

EEE 591
Real-Time DSP Systems

Project 1
Real Time Image Processing

Due Date: March 19th, 2024

Jayaram Atluri
ASU ID: 1228699721

Professor: Dr. Chao Wang

Introduction:

This project aims to apply real-time image enhancement techniques to live video streams, providing hands-on experience with DSP applications. Through tasks like Thresholding, Gray Level Quantization, Gray Level Transformations, and Histogram Equalization, we gained practical skills in MATLAB and hardware interaction. This project bridges theoretical knowledge with real-world applications, enhancing our understanding of image processing. Below are the contents of deliverables in the order as follows:

1. Tables

- **Thresholding**
 - Global thresholding
 - Band thresholding.
 - Semi thresholding
- **Gray Level Quantization**
 - Quantization Levels (128, 64, 32, 16)
- **Gray Level Transformations**
 - Transformation 1
 - Transformation 2
 - Transformation 3
- **Histogram Equalization**
 - Histogram Equalization Observations

2. Screenshots

- Global thresholding
- Band thresholding.
- Semi thresholding
- Gray Level Quantization
- Gray Level Transformation

3. Codes

- **Thresholding Functions**
 - Global Thresholding C
 - Band Thresholding C
 - Band Thresholding C
 - Band Thresholding Hybrid (Grad Students Only)
 - Semi Thresholding C
 - Semi Thresholding Hybrid
- **Gray Level Quantization Functions**
 - Gray Level Quantization C
 - Gray Level Quantization Hybrid
 - Modified MATLAB script to reconstruct quantized image for display.
- **Gray Level Transformation Functions**
 - Gray Level Transformation 1 C
 - Gray Level Transformation 1 Hybrid
 - Gray Level Transformation 2 C
 - Gray Level Transformation 2 Hybrid (Grad Students Only)
 - Gray Level Transformation 3 C (Extra Credit)
 - Gray Level Transformation 3 Hybrid (Extra Credit)
- **Histogram Equalization Functions**
 - Calculate Histogram C
 - Calculate Histogram Hybrid

- Map Levels C
- Map Levels Hybrid
- Transform Image C
- Transform Image Hybrid

1. Tables:

Thresholding:

Task	Parameters	Observation and MATLAB Plot
Global Thresholding	Threshold: 80	The output image is binary, with the background and brighter regions turning white, and the subject and shadows turning black. The output histogram shows two distinct peaks corresponding to the black and white regions in the image.
Band Thresholding	Threshold1: 80, Threshold2: 240	The band thresholding has isolated the mid-tones. The subject's face and shirt are white, indicating these pixel values fall within the specified band. The output histogram is expected to show a concentration of pixel values within the threshold band.
Semi Thresholding	Threshold: 80	Semi thresholding darkens areas below the threshold, keeping brighter areas unchanged. The subject's face is visible against a darker background. The output histogram will likely show a reduction in lower-intensity values.

Gray Level Quantization:

Quantization Levels	Degradation in Image Quality	Missing Values in Histogram
128 (shift right by 1)	Yes, slight degradation with less detail, especially in finer textures.	Yes, there are fewer histogram peaks, indicating reduced gray levels.
64 (shift right by 2)	Yes, moderate degradation with more pronounced loss of detail and contouring effects.	Yes, histogram peaks are more spaced out due to fewer gray levels.
32 (shift right by 3)	Yes, significant degradation with coarse quantization visible, resulting in a blockier appearance.	Yes, histogram shows significant gaps, indicating many gray levels have been combined.
16 (shift right by 4)	Yes, severe degradation with a very blocky and abstract appearance, losing most details.	Yes, the histogram is very sparse, with most original gray levels missing.

Gray Level Transformations:

Transformation	A	B	Observation
Transformation 1			Image inversion is observed, dark regions become light and vice versa, enhancing details in dark areas.
Transformation 2	70	100	With $A < B$, the dynamic range is expanded in the lower gray levels, enhancing details in darker regions.
Transformation 2	100	70	With $A > B$, the dynamic range is compressed, potentially improving visibility in highlights.
Transformation 2	70	70	With $A = B$, no transformation is applied; the image remains unchanged.
Transformation 3	70	100	With $A < B$, the transformation might enhance mid-tones while keeping darker and lighter regions largely unchanged.
Transformation 3	100	70	With $A > B$, there might be an aggressive contrast enhancement effect, particularly affecting mid-tone regions.
Transformation 3	70	70	With $A = B$, the transformation could create a high-contrast image where mid-tones are either significantly darkened or lightened.

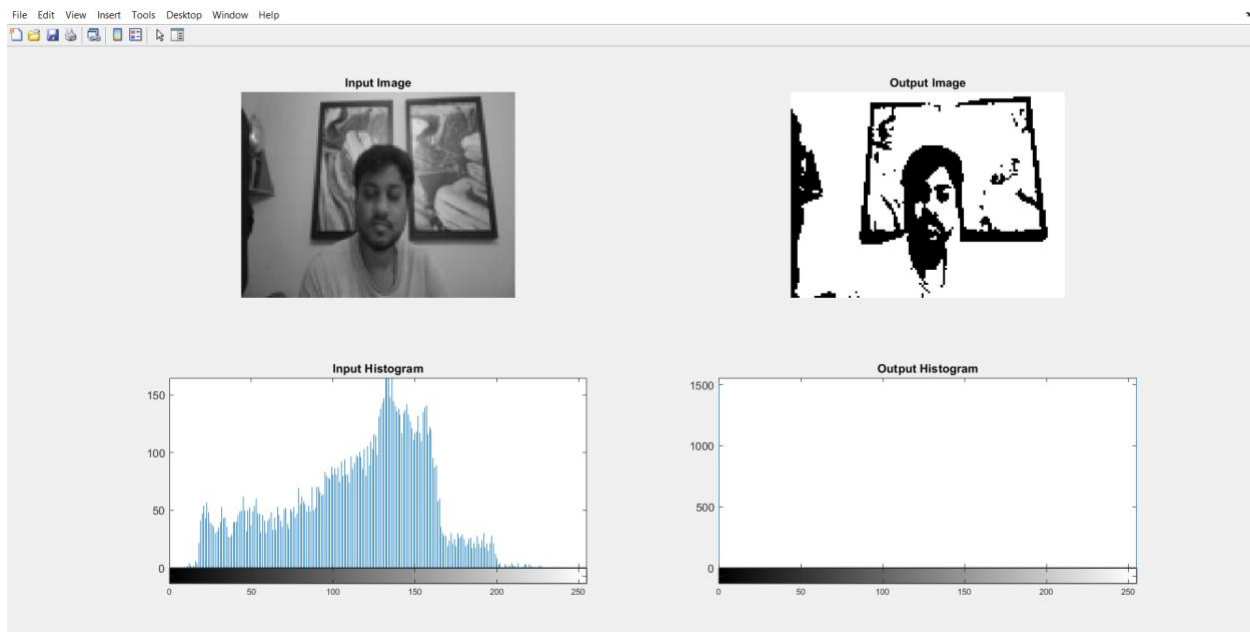
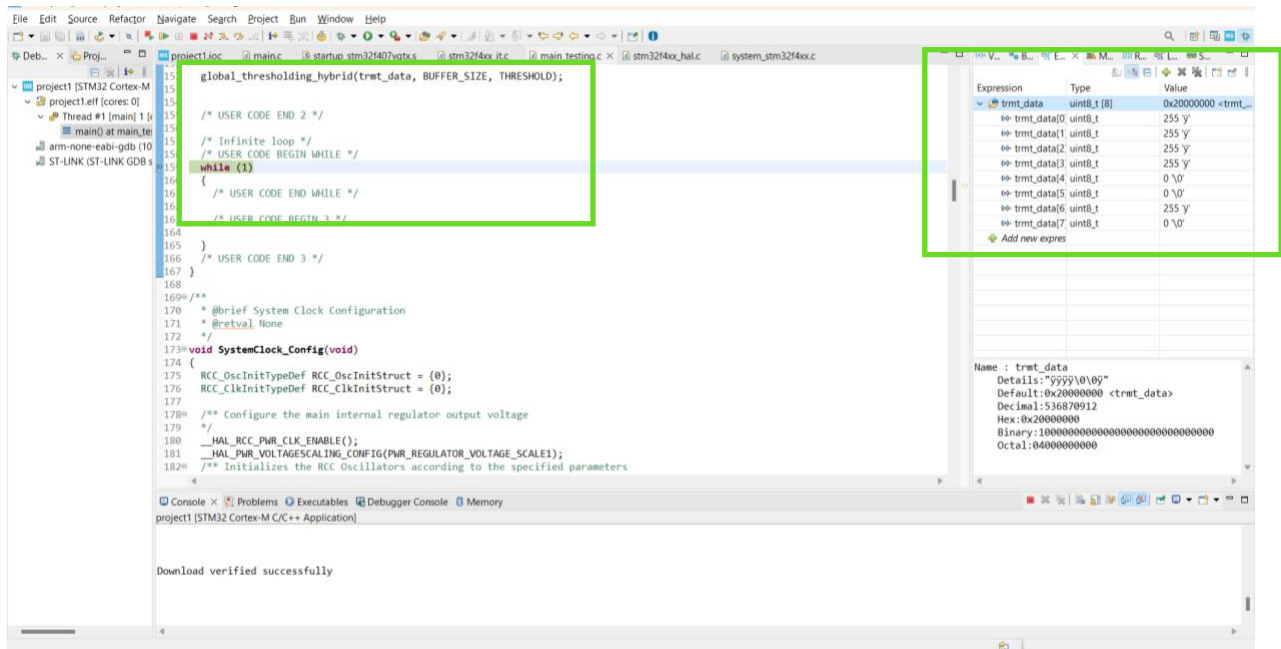
Histogram Equalization Observations:

