Camera-Based System for Automatic Pin Mapping

Date: 25.06.23

Author: 21900179 / JongHyeon Kim

22000188 / ChanJung Kim

Github: repository link

Demo Video: Youtube link

Introduction

The students in the school of Mechanical and Control Engineering frequently encounter a variety of IC (integrated circuit) chips during class experiments. However, since these chips often include both manufacturer names and production dates printed together, it can be difficult to identify the exact model of the chip. Additionally, the manual search for the appropriate datasheet for each chip can be inconvenient and time-consuming. To address these issues, this project proposes a vision-based system for automatic chip identification and pin mapping.

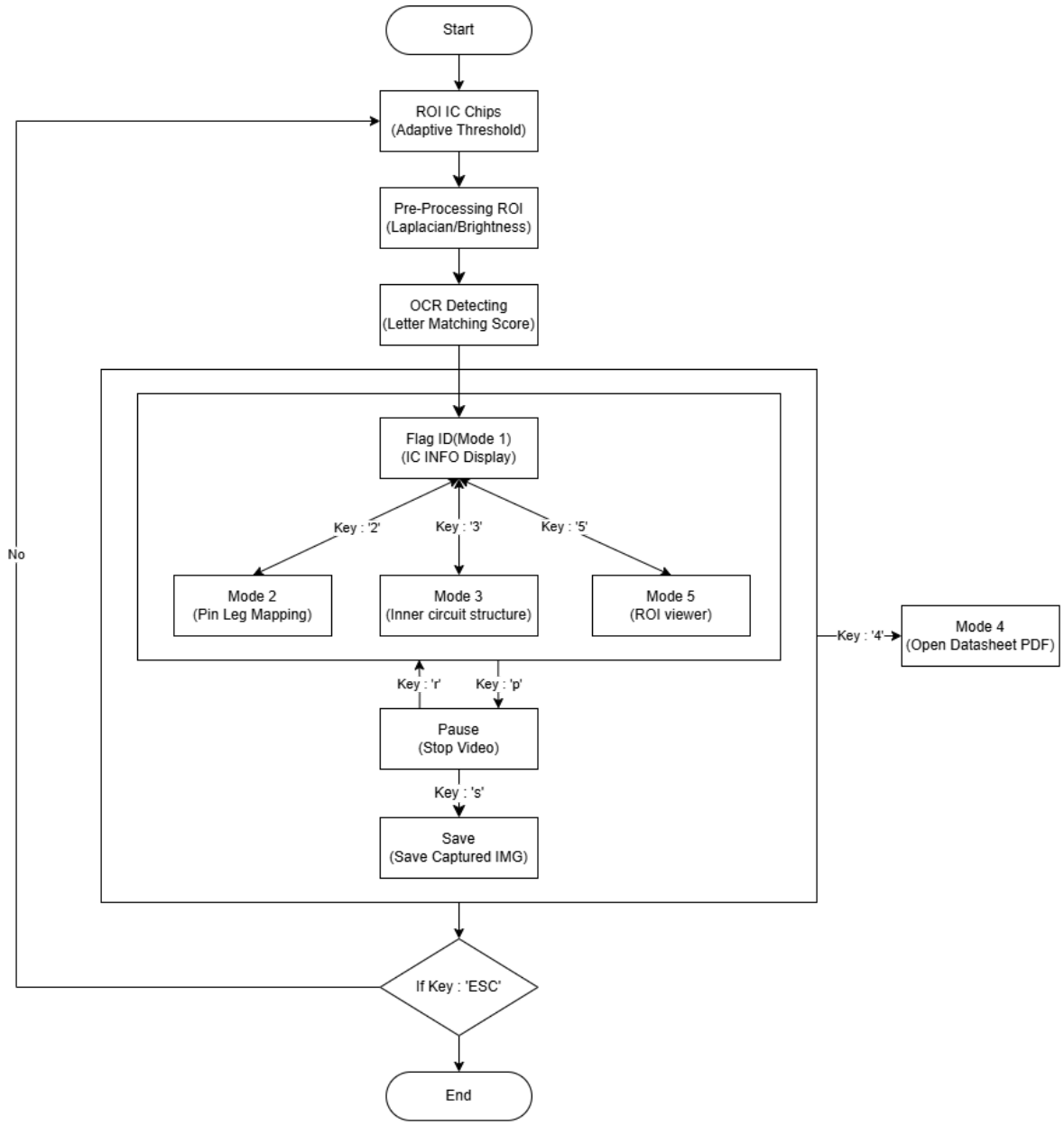
Problem Statement

- The presence of manufacturer names and production dates makes it difficult to accurately identify IC chip models.
- Manually searching for datasheets to determine pin configurations is time-consuming and inefficient.

Project Objectives

- Filter out unnecessary printed information and accurately identify the IC chip models.
- Detect chip pins and label each corresponding node.

Project Overview



Mode	Command Key	**Description**
Mode 1	1	Detects scattered IC chips and identifies what chip it is.
Mode 2	2	Identifies IC chips and displays detailed information about each pin node.
Mode 3	3	Overlay internal circuit diagram
Mode 4	4	Open PDF for detailed specifications
Mode 5	5	Display ROI image for a debugging
Pause	p	Pauses the machine vision and freezes the current frame.
Restart	R	Restarts real-time operation of the machine vision system.
Save Image	S	Saves the currently paused image for reporting or documentation purposes.

When the program is launched, it starts in **Mode 1** by default, where IC chips are detected and their models are displayed on screen. The camera feed is provided by a smartphone connected to the laptop. Users can switch between **Mode 1**, **Mode 2**, **Mode 3**, **Mode 4**, and **Mode 5** by pressing the corresponding numeric keys on the keyboard. Pressing the `P` key pauses the video stream to allow closer inspection of a specific chip. To resume the program, the `R` key can be used. Additionally, while the video is paused, pressing `S` enables users to save the current frame for later use in reports or documentation.

Expected Output

				
Mode 1	Mode 2	Mode 3	Mode 4	Mode 5

In **Mode 1**, the system identifies the IC chip's model name and its general application. In **Mode 2**, it provides additional information, such as detailed pin descriptions and node labeling. **Mode 3** overlays the internal circuit diagram to help users easily identify connection points. When the `4` key is pressed, the corresponding datasheet is displayed for more detailed specifications. Finally, **Mode 5** provides OCR input images to help the user accurately capture the IC chip.

Evaluation Method

To evaluate our vision machine's accuracy, the performance test is conducted. The conducted evaluation environment is following:

- No black objects in the camera view
- The text printed on the IC chip must be clearly visible
- The chip must not be rotated upside down.
- The smart phone placed about 15-20 [cm] above IC chips

			
CD74HC4052E	SN74LS74N	SN74HC00N	GD74LS74A
			
SN74HC04N	SN74HC08N	SN74HC86N	SN74LS47N

The evaluation is conducted with one IC chip at a time for 8 kinds of IC chip in pre-defined environment. The selected IC chips used in evaluation is shown table above. The data to evaluate accuracy of each class are collected by 50 times, counting each detected IC chips with a threshold(0.55) of validity of detected OCR. The confusion matrix is made of collected data, and the result is analyzed in **Result and Analysis**. The evaluation mode is activated by changing

`Eval_Flag` to `True`. Evaluation method is conducted for two OCRs, which are `easyocr` and `paddleocr`.

Requirement

Hardware List

- Smart phone (used as a webcam via Window11 webcam Connection)

Software Installation

- OpenCV-python 4.11
- Python 3.9.21
- Pytorch 2.1.2
- Levenshteind 0.27.1
- easyOCR 1.7.2 **(for CPU user)**
- paddlepaddle-gpu 3.0.0 / paddleOCR 3.0.2 **(Recommended for GPU user)**

Method

OCR(Optical Character Recognition)

Optical Character Recognition (OCR) is a technique that uses automated data extraction to quickly convert text images into machine-readable formats.

PaddleOCR

PaddleOCR is an OCR package developed from Paddle Paddle (a Python-based deep learning platform) called fade driver. PaddleOCR is an ultra-lightweight Optical Character Recognition (OCR) system developed by Baidu. It supports a variety of cutting-edge OCR algorithms and provides value at every stage of the AI pipeline, including data generation, model training, and inference.

This update further **boosts text-recognition accuracy**, adds support for **multiple text-type recognition** and **handwriting recognition**, and meets the growing demand from large-model applications for **high-precision parsing of complex documents**.

- **Model Architecture**

Architecture Type for Text Detector: CNN
Network Architecture for Text Detector: LK-PAN

Architecture Type for Text Recognition: Hybrid Transformer CNN
Network Architecture for Text Recognition: SVTR-LCNet (NRTR Head and CTCLoss head)

- **Input**

Input Type(s): Image
Input Format(s): Red, Green, Blue (RGB)
Input Parameters: Two Dimensional (2D)
Other Properties Related to Input: nd array, or batch of nd arrays are passed in with shape [Batch, Channel, Width, Height]. PaddleOCR does some internal thresholding, but none was implemented from our side.

