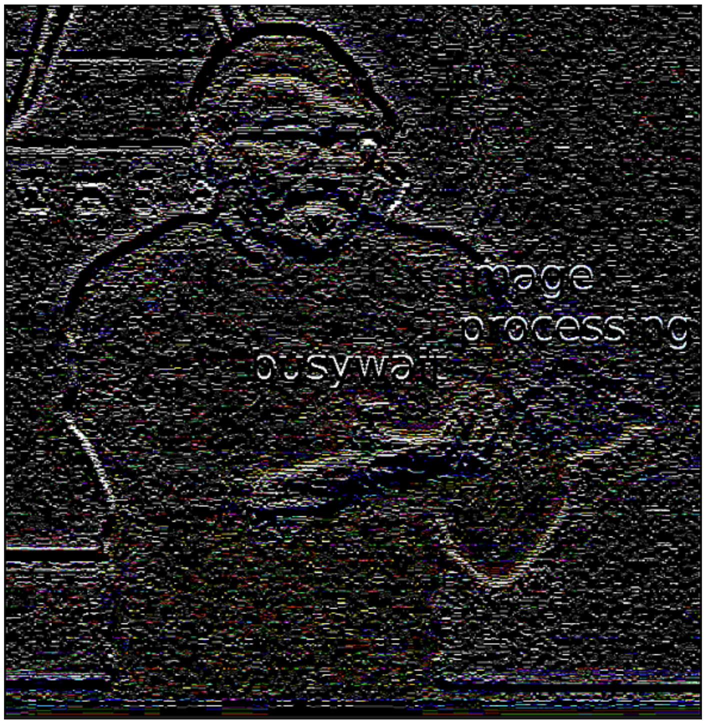
a)



Observations:
It seems that edge detection has been applied to the image to get major outlines. It seems to have gotten the major outline of the body, some of the letters and the graph at the back.

Client Output:

