M9 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.

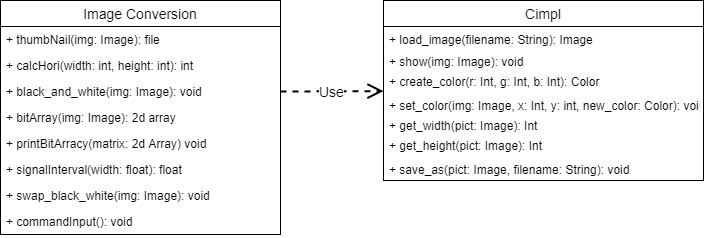
First section is Functionality-Based Input Domain Modeling for ImageConversion.py. The basic principles are; for each function identifies the parameters and characteristics of interest, then identify blocks (which are partitions of the characteristics. **Table for Test Cases after all the derivation.**

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.

# Honestly this method could be simply print a 2d array.

# But we'll have it only print 0s and 1s

- 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: |  |

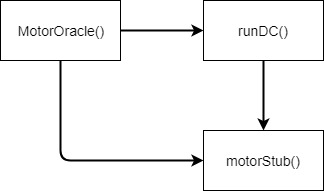
Test Cases derived from ImageConversionTestFrames.doc

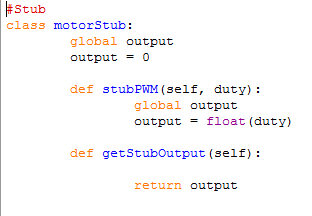
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test ID | Test Name | Test Description | Test Cases (inputs) | Test Steps | Expected Results | Actual Results | Status |
| 1 | test\_thumbNail\_normal() | Load image with valid file format  and non‐empty | img = Test.jpg | thumbnail(Test.jpg) | Return testCreated.jpg where h is shrunk to 48 pixels | testCreatedthumbnail.jpg created and is shrunk | Pass |
| 2 | test\_thumbNail\_invaldFormat() | Load file with invalid image file  format | img = Test.pdf | thumbnail(Test.pdf) | NameError | NameError | Pass |
| 3 | test\_thumbNail\_emptyImg() | Load a created file with 0 width  and 0 height. | img = empty | Create an image of 0 width and 0 height then thumbnail(Created.jpg) | 1. ValueError when creating empty image. 2. thumbNail with empty image gives an UnboundLocalError | 1. ValueError when creating empty image. 2. thumbNail with empty image gives an UnboundLocalError | Pass |
|  |  |  |  |  |  |  |  |
| 4 | test\_calcHori\_invalidW() | Test invalid image width with valid  height | w=‐5, h=10 | calcHori(-5, 10) | Return 0 | 0 | Pass |
| 5 | test\_calcHori\_normal() | Test normal case, valid width and  height | w=100, h=200 | calcHori(100, 200) | Return 24 | 24 | Pass |
| 6 | test\_calcHori\_largeW() | Test very large width, where  width/(h/48) < 250 | w=3000, h=1000 | calcHori(3000, 1000) | Return 144 | 144 | Pass |
| 7 | test\_calcHori\_invalidH() | Test invalid height with valid width | w=48, h=0 | calcHori(48, 0) | Return 0 | 0 | Pass |
| 8 | test\_calcHori\_largeH() | Test very large height with valid  Width, where width/(h/48) < 250 | w=1500, h=4000 | calcHori(1500, 4000) | Return 18 | 18 | Pass |
| 9 | test\_calcHori\_spc() | Test a width and height, where  width/(h/48) >= 250 | w=17000, h=3000 | calcHori(17000, 3000) | Return 250 | 250 | Pass |
|  |  |  |  |  |  |  |  |
| 10 | test\_black\_and\_white\_normal() | Load image with valid file format  and non‐empty | img = Test.jpg | black\_and\_white(Test.jpg). Checking if black and white, call a test function checkBW(img) that checks the color of each pixel | Image converted to black and white | Image converted to black and white | Pass |
| 11 | test\_black\_and\_white\_invalidFormat() | Load file with invalid image file  format | img = Test.pdf | black\_and\_white(Test.pdf). | NameError | NameError | Pass |
| 12 | test\_black\_and\_white\_emptyImg() | Load a created file with 0 width  and 0 height. | img = empty | Create an image of 0 width and 0 height then black\_and\_white(Created.jpg) | 1. ValueError when creating empty image. 2. thumbNail with empty image gives an UnboundLocalError | 1. ValueError when creating empty image. 2. thumbNail with empty image gives an UnboundLocalError | Pass |