Equalization	Observation
Histogram Equalization	The image that has been processed has shown improved contrast. The features that were previously difficult to distinguish, such as edges or textures, are now more visible and clear. The distribution of pixel intensity values across the histogram range is now more uniform, which leads to an improved visual perception of the image. Darker regions may have been brightened, and brighter areas may now show more detail.

2. Screenshots:

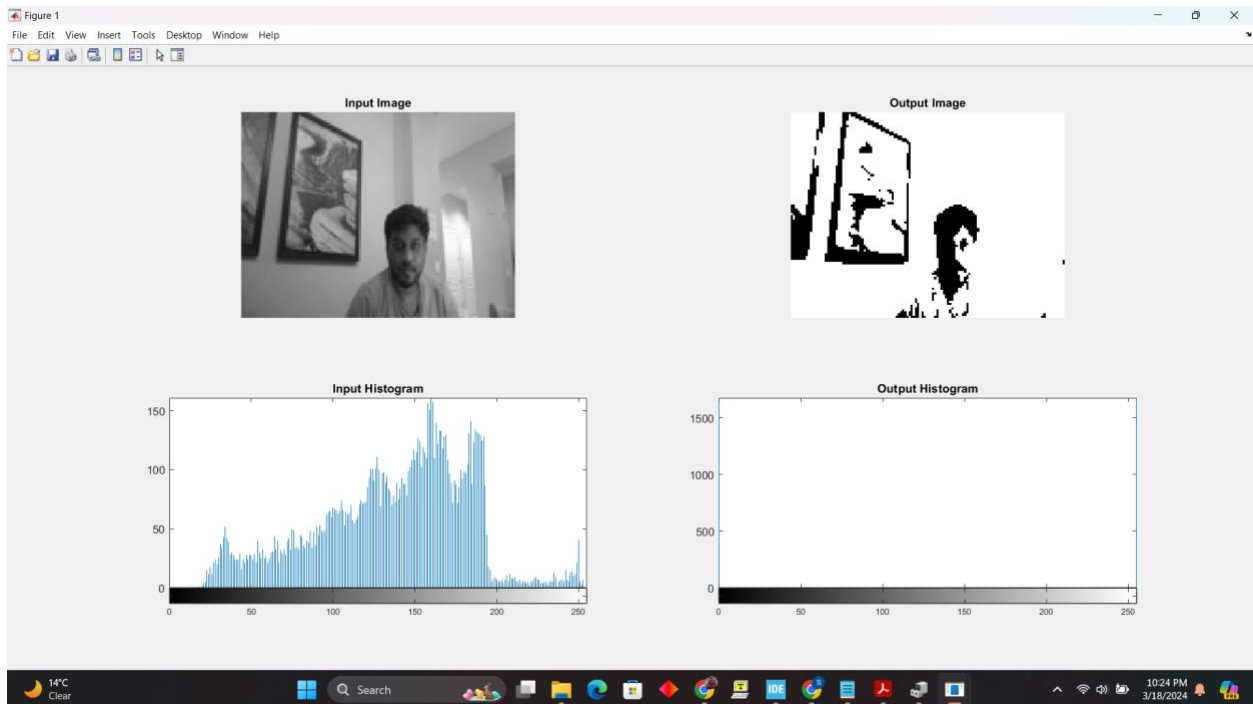
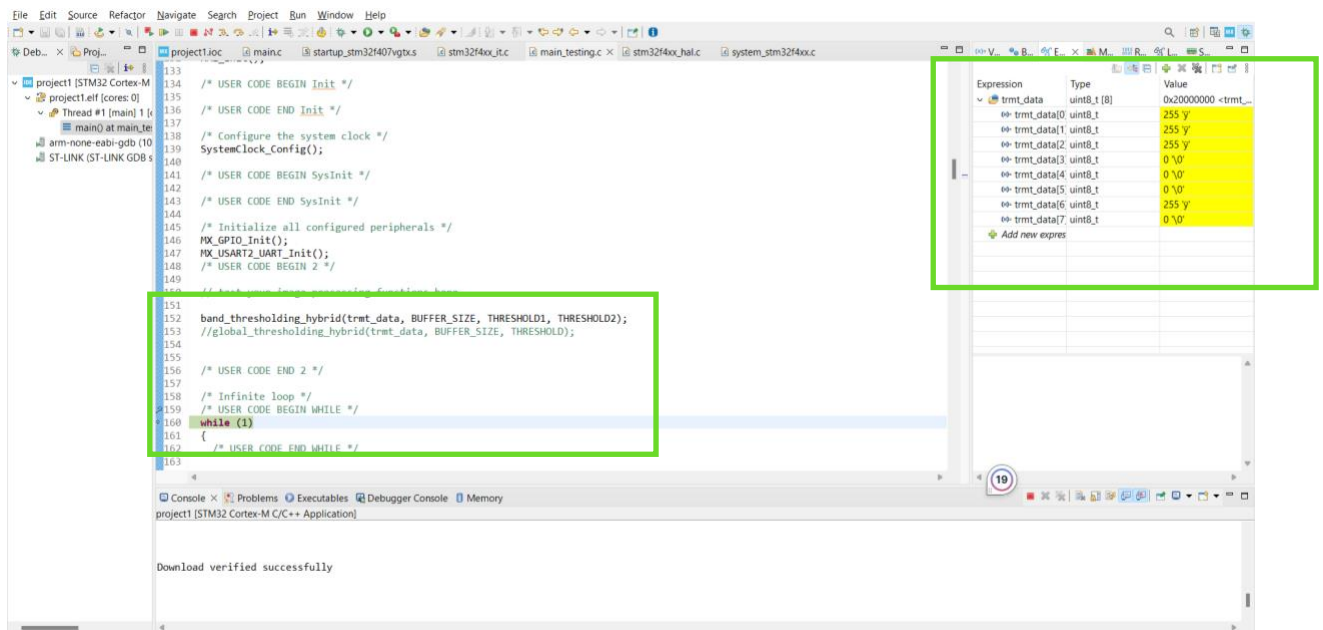
Function Screenshots:

Global thresholding:



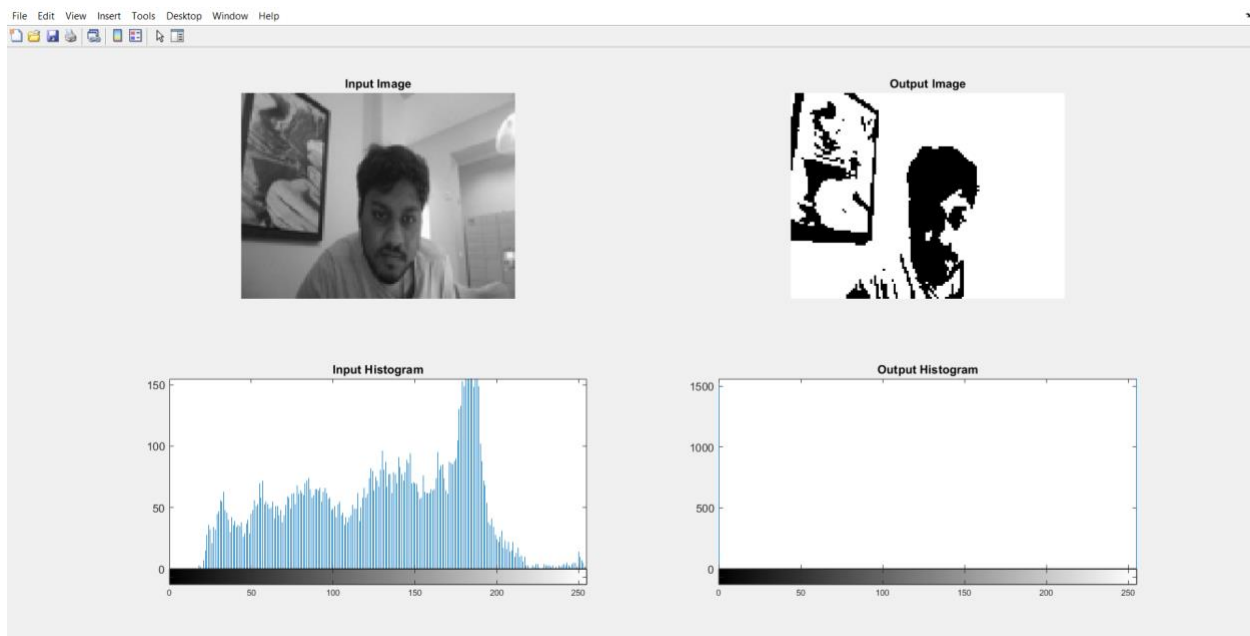
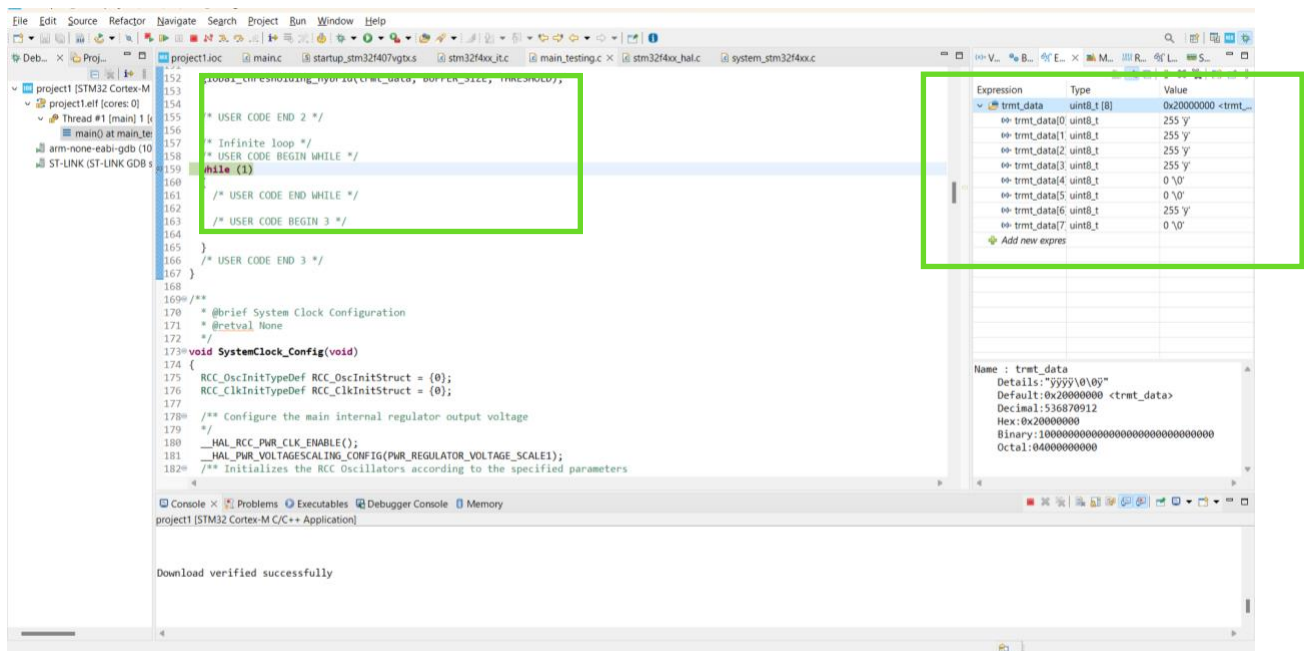
MATLAB plot

Band thresholding:



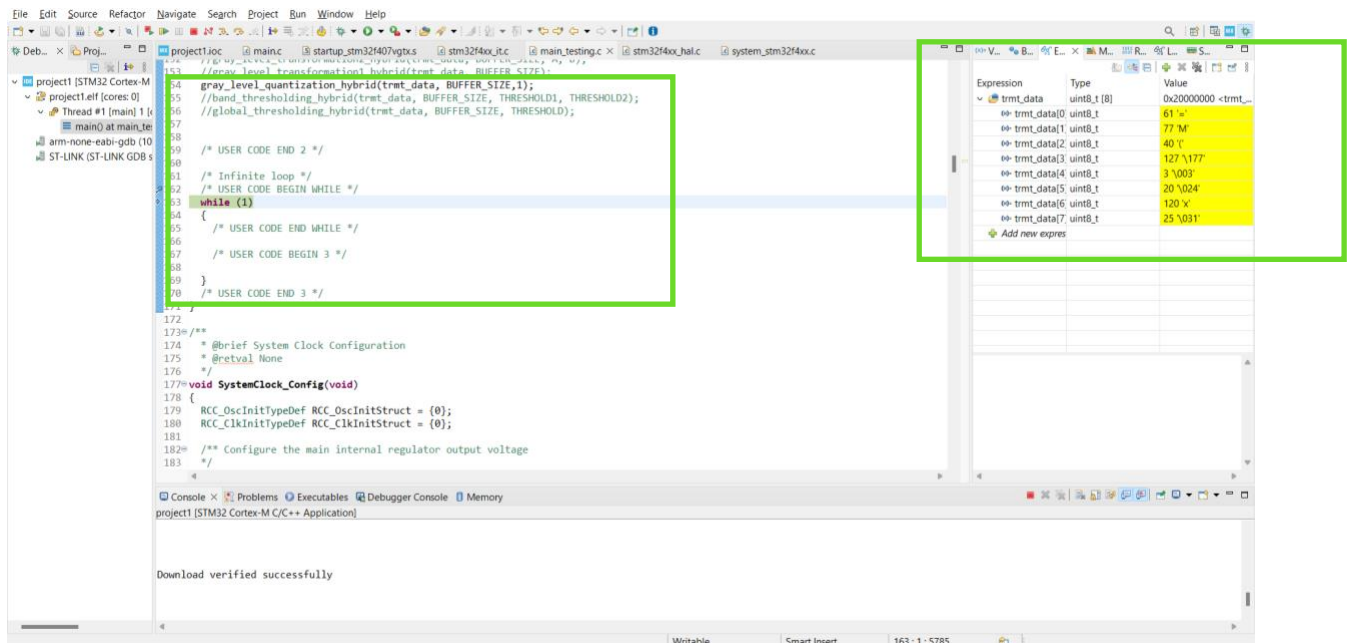
MATLAB plot

Semi thresholding:

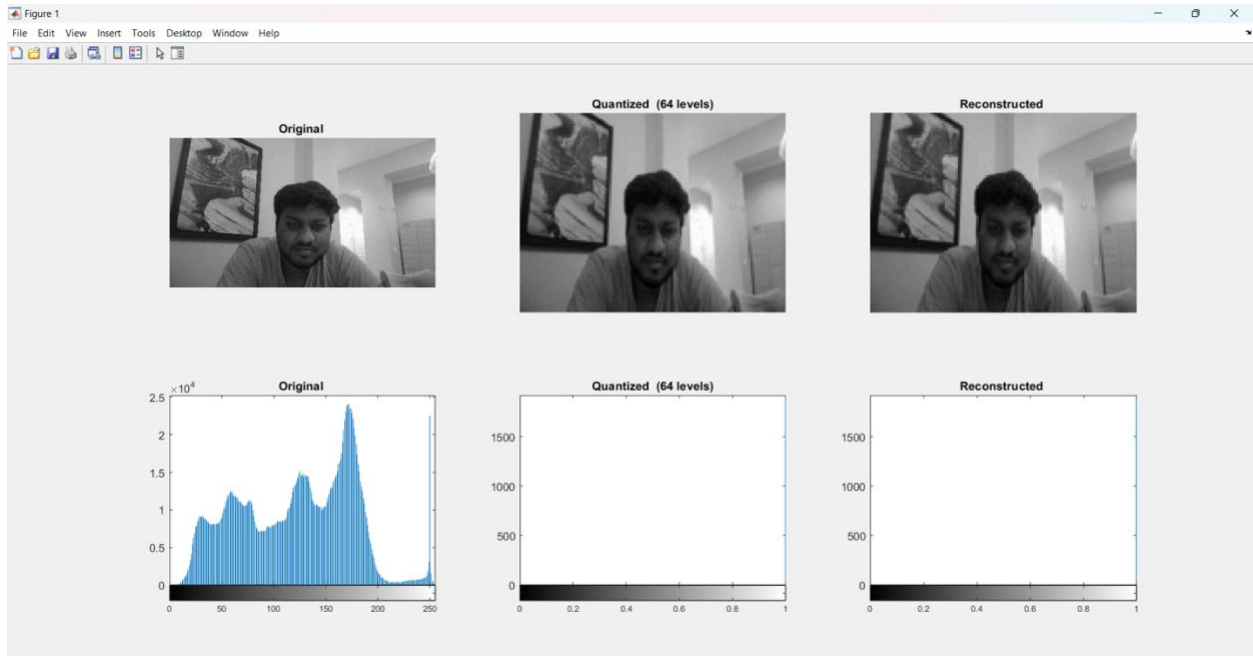


MATLAB plot

128 (shift right by 1):



64 (shift right by 2):



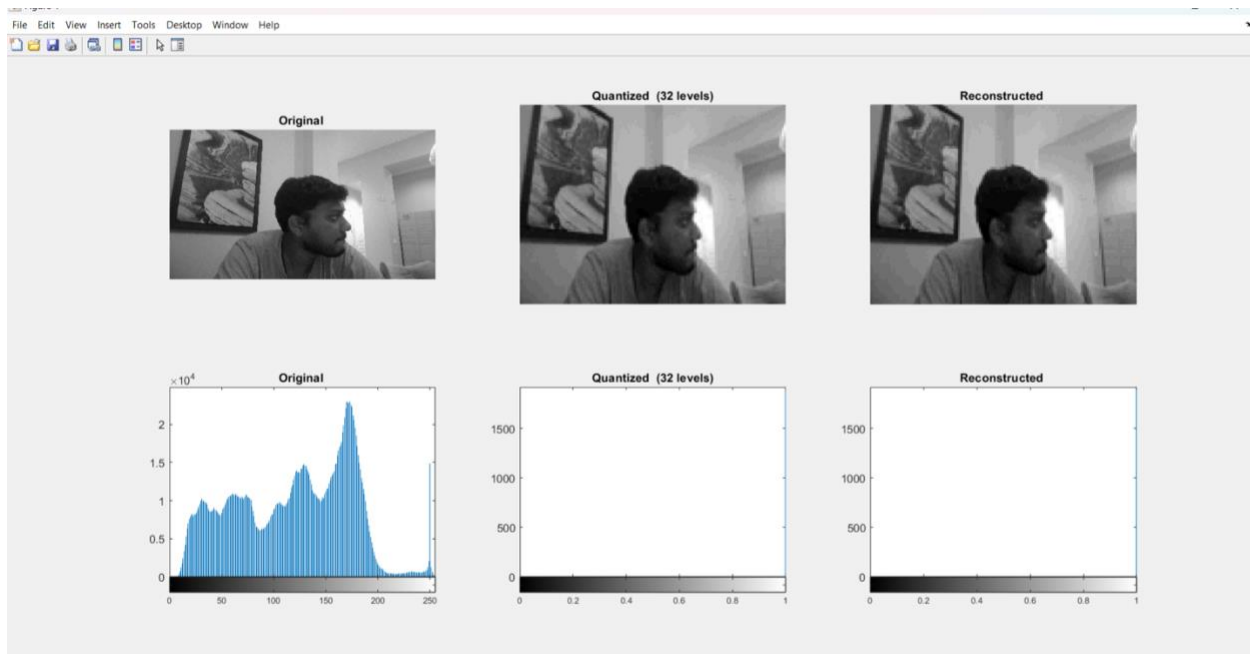
```
File Edit Source Refactor Navigate Search Project Run Window Help
project1ioc | mainc | startup_stm32407vgtxs | stm3240x_itc | main_testing.c | stm3240x_hal.c | system_stm3240x.c
project1 [STM32 Cortex-M]
  project1.c [cores: 0]
    Thread #1 [main] 1
      main() at main.c:10
        arm-none-eabi-gdb: 10
          ST-LINK (ST-LINK GDB)
            /* Configure the system clock */
            SystemClock_Config();
            /* USER CODE BEGIN SysInit */
            /* USER CODE END SysInit */
            /* Initialize all configured peripherals */
            MX_GPIO_Init();
            MX_USART2_UART_Init();
            /* USER CODE BEGIN 2 */
            // test your image processing functions here
            //gray_level_transformation2_hybrid(trmt_data, BUFFER_SIZE, A, B);
            //gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
            //gray_level_transformation_hybrid(trmt_data, BUFFER_SIZE, 2);
            //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
            //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
            /* USER CODE END 2 */
            /* Infinite loop */
            /* USER CODE BEGIN WHILE */
            while (1)
            {
            /* USER CODE END WHILE */
            /* USER CODE BEGIN 3 */
            167
            168
```

Expression	Type	Value
trmt_data	uint8_t [8]	0x20000000 <trmt_data>
trmt_data[0]	uint8_t	30 '\036'
trmt_data[1]	uint8_t	38 '&'
trmt_data[2]	uint8_t	20 '\024'
trmt_data[3]	uint8_t	63 '7'
trmt_data[4]	uint8_t	1 '\001'
trmt_data[5]	uint8_t	10 '\n'
trmt_data[6]	uint8_t	60 '<'
trmt_data[7]	uint8_t	12 '\f'

Name : trmt_data
Details: '\036\024\001\n<\f'
Default: 0x20000000 <trmt_data>
Decimal: 536870912
Hex: 0x20000000
Binary: 10000000000000000000000000000000
Octal: 040000000000

Console | Problems | Executables | Debugger Console | Memory
project1 [STM32 Cortex-M] C/C++ Application
Device name : \RISU-40xxx\40xxx\F411xxx\F411xxx
Flash size : 1 MBytes (default)
Device type : MCU
Device CPU : Cortex-M4
BL Version : --

32 (shift right by 3):



The screenshot shows the STM32CubeIDE IDE with the main.c file open. The code is for a STM32F407VGT6 microcontroller. The main function includes a while loop for user code execution. The Variable View window on the right shows the current values of the trmt_data array, which is of type uint8_t [8]. The values are: trmt_data[0] = 15, trmt_data[1] = 19, trmt_data[2] = 10, trmt_data[3] = 31, trmt_data[4] = 0, trmt_data[5] = 5, trmt_data[6] = 30, and trmt_data[7] = 6.

