Term Project – Testing

**1**. General Utility Class: ImageConversion.py

Specification: Receive image file from the user, if valid the image is reduced or enlarged to a certain pixel width and height. Then it is converted to a black and white image so an array is can be populated with values of only 1 and 0.

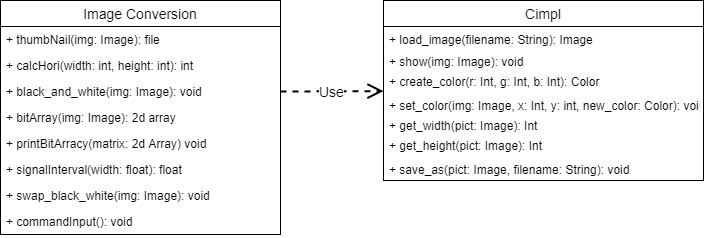
First section is Functionality-Based Input Domain Modeling for ImageConversion.py. The basic principles are; for each function identify the parameters and characteristics of interest, then identify blocks (which are partitions of the characteristics.

Goal: Create test frames with the blocks by following the all combination criterion (all combinations of blocks from all characteristics must be used, excluding [single] and [error]. Then test cases are recorded in the Test plan table.

Special cases:

[single]: Special or redundant conditions that do not have to be combined with all possible choices. It is tested with one test case.

[error]: Assumed that if the parameter has this particular value, any call of the function using that choice will result in the same error. Not combined with other blocks because the other blocks don’t matter in this case.



Function: thumbNail(img: Image):file

Parameter: Takes an image to shrink or expand an image to fit a set image size.

- one Parameter:

- Image img

- Characteristics of interest:

- File format

- image width

- image height

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |

Image img File format valid formats [.bmp,.gif,.jpg,.png,.tif,.tiff] [single] B1

!valid formats other [error] B2

Image width empty [0] [error] B3

!empty [1, max] [h !empty] B4

Image height empty [0] [error] B5

!empty [1, max] [w !empty] B6

All combination coverage

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 (or B1, B4, B6) | Test.jpg |
| B2 | Test.pdf |
| B3 | none |
| B5 | none |

Function: calcHori(width: int, height int): int

Specification: Calculates the number of pixels horizontally by dividing by the ratio that reduced the height to 48 pixels.

Special test frame: The number of horizontal pixels must have an upper bound to prevent unreasonably wide images and to allow the Arduino to signal at a leisurely rate. An arbitrary number of 250 chosen, for 20 micro second intervals.

- Two parameters:

- int w, int h

- Characteristics of parameter w:

- Value

- Characteristics of parameter h:

- Value

- Special test frame:

- A width and height that causes the calculated horizontal pixels to be >= 250.

i.e. (500, 3000) [single] B7

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |

Integer width, w Value Minimum(1) (-∞, 1) [error] B1

Nominal(500) [1, 1920] B2

(*soft constraint)* Maximum(1920) (1920, ∞) [single] B3

Integer height, h Value Minimum(1) (-∞, 1) [error] B4

Nominal(500) [1, 1920] B5

(*soft constraint)* Maximum(1920) (1920, ∞) [single] B6

All Combinations coverage

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 | W=-5, h =10 |
| B2, B5 | W=100, h = 200 |
| B3 (or B3, B5) | W= 3000, h= 1000 |
| B4 | W = 48, h = 0 |
| B6 (or B6, B2) | W= 1500, h = 4000 |
| B7 | W = 500, h = 3000 |

Function: black\_and\_white(img: Image)

Specification: Converts an image to black and white.

- one Parameter:

- Image img

- Characteristics of interest:

- File format

- image width

- image height

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |

Image img File format valid formats [.bmp,.gif,.jpg,.png,.tif,.tiff] [single] B1

!valid formats other [error] B2

Image width empty [0] [error] B3

!empty [1, max] [h !empty] B4

Image height empty [0] [error] B5

!empty [1, max] [w !empty] B6

All combination coverage

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 (or B1, B4, B6) | Test.jpg |
| B2 | Test.pdf |
| B3 | none |
| B5 | none |

Function: bitArray(img: Image): bitMatrix

Specification: Takes the black and white image and fills a 2d array with 0, for black, and 1, for white.

- one Parameter:

- Image img

- Characteristics of interest:

- File format

- image width

- image height

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |

Image img File format valid formats [.bmp,.gif,.jpg,.png,.tif,.tiff] [single] B1

!valid formats other [error] B2

Image width empty [0] [error] B3

!empty [1, max] [h !empty] B4

Image height empty [0] [error] B5

!empty [1, max] [w !empty] B6

All combination coverage

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 (or B1, B4, B6) | Test.jpg |
| B2 | Test.pdf |
| B3 | none |
| B5 | none |

Function: printBitArray(matrix: 2d array)

Specification: Print the 2d array that represents the bit pattern for a black and white image.

- One parameter:

- 2d array “matrix”

- Characteristics:

- array size, rows and cols

- element values

- type of elements

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |

2d array, matrix array size, row empty [0] [error] B1

Not empty [1, 48] [col !empty] B2

> max (48, ∞) [single] B3

array size, col empty [0] [error] B4

Not empty [1, 250] [row !empty] B5

> max (250, ∞) [single] B6

Element values zero [0] B7

one [1] B8

not valid [!0 or !1] [single] B9

type of elements integer int B10

not int !int [single] B11

All combination coverage

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 | Empty matrix |
| B2, B5, B7, B10 | [0][0]  [0][0] |
| B2, B5, B8, B10 | [1][1][1][1]  [1][1][1][1] |
| B3 (or B3, B5, B7, B10) | [0][0]  [0][0]  . .  . . 49 rows |
| B4 | Empty matrix |
| B6 (or B2, B6, B8, B10) | [1][1] . . 251 cols |
| B9 (or B2, B5, B9, B10) | [2][2]  [3][4] |
| B11 (or B2,B5, B9, B11) | [h][i] |

Function: signalInterval(width: Int): float

Specification: Calculates the interval to signal the LEDs for the next bit pattern. An arbitrary number of 250 chosen, for 20 micro second intervals, as the upper bound.

