République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université A. MIRA - BEJAIA

Faculté de Technologie
Département d'Automatique, Télécommunications et d'Électronique (ATE)

Projet de Fin d'Etude

Pour l'obtention du diplôme de Master en Automatique et Informatique industrielle

Thème

Développement et réalisation d'un système de suivi de cible

Préparé par :

DAHLI Fares
BOUHAMDANI Messaoud

Dirigé par :

Mme MEZZAH Samia

Année Universitaire : 2021/2022

Acknowledgements

Above all, we thank God for the health, patience and willpower he gave us during all these years, we really needed it.

We would like to thank Mrs. MEZZAH Samia for agreeing to be our supervisor and the members of the jury for agreeing to judge our work.

We would like to sincerely thank Mr MEZZAH Ibrahim and Mr KERMIA Omar for their kindness, patience, trust, dedication and their invaluable help.

We would like to address a particular mention to Dr ABDELMALEK Omar for his precious advices.

Dedication

To my parents and all their sacrifices,

To my brother who, I hope, will not make the same mistakes as me,

to all my family and anyone who has helped me at any time,

to my brothers whom my mother did not give birth to,

Thank you.

Fares.

I would like to express my deep gratitude to my dear family, starting with my parents and then my sisters Lydia, Djamila, Kahina and Amira who encouraged, supported and helped me throughout my school and university studies.

Not to mention my beloved nieces Ritaj, Ines and Mayline and my nephew Anes.

And to all my friends with whom I spent unforgettable moments at university.

Messaoud.

Contents table

Chapitre III : Implementation

Table of figures

Chapitre I :

Chapitre II :

Chapitre II :

Introduction

In recent years, the success and capabilities of embedded vision have showed up in embedded applications. The embedding of vision into electronic devices such as embedded medical applications is being driven by the availability of high-performance processors, integrating with deep learning algorithms, as well as advances in image processing technology. But, including image processing in embedded vision systems need huge amount of computational capabilities even to process a single image to detect an object and it's extremely challenging to implement in embedded systems. Implementing deep learning algorithms and testing it on a task specific data set could provide enhanced results. In this paper, an approach for enhancing image processing architecture using deep learning for embedded vision systems is proposed and analyzed. Implementing deep learning algorithms and testing it on embedded vision yielded effective results.

In this context, the main purpose of this project is to develop an object tracking system and embed it on a mobile robot.

The dissertation is organized as follows. Chapter I presents deep learning and computer vision. In chapter II, we present a description of the system. Finally, in chapter III, we describe the implementation of our system.

Chapter I: Deep learning and computer vision.

1. Introduction

We live in a hyper connected world in which every simple interaction, phone call, payment transaction, web browsing, is added to an endless ocean of data. With the arrival of IOT, cars, alarms and even wearables contribute with a huge amount of additional data every day. During this chapter, we're going to introduce some definitions and basics of the main technologies of the future and the field in which we are working on.

2. Artificial Intelligence

Artificial Intelligence is a method of making a computer, a computer-controlled robot, or a software think intelligently like the human mind. AI is accomplished by studying the patterns of the human brain and by analyzing the cognitive process. The outcome of these studies develops intelligent software and systems.

It is more about the process and ability to think and analyze data to the maximum depth than any particular format or function.

3. Machine Learning

Machine learning is a branch of artificial intelligence where computer systems are programmed to learn from data that is input without being continually reprogrammed. In other words, they continuously improve their performance on a task without additional help from a human. Machine learning is being used in a wide range of fields. And there are different ways of getting machines to learn. Some are simple, such as a basic decision tree, and some are much more complex.

4. Deep Learning

Deep learning is a subfield of Machine Learning which itself is a subfield of Artificial Intelligence. Its structure and function are inspired by the ones of the human brain. Its algorithms can work with enormous amount of both structured and unstructured data (unlike machine learning, which can work only with structured data). Artificial neural networks

(ANN) are the core concept of deep learning, they are the ones who enable machines to make decisions.

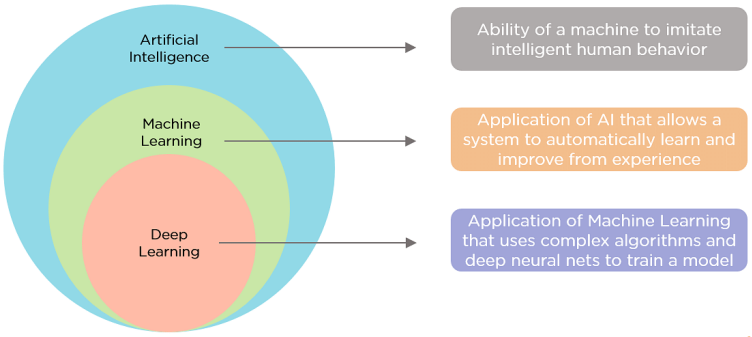

Figure I.1: Differences between Artificial intelligence, machine learning and deep learning.

The artificial neural networks are composed of three types of layers:

- Input layer : initial data.
- Hidden layers : the heart of the network, the place where all the computation is done.
- Output layer: provide the expected results.

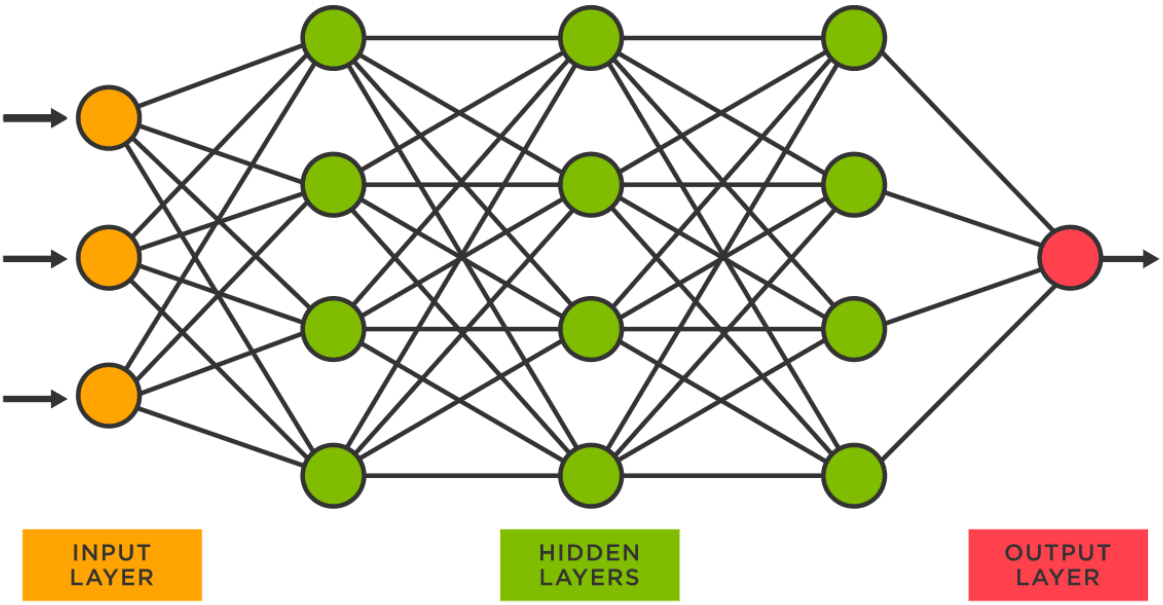

Figure I.2: Composition of an artificial neural network.

