Register for face recognition

Routine Experiment Effect

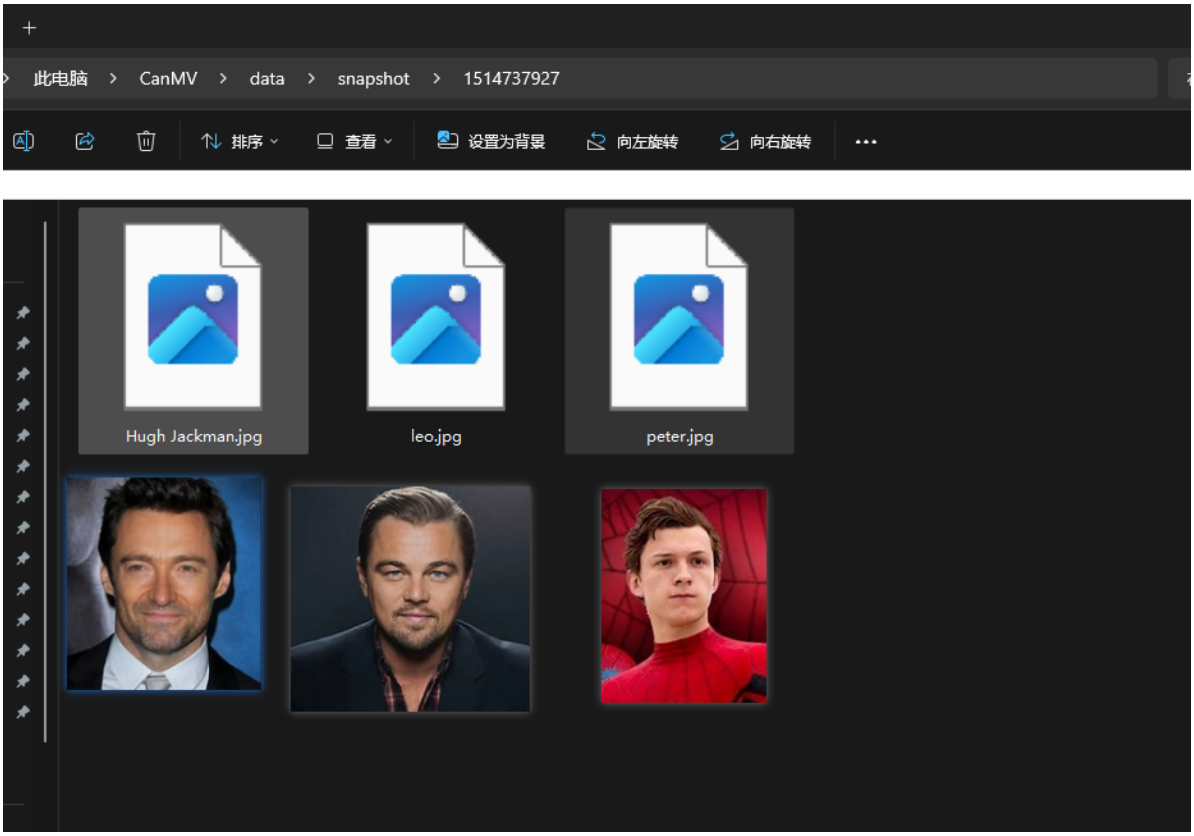
Finally, let's learn how to use K230 to implement face recognition.

Face recognition is divided into two parts: [Registration] and [Recognition]. We need to register the face first.

The registration code provided in the example code is used to register all image files in a directory, so we need to take a photo of the face we want to recognize first.

Here we take the [Camera] provided in the basic routine as an example to take pictures of different faces

Then we find the address where the picture we just took is saved and change the picture file name to the person's name



Next, we open the face registration routine code

Modify the database_img_dir variable value in the exce_demo() method to the path where we save the photos.

In this example, the path is [/data/snapshot/1514737927/]

After modification, we click the run button in the lower left corner and wait for the face to be registered.

```
Start ...
已创建目录: /data/face_databse/1514737927/
/data/snapshot/1514737927/leo.jpg
Success!
/data/snapshot/1514737927/peter.jpg
Success!
/data/snapshot/1514737927/Hugh Jackman.jpg
Success!
人脸注册功能退出
```

After waiting for the program to finish running, we can run the face recognition routine.

Open the face recognition routine in CanMV IDE.

