檔案結構

```
LICENSE-PLATE-CONTROL-SYSTEM
│  readme.txt
│
├─client
│      upload ──────── output.mp4
│      capturecam.py
│      client.py
│      servo.py
│      ultrasonicsensor.py
│
└─server
    │  cropped_letter  ────────  .jpg
    │  upload ──────── output.mp4
    │  cprs.py
    │  database.py
    │  records.json
    │  server.py
    │
```

建立server的程式的虛擬環境

在server or PC上，需要去下載anaconda且activate an environment activate完之後才安裝相對應的安裝包

```
cd server
pip install inference supervision roboflow numpy threading opencv-python re flask
json
```

建立樹梅派的程式的虛擬環境

在樹梅派上安裝Raspi-OS 64bits
需要去下載anaconda且activate an environment activate完之後才分別針對server, client安裝相對應的安裝包

```
conda create -n env_name python==3.9
conda activate env_name
```

```
cd client
pip install requests, opencv-python, time, gpiozero, tikinter
```

查看server IPv4位置

可以用`ipconfig`查看PC的IPv4位置，修改樹梅派client.py中的server_url為server的IPv4位置，預設為port=5000

```
def post_video():
    # Replace 'your_server_ip' with the actual IP address of your server
    server_url = '' # http://xxx.xxx.xx.xx:5000
```

安裝好client, server之環境後

Server執行

```
python ./server/server.py
```

樹梅派執行

```
python ./client/client.py
```

Appendix

Under ./client/

client.py為client主要在執行的程式，集合了所有的程式邏輯，樹梅派不停地感測超聲波的距離，一旦超聲波距離小於20公分，就會開始錄影並且儲存在./videos/output.mp4，然後post_video到server，然後這時候等待server回應，進而控制servo的動作

client.py

```
import requests
import os
import servo
import ultrasonicsensor as us
import cv2
import capturecam as cam
import time

def post_video():
    # Replace 'your_server_ip' with the actual IP address of your server
    server_url = '' # http://xxx.xxx.xx.xx:5000 your server IPv4 address

    # Replace 'image_path' with the path to your image file on your client machine
    file_path = "./videos/output.mp4"
    if not os.path.isfile(file_path):
        print("Error: File not found.")
        return

    filename = os.path.basename(file_path)
    file_extension = os.path.splitext(filename)[1].lower()
    valid_extensions = ('.jpg', '.jpeg', '.png', '.mp4')

    # Open the image file in binary mode
    with open(file_path, "rb") as image_file:

        if file_extension == '.jpg' or file_extension == '.jpeg':
            content_type = "image/jpeg"
        elif file_extension == '.png':
            content_type = "image/png"
        elif file_extension == '.mp4':
            content_type = "video/mp4"
        else:
            content_type = "application/octet-stream"  # Default for unknown types

        files = {"file": (filename, image_file, content_type)}  # Specify content
type

        # Send a POST request to the server's upload endpoint
        response = requests.post(server_url, files=files)
```

```
    if response.status_code == 200:
        print("Image uploaded successfully!")
        return True
    else:
        print(f"Upload failed: {response.text}")
        return False

if __name__ == "__main__" :
    servo.sg90_close()
    time.sleep(3)
    while True :
        distance = int(us.distance_sensor.distance*100) # distance in cm
        if distance < 20 :
            print("[ULTRASONICS SENSOR] IN AREA")
            cam.record_video()
            if post_video():
                servo.sg90_open()
                while int(us.distance_sensor.distance*100) < 30 : pass
                time.sleep(3)
                servo.sg90_close()
```

擷取frames儲存成output.mp4

capturecam.py

```
import cv2
import time

def record_video():
    cap = cv2.VideoCapture(0)
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    out = cv2.VideoWriter('./videos/output.mp4', fourcc, 20.0, (width, height))

    start_time = time.time()
    while(int(time.time() - start_time) < 2):
        ret, frame = cap.read()
        if ret:
            out.write(frame)
            cv2.imshow('frame', frame)
            if cv2.waitKey(1) == ord('q'):
                break
        else:
            break

    cap.release()
    out.release()
    cv2.destroyAllWindows()
```

