Manual Trace of the Daltonization Algorithm

Input RGB Image / Video Feed Frame
Converting RGB image to LMS image
Simulate color blindness on 'LMS Image' -> 'Simulated LMS Image'
Compensate 'Simulated LMS Image' -> 'Simulated RGB Image'
Convert 'Simulated RGB Image'' to 'Compensated/Daltonized RGB Image'
Output Daltonized RGB Image

	Converting RGB image to LMS image					
	RGB2LMS = [[17.8824,43.5161,4.11935],	Line	i	j		
	[3.45565,27.1554,3.86714],	1	0			1st row
	[0.0299566,0.184309,1.46709]]	3		0		
		5-15			LMS multiplication	Row loop
1	i = 0	16		1		
2	WHILE i < height DO	5-15			LMS multiplication	
3	j = 0	16		2		
4	WHILE j < width DO #(j, i) refer to pixel position	....				
5	R = RGB_image[j, i, 0]	16		width		
6	G = RGB_image[j, i, 1]	18	1			2nd row
7	B = RGB_image[j, i, 2]	3		0		
8		5-15			LMS multiplication	Row loop
9	L = RGB2LMS[0][0] * R + RGB2LMS[0][1] * G + RGB2LMS[0][2] * B	16		1		
10	M = RGB2LMS[1][0] * R + RGB2LMS[1][1] * G + RGB2LMS[1][2] * B	....				
11	S = RGB2LMS[2][0] * R + RGB2LMS[2][1] * G + RGB2LMS[2][2] * B	16		width		
12		18	2			3rd row
13	LMS_image[j, i, 0] = L	3		0		
14	LMS_image[j, i, 1] = M	5-15			LMS multiplication	Row loop
15	LMS_image[j, i, 2] = S	16		1		
16	j = j + 1	....				
17	END WHILE	16		width		
18	i = i + 1	....				
19	END WHILE	18	height			End
		Time complexity = O(height x width) = O(n)				

Simulate color blindness on 'LMS Image' -> 'Simulated LMS Image'

```
colorblind_type = 0   # For example; 0 is for Protanope, 1 is for Deuteranope, and 2 is for Tritanope
L, M, and S values from previous process
1  i = 0
2  WHILE i < height DO
3     j = 0
4     WHILE j < width DO
5       L = LMS_image[j, i, 0]
6       M = LMS_image[j, i, 1]
7       S = LMS_image[j, i, 2]
8
9       IF colorblind_type == 0 THEN
10        simulated_L = protanope_matrix[0][0] * L + protanope_matrix[0][1] * M + protanope_matrix[0][2] * S
11        simulated_M = protanope_matrix[1][0] * L + protanope_matrix[1][1] * M + protanope_matrix[1][2] * S
12        simulated_S = protanope_matrix[2][0] * L + protanope_matrix[2][1] * M + protanope_matrix[2][2] * S
13
14      ELSE IF colorblind_type == 1 THEN
15        simulated_L = deuteranope_matrix[0][0] * L + deuteranope_matrix[0][1] * M + deuteranope_matrix[0][2] * S
16        simulated_M = deuteranope_matrix[1][0] * L + deuteranope_matrix[1][1] * M + deuteranope_matrix[1][2] * S
17        simulated_S = deuteranope_matrix[2][0] * L + deuteranope_matrix[2][1] * M + deuteranope_matrix[2][2] * S
18
19      ELSE IF colorblind_type == 2 THEN
20        simulated_L = tritanope_matrix[0][0] * L + tritanope_matrix[0][1] * M + tritanope_matrix[0][2] * S
21        simulated_M = tritanope_matrix[1][0] * L + tritanope_matrix[1][1] * M + tritanope_matrix[1][2] * S
22        simulated_S = tritanope_matrix[2][0] * L + tritanope_matrix[2][1] * M + tritanope_matrix[2][2] * S
23
24      ELSE   # original LMS values
25        simulated_L = L
26        simulated_M = M
27        simulated_S = S
28      END IF
29
30      simulated_image[j, i, 0] = simulated_L
31      simulated_image[j, i, 1] = simulated_M
32      simulated_image[j, i, 2] = simulated_S
33
34       j = j + 1
35       END WHILE
36    i = i + 1
37 END WHILE
```

