ACE - AutoCar Education Kit User Manual

«Bui Phi Hung, Pham Thi Thuc Trinh, Nguyen Quang Hiep»

Author

Contents

44 | Chapter II: Object Detection

57 | Chapter III: Object Tracking

62 | Overview of expected results

63 | Conclusion

Introduce

AutoCad Education Kit is fun, easy to apply, flexible, and provides hands-on solutions for:

- Image processing
- Run in the lane
- Object detection
- Autonomous car

This course is an integration of the above concepts and is not intended to be a beginner's course on any of the above topics. Up to now, the content in this document is most suitable for university/college level, requiring students to study technical fields related to autonomous vehicles.

This course doesn't include:

- Mechatronics
- An alternative to university/college courses on the topics we cover

Request:

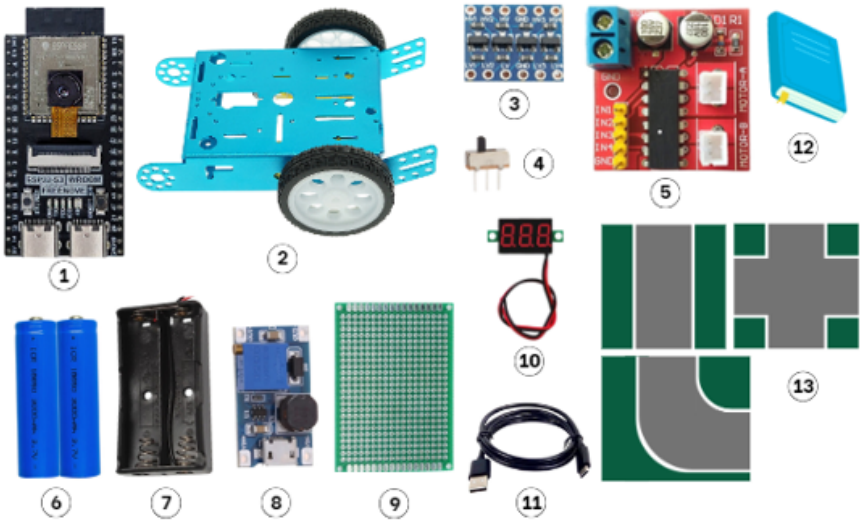
- Basic knowledge of image processing and deep learning
- Programming skills (C/C++ and Python)

Guide

2.1 Hardware

2.1.1 Arduino Car Kit

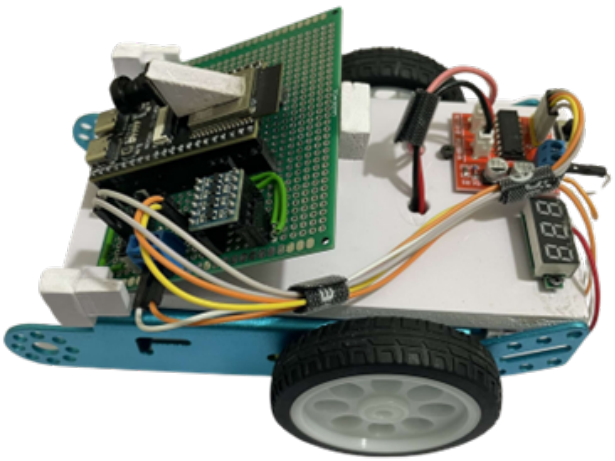
The vehicle's components are described in the following photo.



1. ESP32S3-OV2640/CAM 160°
2. Car chassis FUT5024
3. Logic Level Converter Circuit
4. On/Off switch (x2)
5. Motor control board L298
6. Battery 18650 (x2)
7. Battery box
8. Booster circuit ML3608
9. FR4 PCB Test board 7x9cm
10. Voltage Meter
11. USB-A to USB-C cable
12. Document
13. Map: Straight line (x6)
    Crossroads(x2)
    Turn lane(x2)

The completed vehicle will have the shape below. It will have the following basic parameters:

- Size: 13x16cm
- Voltage: 3.6V - 4V
- Speed: 165
- FPS: 13 - 15

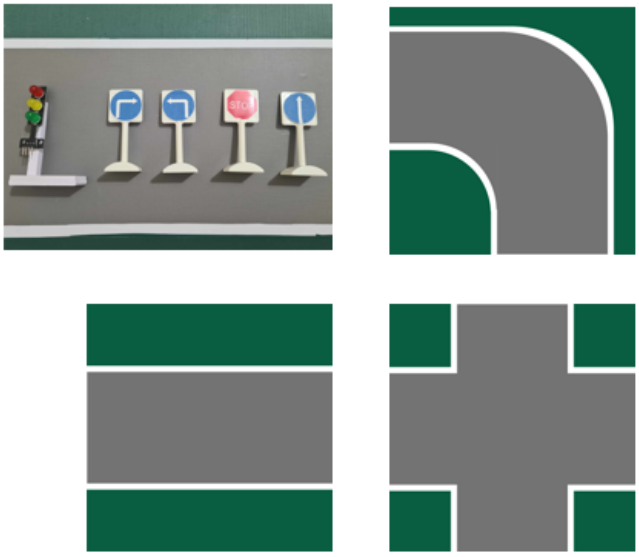


2.1.2 Map shape

The map is made from formex material and has dimensions of 40x40x0.8cm. Lane width is 16cm and road markings are 1cm.

2.1.3 Traffic sign size

The sign will be 3x3cm in size with a height of 7cm.



2.2 Setup Software

2.2.1 Resource

Download project's file from this link.

2.2.2 Transit data by UDP

When transferring data between computers over a local Wi-Fi network, the data may be blocked by the Windows firewall. To fix this, your computer needs to follow these steps.

Set up OpenSSH
Find Optional Features and open this window.

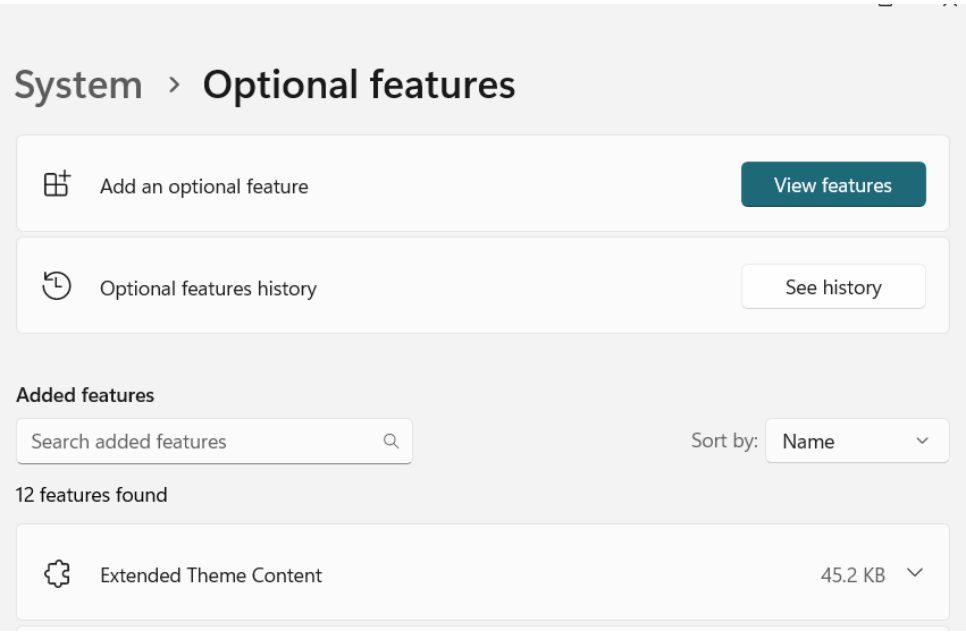


Figure 2.1: Optinal features

Then, select 'View features' and search 'OpenSSH Server'.

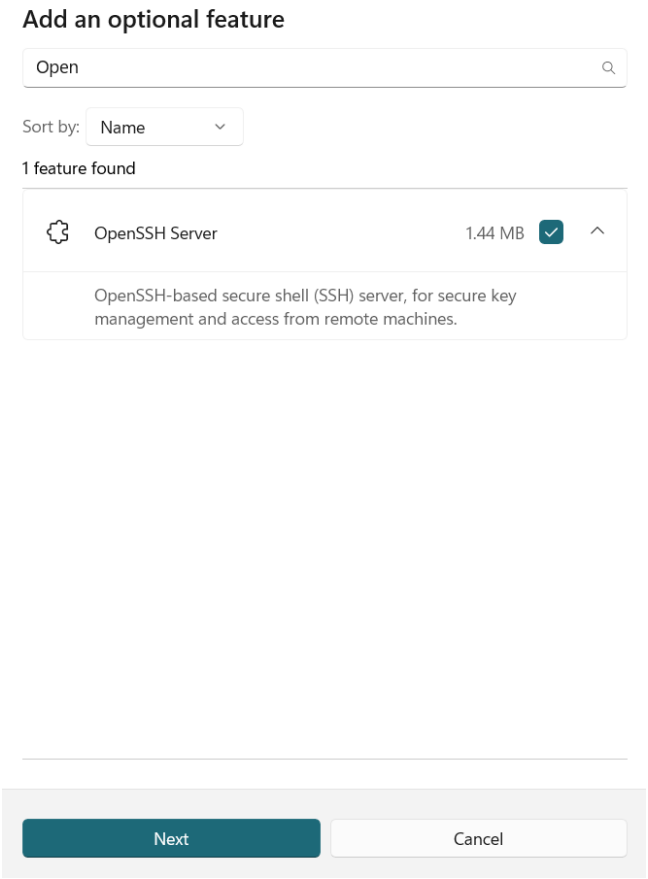


Figure 2.2: Find and add OpenSSH server

This process will take about 15-20 minutes.

Add inbound rules

Open 'Windows Defender Firewall with advanced security' and select action 'New Rule'

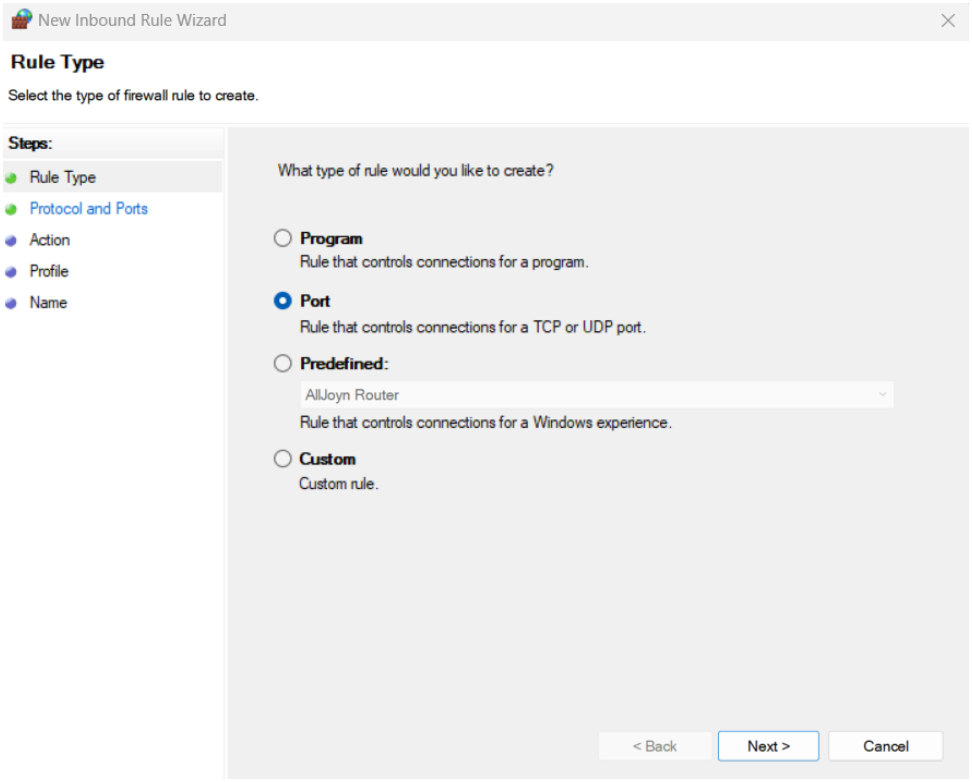


Figure 2.3: New Rule in Inbound rules

In New rule, select Port. Then, select UDP, and enter the ports that you use to receive data. In this document's laboratories, we use port 7000, 8000, 3000, 3001, 8001, 7001.

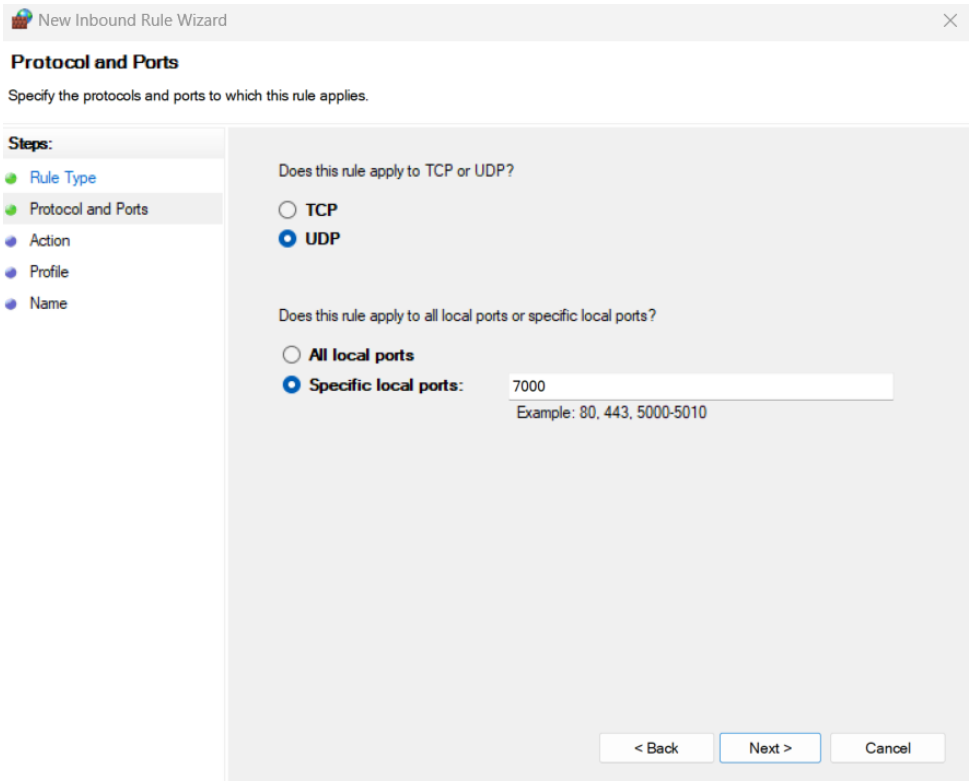


Figure 2.4: Enter the ports receive data

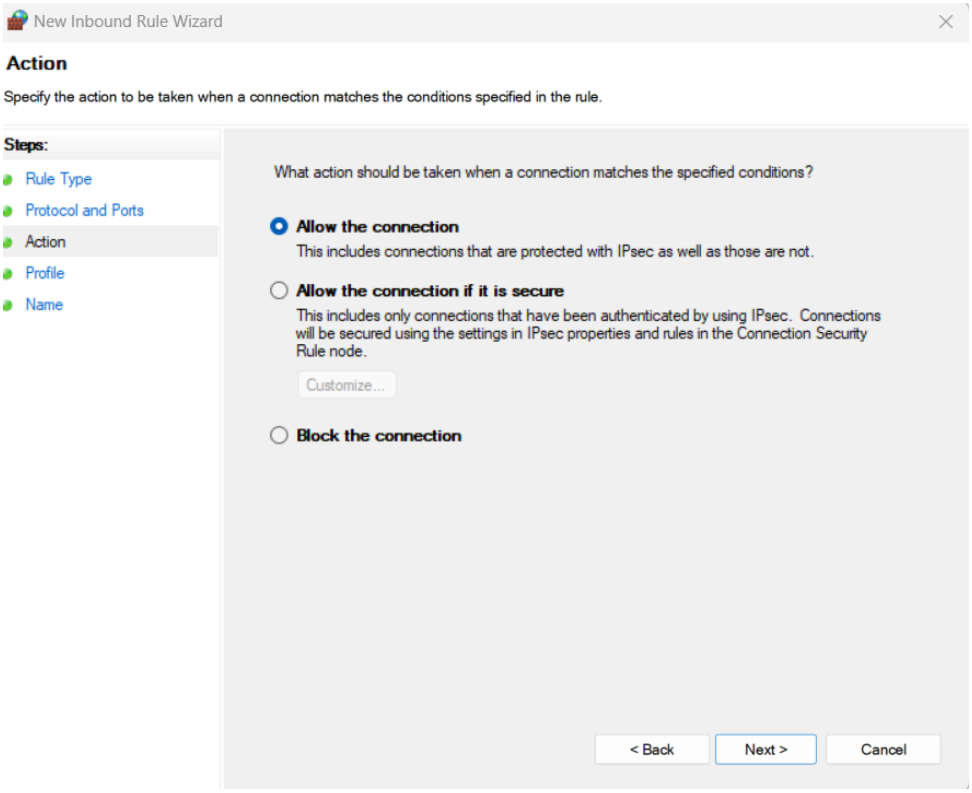


Figure 2.5: Allow the connect

Next, select 'Next' twice. Then, enter name and description and finish. Finally, restart the computer.

