TUNISIAN REUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage

National Institute of Applied Sciences and Technology

Graduation Project

to obtain the

National Engineering Diploma
in Applied Sciences and Technology

Specialty: Industrial Computing and Automation

Autonomous navigation using Machine Learning

Prepared by

BARGAOUI Nourchene

Hosted by **Center For Advancing Electronics Dresden**

Defended on the 22.07.2022 in front of a jury composed of:

Mr. Atef Boulila	President of the jury
Mr. Nejm Eddine Sifi	Reviewer
Mr. Akash Kumar	Supervisor at the Company
Mrs. Olfa Boubaker	Supervisor at INSAT

School year: 2021 / 2022

TUNISIAN REUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and Technology

Graduation Project

to obtain the

National Engineering Diploma
in Applied Sciences and Technology

Specialty: Industrial Computing and Automation

Autonomous navigation using Machine Learning

Prepared by

BARGAOUI Nourchene

Hosted by **Center For Advancing Electronics Dresden**

Supervisor at the Company **Pr. Akash KUMAR**	Supervisor at INSAT **Pr. Olfa BOUBAKER**
Date:	Date:

School year: 2021 / 2022

Dedication

To myself

to always hold on and continue to move forward and excel.
"Success is peace of mind in knowing you did your best to become the best you are capable of becoming" – John Wooden

To my Mother

The one that we have always called "mom", even as an adult, this magic word that is often the first one spoken. But to show it and express it, is often more complicated. I feel today this envy, this need, to leave a trace on paper, of my love for you. Thank you !

To Pr. Olfa BOUBAKER

What is the nobler and more valuable profession for a country, than that of the educator of the next generation?
The greatest lessons come not from books but from a teacher like you. Thank you for taking the time to help me through these last two years and for accompanying me in mastering my knowledge.

To my soulmate

Since I met you, I am totally fulfilled. Our happiness and our love are growing every day. No words, no texts will be able to match my love for you. You are and will remain the most beautiful thing that has happened to me.

To my Friend Ahmed ALOUINI

"Let us be grateful to the people who make us happy; they are the charming gardeners who make our souls blossom."– Marcel Proust.
Thank you for always being supportive !

To all my Friends and Family

Thank you for supporting me through all my life. I love you all !

NOURCHENE

Acknowledgments

I would like to thank the CFAED family for the opportunity they gave me to enhance my abilities and my knowledge.

Special thanks and gratitude to my german supervisor, Pr. Akash KUMAR for his help and constant support during all these six months.

My sincere gratitude goes to my Tunisian supervisor Pr. Olfa BOUBAKER for being an amazing academic supervisor, supportive and kind.

This work is supported by a grant from the Deutscher Akademischer Austauschdienst (DAAD) of Germany.

We gratefully acknowledge the support of the Center of Advancing Electronnics Dresden (CFAED) of the equipment used for this research.

I particularly thank the INSAT family for providing the most ideal environment for students to grow and achieve at high levels.

Finally, I would like to express my sincere acknowledgement to the members of the jury for agreeing to evaluate my work and for their valuable comments.

Abstract

This work conc This work concerns the implementation of an algorithm for autonomous navigation for an Hexapod robot based on a Semantic segmentation model using a Cityscape database. This work is a preliminary and will lead later to an implementation on a real car.

Throughout this project, we will start by the determination of the necessary computer software and equipment required. Followed by the assembly of a Hexapod which will be used in the essential task which is the implementation of an autonomous navigation solution. The geometric models (Inverse and Forward kinematics models) will also be detailed.

This proposed Autonomous navigation project aims to provide a machine learning algorithm using Python language trained on image classification, using OpenCV and TensorFlow, to the task of semantic segmentation by applying the conventional neural network with City Scape dataset.

We further extend it to deep learning algorithm to perform the lane assistance, which encodes the road features to help drivers stay in the middle of the lane. To achieve accurate and precise predictions as well as detailed segmentation images, we also combine ideas from the mediated perception approach and the behavioral reflex approach.

Experimental results have proven that the system proposed in this project is significantly ahead of the state of the art in several challenging research projects with open-source datasets, including PASCAL VOC and PASCAL-Person-Part.

Keywords: Facial recognition, Hexapod, geometric model, semantic segmentation, Python, TensorFlow, OpenCV, Artificial intelligence, robot navigation.

Table of content

L

List of figures

List of tables

L

List of acronyms

DAAD : Deutscher Akademischer Austauschdienst

CFAED : Center For Advancing Electronics Dresden

TUD : Technische Universität Dresden

TCP : Transmission Control Protocol

IP : Internet Protocol

LAN : Local Area Network

iOS : iPhone Operating System

Enet : Efficient Neural Network

Resnet : Residual neural network

PSPnet : Pyramid Scene Parsing Network

VOC : visual object class

FCN : Fully Convolutional Network

CRF-RNN : Conditional Random Fields as Recurrent Neural Networks

DPN : Dual Path Network

OpenCV : Open Computer Vision

IoU : Intersection over union

VGG : Visual Geometry Group

General Introduction

An autonomous robot, also known simply as an autorobot or autobot, is a robot that performs behaviors or tasks with a high degree of autonomy (without external influence). Autonomous robotics is generally considered a subfield of Artificial Intelligence, Robotics and Information engineering.

Autonomous robots are particularly desirable in areas such as spaceflight, housekeeping (such as cleaning), wastewater treatment, and the provision of goods and services. Among the sectors that are working the most on autonomous robots are: Automotive or especially autonomous cars.

An important area of robotics research is enabling the robot to cope with its environment. It is in this context that this work aims at presenting in a rigorous way the development and implementation of an autonomous car capable of navigating a given path and distinguishing.

In order to achieve the mentioned-above objective, an artificial intelligence algorithm for semantic segmentation will be developed so that the robot can recognize its environment.

More importantly, to be able to recognize the owners of the robot, it requires the implementation of a Machine Learning algorithm for facial recognition linked to the user interface of the robot.

Finally, we will have a robot able to distinguish its environment with Semantic Segmentation, its path with lane assistance and even better its owners with deep learning facial recognition.

This report summarizes the different stages of the project's life cycle.

We have four different chapters:

- Chapiter 1 *"Project context"* in which the host organization will be presented as well as the project's topic and sub-problems.
- Chapiter 2 *"Design and Architecture"* in which we will present the mechanical design of the robot and its assembly.

- Chapiter 3 *"Software Development"* in which we will describe the implemented AI code with python.
- Chapiter 4 *"Testing and achievements"* in which a test and verification of the efficiency of the project will be evaluated.

This report will be concluded by a general conclusion that will summarize this work done during this internship and presents some further perspectives.

1

Project context

Introduction

In this chapter, the context of the project will be presented, the host institution unit will be introduced and the problematic around which the project revolves will be described.

This chapter is composed of two parts. In the first part, we will present the *Processor Design* unit, its organization, its activities and the axes on which it works. Then, in the second part of this chapter, we will address the task of presenting the project context, delimiting its scope and establishing the issue on which we will perform all the work.

Keywords: Machine Learning; autonomous navigation; mobile robot; mediated perception; behavior reflex.

1. Host organization

In this part, we will present the host institution and its different chairs as well as the host chair with whom we will perform this work.

1.1. The Host University

This project was carried out inside the Center for Advancing Electronics Dresden in the Technische Universität Dresden [1]. It is a prestigious German university, founded in 1828 and located in Dresden, Saxony. TUD is one of the most important universities in Germany for scientific research. The official logo of the organization is in the following figure 1.1.

TECHNISCHE UNIVERSITÄT DRESDEN

Figure 1.1 Technische Universität Dresden

1.2. The Host institution

1.2.1 Presentation of the research center

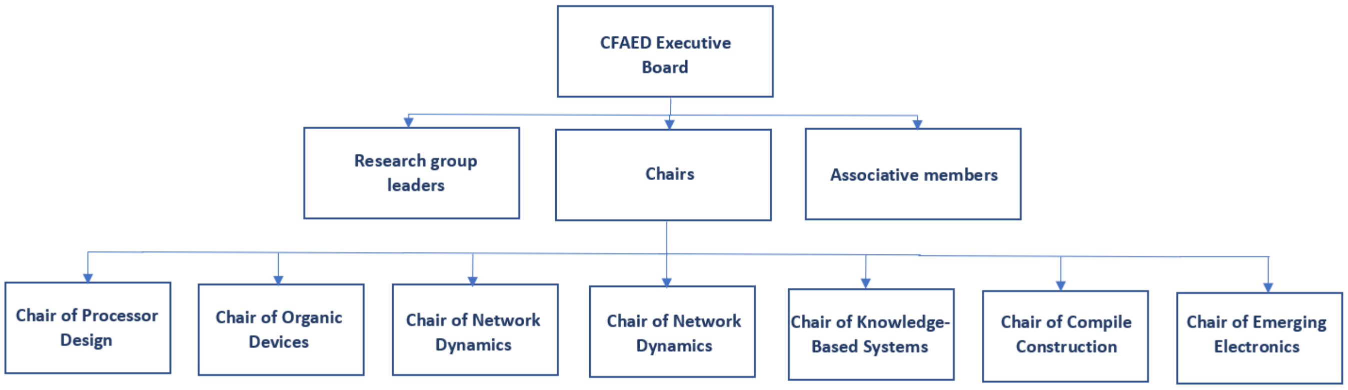
The Center for Advancing Electronics Dresden (CFAED) [2], within the TUD, is one of the well-known centers for scientific research and took part of the "Excellence Initiative of German universities". The present project will be carried out inside this prestigious institution. The official logo of the organisation is in the following figure 1.2.

cfaed CENTER FOR ADVANCING ELECTRONICS DRESDEN

Figure 1.2 The Center for Advancing Electronics Dresden

1.2.2 Constitution of the host organization

The Center for Advancing Electronics Dresden (CFAED) is composed of an executive board, a group of Research Leaders which is composed from the most important scientific researchers in the center, a group of associate members which is composed from the secretary of financial managers and last but not least eight research departments (Figure 1.3) operating in different fields. This project will be processed in the Chair of Processor Design.



CFAED Executive Board
Research group leaders
Chairs
Associative members
Chair of Processor Design
Chair of Organic Devices
Chair of Network Dynamics
Chair of Network Dynamics
Chair of Knowledge-Based Systems
Chair of Compile Construction
Chair of Emerging Electronics

Figure 1.3 Flowchart of the CFAED

1.3. The German Cooperation

The German Academic Exchange Service (Deutscher Akademischer Austauschdienst) [3] promotes TUD through priority programs in several research units and research groups, and this project will be funded by the DAAD. The official logo of the organization is in the following figure 1.4.

DAAD

Deutscher Akademischer Austauschdienst
German Academic Exchange Service

Figure 1.4 The German Academic Exchange Service

1.4. Project description

The aim of this collaboration is to design and implement a Machine Learning algorithm able to detect a path and its obstacles and which will be introduced in Robot-Car. Freshly launched, the goal is to move from a theoretical work to a real computer tool. This tool analyses the working area and provides all the necessary elements to find the navigation path according to certain criteria. This tool can be used in real time. As a result, the navigation plan can be re-configured when a problem arises during the driving operation of this robot-car.

We will now begin the analysis phase of the existing situation in order to obtain full visibility of the project. This will allow us to determine the major problematic of the project and to deduce the action plan to be realized during the internship.

1.5. State-of-the-art survey

This project is based on the principle of autonomous driving of a car using Machine Learning. Today, there are two major paradigms for vision-based autonomous driving systems: The approach called *mediated perception* [4] and another approach called *behavior reflex* [4].

The mediated perception approach [6] starts by analyzing the whole scene in order to make a driving decision. However, the behavioral reflex approach that directly associate an input image with a driving action by a regressor [4] like it is shown in the figure 1.4.

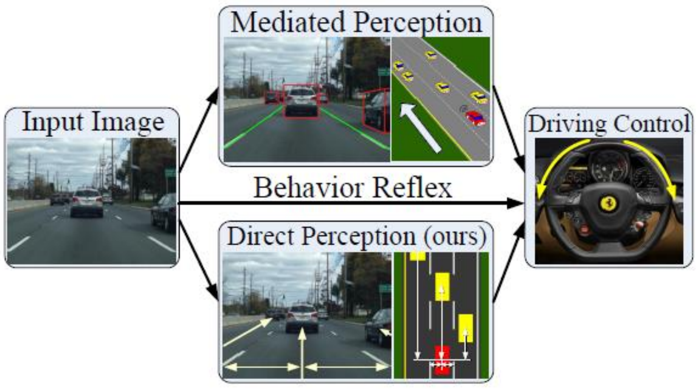


Figure 1.5 The autonomous driving paradigms

These approaches have been essentially explored by the international company for electric cars Tesla [5]. A simple illustration can be found in the next figure 1.5.

Figure 1.6 The autonomous driving paradigms

Solutions to many open challenges, that have been found for general understanding of the scene to solve the problem of controlling a robot-car, increases the complexity and cost of a system.

Current and most responsive autonomous driving systems are based on *mediated perception* approaches [6]. However, these algorithms for lane and path detection can lead to some false detections. For example, solid structures such as large road signs, cracks in the surface of sidewalks surface can be misidentified as lane markings and can lead the AI algorithm to false decisions [4].

Here, in this figure 1.6, we can see that those two driving cases depends on an ego-centric perspective. The driving decisions are made based on the lanes monitored that are marked with light gray.

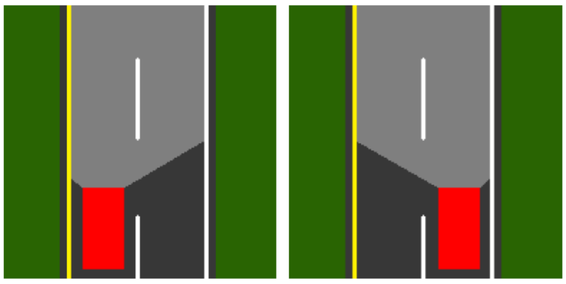


Figure 1.7 (a) two-lane Left (b) two-lane Right

Assuming that efficient results for lane detection have been achieved, some of the essential information for locating the car might be missing. For example, since only two-lane markings are usually detected. However, it would be difficult to recognize whether a car is driving in the right or left lane in a two-lane road.

1.6. Project goal

Given the problems and requirements mentioned, the objective of this internship is the development of an algorithm that will allow a car to navigate autonomously in a known environment using a combination of the two approaches mentioned above. Later, the algorithm will be simulated in a specific software to test and compare results. At the end, we will be able to use a rea- small robot-car to test the efficiency of the Machine Learning algorithm.

1.7. Methodology

Since software development platforms are constantly evolving, the objective is to adapt to the intrinsic and extrinsic constraints of the robot that are necessary for the proper conduct of the project. To meet the specificities of the specifications, this part will present the action plan adopted for the development of the solution.

1.7.1 The spider robot

The following paragraph provides a more detailed look at the biological systems utilized in robot designs.

Two observations can be made : a human-being form and Insects form.

17.1.1 Biological inspiration

1.7.1.1.1 Insects

The Hexapod robot is a 6-legged robot where each leg consists of three rotating joints to imitate the structure of a spider insect [16 , 17]. Insects are among the most successful creatures on Earth. The configuration of the 6-legged robot is inspired by the legs of insects. The legs of an insect are composed of 5 parts which are: Tarsus, Tibia, Femur, Trochanter and Coxa as shown in the next Figure 1.7.

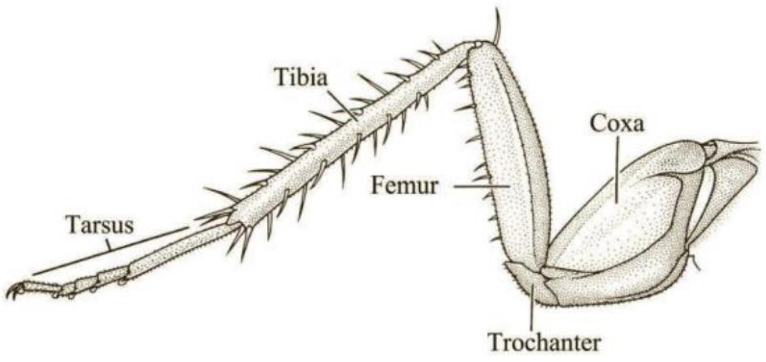


Figure 1.8 Anatomical structure of a typical insect leg

1.7.1.1.2 Human beings

The modern world’s expectations of robotic systems have continued to rise day by day. Indeed, today's robots are required to be flexible, predictable and intelligent.

To meet those specific demands, a robot must be designed intelligently. Nature has always inspired us especially in the robotics field. As a proof, roboticists inspired themselves from human beings to produce intelligent robotics systems. As shown in Figure 1.8, the arm and shoulder of a human inspired roboticists [18] to create a robot with legs similar to human arm with joints.

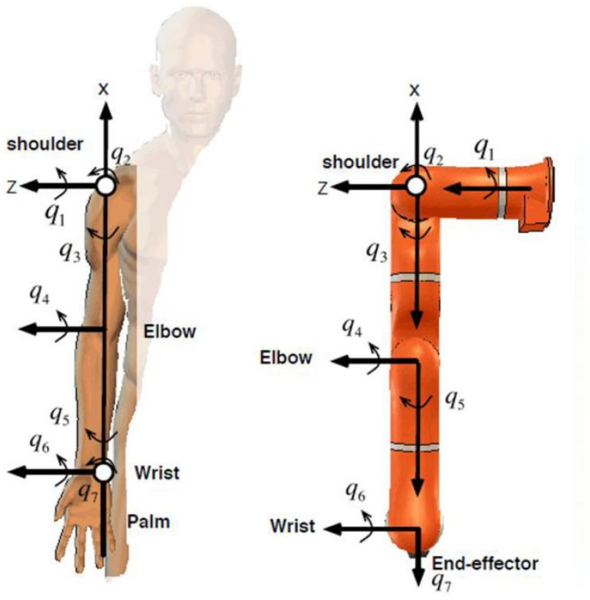


Figure 1.9 Anthropomorphic Arm [Wang and Artemiadis, 2013]

Compared to what precedes, the Figure 1.9 down below shows the structure of a Hexapod with 6 legs.

