



École nationale supérieure d'informatique et d'analyse des  
systèmes

End-of-year Project

---

## R-R: Rescue Robot

---

Realised by:  
Mohamed Manessouri  
Nouhaila Chakouk

Supervised by:  
M. Zine El Abidine Alaoui Ismaili

Jury:

M. FAISSAL ELBOUANANI

M. ZINE EL ABIDINE ALAOUI ISMAILI

2023-2024

## ACKNOWLEDGEMENTS

We are deeply grateful to all the individuals who contributed to the realization of this project. Their expertise, advice, and support have been essential pillars that enabled us to achieve our objectives within the given timeframe.

We would like to express our special gratitude to **Mr. Zine El Abidine Alaoui Ismaili** for his central role in providing insightful suggestions and consistent support. His unwavering commitment to project coordination, especially in drafting this report, has been invaluable.

Finally, we would also like to warmly thank our jury members **Mr. Zine El Abidine Alaoui Ismaili** and **Mr. Faissal Elbouanani** for their valuable guidance and feedback throughout the project. Their constructive feedback and expertise have greatly contributed to improving our presentation skills, which were crucial during our presentations.

.

In the realm of emergency response, time is paramount, where swift action can be the defining factor between survival and tragedy. To confront these challenges head-on, we turn to state-of-the-art technology to aid us in the most daunting scenarios. Our rescue robot project epitomizes this blend of cutting-edge innovation and urgent necessity, with the aim of developing a system capable of promptly and reliably assisting in remote and inaccessible areas.

By seamlessly integrating a myriad of disciplines such as **artificial intelligence**, **mobile development**, and **networking** into **embedded systems**, we've engineered a rescue robot that offers a robust and comprehensive solution to enhance rescue operations and safeguard lives. Throughout this report, we'll delve into the intricacies of our design and construction process, shedding light on how our creation has the potential to transform emergency response efforts in critical situations

# CONTENTS

List of Figures .....	6
<b>1 General Introduction .....</b>	<b>8</b>
<b>2 Requirements Document .....</b>	<b>11</b>
2.1 Model Diagrams .....	12
2.1.1 DFD Diagram .....	12
2.1.2 Use-case Diagram .....	12
2.1.3 BDD Diagram .....	13
2.1.4 Sequence Diagram .....	13
<b>3 Mobile App Development .....</b>	<b>14_15</b>
3.1 Understanding Flutter .....	16
3.1.1 Flutter Framework .....	16
3.1.2 Dart Language .....	16
3.1.3 Flutter Supported Packages .....	17
3.1.3.1 Packages needed .....	17_18
3.2 Back-End Development .....	19
3.2.1 Definition of Back-End development .....	19
<b>4 Computer Vision Revolution .....</b>	<b>20</b>
4 Introduction to computer vision .....	21
4.1 Definition .....	21
4.2 Applications of computer vision .....	22
4.3 Deep learning in computer vision .....	23
4.3.1 CNN layers .....	24
4.4 Real time object detection .....	25
4.4.1 YOLO .....	25

---

4.4.2 YOLO architecture.....	27
4.4.3 Human detection model conception .....	28
4.4.3.1 Loading Dataset.....	28
4.4.3.1.1 Roboflow .....	28
4.4.3.1.2 Dataset.....	28_29
4.4.3.1.3 Yolov5 torch format.....	30
4.4.3.2 Train & Evaluate the model.....	31
<b>5 Video transmission pipeline .....</b>	<b>33_34</b>
5.1 Real time video recording with Mjpg streamer.....	34
5.1.1 Components.....	34_35
5.1.2 Mjpg streamer .....	35_36
5.2 Human detection in Flask server .....	37
5.3 Real time detection in Flutter App .....	38
<b>6 GPS TRACKER .....</b>	<b>39_40</b>
6.1 Neo 6m Module .....	41
6.2 Realtime Database in Firebase .....	42
6.2.1 Definition .....	42
6.2.2 Create a project in Firebase .....	42_43
6.2.3 Realtime Database.....	44
6.3 GPS realtime data pipeline .....	45
6.3.1 Send realtime data to the realtime Database .....	45_46
6.3.2 Fetch realtime data from the realtime Database .....	47_48

<b>7 Robot Control .....</b>	<b>49_50</b>
7.1 Components.....	51
7.2 HTTP requests with Dart.....	52
7.3 RESTFUL API .....	53
7.4 Control Robot's movement with Arduino Uno .....	54
7.5 Final Project .....	55
<b>8 BIBLIOGRAPHY .....</b>	<b>56</b>

## LIST OF FIGURES

1.1 Disaster scenes.....	9
1.2 Rescue Robot .....	10
3 Mobile Application Development .....	15
4.1 Human vision vs computer vision.....	21
4.2 Applications of computer vision.....	22
4.3 Deep learning in computer vision .....	23
4.4.1 Yolo .....	25
4.4.1.1 Histogram (FPS) of different models .....	26
4.4.2. Yolo architecture from the original paper .....	27
4.4.3.1.1 Roboflow .....	28
4.4.3.1.2 Remove nonhuman class from Inrea Dataset .....	29
4.4.3.1.3.1 Verify annotations and labels.....	30
4.4.3.1.3.2 Data augmentation .....	30
4.4.3.1.3.3 Load data in a Yolov5 torch format.....	30
4.4.3.2.1 command for training over 100 epochs .....	31
4.4.3.2.2 metrics .....	31
4.4.3.2.2 Confusion_matrix.....	31
4.4.3.2.3 Detect & Display results.....	32
5.1.1.1 Raspberry pi 3 .....	34
5.1.1.2 Web cam .....	35
5.1.2.1 commands to set up Mjpg-streamer on raspberry pi 3.....	36
5.1.2.2 command to run the server who display the stream .....	36
5.2 Human Detection in Flask server .....	37
5.3 Dart code to access the video .....	38
6 Gps Tracker .....	40
6.1 Gps Neo 6m module.....	41
6.2.2.1 GPS_TRACKER project .....	42
6.2.2.2 Application that generates the API key of the real time database .....	42

---

6.2.2.3 the generated informations .....	43
6.2.2.4 Application to link a flutter app with the project .....	43
6.2.3 Realtime Database in firebase project .....	44
6.3.1.1 gps_send.py .....	46
6.3.1.2 Gps_data sent to the realtime Database .....	46
6.3.2.1 fetch realtime data from the realtime database.....	47
6.3.2.2 The location at latitude 0 & longitude 0.....	48
7. R-R: Rescue Robot .....	50
7.1 Arduino UNO + L293D / Servo Motor.....	51
7.2.1 Http Protocol.....	52
7.2.2 Send commands using http protocol .....	52
7.3 RESTFUL API .....	53
7.4 Arduino code in raspberry pi 3 .....	54
7.5 Final Prototype .....	55



## CHAPTER 1

### GENERAL INTRODUCTION

In emergency and disaster situations, every minute counts. Devastating earthquakes, severe industrial accidents, catastrophic floods, and uncontrollable wildfires are scenarios where human lives may be at stake. Unfortunately, individuals often find themselves trapped in inaccessible or hazardous environments, making their rescue extremely challenging, if not impossible, for traditional rescue teams. Whether it's under rubble, in flooded areas, or in isolated locations, every second can make the difference between life and death.



Figure 1.1: Disaster scenes

In this critical context that rescue robots emerge as an innovative and promising solution.

According to a recent research study conducted by Yutan Li. (2019), rescue robots can significantly reduce the time required to reach victims in challenging environments and increase their chances of survival. Designed to operate in hostile environments where humans cannot safely access, these robots provide a means to swiftly reach victims and provide vital assistance. They can navigate through rugged terrain, detect victims buried under rubble, and provide crucial information to rescue teams, all without endangering the lives of rescuers.

The adoption and development of rescue robots are not just about technological innovation but also an urgent necessity to enhance the effectiveness of rescue operations and save lives. By investing in the research and development of these technologies, we can not only improve the speed and efficiency of disaster relief but also raise public awareness about the crucial importance of this solution.

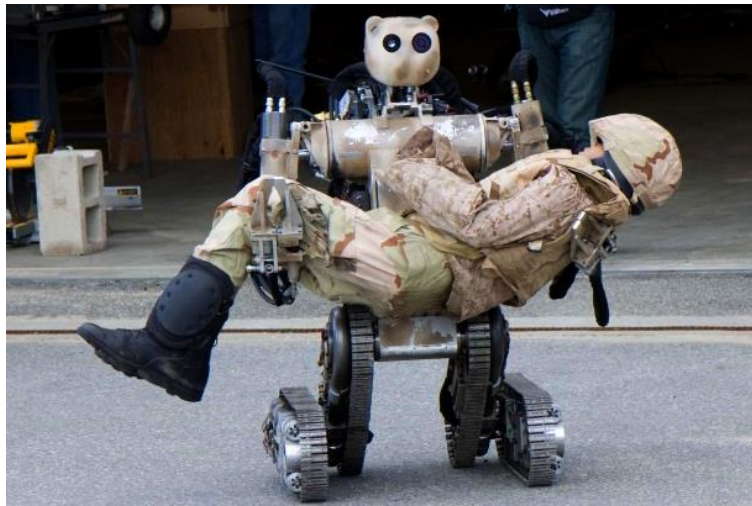


Figure 1.2: Rescue Robot

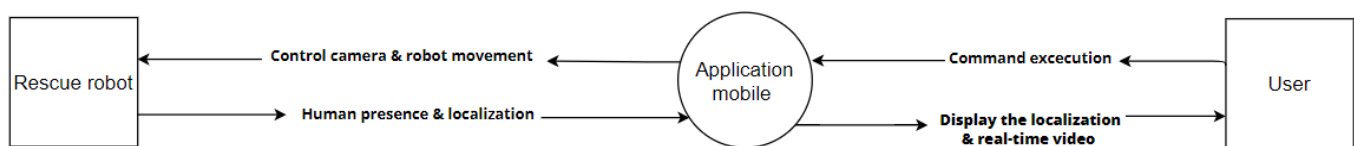
Our innovative rescue robot, crafted through the integration of diverse disciplines including artificial intelligence, mobile development, and networking, excels in exploring expansive areas where conventional search methods may struggle. In situations where individuals face disorientation or lose their way, our robot emerges as a beacon of hope, armed with advanced technologies to swiftly detect humans in distress. Upon identifying a human presence, our accompanying application facilitates real-time video streaming, empowering rescuers to visually assess the situation promptly. Through the intuitive interface of the application, users gain access to a live feed of the robot's surroundings, where humans are highlighted by bounding boxes for quick identification. Moreover, the application boasts remote control functionalities, enabling operators to maneuver the robot through diverse terrain, ensuring precise and efficient assistance to those in need.

