上海交通大学巴黎卓越工程师学院
SJTU Paris Elite Institute of Technology

综合实践项目报告

Practical Project Report

课题名称 Topic：

Machine Vision-based Intelligent Recognition of Building Air Conditioning Interface

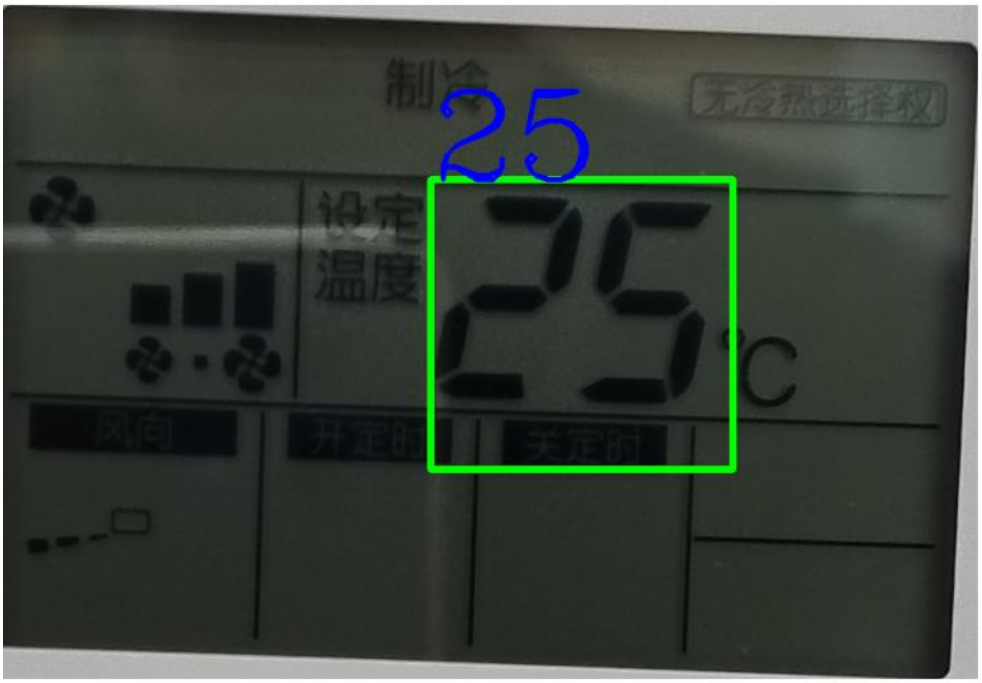


Figure P: Illustration of one project's result

教师姓名 Supervisor：**Helin Gong**

学生姓名 Student：**Babic Marko** 学号 Student ID：**J12426099007**

专业名称 Major：**Computer Science - Information Engineer**

学院(系)：**巴黎卓越工程师学院 SPEIT**

Table of Contents

课题名称 Topic

Purpose of the project

The purpose of this project is to establish a programmed digital recognition model capable of reading the digits from an air-conditioner (annotated **AC**) interface. More precisely, from our own pictures taken by a smartphone.

In fact, we took two hundreds of pictures with a smartphone of AC's interfaces in the Shanghai Jiao Tong University (annotated **SJTU**) campus. Our final goal is to be able to read as many pictures as possible even if they have visual problem with highest accuracy and robustness. Indeed, some images can be taken with different angle, with different brightness, with reflections, camera settings, perspective distortions, glare obscuring digits, digit occlusion by dust or other objects, or resolution and scaling issues.

Significance of the project

The main significance of our project is to become a relevant tool for building maintenance. For instance, a building containing different AC in many rooms.
Our project can be used by people that manages the temperatures of each room. Indeed, a picture could be taken from an AC so as to put it directly in a database. Then this database could be displayed on a monitoring tool that will alert the user when some rooms have abnormal temperatures. Especially for small buildings, rather than investing in linking all AC to a database online or in an internal network. The pictures will help to check regularly the temperatures and regroup it in a cheaper and in a simpler way.

Furthermore, if an abnormal temperature is detected, an action will be taken so as to prevent energy loss, heat loss (that could make money loss) or to get the room more comfortable.

Overview on the actual research situation

After establishing 4 different models, we have a robust image processing before reading the image. The image can be cropped even with brightness problems or angle problems. The last accuracy measured on model 4 is the same as the model 2. However, the execution time is hugely higher.

We started also to think about the whole project purpose and topic. After finishing to learn the basics of machine learning in another course this semester. Experiment this knowledge concretely here could have been relevant. Indeed, we could have done a supervised learning on these 205 pictures in order to establish a model capable of reading new images that we would took.

Moreover, we started to think if it was possible to do this detection without taking a photo. In fact, like scanning a QR code on Alipay for instance, just by activating the camera of the phone. This could be the next step of this project in the future, making it more ergonomic.

课题研究内容 Research content：

Taking a sample of pictures

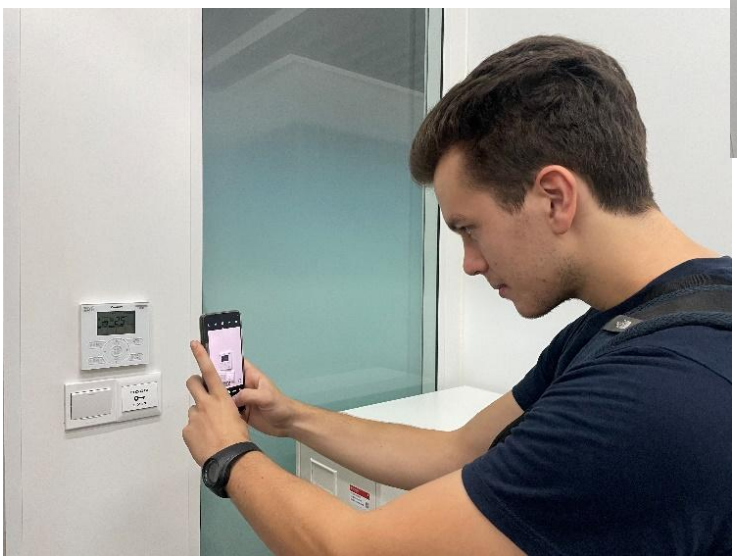
Here is the first main task, we needed to take certain amount of pictures in order to experiment our model.

In our project, these are the 2 main criteria:

- Taking a certain amount of pictures around 200.
- Taking our **own** photos without using a bank of pictures made by someone else already prepared

To take our photos, we used our own smartphone.
More precisely, a **Motorola G84 5G**.

