컴퓨터비전특론 homework 5

problem 1

Practice Image Retrieval using MATLAB or OpenCV. We use the bookCovers dataset (come under the MATLAB folder ’toolbox/vision/visiondata/’ or uploaded in LMS) that contains images of various books’ covers. In addition, under the queries subfolder, there are three images of book covers that
will be used as query images.
The workflow is as follows:

1. Index the images (i.e., book1.jpg to book58.jpg). In other words, create visual vocabulary and create inverted image indexes to map from visual words to images.
2. Use an (unindexed) image (i.e., query1.jpg to query3.jpg) to query the index and find the most similar image.
   The result should be identical to Fig. 1.

---

problem 1 answer code

code 1

```
import cv2
import numpy as np
import os

dir = '/home/sivvon/Desktop/CV_Class_HW/CV_HW5/bookCovers/'
query_path = '/home/sivvon/Desktop/CV_Class_HW/CV_HW5/bookCovers/queries/'
save_dir = '/home/sivvon/Desktop/CV_Class_HW/cv_hw5_results/'

os.makedirs(save_dir,exist_ok=True)

orb = cv2.ORB_create(
    nfeatures=2000,
    scaleFactor=1.4,
    nlevels=8,
    edgeThreshold=31,
    firstLevel=0,
    WTA_K=2,
    scoreType=cv2.ORB_HARRIS_SCORE,
    patchSize=31,
    fastThreshold=20,
)

image_index = {}
training_descriptors = []
```

```
# image index
k=0
t=0
for i in range(1, 59):
    # image load
    img_path = dir+f'book{i}.jpg'
    img_name = f'book{i}.jpg'
    # print(img_path)
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    # feature
    keypoints, descriptors = orb.detectAndCompute(img, None)
    descriptors = np.float32(descriptors)

    training_descriptors.append(descriptors)

# print(training_descriptors[0])

# make visual vocabulary
vocabularySize = 60
descriptors = np.concatenate(training_descriptors)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.01)
attempts = 5
flags = cv2.KMEANS_RANDOM_CENTERS

_, labels, vocabulary = cv2.kmeans(descriptors, vocabularySize, None, criteria, attemp
ts, flags)

# Save the visual vocabulary to a file
fs = cv2.FileStorage(save_dir+"visual_vocabulary.xml", cv2.FILE_STORAGE_WRITE)
fs.write("vocabulary", vocabulary)
fs.release()
```

