OpenVINS Study Week#2

2024. 09. 24

Chanjoon Park 박찬준

chanjoon.park@kaist.ac.kr

KAIST

URL
URBAN ROBOTICS LAB

Contents

- 지난 주차 스터디에서 나온 질문 논의
  - First Esimate Jacobian (FEJ) ↔ marginalization
  - UpdateMSCKF() 와 UpdateSLAM() 차이점

- Code Analysis
  - trackFEATS 객체까지의 플로우차트
  - 차트 흐름에 따라 자세히 살펴보기
  - KLT Tracking vs. Descriptor (FAST + ORB)
    - Code functions, arguments, …
- Visualization
- Discussion

지난 주차 나온 질문

- 1. First Esimate Jacobian (FEJ) ↔ marginalization
  - FEJ
    - 현재 estimated state $\hat{x}_k$의 Jacobian 을 매번 계산하기보다 첫 번째에서 얻은 Jacobian 을 고정해둠.
    - linearization errors 를 줄이고 **consistent covariance estimate** 를 보장

$$\mathbf{z}_k \approx h(\hat{\mathbf{x}}_{k|k-1} + \mathbf{H}_k(\mathbf{x}_k - \hat{\mathbf{x}}_{k|k-1}) + \mathbf{n}_k \qquad (1)$$

$$\mathbf{H}_k = \frac{\partial h(\hat{\mathbf{x}}_{k|k-1})}{\partial \mathbf{x}_k} \qquad \mathbf{H}_{FEJ} = \frac{\partial h(\hat{\mathbf{x}}_0)}{\partial \mathbf{x}_0}$$

    - 참고 함수
    - void UpdateHelper::get_feature_jacobian_representation()
    - void UpdaterHelper::get_feature_jacobian_full()

  - Marginalization
    - sliding window 를 벗어난 이전의 state 들을 marginalize 하여 연산속도를 확보
    - filtering 의 불확실성을 줄이고자 하는 FEJ 와 달리, state vector 의 크기를 줄여서 계산 복잡도를 줄임

지난 주차 나온 질문

- 2. UpdateMSCKF() 와 UpdateSLAM() 차이점
  - 직접적인 구현 내용은 propagation 파트이므로 생략.
  - UpdateMSCKF() 의 경우, feature(or landmark) 를 담지 않음.
  - UpdateSLAM() 에서는 이를 포함함.
  - 따라서, OpenVINS 에서는 MSCKF 기반이지만 선택적으로 slam feature 들을 사용할 수 있게 함.
  - 논문에 따르면 1) inverse MSCKF[4], 2) full inverse depth[5], 3) anchored 3D position[6] 3가지를 선택해서 사용할 수 있다고 함.

Original MSCKF

$$\hat{\mathbf{X}}_k = \begin{bmatrix} \hat{\mathbf{X}}_{\text{IMU}_k}^T & {}^{C_1}_{G}\hat{\bar{q}}^T & {}^{G}\hat{\mathbf{p}}_{C_1}^T & \cdots & {}^{C_N}_{G}\hat{\bar{q}}^T & {}^{G}\hat{\mathbf{p}}_{C_N}^T \end{bmatrix}^T \quad (4)$$

IMU / Camera pose

OpenVINS

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}_I^\top & \mathbf{x}_C^\top & \mathbf{x}_M^\top & \mathbf{x}_W^\top & {}^{C}t_I \end{bmatrix}^\top \quad (1)$$

$$\mathbf{x}_I = \begin{bmatrix} {}^{I_k}_{G}\bar{q}^\top & {}^{G}\mathbf{p}_{I_k}^\top & {}^{G}\mathbf{v}_{I_k}^\top & \mathbf{b}_{\omega_k}^\top & \mathbf{b}_{a_k}^\top \end{bmatrix}^\top \quad (2)$$

$$\mathbf{x}_C = \begin{bmatrix} {}^{I_{k-1}}_{G}\bar{q}^\top & {}^{G}\mathbf{p}_{I_{k-1}}^\top & \cdots & {}^{I_{k-c}}_{G}\bar{q}^\top & {}^{G}\mathbf{p}_{I_{k-c}}^\top \end{bmatrix}^\top \quad (3)$$

$$\mathbf{x}_M = \begin{bmatrix} {}^{G}\mathbf{p}_{f_1}^\top & \cdots & {}^{G}\mathbf{p}_{f_m}^\top \end{bmatrix}^\top \quad (4)$$