Modify the database_dir variable in excel_demo() to [/data/face_databse/1514737927/]

[1514737927] is the name of the directory where the face photo is located when registering the face

Click the button in the lower left corner to run

Let's take Peter as an example. The registered Peter in the picture will be marked with a green frame, while the unregistered face will be marked with a blue frame.

Why not use pictures directly, but take pictures with K230? Because the original picture and the picture read by the K230 camera will have deviations due to various environmental factors. Using the results of taking pictures with K230 as the training set, the resulting model will be more suitable for the current K230 module for recognition.

> The same is true when training the model yourself. Try to use K230 photos as the training set.

> Serial port output function has been added
>
> After detecting a face, the following serial output will be sent
>
> If it is an unknown face, send:
>
> $x,y,w,h,unknown#
>
> If it is a face that has been registered in the database, send
>
> $x,y,w,h,name,score#
>
> The '$' represents the beginning of the data, and the '#' represents the end of the data.
>
> x, y, w, h are the positions of the face detection frame (resolution is 640*480)
>
> unknown indicates an unknown face
>
> name indicates the name of the recognized face
>
> Score represents the recognition score. The higher the score, the higher the possibility of correct recognition.

Code Explanation

Since there is too much code in this chapter, the complete source code file is not included here, only some key code is left for explanation

