Internship Report

Report on the internship carried out from 12/06/2023 to 01/09/2023

Internship tutor: Kim Khoa Nguyen

Student: Sébastien Doyez, student in 4th year Microelectronics and Automation, Polytech Montpellier

Thanks:

Firstly, I want to thank my tutor Kim Khoa Nguyen, and the NG AI laboratory that is part of ETS for their confidence and for believing in my potential.

This internship was beneficial for me, because I was able to work independently, which allowed me to get a real simulation of the engineering profession!

I would also like to thank the teaching team of Polytech Montpellier. I don't forget my parents either, for their unfailing support, throughout my years of schooling.

Introduction:

From June 12, 2023 to September 3, 2023, I did an internship within the ETS' NG-IA laboratory, funded by VMware, located at 1100, street Notre-Dame Ouest, in Montréal (QC). I was able to focus on the implementation of Machine Learning in a robotic project. It was an opportunity for me to deepen my knowledge of robotics, to take on competence in the field of artificial intelligence, particularly machine learning, and to face the reality of the world of work. It also made me understand that machine learning is the future of the industry and its implementation will open up a new horizon of possibilities in terms of robotics projects. The objective of the course was to perform a rescue scenario, when the aerial drone would detect the person, it determines the person's coordinates, and sends them to the rovers, so that the rovers could assist the person.

Summary:

I) Presentation of the companies:

My internship was done at the NGAI lab, funded by VMware, located at ETS Montreal, and is in partnership with the Mitacs Globalink research program. Foremost, let's start with a quick presentation of these partners.

a) ETS:

The ETS, also known as Ecole de Technologie Supérieur, was founded in 1974. It is a public research school in Montreal (QC), Canada. This school is renowned for its strong focus on applied research and innovation. ETS's research centers focus on diverse areas such as advanced manufacturing, sustainable development, aerospace, or robotics. The school's robotics research centers focus on various aspects of robotics, such as autonomous systems, humanoid robots, industrial automation…

https://www.imtl.org/image/big/2014_IMG_7168.jpg

b) VMware:

VMware is a global leader in cloud infrastructure and virtualization software. It has played a pivotal role in revolutionizing the IT industry by introducing innovative solutions that transform the way organizations manage and utilize their technology resources. VMware was founded in 1998, now it is based in Palo Alto, California.

In 2020, the VMware's turnover was of 11.8 billion dollars, the company had more than 34 000 employees. It is present in more than 50 countries.

Based on the principles of energy efficiency, reliability and flexibility, VMware offers products and services focused on server virtualization, cloud computing and data management. It strives to enable businesses to create, deploy and manage their IT environments efficiently and scalably.

This partnership was established through a constructive dialogue: they identified some potentials areas for collaboration, such as the development of new virtualization technology or implementation of machine learning concept. The VMware's industrial chair in Edge IA wants to develop a new generation of smart technologies to improve the life of Canadians: the goal of our project was to improve the first prototype of a system with drones and rovers with object detection and communication, by adding some AI and machine learning feature...

c) Mitacs:

Mitacs is a non-profit organization, in partnership with the Canadian government, which supported internships. It operates six main programs: Accelerate, Elevate, Globalink, Canadian Science Policy Fellowship, Business Strategy Internship and Entrepreneur International.

This internship was carried out as part of the Globalink branch of Mitacs.

In October 2022, VMware and Mitacs became partners and created the NGIA laboratory, at ETS, in order to enable advances in cloud, network and artificial intelligence, all with the aim of helping the development of 6G.

II) The internship:

a) Presentations of the drone and rovers

On this project, we have at our disposal a drone and 2 rovers of the brand DJI.

To meet our needs, we need to add functions to the robots via the Python programming language already built into them.

1) DJI's Mavic Air 2:

This drone released on April 27, 2020, it is a quadri-rotor, has a camera located on a nacelle at the front of the aircraft.

Thanks to this, we will be able to make a machine learning recognition system in order to make it detect a target.

A perceived problem, being a public drone, it is not possible to modify its internal functions to add or remove it. We will therefore have to go through another solution to meet our needs.

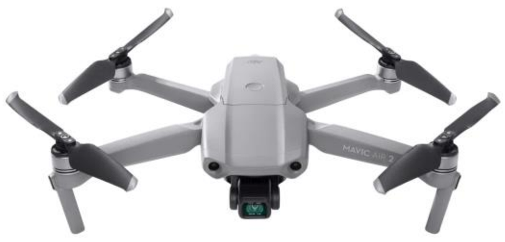


Figure 1: drone Mavic air 2

2) DJI's Robomaster S1:

This rover has a gimbal and a chassis, both independent, that can be coupled to perform actions according to our needs.

It is equipped with a camera that can be used to make perception of objects or movements, using various functions based on the principle of AI.

It is a responsive rover and has "mecanum" wheels, which offers great freedom of movement.

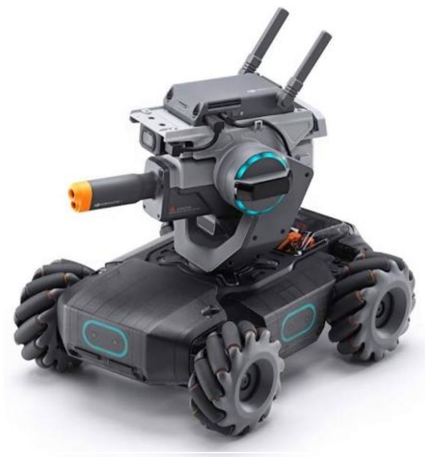


Figure 2: rover Robomaster S1

3) DJI Robomaster EP Core:

Unlike the Robomaster S1, it does not have a carrycot. It is equipped with an arm, with a clamp at the end, whose movements are guided by the chassis.

It can be used to catch and move objects in our work area.

A camera is also present on the arm, which allows us to visualize the movements of the pliers with precision.

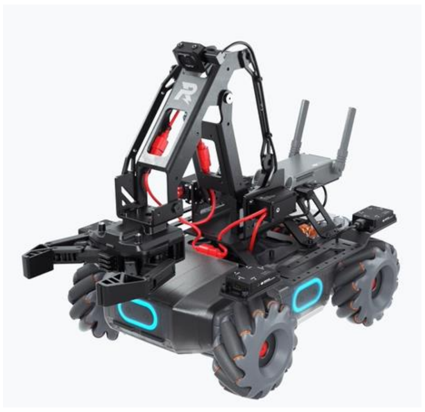
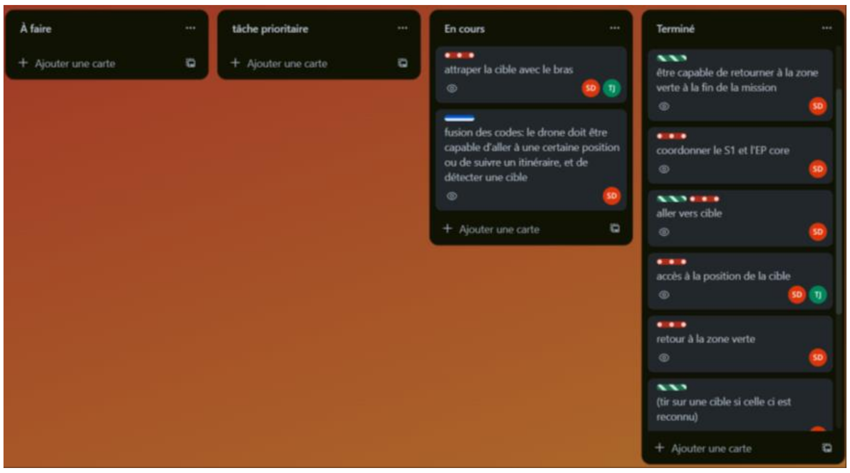


Figure 3: rover Robomaster EP Core

An application has been created by DJI, to allow coding on these rovers, but there is also the Robomaster SDK, which allows more possibilities of movement. Unfortunately, it only works on EP Core rover.

b) Organization of the internship:

As this internship was quite long, so we needed to be well organized. That is why we set up a Trello and a project follow-up that we had to send to our tutor every weekend. This project monitoring not only allowed us to see the progress of the work, but also allowed us to document the problems we had encountered and the solutions we had found.



À faire
+ Ajouter une carte

tâche prioritaire
+ Ajouter une carte

En cours
- attraper la cible avec le bras (SD, TJ)
- fusion des codes: le drone doit être capable d'aller à une certaine position ou de suivre un itinéraire, et de détecter une cible (SD)
+ Ajouter une carte

Terminé
- être capable de retourner à la zone verte à la fin de la mission (SD)
- coordonner le S1 et l'EP core (SD)
- aller vers cible (SD)
- accès à la position de la cible (SD, TJ)
- retour à la zone verte (SD)
- (tir sur une cible si celle ci est reconnu)
+ Ajouter une carte

Figure 4: Screenshot of our Trello

c) Creation of a model for Machine Learning:

During the creation of the model, we took inspiration from the book "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers" written by Pete Warden and Daniel Situnayake. It is a book with a lot of examples from the creation of a model to its deployment in a microcontroller.

1) How does it work?

In order to create artificial intelligence based on machine learning, you first need to create a model. It is a mathematical or statistical representation of a system, designed to capture the relationship between inputs and outputs. In others words, it learns from sample data to understand the link between the inputs and the predictions we want to have from the model.

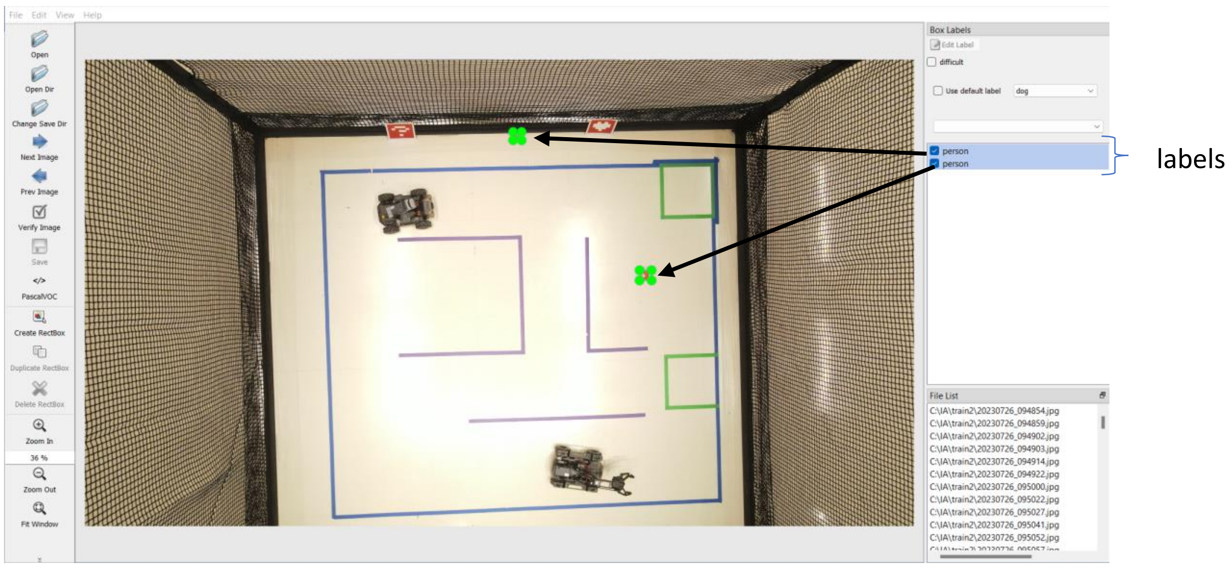
For the creation of a model, we need to choose an architecture, it is the way of how the different parts of the models are structured. It determined how the data is processed and how the information is propagated through the model. This choice must be made in order to respect the constraints of our project (maximum time processing…).

For the training of the model, we use jupyter notebook, which can run on Google collab: it is a free online platform, which can use a GPU, and it is often used for machine learning project.

When we have a model, we need to deploy it in a microcontroller, and for doing that, we will use the TensorFlow libraries. It takes the model as an input, invokes it, and does some prediction with the input data.

2) The creation of our model:

For this project, we need an object detection: the target are small figurines. We can split the code into 3 sub-parts: creation, training, and evaluation. So, we will need data in order to successfully generate the model, we can 200 pictures of our objects, and with the software « LabelIMG », we can put label on areas containing persons.



```
C:\IA\train2\20230726_094854.jpg
C:\IA\train2\20230726_094859.jpg
C:\IA\train2\20230726_094902.jpg
C:\IA\train2\20230726_094903.jpg
C:\IA\train2\20230726_094914.jpg
C:\IA\train2\20230726_094922.jpg
C:\IA\train2\20230726_095000.jpg
C:\IA\train2\20230726_095022.jpg
C:\IA\train2\20230726_095027.jpg
C:\IA\train2\20230726_095041.jpg
C:\IA\train2\20230726_095052.jpg
```

Figure 4: Screenshot from the software LabelIMG while labelling our images

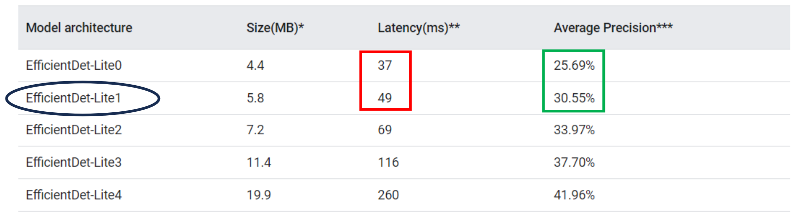
LabelIMG create file .xml (it contains the information like the name, the position in pixels) with the help of the labels that have been placed. When we have all the images ready, we split this into 2 groups: one for the training (60% of the photography) and one for evaluation (40%).

Now let us look at how the notebook works: firstly, we need to set the environment: we need to install tflite-model-maker, it is a library of tensorflow, which aims to simplify the creation of models, and we need to import some libraries. After that, we need to import the two groups of images that we created earlier.