2.2.3 Python environment

Install Python version 3.12.4 at here

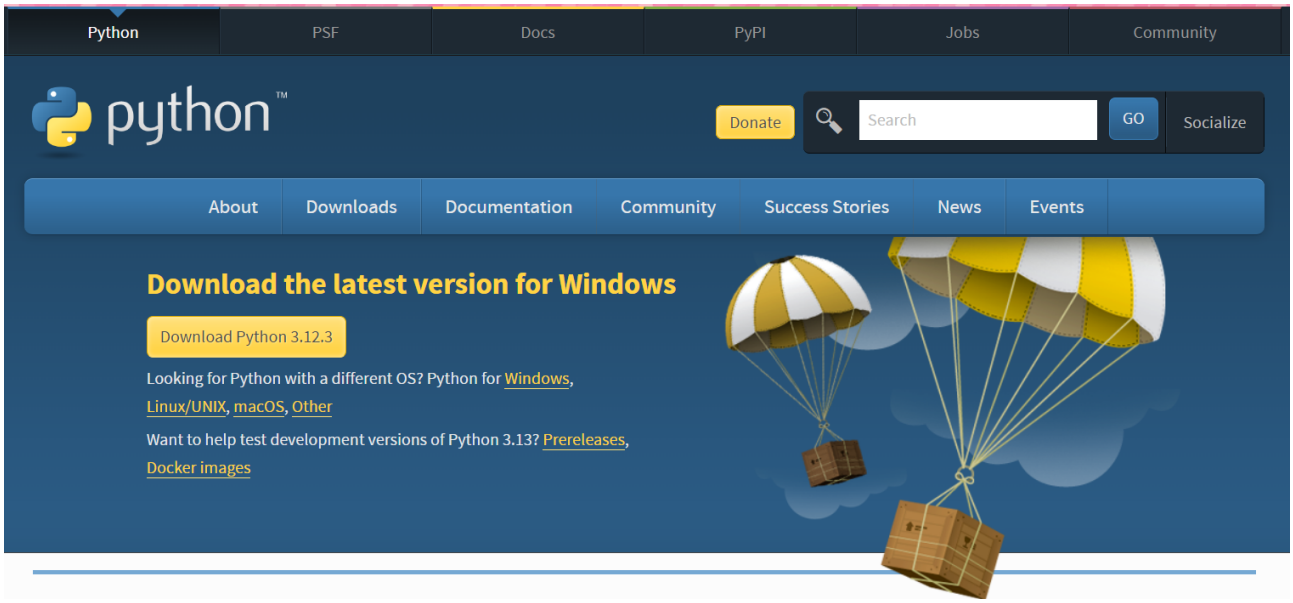


Figure 2.6: Interface of Python website

2.2.4 Visual Studio Code

Install Visual Studio Code at here

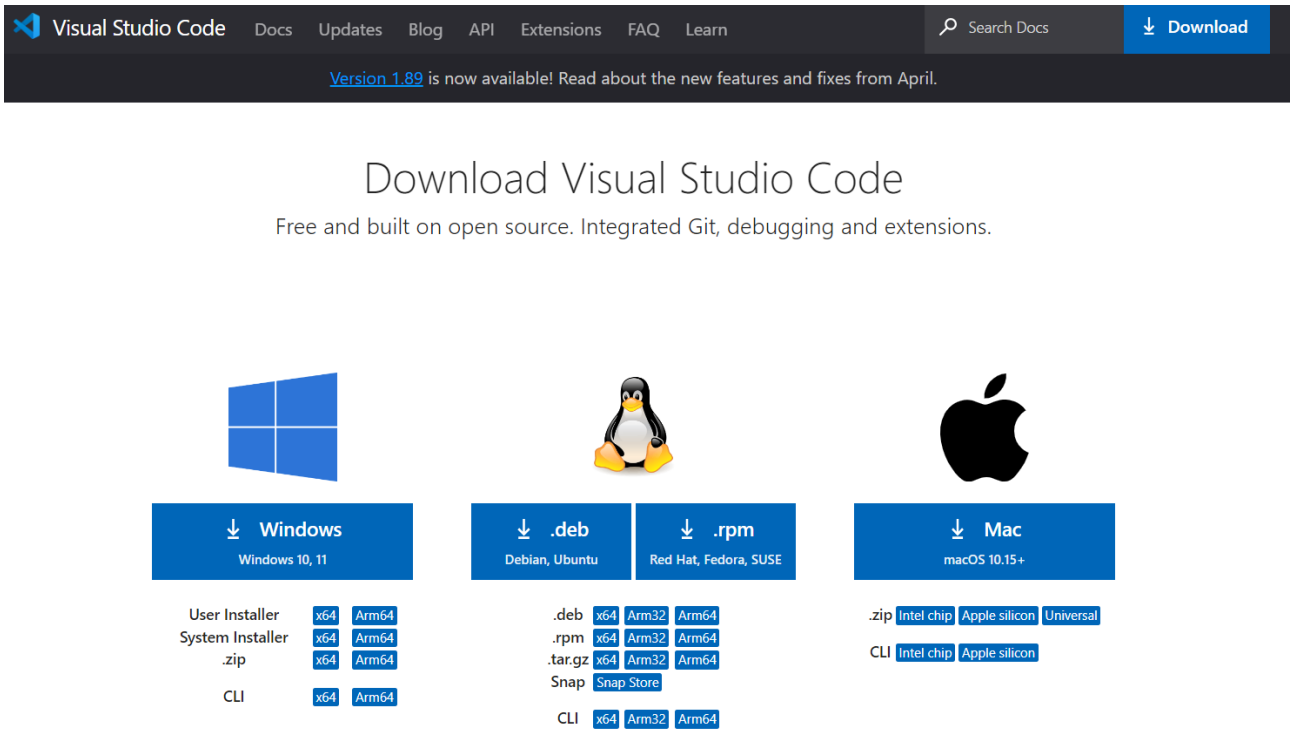


Figure 2.7: Interface of Visual Studio Code Website

2.2.5 Install Jupyter notebook

To run the labs, you need to install Jupyter Notebook. First, go to extensions in Vs Code, look up Jupyter Notebook, select the Microsoft one and press install.

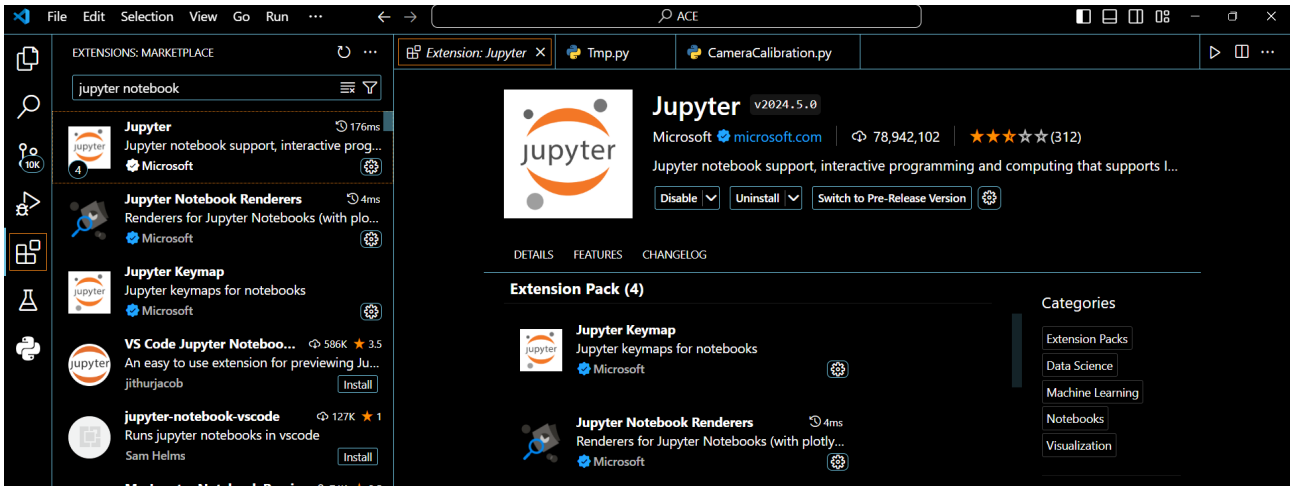


Figure 2.8: Jupyter Notebook

2.2.6 Install necessary library

After installing, we install the necessary libraries. Open terminal and enter command 'pip install -r environments.txt'. Use 'pip list' command to check installed libraries.

```
PS D:\Summer 2024\Capstone project\Final\ACE-main\Lab> pip list
Package            Version
------------------ ---------------
asttokens          2.4.1
certifi            2024.7.4
charset-normalizer 3.3.2
colorama           0.4.6
comm               0.2.2
contourpy          1.2.1
cycler             0.12.1
debugpy            1.8.5
decorator          5.1.1
deep-sort-realtime 1.3.2
```

Figure 2.9: Library installed

2.2.7 Ardunio IDE

Installed Arduino IDE at here

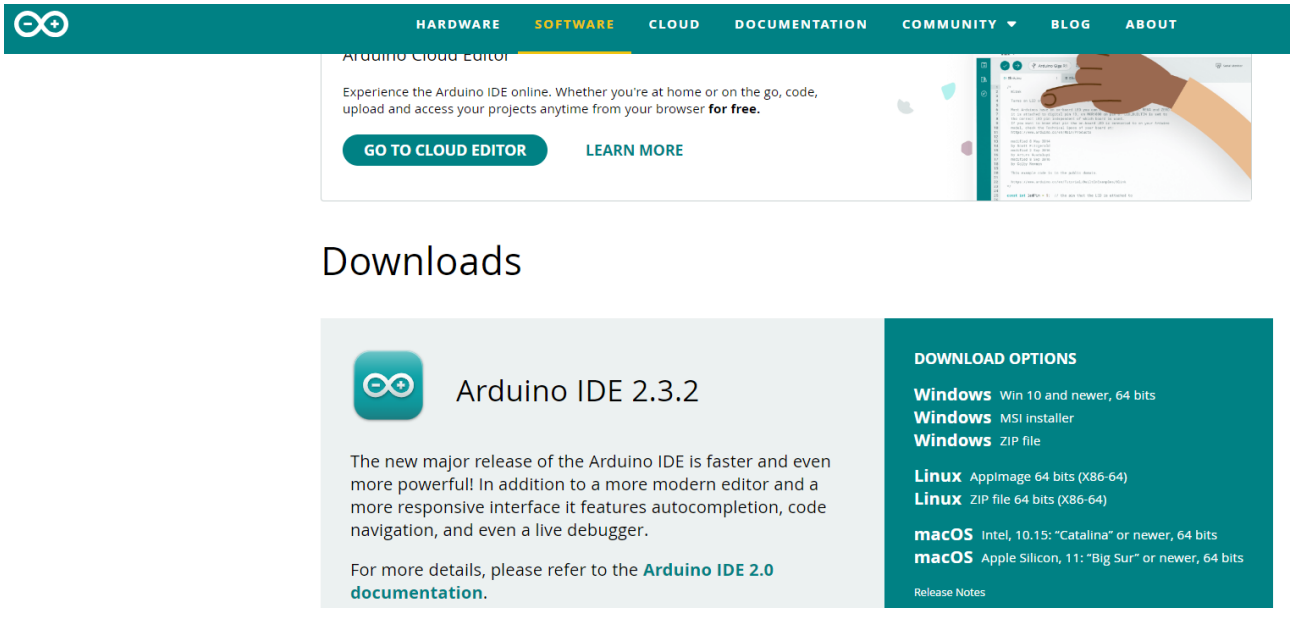


Figure 2.10: Interface of Arduino website

It will then appear on the donation website as follows. If you're generous, please donate. If you are poor like us, choose 'JUST DOWNLOAD'.

Figure 2.11: Interface of Arduino donate website

The downloaded file will have the following format.

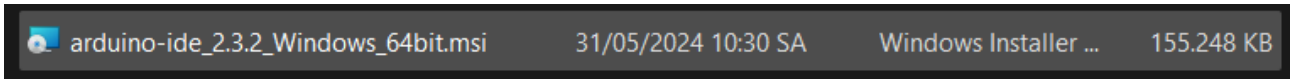


Figure 2.12: Arduino installation file

After the file finishes running, it will automatically create an icon on your screen.

Figure 2.13: Arduino icon

The app will ask you to install the necessary libraries, just press 'install' all.

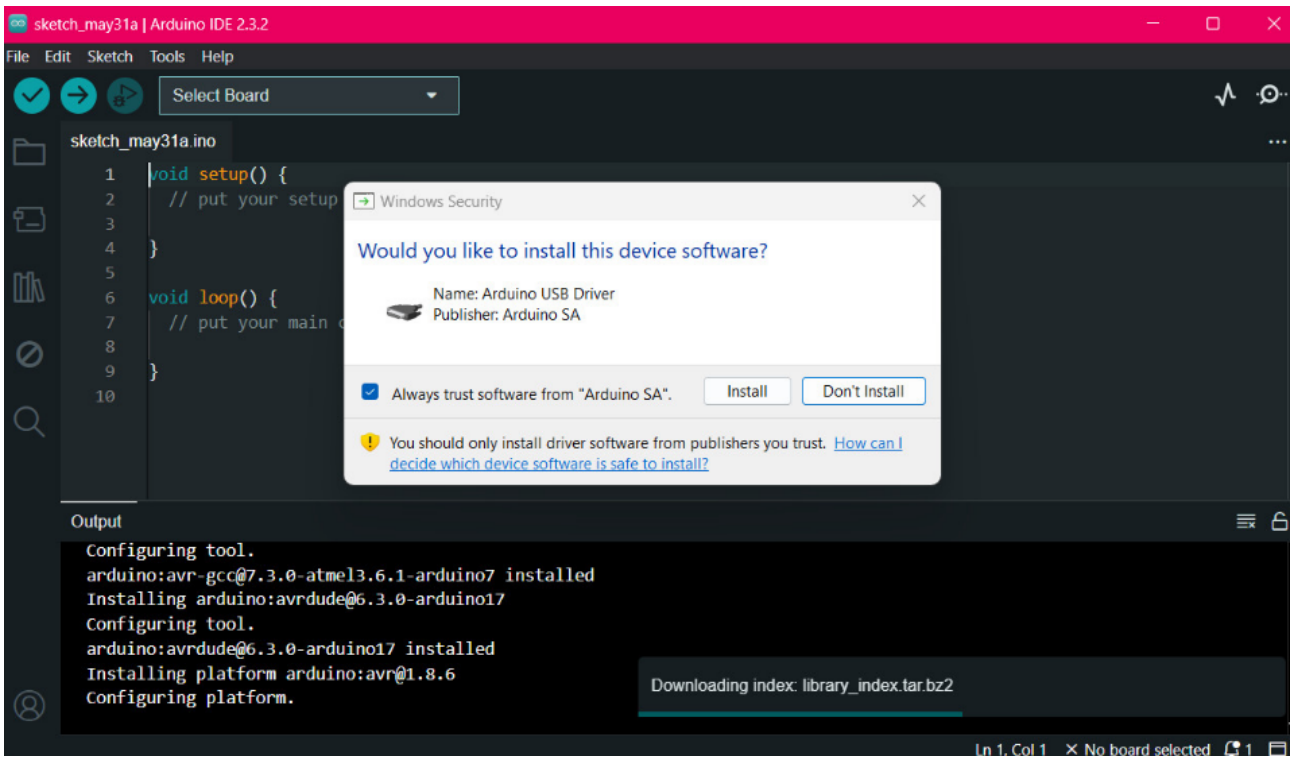


Figure 2.14: Install libraries for arduino

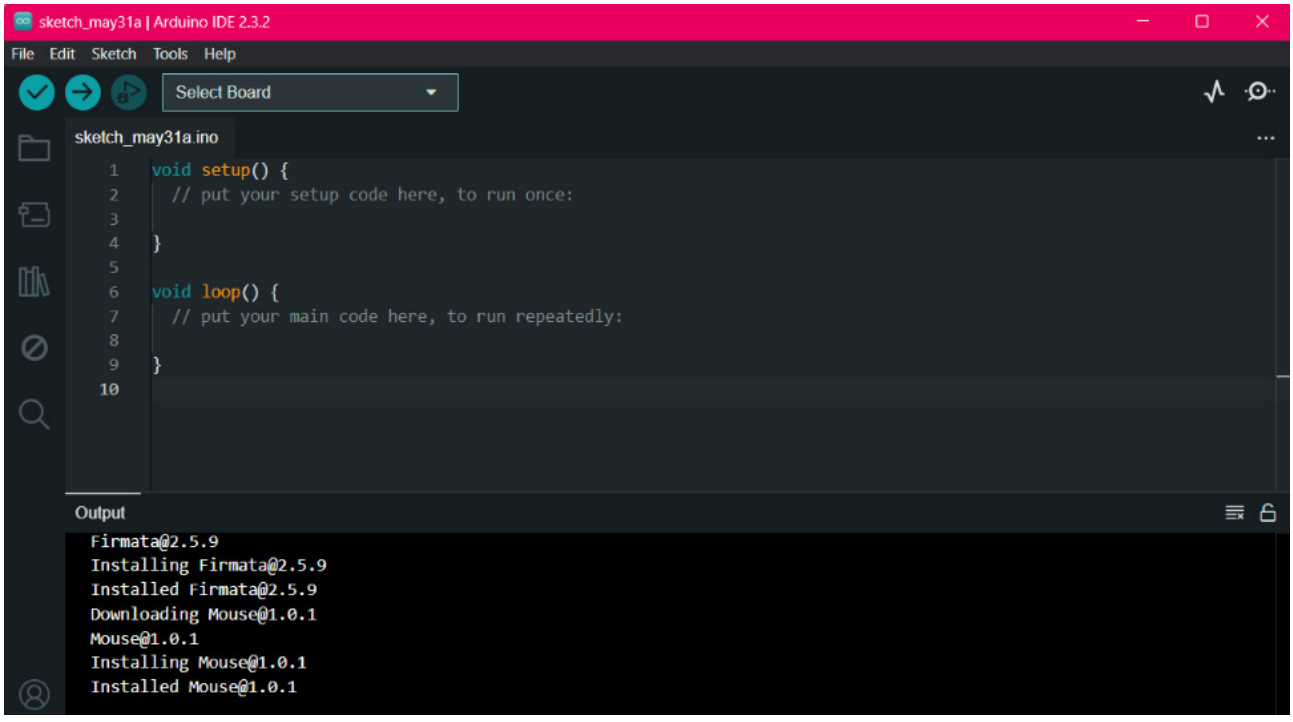


Figure 2.15: Installed the library successfully for arduino

We continue to install the library for the esp32 camera. Select tools -> board -> boards manager. Type 'esp32' in the search box, select 'esp32 by Espressif Systems' and select version 1.0.4. Then press 'INSTALL'.

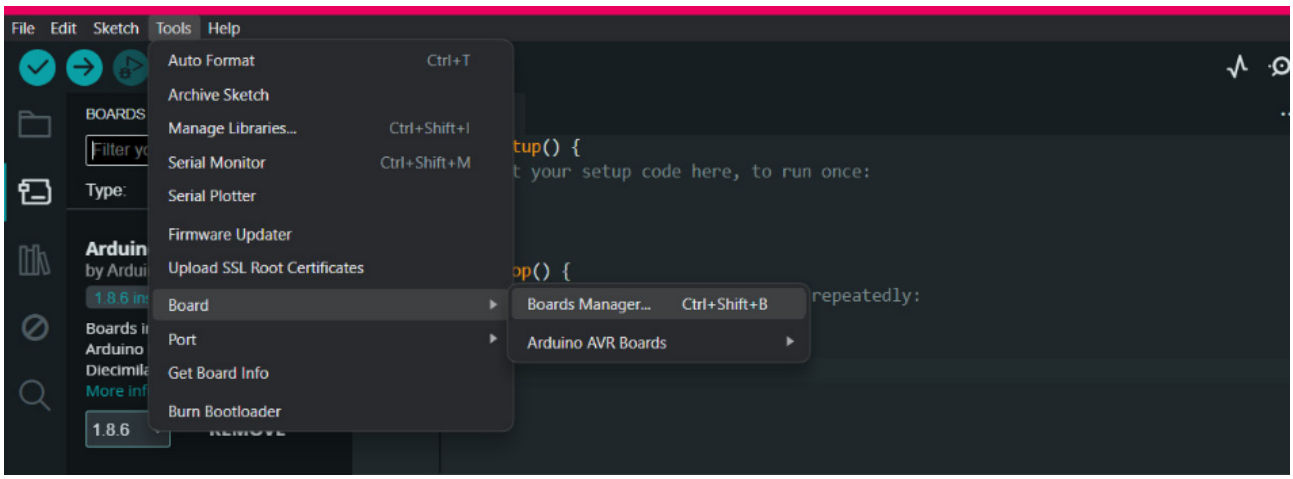


Figure 2.16: Tools -> Board -> Boards Manager

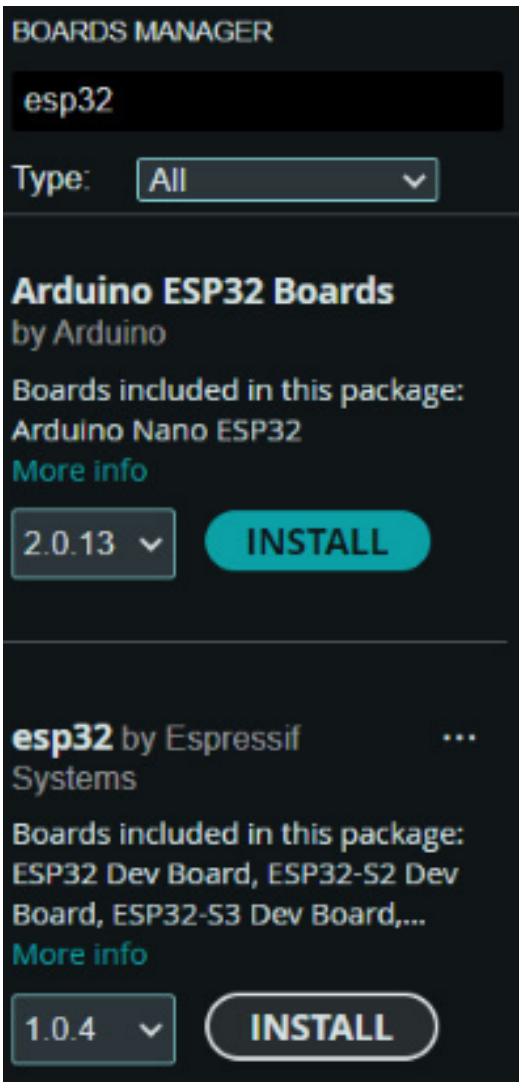


Figure 2.17: Select the appropriate version for the camera library

Figure 2.18: Install camera library successfully

2.2.8 Transmit code to the ACE kit (camera and arduino circuit)

Camera

Open the file 'Camera.ino' using the Arduino IDE app. You will see its interface as follows.