- One parameter:

- int w

- Characteristics of parameter w:

- Value

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Parameters | Characteristics | Blocks | Values | Constraints | Label |

Integer width, w Value Minimum(1) (-∞, 1) [error] B1

Nominal(50) [1, 250) B2

(*arbitrary constraint)* Maximum(150) [250, ∞) [single] B3

All combination coverage

|  |  |
| --- | --- |
| Test Frames | Test Cases |
| B1 | W= -5 |
| B2 | W=50 |
| B3 | W=300 |

|  |  |  |  |
| --- | --- | --- | --- |
| Project Name: | LED globe | Test Designed by: | Jonathan |
| Module Name: | ImageConversion.py | Tests last executed: |  |

Test Cases derived from ImageConversionTestFrames.doc to be added ~ ~ ~

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test ID | Test Name | Test Description | Test Cases (inputs) | Test Steps | Expected Results | Acutal Results | Status |
| 1 | test\_thumbNail\_normal() | Load image with valid file format  and non‐empty | img = Test.jpg |  |  |  |  |
| 2 | test\_thumbNail\_invaldFormat() | Load file with invalid image file  format | img = Test.pdf |  |  |  |  |
| 3 | test\_thumbNail\_emptyImg() | Load a created file with 0 width  and 0 height. | img = empty |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 4 | test\_calcHori\_invalidW() | Test invalid image width with valid  height | w=‐5, h=10 |  |  |  |  |
| 5 | test\_calcHori\_normal() | Test normal case, valid width and  height | w=100, h=200 |  |  |  |  |
| 6 | test\_calcHori\_largeW() | Test very large width, where  width/(h/48) < 250 | w=3000, h=1000 |  |  |  |  |
| 7 | test\_calcHori\_invalidH() | Test invalid height with valid width | w=48, h=0 |  |  |  |  |
| 8 | test\_calcHori\_largeH() | Test very large height with valid  Width, where width/(h/48) < 250 | w=1500, h=4000 |  |  |  |  |
| 9 | test\_calcHori\_spc() | Test a width and height, where  width/(h/48) >= 250 | w=500, h=3000 |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 10 | test\_black\_and\_white\_normal() | Load image with valid file format  and non‐empty | img = Test.jpg |  |  |  |  |
| 11 | test\_black\_and\_white\_invalidFormat() | Load file with invalid image file  format | img = Test.pdf |  |  |  |  |
| 12 | test\_black\_and\_white\_emptyImg() | Load a created file with 0 width  and 0 height. | img = empty |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 13 | test\_bitArray\_normal() | Load image with valid file format  and non‐empty | img = Test.jpg |  |  |  |  |
| 14 | test\_bitArray\_invalidFormat() | Load file with invalid image file  format | img = Test.pdf |  |  |  |  |
| 15 | test\_bitArray\_emptyImg() | Load a created file with 0 width  and 0 height. | img = empty |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 16 | test\_printBitArray\_empty() | Input empty matrix | matrix = empty |  |  |  |  |
| 17 | test\_printBitArray\_valid0() | Input non empty matrix filled with  0s | matrix = [0][0], [0][0] |  |  |  |  |
| 18 | test\_printBitArray\_valid1() | Input non empty matrix filled with  1s | matrix = [1][1][1][1], [1][1][1][1] |  |  |  |  |
| 19 | test\_printBitArray\_largeRow() | Input non empty matrix with  number of rows > 49 filled with 0s | matrix = 49 rows filled with 0s |  |  |  |  |
| 20 | test\_printBitArray\_largeCol() | Input non empty matrix with  number of columns > 251 filled with 1s | matrix = 251 columns filled with 1s |  |  |  |  |
| 21 | test\_printBitArray\_invalidVal() | Input non empty matrix filled with  integers other than 0 or 1 | matrix = [2][2], [3][4] |  |  |  |  |
| 22 | test\_printBitArray\_empty() | Input non empty matrix filled with  not integers | matrix = [h][i] |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 23 | test\_signalInterval\_invalidW() | Input invalid width | w=‐5 |  |  |  |  |
| 24 | test\_signalInterval\_normal() | Input nominal width | w=50 |  |  |  |  |
| 25 | test\_signalInterval\_GmaxW() | Input width >= 250 | w=300 |  |  |  |  |

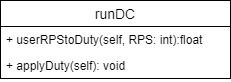
**2.** Hardware Unit Test

Specification: runDC.py constantly reads the Revolutions per second (no load) value and applies the duty cycle to the motor. The stub written is to allow the program to output the duty cycle value for comparison.

Since the program relies on user input, a driver class is included and it contains a function to drive certain inputs for the program to convert. It allows more consistent test cases to be pre-programmed for unit tests.

runDC.py class

Add complete class diagram with driver and stub



MotorStub.py contains the stub and driver



Test cases reasoning

**3.** Distributed Systems Unit Test

**4.** Acceptance Testing