Now, the programming environment is ready, we can create our model: with tflite-model-maker, it only takes 2 lines of code:

```
from tflite_model_maker import object_detector
model = object_detector.create(train_data, model_spec=model_spec.get('efficientdet_lite1'))
```

The train_data is the group with 60% of the photography we take earlier. The model_spec is the architecture we need:

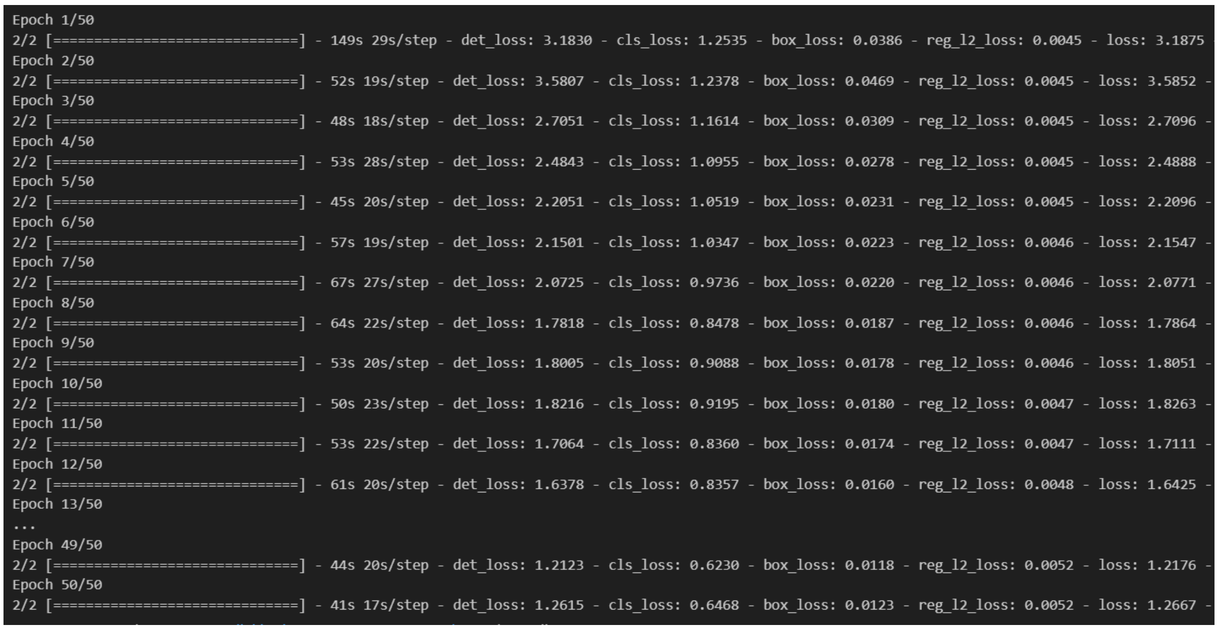


Model architecture	Size(MB)*	Latency(ms)**	Average Precision***
EfficientDet-Lite0	4.4	37	25.69%
EfficientDet-Lite1	5.8	49	30.55%
EfficientDet-Lite2	7.2	69	33.97%
EfficientDet-Lite3	11.4	116	37.70%
EfficientDet-Lite4	19.9	260	41.96%

Figure 5: table grouping the characteristics of the different architectures

There are a lot of models already available, but we must choose the one that best suits our project: at the beginning we turned to the EfficientDet-Lite0 because it was the fastest one. At the beginning of the project, we wanted to study the streaming stream in its entirety, however, in the rest of the project, we realized that it would not be the best choice, since live data processing is heavy in terms of resources used. For the rest, we chose to take only one photo every second and then process it, because our drone performs only a hover. So, we could take a longer but also more accurate model: EfficientDet-Lite1.

This is the output of the function object_detector.create(…):



```
Epoch 1/50
2/2 [==============================] - 149s 29s/step - det_loss: 3.1830 - cls_loss: 1.2535 - box_loss: 0.0386 - reg_l2_loss: 0.0045 - loss: 3.1875
Epoch 2/50
2/2 [==============================] - 52s 19s/step - det_loss: 3.5807 - cls_loss: 1.2378 - box_loss: 0.0469 - reg_l2_loss: 0.0045 - loss: 3.5852 -
Epoch 3/50
2/2 [==============================] - 48s 18s/step - det_loss: 2.7051 - cls_loss: 1.1614 - box_loss: 0.0309 - reg_l2_loss: 0.0045 - loss: 2.7096 -
Epoch 4/50
2/2 [==============================] - 53s 28s/step - det_loss: 2.4843 - cls_loss: 1.0955 - box_loss: 0.0278 - reg_l2_loss: 0.0045 - loss: 2.4888 -
Epoch 5/50
2/2 [==============================] - 45s 20s/step - det_loss: 2.2051 - cls_loss: 1.0519 - box_loss: 0.0231 - reg_l2_loss: 0.0045 - loss: 2.2096 -
Epoch 6/50
2/2 [==============================] - 57s 19s/step - det_loss: 2.1501 - cls_loss: 1.0347 - box_loss: 0.0223 - reg_l2_loss: 0.0046 - loss: 2.1547 -
Epoch 7/50
2/2 [==============================] - 67s 27s/step - det_loss: 2.0725 - cls_loss: 0.9736 - box_loss: 0.0220 - reg_l2_loss: 0.0046 - loss: 2.0771 -
Epoch 8/50
2/2 [==============================] - 64s 22s/step - det_loss: 1.7818 - cls_loss: 0.8478 - box_loss: 0.0187 - reg_l2_loss: 0.0046 - loss: 1.7864 -
Epoch 9/50
2/2 [==============================] - 53s 20s/step - det_loss: 1.8005 - cls_loss: 0.9088 - box_loss: 0.0178 - reg_l2_loss: 0.0046 - loss: 1.8051 -
Epoch 10/50
2/2 [==============================] - 50s 23s/step - det_loss: 1.8216 - cls_loss: 0.9195 - box_loss: 0.0180 - reg_l2_loss: 0.0047 - loss: 1.8263 -
Epoch 11/50
2/2 [==============================] - 53s 22s/step - det_loss: 1.7064 - cls_loss: 0.8360 - box_loss: 0.0174 - reg_l2_loss: 0.0047 - loss: 1.7111 -
Epoch 12/50
2/2 [==============================] - 61s 20s/step - det_loss: 1.6378 - cls_loss: 0.8357 - box_loss: 0.0160 - reg_l2_loss: 0.0048 - loss: 1.6425 -
Epoch 13/50
...
Epoch 49/50
2/2 [==============================] - 44s 20s/step - det_loss: 1.2123 - cls_loss: 0.6230 - box_loss: 0.0118 - reg_l2_loss: 0.0052 - loss: 1.2176 -
Epoch 50/50
2/2 [==============================] - 41s 17s/step - det_loss: 1.2615 - cls_loss: 0.6468 - box_loss: 0.0123 - reg_l2_loss: 0.0052 - loss: 1.2667 -
```

Figure 6: Screenshot of model creation output

The output contains information like: det_loss, cls_loss, box_loss, reg_l2_loss, loss, learning_rate…

An epoch represents a complete iteration of all the drive data through the model. The det_loss evaluates the quality of the prediction relative to the actual location given by the labels. This is usually a combination of losses to assess the location and classification of detected objects. The cls_loss is the accuracy of the prediction. The box_loss measures how close the predicted coordinates of the enveloping boxes are to the coordinates of the actual enveloping boxes

When the model is created, we need to train it with new data: we will use our “train_data” and for the validation we will use the “ test_data”.

```
model.train(epochs=10, batch_size=8, train_data=train_data, validation_data=test_data)
```

The output of this function looks like the same as the previous function.

Now our model is ready, we can evaluate and export it:

```
result = model.evaluate(test_data)
# Show the result of the evaluation
print(result)
model.export(export_dir='C:/IA',tflite_filename='ML_robmaster_EP_core.tflite')
```

The evaluation of our model gives us so clue on how well does our model works. Unfortunately, the photos taken by the Mavic Air 2 drones are not of very good quality, and since it has to fly high enough, the accuracy decreases. However, our model is still accurate enough to be able to detect objects quite easily.

3) The deployment of the model:

For the microcontroller, we chose a raspberry: we needed a WIFI connection and we need to install a NGINX server for receiving video streams. Now, we need to retrieve the stream: the Mavic Air 2 can, with the DJI Fly application, make a video stream transfer via RTMP protocol, but for that we need to be connect to the same WIFI, so we can use a router or a photo with the mobile access point activated. The URL for the RTMP was: " rtmp://+ raspberry pi 's IP address + /live" .

The first function we need was the capture_picture (…): it takes picture every second (we can configure this with the variable interval_sec) and saves it in the output folder. We put the detection function inside because if an object is detected, there are no need to take pictures after that. We will use the content of the folder in the calculation of the coordinate. In the first part of this function, we check if there is an output folder.

```
if not os.path.exists(output_folder):
        os.makedirs(output_folder)
```

After that, while the detection does not identify an object and we do not have taken the maximum number of photography, we take a picture, save it and give to the detection function as an input:

Extracted from the capture_picture(…) function

```
while photo_count < num_photos and verif != 2 :
        cap = cv2.VideoCapture(url)
        OK = check_rtmp_stream(cap)
        if OK != 1:
            # read the stream and take a picture
            ret, frame = cap.read()
            if ret:
                # Generation of a name for the picture
                timestamp = int(time.time())
                photo_filename = os.path.join(output_folder, f"photo_{timestamp}.jpg")

                # Save the picture
                cv2.imwrite(photo_filename, frame)
                print(f"Picture {photo_count + 1} saved: {photo_filename}")
                photo_count += 1
                verif = detection(photo_filename,interpreter,input_details, output_details, height,
width, float_input,output_folder, output_folder_img)

                # Save the picture
                cv2.imwrite(photo_filename, frame)
                # Waiting...
                time.sleep(interval_sec)
                # reboot the stream.
                cap.release()
```

In the detection function, we need to open a picture and prepare it for the model. For the implementation of the model, we need to set the tensors (there are generalizations of scalars, vectors, and matrices to higher dimensions), invoke the model and preparate the result.

Extracted from the detection(…) function

```
    image = cv2.imread(path_img)
    imH, imW, _ = image.shape

    # Resize it
    image_resized = cv2.resize(image, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)

    if float_input:
        input_data = (np.float32(input_data) - input_mean) / input_std

    # object detection:
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()
```

When we have the 3 lists (boxes, scores, and classes), we can make draw red rectangles around zones where the model seems to have issued a score higher than min_conf (which is a parameter we set), and we show it to the user. He can validate the prediction, the function will return 2, and the capture_picture function will stop or if the user does not validate, the program will return 0, and another picture will be taken.

```
# preparation of the result
    boxes = interpreter.get_tensor(output_details[1]['index'])[0] # Bounding box coordinates of
detected objects
    classes = interpreter.get_tensor(output_details[3]['index'])[0] # Class index of detected objects
    scores = interpreter.get_tensor(output_details[0]['index'])[0] # Confidence of detected objects

    # for all the detection:
    i= int(np.argmax(scores))
    if ((scores[i] > min_conf) and (scores[i] <= 1.0)):

        # set the coordonnates
        ymin = int(max(1,(boxes[i][0] * imH)))
        xmin = int(max(1,(boxes[i][1] * imW)))
        ymax = int(min(imH,(boxes[i][2] * imH)))
        xmax = int(min(imW,(boxes[i][3] * imW)))
        object_name = labels[int(classes[i])] # Look up object name from "labels" array using class
index
        # Draw a rectangle:
        if object_name == 'person':
            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (0, 10, 255), 2)
        else:
            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10,255, 0), 2)
```

For the validation by the user, we user the function input(), just after a print, that asked if the user agrees with the object detection:

```
if (user == "Y" or user == "y"):
            # Code de Theodore ici, pour avoir les coordonnees
            print("A person has been detected, send data to rovers...")
            return 2
        else:
            os.remove(photo_filename)
```

The block diagram of this code is in the appendix, at pages 25 – 27.

4) Some tests:

Now, we can test the model with our drone: We can see the detection with the red rectangle around the object.

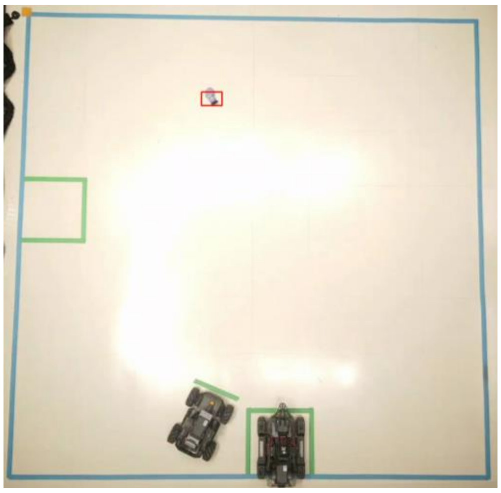


Figure 7: Object detection

As expected, the drone sends its stream video to the NGINX server on the raspberry pi, our program just take picture every second and applies the model. It also draws a red rectangle around the target, and this image will be use after for the determination of the coordinates.

d) The coordinates calculation:

Thanks to the object detection, we are now able to calculate the coordinates of the object in the map. This is how we will do it:

Firstly, we need to prepare the raw image from our RTMP stream. Given that all we're interested in the picture is the inside of the blue lines, we will detect the blue lines and crop the images up to it. After that, we also need to detect the origin: it is a small orange point on our map. According to the concept of perception, we will rotate the picture until the orange point is at the top left of the image.