File Edit Source Refactor Navigate Search Project Run Window Help

Deb... x Proj... x project1.c x main.c x startup_stm32407vgt6.s x stm32f4xx_it.c x main_testing.c x stm32f4xx_hal.c x system_stm32f4xx.c

project1 [STM32 Cortex-M] x

project1.c [cores: 0]

Thread #1 [main]

main.c at main.c

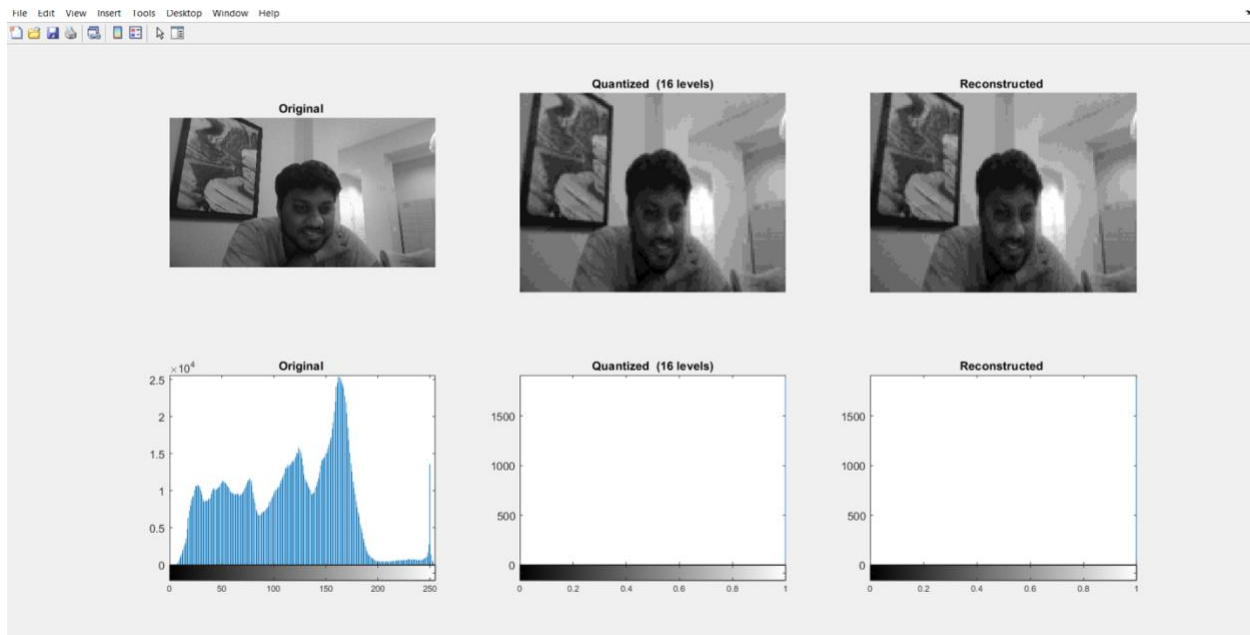
arm-none-eabi-gcc (10)

ST-LINK (ST-LINK GD)

```

153 //gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
154 gray_level_quantization_hybrid(trmt_data, BUFFER_SIZE, 3);
155 //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
156 //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
157
158 /* USER CODE END 2 */
159
160
161 /* Infinite loop */
162 /* USER CODE BEGIN WHILE */
163 while (1)
164 {
165     /* USER CODE END WHILE */
166
167     /* USER CODE BEGIN 3 */
168
169 }
170
171 /* USER CODE END 3 */
172
173 /**
174  * @brief System Clock Configuration
175  * @retval None
176  */
177 void SystemClock_Config(void)
178 {
179     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
180     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
181
182     /** Configure the main internal regulator output voltage
183     */
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1
```

16 (shift right by 4):



```
File Edit Source Refactor Navigate Search Project Run Window Help
project1.ioc main.c startup_stm32f407vgtx.s stm32f4xx_it.c main_testing.c stm32f4xx_hal.c system_stm32f4xx.c
project1 [STM32 Cortex-M]
  project1.elf [cores: 0]
    Thread #1 [main] 1p
      main() at main_testing.c:10
      ST-LINK (ST-LINK GDB)

135 /* USER CODE END Init */
136
137 /* Configure the system clock */
138 SystemClock_Config();
139
140 /* USER CODE BEGIN SysInit */
141
142 /* USER CODE END SysInit */
143
144 /* Initialize all configured peripherals */
145 MX_GPIO_Init();
146 MX_USART2_UART_Init();
147 /* USER CODE BEGIN 2 */
148
149 // test your image processing functions here
150
151
152 //gray_level_transformation2_hybrid(trmt_data, BUFFER_SIZE, A, B);
153 //gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
154 gray_level_quantization_hybrid(trmt_data, BUFFER_SIZE, 4);
155 //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
156 //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
157
158 /* USER CODE END 2 */
159
160 /* Infinite loop */
161 /* USER CODE BEGIN WHILE */
162 while (1)
163 {
164
165 /* USER CODE END WHILE */
```

Expression	Type	Value
trmt_data	uint8_t [8]	0x20000000 <trmt...
trmt_data[0]	uint8_t	7 '\a'
trmt_data[1]	uint8_t	9 '\t'
trmt_data[2]	uint8_t	5 '\005'
trmt_data[3]	uint8_t	15 '\017'
trmt_data[4]	uint8_t	0 '\0'
trmt_data[5]	uint8_t	2 '\002'
trmt_data[6]	uint8_t	15 '\017'
trmt_data[7]	uint8_t	3 '\003'

Download verified successfully

Gray Level Transformation:

Transformation 1:

The screenshot shows the STM32CubeIDE interface with the following components:

- Code Editor:** Displays the `main.c` file. A green box highlights the following code block:

```
151 // test your image processing functions here
152 gray_level_transformation2_hybrid(trmt_data, BUFFER_SIZE, A, B);
153 gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
154 //gray_level_quantization_hybrid(trmt_data, BUFFER_SIZE, 4);
155 //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
156 //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
157
158 /* USER CODE END 2 */
159
160 /* Infinite loop */
161 /* USER CODE BEGIN WHILE */
162 while (1)
163 {
164     /* USER CODE END WHILE */
165 }
166 /* USER CODE BEGIN 3 */
167
168 }
169 /* USER CODE END 3 */
170
171 }
```
- Variable View:** A green box highlights the variable view showing the values of `trmt_data` array elements:

Expression	Type	Value
trmt_data[0]	uint8_t [8]	0x20000000 <trmt...
trmt_data[1]	uint8_t	133 '305'
trmt_data[2]	uint8_t	100 'd'
trmt_data[3]	uint8_t	175 'm'
trmt_data[4]	uint8_t	1 '001'
trmt_data[5]	uint8_t	248 'x'
trmt_data[6]	uint8_t	215 'x'
trmt_data[7]	uint8_t	15 '\017'
- Console:** Shows the message "Download verified successfully".