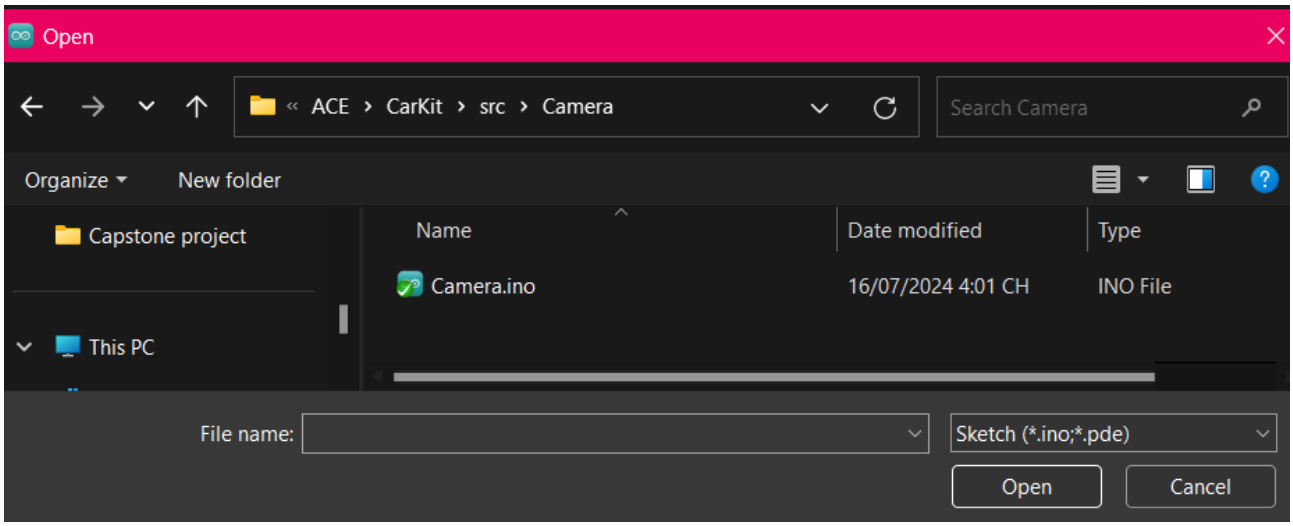


Figure 2.19: Camera.ino file location

Camera.ino | CameraIndex.h | CameraPins.h | CameraWebServerAccessPoint.cpp | CameraWebServerAccessPoint.h | app_httpd.cpp

```
#ifndef _CameraWebServer_AP_H
#define _CameraWebServer_AP_H

#include "esp_camera.h"
#include <WiFi.h>

class CameraWebServerAccessPoint {
  public:
    // Initialize the camera web server
    void CameraWebServerAccessPointInit();

  private:
    const char *ssid = "NoHope";
    const char *password = "33333333";
};

#endif

```

Figure 2.21: Edit wifi information

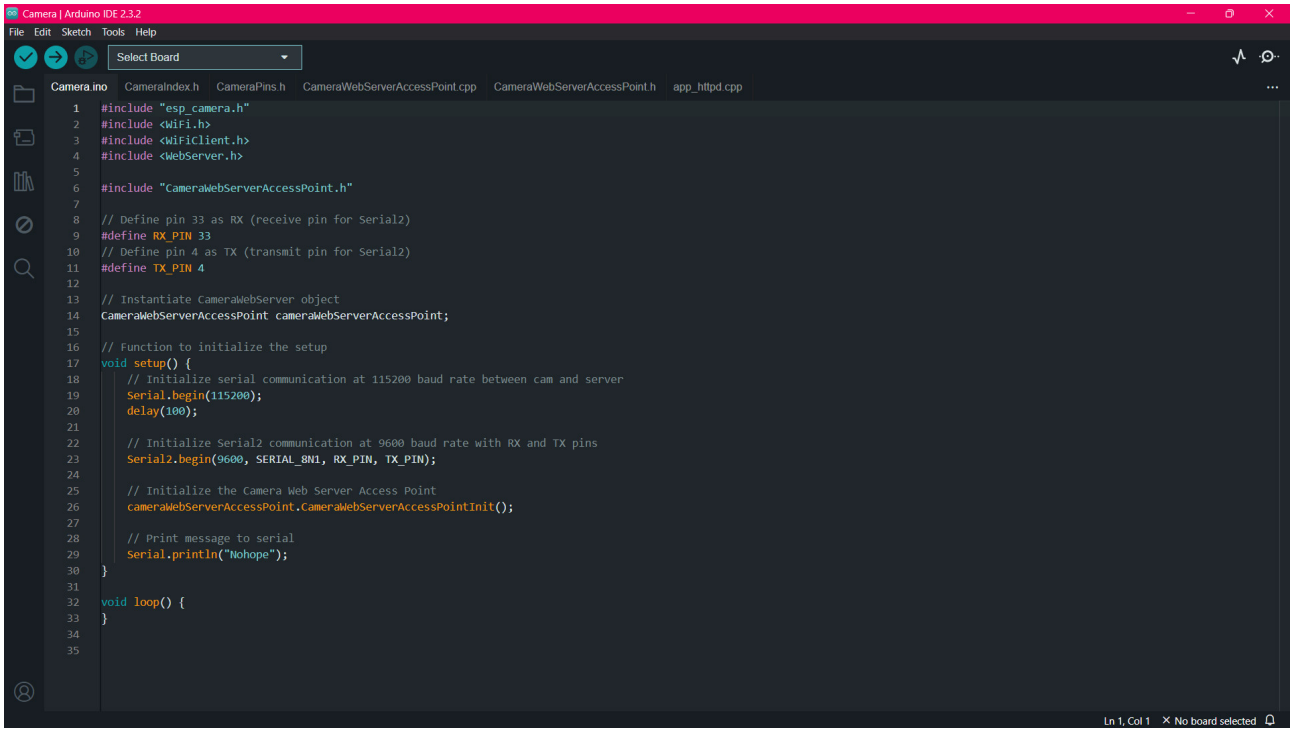


Figure 2.20: Interface of camera.ino file

To export code to the camera, you will need to adjust the following 4 things: 1 is wifi, 2 is the board module, 3 is the wire port, 4 is some default settings.

- 1. Open file 'CameraWebServerAccessPoint.h', edit '*ssid' to your wifi , '*password' to your wifi password..

- 2. Select Tools -> Board -> esp32 -> ESP32 Dev Module.

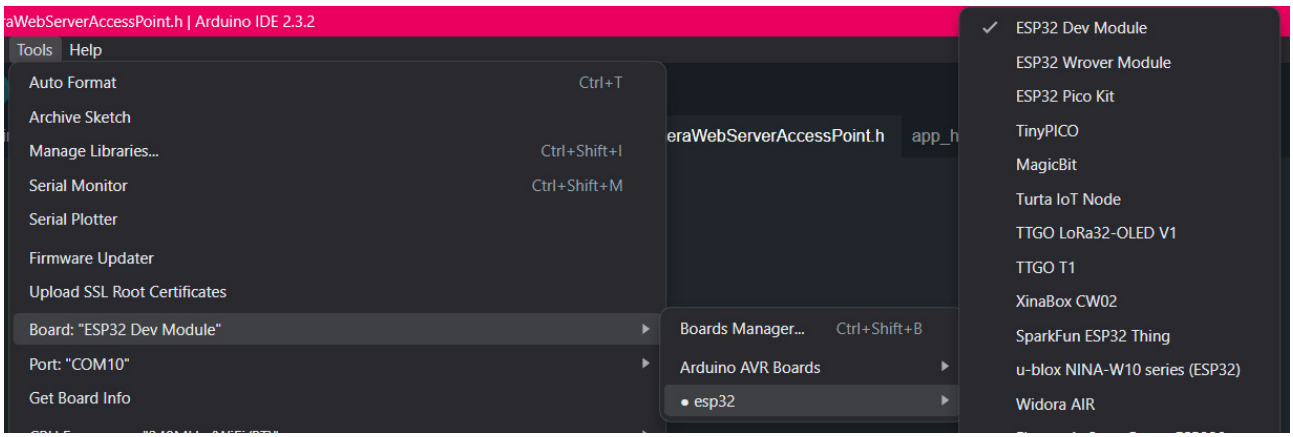


Figure 2.22: Select the board module

- 3. To choose the port exactly, you will need to determine the number of the USB port you plug into. Go to Device Manager of your device, and check. After getting the standard port, select Tools -> Port -> select the correct port you just checked.

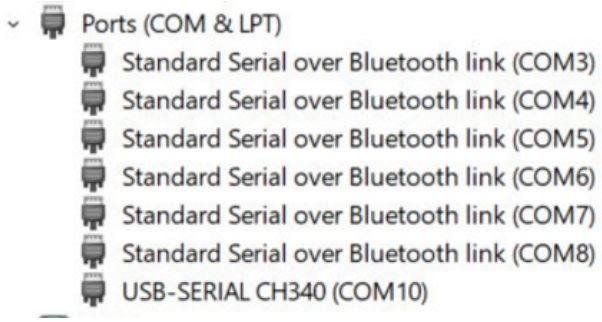


Figure 2.23: Check the USB port number

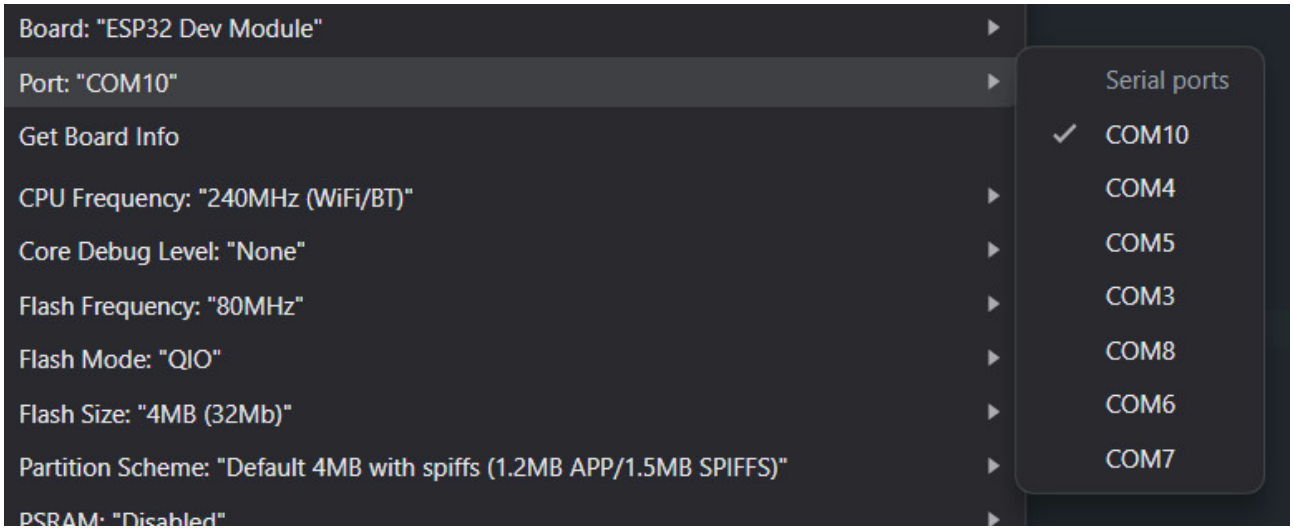


Figure 2.24: Camera's Port

- 4. Select Tools -> Select 'Partition Scheme' Huge APP and enabled 'PSRAM".

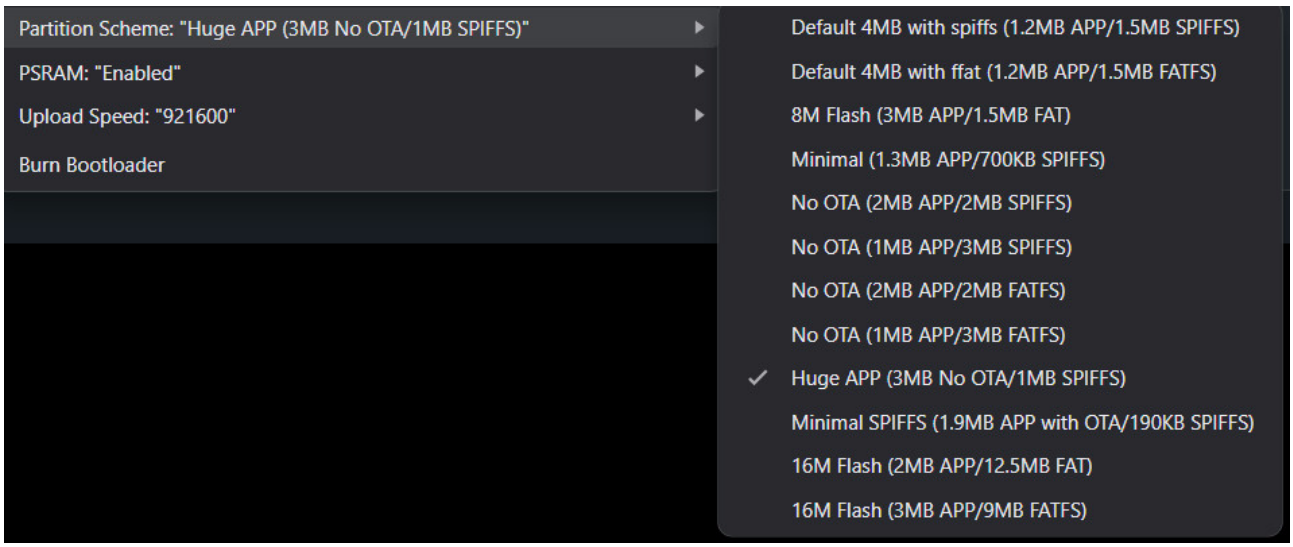


Figure 2.25: Choose the correct scheme and PSRAM

Then we proceed to transmit the code to the camera by pressing the upload button (symbolized by '->') in the left corner of the screen.

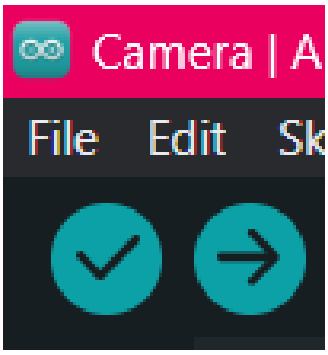


Figure 2.26: Upload file

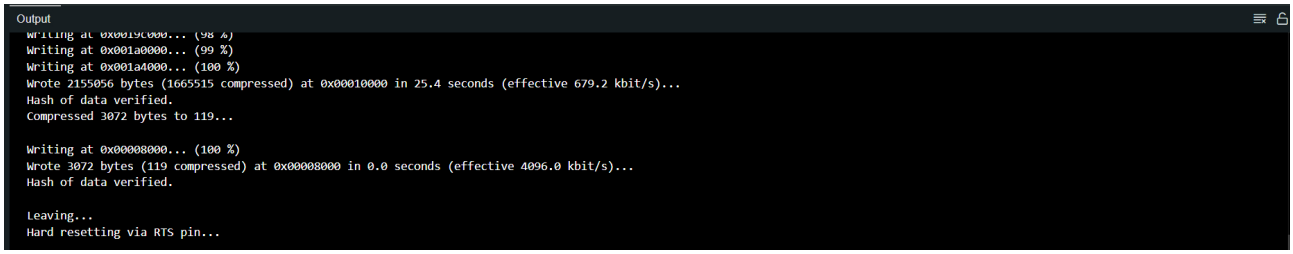


Figure 2.27: Upload file successfully

Car

In the case of Car, do the same as the camera code. We continue to open the car.ino file with the Arduino IDE app, select the port and board as 'arduino Uno', and then proceed to upload.

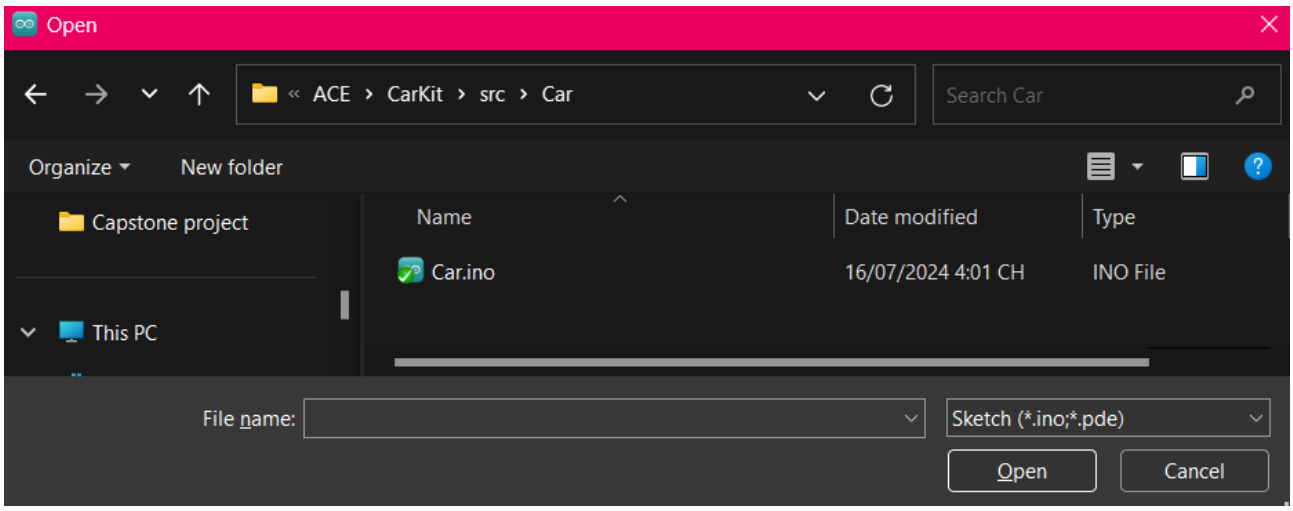


Figure 2.28: Car file location

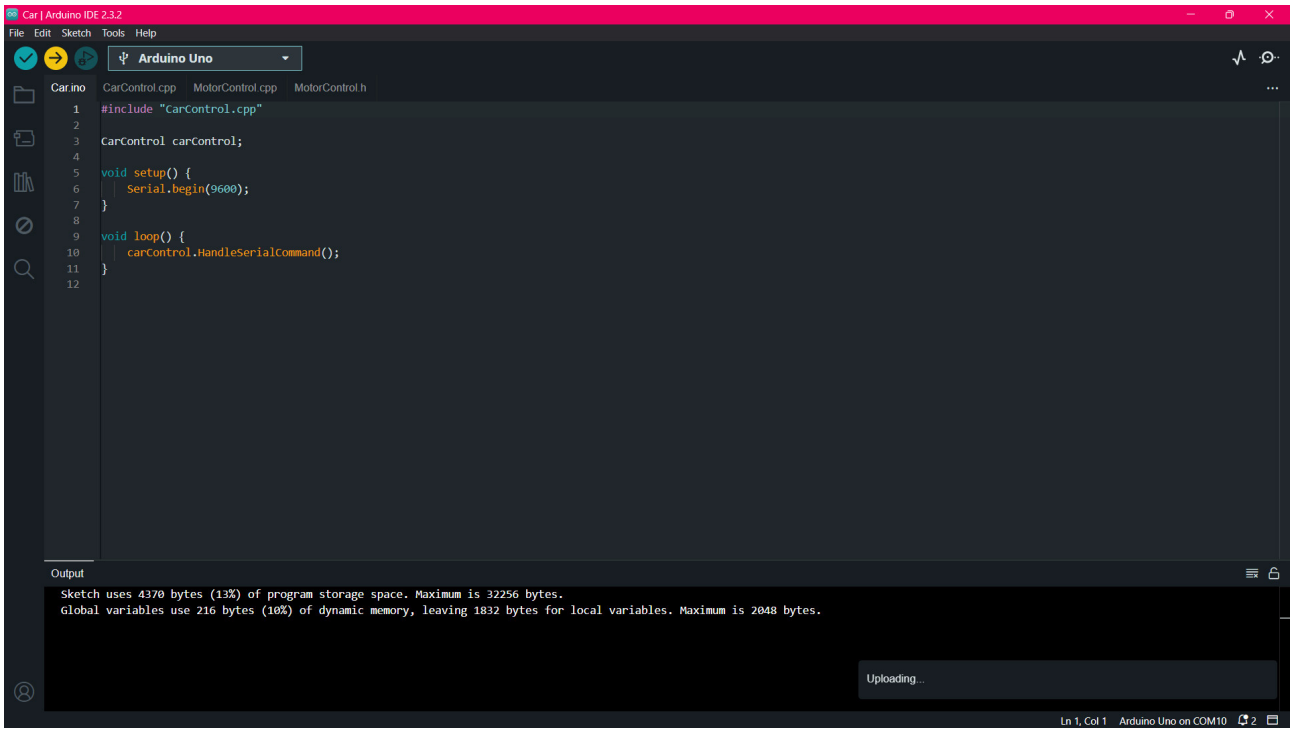


Figure 2.29: Select the appropriate port and board then proceed to upload

2.2.9 Get the camera's IP address

The camera's IP address is very important, it is responsible for transmitting and receiving data from the camera via wifi. Therefore, we need to determine the correct IP to serve the coding process.