Environmental landmarks

$$\mathbf{x}_W = \begin{bmatrix} {}^{I}_{C_1}\bar{q}^\top & {}^{C_1}\mathbf{p}_I^\top & \zeta_0^\top & \cdots & {}^{I}_{C_w}\bar{q}^\top & {}^{C_w}\mathbf{p}_I^\top & \zeta_w^\top \end{bmatrix}^\top \quad (5)$$

Related Papers

[1] Li M, Mourikis AI. High-precision, consistent EKF-based visual-inertial odometry. The International Journal of Robotics Research. 2013;32(6):690-711. doi:10.1177/0278364913481251

[2] Li, Mingyang, and Anastasios I. Mourikis. "High-precision, consistent EKF-based visual-inertial odometry." The International Journal of Robotics Research 32.6 (2013): 690-711. journals.sagepub.com/doi/full/10.1177/0278364913481251

[3] Li, Mingyang, and Anastasios I. Mourikis. "Optimization-based estimator design for vision-aided inertial navigation." Robotics: Science and Systems. Germany: Berlin, 2013. http://roboticsproceedings.org/rss08/p31.pdf

[4] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in Proceedings of the IEEE International Conference on Robotics and Automation, Rome, Italy, Apr. 10–14, 2007, pp. 3565–3572.

[5] J. Civera, A. Davison, and J. Montiel, "Inverse depth parametrization for monocular SLAM," IEEE Transactions on Robotics, vol. 24, no. 5, pp. 932–945, Oct. 2008.

[6] M. K. Paul, K. Wu, J. A. Hesch, E. D. Nerurkar, and S. I. Roumeliotis, "A comparative analysis of tightly-coupled monocular, binocular, and stereo VINS," in Proc. of the IEEE International Conference on Robotics and Automation, Singapore, July 2017, pp. 165–172.

Related Github Issues

1. VIO vs SLAM #279: https://github.com/rpng/open_vins/issues/279
2. environmental landmarks in state vector? #115: https://github.com/rpng/open_vins/issues/115
3. Update Types #108: https://github.com/rpng/open_vins/issues/108
4. Question: MSCKF features in update #107: https://github.com/rpng/open_vins/issues/107