4.1. Types of Deep Neural Networks

There are 4 major types of neural networks in deep learning :

- **Convolutional Neural Network (CNN) :** CNN is a class of deep neural networks most commonly used for image analysis.
- **Recurrent Neural Network (RNN) :** RNN uses sequential information to build a model. It often works better for models that have to memorize past data.
- **Generative Adversarial Network (GAN) :** GAN are algorithmic architectures that use two neural networks to create new, synthetic instances of data that pass for real data. A GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers.
- **Deep Belief Network (DBN) :** DBN is a generative graphical model that is composed of multiple layers of latent variables called hidden units. Each layer is interconnected, but the units are not.

4.2. Deep learning applications

There are many applications, some of them are:

- Self-driving cars
- Voice search and virtual assistants
- Colorization of old black and white pictures
- Real-time object recognition in the image
- Object detection and face recognition
- Caption bot for captioning an image

5. Convolutional Neural Network (CNN)

A convolutional neural network (CNN or ConvNet), is a network architecture for deep learning which learns directly from data, eliminating the need for manual feature extraction.

CNNs are particularly useful for finding patterns in images to recognize objects, faces, and scenes. They can also be quite effective for classifying non-image data such as audio, time series, and signal data.

Like a traditional neural network, a CNN has neurons with weights and biases. The model learns these values during the training process, and it continuously updates them with each new training example. However, in the case of CNNs, the weights and bias values are the same for all hidden neurons in a given layer.

This means that all hidden neurons are detecting the same feature, such as an edge or a blob, in different regions of the image. This makes the network tolerant to translation of objects in an image. For example, a network trained to recognize cars will be able to do so wherever the car is in the image.

5.1. CNN Architecture

Like other neural networks, a CNN is composed of an input layer, an output layer, and many hidden layers in between. (Figure)

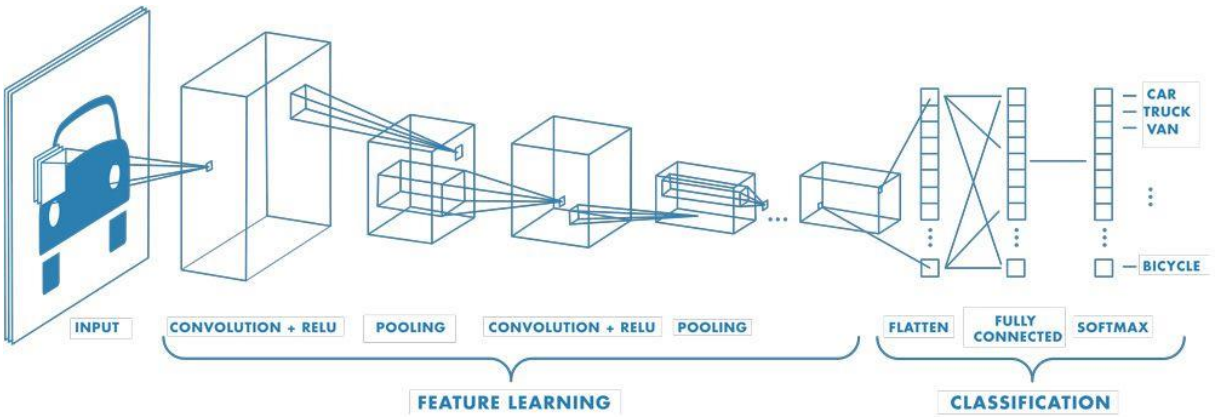

Figure I.3: Example of a network with many convolutional layers.

5.1.1. Feature detection layers:

These layers perform one of three types of operations on the data: convolution, pooling, or rectified linear unit (ReLU).

- **Convolution** puts the input images through a set of convolutional filters, each of which activates certain features from the images.
- **Pooling** simplifies the output by performing nonlinear downsampling, reducing the number of parameters that the network needs to learn about.
- **Rectified linear unit (ReLU)** allows for faster and more effective training by mapping negative values to zero and maintaining positive values.

These three operations are repeated over tens or hundreds of layers, with each layer learning to detect different features.

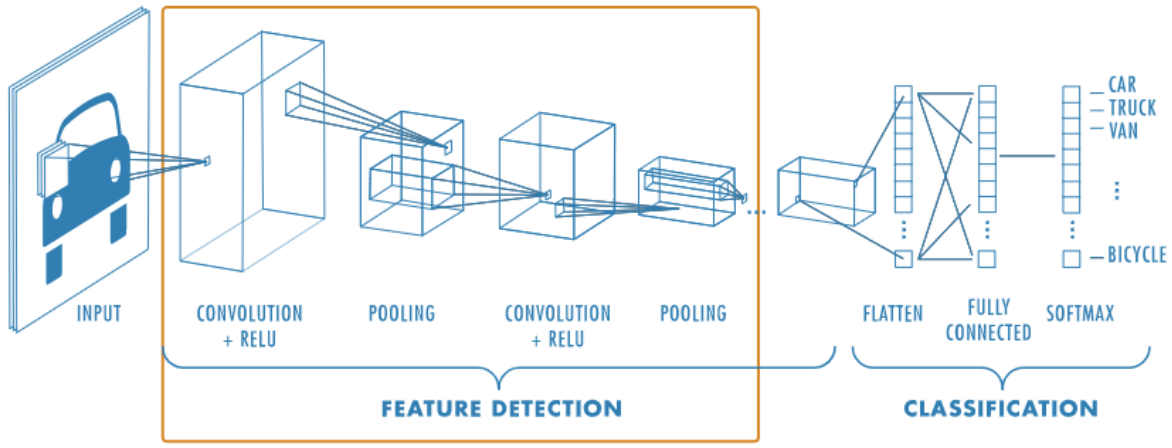

Figure I.4: Feature Detection Layers and their main operations.

5.1.2. Classification layers:

After feature detection, the architecture of a CNN shifts to classification.

The next-to-last layer is a fully connected layer (FC) that outputs a vector of K dimensions where K is the number of classes that the network will be able to predict. This vector contains the probabilities for each class of any image being classified.

The final layer of the CNN architecture uses a softmax function to provide the classification output.

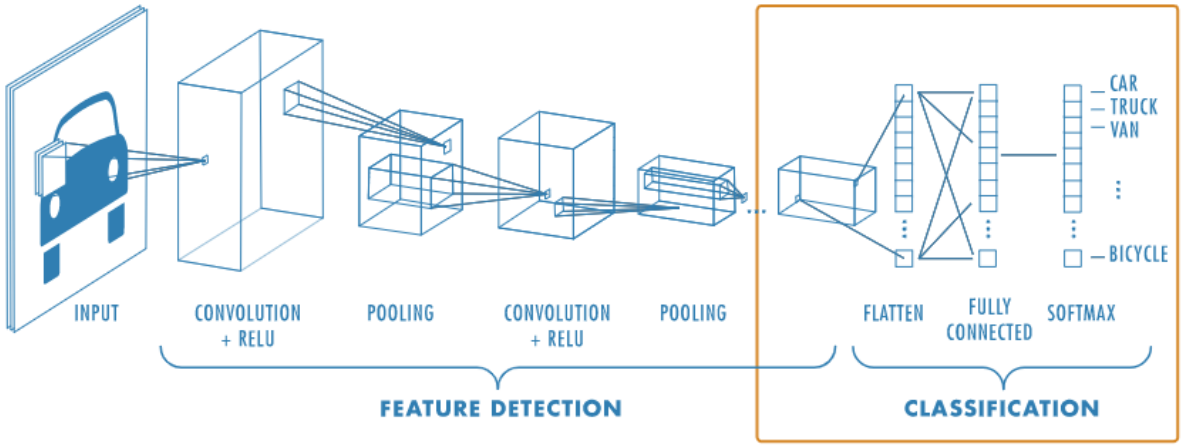

Figure I.5: Classification Layers and their main operations.