- **Output**

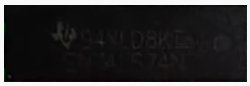
Output Type(s): Text
Output Format: String
Output Parameters: 1D
Other Properties Related to Output: Batch of text strings.

EasyOCR

EasyOCR is a Python package for detecting and extracting text from images such as photos or scanned documents. It comes with pre-trained models designed to make text recognition fast and efficient and supports over 80 languages.

The recognition model is a CRNN (paper). It is composed of 3 main components: feature extraction (we are currently using Resnet) and VGG, sequence labeling (LSTM) and decoding (CTC). The training pipeline for recognition execution is a modified version of the deep-text-recognition-benchmark framework.

Pre-Processing

			
Original ROI	Original ROI + Brightness	Original ROI + Laplacian	Original ROI + Brightness + Laplacian

After reading the text with the ROI of the above four cases, the system extracts the true value with the highest matching score and determines which IC chip it is.

Brightness

```
roiBright = cv2.convertScaleAbs(roiOriginal, alpha=1, beta=40)
```

The brightness was adjusted by adjusting the beta value using the function provided by the openCV library.

Laplacian filter

0	-1	0
-1	6	-1
0	-1	0

```
kernel = np.array([[0, -1, 0], [-1, 6, -1], [0, -1, 0]])

roiOriLaplacian = cv2.filter2D(roiOriginal, -1, kernel)
```

The filter shape was made like a picture, and the effect of the laplacian filter was reflected using the `filter2D` function.

Text recognition algorithms

Levenshtein

```
from Levenshtein import ratio
```

The ratio function provided by Levenshtein was used to display the degree of agreement with the letters as a score to increase the strength and accuracy of character recognition. The advantage of this `ratio` is that even if there are several letters in the middle, you can match them one by one and score points.

Multiple ROI

```
result1 = ocr.predict(roiOriginal)
result2 = ocr.predict(roiBright)
result3 = ocr.predict(roiOriLaplacian)
result4 = ocr.predict(roiBriLaplacian)

allTexts=[]
for item in result1+result2+result3+result4:
    texts = item.get('rec_texts', [])  # Return empty list without 'rec_texts'
    allTexts.extend(texts)

# List to store bestvmatch candidates
bestTrueCandidates = []
for text in allTexts:
    matches = [(true, ratio(Clean(text.upper()), Clean(true))) for true in
IC_INFO]
    best_match = max(matches, key=lambda x: x[1])
    bestTrueCandidates.append(best_match)
```

Rather than recognizing letters with only one ROI, it receives four ROIs in multiple ways to extract the best score among them. This not only improves accuracy, but also increases robustness through consideration of various cases.

Character normalization

```
# text : Text to replace letter
def Clean(text):
    return text.replace('B', '8')\
               .replace('O', '0')\
               .replace('Z', '7')\
               .replace('I', '1')\
               .replace("SN74", "")\
               .replace("SN", "")
```

When characters are recognized, letters and numbers can look similar, which may lead to misrecognition. To prevent this, characters like 'B', 'O', 'Z', and 'I' are converted to '8', '0', '7', and '1', respectively. These substitutions do not overlap with actual IC chip model names, so they can be safely included in a conversion list. If a substitution overlaps with a model name, it becomes difficult to replace.

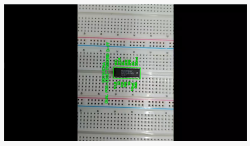
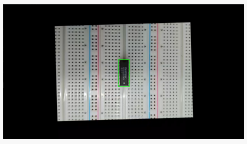
Additionally, many chips commonly include "SN74" in their names. When "SN74" is followed by a large number of incorrect characters, it significantly reduces accuracy. Therefore, if the recognized characters contain "SN74" followed by other characters, those parts are removed, and only "SN74" is used for comparison. This helps reduce false positives and improves accuracy.

Rotate text

```
x, y, w, h = cv2.boundingRect(cnt)

if w > h:
    Horizontal = True
else:
    Horizontal = False

roi = frame[y:y+h, x:x+w]
rect = cv2.minAreaRect(cnt)
box = cv2.boxPoints(rect)
box = np.intp(box)

(yc, xc), (wc, hc), angle = rect
angleROI = angle
angleGUI = angle
rate = wc/hc

if wc < hc:
    rate = hc/wc
    angleROI = angle - 90
if rate > 2.4 and rate < 3.2:
    cv2.drawContours(frame,[box],0,(0,255,0),2)     # Draw a box on the frame.
    if wc > hc and Horizontal == False:
        angleGUI -= 90
```

```
    if Horizontal == True:
        angleGUI -= 90
        if wc < hc:
            angleGUI -= 90

    roi, _ = RotateImage(roi, angleROI)
    M2 = cv2.getRotationMatrix2D((frame.shape[1]//2, frame.shape[0]//2),
angleGUI, 1.0)
    box = np.array(box, dtype=np.float32).reshape(-1, 1, 2)
    box = cv2.transform(box, M2).reshape(-1, 2)
```

By using the `cv2.minAreaRect`, `cv2.boxPoints`, `cv2.getRotationMatrix2D`, and `cv2.transform` functions, you can output the rotated ROI and extract the rotated text.

			
Original frame	Rotated frame	Rotated mask	Inverse rotated mask

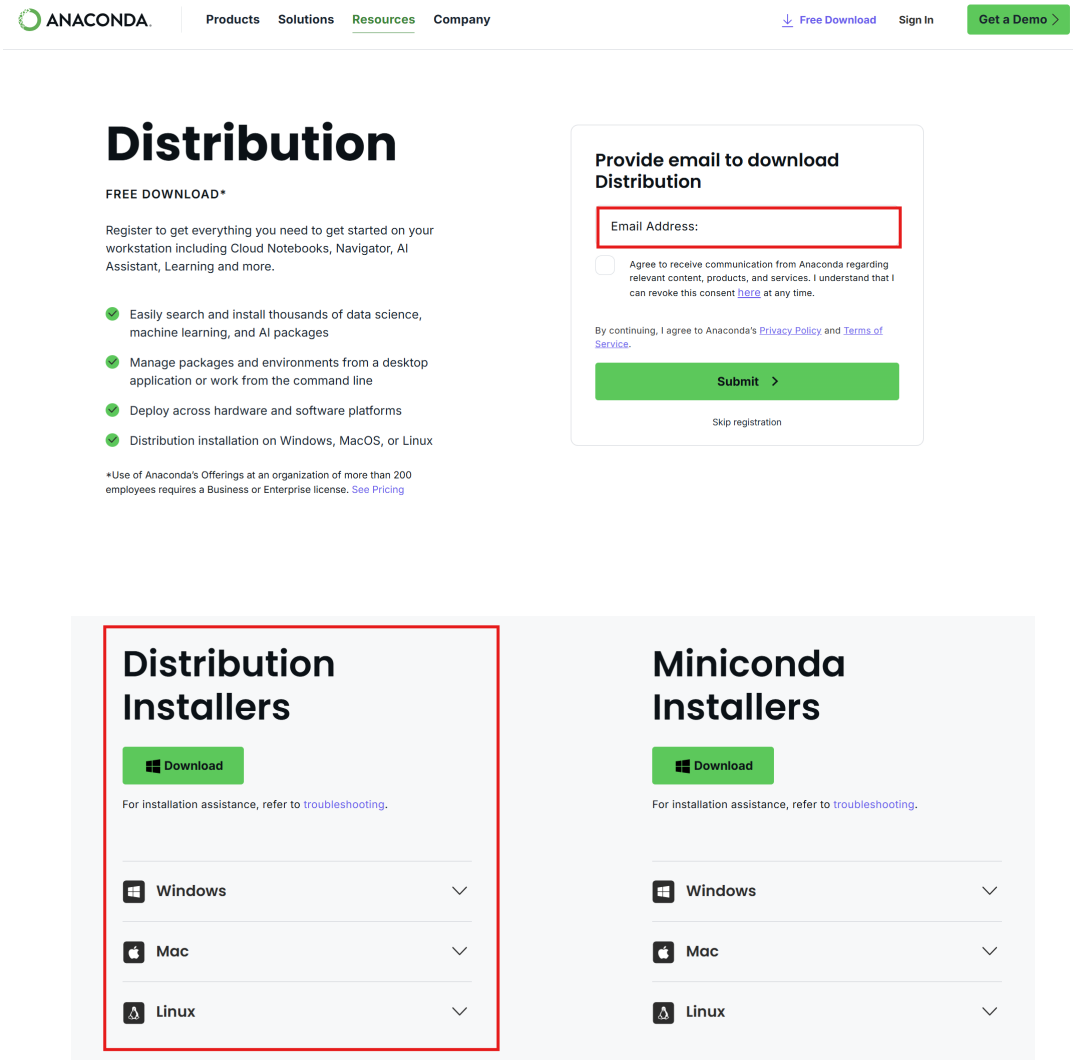
By rotating the coordinates of the box points based on the given angle, the IC chip always appears in a vertically elongated shape, resembling its real orientation. The second image demonstrates this (although in practice, we do not render this intermediate image; instead, we directly use the coordinates to draw text on an empty frame using `putText`). After drawing the text on the empty frame, we rotate it back using the inverse of the original angle so that the text aligns correctly with the original image. Finally, the text is rendered on the original frame through masking.

