Assignment#3-2

25.02.13

1. FFmpeg-based multi-camera (14 connections) streaming w/ Thread

FPS 를 실제 cv2.imshow 에서 시각화 되는 이미지 속도로 정의

(cap.read()로 부터 읽어오는 프레임 수가 아닌)

-> 추후 fps metric 에 대한 조사가 더 필요할듯

또한, 이미지 시각화를 main thread 에서 할 경우 병목현상이 발생함을 확인하였고, 멀티카메러라로 갈수록 필수로 cv2.imshow 를 각 쓰레드에서 불러와야함

Test Cases :

1. *Single camera w/o Thread : 15fps, 0.2gb 메모리 사용*
2. *Single camera w/ Thread : 15fps, 16 개중 4 개의 코어 사용(laptop 기준), 0.2gb 메모리 사용*
3. *Multi camera(14 cons) w/o Thread : 7 fps, 16 개중 16 개의 코어 사용, 2.5gb 메모리사용*

Env1 갯수 변화

Env2 멀티프로세스

멀티프로세스(python multi interpreter) but visualize 시 버퍼활용필요, 멀티쓰레드(one interpreter)

Loss 최소화 목표

Final :

4. *Multi camera(14 cons) w/ Thread : 11 fps, 16 개중 16 개의 코어 사용, 5gb 메모리사용*

Cv2.putText 영향

-> fps 다시 계산 필요

Mult camera w/ Thread 에서의 접근 방법

Try 1. Using Que

FPS 가 증가하나 latency 가 증가함 -> 버퍼 현상이 자주 일어남

FPS 평균 7->15 로 증가, latency 평균 70~80ms 으로 증가

메모리 사용량은 변화없음

FPS 가 증가하는 것은 좋으나 Que 를 사용함으로 인해 바로 바로다음 프레임을 시각화 하려다가, 현재 시점과 시각화 하고 있는 시점간 차이가 발생함. (-> latency 증가 원인, latency 가 증가하다가 limit 에 도달하면 버퍼 발생)

Try 2. Using current frame as variable

따라서 que 를 사용하지 않고 최근 프레임을 가져오는 방식으로 하면 최대한 현재시점 과 시각화 하는 시점을 유사하게 유지하며 버퍼최소화

FPS 가 증가하면서 latency 값은 1~80 사이 에에서 증가하다 1 로 초기화 하기를 반복하면서 버퍼 현상 최소화

2. GStreamer-based multi-camera (14 connections) streaming w/ Thread

(추후예정)