BLUE: 클래스/객체
PINK: 함수명

Code Analysis

- trackFEATS 객체 까지의 플로우차트

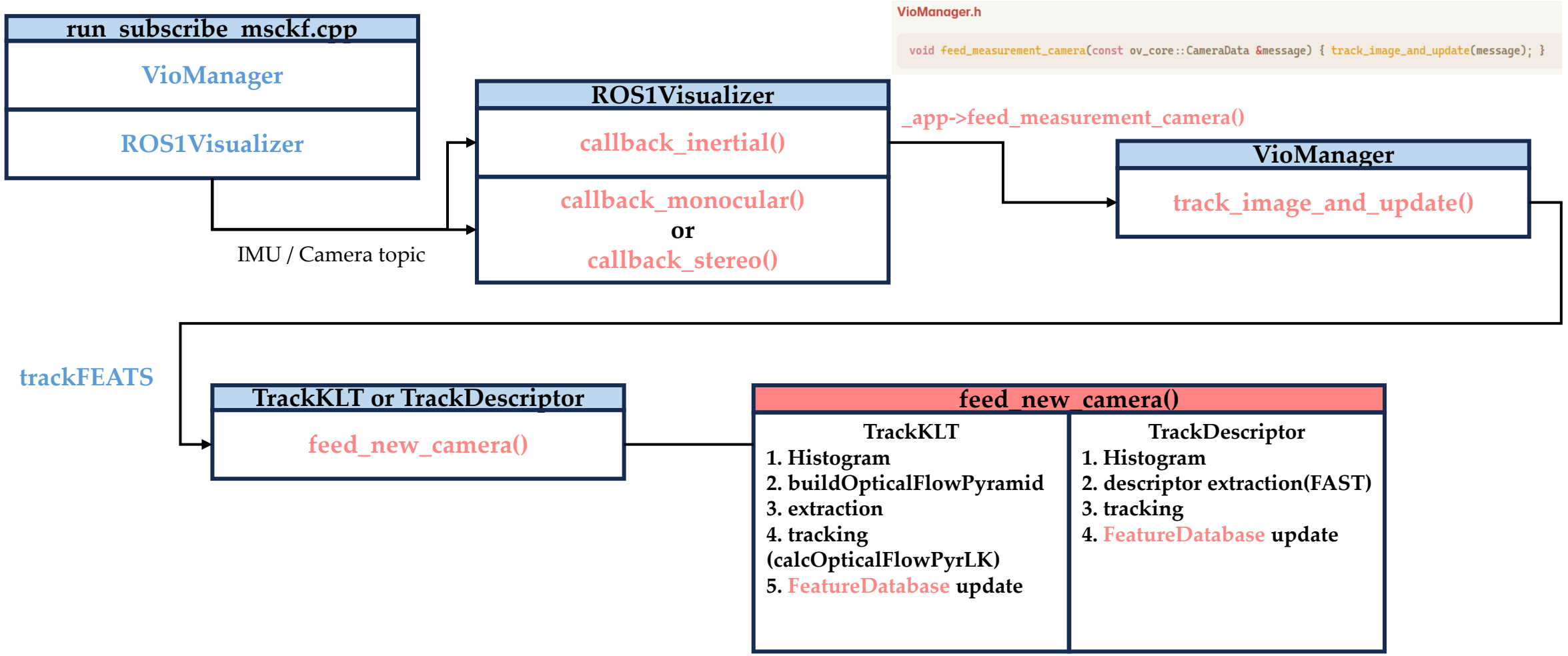


차트 흐름에 따라 자세히 보기

- run_subscribe_msckf.cpp
  - sys: VioManager
  - viz: ROS1Visualizer
- ROS1Visualizer.cpp
  - setup_subscribers()
  - IMU topic: callback_inertial()
  - Camera topic
    - callback_monocular()
    - callback_stereo()
  - 클래스 생성 시 _app 변수로 VioManager 클래스를 사용함.

```
  // Create our VIO system
  VioManagerOptions params;
  params.print_and_load(parser);
  params.use_multi_threading_subs = true;
  sys = std::make_shared<VioManager>(params);
#if ROS_AVAILABLE == 1
  viz = std::make_shared<ROS1Visualizer>(nh, sys);
  viz->setup_subscribers(parser);
```

```
void ROS1Visualizer::setup_subscribers(std::shared_ptr<ov_core::YamlParser> parser) {
  // Create imu subscriber (handle legacy ros param info)
  sub_imu = _nh->subscribe(topic_imu, 1000, &ROS1Visualizer::callback_inertial, this);

  // Logic for sync stereo subscriber
  // https://answers.ros.org/question/96346/subscribe-to-two-image_raws-with-one-function/?answer=96491#post-id-96491
  if (_app->get_params().state_options.num_cameras == 2) {
    // Create sync filter (they have unique pointers internally, so we have to use move logic here...)
    auto image_sub0 = std::make_shared<message_filters::Subscriber<sensor_msgs::Image>>(*_nh, cam_topic0, 1);
    auto image_sub1 = std::make_shared<message_filters::Subscriber<sensor_msgs::Image>>(*_nh, cam_topic1, 1);
    auto sync = std::make_shared<message_filters::Synchronizer<sync_pol>>(sync_pol(10), *image_sub0, *image_sub1);
    sync->registerCallback(boost::bind(&ROS1Visualizer::callback_stereo, this, _1, _2, 0, 1));
    // Append to our vector of subscribers
    sync_cam.push_back(sync);
    sync_subs_cam.push_back(image_sub0);
    sync_subs_cam.push_back(image_sub1);
  } else {
    // Now we should add any non-stereo callbacks here
    for (int i = 0; i < _app->get_params().state_options.num_cameras; i++) {
      // create subscriber
      subs_cam.push_back(_nh->subscribe<sensor_msgs::Image>(cam_topic, 10, boost::bind(&ROS1Visualizer::callback_monocular, this, _1, i)));
    }
  }
}
```