初始設定servo為GPIO17，一旦樹梅派收到server的訊息，可能就會呼叫servo.py裡面開啟和關閉道閘的函示

servo.py

```
from gpiozero import Servo
from time import sleep
servo = Servo(17)
def sg90_open() :
    print("[SERVO] OPEN GATE!!!")
    servo.min()
    sleep(0.5)
    servo.value = None

def sg90_close() :
    print("[SERVO] CLOSE GATE!!!")
    servo.max()
    sleep(0.5)
    servo.value = None
```

用來取得超聲波測出來的距離

ultrasonicsensor.py

```
from gpiozero import DistanceSensor  # Import the DistanceSensor class from the
gpiozero library
from time import sleep  # Import the sleep function from the time module for delay
# Initialize the ultrasonic sensor
distance_sensor = DistanceSensor(echo=24, trigger=23, max_distance=5)
```

Under ./server/

server.py為主要server端運行的程式，集合了所有的程式邏輯，以flask當作架構，接收client端的post，進而開始使用model進行inference，還有使用database搜尋，並把控制的結果回傳給client端

server.py

```
import cprs as RT_CPR # real time car plate recognition system
import os
from flask import Flask, request, redirect, url_for, make_response
from werkzeug.utils import secure_filename
from flask import send_from_directory
import database as db
UPLOAD_FOLDER = './upload/'
ALLOWED_EXTENSIONS = set(['mp4'])
records = db.Record("records.json")
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
```

```
app.config['MAX_CONTENT_LENGTH'] = 64 * 1024 * 1024  # 64MB

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1] in ALLOWED_EXTENSIONS

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            save_path = os.path.join(app.config['UPLOAD_FOLDER'],filename)
            file.save(save_path)
            print(save_path)
            ret, plate_number = RT_CPR.start_predict(save_path)
            if ret :
                if db.search_plate(records, plate_number):
                    return make_response("Plate database search success", 200)
                    #return redirect(url_for('uploaded_file',filename=filename))
                else:
                    # Plate recognized but not found in database (404 Not Found)
                    return make_response("Plate database search failed", 400)
            else:
                # Plate recognition failure (400 Bad Request or custom code)
                response = make_response("Plate recognition predict failed", 400)
                return response

    return '''
    <!doctype html> 車輛門禁系統
    <title>Upload new inference file</title>
    <h1>Upload new inference File</h1>
    <form action="" method=post enctype=multipart/form-data>
      <p><input type=file name=file>
         <input type=submit value=Upload>
    </form>
    '''
#對上傳檔案進行訪問
@app.route('/upload/<filename>')
def uploaded_file(filename):
    return send_from_directory(app.config['UPLOAD_FOLDER'],filename)

if __name__ == '__main__':
    app.run(host='0.0.0.0',port=5000,debug=True)
```