To get that much of pictures, we visited different schools on the SJTU campus where we found different type of AC interfaces from different brands.
When we found an AC, we took 4-5 photos of it. One photo in front of the AC, 2-3 with different angles, and one like the first one but zoomed out.



Our photos are stored in my personal computer. Moreover, I made a copy of them on a SD card and on a cloud drive in any case.

An important point to notice is that the image can be blurred or can get quality issues for the reason that the smartphone and the photographer are not perfect.

Figure RC1: Picture of me taking a photo of AC

The whole task went as planned, we managed to get **205 pictures.** The space used on the computer is 507MB.

Establishing a first model

To establish a first model recognition, we had first to select a programming language. The supervisor told me that it could be done with the most common language like JAVA, C, C++ or Python. Consequently, I decided to use Python for different reasons:

- ✓ **Extensive support libraries** making it suitable for scientific and data-related applications.
- ✓ **Open source and large active community base**
- ✓ **Interpreted language**: which allows easier debugging and code development.

- ✓ **Versatile, easy to read, learn, and write**
- ✓ **High-level language** that abstracts low-level details, making it more user-friendly.
- ✓ **Ideal for prototypes:** Python's concise syntax allows to prototype applications quickly with less code.

However, it is important to notice that Python is not the best language in term of performance. For instance, this language is really far behind the other main languages in term of energy consumption, execution time and memory consumption.

	Energy
(c) C	1.00
(c) Rust	1.03
(c) C++	1.34
(c) Ada	1.70
(v) Java	1.98
(c) Pascal	2.14
(c) Chapel	2.18
(v) Lisp	2.27
(c) Ocaml	2.40
(c) Fortran	2.52
(c) Swift	2.79
(c) Haskell	3.10
(v) C#	3.14
(c) Go	3.23
(i) Dart	3.83
(v) F#	4.13
(i) JavaScript	4.45
(v) Racket	7.91
(i) TypeScript	21.50
(i) Hack	24.02
(i) PHP	29.30
(v) Erlang	42.23
(i) Lua	45.98
(i) Jruby	46.54
(i) Ruby	69.91
(i) Python	75.88
(i) Perl	79.58

	Time
(c) C	1.00
(c) Rust	1.04
(c) C++	1.56
(c) Ada	1.85
(v) Java	1.89
(c) Chapel	2.14
(c) Go	2.83
(c) Pascal	3.02
(c) Ocaml	3.09
(v) C#	3.14
(v) Lisp	3.40
(c) Haskell	3.55
(c) Swift	4.20
(c) Fortran	4.20
(v) F#	6.30
(i) JavaScript	6.52
(i) Dart	6.67
(v) Racket	11.27
(i) Hack	26.99
(i) PHP	27.64
(v) Erlang	36.71
(i) Jruby	43.44
(i) TypeScript	46.20
(i) Ruby	59.34
(i) Perl	65.79
(i) Python	71.90
(i) Lua	82.91

	Mb
(c) Pascal	1.00
(c) Go	1.05
(c) C	1.17
(c) Fortran	1.24
(c) C++	1.34
(c) Ada	1.47
(c) Rust	1.54
(v) Lisp	1.92
(c) Haskell	2.45
(i) PHP	2.57
(c) Swift	2.71
(i) Python	2.80
(c) Ocaml	2.82
(v) C#	2.85
(i) Hack	3.34
(v) Racket	3.52
(i) Ruby	3.97
(c) Chapel	4.00
(v) F#	4.25
(i) JavaScript	4.59
(i) TypeScript	4.69
(v) Java	6.01
(i) Perl	6.62
(i) Lua	6.72
(v) Erlang	7.20
(i) Dart	8.64
(i) Jruby	19.84

Figure RC2: performance criteria between different programming languages

To use python, we decided to take **Pycharm** (developed by JetBrains) as our python Integrated Development Environment(IDE) with a python version 3.12.

Moreover, we started to search which Python libraries would help to read characters on pictures:

- **pytesseract**: A Python wrapper for Tesseract-OCR Engine, which is highly effective for digit recognition.
- **EasyOCR**: A user-friendly library that can recognize digits and other characters from images with ease.
- **PaddleOCR**: A Python library developed by Baidu that is capable of recognizing digits and text from images with high accuracy.

After trying each one, we decided to use **EasyOCR**. This library was easier to test and to install than the other. Another reason comes from the fact that it does not need a heavy syntax to detect text on an image.

While testing this tool, we noticed that the process of reading is longer on bigger image. In addition, it reads every characters on the screen, that means that the number are not isolated.

For this reason, it is necessary to preprocess a picture before reading it. Indeed, cropping will reduce the time of reading and reduce the risk of reading digits outside the AC interface. But also finding a filter that would avoid reading other elements of the screen than the temperature would help to create our first model.

Here are the different libraries related to image processing we started to use to establish this first model:

- **cv2**: An open-source computer vision library that provides a wide range of image and video processing functions.
- **Numpy**: A fundamental package for scientific computing with Python, providing support for arrays, matrices, and a variety of mathematical functions.
- **Matplotlib**: A plotting library used for creating static, animated, and interactive visualizations in Python.
- **Imutils**: a series of convenience functions to make basic image processing functions such as resizing, rotating, and converting between different image formats.

Here are the different steps of our first model, click here to see the related code:

1. Load an image from our files
2. Pre-process the image with a gray filter, a blur filter then a cropping
3. Initialize EasyOCR reader to read text on our pre-processed image using GPU and English language
4. Manage the reading of the Celsius symbol. We suppose that AC should not get a temperature higher than a 99°C. Therefore, if we detect 3 digits read, we select only the first ones.
5. Draw bounding boxes on detected digit and the digits detected.
6. Return a string containing the digit recognized.

When we use the **easyocr.Reader** to read text from an image, the **readtext** method returns a structured output that contains detailed information about each detected text element within the image. Here's a more detailed breakdown of the structure returned:

1. **List of Dictionaries**: It is a list where each element represents a separate text region detected in the image.
2. **Dictionary Components**:

   - **'text'**: This key holds the actual text string detected within the bounding box.
   - **'bbox'**: This key contains the coordinates of the bounding box that encloses the detected text. The coordinates are usually in the format of **(x, y, width, height),** where x and y are the coordinates of the top-left corner of the bounding box, and width and height are the dimensions of the box.
   - **'confidence'**: This key provides a confidence score indicating how certain the OCR engine is about the accuracy of the detected text. The score is a **value between 0 and 1**, with higher values indicating higher confidence.