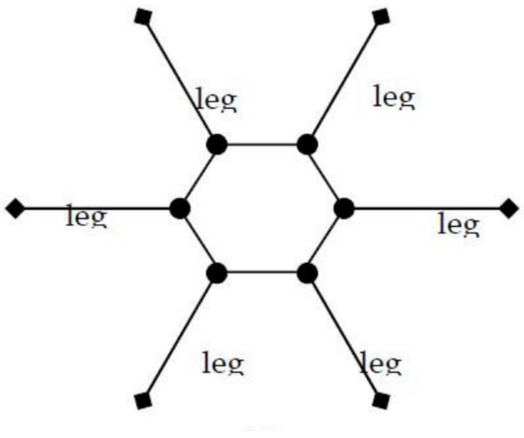


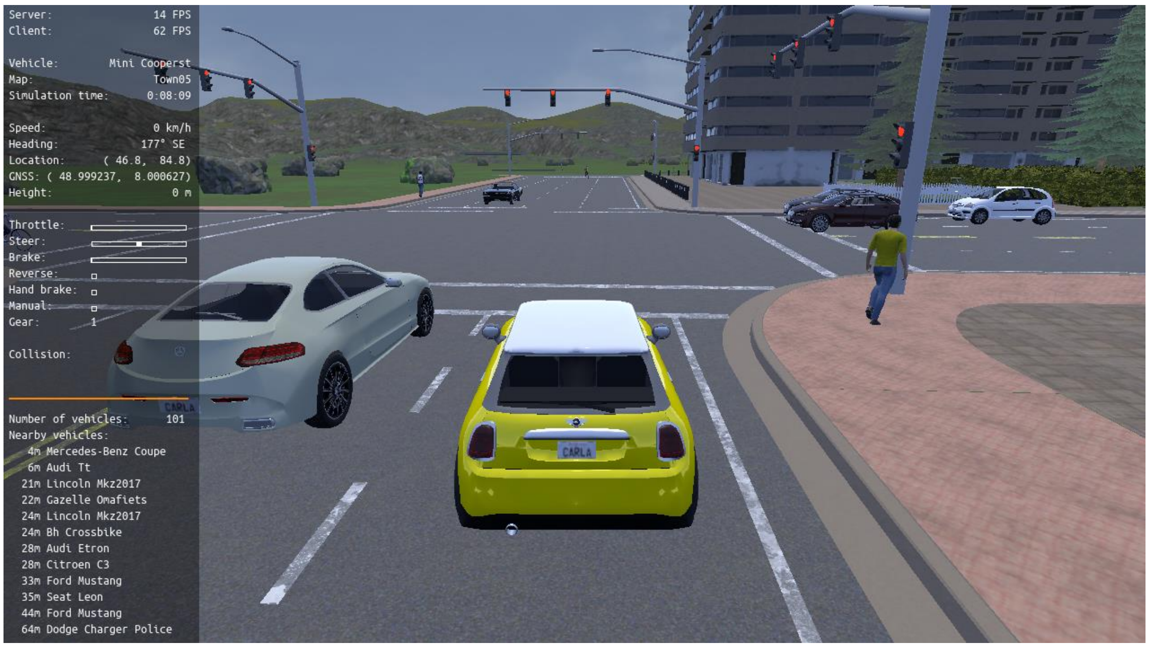
Figure 1.10 Hexapod simple design

17.1.2 Carla Simulator

CARLA [7] has been designed and developed to enable programming, training and testing of autonomous driving cars. It is a virtual environment where engineers can develop and simulate their work in an environment similar to a reality. The official logo of the organization is in the following figure 1.10.

Figure 1.11 Carla official Logo

Having the advantage of available open-source codes, CARLA provides a detailed set of obstacles (static and dynamic) that can be found in the real world (plants, vehicles, pedestrians [12], buildings...). The Carla environment can be shown in Figure 1.11.



```
Server:                14 FPS
Client:                62 FPS

Vehicle:         Mini Cooperst
Map:                    Town05
Simulation time:       0:08:09

Speed:                0 km/h
Heading:             177° SE
Location:       ( 46.8,  84.8)
GNSS: ( 48.999237,  8.000627)
Height:                   0 m

Throttle:
Steer:
Brake:
Reverse:
Hand brake:
Manual:
Gear:         1

Collision:

Number of vehicles:      101
Nearby vehicles:
   4m Mercedes-Benz Coupe
   6m Audi Tt
  21m Lincoln Mkz2017
  22m Gazelle Omafiets
  24m Lincoln Mkz2017
  24m Bh Crossbike
  28m Audi Etron
  28m Citroen C3
  33m Ford Mustang
  35m Seat Leon
  44m Ford Mustang
  64m Dodge Charger Police
```

Figure 1.12 Carla environment

1.7.2 Jupyter platform

To develop the Machine Learning algorithm, we will use the platform of Jupyter Notebook [8]. It is a development environment with virtual city maps for programming in python language.The official logo of the organization is in the following Figure 1.12.

Figure 1.13 Jupyter official Logo

1.8. Action plan

Given the details illustrated above, we will work on the following tasks:

➢ **Preliminary project study (research aspect)**
  - Bibliographic research on autonomous driving.

    - Overview of the market and similar projects.
    - State-of-the-art research
- **Assembly and development on the hardware**
    - Mechanical assembly of the robot.
    - Defining Inverse kinematics model.
    - Testing equipment.
- **Conception and development of the Machine Learning algorithm**
    - Select the most appropriate database with the features we need.
    - Identify existing issues in the mentioned database.
    - Extract the dataset that will be used in the training model.
    - Implement the model of classification for neural network.
    - Evaluate the resulting model.

1.9. Conclusion

This part of the document presented the context of the project. In this chapter, we discussed several concepts of artificial intelligence applied of an Hexapod robot, inspired by biological systems existed in the nature. We presented two important artificial intelligence approaches "Behavioral reflex" and "Mediated perception" which are the most used approaches in the automotive modern industry. These specific approaches allowed us to elaborate a strong plan for this application.

We also presented the action plan as well as the required software and tools that we have followed to achieve the tasks illustrated above in order to reach the desired objectives in the next two chapters.

2

Design and architecture

Introduction

This second chapter is the entry point to this graduation project and consists of four main goals starting with an overview of the designed robot and its architecture. This study is to specify the functionalities and constraints of the system. Then, the Hexapod robot of FREENOVE will be described as well as its geometric models (Forward and Inverse kinematics models) on which we have focused to make it move from an initial position to a further position. Furthermore, the mechanical conception of the robot will be detailed along with its different components. Finally, the communication between the machine and the robot is established with a LAN network using a server that will be described at the end of the chapter.

Keywords: Inverse Kinematic model; forward kinematic model; calibration.

2. Presentation of the Robot

2.1 Hexapod overview

FREENOVE [14] is an open-source electronics platform providing robotics devices for research and academic purposes. The related company helps and supports its clients and users in coding and learning robotics so that they can turn their exiting ideas into new, modern and advanced concepts and prototypes. Many of FREENOVE products were used by customers all over the world and one of the most famous products is the Hexapod robot [15] (or spider robot). The Hexapod is a six-legged robot equipped with sensors and servo to facilitate its movement. Figure 2.1 provides an overview on the Hexapod product. For the Figure 2.2, it presents the official logo of the company.

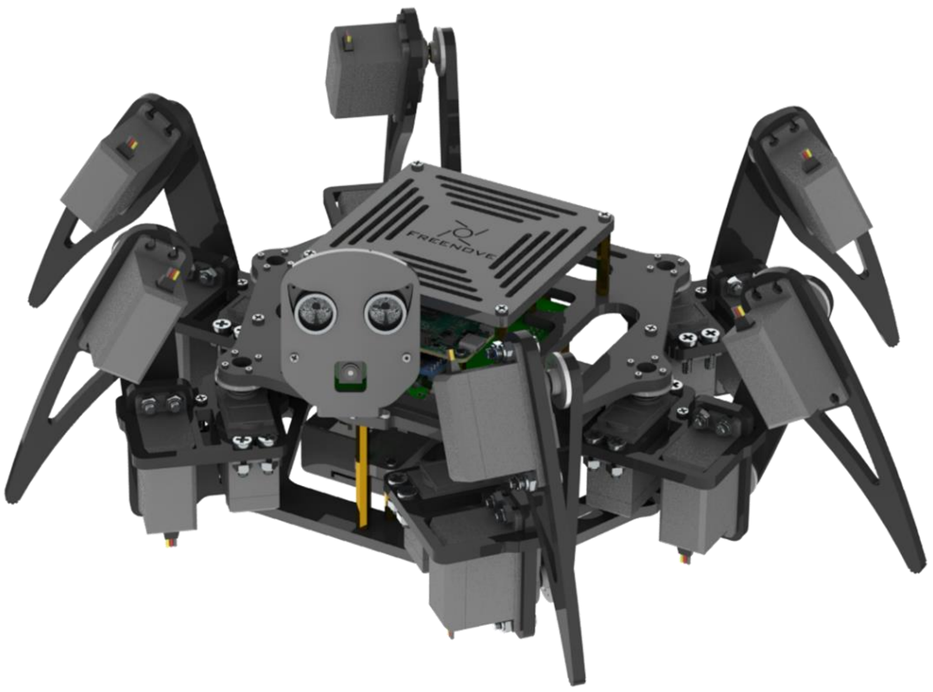


Figure 2.1 Hexapod FREENOVE

™

FREENOVE

Figure 2.2 FREENOVE company official logo

2.2 Mechanical design

The mechanical design of the robot is open source and is developed by the company FREENOVE. Figure 2.3 presents an overview on the different parts of the Hexapod robot. There are 6 legs, 12 servo-base for the legs, 3 pieces for the center of the robot (the body on which we put the electronic cards), and finally 2 pieces for the head.

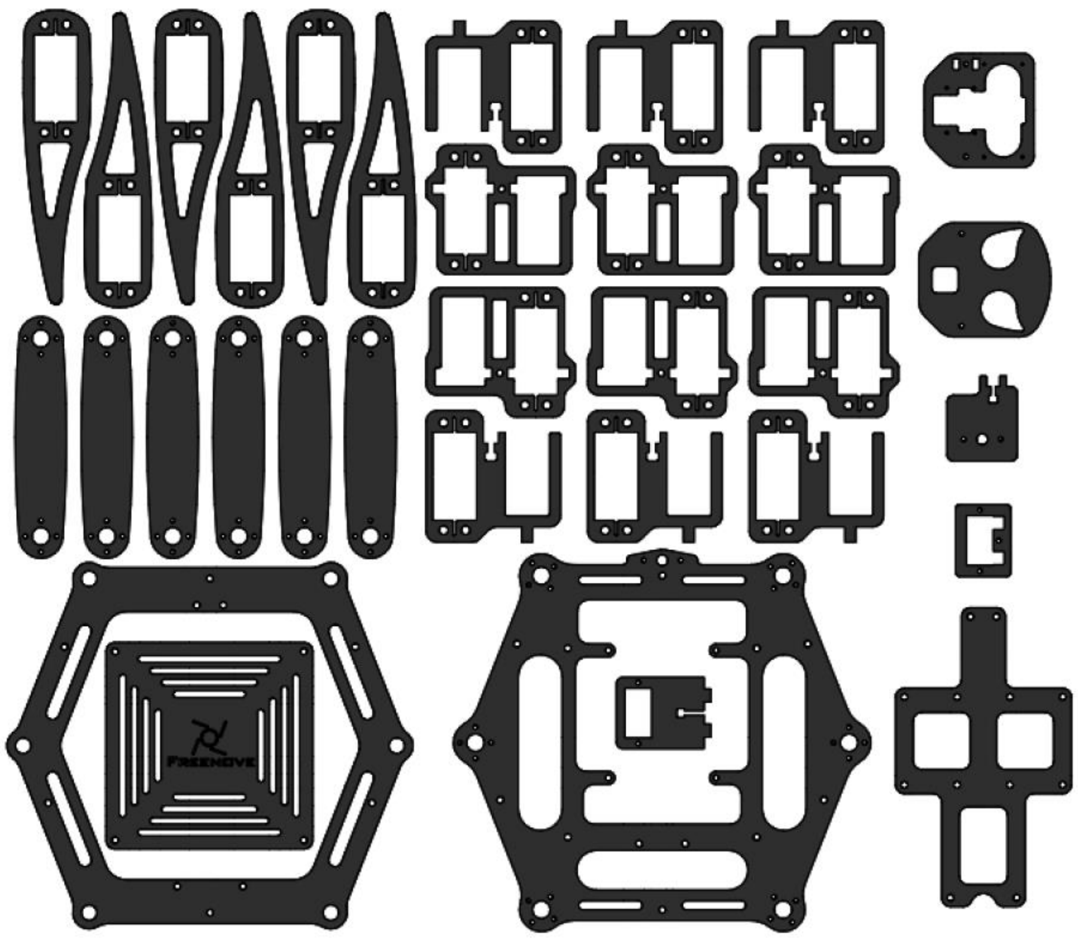


Figure 2.3 Mechanical design for Hexapod robot

2.3 Assembling the robot

The assembly of the robot starting from installing the servo base, assembling legs with servos, assembling the center of the robot, installing the battery holder and finally assembling legs and head to the body of the robot. Figures 2.4, 2.5, 2.6, 2.7, 2.8 and 2.9, in the next pages, present an overview of the process of assembling the Hexapod robot.

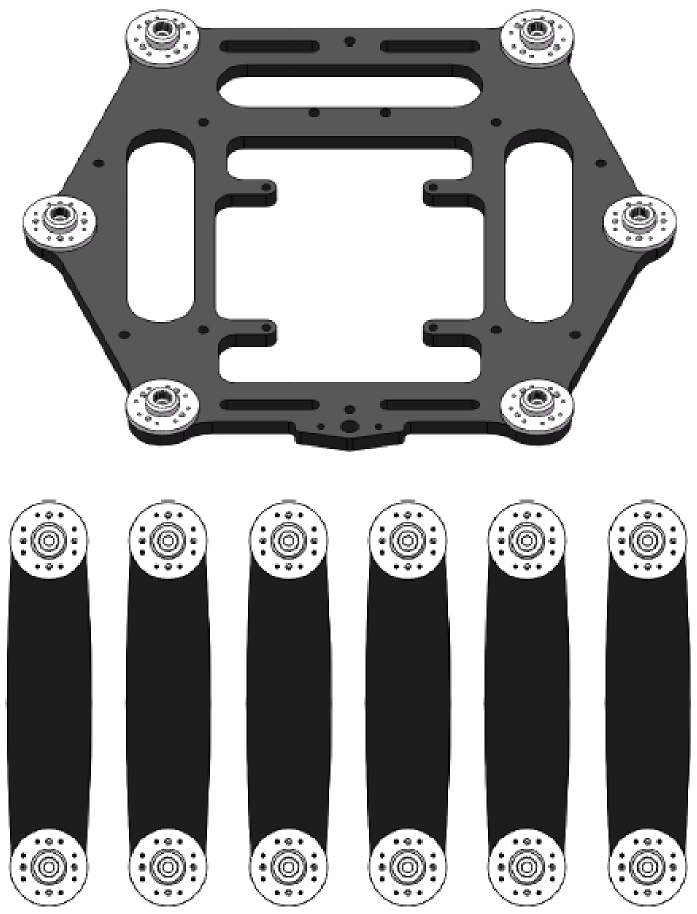


Figure 2.4 Installing the servo base

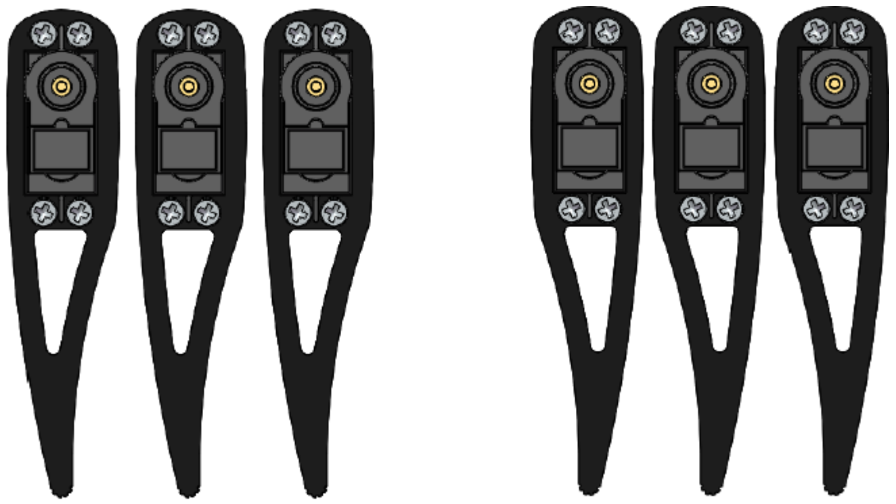


Figure 2.5 Assembling legs with servos

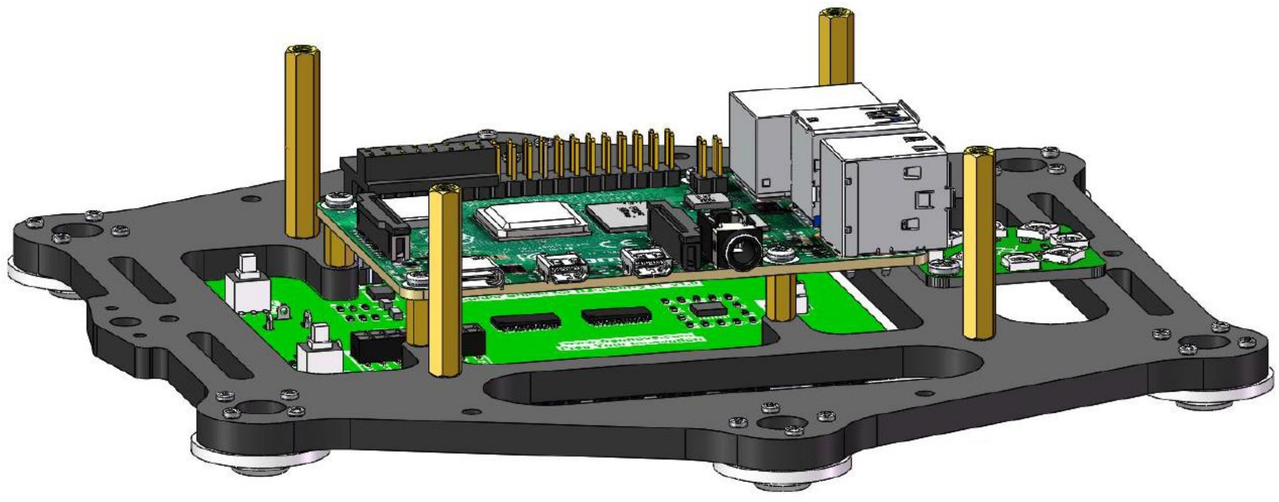


Figure 2.6 Assembling the center of the robot

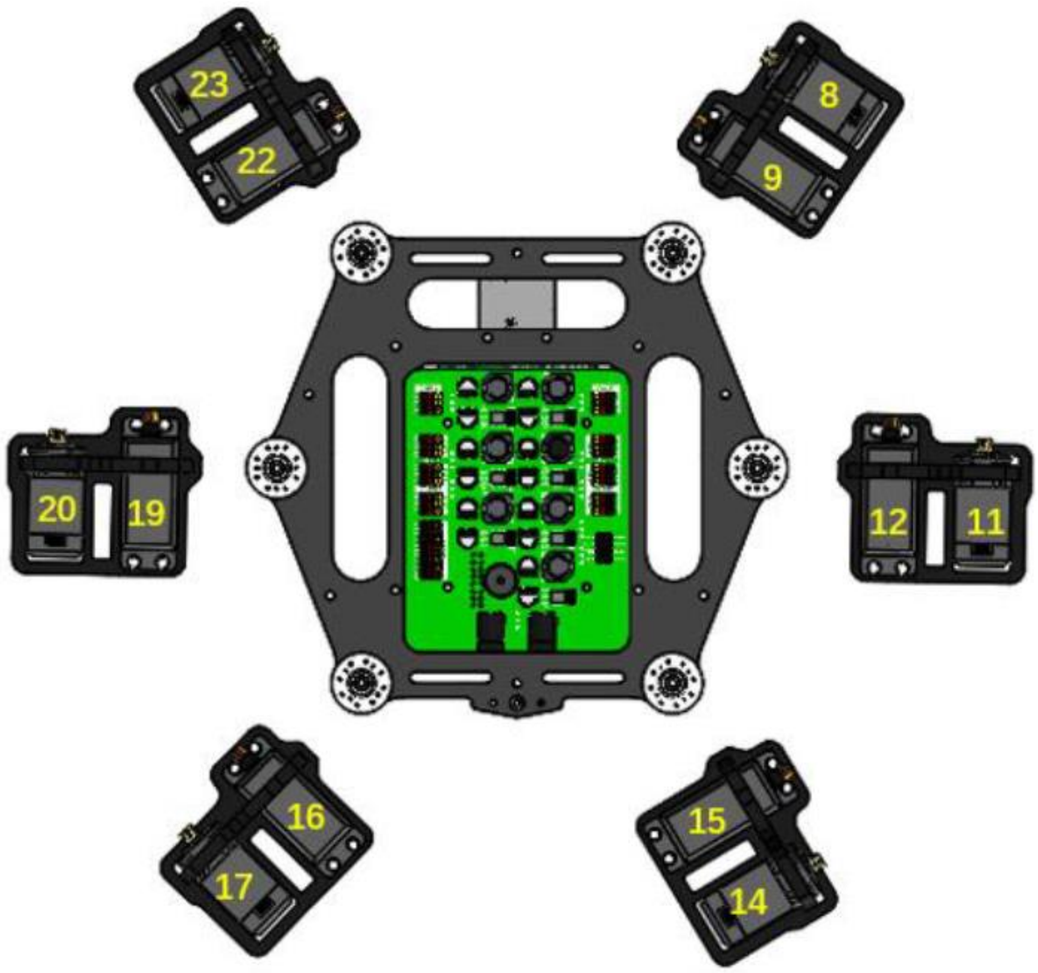


Figure 2.7 Assembling body and legs (Bottom view)