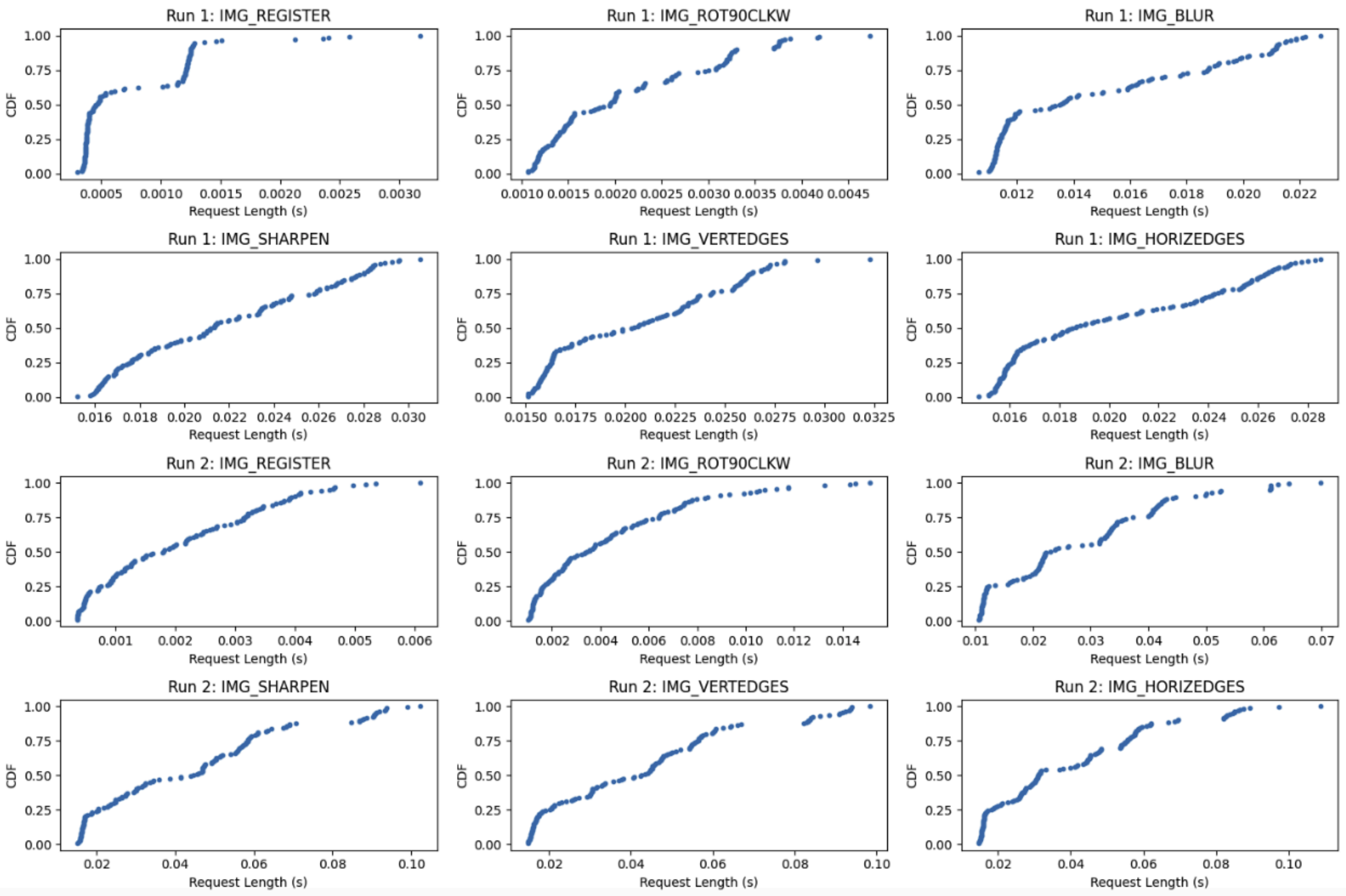
```
[rsudhir@scc-ge2 HW6]$ ./client -a 30 -I images/ -n 5 1234
[#CLIENT#] INFO: CS350 Client Version 4.1
[#CLIENT#] INFO: Found BMP 0: test1.bmp
[#CLIENT#] INFO: Found BMP 1: test6.bmp
[#CLIENT#] INFO: Found BMP 2: test5.bmp
[#CLIENT#] INFO: Found BMP 3: test4.bmp
[#CLIENT#] INFO: Found BMP 4: test3.bmp
[#CLIENT#] INFO: Found BMP 5: test2.bmp
[#CLIENT#] INFO: Reading BMP 0: test1.bmp | HASH = 9f3363f0249c15163d52e60fd9544c31
[#CLIENT#] INFO: Reading BMP 1: test6.bmp | HASH =
11552ac97535bd4433891b63ed1dd45d
[#CLIENT#] INFO: Reading BMP 2: test5.bmp | HASH =
5597b44eaee51bd81292d711c86a3380
[#CLIENT#] INFO: Reading BMP 3: test4.bmp | HASH = 0caaef67aee1775ffca8eda02bd85f25
[#CLIENT#] INFO: Reading BMP 4: test3.bmp | HASH = f2ac174476fb2be614e8ab1ae10e82f0
[#CLIENT#] INFO: Reading BMP 5: test2.bmp | HASH = b6770726558da9722136ce84f12bfac8
[#CLIENT#] INFO: setting client port as: 1234
```

