

# Assignment cover sheet

Complete all sections of this coversheet

|  |  |
| --- | --- |
| Student name | Student number |
| Mohamed Abuklal | 5902514 |

|  |  |
| --- | --- |
| Subject number and name: | Real-Time Embedded Systems ECTE331 |
| Subject coordinator: | Dr Abdsamad binkrid |
| Title of Assignment: | Project Problems (Part A) |
| Date and time due: | 18/06/2024 |
| Total number of pages: | 13 |

Student declaration and acknowledgement (must be read by all students)

By submitting this assignment online, the submitting student declares on behalf of the team that:

1. All team members have read the subject outline for this subject, and this assessment item meets the requirements of the subject detailed therein.
2. This assessment is entirely our own work, except where we have included fully documented references to the work of others. The material contained in this assessment item has not previously been submitted for assessment.
3. Acknowledgement of source information is in accordance with the guidelines or referencing style specified in the subject outline.
4. All team members are aware of the late submission policy and penalty.
5. The submitting student undertakes to communicate all feedback with the other team members.

Place compressed photo of your structure or team here (optional)

**Part A**

**Code**

**For the Single Thread:**

**package** project;

**import** javax.imageio.ImageIO;

**import** java.awt.Graphics2D;

**import** java.awt.image.BufferedImage;

**import** java.awt.Color;

**import** java.io.File;

**import** java.io.IOException;

**public** **class** TemplateMatchingSingleThread {

**public** **static** **void** main(String[] args) {

**try** {

// Reading the input images provided:

BufferedImage templateImage = ImageIO.*read*(**new** File("OneG.jpg"));

BufferedImage sourceImage = ImageIO.*read*(**new** File("TenCardG.jpg"));

// If the template Image couldn't be read, a fail message will be printed as the following:

**if** (templateImage == **null**) {

System.***err***.println("Failed to read template image: OneG.jpg");

**return**;

}

// If the source Image couldn't be read, a fail message will be printed as the following:

**if** (sourceImage == **null**) {

System.***err***.println("Failed to read source image: TenCardG.jpg");

**return**;

}

**long** start\_Time = System.*currentTimeMillis*();

// Performing a template matching for both the template and source images:

BufferedImage resultImage = *templateMatchingSingleThread*(templateImage, sourceImage);

**long** endTime = System.*currentTimeMillis*();

**long** executionTime = endTime - start\_Time;

System.***out***.println("Single-thread execution time: " + executionTime + " ms");

// Saving the output image with rectangles as single-Threadpic.jpg:

**boolean** write\_Success = ImageIO.*write*(resultImage, "jpg", **new** File("single-Threadpic.jpg"));

**if** (write\_Success) {

// Printing a successful message if the output image has been generated successfully:

System.***out***.println("Single-thread result image has been generated successfully.");

} **else** {

// Printing an error message if the output image has not been generated successfully:

System.***err***.println("Failed to generate single-thread result image.");

}

} **catch** (IOException e) {

e.printStackTrace();

}

}

// This method performs a template matching on the source image using the template image:

**private** **static** BufferedImage templateMatchingSingleThread(BufferedImage templateImage, BufferedImage sourceImage) {

// Getting the dimensions of the source image and the template image:

**int** r\_1 = sourceImage.getHeight();

**int** c\_1 = sourceImage.getWidth();

**int** r\_2 = templateImage.getHeight();

**int** c\_2 = templateImage.getWidth();

**int** temp\_Size = r\_2 \* c\_2;

// Initializing a 2D array to store the absolute differences:

**double** minimum = Double.***MAX\_VALUE***;

**double**[][] abs\_Diff\_Mat = **new** **double**[r\_1 - r\_2 + 1][c\_1 - c\_2 + 1];

// Compute the absolute differences between the source image and the template image at each position.