cprs.py為model1,model2,houghTransform等等的圖像辨識程式函示，還有voting等等，為這個系統很主要的程式

cprs.py

api_key需要改成自己的，需註冊roboflow，api_key只供作業用，不提供給其他用途

```
from inference import get_roboflow_model
import supervision as sv
import cv2
import time
import queue
import numpy as np
import threading
import re

# INIT
class CPRS() :
  def __init__(self):
    self.PLATE_MODEL = get_roboflow_model(model_id="taiwan-license-plate-
recognition-research-tlprr/7",api_key="9d2tqEGBk4q34SiofV0Q")
    self.LETTER_MODEL =get_roboflow_model(model_id="license-bha52-
bssnw/2",api_key="9d2tqEGBk4q34SiofV0Q")
    self.INFERENCE_TIME = 100
    self.image_queue = queue.Queue(maxsize=self.INFERENCE_TIME)
    self.car_plate_queue = queue.Queue(maxsize=2)
    self.plate_counts = dict()
    self.image_event = threading.Event()

  def correct_skew(self, image):
      # 旋轉裁切的車牌
      gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
      edges = cv2.Canny(gray, 50, 150, apertureSize=3)
      lines = cv2.HoughLinesP(edges, 1, np.pi/180, threshold=50, minLineLength=50,
maxLineGap=10)

      if lines is not None:
          angles = []
          for line in lines:
              x1, y1, x2, y2 = line[0]
              angle = np.arctan2(y2 - y1, x2 - x1) * 180. / np.pi
              angles.append(angle)

          median_angle = np.median(angles)

          rows, cols = image.shape[:2]
          M = cv2.getRotationMatrix2D((cols / 2, rows / 2), median_angle, 1)
          corrected_image = cv2.warpAffine(image, M, (cols, rows))
          return corrected_image
      else:
          return image # No lines detected, return original image

  def INFER(self,image):
    model = self.PLATE_MODEL
    carplate_box_result = model.infer(image)
    detections =
sv.Detections.from_inference(carplate_box_result[0].model_dump(by_alias=True,
exclude_none=True))
    # create supervision annotators
    bounding_box_annotator = sv.BoxAnnotator()
```

```
      label_annotator = sv.LabelAnnotator()

      # annotate the image with our inference results
      annotated_image = bounding_box_annotator.annotate(
          scene=image, detections=detections)
      annotated_image = label_annotator.annotate(
          scene=annotated_image, detections=detections)
      detection_label = 0
      for i in range (len(detections.class_id)):
        # Get bounding box coordinates (might be named differently)
        if detections.class_id[i] == 0 and detections.confidence[i] > 0.9 :
          detection_label+=1
          #print(detections.confidence[i])
        # Crop the image based on coordinates
        # upper left corner
          x1 = int(detections.xyxy[i][0])
          y1 = int(detections.xyxy[i][1])
        # lower right corner
          x2 = int(detections.xyxy[i][2])
          y2 = int(detections.xyxy[i][3])
          #print("Cropping car plate image\n")
          cropped_image = image[y1:y2, x1:x2]
          cropped_image = self.correct_skew(cropped_image)
          #cv2.imshow("cropped", cropped_image)
          #cv2.waitKey(1)
          if not self.image_queue.full():
            self.image_queue.put(cropped_image,block=True)

    def filter_formatted_match(self,string):
      # AAA-1234 OR 123-AAA
      pattern1 = r"^[A-Z]{3}-\d{4}$"
      pattern2 = r"^\d{3}-[A-Z]{3}$"
      if re.match(pattern1, string) or re.match(pattern2, string):
        #print("CAR PLATE NUMBER : " + string + " MATCH ")
        if string in self.plate_counts:
          self.plate_counts[string]+=1
        else:
          self.plate_counts[string]=1
        return True
      #print("CAR PLATE NUMBER : " + string + " DOES NOT MATCH")
      return False

    def predict_frame_letter(self):
      model=self.LETTER_MODEL
      img_num = 1
      self.image_event.wait()
      while not self.image_queue.empty() :
        image = self.image_queue.get()
        hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        # 取出 V 通道 (亮度)
        v = hsv[:, :, 2]
        # 二值化 V 通道
        ret, thresh = cv2.threshold(v, 127, 255, cv2.THRESH_BINARY)
        # 將二值化後的 V 通道合併成三通道影像
```

```
      image = cv2.merge([thresh, thresh, thresh])
      carplate_box_result = model.infer(image)
      detections =
sv.Detections.from_inference(carplate_box_result[0].model_dump(by_alias=True,
exclude_none=True))
      bounding_box_annotator = sv.BoxAnnotator()
      label_annotator = sv.LabelAnnotator()
      # annotate the image with our inference results
      annotated_image = bounding_box_annotator.annotate(
        scene=image, detections=detections)
      annotated_image = label_annotator.annotate(
        scene=annotated_image, detections=detections)

      #cv2.imshow("letter crop", annotated_image)
      cv2.waitKey(1)

cv2.imwrite(f"cropped_letter/cropped_letter_box_{img_num}.jpg",annotated_image)
      img_num +=1

      xyxy = detections.xyxy # array of bounding box left corner
      data = detections.data["class_name"]
      # 將數字和座標合併成一個列表
      list_xy = []
      for xy in xyxy :
        list_xy.append(int(xy[0]))
      zipped = zip(list_xy,data) # 合併 字母或數字x座標與對應之字母或數字
      # 根據 x 座標排序
      list_zip = list(zipped)
      #print(f'zipped : {list_zip}')
      # 分離排序後的數字和座標
      car_plate_number = ''.join([l[1] for l in sorted(list_zip, key=lambda x:
x[0])])
      self.filter_formatted_match(car_plate_number)

    # VOTING CAR PLATE
    if len(self.plate_counts) > 0:
      #print(plate_counts)
      sorted_plate_counts = sorted(self.plate_counts.items(), key=lambda x:
x[1],reverse=True)
      #print(sorted_plate_counts)
      voted_plate_number, voted_plate_count = sorted_plate_counts[0]
      print("[CPR_SYS_MSG] VOTE FOR CAR PLATE NUMBER : " + voted_plate_number)
      self.car_plate_queue.put(voted_plate_number)
      self.plate_counts.clear()
    else :
      print("[CPR_SYS_MSG] NOT RECOGNIZE ANY CAR PLATE, PLEASE COME CLOSER")
    self.image_event.clear()

  def process_frame(self,video_path):
    cap = cv2.VideoCapture(video_path)
    input_size = 640
    cap.set(cv2.CAP_PROP_FPS,30)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH,input_size)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT,input_size)
```

```
    infer_time = self.INFERENCE_TIME
    while True:
      start_time = time.time()
      ret, frame = cap.read()
      if not ret:
        break
      self.INFER(frame)
      end_time = time.time()
      fps = 1 / (end_time - start_time)
      framefps = "FPS: {:.2f}".format(fps)
      cv2.rectangle(frame, (10,1), (120,20), (0,0,0), -1)
      cv2.putText(frame, framefps, (15,17), cv2.FONT_HERSHEY_SIMPLEX, 0.6,
(0,255,255),2)
      cv2.imshow('anotated_frame', frame)
      if cv2.waitKey(1) & 0xFF == ord('q'):
          break
      infer_time-=1
      #print(infer_time)
    cap.release()

  def car_plate_result(self):
    if not self.car_plate_queue.empty():
      return True, self.car_plate_queue.get()
    return False, ""

def start_predict(path):
  cprs = CPRS()
  cprs.__init__()
  producer_thread = threading.Thread(target=cprs.process_frame,args=(path,))
  consumer_thread = threading.Thread(target=cprs.predict_frame_letter)
  start_time = time.time()
  producer_thread.start()
  consumer_thread.start()
  producer_thread.join()
  cprs.image_event.set() # consumer 等待 producer通知
  consumer_thread.join()
  end_time = time.time()
  total_time = end_time-start_time
  msg = "[CPR_SYS_MSG] Inference time : {:.2f} seconds".format(total_time)
  print(msg)
  return cprs.car_plate_result()
```