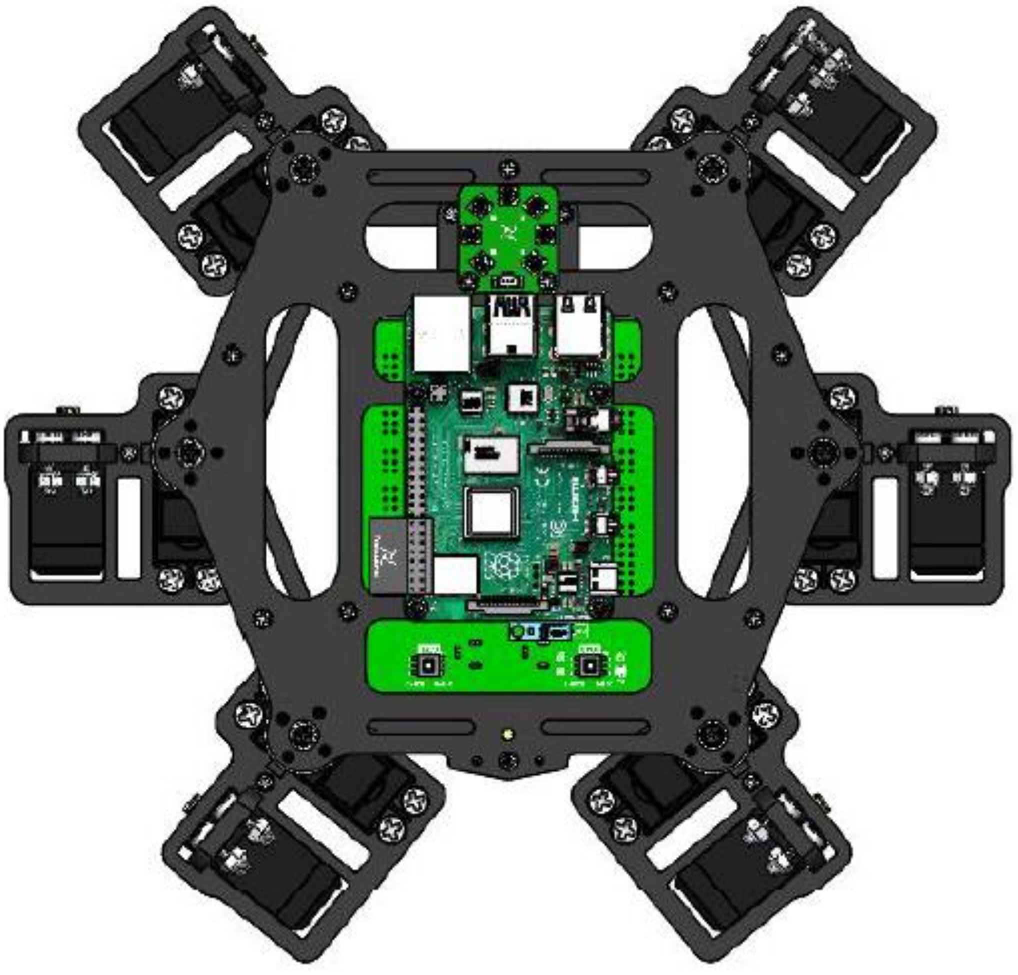


Figure 2.8 Assembling body and legs (Top view)

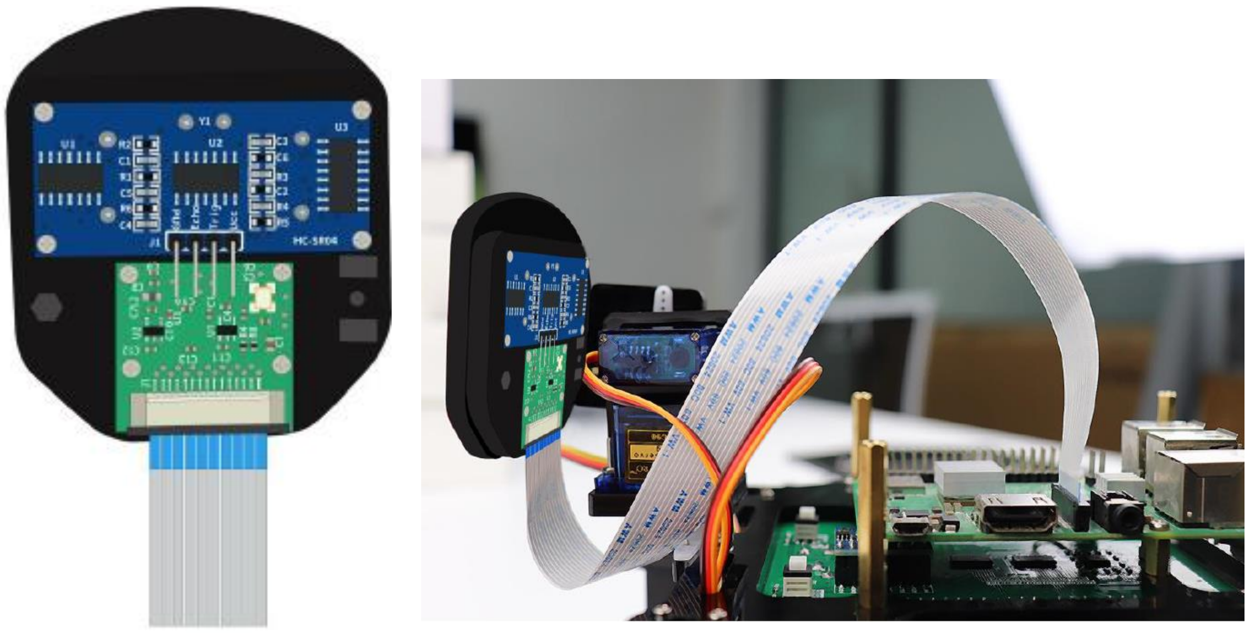


Figure 2.9 Head assembly

2.4 Calibration

In the previous paragraph, we have assembled the robot. In this section, we will calibrate the legs and test all modules (led, ultrasonic, buzzer, Picamera).

To do this, we need to place the robot on horizontal table on which we already put the calibration paper like shown in the Figure 2.10 below.

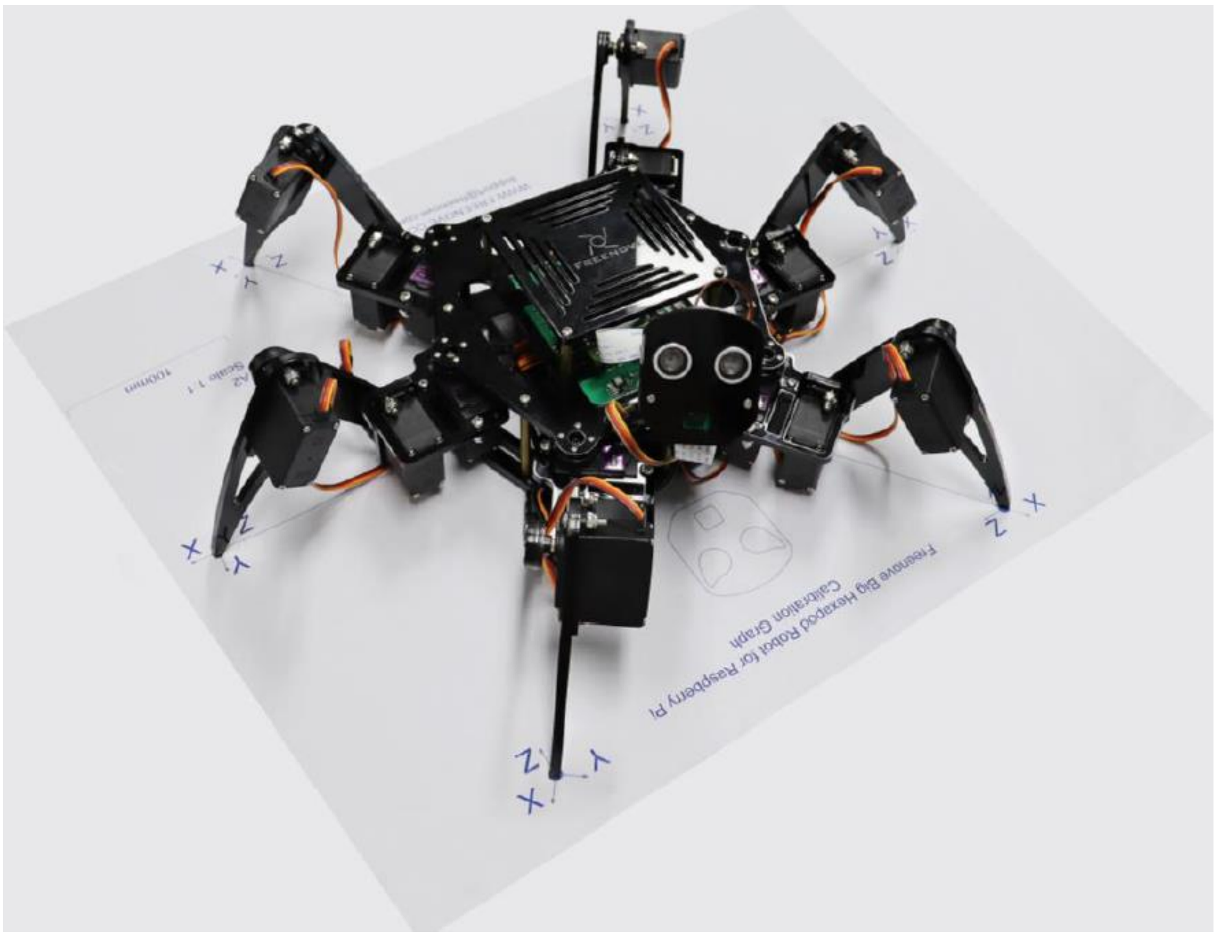


Figure 2.10 Calibration of the Hexapod robot

For the calibration, we need to open the Raspberry Pi of the Hexapod robot and run these commands:

```
cd ~/Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi/Code/Server
```

```
sudo python test.py Servo
```

As an output to this calibration, the robot slowly rotates its six legs clockwise from the original straight state, and then lifts the legs up, and finally stands with the lower leg extended back. The Figure 2.11 shows the rotation angles of the 6 legs.

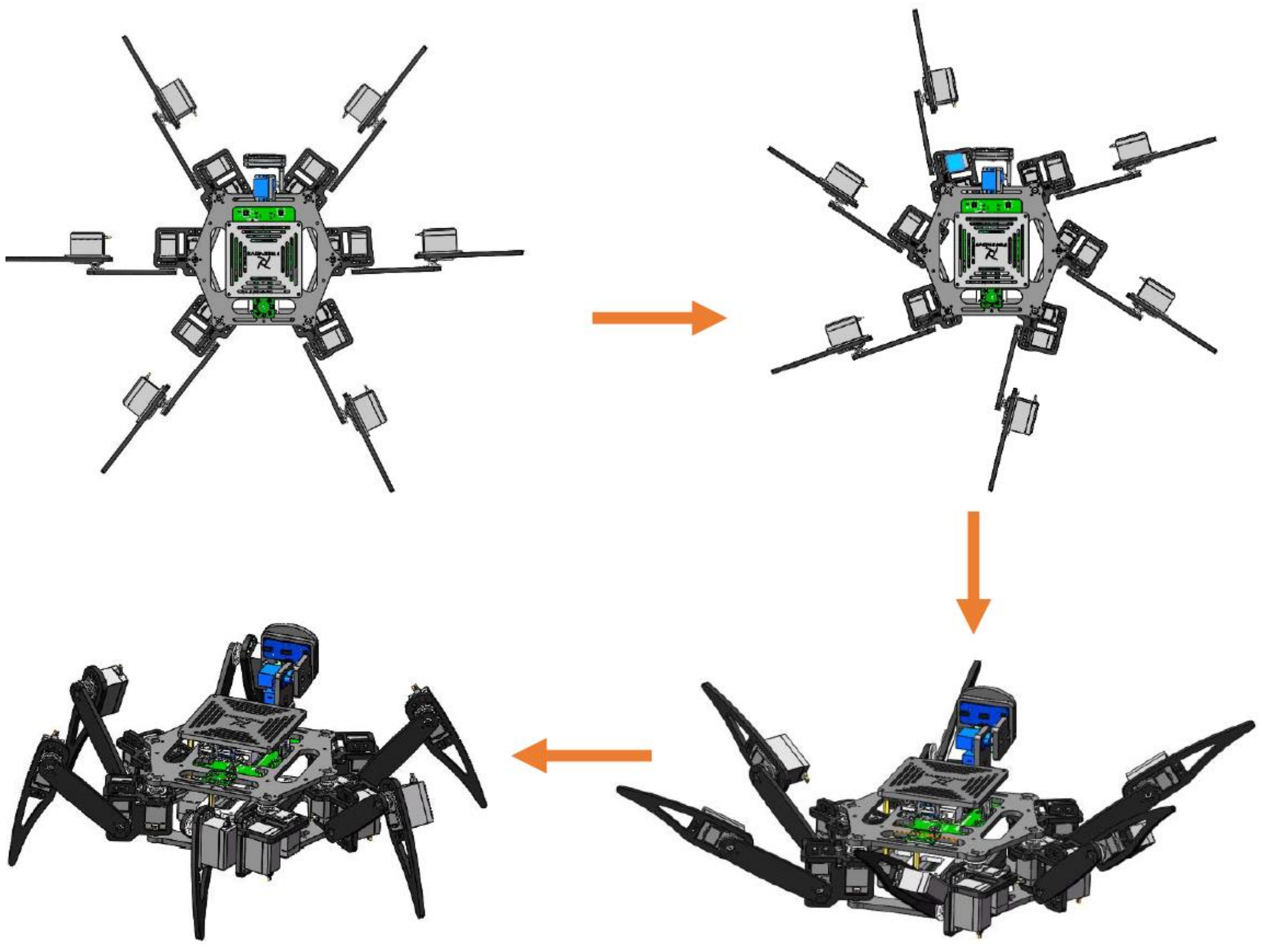


Figure 2.11 Moving legs

2.5 Kinematics models

In this section, the kinematics models of the Hexapod robot with mathematical methodology will be established.

2.5.1 Forward kinematic model

The kinematic model of the Hexapod robot should be developed to be able to move the end-effector to the robot from an initial position to an end position. Coordinate frames are defined and will be used throughout this paragraph. The kinematic model includes both of "Forward Kinematic model" which is the kinematic description of the robot and its legs, and an "Inverse Kinematic model" for the legs that will be developed afterwards. It is thus possible to calculate the articulation angles of the robot legs for a given leg. To derive the kinematic model, the following hypothesis are made:

a) The body of the robot is maintained at a constant height and in at parallel level to the ground plane during the movement.

b) The center of gravity of the robot body is considered to be at the geometric center of the trunk.

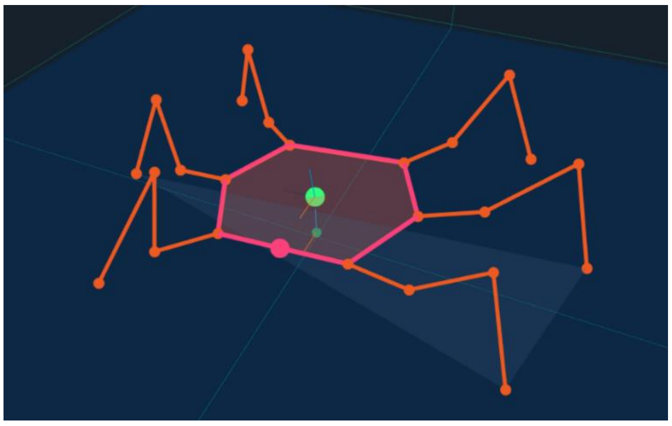


Figure 2.12 3D model of Hexapod

If we consider that the robot is in static equilibrium, the leg that will take a step will have its kinematic model similar to that of a robotic manipulator, so that the Denavit-Hartenberg method can be used.

After defining the segments of the robot leg similar to an insect mentioned in chapter 1 (leg, Coxa, Femur and Tibia), the instructions of the Denavit-Hartenberg algorithm were applied according to [23] , [24] , [25] and [26], given the position of each coordinate axis of the robotic leg joints (Figure 2.13) where there are three rotational joints (J1, J2 and J3) and L1, L2 and L3 links.