차트 흐름에 따라 자세히 보기

- void ROS1Visualizer::callback_inertial()
  - ros topic 을 ov_core::ImuData / CameraData 에 담음.
  - 모든 카메라 topic 들어오면 VioManager 로 전달.
  - OpenVINS 에서는 데이터를 전달하는 함수의 경우 feed_<SOMETHING>()
  - 데이터를 받아올 때는 get_<SOMETHING>() 의 형태로 되어있음.

```
1  // convert into correct format
2  ov_core::ImuData message;
3  message.timestamp = msg->header.stamp.toSec();
4  message.wm << msg->angular_velocity.x, msg->angular_velocity.y, msg->angular_velocity.z;
5  message.am << msg->linear_acceleration.x, msg->linear_acceleration.y, msg->linear_acceleration.z;
6
7  // send it to our VIO system
8  _app->feed_measurement_imu(message);
9
10 // ... 중략
11
12     while (!camera_queue.empty() && camera_queue.at(0).timestamp < timestamp_imu_inC) {
13       double update_dt = 100.0 * (timestamp_imu_inC - camera_queue.at(0).timestamp);
14       _app->feed_measurement_camera(camera_queue.at(0));
```

```
1  // Get our good MSCKF features
2  std::vector<Eigen::Vector3d> feats_msckf = _app->get_good_features_MSCKF();
3  sensor_msgs::PointCloud2 cloud = ROSVisualizerHelper::get_ros_pointcloud(feats_msckf);
4  pub_points_msckf.publish(cloud);
5
6  // Get our good SLAM features
7  std::vector<Eigen::Vector3d> feats_slam = _app->get_features_SLAM();
8  sensor_msgs::PointCloud2 cloud_SLAM = ROSVisualizerHelper::get_ros_pointcloud(feats_slam);
9  pub_points_slam.publish(cloud_SLAM);
```

차트 흐름에 따라 자세히 보기

- trackFEATS 객체 선언
  - VioManager 클래스 생성자에서 선언.
  - 파라미터 YAML 파일에서 use_klt 로 구분.

```
1   int init_max_features = std::floor((double)params.init_options.init_max_features /
(double)params.state_options.num_cameras);
2   if (params.use_klt) {
3     trackFEATS = std::shared_ptr<TrackBase>(new TrackKLT(state->_cam_intrinsics_cameras, init_max_features,
4                                                          state->_options.max_aruco_features,
5                                                          params.use_stereo, params.histogram_method,
6                                                          params.fast_threshold, params.grid_x, params.grid_y,
7                                                          params.min_px_dist));
8   } else {
9     trackFEATS = std::shared_ptr<TrackBase>(new TrackDescriptor(
10        state->_cam_intrinsics_cameras, init_max_features, state->_options.max_aruco_features,
11        params.use_stereo, params.histogram_method,
12        params.fast_threshold, params.grid_x, params.grid_y, params.min_px_dist, params.knn_ratio));
13  }
```

- state->_cam_intrinsics_cameras: camera calibration object which has all camera intrinsics in it
- init_max_features: number of features we want want to track (i.e. track 200 points from frame to frame)
- state->_options.max_aruco_features: the max id of the aruco tags, so we ensure that we start our non-auroc features above this value
- use_stereo: if we should do stereo feature tracking or binocular
- histogram_method: what type of histogram pre-processing should be done (histogram eq?)
- fast_threshold: FAST detection threshold
- grid_x,grid_y: size of grid in the x-direction / u-direction, the y-direction / v-direction
- min_px_dist: features need to be at least this number pixels away from each other
- knnratio: matching ratio needed (smaller value forces top two descriptors during match to be more different)