A=B=70:

The screenshot shows the STM32CubeIDE interface with the following components:

- Code Editor:** Displays the `main.c` file. A green box highlights the following code block:

```
152 gray_level_transformation2_hybrid(trmt_data, BUFFER_SIZE, A, B);
153 //gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
154 //gray_level_quantization_hybrid(trmt_data, BUFFER_SIZE, 1);
155 //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
156 //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
157
158 /* USER CODE END 2 */
159
160 /* Infinite loop */
161 /* USER CODE BEGIN WHILE */
162 while (1)
163 {
164     /* USER CODE END WHILE */
165 }
166 /* USER CODE BEGIN 3 */
167
168 }
169 /* USER CODE END 3 */
170
171 }
```
- Variable View:** A green box highlights the variable view showing the values of `trmt_data` array elements:

Expression	Type	Value
trmt_data[0]	uint8_t [8]	0x20000000 <trmt...
trmt_data[1]	uint8_t	122 'z'
trmt_data[2]	uint8_t	155 '\233'
trmt_data[3]	uint8_t	80 'P'
trmt_data[4]	uint8_t	254 'p'
trmt_data[5]	uint8_t	7 'a'
trmt_data[6]	uint8_t	40 'l'
trmt_data[7]	uint8_t	240 'o'
- Console:** Shows the message "Download verified successfully".

A=100, B=70:

The screenshot shows an IDE with a C++ project for an STM32 Cortex-M. The code is in a while loop, and the variable watch window shows the state of the `trmt_data` array elements.

```
145 /* Initialize all configured peripherals */
146 MX_GPIO_Init();
147 MX_USART2_UART_Init();
148 /* USER CODE BEGIN 2 */
149
150 // test your image processing functions here
151
152 gray_level_transformation2_hybrid(trmt_data, BUFFER_SIZE, A, B);
153 //gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
154 //gray_level_quantization_hybrid(trmt_data, BUFFER_SIZE,4);
155 //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
156 //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
157
158
159 /* USER CODE END 2 */
160
161 /* Infinite loop */
162 /* USER CODE BEGIN WHILE */
163 while (1)
164 {
165     /* USER CODE END WHILE */
166
167     /* USER CODE BEGIN 3 */
168
169     }
170 /* USER CODE END 3 */
171 }
172
173 /**
174  * @brief System Clock Configuration
175  * Details: ...
176  */
177
178
```

Expression	Type	Value
trmt_data	uint8_t [8]	0x20000000 <trmt...
trmt_data[0]	uint8_t	143 '\217'
trmt_data[1]	uint8_t	171 '\r'
trmt_data[2]	uint8_t	108 't'
trmt_data[3]	uint8_t	254 'b'
trmt_data[4]	uint8_t	10 '\n'
trmt_data[5]	uint8_t	57 '9'
trmt_data[6]	uint8_t	242 'o'
trmt_data[7]	uint8_t	71 'G'

Download verified successfully

A=70, B=100:

The screenshot shows an IDE with a C++ project for an STM32 Cortex-M. The code is in a while loop, and the variable watch window shows the state of the `trmt_data` array elements.

```
141 /* USER CODE BEGIN SysInit */
142
143 /* USER CODE END SysInit */
144
145 /* Initialize all configured peripherals */
146 MX_GPIO_Init();
147 MX_USART2_UART_Init();
148 /* USER CODE BEGIN 2 */
149
150 // test your image processing functions here
151
152 gray_level_transformation2_hybrid(trmt_data, BUFFER_SIZE, A, B);
153 //gray_level_transformation1_hybrid(trmt_data, BUFFER_SIZE);
154 //gray_level_quantization_hybrid(trmt_data, BUFFER_SIZE,4);
155 //band_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD1, THRESHOLD2);
156 //global_thresholding_hybrid(trmt_data, BUFFER_SIZE, THRESHOLD);
157
158
159 /* USER CODE END 2 */
160
161 /* Infinite loop */
162 /* USER CODE BEGIN WHILE */
163 while (1)
164 {
165     /* USER CODE END WHILE */
166
167     /* USER CODE BEGIN 3 */
168
169     }
170 /* USER CODE END 3 */
171 }
172
173 /**
174  * @brief System Clock Configuration
175  * Details: ...
176  */
177
178
```

Expression	Type	Value
trmt_data	uint8_t [8]	0x20000000 <trmt...
trmt_data[0]	uint8_t	96 'P'
trmt_data[1]	uint8_t	135 '\207'
trmt_data[2]	uint8_t	56 '8'
trmt_data[3]	uint8_t	253 'y'
trmt_data[4]	uint8_t	4 '\004'
trmt_data[5]	uint8_t	28 '\034'
trmt_data[6]	uint8_t	237 '7'
trmt_data[7]	uint8_t	35 '#'

Download verified successfully

3. Codes:

Thresholding Functions:

Global Thresholding C:

```
void global_thresholding_c(uint8_t *x, uint32_t size, uint8_t threshold) {  
    for(int i=0;i<size;i++) { if (x[i]>=threshold) {  
        x[i]=255; }  
        else {  
            x[i]=0; }  
    } }  
}
```

Band Thresholding C:

```
void band_thresholding_c(uint8_t *x, uint32_t size, uint8_t threshold1, uint8_t threshold2) {  
    for (int i = 0; i < size; i++) {  
        // Check if the pixel value is within the threshold range  
        if ((x[i] >= threshold1) && (x[i] <= threshold2)) {  
            x[i] = 255; // Set the pixel to white if within the range  
        } else {  
            x[i] = 0; // Set the pixel to black otherwise  
        }  
    }  
}  
}
```

Band Thresholding Hybrid (Grad Students Only):

```
__attribute__((naked)) void band_thresholding_hybrid(uint8_t *x, uint32_t size, uint8_t threshold1, uint8_t threshold2) {
```

```
    __asm volatile (
```

```
        "PUSH {r4, r5, r6, r7, lr}\n\t"    // Save registers and the link register
        "MOV r4, #0\n\t"                  // Initialize loop counter r4 to 0
        "MOV r5, #255\n\t"                // r5 will hold the value 255
        "MOV r6, #0\n\t"                  // r6 will hold the value 0
        "band_loop: CMP r4, r1\n\t"        // Compare loop counter with size
        "BGE band_exit\n\t"              // If counter >= size, exit loop
        "LDRB r7, [r0, r4]\n\t"          // Load the pixel value into r7
        "CMP r7, r3\n\t"                  // Compare pixel with upper threshold
        "BHI next_pixel\n\t"             // If pixel > threshold2, go to next_pixel
        "CMP r7, r2\n\t"                  // Compare pixel with lower threshold
        "BLO next_pixel\n\t"             // If pixel < threshold1, go to next_pixel
        "STRB r5, [r0, r4]\n\t"          // Else, pixel is within range, set to 255
        "B increment_counter\n\t"        // Skip to increment_counter
        "next_pixel: STRB r6, [r0, r4]\n\t" // Pixel is outside range, set to 0
        "increment_counter: ADD r4, r4, #1\n\t" // Increment loop counter
        "B band_loop\n\t"                // Continue loop
        "band_exit: POP {r4, r5, r6, r7, pc}\n\t" // Restore registers and return
```

```
);
```

```
}
```

Semi Thresholding C:

```
void semi_thresholding_c(uint8_t *x, uint32_t size, uint8_t threshold) {  
    for(int i = 0; i < size; i++) {  
        if(x[i] < threshold) {  
            x[i] = 0; // Set to zero if the pixel value is below the threshold  
        }  
    }  
}
```

Semi Thresholding Hybrid:

```
__asm volatile (  
    // x -> r0, size -> r1, threshold -> r2  
  
    "PUSH {r4, r5, r6, lr}\n\t"    // Save registers and the return address on the stack.  
  
    "MOV r3, #0\n\t"                // Initialize loop counter r3 to 0.  
  
    "MOV r5, #0\n\t"                // r5 holds the value to set for pixels below the threshold.  
  
    "loop_semi_thresh: CMP r3, r1\n\t" // Compare loop counter (r3) with size (r1).  
  
    "BGE exit_semi_thresh\n\t"      // If r3 >= r1, exit loop.  
  
    "LDRB r6, [r0, r3]\n\t"        // Load the current pixel value into r6.  
  
    "CMP r6, r2\n\t"                // Compare current pixel value (r6) with threshold (r2).  
  
    "BLT thresholding\n\t"          // If current pixel is less than threshold, branch to thresholding.  
  
    "B continue\n\t"                // Otherwise, continue to the next pixel.  
  
    "thresholding: STRB r5, [r0, r3]\n\t" // Store zero in pixel value if below threshold.  
  
    "continue: ADD r3, r3, #1\n\t"    // Increment loop counter.  
  
    "B loop_semi_thresh\n\t"        // Go back to the beginning of the loop.  
  
    "exit_semi_thresh: POP {r4, r5, r6, pc}\n\t" // Restore registers and return.  
  
);
```