code 2

```
import cv2
import numpy as np
import os
from glob import glob
import natsort

dirdir = '/home/sivvon/Desktop/CV_Class_HW/CV_HW5/bookCovers/'
query_path = '/home/sivvon/Desktop/CV_Class_HW/CV_HW5/bookCovers/queries/'
save_dir = '/home/sivvon/Desktop/CV_Class_HW/cv_hw5_results/'

os.makedirs(save_dir,exist_ok=True)

fs = cv2.FileStorage(save_dir+"visual_vocabulary.xml", cv2.FILE_STORAGE_READ)
visualVocabulary = fs.getNode("vocabulary").mat()
fs.release()

queryImages = []
for img in os.listdir(query_path):
    queryImages.append(query_path+img)
```

```
queryImages = natsort.natsorted(queryImages)
print(queryImages)

orb = cv2.ORB_create(
    nfeatures=2000,
    scaleFactor=1.4,
    nlevels=8,
    edgeThreshold=31,
    firstLevel=0,
    WTA_K=2,
    scoreType=cv2.ORB_HARRIS_SCORE,
    patchSize=31,
    fastThreshold=20,
)

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 1)
search_params = dict(checks = 60)
flann = cv2.FlannBasedMatcher(index_params, search_params)

# clustering function
def cluster(des, voca):
    matches = flann.knnMatch(des, voca, k=1)
    clusterAssignment = np.zeros((1, voca.shape[0]), dtype=np.float32)

    for match in matches:
        clusterIndex = match[0].trainIdx
        clusterAssignment[0, clusterIndex] += 1

    return clusterAssignment

for queryIndex, queryImagePath in enumerate(queryImages):
    queryImageName = queryImagePath.split("/")[-1]
    queryImage = cv2.imread(queryImagePath)

    if queryImage is None:
        print("Failed to read query image:", queryImagePath)
        continue

    _, query_descriptor = orb.detectAndCompute(queryImage, None)
    query_descriptor = np.float32(query_descriptor)

    query_clustter_assignment = cluster(query_descriptor, visualVocabulary)

    best_match_score = np.finfo(np.float64).max
    best_match_img = ""

    for bookIndex in range(1,59):
        book_path = dirdir + 'book' + str(bookIndex) + '.jpg'
        book_name = f'book{bookIndex}.jpg'
        book_img = cv2.imread(book_path)

        if book_img is None:
            print(f"book image 를 찾지 못했습니다: {book_path}")
            continue

        _, book_descriptor = orb.detectAndCompute(book_img, None)
```

```
        book_descriptor = np.float32(book_descriptor)

        book_cluster_assignment = cluster(book_descriptor, visualVocabulary)

        # calculate match score (origin book - query)
        match_score = cv2.compareHist(query_clustter_assignment, book_cluster_assignme
nt, cv2.HISTCMP_CHISQR)

        # Update the best match
        if match_score < best_match_score:
            best_match_score = match_score
            best_match_img = book_img

    # show query img matches
    cv2.imshow(f'query{queryIndex}', queryImage)
    save_q = save_dir+f'querty{queryIndex}.jpg'
    cv2.imwrite(save_q, queryImage)
    cv2.waitKey(0)
    cv2.imshow(f'best image {queryIndex}', best_match_img)
    save_b = save_dir+f'best image {queryIndex}.jpg'
    cv2.imwrite(save_b, best_match_img)
    cv2.waitKey(0)
    result = cv2.hconcat([queryImage, best_match_img])
    cv2.imshow(f'matching result {queryIndex}', result)
    save_match = save_dir+f'matching result {queryIndex}.jpg'
    cv2.imwrite(save_match, result)
    cv2.waitKey(0)
```

---

problem 1 code analysis

code 1

```
import cv2
import numpy as np
import os

dir = '/home/sivvon/Desktop/CV_Class_HW/CV_HW5/bookCovers/'
query_path = '/home/sivvon/Desktop/CV_Class_HW/CV_HW5/bookCovers/queries/'
save_dir = '/home/sivvon/Desktop/CV_Class_HW/cv_hw5_results/'

os.makedirs(save_dir,exist_ok=True)

orb = cv2.ORB_create(
    nfeatures=2000,
    scaleFactor=1.4,
    nlevels=8,
    edgeThreshold=31,
    firstLevel=0,
    WTA_K=2,
    scoreType=cv2.ORB_HARRIS_SCORE,
    patchSize=31,
    fastThreshold=20,
)
```

먼저 위와 같이 ORB (Oriented FAST and Rotated BRIEF) 디스크립터를 생성한다.

```
image_index = {}
training_descriptors = []

# image index
k=0
t=0
for i in range(1, 59):
    # image load
    img_path = dir+f'book{i}.jpg'
    img_name = f'book{i}.jpg'
    # print(img_path)
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)

    # feature
    keypoints, descriptors = orb.detectAndCompute(img, None)
    descriptors = np.float32(descriptors)

    training_descriptors.append(descriptors)
```

다음으로 이미지 인덱스를 저장할 딕셔너리 image_index 를 선언하고, training 에 사용할 descriptor 를 저장하는 리스트인 trainging_descriptors 를 생성한다.

for 반복문은 image index 생성을 위한 것이다. 주어진 book cover image 개수만큼 범위를 지정하고, 이미지를 load 한 후 grayscale 해서 descriptor 를 구한다. 구한 디스크립터는 training descriptor list 에 저장한다.

```
# make visual vocabulary
vocabularySize = 60
descriptors = np.concatenate(training_descriptors)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.01)
attempts = 5
flags = cv2.KMEANS_RANDOM_CENTERS

_, labels, vocabulary = cv2.kmeans(descriptors, vocabularySize, None, criteria, attemp
ts, flags)

# Save the visual vocabulary to a file
fs = cv2.FileStorage(save_dir+"visual_vocabulary.xml", cv2.FILE_STORAGE_WRITE)
fs.write("vocabulary", vocabulary)
fs.release()
```

마지막으로 k-means clustering 을 활용해 visual vocabulary 를 생성하는 과정을 거친다.

