

Combining Images: action shots and action removal

Manipulating a sequence of images to compile an action shot or remove the action.

Project due date: You will need to upload your code before

11.59pm on Friday the 13th of September.

From a sequence of images...



To an action shot



OR

The action removed



NOTE: This is version 1 of the project document. If you find a typo please post it to the Piazza project typo thread. If any significant typos are found a newer version of the project will be uploaded to Canvas, with those typos corrected.

Introduction

This project requires you to write code that will allow you to create two different kinds of images from an arbitrary sequence of images that feature someone (or something) moving through a static scene. These images could be from a series of photos or a selection of frames taken from a movie.

One kind of image you will create is known as an “Action shot”, which is a single image that features the moving person or object shown in different locations. These kinds of images are sometimes used in extreme sport magazines, in order to show people moving over a jump or performing some other kind of action. See <https://en.wikipedia.org/wiki/ActionShot> for a few examples. Action shots are also popular in astronomy to show the motion of celestial objects such as the moon, as they track across the sky, as shown in the eclipse scene below.



The other kind of image you will create is one where the action has been removed, eliminating the moving person (or object) from the image, leaving only the static scene visible. A static scene could be useful if you want to remove a pesky pedestrian from an image.

You have been provided with a main script and a few useful helper functions to get you started. On completion of the project you will have written code that allows you to produce both types of images from an arbitrary sequence of images which show action against a static background. These could be a series of still photos or frames from a short movie clip.

You will be supplied with a few image sets to work with but it is expected that you will create your own small images and arrays to test your code (make sure you work out the answers on some **very** small sets of test data by hand, so you know what you should get). When creating test data you could even go as small as an image with just a couple of rows and columns.

A few days prior to the submission date you will be supplied with some test data and test scripts which can be used to help check your code. If your code fails some of the supplied test code there is obviously work to be done. If your code passes the supplied test scripts it is a good sign you are on the right track but please note that the supplied test code isn't identical to that used to mark your code. If you rely just on the supplied test code (without doing tests of your own) you will run the very real risk of submitting code that will not pass all the marking tests. Do NOT rely solely on the supplied test scripts, part of doing the project is to learn how to extensively test your own code on a range of different values (which is why the supplied test scripts are only released at the end of the project).

This project is designed to take the average student around **15 hours** to code up but some people may find it takes upwards of 60 hours (and others will finish it in just a few hours). You will not know ahead of time how long it will take you, so please don't leave it to the last minute to make a start!

Remember the golden rule

Write your own code and don't give your code to anyone. Friends don't let friends share code. This project needs to be YOUR work, it is an individual project, not a group project. Copying and/or sharing code is academic misconduct. Please note that every project submission is passed through software that detects plagiarism so if you copy **you WILL be caught**. Unfortunately every year students discover the hard way that I'm not kidding. Let me repeat,

DO NOT COPY: YOU WILL BE CAUGHT.

If you give your code to another student and they copy it, you will both be guilty of misconduct. Copying or supplying code typically results in both the person who copied and the person who supplied the code being awarded a mark of zero for the project, as well as your names going on the academic misconduct register. If you have already been found guilty of academic misconduct on another course or assignment the penalty may be even greater (e.g. a fine or fail of the course).

To help you avoid academic misconduct I have put together a very short best practices guide which I recommend you read. It is attached to the end of this document.

If you have any queries about what is or isn't academic misconduct, please ask so that I can make sure everyone understands what is acceptable behaviour and what isn't.

How to tackle the project

Do not be daunted by the length of this document. It is long because a lot of explanation is given as to exactly what each function needs to do.

The best way to tackle a big programming task is to break it down into small manageable chunks. A lot of this work has already been done for you in the form of eight functions to write. Each function has a detailed description of what it needs to do and many can be written independently of each other. Have a read through the entire document and then pick a function to start on. Remember you don't have to start with `GenerateFrameList`, although it is one of the simplest functions. You may prefer to start with one of the other fairly straight forward functions such as `PixelDistance` (which can be written in just a few lines of code with a bit of cunning).

Write as many of the functions as you can. When writing some of the specified functions you may find it useful to call one of the other functions you have written. You may even wish to write additional helper functions that have not been listed in the specifications. This is absolutely fine (in fact it is good coding practice to do so). Make sure you remember to submit any extra helper functions you write as they will be needed so that your code runs correctly. Please note that you will be able to submit up to 16 functions in total, which leaves room for up to 8 helper functions (which should be ample).

Several functions require careful thought, particularly those that form the heart of creating the action shot (remember those 5 steps of problem solving!). If you are having trouble understanding how a function should work, remember to work through the problem by hand with a small data set.

Note that I don't expect everyone to get through all of the functions. Some of them are relatively easy (e.g. `GenerateFrameList` and `PixelDistance`) while others are quite tricky (e.g. `GenerateImageList`). You can still get a pretty good mark even if you don't complete all of the functions.