How to get IP is extremely simple. Select Tools -> Serial Monitor, and switch to band 115200. Press the reset button on the esp 32 camera and wait for it to return the IP.

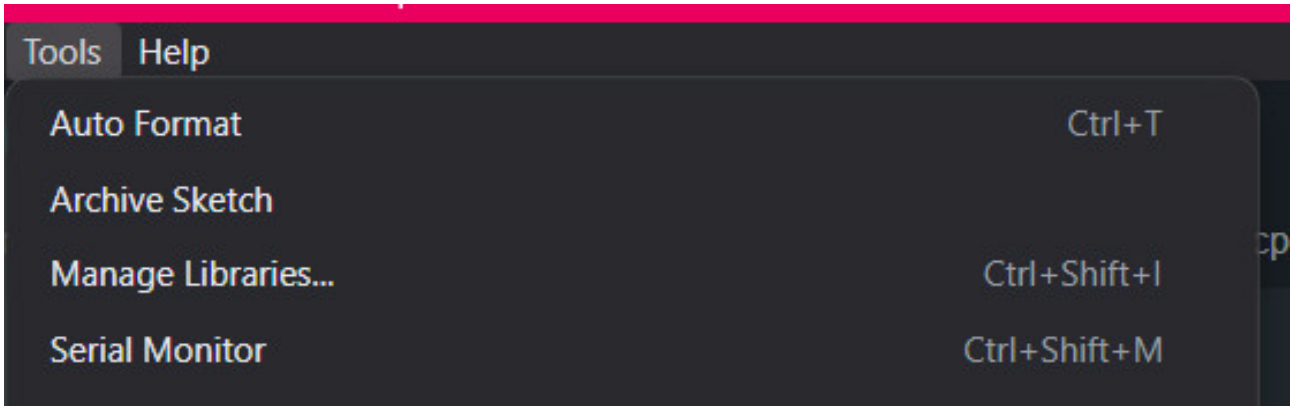


Figure 2.30: Open serial monitor

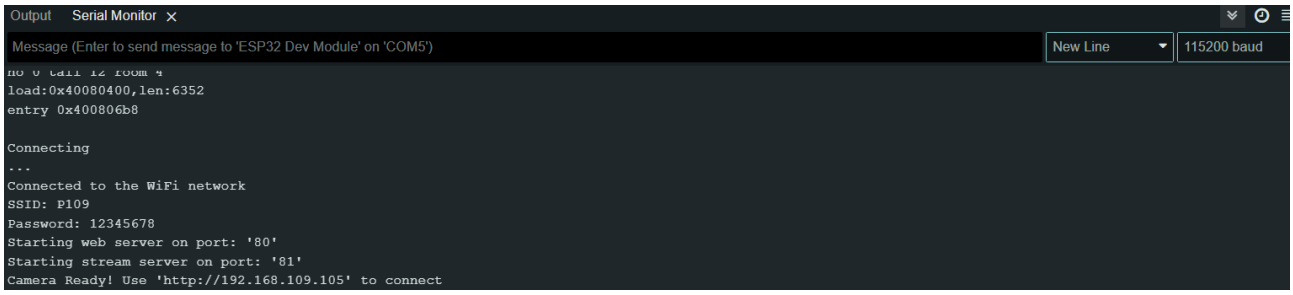


Figure 2.31: IP's camera

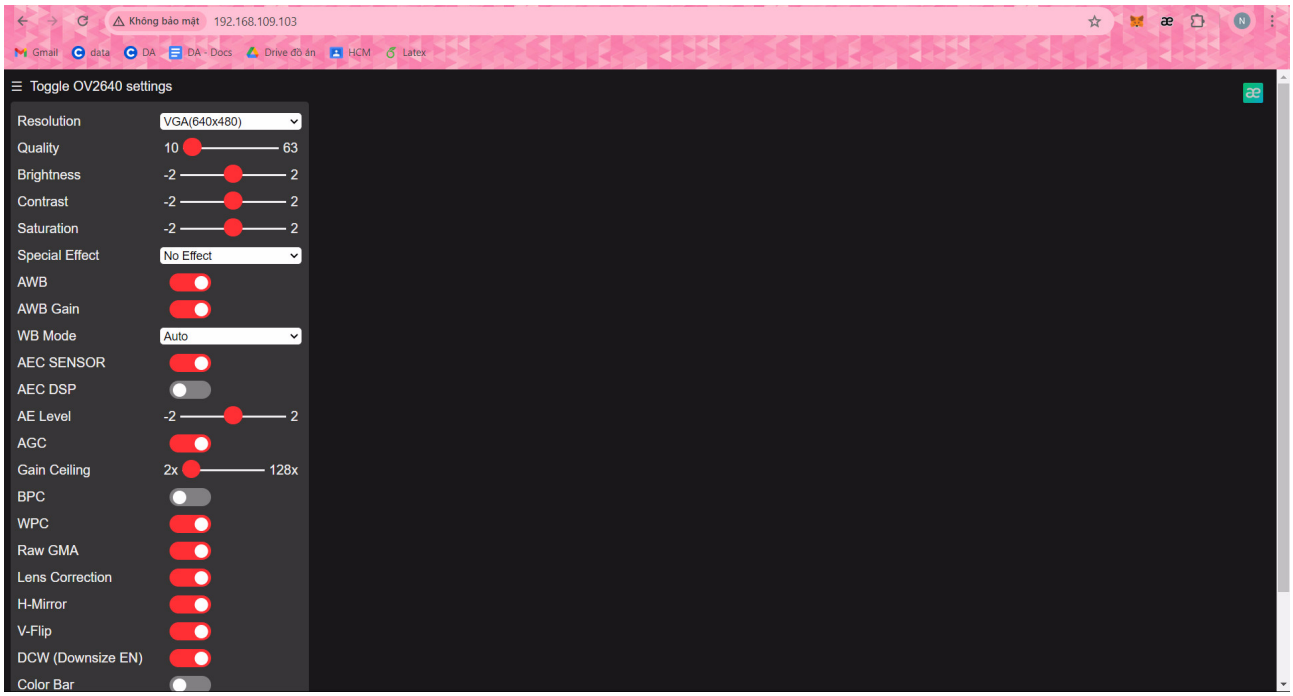


Figure 2.32: You can test on the web and adjust some camera parameters

Chapter I: Lane Detection

3.1 Get acquainted with the OpenCV library

3.1.1 Overview

OpenCV is one of the leading open-source libraries for computer vision, image processing, and machine learning. It provides powerful GPU acceleration capabilities for real-time applications. OpenCV plays a significant role in autonomous vehicles. This practice exercise will guide you through the most basic tasks to help you master the foundational knowledge and how to apply OpenCV.

3.1.2 Learning goals

Upon completing this exercise, learners will gain knowledge of:

- Reading image and video data
- Basic functions in OpenCV

3.1.3 Related knowledge

- Python
- Image processing
- OpenCV

3.1.4 Required background

To complete this exercise, you will need the following knowledge:

- Basic programming skills with Python

3.1.5 Problem

Objective: Process input images/videos using basic operations in OpenCV.
Requirements

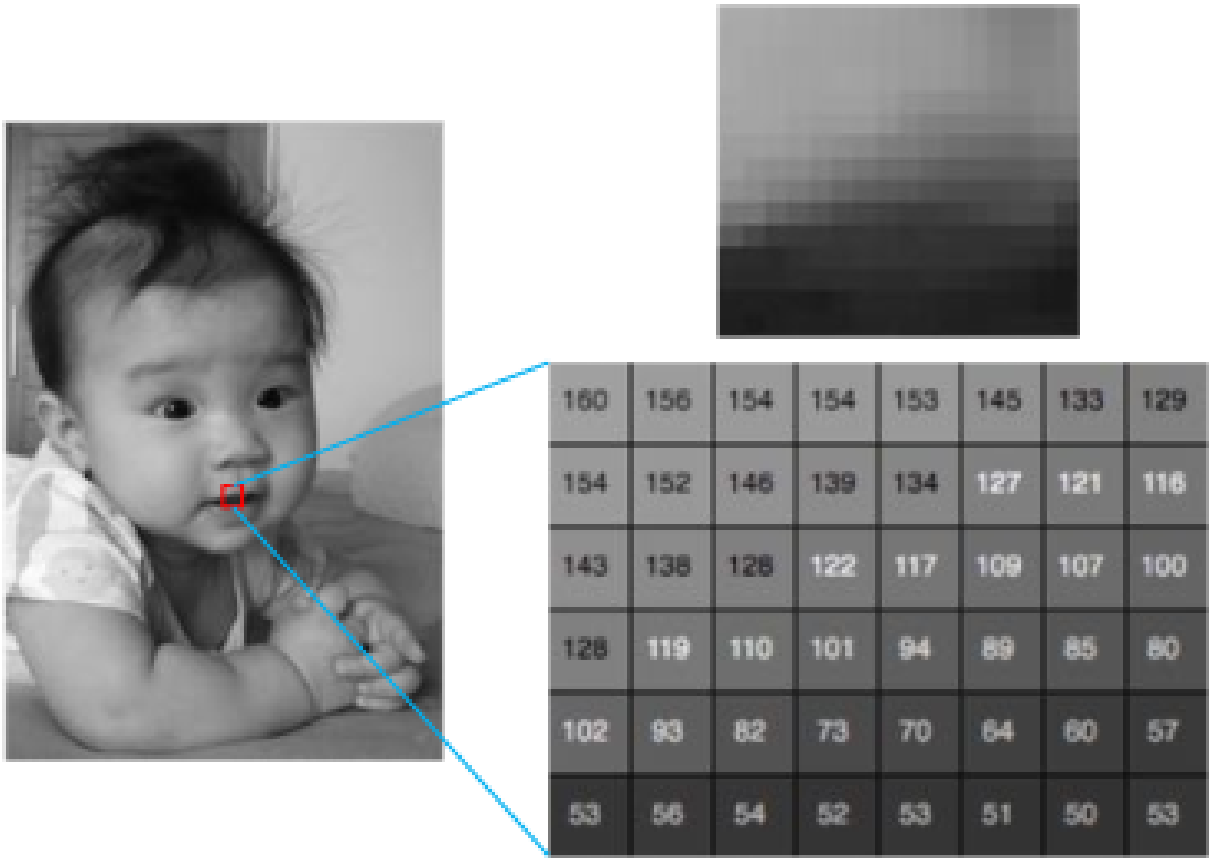
- Input: Image or video.
- Basic processing operations:
  - Reading and displaying images

  - Getting image dimensions
  - Adjusting brightness
  - Cropping images
  - Resizing images
  - Converting images to grayscale
  - Changing image color
  - Drawing on images

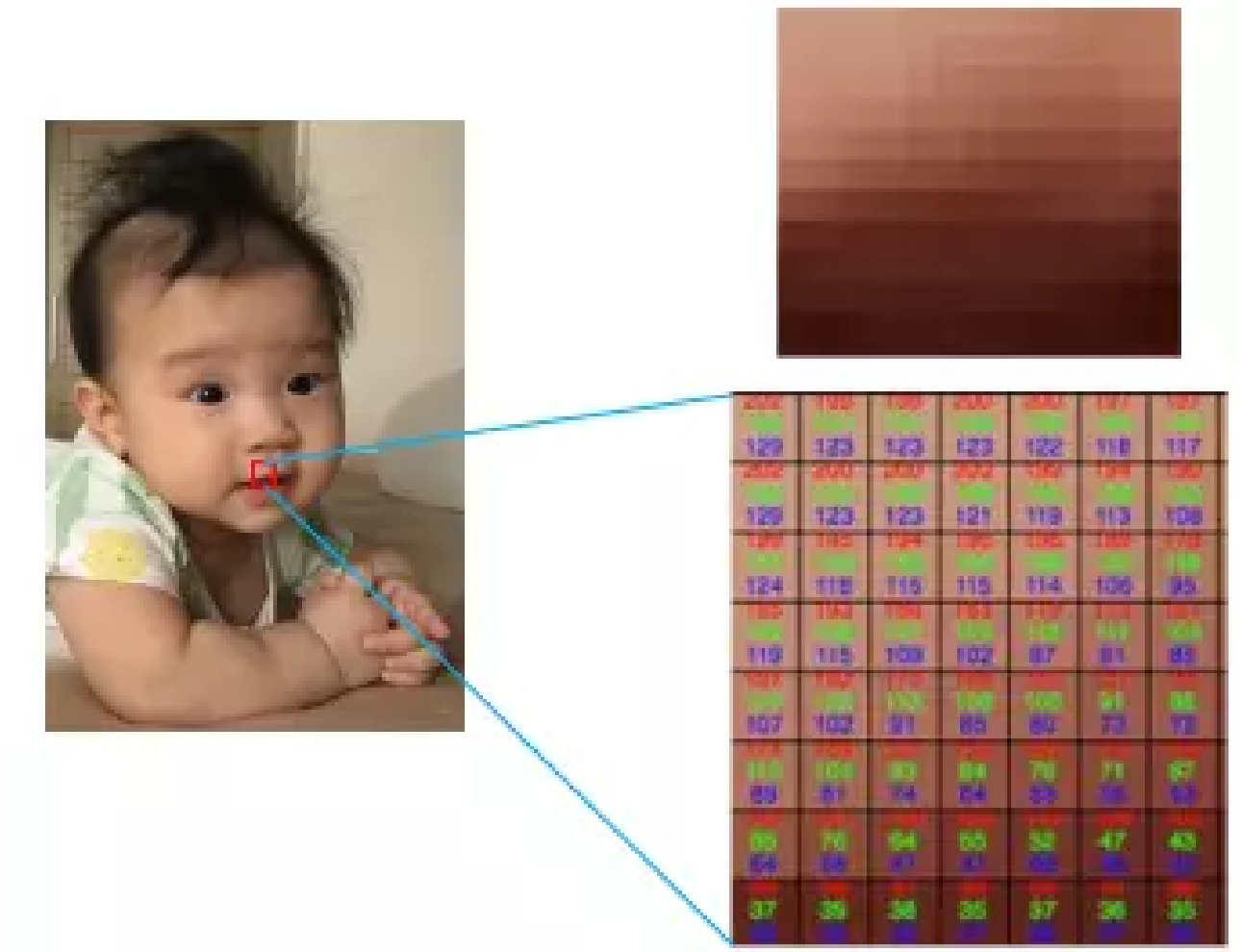
- Output:
  - Images after processing steps
  - Videos after processing steps: grayscaling, brightness adjustment, and drawing overlay

3.1.6 Theory

Read images and image properties
Images are composed of pixels. They have position (x,y) and grayscale l (x,y).



As for color images, each color point will have the brightness value of 3 colors red, green, blue (RGB)



To read images, we have function textbfcv2.imread(). The output will be a 3-dimentional matrix.
To check the shape and size of an image:

- The **img.shape** function returns the dimensions of the image with h, w, d being the height, width, and depth, respectively. For colored images, the depth is usually 3, while for grayscale images, it is 1.
- The **img.size** function returns the size of the image with the following formula: size = width x height x dimension.

To display image, we have 2 methods:

- Method 1: Using the built-in function in OpenCV The **cv2.imshow(window_name, image)** function displays an image in a GUI window.

  Parameters:

  - **window_name**: The name of the window where the image will be displayed. You can name it arbitrarily.
  - **image**: A numpy array representing the image you want to display. This image can be the result of reading an image using **cv2.imread()** or from other sources.

- Method 2: Using the matplotlib library The **plt.imshow** function will display the image in the BGR color format.

Adjust Image Brightness
The value of a pixel ranges from 0 to 255. When the value of x exceeds 255, the new value of x will be: x = x - 255. When the value of x is less than 0, the new value of x will be: x = 255 + x.

Example: For a pixel with a value of 275, its actual value will be x = 275 - 255 = 25.

To increase the brightness of an image, we need to create a matrix of the same size as the image, containing the values to add/subtract to adjust the brightness.

Then use the **cv2.add** function to increase, and the **cv2.subtract** function to decrease the brightness of the image.

Crop Image
Cropping an image is a common and basic technique aimed at extracting the necessary space in an image and removing unnecessary background. To crop an image, two parameters are needed: the coordinates of the top-left and bottom-right corners.

Resize Image
To resize an image in OpenCV, we use the **cv2.resize** function. This function requires specifying the height and width of the image after transformation.

However, the resized image usually maintains the same aspect ratio as the original. We should convert both dimensions according to the same ratio to preserve the original shape of the image.