kmeans clustering 의 input patameter 중 ncluster(클러스터 개수) 에 해당하는 vocabularySize 는 임의로 지정하되, 결과를 잘 찾을 수 있는 값을 test 를 통해 찾아야 한

다. kmeans clustering 은 클러스터 개수만큼의 centroid 를 무작위로 선택하는 초기화 과정을 거친다.

criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 100, 0.01) 에서 cv2.TERM_CRITERIA_EPS 는 특정 정확도에 다다르면 알고리즘 반복을 멈추게 하는 flag 이며, cv2.TERM_CRITERIA_MAX_ITER 는 특정 반복 횟수를 지나면 알고리즘을 멈추게 하는 flage 이다.

위 코드와 같이 cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER 형태로 작성할 경우 두 조건 중 하나만 만족해도 알고리즘의 반복을 멈추게 한다.

100은 최대 100번을 반복하게 하는 파라미터이며, 0.01 은 요구하는 정확도를 의미한다.

cv2.KMEANS_RANDOM_CENTERS 로 지정된 flags 는 어떻게 초기 중심값을 정할지에 대한 것이다.

이후 k-means clustering 통해 라벨을 달아준다. 라벨을 달아 생성된 visual vocabulary 는 xml file 형태로 우선 저장한다.

code2

```
fs = cv2.FileStorage(save_dir+"visual_vocabulary.xml", cv2.FILE_STORAGE_READ)
visualVocabulary = fs.getNode("vocabulary").mat()
fs.release()

queryImages = []
for img in os.listdir(query_path):
    queryImages.append(query_path+img)

queryImages = natsort.natsorted(queryImages)
print(queryImages)
```

먼저 code1 에서 저장한 visual vocabulary 정보를 가진 xml file 을 다시 load 한다.

queryImage 를 이름순으로 path를 불러 온다.

```
orb = cv2.ORB_create(
    nfeatures=2000,
    scaleFactor=1.4,
    nlevels=8,
    edgeThreshold=31,
    firstLevel=0,
    WTA_K=2,
    scoreType=cv2.ORB_HARRIS_SCORE,
    patchSize=31,
    fastThreshold=20,
)

FLANN_INDEX_KDTREE = 0
index_params = dict(algorithm = FLANN_INDEX_KDTREE, trees = 1)
```

```
search_params = dict(checks = 60)
flann = cv2.FlannBasedMatcher(index_params, search_params)
```

위와 같이 feature 추출하는 것은 ORB 로, 추출된 feature descriptor 를 매칭하는 것은 FLANNBasedMatcher 로 지정해 준다.

```
# clustering function
def cluster(des, voca):
    matches = flann.knnMatch(des, voca, k=1)
    clusterAssignment = np.zeros((1, voca.shape[0]), dtype=np.float32)

    for match in matches:
        clusterIndex = match[0].trainIdx
        clusterAssignment[0, clusterIndex] += 1

    return clusterAssignment
```

clustering 은 query image 와 book cover database image 에 각각 한 번씩 두 번 적용해야 하기 때문에 처음부터 따로 함수로 만들어 준다.

flannBasedMatcher 로 descriptor 매칭을 해 주고 clusterAssignment 를 얻는 과정이다. clusterAssignment 는 클러스터 할당 단계를 거친 후 그 결과를 나타내는 array 이다. 디스크립터(des) 와 visual vocabulary(voca) 의 descriptor matching 을 통해 생성된다. 두 값의 가장 가까운 클러스터 인덱스를 찾는다.(k=1 이므로 최근접 1개만 찾음)

for 문을 통해서는 각 match 의 train index 를 클러스터 인덱스로 사용한다. 해당하는 cluster index 는 clusterAssignment array 에서 해당하는 위치의 요소에 1을 더해 주는 역할을 한다. 따라서 return 값인 clusterAssignment array 는 각 cluster 에 속하는 descriptor 의 수를 의미한다.