차트 흐름에 따라 자세히 보기

- void VioManager::track_image_and_update()
  - CameraData 에러 검출 코드 제외
  - Downsampling 파트 제외
    - 기본 파라미터 false
    - true 인 경우 half 로 되도록 하드코딩되어있음.
  - zero velocity update(ZUPT) 파트 제외
    - 기본 파라미터 false
    - 정지한 경우, monocular 에서 feature triangulation, system update 가 불가하므로 이런 경우에 유용하다고 함[1].
  - 따라서 전체적인 알고리즘이 여기 함수에서 이뤄진다고 보면 됨.
    - Feature extraction / tracking / update
    - Initialization
    - Propagation

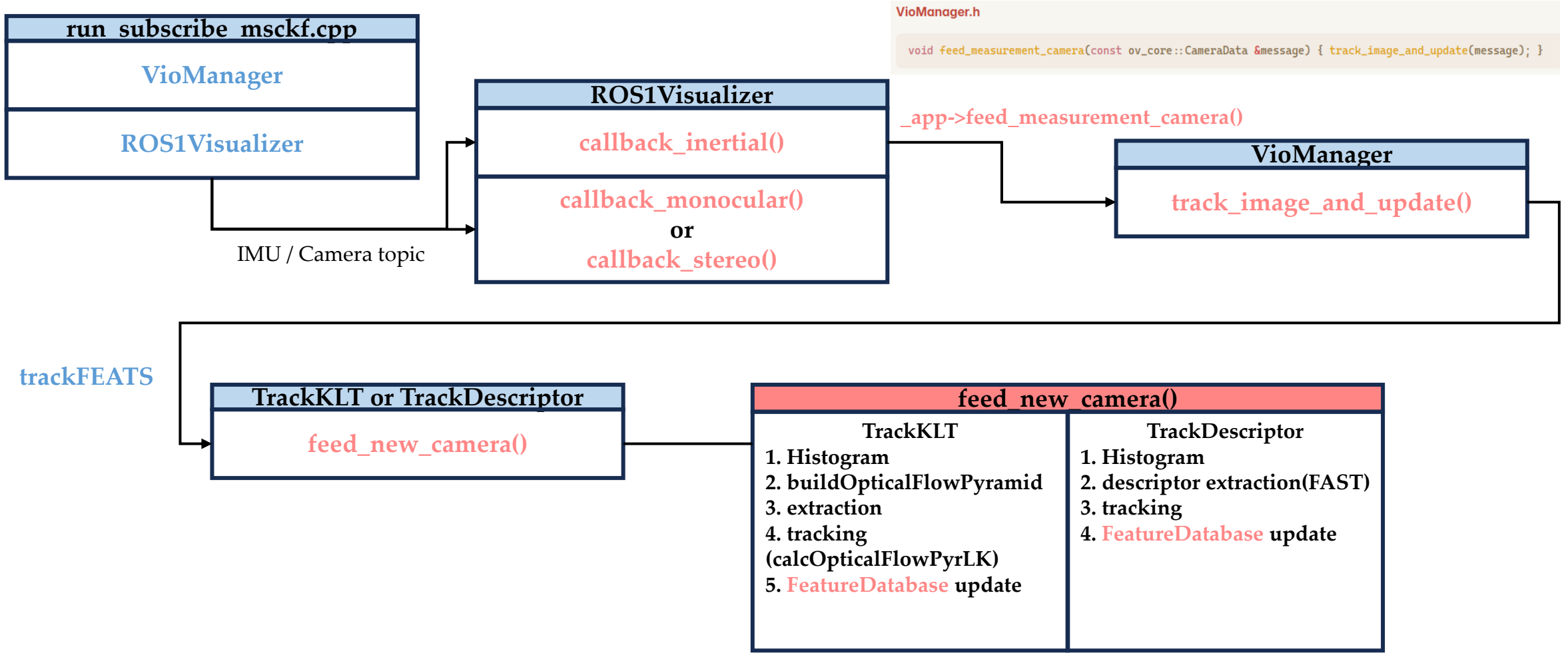
```
1  void VioManager::track_image_and_update(const ov_core::CameraData &message_const) {
2    // Perform our feature tracking!
3    trackFEATS->feed_new_camera(message);
4
5    // If we do not have VIO initialization, then try to initialize
6    // TODO: Or if we are trying to reset the system, then do that here!
7    if (!is_initialized_vio) {
8      is_initialized_vio = try_to_initialize(message);
9      if (!is_initialized_vio) {
10       double time_track = (rT2 - rT1).total_microseconds() * 1e-6;
11       PRINT_DEBUG(BLUE "[TIME]: %.4f seconds for tracking\n" RESET, time_track);
12       return;
13     }
14   }
15
16   // Call on our propagate and update function
17   do_feature_propagate_update(message);
18 }
```