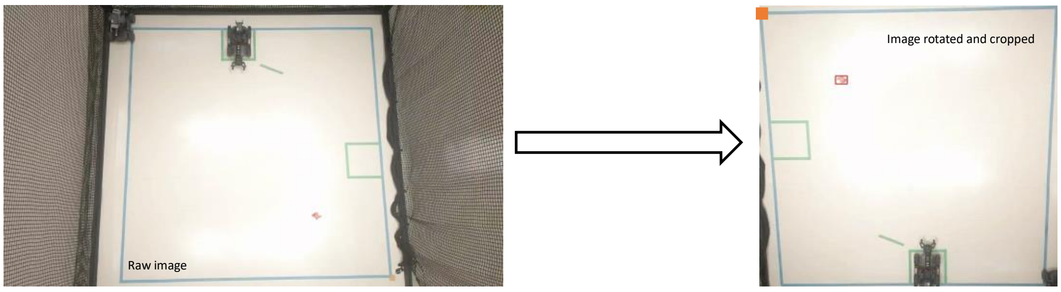


Figure 8: Preparation of the image

Thus, we are ready to calculate the coordinates of our object. The idea is to create a sort of chessboard, the object will have a red rectangle around it, so we need to localize it, and give the position on the checkerboard.

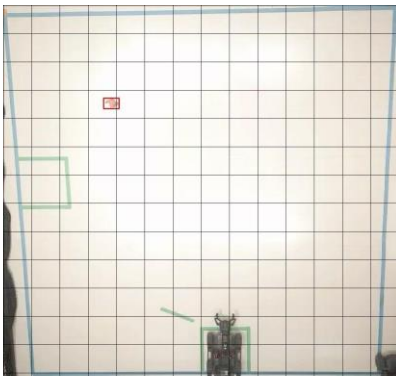


Figure 9: the object detection and the implementation of a chessboard

We will now examine the image, box by box, to detect if there are any red lines contained inside. For the box which contain the object, we will send the coordinates of it to the Robomaster.

e) The communication between the raspberry and the Robomaster EP Core:

Now, we can get the coordinates of the object, but we need to send it to the rovers. The solution we use is a system of client – server. The server will be on the computer, and the client will be the raspberry pi. The software Robomaster cannot use a server, but DJI release an SDK, where we can use this. Unfortunately, the SDK only works with the EP Core, so we will program the rover S1 autonomously. We will use the same WIFI as the Mavic and the raspberry pi.

For the computer: the function start_server() will be use for the creation, connection and the data recovery. We need to create a socket and put it in the listen mode. When there is a connection, we accept it and take in memory the data received. The data will be a string, so we have to convert it into a list of floats. After that, we can close the server

```
def start_server():
    # Server Configuration
    host = ''  # Empty to permite all the connexion
    port = 12345

    # Creation of a socket for the server
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

The socket is the interface of programming which permits to the communication between the client and the server, it allows the connection the send and the receive of data.

```python
    try:
        # Link the socket
        server_socket.bind((host, port))

        # put the socket in bind mode
        server_socket.listen(1)
        print("the server is binding {}:{}".format(host, port))

        # Accept the connexion
        client_socket, client_address = server_socket.accept()
        print("Connection with : {}".format(client_address))

        # Received the data from the Raspberry pi
        data = client_socket.recv(1024)
        print("received : ", data)
        numbers_list = [float(num) for num in data.decode('utf-8').split(',')]

    except Exception as e:
        print("An error happened : {}".format(e))
        return 1

    finally:
        # Close the connexions
        client_socket.close()
        server_socket.close()
        return numbers_list
```

For the raspberry pi: we need to know the IP address of the computer for send the data to his server, and we need to configure the same port for the server and the client. We need to create a socket, which will contain the data. We try to connect to the server, and when we are connected, we can send the data to it. But we need to take care because we cannot send a list but a string, so we need to convert the data and after that, send it. Finally, we can close the client's socket.

```python
def start_client(data):
    # Configuration of the client
    host = '192.168.211.45'  # IP adress of the computer
    port = 12345  # same port as the computer

    # create a socket for the client
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        # connexion client - server
        client_socket.connect((host, port))
        print("Connected to the serveur at {}:{}".format(host, port))

        numbers_list = data
        # Convertir la liste en une chaîne de caractères séparée par des virgules
        message = ','.join(str(num) for num in numbers_list)
        client_socket.sendall(message.encode('utf-8'))
        print("data:  ", data, " sent")

    except Exception as e:
        print("Error : {}".format(e))

    finally:
        # shutdown the connection
        client_socket.close()
```

f) Creation of some functions helpful for our project:

With the SDK, we can create new function for the control of the Robomaster EP Core: we need a function for the moving from one point to another. these points will be defined by coordinates, where the origin will be the orange point located on the map. For doing that, we will need an angle and a distance to travel.

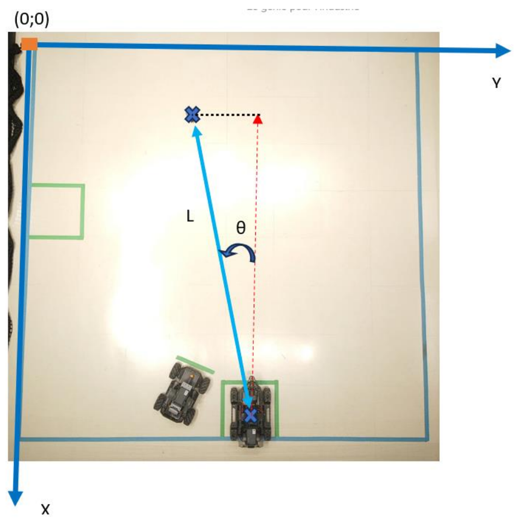


Figure 8: schema representing the distance to travel and the angle

With the Pythagorean formula and some trigonometric properties, we obtain:

$$\vartheta = \arctan2\,(pos1_x - pos2_x\,, pos1_y - pos2_y)$$

$$l = \sqrt{\left(\left(pos[1] - pos_{desire[1]}\right)^2 + \left(pos[0] - pos_{desire[0]}\right)^2\right)}$$

In order to consider all cases and simplify the use of the function, we put the angle of the beginning and the final angle (or "PASS" if we do not care). To do this, we need to do subtractions on the angle: if we need to do a rotation of ϑ, but we already have at the beginning an angle of α, we need to rotate of an angle of $\vartheta - \alpha$. We did the same for the final angle. If we do not care about the final angle, we put "PASS" as an input, and the function will give us ϑ as an output, this will help us for the creation of an autonomous programs.

```python
def go_to_pos_with_angle(pos, pos_desire, actual_angle, angle_d):
    # Angle calculation
    theta = int(math.degrees(math.atan2(pos[0] - pos_desire[0], pos[1] - pos_desire[1])))

    # Keep this angle in memory
    mem_theta = theta
    l = math.sqrt((pos[1] - pos_desire[1])**2 + (pos[0] - pos_desire[0])**2)

    # Rotate the bot
    theta = theta - actual_angle
    chassis_ctrl.move(x=0, y=0, z=theta, z_speed=45).wait_for_completed()

    # Move the robot
    chassis_ctrl.move(x=l, y=0, z=0, xy_speed=0.7).wait_for_completed()

    # Rotate to have angle_d
    if angle_d == "PASS" :
        return mem_theta
    else :
        if (angle_d > 0 and theta > 0) or (angle_d < 0 and theta < 0):
            theta = angle_d - mem_theta
        else:
            theta = -(mem_theta - angle_d)
        chassis_ctrl.move(x=0, y=0, z=theta, z_speed=45).wait_for_completed()
        return 0
```

Now, the rover EP Core can move to a point to another. But if it needs to drop the object into the green zone, it cannot be in the green zone too. We need another function where we give as an input: the actual position, the angle, and the identifier of the target zone. The rover needs to stop before the green zone, and with his arm, put the object into the green sector.

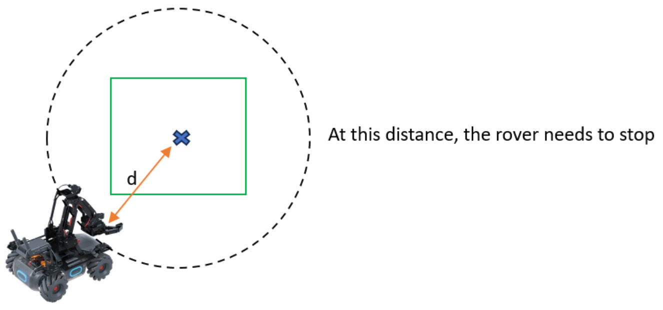


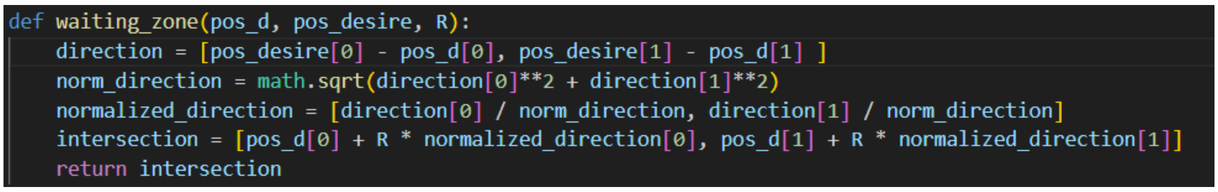
Figure 9: schema representing the stopping area for the rover EP core

The waiting zone is the intersection between the rover trajectory and a circle of radius defined at 40 centimeters. Therefore, we need to calculate the direction of the rover with the actual and the desired position. We also need to normalize this direction to calculate the intersection.

$$intersection = [pos_{des}\,[0] + R.direction_{normalized}[0]\,;\, pos_{des}\,[1] + R.direction_{normalized}[1]]$$

Where: $$direction_{normalized} = \frac{direction}{\sqrt{direction[0]^2 + direction[1]^2}}$$

$$direction = position_{des} - position_{actual}$$



```python
def waiting_zone(pos_d, pos_desire, R):
    direction = [pos_desire[0] - pos_d[0], pos_desire[1] - pos_d[1] ]
    norm_direction = math.sqrt(direction[0]**2 + direction[1]**2)
    normalized_direction = [direction[0] / norm_direction, direction[1] / norm_direction]
    intersection = [pos_d[0] + R * normalized_direction[0], pos_d[1] + R * normalized_direction[1]]
    return intersection
```

As a result, the rover is now able to go to a zone, stop just before it. For the control of the drop, it is a short function where the arm needs to get down, open its grippers and raise the arm again. Moreover, the object capture function will have the same aspect except we close the grippers before open it.

g) Development of an autonomous code for the rover S1:

Although the S1 and EP Core rovers are part of the same DJI product line, the SDK does not work on the Robomaster S1. Instead, we had to find another solution. Only the app provided by DJI Robomaster can program this rover. We have therefore set up an autonomous program for this sector. The goal of the rover S1 is to localize the enemies and to neutralize them with the help of his marbles. Therefore, we do not need the coordinates of the target to rescue. The rover EP Core will be detected by the S1, it is the beginning of the mission. It must go to the center of the map and take a spin with its turret, where it's going to analyze and eliminate all the targets it sees, and then it has to go to the left corner, next to the first zone, which is going to serve as a resting area. Finally, it will continue to analyze for some time in order to eliminate the remaining targets.

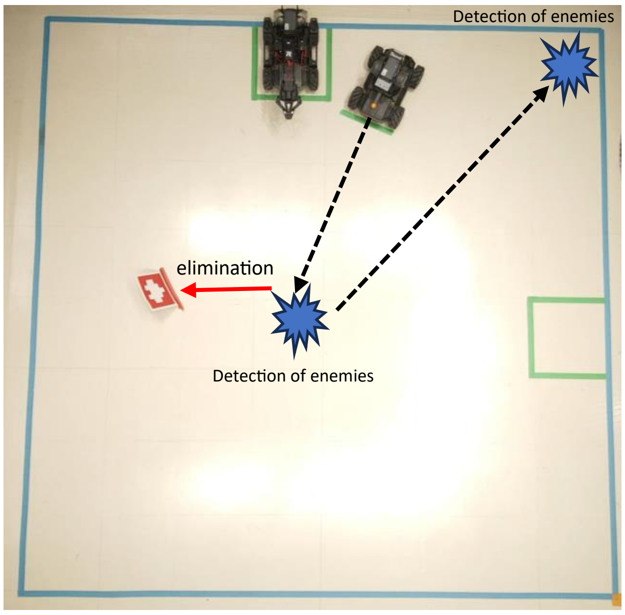


Figure 10: Summary of the mission of rover S1

h) installation of a position correction device:

We are able to detect an object with our drone Mavic air 2, send its position to the rover EP Core, and it is able to go. However, even if the Ep Core is close to the object, it is unable to catch it with its gripper, because it requires a great deal of precision…

For this reason, we added another object detection: but this time, with the camera on the rover. It will be more accurate, because the target is close to it, so the acceptance threshold for this detection will be higher. We will not re-detail the design of the model since we have already done so in part 2). The goal here is to explain how we integrate it, and to criticize its performance.

Once we have the model, the implementation is similar to the first object detection. First and foremost, we need a picture from the rover: In order to avoid excessive memory usage, we only take one photo here. In order not to have to open and close the video stream repeatedly, we only open it once at the beginning of the program and we will only close it at the very end of the hand. The functions “camera_ctrl.start_video_stream“ and “camera_ctrl.stop_video_stream”, extracted from the Robomaster EP Core’s SDK, are the two ones able to open and shut down the stream.

To simplify the code, we created a function “detection”: The first part concerns the deployment of the model, and the preparation of the image.