따라서 return 값인 clusterAssignment 는 input descriptor 가 각 cluster 에 어떻게 할당되었는지를 나타내는 벡터이다.

```
for queryIndex, queryImagePath in enumerate(queryImages):
    queryImageName = queryImagePath.split("/")[-1]
    queryImage = cv2.imread(queryImagePath)

    if queryImage is None:
        print("Failed to read query image:", queryImagePath)
        continue

    _, query_descriptor = orb.detectAndCompute(queryImage, None)
    query_descriptor = np.float32(query_descriptor)

    query_clustter_assignment = cluster(query_descriptor, visualVocabulary)

    best_match_score = np.finfo(np.float64).max
    best_match_img = ""
```

```
    for bookIndex in range(1,59):
        book_path = dirdir + 'book' + str(bookIndex) + '.jpg'
        book_name = f'book{bookIndex}.jpg'
        book_img = cv2.imread(book_path)

        if book_img is None:
            print(f"book image 를 찾지 못했습니다: {book_path}")
            continue

        _, book_descriptor = orb.detectAndCompute(book_img, None)
        book_descriptor = np.float32(book_descriptor)

        book_cluster_assignment = cluster(book_descriptor, visualVocabulary)

        # calculate match score (origin book - query)
        match_score = cv2.compareHist(query_clustter_assignment, book_cluster_assignme
nt, cv2.HISTCMP_CHISQR)

        # Update the best match
        if match_score < best_match_score:
            best_match_score = match_score
            best_match_img = book_img
```

최종적으로 이미지 검색을 수행하고 매칭하는 부분은 위와 같다. query image 에서 orb 로 descriptor 를 추출하고, 이 값을 visualVocabulary 와 함께 clustering 함수의 inptut 으로 넣어 클러스터 할당 결과인 query_clustter_assignment 를 얻을 수 있다.

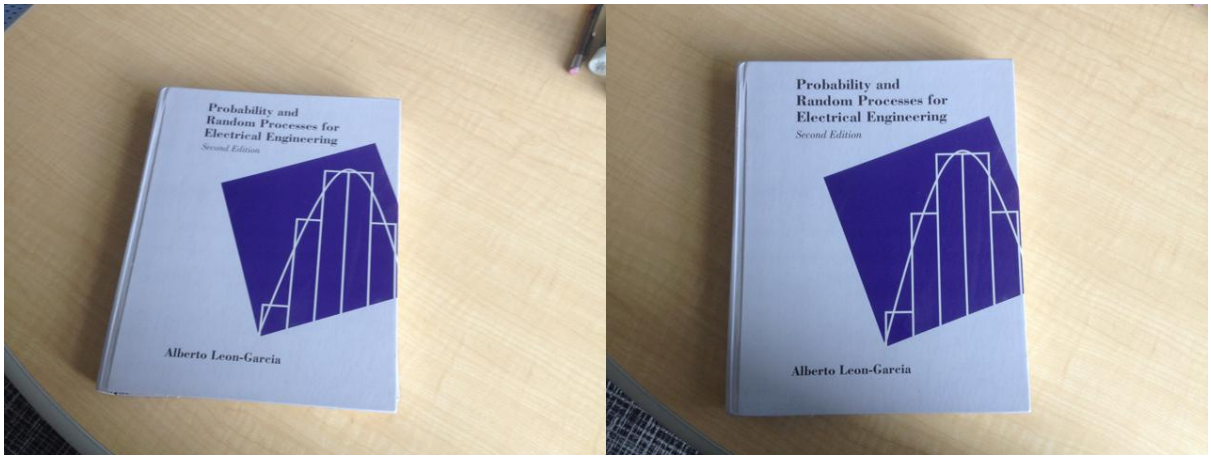
book cover database image 들에 대해서도 동일 과정을 진행한다.

이후 query_clustter_assignment 와 book_clustter_assignment 를 통해 query image 에 대해 가장 높은 match score 를 가지는 것이 book cover image 중 무엇인지 판별하는 과정을 거친다.