Procedure

Installation

How to install Anaconda Prompt

You may download the anaconda and create python environment by following steps in provided reference link: YGKim_Gitbook



How to Install Levenshtein

You can install the `Levenshtein` simply by typing following command in anaconda prompt.

```
pip install Levenshtein
```

How to Install OCR

Installation for GPU user(recommend):

1. **Install paddlepaddle**

Since GPU installation requires specific CUDA versions, the following example is for installing NVIDIA GPU on the Linux platform with CUDA 11.8 & CUDA 12.6. For other platforms, please refer to the instructions in the PaddlePaddle official installation documentation.

```
python -m pip install paddlepaddle-gpu==3.0.0 -i
https://www.paddlepaddle.org.cn/packages/stable/cu118/

python -m pip install paddlepaddle-gpu==3.0.0 -i
https://www.paddlepaddle.org.cn/packages/stable/cu126/
```

2. **Install paddleocr**

```
python -m pip install paddleocr
```

For more information, Please refer the documentation

Installation OCR for CPU user:

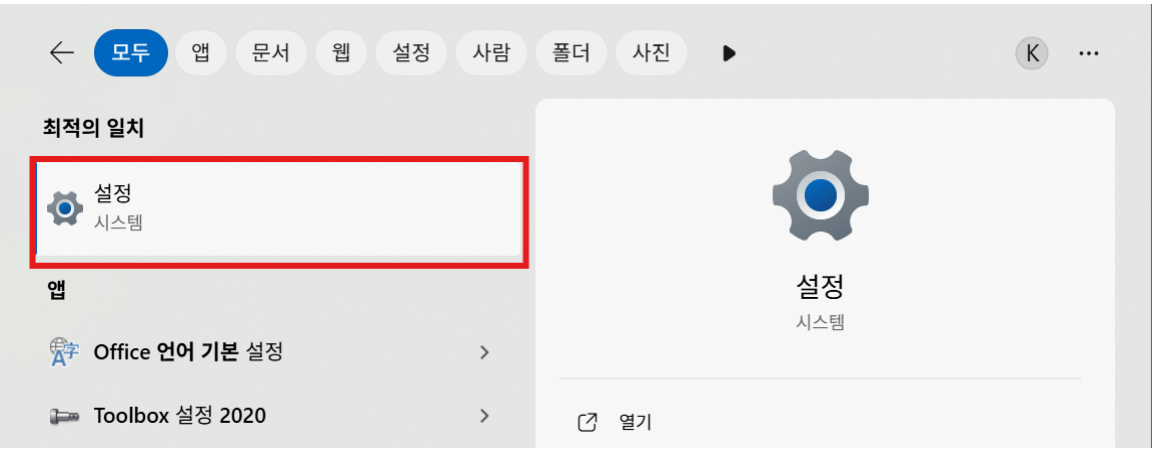
1. **Install EasyOCR**

   Open the anaconda prompt, run the environment, and enter it as follows.

```
pip install easyocr
```

How to connect a smartphone in Window11 as a webcam?

1. Search '**Settings**' in the Windows search box and press it.

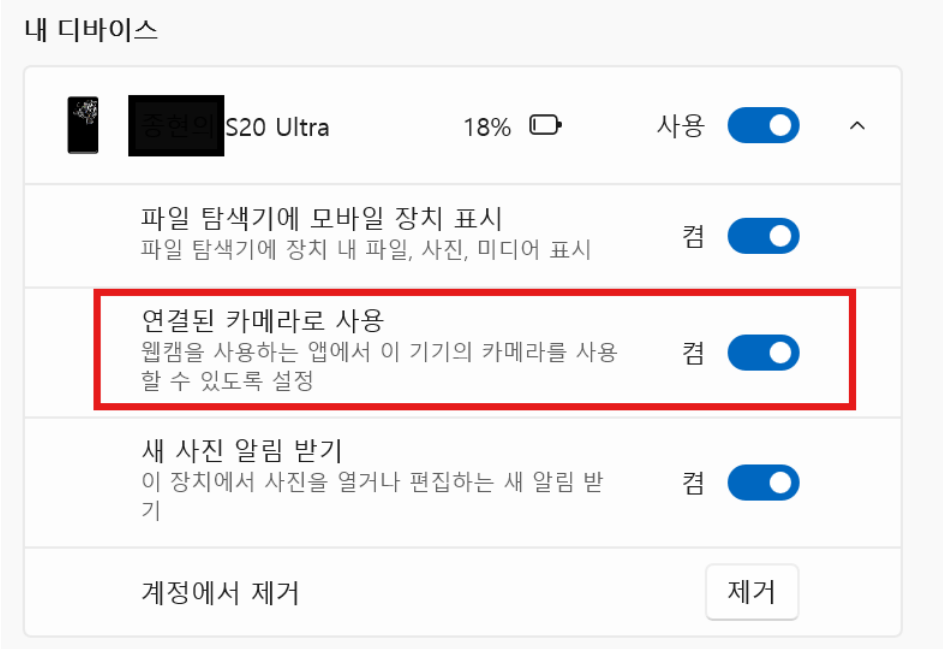


2. In the "**Bluetooth and Devices**" section, click "**Mobile Devices**".

3. Leave Allow to access mobile devices from your PC as "**Turn On**" and click "**Manage Devices**" in the mobile device section as shown in the following figure

4. If a device is not registered on the mobile device, click "**Add a device**" to add a device. When clicking, take a QR code with a smartphone and enter an authentication number as guided.

5. When you add a device, your device will be displayed as in the photo and change it to "**Turn on**" in the "**Use as a Connected Camera**" part.



The code below is the code that brings up the screen of smartphone webcam from Python code.

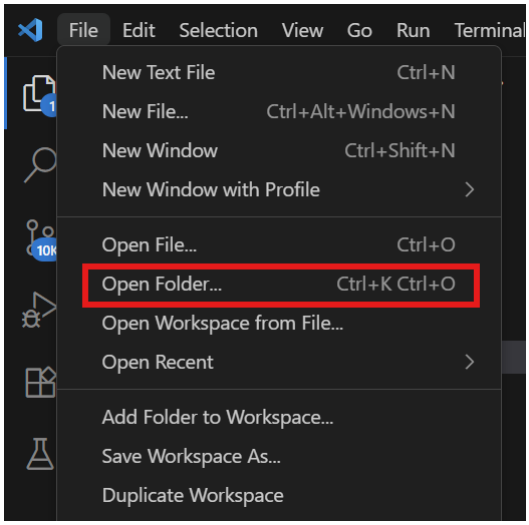
```
import cv2
cap = cv2.VideoCapture(1)
```

Tutorials

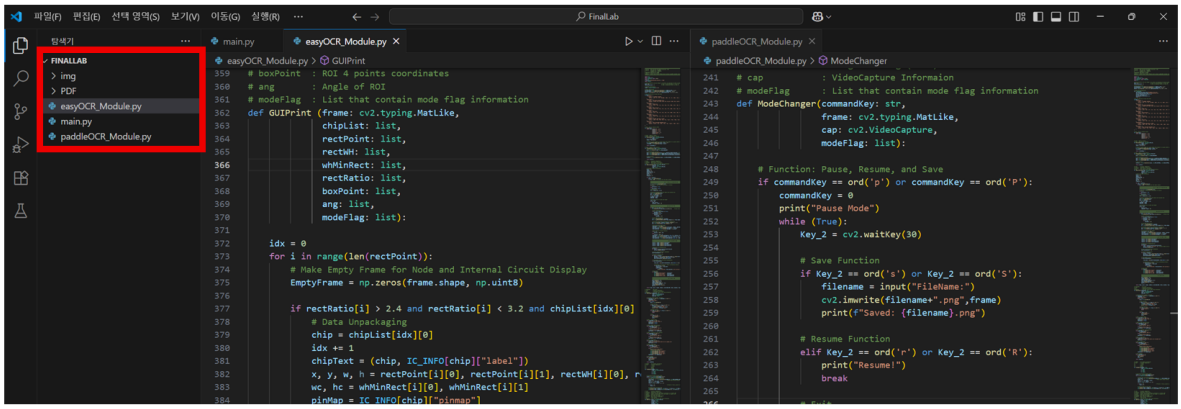
Go to following Github link to download our python code, images, and PDFs. You may download either `easyOCR` or `paddleOCR`.

- For those who downloaded `easyOCR` (for CPU user), refer to this link.
- For those who downloaded `paddleOCR` (for GPU user), refer to this link.
- Download our main code. Click here.
- Download image and PDFs `.zip` file. Click here

Unzip the `img and PDF.zip` file. In `img and PDF.zip` file, two folders, `img` and `PDF`, should be seen. Redirect them in the same directory of your main and module code. Your folder should be seen similar with in the figure above.



