**Instituto Tecnológico de Monterrey Campus Querétaro**


**Programming Languages**
**Final Project**

**August – December 2020**

**Social Distance Detector Alarm System**

**Teacher: Benjamín Valdés Aguirre PhD**

**Ricardo Antonio Vázquez Rodríguez**
**A01209245**

# Table of Contents

# Context of the problem

## Social problem

Due to the COVID-19 pandemic the world has taken on a new normality that has presented new challenges for visiting public and private closed spaces such as workspaces, schools, restaurants, etc. New politics and regulations have been stablished for achieving social distancing and keeping people safe, such as signs and warning posters. [1] However, sometimes people can get distracted and can avoid keeping a safe distance between themselves. Specially on waiting rooms and lobbies that have many people lined up to enter. [2] Staff and security personnel have become vigilant of this issue, keeping an eye on visitors to keep them safe. However, there are many limitations to this approach, and still this issue needs to be attended as quickly as possible.

## Technical problem

Cameras and surveillance devices can help to automate the assurance of personal space distancing between people and alert if a safe social distance is not being respected. However, the detection needs to be done in a fast fashion due to the fast speed of changes in the scene. Time is vital for virus transmission and the processing that can be done to an image in order to analyze and interpret the data. This can take a considerable amount of time due the image's large size in memory and the number of operations needed to obtain relevant information from the image.

# Solution

A distance detector alarm system can help make sure that people are keeping a safe distance from each other while waiting in line at an enclosed space. For this problem to be solved it is necessary to reduce the time it takes to process images and alert as fast as possible if someone is not respecting the stablished safe social distance, hence a parallel paradigm is needed. This means that instead of having a sequential way of analyzing the images with a single thread; multiple threads can be used to work at the same time on different tasks or sections and divide the overall time taken to do the tasks. This solution is intended to prevent more than 2 parties of people from getting too close to each other.

For preventing people to get too close to each other, it is necessary to detect if a person appears in the image and if more than one person is present; then calculate the distances between them. This solution contains different subproblems that need to be tackled in order to achieve this detection.

The coded solution is located in a GitHub open repository.

Figure 1 shows the steps for the distance detector alarm system, the ones in blue squares represent inputs, the ones in yellow represent parallel operations and the one in green represents the final output of the program. Note that there is a sequence between them because they are dependent on each other. The approach to this solution is the following:

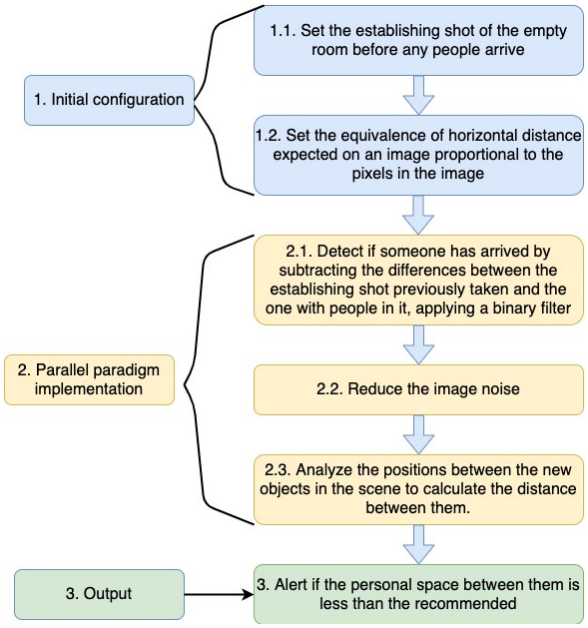


Figure 1: Steps for the solution

Each of these steps is described as follows:

## 1. Initial Configuration

### 1.1. Set the establishing shot of the empty room before any people arrive.

The image of the empty space is stablished. The background needs to be clear and the lightning needs to be consistent (a closed space). The following figure is an example of a room used to test the program.

Figure 3: Empty room initial shot

### 1.2. Set the equivalence of horizontal distance expected on an image proportional to the pixels in the image

For the initial configuration it is necessary to set the horizontal distance in meters of the empty space that is going to be captured by the camera. As well as the minimum distance allowed between two people without alerting them. These values will be used as a reference on the final step to transform the calculated distances in pixels to meters as a scale.

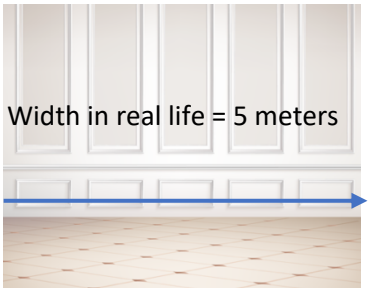



Figure 2: Example of a horizontal distance

## 2. Parallel paradigm implementation

Steps 2.1, 2.2 and 2.3 require the whole image to be parsed pixel by pixel, with one thread; this would be done in a sequential way applying the operations for each pixel. The number of operations equals the total size of the image (width * height). As shown in Figure 4:

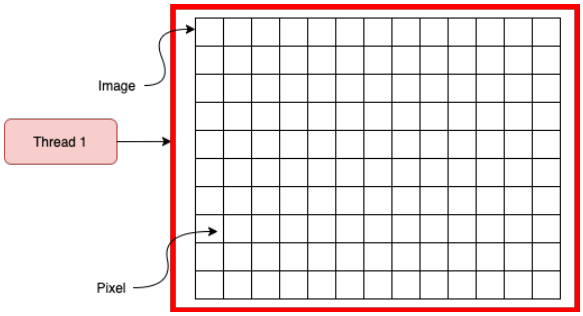



Figure 4: Image assigned pixel tasks using 1 thread

This takes a considerable amount of time proportional to the size of the image. However, if we divide the image in sections, these tasks could be done by more than one thread simultaneously, the total number of operations would still be the same, but the time would be reduced since they do the calculations in parallel, as shown in figure 5. [3]