|  |  |  |  |  |  |  |  |
| 13 | test\_bitArray\_normal() | Load image with valid file format  and non‐empty  For test, full black, w = 100, h = 100 | img = Test.jpg | bitArray(Test.jpg) and a test function will validate if the 2d array only contains 0s and 1s | Bit array filled with all 0s | bit array filled with all 0s | Pass |
| 14 | test\_bitArray\_invalidFormat() | Load file with invalid image file  format | img = Test.pdf | bitArray(Test.pdf) | NameError | NameError | Pass |
| 15 | test\_bitArray\_emptyImg() | Load a created file with 0 width  and 0 height. | img = empty | Create an image of 0 width and 0 height then bitArray(Created.jpg) | 1. ValueError when creating empty image. 2. thumbNail with empty image gives an UnboundLocalError | 1. ValueError when creating empty image. 2. thumbNail with empty image gives an UnboundLocalError | Pass |
|  |  |  |  |  |  |  |  |
| 16 | test\_printBitArray\_empty() | Input empty matrix | matrix = empty | Create an empty 2d array then printBitArray(createdArray) | Prints input array | prints input matrix | Pass |
| 17 | test\_printBitArray\_valid0() | Input non empty matrix filled with  0s | matrix = [[0,0],[0,0]] | Create an empty 2d array with 2 rows and 2 columns of all 0s ,then printBitArray (createdArray) | Prints input array | prints input matrix | Pass |
| 18 | test\_printBitArray\_valid1() | Input non empty matrix filled with  1s | matrix = [[1,1,1,1], [1,1,1,1]] | Create an empty 2d array with 4 rows and 4 columns of all 1s ,then printBitArray (createdArray) | Prints input array | prints input matrix | Pass |
| 19 | test\_printBitArray\_largeRow() | Input non empty matrix with  number of rows > 49 filled with 0s | matrix = 49 rows, and 2 columns filled with 0s | Create an empty 2d array with 49 rows and 2 columns of all 0s ,then printBitArray (createdArray) | Prints input array | prints input matrix | Pass |
| 20 | test\_printBitArray\_largeCol() | Input non empty matrix with  number of columns > 251 filled with 1s | matrix = 251 columns and 10 rows filled with 1s | Create an empty 2d array with 10 rows and 251 columns of all 1s ,then printBitArray (createdArray) | Prints input array | prints input matrix | Pass |
| 21 | test\_printBitArray\_invalidVal() | Input non empty matrix filled with  integers other than 0 or 1 | matrix = [[2,2], [3,4]] | Create an empty 2d array with 2 rows and 2 columns of 2,2;3,4 ,then printBitArray (createdArray) | Does not print input array | Does not print input array | Pass |
| 22 | test\_printBitArray\_nonInt() | Input non empty matrix filled with  not integers | matrix = [[‘h’,’I’],[]] | Create an empty 2d array with 1 rows and 2 columns of h,i ,then printBitArray (createdArray) | Does not print input array | Does not print input array | Pass |
|  |  |  |  |  |  |  |  |
| 23 | test\_signalInterval\_invalidW() | Input invalid width | w=‐5 | Input signalInterval(-5) | Return 3.7 x 10^-3 | Return 3.7 x 10^-3 | Pass |
| 24 | test\_signalInterval\_normal() | Input nominal width | w=50 | Input signalInterval(50) | Return 7.407 x 10^-5 | Return 7.407 x 10^-5 | Pass |
| 25 | test\_signalInterval\_GmaxW() | Input width >= 250 | w=300 | Input signalInterval(300) | Return 1.48 x 10^-5 | Return 1.48 x 10^-5 | Pass |

**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.

The program relies on receiving its input from an external file, so we replace this with by having the applyDuty() to get a value at a set interval from changeRPS() and provide a function to set this value.