{"text": "Detected Text String",
"bbox": [x, y, width, height],
"confidence": confidence_value} is the form of this dictionary.

Nevertheless, it did not work on other images. We can clearly see it on the figure RR2 where the image processing failed. Making then impossible to read digits. The variable containing the information on was consequently empty and we did not create exception for that case. Which finally led to an error.

That is reason why we need to improve this model in order to read at least 5 images.

Establishing a second model

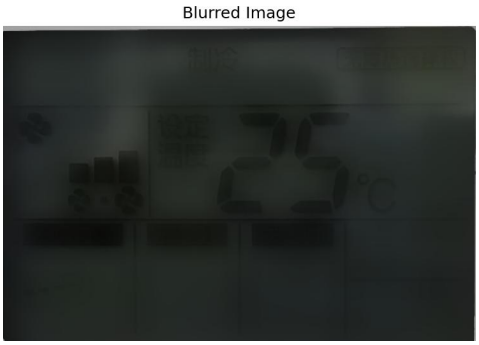
The main issues that came from the first model were related to the pre-processing aspect. Thus, to improve this, we separated the cropping part in another file name **crop.py** and the reading part in **readImage.py**.

Cropping:

The main goal was to crop more pictures by separating this function we observed that it really did not work in the first model on different images. To improve the cropping, we decided to use the fact that the screens of AC's interfaces are usually gray/green. By creating a mask with these colors, the cv2 library permitted to find the contours of the screen based on it.

Thanks to this, we managed to crop more images than before. For example, the image that failed to be processed in the first model can now be cropped correctly:

After upgrading the cropping, we can think that we just need to read the image. However, trying to do so can lead to a failure too. For instance, some other symbols/characters may be read even small details (see Figure RR4)



To correct this problem, the strategy was to apply a blur so as to make the little details less readable and the temperature still distinguishable.

Here is what we done, we applied a bilateral filter so as to make the image more pixelated with a high strength. Then we tried to read the image. If we cannot read the image, we apply half as much blur then we read it again with EasyOCR.

Figure RC3: Example of a blurred image

Thanks to just this 2 processes, without other image processing, we can now read at least 5 images. We managed to read pictures with the following number for instance: **1,2,5,58,74,142,177,204**. We can illustrate the process with picture n°5 in Figure RR5.

In that case, the cropping succeeded. Then we can see that the blur applied is too strong so the digit cannot be detected (step 4). We apply then less blur and the digit are finally found.

Another issue that happened was the incorrect reading of the digit 5. Sometimes it was read as a **"S".** We managed this problem by replacing the character S by a **"5"** for the reason that the S mostly look to this number.

Establishing a testing program

Because when we modify a method we want to see if the image that worked on the previous model still works on them. Rather than testing one by one, it would be faster and better to have a program that test our model on all images. Like this, we would be able to compare our models.

```
Execution time: 2965.750782728195 seconds
This model read correctly: 14/205
```

Figure RC4: Example of the testing program result on model 2

How did we establish our testing program? First we created a directory named "Test" that contains different models adapted for the testing.

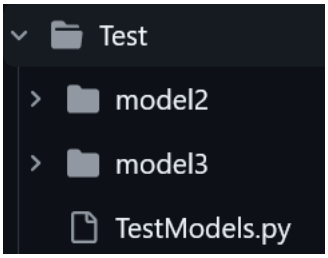


Figure RC5: Testing directories

Indeed, the model usually display all the steps on 1 image and does not necessarily return a result. If we want to test one of our models on all the 205 images, we need to adapt these models in order to just return the value read without displaying any steps.

After doing this part, we created the file **TestModels.py**. We created a list (named labels) containing the value of all the pictures in string. For instance, the element 0 contains in this list is the string **"25".** If there is nothing on the AC's screen, the element related to this picture contains the string **"empty"**.

```
labels=[
    "25","25","25","25","25","24","24","24","24","24",
    "24","23","23","23","24","25","24","25","25","24",
    "26","26","26","26","26","26","25","25","26","26",
    "26","26","20","20","20","20","22","22","22","22",
    "25","25","25","25","23","23","23","23","26","26",
    "26","26","26","25","25","25","23","23","23","23",
    "21","21","21","21","21","26","26","26","26","26",
    "26","26","26","26","26","empty","empty","empty","empty"
```

Figure RC6: A portion of the list of labels

Because there are only 205 pictures, we filled this list one by one. By doing this, we can now directly compare the result read on a picture and compare it with the correct one in this list. It can also be uncertainties about this list because we filled it manually.

Now we are able to do so iteratively and display relevant measures such as:

- Number of pictures read correctly on the total of the 205 images
- Execution time (sometimes it is very long) thanks to the library **time**
- Note which images we could not read with this model

Establishing a third model

After testing model 2, we analyzed different image that could not be read. We noticed that a lot are not cropped correctly. We suppose that if we improve cropping, the execution time will reduce and the accuracy may increase. Therefore, in this model 3, we only focus on upgrading cropping.

Here are suppositions on the cropping fails:

- Angle of the picture
- Brightness that makes the screen harder to be detected by the HSV mask
- Elements in the background that have a color more near to the mask and that are rectangular
- Quality of the picture

To test different solutions, we grouped some photos per suppositions and we tried different new preprocess before the cropping. Finally, here are the 3 processes that improved the cropping on previous pictures:

- Converting the image to **grayscale** with the open-CV constant cv2.COLOR_BGR2GRAY
- Applying a **gamma correction** to reduce the problems related to the brightness of the picture
- Using an **adaptive thresholding** on the image, which is particularly useful for converting a grayscale image to a binary image. The main objective is to handle images with varying illumination conditions, where a global threshold is suitable only for one specific light situation.

Hypothesis	Angle	Brightness and Color	Objects in the background	Poor quality
Image n°	4	11,26,31	6,7,8,9,10,28	27,29

Figure RC7: Different case of problems on different samples of images

An important thing to say is that we perform 3 different adaptive thresholding. In fact, by testing different values of the parameters in the cv2.adaptiveThreshold function. Some photos can be read with a "block_sizes" parameter of 21, some with one of 51 and others with 81. After cropping using these 3 different thresholding, we take the picture cropped with the biggest width as the real cropped image.

The goal of the "block sizes" parameter is to define the size of the local neighborhood around each pixel in the image, which is used to calculate a threshold value adaptively.
A smaller "block_sizes" results in a threshold that are more adaptive, which can be useful for capturing edges and details in the image. A larger "block_sizes" smooths out more of the image, making it more suitable for general thresholding where the goal is to separate the foreground from the background.