That's why steps 2.1, 2.2 and 2.3 use a java executor known as fork join, the most common kind of executor according to Oracle. [4] An executor distributes the total work to smaller tasks for threads to do the operation. This improves the performance by using the processing capacity of multiple processors. [5]

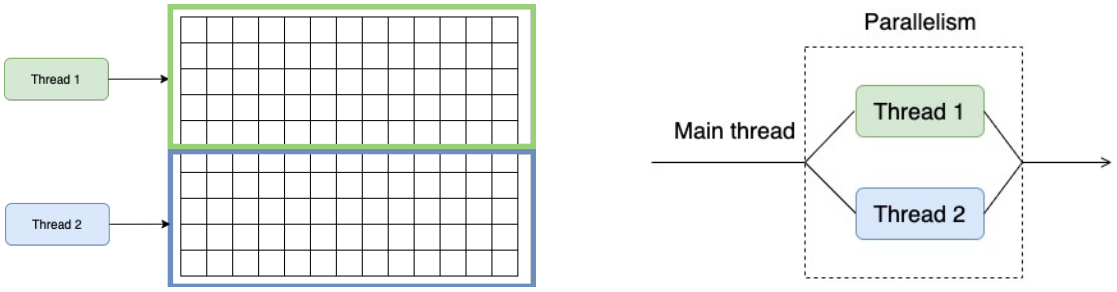



Figure 5: Image division of pixel tasks using multiple threads.

Previous to these operations, the image is first linearized, that means, transformed from an image (Buffered Image) to an array to be worked with without doing the reading and writing operations for each thread.

### 2.1. Detect if someone has arrived by subtracting the differences between the establishing shot previously taken and the one with people in it, applying a binary filter.

To analyze the differences detected between two images, that is, knowing which are the new objects in the scene, it is necessary to do a subtraction between the actual image with people in it and the establishing shot (image without people in it), as shown in figure 6. This concept is known as foreground detection in the field of image processing. However, a simple subtraction between them can also carry over noise we are not interested in, such as luminance and shadows, that's why it is necessary to stablish a tolerance value for detecting new changes between two images, this value is commonly referred to as a threshold.

Figure 6: Detecting the differences between two images by subtraction

After detecting the changes, the image is transformed from RGB colors to black and white, such that black pixels represent new changes on the image and white values represent no changes in the image (the background). This is commonly referred to as image binarization. A binarization consists of transforming a colored or even a grayscale image to a completely black and white representation for the pixels as shown in figure 7.

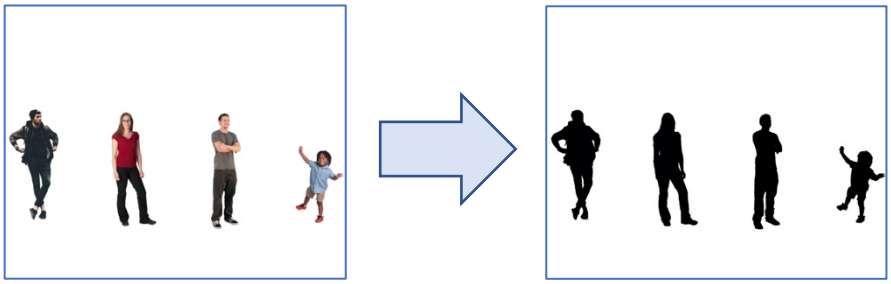


Figure 7: Binarization of the result of the differences between two images

In order to stablish if a colored RGB pixel will be transformed to a black or a white pixel, a threshold value is used to delimit this establishment, that is, for each red, green and blue value of the pixel within the image, if they are lower or equal than the threshold it will be transformed to a white pixel, and in case it is greater than the threshold, it will be converted to black. This binarization is needed in order to reduce the image noise for the next step.

## 2.2. Reduce the image noise.

After detecting the new objects on the scene, noise can be carried over because of the threshold of the previous filter, which is the value used to differentiate between the people and the background. As well as the image size. As seen in the following figure:

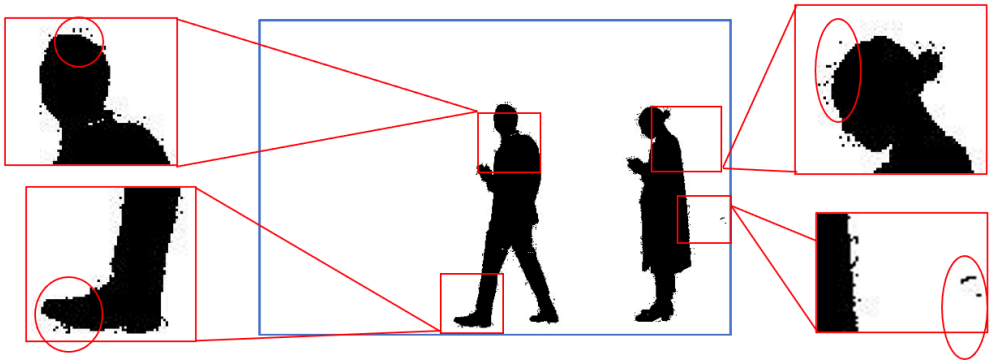
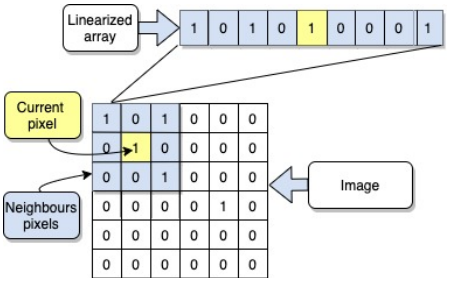


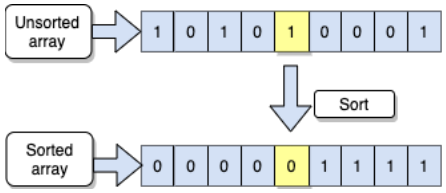
Figure 8: Example of image noise

So, in order to delete these small “particles” seen at the border of each person a median filter can be used. Which is a popular algorithm that helps reduce image noise [6]. This filter is called in a parallel way and works in the following manner:

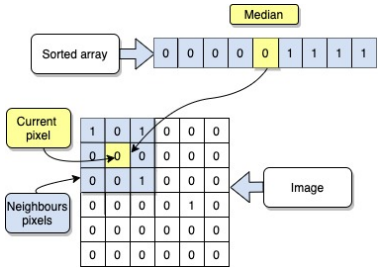
1. For each pixel it gets all of its nearby 8 neighbors.