6. Computer Vision

Computer vision is the branch of artificial intelligence that teaches computers to “see” digital images such as photos and videos. It seeks to understand and automate tasks that the human visual system can do. Its tasks include methods for acquiring, processing, analyzing and understanding digital images.

6.1. Computer Vision and Image Processing

Image Processing is mostly related to the usage and application of mathematical functions and transformations over images regardless of any intelligent inference being done over the image itself. It simply means that an algorithm does some transformations on the image such as smoothing, sharpening, contrasting, stretching on the image.

Image processing is a subset of computer vision. A computer vision system uses the image processing algorithms to try and perform emulation of vision at human scale.

6.2. Computer vision examples

Here are a few examples of established computer vision tasks:

- **Image classification** sees an image and can classify it (a dog, an apple, a person's face). More precisely, it is able to accurately predict that a given image belongs to a certain class. For example, a social media company might want to use it to automatically identify and segregate objectionable images uploaded by users.
- **Object detection** can use image classification to identify a certain class of image and then detect and tabulate their appearance in an image or video. Examples include detecting damages on an assembly line or identifying machinery that requires maintenance.
- **Object tracking** follows or tracks an object once it is detected. This task is often executed with images captured in sequence or real-time video feeds. Autonomous vehicles, for example, need to not only classify and detect objects such as pedestrians, other cars and road infrastructure, they need to track them in motion to avoid collisions and obey traffic laws.
- **Content-based image retrieval** uses computer vision to browse, search and retrieve images from large data stores, based on the content of the images rather than metadata tags associated with them. This task can incorporate automatic image annotation that replaces manual image tagging. These tasks can be used for digital asset management systems and can increase the accuracy of search and retrieval.

7. Conclusion

We can retain from all the definitions above that the deep learning allows us to manage and process huge amounts of data and make complex mathematical operations over it. That will allow us to create models that will either do the same work than an expert in his field in some cases, or do calculations that are just too complex for a human that it would require many years to solve in the other cases.

This huge advantage does not come without disadvantages, since applying a complex model requires high computing resources and can only be done by either high end computers or computing accelerators. In addition, the development of a model takes a lot more resources. And those resources are so expensive that big companies such as google are renting these resources for those who desire to develop a model (google colab).

Chapter II : System description.

1. Introduction

In this chapter, we will present the main components and technologies on which our project will be based, their specifications, and the reasons that led us to choose them.

2. The mobile platform

The best mobile platform choice in our case is an omnidirectional one, this structure will be able to perform movements in all possible directions of a plane (translations and rotations) thanks to its mecanum wheels.

Based on availability, we decided to go with the Keyestudio KS4032 4WD mecanum robot car for micro:bit as a platform.

This Keyestudio 4WD Mecanum Robot Car is a smart DIY car specially designed for micro:bit. The smart car kit consists of a car body with extended functions, a PCB base plate with integrated motor drive sensors, 4 decelerating DC motors, Mecanum wheels, various modules and sensors and acrylic boards.

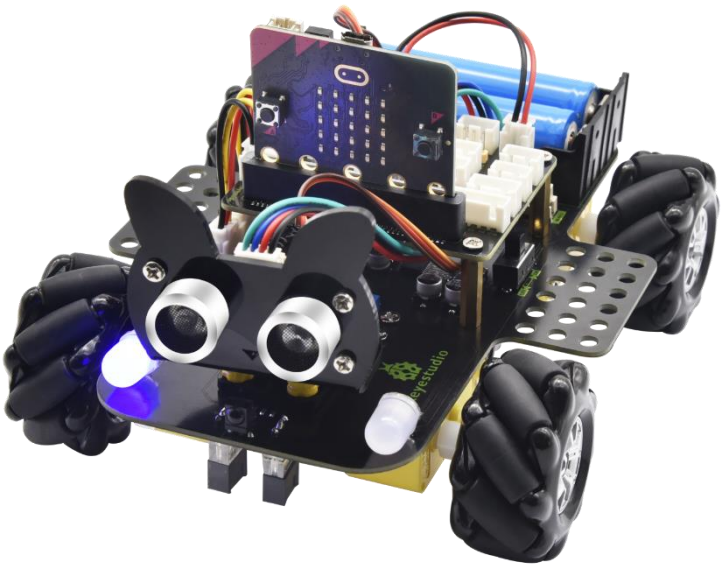

Figure II.1: KS4031(4032) Keyestudio 4WD Mecanum Robot Car for Micro:bit.

2.1. Keyestudio KS0528 PCB Driver Board

This driver board integrates an infrared tracking module, a motor module, an IR remote control module, four WS2812RGB lights to display different colors; two multicolor lights and one 18650 battery holder for supply power. Additionally, its 2.54mm anti-reverse interfaces are compatible with Micro:bit, Arduino and other control boards.

Plus, LEGO building blocks can be built up at its both sides. Its chip PCA9685PW and TB6612FNG are used to save IO ports. The TB6612FNG motor chip is used to control the rotation direction and speed of the four DC gear motors.

Specification

- Connector port input: DC 6V---9V
- Driver board system operating voltage: 5V
- Standard operating power consumption: about 2.2W
- Maximum power: 12W
- Operating temperature range: 0-50℃
- Size: 130.3*185*23mm
- Motor interface: PH2.0mm-2P
- Control interface: PH2.54 anti-reverse port
- Environmental attributes: ROHS

