Lab 11 – Images and Movies

1 Images

1.1 Shrinking for a different ratio

Write code reads the file **lake.jpg** (download from course website), then shrinks the image along both the x and y axes to 0.7 of the original size.

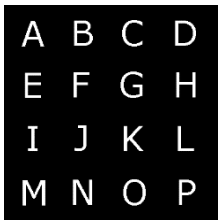
1.2 Many modifications to a picture

This Q was recently used in the Final Exam

Given a photo-quality image called **oldPicture.jpg** (download from course website) that is saved in the same folder as your code, write Matlab code to do the following in this order:

- First, the image should be read from file.
- The top-left and bottom-right quadrants of the image should be extracted, and only the top-left quadrant should change so that the first row of pixels becomes instead the first column of pixels, the second row of pixels becomes the second column of pixels, etc. These quadrants should then be placed beside each other, with the adjusted top-left quadrant on the left and the non-rotated bottom-right quadrant on the right.
- The resultant image should then be magnified to double the size.
- Finally, this image should be saved to a file called **newPicture.jpg**

For example, if **oldPicture.jpg** looked like the image on the left, then **newPicture.jpg** should look like the image on the right:



These images are drawn to scale relative to each other.

Do not use any inbuilt Matlab functions except **imread()**, **imwrite()**, **round()** and **size()**.

Note: When you are doing hard Q's, the most important rule to keep in mind is to "keep it as simple as possible". For example, you should be running your code often (even if you have not finished the exercise), just to see if the most recent changes that you have made are in fact correct and make sense.

You must NEVER write large amounts in one hit, without testing it. Otherwise, you will have too many errors, and it will be extremely difficult to fix. Get into good habits now, follow this rule seriously, so that you can handle the harder Q's in Lab Exam 3.

2 Simple Movies

2.1 Create a single image

Use the techniques taught in lecture 11 to create a single color / heat map (not a movie). The image contains 32 rows and 32 columns, and every pixel in the image is one color, except for a single pixel, which has a different color. This pixel should be at coordinates (1, 4).

As discussed in the lecture, the x value (in this case, 1) represents the column number, and the y value (in this case, 4) represents the row number. So that means when you deal with matrices, you need to give the y value first (not the x value), then the x value (not the y value).

Also, remember that since counting of rows starts from top to bottom, the higher numbered rows are actually further down. This is the opposite of the standard x-y plane in maths, where increasing y values means going up the y-axis. That is, the origin when dealing with matrices and images is actually at the top left.

2.2 Create a simple movie

Now create a movie, where all the 'dot' does is move to the right of the screen. On each iteration of the movie, the dot should move one position to the right. That means the x value increases by 1 each time.

When the dot reaches the right side of the screen, it can just stay there.

The screen should remain the same size of 32 rows and 32 columns for the entire time.

Your movie should run for exactly 100 iterations, no more and no less.

Again, use the example code in lecture 11 to help you do this.

2.3 Create a movie with some calculations

Modify the movie, so that each time the ball reaches a vertical edge, it stays there for a while (15 iterations) before bouncing back and moving in the opposite direction (but still moving horizontally). Run your movie for exactly 200 iterations, no more and no less.

Note: When you are doing hard Q's, the most important rule to keep in mind is to "keep it as simple as possible". For example, you should be running your code often (even if you have not finished the exercise), just to see if the most recent changes that you have made are in fact correct and make sense.

You must NEVER write large amounts in one hit, without testing it. Otherwise, you will have too many errors, and it will be extremely difficult to fix. Get into good habits now, follow this rule seriously, so that you can handle the harder Q's in Lab Exam 3.

2.4 A more complex example

This Q was recently used in Lab Exam 3

In this question, modify the sample solution file from Lab 11, Exercise 2-3 (go to the course website, click on "Laboratory Exercises", then in Lab 11, right click on "Exercise 2-3" and click "Save Target As" or "Save Link As"). [If the sample solutions are not yet up, then use your solution that you wrote for that exercise]

Change the program so that:

- It is written as a function, whose parameter is the filename of a CSV file
- The image represents a map with walls, whose locations are based on the locations of 1's in the CSV file
- The ball starts at row 7, column 2 and is initially moving bottom-right (diagonally)
- The ball can potentially move up and down, not just left and right
- The ball should no longer stick to the edge of the screen when it reaches it, the ball should just bounce back immediately
- The ball bounces off walls in the same way that it bounces off the edge of screen
- When a pixel of a wall has been hit 3 times by the ball, that pixel of the wall disappears
- Each time a pixel of the wall is hit by the ball, it changes color
- Run your movie for exactly 90 iterations

Your program should continue to do everything else the same as in Lab 11, Exercise 2-3.