Run `Visual Studio Code` open folder where you set all the provided files, and make sure that the files appear well in the `EXPLORER` window, as shown below in the red box.



Open the main code, and edit the importing module section. You must change `ModuleYouDownloaded` to either `paddleOCR_Module` or `easyOCR_Module` as shown below in `main.py`

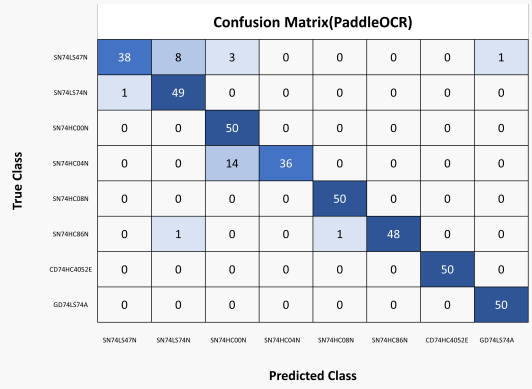
```
import cv2
# Insert appropriate module you downloaded
import paddleOCR_Module as OCRModule
```

Now, you may run the `main.py` code!

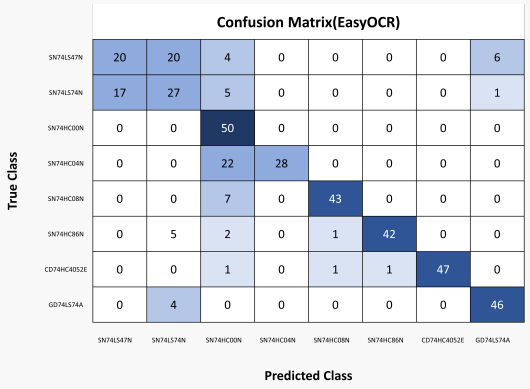
If you want to know the operation modes of various command keys and various functions, click on the Project Overview section above for reference.

Results and Analysis

Since the system displays all information related to the IC chip once it is recognized, the accuracy of the IC chip identification is the most critical aspect.



PaddleOCR



EasyOCR

This is the result of the fusion matrix extracted according to the evaluation method. The following is a formula used in calculating its accuracy.

$$Accuracy = \frac{Number\ of\ correctly\ detected\ IC\ chips}{Total\ number\ of\ attempts} \times 100\ [\%]$$

	PaddleOCR	EasyOCR
SN74LS47N	76%	40%
SN74LS74N	98%	54%
SN74HC00N	100%	100%
SN74HC04N	72%	56%
SN74HC08N	100%	86%
SN74HC86N	96%	84%
CD74HC4052E	100%	94%
GD74LS74A	100%	92%
Average	**92.75%**	**75.75%**

Using the defined formula, the accuracy for each class was calculated and averaged. The system utilizes both PaddleOCR and EasyOCR, with PaddleOCR achieving an accuracy of **92.75%** and EasyOCR achieving **75.75%**. Given PaddleOCR's relatively high accuracy, it is expected to be highly applicable in practical, real-world scenarios.

Therefore, the system effectively filters out irrelevant printed information, enabling accurate identification of IC chip models, and simultaneously detects chip pins and assigns correct labels to each corresponding node.

Discussion

According to the confusion matrix, PaddleOCR demonstrated an average accuracy that was 17% higher than EasyOCR. When analyzing the recognition accuracy of individual chips, both models showed poor performance on specific ICs, namely SN74LS47N and SN74HC04N. A common characteristic of these chips is that their printed names were severely worn, to the point of being nearly invisible to the naked eye unless viewed from an angle that reflects light.

Despite these adverse visual conditions, PaddleOCR achieved recognition accuracies of 76% and 72% for SN74LS47N and SN74HC04N, respectively, whereas EasyOCR only reached 40% and 56%. Furthermore, EasyOCR also showed low accuracy with the SN74LS74N chip. This can be attributed to the similarity in naming between SN74LS74N and SN74LS47N—only two digits differ—which likely caused the model to confuse the two during classification. In fact, the confusion matrix confirms that these two chips were frequently misidentified as each other.

Conclusion

We have successfully developed a machine vision-based system capable of detecting IC chips and identifying their pin nodes, with multiple operational modes and functionalities. The system achieved an overall accuracy of **92.75%**, even under visually degraded conditions. While the physical state of IC chips may affect recognition performance, the system demonstrated robust results, maintaining over **70% accuracy** in challenging scenarios. This machine vision solution is

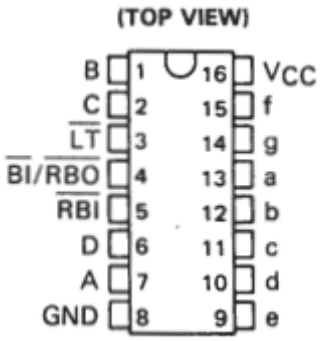
expected to support students conducting experiments with IC chips by providing functional identification, pin configuration, and access to datasheets for detailed specifications.

FAQ

How to customize IC chips information?

```
IC_INFO = {
    "SN74LS47N": {
        "label": "7-SEGMENT DECODER",
        "pinmap": ["B", "C", "LT", "BI/RBO", "RBI", "D", "A", "GND", "VCC", "f",
"g", "a", "b", "c","d","e"]
    },
```

The code above is part of IC chips dictionary. This code can be customized by the user, if there is a list of IC chips that you want to add or remove. The first part refers to IC chip name itself. Under it, are two data types which are `label` and `pinmap`. The label refers to its function type, while pinmap refers to node name following order from 1 to last node. For pinmap order, please refers to figure below and `pinmap` in above code.




```
# These parameters you may change if no letters are detected from ROI at Debug
Mode
beta = 40
LaplacianConstant = 6
```

To support mode 3 and 4, you must also find internal circuit and data-sheet of your added IC chip. The name must match with IC chip name you typed in IC_INFO. Please refer to images and PDFs we provide and add .

There is NO letters detected in ROIs in the Debug Mode

Checking ROIs shown by entering the Debug Mode(5), you may change following parameter `beta` and `LaplacianConstant`. It is because images from one's smart phone may differ from one to another. The recommended range for `beta` and `LaplacianConstant` are `[30 50]` and `[5.5 6.5]` respectively. The parameters are located at the top of each module.

```
# These parameters you may change if no letters are detected from ROI at Debug
Mode
beta = 40
LaplacianConstant = 6
```

Reference


```
https://docs.opencv.org/4.11.0/
https://www.paddlepaddle.org.cn/en
https://paddlepaddle.github.io/PaddleOCR/latest/en/index.html
https://build.nvidia.com/baidu/paddleocr/modelcard
Blog - 윈도우11 안드로이드 스마트폰 웹캠 사용 방법 [Link]
(https://blog.naver.com/kwy3000/223715853751)
```

Appendix

A. Main code

```
################################################################################
# @course   Deep Learning and Image Processing - HGU
# @author   ChanJung Kim / 22000188
#           JongHyeon Kim / 21900179
# @Created  2025-06-7 by CJKIM
# @Modified 2025-06-21 by CJKIM
# @brief    [DLIP] Final Project: Camera-Based System for Automatic Recognition
#           and Node Pin Mapping
################################################################################

import cv2
# Insert appropriate module you downloaded
import paddleOCR_Module as OCRModule

# Variable Initiation
lastValidClass = [("", 0.0) for _ in range(10)]

# ==================== Change Eval_Flag as True to Evauluate Project Accuracy
====================
evalFlag = False
evalNum = 50
count = 1
totalCount = -5

############## |Mode2| Mode3| Mode4| Mode5| Evaluation | ##############
pinModeFlag  = [False, False, False, False,  evalFlag  ]
ICName = list(OCRModule.IC_INFO.keys())
ICLength = len(OCRModule.IC_INFO)
number = [0 for _ in range(ICLength)]

# Connect Smartphone as Webcam
```

```
cap = cv2.VideoCapture(1)

if not cap.isOpened():
    print("Error: Could not open video stream.")
    exit()

#
==========================================================================
==============
while True:
    ret, frame = cap.read()
    if not ret:
        print("Error: Failed to capture image.")
        break

    # Find IC Chip Orientation
    contours = OCRModule.FindingContours(frame)

    # Identify IC Chip through Contour
    frame, point, length, rate, whMinRect, angle, box, Flag =
OCRModule.IdentifyICchip(frame, lastValidClass, contours, pinModeFlag)
    # GUI Print
    OCRModule.GUIPrint(frame, lastValidClass, point, length, whMinRect, rate,
box, angle, pinModeFlag)

    # Result Show
    cv2.imshow("Detecting",frame)
    Key = cv2.waitKey(10)
    # Mode Changing
    pinModeFlag  = OCRModule.ModeChanger(Key, frame, cap, pinModeFlag)


################################## Evaluation Accuracy
###################################
    totalCount += 1
    if  Flag and evalFlag:
        count += 1
        idx = ICName.index(lastValidClass[0][0])
        number[idx] += 1
        print(f"Total Iteration: {totalCount}\tDetected Count: {count}\n Class
Name: {lastValidClass[0][0]}\tAccuracy: {lastValidClass[0][1]*100:.2f}%")

    if count > evalNum:
        break

# Evaluation Result Print
if evalFlag:
    for ch in ICName:
        print(f"|{ch}|\t", end='')
    print(f'')
    for i in range(8):
        print(f"|\t{number[i]} |\t", end='')
```