## CHAPTER 2

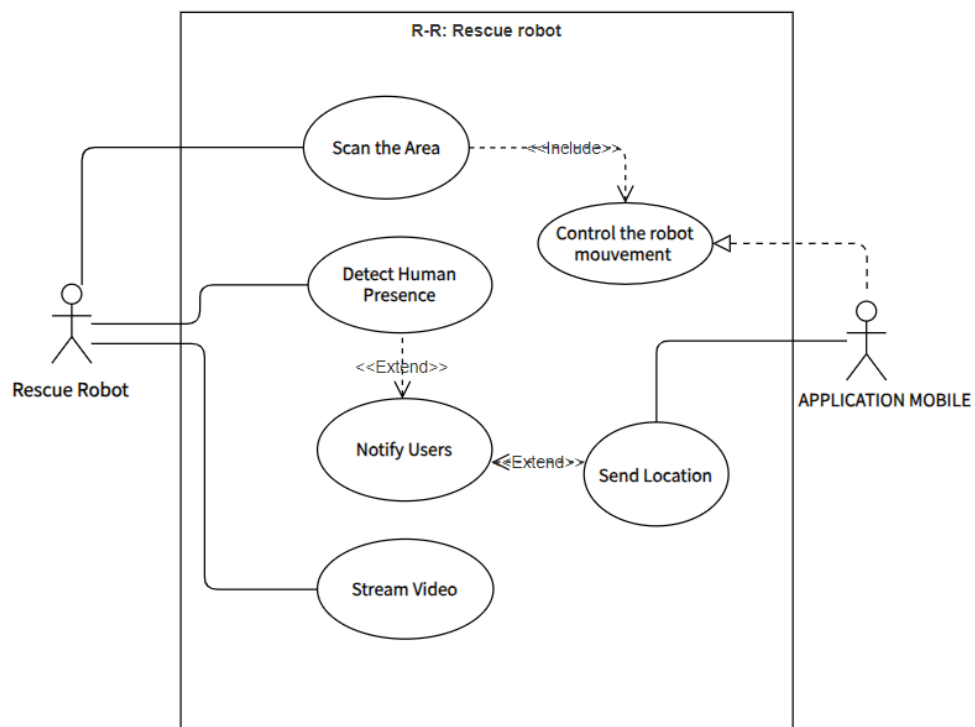
# Requirements Document

## Model Diagrams

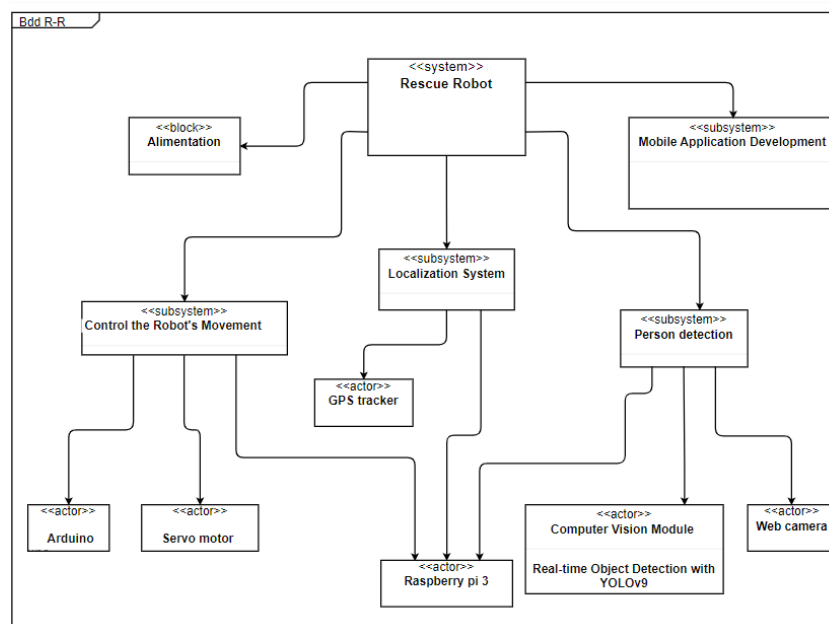
### 1. DFD Diagram



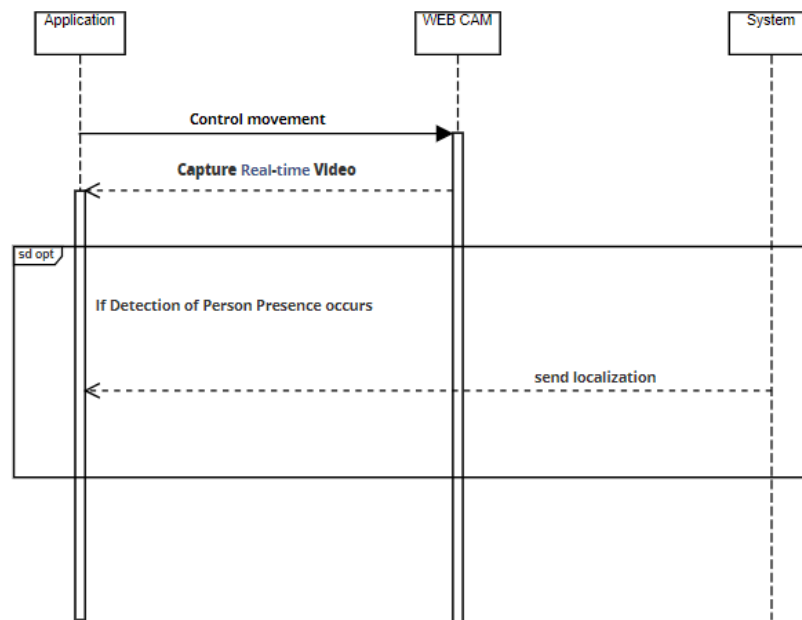
### 2. Use-case Diagram



### 3. BDD Diagram



### 4. Sequence Diagram



## CHAPTER 3

# Mobile app Development

### 3. Mobile Application Development

In today's rescue missions, mobile application development plays a pivotal role in controlling robots and visualizing crucial data. These applications serve as the nerve center, enabling operators to remotely guide robots and access real-time information essential for decision-making. Through intuitive interfaces and seamless connectivity, mobile apps empower responders to navigate complex environments and optimize rescue efforts with unparalleled efficiency and precision.

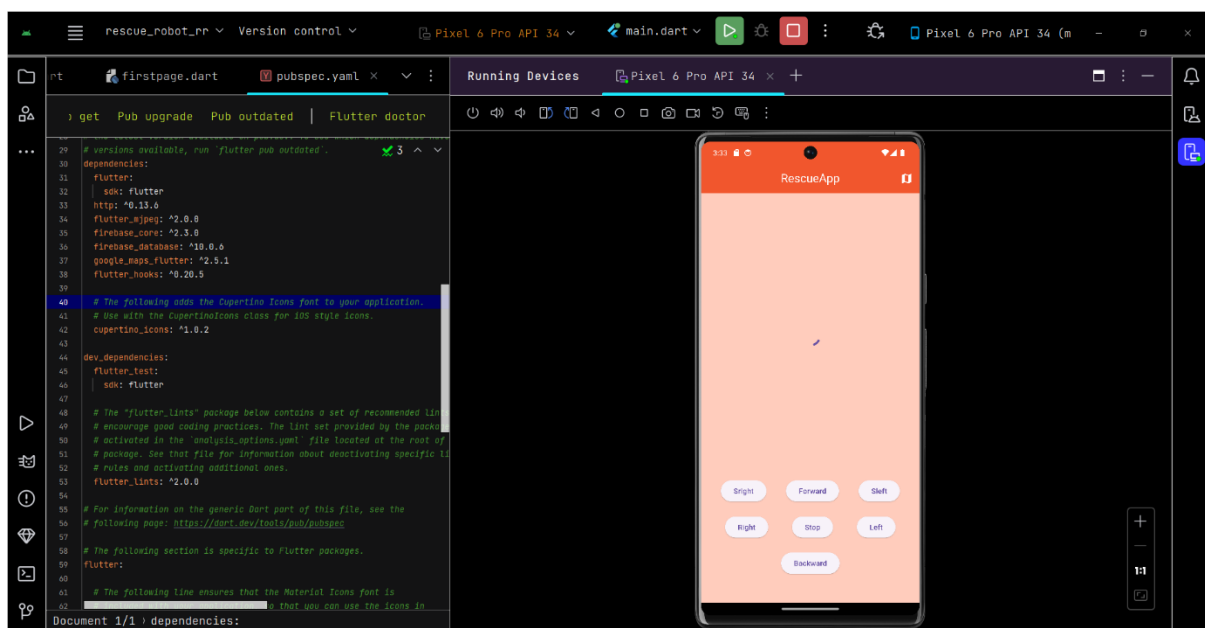


Figure 3: Mobile Application Development



## 3.1 Understanding Flutter

### 3.1.1 Flutter framework



Flutter is a free and open-source UI framework for creating native mobile applications from Google. Released in 2017, Flutter allows developers to build mobile applications with a single codebase and programming language. This capability makes building both iOS and Android apps simpler and faster.

### 3.1.2 Dart Language



Dart is a versatile object-oriented programming language developed by Google, designed for building web, mobile, server, and desktop applications. It's known for its clean syntax, powerful features, and excellent tooling, making it a great choice for beginners and

### 3.1.3 Flutter Supported Packages

Flutter, known for its flexibility in mobile app development, offers a wealth of supported packages. These packages cover a diverse range of functions, from UI design to networking and beyond, empowering developers to swiftly integrate essential features into their applications with ease.

#### 3.1.3.1 Packages needed

- *Http*: Handles HTTP requests and responses, facilitating communication between the Flutter app and web servers.

```
dependencies:  
  http: ^1.2.1
```

- *Flutter\_mjpeg*: Enables streaming of MJPEG video, often used for live video feeds in applications such as surveillance or monitoring.

```
dependencies:  
  flutter_mjpeg: ^2.0.4
```

- *Google maps\_flutter*: Integrates Google Maps functionality into Flutter apps, allowing developers to display maps, add markers, and implement various location-based features.

```
dependencies:  
  google_maps_flutter: ^2.6.1
```

- *Firestore\_core*: Provides the core functionality for Firestore services integration in Flutter apps, including initialization and configuration.

```
dependencies:  
  firestore_core: ^2.32.0
```

- *Firestore\_database*: Offers real-time database capabilities through Firestore, allowing developers to store and sync data across devices and platforms.

```
dependencies:  
  firestore_database: ^10.5.7
```

- *Flutter\_hooks*: Facilitates the implementation of stateful logic in Flutter widgets, offering a more concise and efficient way to manage widget state compared to traditional methods.

```
dependencies:  
  flutter_hooks: ^0.20.5
```

## 3.2 Back-End Development

### 3.2.1 Definition of Back-End development

Back-end development refers to the server-side of development. It encompasses all the processes that happen behind the scenes, away from the user's view. It is a combination of servers and databases. Servers control how users access files. Databases are organized and structured collections of data.

#### The Role of Back End Development

- **Processing User Requests:** One of the primary functions of the back end is to handle user requests. Whether it's submitting a form, logging in, or fetching data, the back end processes these requests and ensures the appropriate response is generated.
- **Database Management:** This involves creating efficient database structures and optimizing queries for faster data retrieval. Also, the data stored in the databases must be accurate and secure.
- **Server Management:** The back end is responsible for managing servers, ensuring they run smoothly, and deploying updates or patches when necessary.

Back-end developers also design and develop **API** or "application programming interface", is a set of rules or protocols that enables software applications to communicate with each other to exchange data, features and functionality.

*Back-End is crucial for the next chapters.*

## CHAPTER 4

# Computer Vision Revolution

## 4. Introduction to COMPUTER VISION

### 4.1 Definition

Computer vision is a field of artificial intelligence (AI) that uses machine learning and neural networks to teach computers and systems to derive meaningful information from digital images, videos and other visual inputs—and to make recommendations or take actions when they see defects or issues.

