VIT-LSTM : CLASSICAL VO INTRO

최병찬 (석사 4기 / 지능통신시스템 연구실)

목 차

Reference Materials

- Visual Odometry Tutorial – Davide Scaramuzza, Friedrich Fraundorfer

  (IEEE Robotics & Automation Magazine / https://ieeexplore.ieee.org/document/6096039)
- Pose from Epioplar Geometry – Thomas Opsahl (University of Oslo)

  (https://www.uio.no/studier/emner/matnat/its/nedlagte-emner/UNIK4690/v16/forelesninger/lecture_7_3-pose-from-epipolar-geometry.pdf)
- Introduction to Machine Vision (ENB339) – Peter Corke (Queensland University of Technology)

  (https://www.youtube.com/watch?v=N_a6IP6KUSE&list=PL-cZT00o1h6JGiQcrlNFGi2aVdWzH6ios)
- QUT Robot Academy

  (https://robotacademy.net.au/)
- 컴퓨터 비전 특강 – 2020 컴퓨터공학과 대학원 수업 (문영식 교수님)
- Optical Flow in OpenCV (C++/Python) (https://learnopencv.com/optical-flow-in-opencv/)
- Depth Estimation : Basics and Intuition (https://towardsdatascience.com/depth-estimation-1-basics-and-intuition-86f2c9538cd1)
- Monocular Visual Odometry using OpenCV – Avi Singh (https://avisingh599.github.io/vision/monocular-vo/)

OPTICAL FLOW OPENCV IMPLEMENTATION

OPTICAL FLOW OPENCV IMPLEMENTATION

[ORB Feature Extraction + Lucas-Kanade Optical Flow]

- ORB Feature Extraction
  - FAST 알고리즘과 BRIEF를 섞어서 사용하는 알고리즘 (Orientation은 FAST / Rotation을 BRIEF 산출)
  - cv.ORB를 통해 ORB Feature Extractor 객체를 생성하고, detect함수와 compute함수를 통해 Feature Descriptor와 Feature Keypoint를 산출함
- Lucas-Kanade Optical Flow
  - Sparse / Indirect / Feature-based Method
  - 연속된 이미지 전체 Pixel 중 Feature Keypoint에 대해서만 Search Window Size 내에서 이동 변화(Flow) 추적하는 방식
  - Scale Invariance와 Robustness를 위해 Image Pyramid를 사용함
  - Grayscale 이미지에서의 Feature Keypoint를 요구함
  - OpenCV cv.calcOpticalFlowPyrLK API 사용

OpenCV Documentation – ORB Feature Extractor

ORB OpenCV Documentation : https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html

Class implementing the ORB (*oriented BRIEF*) keypoint detector and descriptor extractor. More...

```
#include <opencv2/features2d.hpp>
```

Inheritance diagram for cv::ORB:

cv::Algorithm — Feature2D의 부모 Class

cv::Feature2D — ORB의 부모 Class

cv::ORB — ORB는 Feature2D와 Algorithm에 대해 member 함수와 변수를 상속받음

Public Types

```
enum ScoreType {
  HARRIS_SCORE =0,
  FAST_SCORE =1
}
```

Public Member Functions

ORB Class가 가지는 member 함수

virtual String	getDefaultName () const CV_OVERRIDE
virtual int	getEdgeThreshold () const =0
virtual int	getFastThreshold () const =0
virtual int	getFirstLevel () const =0
virtual int	getMaxFeatures () const =0
virtual int	getNLevels () const =0
virtual int	getPatchSize () const =0
virtual double	getScaleFactor () const =0
virtual ORB::ScoreType	getScoreType () const =0
virtual int	getWTA_K () const =0
virtual void	setEdgeThreshold (int edgeThreshold)=0
virtual void	setFastThreshold (int fastThreshold)=0
virtual void	setFirstLevel (int firstLevel)=0
virtual void	setMaxFeatures (int maxFeatures)=0
virtual void	setNLevels (int nlevels)=0
virtual void	setPatchSize (int patchSize)=0
virtual void	setScaleFactor (double scaleFactor)=0
virtual void	setScoreType (ORB::ScoreType scoreType)=0
virtual void	setWTA_K (int wta_k)=0

▸ Public Member Functions inherited from cv::Feature2D

▸ Public Member Functions inherited from cv::Algorithm

cv::Feature2D Class Reference

2D Features Framework » Feature Detection and Description

Public Member Functions

virtual	~Feature2D ()
virtual void	compute (InputArray image, std::vector< KeyPoint > &keypoints, OutputArray descriptors) Computes the descriptors for a set of keypoints detected in an image (first variant) or image set (second variant). More...
virtual void	compute (InputArrayOfArrays images, std::vector< std::vector< KeyPoint > > &keypoints, OutputArrayOfArrays descriptors)
virtual int	defaultNorm () const
virtual int	descriptorSize () const
virtual int	descriptorType () const
virtual void	detect (InputArray image, std::vector< KeyPoint > &keypoints, InputArray mask=noArray()) Detects keypoints in an image (first variant) or image set (second variant). More...
virtual void	detect (InputArrayOfArrays images, std::vector< std::vector< KeyPoint > > &keypoints, InputArrayOfArrays masks=noArray())
virtual void	detectAndCompute (InputArray image, InputArray mask, std::vector< KeyPoint > &keypoints, OutputArray descriptors, bool useProvidedKeypoints=false)
virtual bool	empty () const CV_OVERRIDE Return true if detector object is empty. More...
virtual String	getDefaultName () const CV_OVERRIDE
void	read (const String &fileName)
virtual void	read (const FileNode &) CV_OVERRIDE Reads algorithm parameters from a file storage. More...
void	write (const String &fileName) const
virtual void	write (FileStorage &) const CV_OVERRIDE Stores algorithm parameters in a file storage. More...
void	write (const Ptr< FileStorage > &fs, const String &name=String()) const

▸ Public Member Functions inherited from cv::Algorithm

Additional Inherited Members

▸ Static Public Member Functions inherited from cv::Algorithm

▸ Protected Member Functions inherited from cv::Algorithm

OpenCV Documentation – ORB Feature Extractor

ORB OpenCV Documentation : https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html

◆ create() **ORB Feature Extractor 생성자 함수**

```
static Ptr<ORB> cv::ORB::create ( int              nfeatures = 500 ,
                                  float            scaleFactor = 1.2f ,
                                  int              nlevels = 8 ,
                                  int              edgeThreshold = 31 ,
                                  int              firstLevel = 0 ,
                                  int              WTA_K = 2 ,
                                  ORB::ScoreType   scoreType = ORB::HARRIS_SCORE ,
                                  int              patchSize = 31 ,
                                  int              fastThreshold = 20
                                )
```

static

C/C++에서의 함수 사용 버전

동일 기능에 대해 C/C++ 함수 버전과 Python 함수 버전이 서로 다를 수도 있음

Python:

```
cv.ORB_create( [, nfeatures[, scaleFactor[, nlevels[, edgeThreshold[, firstLevel[, WTA_K[, scoreType[, patchSize[, fastThreshold]]]]]]]]] ) -> retval
```

Python에서의 함수 사용 버전

The ORB constructor.

Parameters

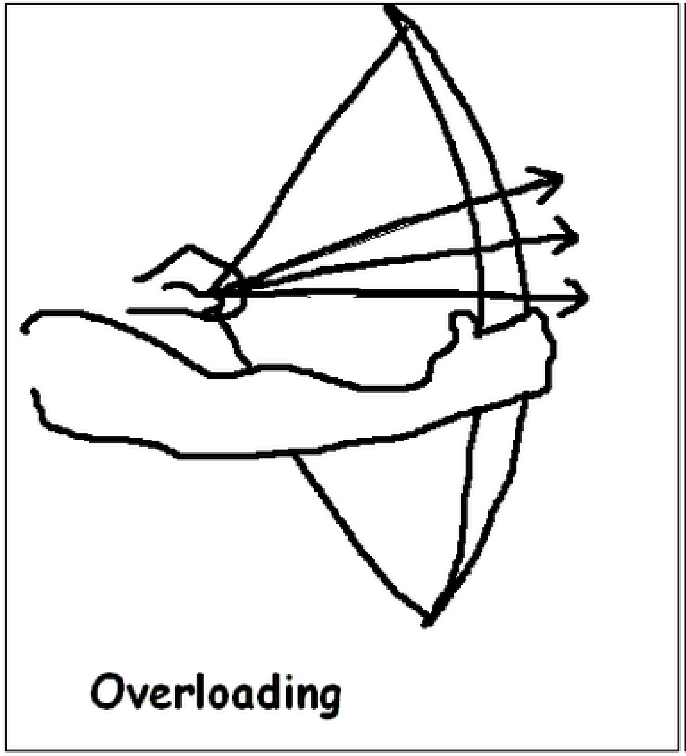
nfeatures	The maximum number of features to retain.
scaleFactor	Pyramid decimation ratio, greater than 1. scaleFactor==2 means the classical pyramid, where each next level has 4x less pixels than the previous, but such a big scale factor will degrade feature matching scores dramatically. On the other hand, too close to 1 scale factor will mean that to cover certain scale range you will need more pyramid levels and so the speed will suffer.
nlevels	The number of pyramid levels. The smallest level will have linear size equal to input_image_linear_size/pow(scaleFactor, nlevels - firstLevel).
edgeThreshold	This is size of the border where the features are not detected. It should roughly match the patchSize parameter.
firstLevel	The level of pyramid to put source image to. Previous layers are filled with upscaled source image.
WTA_K	The number of points that produce each element of the oriented BRIEF descriptor. The default value 2 means the BRIEF where we take a random point pair and compare their brightnesses, so we get 0/1 response. Other possible values are 3 and 4. For example, 3 means that we take 3 random points (of course, those point coordinates are random, but they are generated from the pre-defined seed, so each element of BRIEF descriptor is computed deterministically from the pixel rectangle), find point of maximum brightness and output index of the winner (0, 1 or 2). Such output will occupy 2 bits, and therefore it will need a special variant of Hamming distance, denoted as NORM_HAMMING2 (2 bits per bin). When WTA_K=4, we take 4 random points to compute each bin (that will also occupy 2 bits with possible values 0, 1, 2 or 3).
scoreType	The default HARRIS_SCORE means that Harris algorithm is used to rank features (the score is written to KeyPoint::score and is used to retain best nfeatures features); FAST_SCORE is alternative value of the parameter that produces slightly less stable keypoints, but it is a little faster to compute.
patchSize	size of the patch used by the oriented BRIEF descriptor. Of course, on smaller pyramid layers the perceived image area covered by a feature will be larger.
fastThreshold	the fast threshold