Convert Image to Grayscale
There are two ways to convert an image from normal to grayscale:

- Method 1: Average (G) = (R + G + B) / 3
- Method 2: Use the OpenCV library (G) = 0.299R + 0.587G + 0.114B

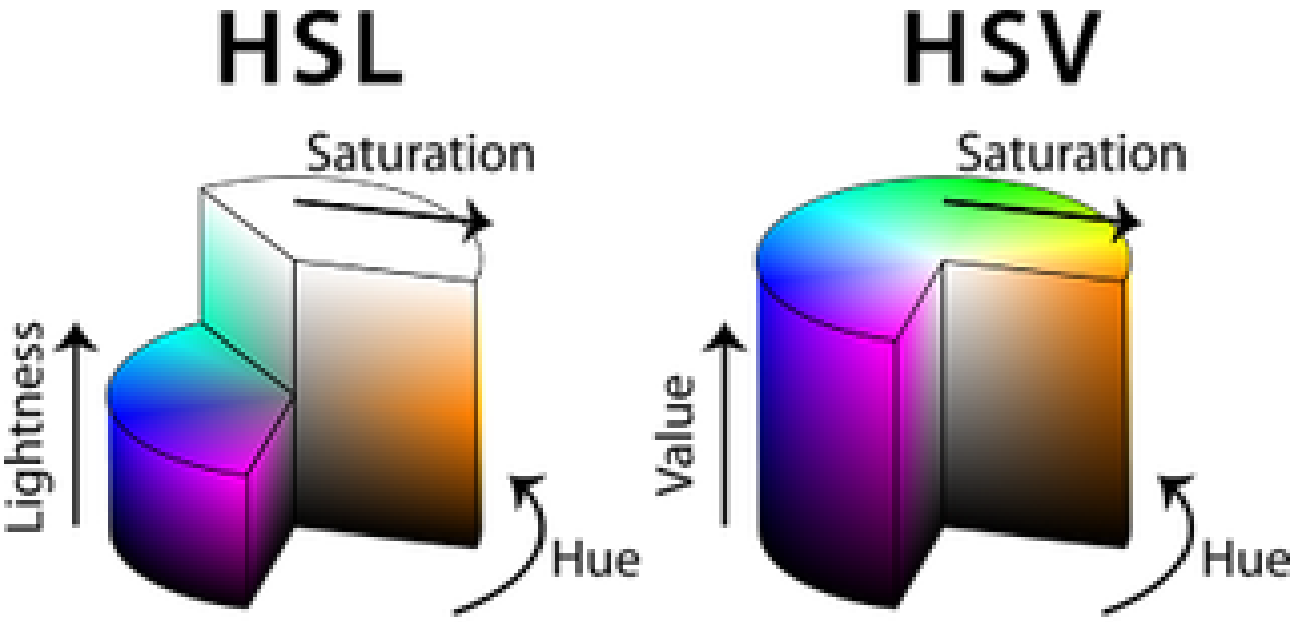
Experimental results show that when using the **cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)** function, the output image will be better balanced compared to using the average method.

Convert Image to Other Color Spaces (HLS and HSV)
HSV Color Space:

- The HSV color space is very convenient for color processing, as each color is assigned a Hue value ranging from [0; 360], while the Value and Saturation components indicate the brightness and saturation of the color, respectively. This color space is particularly useful for object tracking based on color.
- In some contexts, it may be referred to as HSB, where ”Value” is replaced with ”Brightness,” but the meaning remains the same.

HLS Color Space:

- The HSL color space stands for Hue, Saturation, and Luminance. It is similar to HSV and is also suitable for color filtering tasks.



The **cv2.cvtColor(img, color_space)** function is used to convert an image between different color spaces. This function requires two parameters:

- img: The variable containing the image to be processed.
- color_space: The code representing the target color space.

Drawing on Images

There are many functions available to draw on images. Here, we will introduce the **cv2.line** function, which is used to draw a straight line on an image. The function requires four parameters:

- The input image
- The starting point
- The ending point
- The color

- The thickness

Work with video
In essence, a video is a collection of continuous frames (images). Therefore, we simply process each frame in the video and combine them.
We will go through the following steps:

- Reading the video:
  - **cv2.VideoCapture()**: Opens the input video.
- Getting video information:
  - frame_width, frame_height, fps: Retrieve the frame dimensions and frames per second (fps) of the input video.
  - fourcc: Defines the codec format for the output video.
- Creating VideoWriter:
  - **cv2.VideoWriter()**: Creates an object to write the output video.
- Loop to process each frame:
  - **cv2.cvtColor()**: Converts the frame to grayscale.
  - **cv2.subtract()**: Decreases the brightness of the image.
  - **cv2.line()**: Draws a line on the image.
  - **out.write()**: Writes the processed frame to the output video.
- Releasing resources:
  - **cap.release()**, **out.release()**, **cv2.destroyAllWindows()**: Closes the resources.

3.1.7 Laboratory

Open Visual Studio Code, open Lab1_1 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab1_1_solve.

3.2 Get familiar with the Canny edge recognition algorithm

3.2.1 Overview

You might not know this, but when it comes to image recognition, the human eye takes only a few milliseconds to process and identify the content of an image. This remarkable ability can be achieved whether the data is a drawing or a photograph. This is also the goal of computer vision. One of the current ideas is to develop an algorithm that can outline the edges of any object in an image using the Canny edge detection algorithm. Its effectiveness has been proven by many papers in the past, achieving an accuracy of over 90% [1].

First, it's essential to understand what the Canny edge detection algorithm is:

The Canny edge detector is an algorithm that involves multiple stages to detect a wide range of edges in an image. It was developed by John F. Canny in 1986. Canny also proposed a computational theory of edge detection that explains why this technique works.

3.2.2 Learning goals

Upon completing this exercise, learners will gain knowledge of:

- A deeper understanding of the Canny algorithm
- How to program the processing steps in the Canny algorithm for edge detection

3.2.3 Related knowledge

- Python
- Image processing
- OpenCV

3.2.4 Required background

To complete this exercise, you will need the following knowledge:

- Basic programming skills with Python

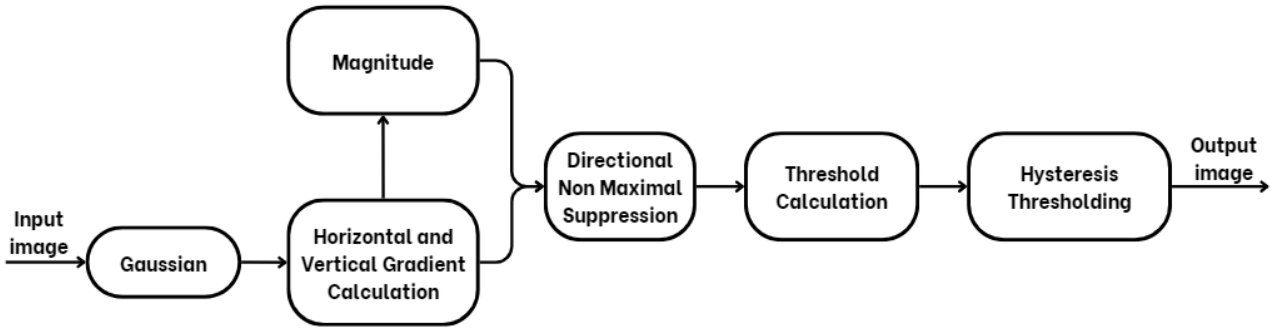
3.2.5 Problem

Objective: Identify edges in the input image/video using the Canny algorithm.
Requirements

- Input: Image or video.
- Output: Image or video processed using Canny edge detection.

3.2.6 Theory

In terms of theory, the Canny algorithm consists of the steps.



Gaussian Blur
Eliminating noise is crucial in the edge detection process. A useful method is applying a Gaussian filter. This is done by convolving the image with a Gaussian kernel (with sizes like 3×3, 5×5, 7×7, etc.). The kernel size depends on the desired blurring effect. Essentially, the smaller the kernel, the less blurring effect it will produce.

$$H_{ij} = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{(i-(k+1))^2 + (j-(k+1))^2}{2\sigma^2}\right); 1 \leq i, j \leq (2k+1)$$

Figure 3.1: Equation for a Gaussian filter kernel of size (2k + 1) × (2k + 1)

The **cv2.GaussianBlur** function applies a Gaussian filter to blur the image, which helps reduce noise and smooth the edges.

The function includes three parameters:

- **image**: The input image that needs to be blurred.
- **kernel_size**: The size of the kernel used for blurring the image. The kernel is a square matrix with the side length being **kernel_size**. This size must be an odd number and greater than 1, such as (3, 3), (5, 5), etc.
- **sigma**: The standard deviation of the Gaussian, which determines the level of blurring in the image. If **sigma** is 0, it will be calculated automatically from the kernel size.

Gradient Calculation
Calculating the gradient of an image helps to identify the change in intensity at each point in the image to determine edges. To do this most easily, we use the Sobel filter, which highlights intensity changes across both the horizontal (x) and vertical (y) axes.

After the image has been smoothed, we calculate the derivatives in the x-direction (**Ix**) and y-direction (**Iy**). This will allow us to determine the magnitude and direction of the gradient.

$$K_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, K_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}.$$

(a)

$$|G| = \sqrt{I_x^2 + I_y^2},$$

$$\theta(x,y) = arctan\left(\frac{I_y}{I_x}\right)$$

(b)

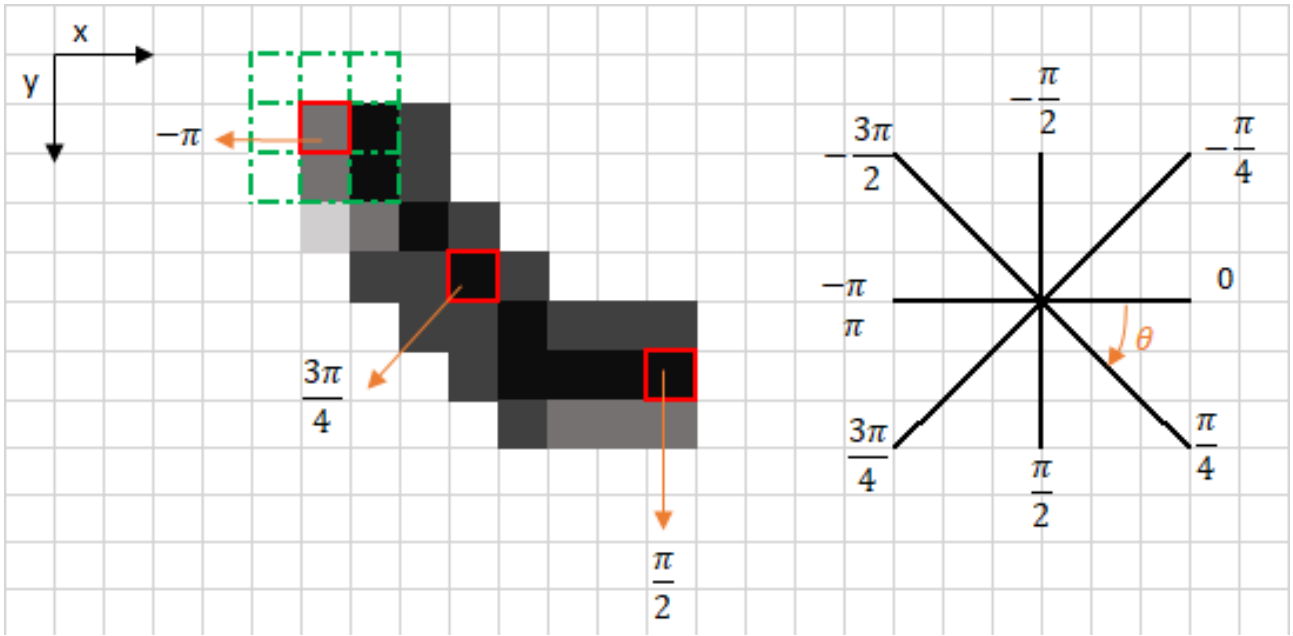
To calculate the gradient of an image and determine the intensity change at each point, this process involves two main steps:

- Calculating the first derivative of the image
  - The **cv2.Sobel** function calculates the first derivative of the image using the Sobel filter, enabling edge detection in the image.
  - This function has five parameters
    - **image**: The input image.
    - The data type of the output image, typically **CV_64F** is used for higher precision because the gradient values can be negative.
    - These parameters define the direction of the derivative. **1, 0** calculates the derivative in the x-direction, while **0, 1** calculates the derivative in the y-direction.
    - **ksize**: The size of the Sobel kernel. This is typically an odd number, such as 3, 5, 7, etc.
- Calculating the magnitude and direction of the gradient
  - This step uses the **np.hypot** and **np.arctan2** functions, with the inputs being the results from the Sobel filters. This allows you to calculate the magnitude and direction of the gradient, providing information on how pixel values are changing in the image.

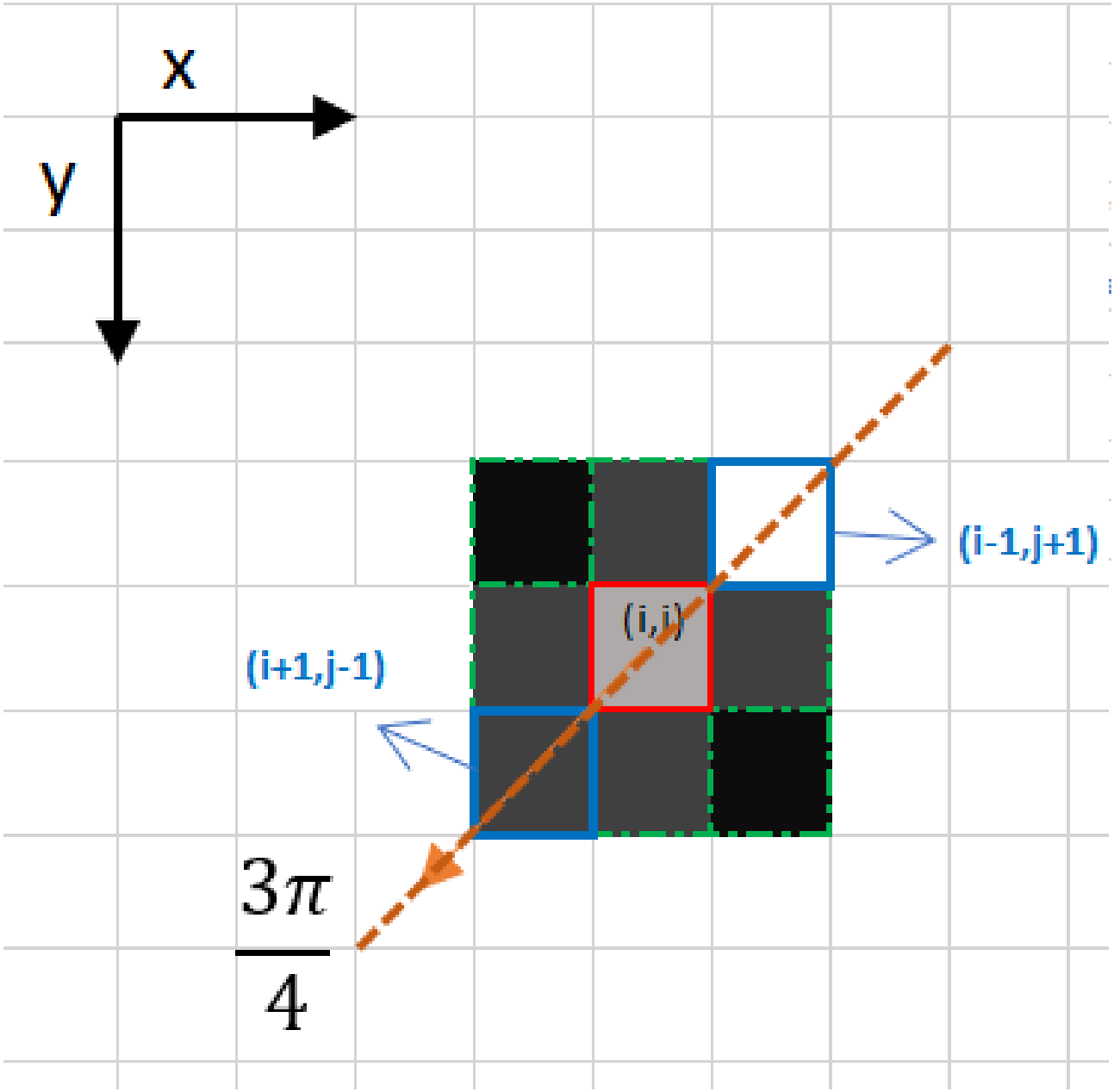
This approach is crucial in edge detection, as it reveals the boundaries within an image based on intensity variations.

Non-maximum Suppression
To achieve the best results, the final image should retain only the local maxima points in the gradient magnitude.

Principle: The algorithm scans all the points in the gradient magnitude matrix and identifies the pixels with the highest values in the direction of the edges.



The red box in the top left corner represents a pixel intensity in the Gradient Magnitude matrix currently being processed. The corresponding gradient direction is represented by an orange arrow with an angle of -π radians (+/- 180 degrees).



The gradient direction is indicated by the dotted orange line (horizontal from left to right). The purpose of the algorithm is to check if the pixels along the same direction have a higher or lower intensity than the pixel currently being processed. In the example above, the pixel (i, j) is being processed, and the pixels along the same direction are marked in blue as (i, j-1) and (i, j+1). If either of these pixels has a higher intensity than the one being processed, then only the pixel with the higher intensity is retained. The pixel (i, j-1) appears brighter since it is white (value 255). Therefore, the intensity value of the current pixel (i, j) is set to 0. If neither of the pixels in the edge direction has a higher intensity value, then the value of the current pixel remains unchanged.

To summarize, each pixel is evaluated based on two main criteria (edge direction in radians and pixel intensity ranging from 0–255). Based on these inputs, the steps of non-maximum suppression are:

- Create a matrix initialized with 0 having the same size as the original gradient intensity matrix.
- Determine the gradient direction based on the angle value from the angle matrix.
- Check if the pixel in the same direction has a higher intensity than the currently processed pixel.
- Return the image processed by the non-maximum suppression algorithm.

Double Threshold
This step categorizes pixels into three types: strong, weak, and non-edges.

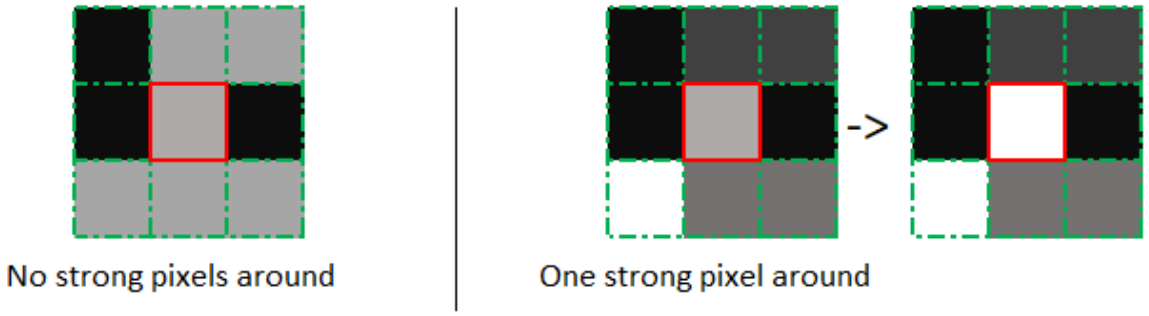
Apply two thresholds: high threshold and low threshold.

- Pixels with values greater than the high threshold are considered strong edges.
- Pixels with values between the high and low thresholds are considered weak edges.
- Pixels with values below the low threshold are discarded.

Choosing the right threshold directly affects the output result as it is the basis for determining which pixels are edges. Experiment with different values to find the parameter pair that best suits the input image.

Edge Tracking by Hysteresis The goal of this step is to identify weak edges that are connected to strong edges.

- Weak edges connected to strong edges are considered true edges.
- Use depth-first search or breadth-first search to connect edges.



Using the Built-in Function in the OpenCV Library In the OpenCV library, there is a built-in function **cv2.Canny()**. To use this function, you need to pass three variables to it:

- **img**: The input image (grayscale image).
- **threshold1**: The lower threshold for the hysteresis procedure.
- **threshold2**: The upper threshold for the hysteresis procedure.

3.2.7 Laboratory

Open Visual Studio Code, open Lab1_2 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab1_2_solve.

3.3 Apply the Canny algorithm to lane-following vehicles

3.3.1 Overview

Self-driving cars are one of the most groundbreaking innovations in the field of AI. One of the many steps involved in the self-driving car training process is lane detection, which is a preliminary step. In this exercise, we will learn how to apply the Canny algorithm to lane recognition and operate the car in lane.