Finally, we used our new testing program on this new model. We observed 2 main results (Figure RR6,7):

- The accuracy reduced: some images lost quality after cropping.
- The execution time increased: the new pre-processes on each image is longer, especially after the 3 adaptive thresholds realized.

Establishing a fourth model

Now that the cropping is upgraded, the part to improve is the image processing after the image is cropped. Before the reading, this image processing will help to reduce a lot of noise and visual problems on the picture. We created a dedicated function in a new file "imageProcessing.py". We searched different way to clear the image on different images and with different values on their parameters.

Image Processing
Here are the different processed done one after another on the cropped image:

I. **Grayscale conversion**: converts a color image (with three color channels: blue, green, and red) to a grayscale image (with only one channel representing the intensity of light). This is done to simplify the image data for further processing, as it reduces the amount of data to work with and can sometimes improve the performance of image analysis algorithms. Each pixel value in the grayscale image is an unsigned 8-bit integer, ranging from 0 to 255, where 0 represents black and 255 represents white.

II. **Histogram Equalization:** enhance the contrast of the image, making it easier to distinguish between different regions of the image. It spreads out the most frequent intensity values so that the image has a more uniform distribution of pixel intensities. This is achieved by applying a non-linear mapping to the pixel intensities. By doing this, areas of the image that were originally dark and had similar pixel values can be differentiated more clearly.

III. **Adaptive Thresholding:** useful when the image has varying illumination conditions across different regions, as it can adapt the threshold value locally to account for these variations. This can result in better segmentation of objects in the image compared to a global thresholding approach. The result of this function call is a binary image.

IV. **Morphological dilatation:** expands the boundaries of the foreground (white) regions in an image. It does this by adding pixels to the boundaries of these regions. The effect of dilation is controlled by the kernel that we select size.

Figure RC8: example of dilation

V. **Morphological closing:** a two-step process that first performs an erosion which removes pixels on the boundaries of the foreground regions. This is done by comparing each pixel and its neighbors with the kernel. If the kernel does not completely overlap with the foreground region, the central pixel is turned to black (set to 0). After erosion, dilation is applied. We also control the effect by the kernel.

Figure RC9: example of closing

VI. **Median blur:** a non-linear smoothing filter that replaces each pixel value with the median value of its neighbors, which is effective at removing noise while preserving edges.

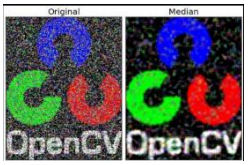


Figure RC10: example of median blur

VII. **Second Morphological closing + VIII. Second Median blur**

IX. Reprocessing image in case of a reading failure: If the previous processes fail. We redo the part V and VI with different parameters, then we apply a morphological operation called "**rectification**". It refers to the process of correcting or adjusting an image to remove distortions or to align it with a certain standard or coordinate system.

Reading conditions

To make the reading easier, we verify different things:

- If the reader from easyOCR read nothing
- If it read a string longer than 3 digits: we suppose that ACs in buildings are rarely under 10°C
- If there is one of the 2 characters that is not a digit

If one of this condition is fulfilled, then we redo the reading on the image but with different parameters used in image processing part IX.

Problems and improvements

The main problem is the tool used to read the image, easyOCR cannot be configured to read only digits and cannot be configured in details. Furthermore, this reader is not accurate when there a lot of details on the image. Even a triangular shape can be read as a digit for instance.

Another problem is the execution time of our last model, this could be done by changing the model or using a more powerful computer. A relevant tool could be Google Colab that gives free access to computing resources more powerful online.

研究方法和研究思路（技术路线）Research methods and clues：

Supervisor's clues

The teacher supervisor advised me to use python as my programming language in this project. Moreover, during all along the project, he guided me through different steps so as to improve the model progressively. I learnt from him that the cropping of the image would help to reduce the calculations. We had also few meetings to discuss about results and the next steps, usually, it was done online.

Official Documentation reference

Every library on python have a dedicated documentation on its function in detailed. It is an essential tool to understand the different parameters of a function, the different options, what are the limits of a parameter and its type for instance.

Stack Overflow

Stack Overflow is a question and answer website for programmers. It was created in 2008 by Jeff Atwood and Joel Spolsky. The site serves as a platform where developers can collaborate, share knowledge, and learn from each other. This website site was a good help when I had a problem or an error. Here I could see other people with the same issue and see what are the solutions brought by other programmers that worked or they tried.

Debugging

Like in every programming project, we had to debug the code by isolating the portion of the code that creates issues. By printing, displaying variables or processes, we could understand which section of the program had to be corrected and how. This step led to create a specific code for cropping that helped me to focus only on this part.

Videos

Especially for finding a library that reads text. A YouTube video helped me to use the "**EasyOCR**" library and to display examples of use for other libraries for reading image such as "**pytesseract**".

GitHub

GitHub is a web tool for storing programming project containing many different type of files. It is reliable for using the code on different computer rather than sending it by email. Furthermore, after each "push" (upload/update of our code), we have an history that can be used in case we need an older version of our code.

Websites

A lot of different websites related to image recognition use cropping. For example, the plate recognition on cars needs to crop the plate before reading it. In order to do so, image processing functions from the CV2 library are also used to isolate the plate in different cases. The problem here is that the type of mask used was not relevant for our case because it was a white one instead of a gray/green that vary even more with different brightness variations. In addition, it was usually not robust to image with poor quality or with all the visual problems. Other websites helped me also to understand how different functions from open-CV work. Especially, the morphological operations and the smoothing operations.