```
# Setting the interpreter
    input_details  =  interpreter.get_input_details()
    output_details =  interpreter.get_output_details()
    height         =  input_details[0]['shape'][1]
    width          =  input_details[0]['shape'][2]

    print("Analysing the picture...")
    imH, imW, _ = image.shape

    # Resize it
    image_resized = cv2.resize(image, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)
    name = f"D:/IA/IA_photo_robomaster/photo{index}.jpg"
    cv2.imwrite(name, image)
    # Object detection :
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()
```

The function read_cv2_image (strategy="newest") is extract from the SDK; it returns the newest frame as picture. For our model, we need to expand the dimension and resize the picture. After that the image is ready, we can set the tensor and invocate the interpreter.

With the result of the object detection, we need to determinate if there are a target (if the score is higher than the threshold defined at the beginning of the code) and if it is the case, what the rover needs to do to get closer to it.

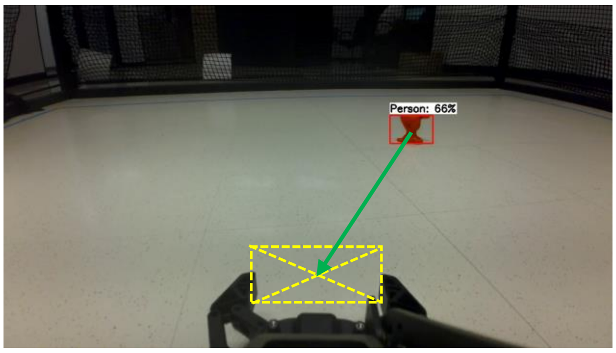
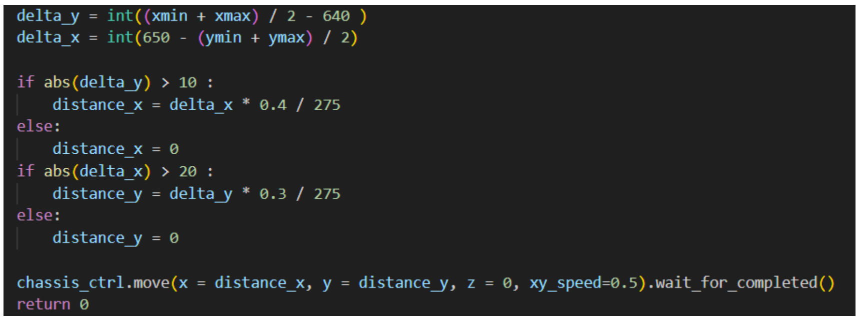


Figure 10: detection of a target

If we look at the score of the detection: it is 66%, we can say it is a good score, the pixel coordinates are satisfying because the target is fully included in the red rectangle.

According to the notions of perception, let see the space in the gripper as a rectangle, and we need to put the center point of the red rectangle to the center of this one. For this reason, we will calculate the deviation in pixels on the horizontal and vertical axis. Once calculated, we convert it into deviations in meters with a cross product.



```
delta_y = int((xmin + xmax) / 2 - 640 )
delta_x = int(650 - (ymin + ymax) / 2)

if abs(delta_y) > 10 :
    distance_x = delta_x * 0.4 / 275
else:
    distance_x = 0
if abs(delta_x) > 20 :
    distance_y = delta_y * 0.3 / 275
else:
    distance_y = 0

chassis_ctrl.move(x = distance_x, y = distance_y, z = 0, xy_speed=0.5).wait_for_completed()
return 0
```

Now the rover EP Core is able to regulate its position to get as close as possible to the object. However, we will integrate his function into the catch_target function in order to simplify the program. The function will bring down the arm of the EP Core so as to orient the camera to be able to see and catch the object and orient the camera. It will then use detection to adjust the position of the rover and finally it will close the clamp and lift the arm of the rover. After that the rover will go to the first green zone in order to drop the object and will go to the second green zone.

Nevertheless, sometimes the camera gives us a picture pixelated. The solution we find was to have the photography checked by the user, if it is too much pixelated, the program needs to be able to take another image. Like the code in the raspberry pi, we will use the input() function:

```
while not check:
        print("Taking a picture")
        image = camera_ctrl.read_cv2_image(strategy="newest")
        # Ask the user if the photo is ok
        # Somestimes the camera does not work very well (it pixelates)
        cv2.imshow("image", image)
        cv2.waitKey(5000)
        cv2.destroyAllWindows()
        print("Is the photo OK for detection?    (Y/N)")
        letter = input()
        if letter == 'y' or letter == 'Y':
            check = True
        else :
            check = False
```

if the object is not centered at the level of the clip, the user can also adjust the position by pressing the 'd' or 'q' keys on the keyboard:

```
# The user will ajust the position:
print("manual setting: press 'd' to go right and 'q' to go left, then press 'f' for finish ")