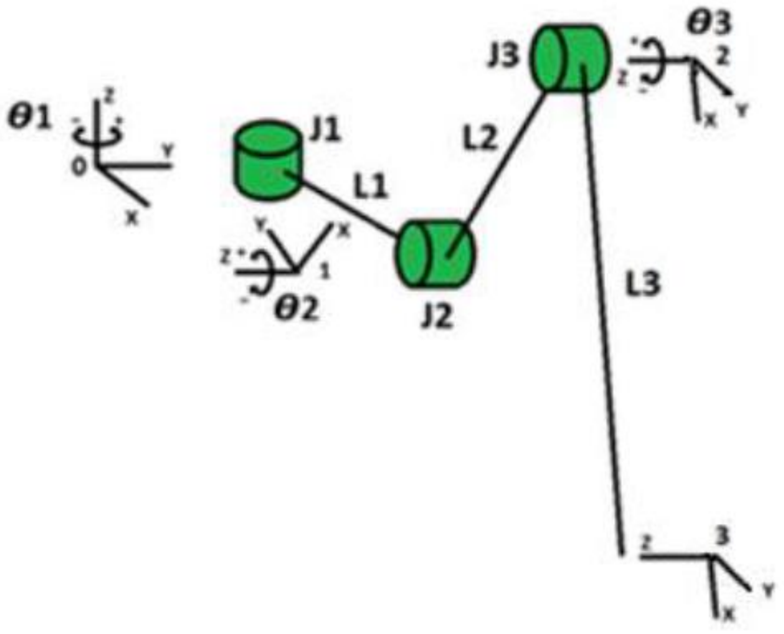


Figure 2.13 Coordinate systems for one leg

According to the method of Denavit - Hartenberg, the robotic leg is positioned in the zero setting which corresponds to the position where all the angles of the joints are equal to zero and all "x" directions are aligned (figure 2.14).

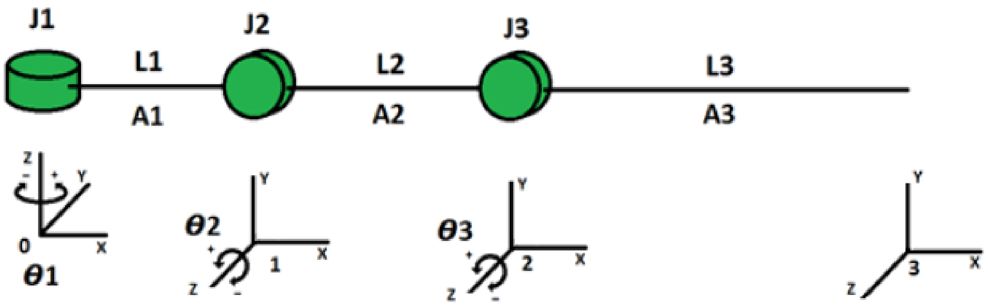


Figure 2.14 Zero position of Robotics Leg for one leg

Homogeneous transformation matrices can be defined as follows: A1, A2 and A3 using the parameters of the Denavit-Hartenberg method (Table 1). The following matrices were obtained:

- Link length ai is the distance from Zi to Zi+1 measured along Xi.
- Link twist αi is the angle from Zi to Zi+1 measured about Xi.
- Link offset di is the distance from Xi-1 to Xi measured along Zi.
- Joint angle θi is the angle from Xi-1 to Xi measured about Zi.

Link	θi (°)	αi (°)	ai (m)	di (m)
1	$\theta 1$	90	L1	0
2	$\theta 2$	0	L2	0
3	$\theta 3$	0	L3	0

Table 1 Parameters of DENAVIT-HATEMBERG

$$A1 = \begin{bmatrix} \cos(\theta 1) & 0 & \sin(\theta 1) & L1 * \cos(\theta 1) \\ \sin(\theta 1) & 0 & -\cos(\theta 1) & L1 * \sin(\theta 1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.15)$$

$$A2 = \begin{bmatrix} \cos(\theta 2) & -\sin(\theta 2) & 0 & L2 * \cos(\theta 2) \\ \sin(\theta 2) & \cos(\theta 2) & 0 & L2 * \sin(\theta 2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.16)$$

$$A3 = \begin{bmatrix} \cos(\theta 3) & -\sin(\theta 3) & 0 & L3 * \cos(\theta 3) \\ \sin(\theta 23) & \cos(\theta 3) & 0 & L3 * \sin(\theta 3) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.17)$$

By multiplying the matrices A1, A2, and A3, the transformation matrix ${}^{0}_{3}A$ down below was obtained.

$${}^{0}_{3}A = A_1 * A_2 * A_3 \tag{2.18}$$

$${}^{0}_{3}A = \begin{bmatrix} \mathrm{n}_x & \mathrm{S}_x & \mathrm{n}_x & \mathrm{p}_x \\ \mathrm{n}_y & \mathrm{S}_x & \mathrm{n}_x & \mathrm{p}_y \\ \mathrm{n}_z & \mathrm{S}_z & \mathrm{a}_z & \mathrm{p}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

The terms "n", "S", and "a" are identified as the unit direction vectors of the coordinate frame number three, and the terms px, py and pz are the terms of the position of the coordinate frame number three relative to the number zero. thus, we conclude that px, py and pz has the following forward kinematic equations :

$$px = L1 * \cos\theta1 + L2 * \cos\theta2 * \cos(\theta1) - L3 * \sin\theta3 * \sin\theta2 * \cos\theta1 + L3 * \cos(\theta3) * \cos(\theta2) * \cos(\theta1)$$

$$py = L1 * \sin\theta1 + L2 * \cos\theta2 * \sin\theta1 - L3 * \sin\theta3 * \sin\theta2 * \sin(\theta1) + L3 * \cos\theta3 * \cos\theta2 * \sin(\theta1)$$

$$\mathrm{pz} = L2 * \sin\theta2 + L3 * \sin\theta3 * \cos\theta2 + L3 * \cos\theta3 * \sin(\theta2)$$

Where L1, L2 and L3 are the distances of each link of the leg. Θ1, Θ2 and Θ3 are the angles of the joints and px, py and pz is the Cartesian position of the tip-end of the leg.

2.5.2 Inverse kinematic model

The forward kinematics results obtained in the previous paragraph in direct expressions, gives the position of the tip of the leg (px, py, pz) with given angle values. However, the inverse kinematics aims to determine the mathematic expression of the angles Θ1, Θ2 and Θ3.

From the equation (2.18) above, the next equation (2.20) can be found:

$$A_1^{-1} * {}^{0}_{3}A = A_2 * A_3 \tag{2.20}$$

Considering the matrix (2.19) and the inverse of A1 being the following :

$$A_1^{-1} = \begin{bmatrix} \cos(\theta1) & \sin(\theta1) & 0 & -L1 \\ 0 & 0 & 1 & 0 \\ \sin(\theta1) & -\cos(\theta1) & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.21}$$

With the matrices A2 and A3, we solve the equation (2.20) above and we obtain the following equations (2.22, 2.23 and 2.24):

$$px * \cos\theta1 + py * \sin\theta1 - L1 = L2 * \cos\theta2 + L3 * \cos\theta2 * \cos\theta3 - L3 * \sin\theta2 * \sin(\theta3) \tag{2.22}$$

$$pz = L2 * \sin\theta2 + L3 * \cos\theta2 * \sin\theta3 + L3 * \cos\theta3 * \sin\theta2 \tag{2.23}$$

$$px * \sin\theta1 - py * \cos\theta1 = 0 \tag{2.24}$$

Using the last equation (2.24), we found :

$$\theta 1 = tan^{-1}(\frac{py}{px}) \tag{2.25}$$

Furthermore, to find the expression of $\theta 3$ by substitution, we use the equation number (2.22) as follows :

$$px{*}\cos\theta 1 + py{*}\sin\theta 1 - L1 = L2{*}\cos\theta 2 + L3{*}\cos\theta 2 {*}\cos\theta 3 - L3{*}\sin\theta 2 {*}\sin(\theta 3) \tag{2.22}$$

Finally, we obtain :

$$\mathrm{V} = L2{*}\cos\theta 2 + L3{*}\cos\theta 2 {*}\cos\theta 3 - L3{*}\sin\theta 2 {*}\sin(\theta 3) \tag{2.26}$$

Then, we can simplify the equation (2.23) by inserting the last equation (2.26), we obtain two new equations (2.27) and (2.28).

$$\mathrm{V} = L2{*}\cos\theta 2 + L3{*}\cos\theta 2 + \theta 3 \tag{2.27}$$

$$pz = L2{*}\sin\theta 2 + L3{*}\sin\theta 2 + \theta 3 \tag{2.28}$$

The equations (13) and (14) can be simplified by summing up $\mathrm{V}^2 + \mathrm{pz}^2$ and using adequate trigonometric properties, we finally obtain the next equation (2.29).

$$\mathrm{V}^2 + pz^2 = L2^2 + L3^2 + 2 * L2 * L3 * \cos(\theta 3) \tag{2.29}$$

Finally, we get to the point of finding the expression of $\theta 3$ as follows.

$$\theta 3 = \pm\, cos^{-1}\left(\frac{\mathrm{V}^2 + \mathrm{pz}^2 - \mathrm{L2}^2 - \mathrm{L3}^2}{2{*}L3{*}L2}\right) \tag{2.30}$$

As the values of $\theta 1$ and $\theta 3$ are found, we proceed to look for $\theta 2$.

To find the equation $\theta 2$, we establish the following XZ coordinate system as shown in the next figure 2.31.

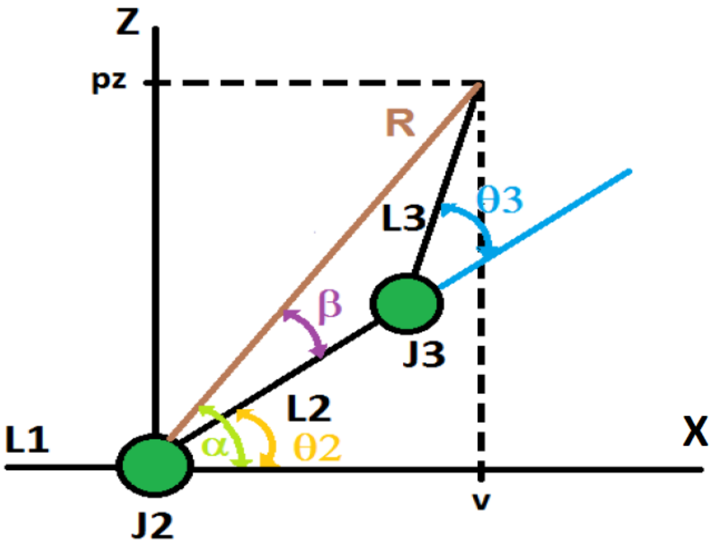


Figure 2.31 Coordinate system XZ with angles

To find Observing the above figure, we can conclude the following statements:

$$\theta 2 = \alpha - \beta \quad ; \quad tan\,(\alpha) = \frac{pz}{v} \quad ; \quad tan\,(\beta) = \frac{L3 * sin(\theta 3)}{L2 + L3 * cos(\theta 3)}$$

Using the next trigonometric property : $tan\,(A - B) = \frac{\tan(A) - \tan(B)}{1 + \tan(A) * \tan(B)}$

$$\tan(\theta 2) = tan\,(\alpha - \beta) = \frac{pz * L2 + L3 * cos\,\theta 3 - v * L3 * sin(\theta 3)}{v * L2 + L3 * cos\,\theta 3 + pz * L3 * sin(\theta 3)}$$

$$\theta 2 = tan^{-1}\left(\frac{pz * L2 + L3 * \cos\theta 3 - v * L3 * \sin(\theta 3)}{v * L2 + L3 * \cos\theta 3 + pz * L3 * \sin(\theta 3)}\right) \qquad (2.32)$$

Finally, the inverse kinematic model is the following :

$$\theta 1 = tan^{-1}\left(\frac{py}{px}\right) \qquad (2.25)$$

$$\theta 2 = tan^{-1}\left(\frac{pz * L2 + L3 * \cos\theta 3 - v * L3 * \sin(\theta 3)}{v * L2 + L3 * \cos\theta 3 + pz * L3 * \sin(\theta 3)}\right) \qquad (2.32)$$

$$\theta 3 = \pm\, cos^{-1}\left(\frac{V^2 + pz^2 - L2^2 - L3^2}{2 * L3 * L2}\right) \qquad (2.30)$$

2.6 Set up of the server

In this paragraph, we will go through the robot features with live video, ultrasonic ranging and other functions. Based on Python3 and PyQt5, it has also been built with server and client, which communicate with each other through TCP/IP protocol and can be controlled remotely with the same LAN.

The server can send image data obtained by camera and ultrasonic data to client as well as receive commands from client. The server organigram in the next figure 2.14 shows the entire process and how it works.

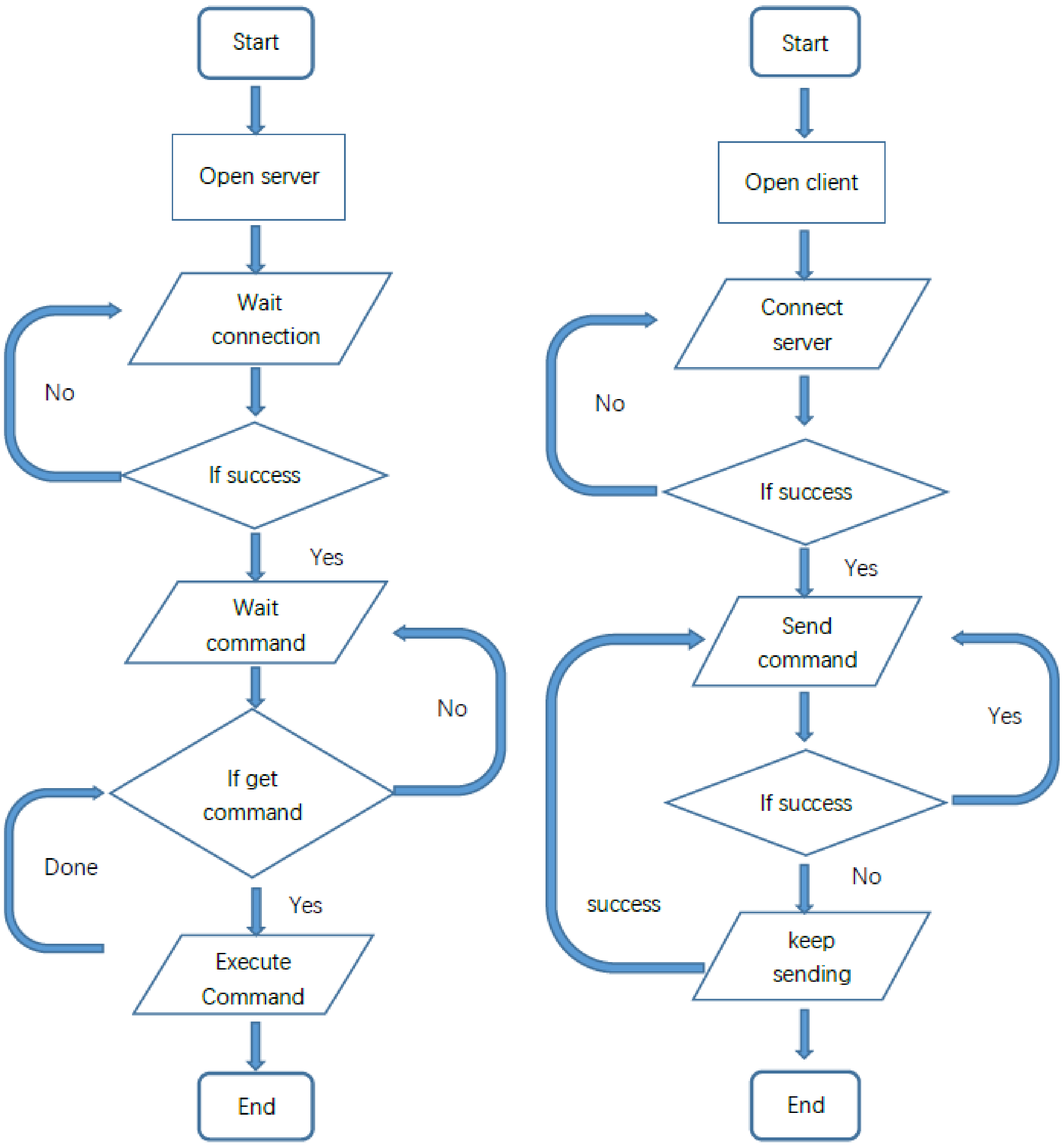


Figure 2.33 Server flowchart

To open the server, we need to boot the Raspberry Pi and run the following commands:

```
cd ~/Freenove_Big_Hexapod_Robot_Kit_for_Raspberry_Pi/Code/Server
sudo python main.py
```

After that, this window will appear as shown in the next figure 2.15 on the next page.

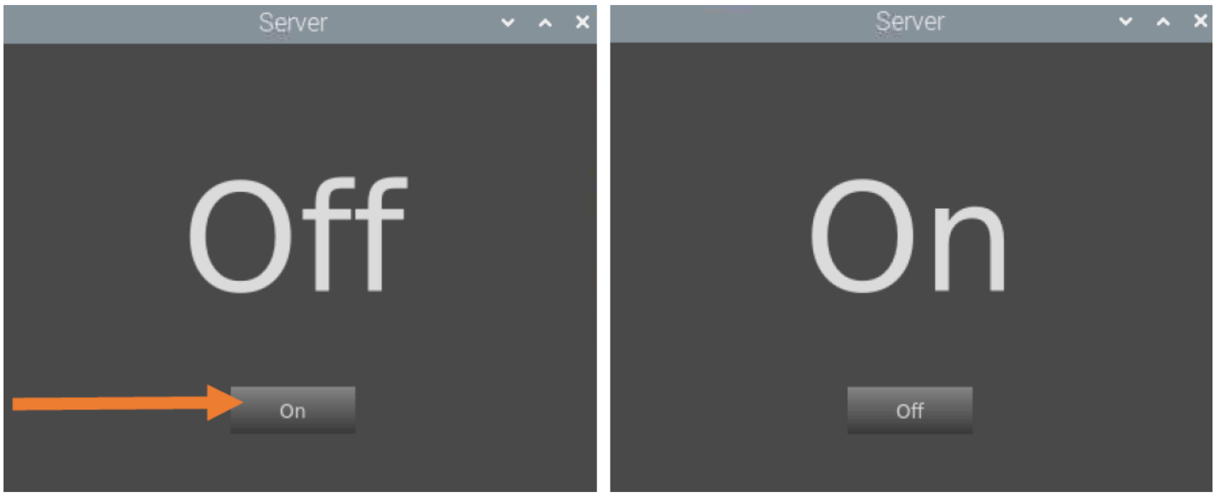


Figure 2.34 Server interface

After that, we go back to Windows and open the client interface as shown in the Figure 2.16.