**for** (**int** i = 0; i <= r\_1 - r\_2; i++) {

**for** (**int** j = 0; j <= c\_1 - c\_2; j++) {

**double** abs\_Diff = *computeAbsDiff*(templateImage, sourceImage, i, j, r\_2, c\_2) / temp\_Size;

abs\_Diff\_Mat[i][j] = abs\_Diff;

**if** (abs\_Diff < minimum) {

minimum = abs\_Diff;

}

}

}

**double** threshold = 10 \* minimum;

System.***out***.println("Minimum absolute difference: " + minimum);

System.***out***.println("Threshold: " + threshold);

BufferedImage resultImage = **new** BufferedImage(sourceImage.getColorModel(),

sourceImage.copyData(**null**), sourceImage.isAlphaPremultiplied(), **null**);

**for** (**int** i = 0; i <= r\_1 - r\_2; i++) {

**for** (**int** j = 0; j <= c\_1 - c\_2; j++) {

**if** (abs\_Diff\_Mat[i][j] <= threshold) {

*drawRectangle*(resultImage, i, j, r\_2, c\_2);

}

}

}

**return** resultImage;

}

// This method computes the absolute difference between two images at a given position:

**private** **static** **double** computeAbsDiff(BufferedImage templateImage, BufferedImage sourceImage, **int** startX, **int** startY, **int** height, **int** width) {

**double** abs\_Diff = 0.0;

// Computing the sum of absolute differences between the corresponding pixels in the two images.

**for** (**int** x = 0; x < height; x++) {

**for** (**int** y = 0; y < width; y++) {

**int** template\_RGB = templateImage.getRGB(y, x);

**int** source\_RGB = sourceImage.getRGB(startY + y, startX + x);

abs\_Diff += Math.*abs*(((template\_RGB >> 16) & 0xFF) - ((source\_RGB >> 16) & 0xFF)); // Red

abs\_Diff += Math.*abs*(((template\_RGB >> 8) & 0xFF) - ((source\_RGB >> 8) & 0xFF)); // Green

abs\_Diff += Math.*abs*((template\_RGB & 0xFF) - (source\_RGB & 0xFF)); // Blue

}

}

**return** abs\_Diff;

}

// Drawing the rectangles around the needed shapes for the output image:

**private** **static** **void** drawRectangle(BufferedImage image, **int** startX, **int** startY, **int** height, **int** width) {

Graphics2D g2d = image.createGraphics();

g2d.setColor(Color.***RED***);

g2d.drawRect(startY, startX, width, height);

g2d.dispose();

}

}

**For the Multi-Thread:**

**package** project;

**import** javax.imageio.ImageIO;

**import** java.awt.Graphics2D;

**import** java.awt.image.BufferedImage;

**import** java.awt.Color;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.concurrent.\*;

**public** **class** TemplateMatchingMultiThread {

**public** **static** **void** main(String[] args) {

**try** {

// Reading the input images provided:

BufferedImage templateImage = ImageIO.*read*(**new** File("OneG.jpg"));

BufferedImage sourceImage = ImageIO.*read*(**new** File("TenCardG.jpg"));

// If the template Image couldn't be read, a fail message will be printed as the following:

**if** (templateImage == **null**) {

System.***err***.println("Failed to read template image: OneG.jpg");

**return**;

}

// If the source Image couldn't be read, a fail message will be printed as the following:

**if** (sourceImage == **null**) {

System.***err***.println("Failed to read source image: TenCardG.jpg");

**return**;

}

**long** start\_Time = System.*currentTimeMillis*();

**int** numOfThreads = 4; // The number of threads that have been used.

// Performing a template matching for both the template and source images:

BufferedImage resultImage = *templateMatchingMultiThread*(templateImage, sourceImage, numOfThreads);

**long** end\_Time = System.*currentTimeMillis*();

**long** executionTime = end\_Time - start\_Time;

System.***out***.println("Multi-thread execution time: " + executionTime + " ms");

// Saving the output image with rectangles as single-Threadpic.jpg:

**boolean** write\_Success = ImageIO.*write*(resultImage, "jpg", **new** File("multi-Threadpic.jpg"));

**if** (write\_Success) {

// Printing a successful message if the output image has been generated successfully:

System.***out***.println("Multi-thread result image written successfully.");

} **else** {

// Printing an error message if the output image has not been generated successfully:

System.***err***.println("Failed to write multi-thread result image.");

}

} **catch** (IOException | InterruptedException | ExecutionException e) {

e.printStackTrace();

}

}

// This method performs a template matching on the source image using the template image for the MULTI-THREADS:

**private** **static** BufferedImage templateMatchingMultiThread(BufferedImage templateImage, BufferedImage sourceImage, **int** numOfThreads) **throws** InterruptedException, ExecutionException {

// Getting the dimensions of the source image and the template image:

**int** r\_1 = sourceImage.getHeight();

**int** c\_1 = sourceImage.getWidth();

**int** r\_2 = templateImage.getHeight();

**int** c\_2 = templateImage.getWidth();

**int** temp\_Size = r\_2 \* c\_2;

// Initializing a 2D array to store the absolute differences:

**double**[][] abs\_Diff\_Mat = **new** **double**[r\_1 - r\_2 + 1][c\_1 - c\_2 + 1];

**double**[] minimum = {Double.***MAX\_VALUE***};

ExecutorService executor = Executors.*newFixedThreadPool*(numOfThreads);

List<Future<Void>> futures = **new** ArrayList<>();

**for** (**int** t = 0; t < numOfThreads; t++) {

**int** start\_Row = t \* (r\_1 - r\_2 + 1) / numOfThreads;

**int** end\_Row = (t + 1) \* (r\_1 - r\_2 + 1) / numOfThreads;

futures.add(executor.submit(() -> {

// Compute the absolute differences between the source image and the template image at each position.

**for** (**int** i = start\_Row; i < end\_Row; i++) {

**for** (**int** j = 0; j <= c\_1 - c\_2; j++) {

**double** abs\_Diff = *computeAbsDiff*(templateImage, sourceImage, i, j, r\_2, c\_2) / temp\_Size;

abs\_Diff\_Mat[i][j] = abs\_Diff;

**synchronized** (minimum) {

**if** (abs\_Diff < minimum[0]) {

minimum[0] = abs\_Diff;

}

}

}

}

**return** **null**;

}));

}

**for** (Future<Void> future : futures) {

future.get();

}

executor.shutdown();

**double** threshold = 10 \* minimum[0];

System.***out***.println("Minimum absolute difference (multi-thread): " + minimum[0]);

System.***out***.println("Threshold (multi-thread): " + threshold);

BufferedImage resultImage = **new** BufferedImage(sourceImage.getColorModel(),

sourceImage.copyData(**null**), sourceImage.isAlphaPremultiplied(), **null**);

**for** (**int** i = 0; i <= r\_1 - r\_2; i++) {

**for** (**int** j = 0; j <= c\_1 - c\_2; j++) {

**if** (abs\_Diff\_Mat[i][j] <= threshold) {

*drawRectangle*(resultImage, i, j, r\_2, c\_2);

}

}

}

**return** resultImage;

}

// This method computes the absolute difference between two images at a given position:

**private** **static** **double** computeAbsDiff(BufferedImage templateImage, BufferedImage sourceImage, **int** startX, **int** startY, **int** height, **int** width) {

**double** abs\_Diff = 0.0;

// Computing the sum of absolute differences between the corresponding pixels in the two images.

**for** (**int** x = 0; x < height; x++) {

**for** (**int** y = 0; y < width; y++) {

**int** template\_RGB = templateImage.getRGB(y, x);

**int** source\_RGB = sourceImage.getRGB(startY + y, startX + x);

abs\_Diff += Math.*abs*(((template\_RGB >> 16) & 0xFF) - ((source\_RGB >> 16) & 0xFF)); // Red

abs\_Diff += Math.*abs*(((template\_RGB >> 8) & 0xFF) - ((source\_RGB >> 8) & 0xFF)); // Green

abs\_Diff += Math.*abs*((template\_RGB & 0xFF) - (source\_RGB & 0xFF)); // Blue

}

}

**return** abs\_Diff;

}

// Drawing the rectangles around the needed shapes for the output image:

**private** **static** **void** drawRectangle(BufferedImage image, **int** startX, **int** startY, **int** height, **int** width) {

Graphics2D g2d = image.createGraphics();

g2d.setColor(Color.***RED***);

g2d.drawRect(startY, startX, width, height);