You will receive marks both for correctness (does your code work?), style (is it well written?) and execution time (how fast does it run?). An "A" grade student should be able to nut out everything except perhaps full marks for execution time (it can be **extremely** challenging to get things running quickly on larger images). B and C grade students might not get a fully working solution. Even if you only get three of the functions working, you can still get over 50% for the project, as long as the code you submit is well written (since you get marks for using good style).

What Matlab functions do I need to know?

Almost the entire project can be completed using just the functions we have covered in the course manual, lectures and labs. Note however that exploring unfamiliar functions and getting to grips with additional features of a language is an important part of coding so I have designed one task (writing `GenerateImageList`) which will require you to explore some unfamiliar functions (e.g. `dir`) and teach yourself some new ideas (including how to access elements of a struct in Matlab). Remember that documentation, google and youtube are great resources when trying to learn about new ideas. You can also refer to library books (there are a huge number of books in the library on Matlab).

When writing the other functions, while you don't need to use functions we haven't covered in class, you may wish to do so.

What Matlab functions can I use?

You can use any functions we have covered in class and any other functions that are part of Matlab's **core** distribution (i.e. not functions from any toolboxes you may have installed). Note that for this project *the use of functions that are only available in toolboxes is not permitted*. Matlab features a large number of optional toolboxes that contain extra functions. The purpose of this project is for you to get practice at coding, not to simply call some toolbox functions that do all the hard work for you.

A list of some of the core Matlab functions can be found on pages 285 to 287 of the course manual. If you are uncertain whether a particular function is part of the Matlab's core distribution or not, type the following at the command line

```
which <function>
```

where the term *<function>* is replaced by the name of the function you are interested in. Check the text displayed to see if the directory directly after the word `toolbox` is `matlab` (which means it is core) or something else (which means it is from a toolbox).

For example:

```
>> which median
```

```
C:\Matlab\R2017b\Pro\toolbox\matlab\datafun\median.m
```

This shows us that the `median` function is part of the standard core Matlab distribution (notice how the word `matlab` follows the word `toolbox`). You may use the `median` function if you wish to.

```
>> which imadd
```

```
C:\Matlab\R2017b\Pro\toolbox\images\images\imadd.m
```

This shows us that the `imadd` function is part of the image processing toolbox and is therefore not permitted to be used for the project (notice how the word `image` follows the word `toolbox`).

An overview of creating action shots and removing action.

Before we delve into some more detail, let's take a quick high level outline of how removing action and creating action shots works. Removing action is conceptually a little simpler, so we will begin with that.

Removing Action

Suppose we have the following three images stored in a directory



We want to remove the pedestrian and obtain only the static background. Imagine that we stack the three images on top of each other. If you pick a point on the top image there will be a pixel with a particular set of RGB values at that position (e.g. the pixel in row 100, column 20). We can then move through the stack of images and find the other pixels that are in the corresponding position. This will give us a little stack of pixels, some of which may be from the pedestrian and some of which may be from the background. If the pedestrian moved around a lot it is likely that the majority of the pixels will be from the static background (and will be very close to the same colour – although likely not identical as the lighting may have changed a little between photos). We want to pick one of the pixels from the background. If we were able to sort the pixels in order (using their colour values) and then pick the middle one, then we should get one from the static background (rather than the pedestrian). One way to calculate a “middle pixel” is to calculate the median value for each of the three colours of the pixel in turn. E.g. to get the red value of the “middle pixel”, we would take the median value of all the red values in our little stack of pixels. To get the green value of the middle pixel we would take the median value of all the green values in our little stack of pixels and finally we would get the blue value by taking the median value of all the blue values in our little stack of pixels. Once we have the **median pixel** values we can write those colour values to a new image in an appropriate position (in our example they would go be the colour values for row 100, column 20 of our new image).

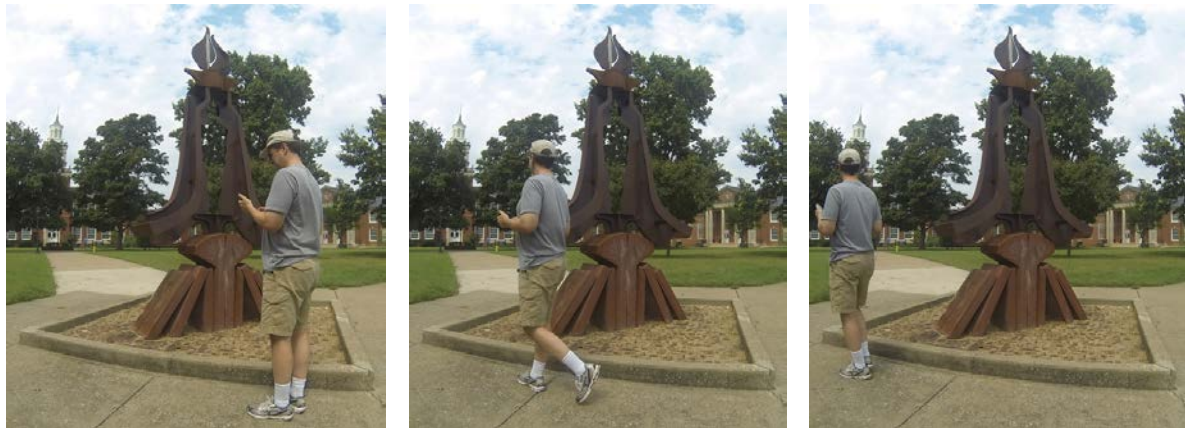
If you take the above idea and iterate over all the points in our stack of images you will obtain a filtered image that should have the pedestrian removed, as shown on the right.