**If AI enables computers to think, computer vision enables them to see, observe and understand**

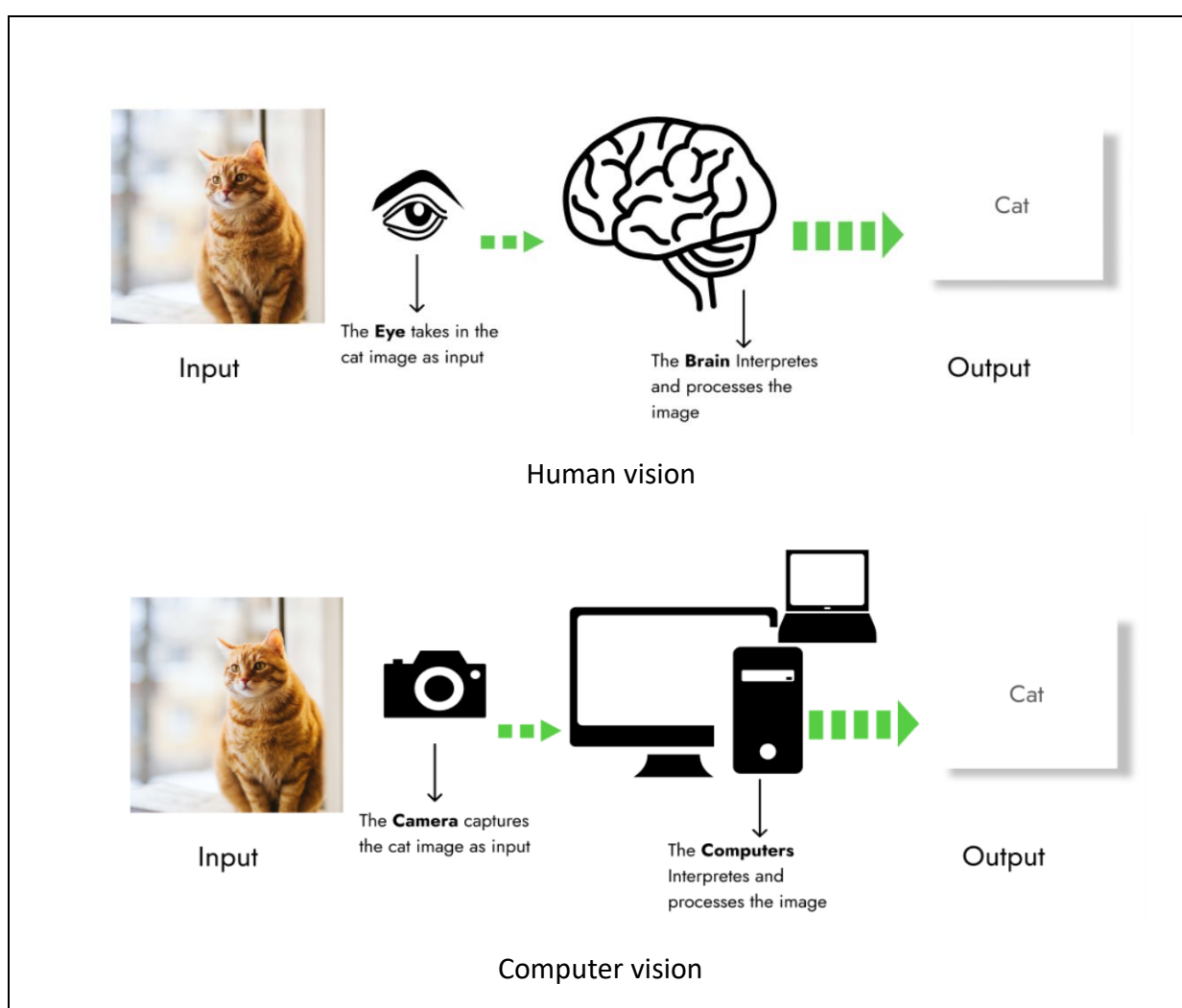


Figure 4.1: Human vision vs Computer vision

## 4.2 Applications of computer vision



Figure 4.2: Applications of computer vision

## 4.3 Deep Learning in Computer vision

Deep learning has transformed computer vision by enabling automatic feature extraction and improving accuracy in tasks like object detection and image classification. With models such as convolutional neural networks (CNNs), deep learning processes vast amounts of visual data to recognize patterns, making it crucial for advancements in various fields.

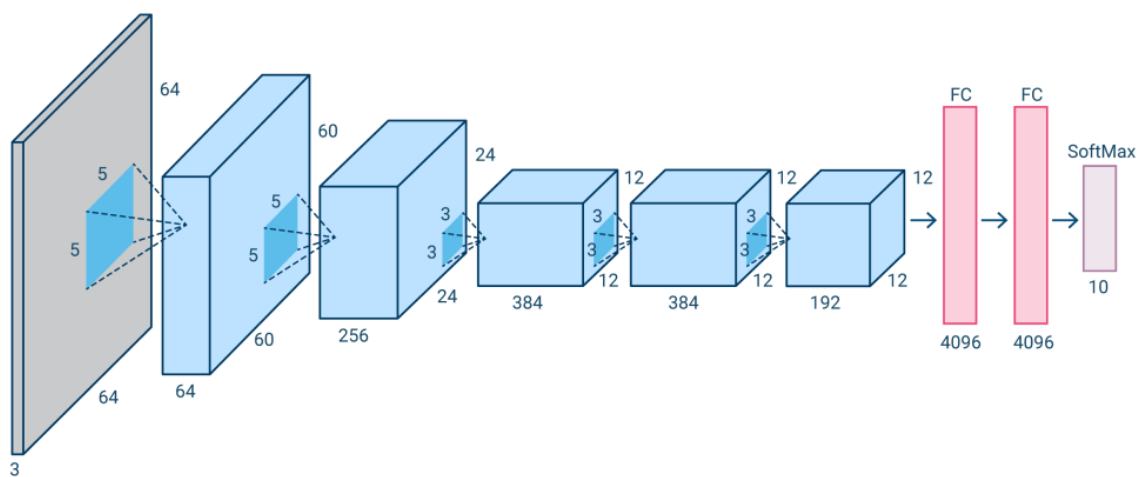


Figure 4.3: Deep Learning in Computer Vision



### 4.3.1 CNN Layers

- **Convolutional Layers:** These layers apply filters to input images, capturing spatial hierarchies of features like edges, textures, and patterns
- **Pooling Layers:** Pooling layers reduce the spatial dimensions of the feature maps while retaining important information. Max pooling, for example, selects the maximum value from a region of the feature map, effectively down sampling it. This helps in making the network more robust to variations in input and reduces computational complexity.
- **Fully connected Layers:** After convolutional and pooling layers, the feature maps are flattened into a single vector and passed through fully connected layers. These layers perform classification by learning complex relationships between features extracted in earlier layers and mapping them to specific output classes. Fully connected layers utilize dense connections between all neurons in adjacent layers.
- **Activation Functions:** Activation functions introduce non-linearity into the network, allowing it to learn complex patterns.

## 4.4 Real-time object detection

Real-time object detection has emerged as a critical component in numerous applications across various fields, including autonomous vehicles, robotics, video surveillance, and augmented reality.

### 4.4.1 YOLO

The YOLO (You Only Look Once) model is a revolutionary approach in the field of computer vision, particularly for object detection tasks. YOLO stands out for its speed, its good generalization and efficiency, making real-time object detection a reality.

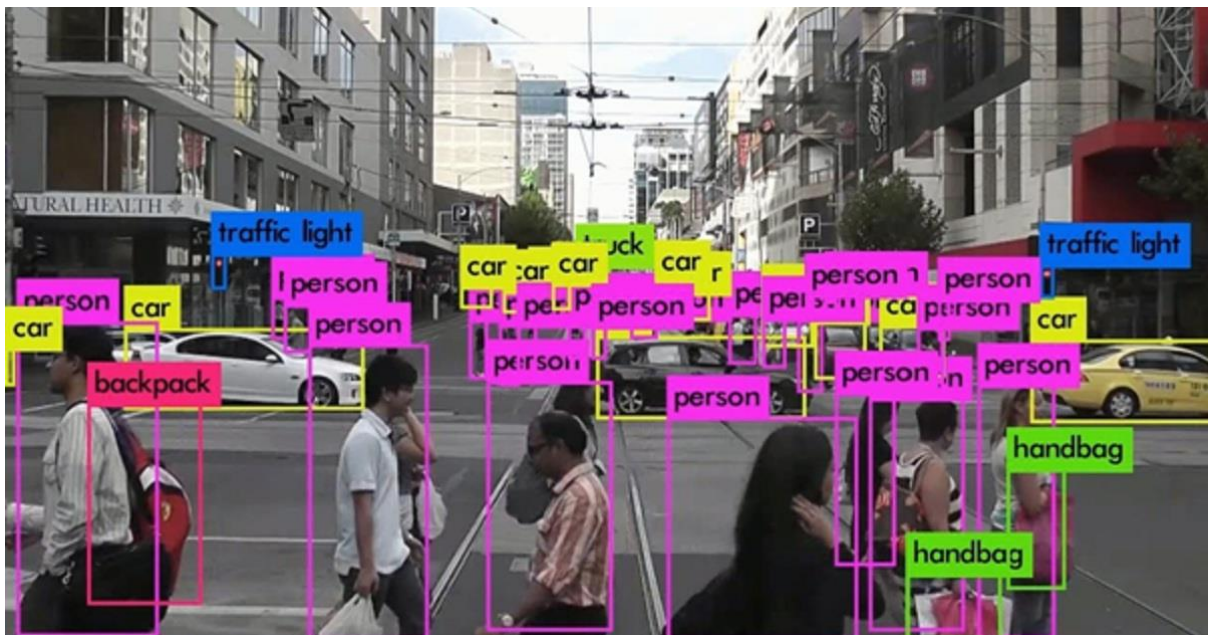


Figure 4.4.1: Yolo

- Speed

YOLO is extremely fast because it does not deal with complex pipelines. It can process images at 45 Frames Per Second (FPS). In addition, YOLO reaches more than twice the mean Average Precision (mAP) compared to other real-time systems, which makes it a great candidate for real-time processing.

From the graphic below, we observe that YOLO is far beyond the other object detectors with 91 FPS.

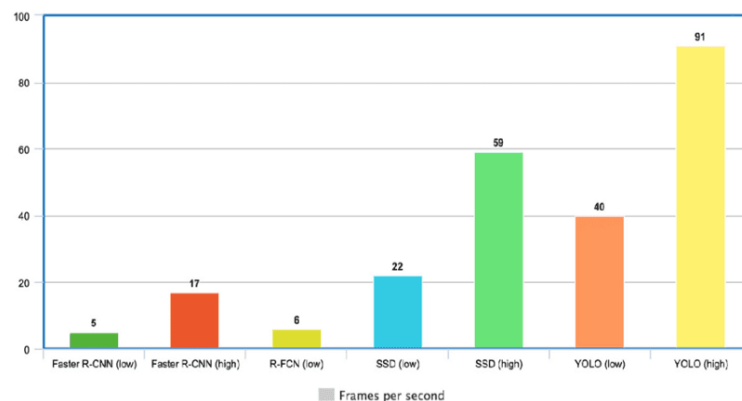


Figure 4.4.1.1: Histogram (FPS) of different models

- High detection accuracy

YOLO is far beyond other state-of-the-art models in accuracy with very few background errors.

- Better generalization

YOLO (You Only Look Once) has a strong generalization capability due to its innovative architecture, which simultaneously predicts multiple bounding boxes and class probabilities for objects in an image.

- Open Source

Making YOLO open-source led the community to constantly improve the model. This is one of the reasons why YOLO has made so many improvements in such a limited time.

## 4.4.2 YOLO architecture

YOLO architecture is similar to GoogleNet. As illustrated below, it has overall 24 convolutional layers, four max-pooling layers, and two fully connected layers.

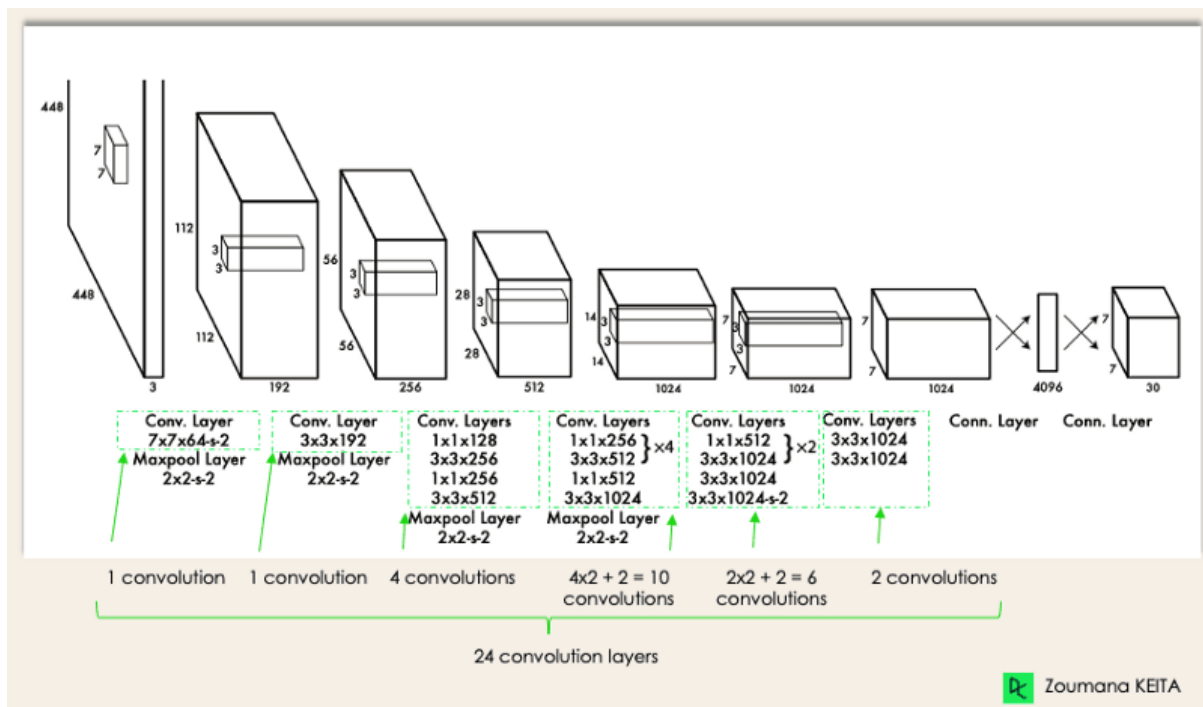


Figure 4.4.2: Yolo Architecture from the original paper

The architecture works as follows:

- Resizes the input image into 448x448 before going through the convolutional network.
- A 1x1 convolution is first applied to reduce the number of channels, which is then followed by a 3x3 convolution to generate a cuboidal output.
- The activation function under the hood is ReLU, except for the final layer, which uses a linear activation function.
- Some additional techniques, such as batch normalization and dropout, respectively regularize the model and prevent it from overfitting.