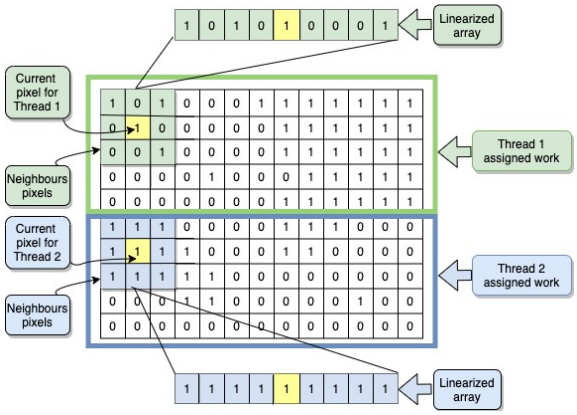

2. The resulting array with 9 elements is sorted in an increasing order.




3. The value in the middle, the median, is set to the current pixel.




4. This is done for each pixel in the image, the following would be a first iteration example using 2 threads in parallel.




The result after removing the noise in this example was the following:

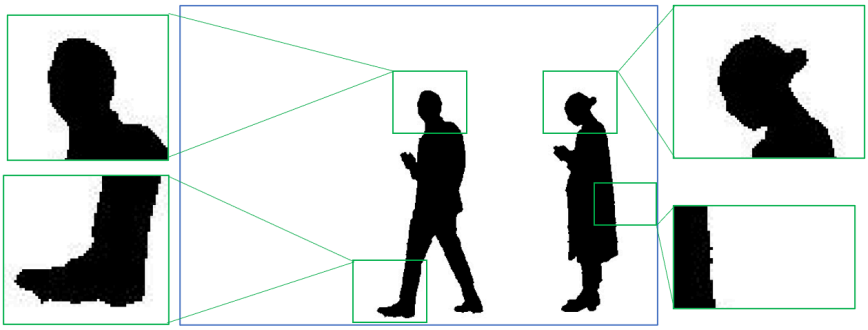


Figure 9: Example of image after applying the median filter

## 2.3. Analyze the positions between the new objects in the scene to calculate the distance between them.

After binarizing the new objects in the scene, that is, the people that have arrived at the scene, it is necessary to detect the distances between them. After analyzing how to segment the image properly into smaller sections to achieve parallelism we can notice that this segmentation is not trivial, because of the patterns in people's anatomy and movement. Some differences may occur in a person's body if analyzed in a horizontal manner. That's why the image needs to be divided horizontally and analyzed on a vertical manner, in order to properly differentiate between two separated individuals.

In this approach each thread can detect the column in which a person starts to appear and the column in which it ends, that is, their horizontal boundaries. Each thread stores these resulting boundaries. Finally, they are sorted to join the results of the threads together. That's why the threads share an object that has access to the list of people boundaries. As the threads can modify the list of start and end boundaries at the same time, this object needs to be synchronized in order to achieve consistency and avoid running condition errors. As seen in figure 10, the image can be analyzed using more than one thread, each one getting the corresponding boundaries.

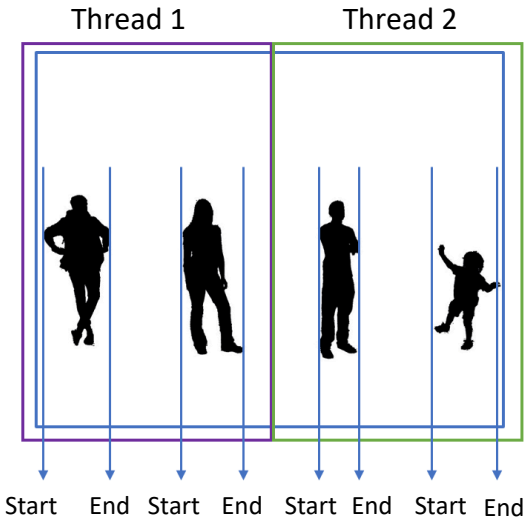



Figure 10: Two threads analyzing the start and end boundaries of people in parallel.

### Why using this specific algorithm?

Another relevant algorithm for object detection in an image is known as "connected components", as in the proposed solution in this document, this approach consists of binarizing a given image (turn each pixel into black or white) to discard the background behind the objects or the irrelevant information that we are not interested in, using a threshold value to stablish the discrimination or tolerance of the image colors. This filtered image can be saved into an array with the values. Then, a verification of each binarized pixel (value in the array) with the nearby previous neighbor pixels is done, so that if the previous left pixel and the previous upper pixel is different from the pixel being verified, it is tagged with a new different value. Tagging consists of assigning a value. [7]

However, this makes the processing dependent between the pixels in a sequential way. The problem is that the connected components algorithm can't be done in parallel because of the relationships between the pixels and their dependencies, making the threads dependent on each other. So, if an actual connected component is divided to be processed by multiple threads the relationship between them is broken. The complexity of this problem solution in big O notation is O ($n^2$), because it is necessary to transverse each row and column in the image, so it can take a considerable amount of time to detect different elements (people) in an image, proportional to the image size.

### 3. Alert if the personal space between people is less than the recommended.

After detecting the horizontal distance between people in pixels in the previous step, the distance is converted to the corresponding length in meters using the horizontal distance in meters of the stablishing shot provided on step 1. In case the distance is lower than the one specified, an alert is shown in the graphic interface as a red light and the output of the terminal shows that there is no safe social distance.