Creating Action shots

To create an action shot we want to ensure that the portions of each image that show a moving person (or object) all end up on the final image.

Suppose we have the following three images stored in a directory



As before, imagine that we stack the three images on top of each other and we pick a point on the top image, where there will be a pixel with a particular set of RGB values at that position (e.g. the pixel in row 100, column 20). As before we can then move through the stack and find the other pixels that are in the corresponding position to give us a little stack of pixels. If the pedestrian moved around a lot some of those pixels may be from the pedestrian but the majority of the pixels will be from the static background. This time we WANT to pick a pixel from the pedestrian (if there is one).

Recall that if we sort the pixels in order (by colour) and then calculate the median pixel we should get one from the static background (rather than the pedestrian). If there is a pixel from the pedestrian it is likely to be very different from the background. This time we want to choose the pixel from our little stack that is the most different, i.e. **the most distant pixel** from the median pixel in terms of colour. This requires us being able to measure how far away each pixel from our stack is from the median pixel (in terms of colour). If we think about the three colour values of each pixel as points in 3D space, then one way to measure how far away a pixel is from another pixel is to calculate the distance between the two points in this 3D space (this can be done using Pythagoras or linear algebra calculations). An even simpler measure of **pixel distance** is to calculate the squared distance (this is likely to work just as well and may be faster as we avoid having to calculate square roots if using Pythagoras)

Once we have found the pixel from our stack that is the most distant from the median pixel, we can write its colour values to a new image in an appropriate position (in our example they would be the colour values for row 100, column 20 of our new image).

If you take the above idea and iterate over all the points in our stack of images you will obtain a filtered image that should have all the action included, as shown on the right.



Overview of Functions to Write

There are eight in total:

Function	Difficulty Level	Where used
<code>GenerateFrameList</code>	Very easy, a good starting point	Called from the main script
<code>GenerateImageList</code>	Challenging, requires you to learn some new ideas!	Called from the main script
<code>ReadImages</code>	Fairly easy	Called from the main script
<code>RemoveAction</code>	Fairly challenging	Called from the main script
<code>ActionShot</code>	Fairly challenging but very similar to <code>RemoveAction</code>	Called from the main script
<code>PixelDistance</code>	Fairly easy, another good starting point	Helper function, likely to be called from <code>MostDistantPixel</code>
<code>MedianPixel</code>	Fairly easy	Helper function, likely to be called from <code>MostDistantPixel</code> and <code>RemoveAction</code>
<code>MostDistantPixel</code>	Fairly challenging	Helper function, likely to be called from <code>ActionShot</code>

`GenerateFrameList`, `GenerateImageList`, `ReadImages`, `RemoveAction` and `ActionShot` are all called directly from the supplied main script file. You will need to implement these functions before being able to use the supplied script file to generate the two types of images.

The remaining three functions mentioned above are not called directly from the supplied main script file but should prove helpful to call when writing some of the other functions (even if you decide not to call them in this way you should still write them as they will be marked). Note that some of these “helper” functions are relatively easy to write so are a good starting point (you can leave the more challenging ones until after these have been written).

The pages following have detailed specifications of what each function should do. Note that while the functions are designed to work together with the supplied main script, to allow the creation of a action shots and images with action removed, many of the functions can be written and tested in isolation. Even if you get stuck on one function you should still try and write the others.

Keep the following general principles in mind:

- Note the capital letters used in function names. Case matters in Matlab and you should take care that your function names EXACTLY match those in the projection specification. E.g. if the function name is specified as `PixelDistance` don't use the name `pixeldistance` or `pixel_distance`.
- The order type and dimension of input arguments matters. Make sure you have the required number of inputs in the correct order. E.g. if a function requires 2 inputs, as `GenerateImageList` does (where the first input is a string containing a directory and the second input is a string containing a file extension type) then the order matters (i.e. the first input should be the directory not the file type, to match the specifications)
- The order, type and dimensions of the output matter, e.g. `MedianPixel` returns three outputs, the first of which is a red value, then a green value then a blue value. Muddling up the order would lose you all the marks for this function.

The GenerateFrameList function

Purpose	GenerateFrameList generates a list of frames we are interested in which can be used by other functions. In particular it will be useful for determining which frames to extract from a movie file.
Input(s)	It takes three inputs in the following order: <ol style="list-style-type: none"> 1) A starting frame number 2) A step size 3) The number of frames to generate (n)
Output(s)	It returns a single output, a 1xn 1D array, where n is the desired number of frames (n). The first element of the array will be the starting frame number and each subsequent element will have a frame value that is the step size greater than the last.