함수의 입력 Argument에 대한 설명

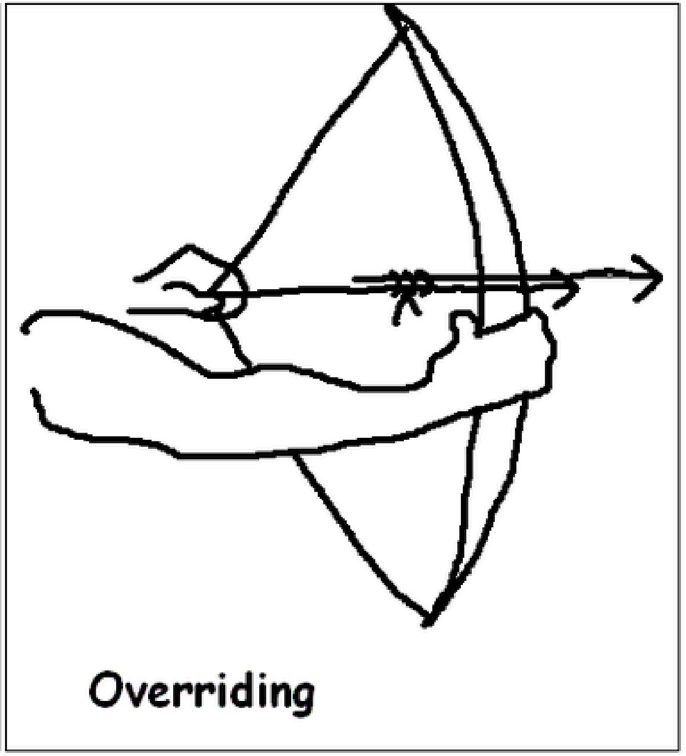
OpenCV Documentation – ORB Feature Extractor

ORB OpenCV Documentation : https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html

Feature2D Class OpenCV Documentation : https://docs.opencv.org/4.x/d0/d13/classcv_1_1Feature2D.html



특정 이름의 함수에 대해
입력/출력 Argument 조합에 따른
여러 버전을 만듦



기존에 있는 함수의 형태를 유지하되
기존의 기능을 다르게 재정의함

OpenCV Documentation – ORB Feature Extractor

ORB OpenCV Documentation : https://docs.opencv.org/4.x/db/d95/classcv_1_1ORB.html

Feature2D Class OpenCV Documentation : https://docs.opencv.org/4.x/d0/d13/classcv_1_1Feature2D.html

◆ detect() [1/2] ORB Keypoint 연산 함수 – Overloading 버전 1

```
virtual void cv::Feature2D::detect ( InputArray                 image,
                                     std::vector< KeyPoint > &  keypoints,
                                     InputArray                 mask = noArray()
                                   )
```
virtual

Python:

cv.Feature2D.detect(image[, mask]) -> keypoints

cv.Feature2D.detect(images[, masks]) -> keypoints

Detects keypoints in an image (first variant) or image set (second variant).

Parameters

- **image** Image.
- **keypoints** The detected keypoints. In the second variant of the method keypoints[i] is a set of keypoints detected in images[i] .
- **mask** Mask specifying where to look for keypoints (optional). It must be a 8-bit integer matrix with non-zero values in the region of interest.

◆ detect() [2/2] ORB Keypoint 연산 함수 – Overloading 버전 2

```
virtual void cv::Feature2D::detect ( InputArrayOfArrays                         images,
                                     std::vector< std::vector< KeyPoint > > &   keypoints,
                                     InputArrayOfArrays                         masks = noArray()
                                   )
```
virtual

Python:

cv.Feature2D.detect(image[, mask]) -> keypoints

cv.Feature2D.detect(images[, masks]) -> keypoints

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **images** Image set.
- **keypoints** The detected keypoints. In the second variant of the method keypoints[i] is a set of keypoints detected in images[i] .
- **masks** Masks for each input image specifying where to look for keypoints (optional). masks[i] is a mask for images[i].

◆ compute() [1/2] ORB Descriptor 연산 함수 – Overloading 버전 1

```
virtual void cv::Feature2D::compute ( InputArray                 image,
                                      std::vector< KeyPoint > &  keypoints,
                                      OutputArray                descriptors
                                    )
```
virtual

Python:

cv.Feature2D.compute(image, keypoints[, descriptors]) -> keypoints, descriptors

cv.Feature2D.compute(images, keypoints[, descriptors]) -> keypoints, descriptors

Computes the descriptors for a set of keypoints detected in an image (first variant) or image set (second variant).

Parameters

- **image** Image.
- **keypoints** Input collection of keypoints. Keypoints for which a descriptor cannot be computed are removed. Sometimes new keypoints can be added, for example: SIFT duplicates keypoint with several dominant orientations (for each orientation).
- **descriptors** Computed descriptors. In the second variant of the method descriptors[i] are descriptors computed for a keypoints[i]. Row j is the keypoints (or keypoints[i]) is the descriptor for keypoint j-th keypoint.

Reimplemented in **cv::xfeatures2d::DAISY**.

◆ compute() [2/2] ORB Descriptor 연산 함수 – Overloading 버전 2

```
virtual void cv::Feature2D::compute ( InputArrayOfArrays                         images,
                                      std::vector< std::vector< KeyPoint > > &   keypoints,
                                      OutputArrayOfArrays                        descriptors
                                    )
```
virtual

Python:

cv.Feature2D.compute(image, keypoints[, descriptors]) -> keypoints, descriptors

cv.Feature2D.compute(images, keypoints[, descriptors]) -> keypoints, descriptors

This is an overloaded member function, provided for convenience. It differs from the above function only in what argument(s) it accepts.

Parameters

- **images** Image set.
- **keypoints** Input collection of keypoints. Keypoints for which a descriptor cannot be computed are removed. Sometimes new keypoints can be added, for example: SIFT duplicates keypoint with several dominant orientations (for each orientation).
- **descriptors** Computed descriptors. In the second variant of the method descriptors[i] are descriptors computed for a keypoints[i]. Row j is the keypoints (or keypoints[i]) is the descriptor for keypoint j-th keypoint.

Reimplemented in **cv::xfeatures2d::DAISY**.

OpenCV Documentation – ORB Feature Extractor

Basic Optical Flow Documentation : https://docs.opencv.org/3.4/dc/d6b/group__video__track.html

Classes

class	cv::DenseOpticalFlow
class	cv::DualTVL1OpticalFlow "Dual TV L1" Optical Flow **Algorithm**. More...
class	cv::FarnebackOpticalFlow Class computing a dense optical flow using the Gunnar Farneback's algorithm. More...
class	cv::KalmanFilter Kalman filter class. More...
class	cv::SparseOpticalFlow Base interface for sparse optical flow algorithms. More...
class	cv::SparsePyrLKOpticalFlow Class used for calculating a sparse optical flow. More...

OpenCV의 특정 영역에 소속된 Class

Enumerations

```
enum {
  cv::OPTFLOW_USE_INITIAL_FLOW = 4,
  cv::OPTFLOW_LK_GET_MIN_EIGENVALS = 8,
  cv::OPTFLOW_FARNEBACK_GAUSSIAN = 256
}
enum {
  cv::MOTION_TRANSLATION = 0,
  cv::MOTION_EUCLIDEAN = 1,
  cv::MOTION_AFFINE = 2,
  cv::MOTION_HOMOGRAPHY = 3
}
```

"열거형"

- 라이브러리의 Global Parameter 상수
- 상수 이름 그대로 사용 가능함
- C/C++에서 #define으로 정의한 상수와 유사함

Functions

int	cv::buildOpticalFlowPyramid (InputArray img, OutputArrayOfArrays pyramid, Size winSize, int maxLevel, bool withDerivatives=true, int pyrBorder=BORDER_REFLECT_101, int derivBorder=BORDER_CONSTANT, bool tryReuseInputImage=true) Constructs the image pyramid which can be passed to calcOpticalFlowPyrLK. More...
void	cv::calcOpticalFlowFarneback (InputArray prev, InputArray next, InputOutputArray flow, double pyr_scale, int levels, int winsize, int iterations, int poly_n, double poly_sigma, int flags) Computes a dense optical flow using the Gunnar Farneback's algorithm. More...
void	cv::calcOpticalFlowPyrLK (InputArray prevImg, InputArray nextImg, InputArray prevPts, InputOutputArray nextPts, OutputArray status, OutputArray err, Size winSize=Size(21, 21), int maxLevel=3, TermCriteria criteria=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, 0.01), int flags=0, double minEigThreshold=1e-4) Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids. More...
RotatedRect	cv::CamShift (InputArray probImage, Rect &window, TermCriteria criteria) Finds an object center, size, and orientation. More...
double	cv::computeECC (InputArray templateImage, InputArray inputImage, InputArray inputMask=noArray()) Computes the Enhanced Correlation Coefficient value between two images [64] . More...
Ptr< DualTVL1OpticalFlow >	cv::createOptFlow_DualTVL1 () Creates instance of cv::DenseOpticalFlow. More...
Mat	cv::estimateRigidTransform (InputArray src, InputArray dst, bool fullAffine) Computes an optimal affine transformation between two 2D point sets. More...
Mat	cv::estimateRigidTransform (InputArray src, InputArray dst, bool fullAffine, int ransacMaxIters, double ransacGoodRatio, int ransacSize0)
double	cv::findTransformECC (InputArray templateImage, InputArray inputImage, InputOutputArray warpMatrix, int motionType, TermCriteria criteria, InputArray inputMask, int gaussFiltSize) Finds the geometric transform (warp) between two images in terms of the ECC criterion [64] . More...
double	cv::findTransformECC (InputArray templateImage, InputArray inputImage, InputOutputArray warpMatrix, int motionType=MOTION_AFFINE, TermCriteria criteria=TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 50, 0.001), InputArray inputMask=noArray())
int	cv::meanShift (InputArray probImage, Rect &window, TermCriteria criteria) Finds an object on a back projection image. More...

라이브러리에서 사용할 수 있는 함수 종류

OpenCV Documentation – ORB Feature Extractor

Basic Optical Flow Documentation : https://docs.opencv.org/3.4/dc/d6b/group__video__track.html

◆ calcOpticalFlowPyrLK()

```
void cv::calcOpticalFlowPyrLK ( InputArray        prevImg,
                                InputArray        nextImg,
                                InputArray        prevPts,
                                InputOutputArray  nextPts,
                                OutputArray       status,
                                OutputArray       err,
                                Size              winSize = Size(21, 21) ,
                                int               maxLevel = 3 ,
                                TermCriteria      criteria = TermCriteria(TermCriteria::COUNT+TermCriteria::EPS, 30, 0.01) ,
                                int               flags = 0 ,
                                double            minEigThreshold = 1e-4
                              )
```

Python:

```
cv.calcOpticalFlowPyrLK( prevImg, nextImg, prevPts, nextPts[, status[, err[, winSize[, maxLevel[, criteria[, flags[, minEigThreshold]]]]]]] ) -> nextPts, status, err
```

Lucas-Kanade Optical Flow 연산 특성을 결정하는 Parameter

- **winSize : Optical Flow를 감지할 Window 크기**
- **maxLevel : Image Pyramid 개수**
- **flags : Flow 연산 작동 특성 결정**
- **minEigThreshold : flags=1인 경우 Eigen Value 필터링 기준**

`#include <opencv2/video/tracking.hpp>`

Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.