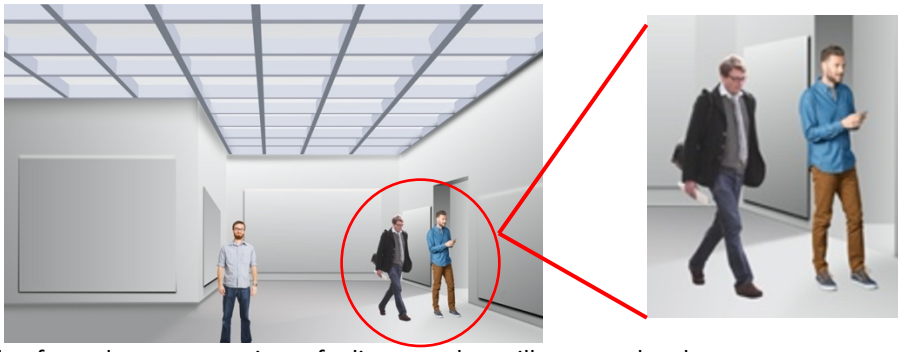


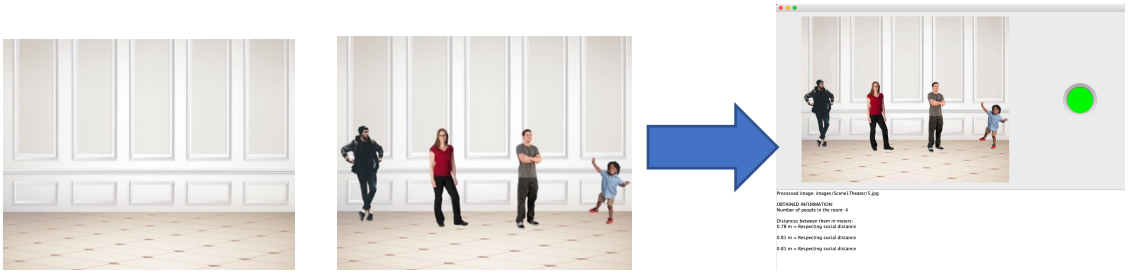
Figure 11: Example of people not respecting safe distance, that will turn on the alarm

## Results

The solution works as expected for both the distance detection and the improvement of processing times by testing it with different scenarios with large image sizes (2.5 MB, 2 MB and 6.2 MB). Proving that parallelism favors image processing and analysis. After running steps 2.1,

2.2 and 2.3 in a sequential manner instead of using parallelism the results were the following for the runtimes and detection:

**Test case 1:** Using images emptyTheaterWaitingLine.jpg and images/Scene1Theater/5.jpg



**Input:**

Initial configuration
Please enter the horizontal distance of the room in meters
**5**
Please enter the social distance of the room in meters (recommended 1 meter)
**0.5**

**Output:**

OBTAINED INFORMATION:
* Number of people in the room: **4**

Starts in pixels: 37 940 1819 2567
Ends in pixels: 472 1334 2081 2991
Distances between them in pixels:
468 485 486

Distances between them in meters:

**0.78 = Respecting social distance**
**0.81 = Respecting social distance**
**0.81 = Respecting social distance**

Expected results:

✓ Detect 4 people in the room.

✓ Does not alert, social distance is respected.

Analysis of the given results:

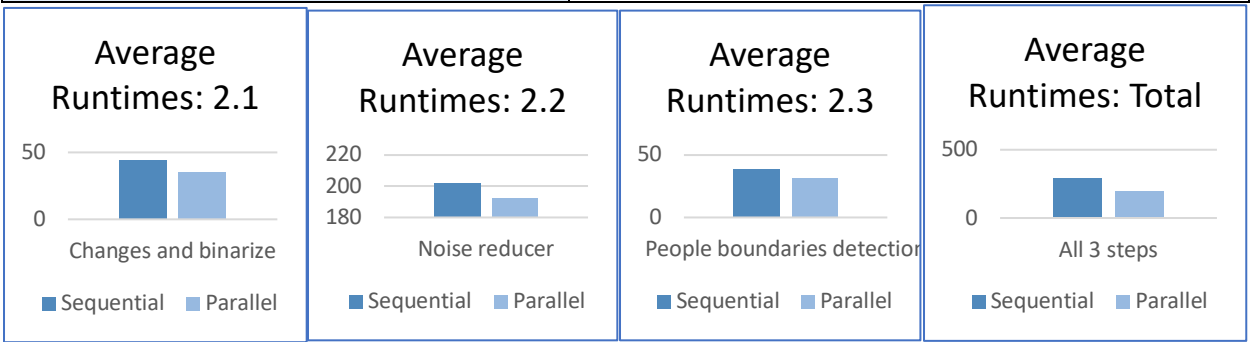
✓ As we can see, 4 people were correctly detected.

✓ The distances between them are also correct (3 distances between 4 people)

✓ The distances were correctly transformed to meters.

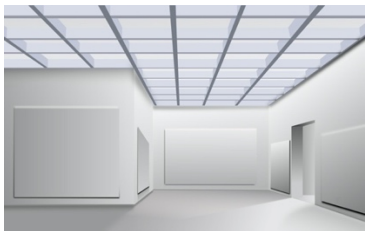
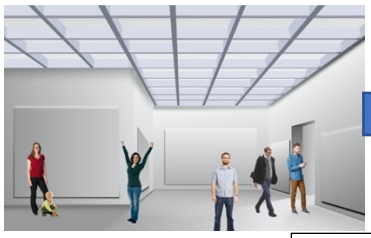
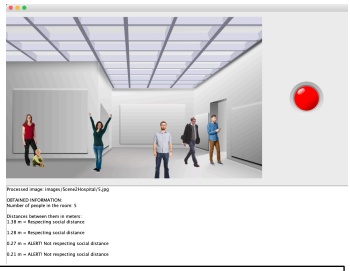
✓ The alerts are correct, they do not need to show up, all of them are respecting the stablished social distance (0.5 meters).