# While the user does not say it is ready, he can ajust the position:
f = ''
while f != 'f':
        f = input()
        if f == 'd':
                # At the right
                chassis_ctrl.move(x = 0, y = 0.1, z = 0, xy_speed=0.7).wait_for_completed()
                distance_y += 0.1
        else:
                if f == 'q':
                # At the left
                chassis_ctrl.move(x = 0, y = -0.1, z = 0, xy_speed=0.7).wait_for_completed()
                distance_y -= 0.1
```

i) Some improvements possibles:

Now our system of rescue is able to detect an object, catch it, drop it into the first green zone and finish the mission at the second green zone. Here is a summary diagram of the mission carried out:

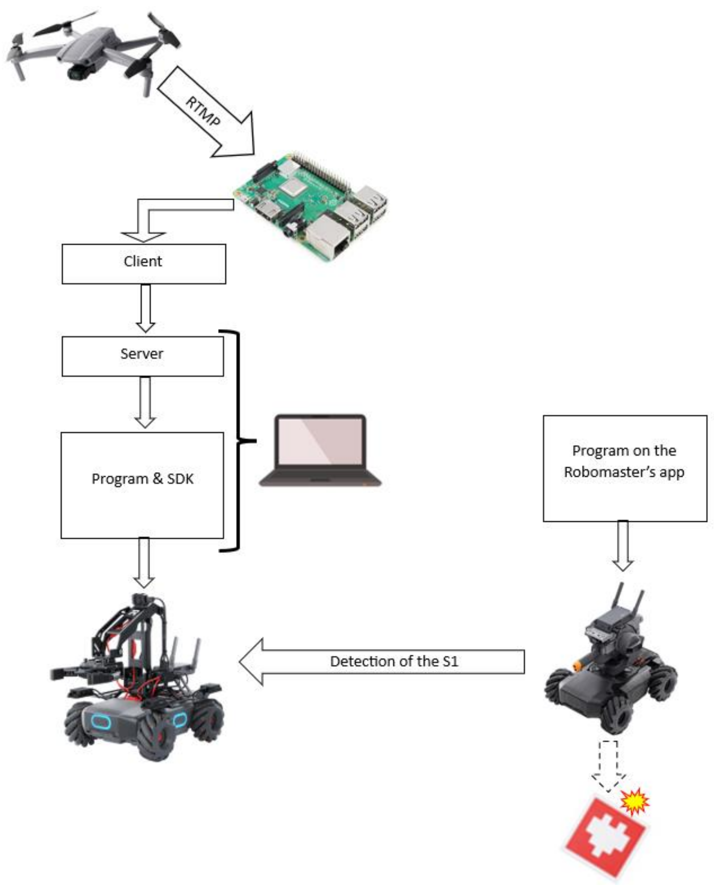


Figure 11: Diagram of system design

In the appendices at the end of this document, there are some block diagrams of the code in order to better understand the architecture of the code. But there are always ways to improve our system:

Even if the rover EP Core is able to catch a target, it is not autonomous because the video stream from the camera on its arm is not stable: sometimes, we received picture pixelated… So, we always need a supervision from the user. This problem can be solved by adding a better camera. The algorithm of object detection for the Robomaster EP Core is slow, we can use the architecture faster than the EfficientDet-lite1, moreover the target is close enough to the camera, so it will be easier to detect than in the drone case. The accurate from the object detection algorithm from the drone is not as good as the one of the Robomaster EP Core, because the picture are far away from the object. The accuracy from the machine learning program from the rover is around 60%, while that of the drone only reaches 30%. Even if we add more photo to train this object detection, we cannot improve it like that. The best way should be to add a better camera, or to do some zoom on the map.

To put it in a nutshell, this internship allowed us to learn about machine learning concepts, this allowed us to better understand the value of machine learning and that they are the future of robotics. The first part was to create a machine learning notebook, according to the book “TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers" written by Pete Warden and Daniel Situnayake. Thanks to it, we built our model for the object detection done with the video stream send by the drone. After that, we create a system client – server to be able to send the coordinates calculated by the raspberry pi. Accordingly, we were able to have the coordinates of the target on the computer where is the server. We created, with the help of the SDK, some simple functions of movement for the rover EP Core. But the coordinates were not accurate enough, so we built another machine learning for the EP Core, and to regulates the problem of pixelated pictures, the user needs to check the pictures before the model processing. The rover S1 can’t use the SDK, so we use the Robomaster’s application to code it. It is more an autonomous program, because it can have access to the coordinates of the target, but his goal is not to go to the coordinates, but to eliminate the enemies.

During my internship, I had the opportunity to gain significant technical skills by diving into the concepts of machine learning and artificial intelligence. I was able to work on real projects where I explored and applied machine learning algorithms to solve real problems. This experience has given me a better understanding of the nuances of machine learning, classification and prediction, strengthening my technical skills in this rapidly expanding field. At the same time, I was able to apply processes that allowed me to better manage my time, in order to be as productive as possible.

These skills that I have acquired and applied will be useful to me in my future professional development. They will allow me to contribute to new artificial intelligence and machine learning projects.

Bibliography:

For the machine learning part:

- "TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers" written by Pete Warden and Daniel Situnayake.

- "Tensorflow 2 Custom Object Detection Model (Google Colab and Local PC)" (Youtube video by Lazy Tech : https://www.youtube.com/watch?v=8ktcGQ-XreQ)

- Tensorflow 's tutorial on the tensorflow model maker: https://www.tensorflow.org/lite/models/modify/model_maker?hl=fr

- "How to Train TensorFlow Lite Object Detection Models Using Google Colab | SSD MobileNet" (Youtube video by Edje Electronics) https://www.youtube.com/watch?v=XZ7FYAMCc4M&t=855s

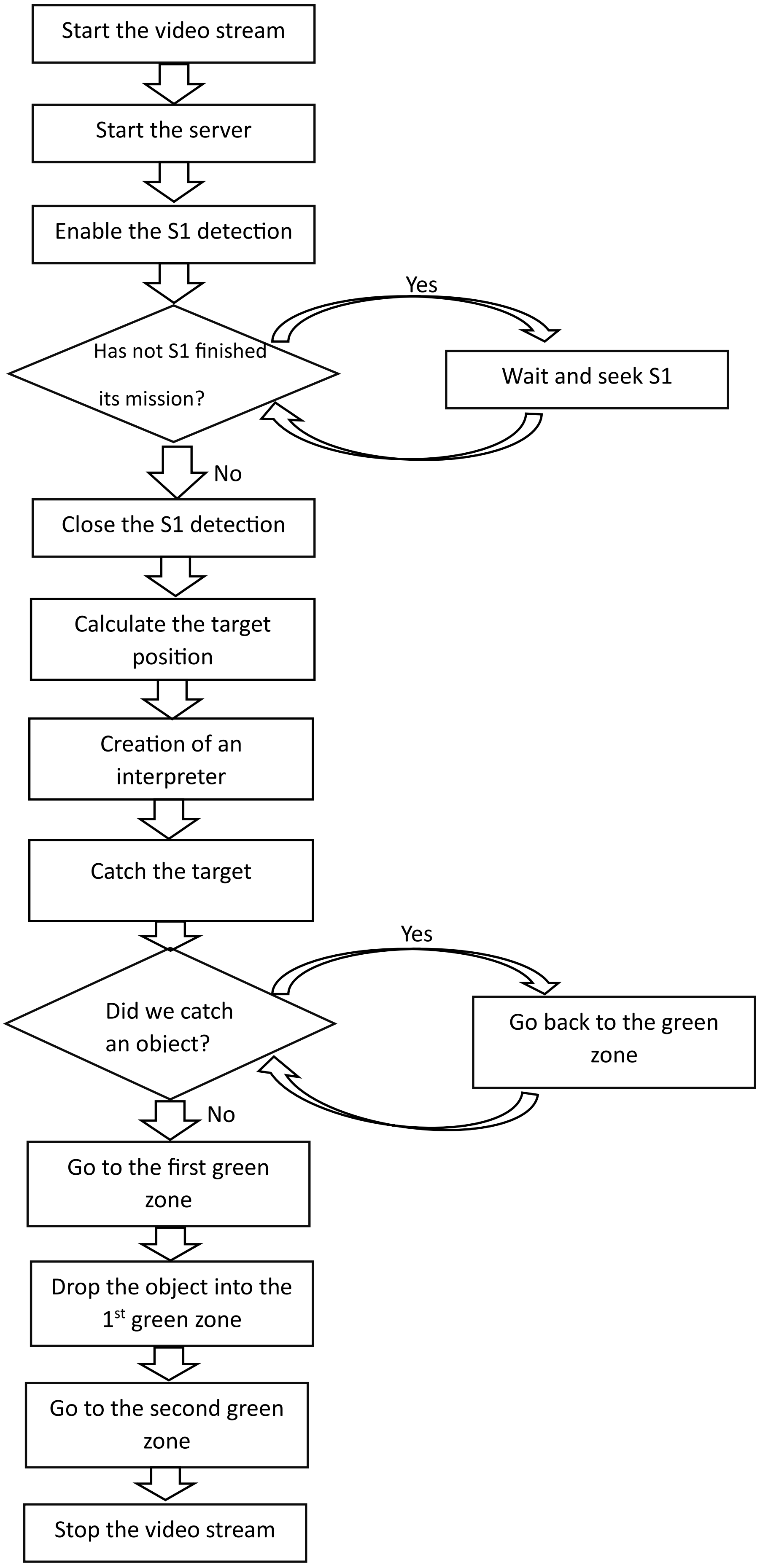
For the programming of the rovers:

- The Robomaster developer guide: https://robomaster-dev.readthedocs.io/en/latest/

- The DJI's forum: https://sdk-forum.dji.net/hc/en-us

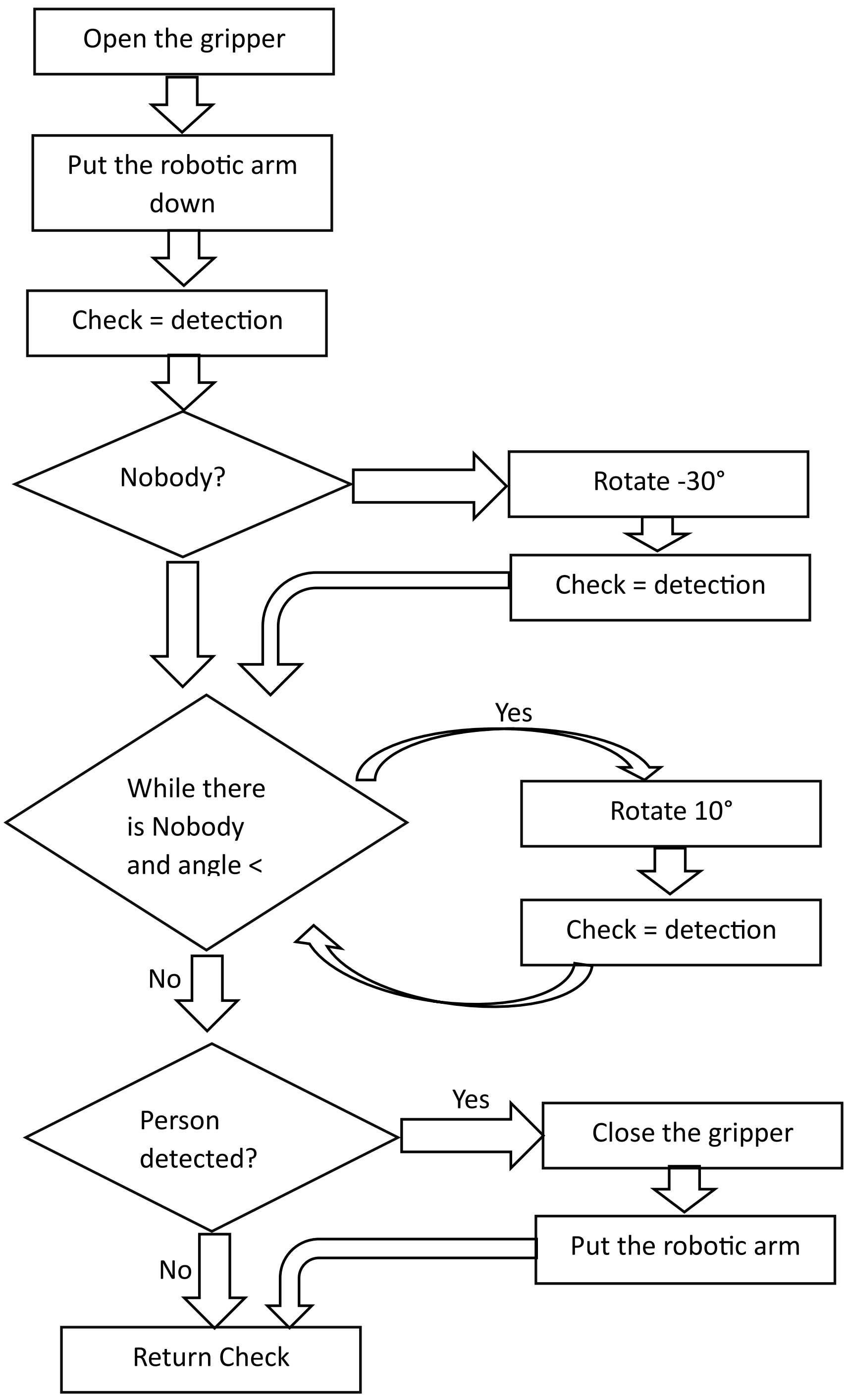
Block diagram of the algorithm:

1) Main of the EP Core's program:

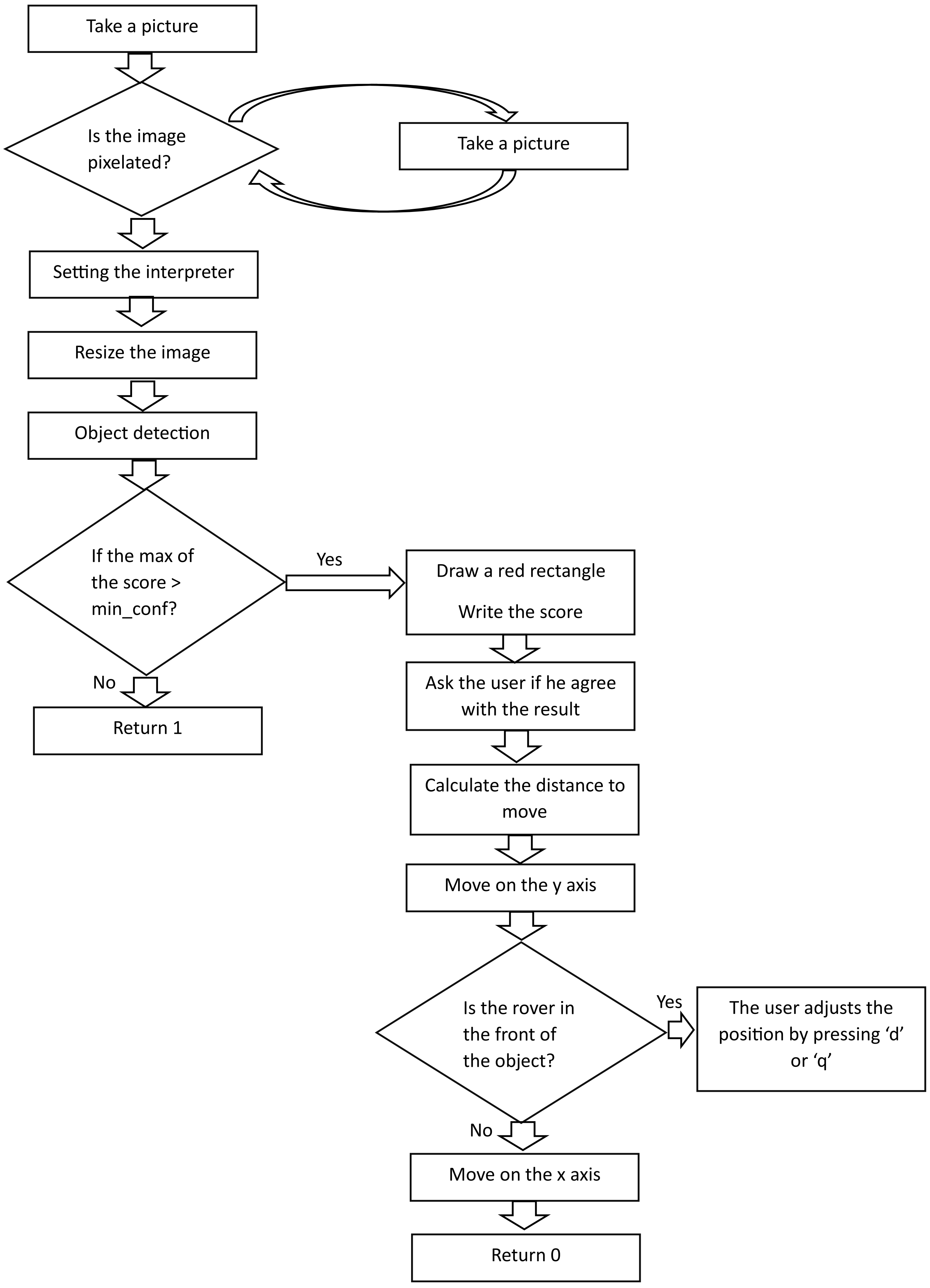

Start the video stream
Start the server
Enable the S1 detection
Has not S1 finished its mission?
Yes
Wait and seek S1
No
Close the S1 detection
Calculate the target position
Creation of an interpreter
Catch the target
Did we catch an object?
Yes
Go back to the green zone
No
Go to the first green zone
Drop the object into the 1st green zone
Go to the second green zone
Stop the video stream

2) Some functions of the EP Core's Program:

a) Catch target:

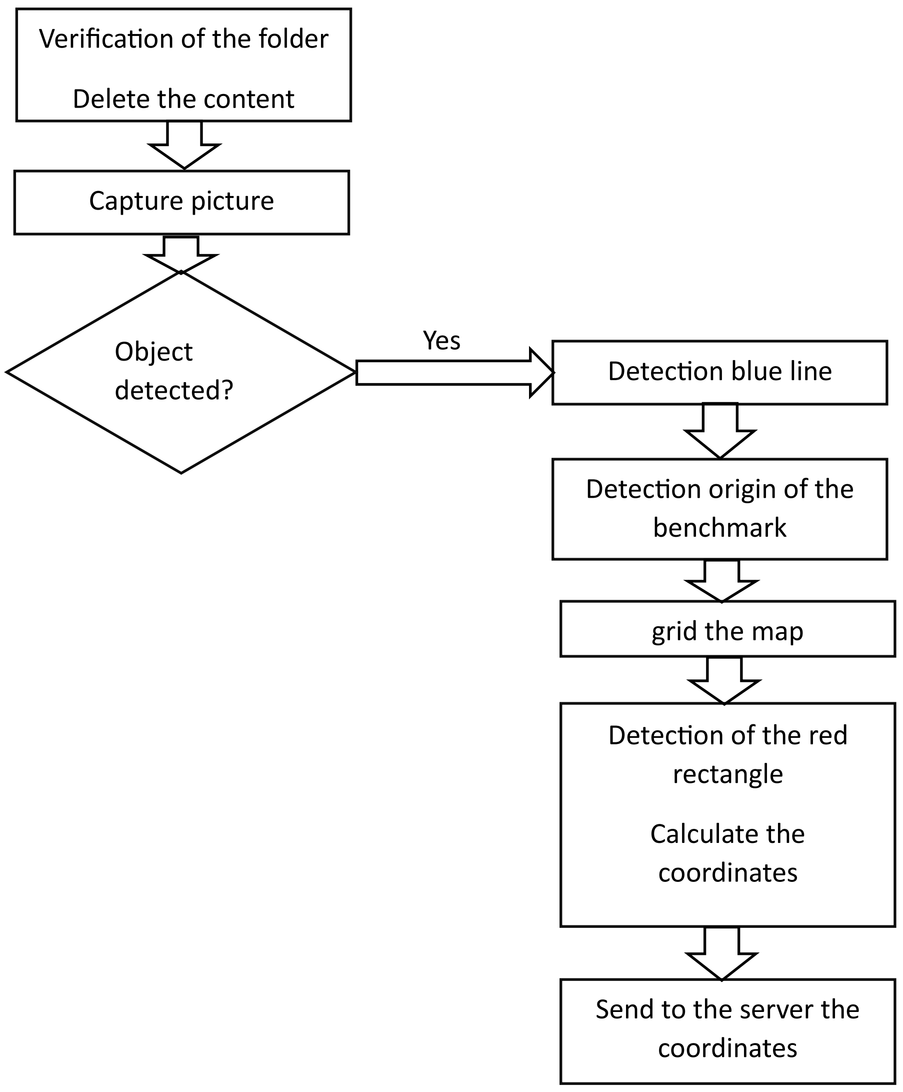


b) Detection:

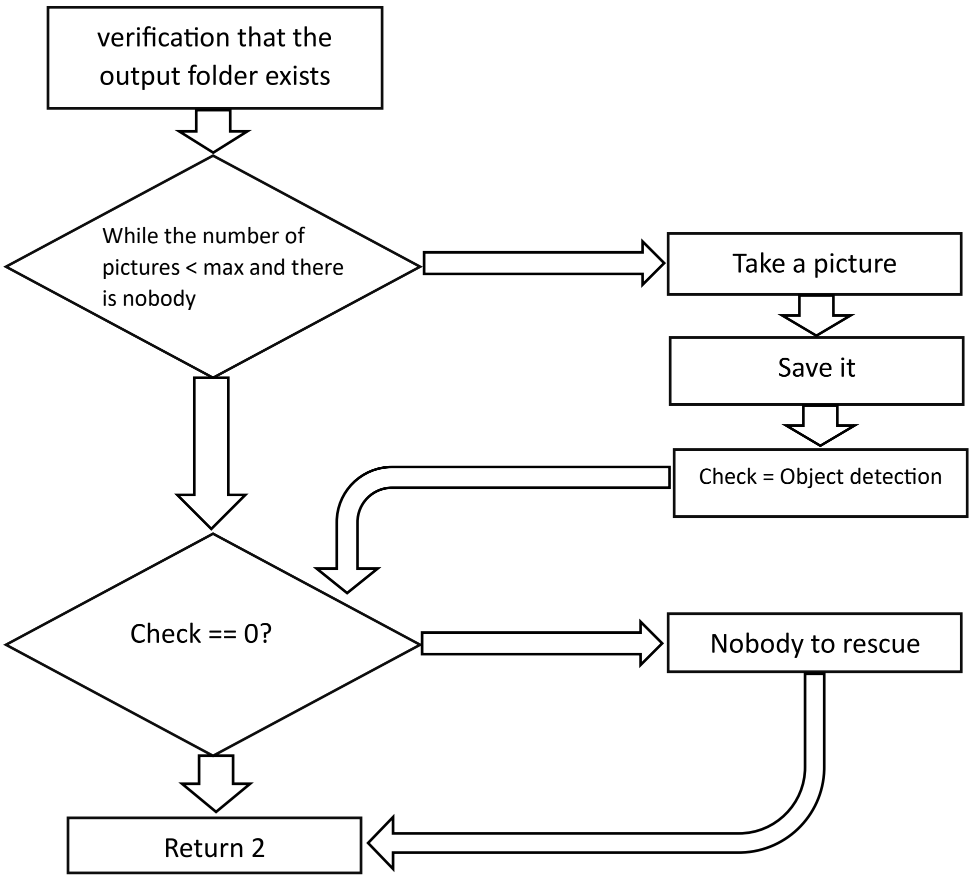

Take a picture
Is the image pixelated?
Take a picture
Setting the interpreter
Resize the image
Object detection
If the max of the score > min_conf?
Yes
Draw a red rectangle
Write the score
No
Return 1
Ask the user if he agree with the result
Calculate the distance to move
Move on the y axis
Is the rover in the front of the object?
Yes
The user adjusts the position by pressing 'd' or 'q'
No
Move on the x axis
Return 0

3) Raspberry pi ‘s program:

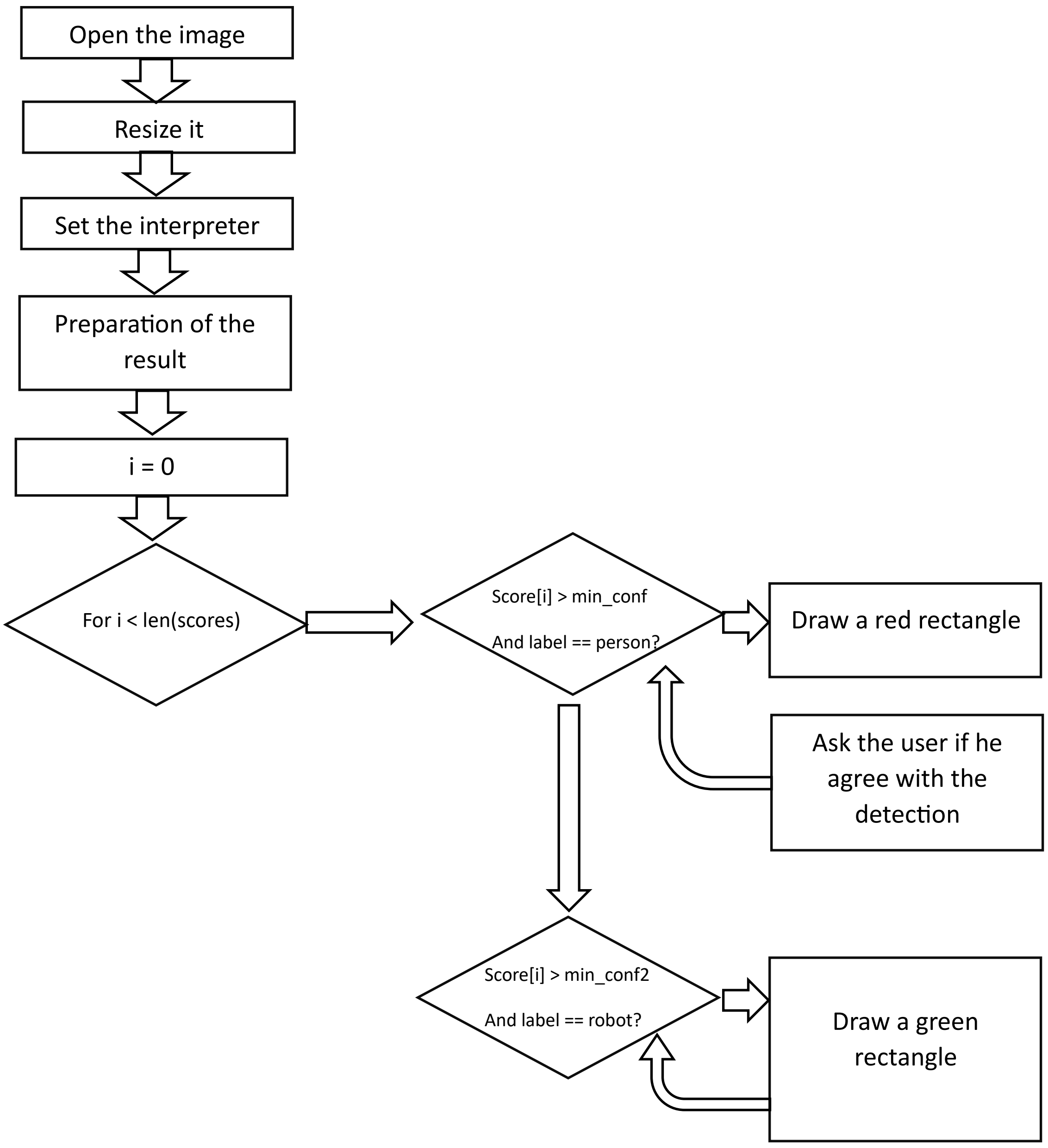
a) Main:


Verification of the folder
Delete the content
Capture picture
Object detected?
Yes
Detection blue line
Detection origin of the benchmark
grid the map
Detection of the red rectangle
Calculate the coordinates
Send to the server the coordinates

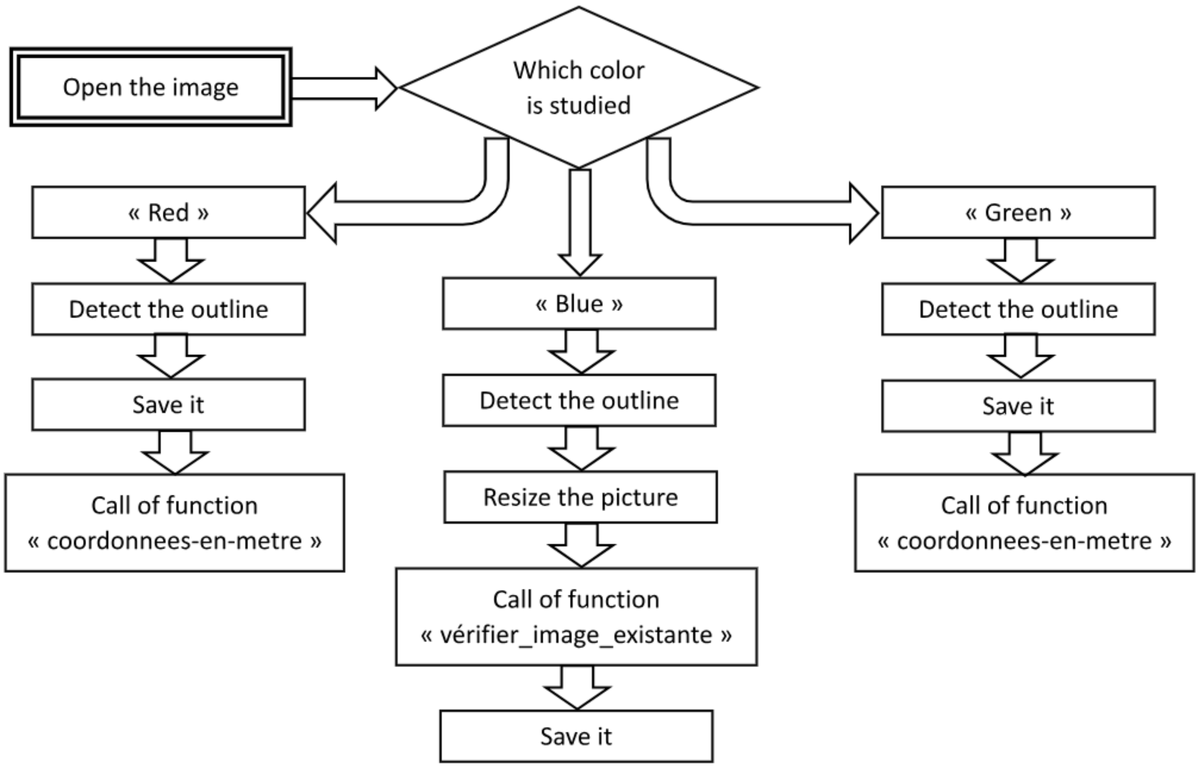
b) Capture picture:


verification that the output folder exists
While the number of pictures < max and there is nobody
Take a picture
Save it
Check = Object detection
Check == 0?
Nobody to rescue
Return 2

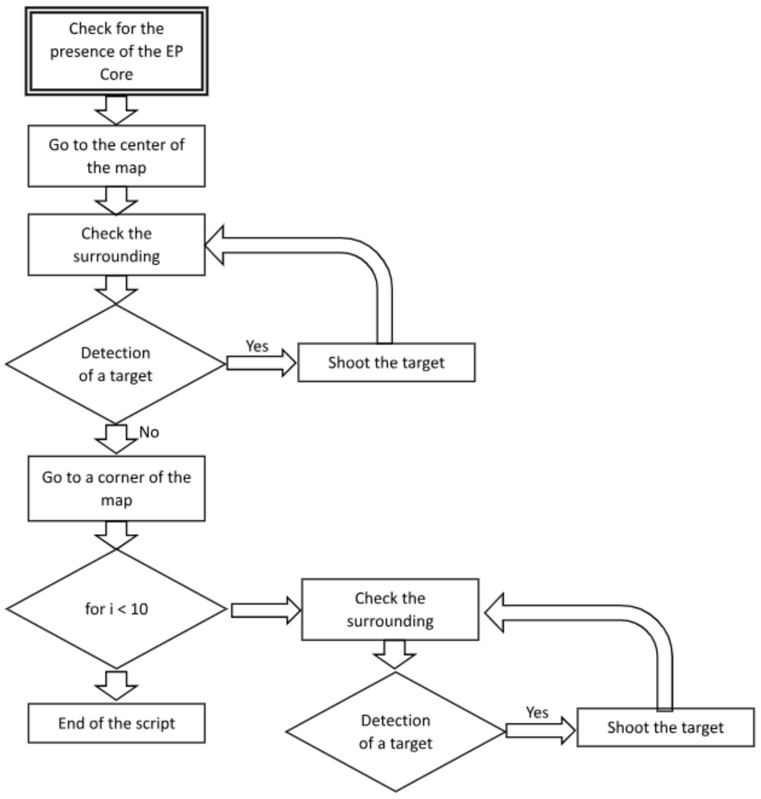
c) Detection:


Open the image
Resize it
Set the interpreter
Preparation of the result
i = 0
For i < len(scores)
Score[i] > min_conf
And label == person?
Draw a red rectangle
Ask the user if he agree with the detection
Score[i] > min_conf2
And label == robot?
Draw a green rectangle

d) Analyzing the lines



4) Robomaster S1's program:



Program of the Raspberry pi (object detection and client):

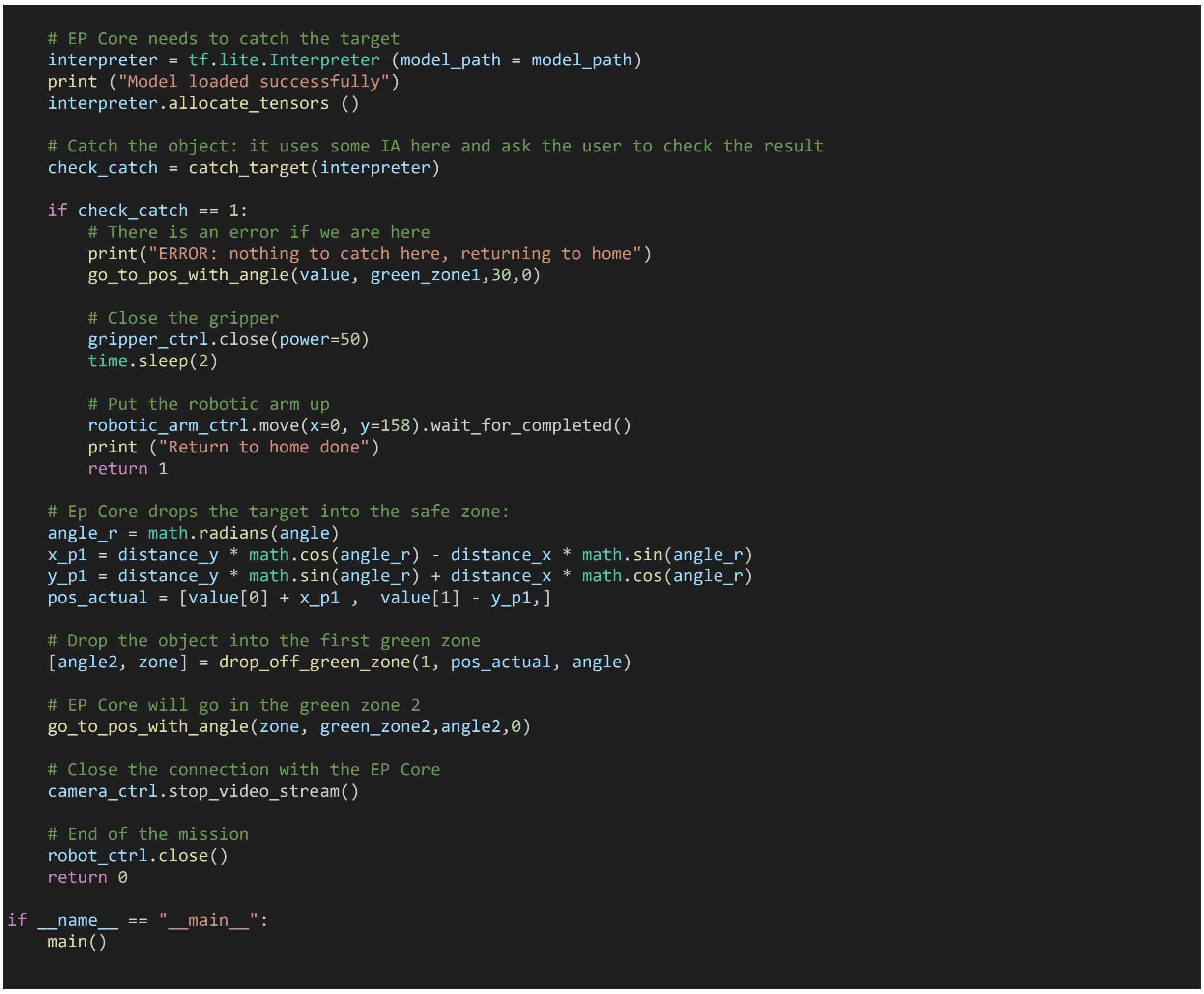
```python
import socket
import os
import cv2
import numpy as np
import sys
import glob
import random
import importlib.util
import tflite_runtime.interpreter as tf
import time
from PIL import Image, ImageDraw