Parameters

prevImg	first 8-bit input image or pyramid constructed by buildOpticalFlowPyramid.
nextImg	second input image or pyramid of the same size and the same type as prevImg.
prevPts	vector of 2D points for which the flow needs to be found; point coordinates must be single-precision floating-point numbers.
nextPts	output vector of 2D points (with single-precision floating-point coordinates) containing the calculated new positions of input features in the second image; when OPTFLOW_USE_INITIAL_FLOW flag is passed, the vector must have the same size as in the input.
status	output status vector (of unsigned chars); each element of the vector is set to 1 if the flow for the corresponding features has been found, otherwise, it is set to 0.
err	output vector of errors; each element of the vector is set to an error for the corresponding feature, type of the error measure can be set in flags parameter; if the flow wasn't found then the error is not defined (use the status parameter to find such cases).
winSize	size of the search window at each pyramid level.
maxLevel	0-based maximal pyramid level number; if set to 0, pyramids are not used (single level), if set to 1, two levels are used, and so on; if pyramids are passed to input then algorithm will use as many levels as pyramids have but no more than maxLevel.
criteria	parameter, specifying the termination criteria of the iterative search algorithm (after the specified maximum number of iterations criteria.maxCount or when the search window moves by less than criteria.epsilon.
flags	operation flags: • **OPTFLOW_USE_INITIAL_FLOW** uses initial estimations, stored in nextPts; if the flag is not set, then prevPts is copied to nextPts and is considered the initial estimate. • **OPTFLOW_LK_GET_MIN_EIGENVALS** use minimum eigen values as an error measure (see minEigThreshold description); if the flag is not set, then L1 distance between patches around the original and a moved point, divided by number of pixels in a window, is used as a error measure.
minEigThreshold	the algorithm calculates the minimum eigen value of a 2x2 normal matrix of optical flow equations (this matrix is called a spatial gradient matrix in [30]), divided by number of pixels in a window; if this value is less than minEigThreshold, then a corresponding feature is filtered out and its flow is not processed, so it allows to remove bad points and get a performance boost.

The function implements a sparse iterative version of the Lucas-Kanade optical flow in pyramids. See [30] . The function is parallelized with the TBB library.

Note

- An example using the Lucas-Kanade optical flow algorithm can be found at opencv_source_code/samples/cpp/lkdemo.cpp
- (Python) An example using the Lucas-Kanade optical flow algorithm can be found at opencv_source_code/samples/python/lk_track.py
- (Python) An example using the Lucas-Kanade tracker for homography matching can be found at opencv_source_code/samples/python/lk_homography.py

Examples:

samples/cpp/lkdemo.cpp.

OPTICAL FLOW OPENCV IMPLEMENTATION

[Example 1 – Slow Traffic (1/3)]

```
import numpy as np
import cv2 as cv

lk_params = dict( winSize  = (15,15),
                  maxLevel = 2,
                  criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 10, 0.03))

cap = cv.VideoCapture('./slow_traffic_small.mp4')

color = np.random.randint(0, 255, (2000, 3))

orb = cv.ORB_create(nfeatures=500)

ret, prev_frame = cap.read()
prev_gray = cv.cvtColor(prev_frame, cv.COLOR_BGR2GRAY)

prev_keypoints = orb.detect(prev_gray, None)
prev_keypoints, prev_descriptors = orb.compute(prev_gray, prev_keypoints)
prev_keypoints = np.array((prev_keypoints))

### Convert ORB keypoints into numpy format (single point precision / column keypoint vector) ###########
correct_keypoints = []
for i in range(len(prev_keypoints)):
    correct_keypoints.append([[np.float32(prev_keypoints[i].pt[0]), np.float32(prev_keypoints[i].pt[1])]])

np_prev_correct_keypoints = np.array(correct_keypoints)
#####################################################################################################

mask = np.zeros_like(prev_frame)
```

Optical Flow 작동 특성 Parameter

테스트 영상 Loading

최대 Feature 500개를 감지하는 ORB Feature Extractor 생성

ORB를 적용하기 위해 Grayscale 이미지로 전환함

ORB Keypoint 및 Descriptor 계산

Optical Flow를 그려내기 위한 더미 Mask 이미지

OPTICAL FLOW OPENCV IMPLEMENTATION

[Example 1 – Slow Traffic (2/3)]

```
while(1):

    ret, frame = cap.read()
    frame_gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

    current_keypoints, st, err = cv.calcOpticalFlowPyrLK(prev_gray, frame_gray, np_prev_correct_keypoints, None, **lk_params)

    good_new = current_keypoints[st==1]
    good_prev = np_prev_correct_keypoints[st==1]

    ### Feature Retracking : If the feature is below certain number, re-conduct feature extraction & tracking ####################
    print('Current Feature Num : ', len(good_new))
    if len(good_new) <= 400:

        print('[Re-Tracking] Current Feature Num : ', len(good_new))

        prev_keypoints = orb.detect(prev_gray, None)
        prev_keypoints, prev_descriptors = orb.compute(prev_gray, prev_keypoints)
        prev_keypoints = np.array((prev_keypoints))

        ### Convert ORB keypoints into numpy format (single point precision / column keypoint vector) #############################
        correct_keypoints = []
        for i in range(len(prev_keypoints)):
            correct_keypoints.append([[np.float32(prev_keypoints[i].pt[0]), np.float32(prev_keypoints[i].pt[1])]])

        np_prev_correct_keypoints = np.array(correct_keypoints)
        ##########################################################################################################################
        current_keypoints, st, err = cv.calcOpticalFlowPyrLK(prev_gray, frame_gray, np_prev_correct_keypoints, None, **lk_params)
        good_new = current_keypoints[st==1]
        good_prev = np_prev_correct_keypoints[st==1]
    ################################################################################################################################
```

새롭게 들어온 이미지 Grayscale 전환

이전 이미지의 Keypoint 중심으로
현재 이미지에서 Feature Correspondence를
Optical Flow로 연산함

status=1에 해당되는 Index를 찾아서 Feature Correspondence를 찾음

만약 Feature Correspondence 개수가
400개 이하인 경우
이전 이미지와 현재 이미지에 대해
Feature Extraction부터 다시 수행함

OPTICAL FLOW OPENCV IMPLEMENTATION

[Example 1 – Slow Traffic (3/3)]

```python
for i, (new, prev) in enumerate(zip(good_new, good_prev)):
    a, b = new.ravel()
    c, d = prev.ravel()
    mask = cv.line(mask, (a, b), (c, d), color[i].tolist(), 2)
    frame = cv.circle(frame, (a, b), 5, color[i].tolist(), -1)

img = cv.add(frame.copy(), mask)
```

Optical Flow로 추척되는 Feature Correspondence 경로를 그림

```python
cv.imshow('ORB-based Optical Flow + ReTracking', img)
k = cv.waitKey(30) & 0xff
if k == 27:
    break
```

최종 결과 이미지를 보여줌

```python
prev_gray = frame_gray.copy()
np_prev_correct_keypoints = good_new.reshape(-1, 1, 2)
```

현재 이미지와 Feature Keypoint를 이전 이미지와 Keypoint로 업데이트함

DATASET SETUP

DATASET SETUP

[KITTI Sensor & Groundtruth Setup]

Visual Odometry / SLAM Evaluation 2012

The odometry benchmark consists of 22 stereo sequences, saved in loss less png format: We provide 11 sequences (00-10) with ground truth trajectories for training and 11 sequences (11-21) without ground truth for evaluation. For this benchmark you may provide results using monocular or stereo visual odometry, laser-based SLAM or algorithms that combine visual and LIDAR information. The only restriction we impose is that your method is fully automatic (e.g., no manual loop-closure tagging is allowed) and that the same parameter set is used for all sequences. A development kit provides details about the data format.

- Download odometry data set (grayscale, 22 GB)
- Download odometry data set (color, 65 GB)
- Download odometry data set (velodyne laser data, 80 GB)
- Download odometry data set (calibration files, 1 MB)
- Download odometry ground truth poses (4 MB)
- Download odometry development kit (1 MB)
- Lee Clement and his group (University of Toronto) have written some python tools for loading and parsing the KITTI raw and odometry datasets

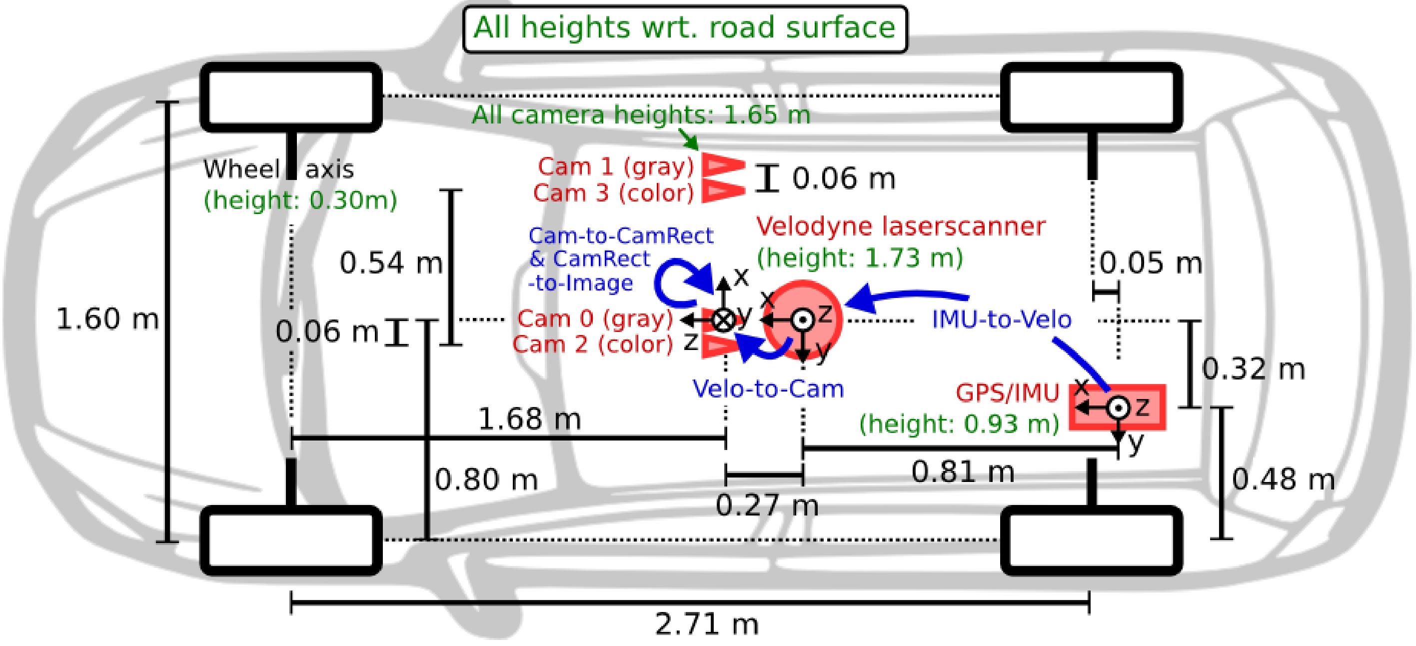
Devkit Readme에 Odometry 데이터셋의 구성에 대한 설명이 포함됨