3.3.2 Learning goals

After completing this exercise, students will learn the following knowledge:

- Real-time image processing techniques.
- Apply Canny to detect lanes.

3.3.3 Related knowledge

- Python
- Image processing
- OpenCV

3.3.4 Required background

To perform the exercise, you will need the following knowledge:

- Basic programming skills with Python
- Can use basic functions of OpenCV
- Edge detection Canny algorithm.

3.3.5 Problem

Target: The vehicle model can drive in the lane at a slow speed.
Request:

- Processing lane detection using the Canny algorithm.

Processing steps:

- Get input image from camera.
- Process the input image by calibration and converting image perspective.
- Identify road markings with Canny.
- Mix the output image with the input image for display.
- Improve processing speed by multithreading.

3.3.6 Theory

Get input image

To retrieve image data from the camera, we need to obtain the IP link that we have acquired from the setup. Initially, the function attempts to open the URL using **urlopen** with a timeout period to get the image data. The obtained data is then converted into a NumPy array, and subsequently decoded into an image format using OpenCV's **imdecode** function. After decoding, the image is resized to 1280x720 pixels. The processed image is then added to the queue.

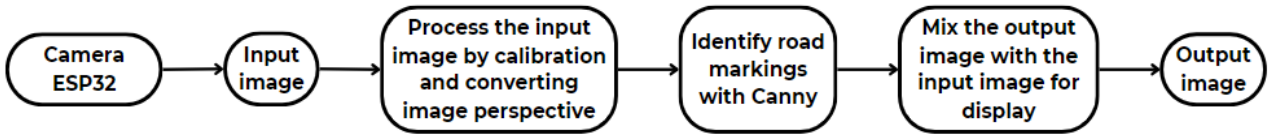


Figure 3.3: Image processing steps

Calibration

Optical distortion is a common phenomenon in photography. Radial distortion causes straight lines to appear curved. Radial distortion becomes more pronounced as points are farther from the center of the image.

For example, an image below shows two edges of a chessboard (marked by red lines).

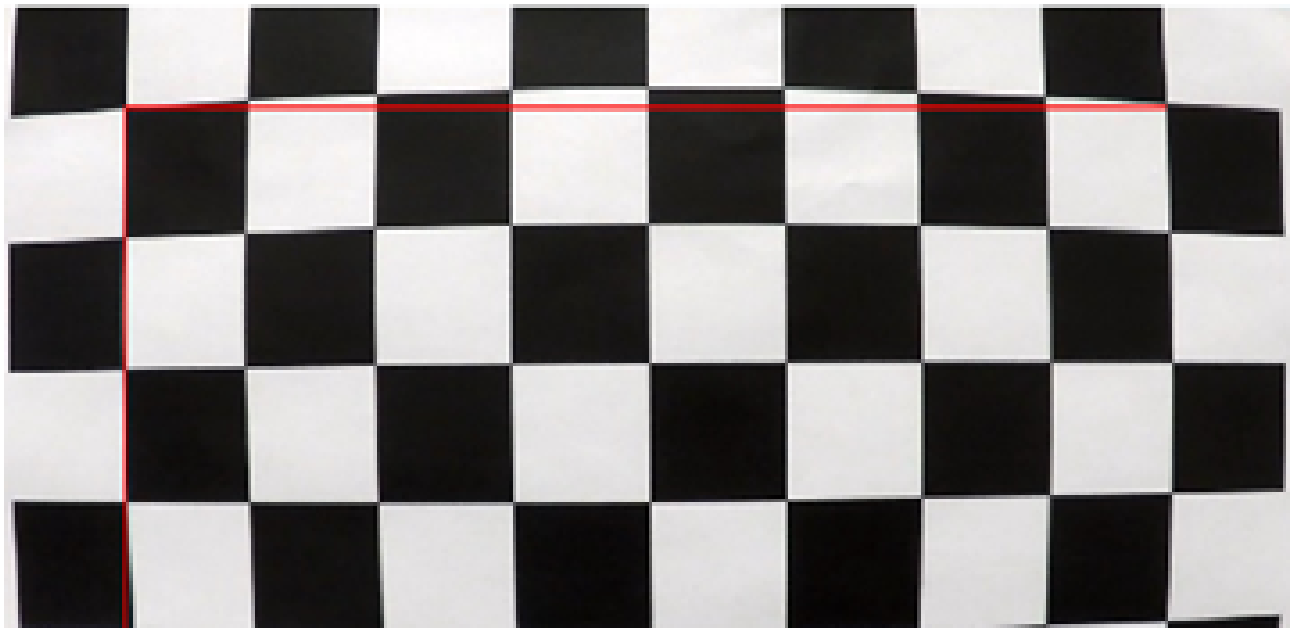


Figure 3.4: Image before applying Calibration technique

However, you can see that the edges of the chessboard are not straight lines and do not align with the red lines.

To solve this problem, we need Camera Calibration.

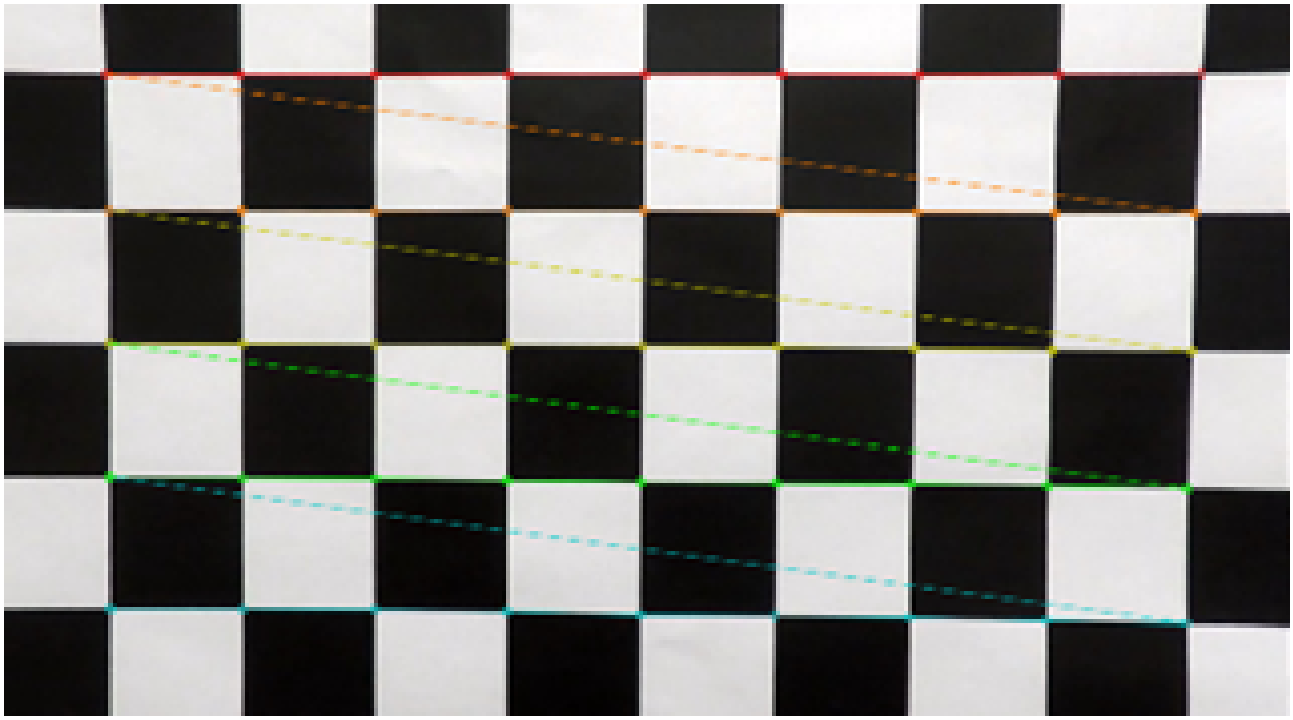


Figure 3.5: Image after applying Calibration technique

The input data for camera calibration is a set of 3D real-world points and their corresponding 2D coordinates in the image.

Calibration in image processing is the process of adjusting the camera's sensors and optical systems to ensure that the captured image accurately reflects reality.

The **CameraCalibration** function will be responsible for correcting Lens Distortion by using images of a chessboard. The chessboard is used because its corners are easily recognizable by the algorithm.

Perspective transform

In autonomous vehicle control, lane detection is extremely important.
To detect the road more easily and efficiently, we can use the perspective transformation method to switch between the front view and the bird's-eye view.

Figure 3.6: Image before applying the Perspective transform technique

The forward function of the class **PerspectiveTransformation** will convert the source points (src_points) to the destination points (dst_points) to change the perspective of the image. The points are the coordinates of the pixels on the input image.

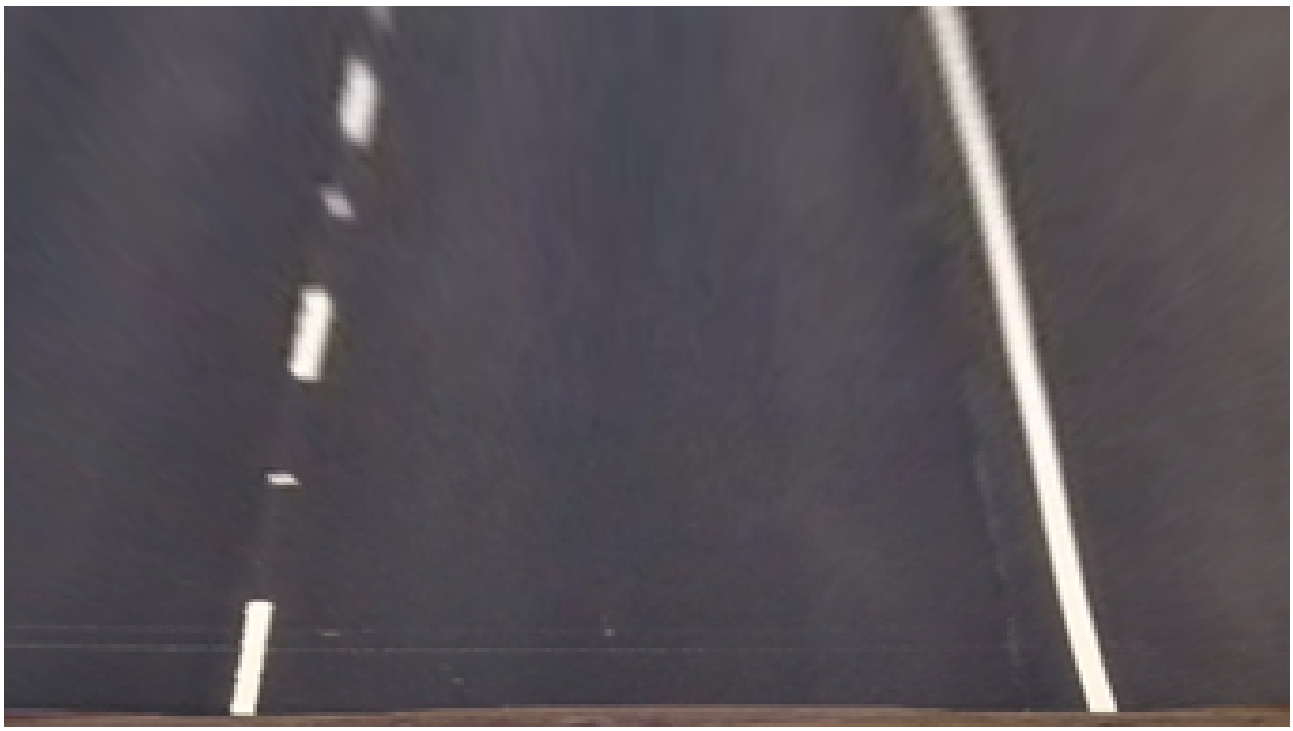


Figure 3.7: Image after applying the Perspective transform technique

The top-left and bottom-left points of the src_points will align with the lane marking when the right tire of the ACE-kit vehicle touches the right lane marking, and similarly for the left side.

Due to hardware limitations, we should use the bottom 1/3 of the image (y = 450) to ensure high stability in lane detection (in the later stages).

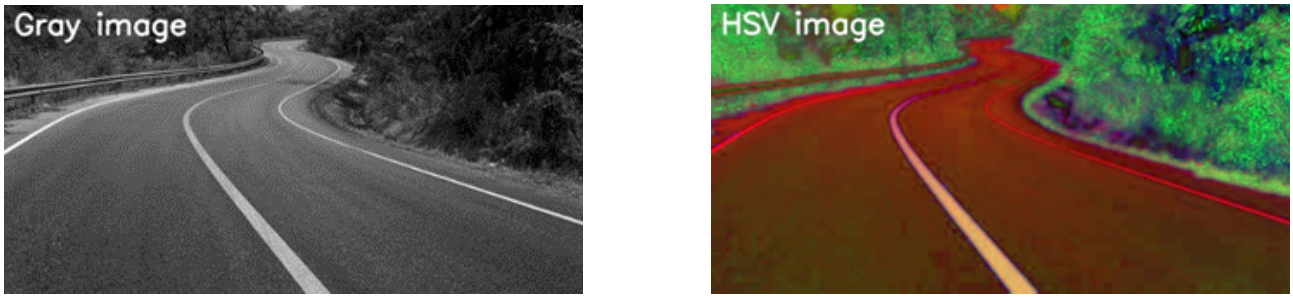
Table 3.1: Performance of transformation technique [2]

Number images	None-transform	Transform
300	89%	92.5%
500	89%	94%
800	90%	94%
1000	90%	94%
1500	92%	95%

Detect lane lines by Canny algorithm

The theory behind the Canny algorithm has been detailed in our Lab 2. Therefore, in Lab 3, we will discuss the differences we have implemented compared to the original Canny algorithm.

We use a color space HSV, instead of grayscale. The reason for this is that in practice, there are lane markings that are yellow (separating two opposite lanes).

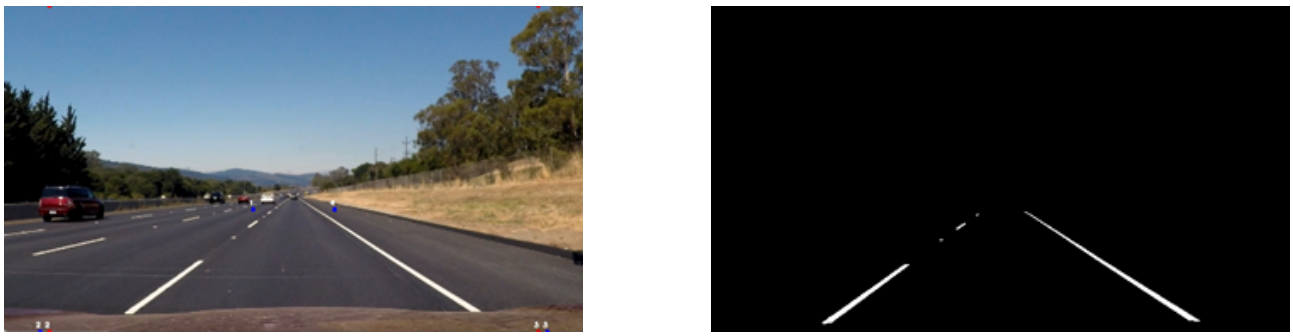


(a) (b)

Figure 3.8: Gray and HSV images

Output image

The input and output image will look like this.



(a) (b)

Figure 3.9: Input and output images

To provide a more intuitive display for the user, we should combine the input image and the output image.

Multiprocessing

Multithreading is a programming technique that allows a program to execute multiple threads concurrently. A thread is the basic unit of CPU utilization and is scheduled for execution. Threads within a process share the same address space and system resources but have separate execution contexts.

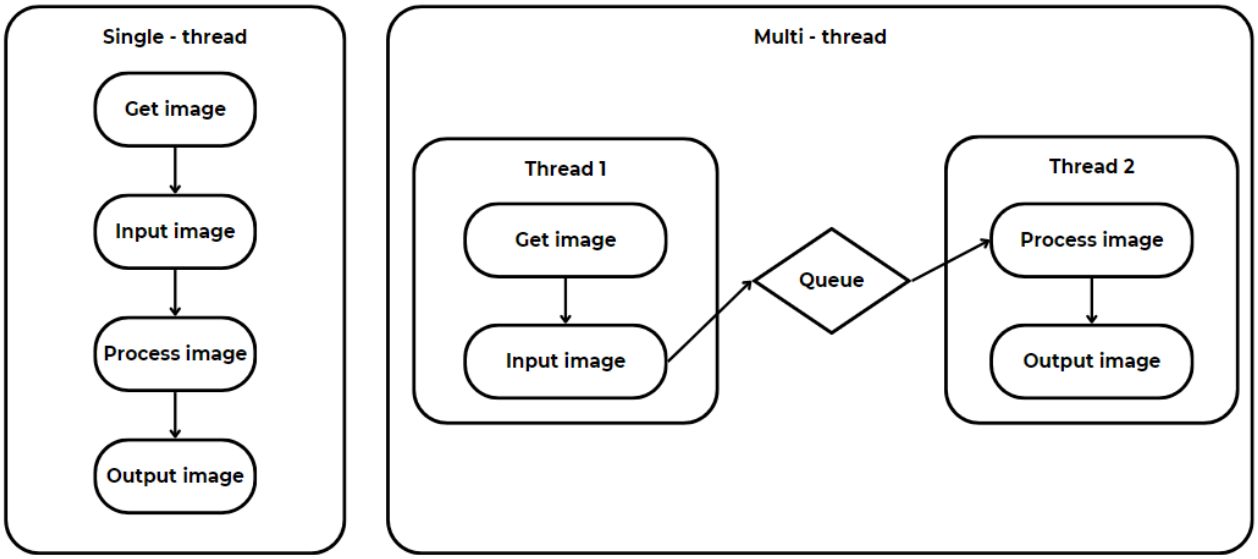


Figure 3.10: Single-thread vs multi-thread

Purpose of Multithreading

- **Increase Performance**: Multithreading can enhance the performance and speed of a program by fully utilizing the CPU's power, especially on multi-core systems. Each core can execute a separate thread, reducing waiting time and increasing overall processing speed.
- **Improve Responsiveness**: For applications requiring quick user responses, such as graphical user interfaces, multithreading can make the application more responsive. For example, a user interface application can maintain smooth interaction with the user even while performing complex background tasks.
- **Efficient Resource Utilization**: Multithreading helps utilize system resources more efficiently by allowing different threads to perform different tasks simultaneously. This can reduce waiting times for resources like CPU, memory, and I/O.
- **Concurrent Task Execution**: Independent tasks can be executed concurrently in separate threads. This is particularly useful in applications that need to handle multiple tasks simultaneously, such as data processing, file downloading, or performing complex calculations.

Through practical experiments, we observed a significant performance improvement when applying the multithreading approach.

Table 3.2: Performance comparison table between single-thread and multi-thread

	Lane Detection	
	FPS	ACC
Single-thread	1-3	42%
Multi-thread	12-15	92%

3.3.7 Laboratory

Open Visual Studio Code, open Lab1_3 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab1_3_solve.

3.4 Expected performance Results

In terms of performance, the vehicle will be able to maintain lane-keeping with an FPS of around 12-15 and an accuracy of over 90%.

Chapter II: Object Detection

4.1 Prepare data

Welcome to Lab 2.1. In this lab, we will learn how to process raw data to prepare it for training and evaluating a model.

4.1.1 Overview

Building a dataset for an AI model is extremely important. A good dataset will help the model accurately identify and perform effectively. Because we want the vehicle to best recognize traffic signals in the map, we prioritize collecting data ourselves rather than using publicly available sources. After obtaining the original images, we will sequentially process them to clean and enrich the data. The final step is to label them and divide them into training, validation, and test files according to a customized ratio.