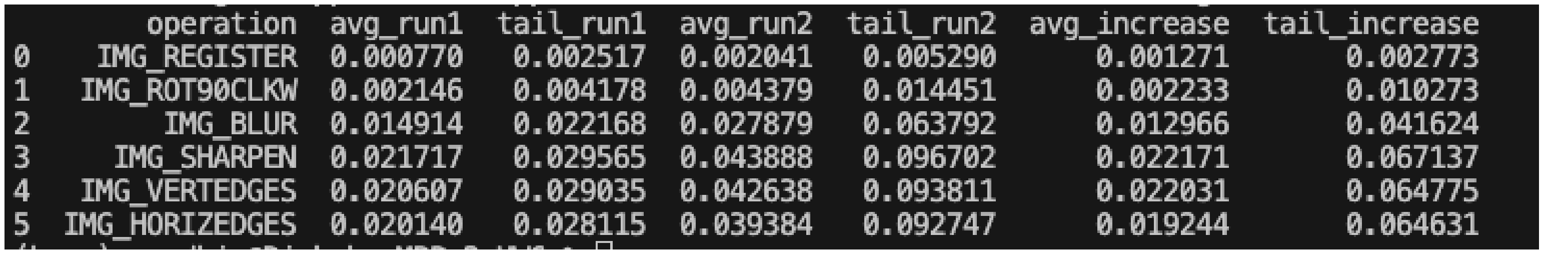
```
[#CLIENT#] INFO: setting distribution: 0
[#CLIENT#] INFO: Initiating connection...
[#CLIENT#] PREP REQ 0
[#CLIENT#] SENT REQ 0 [OPCODE = IMG_REGISTER, OW = 1, IMG_ID = 4]
[#CLIENT#] RESP REQ 0
[#CLIENT#] PREP REQ 1
[#CLIENT#] SENT REQ 1 [OPCODE = IMG_HORIZEDGES, OW = 1, IMG_ID = 0]
[#CLIENT#] RESP REQ 1
[#CLIENT#] PREP REQ 2
[#CLIENT#] SENT REQ 2 [OPCODE = IMG_SHARPEN, OW = 1, IMG_ID = 0]
[#CLIENT#] PREP REQ 3
[#CLIENT#] SENT REQ 3 [OPCODE = IMG_SHARPEN, OW = 1, IMG_ID = 0]
[#CLIENT#] PREP REQ 4
[#CLIENT#] SENT REQ 4 [OPCODE = IMG_RETRIEVE, OW = 1, IMG_ID = 0]
[#CLIENT#] RESP REQ 2
[#CLIENT#] RESP REQ 3
[#CLIENT#] RESP REQ 4
[#CLIENT#] ==== REPORT ====
[#CLIENT#] R[0]: Sent: 3059875.165319008 Recv: 3059875.183168635 Opcode:
IMG_REGISTER    OW: 1 ClientImgID: 4    ServerImgID: 0    Rejected: No HASH:
f2ac174476fb2be614e8ab1ae10e82f0
[#CLIENT#] R[1]: Sent: 3059875.244289655 Recv: 3059875.283796780 Opcode:
IMG_HORIZEDGES  OW: 1 ClientImgID: 0    ServerImgID: 0    Rejected: No
[#CLIENT#] R[2]: Sent: 3059875.295242851 Recv: 3059875.335246850 Opcode:
IMG_SHARPEN     OW: 1 ClientImgID: 0    ServerImgID: 0    Rejected: No
[#CLIENT#] R[3]: Sent: 3059875.302584848 Recv: 3059875.365160111 Opcode:
IMG_SHARPEN     OW: 1 ClientImgID: 0    ServerImgID: 0    Rejected: No
[#CLIENT#] R[4]: Sent: 3059875.313435683 Recv: 3059875.392362818 Opcode:
IMG_RETRIEVE    OW: 1 ClientImgID: 0    ServerImgID: 0    Rejected: No HASH:
e42f270e6a20bfc2e58c6b80efff5218
[#CLIENT#] DONE!
```

Observations on Client Output:

From the client output, we can see that IMG_HORIZEDGES, IMG_SHARPEN have been applied, which confirms that some form of edge detection has occurred. Multiple image sharpens occurred, which explained the color of the image.

b)





```
        operation  avg_run1  tail_run1  avg_run2  tail_run2  avg_increase  tail_increase
0    IMG_REGISTER  0.000770   0.002517  0.002041   0.005290      0.001271       0.002773
1   IMG_ROT90CLKW  0.002146   0.004178  0.004379   0.014451      0.002233       0.010273
2        IMG_BLUR  0.014914   0.022168  0.027879   0.063792      0.012966       0.041624
3     IMG_SHARPEN  0.021717   0.029565  0.043888   0.096702      0.022171       0.067137
4   IMG_VERTEDGES  0.020607   0.029035  0.042638   0.093811      0.022031       0.064775
5  IMG_HORIZEDGES  0.020140   0.028115  0.039384   0.092747      0.019244       0.064631
```

Now look at the plots, and answer the following: (1) within the same run, what operations are similar in behavior and which ones behave differently? (2) Within the same run, which ones are the least predictable operations? (3) Across runs, by how much does the average response time increase for each operation? (4) Across runs, by how much does the 99% tail latency increase for each operation?