rtmp_url = "rtmp://192.168.211.193/live"
model_path = '/home/theodore/Desktop/ML_VmWare_v2.tflite'
output_folder_path = "/home/theodore/Documents/ML_Mavic/test_connexion/images_stock"
output_folder_img = "/home/theodore/Documents/ML_Mavic/test_connexion/images"
labels = ['person', 'robot']
interval_seconds = 1
total_photos = 10
input_mean = 127.5 #= 255/2
input_std = 127.5
min_conf = 0.15
tab = []

cheminProjet = os.getcwd()
dossierImages = "images"
dossierImagesRedimendionnees = "cellule avec personne dedans"
dossierImagesRedimensioneesBleues = "plateau lignes bleues"
dossierImagesQuadrillees = "images quadrillees"
dossierCellules = "cellules"
grille = 120 # 1 carreau = 12in (= 0.3048m)
taille_image = None



def capture_picture(url, output_folder, output_folder_img, interval_sec, num_photos):
    # Verification:
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)
    # Initialization of the counter
    photo_count = 0
    verif = 0
    while photo_count < num_photos and verif != 2 :
        cap = cv2.VideoCapture(url)
        OK = check_rtmp_stream(cap)
        if OK != 1:
            # read the stream and take a picture
            ret, frame = cap.read()
            if ret:
                # Generation of a name for the picture
                timestamp = int(time.time())
                photo_filename = os.path.join(output_folder, f"photo_{timestamp}.jpg")