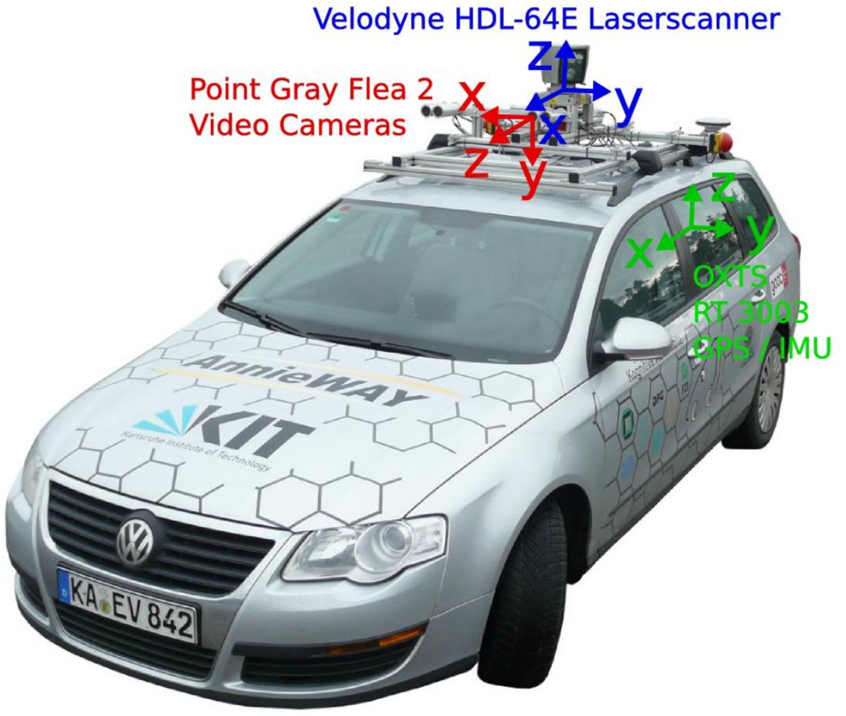
DATASET SETUP

[KITTI Sensor & Groundtruth Setup]

- 1 Inertial Navigation System (GPS/IMU): OXTS RT 3003
- 1 Laserscanner: Velodyne HDL-64E
- 2 Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3M-C)
- 2 Color cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3C-C)
- 4 Varifocal lenses, 4-8 mm: Edmund Optics NT59-917

The laser scanner spins at 10 frames per second, capturing approximately 100k points per cycle. The vertical resolution of the laser scanner is 64. The cameras are mounted approximately level with the ground plane. The camera images are cropped to a size of 1382 x 512 pixels using libdc's format 7 mode. After rectification, the images get slightly smaller. The cameras are triggered at 10 frames per second by the laser scanner (when facing forward) with shutter time adjusted dynamically (maximum shutter time: 2 ms). Our sensor setup with respect to the vehicle is illustrated in the following figure. Note that more information on calibration parameters is given in the calibration files and the development kit (see raw data section).





DATASET SETUP

[KITTI Sensor & Groundtruth Setup – KITTI Devkit Readme]

This file describes the KITTI visual odometry / SLAM benchmark package. Accurate ground truth (<10cm) is provided by a GPS/IMU system with RTK float/integer corrections enabled. In order to enable a fair comparison of all methods, only ground truth for the sequences 00-10 is made publicly available. The remaining sequences (11-21) serve as evaluation sequences.	▪ KITTI 데이터셋에 대한 간단한 소개 ▪ Groundtruth는 GPS/IMU 사용하여 6DOF IMU 값으로 생성되었으며, RTK를 사용하여 10cm 이내까지 정확도를 보정함 ▪ Sequence 00 ~ 10까지 Groundtruth가 제공되지만, 11 ~ 21은 평가를 위해 Groundtruth를 별도로 제공하지 않음
Folder 'sequences': Each folder within the folder 'sequences' contains a single sequence, where the left and right images are stored in the sub-folders image_0 and image_1, respectively. The images are provided as greyscale PNG images and can be loaded with MATLAB or libpng++. All images have been undistorted andrectified. Sequences 0-10 can be used for training, while results must be provided for the test sequences 11-21.	▪ 각 Sequence별 데이터셋 구성 소개 ▪ 전방 좌측 Grayscale 카메라는 image_0, 전방 우측 Grayscale 카메라는 image_1에 저장되어있음 ▪ 전방 좌측 Color 카메라는 image_2, 전방 우측 Color 카메라는 image_3에 저장되어있음

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – KITTI Devkit Readme]

Additionally we provide the velodyne point clouds for point-cloud-based methods. To save space, all scans have been stored as Nx4 float matrix into a binary file using the following code:

Here, data contains 4*num values, where the first 3 values correspond to x,y and z, and the last value is the reflectance information. All scan sare stored row-aligned, meaning that the first 4 values correspond to the first measurement. Since each scan might potentially have a different number of points, this must be determined from the file size when reading the file, where 1e6 is a good enough upper bound on the number of values:

x,y and y are stored in metric (m) Velodyne coordinates.

- 3D LiDAR Point Cloud 데이터셋 구성 소개
- 센서 수집 차량의 Velodyne 64 Channel 3D LiDAR 1개만 설치되어있기에 각 Sequence마다 1개의 LiDAR 데이터셋 폴더를 가짐
- 3D LiDAR Point Cloud는 binary 파일로 저장되어있기에 binary를 해제해서 사용해야함
- Binary 파일 1개는 한 순간 (1개 Frame)의 Point Cloud 모음

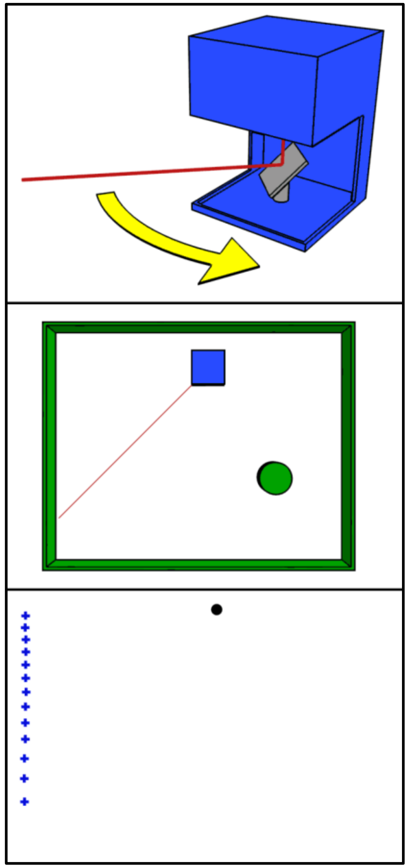
  (※ 매 순간마다 수집된 Point Cloud의 개수는 다름)
- Binary 해제하면 4개 값(X, Y, Z, Reflectance)으로 구성된 리스트 모음이 나옴
- X : LiDAR 기준으로 1개 Point Cloud의 X축 위치 / 단위 : m
- Y : LiDAR 기준으로 1개 Point Cloud의 Y축 위치 / 단위 : m
- Z : LiDAR 기준으로 1개 Point Cloud의 Z축 위치 / 단위 : m
- Reflectance : LiDAR 레이저 반사 강도 ≠ Point Cloud의 Depth 또는 거리값

Raw 3D LiDAR Point Cloud Numpy 예시 : $[\,[x_1, y_1, z_1, \text{reflectance}_1],\ [x_2, y_2, z_2, \text{reflectance}_2],\ [x_3, y_3, z_3, \text{reflectance}_3],\ \cdots\,]$

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – Understanding LiDAR Point Cloud]

▪ 2D LiDAR 기본 원리



https://en.wikipedia.org/wiki/Lidar

❖ 레이저 스캐너가 수평으로 360도 회전하면서 반사된 레이저의 이동 거리 시간을 수집하여 주변에 반사된 물체 또는 평면과의 거리를 종합함

❖ 데이터는 주로 $1 \times \frac{360}{\text{분해능}\ (\textit{Angular Resolution})}$ 구조의 배열이며,
배열 내에는 Intensity (반사율) 또는 거리값 (Time of Flight에 근거한 거리값)을 저장함

❖ 특징 : 2D LiDAR는 수평 회전만 하기 때문에 반사되는 레이저에 따른 거리 측정값은
레이저가 소실되지 않는 이상 (소실 시 Max or NaN로 처리) 회전 방향 순서에 맞춰서 쌓임.
('Ordered & Structured Data Structure')

0도 레이저 거리 측정결과	1도 레이저 거리 측정결과	2도 레이저 거리 측정결과	· · ·	358도 레이저 거리 측정결과	359도 레이저 거리 측정결과

(※ 예시 : 1회전 분해능 1도 → 1회전 1 x 360 거리 데이터 측정)

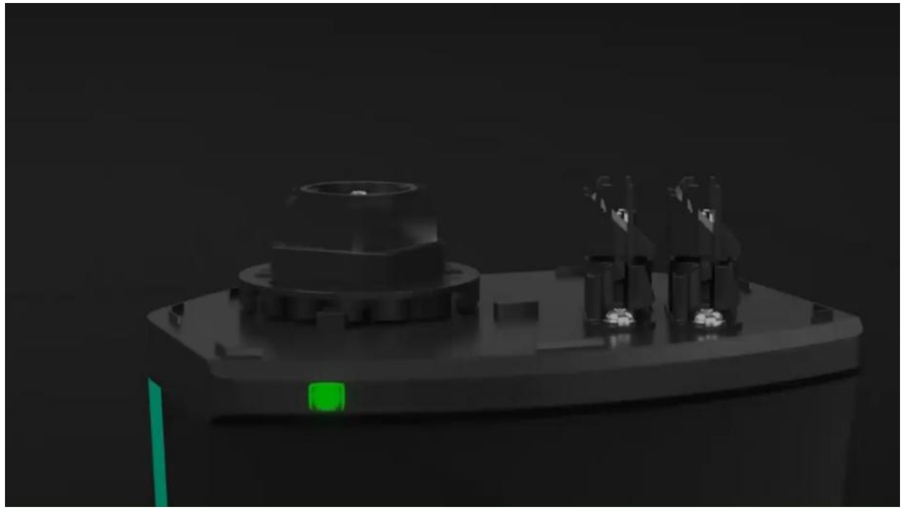
: 2D LiDAR는 회전 속도가 높아질수록 1회전에 획득할 수 있는 데이터의 개수가 감소함.

(∵ LiDAR가 빠르게 회전할수록 반사된 레이저를 감지하기 힘들기 때문에 분해능이 감소함)

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – Understanding LiDAR Point Cloud]

- **3D LiDAR 기본 원리**



3-D LiDAR Sensor | R2300 Multi-Layer Scanner | Overview
https://www.youtube.com/watch?v=-qQgvJ6FJdc

❖ 레이저 스캐너가 수평으로 360도 + 수직으로 N개의 채널로 레이저를 발사하여 반사된 레이저의 이동 거리 시간을 수집하여 주변에 반사된 물체 또는 평면과의 거리를 종합함

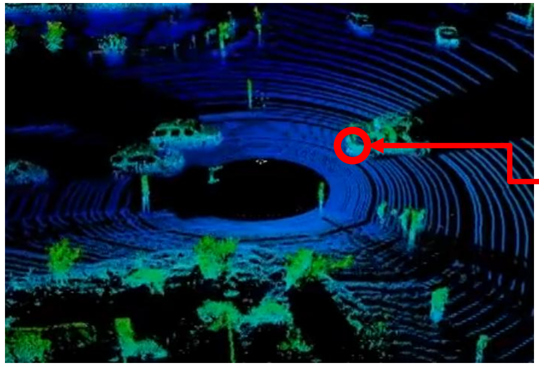
❖ 3D LiDAR의 경우 1회전 최대 인식 데이터량은 $1 \times \frac{360}{\text{분해능}\ (Angular\ Resolution)}$ x Channel 개수임.