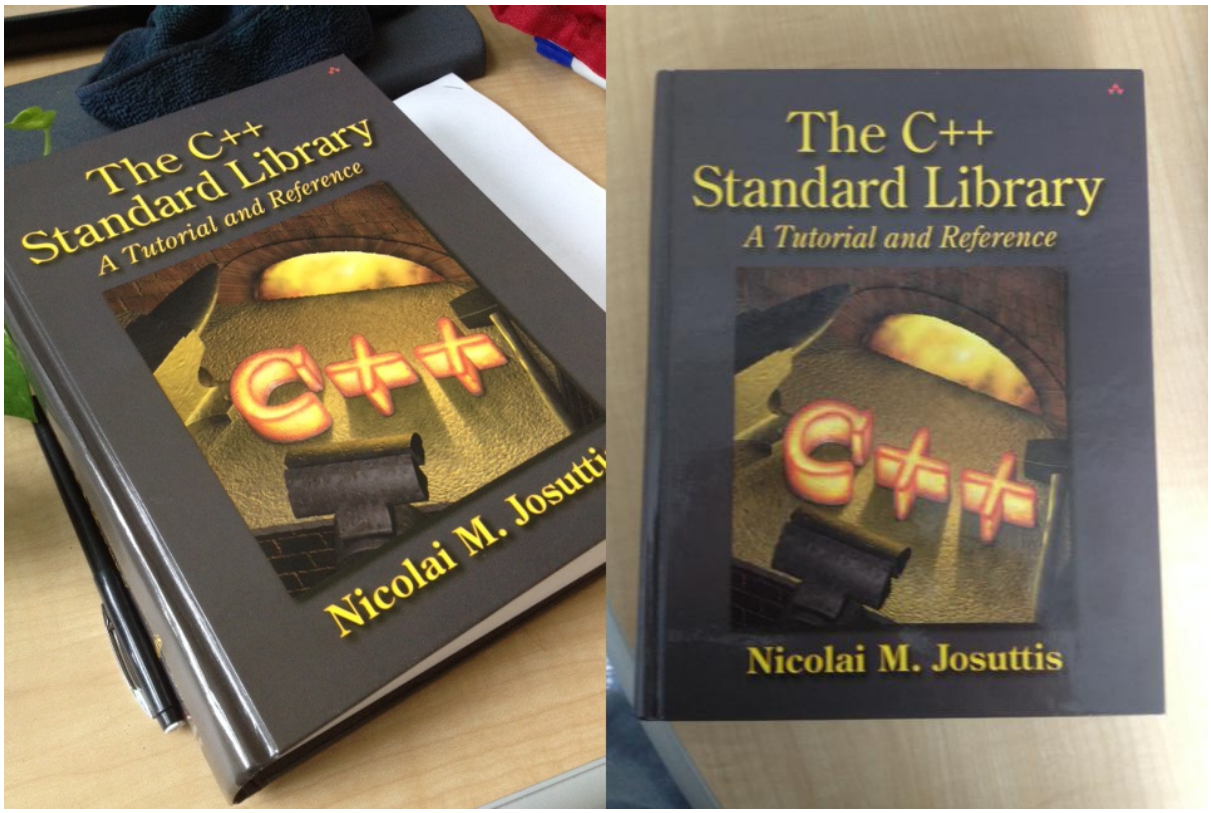
```
    # show query img matches
    cv2.imshow(f'query{queryIndex}', queryImage)
    save_q = save_dir+f'querty{queryIndex}.jpg'
    cv2.imwrite(save_q, queryImage)
    cv2.waitKey(0)
    cv2.imshow(f'best image {queryIndex}', best_match_img)
    save_b = save_dir+f'best image {queryIndex}.jpg'
    cv2.imwrite(save_b, best_match_img)
    cv2.waitKey(0)
    result = cv2.hconcat([queryImage, best_match_img])
    cv2.imshow(f'matching result {queryIndex}', result)
    save_match = save_dir+f'matching result {queryIndex}.jpg'
    cv2.imwrite(save_match, result)
    cv2.waitKey(0)
```

결과를 imshow 로 display 하고 저장하는 부분은 위와 같다. 저장된 결과 이미지는 아래 (problem 1 results) 에 첨부하였다.