B. easyOCR_Module.py

```
################################################################################
# @course   Deep Learning and Image Processing - HGU
# @author   ChanJung Kim / 22000188
#           JongHyeon Kim / 21900179
# @Created  2025-06-13 by CJKIM
# @Modified 2025-06-21 by JHKIM
# @brief    [DLIP] Final Project Module which contains easyOCR related function
################################################################################

import cv2
import numpy as np
from Levenshtein import ratio
import os
os.environ["KMP_DUPLICATE_LIB_OK"] = "TRUE"
import easyocr

# These parameters you may change if no letters are detected from ROI at Debug
Mode
beta = 40                           # used in convertScaleAbs in lines 174
LaplacianConstant = 6               # used in laplacian kernel in lines 27

# EasyOCR Configuration
reader = easyocr.Reader(['en'])

# Kernel Declaration
kernelMorph2x2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2,2))
kernelMorph3x3 = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
kernel = np.array([[0, -1, 0], [-1, LaplacianConstant, -1], [0, -1, 0]])

# You can add more IC chip lists here
IC_INFO = {
    "SN74LS47N": {
        "label": "7-SEGMENT DECODER",
        "pinmap": ["B", "C", "LT", "BI/RBO", "RBI", "D", "A", "GND", "VCC", "f",
"g", "a", "b", "c","d","e"]
    },
    "SN74LS74N": {
        "label": "POS-EDGE-TRIGGERED D-FF",
        "pinmap":
["1CLR","1D","1CK","1PR","1Q","1Q_not","GND","VCC","2CLR","2D","2CK","2PR","2Q",
"2Q_not"]
    },
    "SN74HC00N": {
        "label": "NAND Gate",
        "pinmap": ["1A", "1B", "1Y", "2A", "2B", "2Y", "GND", "VCC", "4B", "4A",
"4Y", "3B", "3A", "3Y"]
    },
    "SN74HC04N": {
        "label": "HEX Inverter",
        "pinmap": ["1A", "1Y", "2A", "2Y", "3A", "3Y", "GND", "VCC", "6A", "6Y",
"5A", "5Y", "4A", "4Y"]
    },
    "SN74HC08N": {
        "label": "AND Gate",
```

```
        "pinmap": ["1A", "1B", "1Y", "2A", "2B", "2Y", "GND", "VCC", "4B", "4A",
"4Y", "3B", "3A", "3Y"]
    },
    "SN74HC86N": {
        "label": "XOR Gate",
        "pinmap": ["1A", "1B", "1Y", "2A", "2B", "2Y", "GND", "VCC", "4B", "4A",
"4Y", "3B", "3A", "3Y"]
    },
    "CD74HC4052E": {
        "label": "H-SPEED CMOS MUX & D-MUX",
        "pinmap": ["B0", "B2", "BN", "B3", "B1", "Enot", "VEE", "GND", "VCC",
"A2", "A1", "AN", "A0", "A3", "S0", "S1"]
    },
    "GD74LS74A": {
        "label": "DUAL POS-EDGE-TRIGGERD D-FF",
        "pinmap": ["1CLR", "1D", "1CK", "1PR", "1Q", "1Qnot", "GND", "VCC",
"2CLR", "2D", "2CK", "2PR", "2Q", "2Qnot"]
    }
}

# text : Text to replace letter
def Clean(text):
    return text.replace('B', '8')\
               .replace('O', '0')\
               .replace('Z', '7')\
               .replace('I', '1')\
               .replace("SN74", "")\
               .replace("SN", "")

# frame : Image to rotate
# angle : The angle of rotation
def RotateImage(frame: cv2.typing.MatLike,
                angle: float):
    h,w = frame.shape[:2]
    center = (w//2,h//2)

    # Rotates a given image by a given angle with respect to a given center
point.
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotatedImg = cv2.warpAffine(frame, M,(w,h))

    return rotatedImg, M

# frame : Image to find contours
def FindingContours(frame: cv2.typing.MatLike):
    # Color Convert: BGR -> GRAY
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Pre-Processing & Fillter
    mask =
cv2.adaptiveThreshold(gray,272,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_
INV,167,34)
    mask = cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernelMorph2x2,iterations=2)
    mask = cv2.morphologyEx(mask,cv2.MORPH_DILATE,kernelMorph2x2,iterations=1)

    # Find Contours
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
```

```
    return contours

# frame     : Original image(main)
# chipList  : Detected IC Chip lists
# contours  : ROI contours information
# flag      : List that contain mode flag information
def IdentifyICchip(frame: cv2.typing.MatLike,
                   chipList: list,
                   contours: cv2.typing.MatLike,
                   flag: list):
    # Variable to Return Initiation
    point = []
    length = []
    whRatio = []
    minAreaRectWH = []
    theta = []
    boxData = []
    idx = 0
    flagCount = 0

    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 1000:  # Exception for Too small contour


 ################################################################################
#########
            ############################### Needed Information Diriven
##############################

 ################################################################################
#########
            x, y, w, h = cv2.boundingRect(cnt)
            # Identify if Chip lays as horizontal or vetical
            if w > h:
                Horizontal = True
            else:
                Horizontal = False
            # ROI Selection
            roi = frame[y:y+h, x:x+w]

            # Finding rectangle points and angle with minimum area
            rect = cv2.minAreaRect(cnt)
            box = cv2.boxPoints(rect)
            box = np.intp(box)
            _, (wc, hc), angle = rect
            angleROI = angle
            angleGUI = angle

            # Calculate ratio of width and height
            rate = wc/hc
            if wc < hc:
                rate = hc/wc
                angleROI = angle - 90
```

```
 ##########################################################################
#########
            ##################################### Pre-Processing
##################################

 ##########################################################################
#########
            if rate > 2.4 and rate < 3.2:
                cv2.drawContours(frame,[box],0,(0,255,0),2)     # Draw a box on
the frame.

                # Angle compensation
                if wc > hc and Horizontal == False:
                    angleGUI -= 90
                if Horizontal == True:
                    angleGUI -= 90
                    if wc < hc:
                        angleGUI -= 90

                # Align ROI Horizontally
                roi, _ = RotateImage(roi, angleROI)
                M2 = cv2.getRotationMatrix2D((frame.shape[1]//2,
frame.shape[0]//2), angleGUI, 1.0)

                # Calculate Rotated Box Coordinate for Future Use
                box = np.array(box, dtype=np.float32).reshape(-1, 1, 2)
                box = cv2.transform(box, M2).reshape(-1, 2)

                # Extracting OCR Input Images
                roiOriginal = roi.copy()
                roiBright = cv2.convertScaleAbs(roiOriginal, alpha=1, beta=beta)
                roiOriLaplacian = cv2.filter2D(roiOriginal, -1, kernel)
                roiBriLaplacian = cv2.filter2D(roiBright, -1, kernel)

                # Resize ROI to Increase Validity
                if wc > hc:
                    resize_y = 200 / (hc)
                else :
                    resize_y = 200 / wc
                roiOriginal = cv2.resize(roiOriginal, None, fx = resize_y, fy =
resize_y, interpolation=cv2.INTER_CUBIC)
                roiBright = cv2.resize(roiBright, None, fx = resize_y, fy =
resize_y, interpolation=cv2.INTER_CUBIC)
                roiOriLaplacian = cv2.resize(roiOriLaplacian, None, fx =
resize_y, fy = resize_y, interpolation=cv2.INTER_CUBIC)
                roiBriLaplacian = cv2.resize(roiBriLaplacian, None, fx =
resize_y, fy = resize_y, interpolation=cv2.INTER_CUBIC)


 ##########################################################################
###########
                ##################################### Identify Letters
##################################

 ##########################################################################
###########
                result1 = reader.readtext(roiOriginal)
```

```
                result2 = reader.readtext(roiBright)
                result3 = reader.readtext(roiOriLaplacian)
                result4 = reader.readtext(roiBriLaplacian)

                # Sum-up all the Detected Text
                allTexts=[]
                for item in result1+result2+result3+result4:
                    allTexts.append(item[1])

                # List to store bestvmatch candidates
                bestTrueCandidates = []

                # Evaluate Detected Text
                for text in allTexts:
                    matches = [(true, ratio(Clean(text.upper()), Clean(true)))
for true in IC_INFO]
                    best_match = max(matches, key=lambda x: x[1])
                    bestTrueCandidates.append(best_match)

                # Find Highest Score from bestTrueCandidates
                if bestTrueCandidates:

                    final_best = max(bestTrueCandidates, key=lambda x: x[1])

                    # Threshold of Accuracy for Better Confidential Result
                    if final_best[1] > 0.55:
                        chipList[idx] = final_best
                        flagCount=1
                        if not flag[4]:
                            print(f"Detected: {final_best[0]} with accuracy of
{final_best[1]*100:.2f}%")

                # Debug Mode!
                if (flag[3]):