MotorStub.py contains the stub and Inputs for the moto

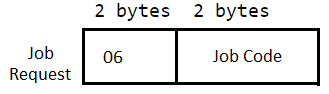
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Project Name: | LED globe | Test Designed by: | Jonathan |  |  |  |
|  | Module Name: | runDC.py | Tests last executed: | Nov 15 |  |  |  |
|  | Using a stub MotoStub.py to provide inputs (motorInputs class) and allow output (MotorStub class) | | | |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
| Test ID | Test Name | Test Description | Test Cases (inputs) | Test Steps | Expected Results | Acutal Results | Status (pass/fail) |
| 1 | test\_userRPStoDutyNeg() | Input of a negative value | RPS = -50 | Input RPS to userRPStoDuty and compare the result with expected duty | return 0 | 0 | Pass |
| 2 | test\_userRPStoDutyZero() | Input of a value 0 | RPS = 0 | Input RPS to userRPStoDuty and compare the result with expected duty | return 0 | 0 | Pass |
| 3 | test\_userRPStoDutyNorm() | Input of a value within range | RPS = 50 | Input RPS to userRPStoDuty and compare the result with expected duty | return 35.46 | 35.46 | Pass |
| 4 | test\_userRPStoDutyMax() | Input of a max value | RPS = 141 | Input RPS to userRPStoDuty and compare the result with expected duty | return 100 | 100 | Pass |
| 5 | test\_userRPStoDutyLarge() | Input of value larger than max | RPS = 200 | Input RPS to userRPStoDuty and compare the result with expected duty | return 100 | 100 | Pass |
| 6 | test\_userRPStoDutyInvalid() | Input an input type like char | RPS = aaa | Input RPS to userRPStoDuty and compare the result with expected duty | NameError | NameError | Pass |
|  |  |  |  |  |  |  |  |
| 7 | test\_applyDutyNeg() | In motorInput class, set the input value to a negative value. | applyDuty(-50, stubInst) | Call function applyDuty(inRPS, stubInst). Where "inRPS" and "stubInst" are parameters added to the function for testing purposes. inRPS is the revolutions per second and stubInst is the instance the of the stub class. | 0 | 0 | Pass |
| 8 | test\_applyDutyNegOne() | In motorInput class, set the input value to -1, break condition for applyDuty loop. | applyDuty(-1, stubInst) | Call function applyDuty(inRPS, stubInst). Where "inRPS" and "stubInst" are parameters added to the function for testing purposes. inRPS is the revolutions per second and stubInst is the instance the of the stub class. | 0 | 0 | Pass |
| 9 | test\_applyDutyZero() | In motorInput class, set the input value to 0. | applyDuty(0, stubInst) | Call function applyDuty(inRPS, stubInst). Where "inRPS" and "stubInst" are parameters added to the function for testing purposes. inRPS is the revolutions per second and stubInst is the instance the of the stub class. | 0 | 0 | Pass |
| 10 | test\_applyDutyNorm() | In motorInput class, set the input value to a normal value. | applyDuty(75, stubInst) | Call function applyDuty(inRPS, stubInst). Where "inRPS" and "stubInst" are parameters added to the function for testing purposes. inRPS is the revolutions per second and stubInst is the instance the of the stub class. | 53.19 | 53.19 | pass |
| 11 | test\_applyDutyMax() | In motorInput class, set the input value to max value. | applyDuty(141, stubInst) | Call function applyDuty(inRPS, stubInst). Where "inRPS" and "stubInst" are parameters added to the function for testing purposes. inRPS is the revolutions per second and stubInst is the instance the of the stub class. | 100 | 100.0 | Pass |
| 12 | test\_applyDutyLarge() | In motorInput class, set the input value to a value larger than max. | applyDuty(200, stubInst) | Call function applyDuty(inRPS, stubInst). Where "inRPS" and "stubInst" are parameters added to the function for testing purposes. inRPS is the revolutions per second and stubInst is the instance the of the stub class. | 100 | 100.0 | Pass |
|  |  |  |  |  |  |  |  |

**3.** Distributed Systems Unit Test

Communication protocol of the server follows TFTP

Type of messages received by the server:

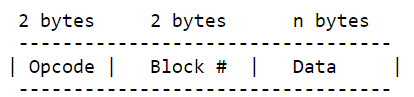
1. Job request, opcode 06: A job request packet contains 2 bytes for the opcode and 2 bytes for the Job code which specifies the job to initialize.

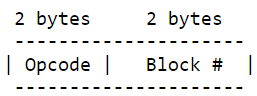


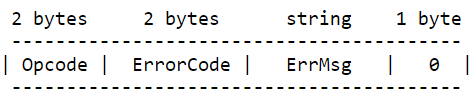
Job request will be mainly used to operate the LED ring where job 1 will initialize and control the DC motor and job 2 execute a program with the image bit pattern to operate the LEDs.

2. Write request, opcode 02: 

3. Read request, opcode 01: 

4. Data packet, opcode 03: 

5. ACK packet, opcode 04: 

6. Error packet, opcode 05: 

Types of messages sent by the server are Data, ACK, and Error packets.