| Time in milliseconds | Step 3: Changes and binarize | | Step 4: Noise reducer | | Step 5: People boundaries detection | |
|---|---|---|---|---|---|---|
| **Runtimes** | **Sequential** | **Parallel** | **Sequential** | **Parallel** | **Sequential** | **Parallel** |
| **Run 1** | 44 | 31 | 240 | 132 | 43 | 38 |
| **Run 2** | 49 | 39 | 194 | 110 | 37 | 27 |
| **Run 3** | 42 | 31 | 191 | 163 | 39 | 31 |
| **Run 4** | 43 | 33 | 192 | 133 | 38 | 31 |
| **Run 5** | 42 | 40 | 192 | 108 | 38 | 31 |
| **Average** | 44 | 34.8 | 201.8 | 129.2 | 39 | 31.6 |
| **Difference (averages)** | | 9.2 | **Difference** | 72.6 | **Difference** | 7.4 |
| **Total difference** | | | 89.2 | | | |




The total average save time in milliseconds is 89.2, this represents a **31.32% decrease in time** when compared with the total 284.8 milliseconds in average of the sequential approach.

**Test case 2:** Using images: emptyHospitalWaitingLine.jpg and images/Scene2Hospital/5.jpg

**Given input:**
Initial configuration
Please enter the horizontal distance of the room in meters
**8**
Please enter the social distance of the room in meters (recommended 1 meter)
**1**

**Given output:**
OBTAINED INFORMATION:
* Number of people in the room: **5**

Starts in pixels: 431 2191 3900 4632 5328
Ends in pixels: 1023 2811 4406 5148 5762
Distances between them in pixels:
**1168 1089 226 180**

Distances between them in meters:
**1.38 = Respecting social distance**
**1.28 = Respecting social distance**
**0.27 = ALERT! Not respecting social distance**
**0.21 = ALERT! Not respecting social distance**

Expected results:

✓ Detect 5 people in the room.

✓ Alert no social distance being respected between the three people on the right.

Analysis of the given results:

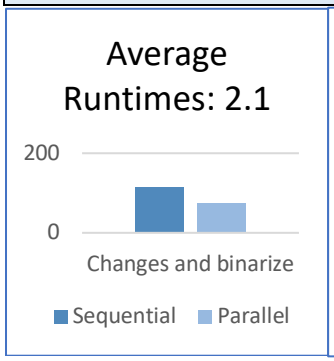
✓ As we can see, 5 people were correctly detected (the woman with a child in the left is a party)

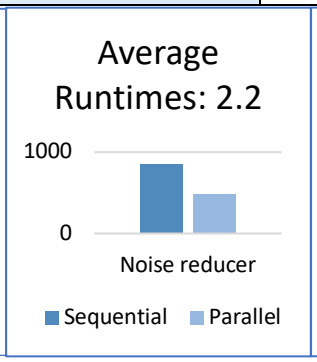
✓ The distances between them are also correct (4 distances between 5 people)

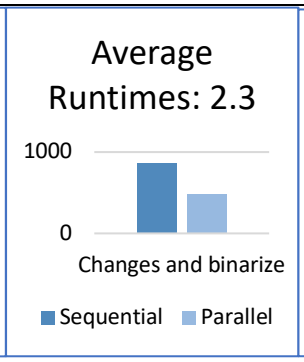
✓ The distances were correctly transformed to meters.

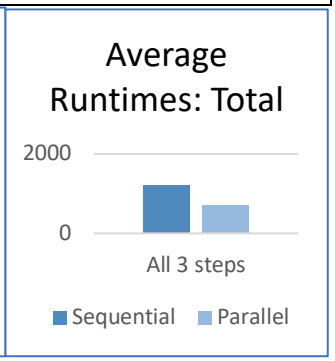
✓ The alerts are correct, as the first 2 distances are greater than the stablished tolerance (1 meter) but the other 2 are lower than the established distance.

| Time in milliseconds | Step 2.1: Changes and binarize | | Step 2.2: Noise reducer | | Step 2.3: People boundaries detection | |
|---|---|---|---|---|---|---|
| **Runtimes** | **Sequential** | **Parallel** | **Sequential** | **Parallel** | **Sequential** | **Parallel** |
| **Run 1** | 114 | 77 | 760 | 572 | 248 | 136 |
| **Run 2** | 115 | 72 | 949 | 438 | 256 | 139 |
| **Run 3** | 112 | 76 | 924 | 398 | 251 | 135 |
| **Run 4** | 115 | 74 | 769 | 524 | 247 | 136 |
| **Run 5** | 110 | 78 | 887 | 475 | 246 | 135 |
| **Average** | 113.2 | 75.4 | 857.8 | 481.4 | 249.6 | 136.2 |
| **Difference (averages)** | | 37.8 | **Difference** | 376.4 | **Difference** | 113.4 |
| **Total difference** | | | 527.6 | | | |




The total average save time in milliseconds is 527.6, this represents a **43.22% decrease in time** when compared with the total 1220.6 milliseconds in average of the sequential approach.

**Test case 3:** Using image emptyElevator.jpg and images/Scene3Elevator/5.jpg


## Input:

Initial configuration
Please enter the horizontal distance of the room in meters
**3**
Please enter the social distance of the room in meters (recommended 1 meter)
**0.35**

## Output:

OBTAINED INFORMATION:
* Number of people in the room: **3**

Starts in pixels: 903 1690 3073
Ends in pixels: 1621 2451 3791
Distances between them in pixels:
69 622

Distances between them in meters:

**0.05 = ALERT! Not respecting social distance**
**0.43 = Respecting social distance**

## Expected results:

✓ Detect 3 people in the room.

✓ Alert no social distance being respected between the two people on the left.