Gray Level Quantization Functions:

Gray Level Quantization C:

```
void gray_level_quantization_c(uint8_t *x, uint32_t size, uint8_t shift_factor) {  
    for(int i = 0; i < size; i++) {  
        x[i] = x[i] >> shift_factor; // Quantize pixel by shifting  
    }  
}
```

Gray Level Quantization Hybrid:

```
__attribute__((naked)) void gray_level_quantization_hybrid(uint8_t *x, uint32_t size, uint8_t  
shift_factor) {  
    __asm volatile (  
        "PUSH {r4, r5, lr}\n\t"        // Save registers  
        "MOV r4, #0\n\t"              // Loop counter  
        "quantize_loop: CMP r4, r1\n\t"  // Compare counter with size  
        "BGE quantize_exit\n\t"        // Exit if done  
        "LDRB r5, [r0, r4]\n\t"        // Load pixel into r5  
        "LSR r5, r5, r2\n\t"          // Logical Shift Right by shift_factor  
        "STRB r5, [r0, r4]\n\t"        // Store back the quantized pixel  
        "ADD r4, r4, #1\n\t"          // Increment counter  
        "B quantize_loop\n\t"          // Loop back  
        "quantize_exit: POP {r4, r5, pc}\n\t" // Restore registers and return  
    );  
}
```

Modified MATLAB script to reconstruct quantized image for display:

```
close all; clear all; % Close figures, clear variables

s1 = serialport('COM4', 115200); % Setup serial comms

configureTerminator(s1, "CR/LF");

flush(s1);

cam = webcam; % Initialize webcam

frame = snapshot(cam); % Capture frame

original_image = rgb2gray(frame); % Grayscale conversion

original_image_resized = imresize(original_image, [96, 128]); % Resize image

% Send image data

imageData = original_image_resized(:)';

write(s1, imageData, "uint8");

% Wait for return data

while (s1.NumBytesAvailable < numel(imageData))

    pause(0.1);

end

% Read and reshape quantized data

quantizedData = read(s1, s1.NumBytesAvailable, "uint8");

quantized_image = reshape(quantizedData, size(original_image_resized));

% Define quantization level and reconstruct image

selected_level = 16; % Example for 16 levels, change from 64,32,14,128

shift_factor = log2(256/selected_level);

reconstructed_image = bitshift(quantized_image, shift_factor);
```

```

% Display images

figure;

subplot(2, 3, 1); imshow(original); title('Original ');

subplot(2, 3, 2); imshow(quantized, []); title(['Quantized (' num2str(selected_level) ' levels)']);

subplot(2, 3, 3); imshow(reconstructed, []); title('Reconstructed ');

% Display histograms

subplot(2, 3, 4); imhist(original); title(' Original Image');

subplot(2, 3, 5); imhist(quantized); title(['Quantized Image (' num2str(selected_level) ' levels)']);

subplot(2, 3, 6); imhist(reconstructed); title('Reconstructed ');

% Cleanup

clear('cam'); % Release webcam

clear s1; % Close serial port

```

Gray Level Transformation Functions:

Gray Level Transformation 1 C:

```

void gray_level_transformation1_c(uint8_t *x, uint32_t size) {
    for(uint32_t i = 0; i < size; i++) {
        x[i] = 255 - x[i]; // Invert pixel
    }
}

```

Gray Level Transformation 1 Hybrid:

```

__attribute__((naked)) void gray_level_transformation1_hybrid(uint8_t *x, uint32_t size) {
    __asm volatile (
        "MOV r2, #0\n\t"           // Counter

```

```

"MOV r3, #255\n\t"          // Inversion constant

"gl_trans1_loop: CMP r2, r1\n\t" // Compare counter with size

"BGE gl_trans1_exit\n\t"     // Exit loop if counter >= size

"LDRB r4, [r0, r2]\n\t"      // Load byte

"SUB r4, r3, r4\n\t"        // Subtract from 255

"STRB r4, [r0, r2], #1\n\t"  // Store byte and post-increment address

"ADD r2, r2, #1\n\t"        // Increment counter

"B gl_trans1_loop\n\t"       // Loop back

"gl_trans1_exit: BX LR\n\t"  // Exit

);

}

```

Gray Level Transformation 2 C:

```

void gray_level_transformation2_c(uint8_t *x, uint32_t size, uint8_t a0, uint8_t b0) {

    uint8_t a2 = 255 - a0;

    uint8_t b2 = 255 - b0;

    for(uint32_t i = 0; i < size; i++) {

        if(x[i] <= b0) {

            x[i] = (uint8_t)((float)a0 * x[i] / b0); // Scale up to a0 for x <= b0

        } else {

            uint8_t p = x[i] - b0;

            x[i] = (uint8_t)((float)p * a2 / b2) + a0; // Scale from a0 to 255 for x > b0

        }

    }

}

```

Gray Level Transformation 2 Hybrid (Grad Students):

```
__attribute__((naked)) void gray_level_transformation2_hybrid(uint8_t *x, uint32_t size, uint8_t a0, uint8_t b0) {
```

```
    __asm volatile (
```

```
        "PUSH {r4-r9, lr}"           // Save registers
```

```
        "MOV r6, #0"                 // Loop counter
```

```
        "MOV r5, #255"               // 255 constant for calculations
```

```
        "loopg2: CMP r6, r1"         // Compare counter with size
```

```
        "BGE exit2"                  // Exit if done
```

```
        "LDRB r4, [r0, r6]"          // Load current pixel value
```

```
        "CMP r4, r3"                 // Compare with b0
```

```
        "BGE greater_than_b0"        // Branch if greater than b0
```

```
        // If less than or equal to b0
```

```
        "SUB r7, r5, r2"              // r7 = 255 - a0
```

```
        "SUB r8, r5, r3"              // r8 = 255 - b0
```

```
        "SUB r9, r4, r3"              // r9 = current pixel value - b0
```

```
        "MUL r7, r7, r9"              // r7 = (255 - a0) * (current pixel value - b0)
```

```
        "UDIV r7, r7, r8"             // Divide r7 by r8
```

```
        "ADD r7, r7, r2"              // Add a0
```

```
        "STRB r7, [r0, r6], #1"       // Store result and post-increment address
```

```
        "ADD r6, #1"                  // Increment counter
```

```
        "B loopg2"                    // Loop back
```

```
        // If greater than b0
```

```
        "greater_than_b0: "
```

```
        "MUL r4, r4, r2"              // Multiply by a0
```

```

    "UDIV r4, r4, r3"           // Divide by b0
    "STRB r4, [r0, r6], #1"     // Store result and post-increment address
    "ADD r6, r6, #1"           // Increment counter
    "B loopg2"                  // Loop back
    "exit2: POP {r4-r9, pc}"    // Restore registers and return
);
}

```

Gray Level Transformation 3 C (Extra Credit):

```

void gray_level_transformation3_c(uint8_t *x, uint32_t size, uint8_t a, uint8_t b) {
    for(uint32_t i = 0; i < size; i++) {
        // Example transformation logic:

        // If the pixel value is less than 'a', set it to 0 (black).

        // If the pixel value is greater than 'b', set it to 255 (white).

        // Otherwise, scale it linearly between 'a' and 'b'.

        if(x[i] < a) {
            x[i] = 0;
        } else if(x[i] > b) {
            x[i] = 255;
        } else {
            // Scale between 'a' and 'b'. Adjust this part to match the desired transformation.

            x[i] = (uint8_t)((x[i] - a) / (float)(b - a) * 255);
        }
    }
}