                # Save the picture
                cv2.imwrite(photo_filename, frame)
                print(f"Picture {photo_count + 1} saved: {photo_filename}")
                photo_count += 1
                verif = detection(photo_filename,interpreter,input_details, output_details, height, width,
float_input,output_folder, output_folder_img)

                # Save the picture
                cv2.imwrite(photo_filename, frame)
                # Waiting...
                time.sleep(interval_sec)
                # reboot the stream.
                cap.release()
    if verif == 0:
        print("Nobody to rescue here.")
    else:
        return 2

def detection(path_img, interpreter, input_details, output_details, height, width, float_input, output_path_IA,
output_folder_img):
    # Open the picture
    image = cv2.imread(path_img)
    imH, imW, _ = image.shape

    # Resize it
    image_resized = cv2.resize(image, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)

    if float_input:
        input_data = (np.float32(input_data) - input_mean) / input_std
    # object detection :
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()

    # preparation of the result
    boxes = interpreter.get_tensor(output_details[1]['index'])[0] # Bounding box coordinates of detected objects
    classes = interpreter.get_tensor(output_details[3]['index'])[0] # Class index of detected objects
    scores = interpreter.get_tensor(output_details[0]['index'])[0] # Confidence of detected objects

    # for all the detection :
    i= int(np.argmax(scores))
    if ((scores[i] > min_conf) and (scores[i] <= 1.0)):

        # set the coordonnates
        ymin = int(max(1,(boxes[i][0] * imH)))
        xmin = int(max(1,(boxes[i][1] * imW)))
        ymax = int(min(imH,(boxes[i][2] * imH)))
        xmax = int(min(imW,(boxes[i][3] * imW)))
        object_name = labels[int(classes[i])] # Look up object name from "labels" array using class index
        # draw a rectangle:
        if object_name == 'person':
            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (0, 10, 255), 2)
        else:
            cv2.rectangle(image, (xmin,ymin), (xmax,ymax), (10,255, 0), 2)

        timestamp = int(time.time())
        photo_filename = os.path.join(output_folder_img, f"photo_{timestamp}.jpg")
        cv2.imwrite(photo_filename, image)

        image = Image.open(photo_filename)
        image.show()
        print("\nIt looks like there is a person here\nDo you agree ? (Y/N)")
        user = input()

        if (user == "Y" or user == "y"):
            # Code de Theodore ici, pour avoir les coordonnees
            print("A person has been detected, send data to rovers...")
            return 2
        else:
            os.remove(photo_filename)
    return 0


def check_rtmp_stream(cap):
    # Check if the stream is OK
    if cap.isOpened():
        return cap
    else:
        # if we are here, there is a problem for the stream
        print("Unable to connect to RTMP stream, please check your connection...")
        return 1

def start_client(data):
    # Configuration of the client
    host = '192.168.211.45'  # IP adress of the computer
    port = 12345  # same port as the computer

    # create a socket for the client
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        # connexion client - server
        client_socket.connect((host, port))
        print("Connected to the serveur at {}:{}".format(host, port))

        numbers_list = data

        # Convertir la liste en une chaîne de caractères séparée par des virgules
        message = ','.join(str(num) for num in numbers_list)
        client_socket.sendall(message.encode('utf-8'))
        print("data:  ", data, " sent")

    except Exception as e:
        print("Error : {}".format(e))

    finally:
        # shutdown the connection
        client_socket.close()

def delete_folder(path):
    # Delete the file inside the folder
    try :
        if not os.path.isdir(path):
            print("The folder does not exist...")
            return 0
        for name in os.listdir(path):
            path_file = os.path.join(path, name)
            if os.path.isdir(path_file):
                delete_folder(path_file)
            else:
                os.remove(path_file)
    except:
        print("An error occurred")
        return 1

def detecter_origine_repere(image):
    try:
        # Open the image
        img = Image.open(image)

        x, y, p = 0, 0, 0

        pixel_color = img.getpixel((x, y))

        r = list (range (200, 255))
        g = list (range (100, 200))
        b = list (range (100, 150))

        plage_orange = (r, g, b)
        detect = False


        # Check if the color of the pixel is orange
        while ((not detect) and (p != 4)) :

            # collect the  color of the pixel at (0, 0)
            pixel_color = img.getpixel((x, y))

            if pixel_color [0] in plage_orange [0] and \
               pixel_color [1] in plage_orange [1] and \
               pixel_color [2] in plage_orange [2] :
                detect = True

            if x >= 50 :
                x = 0
                if y >= 50 :
                    # Rotate the image:
                    img_rotated = img.transpose(Image.ROTATE_90)
                    img_rotated.save (image)

                    img = Image.open(image)
                    p += 1
                    y = 0
                y += 1
            x += 1

    except Exception as e:
        print(f"An error occurred : {e}")



def verifDossier (nom_dossier) :
    # Check if the folder exist
    if not os.path.exists(nom_dossier):
        print (f"Do you want to create the folder '{nom_dossier}' ? (Y/N)")
        user = input()
        if (user == "Y" or user == "y"):
            # Create the folder
            os.mkdir(nom_dossier)
            print(f"folder created\n\n")


def quadriller_image (image, taille_grille) :
    # Open the i;age
    img = Image.open (image)

    # Create an object "draw" on the picture
    draw = ImageDraw.Draw (img)

    # collect the shape of the picture
    largeur, hauteur = img.size

    # draw verticals lines
    for x in range (0, largeur, taille_grille):
        draw.line ([(x, 0), (x, hauteur)], fill = (0, 0, 0))

    # draw horizontal lines
    for y in range (0, hauteur, taille_grille):
        draw.line ([(0, y), (largeur, y)], fill = (0, 0, 0))


    verifier_image_existante (nomImage)

    # Save the new image
    img.save (f"{cheminProjet}/{dossierImagesQuadrillees}/{nomImage}.jpg")


def etudier_image_quadrillee(image, taille_grille):

    global  cellule,            \
            coin_inf_droit,     \
            coin_sup_gauche,    \
            coordonneeX,        \
            coordonneeY,        \
            largeur

    # Open the image
    img_quad = Image.open (image)

    # Collect the shape of the picture
    largeur, hauteur = img_quad.size

    # iterate through grid cells
    for ligne, coordonneeX in enumerate(range (0, largeur, taille_grille)):
        for colonne, coordonneeY in enumerate(range (0, hauteur, taille_grille)):

            # Collect the coordinates of the cells
            coin_sup_gauche = (coordonneeX, coordonneeY)
            coin_inf_droit = (coordonneeX + taille_grille, coordonneeY + taille_grille)

            # Verification: are the coordinates in the limits of the image?
            if coin_inf_droit[0] > largeur:
                coin_inf_droit = (largeur, coin_inf_droit[1])
            if coin_inf_droit[1] > hauteur:
                coin_inf_droit = (coin_inf_droit[0], hauteur)

            # Extraction of the cell
            cellule = img_quad.crop((coin_sup_gauche[0], coin_sup_gauche[1], coin_inf_droit[0], coin_inf_droit[1]))

            cellule = extraire_cellule(img_quad, ligne, colonne, taille_grille)

            # Convert the image PIL in a image OpenCV (BGR)
            cellule_cv2 = cv2.cvtColor(np.array(cellule), cv2.COLOR_RGB2BGR)

            # Save the new image
            cv2.imwrite(f"{cheminProjet}/{dossierCellules}/{nomImage} cellule {(int (coordonneeX/taille_grille), int
(coordonneeY/taille_grille))}.jpg", cellule_cv2)
            tab = detecter_lignes(f"{cheminProjet}/{dossierCellules}/{nomImage} cellule {(int
(coordonneeX/taille_grille), int (coordonneeY/taille_grille))}.jpg", "rouge")
            if tab != None :
                list_float = [float(s) for s in tab]
                return list_float

def coordonnees_en_metre (largeurImage, axeX, axeY) :

    nbPixelsParCarreau = largeurImage / 8 # = 0.3048 m

    positionX = str(round ((axeX / nbPixelsParCarreau) * 0.3048, 2))
    positionY = str(round ((axeY / nbPixelsParCarreau) * 0.3048, 2))

    positionFinX = str(round (((axeX + grille) / nbPixelsParCarreau) * 0.3048, 2))
    positionFinY = str(round (((axeY + grille) / nbPixelsParCarreau) * 0.3048, 2))

    return[positionX, positionY,positionFinX, positionFinY]


def extraire_cellule(image, ligne, colonne, taille_cellule):
    # Calculations of the coordinates of the cell
    x1 = colonne * taille_cellule
    y1 = ligne * taille_cellule
    x2 = x1 + taille_cellule
    y2 = y1 + taille_cellule

    return image.crop((x1, y1, x2, y2))


def detecter_lignes (image, couleur) :  # sourcery skip: extract-duplicate-method, extract-method
    # Open the image
    img = cv2.imread (image)

    # Convertion BGR => HSV
    hsv = cv2.cvtColor (img, cv2.COLOR_BGR2HSV)


    # Definition of the blue range in HSV
    bas_bleu = np.array     ([90, 50, 50])
    haut_bleu = np.array    ([130, 255, 255])
    plage_bleue = cv2.inRange (hsv, bas_bleu, haut_bleu)
    # Applying a blur
    plage_bleue = cv2.GaussianBlur (plage_bleue, (5, 5), 0)


    # Definition of the red range in HSV
    bas_rouge = np.array    ([0, 100, 100])
    haut_rouge = np.array   ([10, 255, 255])
    plage_rouge = cv2.inRange (hsv, bas_rouge, haut_rouge)
    # Applying a blur
    plage_rouge = cv2.GaussianBlur (plage_rouge, (5, 5), 0)


    if couleur == "bleue" :# blue edge detection
        contours, _ = cv2.findContours (plage_bleue, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    elif couleur == "rouge" :# Red edge detection
        contours, _ = cv2.findContours(plage_rouge, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    if contours :
        # find the smallest rectangle
        x_min, y_min, x_max, y_max = img.shape[1], img.shape[0], 0, 0
        for contour in contours:
            x, y, w, h = cv2.boundingRect (contour)
            x_min = min (x_min, x)
            y_min = min (y_min, y)
            x_max = max (x_max, x + w)
            y_max = max (y_max, y + h)



        if couleur == "bleue" :
            # crop the image around the detected lines
            img_cropee = img [y_min:y_max, x_min:x_max]

            # verification
            verifier_image_existante (nomImage)
            # Save the new image
            cv2.imwrite(f"{cheminProjet}/{dossierImagesRedimensioneesBleues}/{nomImage}.jpg", img_cropee)
        else :
            verifier_image_existante (nomImage)

            # Show the data of the pixels
            valeurs_pixels = cellule.getdata()

            # Save the new picture
            cv2.imwrite(f"{cheminProjet}/{dossierImagesRedimendionnees}/{nomImage} cellule {(int
(coordonneeX/grille), int (coordonneeY/grille))}.jpg", img)

            [coordonneeX1,coordonneeY1,coordonneeX2, coordonneeY2] = coordonnees_en_metre (largeur, coordonneeX,
coordonneeY)
            return [coordonneeX1,coordonneeY1,coordonneeX2, coordonneeY2]


def verifier_image_existante (image) :
    # deleted if the file already exist
    if os.path.exists(image):
        os.remove(image)


def main():
    verifDossier (f"{dossierImagesRedimendionnees}")
    verifDossier (f"{dossierImagesRedimensioneesBleues}")
    verifDossier (f"{dossierImagesQuadrillees}")
    verifDossier (f"{dossierCellules}")

    delete_folder("/home/theodore/Documents/ML_Mavic/test_connexion/cellule avec personne dedans")
    delete_folder("/home/theodore/Documents/ML_Mavic/test_connexion/cellules")
    delete_folder("/home/theodore/Documents/ML_Mavic/test_connexion/images")
    delete_folder("/home/theodore/Documents/ML_Mavic/test_connexion/images quadrillees")
    delete_folder("/home/theodore/Documents/ML_Mavic/test_connexion/images_stock")
    delete_folder("/home/theodore/Documents/ML_Mavic/test_connexion/plateau lignes bleues")

    check = capture_picture(rtmp_url, output_folder_path,output_folder_img, interval_seconds, total_photos)
    os.system ('cls')

    global nomImage
    print(check)
    if check == 2:
        for i in os.listdir (f"{cheminProjet}/{dossierImages}") :
            print(i)
            img = os.path.splitext (i)

            nomImage = img [0]

            detecter_lignes(f"{cheminProjet}/{dossierImages}/{nomImage}.jpg", "bleue")
            detecter_origine_repere (f"{cheminProjet}/{dossierImagesRedimensioneesBleues}/{nomImage}.jpg")
            quadriller_image (f"{cheminProjet}/{dossierImagesRedimensioneesBleues}/{nomImage}.jpg", grille)
            tab = etudier_image_quadrillee (f"{cheminProjet}/{dossierImagesQuadrillees}/{nomImage}.jpg", grille)
            if tab != None:
                start_client(tab)
                break


if __name__ == "__main__":
     # Set the interpreter:
    interpreter = tf.Interpreter(model_path)
    print("Model loaded")
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height = input_details[0]['shape'][1]
    width = input_details[0]['shape'][2]
    float_input = (input_details[0]['dtype'] == np.float32)
    main()
```

Program of the Robomaster EP Core (server and the movement functions):