                    cv2.imshow("ROI Original",  roiOriginal)
                    cv2.imshow("ROI Bright",    roiBright)
                    cv2.imshow("ROI Ori Lap",   roiOriLaplacian)
                    cv2.imshow("ROI Bri Lap",   roiBriLaplacian)


 ##########################################################################
######
                ################################### DATA Stored
#################################

 ##########################################################################
######
                point.append((x, y))
                length.append((w, h))
                whRatio.append(rate)
                minAreaRectWH.append((wc, hc))
                theta.append(angleGUI)
                boxData.append(box.tolist())
                idx += 1

    return frame, point, length, whRatio, minAreaRectWH, theta, boxData,
flagCount
```

```python
# commandKey    : Key input
# frame         : Original image(main)
# cap           : VideoCapture Informaion
# modeFlag      : List that contain mode flag information
def ModeChanger(commandKey: str,
                frame: cv2.typing.MatLike,
                cap: cv2.VideoCapture,
                modeFlag: list):

    # Function: Pause, Resume, and Save
    if commandKey == ord('p') or commandKey == ord('P'):
        commandKey = 0
        print("Pause Mode")
        while (True):
            Key_2 = cv2.waitKey(30)

            # Save Function
            if Key_2 == ord('s') or Key_2 == ord('S'):
                filename = input("FileName:")
                cv2.imwrite(filename+".png",frame)
                print(f"Saved: {filename}.png")

            # Resume Function
            elif Key_2 == ord('r') or Key_2 == ord('R'):
                print("Resume!")
                break

            # Exit
            elif Key_2 == 27:
                print("ByeBye!!")
                cap.release()
                exit()

    # Mode 1: Default Mode
    elif commandKey == ord('1'):
        print("Current Mode: Mode 1")
        modeFlag[0] = False
        modeFlag[1] = False

    # Mode 2: Node Identification
    elif commandKey == ord('2'):
        if not modeFlag[0]:
            print("Mode-On: Mode 2")
        else:
            print("Release Mode 2")
        modeFlag[0] = not modeFlag[0]

    # Mode 3: Internal Circuit Display
    elif commandKey == ord('3'):
        if not modeFlag[1]:
            print("Mode-On: Mode 3")
        else:
            print("Release Mode 3")
        modeFlag[1] = not modeFlag[1]

    # Mode 4: Data-Sheet
    elif commandKey == ord('4'):
        print("Corresponding data-sheet will be poped up!")
```

```
        modeFlag[2] = True

    # Mode 5: Debug Mode
    elif commandKey == ord('5'):
        if not modeFlag[3]:
            print("Debug Mode!!")
        else:
            print("Release Debug Mode")
            # Destroy ROI Windowss
            cv2.destroyWindow("ROI Original")
            cv2.destroyWindow("ROI Bright")
            cv2.destroyWindow("ROI Ori Lap")
            cv2.destroyWindow("ROI Bri Lap")
        modeFlag[3] = not modeFlag[3]

    # Exit Command
    elif commandKey == 27:
        print("ByeBye!!")
        cap.release()
        exit()

    return modeFlag

# pinMaps   : Pin mapping information
# frame     : Original image(main)
# x         : x point
# y         : y point
# w         : Width
# h         : Height
def PinMapDraw(pinMaps: list,
               frame: cv2.typing.MatLike,
               x: float,
               y: float,
               w: float,
               h: float):

    # Calculate Distance between Nodes
    offset = h / (((int)(len(pinMaps)/2))+1)
    for j in range((int)(len(pinMaps)/2)):
        # Assign Text Printing Coordinates
        (text_width, _), _ = cv2.getTextSize(pinMaps[j],
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
        margin = 10                             # Empty space margun between
text and roi
        text_x = x - text_width - margin
        text_x2 = x + w + margin
        text_y = int(y + offset * (j + 1))

        if text_x < 0:                          # Prevent if text_x points
locate out of frame
            text_x = 0

        # Display texts side and side.
        cv2.putText(frame, pinMaps[j], (int(text_x), int(text_y)),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        cv2.putText(frame, pinMaps[j+(int)(len(pinMaps)/2)], (int(text_x2),
int(y + offset * (j + 1))), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
```

```
# frame     : Original image(main)
# chipList  : Detected IC Chip lists
# rectPoint : Square enclosing ROI points coordinate in original IMG
# rectWH    : Square enclosing ROI width and height in original IMG
# whMinRect : ROI width and height
# rectRatio : ROI width and height ration
# boxPoint  : ROI 4 points coordinates
# ang       : Angle of ROI
# modeFlag  : List that contain mode flag information
def GUIPrint (frame: cv2.typing.MatLike,
              chipList: list,
              rectPoint: list,
              rectWH: list,
              whMinRect: list,
              rectRatio: list,
              boxPoint: list,
              ang: list,
              modeFlag: list):

    idx = 0
    for i in range(len(rectPoint)):
        # Make Empty Frame for Node and Internal Circuit Display
        EmptyFrame = np.zeros(frame.shape, np.uint8)

        if rectRatio[i] > 2.4 and rectRatio[i] < 3.2 and chipList[idx][0] != '':
            # Data Unpackaging
            chip = chipList[idx][0]
            idx += 1
            chipText = (chip, IC_INFO[chip]["label"])
            x, y, w, h = rectPoint[i][0], rectPoint[i][1], rectWH[i][0],
rectWH[i][1]
            wc, hc = whMinRect[i][0], whMinRect[i][1]
            pinMap = IC_INFO[chip]["pinmap"]
            BoxTemp = boxPoint[i]

            # Make Rotation Matrix for Node and Internal Circuit Display
            M = cv2.getRotationMatrix2D((frame.shape[1]//2, frame.shape[0]//2),
ang[i], 1.0)

            smallest_two = sorted(BoxTemp, key=lambda p: p[0])[:2]  # Only two
small x-point box coordinates were received.
            smallest = sorted(smallest_two, key=lambda p:p[1])[:1]  # Then, only
one of the smallest box coordinates in y-point is received.

            # Conpensate for PinMapDraw
            if (wc > hc):
                wc, hc = hc, wc

            # Mode 2 / 3 / 4
            if modeFlag[0] or modeFlag[1] or modeFlag[2]:

                # Mode 2 - Node pin mapping.
                if modeFlag[0]:
                    PinMapDraw(pinMap, EmptyFrame, smallest[0][0], smallest[0]
[1], wc, hc)

                # Mode 3 - Display circuit diagram image.
                if modeFlag[1]:
```

```
                    ChipMap = cv2.imread("img/"+chip+".png")     # Use relative
path
                    if ChipMap is not None:
                        resized_ChipMap = cv2.resize(ChipMap,(int(wc),int(hc)))

                        # Make an Exception for when Size of resized_ChipMap
does not fit
                        x = int(smallest[0][0]); y = int(smallest[0][1])
                        h_frame, w_frame = EmptyFrame.shape[:2]
                        if y + int(hc) <= h_frame and x + int(wc) <= w_frame:
                            EmptyFrame[y:y+int(hc), x:x+int(wc)] =
resized_ChipMap
                    else:
                        print(f"Circuit Images Not Found 404")

                # Mode 4 - Open PDF datasheet.
                if modeFlag[2]:
                    pdf_path = os.path.join("pdf", chip+".pdf")  # Use relative
path
                    if os.path.exists(pdf_path):
                        os.startfile(pdf_path)
                    else:
                        print("PDF 파일이 존재하지 않습니다:", pdf_path)

                # Print IC Name for Mode 2, 3, 4.
                for k in range(len(chipText)):
                    size, _ = cv2.getTextSize(chipText[k],
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
                    dx = size[0]/2
                    dy = size[1]/2 + (size[1] + size[1]/3) * (len(chipText) - k)
                    cv2.putText(EmptyFrame,chipText[k],((int)(smallest[0][0] +
wc/2 - dx),(int)(smallest[0][1]-dy)),cv2.FONT_HERSHEY_SIMPLEX,0.6, (36, 255,
12), 2)

                # Rotate the Frame Back to its Original Position.
                M_inv = cv2.invertAffineTransform(M)
                EmptyFrame = cv2.warpAffine(EmptyFrame, M_inv, (frame.shape[1],
frame.shape[0]))
                # Mask and Print out Shaped Letters and Images in the Original
Image.
                mask = cv2.cvtColor(EmptyFrame, cv2.COLOR_BGR2GRAY)
                _, mask = cv2.threshold(mask, 1, 255, cv2.THRESH_BINARY)
                cv2.copyTo(EmptyFrame, mask, frame)
            else:
                # Print IC Name for Mode 1.
                for k in range(len(chipText)):
                    size, _ = cv2.getTextSize(chipText[k],
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
                    dx = size[0]/2
                    dy = size[1]/2 + (size[1] + size[1]/3) * (len(chipText) - k)
                    cv2.putText(frame,chipText[k],((int)(x + w/2 - dx),(int)(y -
dy)),cv2.FONT_HERSHEY_SIMPLEX,0.6, (36, 255, 12), 2)

    modeFlag[2] = False
```