Example calls

Here are some examples of calls to `GenerateFrameList`

```
>> frameNumbers = GenerateFrameList (1,4,3)

frameNumbers =

     1     5     9

>> frameList = GenerateFrameList (10,20,4)

frameList =

    10    30    50    70

>> frameValues = GenerateFrameList (20,1,5)

frameValues =

    20    21    22    23    24
```

The GenerateImageList function

Purpose	Fetch a list of the names of all the images with a particular file extension that are contained in a specified directory
Input(s)	It takes two inputs in the following order: <ol style="list-style-type: none"> 1) A string containing the name of the directory the images are contained in 2) A string containing the file extension of the images to fetch
Output(s)	It returns one output: A 1xn 1D cell array containing n strings where each element is the filename of an image from the specified directory that has a particular file extension

Example Calls

Suppose the following directory exists on my computer:

C:\Users\peter\Documents\pics

And it contains the following five files

Moon1.png
Moon2.png
Unicycle1.jpg
Unicycle2.jpg
Unicycle3.jpg

Here are some examples of calls to `GenerateImageList`

```
>> filenames = GenerateImageList('C:\Users\peter\Documents\pics','png')
```

The output assigned to the variable `filenames` would be a 1x2 cell array where the first element will contain the string `'Moon1.png'` and the second element will contain the string `'Moon2.png'`.

```
>> filenames = GenerateImageList('C:\Users\peter\Documents\pics','jpg')
```

The output assigned the variable `filenames` would be a 1x3 cell array where the first element will contain the string `'Unicycle1.jpg'`, the second element will contain the string `'Unicycle2.jpg'` and the third element will contain the string `'Unicycle3.jpg'`

Note if our working director was set to `C:\Users\peter\Documents` and this contained the subdirectory `pics` then we could just use the following call to achieve the same result:

```
>> filenames = GenerateImageList('pics','jpg')
```

(i.e. you don't need the full directory path).

IMPORTANT NOTE

One of the reasons `GenerateImageList` is a more difficult function to write is that it is going to require you to do a little research on how to get hold of the names of files within a specified directory. This is a relatively common task to do when trying to process a bunch of files but we haven't discussed it in class. Exploring unfamiliar functions and syntax and getting to grips with them is an extremely important part of coding and this task tests that ability. Armed with the Matlab documentation and a little online searching you should be able to teach yourself the required skills. In addition to the online Matlab documentation there are many websites, forums and videos which have discussions on how to solve common programming tasks, which could prove useful. Remember however that you want to teach yourself the required skills and understand them, rather than just copying some cryptic piece of code that you don't understand. You don't need to reference the Matlab help or documentation but if you find another useful website that you use concepts from, please remember to reference it in your comments.

To point you in the right direction for your research on this task:

- 1) You may find the **`dir`** function very useful. Check out the Matlab documentation on what the `dir` function does (remember the documentation typically gives you example fragments of code which may prove very helpful).
- 2) You will need to figure out how to access elements from a Matlab structure array. The Matlab documentation on `struct` may prove useful

The ReadImages function

Purpose	Read in a specified list of images given the filenames and the directory the files are located in.
Input(s)	It takes two inputs in the following order: <ol style="list-style-type: none"> 1) A string containing the name of the directory the images are contained in 2) A 1xn 1D cell array containing n strings where each element is a filename of an image to read
Output(s)	It returns one output A 1xn 1D cell array containing n images, where each element is an RGB image (recall RGB images stored as 3D arrays of uint8 values ranging from 0 to 255). The first image will correspond to the first filename from the list of filenames

Example Calls

Suppose the following directory exists on my computer:

C:\Users\peter\Documents\pics

And it contains the following five files

Moon1.png
Moon2.png
Unicycle1.jpg
Unicycle2.jpg
Unicycle3.jpg

Here are some examples of calls to `ReadImages`

```
>> filenames = {'Moon1.png', 'Moon2.png'}
>> pics = ReadImages('C:\Users\peter\Documents\pics', filenames)
```

The output assigned to the variable `pics` would be a 1x2 cell array. The first element of the array would contain a 3D array with the colour values for Moon1.png and the second element would contain a 3D array with the colour values for Moon2.png.

In this instance if you wanted to display the first image with its name as a title you could do the following:

```
>> image(pics{1}) % note curly braces
>> title(filenames{1}) % note curly braces
```

```
>> filenames = {'Unicycle1.png', 'Unicycle2.png', 'Unicycle3.png'}  
>> pics = ReadImages('C:\Users\peter\Documents\pics', filenames)
```

The output assigned to the variable `pics` would be a 1x3 cell array. The first element of the array would contain a 3D array with the colour values for Unicycle1.jpg, the second element would contain a 3D array with the colour values for Unicycle2.jpg and the third element would contain a 3D array with the colour values for Unicycle3.jpg.