Pinout

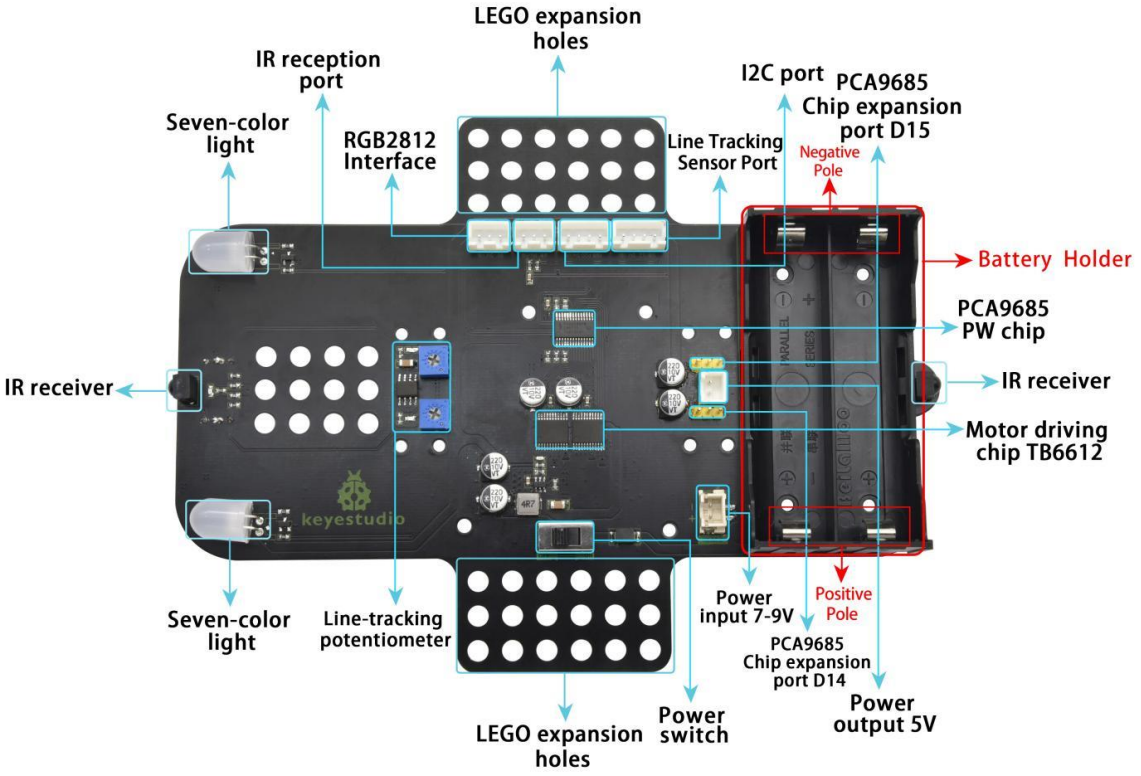

Figure II.2: Top view of the KS0528 Keyestudio PCB Driver Board.

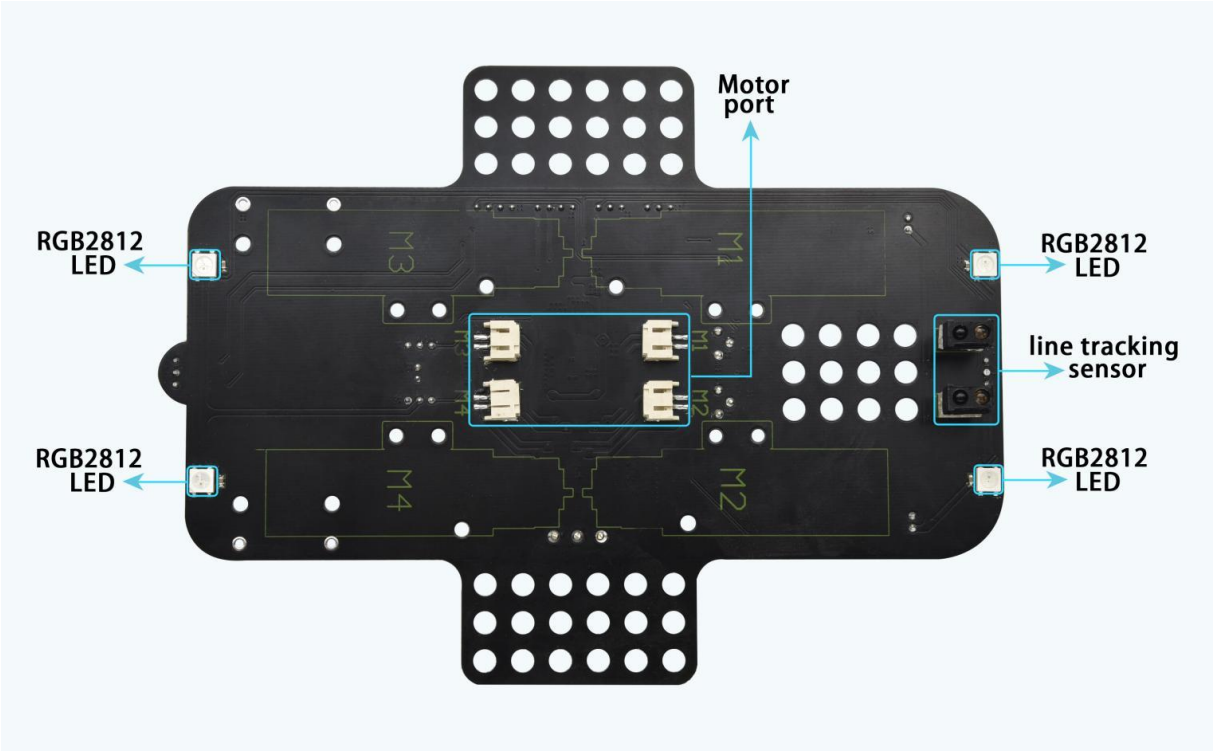

Figure II.3: Bottom view of the KS0528 Keyestudio PCB Driver Board.

Schematic Diagram

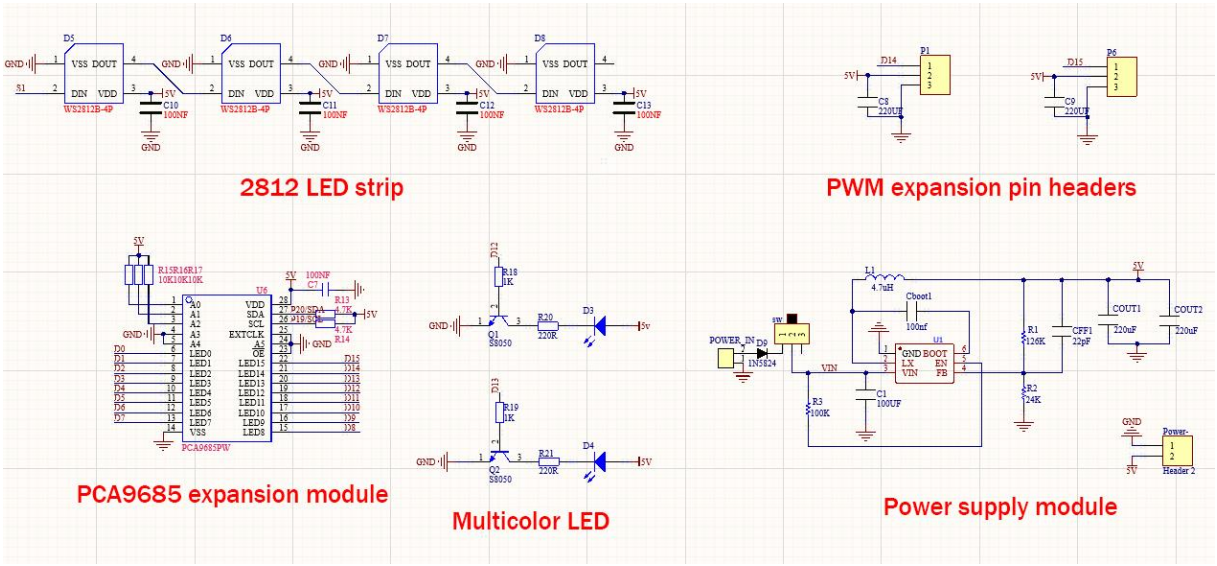

Figure II.4: Schematic diagram of some of the KS0528 Keyestudio PCB Driver Board modules.