Face registration part

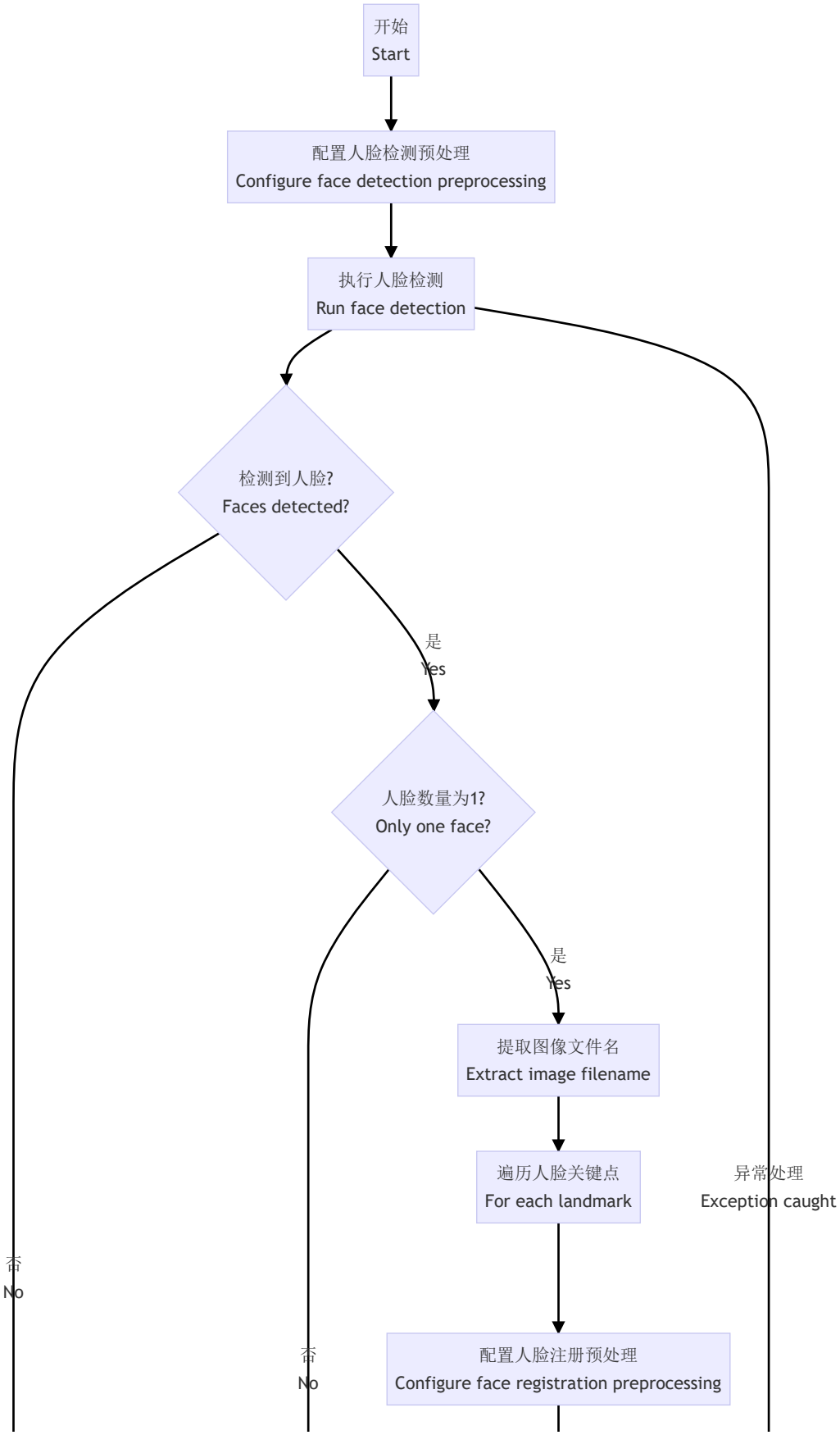
FaceRegistration's run method

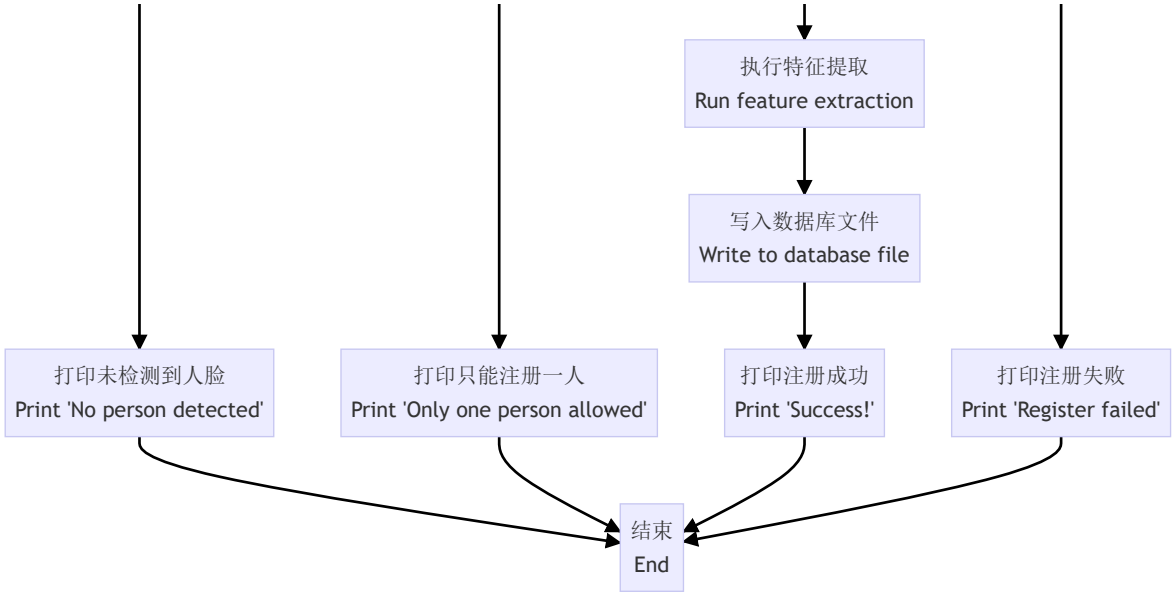
```
    def run(self, input_np, img_file):
        """运行人脸注册流程 / Run face registration process"""
        # 配置人脸检测预处理 / Configure face detection preprocessing
```

```
        self.face_det.config_preprocess(input_image_size=
[input_np.shape[3],input_np.shape[2]])
        # 执行人脸检测 / Perform face detection
        det_boxes, landms = self.face_det.run(input_np)

        try:
            if det_boxes:
                if det_boxes.shape[0] == 1:
                    # 若只检测到一张人脸，进行注册 / If only one face is detected,
proceed with registration
                    db_i_name = img_file.split('.')[0]
                    for landm in landms:
                        # 配置人脸注册预处理 / Configure face registration
preprocessing
                        self.face_reg.config_preprocess(landm, input_image_size=
[input_np.shape[3],input_np.shape[2]])
                        # 执行人脸特征提取 / Perform face feature extraction
                        reg_result = self.face_reg.run(input_np)
                        # 保存特征到数据库 / Save features to database
                        with open(self.database_dir+'{}.bin'.format(db_i_name),
"wb") as file:
                            file.write(reg_result.tobytes())
                            print('Success!')
                else:
                    print('Only one person in a picture when you sign up')
            else:
                print('No person detected')
        except:
            print("Register failed")
```