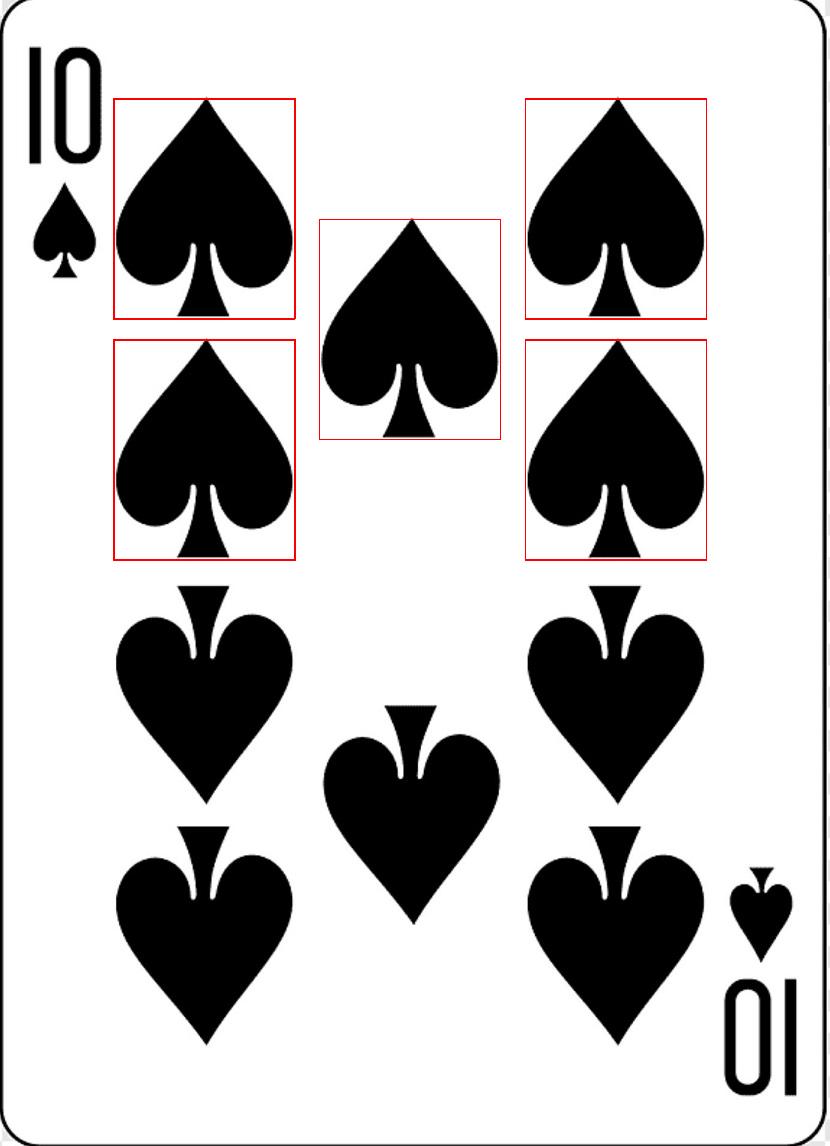
g2d.dispose();