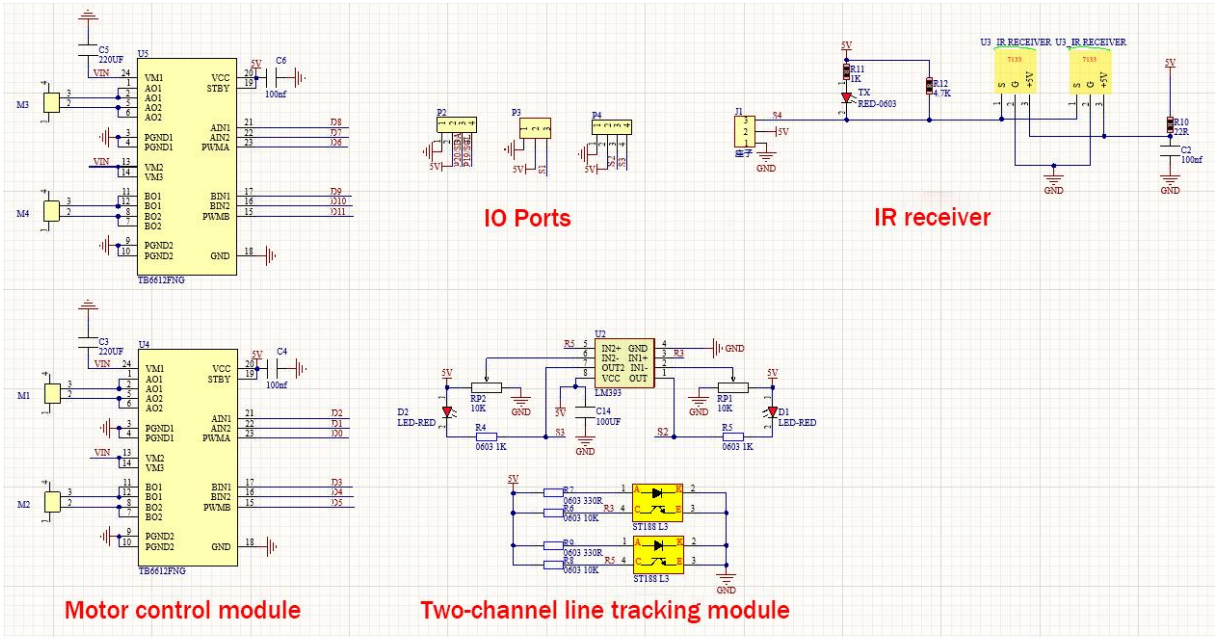

Figure II.5: Schematic diagram of some of the KS0528 Keyestudio PCB Driver Board modules.

3. The control platform

Our image processing algorithms need to be performed on-board. There are some embedded processing platforms known for that, like the NVIDIA Jetson, Odroid, Asus Tinker Board…

Based on availability once again, we decided to go with the Rapsberry Pi 4 model B 8GB RAM.

3.1. Raspberry Pi

Raspberry Pi offers a versatile set of tools for solving many automation challenge and it operates in the open source ecosystem, it runs Linux (a variety of distributions), and has its own supported operating system, Raspberry Pi OS.

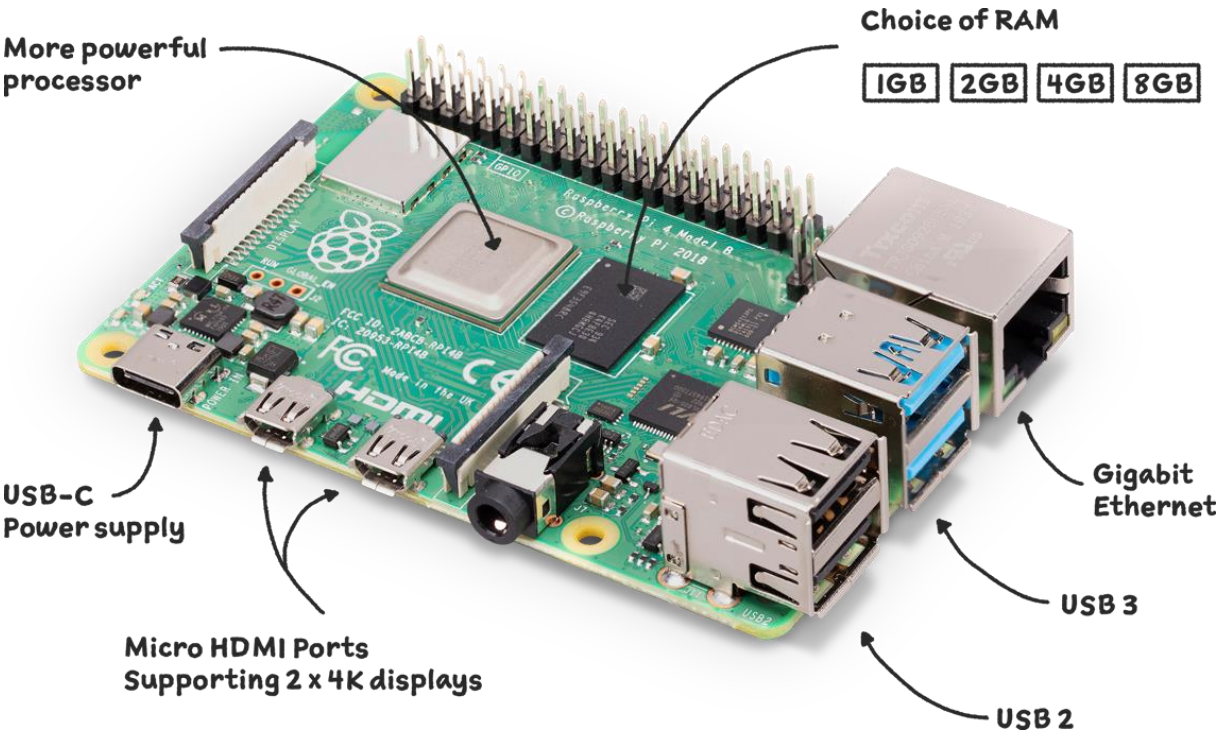

Figure II.6: Raspberry Pi 4 board.

Raspberry Pi 4 specifications

- Processor: Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memory: 8GB LPDDR4 with on-die ECC
- Connectivity: 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless, LAN, Bluetooth 5.0, BLE Gigabit Ethernet, 2 × USB 3.0 ports, 2 × USB 2.0 ports.
- GPIO: Standard 40-pin GPIO header (fully backwards-compatible with previous boards)
- Video & sound: 2 × micro HDMI ports (up to 4Kp60 supported), 2-lane MIPI DSI display port, 2-lane MIPI CSI camera port, 4-pole stereo audio and composite video port
- Multimedia: H.265 (4Kp60 decode); H.264 (1080p60 decode, 1080p30 encode); OpenGL ES, 3.0 graphics
- SD card support: Micro SD card slot for loading operating system and data storage
- Input power: 5V DC via USB-C connector (minimum 3A1), 5V DC via GPIO header (minimum 3A1), Power over Ethernet (PoE)–enabled (requires separate PoE HAT)

Additional components

The recommended ambient operating temperature range is 0 to 50°C. Since we deal with high computational power project (Computer Vision) that causes high CPU & GPU usage. A lot of heat is then generated on the surface when the system operates at full performance, thus high temperatures are reached, which require further cooling. That is why we need to add heatsinks and a cooling fan.