## 4.4.3 Human Detection Model conception

### 4.4.3.1 Loading the Dataset

#### 4.4.3.1.1 RoboFlow

**Definition:** Roboflow is a platform designed to facilitate the development and deployment of computer vision models. It provides tools and services for managing the entire lifecycle of machine learning projects focused on image and video data.

In this project, it is used to generate our dataset in a YOLOv5 torch format.

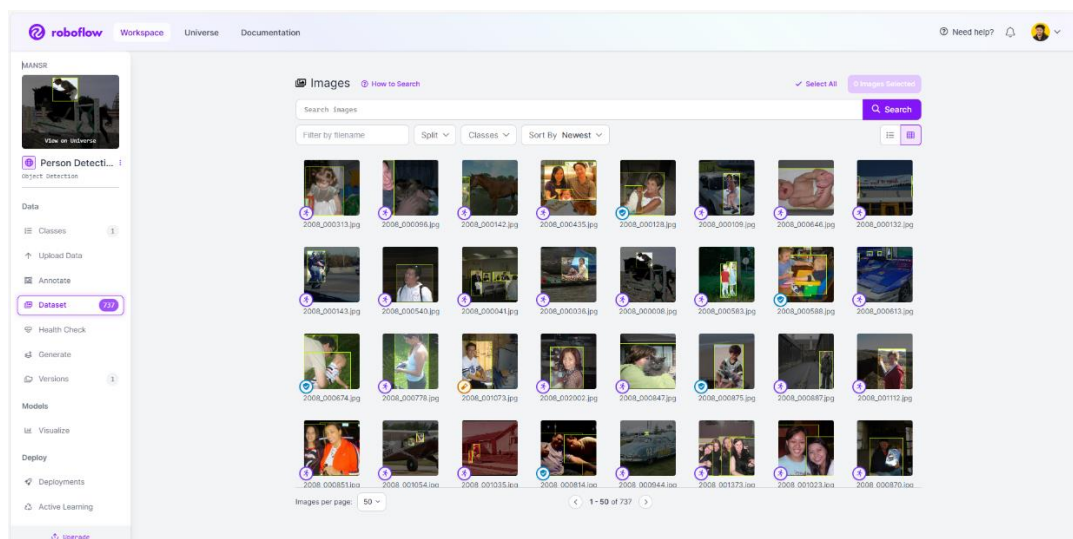


Figure 4.4.3.1.1: Roboflow

#### 4.4.3.1.2 Dataset

The INRIA PERSON dataset is imported from Kaggle (platform for data scientists and machine learning enthusiasts to collaborate, compete, and learn.)

The INRIA Person Dataset is organized into three main folders: Train, Test, and Val. Each of these folders contains two subfolders: Images, which hold the image files, and Annotations, which include the corresponding XML files with annotations. The dataset encompasses two classes: person and non-human.

we are interested in removing the non-human class to ensure that only annotations related to the person class are utilized since we are building a human detection model.

```
[ ] import os
    import xml.etree.ElementTree as ET

[ ] remove_class_name = 'nonhuman'

[ ] subdirs = ['Train', 'Val', 'Test']

[ ] image_dirs = {subdir: os.path.join(Dataset_path, subdir, 'Images') for subdir in subdirs}
    annot_dirs = {subdir: os.path.join(Dataset_path, subdir, 'Annotations') for subdir in subdirs}

[ ] # Function to remove images and annotations containing the remove_class_name
    def remove_nonhuman_images_and_annotations(image_dirs, annot_dirs, remove_class_name):
        for subdir in subdirs:
            images_path = image_dirs[subdir]
            print(images_path)
            annotations_path = annot_dirs[subdir]

            if not os.path.exists(images_path) or not os.path.exists(annotations_path):
                print(f"Images or annotations directory does not exist: {images_path}, {annotations_path}")
                continue

            for annotation_file in os.listdir(annotations_path):
                annotation_path = os.path.join(annotations_path, annotation_file)
                image_file = annotation_file.replace('.xml', '.jpg') # Adjust if image extension is different
                image_path = os.path.join(images_path, image_file)

                tree = ET.parse(annotation_path)
                root = tree.getroot()

                # Check if any object contains the class name to remove
                contains_remove_class = any(obj.find('name').text == remove_class_name for obj in root.findall('object'))

                if contains_remove_class:
                    # Remove annotation and image file
                    os.remove(annotation_path)
                    if os.path.exists(image_path):
                        os.remove(image_path)
                    print(f"Removed {annotation_file} and corresponding image {image_file}.")
```

Figure 4.4.3.1.2: remove nonhuman class from Inrea Dataset

### 4.4.3.1.3 YOLOv5 Torch Format

- Verify the annotations and the labels

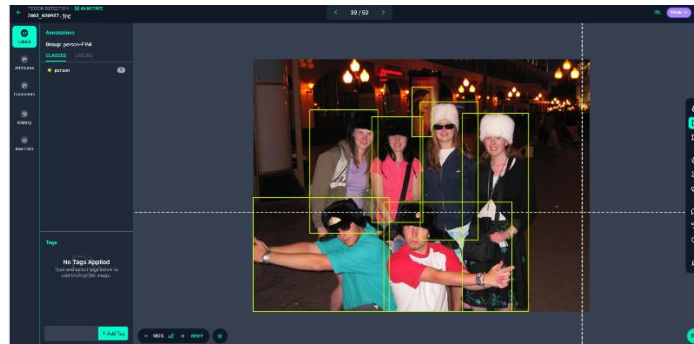


Figure 4.4.3.1.3.1: verify annotations and labels

- Apply some Data augmentation

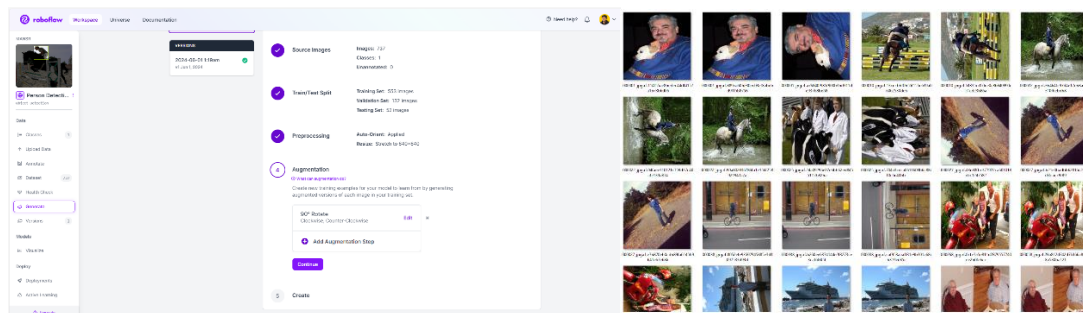


Figure 4.4.3.1.3.2: Data augmentation

- Load the data in a YOLOv5 Torch format

```
[ ] from roboflow import Roboflow

rf = Roboflow(api_key="YdVF2nLebFSVBXfodM9N")
project = rf.workspace("mansr").project("person-detection-5u4uo")
version = project.version(1)
dataset = version.download("yolov5")

loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in Person-Detection-1 to yolov5pytorch:: 100%|██████████| 99371/99371 [00:02<00:00, 41894.83it/s]
Extracting Dataset Version Zip to Person-Detection-1 in yolov5pytorch:: 100%|██████████| 3330/3330 [00:01<00:00, 3187.92it/s]
```

Figure 4.4.3.1.3.3: Load data in a YOLOv5 torch format

### 4.4.3.2 Train & evaluate the model

- Training

```
[ ] !python /content/drive/MyDrive/ColabNotebooks/yolov5/train.py --img 416 --batch 16 --epochs 100 --data /content/drive/MyDrive/ColabNotebooks/yolov5/Person-Detection-1/data.yaml --weights yolov5s.pt --cache
```

Show hidden output

Figure 4.4.3.2.1: command for training over 100 epochs

- Evaluation

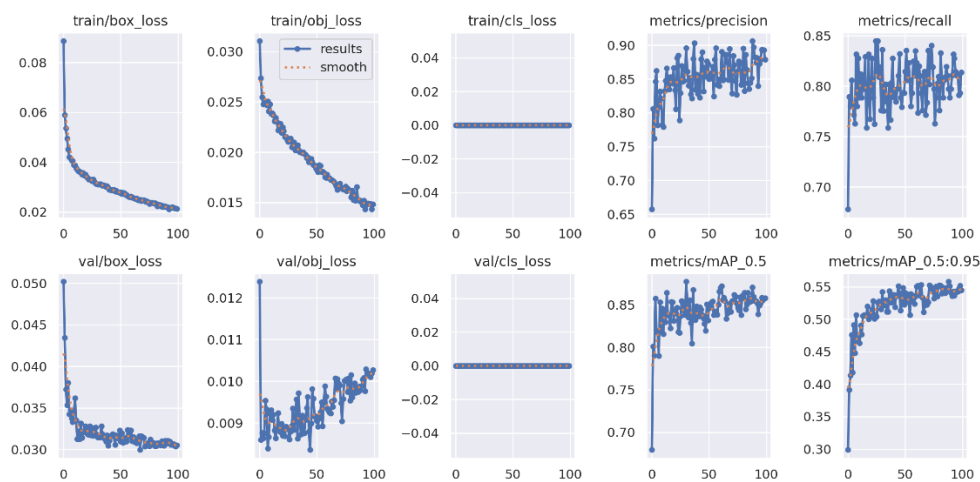


Figure 4.4.3.2.2: metrics

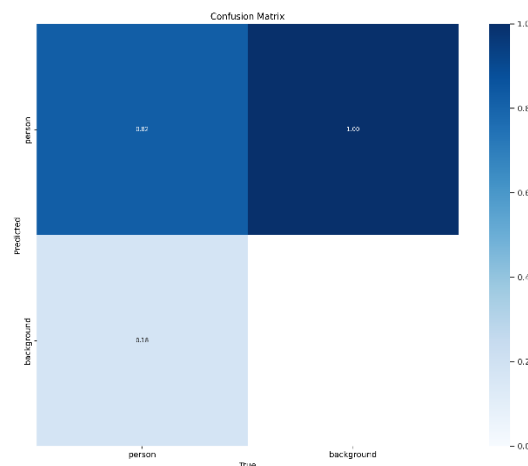


Figure 4.4.3.2.2: Confusion\_matrix



### 4.4.3.2.3 Prediction

```
[ ] | python /content/drive/MyDrive/ColabNotebooks/yolov5/detect.py --weights /content/drive/MyDrive/ColabNotebooks/yolov5/runs/train/exp4/weights/best-fp16.tflite --img 416 --conf 0.2 --source /content/drive/MyDrive/ColabNotebooks  
Show hidden output  
i = 0  
for imageName in glob.glob('/content/drive/MyDrive/ColabNotebooks/yolov5/runs/detect/exp3/*-jpg'):  
    i += 1  
    if i < 52:  
        display(Image(filename=imageName))  
        print("\n")
```



Figure 4.4.3.2.3 : Detect & Display results

## CHAPTER 5

# Video transmission pipeline

## 5. Video transmission pipeline

The use of a video transmission pipeline for real-time human detection addresses hardware limitations and computational efficiency. Leveraging a server-based approach via Flask offloads detection tasks from the Raspberry Pi 3, optimizing performance and enabling seamless transmission. This strategic division enhances system responsiveness and scalability while facilitating future updates with ease.

### 5.1 Real-time Video Recording with MJPG Streamer

#### 5.1.1 Components

**Raspberry pi:** is a small computer that's about the size of a computer mouse (85 mm x 56 mm is the standard size). It looks like a PC motherboard, but all the components are already on it (CPU, memory, wireless module, USB, network, etc.).



Figure 5.1.1.1: raspberry pi 3

**Web cam:** Unlike a digital camera, a webcam does not include a built-in memory chip or flash memory card since it is for recording and sending images to a computer. It is why cameras have USB cords protruding from the rear. The USB connection provides power to the camera from the computer. It returns the digital data acquired by the webcam's image sensor to the computer, which then transmits it to the Internet. Some webcams operate wirelessly and do not need a computer connection but use Wi-Fi to transfer their images to your Internet router. It makes them accessible to other devices on your home network or the Internet anywhere.