1. Pretreatment and detection:
    - Configure preprocessing parameters for face detection
    - Performing face detection
2. Test result processing:
    - Check if a face is detected
    - Confirm whether there is only one face
3. Registration Process:
    - Processing image file names
    - Processing facial feature points
    - Configuring registration preprocessing
    - Extract facial features
    - Save to database
4. Exception handling:
    - Handling registration failures
    - Output the corresponding error message

The complete flow chart is as follows:


开始
Start
配置人脸检测预处理
Configure face detection preprocessing
执行人脸检测
Run face detection
检测到人脸?
Faces detected?
是
Yes
否
No
人脸数量为1?
Only one face?
是
Yes
否
No
提取图像文件名
Extract image filename
遍历人脸关键点
For each landmark
配置人脸注册预处理
Configure face registration preprocessing
异常处理
Exception caught



Face recognition part

> If an error occurs:
>
> Traceback (most recent call last): File "", line 531, in File "", line 486, in exce_demo File "", line 349, in **init** File "", line 374, in database_init OSError: [Errno 2] ENOENT
>
> Please check whether the path in the code is correct.

```
    def run(self, input_np):
        """
        运行人脸识别 / Run face recognition
        """
        # 人脸检测 / Face detection
        det_boxes, landms = self.face_det.run(input_np)
        recg_res = []

        # 对每个检测到的人脸进行识别 / Recognize each detected face
        for landm in landms:
            self.face_reg.config_preprocess(landm)
            feature = self.face_reg.run(input_np)
```

```
            res = self.database_search(feature)
            recg_res.append(res)

        return det_boxes, recg_res

    def database_init(self):
        """
        初始化人脸数据库 / Initialize face database
        """
        with ScopedTiming("database_init", self.debug_mode > 1):
            # 读取数据库文件 / Read database files
            db_file_list = os.listdir(self.database_dir)
            for db_file in db_file_list:
                if not db_file.endswith('.bin'):
                    continue
                if self.valid_register_face >= self.max_register_face:
                    break

                valid_index = self.valid_register_face
                full_db_file = self.database_dir + db_file

                # 读取特征数据 / Read feature data
                with open(full_db_file, 'rb') as f:
                    data = f.read()
                feature = np.frombuffer(data, dtype=np.float)
                self.db_data.append(feature)

                # 保存人名 / Save person name
                name = db_file.split('.')[0]
                self.db_name.append(name)
                self.valid_register_face += 1

    def database_reset(self):
        """
        重置数据库 / Reset database
        """
        with ScopedTiming("database_reset", self.debug_mode > 1):
            print("database clearing...")
            self.db_name = []
            self.db_data = []
            self.valid_register_face = 0
            print("database clear Done!")

    def database_search(self, feature):
        """
        在数据库中搜索匹配的人脸 / Search for matching face in database
        """
        with ScopedTiming("database_search", self.debug_mode > 1):
            v_id = -1
            v_score_max = 0.0

            # 特征归一化 / Feature normalization
            feature /= np.linalg.norm(feature)

            # 遍历数据库进行匹配 / Search through database for matches
            for i in range(self.valid_register_face):
                db_feature = self.db_data[i]
                db_feature /= np.linalg.norm(db_feature)
```