Not testing the TFTP, but the “Jobs.” The mock object will be a TFTP client that sends job requests to so the server initializes some programs to meet pre-conditions for the job and establish connection, then subsequently send pre-determined data.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **No.** | **Title** | **Description** | **Context** | **Trigger** | **Response** |
| 1 | test\_job1\_RQ() | Sends a correct Job 1 request.  After correct response ACK to the client, close the client and allow server to time out. | Server executes a setup and sends a correct ACK response. | Job RQ = 0601 | ACK from server with block number 0. [0400] |
| 2 | test\_job2\_RQ() | Sends a correct Job 2 request. After correct response ACK to the client, close the client and allow the server to time out. | Server executes a setup and sends a correct ACK response. | Job RQ = 0602 | ACK from server with block number 0. [0400] |
| 3 | test\_job1\_operation() | Sends a correct Job 1 request and wait until timeout “t” for ACK with block 0.  If received, Mock sends a test value one in a data packet.  The server either times out or the client sends a data packet with no data to indicate end. Close server socket and thread dies. | Pre-condition: setup correct.  Server receives correct data and writes it to the correct file specified on the server. | Data packet (1st) = 030150  (value 50) | Motor RPS successfully changes.  ACK from server with block number 1. [0401] |
| 4 | test\_job2\_operation() | Sends a correct Job 2 request and wait until timeout “t” for ACK with block 0.  If received, Mock sends bit array in data packet(s)  Server closes when a data packet is less than 516 bytes. | Pre-condition: setup correct.  Server receives correct data and writes it to the correct file specified on the server.  Server executes the correct LED program with the bit pattern and signal interval. | Data packet (1st) = 0301100010110110…..  (value 100010110110…..) until 512 bytes. If more:  Data packet (2nd) = 03010000000000…..  (value 0000000000…..)  Continues until bit pattern and signal interval sent. | ACK from server with block number n. [04n], where “n” echoes the block of the data packet.  After complete successful transfer, LED program executed with the bit pattern and signal interval. |

**4.** Acceptance Testing

• Control the motor speed to be able to simulate the optical illusion, reach an appropriate number of frames per second so the image looks complete. A (1)raspberry pi PWM output could be used to drive the motor or use an external PWM controller.

• A separate (2)Raspberry pi can display the interface to select an image to send to the globe. If an Ethernet cable is required to send the image, the procedure would be to connect, send it, and then disconnect it.

• The second (2)Raspberry pi could also be used to send the take a user’s desired speed and send it to the raspberry pi controlling the motor, then poll for new user input speed.

• Set the correct timing of the LEDs to change colour or blink when spinning to present the correct image on the globe. An Arduino can be used to control the LEDs.

• Using a Hall sensor to detect when one revolution is complete, this helps with synchronization for the pattern to be sent to the LEDs. This may be controlled by the same Arduino.

• To convert images to be able to be displayed on the globe, the image has to be converted into a pixelated version then determine the (r,g,b) array for each pixel. The image processing can be automated with Python then the arrays can be directly copied to Arduino Sketch.

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Successful** | **Erroneous** |
| User inputs an image they would like to display on the LED globe. | 1. The image’s height is re-sized to the number of LEDs and width re-sized with the same ratio when re-sizing height.  2. The image is converted to a black and white image (due to no (r, g, b) LEDs), then a bit array is populated.  3. The signaling interval to the LEDs is also calculated.  4. Client Job request to the server and successful ACK response. Then the bit array and signaling interval are transferred to the server where it initializes the program that controls the LEDs with that bit pattern and signaling interval. | - Job request yield in a correct ACK from the server.  - Incomplete transfer of the bit pattern to the server  - Incorrect or no signaling interval value sent to the server  - LED control program incorrectly loads the bit pattern and/or signaling interval. |
| User enters a value for revolutions per second (unloaded) they would like to see the motor spin at. | 1. Client Job request to the server and successful ACK response.  2. Subsequent data packets are the user inputted values for the revolutions per second for the motor. | - Job request yield in a correct ACK from the server.  - Incorrect transfer or no transfer of the user value to the server  - Motor control program incorrect applies the revolutions per second to the motor. |
| Hall sensor sends a signal as an interrupt to allow a consistent timing for the pattern to start. | 1. Hall sensor passes the stationary magnet.  2. The signal used to trigger an interrupt so the LEDs pattern has a consistent point for synchronization | - Hall sensor does not send a signal at the point it detects the stationary magnet.  - Interrupt triggers incorrect behavior.  - Correct function triggered by the interrupt does not output the correct behavior. |
| TFTP server executes job 1 correctly | 1. Initialize DC motor with 0 duty cycle  2. User value written to a text file the motor program reads from, so it can convert the value to a duty cycle for the motor. | - Server fails to initialize the program that starts the DC motor loop.  - Server writes to the incorrect file or writes the incorrect value |
| TFTP server executes job 2 correctly | 1. Writes the bit pattern with correct format to a text file.  2. Starts the LED program with the loaded bit pattern and signaling interval. | - Server fails to write the bit pattern to the text file or writes incorrect data  - Server fails to start the LED program with the required parameters. |