Do ***not*** assume that the CSV file will contain walls on the edge of the screen.

For simplicity, you may assume that all walls that are parallel to each other (including edge of image) are at least 2 pixels apart; this means that the ball can always move diagonally.

For simplicity, you may also assume that the ball never hits the sharp edge of a corner.

You can download a sample CSV file called **map.csv** from the course website, which is available in the lab 11 exercises.

Note: The hardest Q in Lab Exam 3 will be similar in style and difficulty to this Q. Included in the 8 marks for this Q will also be Comments and Coding Style. This means that even if your program works, it may not score 8 / 8, and may even score very few marks if the code has been written to only work in one specific case.

To score significant marks, your solution should work in the general case, so that if the Q changed slightly, there would be very little modification needed to get it to work in that new case. And of course, you should still satisfy the marking scheme for Comments and Coding Style (see Lab Exam 3 Preview).

2.5 An even more complex example

In this question, modify the sample solution file from Lab 11, Exercise 2-4 (go to the course website, click on "Laboratory Exercises", then in Lab 11, right click on "Exercise 2-4" and click "Save Target As" or "Save Link As"). [If the sample solutions are not yet up, then use your solution that you wrote for that exercise]

Change the program so that the two paragraphs beginning with "for simplicity" are removed. That is, there now may be walls that are parallel to each other (including edge of image) and they are only 1 pixel apart; this means that the ball may not always be able to move diagonally. The ball may now also hit a sharp corner of a wall.

Run your movie for exactly 300 iterations.

Your program should continue to do everything else the same as in Lab 11, Exercise 2-4.

You can download a sample CSV file called **tunnel.csv** from the course website, which is available in the lab 11 exercises.

Note: Although your questions in Lab Exam 3 will not have this much code, this is still a good exercise and good practice for Lab Exam 3

2.6 Goalkeeper

This Q was recently used in Lab Exam 3

In this question, modify the file **bouncingBallMovie** from Lecture 11 (go to the course website, click on "Lectures", then on the right of Lecture 11, right click on "Sample Files" click "Save Target As" or "Save Link As", then extract the file as shown to you in the labs).

Change the program so that:

- The movie runs for 300 iterations
- The ball is initially moving diagonally bottom-right
- In addition to the ball, there is also a goalkeeper of 5 horizontal pixels in a row; each pixel of the goalkeeper is always on the screen
    - You may assume that the goalkeeper always has a width which is an odd number of pixels, and a minimum of 3 pixels
- The goalkeeper is always on the 30th row of the image, directly below the ball, with the center of the goalkeeper directly below the ball wherever possible (if not possible, then as close as possible)
- The ball can bounce off the goalkeeper just like how it bounces off the edge of the screen

Your program should continue to do everything else the same as in the bouncing ball movie in lecture 11.

3 More Complex Images and Movies

If you get stuck on the next 3 exercises, study the sample solutions and notice how they have simplified things. Then try again, but not while looking at the sample solutions.

The Q's in Lab Exam 3 will not be as long as these, but you may be asked to modify the sample solutions to do something different, or write something of similar difficulty (although shorter in amount of code).

After passing ENGG1801, you will have to write similar amounts of code of roughly the same complexity as these exercises. As you can see from the sample solutions, it's just a matter of breaking up your problem into many smaller functions, and using all the things that we have learnt so far in this course. You can definitely do it!

3.1 Smoothing images

This Q is based on a project used in previous semesters

Write code that:

- Reads the file **astro001.png** downloaded from the course website
- Displays the original image on screen
- Creates another image, where each pixel at a location in the image is the average of the 9 pixels surrounding at that location in the original image
    - If a pixel is on the edge of the image, then it does not need to be smoothed, it should remain unchanged
- Displays this new smoothed image

Recall from lecture 11 that images contain "special" types of numbers that can only have values between 0 and 255. So if you add up two of these "special" numbers up, you still have a "special" number that can only be between 0 and 255.

Therefore, you cannot find the average by first adding them up, you should just put all of the values that you want to average into an array, then give that array to the **mean** function, which returns the average of values in the given array.

There are many parts to this question, so it will be much easier if you break this question up into several functions, where each function just does one thing.

Hint: You should have 2 separate images:

- The first image is the original image, which you never change – that means you never put anything into this matrix
- The second image is the new smoothed image, which you must create as you go through the original image.