```
import socket
import os
import math
import cv2
import time
import robomaster
import numpy as np
from robomaster import robot, camera
from robomaster import vision
import tensorflow as tf
from PIL import Image, ImageDraw

# Variables definition:
reg_value_x = 0
reg_value_y = 0.35 # Unit = meters

distance_x = 0
distance_y = 0

index = 0

angle = 0
xmin = xmax = 0
ymin = ymax = 0
x = y = z = 0
photo_interval = 0.1
input_mean = 127.5
input_std  = 127.5

model_path = 'D:/IA/ML_robmaster_EP_core.tflite'
labels = ['person']

min_conf = 0.4

robot_ctrl = robot.Robot()
robot_ctrl.initialize(conn_type="sta")
chassis_ctrl = robot_ctrl.chassis
gimbal_ctrl = robot_ctrl.gimbal
vision_ctrl = robot_ctrl.vision
camera_ctrl = robot_ctrl.camera
robotic_arm_ctrl = robot_ctrl.robotic_arm
gripper_ctrl = robot_ctrl.gripper
interpreter = None

# definition of the green zone:
green_zone1 = [1.37, 2.28]
green_zone2 = [0.15, 1.06]
line = []
robots = []
var = True
```

```python
class RobotInfo:

    def __init__(self, x, y, w, h):
        self._x = x
        self._y = y
        self._w = w
        self._h = h

    @property
    def pt1(self):
        return int((self._x - self._w / 2) * 1280), int((self._y - self._h / 2) * 720)

    @property
    def pt2(self):
        return int((self._x + self._w / 2) * 1280), int((self._y + self._h / 2) * 720)

    @property
    def center(self):
        return int(self._x * 1280), int(self._y * 720)

class PointInfo:

    def __init__(self, x, y, theta, c):
        self._x = x
        self._y = y
        self._theta = theta
        self._c = c

    @property
    def pt(self):
        return int(self._x * 1280), int(self._y * 720)

    @property
    def color(self):
        return 255, 255, 255


def sub_position_handler(position_info):
    global x,y,z
    x, y, z = position_info
    print("chassis position: x:{0}, y:{1}, z:{2}".format(x, y, z))


def detection (robot_ctrl, interpreter):
    global index
    index += 1
    check = False
    global xmin, xmax, ymin, ymax
    global distance_x, distance_y
    camera_ctrl = robot_ctrl.camera
    while not check:
        print("Taking a picture")
        image = camera_ctrl.read_cv2_image(strategy="newest")

        # Ask the user if the photo is ok
        # Somestimes the camera does not work very well (it pixelates)
        cv2.imshow("image", image)
        cv2.waitKey(5000)
        cv2.destroyAllWindows()
        print("Is the photo OK for detection?    (Y/N)")
        letter = input()
        if letter == 'y' or letter == 'Y':
            check = True
        else :
            check = False
    time.sleep(2)

    # Setting the interpreter
    input_details  =  interpreter.get_input_details()
    output_details =  interpreter.get_output_details()
    height         =  input_details[0]['shape'][1]
    width          =  input_details[0]['shape'][2]

    print("Analysing the picture...")
    imH, imW, _ = image.shape

    # Resize it
    image_resized = cv2.resize(image, (width, height))
    input_data = np.expand_dims(image_resized, axis=0)
    name = f"D:/IA/IA_photo_robomaster/photo{index}.jpg"
    cv2.imwrite(name, image)
    # Object detection :
    interpreter.set_tensor(input_details[0]['index'],input_data)
    interpreter.invoke()
    # Preparation of the result
    boxes = interpreter.get_tensor(output_details[1]['index'])[0] # Bounding box coordinates of detected objects
    scores = interpreter.get_tensor(output_details[0]['index'])[0] # Confidence of detected objects

    # For all the detection :
    i = np.argmax(scores)
    if ((scores[i] >= min_conf) and (scores[i] <= 1.0)):
        proba =  scores[i]*100
        msg = f"A person has been detected: {proba} %"
        print(msg)

        # Set the coordonnates
        ymin = int(max(1,(boxes[i][0] * imH)))
        xmin = int(max(1,(boxes[i][1] * imW)))
        ymax = int(min(imH,(boxes[i][2] * imH)))
        xmax = int(min(imW,(boxes[i][3] * imW)))

        # Draw a rectangle around the object and show the score
        cv2.rectangle(image,(xmin,ymin),(xmax,ymax),(0,10,255),2)
        object_name = "Person" # Look up object name from "labels" array using class index
        label = '%s: %d%%' % (object_name, int(proba)) # Example: 'person: 72%'
        labelSize, baseLine = cv2.getTextSize(label, cv2.FONT_HERSHEY_SIMPLEX, 0.7, 2) # Get font size
        label_ymin = max(ymin, labelSize[1] + 10) # Make sure not to draw label too close to top of window
        cv2.rectangle(image, (xmin, label_ymin-labelSize[1]-10), (xmin+labelSize[0], label_ymin+baseLine-10), (255,
255, 255), cv2.FILLED) # Draw white box to put label text in
        cv2.putText(image, label, (xmin, label_ymin-7), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 0), 2) # Draw label
text
        cv2.imwrite("D:/IA/IA_photo_robomaster/photodetec.jpg", image) # Save the image

        # Show the image with the object detection
        cv2.imshow("Person detected", image)

        # Wait for 5 s
        cv2.waitKey(5000)
        cv2.destroyAllWindows()

        # Ask the user if he agrees
        print("It looks like there is a person here, \n do you agree?   (Y/N)")
        if input() == 'y' or input() == 'Y':
            print("Going to the target position")

            # Calculate the distance to move:
            delta_x = int((xmin + xmax) / 2 - 640 )
            delta_y = int(650 - (ymin + ymax) / 2)

            if abs(delta_x) > 10 :
                distance_y = delta_x * 0.4 / 275
            else:
                distance_y = 0
            if abs(delta_y) > 10 :
                distance_x = delta_y * 0.3 / 275
            else:
                distance_x = 0

            # The rover move to the target position
            chassis_ctrl.move(x = distance_x, y = distance_y, z = 0, xy_speed=0.7).wait_for_completed()
            return 0
        else :
            return 1
    else:
        print("Nobody here")
        return 1




def start_server():
    # Server Configuration
    host = ''  # Empty to permite all the connexion
    port = 12345

    # Creation of a socket for the server
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    try:
        # Link the socket
        server_socket.bind((host, port))

        # put the socket in bind mode
        server_socket.listen(1)
        print("the server is binding {}:{}".format(host, port))

        # Accept the connexion
```

```python
        client_socket, client_address = server_socket.accept()
        print("Connection with : {}".format(client_address))

        # Received the data from the Raspberry pi
        data = client_socket.recv(1024)
        print("received : ", data)
        numbers_list = [float(num) for num in data.decode('utf-8').split(',')]

    except Exception as e:
        print("An error happened : {}".format(e))
        return 1

    finally:
        # Close the connexions
        client_socket.close()
        server_socket.close()
        return numbers_list


def go_to_pos_with_angle(pos, pos_desire, actual_angle, angle_d):
    # Angle calculation
    theta = int(math.degrees(math.atan2(pos[0] - pos_desire[0], pos[1] - pos_desire[1])))

    # Keep this angle in memory
    mem_theta = theta
    l = math.sqrt((pos[1] - pos_desire[1])**2 + (pos[0] - pos_desire[0])**2)

    # Rotate the bot
    theta = theta - actual_angle
    chassis_ctrl.move(x=0, y=0, z=theta, z_speed=45).wait_for_completed()

    # Move the robot
    chassis_ctrl.move(x=l, y=0, z=0, xy_speed=0.7).wait_for_completed()

    # Rotate to have angle_d
    if angle_d == "PASS" :
        return mem_theta
    else :
        if (angle_d > 0 and theta > 0) or (angle_d < 0 and theta < 0):
            theta = angle_d - mem_theta
        else:
            theta = -(mem_theta - angle_d)
        chassis_ctrl.move(x=0, y=0, z=theta, z_speed=45).wait_for_completed()
        return 0

def waiting_zone(pos_d, pos_desire, R):
    """
    The goal here is to calculate a zone where the EP Core will stop to drop
    the target correctly into the green zone
    """

    # Calculate the direction of the EP Core
    direction = [pos_desire[0] - pos_d[0], pos_desire[1] - pos_d[1] ]

    # Normalized it
    norm_direction = math.sqrt(direction[0]**2 + direction[1]**2)
    normalized_direction = [direction[0] / norm_direction, direction[1] / norm_direction]

    # Calculate the intersection between the circle with radius R and the direction
    intersection = [pos_d[0] + R * normalized_direction[0], pos_d[1] + R * normalized_direction[1]]

    # Return the intersection coordinates
    return intersection


def drop_off_green_zone(id_zone, pos_actuel, angle):
    global green_zone1
    global green_zone2

    # Where will we drop the object?
    if id_zone == 1:
        zone = green_zone1
    else:
        zone = green_zone2

    # The waiting zone is where the EP Core needs to stop to drop correctly the object into the green zone
    zone_att = waiting_zone(zone,pos_actuel,0.4)

    # EP Core will go to the waiting zone
    angle1 = go_to_pos_with_angle(pos_actuel, zone_att,angle,"PASS")

    # Drop off :
    robotic_arm_ctrl.move(x=20, y=-200).wait_for_completed()
    gripper_ctrl.open(power=50)
    # Let the object drop
    time.sleep(2)

    # lift the arm up
    robotic_arm_ctrl.move(x=0, y=158).wait_for_completed()

    # close the gripper
    gripper_ctrl.close(power=50)
    time.sleep(2)

    # return the angle and the position where the EP Core is
    return [(angle1), zone_att]

def catch_target(interpreter):
    global angle
    angle = 0
    # Open the gripper:
    gripper_ctrl.open(power=50)
    time.sleep(2)

    # Put the robotic arm down
    robotic_arm_ctrl.move(x=20, y=-200).wait_for_completed()
    time.sleep(2)

    # Add a IA detection object here:
    check = detection (robot_ctrl, interpreter)

    # If there is nobody here:
    if check == 1:
        angle = -30
        chassis_ctrl.move(x = 0, y = 0, z = angle, z_speed = 45).wait_for_completed()
        time.sleep(2)
        check = detection(robot_ctrl, interpreter)

    # Try to find a person around the rover:
    while check == 1 and angle < 30:
        chassis_ctrl.move(x=0, y=0, z=10, z_speed=45).wait_for_completed()
        time.sleep(2)
        check = detection(robot_ctrl, interpreter)
        angle += 10

    # The rover find somebody:
    if check == 0:
        # Close the gripper
        gripper_ctrl.close(power=50)
        time.sleep(2)

        # Put the robotic arm up
        robotic_arm_ctrl.move(x=0, y=158).wait_for_completed()
    return check

def on_detect_person(person_info):
    global var
    var = True
    number = len (person_info)
    robots.clear ()
    if number == 0 :
        var = False

def main():
    global interpreter

    # Start the video stream of EP Core
    camera_ctrl.start_video_stream(display=False)

    # Start the server
    tab = start_server()

    # Waiting: the S1 needs to go in the center of the map and kill the ennemies
    time.sleep(5)

    # Has the S1 finished its mission?
    while (var) :
        time.sleep (2)
        vision_ctrl.sub_detect_info(name = "robot", callback = on_detect_person)

    # Close the detection of S1 with the camera
    vision_ctrl.unsub_detect_info(name="robot")

    # Now the zone is safe,
    # EP Core will go to the target position
    value = [tab[1] + reg_value_x, tab[0] + reg_value_y] # value [CoordonnéesY, CoordonnéesX]
    print ("Target position  = ", value)
    go_to_pos_with_angle (green_zone1, value, 0, 0)
```



```python
    # EP Core needs to catch the target
    interpreter = tf.lite.Interpreter (model_path = model_path)
    print ("Model loaded successfully")
    interpreter.allocate_tensors ()

    # Catch the object: it uses some IA here and ask the user to check the result
    check_catch = catch_target(interpreter)

    if check_catch == 1:
        # There is an error if we are here
        print("ERROR: nothing to catch here, returning to home")
        go_to_pos_with_angle(value, green_zone1,30,0)

        # Close the gripper
        gripper_ctrl.close(power=50)
        time.sleep(2)

        # Put the robotic arm up
        robotic_arm_ctrl.move(x=0, y=158).wait_for_completed()
        print ("Return to home done")
        return 1

    # Ep Core drops the target into the safe zone:
    angle_r = math.radians(angle)
    x_p1 = distance_y * math.cos(angle_r) - distance_x * math.sin(angle_r)
    y_p1 = distance_y * math.sin(angle_r) + distance_x * math.cos(angle_r)
    pos_actual = [value[0] + x_p1 ,  value[1] - y_p1,]

    # Drop the object into the first green zone
    [angle2, zone] = drop_off_green_zone(1, pos_actual, angle)

    # EP Core will go in the green zone 2
    go_to_pos_with_angle(zone, green_zone2,angle2,0)

    # Close the connection with the EP Core
    camera_ctrl.stop_video_stream()

    # End of the mission
    robot_ctrl.close()
    return 0

if __name__ == "__main__":
    main()
```