## Analysis of the given results:

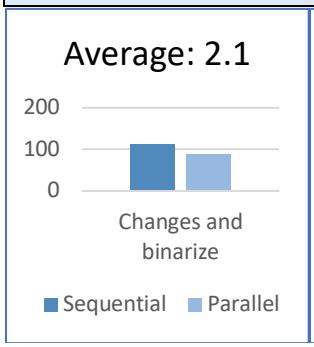
✓ As we can see, 3 people were correctly detected.

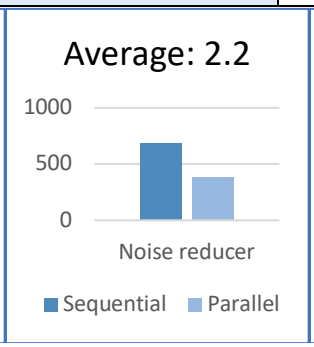
✓ The distances between them are also correct (2 distances between 3 people)

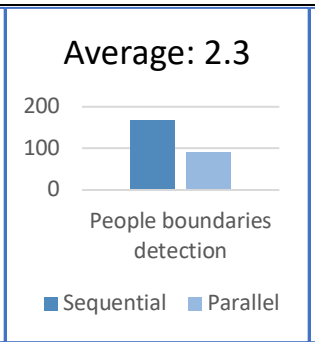
✓ The distances were correctly transformed to meters.

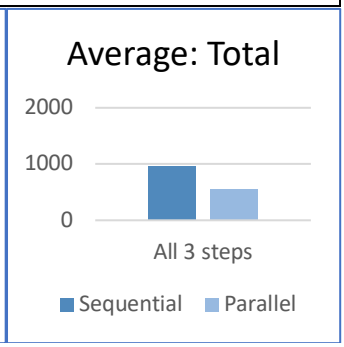
✓ The alerts are correct, as the first distance is greater than the stablished tolerance (0.35 meter) and the second distance is being respected

| Time in milliseconds | Step 3: Changes and binarize | | Step 4: Noise reducer | | Step 5: People boundaries detection | |
|---|---|---|---|---|---|---|
| **Runtimes** | **Sequential** | **Parallel** | **Sequential** | **Parallel** | **Sequential** | **Parallel** |
| **Run 1** | 112 | 89 | 636 | 473 | 164 | 88 |
| **Run 2** | 112 | 87 | 766 | 335 | 172 | 90 |
| **Run 3** | 112 | 86 | 623 | 325 | 166 | 97 |
| **Run 4** | 112 | 87 | 634 | 389 | 166 | 87 |
| **Run 5** | 112 | 87 | 759 | 398 | 164 | 88 |
| **Average** | 112 | 87.2 | 683.6 | 384 | 166.4 | 90 |
| **Difference (averages)** | | 24.8 | **Difference** | 299.6 | **Difference** | 76.4 |
| **Total difference** | | | 400.8 | | | |




The total average save time in milliseconds is 400.8, this represents a **41.66% decrease in time** when compared with the total 962 milliseconds in average of the sequential approach. More tests can be found in the GitHub Repository at Tests.

# Conclusions

The parallel programming paradigm is useful for reducing the amount of time it takes for a large number of repetitive tasks to be executed, by dividing them and assigning them to different threads, executing on real time on multicore systems and GPUs. Image processing and pattern detection on images result ideal for parallelism, and a lot of information can be obtained in a fast way. [8] This implementation was designed considering a 2D image, meaning that no depth of the person within the room was considered; however, it could also be useful if desired to add the functionality of differentiating from two people when they get really close to each other to the point they overlap, using a 3D camera and deep learning algorithms. [9]

There are other useful frameworks that can be used for image processing in parallel, such as OpenCV [10], and it would be interesting to use them, however due that the main part of the paradigm programming was intended to be done by myself they were not used on this implementation.

The obtained information from this program can be used for analyzing other important factors and considerations to help businesses and communities to prevent virus transmission such as using statistics to predict the number of visitors and recommend/regulate visiting times where the number of people is the lowest expected. This project can be used to reduce the number of virus transmission by helping people to prevent getting close to each other.

# Setup instructions

## Deployed JAR:

If you want to just run the program you can download the parallel implementation (without comparing the times with the sequential solution) or the JAR that

1. Download the corresponding JAR

The JAR can be downloaded from the following link:
https://drive.google.com/file/d/1R2P_M2zcwfwaSvvXN2TvCUJwTwdy_T3_/view?usp=sharing

How to run JAR file:

1. Open a shell or terminal.

2. Inside the terminal, go to the folder containing the JAR file.

3. Run it as:
   **java -jar SocialDistance.jar**

4. Insert the desired input on the terminal. The test configurations can be found in the next section.

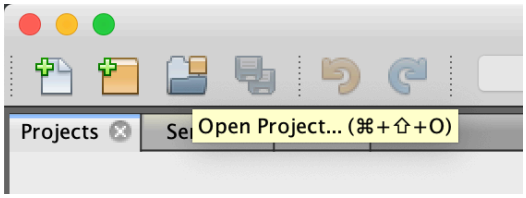
## Code Setup and run

1. Clone GitHub repository on the terminal:

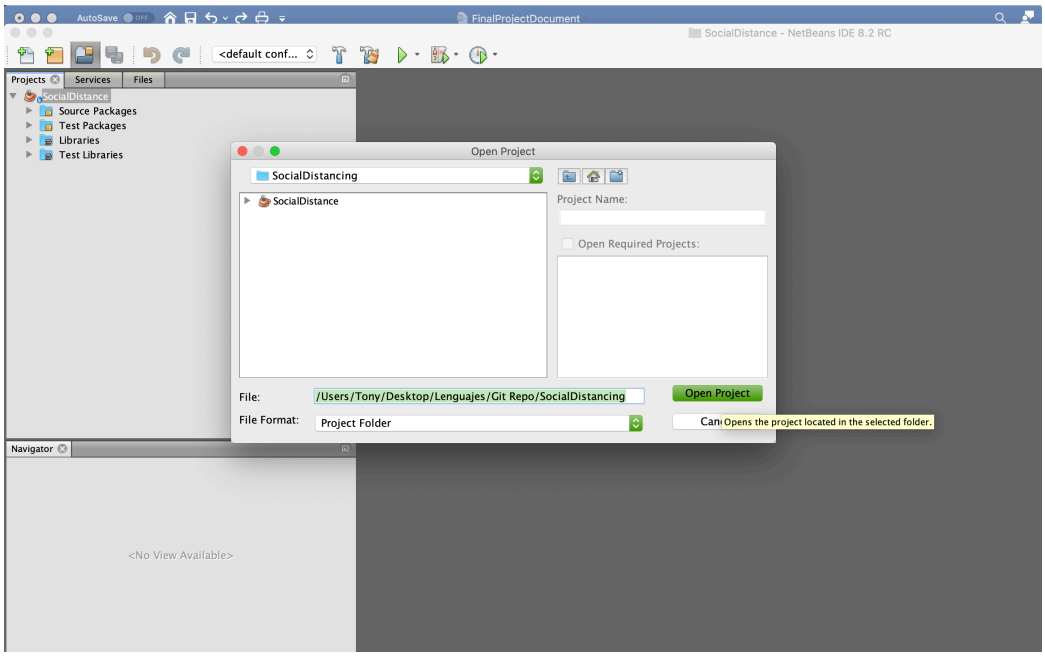
```
git clone https://github.com/RAnthonyVR/SocialDistancing.git
```