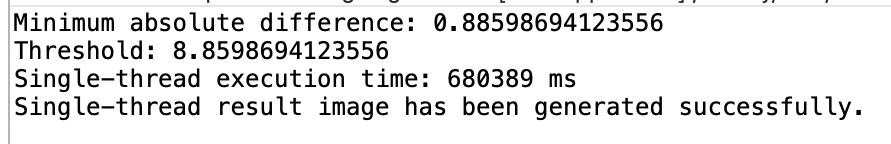
}

}

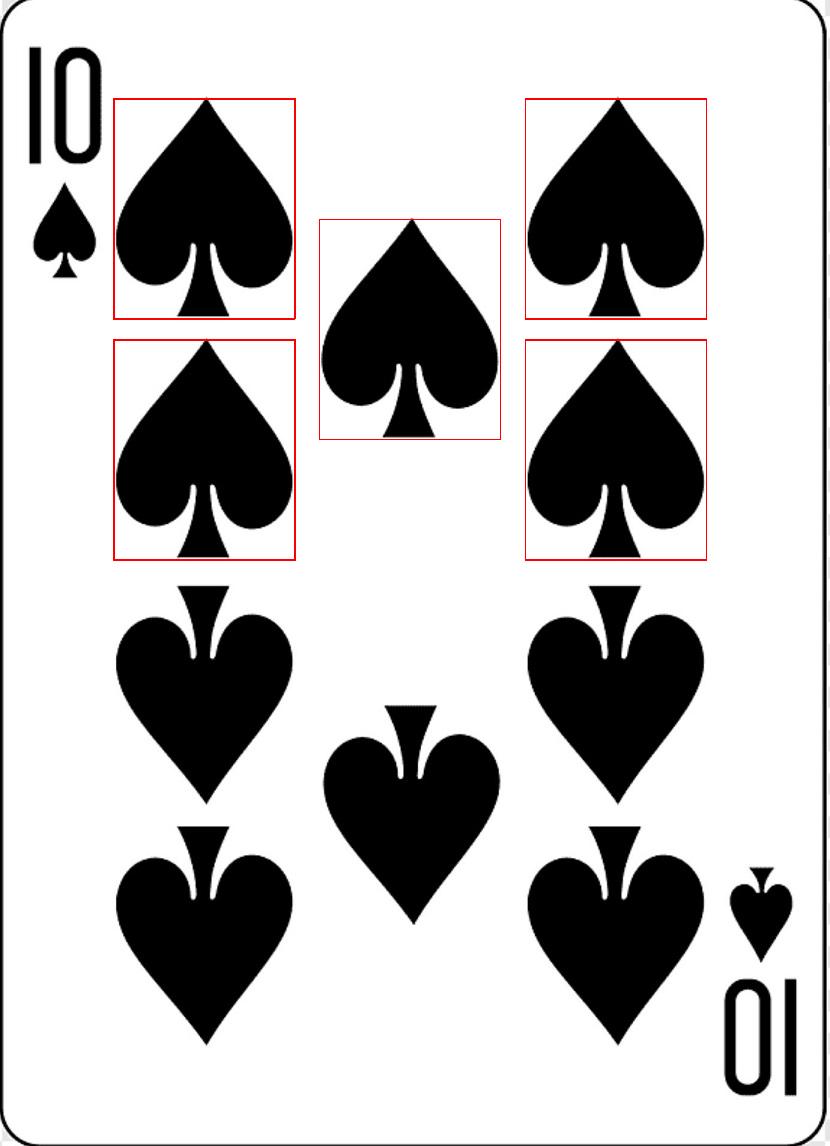
**Output**

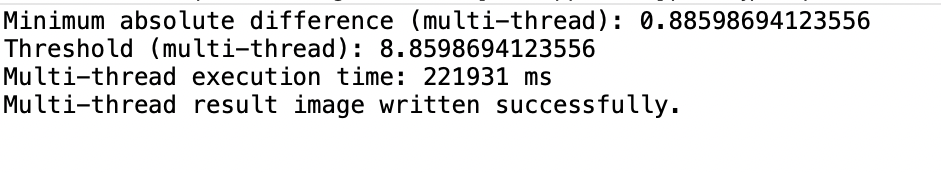
**For the Single-Thread:**

****

****

**For the Multi-Thread:**

****



**Discussion**

* The **single-thread** implementation performs the **following steps**:

1. Creates a 2D array called abs\_Diff\_Mat and initializes it with the absolute differences between the source and template images.   
2. Uses a nested loop to calculate the absolute differences at each position between the template and source images.   
3. Makes use of a synchronized block to update the minimum absolute difference.   
4. In the output image, draws rectangles surrounding the matched areas.

* The **multi-thread** implementation performs the **following steps**:

1. Establishes a fixed number of threads (numOfThreads) in an ExecutorService.   
2. Turns in assignments to calculate the absolute differences at each position between the template image and the source image.   
3. A synchronized block is used by each task to update the minimum absolute difference.

4. Waits for all tasks to complete using future.get().

5. Draws rectangles around the matched regions in the output image.

* **The following are the main distinctions between the multi-thread and single-thread implementations:**

1. All calculations and updates are carried out by a single thread in the single-thread implementation.
2. Several threads are created in the multi-thread implementation to carry out computations simultaneously. After updating a local copy of minimum and abs\_Diff\_Mat, each thread synchronizes with other threads to update the value of the global minimum.
3. By using multiple CPU cores and minimizing conflict for shared resources, the multi-thread implementation may be able to shorten the execution time by employing multiple threads. But because of the synchronization and thread creation, it also adds extra overhead.