(예시 : KITTI 데이터셋 – Velodyne HDL 64E : $1 \times \frac{360}{0.08} \times 64 = 288000$ → 1회전 최대 288000개 데이터 수집)

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – Understanding LiDAR Point Cloud]

- Raw 3D LiDAR Point Cloud Data Structure



https://velodynelidar.com/products/hdl-64e/

Point Cloud 점 1개

1차 Point Cloud 입력의 X좌표	2차 Point Cloud 입력의 X좌표	. . .	N-1차 Point Cloud 입력의 X좌표	N차 Point Cloud 입력의 X좌표
1차 Point Cloud 입력의 Y좌표	2차 Point Cloud 입력의 Y좌표	. . .	N-1차 Point Cloud 입력의 Y좌표	N차 Point Cloud 입력의 Y좌표
1차 Point Cloud 입력의 Z좌표	2차 Point Cloud 입력의 Z좌표	. . .	N-1차 Point Cloud 입력의 Z좌표	N차 Point Cloud 입력의 Z좌표
1차 Point Cloud 입력의 Intensity	2차 Point Cloud 입력의 Intensity	. . .	N-1차 Point Cloud 입력의 Intensity	N차 Point Cloud 입력의 Intensity

(※ 예시 : KITTI 데이터셋 3D LiDAR Point Cloud 데이터셋 구조)

- ❖ 레이저 스캐너가 수평 + 수직 이동을 모두 하기 때문에 2D LiDAR와 다르게 반사된 레이저가 발사된 최초의 수평 각도, 수직 각도로 들어올 보장이 없음. (레이저를 발사한 순간 이미 수직 각도가 변하기 때문임.)
- ❖ 그러므로 3D LiDAR는 입력된 데이터의 순서대로 데이터를 쌓으면 2D LiDAR와 달리 레이저 스캔의 각도에 맞춰서 데이터를 구성할 수 없음. 3D LiDAR Point Cloud는 발사한 레이저 스캔의 각도나 Channel 기준으로 데이터를 구성하는 것이 아닌 단순 입력 순서로 구성하기 때문에 'Unordered & Unstructured Data Structure'를 가지게됨.
- ❖ 3D LiDAR의 각 데이터는 입력 순서대로 반사되어 인식된 레이저 스캔의 LiDAR 좌표계상 반사된 지점의 X, Y, Z 위치 (또는 각좌표 (θ, Φ, r)) + 반사율 (Intensity / Reflectance)로 구성됨.
- ❖ 단순히 1회 회전 동안 입력된 데이터를 순서대로 쌓기만 하기 때문에 매 회전마다 입력되는 데이터의 양이 달라서 매 회전마다 데이터의 Dimension이 달라짐.

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – Understanding LiDAR Point Cloud]

- Unordered & Unstructured 3D LiDAR Point Cloud의 문제점
  - 매 회전마다 수집되는 데이터의 개수가 달라지기 때문에 시스템의 입력으로서 부적합함.
  - 센서의 입력 구조를 반영한 데이터 구조가 아니기 때문에 직접적으로 Feature Extraction이 매우 힘듬.

    ➔ 3D LiDAR Point Cloud를 구조화된 Grid 형태로 재구성해서 표현 해야할 필요가 있음.

- Raw 3D LiDAR Point Cloud 데이터 재구성 방법
  - Range Image Representation : 3D Point Cloud를 2D 구조로 펼쳐서 구성하는 방법
  - Voxel Map : 3D 좌표 공간을 Quantization하여 영역별로 데이터를 관리하는 방법 (Volumetric Representation)
  - Octo Map : 3D 좌표 공간을 Quantization하여 3D Occupancy Grid를 생성하는 방법 (Volumetric Representation)

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – KITTI Devkit Readme]

Folder 'poses': The folder 'poses' contains the ground truth poses (trajectory) for the first 11 sequences. This information can be used for training/tuning your method. Each file xx.txt contains a N x 12 table, where N is the number of frames of this sequence. Row i represents the i'th pose of the left camera coordinate system (i.e., z pointing forwards) via a 3x4 transformation matrix. The matrices are stored in row aligned order (the first entries correspond to the first row), and take a point in the i'th coordinate system and project it into the first (=0th) coordinate system. Hence, the translational part (3x1 vector of column 4) corresponds to the pose of the left camera coordinate system in the i'th frame with respect to the first(=0th) frame. Your submission results must be provided using the same data format.	▪ XX번째 Sequence의 6 DOF IMU Groundtruth 값은 poses 폴더에 XX.txt 파일에 저장되어있음 ▪ XX.txt의 매 줄에는 12개의 실수값으로 구성되어있음 ▪ 각 줄의 12개 실수값은 3x4 Transformation Matrix 구성값이며, Z축이 전진 방향으로 좌표계가 설정되어있음. ▪ 각 줄의 12개 실수값으로 구성된 3x4 Transformation Matrix는 왼쪽 카메라의 Translation과 Orientation임 ▪ i번째 줄의 Transformation Matrix은 Sequence별 최초 지점 (0번째 위치) 기준으로 Transformation한 것임.

DATASET SETUP

[KITTI Sensor & Groundtruth Setup – 6 DOF Pose]

0^{th} : 1.00e+00 9.04e-12 2.32e-11 5.55e-17 9.04e-12 1.00e+00 2.39e-10 3.33e-16 2.32e-11 2.39e-10 9.99e-01 -4.44e-16

$$\left[\begin{array}{ccc|c} 1.00e{-}00 & 9.04e{-}12 & 2.32e{-}11 & 5.55e{-}17 \\ 9.04e{-}12 & 1.00e{+}00 & 2.39e{-}10 & 3.33e{-}16 \\ 2.32e{-}11 & 2.39e{-}10 & 9.99e{-}01 & -4.44e{-}16 \end{array}\right]$$

Rotation Matrix **Translation Matrix**

→ **X축값과 Z축값을 이용하여 Grountruth Trajectory를 그릴 수 있음**

1^{st} : 9.99e-01 5.27e-04 -2.06e-03 -4.69e-02 -5.29e-04 9.99e-01 -1.15e-03 -2.83e-02 2.06e-03 1.15e-03 9.99e-01 8.58e-01

→ 0^{th} 위치를 기준으로 1^{st} 순간의 왼쪽 카메라의 Transformation Matrix

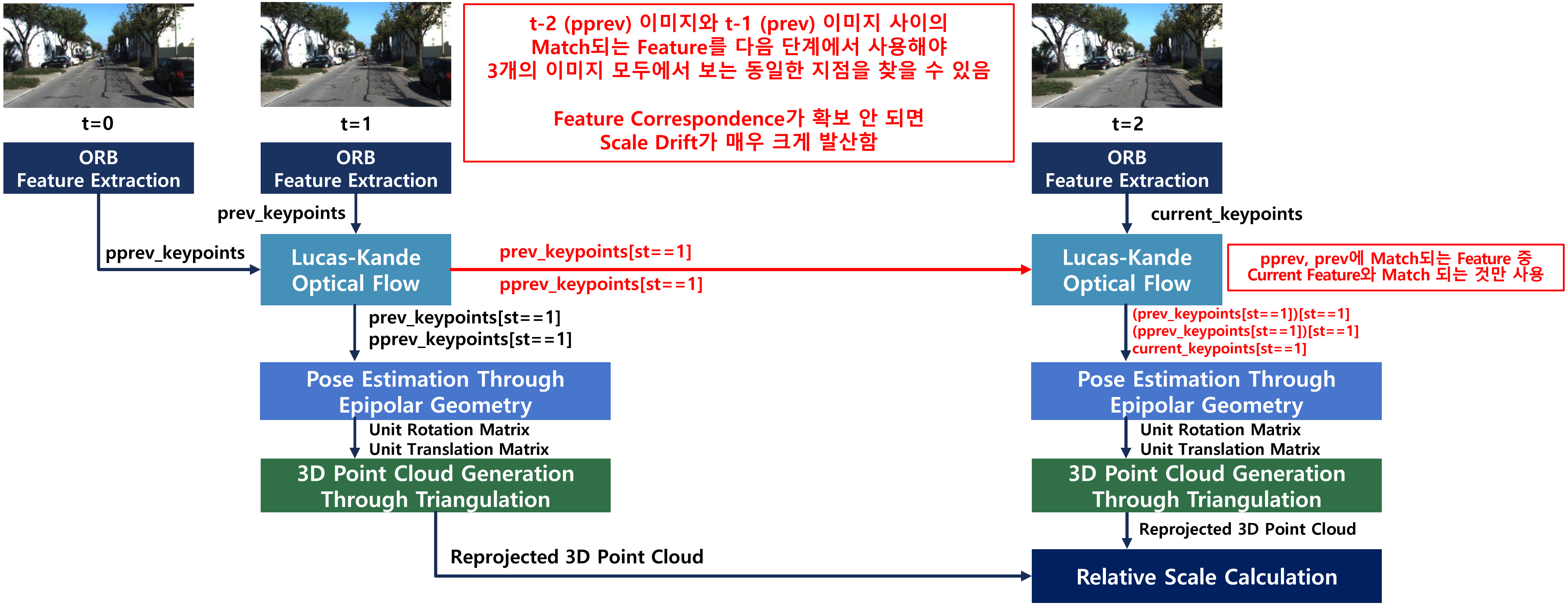
2^{nd} : 9.99e-01 1.04e-03 -4.13e-03 -9.37e-02 -1.05e-03 9.99e-01 -2.30e-03 -5.67e-02 4.12e-03 2.31e-03 9.99e-01 1.71e+00