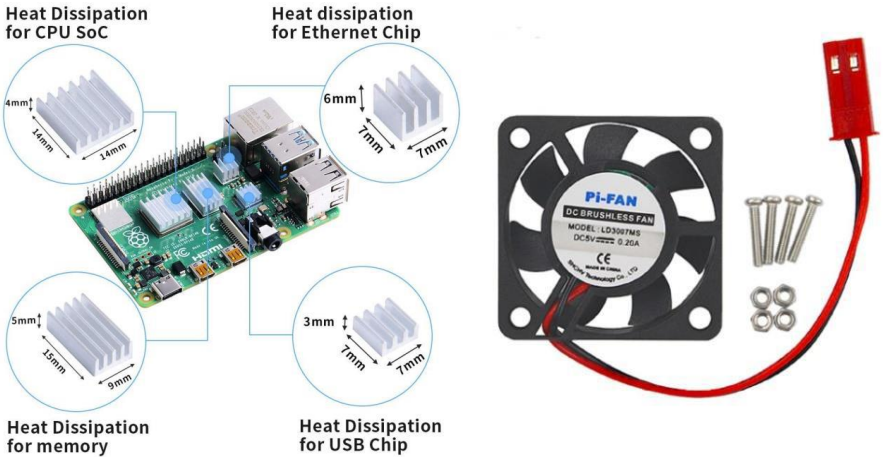

Figure II.7: Raspberry Pi 4 with heatsinks and cooling fan.

4. Camera module

There are a lot of cameras adapted for this use. We chose to go with the Raspberry Pi Camera Module Rev 1.3 for availability and compatibility with our Raspberry Pi.

4.1. Raspberry Pi Camera Module Rev 1.3

This camera module is specially designed for Raspberry Pi. It uses the dedicated CSI interface.

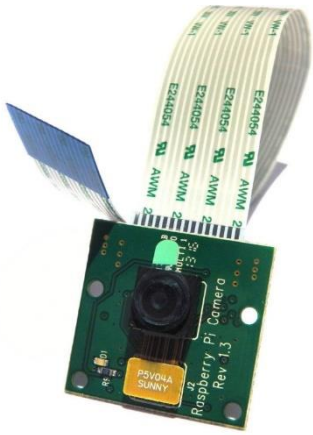

Figure II.8: Raspberry Pi Camera Module Rev 1.3.

Specifications

- Fully Compatible with Both the Model A and Model B Raspberry Pi
- 5MP Omnivision 5647 Camera Module
- Still Picture Resolution: 2592 x 1944
- Video: Supports 1080p @ 30fps, 720p @ 60fps and 640x480p 60/90 Recording
- 15-pin MIPI Camera Serial Interface - Plugs Directly into the Raspberry Pi Board
- Size: 20 x 25 x 9mm
- Weight 3g
- Fully Compatible with many Raspberry Pi cases

5. Software part

5.1. Operating system

Raspberry Pi OS has been specially designed for Raspberry Pi. It is a Linux operating system optimized for Raspberry Pi boards. It is known for its reliability, versatility, low power consumption and security. It also has some tools and programming languages preinstalled like C, C++ and Python.

5.2. Programming language and required libraries

We can use any programming language depending on personal preferences, but when it comes to AI, ML, DL and data science, Python is the most adapted choice.

The libraries that are going to be used are:

- **OpenCV-Python:**

OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

Python is a general purpose programming language started by Guido van Rossum that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: first, the code is as fast as the original C/C++ code (since it is the actual C++ code working in background) and second, it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

- **NumPy:**

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

- **Imutils**:

A series of convenience functions to make basic image processing functions such as translation, rotation, resizing, skeletonization, and displaying Matplotlib images easier with OpenCV and both Python 2.7 and Python 3.

- **Smbus:**

It supports the i2C protocol and several low-level read and write access methods. It accesses its host built-in smbus kernel module, from which it can get an I2C instance.

6. Conclusion

In this chapter, we presented the main components and technologies on which our project was based, their specifications, and the reasons that led us to choose them.

Chapter III : Implementation.

1. Introduction

In this chapter, we will proceed to the implementation of our project step by step. Explain how the connection and assembly are done, how the the system is developed and the logic behind its functioning, the problems encountered and the how we handled them.

2. Hardware part

2.1. Connection and electronics

To control the robot with the Raspberry Pi, there must be communication between the two. There are many communication interfaces that allow us to achieve that, but in our case we chose the I2C communication since the raspberry pi comes with one and the robot provides a I2C interface that allows us to communicate directly with the motors drivers.

The wiring is done between the SDA SCL Ground pins of the Raspberry Pi and those of the PCA9685 (the expansion module of th robot).

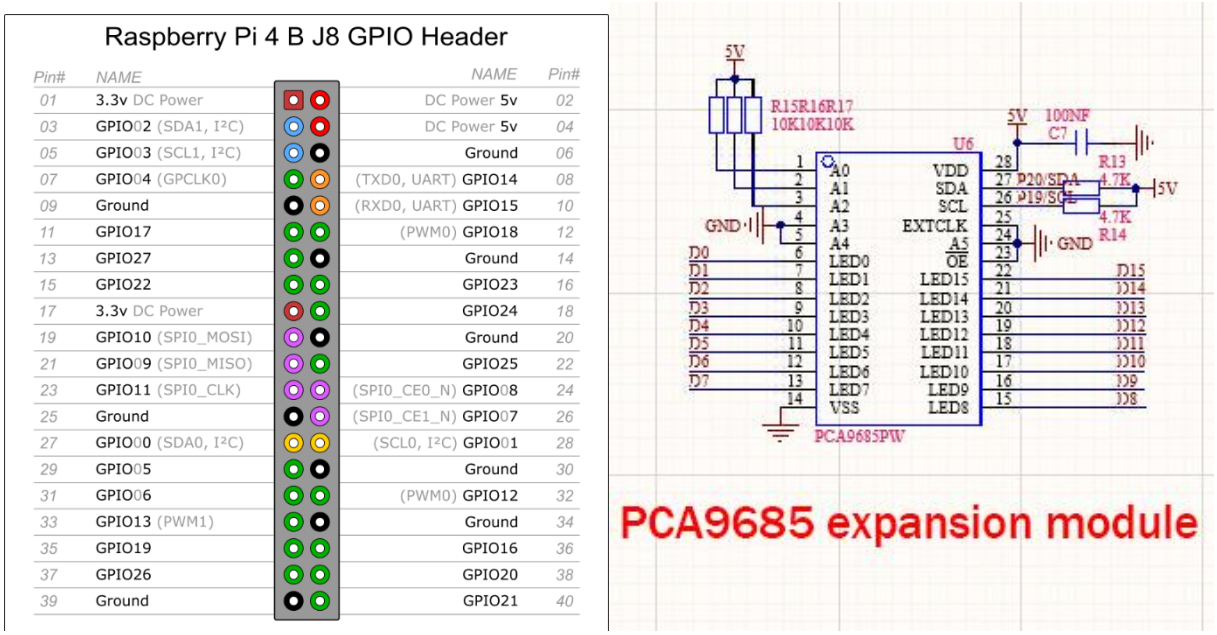

Figure III.1: Connecting pins of the Raspberry Pi (on the left) and of the PCA9685 expansion module (on the right).

2.2. Robot assembly

The smart car kit is designed for micro:bit. In order to control it with a Raspberry Pi, we need to apply some changes.

We assemble the robot following the instructions given by the constructor. We remove all the unnecessary parts and accessories to free up space and fix those that will be used instead.