4.1.2 Learning goals

After completing this exercise, learners will have acquired the following knowledge:

- Image preprocessing
- Some data augmentation methods

4.1.3 Related knowledge

- Python
- Image processing
- OpenCV

4.1.4 Required background

To complete this exercise, you will need the following knowledge:

- Basic programming skills in Python
- Basic understanding of functions in the OpenCV library

4.1.5 Problem

Objective: Data processing and augmentation
Requirements

- Input: Raw data
- Output: Data processed and ready for model training.

4.1.6 Theory

Adjusting Image Size
When developing an object detection system for autonomous vehicles, it is crucial to ensure that the images used for training the model are consistent with the images from the input camera.

- **Resolution and Aspect Ratio**: The input camera may have a specific resolution and aspect ratio. If the training images do not match the resolution and aspect ratio, the model may struggle to detect small and important details.
- **Data Similarity**: Machine learning models perform best when the training data resembles the real-world data they will process. If the training images are not similar to the images from the input camera, the model may not accurately detect objects.
- **Real-World Performance**: The ultimate goal is to ensure that the autonomous vehicle operates safely and effectively. By training the model with images that match the input camera, you can improve the accuracy and performance of the object detection system, thereby enhancing the safety and effectiveness of the autonomous vehicle.

Therefore, to achieve the best results, training images should be collected and processed similarly to how the input camera will capture them in real-world scenarios.

When collecting images, they are often large in size and not proportionate to the camera's aspect ratio.

Thus, a step is needed to resize the images to match the aspect ratio of the ESP32 camera's input image at 320x240.

Data Augmentation
Data augmentation is the process of creating variations of existing data by applying transformations such as rotation, translation, brightness adjustment, etc., ensuring the data covers a wide range of situations. This is a crucial factor in training machine learning and artificial intelligence models.

This process contributes to improving the model's generalization ability, reducing bias, improving accuracy and performance, as well as detecting and handling outliers.

Data augmentation is a critical and necessary step in developing effective and fair machine learning models. It ensures that the model can perform well in real-world scenarios under different conditions.

Working with Directories
In practice, we often need to process a large number of images simultaneously. Therefore, applying a processing function to each file as above is inefficient.

In this exercise, we will introduce some common functions in the **os** module for manipulating the file and directory system:

- **os.path.exists(path)**: Checks whether a path exists.
- **os.makedirs(path)**: Creates a new directory along with the necessary subdirectories. If the directory already exists, it will not raise an error.
- **os.listdir(path)**: Lists all files and subdirectories in the specified directory.
- **os.path.join(*paths)**: Combines multiple path components into a single, well-formed path, helping to avoid errors when manually concatenating paths.

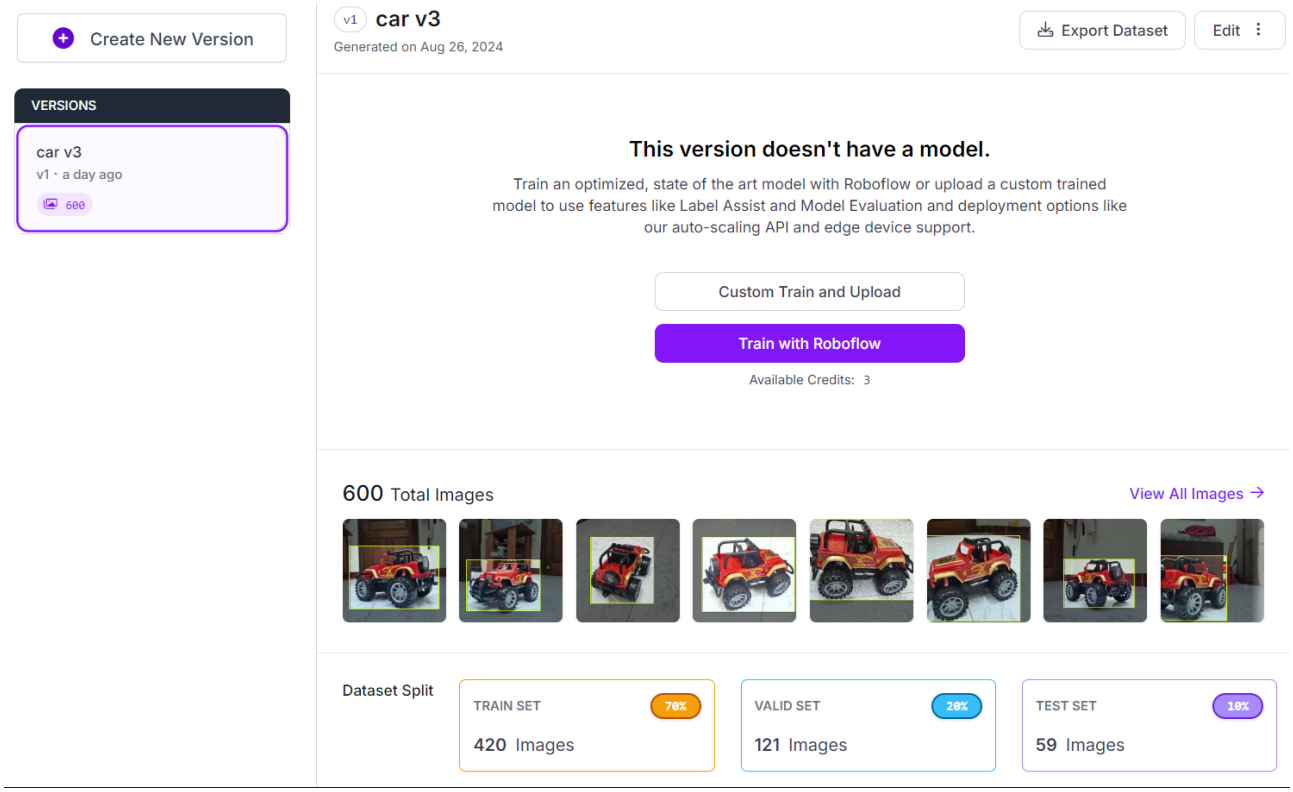
Data Labeling
Data labeling is the process of assigning labels or additional information to raw data, making it usable for model training. These labels can be identifiers, classifications, descriptions, or any necessary information that a machine learning model needs to understand and learn from the data.

Data labeling is an indispensable step to ensure that machine learning and deep learning models work well and provide reliable results. It helps increase accuracy, detect errors, improve training efficiency, evaluate, and test model accuracy.

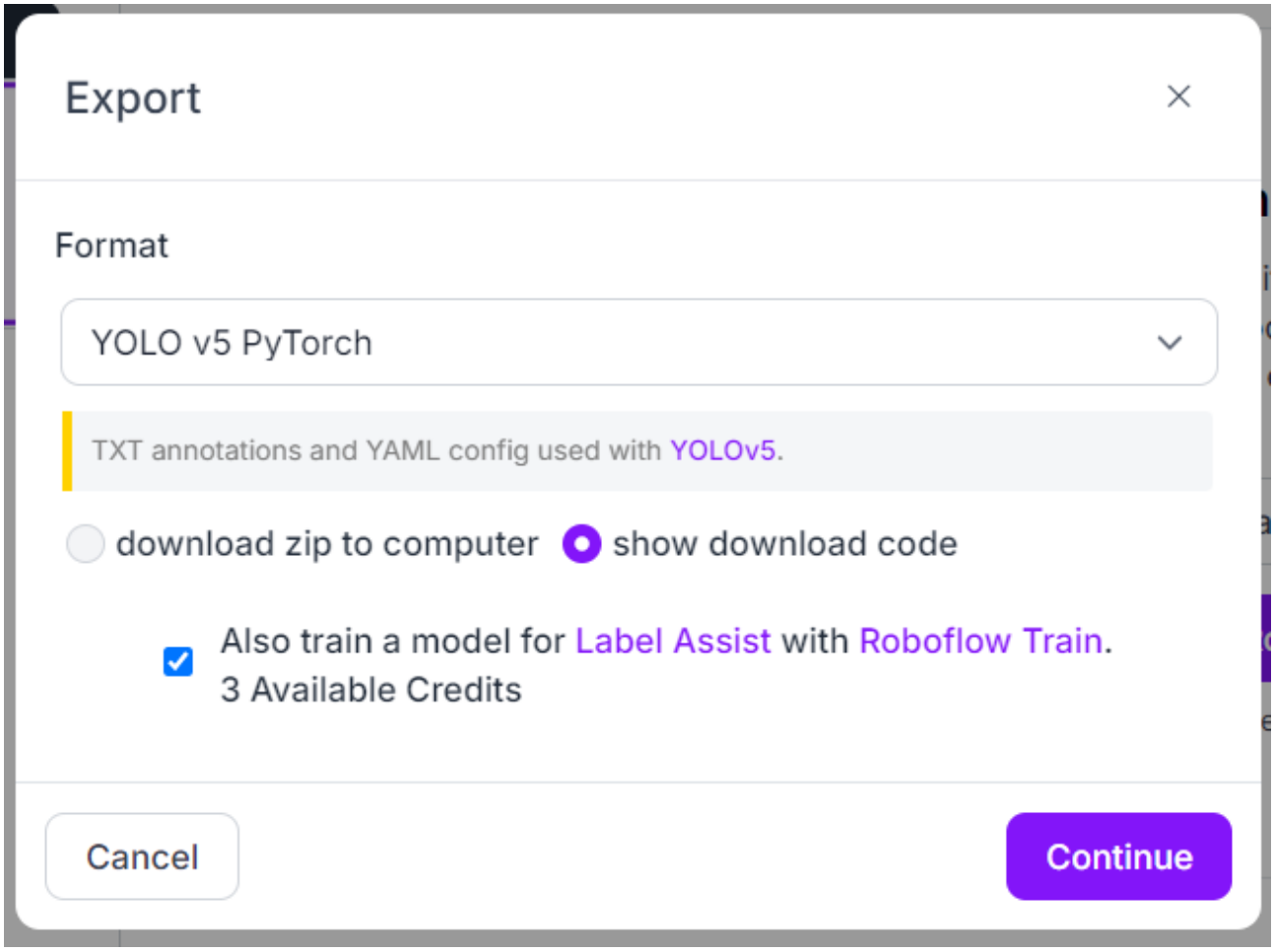
Building the Dataset
The final result of data labeling will include images and an accompanying text file containing information about the class and object location in the image. After that, we will need to divide the data into three directories: train, valid, and test (usually at a 7:2:1 ratio).

To facilitate labeling and data division, we need a label tool. We recommend an online website called roboflow.
After experiencing it with a team of three, the team found this tool to be quite useful and convenient for the following reasons:

- **Suitable for teamwork:** It allows tasks to be divided among members for collaborative labeling.
- **Supports various dataset formats for different models:** It can split data into train, validation, and test files and customize them for each model.
- **Online storage:** It enables online data storage and download via a link, which is especially suitable since exercises 2 and 3 involve training the model on Colab.



In addition, it also has the option to download data sets according to YOLOv5 standard.



4.1.7 Laboratory

Open Visual Studio Code, open Lab2_1 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab2_1_solve.

4.2 Train model Yolo V5 basic

YOLO (You Only Look Once) is one of the leading architectures for object detection tasks in computer vision applications. YOLOv5, released in 2020 by Ultralytics, is a significant advancement in the YOLO model family. Despite not being an official successor to YOLOv4, YOLOv5 quickly gained popularity due to its substantial improvements in performance and ease of use.

Summary

YOLOv5 has introduced major improvements in speed and performance for object detection tasks. With its flexibility and efficiency, it has become one of the primary tools used in autonomous driving systems, enabling vehicles to detect and respond to their surroundings swiftly and accurately. This model has been proven effective in real-world vehicle detection with an accuracy of over 80% [3].

4.2.1 Learning goals

After completing this exercise, students will gain knowledge in:

- Understanding the YOLOv5 model in depth
- Training the model with a custom dataset
- Evaluating the model's performance with key metrics

4.2.2 Related knowledge

- Python
- YOLO

4.2.3 Required background

To carry out this exercise, you will need the following knowledge:

- Basic Python programming skills

4.2.4 Problem

Objective: Train the YOLOv5 model with a custom dataset.
Requirements:

- Input: The dataset prepared from Exercise 4.
- Output: Trained model weights and their performance metrics.

4.2.5 Theory

YOLOv5

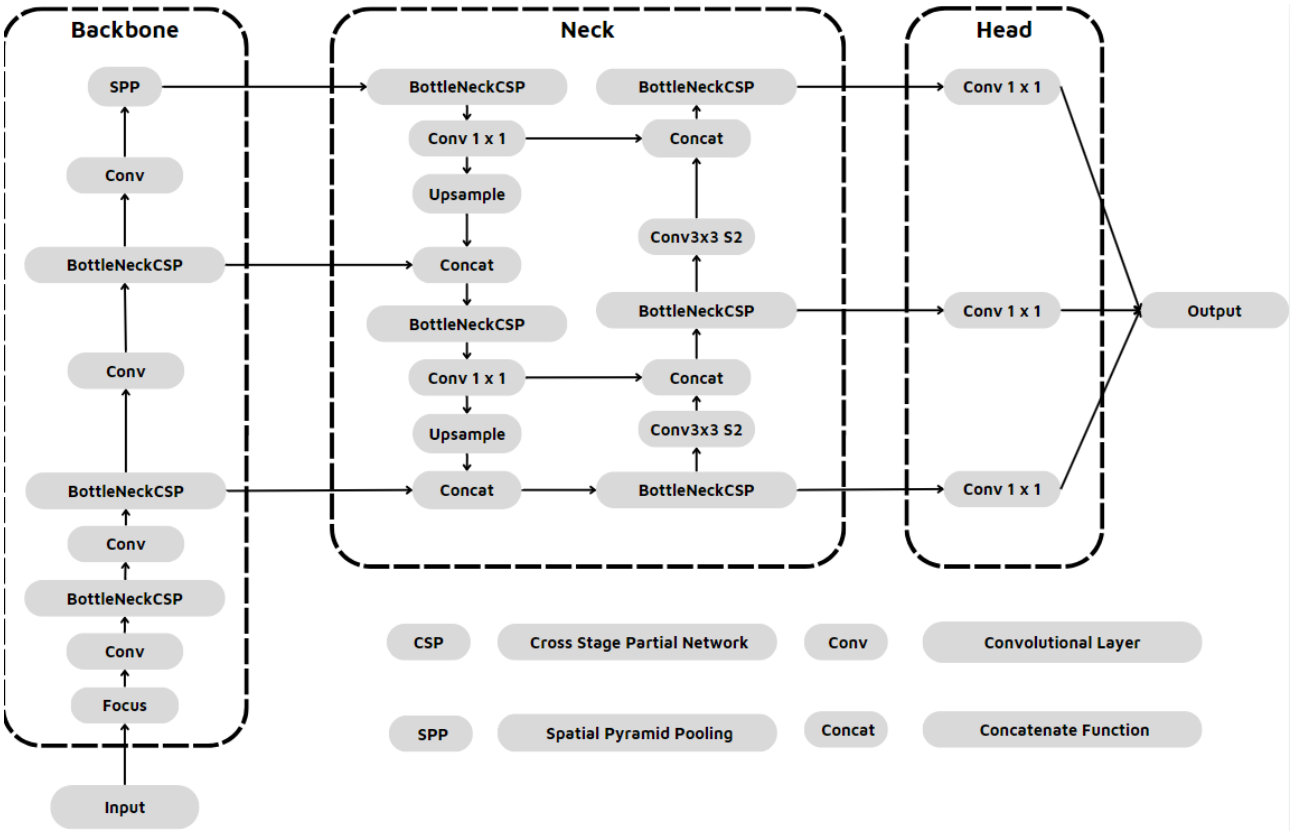


Figure 4.1: The basic architecture of YOLOv5

The network structure is illustrated in the YOLOv5 diagram (default setup), consisting of three main parts: (1) “Backbone,” (2) “Neck,” and (3) “Head.” After the initial preprocessing, the image undergoes feature extraction in the Backbone, followed by feature aggregation in the Neck. Then, the Head processes this information to produce detection results, including class, score, location, and size.
Layers in YOLOv5:

- **Backbone:** Extracts the main features from the input image.
  - Focus: Splits the input image and extracts initial features.
  - Conv (Convolutional Layer): Computes and detects features.

  - BottleneckCSP (Cross Stage Partial): Reduces computational complexity and improves feature learning.
  - SPP (Spatial Pyramid Pooling): Extracts information at different levels of resolution.

- **Neck:** Combines information from different layers for more accurate predictions.
  - BottleneckCSP: Similar to the Backbone, compresses information and enhances computational efficiency.
  - Upsample: Increases the image size to restore information from previous layers.
  - Concat (Concatenate): Combines information from different layers.
  - Conv 3x3 S2: Balances capturing local patterns while keeping the computational cost moderate.

- **Head**: Produces the final prediction output, including bounding boxes and prediction probabilities.
  - Conv 1x1: Final connection layer that narrows down and converts features into prediction outputs.

Dataset
To train our custom model, we need to assemble a dataset of representative images with bounding box annotations around the objects we want to detect. We also need our dataset in YOLOv5 format.

Model Training

A few basic parameters:

- **img:** input image size, a multiple of 32
- **batch:** batch size
- **epochs:** number of epochs
- **data:** path to the dataset
- **weights:** path to the weights to start training from. Here, we choose pre-trained weights on the COCO dataset.

Model Evaluation

The training loss and performance metrics are saved in Tensorboard as well as in the log file.

We will guide you on how to evaluate the YOLOv5 model's performance through three key metrics: **mAP_0.5**, **Precision**, **Confidence**, and **Loss**.

mAP_0.5 (Mean Average Precision at IoU 0.5)

- **Explanation**: mAP_0.5 measures the average precision of the model when detecting objects with an IoU (Intersection over Union) threshold of 0.5. IoU measures the overlap between the model's predicted box and the actual box.
- **Evaluation Criterion**: The mAP_0.5 value ranges from 0 to 1; the higher, the better.
- **Example**: mAP_0.5 = 0.85 means the model correctly predicts 85%

Precision

- **Explanation:** Precision is the ratio of true positive predictions to the total number of predictions (including false positives).
- **Evaluation Criterion:** Precision ranges from 0 to 1; the higher, the better. High precision means the model has fewer false positives.
- **Example:** If Precision = 0.9, then out of 100 predictions, the model correctly predicts objects 90 times.

Confidence

- **Explanation:** Confidence is the model's confidence level that an object exists in a detected region.
- **Evaluation Criterion:** The confidence threshold should be set appropriately. If set too high, the model will miss many objects (lower recall); if too low, the model will detect many false objects (lower precision).
- **Example:** With a confidence threshold of 0.8, the model only retains predictions where it is more than 80% sure of being correct.

Loss

- **Explanation:** Loss represents the model's error during training. The lower the loss, the better the model learns.
- **Evaluation Criterion:** The loss value should be as low as possible.
- **Example:** If the loss decreases from 1.2 to 0.3 after training, it indicates that the model is improving.

Summary:

- **mAP_0.5** evaluates the overall accuracy of the model.
- **Precision** indicates how many correct predictions the model makes out of all predictions.
- **Confidence** controls the quality of the predictions.
- **Loss** evaluates the model's learning process; the lower, the better.

Loading the Detection Model

The ‘torch.hub.load‘ function in PyTorch is a powerful tool for loading models or modules from repositories shared on GitHub. It allows you to easily use pre-trained models or other modules without having to build them from scratch.
Structure of the ‘torch.hub.load‘ function:

model = torch.hub.load(repo_or_dir, model, *args, **kwargs)

Parameters of the function:
1. **‘repo_or_dir‘**:

- This parameter specifies the GitHub repository containing the model you want to load. It is provided as a string, e.g., ‘’ultralytics/yolov5’‘.
- You can also specify a local path to the directory containing the model if the model has already been downloaded.