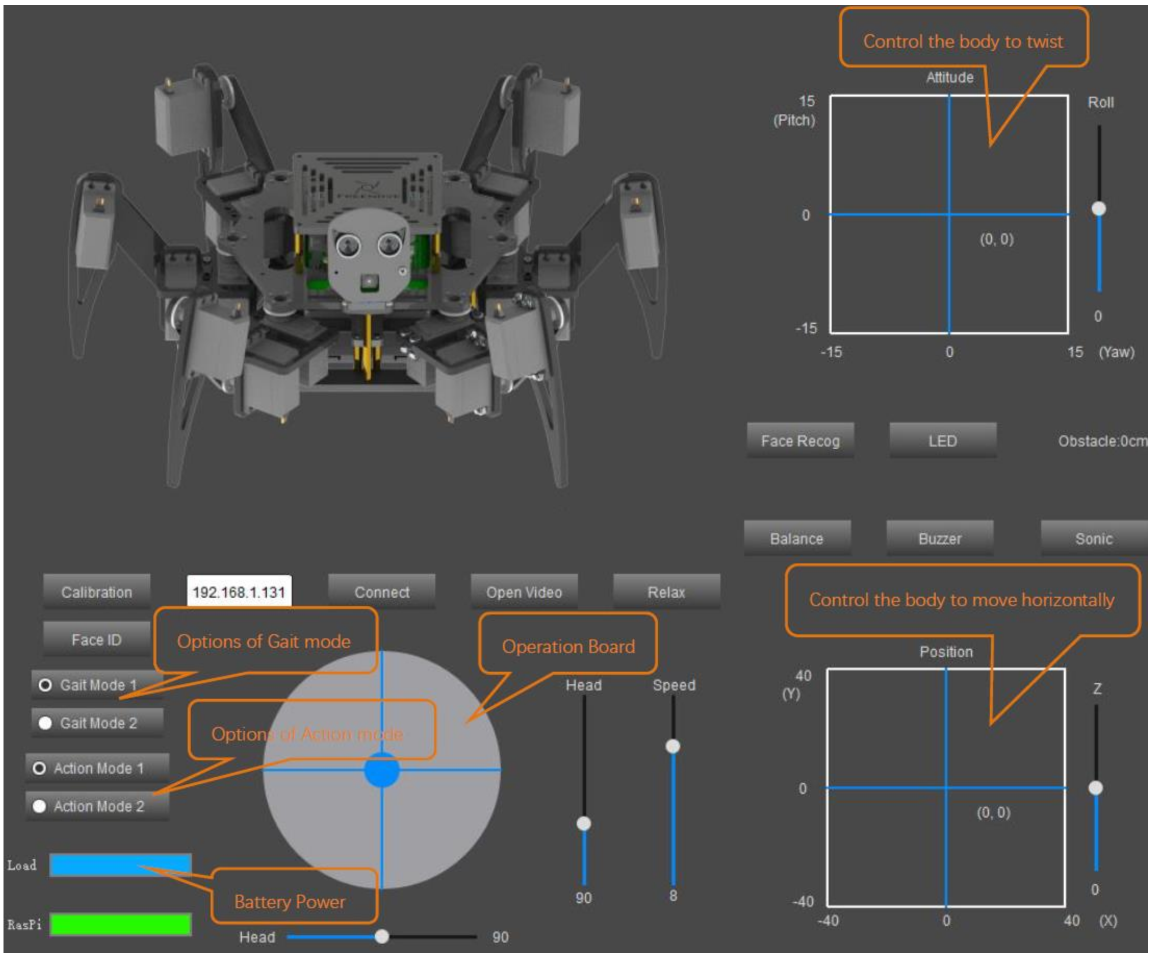


Figure 2.35 Client interface

After opening the serve of the raspberry Pi and opening the client interface, we can test the different modules of the Hexapod robot and here we take an example of the Pi Camera. As soon as we run the Picamera Python script, we can notice that my face is displayed on the client interface screen in the figure 2.17 down below.

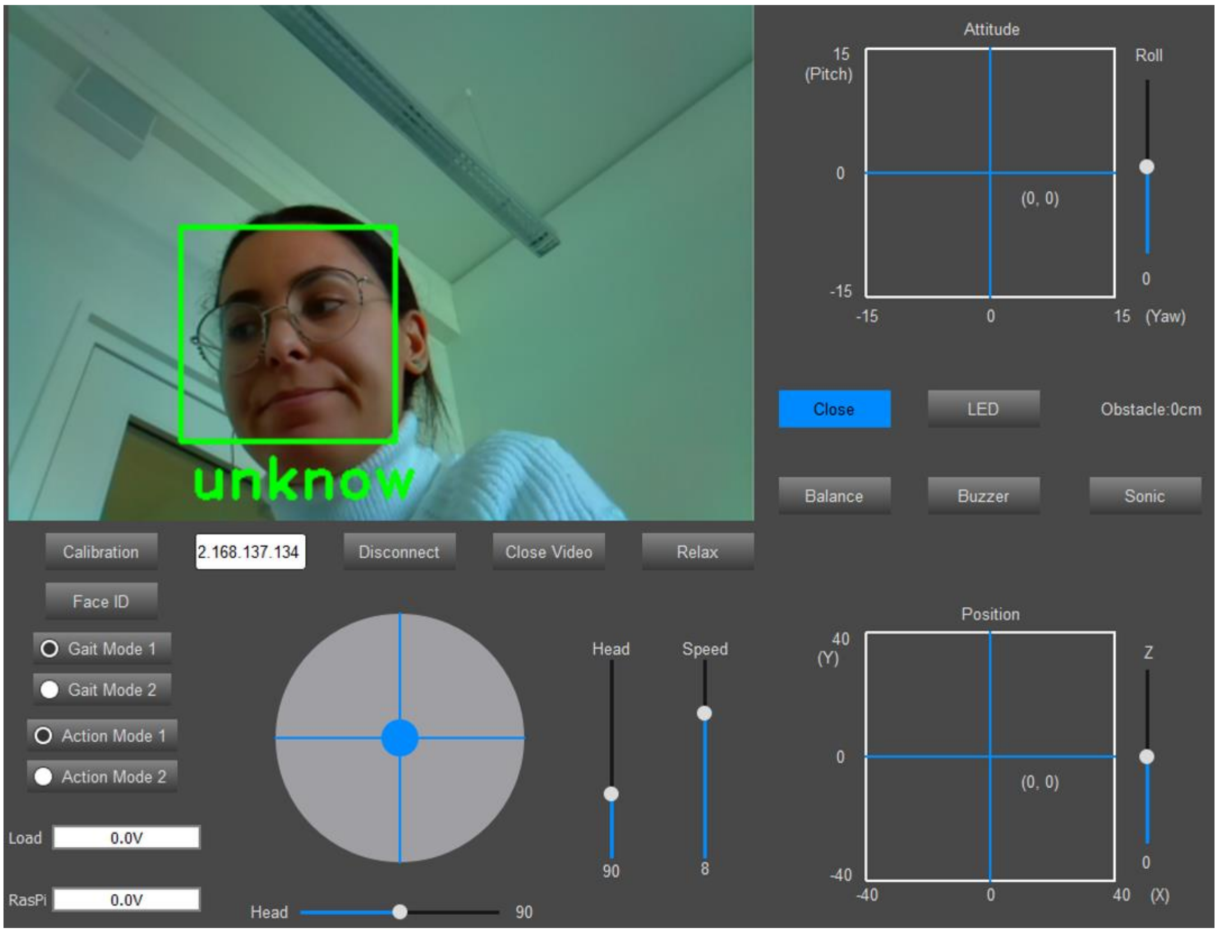


Figure 2.36 Testing Camera with client interface

2.7 Android and iOS app

In this paragraph, we will go through the Freenove Android app for Hexapod robot and how to use it.

Indeed, the Freenove company designed a mobile application so that customers and users can manipulate the robot using their phone instead of the PC. The set up of the server interface on the Raspberry Pi that we introduced in the previous paragraph remains the same. Figure 2.18 in the next page shows the display of the Freenove mobile application for the Hexapod robot.

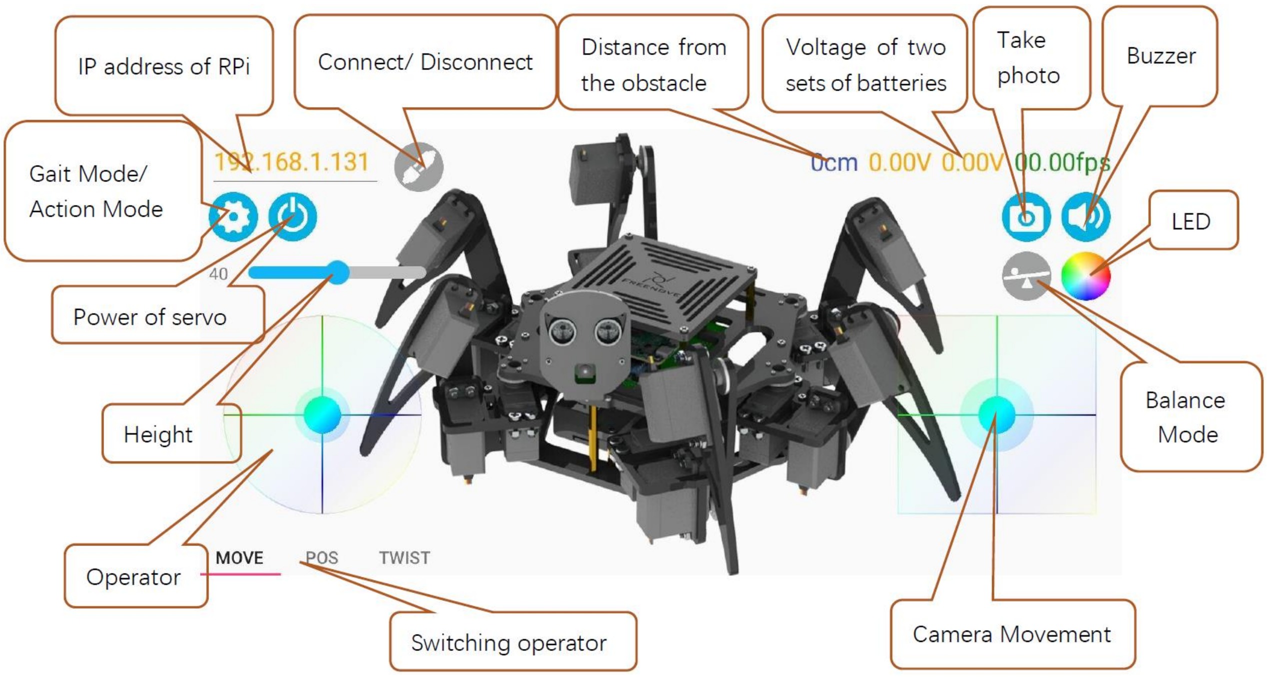


Figure 2.37 Android iOS app for Hexapod robot

After setting up the mobile application, we have to insert the IP address of the raspberry Pi and open the server interface and finally connect the app to the robot.

The result of this connection can be shown in the next figure 2.19 where we can see the live video streaming of the Picamera of the raspberry Pi using the mobile app.

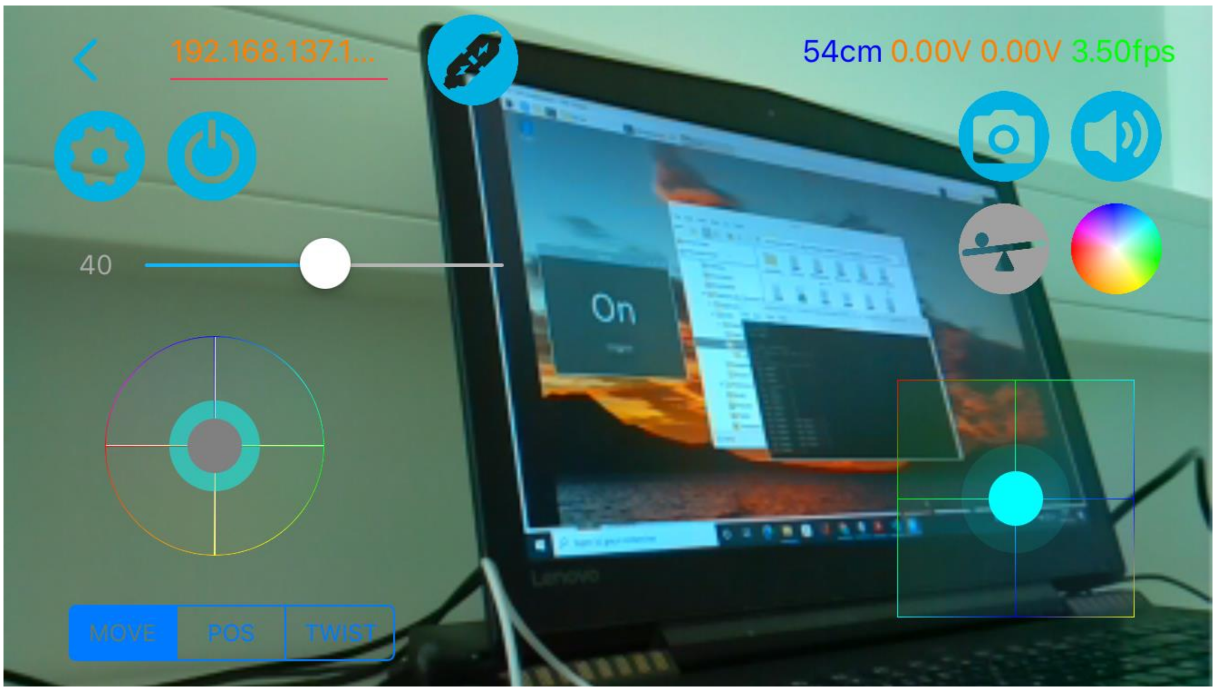


Figure 2.38 Testing Camera with iOS android app

2.8 Conclusion

In this chapter, we have assembled the robot and executed the servo installation program, which makes the six legs of the robot straight. We went through the mechanical design of the Hexapod robot, as well as the Kinematics models (forward and inverse models) that are the most important part in a robot movement description. The forward and inverse kinematics models were carefully detailed and concluded by an analytical. We successfully calibrated the 6 legs of the robot. We installed, as well, the server interface and client interface. Finally, we tested the mobile iOS android app for Hexapod robot.

Overall, one can conclude that by establishing a correct geometric model, adopting an adequate design of the robot, it can be used to perform several robotics achievements. However, given the complexity of such system, one should perform a precise and accurate Machine Learning model in order to get the best results.

This will be addressed in the next chapter of this document.

3

Software development

Introduction

In this third chapter and as a first step, we will discuss the implementation of a software system for environment identification and facial recognition.

In a neural network, two modes of operation can be distinguished [21]. In the first one, the parameters of the network are adjusted by the back-propagation algorithm through the presentation of examples for which the desired response is known: this mode of operation is called "learning phase". In the second mode, it is a concern of operating the network by presenting it with new unlabeled data. This mode of use is often called the "test phase", or the "generalization phase".

In this chapter, we will discuss the different cases at the end of the training phase and the possible solutions to remedy the problems encountered.

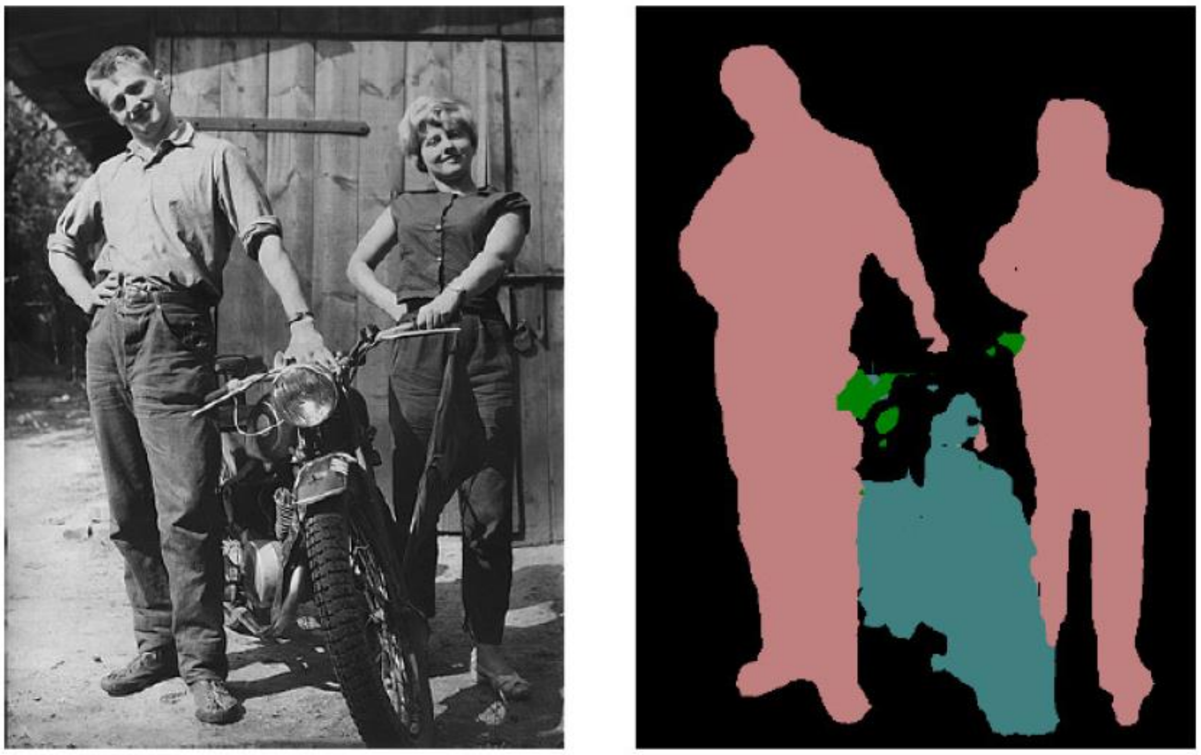
Keywords: Autonomous navigation; semantic segmentation; convolutional networks; obstacle detection; Enet.

3. Recognition of the environment

3.1 About Semantic Segmentation

Semantic segmentation is a Deep Learning algorithm that associates a label or category to each pixel in an image [27]. It allows us to recognize a set of pixels that form distinct categories. For example, an autonomous vehicle needs to identify vehicles, pedestrians, traffic signs, sidewalks and other elements of the road environment.

Furthermore, separating images into two classes is a simple example of semantic segmentation. For example, in Figure 3.1, an image showing people with motorbike is associated with a version showing the pixels of the image segmented into three distinct classes: the person (pink) ,the motorbike (blue) and the background (black).