Figure 5.1.1.2: web cam

## 5.1.2 Mjpg streamer

**Definition:** MJPG Streamer is a tool used for streaming live video from a webcam to a web browser or other devices over a network. It captures video frames from a webcam, encodes them in the MJPEG (Motion JPEG) format, and then transmits them over HTTP. It provides a flexible interface that allows users to specify the video source and configure various parameters such as image size, quality, and FPS.

**Set up Mjpg streamer on raspberry pi:** These commands should be written in the Raspberry Pi's command line interface to install MJPG Streamer on Raspberry Pi OS.

```
sudo apt-get install cmake libjpeg8-dev
sudo apt-get install gcc g++
cd mjpg-streamer
cd mjpg-streamer-experimental
make
sudo make install
```

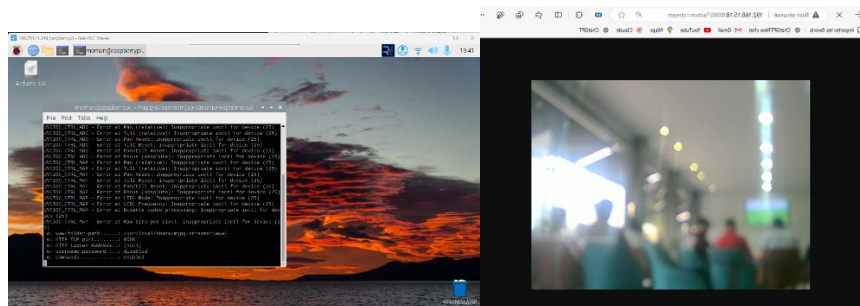
Figure 5.1.2.1: commands to set up Mjpg-streamer on raspberry pi 3

**Stream the video:** After navigating to the Mjpg-streamer-experimental directory, this command displays the stream on a web server:

```
/usr/local/bin/mjpg_streamer -i "input_uvc.so -r 640x480 -d /dev/video0 -f 24  
-q 80" -o "output_http.so -p 8080 -w /usr/local/share/mjpg-streamer/www"
```

Figure 5.1.2.2: command to run the server who display the stream

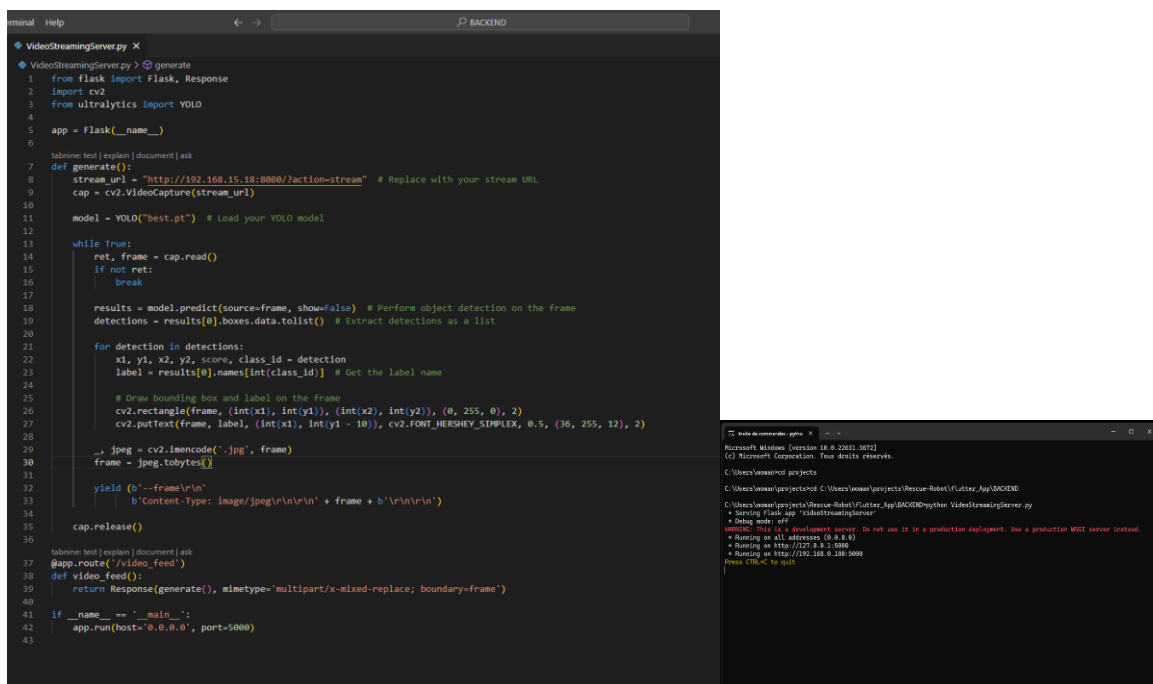
**Result:**



## 5.2 Human Detection in Flask Server

**Flask:** Flask is a lightweight and flexible web framework for Python that is designed to help developers create web applications quickly and easily. It is known for its simplicity and minimalism, providing the essential tools needed for web development without enforcing a particular project structure or dependencies. Flask supports extensions for adding advanced functionality, such as database integration, form validation, and user authentication. Its ease of use and scalability make it a popular choice for both small and large web projects.

**Create Flask server:** This server is responsible of hosting the video after the detections are made so it can be accessible to the flutter app.



```

1 from flask import Flask, Response
2 import cv2
3 from ultralytics import YOLO
4
5 app = Flask(__name__)
6
7 def generate():
8     stream_url = "http://192.168.15.18:8080/action-stream" # Replace with your stream URL
9     cap = cv2.VideoCapture(stream_url)
10
11     model = YOLO("best.pt") # Load your YOLO model
12
13     while True:
14         ret, frame = cap.read()
15         if not ret:
16             break
17
18         results = model.predict(source=frame, show=False) # Perform object detection on the frame
19         detections = results[0].boxes.data.tolist() # Extract detections as a list
20
21         for detection in detections:
22             x1, y1, x2, y2, score, class_id = detection
23             label = results[0].names[int(class_id)] # Get the label name
24
25             # Draw bounding box and label on the frame
26             cv2.rectangle(frame, (int(x1), int(y1)), (int(x2), int(y2)), (0, 255, 0), 2)
27             cv2.putText(frame, label, (int(x1), int(y1 - 10)), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (30, 255, 12), 2)
28
29         _, jpeg = cv2.imencode('.jpg', frame)
30         frame = jpeg.tobytes()
31
32         yield (b'--frame\r\n'
33               b'Content-type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
34
35     cap.release()
36
37 @app.route('/video_feed')
38 def video_feed():
39     return Response(generate(), mimetype='multipart/x-mixed-replace; boundary=frame')
40
41 if __name__ == '__main__':
42     app.run(host='0.0.0.0', port=5000)
43

```

Figure 5.2: Human Detection in Flask Server

## 5.3 Real-time Detection Displayed in Flutter App

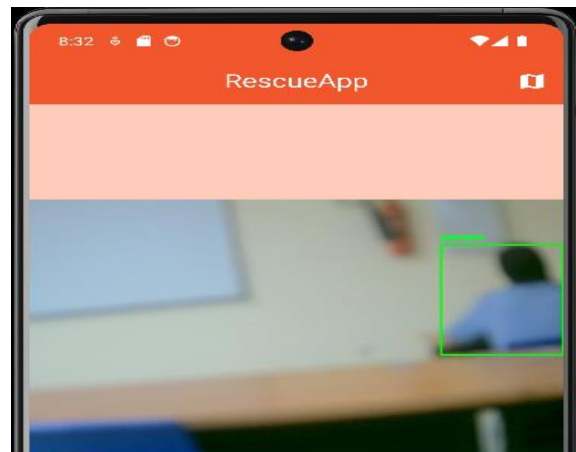
**Mjpg\_flutter package to visualize the results:** Using the URL generated from the flask server and using the Mjpg\_flutter package, the flutter app can access the video resulting and visualize it.

**Dart code to access the Video:**

```
final String videoStreamUrl = 'http://10.1.6.71:5000/video_feed';
```

```
body: Column(  
  children: [  
    // Video Streaming Widget  
    Expanded(  
      child: Mjpeg(  
        isLive: isRunning.value,  
        error: (context, error, stack) {  
          print(error);  
          print(stack);  
          return Text(error.toString(), style: TextStyle(color: Colors.red));  
        },  
        stream: videoStreamUrl,  
      ), // Mjpeg  
    ), // Expanded  
  ],  
)
```

Figure 5.3: Dart code to access the Video







## 6. GPS tracker

Integrating a GPS tracker into the rescue robot is crucial for enhancing its operational effectiveness. The GPS tracker provides real-time location data, allowing rescue teams to monitor the robot's position accurately and efficiently. This capability ensures precise navigation and coordination, particularly in challenging or unfamiliar terrains. Additionally, real-time location tracking enables quick response and retrieval in case of emergencies, significantly improving the safety and success rate of rescue missions.

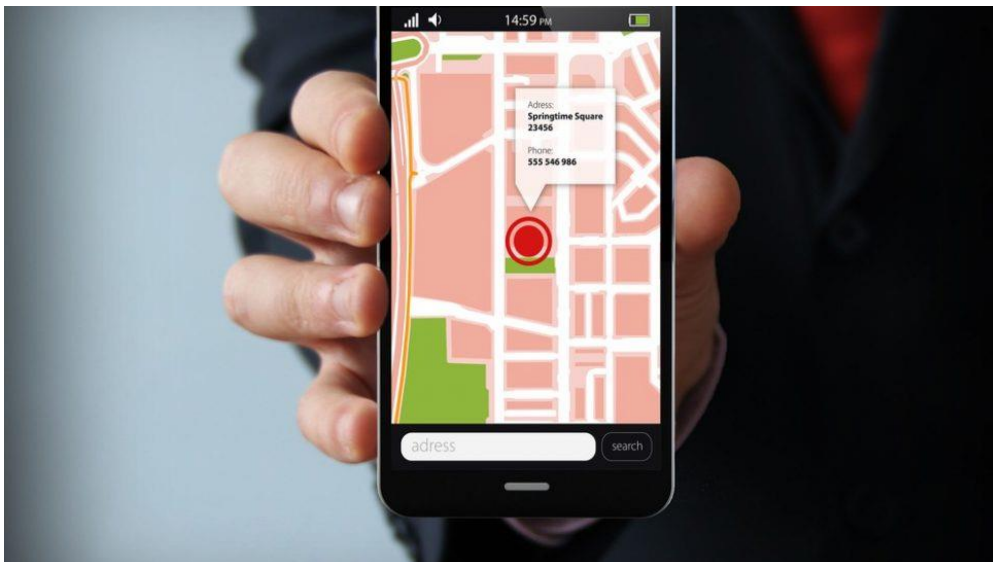


Figure 6: GPS tracker conception

## 6.1 Neo 6m Module

**Definition:** The NEO-6M module is a GPS receiver that provides accurate positioning and timing information. It features high sensitivity, low power consumption, and fast acquisition times. The NEO-6M typically includes a built-in antenna and supports communication via serial interfaces, making it a versatile choice for adding GPS functionality to projects.

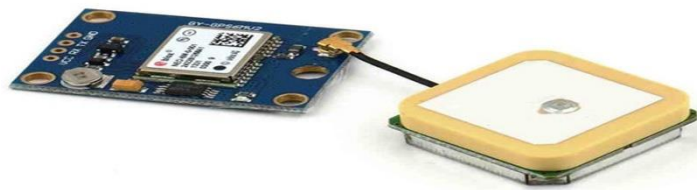


Figure 6.1: Gps Neo 6m module

## 6.2 Realtime Database in Firebase

### 6.2.1 definition

Firebase (Backend as a service platform) is a comprehensive mobile and web application development platform developed by Google. It provides developers with a suite of cloud-based tools and services designed to help create, maintain, and improve applications.

Firebase works as the intermediary between the application and the cloud-based services it needs to function effectively, handling data storage, user authentication, analytics, and much more.