You need 2 separate images because if you have only one image, you will change it as you are looking at it, so later calculations will be incorrectly influenced by earlier calculations.

3.2 Gas leak

This Q is based on a project used in previous semesters

Create a movie that shows how gas spreads through a room when there is a gas leak.

First read the file **plan1.csv** (download from course website), which represents the map of a room. 1's represent walls, and 0's represent empty space which can be filled with gas.

Then select a location in the room where a gas leak occurs. You can assume that the concentration of gas at this location is a fixed amount.

During each step of your simulation, the amount of gas at a square will equal the average of gas concentrations of the surrounding 9 squares on the previous step of the simulation.

Hint: You should have 3 separate matrices:

- The first matrix represents the map, so it contains only 1's and 0's
- The second matrix only contains numbers that represent the concentrations of gas at each square, on the current step of the simulation
- The third matrix only contains numbers that represent the concentrations of gas at each square, on the next step of the simulation

The reason you want 3 separate matrices is because if you had just one matrix, it will be ambiguous whether a 0 or 1 represents empty space / walls, or whether that means the concentration of the gas at that location is 0 or 1.

You need separate matrices for the current step and the next step of the simulation is because as you calculate what the gas concentration will be at a particular square on the next step of the simulation, this new number should not be used to influence the calculation for other squares.

3.3 Random-moving robot

This Q is based on a project used in previous semesters

Create a movie that shows how a robot can randomly move around a map.

First, download the file **wharf.csv** from the course website.

Write code that reads the above CSV file, which represents the map of a wharf. 1's represent walls, and 0's represent empty space where the robot can move to.

Then select a location in the room where the robot starts, and another location where the robot is trying to get to (the goal).

During each step of your simulation, the robot must move either up, down, left or right (it cannot move diagonally), and each direction should be of equal probability.

Your movie should also show different colors for different squares to represent how many times the robot has been at that location.

Hint: You should have 2 separate matrices:

- The first matrix represents the map, so it contains only 1's and 0's. This matrix will also contain other numbers, where each number represents a physical object, such as the robot, the start location, and the goal location
- The second matrix only contains numbers that represent how many times the robot has visited each square

The reason you want 2 separate matrices is because if you had just one matrix, it will be ambiguous whether a 0 or 1 represents empty space / walls, or whether that means the robot has visited that location 0 or 1 times.

4 Making Fun Computer Games

(Note: Sample solutions will not be provided for this question, as it is more of an extension exercise for fun)

Since a computer game is just a movie that can react to user input, you can now create very basic computer games, similar to the classics:

- **Pac-Man**
  - A dot moves around the screen, based on what the user types. Different letters means different directions. The user must press **enter** each time.
  - The moving dot (Pacman) tries to run into stationary dots (food), and when it does, it eats the food, which are then removed from the screen
  - If Pac-Man runs into a ghost (other dots moving randomly), the game is over
  - https://www.youtube.com/watch?v=uswzriFIf_k
- **Snake**
  - The snake is not just a dot, but a long snake and moves around the map
  - The snake tries to run into stationary dots (food) or go through gaps in walls
  - As the snake eats more food, it gets longer
  - If the snake crashes into itself or the walls, the game is over
  - https://www.youtube.com/watch?v=kZr8sR9Gwag
- **Tetris**
  - Blocks composed of many single blocks of different shapes fall and when you create a complete row of single blocks, the row disappears and the blocks on higher rows fall down
  - If the blocks stack up too high, the game is over
  - https://www.youtube.com/watch?v=umt4ZESWSeQ
- **Arkanoid**
  - The player controls a small horizontal plate that can only move horizontally, positioned near the bottom of screen
  - The ball bounces off this plate at different angles depending on which part of the plate it hits
  - When the ball hits blocks at the top of screen, the ball bounces off, and the blocks disappear
  - If the ball falls past the plate, the game is over
  - https://www.youtube.com/watch?v=Th-Z6QQ5AOQ

After the game is over, you can create a movie **(.avi** file) of what happened and then replay it.

Using only what we have learnt so far in the semester, we can only create very simple games. The user will have to press **Enter** each time they want to do something. And your program must wait for the user to press **Enter** before you can change the image on screen.

To create more fun games, such as where the screen can change image without waiting for the user to press something, and where the image will also change if the user types something, you will need to use more advanced computer science techniques not taught in this course. This is what we teach you in other courses at the School of CS! ☺