(a) (b)

Figure 3.1 Image before (a) and after (b) Semantic Segmentation

3.2 CityScape dataset

Cityscapes [23] is a recently published large-scaled dataset for the semantic understanding of urban scenes. It includes 5,000 high quality images with fine annotations and 20 000 images wih coarse annotations from different street scenes from 50 different cities around the world. The scene contains pedestrians, the street, traffic signs, sky, vehicles, animals, plants and in all seasons. This dataset is divided into 2,975 images for training and 1,525 images for validation and testing.

Group	Classes
Flat	Road, sidewalk, parking, rail track
Human	Person, rider
Vehicule	Car, truck, bus, on rails, motorcycle
Construction	Building, wall, fence, guard rail, bridge
Objet	Traffic sign, traffic light
Nature	Vegetation, terrain
Sky	Sky
Void	Ground, static, dynamic

Table 2 Group and classes of CityScape dataset

There are 19 different categories of Cityscape testing methods, but the most known ones are CRF-RNN, FCN, SiCNN, DPN, Dilation10, DeepLab, Piecewise and PSPNet.

The next figure 3.2 shows examples of training with the PSPNet method on Cityscape dataset (images with fine annotations). These results were developed in a recent released scientific article [34] on comparing training methods on Cityscape dataset.

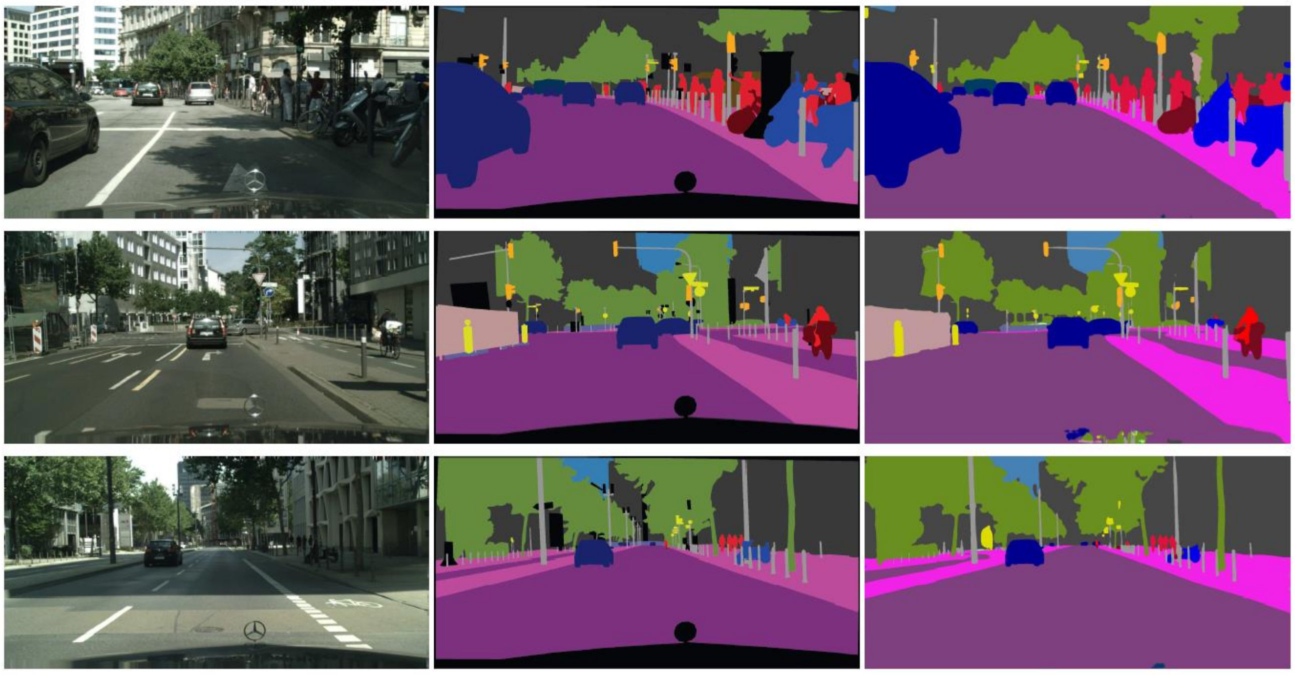


Figure 3.2 Examples of PSPNet results on Cityscapes dataset

3.3 Implementation details

3.3.1 Dataset

For the semantic segmentation model, we worked on training the model with ENET, a Deep Neural Network Architecture for Real-Time Semantic Segmentation using the backbone of Resnet50.

For the labels, we will to use PASCAL Visual Object Classs as a segmentation benchmark. It provides normalized image datasets for object class recognition. The standardized colors are as follows:

B-ground	Aero plane	Bicycle	Bird	Boat	Bottle	Bus
Car	Cat	Chair	Cow	Dining-Table	Dog	Horse
Motorbike	Person	Potted-Plant	Sheep	Sofa	Train	TV/Monitor

Table 3 Standardized colors for PASCAL VOC

From the cityscape database, we used:

- *gtFine_trainvaltest* : a file with fine annotations for train and valisation sets (3475 annotated images) and for the test set (1525 images).
- *gtCoarse* : a file with coarse annotations for train and valisation set (3475 annotated images).
- *leftImg8bit_trainvaltest* : a file for train, validation, and test sets (5000 images)

3.3.2 Software requirements

After identifying the adequate dataset, we make sure of the software requirements as follows :

a) **Pytorch** [35], built on the Torch library, is an open-source framework for machine learning. For this case, we will need Pytorch version 1.1 The next figure 3.3 shows the official logo of Pytorch developed by Meta [36].
b) **Torch vision**, is a library that is part of the Pytorch framework. It is a package that contains common datasets as well as model architectures (CNN, RNN…).
c) **OpenCV for Python** [37], is an open-source software library, initially developed by Intel company. It is dedicated to real-time image processing.
d) **Tensorflow Board** [38], is a software tool that provides a visualization solution needed for machine learning testing such as tracking and visualization of metrics such as loss and accuracy, display of histograms of weights alongside other functionalities.

After identifying the adequate software requirements, we need to add the loss Dice-Loss, which measures the overlap between two samples (images).

Dice loss or the Dice coefficient [39], is largely used in medical image segmentation studies to solve the issue of data mismatch. However, in this project, we will used in Segmentation of a street scene.

3.4 Code structure

The structure of the Semantic segmentation code is based on a pytorch project template as follows:

```
pytorch-template/
│
├── train.py - main script to start training
├── inference.py - inference using a trained model
├── trainer.py - the main trained
├── config.json - holds configuration for training
│
├── base/ - abstract base classes
│   ├── base_data_loader.py
│   ├── base_model.py
│   ├── base_dataset.py - All the data augmentations are implemented here
│   └── base_trainer.py
│
├── dataloader/ - loading the data for different segmentation datasets
│
├── models/ - contains semantic segmentation models
│
├── saved/
│   ├── runs/ - trained models are saved here
│   └── log/ - default logdir for tensorboard and logging output
│
```

```
└── utils/ - small utility functions
    ├── losses.py - losses used in training the model
    ├── metrics.py - evaluation metrics used
    └── lr_scheduler - learning rate schedulers
```

Furthermore, the configuration elements specific to this project have been mentioned in the Config.json file. We mentioned that the method used is ENET and the training will take 80 epochs. The complete code of the config.json can be found in Appendix 1 Configuration file of semantic segmentation trained model. The final Semantic Segmentation algorithm can be found in Appendix 2 Semantic segmentation algorithm.

Due to the fact that the model doesn't include road sign, light or any infrastructure that might help us include city center navigation we opt to include a lane assistant feature. Lane assistance is a feature implemented in many modern vehicles that help drivers stay in the middle of the lane in case they start drifting from it due to lack of attention.

To implement such feature, we need to detect the curvature of the road up ahead of the driver and compare the expected steering of wheel that the driver needs to make to the actual action. To do so, given a segmented input of the road, we proceed to filter out everything but the actual asphalt of the road and to detect the curvature a good method is to calculate the center of the asphalt shape left behind using the OpenCv simple blob detection function.

In case of a problem detection due to drifting away from the expected path two actions could be taking :

-Active correction : is the action of lightly turning the wheel in the hand of the driver in the target direction
which will turn his attention back to road.

-passive notification : could be a vocal signal or a visual signal like light that notify the driver that he is drifting about from the assigned lane.

3.5 Face recognition

Face recognition is an artificial intelligence section that is used widely for recognizing an individual's identity using the features of their faces. There are several facial recognition algorithms, but the accuracy varies. Face recognition is widely used in smart phones. Adapted to the automotive industry and as part of the autonomous navigation project, the face recognition will allow cars to recognize their owners from a distance. it will replace the traditional key with a facial recognition system, at least for the locking or unlocking function.

For the present case, the facial recognition process starts by capturing images with the PiCamera relative to the robot's Raspbeery Pi. Then, the images are processed with the OpenCV image processing library.

3.5.1 Face detection

The first task is to detect the faces in the image or video sequence. We will recognize the coordinates and the exact location of the face in the given image.

We extract, therefore, these faces for further processing of facial recognition. For a given image, these command lines in the figure below initialize OpenCV and allow to import of functions and classes needed for image processing.

Now we need to convert the image to grayscale in order to use the face detection function. The conversion to gray level is a transformation in the RGB space (Red/Green/Blue).

```python
import cv2
import sys
from matplotlib import pyplot as plt

imagePath = r'image0.jpg'
dirCascadeFiles = r'../opencv/haarcascades_cuda/'
cascadefile = dirCascadeFiles + "haarcascade_frontalface_default.xml"
classCascade = cv2.CascadeClassifier(cascadefile)
image = cv2.imread(imagePath)
plt.imshow(image)
```

Furthermore, we proceed by converting the image to grayscale in order to use the face detection function. The conversion to gray level is a transformation in the RGB space (Red/Green/Blue).

```python
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
plt.imshow(gray)
```

Following that, we can call for the face detection function.

```python
faces = classCascade.detectMultiScale(
    gray,
    scaleFactor=1.1,
    minNeighbors=5,
    minSize=(30, 30),
    flags = cv2.CASCADE_SCALE_IMAGE
)
print("Il y a {0} visage(s).".format(len(faces)))
```

Finally, we get an image where the faces are surrounded by a rectangular shape. The complete algorithm of Face detection can be found in Appendix 3 Face detection algorithm.

3.5.2 Feature extraction

After extracting the faces present in an image, we extract the specific features. To do this, the conventional neural network capture an image of the individual in front of the camera and takes its face as input to the machine learning model and generates a vector that represents that it consists of the important features of that given face that makes it distinguished. It is called "features vector". During the learning process, the neural network learns to generate similar vectors for faces that look similar.

Following that, we train the model with the extracted features.

Finally, we extracted the features. The complete algorithm of Feature extraction and training can be found in Appendix 4 Face recognition algorithm.

3.5.3 Face comparing

After training the model, we now have face overlays for each face in the data stored in a file in the Raspberry Pi's memory.

The next step is to give the system a new image that is not in its initial database and see if it can compare it with the learned faces. This step consists in calculating the face features for the image using the same grid that we used in the previous code. Then, we must compare this overlay with the rest of the features present in the system after the training.

Then, Finally, the complete algorithm of Face comparing can be found in Appendix 5 Face comparing algorithm, and to sum up, the face recognition path can be represented as follows in Figure 3.4

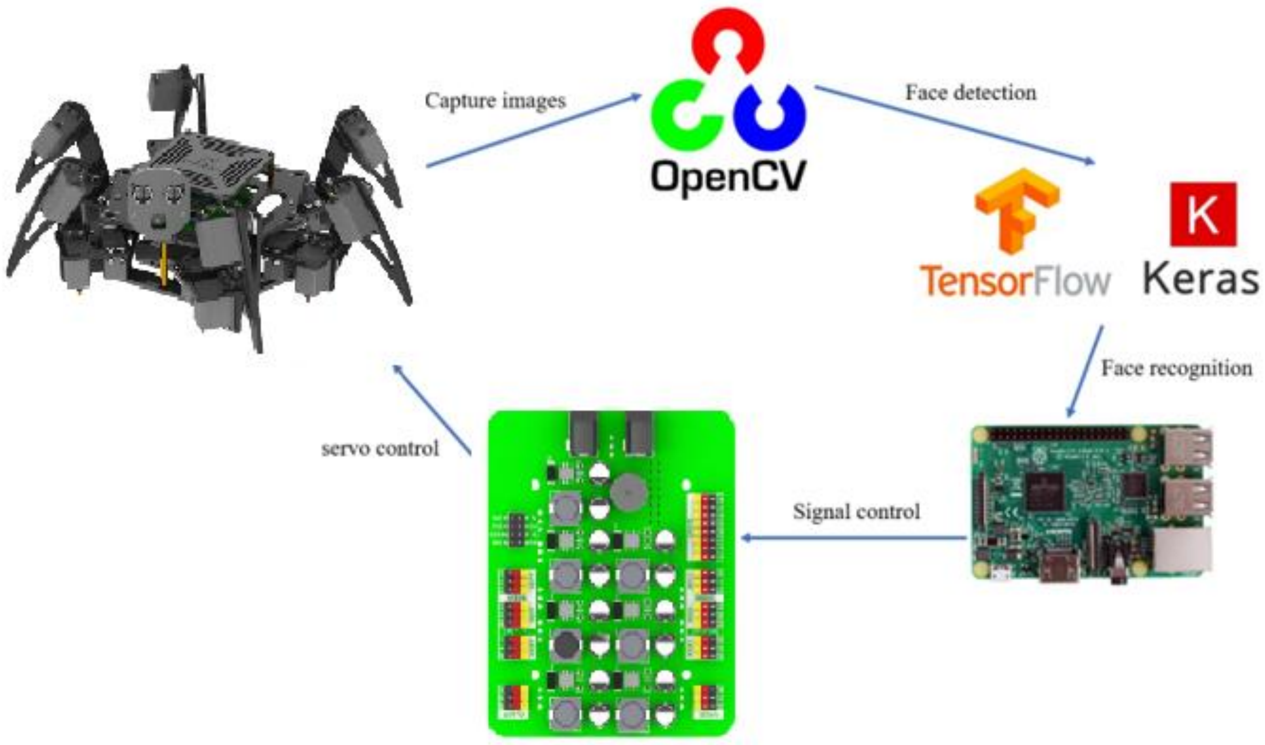


Figure 3.3 Face recognition for Hexapod

3.6 Conclusion

Throughout this chapter, we had engineered all the steps required to make the robot recognize his environments and his owners, from a deep learning algorithm for semantic segmentation to a machine learning algorithm for facial recognition.

First, we highlighted the importance of the semantic segmentation for this specific robotics system as a powerful tool for environment recognition. Indeed, a Convolutional neural network is performed to reduce the feature resolution and filter every category in the image.

Hence, we have now given the robot both an "eye" (Face recognition) and a "brain" (semantic segmentation). However, these "eye "and "brain" should be tested and verified for their accuracy and efficiency.

In the upcoming chapter, a full evaluation of the deep Semantic Segmentation algorithm will be descripted.

4

Testing and achievements

Introduction

Now that the algorithms are set up, this final chapter will be the topic of the testing of the proposed solution. First, we will detail the results we got from the semantic segmentation training. Next, we will detail the parameters we got as well as graphs with interpretations. Then, the navigation part will be descripted in order to deduce the centroid of the Semantic segmentation image. Finally, we will interpret the results gotten with the deep learning facial recognition algorithm.

Keywords: Navigation, lane assistance, Intersection over union, centroid.

4. Semantic segmentation results

4.1 Evaluate the trained

4.1.1 Loss Function

Semantic segmentation for Hexapod robot has shown the results below (Figure 4.1).

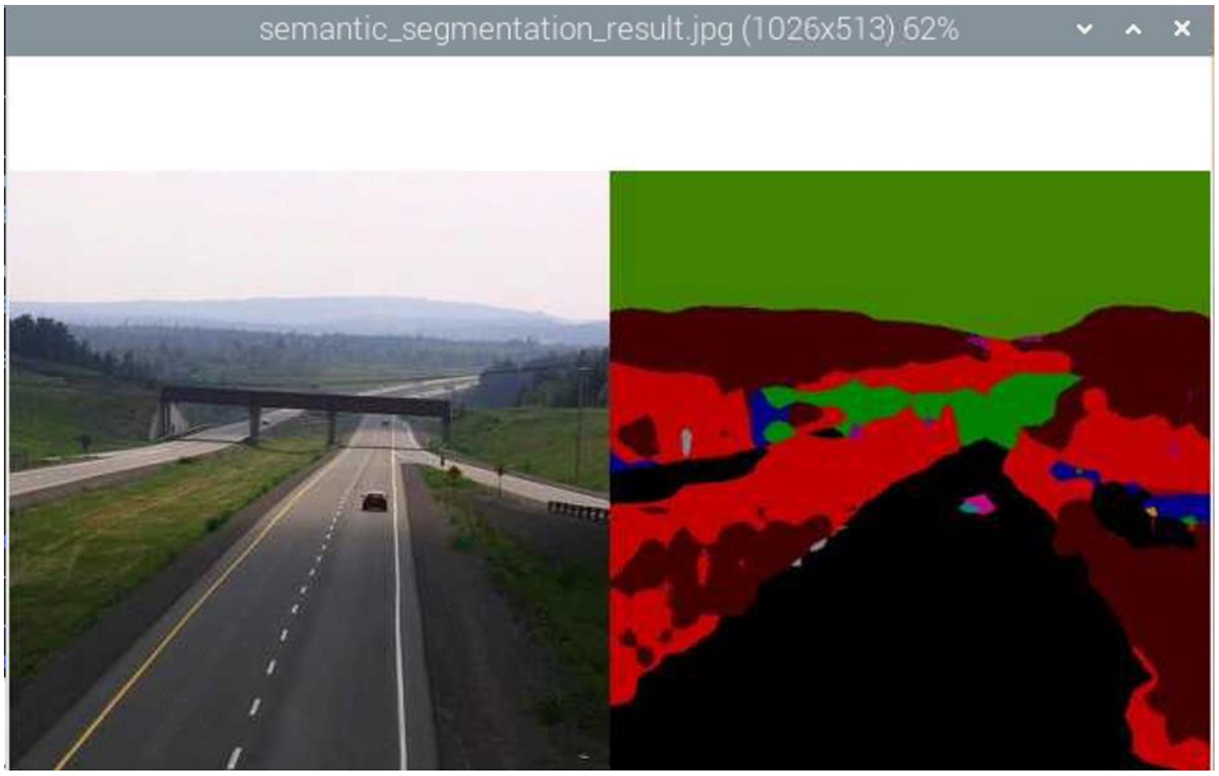
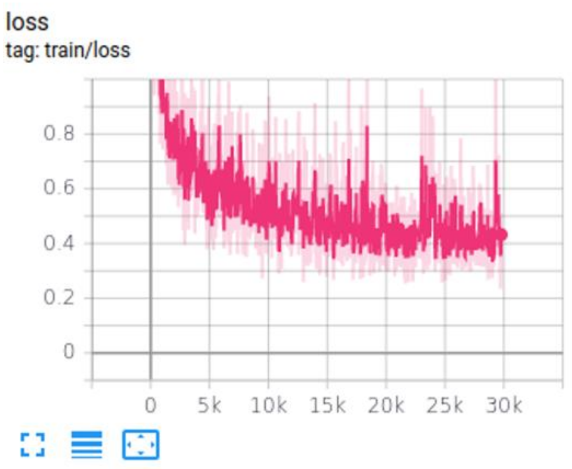


Figure 4.1 Semantic Segmentation

Indeed, after training for 4 days, the model finished the 80 epochs. Now it is time to see how well the training went. First thing is to plot (Figure 4.2) the loss function of both training and validation sets. A loss function is a function that evaluates the difference between the predictions made by the neural network and the actual values of the observations used during training. The more the result of this function is minimized, the better the neural network is performing. Its minimization, i.e. reducing to the minimum the difference between the predicted value and the real value for a given observation, is done by adjusting the different weights of the neural network. It is remarkable to see that both training and validation losses declined rapidly together, and then stayed very low after epoch 10. There didn't seem to be any overfitting [40] issue, as validation loss stayed low with training loss.