2. **‘model‘**:

- The name of the model or module you want to load from the repository. For example, ‘’custom’‘ for a custom model or ‘’yolov5s’‘ for a pre-trained YOLOv5s model.
- Depending on the repository, this name may represent a specific model or a set of predefined models.

3. **‘path‘**:

- The path to the ‘.pt‘ file containing the trained model weights. This parameter is usually used when you want to load a pre-trained model that you have built or customized.

4. **‘force_reload‘**:

- This parameter is a boolean value (‘True‘ or ‘False‘).
- When ‘force_reload=True‘, the function will reload the model from the repository or the specified path even if the model has already been loaded before. This is useful when you want to avoid conflicts or ensure that you are using the latest version of the model.

Testing with Images

Before working with videos, we will first apply the model to a few images to observe its effectiveness.

‘model(img_path)‘: This command uses the YOLOv5 model that was previously loaded to detect objects in the image.

- **‘img_path‘** is the path to the image you want to run object detection on.
- When you call the model with an image path like this, the model processes the image and returns the detection results as a ‘results‘ object.

‘test_detect.show()‘: This method displays the image with bounding boxes drawn around the detected objects.

- The detected objects are shown with information such as class labels and the confidence level of each object.

4.2.6 Laboratory

Open Visual Studio Code, open Lab2_2 and Lab2_2_1 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab1_2_solve and Lab2_2_1_solve.

4.3 Config model parameters for better performance

4.3.1 Overview

Most good results can be obtained without changing the model or training settings, as long as your dataset is large enough and clearly labeled. If you don’t get good results initially, you can take some steps to improve, but we always recommend training first with all default settings before considering any changes. This helps us establish the initial performance and make more accurate judgments.

4.3.2 Learning goals

After completing this exercise, students will learn:

- Methods to improve the performance of the model
- The impact of important parameters on the model

4.3.3 Related knowledge

- Python
- YOLO

4.3.4 Required background

To carry out this exercise, you will need the following knowledge:

- Basic Python programming skills
- Basic theory about YOLO

4.3.5 Problem

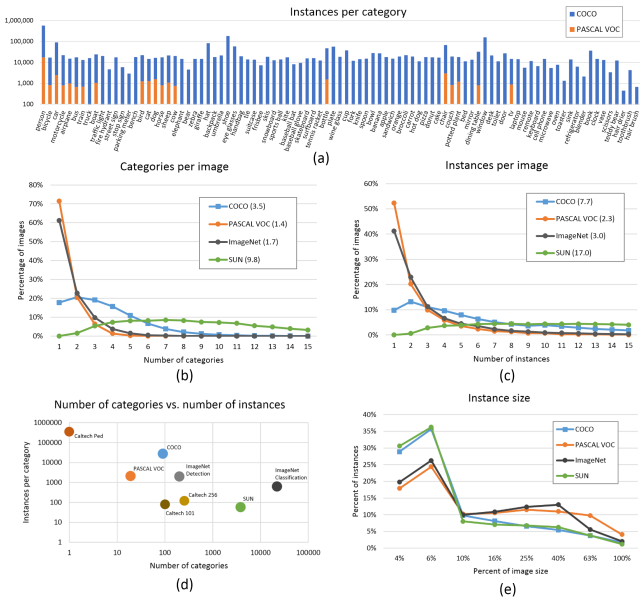
Goal: Improve the performance of the YOLOv5 model

4.3.6 Theory

Dataset

1. **Number of images per class**: There should be at least **1500 images** for each object class.
2. **Number of objects per class**: Each object class should have **at least 10,000 objects** (labeled objects).
3. **Diversity of images**: The dataset should reflect the real-world environment in which the model will be deployed. For real-world use cases, we recommend collecting images at different times of the day, in different seasons, weather conditions, lighting conditions, viewpoints, and from various sources (taken from the internet, on-site collection, using different types of cameras, etc.).
4. **Label consistency**: All objects of all classes in every image must be fully labeled. Incomplete labeling will not be effective.
5. **Label accuracy**: Labels must precisely enclose the object. There should be no gaps between the object and its bounding box. All objects must be fully labeled.
6. **Label verification**: When starting the training process, check collages such as 'train_batch.jpg' to verify that your labels appear correctly.
7. **Background images**: Background images are those that do not contain any objects and are added to the dataset to reduce false positives (FP). We recommend approximately **0-10%** background images in the dataset to help reduce FP (the COCO dataset has 1000 background images, accounting for 1% of the total images). Background images do not need to be labeled.

Compare the performance of the datasets:

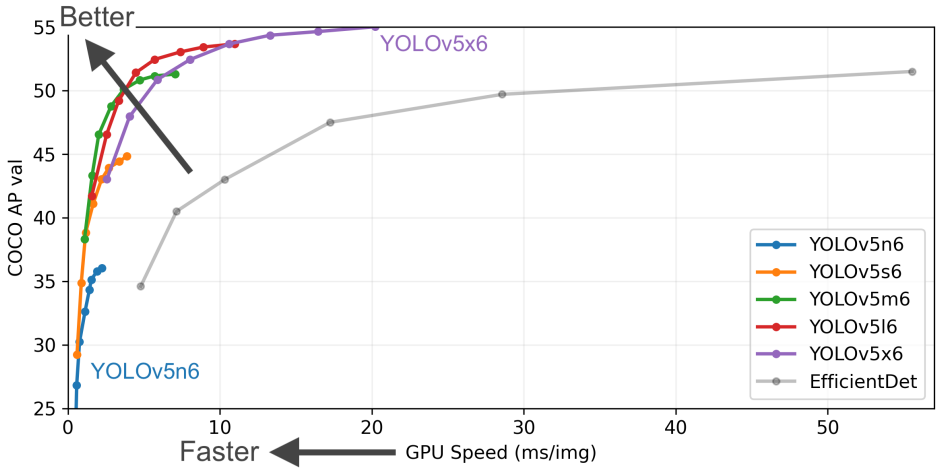


Model Selection

Larger models like YOLOv5x and YOLOv5x6 will produce better results in most cases, but they have more parameters, require more CUDA memory to train, and run slower. For mobile deployment, we recommend YOLOv5s/m, and for cloud deployment, we recommend

YOLOv5l/x.

The performance and speed of the models are described in the following image:



Parameter Tuning

These parameters play an important role in adjusting and optimizing the effectiveness of the YOLOv5 model:

1. **img**:
   - Input image size (e.g., 640, 416, ...). This size affects the resolution of the image and, therefore, the model's ability to detect details. Larger images can help the model detect more details but also require more computational resources and may slow down processing speed.
2. **rect**:
   - When enabled (True), input images are cropped and resized according to the original aspect ratio (rectangular training). This can help the model detect objects that are not square-shaped better.
3. **batch**:
   - Batch size (number of images processed simultaneously in one run) affects the training of the model. Larger batches can help stabilize the training process but require more GPU memory. Smaller batches may allow the model to learn more details but are more susceptible to noise.
4. **epochs**:
   - The number of iterations to train the entire dataset. More epochs help the model learn better, but too many epochs can lead to overfitting, where the model learns too well on the training data and does not generalize well to new data.
   - Start with 300 epochs. If this causes the model to overfit, you can reduce the number of epochs. If overfitting does not occur after 300 epochs, train longer, i.e., 600, 1200 epochs, etc.

5. **weights**:

   - Initial model weights. Pre-trained weights on other datasets can be used, or the model can be trained from scratch (no initial weights). Using pre-trained weights can help the model learn faster and achieve better performance.

6. **cfg**:

   - Configuration file specifying the model architecture. yolov5s is the smallest and fastest version, suitable for limited computational environments or when high speed is needed. However, it may not perform as well as larger versions like yolov5m, yolov5l, or yolov5x.

7. **patience**:

   - The maximum number of epochs the model can continue to train without improvement (early stopping). This parameter helps avoid overtraining, saving time and resources.

8. **hyp**:

   - Hyperparameters, including settings like learning rate, momentum, weight decay, etc.
   - The default hyperparameters are in the **hyp.scratch-low.yaml** file. We recommend training with the default hyperparameters before considering adjusting any parameters. Generally, increasing augmentation hyperparameters will help reduce and delay overfitting, allowing longer training and achieving higher final mAP. Reducing the hyperparameters that adjust loss components like **hyp['obj']** will help reduce overfitting in those specific loss components.

9. **data**:

   - The data configuration file contains information about the paths to the training set, test set, and object classes to detect. This parameter determines what data the model will be trained on and what the objectives are.

Understanding and adjusting these parameters is essential for optimizing the YOLOv5 model for specific problems.

4.3.7 Laboratory

Open Visual Studio Code, open Lab2_3 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab2_3_solve.

4.4 Expected results

Developing a sufficiently good model and preparing for its application to the vehicle model in Chapter 3.

Chapter III: Object Tracking

5.1 Basic knowledge about DeepSORT

5.1.1 Overview

DeepSORT (Simple Online and Realtime Tracking) is a powerful online object tracking algorithm, developed as an enhancement to the SORT algorithm. DeepSORT improves SORT by integrating deep learning features to track objects in video with higher accuracy, especially in complex situations such as multiple objects and objects appearing or disappearing from the frame.

DeepSORT was first introduced in 2017 in the paper titled ”Simple Online and Realtime Tracking with a Deep Association Metric” by Wojke, Bewley, and Paulus. The main improvement of DeepSORT over SORT is the use of a deep neural network to generate feature vectors that help distinguish between different objects, even when there are changes in position or shape.

In the field of autonomous vehicles, DeepSORT can be used in object tracking systems to identify and track other vehicles, pedestrians, and obstacles on the road. The continuous and accurate tracking capability of DeepSORT allows autonomous vehicles to make safer movement decisions, avoid collisions, and maintain an efficient travel path.

5.1.2 Learning goals

After completing this exercise, learners will gain knowledge of:

- DeepSORT

5.1.3 Related knowledge

- Python
- DeepSORT

5.1.4 Required background

To complete this exercise, you will need the following knowledge:

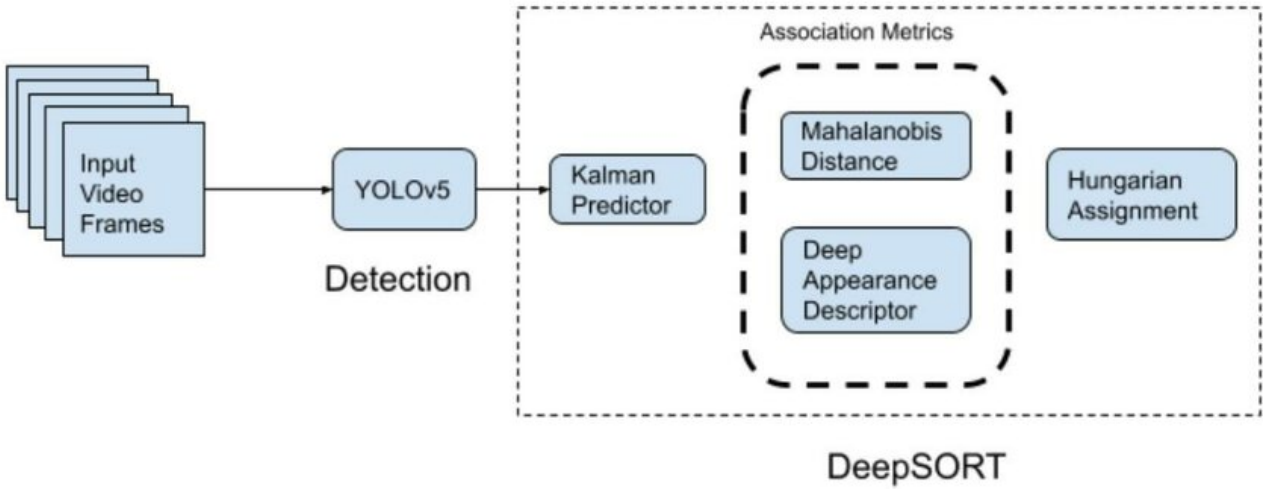
- Basic programming skills in Python

5.1.5 Problem

Objective: Apply DeepSORT with YOLO model to track objects in a video.
Requirements:

- Input: video
- Output: video with object tracking boxes

5.1.6 Theory



Steps in DeepSORT:

- YOLO for Object Detection: The input video sequence is processed frame by frame using the YOLO (You Only Look Once) algorithm, which detects objects in each frame.
- Detection Processing: Detected objects are processed by a Difference Detector to identify changes between frames, which helps in distinguishing between static and dynamic objects.
- Kalman Predict: For each detected object, a Kalman filter predicts the next position of the object based on its previous motion.
- Feature Extraction: A deep appearance descriptor is used to extract features from detected objects, which are used to maintain consistent tracking over time.
- Data Association: The Mahalanobis distance metric is employed to measure the similarity between predicted and detected objects, assisting in the association of objects across frames.
- Hungarian Assignment: The Hungarian algorithm is used to assign detected objects to existing tracks, ensuring accurate tracking of multiple objects simultaneously.

Load Detection Mode
The objects that will be tracked depend on whether the model can detect them. In the labs of Chapter 2, we have guided you on how to effectively train models for different purposes.

Please reload the weights of the model you find suitable to complete the following task.

Initialize DeepSORT
The meaning of the parameters used in the **DeepSort** function:
1. **max_age**:

   - Meaning: Specifies the maximum number of frames that an object can be lost before it is removed from the tracker.
   - Explanation: If an object is not detected for **max_age** consecutive frames, it will be considered lost and removed.

2. **n_init**:

   - Meaning: Specifies the number of initial frames in which an object must be detected consecutively before it is considered a valid tracked object.
   - Explanation: A new object must appear in at least **n_init** frames before it is confirmed as a valid object and starts being tracked.

3. **nms_max_overlap**:

   - Meaning: This parameter relates to Non-Maximum Suppression (NMS), an algorithm used to eliminate overlapping bounding boxes with different objects.
   - Explanation: **nms_max_overlap** defines the maximum overlap between bounding boxes that is still considered as two separate objects. The default value is **1.0**, meaning that bounding boxes can overlap completely without being removed.

4. **max_cosine_distance**:

   - Meaning: Specifies the maximum cosine distance between the features of objects (e.g., embedding vectors) to be considered the same object.
   - Explanation: Cosine distance is a measure of similarity between two vectors. A low **max_cosine_distance** value indicates a high requirement for similarity between two objects to be matched. The default value is typically **0.3**, meaning that only highly similar objects will be matched.

These parameters can be adjusted based on the data and specific problem you are working on to optimize the performance of the object tracking system.

Apply DeepSORT and YOLO to Video
Based on the instructions from chapter 1, try applying DeepSORT to a video.

5.1.7 Laboratory

Open Visual Studio Code, open Lab3_1 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab3_1_solve.

5.2 Apply Yolo V5 with DeepSORT to tracking trafffic signals and car

5.2.1 Learning goals

By the end of this exercise, participants will have learned the following:

- Application of YOLO for object detection
- Using DeepSORT for object tracking

5.2.2 Related knowledge

- Basic programming skills with Python
- Basic functions of OpenCV
- YOLO and DeepSORT

5.2.3 Required background

5.2.4 Problem

Objective: The vehicle model should be able to detect traffic signs and issue appropriate commands to the vehicle.
Requirements: Perform traffic sign detection using YOLO.

5.2.5 Theory

Implement the YOLOv5 Pre-trained Model and DeepSORT

Define the IP: To connect with the leader laptop to receive images and send commands you need specify the IP address. To do that, open Command Prompt, and type 'ipconfig.' The IP address will be displayed in the terminal.

Load YOLOv5 Model: YOLOv5 (You Only Look Once version 5) is a state-of-the-art model for real-time object detection. For this application, we will use a pre-trained version of YOLOv5. The smallest model, 'yolov5s', is a good starting point due to its balance between performance and computational requirements.

Initialize DeepSORT: DeepSORT (Deep Simple Online and Realtime Tracking) is used for tracking objects across frames. After detecting objects with YOLOv5, DeepSORT helps maintain the identities of these objects and track them through subsequent frames. Initialize DeepSORT with the appropriate parameters for tracking.

5.2.6 Laboratory

Open Visual Studio Code, open Lab3_2 and Lab3_3 in the Lab file to practice the above knowledge.

If you have difficulty or want to check the results, check the results in file Lab2_3_solve and Lab3_3_solve.

5.3 Expected performance results

In terms of performance, the vehicle will be able to obey traffic signals and stop when encountering vehicles.

Table 5.1: Comparison results between multithread and single thread

	Lane Detection	
	FPS	mAP@0.5
Single-thread	1-2	38%
Multi-thread	17-23	86%

Overview of expected results

Theoretical Understanding: Students will grasp the theoretical knowledge of the Canny algorithm, YOLOv5 model, and DeepSORT algorithm, and learn how to apply them to autonomous vehicles and tackle real-world problems.

Table 6.1: Configuration of the laptop used in the experiment

Configuration	CPU	RAM	GPU
Laptop leader	R5 5500U	16GB	No
Laptop member 1	I5 11300H	16GB	NVIDIA GeForce RTX 3050 4GB
Laptop member 2	I5 1240P	16GB	NVIDIA GeForce MX570 2GB

Performance: With the above configuration, the lead computer has a processing performance of 14-18 fps, the traffic signal tracking computer operates at 22-25 fps, and the vehicle tracking computer performs at 11 fps.
Optimization Consideration: These results are not the most optimal and will depend on code optimization, the configuration of the machines, and network speed.

Conclusion

We are living in an era where technology continuously advances and profoundly transforms human life. Autonomous vehicles are one of the fields with the potential to bring about revolutionary changes, helping to reduce traffic accidents, save energy, and improve the quality of life. This document has been created with the goal of providing foundational knowledge and practical applications in the development of autonomous vehicle systems, focusing on three key technologies: Canny edge detection, the YOLO model, and DeepSORT.

We would like to extend our deepest gratitude to everyone who has contributed, supported, and accompanied us in the process of building this document. In particular, we express our thanks to the pioneers in research and development of advanced technologies, who have paved the way and laid a solid foundation for the next steps.

Autonomous vehicles are not only a new technology but also an important step toward a safer, more convenient, and sustainable future. The knowledge and applications presented in this document not only help you understand the technology more deeply but also provide useful tools for you to participate in the development of autonomous vehicle systems in the future.

However, the road ahead is still long and full of challenges. The progress of technology always comes with new questions and challenges. We hope that this document will be a source of inspiration and motivation for those who are seeking opportunities to explore, learn, and contribute to the development of autonomous vehicles.

The future of autonomous vehicles is in our hands. With a combination of knowledge, creativity, and passion, we can build smarter, safer, and more efficient systems, contributing to shaping the future of transportation and urban life.

We believe that the knowledge and tools you have gained through this document will greatly benefit your work and research. We also hope that you will continue to pursue research and development in this field, while sharing your insights and experiences with the community, as we work together to build a better future.

Thank you sincerely, and we wish you success on the path you have chosen.

Bibliography

[1] M. V. S. Pavan, D. Ranjith, K. Reddy, and M. Ismail. Design and development of automated lane detection using improved canny edge detection method. *Psychology and Education*, 57(9):1350–1358, 2020.

[2] A. Ahmed, A. ElKattan, A. Omar, and H. Abdelaal. Extraction of features from a bird's-eye view for path detection. *International Journal of Artificial Intelligence and Emerging Technology*, 6(1):1–14, 2023.

[3] S. Kumar, S. K. Singh, S. Varshney, S. Singh, P. Kumar, B. Kim, and I. Ra. Fusion of deep sort and yolov5 for effective vehicle detection and tracking scheme in real-time traffic management sustainable system. *Sustainability*, 15(24):16869, 2023.