1)

Similar Behaviors:

- In Run 1 IMG_SHARPEN and IMG_HORIZEDGES seem to have somewhat similar CDF shapes, indicating that they may have comparable request lengths. Likewise, IMG_BLUR and IMG_VERTEDGES in Run 1 have gradual slopes, showing similar response times.
- In Run 2, IMG_VERTEDGES and IMG_SHARPEN have comparable, steeper slopes indicating a similar distribution of response times for these operations.

Different Behaviors:

- Different slopes or shapes indicate variability in request lengths. For example, IMG_REGISTER in Run 1 has a very steep CDF, while IMG_ROT90CLKW has a more gradual slope, indicating higher variability in request times for IMG_ROT90CLKW compared to IMG_REGISTER.
- In Run 2, IMG_BLUR shows a wide spread in request times (gradual CDF), while IMG_REGISTER has a steeper slope, suggesting that IMG_BLUR is more variable and possibly more complex to process.

2)

less predictable operations tend to have a gradual, less steep CDF, as this suggests higher variability in request lengths.

- **Run 1:** IMG_ROT90CLKW and IMG_VERTEDGES have more gradual slopes, suggesting they are less predictable than operations like IMG_REGISTER, which is faster and more consistent.
- **Run 2:** IMG_BLUR and IMG_VERTEDGES exhibit more gradual CDF slopes, indicating they may be less predictable. IMG_BLUR especially has a longer tail, suggesting occasional long response times.

3)

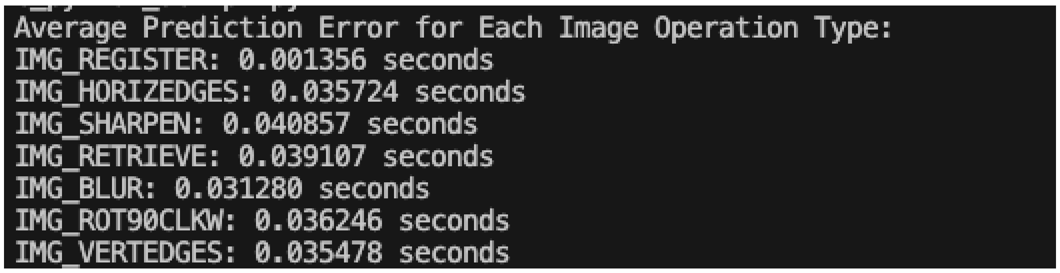
Comparing avg_run1 and avg_run2 for each operation in the summary table provides insights into how much average response time increased between Run 1 and Run 2.

- The column avg_increase in the summary table shows that most operations have experienced an increase in average response time, with IMG_BLUR showing a significant increase (0.012966 seconds) between runs.
- This suggests that processing a larger variety of images (as in Run 2) increases the average response time across the board, likely due to handling larger images or more computationally intensive operations.

4)

Tail latency measures the "worst-case" performance. The tail_increase column shows how much the 99th percentile of response time has increased for each operation between the two runs.
Notably, IMG_BLUR and IMG_SHARPEN have significant tail latency increases (0.041624 seconds and 0.067137 seconds, respectively). This indicates that these operations occasionally experience very high processing times, likely due to computational complexity or larger image sizes in Run 2.
Operations like IMG_REGISTER and IMG_ROT90CLKW have smaller tail latency increases, suggesting they are generally more predictable, even with the added complexity in Run 2.