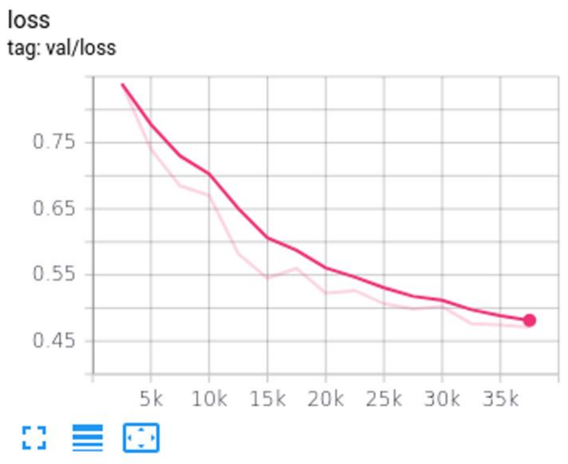


Figure 4.2 (a) Loss function for Training (b) Loss function for validation

4.1.2 Intersection over union

The IoU (Intersection over union) [41] is used as a prediction element for a given class or entity and defined as the intersection of the predicted image and actual image divided by their union (Figure 4.3).

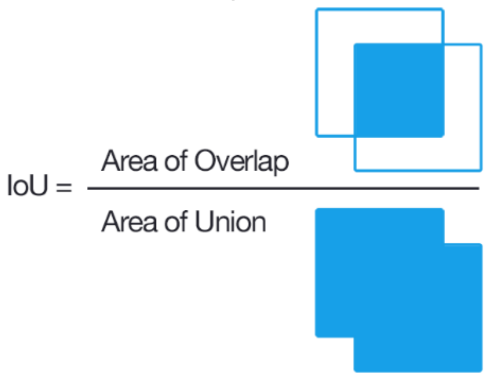


Figure 4.3 Intersection over union

The **mean_IoU** is defined as follows :

Mean_IoU = true_positives / (true_positives + false_positives + false_negatives)

The **mean_IoU** in the Figure 4.4 is low for a value of 0.56599. An IoU close to 0 means that there is no overlap between the images.

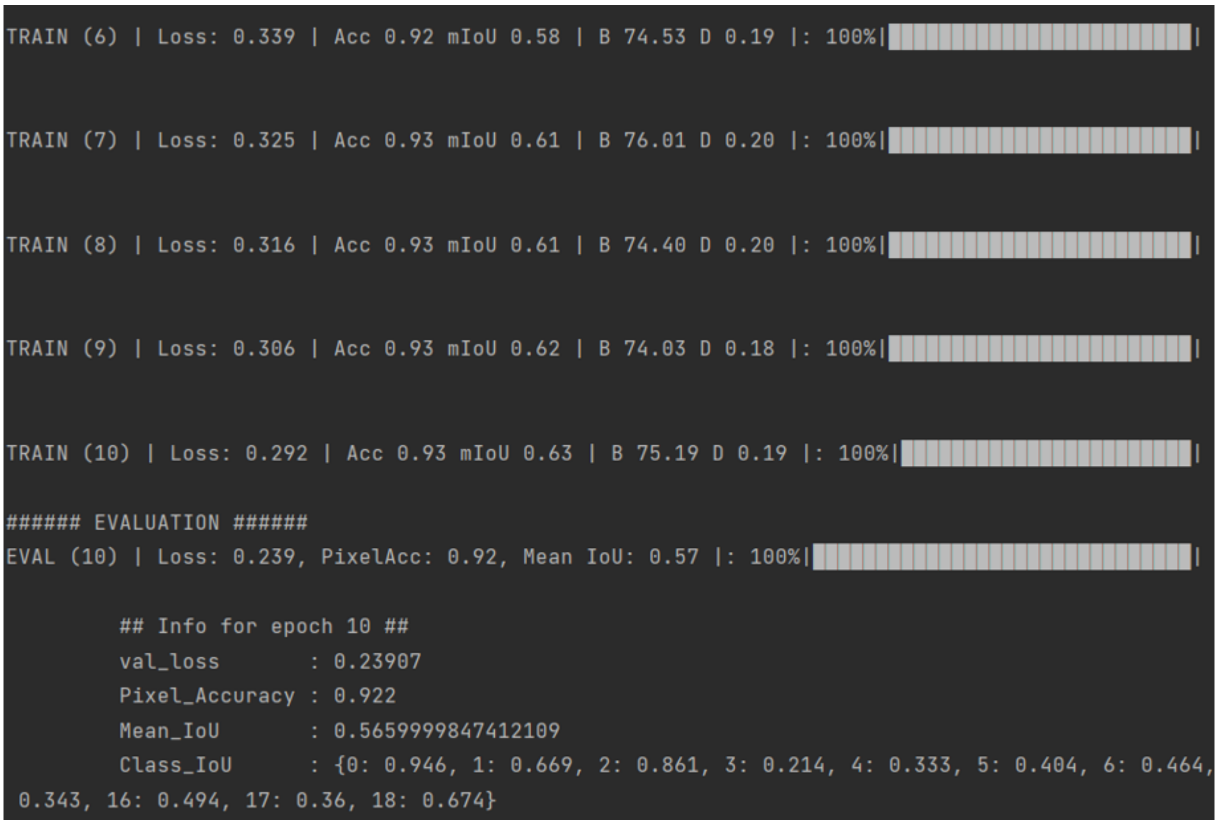
Also, another parameter is the **class IoU** which contains several IDs that refers to every class in a given image. Every pixel in the image is representing a certain class (e.g., person, car, street, sidewalk, speed limit, etc…).

class IoU for ID number zero has a value of 0.946 which shows that the system classifies the roads in a given image more than the other classes (person, nature, vehicles). This is an expected results since the cityscapes dataset that we used for the training contains different images that are mostly roads from different cities.

4.1.3 Pixel accuracy

To evaluate the semantic segmentation model precisely. We will calculate the percentage of pixels that are correctly classified. Pixel accuracy [42] is calculated for each given class individually and for a whole category of classes as well.
Hence, the pixel accuracy is defined as follows :

Pixel_accuracy=TP +TN / TP+TN+FP+FN



```
TRAIN (6) | Loss: 0.339 | Acc 0.92 mIoU 0.58 | B 74.53 D 0.19 |: 100%|████████████████████████|


TRAIN (7) | Loss: 0.325 | Acc 0.93 mIoU 0.61 | B 76.01 D 0.20 |: 100%|████████████████████████|


TRAIN (8) | Loss: 0.316 | Acc 0.93 mIoU 0.61 | B 74.40 D 0.20 |: 100%|████████████████████████|


TRAIN (9) | Loss: 0.306 | Acc 0.93 mIoU 0.62 | B 74.03 D 0.18 |: 100%|████████████████████████|


TRAIN (10) | Loss: 0.292 | Acc 0.93 mIoU 0.63 | B 75.19 D 0.19 |: 100%|███████████████████████|

###### EVALUATION ######
EVAL (10) | Loss: 0.239, PixelAcc: 0.92, Mean IoU: 0.57 |: 100%|██████████████████████████████|

         ## Info for epoch 10 ##
         val_loss       : 0.23907
         Pixel_Accuracy : 0.922
         Mean_IoU       : 0.5659999847412109
         Class_IoU      : {0: 0.946, 1: 0.669, 2: 0.861, 3: 0.214, 4: 0.333, 5: 0.404, 6: 0.464,
 0.343, 16: 0.494, 17: 0.36, 18: 0.674}
```

Figure 4.4 Semantic Segmentation results

4.2 Lane assistance

In this paragraph, the concentration will be on the navigation part of the system. For a given photo (Figure 4.5), we extract the element with ID number 0 that corresponds to the "asphalt" and eliminate all other elements from the input. Then, we proceed calculate the centroid of the shape we obtained.
The centroid of the blob will hint us as to which direction the road is headed.
If the centroid is in around the middle (between pixel 650 and 750) no action is to be taken. In the other scenario we need calculate the off-center (the distance between the centroid and the center of the road).
In this example that we find that the centroid has the value of 800 on the X axis.
From that we deduce that the direction the steering wheel needs to turn to off-set this error is a slight turn to the left. As for how hard to turn the steering wheel is depends on the steering ratio (which is the link between how much a turn in the

steering wheel affect the angle of the actual wheels) which is a car dependent value.

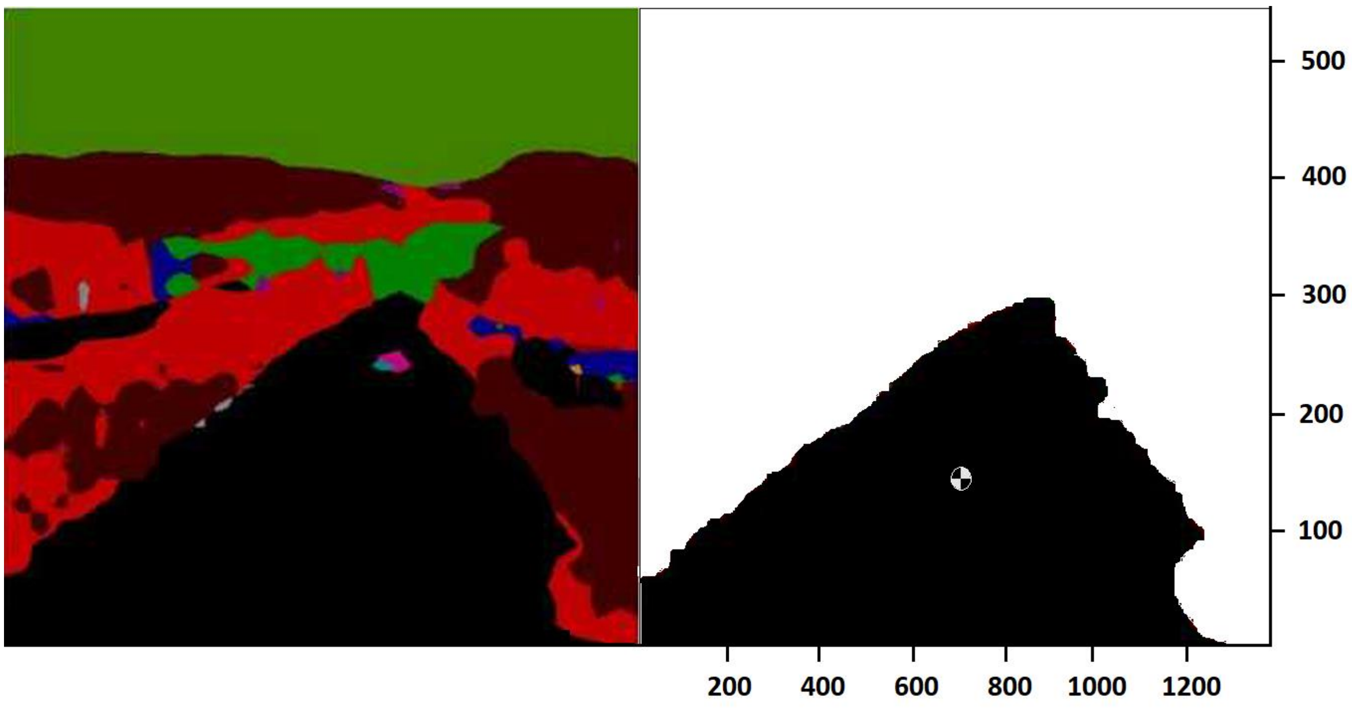


Figure 4.5 Analysis of segmentation results

4.3 Face recognition results

After testing the Face recognition algorithm, we got the results below in Figure 4.6. Indeed, from the facial features in this image, we can conclude:

- To authenticate a person: in this case Nourchene and Youssef.
- To identify a person: that is to say, to find a person within a group of individuals, in a place, an image or a database. In the present case there are 2 people identified.

We used the VGG model. This results in a code that can be divided into two phases:

- Construction of the convolutional network model which is identical to VGG
- Loading the weights of all the variables of the network

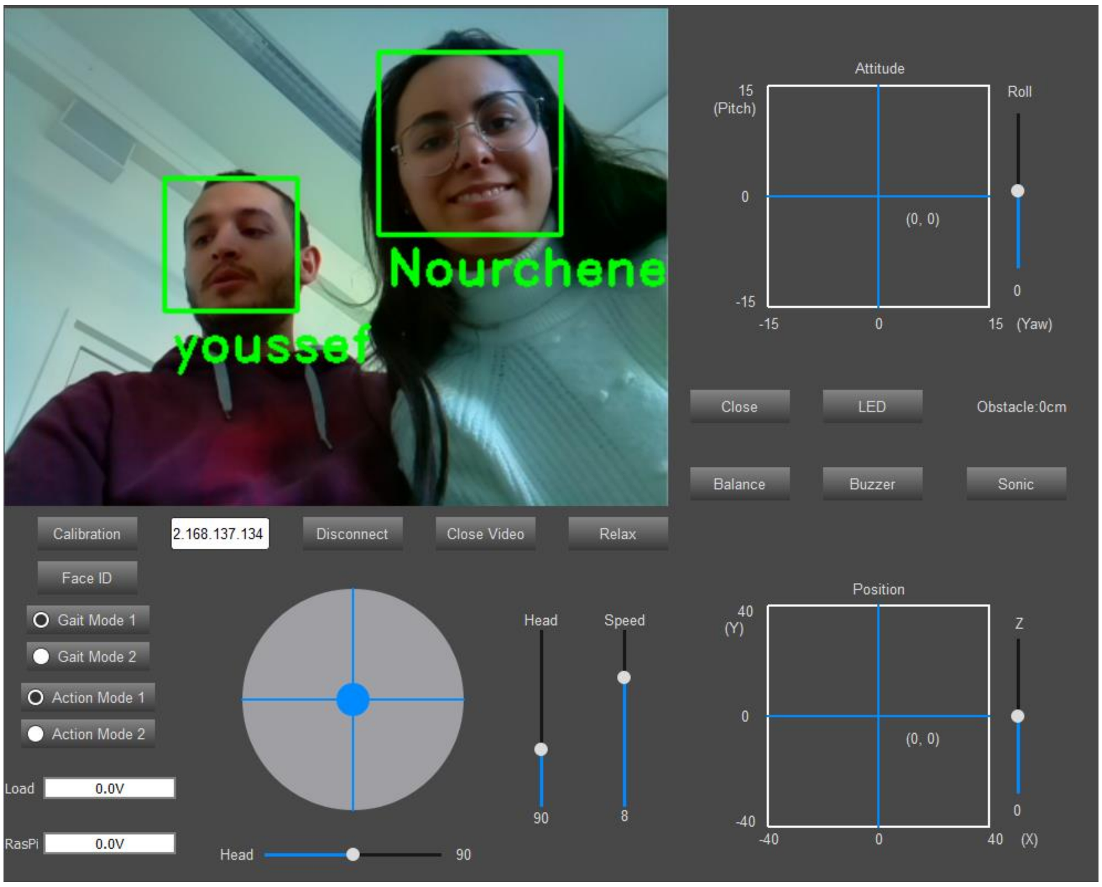


Figure 4.6 Face recognition testing results

This function allows the car to recognize its owner(s), and this by installing an infrared facial recognition sensor on the driver's door of the car. This sensor can record two faces. And depending on the recognized face, the car will adapt the settings previously defined: the position of the seat, the head-up display, the mirrors, or the infotainment (the automatic launch of a radio or a playlist depending on who is driving). Who says infrared camera also says that this sensor works even at night or when the luminosity is low.

4.4 Conclusion

In this chapter, we went through the results of the Deep Learning implemented algorithms in the previous chapter. Finally, one can conclude that the Semantic Segmentation is an important branch of image processing and highly used in robotics especially when it comes to recognize its own environment. For this case, the objective achieved was to understand accurately the scene and the content of the image that was, subsequently, used in the analysis and prediction of steering wheel orientation for the lane assistance section.

As for the Machine Learning facial recognition part, it was proved that the robot can recognize and identify a maximum of two owners in a real-time streaming.

Conclusion and perspectives

This graduation project, carried out at the Center for Advanced Electronics in Dresden, aims to develop a new technology for the modern automotive industry. During this project, we performed the assembly of a Hexapod robot and the development of its geometrical model (Forward and Inverse kinematics models).

Also, we have achieved the development of its movement thanks to a client server and an Android application on smart phone that can control the movement of the robot to move on the right, left, forward and backwards using the servo motors.

Finally, we conducted the project by developing a Machine Learning module for environment recognition by semantic segmentation, an artificial intelligence model used with open-source City Scape dataset, and to finish with the deep learning algorithm for facial recognition using OpenCV and TensorFlow which allows the robot to recognize its owner(s).

As a perspective of this project, we aim to add new features. Indeed, we propose the development of a solution with fingerprints to eliminate the car key: no need to have a key with you systematically. Your fingerprints are the key.

Facial recognition could simplify the identification of the driver to adapt the seat and mirrors to his or her preferences. The same could be true for the settings of the car radio, navigation, climate control and even the driving mode. In addition, facial recognition could also notice signs of discomfort or distraction, even stress or anger. And thus alert the driver and adapt the semi-autonomous driving systems.

B

Bibliography

[1] Technische Universität Dresden website. https://tu-dresden.de/ visited on 07.02.2022.

[2] Center for Advancing Electronics Dresden website. https://cfaed.tu-dresden.de/news visited on 07.02.2022.

[3] Deutscher Akademischer Austauschdienst website. https://www.daad.de/de/ visited on 07.02.2022.