研究结果 Research results：

Model's outputs

Model 1

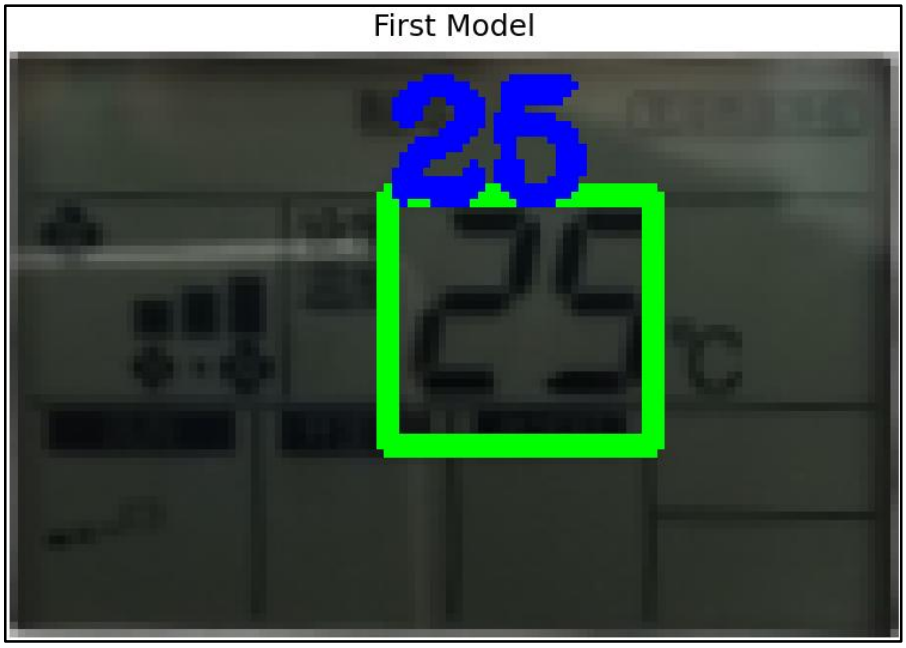


Figure RR1: First model successful result on picture n°1

Figure RR2: First model's failure on picture n°2

Model 2

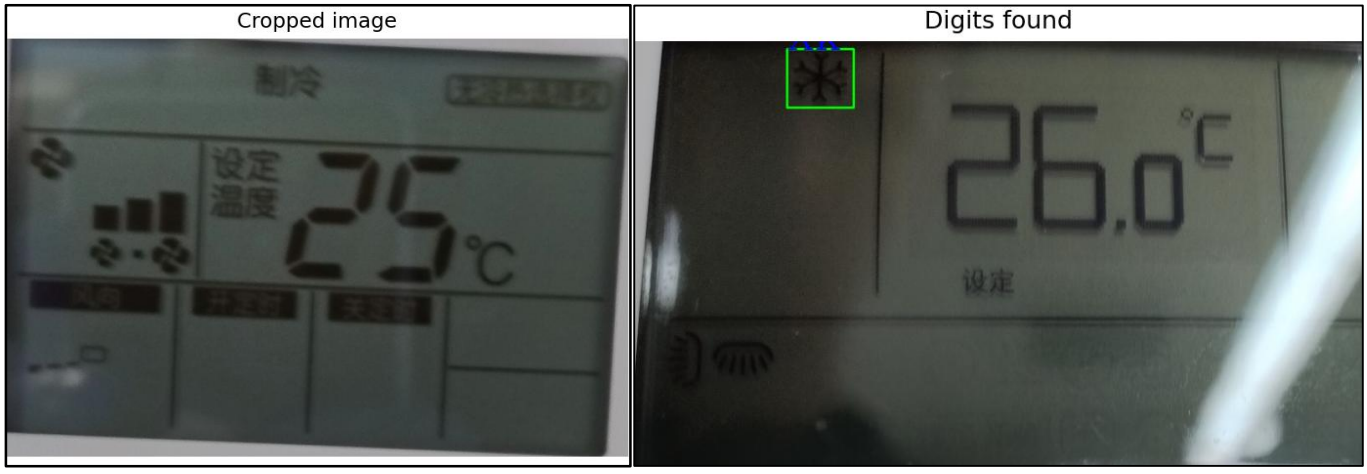


Figure RR3: Result of cropping the picture n°2 (on the left)

Figure RR4: Failure of reading a cropped image n°204 (on the right)

Even if the cropping succeeded, the reading may fail. On RR4, model 2 read **"*".**

After the model 2 was organized and divided correctly, here are the different steps of it on figure RR5:

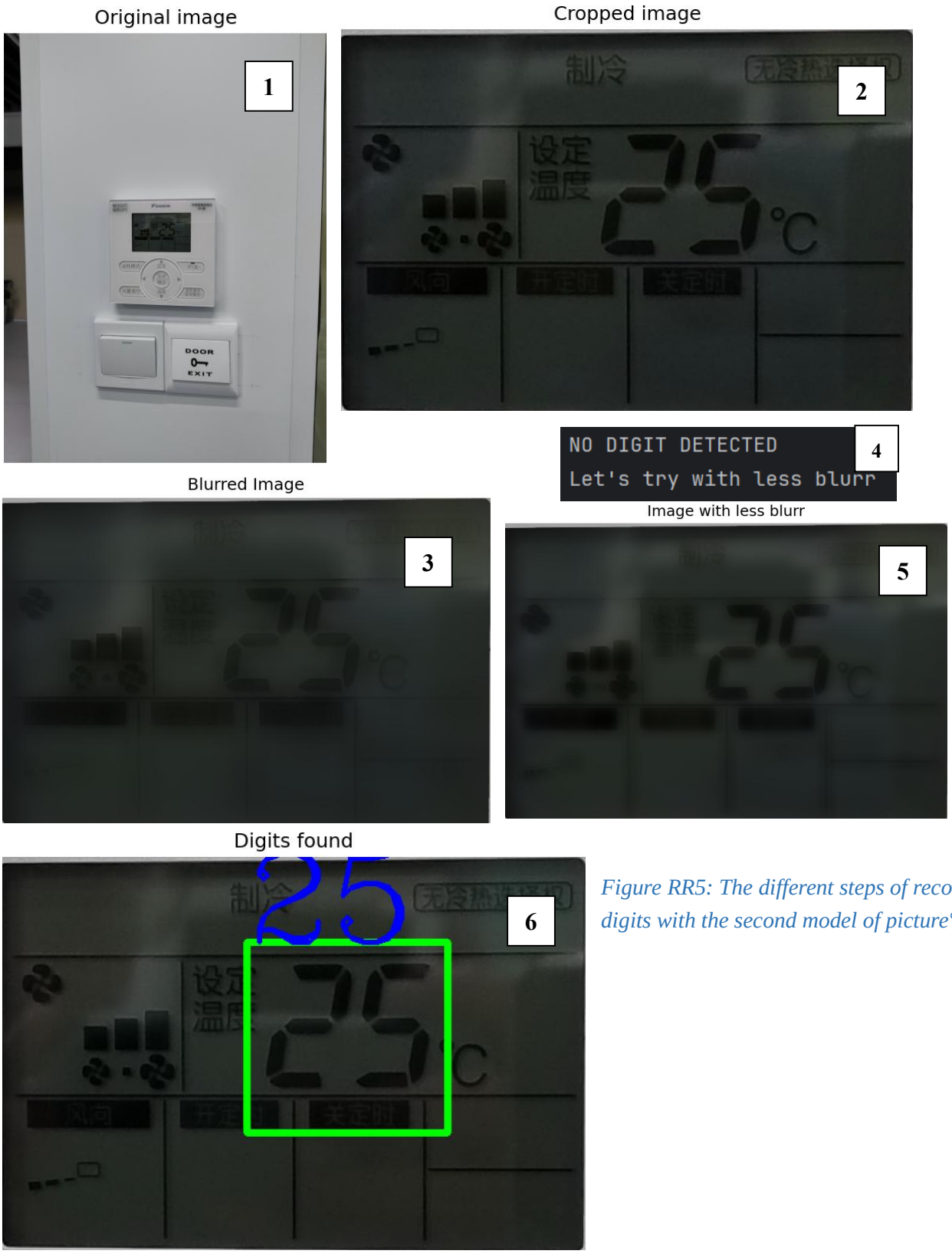


Figure RR5: The different steps of recognizing the digits with the second model of picture°5

Model 3

The most interesting examples are image 7 and 11. In fact, the image 7 helps to see the problems related to objects in the background that could be cropped instead of the AC. Concerning the image 11, the problem was related to the brightness and the quality of the image that is poor.