```

Gray Level Transformation 3 Hybrid (Extra Credit):

```
__attribute__((naked)) void gray_level_transformation3_hybrid(uint8_t *x, uint32_t size, uint8_t A, uint8_t B) {
```

```
    __asm volatile (
```

```
        "PUSH {r4-r8, lr}"           // Save registers and the link register
```

```
        "MOV r4, #0"                 // Loop counter
```

```
        "MOV r7, #255"               // Constant 255
```

```
        "SUB r8, r7, r2"              // 255 - A
```

```
        "SUB r7, r7, r3"              // 255 - B
```

```
        "SUB r7, r7, r8"              // 255 - A - B
```

```
        "MOV r8, #2"                  // Constant 2
```

```
        "MUL r8, r8, r3"              // 2 * B
```

```
        "SUB r7, #255, r8"            // 255 - 2 * B
```

```
        "transform3_loop: CMP r4, r1" // Compare counter with size
```

```
        "BGE transform3_exit"         // Exit if done
```

```
        "LDRB r5, [r0, r4]"           // Load current pixel value into r5
```

```
        // First condition
```

```
        "CMP r5, r2"                  // Compare pixel with A
```

```
        "BLE less_than_A"             // If pixel <= A, branch to less_than_A
```

```
        // Second condition
```

```
        "CMP r5, #255"                // Compare pixel with 255 - B
```

```
        "BHI greater_than_255_minus_B" // If pixel > 255 - B, branch to  
greater_than_255_minus_B
```

```
        // If A < pixel <= 255 - B
```

```
        "SUB r5, r5, r2"              // pixel - A
```

```

    "MUL r5, r5, #255"           // (pixel - A) * 255
    "UDIV r5, r5, r7"           // Divide by 255 - A - B
    "ADD r5, r5, r3"            // Add B
    "STRB r5, [r0, r4], #1"      // Store result and post-increment address
    "B increment_counter"        // Skip to increment_counter

// If pixel <= A
"less_than_A: "
    "MUL r5, r5, #255"           // pixel * 255
    "UDIV r5, r5, r2"           // Divide by A
    "STRB r5, [r0, r4], #1"      // Store result and post-increment address
    "B increment_counter"        // Skip to increment_counter

// If pixel > 255 - B
"greater_than_255_minus_B: "
    "SUB r5, r5, #255"           // pixel - (255 - B)
    "ADD r5, r5, r8"            // Add (255 - 2 * B)
    "STRB r5, [r0, r4], #1"      // Store result and post-increment address
    "increment_counter: "
    "ADD r4, r4, #1"            // Increment counter
    "B transform3_loop"          // Loop back

"transform3_exit: POP {r4-r8, pc}" // Restore registers and return
);
}

```


Histogram Equalization Functions:

Calculate Histogram C:

```
void calculate_histogram_c(uint8_t *image, uint32_t *hist, uint32_t size) {
    for (uint32_t i = 0; i < 256; i++) {
        hist[i] = 0; // Initialize histogram
    }
    for (uint32_t i = 0; i < size; i++) {
        hist[image[i]]++; // Count occurrences of pixel values
    }
}
```

Calculate Histogram Hybrid:

```
__attribute__((naked)) void calculate_histogram_hybrid(uint8_t *image, uint32_t *hist, uint32_t size) {
    __asm volatile (
        "PUSH {r4-r7, lr}\n\t"           // Save used registers and link register
        // Initialize histogram to zero
        "MOV r3, #0\n\t"                 // Set histogram index to 0
        "hist_zero_loop:\n\t"
        "STR r3, [r1, r3, LSL #2]\n\t"    // Zero the histogram bin
        "ADD r3, r3, #1\n\t"             // Increment histogram index
        "CMP r3, #256\n\t"               // Check if we've reached 256 bins
        "BLT hist_zero_loop\n\t"        // Loop if not done
        // Calculate histogram
        "MOV r4, #0\n\t"                 // Set image index to 0
        "calc_hist_loop:\n\t"
        "CMP r4, r2\n\t"                 // Compare image index to size
        "BGE hist_done\n\t"              // If done, branch to end
        "LDRB r5, [r0, r4]\n\t"          // Load pixel value from image
        "LDR r6, [r1, r5, LSL #2]\n\t"    // Load current bin value from histogram
        "ADD r6, r6, #1\n\t"             // Increment bin value
        "STR r6, [r1, r5, LSL #2]\n\t"    // Store updated bin value back to histogram
        "ADD r4, r4, #1\n\t"             // Increment image index
    );
}
```

```

        "B calc_hist_loop\n\t"          // Loop back
        "hist_done:\n\t"
        "POP {r4-r7, pc}\n\t"          // Restore registers and return
        :
        : "r" (image), "r" (hist), "r" (size)
        : "r3", "r4", "r5", "r6", "memory"
    );
}

```

Map Levels C:

```

void map_levels_c(uint32_t *hist, uint8_t *mapping_table, uint32_t size) {
    uint32_t sum = 0;
    for (uint32_t i = 0; i < 256; i++) {
        sum += hist[i];
        mapping_table[i] = (uint8_t)((sum * 255) / size); // Cast the result to uint8_t
    }
}

```

Map Levels Hybrid:

```

__attribute__((naked)) void map_levels_hybrid(uint32_t *hist, uint8_t *mapped_levels, uint32_t size,
uint16_t levels) {
    __asm volatile (
        "PUSH {r4-r7, lr}\n\t"          // Save used registers and link register
        "MOV r4, #0\n\t"                // Initialize sum to 0
        "MOV r5, #0\n\t"                // Loop counter for levels
        "map_loop:\n\t"
        "CMP r5, r3\n\t"                 // Compare counter with levels
        "BGE map_done\n\t"              // If done, branch to end
        "LDR r6, [r0, r5, LSL #2]\n\t"   // Load histogram bin value
        "ADD r4, r4, r6\n\t"             // Add to sum
        "MOV r6, r4\n\t"                 // Copy sum to r6
        "MUL r6, r6, #255\n\t"           // Multiply by 255 (L-1)
        "UDIV r6, r6, r2\n\t"            // Divide by total number of pixels
        "STRB r6, [r1, r5]\n\t"          // Store mapped level
        "ADD r5, r5, #1\n\t"             // Increment loop counter
        "B map_loop\n\t"                 // Loop back
    );
}

```

```

    "map_done:\n\t"
    "POP {r4-r7, pc}\n\t"          // Restore registers and return
    :
    : "r" (hist), "r" (mapped_levels), "r" (size), "r" (levels)
    : "r4", "r5", "r6", "memory"
);
}

```

Transform Image C:

```

void transform_image_c(uint8_t *image, uint8_t *mapping_table, uint32_t size) {
    for (uint32_t i = 0; i < size; i++) {
        image[i] = mapping_table[image[i]]; // Apply mapping to each pixel
    }
}

```

Transform Image Hybrid:

```

__attribute__((naked)) void transform_image_hybrid(uint8_t *image, uint8_t *mapping_table, uint32_t
size) {
    __asm volatile (
        "PUSH {r4-r6, lr}\n\t"          // Save registers
        // Loop over the image to apply the mapping table
        "MOV r4, #0\n\t"                // Counter for loop
        "transform_loop:\n\t"
        "CMP r4, r2\n\t"                // Compare counter with size
        "BGE transform_end\n\t"         // If end of image, exit loop
        "LDRB r5, [r0, r4]\n\t"         // Load the current pixel value
        "LDRB r5, [r1, r5]\n\t"         // Get the mapped value from the mapping table
        "STRB r5, [r0, r4]\n\t"         // Store the transformed value back to the image
        "ADD r4, r4, #1\n\t"            // Increment the loop counter
        "B transform_loop\n\t"          // Loop back
        "transform_end:\n\t"
        "POP {r4-r6, pc}\n\t"           // Restore registers and return
    );
}

```