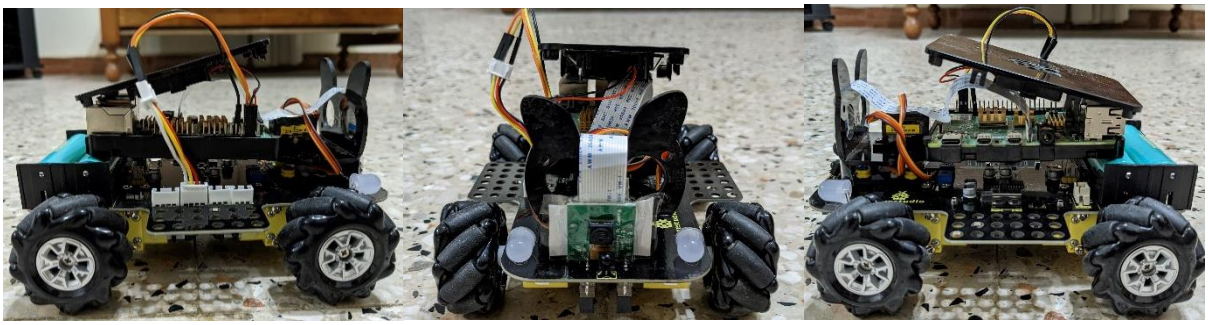

Figure III.2: The final aspect of the assembled robot.

3. Software part

3.1. Image acquisition and processing

The image acquisition is done by the Pi Camera. The image is then directly sent to processing.

The initial plan

We use tiny-YOLO detection system to process the image flow. The system resizes the input image to 320 x 320, runs a single convolutional network on the image, and thresholds the resulting detections by the model's confidence. We create a function that draws a box

around the detected object and writes its name above with the accuracy rate. We finish by displaying the result in realtime.

The system works and correctly detects the objects, but it is too slow. We can't even get 1 FPS. Which is unacceptable for our real-time application.

We try other detection systems (SSD MobileNet lite, TensorFlow lite), we optimize our algorithm, we reduce the used database to keep only one class of object, the simpliest one (tennis ball). The results get a little bit better (3-4 FPS) but are still unusable.

To conclude; the Raspberry Pi does not have enough computing power to perform this processing at the speed that we need.

The Raspberry Pi 4 being the only platform available, we only have two possibilities now, abandon the project at this stage, or replace the object detection system by an image processing algorithm, specially adapted to our object (tennis ball), without using deep learning and other computationally intensive techniques.

We choose to continue and finalize our project despite the very interesting part that we have to sacrifice.

The alternative

The target that we need to detect being a tennis ball, we can say that its specificity is its fluorescent yellow color, and we will base our model on this.

We start by defining the lower and upper boundaries of the ball in the HSV color space.

```
38    greenLower = (29, 150, 100)
39    greenUpper = (64, 255, 255)
```

Figure III.3: Defining the boundaries in the HSV color space.

We retrieve the video stream and perform some processing on it. We flip the frame (because of the camera layout), resize it, blur it and convert it to the HSV color space.

```
63      frame = cv2.flip(frame, 0)
64      frame = imutils.resize(frame, width=600)
65      blurred = cv2.GaussianBlur(frame, (11, 11), 0)
66      hsv = cv2.cvtColor(blurred, cv2.COLOR_BGR2HSV)
```

Figure III.4: Performing some processing on the video stream.

We construct a mask for the specific color, then perform a series of dilations and erosions to remove any small blobs left in the mask

```
70      mask = cv2.inRange(hsv, greenLower, greenUpper)
71      mask = cv2.erode(mask, None, iterations=2)
72      mask = cv2.dilate(mask, None, iterations=2)
```

Figure III.5: Construction the mask and performing dilation and erosion.

We find contours in the mask and initialize the current (x, y) center of the ball

```
76      cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
77                              cv2.CHAIN_APPROX_SIMPLE)
78      cnts = imutils.grab_contours(cnts)
79      center = None
```

Figure III.6: Finding contours and initializing the center.

If at least one countour was found, we proceed to the next step. We find the largest contour in the mask, then use it to compute the minimum enclosing circle and centroid.

```
87          c = max(cnts, key=cv2.contourArea)
88          ((x, y), radius) = cv2.minEnclosingCircle(c)
89          M = cv2.moments(c)
90          center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))
91          center_x = int(M["m10"] / M["m00"])
92          center_y = int(M["m01"] / M["m00"])
93          print("center : ", center, "radius : ", radius)
```

Figure III.7: Finding the largest contour and computing the minimum enclosing circle and centroid.

We update the points queue, loop over the set of tracked points and draw a circle around the object.

```
135     # update the points queue
136     pts.appendleft(center)
137
138     # loop over the set of tracked points
139     for i in range(1, len(pts)):
140         # if either of the tracked points are None, ignore
141         # them
142         if pts[i - 1] is None or pts[i] is None:
143             continue
144         # otherwise, compute the thickness of the line and
145         # draw the connecting lines
146             thickness = int(np.sqrt(args["buffer"] / float(i + 1)) * 2.5)
147             cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)
```

```
130                 cv2.circle(frame, (int(x), int(y)), int(radius),
131                            (0, 0, 255), 2)
132                 cv2.circle(frame, center, 5, (0, 0, 255), -1)
```

Figure III.8: Updating thethe points queue and drawing a circle around the object.

3.2. Robot control

We create a class where we initialize some parameters and declare some functions that we are going to use.

Class PCA9685

We start by setting all the addresses that we are going to use

```
    _MODE1 = 0x00
    _LED0_ON_L = 0x06
    _LED0_ON_H = 0x07
    _LED0_OFF_L = 0x08
    _LED0_OFF_H = 0x09
    _ALL_LED_ON_L = 0xFA
    _ALL_LED_ON_H = 0xFB
    _ALL_LED_OFF_L = 0xFC
    _ALL_LED_OFF_H = 0xFD
    _PRE_SCALE = 0xFE
```

```
def __init__(self, i2c_address=0x47, bus_num=1) -> None:
    """
    :param int i2c_address: The I2C address of the PCA9685 (default is 0x40).
    :param int bus_num: The SMBus instance of the I2C port (0 for the Raspberry Pi 1,
    and 1 for all future models).
    :return None
    """
    self._i2c_address = i2c_address
    self._bus = self.SMBus(bus_num)
    self._write_byte(self._MODE1, 0x00)
```

Figure III.9: Setting the addresses.

We set the ON and OFF positions of all channels to 0

```
34          self._write_byte(self._ALL_LED_ON_L, 0)
35          self._write_byte(self._ALL_LED_ON_H, 0)
36          self._write_byte(self._ALL_LED_OFF_L, 0)
37          self._write_byte(self._ALL_LED_OFF_H, 0)
```

Figure III.10: Setting the positions of all channels to 0.

We create the primary functions that will let us communicate with the motors drivers and write our instructions.

```
def set_duty_cycle(self, channel_num: int, duty_cycle) -> None:
    """Sets the duty cycle of a specified channel.
    :param int channel_num: The specified channel number (0 - 15).
    :param duty_cycle: The duty cycle in % (0 - 100)
    :return None"""
    off = round(duty_cycle * 4095.0 / 100.0)  # maps 0-100 to 0-4095
    on = 0
    self._write_channel(channel_num, on, off)

def _write_channel(self, channel_num: int, on: int, off: int) -> None:
    # if you want to write the starting "on" tick:
    self._write_byte(self._LED0_ON_L + 4 * channel_num, on & 0xFF)  # writes the lower 8 bits of on
    self._write_byte(self._LED0_ON_H + 4 * channel_num, on >> 8)  # writes the upper 4 bits of on

    # Each LED address is separated by 4 which is why we add a multiple of 4 to the LED0 address
    self._write_byte(self._LED0_OFF_L + 4 * channel_num, off & 0xFF)  # writes the lower 8 bits of off
    self._write_byte(self._LED0_OFF_H + 4 * channel_num, off >> 8)  # writes the upper 4 bits of off

def _write_byte(self, location, value):
    try:
        self._bus.write_byte_data(self._i2c_address, location, value)
    except IOError:
        print(f"IOError when writing to the bus at {hex(self._i2c_address)}.")
```

Figure III.11: Primary functions.