Original image

crop.py from model 2

Cropped image

crop.py from model 3

Gamma Corrected Image

Largest Cropped image by Width

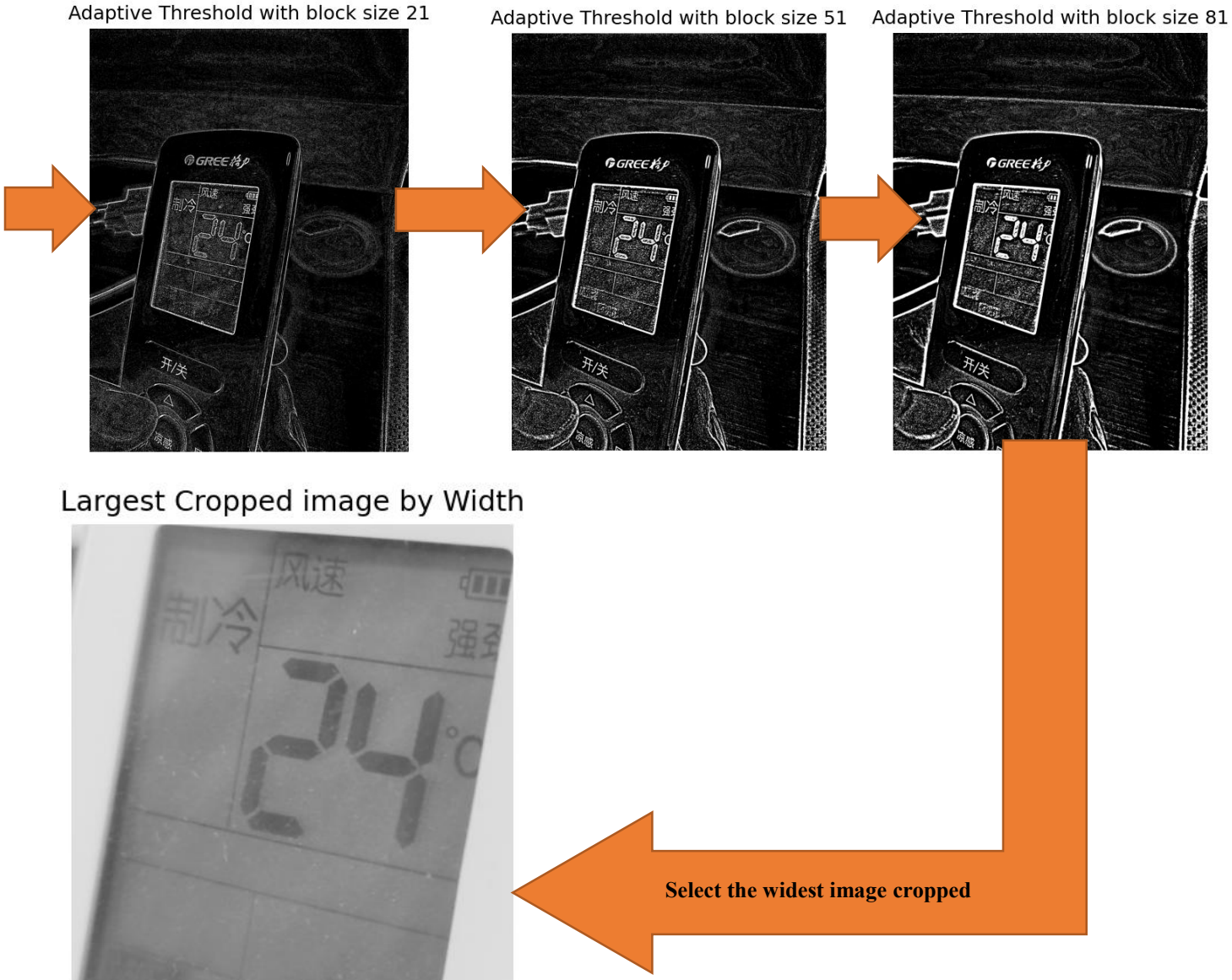


Figure RR6: cropping process on the image 7 from model 2 and 3

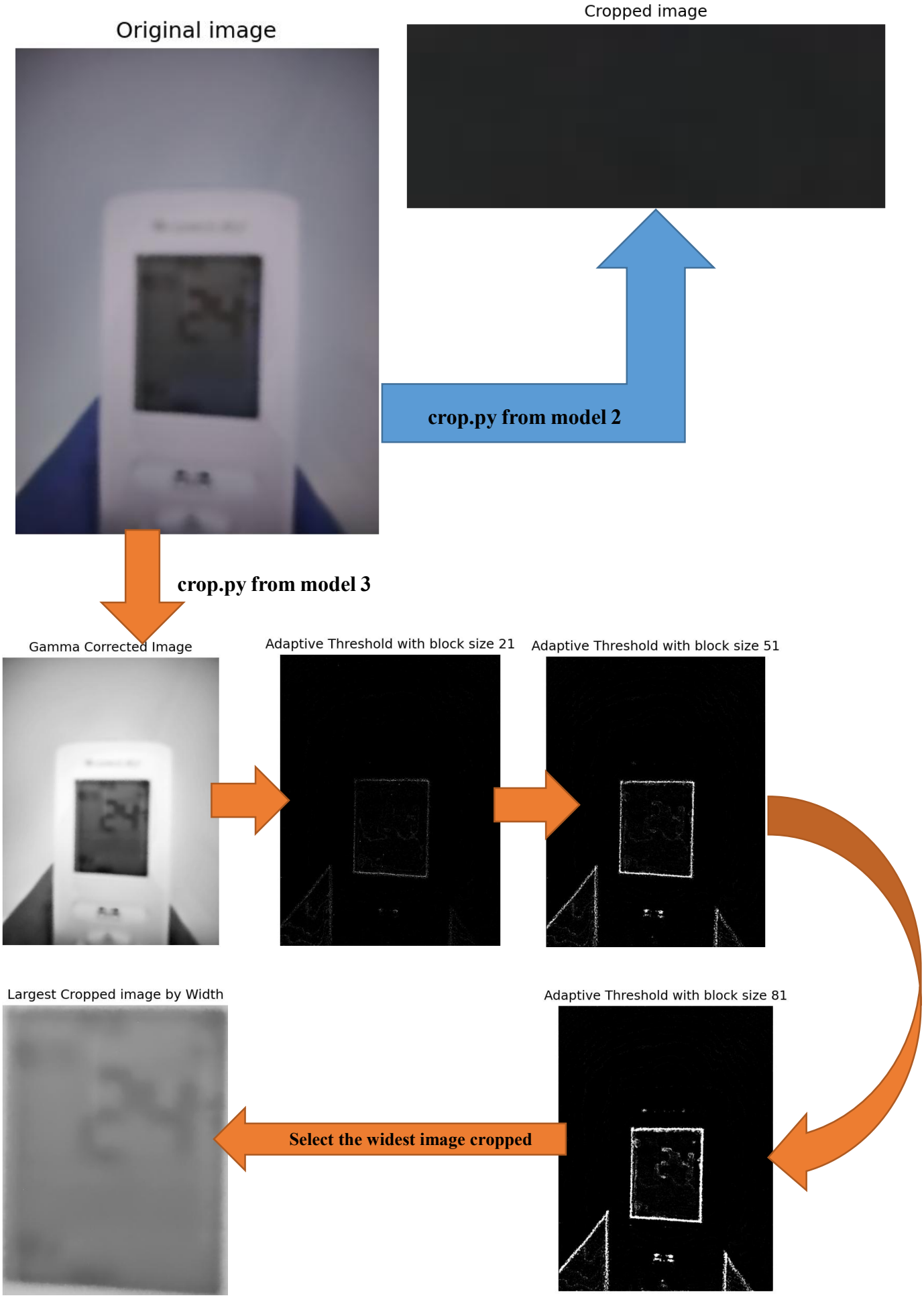


Figure RR7: cropping process on the image 11 from model 2 and 3

Model 4

Let us now observe the different image processes that happens on image n°10(from left to right):

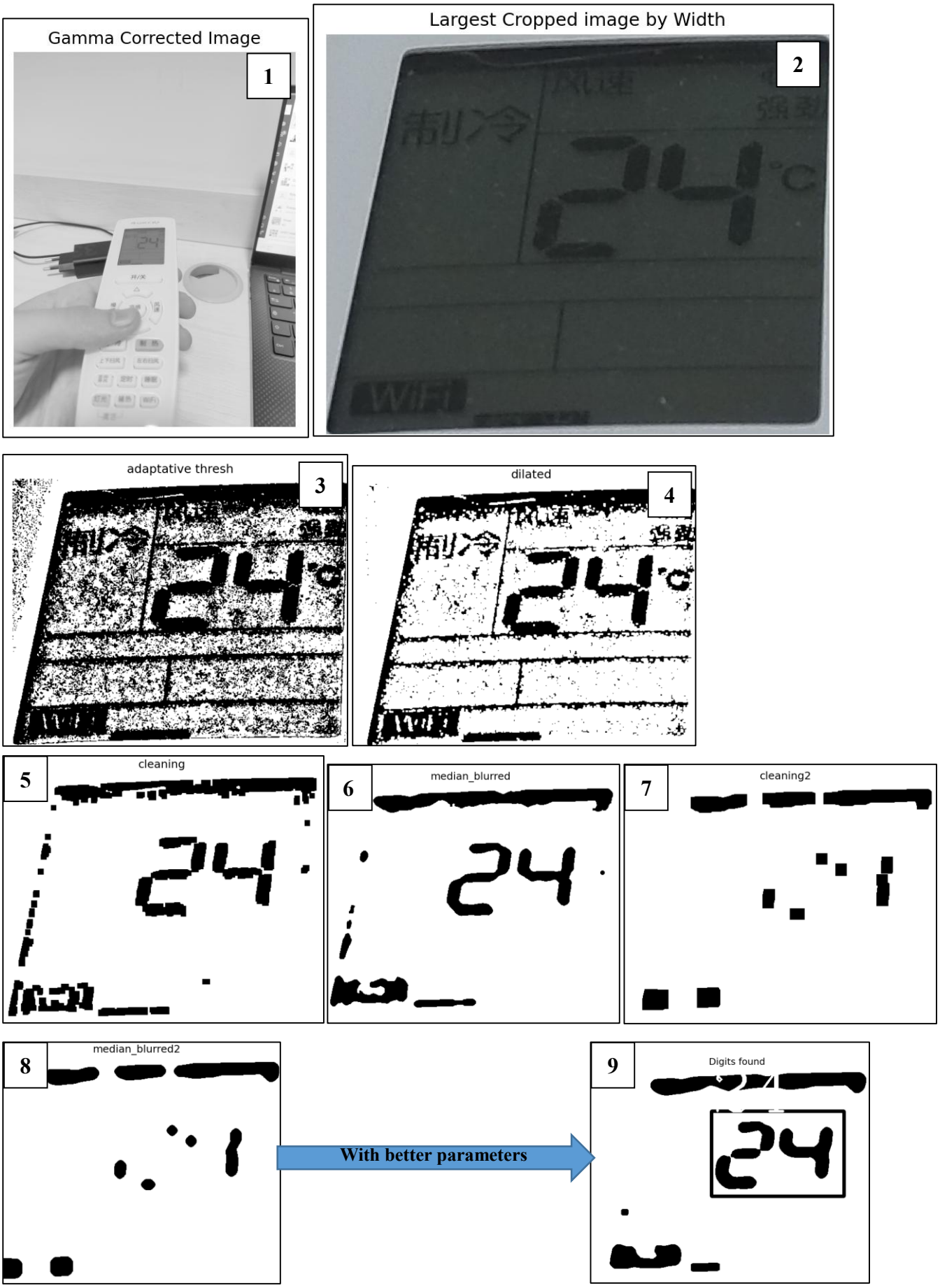


Figure RR8: model 4 processing

Testing program results

Model 2

```
This model read correctly: 20/205
Execution time: 2303.5698223114014 seconds
```

Figure RR9: model 2 accuracy and execution time

Indeed, we managed to get approximately **9,7% of accuracy** and with an execution time around **38 minutes**.