### 6.2.2 Create a Project in Firebase

#### 1. Set Up Your Firebase Account

#### 2. Create a New Project

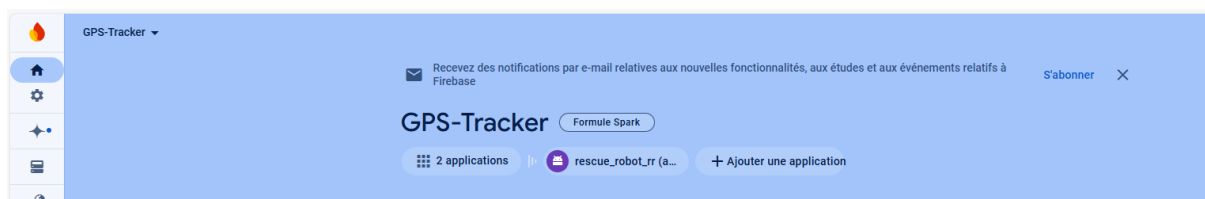


Figure 6.2.2.1: GPS-TRACKER project

#### 3. Add an App to Your Project

- Application to receive data

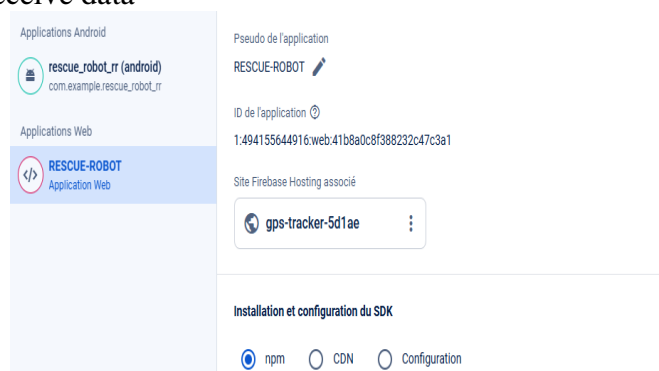


Figure 6.2.2.2: Application that generates the API key of the real time database

```
// Import the functions you need from the SDKs you need
import { initializeApp } from "firebase/app";
// TODO: Add SDKs for Firebase products that you want to use
// https://firebase.google.com/docs/web/setup#available-libraries

// Your web app's Firebase configuration
const firebaseConfig = {
  apiKey: "AIzaSyAIfcHI0KJjDLkR5d0Rdf5bX31Ih6xQEvE",
  authDomain: "gps-tracker-5d1ae.firebaseio.com",
  databaseURL: "https://gps-tracker-5d1ae.firebaseio.com",
  projectId: "gps-tracker-5d1ae",
  storageBucket: "gps-tracker-5d1ae.appspot.com",
  messagingSenderId: "494155644916",
  appId: "1:494155644916:web:41b8a0c8f388232c47c3a1"
};

// Initialize Firebase
const app = initializeApp(firebaseConfig);
```

Figure 6.2.2.3: the generated informations

- Application to connect with a flutter app

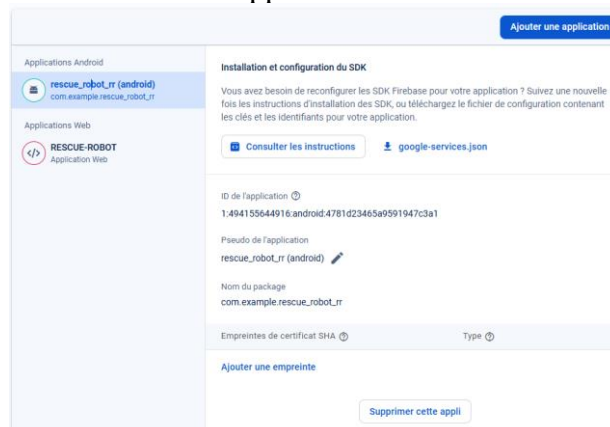


Figure 6.2.2.4: Application to link a flutter app with the project

## 6.2.3 Realtime Database

**Realtime Database:** Firebase Realtime Database is a cloud-hosted NoSQL database that enables real-time data synchronization across all clients connected to the database. When data changes, the connected clients receive updates immediately.

The database stores data as JSON (JavaScript Object Notation) objects which allows for a flexible and hierarchical data structure.

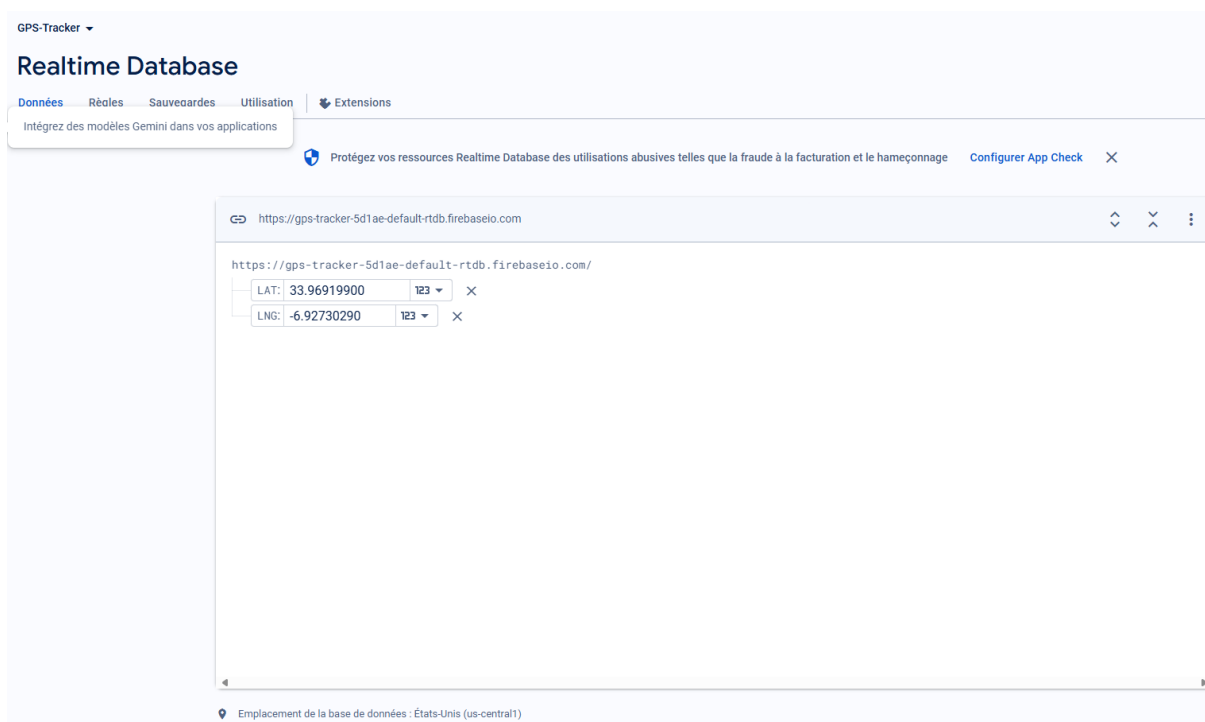
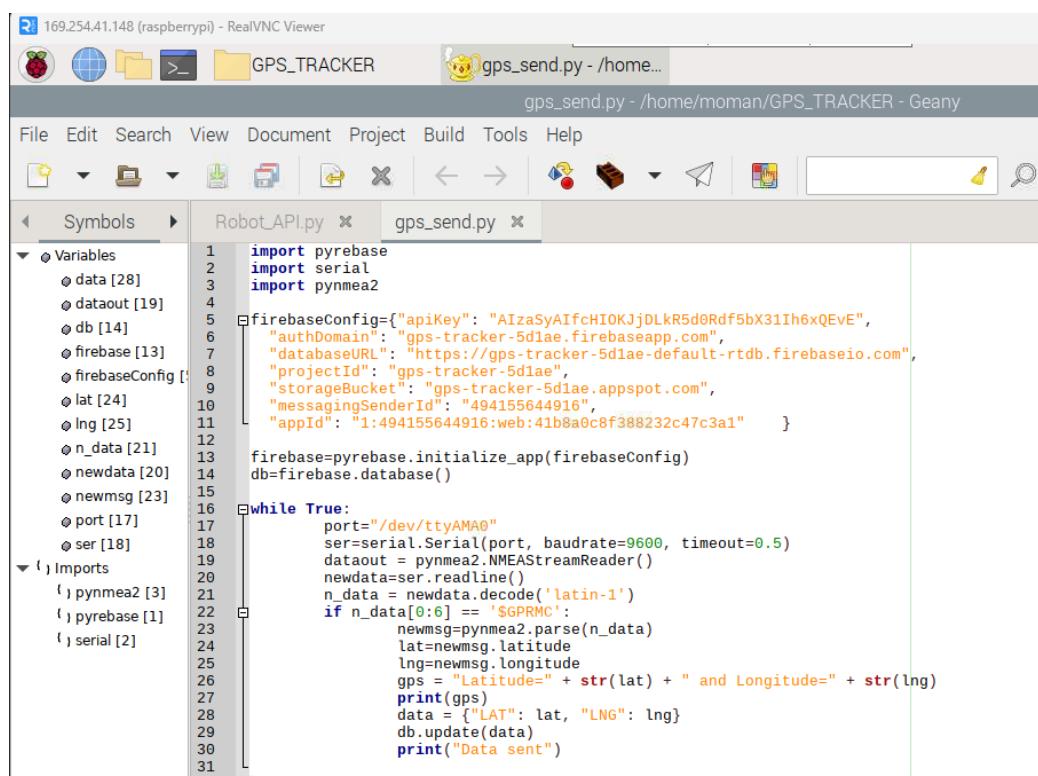


Figure 6.2.3: Realtime Database in firebase project

## 6.3 Gps real time data pipeline

### 6.3.1 Send real time data to the Realtime Database

After linking the NEO 6M GPS module to the Raspberry Pi and obtaining the necessary information about the Realtime Database from the application, a Python script is employed to send the location data to the Realtime Database. This setup ensures that the GPS coordinates are continuously updated and available for real-time tracking and monitoring.

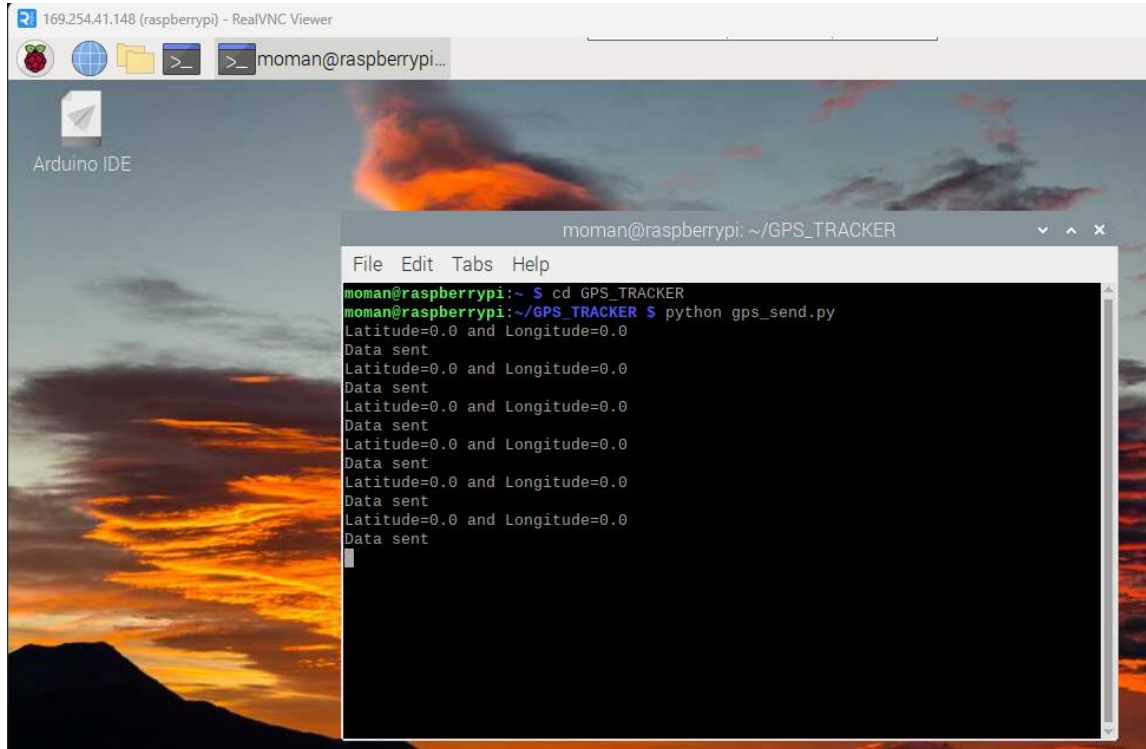


```

1  import pyrebase
2  import serial
3  import pynmea2
4
5  firebaseConfig={
6      "apiKey": "AIzaSyAIfcHI0KJjDLkR5d0Rdf5bX31Ih6xQEvE",
7      "authDomain": "gps-tracker-5d1ae.firebaseio.com",
8      "databaseURL": "https://gps-tracker-5d1ae.firebaseio.com",
9      "projectId": "gps-tracker-5d1ae",
10     "storageBucket": "gps-tracker-5d1ae.appspot.com",
11     "messagingSenderId": "494155644916",
12     "appId": "1:494155644916:web:41b8a0c8f388232c47c3a1"
13 }
14
15 firebase=pyrebase.initialize_app(firebaseConfig)
16 db=firebase.database()
17
18 while True:
19     port="/dev/ttyAMA0"
20     ser=serial.Serial(port, baudrate=9600, timeout=0.5)
21     dataout = pynmea2.NMEAStreamReader()
22     newdata=ser.readline()
23     n_data = newdata.decode('latin-1')
24     if n_data[0:6] == '$GPRMC':
25         newmsg=pynmea2.parse(n_data)
26         lat=newmsg.latitude
27         lng=newmsg.longitude
28         gps = "Latitude=" + str(lat) + " and Longitude=" + str(lng)
29         print(gps)
30         data = {"LAT": lat, "LNG": lng}
31         db.update(data)
32         print("Data sent")
33