2. Have Java 1.8 installed.
3. Open the project on NetBeans (I personally use NetBeans 8.2 RC):

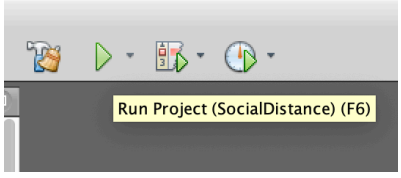
3.1 Select Open Project



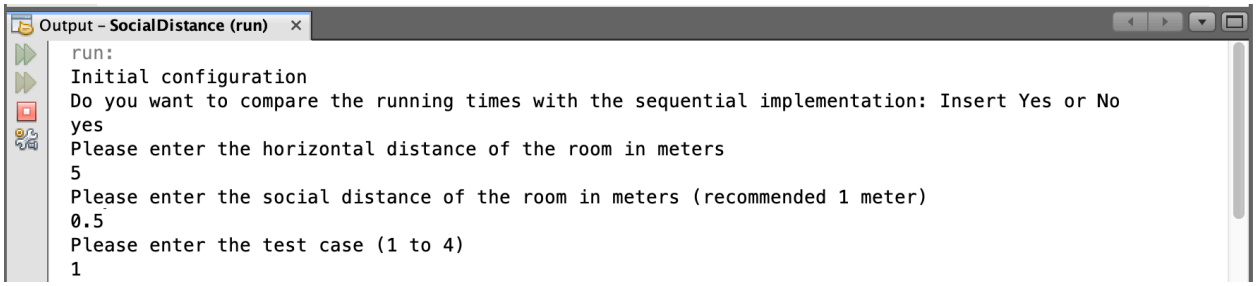

3.2 Select the project from the file browser and click open



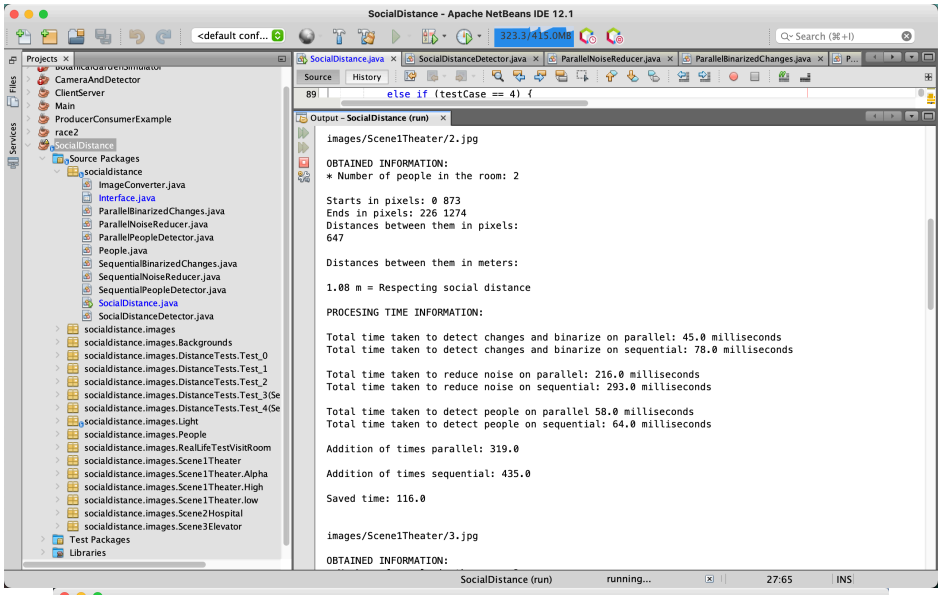

3.3 Run the project by clicking on the play button

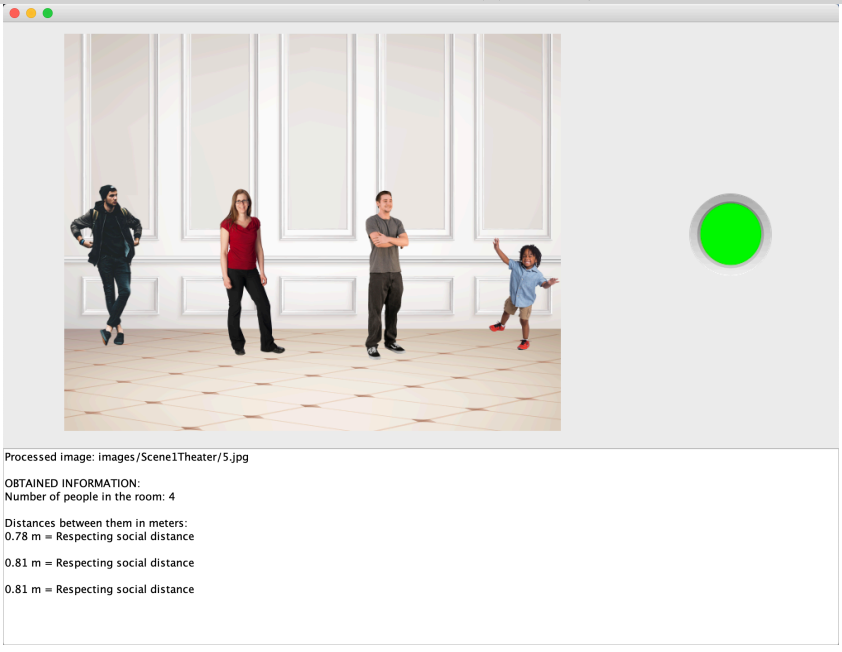


4. Insert the desired input.




5. The program will show information on the output shell and will open a JFrame to view the tests.



**Test configurations:** The suggested inputs for the tests are the following:

### Test case 1: It is expected at all images respect social distance

Initial configuration
Please enter the horizontal distance of the room in meters
**5**
Please enter the social distance of the room in meters (recommended 1 meter)
**0.5**
Please enter the test case (1 to 4)
**1**

### Test case 2: It is expected at not all images respect social distance (images 4-5)

Initial configuration
Please enter the horizontal distance of the room in meters
**8**
Please enter the social distance of the room in meters (recommended 1 meter)
**1**
Please enter the test case (1 to 4)
**2**

### Test case 3: It is expected at not all images respect social distance (images 2-5)

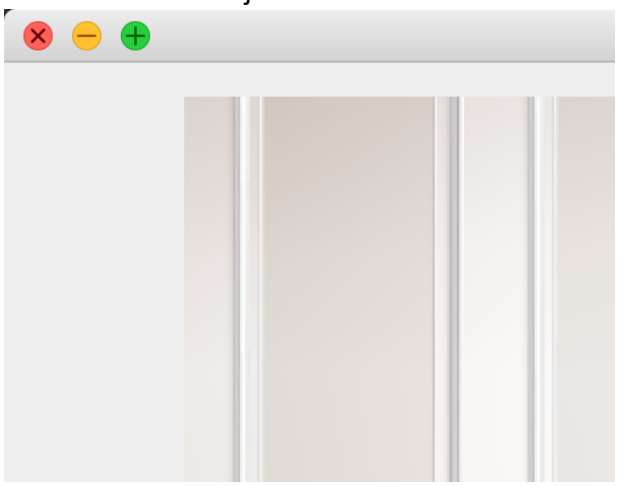
Initial configuration
Please enter the horizontal distance of the room in meters
**3**
Please enter the social distance of the room in meters (recommended 1 meter)
**0.35**
Please enter the test case (1 to 4)
**3**

### Test case 4: It is expected to detect 1 person

Initial configuration
Please enter the horizontal distance of the room in meters
**1**
Please enter the social distance of the room in meters (recommended 1 meter)
**2**
Please enter the test case (1 to 4)
**4**

6. Closing the project

6.1 The program will close until the jFrame has been closed.



# References


[1] Tšernov, K. (2020, March 24). *COVID-19 Safety Guidelines: 7 Tips on Safe Queuing and Workplace Practices*. https://www.qminder.com/queuing-safety-tips-coronavirus/.

[2] CDC. (2020). *Social Distancing, Quarantine, and Isolation*. https://www.cdc.gov/coronavirus/2019-ncov/prevent-getting-sick/social-distancing.html.

[3] ResearchGate. An Improved AES Encryption of Audio Wave Files - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/Fork-Join-model-WebSite-4_fig12_312277403

[4] Oracle. (2020). *Fork/Join*. Fork/Join (The Java™ Tutorials > Essential Classes > Concurrency). https://docs.oracle.com/javase/tutorial/essential/concurrency/forkjoin.html.

[5] Oracle. (2020). *Executors*. Executors (The Java™ Tutorials > Essential Classes > Concurrency). https://docs.oracle.com/javase/tutorial/essential/concurrency/executors.html.

[6] Fisher, R. (2000). *Median Filter*. Spatial Filters - Median Filter. https://homepages.inf.ed.ac.uk/rbf/HIPR2/median.html

[7] Becker, A. (2017, October 2017). Intro2Robotics: Connected Components in a Binary Image. YouTube. https://www.youtube.com/watch?v=ticZclUYy88

[8] Akgün, Devrim. (2013). Performance Evaluations for Parallel Image Filter on Multi - Core Computer using Java Threads. International Journal of Computer Applications. 74. 13-19. 10.5120/12928-9836.https://www.researchgate.net/publication/260632404_Performance_Evaluations_for_Parallel_Image_Filter_on_Multi_-_Core_Computer_using_Java_Threads

[9] Dwivedi, P. (2019, March 27). *People Tracking using Deep Learning*. Medium. https://towardsdatascience.com/people-tracking-using-deep-learning-5c90d43774be.

[10] OpenCV team. (2020). OpenCV. https://opencv.org/.

[11] Gamemaker Game Programming Course. (2018, October 2). *Java Add Image on JFrame*. YouTube. https://www.youtube.com/watch?v=QiLeau29fOQ.

[12] Oracle. (2020, June 24). *Package javax.swing*. javax.swing (Java Platform SE 7). https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

[13] Oracle. (2020). *Guarded Blocks*. Guarded Blocks (The Java™ Tutorials > Essential Classes > Concurrency). https://docs.oracle.com/javase/tutorial/essential/concurrency/guardmeth.html

[14] TrueCoder. (2020, May 8). *How to create a beep sound in Java | Very easy java program for beginners*. YouTube. https://www.youtube.com/watch?v=8NUSbY_7Joc