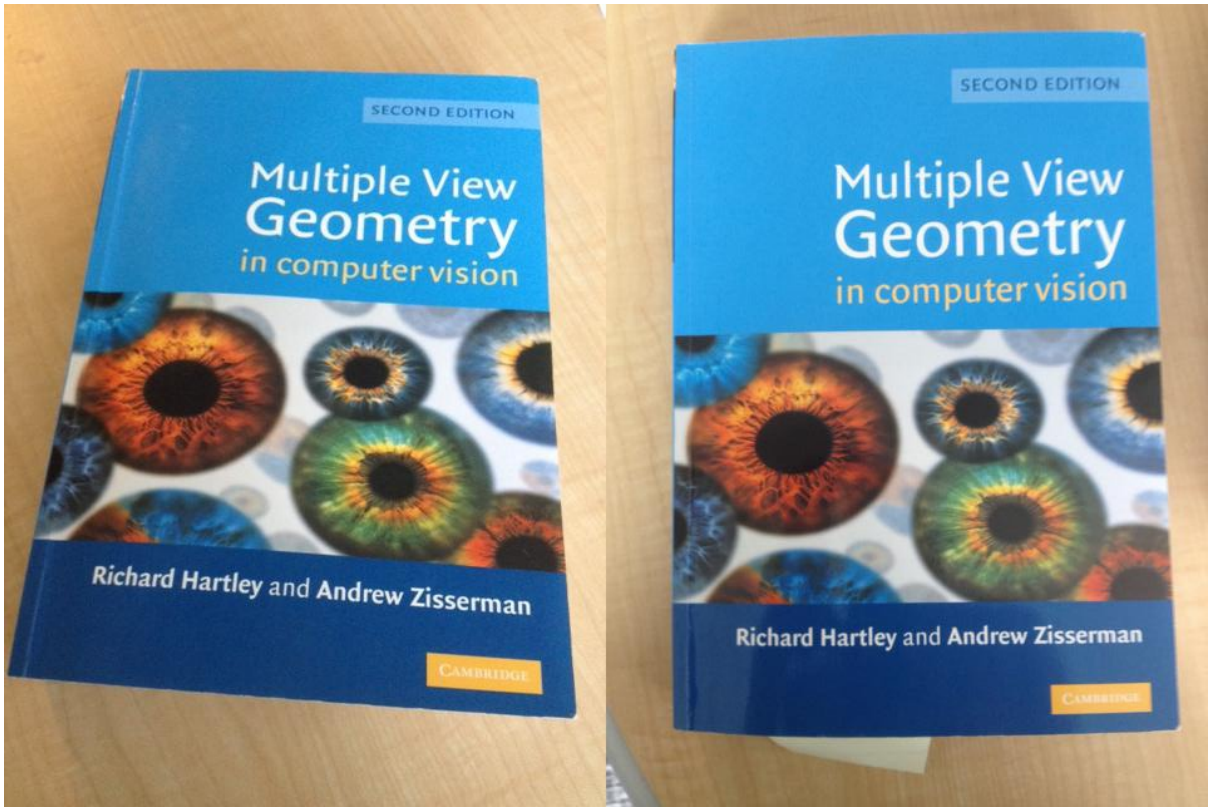
problem 1 results



(위) matching result 0 (좌 : query image, 우 : best match image)



(위) matching result 1 (좌 : query image, 우 : best match image)



(위) matching result 2 (좌 : query image, 우 : best match image)

두 코드(code1, code2)를 순차적으로 실행시킴을 통해 위와 같이 올바른 매칭 결과를 얻을 수 있었다.

---

problem 2

Compare a filtering approach and a smoothing method for visual-inertial fusion.
MSCKF (Geneva et al., ”Openvins: A research platform for visualinertial estimation,” IEEE International Conference on Robotics and Automation (ICRA), 2020. https://github.com/rpng/open vins) and VINS-MONO (Qin et al, ”Vins-mono: A robust and versatile monocular visual-inertial state estimator,” IEEE Transactions on Robotics (T-RO), 2018. https://github.com/HKUST-Aerial-Robotics/VINS-Fusion) are the most widely used filtering and smoothing algorithms, respectively.
Compare them in terms of front-end and back-end. (Hint: See lecture note 14. Read both papers and READMEs. Watch their results on YouTube.)

Answer of problem 2

MSCKF 란 IMU 와 Camera 센서를 사용하여 고정된 feature 에 대해서 measurement update 를 수행하여 odometry 를 제공하는 알고리즘이다.

MSCKF 의 기본 알고리즘은 다음과 같다.