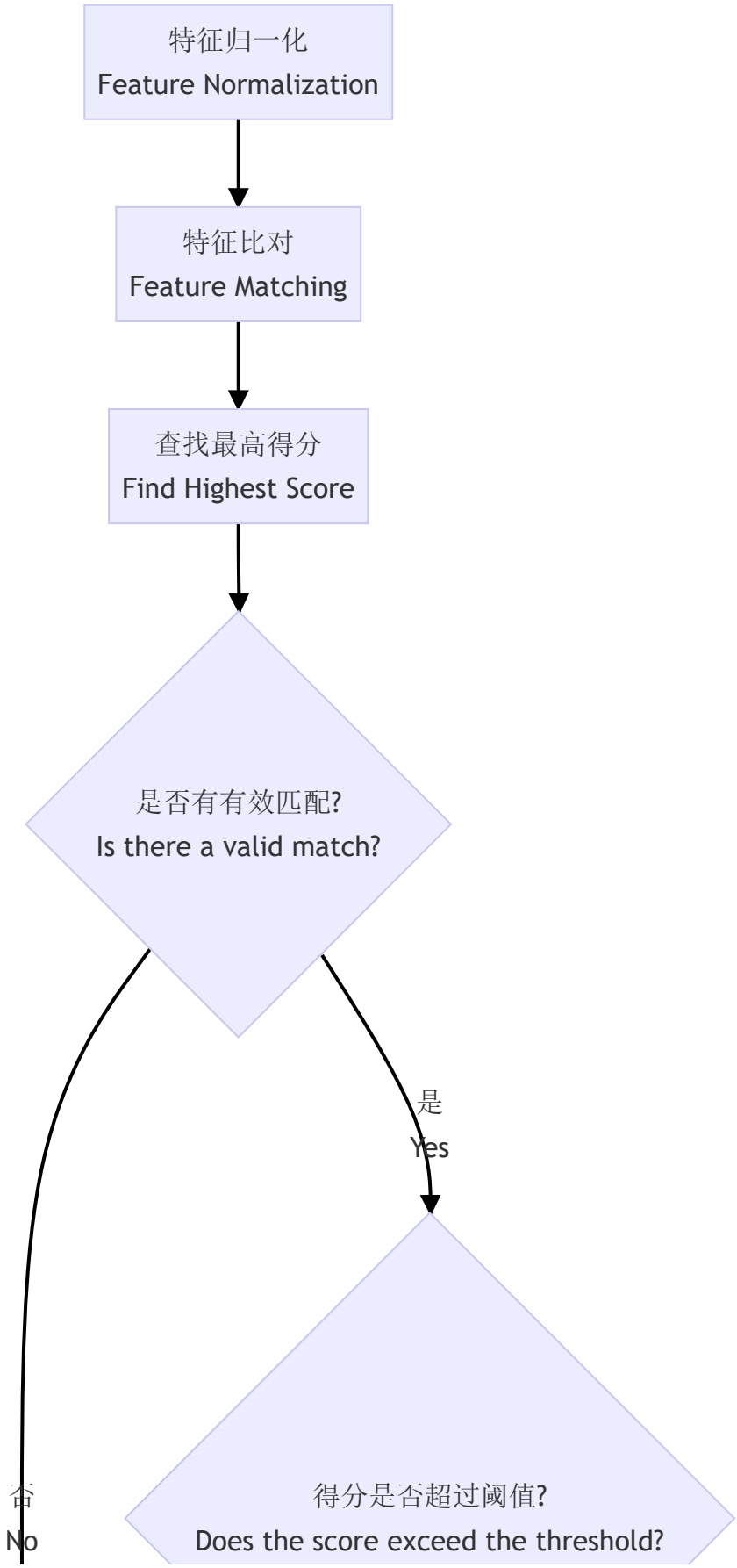
```
                v_score = np.dot(feature, db_feature)/2 + 0.5

                if v_score > v_score_max:
                    v_score_max = v_score
                    v_id = i

            # 返回识别结果 / Return recognition result
            if v_id == -1:
                return 'unknown'
            elif v_score_max < self.face_recognition_threshold:
                return 'unknown'
            else:
                result = 'name: {}, score:{}'.format(self.db_name[v_id],
v_score_max)
                return result

    def draw_result(self, pl, dets, recg_results):
        """
        绘制识别结果 / Draw recognition results
        """
        pl.osd_img.clear()
        if dets:
            for i, det in enumerate(dets):
                # 绘制人脸框 / Draw face box
                x1, y1, w, h = map(lambda x: int(round(x, 0)), det[:4])
                x1 = x1 * self.display_size[0]//self.rgb888p_size[0]
                y1 = y1 * self.display_size[1]//self.rgb888p_size[1]
                w = w * self.display_size[0]//self.rgb888p_size[0]
                h = h * self.display_size[1]//self.rgb888p_size[1]

                # 绘制识别结果 / Draw recognition result
                recg_text = recg_results[i]
                if recg_text == 'unknown':
                    pl.osd_img.draw_rectangle(x1, y1, w, h, color=(255,0,0,255),
thickness=4)
                else:
                    pl.osd_img.draw_rectangle(x1, y1, w, h, color=(255,0,255,0),
thickness=4)

                pl.osd_img.draw_string_advanced(x1, y1, 32, recg_text, color=
(255,255,0,0))
```