C)



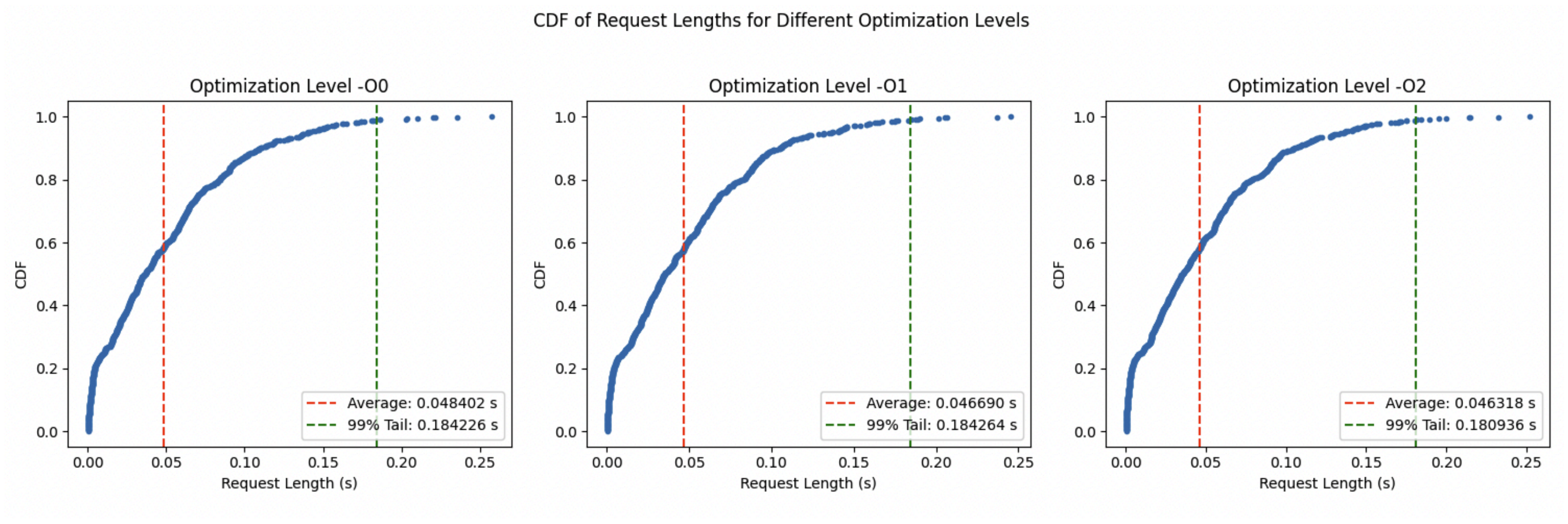
```
Average Prediction Error for Each Image Operation Type:
IMG_REGISTER: 0.001356 seconds
IMG_HORIZEDGES: 0.035724 seconds
IMG_SHARPEN: 0.040857 seconds
IMG_RETRIEVE: 0.039107 seconds
IMG_BLUR: 0.031280 seconds
IMG_ROT90CLKW: 0.036246 seconds
IMG_VERTEDGES: 0.035478 seconds
```

Most Predictable Operation: IMG_REGISTER is the most predictable operation, with the lowest average prediction error. This consistency might be due to the simplicity or lower computational intensity of this operation compared to others.

Least Predictable Operations: IMG_SHARPEN, IMG_HORIZEDGES, and IMG_ROT90CLKW have relatively high prediction errors, suggesting more variability in their processing times.

Operations with low error, like IMG_REGISTER, are relatively easy to forecast with the EWMA model due to their consistent processing times. However, the higher errors in operations such as IMG_SHARPEN and IMG_HORIZEDGES imply that these operations might depend on factors that introduce variability, making them harder to predict with EWMA alone.

D)



The -0 flags:

The -O flags control the optimization level of the compilation:

- **-O0: No optimization.** This setting prioritizes quick compilation over performance, leading to unoptimized code that is slower but more straightforward.
- **-O1: Basic optimization.** This performs optimizations that don't significantly affect compile time, aiming to improve execution speed slightly.
- **-O2: More advanced optimization.** This includes all -O1 optimizations, along with more aggressive ones, for better runtime performance.

Higher optimization levels should theoretically yield faster processing times.

The optimization flags appear to be functioning as expected. As the optimization level increases, the performance improves, with slight decreases in both average and 99% tail latency. This suggests that the compiler optimizations are indeed making the code more efficient.

The difference in tail latencies across the levels is minor, suggesting that while optimizations help, there may be limits to how much can be gained without more significant changes to the code structure or algorithmic efficiency.