→ 0^{th} 위치를 기준으로 2^{nd} 순간의 왼쪽 카메라의 Transformation Matrix

CLASSICAL MONOCULAR VO IMPLEMENTATION

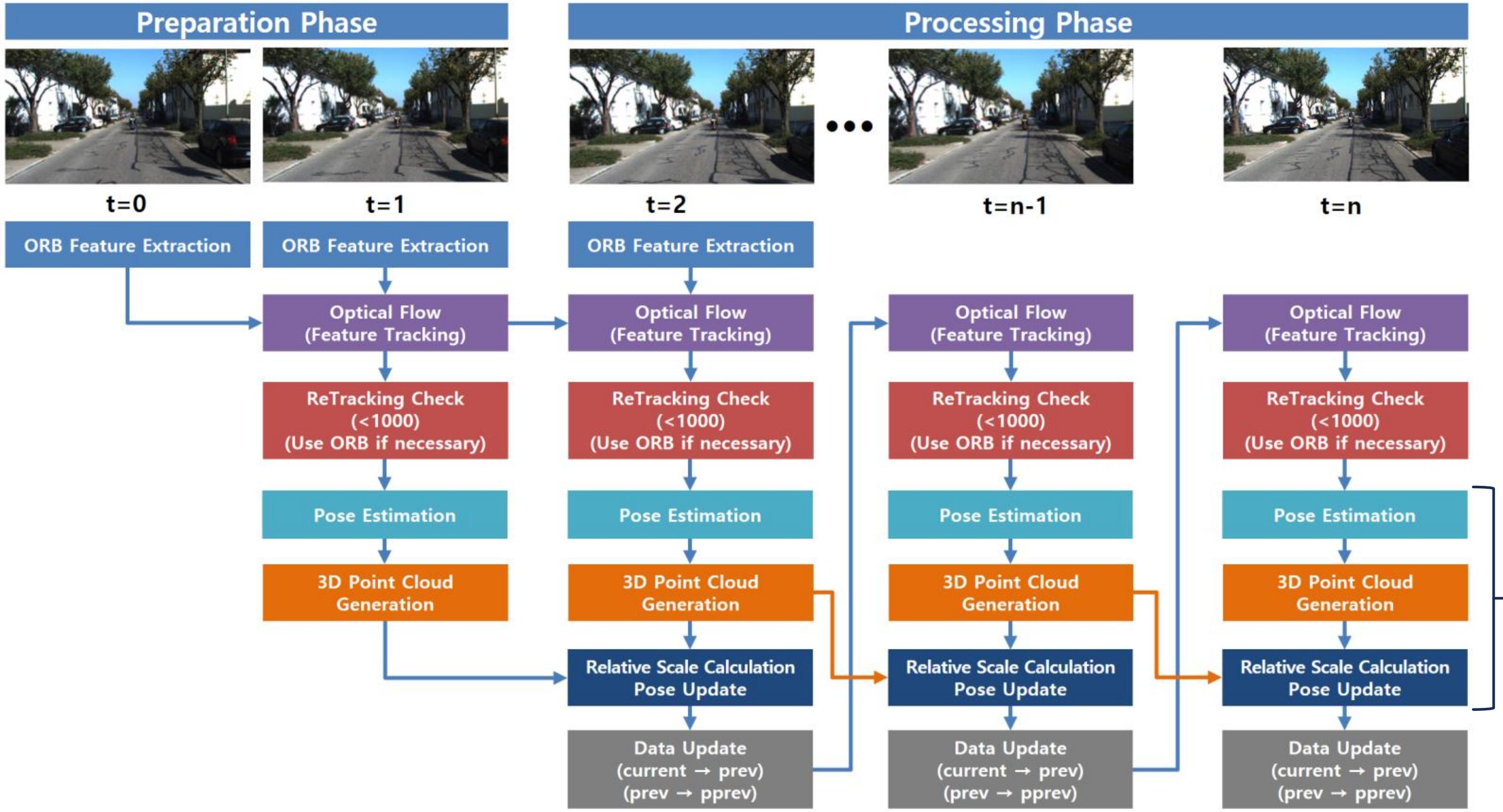
CLASSICAL MONOCULAR VO IMPLEMENTATION

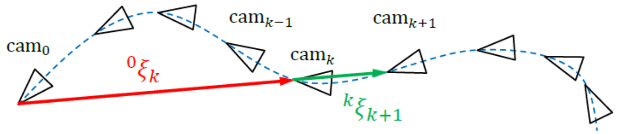
[Monocular VO Pipeline Implementation – Common Feature Selection & Relative Scale Estimation]



CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – Overall Algorithm Process]





Visual odometry from 2D-correspondences

1. Capture new frame img_{k+1}
2. Extract and match features between img_{k+1} and img_k
3. Estimate the essential matrix $E_{k,k+1}$
4. Decompose the $E_{k,k+1}$ into ${}^kR_{k+1}$ and ${}^k\boldsymbol{t}_{k+1}$ to get the relative pose
   $${}^k\xi_{k+1} = [{}^kR_{k+1} \quad {}^k\boldsymbol{t}_{k+1}]$$
5. Compute $\|{}^k\boldsymbol{t}_{k+1}\|$ from $\|{}^{k-1}\boldsymbol{t}_k\|$ and rescale ${}^k\boldsymbol{t}_{k+1}$ accordingly
6. Calculate the pose of camera $k+1$ relative to the first camera
   $${}^0\xi_{k+1} = {}^0\xi_k \boxed{{}^k\xi_{k+1}}$$

$$R_{pos} = RR_{pos}$$

$$t_{pos} = t_{pos} + tR_{pos}$$

t_{pos}의 X축, Z축 값을 Trajectory로 그려내면

Odometry Trajectory가 됨

소스코드 : https://github.com/luwis93choi/Multi-View_Monocular_Visual_Odometry

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – main.py]

```python
import visual_odometry_ORB_OptFlow_KITTI as vo
from matplotlib import pyplot as plt
VO_KITTI = vo.mono_VO_ORBFlow_KITTI(718.856, 718.856, 718.856, 607.1928, 185.2157,
                                    '--PATH_TO_SEQUENCE_SENSOR_DATASET--/sequences/--SEQUENCE_NUMBER--/image_2/',
                                    '--PATH_TO_SEQUENCE_POSE_DATASET--/dataset/poses/--SEQUENCE_NUMBER--.txt')
idx = 0
while True:
    VO_KITTI.img_buffer_feature_tracking(disp_img=True)
    if idx >= 3:
        if VO_KITTI.frame_Skip() == False:
            VO_KITTI.geometric_change_calc()
            VO_KITTI.img_common3Dcloud_triangulate()
            VO_KITTI.pose_estimate()
            VO_KITTI.update()

            # Draw the trajectory
            plt.title('KITTI Dataset - Monocular Visual Odometry (Relative Translation Scaling)\n[ORB-based Optical Flow]')
            plt.plot(VO_KITTI.pose_T[0][0], VO_KITTI.pose_T[2][0], 'ro')

            # Draw the groundtruth
            plt.plot(VO_KITTI.ground_truth_T[VO_KITTI.dataset_current_idx-1][0], VO_KITTI.ground_truth_T[VO_KITTI.dataset_current_idx-1][2], 'bo')

            plt.pause(0.000001)
            plt.show(block=False)

        else:

            print('[FRAME SKIPPED] : Camera is stationary / No need to accumulate pose data')
    else:

        idx += 1

    print('----------------------------------------------------')
```

- Monocular VO Pipeline 객체 라이브러리
- Trajectory 그리기 위한 Matplotlib
- Monocular VO Pipeline 객체 생성
- Common Feature Selection 수행 (Image Buffer 생성 • 업데이트 → ORB → Lucas-Kanade Optical Flow)
- Frame Skip : 연속된 이미지가 3개 미만인 경우 or Feature 개수가 적은 경우 or 정지 상태에서 이미지 변화가 거의 없는 경우
- Odometry 연산 수행
  ① geometric_change_calc : Epipolar Geometry
  ② Img_common3Dcloud_triangulate : Triangulation & Relative Scale Calculation
  ③ pose_estimation : Scale Estimation & Odometry Calculation
  ④ update : 현재값을 이전값으로 업데이트
- VO 결과 Trajectory 그리기
- Groundtruth Trajectory 그리기

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (1/13)]

(1) VO 객체 생성자

```python
class mono_VO_ORBFlow_KITTI:

    def __init__(self, _focal_length, _fx, _fy, _cx, _cy, _dataset_path, _dataset_pose_path):

        self.image_buffer = [None, None, None]      # image_buffer : [pprev_image, prev_image, current_image]
                                                    # ==> Store 3 consecutive images for translation rescaling

        self.img_features_buffer = [None, None, None]

        # Camera intrinsic matrix params for 2D-3D triangulation
        self.focal_length = _focal_length
        self.fx = _fx
        self.fy = _fy
        self.cx = _cx
        self.cy = _cy

        self.intrinsic_CAM_Mat = np.array([[self.fx, 0,       self.cx],
                                           [0,       self.fy, self.cy],
                                           [0,       0,       1      ]])

        print('focal length : ', self.focal_length)
        print('fx : ', self.fx)
        print('fy : ', self.fy)
        print('ppx : ', self.cx)
        print('ppy : ', self.cy)

        self.processing_Stage = '1st'
```

연속되는 이미지 3개를 저장하기 위한 Buffer 공간

연속되는 이미지 3개의 Feature를 저장하기 위한 Feature Buffer 공간

카메라 Intrinsic Parameter 및 Intrinsic Matrix

카메라의 Pixel 생성 특성 결정
Triangulation 연산 시 카메라의 물리적 특성 반영을 위해 사용함

현재 VO Pipeline 실행 단계 변수

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (2/13)]

(1) VO 객체 생성자

```python
# ORB Feature Extractor
self.orb = cv.ORB_create(nfeatures=2000)

# Optical Flow Parameters
self.lk_params = dict( winSize  = (21, 21),
                       criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 30, 0.01))

self.init_optical = True
self.init_cloud = True
self.initial_common_match_num = 0
self.retracking_ratio = 0.5

self.min_matches_num = 8

self.pose_T = np.array([[0],
                        [0],
                        [0]])

self.pose_R = np.array([[1, 0, 0],
                        [0, 1, 0],
                        [0, 0, 1]])

self.prev_pose_T = np.array([[0],
                             [0],
                             [0]])

self.prev_pose_R = np.array([[1, 0, 0],
                             [0, 1, 0],
                             [0, 0, 1]])

self.pprev_pose_T = np.array([[0],
                              [0],
                              [0]])

self.pprev_pose_R = np.array([[1, 0, 0],
                              [0, 1, 0],
                              [0, 0, 1]])

self.pprev_prev_cloud = None
self.prev_current_cloud = None
```

ORB Feature Extractor 선언 (최대 Feature 2000개 감지할 수 있도록 설정함)

Lucas-Kanade Optical Flow Parameter

pprev, prev, curren에 대한 Pose의 Translation과 Rotation 초기화

pprev-prev / prev-current 사이 Triangulation Point Cloud 변수

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (3/13)]

(1) VO 객체 생성자

```
self.dataset_path = _dataset_path
self.images = sorted(os.listdir(_dataset_path))
self.dataset_end_idx = len(os.listdir(_dataset_path)) - 1
self.dataset_current_idx = 0

self.geometric_unit_changes = {'R_pprev_prev' : None, 'T_pprev_prev' : None,
                               'R_prev_current' : None, 'T_prev_current' : None}

### Prepare groundtruth list ######################################################################
self.ground_truth_T = []
f = open(_dataset_pose_path, 'r')
init_flag = 0
while True:
    line = f.readline()
    if not line: break

    pose = line.strip().split()

    self.ground_truth_T.append([float(pose[3]), float(pose[7]), float(pose[11])])

    # Set up initial pprev_pose as 0th groundtruth
    if init_flag == 0:
        init_flag = 1
        self.pprev_pose_T = np.array([[float(pose[3])],
                                      [float(pose[7])],
                                      [float(pose[11])]])

        self.pprev_pose_R = np.array([[float(pose[0]), float(pose[1]), float(pose[2])],
                                      [float(pose[4]), float(pose[5]), float(pose[6])],
                                      [float(pose[8]), float(pose[9]), float(pose[10])]])

    # Set up inital prev_pose as 1st ground truth
    elif init_flag == 1:
        init_flag = 2
        self.prev_pose_T = np.array([[float(pose[3])],
                                     [float(pose[7])],
                                     [float(pose[11])]])

        self.prev_pose_R = np.array([[float(pose[0]), float(pose[1]), float(pose[2])],
                                     [float(pose[4]), float(pose[5]), float(pose[6])],
                                     [float(pose[8]), float(pose[9]), float(pose[10])]])

f.close()
##################################################################################################

print('Dataset Path : ', self.dataset_path)
print('Ground Truth Path : ', _dataset_pose_path)
```