database.py用來讀取json資料庫，並且提供搜尋車牌的功能

database.py

```
import json

class Record():
    def __init__(self,filename):
        with open(filename,"r") as file:
```

```
            data = json.load(file)
        file.close()
        self.data = data
    def dump(self):
        print(self.data)
        return self.data
    def IS_LEGAL(self, plate_number):
        data = self.data["car_plate"]
        for car in data:
            if car["plate_number"] == plate_number and car["registered"] == "YES"
:
                return True
        return False

def search_plate(records,str):
    record = records
    #record.dump()
    #data = record.dump()
    plate_number = str
    if record.IS_LEGAL(plate_number):
        print(plate_number + " is in the list.")
        return True
    print(plate_number + " is not in the list.")
    return False
```

records.json

範例格式如下

```
{
    "car_plate":[
        {
            "plate_number": "NBC-5516",
            "registered": "NO",
            "start_time": 0,
            "end_time": 0,
            "total_time": 0,
            "amounts": 0
        },
        {
            "plate_number": "AQW-0237",
            "registered": "YES",
            "start_time": 0,
            "end_time": 0,
            "total_time": 0,
            "amounts": 0
        },
        {
            "plate_number": "NBX-3388",
            "registered": "YES",
            "start_time": 0,
```

```
            "end_time": 0,
            "total_time": 0,
            "amounts": 0
        }
    ]
}
```