In this instance if you wanted to display the third image with its name as a title you could do the following:

```
>> image(pics{3}) % note curly braces  
>> title(filenames{3}) % note curly braces
```

If your Matlab working directory was set to `C:\Users\peter\Documents` and this directory contained the subdirectory `pics`, then you could achieve the same result as above by using

```
>> pics = ReadImages('pics', filenames)
```

IMPORTANT NOTE

You've already met the idea of cell arrays in the context of storing strings (i.e. 1D arrays of characters) in cell arrays (see page 150 of your course manual). Cell arrays can contain other types of data too, including 1D arrays of numbers, 2D arrays and 3D arrays. There is no problem with assigning a 3D array of RGB colour values to an element of a cell array.

A 1D cell array is a very convenient way of storing a sequence of images (where each element of the cell array contains a 3D array representing an image).

The PixelDistance function

Purpose	Calculates the square of the distance between two points in colour space
Input(s)	It takes two inputs in the following order: <ol style="list-style-type: none"> 1) An array containing three elements representing a point in 3D colour space 2) An array containing three elements representing a second point in 3D colour space
Output(s)	A single output, the square of the distance between the two points in 3D colour space.

Notes

The distance referred to is the geometric straight line distance between the two points when interpreting them as points in 3D space. This function returns the SQUARE of this distance, so mathematically it is equivalent to the following calculation for two 3D points, P and Q

$$D = (P_1 - Q_1)^2 + (P_2 - Q_2)^2 + (P_3 - Q_3)^2$$

Note that the colour values for a pixel can be interpreted as a point in 3D space $P = (P_1, P_2, P_3)$

The three coordinates are the colour values for the pixel, i.e. the first coordinate P_1 is the amount of red, the second P_2 is the amount of green and the third P_3 is the amount of blue.

You may assume that both P and Q will both be the same size.

Your code should be written to handle the case where the two points P and Q are 1D arrays (e.g. both having 1 row and 3 columns or both having 3 rows and 1 column)

In addition your code should also handle the case where P and Q are both 1x1x3 arrays (i.e. a point may be passed in as a 3D RGB image having 1 row 1 column and 3 layers, where each layer contains a colour value).

WARNING: Your code should handle the inputs being passed in as uint8 values. When doing distance calculations with uint8 values it is entirely likely you will get answers greater than 255, so you will want to convert your uint8 values to doubles before doing any calculations (as described on page 254 of your manual).

Worked example

Suppose we had the two points

$$P = (0,0,0) \text{ and } Q = (3,4,0)$$

Then the squared distance would be given by

$$D = (0 - 3)^2 + (0 - 4)^2 + (0 - 0)^2$$

$$D = 9 + 16 = 25$$

Note that the square distance can never be negative (as it is a squared value).

Example calls

Here are some examples of calls to `PixelDistance`

```
>> P = [0 0 0] % this as a 1x3 1D array
>> Q = [255 255 255] % this as a 1x3 1D array

>> % find the squared distance between P and itself
>> squaredDistance = PixelDistance(P,P)
```

This will produce the result:

```
squaredDistance =

    0
```

```
>> % find the squared distance between P and Q
>> squaredDistance = PixelDistance(P,Q)
```

This will produce the result:

```
squaredDistance =

   195075
```

```
>> P = [0; 0; 0] % this as a 3x1 1D array
>> Q = [3; 4; 0] % this as a 3x1 1D array

>> % find the squared distance between P and Q
>> squaredDistance = PixelDistance(P,Q)
```

This will produce the result:

```
squaredDistance =

    25
```

Suppose we have a 3D array of colours containing the following values:

Colours(:, :, 1) =	Colours(:, :, 2) =	Colours(:, :, 3) =
192	50	135
66	73	203
65	192	135

Then we could calculate the squared distance between the first pixel and the third pixel as follows:

```
>> P = Colours(1,1,:) % this is the first pixel, in row 1 column 1
>> Q = Colours(3,1,:) % this is the third pixel in row 3, column 1

>> % find the squared distance
>> squaredDistance = PixelDistance(P,Q)
```

This will produce the result:

```
squaredDistance =

   36293
```

The MedianPixel function

Purpose	MedianPixel calculates the median RGB values from a list of pixels.
Input(s)	It takes one input: A 1xn3 3D array of RGB values representing a list of pixels (pixel 1 will be in column 1, pixel 2 in column 2 etc). Typically n will be greater than 1 (i.e. there are two or more pixels in the list) but your code should still be able to handle the special case of n being 1.
Output(s)	It returns three outputs in the following order: <ol style="list-style-type: none"> 1) The median red value, which will be the median of all the R values from the list of pixels 2) The median green value, which will be the median of all the G values from the list of pixels 3) The median blue value, which will be the median of all the B values from the list of pixels

Example calls

Here are some examples of calls to MedianPixel

Suppose we have a 1x3x3 array called pixels. Displaying the array gives the following:

```
>> disp(pixels)

(:, :, 1) =
    54    50    45

(:, :, 2) =
    48    52    43

(:, :, 3) =
    50    41    47
```

Here we can see the red values (in layer 1), the green values (in layer 2) and the blue values (in layer 3)

We now call

```
>> [R,G,B] = MedianPixel(pixels)
```

The above call will result in R=50, G=48, B=47

Suppose we have a 1x4x3 array called pixels. Displaying the array gives the following:

```
>> disp(pixels)
(:, :, 1) =
    54    50    48    43
(:, :, 2) =
    48    52    48    44
(:, :, 3) =
    50    41    47    52
```

Here we can see the red values (in layer 1), the green values (in layer 2) and the blue values (in layer 3)

We now call

```
>> [R,G,B] = MedianPixel(pixels)
```

The above call will result in R=49, G=48, B=49

Note that the median value of layer 3 is 48.5, as it is the number between 47 and 50 BUT uint8 values can only be integers. In the case of a non-integer median we will round to the nearest integer using conventional rounding rules (i.e. 48.5 rounds up to 49).