Model 3

```
Execution time: 5204.062310934067 seconds
This model read correctly: 16/205
```

Figure RR10: Testing output from model 3

We managed to get approximately **7,8% of accuracy** and with an execution time around **86 minutes**.

Model 4

```
Execution time: 19472.85061979294 seconds
This model read correctly: 20/205
```

Figure RR11: Testing output from model 4

We managed to get approximately **9,7% of accuracy** and with an execution time around **324 minutes**.

研究进度安排 Research schedule：

综合实践项目进度安排 **Researching plan：**			
No.	各阶段内容 Progress	时间安排 Period	备注 Notes
1 - 4			
5	Find a supervisor and start of the project	10th – 12th October	Short amount of time
6	Taking 200 hundreds of photos in SJTU	13th – 19th October	Long task
7	Testing different python libraries	20th – 27th October	Many technical issues related to installations
8	Establishing a first model capable to recognize 1 picture	28th October– 4th November	Find strategies on the image processing so as to isolate the digits
9	Testing and upgrading the first model	4th– 14th November	Separate the different functions of the code and upgrade each one
10	New version capable of reading at least 7 pictures. Finishing the midterm report.	15th– 21th November	A code more organized
11	Starting the final report	22th– 27th November	Organize the ideas and the collected resources quoted or used. Fill it with the work done at this moment.
12	Establishing a testing program	28th November– 2nd December	Get an idea of the accuracy of our model. Test the second model.
13	Upgrading cropping	4th– 12th December	Find the common problems in cropping to improve the accuracy and execution time.
14	Upgrading the rest of the image processing	13th– 22th December	Solve the problems found on pictures unread even after a good cropping.
15	Getting the best model as possible	23th– 27th December	Increase the accuracy and the execution time.
16	Finishing the final report and clear the code	28th – 31st December	Make the code easier to read and the report more easier to understand.

参考文献 References：

Code

GitHub link of the code: LINK

Figures

P: Marko Babic – Screenshot of Pycharm's figure – Beginning of October 2024 - [BACK]

RC1: Helin Gong - Picture of me taking a photo of AC – Mid October 204 - [BACK]
RC2: Aurélien Max - Concepts, Objects, and Java: First Year of the Engineering Cycle in Computer Science at Polytech Paris-Saclay - 2023-24 - [BACK]
RC3: Marko Babic – Screenshot of Pycharm's figure – End of October 2024 - [BACK]
RC4: Marko Babic – Screenshot of Pycharm's interface – End of November 2024 - [BACK]
RC5: Marko Babic – Screenshot of Pycharm's interface computer – End of November 2024 - [BACK]
RC6: Marko Babic – Screenshot of Pycharm's interface – End of November 2024 - [BACK]
RC7: Marko Babic – Screenshot of Pycharm's terminal – Beginning of December - [BACK]
RC8: doxygen - Eroding and Dilating - December 29 2024 - [LINK] – [BACK]
RC9: doxygen - More Morphology Transformations - December 29 2024 - [LINK] - [BACK]
RC10: doxygen - Smoothing Images- December 29 2024 - [LINK] - [BACK]

RR1: Marko Babic – Screenshot of Pycharm's figure – Beginning of November 2024 - [BACK]
RR2: Marko Babic – Screenshot of Pycharm's figure – Beginning of November 2024 - [BACK]
RR3: Marko Babic – Screenshot of Pycharm's figure – Mid November 2024 - [BACK]
RR4: Marko Babic – Screenshot of Pycharm's figure – Mid November 2024 - [BACK]
RR5: Marko Babic – Screenshots of Pycharm's figures – Mid November 2024 - [BACK]
RR6: Marko Babic – Screenshots of Pycharm's figures – Beginning of December 2024 - [BACK]
RR7: Marko Babic – Screenshots of Pycharm's figures – Beginning of December 2024 - [BACK]
RR8: Marko Babic – Screenshot of Pycharm's terminal – End of December - [BACK]
RR9: Marko Babic – Screenshot of Pycharm's terminal – Beginning of December 2024 - [BACK]
RR10: Marko Babic – Screenshot of Pycharm's terminal – Beginning of December 2024 - [BACK]
RR11: Marko Babic – Screenshot of Pycharm's terminal – End of December 2024 - [BACK]

Quotes

[RCQ1] - Python Language advantages and applications - Last Updated : Nov 1, 2023 - [LINK] - [BACK]

Libraries

cv2 - Released: June 18, 2024 – [LINK]
Numpy - Released August 18, 2024 - [LINK]
Matplotlib - Released May 16, 2024 **-** [LINK]
Imutils - Last modification 2022 - [LINK]
Pytesseract - Released: August 16, 2024 - [LINK]
PaddleOCR - Last release: October 18, 2024 - [LINK]
EasyOCR - Released: September 24, 2024 - [LINK]
Time - Released: October 5, 2020 - [LINK]

Videos and websites

YouTube Channel: Computer vision engineer – Text detection with Python and Opencv | OCR using Easy OCR | Computer vision tutorial – 2023 - [LINK]

doxygen - OpenCV Morphological Transformations - December 29 2024 - [LINK]

指导教师意见 Supervisor’s opinion

指导教师意见（课题难度是否适中、工作量是否饱满、进度安排是否合理、研究是否达到预期要求等）：
Supervisor's opinions (whether the project difficulty is suitable, whether the workload is full, whether the schedule is reasonable, whether the research is qualified, etc.)

This project aims to investigate the techniques on AC Image Recognition of SJTU Teaching Building. This images were all collected at SJTU Teaching Buildings/Offices. The research results are sufficient, satisfactory and have a certain supportive effect on the construction of a smart energy teaching building.

指导教师签名 Supervisor signature： 

日期 Date(YYYY-MM-DD)：2024-12-31

学院（系）意见 Institute's opinions

院长（系主任）签名 Dean signature：

日期 Date(YYYY-MM-DD)：