[4] Chenyi Chen, Ari Seff, Alain Kornhauser, Jianxiong Xiao : DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving, 2015.

[5] Tesla AutoPilot https://en.wikipedia.org/wiki/Tesla_Autopilot visited on 16.02.2022.

[6] S. Ullman. Against direct perception. Behavioral and Brain Sciences, 3(03):373–381, 1980.

[7] Atlanta server website https://atlantaservers.com/ visited on 18.02.2022.

[8] Jupyter website. https://jupyter.org/ visited on 18.02.2022.

[9] Inverse Kinematics website. https://en.wikipedia.org/wiki/Inverse_kinematics visited on 20.02.2022.

[10] Liang-Chieh Chen, George Papandreou, Senior Member -IEEE Iasonas Kokkinos. Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs, 2017.

[11] Adam Paszke. A Deep Neural Network Architecture for Real-Time Semantic Segmentation, 2016.

[12] Xianzhi Du, Mostafa El-Khamy, Jungwon Lee, Larry S. Davis. A deep neural network fusion approach to fast and robust pedestrian detection, 2017.

[13] Olfa Boubaker. Systèmes Robotiques Poly-articulés, Centre de publication universitaire, Tunisie, 2015.

[14] Freenove official website https://www.freenove.com/ visited on 13.03.2022.

[15] Freenove Hexapod official website https://www.freenove.com/tutorial.html# visited on 13.03.2022

[16] A. S. Zaghloul, W. M. Hussein, A. Badawy, Y. H. Hossam El-din. On the Modeling of Phantom AX12 Six-Legged Mobile Robot. 15th International Conference on AEROSPACE SCIENCES & AVIATION TECHNOLOGY, ASAT - 15 – May 28 - 30 2013, Cairo, Egypt.

[17] Gabriel Duarte Gonçalves Pedro, Pedro Américo Almeida Magalhães Junior. Analytical Development of the Forward and Inverse Kinematics of A Robotic Leg Biologically-Inspired In Insect, 2014.

[18] Shivesh Kumar. Modular and Analytical Methods for Solving Kinematics and Dynamics of Series-Parallel Hybrid Robots, 2019.

[19] John J. Craig, Introduction to Robotics, 2005.

[20] K.S. Fu, R. C. Gonzalez, C. S. G. Lee. Robotics: Control, Sensing, Vision, and Intelligence, 2016.

[21] Ines Dlimi, Hichem Kallel. Robust Neural Control for Robotic Manipulators, 2016.

[22] Xilun Ding, Zhiying Wang, Alberto Rovetta and J.M. Zhu. Locomotion Analysis of Hexapod Robot Climbing and Walking Robots, 2010.

[23] CRAIG, John J. *Introduction to Robotics – Mechanics and Control*. 2º Ed – 1989 – Addison –Wesley Publishing Company.

[24] SPONG, Mark W.; Hutchinson, Seth; Vidyasagar, M.; *Robot Modeling and Control*; 2006. Editora. JOHN WILEY & SONS, INC.

[25] FILHO, Alberto Adade. *Fundamentos de Robótica- Cinemática, Dinâmica e Controle de Manipuladores Robóticos*. 2001 – Centro Técnico Aeroespacial – Instituto Tecnológico de Aeronáutica - São José dos Campos – SP

[26] SANTOS, Víctor M. F.; Robótica Industrial, 2004. *Departamento de Engenharia Mecânica da Universidade de Aveiro – Portugal.*

[27] Cityscape dataset website. https://www.cityscapes-dataset.com/ visited on 12.04.2022

[28] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, Jiaya Jia, 2017. Pyramid Scene Parsing Network, 2017

[29] Pytorch website. https://pytorch.org/ visited on 29.04.2022

[30] Meta Wikipedia. https://de.wikipedia.org/wiki/Meta visited on 29.04.2022

[31] OpenCV website. https://opencv.org/ visited on 03.05.2022

[32] Tensorflow board website. https://www.tensorflow.org/tensorboard?hl=fr visited on 03.05.2022

[33] The Dice loss wikipedia. https://en.wikipedia.org/wiki/Dice_coefficient visited on 07.05.2022

[34] Overfiting wikipdia. https://en.wikipedia.org/wiki/Overfitting visited on 13.05.2022

[35] Intersection over union website. https://kharshit.github.io/blog/2019/09/20/evaluation-metrics-for-object-detection-and-segmentation visited on 23.05.2022

[36] Pixel accuracy https://www.jeremyjordan.me/evaluating-image-segmentation-models/ visited on 16.06.2022

A

Appendix 1: Configuration file of semantic segmentation trained model

Config file

```
Config files are in .json format:
{
  "name": "ENet",         // training session name
  "n_gpu": 1,               // number of GPUs to use for training.
  "use_synch_bn": true,     // Using Synchronized batchnorm (for multi-GPU usage)

    "arch": {
        "type": "PSPNet", // name of model architecture to train
        "args": {
            "backbone": "resnet50",     // encoder type type
            "freeze_bn": false,     // When fine tuning the model this can be used
            "freeze_backbone": false   // In this case only the decoder is trained
        }
    },

    "train_loader": {
        "type": "VOC",          // Selecting data loader
        "args»: {
            "data_dir": "data/», // dataset path
            "batch_size": 32,     // batch size
            "augment": true,      // Use data augmentation
            "crop_size": 380,     // Size of the random crop after rescaling
            "shuffle": true,
            "base_size": 400,     // The image is resized to base_size
            "scale": true,        // Random rescaling between 0.5 and 2
            "flip": true,         // Random H-FLip
            "rotate": true,       // Random rotation between 10 and -10 degrees
            "blur": true,         // Adding a slight amount of blut to the image
            "split": "train_aug", // Split to use, depend of the dataset
            "num_workers": 8
        }
    },

    "val_loader": {  // Same for val, but no data augmentation, only a center crop
        "type": "VOC",
        "args":{
            "data_dir": "data/",
            "batch_size": 32,
            "crop_size": 480,
            "val": true,
            "split": "val",
            "num_workers": 4
        }
    },

    "optimizer": {
        "type": "SGD",
        "differential_lr": true,      // Using lr/10 for the backbone
        "args":{
"lr": 0.01,                // Learning rate
            "weight_decay": 1e-4,     // Weight decay
            "momentum": 0.9
        }
    },

    "loss": "CrossEntropyLoss2d",     // Loss (see utils/losses.py)
    "ignore_index": 255,              // Class to ignore (must be set to -1 for ADE20K) dataset
    "lr_scheduler": {
        "type": "Poly",               // Learning rate scheduler (Poly or OneCycle)
        "args": {}
    },

    "trainer": {
        "epochs": 80,                 // Number of training epochs
        "save_dir": "saved/",         // Checkpoints are saved in save_dir/models/
        "save_period": 10,            // Saving chechpoint each 10 epochs

        "monitor": "max Mean_IoU",    // Mode and metric for model performance
        "early_stop": 10,             // Number of epochs to wait before early stoping (0 to disable)

        "tensorboard": true,        // Enable tensorboard visualization
        "log_dir": "saved/runs",
        "log_per_iter": 20,

        "val": true,
        "val_per_epochs": 5         // Run validation each 5 epochs
    }
}
```

A

Appendix 2: Semantic segmentation algorithm

```
# Lint as: python3
# Copyright 2019 Google LLC
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
r"""An example of semantic segmentation.

The following command runs this script and saves a new image showing the
segmented pixels at the location specified by `output`:

```
bash examples/install_requirements.sh semantic_segmentation.py

python3 examples/semantic_segmentation.py \
  --model test_data/deeplabv3_mnv2_pascal_quant_edgetpu.tflite \
  --input test_data/bird.bmp \
  --keep_aspect_ratio \
  --output ${HOME}/segmentation_result.jpg
```
"""

import argparse

import numpy as np
from PIL import Image

from pycoral.adapters import common
from pycoral.adapters import segment
from pycoral.utils.edgetpu import make_interpreter

def create_pascal_label_colormap():
  """Creates a label colormap used in PASCAL VOC segmentation benchmark.


    Returns:
      A Colormap for visualizing segmentation results.
    """

    colormap = np.zeros((256, 3), dtype=int)
    indices = np.arange(256, dtype=int)
```

A

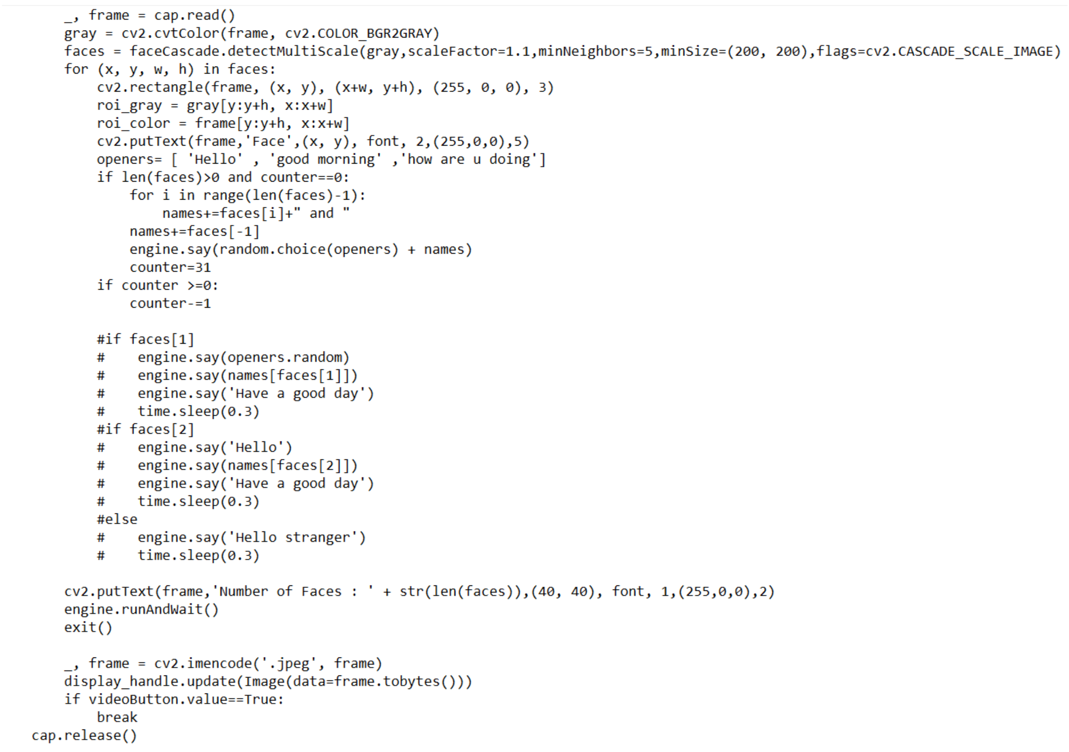
Appendix 3: Face detection algorithm

```
import matplotlib.pyplot as plt
import cv2
import numpy as np
from IPython.display import display, Image
import ipywidgets as widgets
import threading
import time
import pyttsx
import random

#setting property of text to speech module
engine = pyttsx.init()
rate = engine.getProperty('rate')
engine.setProperty('rate', rate-40)

#cascPath = "~/.local/lib/python3.8/site-packages/cv2/datahaarcascade_frontalface_default.xml"
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
faceCascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
font = cv2.FONT_HERSHEY_SIMPLEX
# Stop button
# ================
videoButton = widgets.ToggleButton(
    value=False,
    description='Stop',
    disabled=False,
    button_style='danger',
    tooltip='Description',
    icon='square'
)

# Display function
# ================
def view(button):
    cap = cv2.VideoCapture(0)
    cap.set(3, 640)
    cap.set(4, 480)
    display_handle=display(None, display_id=True)
    counter=5
    while True:
        time.sleep(0.5)
```



```
        _, frame = cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = faceCascade.detectMultiScale(gray,scaleFactor=1.1,minNeighbors=5,minSize=(200, 200),flags=cv2.CASCADE_SCALE_IMAGE)
        for (x, y, w, h) in faces:
            cv2.rectangle(frame, (x, y), (x+w, y+h), (255, 0, 0), 3)
            roi_gray = gray[y:y+h, x:x+w]
            roi_color = frame[y:y+h, x:x+w]
            cv2.putText(frame,'Face',(x, y), font, 2,(255,0,0),5)
            openers= [ 'Hello' , 'good morning' ,'how are u doing']
            if len(faces)>0 and counter==0:
                for i in range(len(faces)-1):
                    names+=faces[i]+" and "
                names+=faces[-1]
                engine.say(random.choice(openers) + names)
                counter=31
            if counter >=0:
                counter-=1

            #if faces[1]
            #    engine.say(openers.random)
            #    engine.say(names[faces[1]])
            #    engine.say('Have a good day')
            #    time.sleep(0.3)
            #if faces[2]
            #    engine.say('Hello')
            #    engine.say(names[faces[2]])
            #    engine.say('Have a good day')
            #    time.sleep(0.3)
            #else
            #    engine.say('Hello stranger')
            #    time.sleep(0.3)

        cv2.putText(frame,'Number of Faces : ' + str(len(faces)),(40, 40), font, 1,(255,0,0),2)
        engine.runAndWait()
        exit()

        _, frame = cv2.imencode('.jpeg', frame)
        display_handle.update(Image(data=frame.tobytes()))
        if videoButton.value==True:
            break
    cap.release()
```

A

Appendix 4: Face recognition algorithm

```
#! /usr/bin/python
# import the necessary packages
from imutils import paths
import face_recognition
#import argparse
import pickle
import cv2
import os

# our images are located in the dataset folder
print("[INFO] start processing faces...")
imagePaths = list(paths.list_images("dataset"))

# initialize the list of known encodings and known names
knownEncodings = []
knownNames = []

# loop over the image paths
for (i, imagePath) in enumerate(imagePaths):
    # extract the person name from the image path
    print("[INFO] processing image {}/{}".format(i + 1,
        len(imagePaths)))
    name = imagePath.split(os.path.sep)[-2]

    # load the input image and convert it from RGB (OpenCV ordering)
    # to dlib ordering (RGB)
    image = cv2.imread(imagePath)
    rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # detect the (x, y)-coordinates of the bounding boxes
    # corresponding to each face in the input image
    boxes = face_recognition.face_locations(rgb,
        model="hog")

    # compute the facial embedding for the face
    encodings = face_recognition.face_encodings(rgb, boxes)
    # loop over the encodings
    for encoding in encodings:
        # add each encoding + name to our set of known names and
        # encodings
        knownEncodings.append(encoding)
```

A

Appendix 5: Face comparing algorithm

```
  for shift in reversed(range(8)):
    for channel in range(3):
      colormap[:, channel] |= ((indices >> channel) & 1) << shift
    indices >>= 3

  return colormap

def label_to_color_image(label):
  """Adds color defined by the dataset colormap to the label.

  Args:
    label: A 2D array with integer type, storing the segmentation label.

  Returns:
    result: A 2D array with floating type. The element of the array
      is the color indexed by the corresponding element in the input label
      to the PASCAL color map.

  Raises:
    ValueError: If label is not of rank 2 or its value is larger than color
      map maximum entry.
  """
  if label.ndim != 2:
    raise ValueError('Expect 2-D input label')

  colormap = create_pascal_label_colormap()

  if np.max(label) >= len(colormap):
    raise ValueError('label value too large.')

  return colormap[label]

def main():
  parser = argparse.ArgumentParser()
  parser.add_argument('--model', required=True,
                      help='Path of the segmentation model.')
  parser.add_argument('--input', required=True,
                      help='File path of the input image.')
  parser.add_argument('--output', default='semantic_segmentation_result.jpg',
                      help='File path of the output image.')
  parser.add_argument(
  interpreter = make_interpreter(args.model, device=':0')
  interpreter.allocate_tensors()
  width, height = common.input_size(interpreter)

  img = Image.open(args.input)
  if args.keep_aspect_ratio:
    resized_img, _ = common.set_resized_input(
        interpreter, img.size, lambda size: img.resize(size, Image.ANTIALIAS))
  else:
    resized_img = img.resize((width, height), Image.ANTIALIAS)
    common.set_input(interpreter, resized_img)

  interpreter.invoke()

  result = segment.get_output(interpreter)
  if len(result.shape) == 3:
    result = np.argmax(result, axis=-1)

  # If keep_aspect_ratio, we need to remove the padding area.
  new_width, new_height = resized_img.size
  result = result[:new_height, :new_width]
  mask_img = Image.fromarray(label_to_color_image(result).astype(np.uint8))

  # Concat resized input image and processed segmentation results.
  output_img = Image.new('RGB', (2 * new_width, new_height))
  output_img.paste(resized_img, (0, 0))
  output_img.paste(mask_img, (width, 0))
  output_img.save(args.output)
  print('Done. Results saved at', args.output)

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

Abstract

This work conc This work concerns the implementation of an algorithm for autonomous navigation for an Hexapod robot based on a Semantic segmentation model using a Cityscape database. This work is a preliminary and will lead later to an implementation on a real car.

This proposed Autonomous navigation project aims to provide a Machine learning algorithm using Python language trained on image classification, using OpenCV and TensorFlow, to the task of semantic segmentation by applying the conventional neural network with City Scape dataset.

Keywords: Facial recognition, Hexapod, geometric model, semantic segmentation, Python, TensorFlow, OpenCV, Artificial intelligence, robot navigation.

Résumé

Ce travail consiste en l'implémentation d'un algorithme de navigation autonome pour un robot Hexapode qui repose sur un modèle de segmentation sémantique exploitant une base de données de type Cityscape. Ce travail est préliminaire et conduira ultérieurement à une implémentation sur une voiture réelle.

Ce projet commencera par la détermination des logiciels et des équipements informatiques nécessaires. Suivi par l'assemblage d'un Hexapode qui sera utilisé dans la tâche essentielle qui est l'implémentation d'une solution de navigation autonome.

Mots-clés : Reconnaissance faciale, Hexapod, modèle géométrique, segmentation sémantique, Python, TensorFlow, OpenCV, intelligence artificielle, navigation d'un robot.

ملخص

يتكون هذا العمل من تنفيذ خوارزمية تنقل مستقلة لروبوت Hexapod الذي يعتمد على نموذج تجزئة دلالي يستغل قاعدة بيانات من نوع Cityscape. هذا العمل تمهيدي وسيؤدي لاحقًا إلى تنفيذ على سيارة حقيقية.

سيبدأ هذا المشروع بتحديد برامج ومعدات الكمبيوتر اللازمة. يتبعه تجميع Hexapod الذي سيتم استخدامه في المهمة الأساسية وهي تنفيذ حل السياقة المستقلة.

الكلمات الرئيسية: التعرف على الوجه ، Hexapod ، النموذج الهندسي ، التجزئة الدلالية ، Python، TensorFlow، OpenCV ، الذكاء الاصطناعي ، السياقة الروبوتية.