C. paddleOCR_Module.py

```
################################################################################
#
# @course   Deep Learning and Image Processing - HGU
# @author   ChanJung Kim / 22000188
#           JongHyeon Kim / 21900179
# @Created  2025-06-20 by JHKIM
# @Modified 2025-06-21 by CJKIM
# @brief    [DLIP] Final Project Module which contains paddleOCR related
function
################################################################################
#

import cv2
import numpy as np
from Levenshtein import ratio
from paddleocr import PaddleOCR
import os

# These parameters you may change if no letters are detected from ROI at Debug
Mode
beta = 40                           # used in convertScaleAbs in lines 180
LaplacianConstant = 6               # used in laplacian kernel in lines 27

# Kernel Declaration
kernelMorph2x2 = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (2,2))
kernelMorph3x3 = cv2.getStructuringElement(cv2.MORPH_RECT, (3,3))
kernel = np.array([[0, -1, 0], [-1, LaplacianConstant, -1], [0, -1, 0]])

# Paddle OCR Configuration
ocr = PaddleOCR(
    text_detection_model_name="PP-OCRv5_server_det",
    text_recognition_model_name="PP-OCRv5_server_rec",
    use_doc_orientation_classify=False,
    use_doc_unwarping=False,
    use_textline_orientation=False,
    lang='en')

# You can add more IC chip lists here
IC_INFO = {
    "SN74LS47N": {
        "label": "7-SEGMENT DECODER",
        "pinmap": ["B", "C", "LT", "BI/RBO", "RBI", "D", "A", "GND", "VCC", "f",
"g", "a", "b", "c","d","e"]
    },
    "SN74LS74N": {
        "label": "POS-EDGE-TRIGGERED D-FF",
        "pinmap":
["1CLR","1D","1CK","1PR","1Q","1Q_not","GND","VCC","2CLR","2D","2CK","2PR","2Q",
"2Q_not"]
    },
    "SN74HC00N": {
        "label": "NAND Gate",
        "pinmap": ["1A", "1B", "1Y", "2A", "2B", "2Y", "GND", "VCC", "4B", "4A",
"4Y", "3B", "3A", "3Y"]
    },
```

```
    "SN74HC04N": {
        "label": "HEX Inverter",
        "pinmap": ["1A", "1Y", "2A", "2Y", "3A", "3Y", "GND", "VCC", "6A", "6Y",
"5A", "5Y", "4A", "4Y"]
    },
    "SN74HC08N": {
        "label": "AND Gate",
        "pinmap": ["1A", "1B", "1Y", "2A", "2B", "2Y", "GND", "VCC", "4B", "4A",
"4Y", "3B", "3A", "3Y"]
    },
    "SN74HC86N": {
        "label": "XOR Gate",
        "pinmap": ["1A", "1B", "1Y", "2A", "2B", "2Y", "GND", "VCC", "4B", "4A",
"4Y", "3B", "3A", "3Y"]
    },
    "CD74HC4052E": {
        "label": "H-SPEED CMOS MUX & D-MUX",
        "pinmap": ["B0", "B2", "BN", "B3", "B1", "Enot", "VEE", "GND", "VCC",
"A2", "A1", "AN", "A0", "A3", "S0", "S1"]
    },
    "GD74LS74A": {
        "label": "DUAL POS-EDGE-TRIGGERD D-FF",
        "pinmap": ["1CLR", "1D", "1CK", "1PR", "1Q", "1Qnot", "GND", "VCC",
"2CLR", "2D", "2CK", "2PR", "2Q", "2Qnot"]
    }
}

# text : Text to replace letter
def Clean(text):
    return text.replace('B', '8')\
               .replace('O', '0')\
               .replace('Z', '7')\
               .replace('I', '1')\
               .replace("SN74", "")\
               .replace("SN", "")


# frame : Image to rotate
# angle : The angle of rotation
def RotateImage(frame: cv2.typing.MatLike,
                angle: float):
    h,w = frame.shape[:2]
    center = (w//2,h//2)

    # Rotates a given image by a given angle with respect to a given center
point.
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotatedImg = cv2.warpAffine(frame, M,(w,h))

    return rotatedImg, M

# frame : Image to find contours
def FindingContours(frame: cv2.typing.MatLike):
    # Color Convert: BGR -> GRAY
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # Pre-Processing & Fillter
```

```
    mask =
cv2.adaptiveThreshold(gray,272,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY_
INV,167,34)
    mask = cv2.morphologyEx(mask,cv2.MORPH_OPEN,kernelMorph2x2,iterations=2)
    mask = cv2.morphologyEx(mask,cv2.MORPH_DILATE,kernelMorph2x2,iterations=1)

    # Find Contours
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    return contours

# frame     : Original image(main)
# chipList  : Detected IC Chip lists
# contours  : ROI contours information
# flag      : List that contain mode flag information
def IdentifyICchip(frame: cv2.typing.MatLike,
                chipList: list,
                contours: cv2.typing.MatLike,
                flag: list):
    # Return Variable Initiation
    point = []
    length = []
    whRatio = []
    minAreaRectWH = []
    theta = []
    boxData = []
    idx = 0
    flagCount = 0

    for cnt in contours:
        area = cv2.contourArea(cnt)
        if area > 1000:  # Exception for Too small contour


 ##############################################################################
#########
            ############################### Needed Information Diriven
##############################

 ##############################################################################
#########
            x, y, w, h = cv2.boundingRect(cnt)
            # Identify if Chip lays as horizontal or vetical
            if w > h:
                Horizontal = True
            else:
                Horizontal = False
            # ROI Selection
            roi = frame[y:y+h, x:x+w]

            # Finding rectangle points and angle with minimum area
            rect = cv2.minAreaRect(cnt)
            box = cv2.boxPoints(rect)
            box = np.intp(box)
            _, (wc, hc), angle = rect
            angleROI = angle
            angleGUI = angle
```

```
            # Calculate ratio of width and height
            rate = wc/hc
            if wc < hc:
                rate = hc/wc
                angleROI = angle - 90


 ##############################################################################
#########
            ##################################### Pre-Processing
##################################

 ##############################################################################
#########
            if rate > 2.4 and rate < 3.2:
                cv2.drawContours(frame,[box],0,(0,255,0),2)     # Draw a box on
the frame.

                # Angle compensation
                if wc > hc and Horizontal == False:
                    angleGUI -= 90
                if Horizontal == True:
                    angleGUI -= 90
                    if wc < hc:
                        angleGUI -= 90

                # Align ROI Horizontally
                roi, _ = RotateImage(roi, angleROI)
                M2 = cv2.getRotationMatrix2D((frame.shape[1]//2,
frame.shape[0]//2), angleGUI, 1.0)

                # Calculate Rotated Box Coordinate for Future Use
                box = np.array(box, dtype=np.float32).reshape(-1, 1, 2)
                box = cv2.transform(box, M2).reshape(-1, 2)

                # Extracting OCR Input Images
                roiOriginal = roi.copy()
                roiBright = cv2.convertScaleAbs(roiOriginal, alpha=1, beta=beta)

                roiOriLaplacian = cv2.filter2D(roiOriginal, -1, kernel)
                roiBriLaplacian = cv2.filter2D(roiBright, -1, kernel)


 ##############################################################################
###########
                ################################### Identify Letters
###################################

 ##############################################################################
###########
                result1 = ocr.predict(roiOriginal)
                result2 = ocr.predict(roiBright)
                result3 = ocr.predict(roiOriLaplacian)
                result4 = ocr.predict(roiBriLaplacian)

                # Sum-up all the Detected Text
                allTexts=[]
```

```
                for item in result1+result2+result3+result4:
                    texts = item.get('rec_texts', [])  # Return empty list
without 'rec_texts'
                    allTexts.extend(texts)

                # List to Store Bestmatch Candidates
                bestTrueCandidates = []

                # Evaluate Detected Text
                for text in allTexts:
                    matches = [(true, ratio(Clean(text.upper()), Clean(true)))
for true in IC_INFO]
                    best_match = max(matches, key=lambda x: x[1])
                    bestTrueCandidates.append(best_match)

                # Find Highest Score from bestTrueCandidates
                if bestTrueCandidates:

                    final_best = max(bestTrueCandidates, key=lambda x: x[1])

                    # Threshold of Accuracy for Better Confidential Result
                    if final_best[1] > 0.55:
                        chipList[idx] = final_best
                        flagCount=1
                        if not flag[4]:
                            print(f"Detected: {final_best[0]} with accuracy of
{final_best[1]*100:.2f}%")

                # Debug Mode!
                if (flag[3]):
                    cv2.imshow("ROI Original",  roiOriginal)
                    cv2.imshow("ROI Bright",    roiBright)
                    cv2.imshow("ROI Ori Lap",   roiOriLaplacian)
                    cv2.imshow("ROI Bri Lap",   roiBriLaplacian)