The MostDistantPixel function

Purpose	MostDistantPixel calculates the pixel from a list that is most distant from the median RGB values of that list. The distance metric to be used is that described in the PixelDistance task (i.e. the squared distance between points in RGB colour space)
Input(s)	It takes one input: A 1xn3 3D array of RGB values representing a list of pixels (pixel 1 will be in column 1, pixel 2 in column 2 etc). Typically n will be greater than 1 (i.e. there are two or more pixels in the list) but your code should still be able to handle the special case of n being 1.
Output(s)	It returns three outputs in the following order: 1) The red value of the most distant pixel 2) The green value of the most distant pixel 3) The blue value of the most distant pixel

Example calls

Here is an example of a call to MostDistantPixel

Suppose we have a 1x3x3 array called pixels. Displaying the array gives the following:

```
>> disp(pixels)
(:, :, 1) =
    54    50    45
(:, :, 2) =
    48    52    43
(:, :, 3) =
    50    41    47
```

Here we can see the red values (in layer 1), the green values (in layer 2) and the blue values (in layer 3)

We now call

```
>> [R,G,B] = MostDistantPixel(pixels)
```

Note that the median values for this particular list of pixels are (50,48,47)

The above call will result in R=50, G=52, B=41 as the point in 3D colour space (50,52,41) is more distant from the median values than either of (54,48,50) or (45,43,47)

We can check this by using the pixel distance function

```
d1 = PixelDistance([50,48,47],[54,48,50]) % will be 25
d2 = PixelDistance([50,48,47],[50,52,41]) % will be 52 (most distant)
d3 = PixelDistance([50,48,47],[45,43,47]) % will be 50
```


The RemoveAction function

Purpose	<code>RemoveAction</code> creates an image that has the action removed by applying a median filter (i.e. each pixel in the new image is obtained by taking the median RGB values of the stack of corresponding pixels taken from the source images).
Input(s)	It takes one input: A 1xn 1D cell array containing n images, where each element is an RGB image (recall RGB images stored as 3D arrays of uint8 values ranging from 0 to 255).
Output(s)	It returns one output: An RGB image (stored as 3D arrays of uint8 values ranging from 0 to 255) that was obtained by taking the median RGB values of the stack of corresponding pixels from the source images.

Example calls

Here are some examples of calls to `RemoveAction`

Suppose a cell array called `imageList` contains the three images below



We then make the following call

```
>> actionRemoved = RemoveAction(imageList)
```

`actionRemoved` will be a 3D array than contains the following image



Suppose a cell array called `frameList` contains the three images below (frames 1, 51 and 101 from the `zylophone.mp4`)



We then make the following call

```
>> actionRemoved = RemoveAction(frameList)
```

`actionRemoved` will be a 3D array than contains the following image



The ActionShot function

Purpose	ActionShot creates an action shot image by finding the pixels from a stack of images that are most distant from the median RGB values (i.e. each pixel in the new image is obtained by finding the pixel in the same row and column of the source images that is most distant from the median RGB values of the stack of corresponding pixels).
Input(s)	It takes one input: A 1xn 1D cell array containing n images, where each element is an RGB image (recall RGB images stored as 3D arrays of uint8 values ranging from 0 to 255).
Output(s)	It returns one output: An action shot in the form of an RGB image (stored as 3D arrays of uint8 values ranging from 0 to 255)

Example calls

Here are some examples of calls to ActionShot

Suppose a cell array called `imageList` contains the three images below



We then make the following call

```
>> actionImage = ActionShot(imageList)
```

`actionImage` will be a 3D array than contains the following image



Suppose a cell array called `frameList` contains the three images below (frames 1, 51 and 101 from the `zylophone.mp4`)