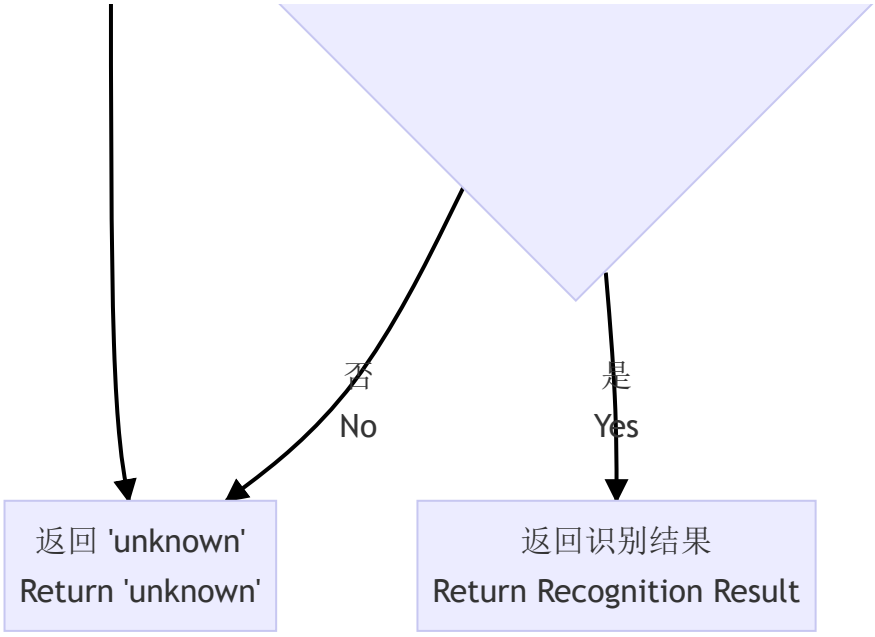
1. Main running process (RunFlow):
    - Performing face detection
    - Processing detected faces
    - Feature extraction and recognition
2. Database initialization (DBInit):
    - Loading signature file
    - Checking file validity
    - Storing feature data
3. Database Search (DBSearch):
    - Feature Normalization
    - Feature comparison
    - Result determination
4. DrawResult:

- Clear old display
- Draw the detection box
- Display recognition results

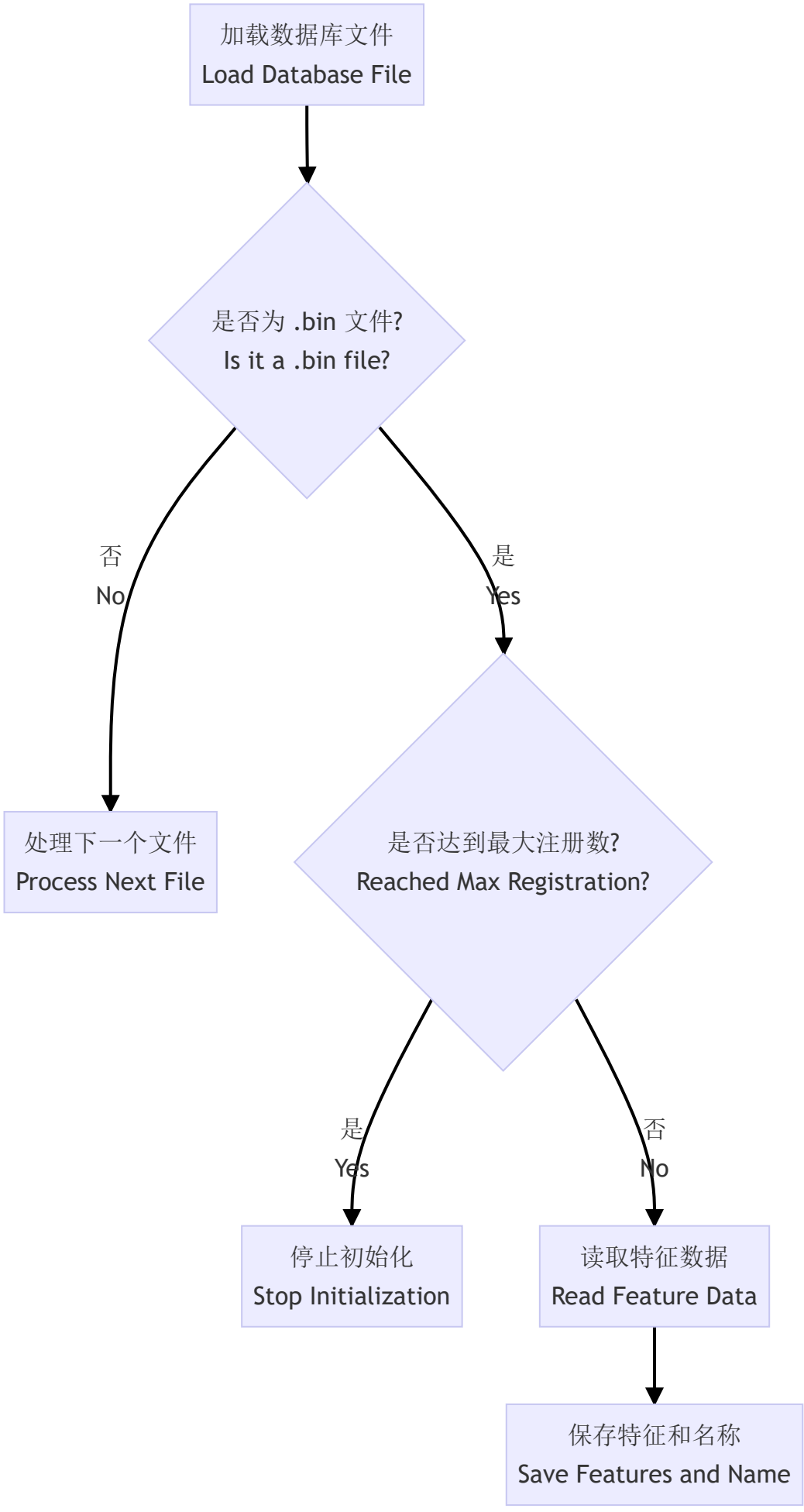
The complete process is shown in the figure