```

Figure 6.3.1.1: gps\_send.py



The screenshot shows a terminal window titled "momman@raspberrypi: ~/GPS\_TRACKER". The terminal output is as follows:

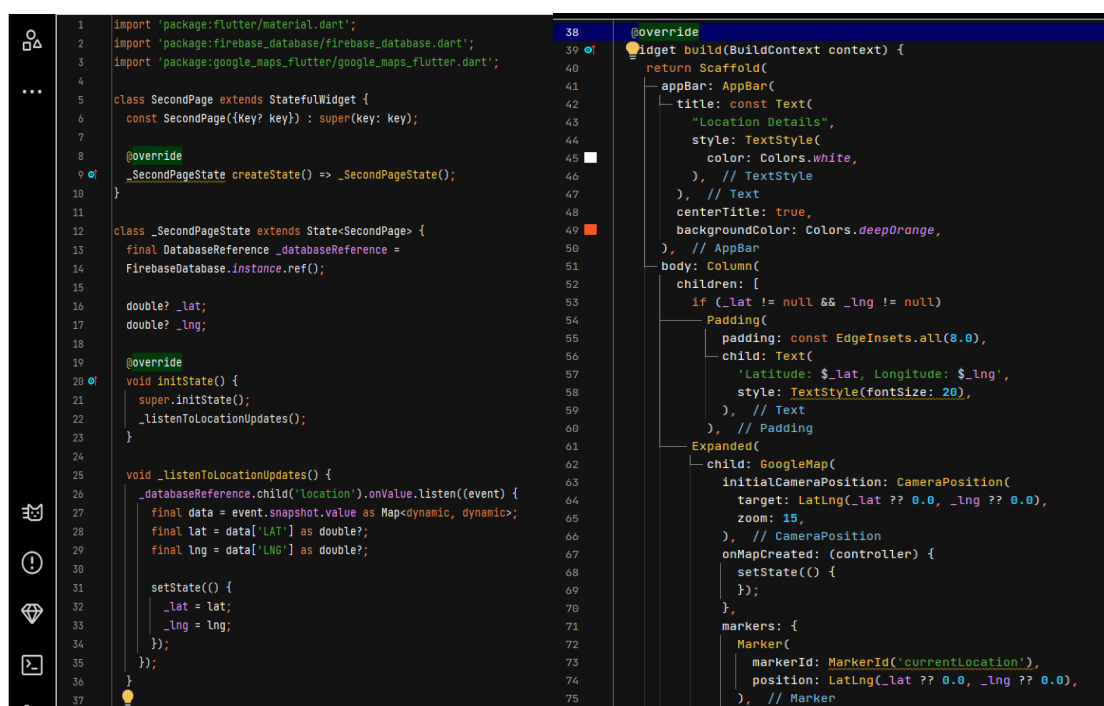
```
momman@raspberrypi:~ $ cd GPS_TRACKER
momman@raspberrypi:~/GPS_TRACKER $ python gps_send.py
Latitude=0.0 and Longitude=0.0
Data sent
Latitude=0.0 and Longitude=0.0
Data sent
Latitude=0.0 and Longitude=0.0
Data sent
Latitude=0.0 and Longitude=0.0
Data sent
Latitude=0.0 and Longitude=0.0
Data sent
Latitude=0.0 and Longitude=0.0
Data sent
Latitude=0.0 and Longitude=0.0
Data sent
```

Figure 6.3.1.2: Gps\_data sent to the realtime database

The Gps module can't detect signal, that is why the data sent is 0 for Latitude, 0 for Longitude.

## 6.3.2 Fetch realtime data from the realtime database

A Dart code is written to obtain real-time data from the Realtime Database. This code continuously fetches the latest information, ensuring that the application reflects current data and provides up-to-date insights and tracking capabilities, beside that using Google maps API key and Google\_maps\_flutter package, we can visualize the updated location of the robot.



```

1 import 'package:flutter/material.dart';
2 import 'package:firebase_database/firebase_database.dart';
3 import 'package:google_maps_flutter/google_maps_flutter.dart';
4
5 class SecondPage extends StatefulWidget {
6   const SecondPage({Key? key}) : super(key: key);
7
8   @override
9   _SecondPageState createState() => _SecondPageState();
10 }
11
12 class _SecondPageState extends State<SecondPage> {
13   final DatabaseReference _databaseReference =
14     FirebaseDatabase.instance.ref();
15
16   double? _lat;
17   double? _lng;
18
19   @override
20   void initState() {
21     super.initState();
22     _listenToLocationUpdates();
23   }
24
25   void _listenToLocationUpdates() {
26     _databaseReference.child('location').onValue.listen((event) {
27       final data = event.snapshot.value as Map<dynamic, dynamic>;
28       final lat = data['LAT'] as double?;
29       final lng = data['LNG'] as double?;
30
31       setState(() {
32         _lat = lat;
33         _lng = lng;
34       });
35     });
36   }
37
38   @override
39   Widget build(BuildContext context) {
40     return Scaffold(
41       appBar: AppBar(
42         title: const Text(
43           "Location Details",
44           style: TextStyle(
45             color: Colors.white,
46           ), // TextStyle
47         ), // Text
48         centerTitle: true,
49         backgroundColor: Colors.deepOrange,
50       ), // AppBar
51       body: Column(
52         children: [
53           if (_lat != null && _lng != null)
54             Padding(
55               padding: const EdgeInsets.all(8.0),
56               child: Text(
57                 'Latitude: $_lat, Longitude: $_lng',
58                 style: TextStyle(fontSize: 20),
59               ), // Text
60             ), // Padding
61           Expanded(
62             child: GoogleMap(
63               initialCameraPosition: CameraPosition(
64                 target: LatLng(_lat ?? 0.0, _lng ?? 0.0),
65                 zoom: 15,
66               ), // CameraPosition
67               onMapCreated: (controller) {
68                 setState(() {
69                   //
70                 });
71               },
72               markers: {
73                 Marker(
74                   markerId: MarkerId('currentLocation'),
75                   position: LatLng(_lat ?? 0.0, _lng ?? 0.0),
76                 ), // Marker
77             },
78           ),
79         ],
80       ),
81     );
82   }
83 }

```

Figure 6.3.2.1: fetch real time data from the realtime database



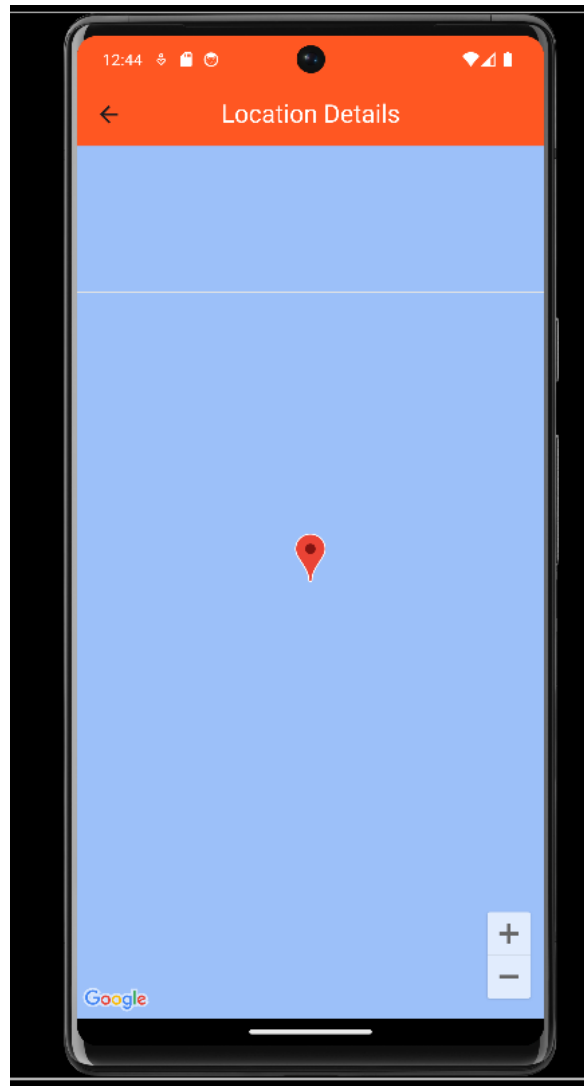
**Visualization:**

Figure 6.3.2.2: the location at latitude 0 & longitude 0



## 7. Robot Control

Controlling the movements of the rescue robot involves a seamless integration of software and hardware components, orchestrated through HTTP requests. In the Flutter app, users can command the robot to move forward, backward, and perform other maneuvers by clicking buttons. These commands are transmitted to a REST API hosted on the Raspberry Pi 3, which then relays the instructions to an Arduino UNO. The Arduino, equipped with the necessary circuitry, manages the DC motors for locomotion and a servo motor for camera orientation, enabling precise and responsive control of the robot's movements.



Figure 7: Robot Control

## 7.1 Components

### Arduino Uno

Arduino UNO is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with an AC-to-DC adapter or battery to get started.

### L239D

The L293D is a 16-pin Motor Driver IC which can control a set of two DC motors simultaneously in any direction. The L293D is designed to provide bidirectional drive currents of up to 600 mA (per channel) at voltages from 4.5 V to 36 V (at pin 8!). You can use it to control small dc motors - toy motors.

### SERVO-MOTOR

A servo motor is defined as an electric motor that provides precise control of angular or linear position, speed, and torque using a feedback loop system.

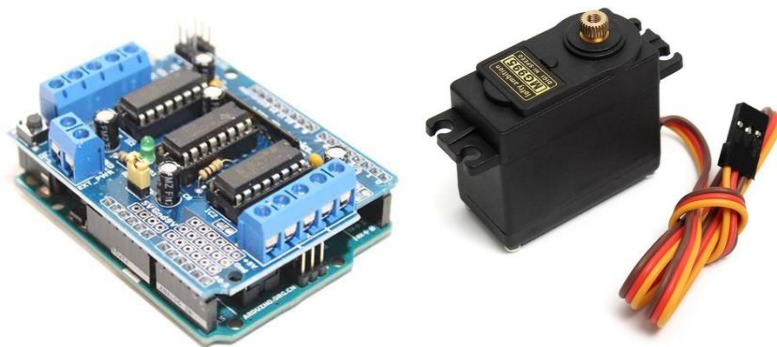


Figure 7.1: Arduino UNO +L293D / Servo Motor

## 7.2 HTTP requests with Dart

**Http protocol:** It is a client-server protocol. Clients and servers communicate by exchanging individual messages. The messages sent by the client, usually a web browser, are called requests and the messages sent by the server as an answer are called responses

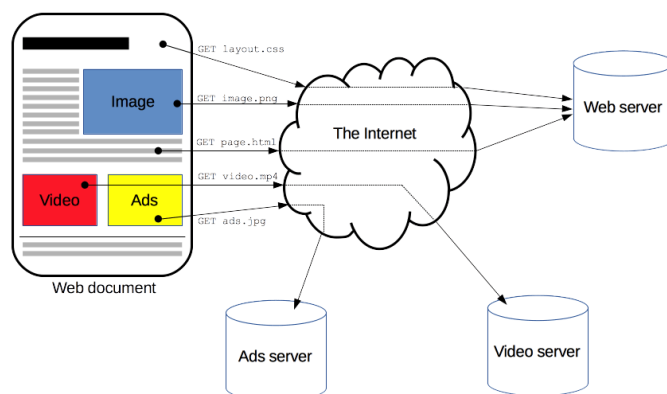


Figure 7.2.1: Http Protocol

Dart code using http package:

```
final String serverUrl = 'http://192.168.0.105:5000';
```

```

Future<void> moveForward() async {
  try {
    final response = await http.post(Uri.parse('$serverUrl/forward'));
    print(response.body);
  } catch (error) {
    print('Error: $error');
  }
}

```

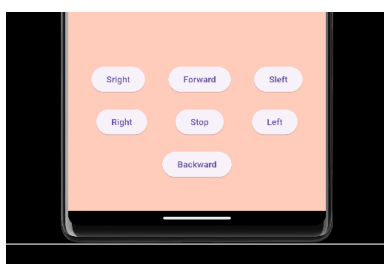


Figure 7.2.2: Send commands using http protocol

## 7.3 RESTFUL API

**Definition:** Restful API is an interface that two computer systems use to exchange information securely over the internet. Restful APIs support this information exchange because they follow secure, reliable, and efficient software communication standards.