We then make the following call

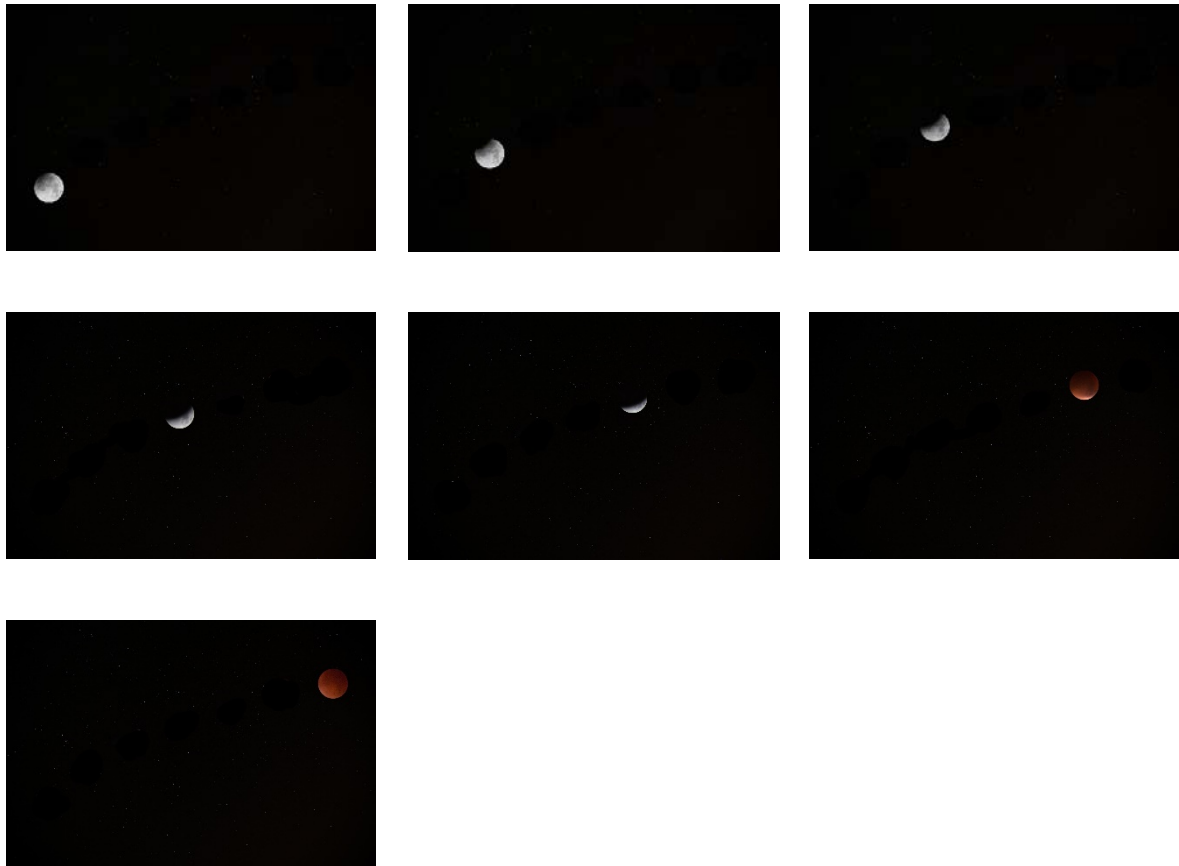
```
>> actionImage = ActionShot(frameList)
```

`actionImage` will be a 3D array than contains the following image



Note the image is a little blurry in parts. This sort of problem can be reduced by using a tripod when filming.

Suppose a cell array called `lunarList` contains the seven images of an eclipse shown below:



We then make the following call

```
>> actionImage = ActionShot(lunarList)
```

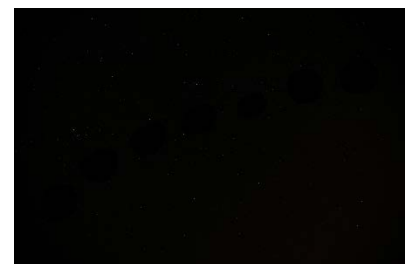
`actionImage` will be a 3D array than contains the image to the right:



For comparison, removing the action in this case, isn't very exciting:

```
>> actionRemoved = RemoveAction(lunarList)
```

`actionRemoved` will be a 3D array than contains the image to the right:



How the project is marked

A mark schedule and some test scripts will be published prior to the due date, outlining exactly how marks will be allocated for each part of the project and giving you the opportunity to test your code. You will receive marks both for correctness (does your code work?), style (is it well written?) and execution time (how fast does it run?).

Each of the **eight** required functions will be marked independently for correctness, so even if you can't get everything to work, please do submit the functions you have written. Some functions may be harder to write than others, so if you are having difficulty writing a function you might like to try working on a different function for a while and then come back to the harder one later.

Note it is still possible to get marks for good style, even if your code does not work at all! Style includes elements such as using sensible variable names, having header comments, commenting the rest of your code, avoiding code repetition and using correct indentation.

Each function can be written in less than 15 lines of code (not including comments) and some functions can be written using just a few lines but do not stress if your code is longer. It is perfectly fine if your project solution runs to several hundred lines of code, as long as your code works and uses good programming style.

How the project is submitted

Submission is done by uploading your code to the Aropa website. More information will be provided on how to submit the project to Aropa in a Canvas email announcement. If you want to be out of Auckland over the break and still want to work on the project that will be perfectly fine, as long as you have access to a computer with Matlab installed and can access the internet to upload your final submission.