이미지 데이터셋, 6DOF IMU 데이터셋이 저장된 공간의 파일 리스트를 불러옴

pprev-prev / prev-current 사이
Translation & Rotation 변화를 기록하기 위한 Dictionary 변수

- pose 텍스트 파일을 한 줄씩 읽음
- 매 번마다 읽은 데이터를 띄어쓰기 기준으로 분리시켜서, 12개 실수를 읽음

매 번마다 읽은 12개 실수에서 Translation에 해당되는 4, 8, 12번째 값을 Grondtruth Translation으로 기록하여 저장함

0번째로 읽은 12개의 실수를 pprev 최초 Transformation (Translation & Rotation)으로 초기화함

1번째로 읽은 12개의 실수를 prev 최초 Transformation (Translation & Rotation)으로 초기화함

읽고 있는 pose 텍스트 파일을 닫음

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (4/13)]

(2) Frame Skip

```
def frame_Skip(self):

    pprev_match_keypoints_pts = np.float32(self.img_features_buffer[0])
    prev_match_keypoints_pts = np.float32(self.img_features_buffer[1])
    current_match_keypoints_pts = np.float32(self.img_features_buffer[2])

    pixel_diff = np.mean(abs(current_match_keypoints_pts - prev_match_keypoints_pts))

    print('[Pixel DIFF] : ',  pixel_diff)

    return pixel_diff < 3
```

3개 연속 이미지에서 동시에 감지된 Feautre의 Keypoint 위치들이 평균적으로 거의 변화가 없다면 정지상태로 인식하고 Frame Skip을 수행해서 Odometry를 누적하지 않음

➔ 정지 상태에서 Odometry Error 누적을 방지함

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (5/13)]

(3) Common Feature Selection – 첫번째 단계 : 0번째 이미지 (pprev) 로딩 및 Feature Extraction

```python
def img_buffer_feature_tracking(self, disp_img=True):

    # At 1st Frame image, conduct ORB feature extraction and convert it into an appropriate numpy format for optical flow
    if self.processing_Stage == '1st':
        print('[INFO] 1st image')

        query_img = cv.imread(self.dataset_path + self.images[self.dataset_current_idx])

        pprev_gray = cv.cvtColor(query_img, cv.COLOR_BGR2GRAY)
        self.image_buffer[0] = pprev_gray

        # Grayscale ORB feature extraction
        pprev_keypoints = self.orb.detect(pprev_gray, None)
        pprev_keypoints, pprev_descriptors = self.orb.compute(pprev_gray, pprev_keypoints)
        pprev_keypoints = np.array((pprev_keypoints))

        # Datatype conversion for optical flow
        temp_pprev_keypoints = []
        for keypoint in pprev_keypoints:
            temp_pprev_keypoints.append([[np.float32(keypoint.pt[0]), np.float32(keypoint.pt[1])]])

        self.img_features_buffer[0] = np.array(temp_pprev_keypoints)

        self.processing_Stage = '2nd'
        self.dataset_current_idx += 1
```

0번째 이미지 (pprev) 로딩 및 이미지 Buffer에 저장
Grayscale 전환

0번째 이미지에 대한
ORB Feature Extraction 수행

Optical Flow에 적합한 형태로
Keypoint 리스트를 재구성하고
Feature Buffer에 저장함

VO Pipeline 다음 단계로 넘어가도록 설정하고
다음 이미지를 읽도록 Index를 증가시킴

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (6/13)]

(3) Common Feature Selection – 두번째 단계 : 1번째 이미지 (prev) 로딩 / pprev-prev Feature Tracking

```python
elif self.processing_Stage == '2nd':
    print('[INFO] 2nd image')

    query_img = cv.imread(self.dataset_path + self.images[self.dataset_current_idx])

    prev_gray = cv.cvtColor(query_img, cv.COLOR_BGR2GRAY)
    self.image_buffer[1] = prev_gray

    # Optical Flow Feature Tracking
    self.img_features_buffer[1], st, err = cv.calcOpticalFlowPyrLK(self.image_buffer[0], self.image_buffer[1],
                                                                   self.img_features_buffer[0], None, **self.lk_params)

    self.img_features_buffer[0] = (self.img_features_buffer[0][st==1]).reshape(-1, 1, 2)
    self.img_features_buffer[1] = (self.img_features_buffer[1][st==1]).reshape(-1, 1, 2)

    print('[INFO] pprev-prev common feature num : ', len(self.img_features_buffer[1]))

    ### Feature Retracking over 2 images ##########################################################################################
    if len(self.img_features_buffer[0]) < 10:
        print('[Re-Tracking] Not Enough Features between pprev and prev')

        # pprev Grayscale ORB feature extraction
        pprev_keypoints = self.orb.detect(self.image_buffer[0], None)
        pprev_keypoints, pprev_descriptors = self.orb.compute(self.image_buffer[0], pprev_keypoints)
        pprev_keypoints = np.array((pprev_keypoints))

        # pprev Datatype conversion for optical flow
        temp_pprev_keypoints = []
        for keypoint in pprev_keypoints:
            temp_pprev_keypoints.append([[np.float32(keypoint.pt[0]), np.float32(keypoint.pt[1])]])

        self.img_features_buffer[0] = np.array(temp_pprev_keypoints)

        # Re-Track between pprev and prev
        self.img_features_buffer[1], st, err = cv.calcOpticalFlowPyrLK(self.image_buffer[0], self.image_buffer[1],
                                                                       self.img_features_buffer[0], None, **self.lk_params)

        self.img_features_buffer[0] = (self.img_features_buffer[0][st==1]).reshape(-1, 1, 2)
        self.img_features_buffer[1] = (self.img_features_buffer[1][st==1]).reshape(-1, 1, 2)
    ###############################################################################################################################

    self.processing_Stage = 'process'
    self.dataset_current_idx += 1
```

1번째 이미지 (prev) 로딩 및 이미지 Buffer에 저장
Grayscale 전환

0번째 (pprev), 1번째 (prev) 이미지 사이에
Lucas-Kanade Optical Flow를 적용하여
동일 Feature를 추적함

Feature Correspondence에 대해 status가 1이 되는 것을 이용하여
status 1에 해당되는 Index를 찾아서 필터링함

필터링된 최종 Feature들을 Feature Buffer에 저장함

Feature 개수가 너무 적으면 (<10)
pprev, prev에 대해 Feature Extraction 부터
다시 적용함

Feature Correspondence에 대해 status가 1이 되는 것을 이용하여
status 1에 해당되는 Index를 찾아서 필터링함

VO Pipeline 다음 단계로 넘어가도록 설정하고
다음 이미지를 읽도록 Index를 증가시킴

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (7/13)]

(3) Common Feature Selection – 세번째 ~ 그 이후 단계 : 연속 3개 이미지에 대해 Feature Tracking 수행

```
elif self.processing_Stage == 'process':

    query_img = cv.imread(self.dataset_path + self.images[self.dataset_current_idx])

    current_gray = cv.cvtColor(query_img, cv.COLOR_BGR2GRAY)
    self.image_buffer[2] = current_gray

    # Optical Flow Feature Tracking
    self.img_features_buffer[2], st, err = cv.calcOpticalFlowPyrLK(self.image_buffer[1], self.image_buffer[2],
                                                                    self.img_features_buffer[1], None, **self.lk_params)

    self.img_features_buffer[1] = (self.img_features_buffer[1][st==1]).reshape(-1, 1, 2)
    self.img_features_buffer[2] = (self.img_features_buffer[2][st==1]).reshape(-1, 1, 2)

    self.img_features_buffer[0] = (self.img_features_buffer[0][st==1]).reshape(-1, 1, 2)

    if self.init_optical == True:
        self.initial_common_match_num = len(self.img_features_buffer[2])
        self.init_optical = False

    print('[INFO] pprev-prev-current common feature num : ', len(self.img_features_buffer[2]))

    print('pprev common num : ', len(self.img_features_buffer[0]))
    print('prev common num : ', len(self.img_features_buffer[1]))
    print('current common num : ', len(self.img_features_buffer[2]))
```

3번째 이후 이미지 (current) 로딩 및 이미지 Buffer에 저장
Grayscale 전환

이전 단계에서 이미 필터링된 Feature들만
사용하여 Feature를 추적함

최종적으로 Optical Flow를 통해
필터링된 Feature들은
3개 이미지에서 존재하는 Feature들임

pprev-prev / prev-current 사이 동시 감지되는 Feature
Keypoint 지점을 Optical Flow를 통해서 찾아냄

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (8/13)]

(3) Common Feature Selection – 세번째 ~ 그 이후 단계 : 연속 3개 이미지에 대해 Feature Tracking 수행

```python
### Feature Retracking over all 3 images ##########################################################################
#if len(self.img_features_buffer[2]) < (self.initial_common_match_num * self.retracking_ratio):
if len(self.img_features_buffer[2]) < 500:
    print('[Re-Tracking] Not Enough Features between pprev and prev / Too many have been consumed')

    ### pprev-prev ###
    # pprev Grayscale ORB feature extraction
    pprev_keypoints = self.orb.detect(self.image_buffer[0], None)
    pprev_keypoints, pprev_descriptors = self.orb.compute(self.image_buffer[0], pprev_keypoints)
    pprev_keypoints = np.array((pprev_keypoints))

    # pprev Datatype conversion for optical flow
    temp_pprev_keypoints = []
    for keypoint in pprev_keypoints:
        temp_pprev_keypoints.append([[np.float32(keypoint.pt[0]), np.float32(keypoint.pt[1])]])

    self.img_features_buffer[0] = np.array(temp_pprev_keypoints)

    # Re-Track between pprev and prev
    self.img_features_buffer[1], st, err = cv.calcOpticalFlowPyrLK(self.image_buffer[0], self.image_buffer[1],
                                                                   self.img_features_buffer[0], None, **self.lk_params)

    self.img_features_buffer[0] = (self.img_features_buffer[0][st==1]).reshape(-1, 1, 2)
    self.img_features_buffer[1] = (self.img_features_buffer[1][st==1]).reshape(-1, 1, 2)

    # Re-Track between prev and current
    self.img_features_buffer[2], st, err = cv.calcOpticalFlowPyrLK(self.image_buffer[1], self.image_buffer[2],
                                                                   self.img_features_buffer[1], None, **self.lk_params)

    self.img_features_buffer[1] = (self.img_features_buffer[1][st==1]).reshape(-1, 1, 2)
    self.img_features_buffer[2] = (self.img_features_buffer[2][st==1]).reshape(-1, 1, 2)

    self.img_features_buffer[0] = (self.img_features_buffer[0][st==1]).reshape(-1, 1, 2)

    # Update common cloud number
    self.initial_common_match_num = len(self.img_features_buffer[2])

    self.init_cloud = True
##############################################################################################################
```

Feature 개수가 너무 적으면 (<500)
pprev, prev, current에 대해
Feature Extraction 부터 다시 적용함

1차 Feature Correspondence Filtering (pprev-prev)