1. IMU message 를 통한 propagation
2. Camera message 를 받아 Camera State 추가
3. Camera message 에서 새로운 feature 혹은 기존에 인식한 feature 확인
4. 기존에 계속 측정이 되었으나, 더 이상은 측정되지 않는 feature 에 대해 measurement update
5. camera state 의 개수가 일정 값 이상이면 measurement update

MSCKF 의 Front End 에서는 vision sensor 에서 제공하는 이미지 데이터를 처리하기 위해 특징점 추출 및 tracking 을 이용한다. 이전 프레임과 현재 프레임에서 추출 및 tracking 된 feature 들을 비교해 매칭 특징점을 찾고, 이를 통해 tracking 된 특징점들의 위치 정보를 업데이트 한다. MSCKF 에서는 재검출 과정을 통해 현재 프레임에서 새로운 특징점을 검출하기도 한다. 이는 tracking 과정에서 누락된 특징점도 활용하기 위함이다.

MSCKF 의 Back End 는 Front End 과정에서 얻은 이미지 특징점의 위치 및 이동 정보를 전달받는다. 칼만 필터를 기반으로 하며, 시간에 따라 system state 를 추정 및 업데이트한다. 이 알고리즘은 global 최적화를 이용해 센서 데이터와 제약 조건을 동시에 고려하여 최상의 추정 결과를 업데이트한다.

VINS-MONO 란 monocular cam 과 imu 를 결합한 것이다.

VINS-MONO 의 기본 알고리즘은 다음과 같다.

1. measurement processing : 이미지로부터 feature 를 추출하고 tracking 하면서 연속적인 두 이미지 사이에 imu measurements 를 preintegrated 한다.
2. measurement 과정이 끝나면, 초기화 과정을 거친다. 이때, 추후 nonlinear optimization 을 할 때 필요한 pose, velocity, gravity vector, gyroscope bias, 3d 특징점의 위치 등을 구하게 된다.
3. 초기화로 구한 값을 통해 relocalization module 이 있는 VIO module 은 preintegrated 된 IMU 측정값과, feature observations 를 tightly coupled 방법으로 fusion 을 진행한다.

VINS-MONO 의 Front End 에서는 새로운 이미지가 들어오면, KLT sparse optical flow 알고리즘을 수행한다. feature 를 찾을 때는 opencv 로 제공되는 GoodFeaturesToTrack 함수를 사용한다. (사이-토마시 알고리즘) 한 이미지당 feature 개수는 100~300 개 정도이다. RANSAC 알고리즘을 이용해 outlier 제거한 후, 특징이 추출된 이미지를 unit sphere 에 투영한다. 이 과정에서 keyFrame 선별을 하게 된다.

VINS-MONO 의 Back End 는 최적화를 기반으로 하여, 추정한 상태를 계속해서 업데이트 한다. 이 알고리즘은 비선형 최적화 문제를 해결하며, VI 결합을 통해 상태를 추정할 수 있다.

MSCKF는 필터링(Filtering)과 스무딩(Smoothing)을 동시에 수행한다. 필터링은 현재 시점에서 이용 가능한 모든 정보를 사용하여 상태 추정을 수행하고, 스무딩을 통해 과거 정보를 활용하여 추정 결과를 개선한다. VINS-Mono는 필터링과 스무딩을 분리하여 수행한다. 필터링을 통해 IMU와 카메라 이미지를 사용하여 단기적인 추정을 수행하고, 스무딩을 통해 과거 데이터를 활용하여 전반적인 추정 결과를 개선한다. MSCKF는 필터링과 스무딩을 동시에 수행하므로 상대적으로 낮은 지연 속도를 보인다. VINS-Mono는 필터링과 스무딩을 분리하여 수행하므로 스무딩 단계에서는 과거 데이터를 사용하기 때문에 상대적으로 지연이 커질 수 있다.

참고 자료

https://bkshin.tistory.com/entry/OpenCV-28-특징-매칭Feature-Matching

https://flower0.tistory.com/103

https://docs.opencv.org/3.4/d1/d5c/tutorial_py_kmeans_opencv.html

https://leechamin.tistory.com/379

https://needjarvis.tistory.com/140

https://needjarvis.tistory.com/719

https://076923.github.io/posts/AI-3/

https://gisbi-kim.github.io/blog/2021/04/27/msckf-history.html

https://wsstudynote.tistory.com/32