Database Search:

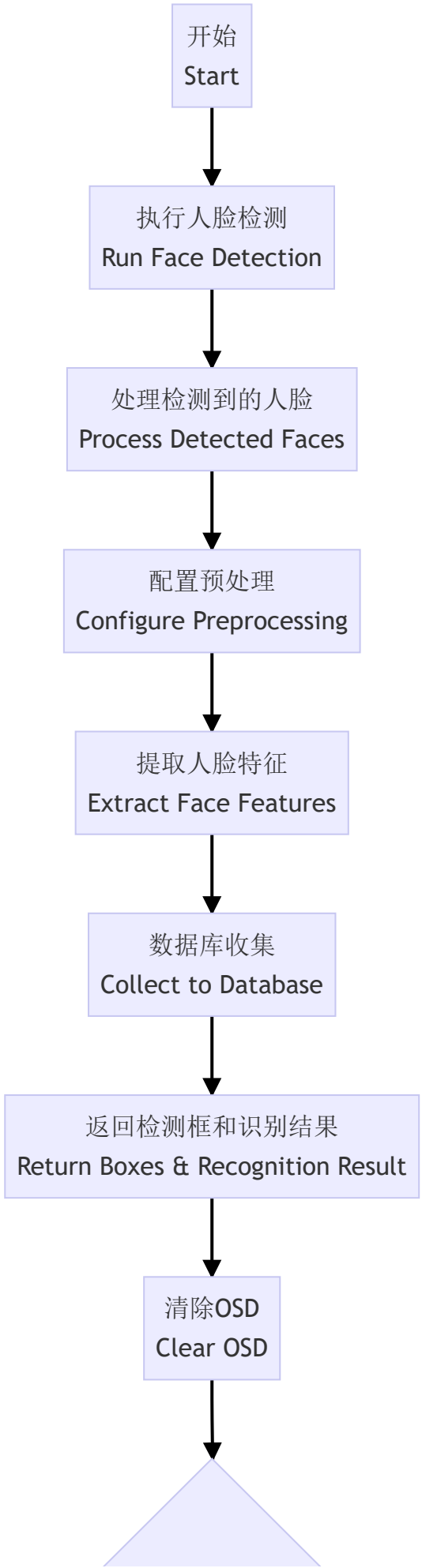


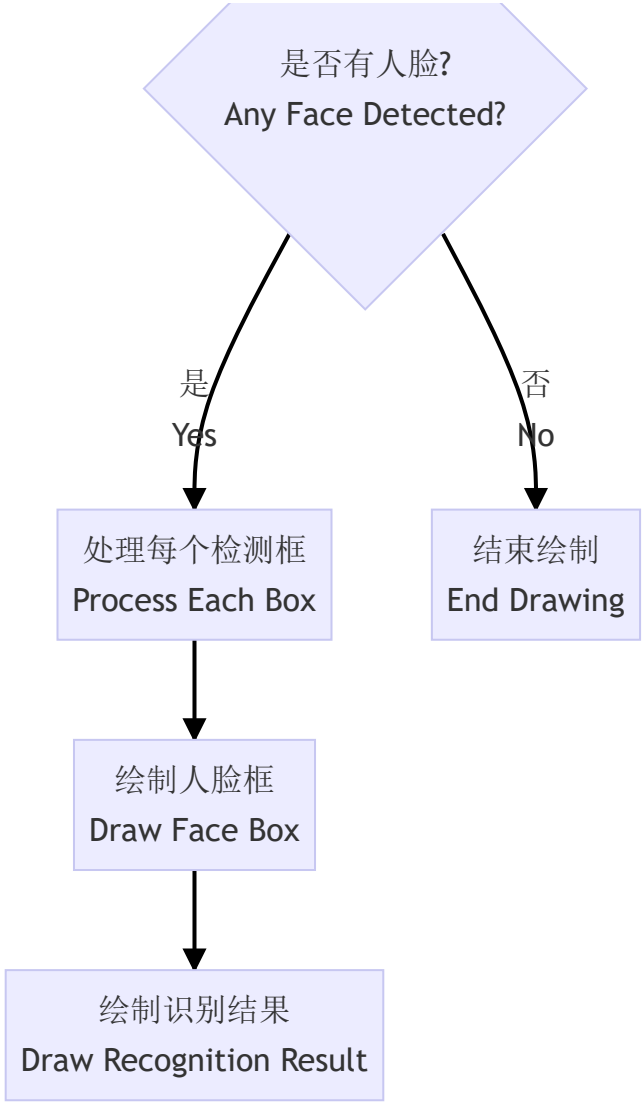


Database Initialization:


加载数据库文件
Load Database File
是否为 .bin 文件?
Is it a .bin file?
否
No
是
Yes
处理下一个文件
Process Next File
是否达到最大注册数?
Reached Max Registration?
是
Yes
否
No
停止初始化
Stop Initialization
读取特征数据
Read Feature Data
保存特征和名称
Save Features and Name

Overall process:





Expand

Create the Directory Section

When you use the K230 os module to access a non-existent directory, an OSError will be thrown, causing the program to terminate.

The following code can detect whether the specified directory exists, and recursively create it if it does not exist

```
def ensure_dir(directory):
    """
    递归创建目录
    (Recursively create directory)
    """
    # 如果目录为空字符串或根目录，直接返回
    # (If directory is empty string or root directory, return directly)
    if not directory or directory == '/':
        return

    # 处理路径分隔符，确保使用标准格式
    # (Process path separators to ensure standard format)
```

```
    directory = directory.rstrip('/')

    try:
        # 尝试获取目录状态，如果目录存在就直接返回
        # (Try to get directory status, if directory exists then return
directly)
        os.stat(directory)
        print(f'目录已存在: {directory}')
        # (Directory already exists: {directory})
        return
    except OSError:
        # 目录不存在，需要创建
        # (Directory does not exist, need to create)

        # 分割路径以获取父目录
        # (Split path to get parent directory)
        if '/' in directory:
            parent = directory[:directory.rindex('/')]
            if parent and parent != directory:  # 避免无限递归
                                                # (Avoid infinite recursion)
                ensure_dir(parent)

        try:
            # 创建目录
            # (Create directory)
            os.mkdir(directory)
            print(f'已创建目录: {directory}')
            # (Directory created: {directory})
        except OSError as e:
            # 可能是并发创建导致的冲突，再次检查目录是否存在
            # (Possible conflict due to concurrent creation, check again if
directory exists)
            try:
                os.stat(directory)
                print(f'目录已被其他进程创建: {directory}')
                # (Directory has been created by another process: {directory})
            except:
                # 如果仍然不存在，则确实出错了
                # (If it still doesn't exist, there is definitely an error)
                print(f'创建目录时出错: {e}')
                # (Error creating directory: {e})
    except Exception as e:
        # 捕获其他可能的异常
        # (Catch other possible exceptions)
        print(f'处理目录时出错: {e}')
        # (Error processing directory: {e})
```