### Restful API with flask

```

1 from flask import Flask
2 app = Flask(__name__)
3
4
5 import serial
6
7 # Define the serial port and baud rate
8 SERIAL_PORT = '/dev/ttyACM0' # This may vary depending on your Raspberry Pi configuration
9 BAUD_RATE = 9600
10
11 # Initialize serial connection
12 ser = serial.Serial(SERIAL_PORT, BAUD_RATE)
13
14 # Function to send forward command to Arduino
15 def send_forward_command():
16     try:
17         ser.write(b'F') # Send 'F' to Arduino (assuming 'F' is the command for forward movement)
18         print('Forward command sent to Arduino')
19         return 'Forward command sent to Arduino'
20     except Exception as e:
21         print(f'Error sending command to Arduino: {e}')
22         return f'Error sending command to Arduino: {e}'
23
24 @app.route('/forward', methods=['POST'])
25 def forward():
26     return send_forward_command()
27
28 # Function to send backward command to Arduino
29 def send_backward_command():
30     try:
31         ser.write(b'B') # Send 'B' to Arduino (assuming 'B' is the command for backward movement)
32         print('Backward command sent to Arduino')
33         return 'Backward command sent to Arduino'
34     except Exception as e:
35         print(f'Error sending command to Arduino: {e}')
36         return f'Error sending command to Arduino: {e}'
37
38 @app.route('/backward', methods=['POST'])
39 def backward():
40     return send_backward_command()
41
42 # Function to send left servo command to Arduino
43 def send_left_servo():
44     try:
45         ser.write(b'L') # Send 'L' to Arduino (assuming 'L' is the command for left servo movement)
46         print('Left servo command sent to Arduino')
47         return 'Left servo command sent to Arduino'
48     except Exception as e:
49         print(f'Error sending command to Arduino: {e}')
50         return f'Error sending command to Arduino: {e}'
51
52 @app.route('/servoLeft', methods=['POST'])
53 def servoLeft():
54     return send_left_servo()
55
56 # Function to send right servo command to Arduino
57 def send_right_servo():
58     try:
59         ser.write(b'R') # Send 'R' to Arduino (assuming 'R' is the command for right servo movement)
60         print('Right servo command sent to Arduino')
61         return 'Right servo command sent to Arduino'
62     except Exception as e:
63         print(f'Error sending command to Arduino: {e}')
64         return f'Error sending command to Arduino: {e}'
65
66 @app.route('/servoRight', methods=['POST'])
67 def servoRight():
68     return send_right_servo()
69
70 if __name__ == '__main__':
71     app.run(host='0.0.0.0', port=5000, debug=True)

```

Figure 7.3: RESTFUL API

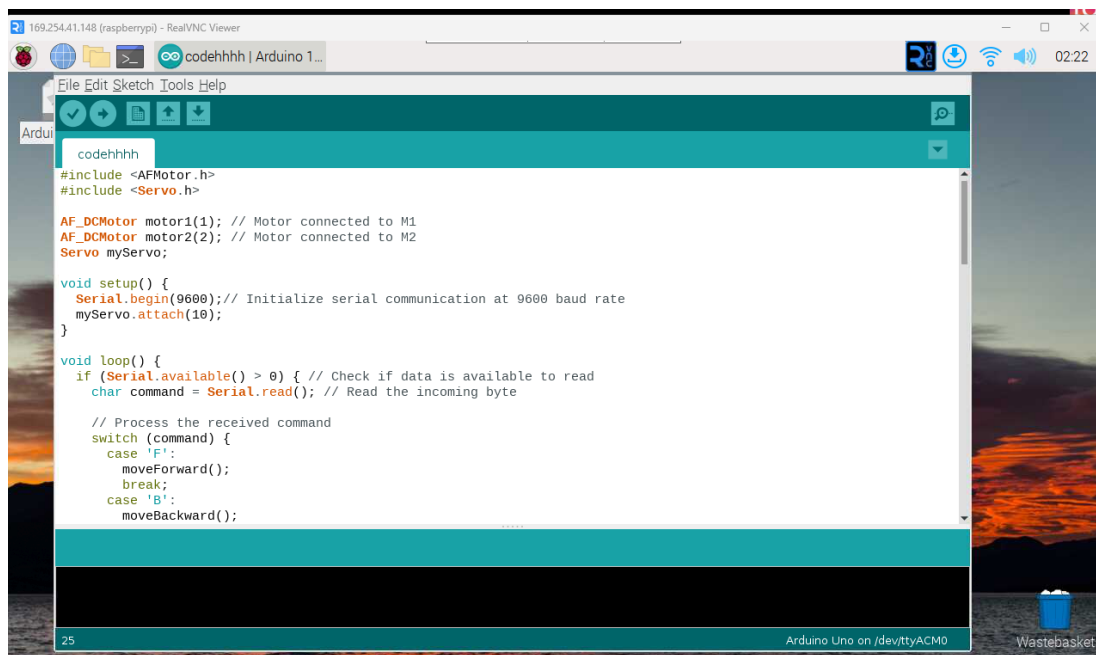
## 7.4 Control robot's movements with Arduino UNO

### Libraries:

**AFMotor.h:** a library designed for controlling DC motors, stepper motors, and servo motors using the Adafruit Motor Shield. This library provides an easy-to-use interface to control the speed and direction of these motors.

**Servo.h:** a standard library for controlling servo motors. This library simplifies the process of interfacing with servo motors by providing functions to easily set the position of the servo's shaft.

### Arduino Code:



```
codehnhh
#include <AFMotor.h>
#include <Servo.h>

AF_DCMotor motor1(1); // Motor connected to M1
AF_DCMotor motor2(2); // Motor connected to M2
Servo myServo;

void setup() {
  Serial.begin(9600); // Initialize serial communication at 9600 baud rate
  myServo.attach(10);
}

void loop() {
  if (Serial.available() > 0) { // Check if data is available to read
    char command = Serial.read(); // Read the incoming byte

    // Process the received command
    switch (command) {
      case 'F':
        moveForward();
        break;
      case 'B':
        moveBackward();
    }
  }
}
```

Figure 7.4: Arduino code in raspberry pi 3

## 7.5 Final Prototype

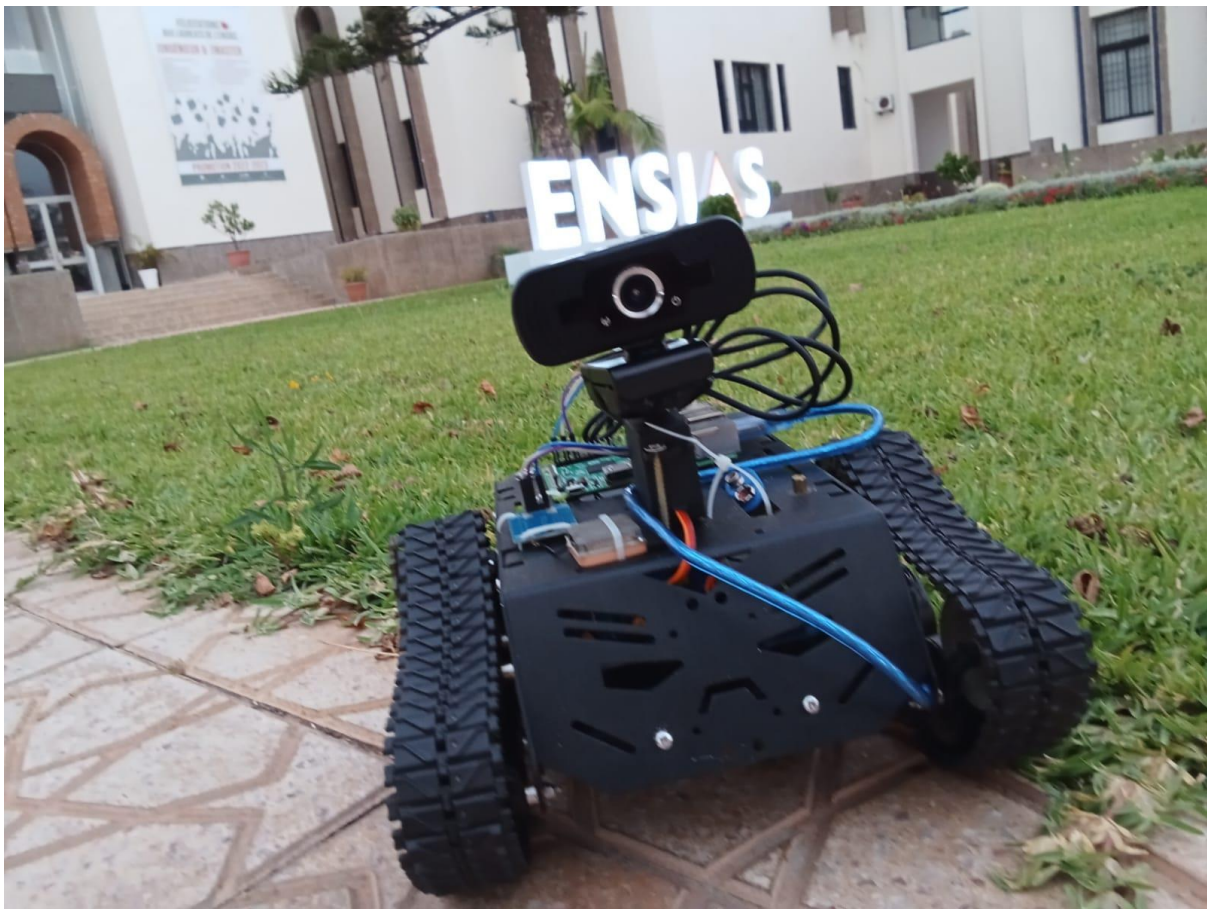


Figure 7.5: Final Prototype



## BIBLIOGRAPHY

- [1] Medium, VAISHAK: Python for IOT: connecting and controlling Devices, Sept 20, 2023  
[Python for IoT: Connecting and Controlling Devices | by VAISHAK | Medium](#)
- [2] How to build run Mjpg-streamer on the Raspberry pi. [How to build and run MJPG-Streamer on the Raspberry Pi - miguelgrinberg.com](#)
- [3] Wei Liu (University of North Carolina), Scott Reed (University of Michigan) , Christian Szegedy (Google inc), Going deeper with convolutions. 17 Sept 2014
- [4] Zoumana Kelta (Senior Data Scientist at international Finance Corporation; Co-founder at ETP4Africa) Yolo object Detection at DataCamp. [YOLO Object Detection Explained: A Beginner's Guide | DataCamp](#)
- [5] You only Look Once: Unified, Real-Time Object Detection Original Paper by Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi from University of Washington, Allen Institute for AI & Facebook AI research. 9 May 2016
- [6] Integrating Flask and Flutter apps from LogRocket. [Integrating Flask and Flutter apps - LogRocket Blog](#)
- [7] Use Neo 6M GPS Module with Raspberry pi and python. [Use Neo 6M GPS Module with Raspberry Pi and Python \(sparklers-the-makers.github.io\)](#)
- [8] An overview of HTTP in mdn web docs. [An overview of HTTP - HTTP | MDN \(mozilla.org\)](#)