Submission Checklist

Here is a list of the **eight** functions specified in this document. Remember to include all eight (or as many of them as you managed to write) in your project submission. The filenames should be EXACTLY the same as shown in this list (including case). They should also work EXACTLY as described in this document (**pay close attention to the prescribed inputs and outputs and their order**). In addition if your functions call any other "helper" functions that you may have written, remember to include them too (you can submit up to a total of 16 files, so may include up to 8 helper functions). The eight required functions are:

- `GenerateFrameList`
- `GenerateImageList`
- `ReadImages`
- `RemoveAction`
- `ActionShot`
- `PixelDistance`
- `MedianPixel`
- `MostDistantPixel`

Any questions?

If you have any questions regarding the project please first check through this document. If it does not answer your question then feel free to ask the question on the class Piazza forum. Remember that you should **NOT** be publicly posting any of your project code on Piazza, so if your question requires that you include some of your code, make sure that it is posted as a **PRIVATE** piazza query.

Have fun!

Enggen131 Good Practice Guidelines

Acceptable

You used some of the project specification text in the header comments for your function

```
% wholeNumberAverage function is responsible for checking whether or not
% the average of the digits is a whole number
% Inputs:    s, an array of numbers representing a sequence to check
% Outputs:   b, zero if the array does NOT average to a whole number,
%            otherwise it returns the whole number average (which corresponds
%            to the number of balls required to juggle the pattern)
```

```
function b = wholeNumberAverage(s)
```

```
% calculate average
```

```
av = mean(s);
```

A friend told you about the mean function, which can be used to find the average of an array. You looked up the matlab help on mean and decide to use it, then WROTE YOUR OWN CODE

```
% check if the rounded average is equal to the original average
```

```
roundedAv = round(av);
```

```
if av > 0 && roundedAv == av
```

```
    b = av;
```

```
else
```

```
    b = 0;
```

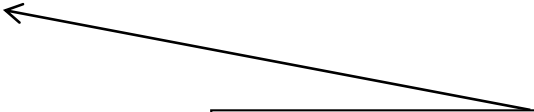
```
end
```

A group of you discussed ideas about how to check if a value is a whole number. As part of the group discussion, everyone figured out that if the original value is the same as the rounded value, then it must be a whole number. You then WROTE YOUR OWN CODE, using this idea.

Acceptable

```
% wholeNumberAverage function is responsible for checking whether or not
% the average of the digits is a whole number
% Inputs:    s, an array of numbers representing a sequence to check
% Outputs:   b, zero if the array does NOT average to a whole number,
%            otherwise it returns the whole number average (which corresponds
%            to the number of balls required to juggle the pattern)
function b = wholeNumberAverage(s)

% check if the rounded average is equal to the original average
% by seeing if the remainder after division by one is positive
% algorithm retrieved from www.mycoolalgorithm.com/aGreatIdea
if rem(s,1) > 0
    b = mean(s);
else
    b = 0;
end
```



You found an algorithm on the internet that described a method for checking for whole number values, by seeing if the remainder after division by one was a positive number.

You credited the source of the algorithm and then WROTE YOUR OWN matlab code to implement it

Unacceptable - DO NOT DO

```
% checks whether a string consists of nonzero digits only
% (i.e. the characters 1 to 9 inclusive)
% Inputs:    s, a string to check
% Outputs:   isValid, a Boolean variable, true if the string contains only
%            non zero digits and false otherwise
function isValid = nonZeroDigitsOnly(s)

isValid = true;
i = 1;

% loop through characters to see if they are within a valid ascii range
while isValid==true && i <= length(s)
    if (s(i) < 49 || s(i) > 57)
        % if out of range, this sequence is invalid
        isValid = false;
    end
    i = i + 1;
end
```

You were unsure what to write for the function header comments so took a photo of your neighbour's work with your iphone and then typed in their comment lines later

You were having trouble late at night with debugging your while loop. You asked your friend about it on msn and they sent through a few lines of their working code, so you used their while loop condition.

You didn't know how to write an if statement that worked, so your friend came to your computer and typed out this line for you

Unacceptable - DO NOT DO

```
% takes a string containing digits and converts it to the corresponding
% array of characters
% Inputs:      c, a string of characters containing only nonzero digits
% Outputs:     n, an array of numbers corresponding to the digits of the string
function [n] = char2num(c)
```

```
% convert to an array of numbers
for i=1:length(c)
    n(i) = str2num(c(i));
end
```

← You didn't know how to write this function, so a friend emailed you their code, which you then copied. To make it look different you changed the wording of the comments and the variable names

```
% generate carry path starting at height 0, going from an initial x position
% to a final x position (specified in terms of cm from origin)
% While this could be a straight line, it looks prettier if the ball dips
% down below the y axis
%
% Inputs:      1) initial x co-ordinate (catch location)
%              2) final x co-ordinate (throw location)
% Outputs:     1) An array of x co-ordinates for the path through the hand
%              2) An array of y co-ordinates for the path through the hand
function [x,y]= generateParabolicPath(xinit,xfinal)
```

```
x = linspace(xinit,xfinal,5);
y = [0 -1 -2 -1 0];
```

← You couldn't write this function by yourself so you and a friend worked on it together. After writing the entire thing together you each took a copy and put it into your project.