Line	i	j		
1	0			1st Row
3		0		
9-34			LMS multiplication	Row loop
34		1		
9-34			LMS multiplication	
34		2		
...				
34		width		
36	1			2nd Row
3		0		
9-34			LMS multiplication	Row loop
34		1		
...				
34		width		
...				
36	height			End

Time complexity = O(height x width) = O(n)

Compensate 'Simulated LMS Image' -> 'Simulated RGB Image'

	LMS2RGB = [[0.0809444479, -0.130504409, 0.116721066],	Line	i	j		
	[-0.0102485335, 0.0540193266, -0.113614708],	1	0			1st row
	[-0.000365296938, -0.00412161469, 0.693511405]]	3		0		
		5-15			RGB multiplication	Row loop
1	i = 0	16		1		
2	WHILE i < height DO	5-15			RGB multiplication	
3	j = 0	16		2		
4	WHILE j < width DO #(j, i) refer to pixel position	....				
5	L = LMS_image[j, i, 0]	16		width		
6	M = LMS_image[j, i, 1]	18	1			2nd row
7	S = LMS_image[j, i, 2]	3		0		
8		5-15			RGB multiplication	Row loop
9	R = LMS2RGB[0][0] * L + LMS2RGB[0][1] * M + LMS2RGB[0][2] * S	16		1		
10	G = LMS2RGB[1][0] * L + LMS2RGB[1][1] * M + LMS2RGB[1][2] * S	....				
11	B = LMS2RGB[2][0] * L + LMS2RGB[2][1] * M + LMS2RGB[2][2] * S	16		width		
12		18	2			3rd row
13	RGB_image[j, i, 0] = R	3		0		
14	RGB_image[j, i, 1] = G	5-15			RGB multiplication	Row loop
15	RGB_image[j, i, 2] = B	16		1		
16	j = j + 1	....				
17	END WHILE	16		width		
18	i = i + 1	....				
19	END WHILE	18	height			End
		Time complexity = O(height x width) = O(n)				

	Convert 'Simulated RGB Image" to 'Compensated/Daltonized RGB Image'					
1	i = 0	Line	i	j		
2	WHILE i < height DO	1	0			1st row
3	j = 0	3		0		
4	WHILE j < width DO	5-19			Conversion process	Row loop
5	simulated_R = simulated_RGB_image[j, i, 0]	21		1		
6	simulated_G = simulated_RGB_image[j, i, 1]	5-19			Conversion process	
7	simulated_B = simulated_RGB_image[j, i, 2]	21		2		
8	# Isolate "invisible colors" (colors that people with a CVD can't see)	....				
9	error_R = R - simulated_R	21		width		
10	error_G = G - simulated_G	23	1			2nd row
11	error_B = B - simulated_B	3		0		
12	# Shift the colors towards the visible spectrum	5-19			Conversion process	Row loop
13	compensated_R = (0.0 * error_R) + (0.0 * error_G) + (0.0 * error_B)	21		1		
14	compensated_G = (0.7 * error_R) + (1.0 * error_G) + (0.0 * error_B)	....				
15	compensated_B = (0.7 * error_R) + (0.0 * error_G) + (1.0 * error_B)	21		width		
16	# Add compensation	23	2			3rd row
17	compensated_R = compensated_R + R	3		0		
18	compensated_G = compensated_G + G	5-19			Conversion process	Row loop
19	compensated_B = compensated_B + B	21		1		
20		....				
21	j = j + 1	21		width		
22	END WHILE	....				
23	i = i + 1	23	height			End
24	END WHILE					