 ##########################################################################
######
                ################################### DATA Stored
###################################

 ##########################################################################
######
                point.append((x, y))
                length.append((w, h))
                whRatio.append(rate)
                minAreaRectWH.append((wc, hc))
                theta.append(angleGUI)
                boxData.append(box.tolist())
                idx += 1

    return frame, point, length, whRatio, minAreaRectWH, theta, boxData,
flagCount

# commandKey    : Key input
# frame         : Original image(main)
# cap           : VideoCapture Informaion
# modeFlag      : List that contain mode flag information
```

```python
def ModeChanger(commandKey: str,
                frame: cv2.typing.MatLike,
                cap: cv2.VideoCapture,
                modeFlag: list):

    # Function: Pause, Resume, and Save
    if commandKey == ord('p') or commandKey == ord('P'):
        commandKey = 0
        print("Pause Mode")
        while (True):
            Key_2 = cv2.waitKey(30)

            # Save Function
            if Key_2 == ord('s') or Key_2 == ord('S'):
                filename = input("FileName:")
                cv2.imwrite(filename+".png",frame)
                print(f"Saved: {filename}.png")

            # Resume Function
            elif Key_2 == ord('r') or Key_2 == ord('R'):
                print("Resume!")
                break

            # Exit
            elif Key_2 == 27:
                print("ByeBye!!")
                cap.release()
                exit()

    # Mode 1: Default Mode
    elif commandKey == ord('1'):
        print("Current Mode: Mode 1")
        modeFlag[0] = False
        modeFlag[1] = False

    # Mode 2: Node Identification
    elif commandKey == ord('2'):
        if not modeFlag[0]:
            print("Mode-On: Mode 2")
        else:
            print("Release Mode 2")
        modeFlag[0] = not modeFlag[0]

    # Mode 3: Internal Circuit Display
    elif commandKey == ord('3'):
        if not modeFlag[1]:
            print("Mode-On: Mode 3")
        else:
            print("Release Mode 3")
        modeFlag[1] = not modeFlag[1]

    # Mode 4: Data-Sheet
    elif commandKey == ord('4'):
        print("Corresponding data-sheet will be poped up!")
        modeFlag[2] = True

    # Mode 5: Debug Mode
    elif commandKey == ord('5'):
```

```
        if not modeFlag[3]:
            print("Debug Mode!!")
        else:
            print("Release Debug Mode")
            # Destroy ROI Windows
            cv2.destroyWindow("ROI Original")
            cv2.destroyWindow("ROI Bright")
            cv2.destroyWindow("ROI Ori Lap")
            cv2.destroyWindow("ROI Bri Lap")
        modeFlag[3] = not modeFlag[3]

    # Exit Command
    elif commandKey == 27:
        print("ByeBye!!")
        cap.release()
        exit()

    return modeFlag


# pinMaps   : Pin mapping information
# frame     : Original image(main)
# x         : x point
# y         : y point
# w         : Width
# h         : Height
def PinMapDraw(pinMaps: list,
               frame: cv2.typing.MatLike,
               x: float,
               y: float,
               w: float,
               h: float):

    # Calculate Distance between Nodes
    offset = h / (((int)(len(pinMaps)/2))+1)
    for j in range((int)(len(pinMaps)/2)):

        # Assign Text Printing Coordinates
        (text_width, _), _ = cv2.getTextSize(pinMaps[j],
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
        margin = 10                                 # Empty space margun between
text and roi
        text_x = x - text_width - margin
        text_x2 = x + w + margin
        text_y = int(y + offset * (j + 1))

        if text_x < 0:                              # Prevent if text_x points
locate out of frame
            text_x = 0

        # Display Nodes by each Side
        cv2.putText(frame, pinMaps[j], (int(text_x), int(text_y)),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        cv2.putText(frame, pinMaps[j+(int)(len(pinMaps)/2)], (int(text_x2),
int(y + offset * (j + 1))), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)


# frame     : Original image(main)
```

```
# chipList  : Detected IC Chip lists
# rectPoint : Square enclosing ROI points coordinate in original IMG
# rectWH    : Square enclosing ROI width and height in original IMG
# whMinRect : ROI width and height
# rectRatio : ROI width and height ration
# boxPoint  : ROI 4 points coordinates
# ang       : Angle of ROI
# modeFlag  : List that contain mode flag information
def GUIPrint (frame: cv2.typing.MatLike,
              chipList: list,
              rectPoint: list,
              rectWH: list,
              whMinRect: list,
              rectRatio: list,
              boxPoint: list,
              ang: list,
              modeFlag: list):

    idx = 0
    for i in range(len(rectPoint)):
        # Make Empty Frame for Node and Internal Circuit Display
        EmptyFrame = np.zeros(frame.shape, np.uint8)

        if rectRatio[i] > 2.4 and rectRatio[i] < 3.2 and chipList[idx][0] != '':
            # Data Unpackaging
            chip = chipList[idx][0]
            idx += 1
            chipText = (chip, IC_INFO[chip]["label"])
            x, y, w, h = rectPoint[i][0], rectPoint[i][1], rectWH[i][0],
rectWH[i][1]
            wc, hc = whMinRect[i][0], whMinRect[i][1]
            pinMap = IC_INFO[chip]["pinmap"]
            BoxTemp = boxPoint[i]

            # Make Rotation Matrix for Node and Internal Circuit Display
            M = cv2.getRotationMatrix2D((frame.shape[1]//2, frame.shape[0]//2),
ang[i], 1.0)

            smallest_two = sorted(BoxTemp, key=lambda p: p[0])[:2]  # Only two
small x-point box coordinates were received.
            smallest = sorted(smallest_two, key=lambda p:p[1])[:1]  # Then, only
one of the smallest box coordinates in y-point is received.

            # Conpensate for PinMapDraw
            if (wc > hc):
                wc, hc = hc, wc

            # Mode 2 / 3 / 4
            if modeFlag[0] or modeFlag[1] or modeFlag[2]:

                # Mode 2 - Node pin mapping.
                if modeFlag[0]:
                    PinMapDraw(pinMap, EmptyFrame, smallest[0][0], smallest[0]
[1], wc, hc)

                # Mode 3 - Display circuit diagram image.
                if modeFlag[1]:
```

```
                    ChipMap = cv2.imread("img/"+chip+".png")     # Use relative
path
                    if ChipMap is not None:
                        resized_ChipMap = cv2.resize(ChipMap,(int(wc),int(hc)))

                        # Make an Exception for when Size of resized_ChipMap
does not fit
                        x = int(smallest[0][0]); y = int(smallest[0][1])
                        h_frame, w_frame = EmptyFrame.shape[:2]
                        if y + int(hc) <= h_frame and x + int(wc) <= w_frame:
                            EmptyFrame[y:y+int(hc), x:x+int(wc)] =
resized_ChipMap
                    else:
                        print(f"Circuit Image Not Found 404")

                # Mode 4 - Open PDF datasheet.
                if modeFlag[2]:
                    pdf_path = os.path.join("pdf", chip+".pdf")  # Use relative
path
                    if os.path.exists(pdf_path):
                        os.startfile(pdf_path)
                    else:
                        print(f"PDF file Not Found 404 {pdf_path}")

                # Print IC Name for Mode 2, 3, 4.
                for k in range(len(chipText)):
                    size, _ = cv2.getTextSize(chipText[k],
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
                    dx = size[0]/2
                    dy = size[1]/2 + (size[1] + size[1]/3) * (len(chipText) - k)
                    cv2.putText(EmptyFrame,chipText[k],((int)(smallest[0][0] +
wc/2 - dx),(int)(smallest[0][1]-dy)),cv2.FONT_HERSHEY_SIMPLEX,0.6, (36, 255,
12), 2)

                # Rotate the Frame Back to its Original Position.
                M_inv = cv2.invertAffineTransform(M)
                EmptyFrame = cv2.warpAffine(EmptyFrame, M_inv, (frame.shape[1],
frame.shape[0]))
                # Mask and Print out Shaped Letters and Images in the Original
Image.
                mask = cv2.cvtColor(EmptyFrame, cv2.COLOR_BGR2GRAY)
                _, mask = cv2.threshold(mask, 1, 255, cv2.THRESH_BINARY)
                cv2.copyTo(EmptyFrame, mask, frame)
            else:
                # Print IC Name for Mode 1.
                for k in range(len(chipText)):
                    size, _ = cv2.getTextSize(chipText[k],
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
                    dx = size[0]/2
                    dy = size[1]/2 + (size[1] + size[1]/3) * (len(chipText) - k)
                    cv2.putText(frame,chipText[k],((int)(x + w/2 - dx),(int)(y -
dy)),cv2.FONT_HERSHEY_SIMPLEX,0.6, (36, 255, 12), 2)

    modeFlag[2] = False
```