We then create the functions that will use the previous ones, but in a much easier way to read and understand. There will be 5 functions. MoveForward, MoveBackward, RotateRight, RotateLeft and Stop. Those are the ones that will be used in the main section. In each one of

those functions, there will be a combination of speeds and directions of rotation of all the motors. One of those functions will be called according to the logic of object tracking.

```
63      def MoveForward(self, value):
64          self.set_duty_cycle(1, 100)
65          self.set_duty_cycle(2, 0)
66          self.set_duty_cycle(0, value)
67          self.set_duty_cycle(3, 100)
68          self.set_duty_cycle(4, 0)
69          self.set_duty_cycle(5, value)
70          self.set_duty_cycle(7, 100)
71          self.set_duty_cycle(8, 0)
72          self.set_duty_cycle(6, value)
73          self.set_duty_cycle(9, 100)
74          self.set_duty_cycle(10, 0)
75          self.set_duty_cycle(11, value)
```

Figure III.12: Example of MoveForward function.

3.3. The logic of object tracking

In order to track our target and its movements, we need to precisely localise its position. To do this only with the image flow and without using any other technology, we need to divide our frame into a grid.

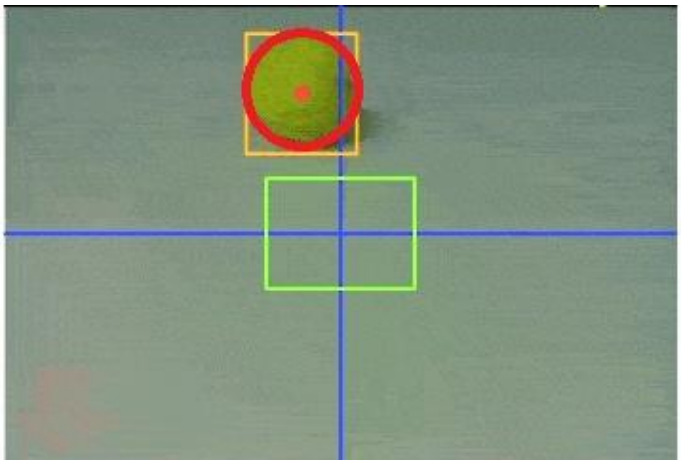

Figure III.13: View of the object tracking system.

Locating the object in the frame

We saw that the model provides the center of the object. Which is marked with a red dot. The dot follows the center as the object moves.

Defining the tolerance zone

Tolerance zone (green rectangle in the figure) is the area around the center of the frame within which the center of the object must fall in order to stop the tracking. As long as the red dot (center of the object) is outside the tolerance zone, the robot continues to move and track the object.

Moving the robot to minimise the deviation

Every time a new frame is processed, the center and radius values are calculated. Depending upon the location of the object in the frame, one of the following cases is handled by the code:

- If 250 < center < 350
  - If radius < 85

    The robot moves forward.
  - If radius > 115

    The robot moves backward.
  - else

    The robot does not move.
- If center < 250

  The robot rotates right.
- If center > 350

  The robot rotates left.

Conclusion

In this project, the main objective was to make an object tracking system. The goal was finally achieved, not exactly as we wanted, but finally achieved and the system is operational anyway.

First, we had to learn about those new technologies and understand its fundamentals. Then start the development et the tests.

In the beginning, we wanted to develop a human tracking system, which can be used in surveillance and many other applications. But given our limitations in terms of available hardware, we had to change our project and adapt several times. The size of the robot forced us to abandon the “human” part of the project and replace the target with another one with an adapted size. We also planned to use some techniques (deep learning) in our models, but again, the hardware did not allow us.

This project allowed us to improve our technical skills by discovering the field of artificial intelligence and its subfields, the python programming language, the Raspberry Pi development platform and the i2c communication bus. It also allowed us to learn to work in a group, to distribute tasks well and to be complementary.

There are several improvements that can be added to this robot. First by solving the already mentioned problems. Then, by adding for example a lidar sensor which will allow the robot to do some mapping and planification, which will make it even more autonomous.

Bibliography

"Integrating Stereo Vision with a CNN Tracker for a Person-Following Robot", By Bao Xin Chen, Raghavender Sahdev and John K. Tsotsos, In the 11th International Conference on Computer Vision Systems, Schezhen, China, July 10-13, 2017.

B. Karthikeyan M.E, L. R,, K. M. and M. R., "Real-time detection and tracking of human based on image processing with laser pointing," 2020 International Conference on System, Computation, Automation and Networking (ICSCAN), 2020, pp. 1-5, doi: 10.1109/ICSCAN49426.2020.9262270.

M. -C. Le and M. -H. Le, "Human Detection and Tracking for Autonomous Human-following Quadcopter," 2019 International Conference on System Science and Engineering (ICSSE), 2019, pp. 6-11, doi: 10.1109/ICSSE.2019.8823343.

K. R. Jayasree, P. R. Jayasree and A. Vivek, "Dynamic target tracking using a four wheeled mobile robot with optimal path planning technique," 2017 International Conference on Circuit ,Power and Computing Technologies (ICCPCT), 2017, pp. 1-6, doi: 10.1109/ICCPCT.2017.8074365.

H. Gao et al., "DupNet: Towards Very Tiny Quantized CNN With Improved Accuracy for Face Detection," 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), 2019, pp. 168-177, doi: 10.1109/CVPRW.2019.00026.

Ramey A, Malfaz M, Castillo JC, Castro-González Á, Pérez I, Salichs MA. A local user mapping architecture for social robots. International Journal of Advanced Robotic Systems. 2017;14(6). doi:10.1177/1729881417736950

T. Feng, Y. Yu, L. Wu, Y. Bai, Z. Xiao and Z. Lu, "A Human-Tracking Robot Using Ultra Wideband Technology," in IEEE Access, vol. 6, pp. 42541-42550, 2018, doi: 10.1109/ACCESS.2018.2859754.

O. Ferm, ‘Real-time Object Detection on Raspberry Pi 4 : Fine-tuning a SSD model using Tensorflow and Web Scraping’, Dissertation, 2020.

Shafiee, Mohammad Javad, Brendan Chywl, Francis Li and Alexander Wong. “Fast YOLO: A Fast You Only Look Once System for Real-time Embedded Object Detection in Video.” *ArXiv* abs/1709.05943 (2017): n. pag.

https://wiki.keyestudio.com/KS4031(4032)Keyestudio_4WD_Mecanum_Robot_Car_for_Micro:bit

https://wiki.keyestudio.com/KS0528_Keyestudio_4WD_Mecanum_Robot_Car_PCB_Driver_Board

https://biochimej.univ-angers.fr/Page2/COURS/Zsuite/6BiochMetabSUITE/5IntelligenceArtificielle/1IntelligenceArtificielle.htm

https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html

https://www.ibm.com/topics/computer-vision

https://helloworld.co.in/article/ai-robot-human-following-robot-using-tensorflow-lite-raspberry-pi

https://viso.ai/deep-learning/object-tracking/

https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/ai-vs-machine-learning-vs-deep-learning