2차 Feature Correspondence Filtering (pprev-prev / prev-current)

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (9/13)]

(3) Common Feature Selection – 기타 기능 : 연속 3개 이미지의 Feature Correspondence 시각화

```python
    # Display Optical Flow results if the display option is True
    if disp_img == True:

        color = np.random.randint(0, 255, (4500, 3))

        query_img = cv.imread(self.dataset_path + self.images[self.dataset_current_idx-1])

        # Display Optical Flow Results
        mask = np.zeros_like(query_img)

        for i, (new, old) in enumerate(zip(self.img_features_buffer[1], self.img_features_buffer[2])):
            a, b = new.ravel()
            c, d = old.ravel()
            mask = cv.line(mask, (a, b), (c, d), color[i].tolist(), 2)
            query_img = cv.circle(query_img, (a, b), 5, color[i].tolist(), -1)

        img = cv.add(query_img.copy(), mask)

        cv.imshow('ORB-based Optical Flow + ReTracking', img)
        k = cv.waitKey(30)

    self.dataset_current_idx += 1
```

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (10/13)]

(3) Epipolar Geometry – Essential Matrix 연산 / Decomposition을 통한 Translation & Rotation 연산

```
def geometric_change_calc(self):

    pprev_match_keypoints_pts = self.img_features_buffer[0]

    prev_match_keypoints_pts = self.img_features_buffer[1]

    current_match_keypoints_pts = self.img_features_buffer[2]

    ### Essential Matrix Calcuation & Rotation/Translation Matrix Calculation ###
    Essential_Mat_pprev_prev, mask_pprev_prev = cv.findEssentialMat(np.int32(prev_match_keypoints_pts),
                                                                    np.int32(pprev_match_keypoints_pts),
                                                                    cameraMatrix=self.intrinsic_CAM_Mat,
                                                                    method=cv.RANSAC, prob=0.99, threshold=1.0)

    Essential_Mat_prev_current, mask_prev_current = cv.findEssentialMat(np.int32(current_match_keypoints_pts),
                                                                        np.int32(prev_match_keypoints_pts),
                                                                        cameraMatrix=self.intrinsic_CAM_Mat,
                                                                        method=cv.RANSAC, prob=0.99, threshold=1.0)

    retval, Rotation_Mat_pprev_prev, Translation_Mat_pprev_prev, r_mask_pprev_prev = cv.recoverPose(Essential_Mat_pprev_prev,
                                                                                                    np.int32(prev_match_keypoints_pts),
                                                                                                    np.int32(pprev_match_keypoints_pts),
                                                                                                    cameraMatrix=self.intrinsic_CAM_Mat)

    retval, Rotation_Mat_prev_current, Translation_Mat_prev_current, r_mask_prev_current = cv.recoverPose(Essential_Mat_prev_current,
                                                                                                          np.int32(current_match_keypoints_pts),
                                                                                                          np.int32(prev_match_keypoints_pts),
                                                                                                          cameraMatrix=self.intrinsic_CAM_Mat)

    self.geometric_unit_changes['R_pprev_prev'] = Rotation_Mat_pprev_prev
    self.geometric_unit_changes['T_pprev_prev'] = Translation_Mat_pprev_prev
    self.geometric_unit_changes['R_prev_current'] = Rotation_Mat_prev_current
    self.geometric_unit_changes['T_prev_current'] = Translation_Mat_prev_current

    print('[R_pprev_prev]')
    print(Rotation_Mat_pprev_prev)
    print('[T_pprev_prev]')
    print(Translation_Mat_pprev_prev)
    print('[R_prev_current]')
    print(Rotation_Mat_prev_current)
    print('[T_prev_current]')
    print(Translation_Mat_prev_current)

    return self.geometric_unit_changes
```

pprev-prev 사이의 Epipolar Geometry를 이용한
Essential Matrix 연산 수행

prev-current 사이의 Epipolar Geometry를 이용한
Essential Matrix 연산 수행

pprev-prev 사이의
Essential Matrix Decomposition을 통한
Translation & Rotation 연산

prev-current 사이의
Essential Matrix Decomposition을 통한
Translation & Rotation 연산

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (11/13)]

(4) Triangulation – Relative Scale 연산을 위한 기초 작업

```python
def img_common3Dcloud_triangulate(self):

    pprev_match_keypoints_pts = np.float32(self.img_features_buffer[0]).reshape(2, -1)
    prev_match_keypoints_pts = np.float32(self.img_features_buffer[1]).reshape(2, -1)
    current_match_keypoints_pts = np.float32(self.img_features_buffer[2]).reshape(2, -1)

    Rotation_Mat_pprev_prev = self.geometric_unit_changes['R_pprev_prev']
    Translation_Mat_pprev_prev = self.geometric_unit_changes['T_pprev_prev']
    Rotation_Mat_prev_current = self.geometric_unit_changes['R_prev_current']
    Translation_Mat_prev_current = self.geometric_unit_changes['T_prev_current']

    #if self.init_cloud == True:
    print('[INFO] pprev-prev Triangulation')
    ### Triangluation between pprev and prev
    # The canonical matrix (set as the origin)
    P0 = np.array([[1, 0, 0, 0],
                    [0, 1, 0, 0],
                    [0, 0, 1, 0]])
    P0 = self.intrinsic_CAM_Mat.dot(P0)
    # Rotated and translated using P0 as the reference point
    P1 = np.hstack((Rotation_Mat_pprev_prev, Translation_Mat_pprev_prev))
    P1 = self.intrinsic_CAM_Mat.dot(P1)

    self.pprev_prev_cloud = cv.triangulatePoints(P0, P1, pprev_match_keypoints_pts, prev_match_keypoints_pts).reshape(-1, 4)[:, :3]

    self.init_cloud = False

    print('[INFO] prev-current Triangulation')
    ### Triangluation between prev and current
    # The canonical matrix (set as the origin)
    P0 = np.array([[1, 0, 0, 0],
                   [0, 1, 0, 0],
                   [0, 0, 1, 0]])
    P0 = self.intrinsic_CAM_Mat.dot(P0)
    # Rotated and translated using P0 as the reference point
    P1 = np.hstack((Rotation_Mat_prev_current, Translation_Mat_prev_current))
    P1 = self.intrinsic_CAM_Mat.dot(P1)

    self.prev_current_cloud = cv.triangulatePoints(P0, P1, prev_match_keypoints_pts, current_match_keypoints_pts).reshape(-1, 4)[:, :3]

    return self.pprev_prev_cloud, self.prev_current_cloud
```

pprev-prev 사이의
Triangulation을 통한
Feautre Keypoint에 대한 거리값 계산

prev-current 사이의
Triangulation을 통한
Feautre Keypoint에 대한 거리값 계산

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (12/13)]

(5) Pose Estimation – Relative Scale 연산 / IMU를 이용한 Absolute Scale 연산 / Forward Motion Model 적용

```python
def pose_estimate(self):

    print('pprev-prev cloud num : ', len(self.pprev_prev_cloud))
    print('prev-current cloud num : ', len(self.prev_current_cloud))

    ratios = []
    for i in range(len(self.prev_current_cloud)):
        if i > 0:
            current_Xk1 = self.prev_current_cloud[i]
            current_Xk2 = self.prev_current_cloud[i-1]

            prev_Xk1 = self.pprev_prev_cloud[i]
            prev_Xk2 = self.pprev_prev_cloud[i-1]

            if (np.linalg.norm(current_Xk1 - current_Xk2) != 0):
                ratios.append(np.linalg.norm(prev_Xk1 - prev_Xk2) / np.linalg.norm(current_Xk1 - current_Xk2))

    T_relative_scale = np.median(ratios)
    #T_relative_scale = np.mean(ratios)

    print('Relative Scale for Translation : ', T_relative_scale)

    absolute_scale = np.linalg.norm(np.array(self.ground_truth_T[self.dataset_current_idx]) - np.array(self.ground_truth_T[self.dataset_current_idx-1]))
    print('Absolute Sclae for Translation : ', absolute_scale)

    # Apply Forward Dominant Motoin Model with Absolute Value Comparison.
    # This is implemented to prevent the error of accumulating Translation and Rotation when the camera is stationary, but the image is changing.
    # This condition is used to prevent nearby moving object's feature keypoint changes from affecting the calculation of translation and rotation.
    # Absolute value of Z is compared to absolute value of Y and X. Absolute value is used to make this condition work under both foward and backward movements of the camera.
    if ( (abs(self.geometric_unit_changes['T_prev_current'][2]) > abs(self.geometric_unit_changes['T_prev_current'][0])) and
         (abs(self.geometric_unit_changes['T_prev_current'][2]) > abs(self.geometric_unit_changes['T_prev_current'][1])) ):
        self.pose_T = self.prev_pose_T + absolute_scale * self.prev_pose_R.dot(self.geometric_unit_changes['T_prev_current'])
        self.pose_R = self.geometric_unit_changes['R_prev_current'].dot(self.prev_pose_R)

        print('[INFO] Dominant Forward : Pose Estimation Results')
        print(self.pose_T)
        print(self.pose_R)

    return self.pose_T, self.pose_R
```

Triangulation 결과에 대해
Relative Scale을 연산함

Groundtruth IMU를 이용하여
Absolute Scale을 연산함

Odometry 누적 연산 수행

$$R_{pos} = RR_{pos} \qquad t_{pos} = t_{pos} + tR_{pos}$$

Forward Motion Model 적용

- 4륜 차량의 조향 특성을 고려하여 전진 변화가 더 큰 경우에 대해서만 Odometry를 적용함

CLASSICAL MONOCULAR VO IMPLEMENTATION

[Monocular VO Pipeline Implementation – visual_odometry_ORB_OptFlow_KITTI.py (13/13)]

(6) Update – 현재 결과를 다음 단계에서 이전 값으로 사용할 수 있도록 업데이트

```python
def update(self):

    # Update current values as prev values
    self.image_buffer[0] = self.image_buffer[1]
    self.img_features_buffer[0] = self.img_features_buffer[1]

    self.image_buffer[1] = self.image_buffer[2]
    self.img_features_buffer[1] = self.img_features_buffer[2]

    self.geometric_unit_changes['R_pprev_prev'] = self.geometric_unit_changes['R_prev_current']
    self.geometric_unit_changes['T_pprev_prev'] = self.geometric_unit_changes['T_prev_current']

    self.pprev_prev_cloud = self.prev_current_cloud

    self.prev_pose_T = self.pose_T
    self.prev_pose_R = self.pose_R
```

다음주 주제 / 이번주 과제

[다음주 주제]

- Basic Dataset Processing using Python
- Basics of Machine Learning

[이번주 과제]

① Classical Monocular VO 구현 / KITTI Sequence 01에 대한 VO Trajectory Estimation 결과 그리기

- 참고자료 1 : https://github.com/luwis93choi/Multi-View_Monocular_Visual_Odometry/blob/main/Multi_View_Visual_Odometry/main_ORBoptflow.py
- 참고자료 2 : https://github.com/luwis93choi/Multi-View_Monocular_Visual_Odometry/blob/main/Multi_View_Visual_Odometry/visual_odometry_ORB_OptFlow_KITTI.py
- 참고자료 3 : https://avisingh599.github.io/vision/monocular-vo/

감사합니다