For more details about ZUPT, please refer [1] https://docs.openvins.com/update-zerovelocity.html

BLUE: 클래스/객체
PINK: 함수명

Code Analysis

- trackFEATS 객체 까지의 플로우차트



Feature Tracking 1. Optical Flow

- void TrackKLT::feed_new_camera()
  - 에러 검출 / print 문 / 쓰레드 제외
  - Histogram equalization
    - 이미지의 균일한 밝기를 위해 OpenCV 함수 사용.
    - 전체 이미지를 균일화 하는 기본적인 방식
    - 구역분할, 노이즈 방지 기법이 포함되어 있는 CLAHE 방식을 두 가지를 지원.[1]
    - 기본 파라미터 HISTOGRAM
  - Image pyramid
    - win_size = cv::Size(15, 15)
    - pyr_levels = 5
    - 로 헤더파일에 고정되어 있음 *(Why?)*
    - (다른 레포[2]에서는 21x21,3 사용. 원본 논문[3]도 4 이상 쓸 필요가 없다고 함.)
  - 카메라 종류, 개수에 따라 함수사용
    - monocular / stereo
    - binocular: 서로 다른 카메라

```
1 void TrackKLT::feed_new_camera(const CameraData &message) {
2   size_t num_images = message.images.size();
3   for (size_t msg_id = 0; msg_id < num_images; msg_id++) {
4     // Histogram equalize
5     cv::Mat img;
6     if (histogram_method == HistogramMethod::HISTOGRAM) {
7       cv::equalizeHist(message.images.at(msg_id), img);
8     } else if (histogram_method == HistogramMethod::CLAHE) {
9       double eq_clip_limit = 10.0;
10      cv::Size eq_win_size = cv::Size(8, 8);
11      cv::Ptr<cv::CLAHE> clahe = cv::createCLAHE(eq_clip_limit, eq_win_size);
12      clahe->apply(message.images.at(msg_id), img);
13    } else {
14      img = message.images.at(msg_id);
15    }
16
17    // Extract image pyramid
18    std::vector<cv::Mat> imgpyr;
19    cv::buildOpticalFlowPyramid(img, imgpyr, win_size, pyr_levels);
20
21    // Save!
22    img_curr[cam_id] = img;
23    img_pyramid_curr[cam_id] = imgpyr;
24  }
25
26  // Either call our stereo or monocular version
27  // If we are doing binocular tracking, then we should parallize our tracking
28  if (num_images == 1) {
29    feed_monocular(message, 0);
30  } else if (num_images == 2 && use_stereo) {
31    feed_stereo(message, 0, 1);
32  } else if (!use_stereo) {
33    parallel_for_(cv::Range(0, (int)num_images), LambdaBody([&](const cv::Range &range) {
34                    for (int i = range.start; i < range.end; i++) {
35                      feed_monocular(message, i);
36                    }
37    }));
38  } else {
39    std::exit(EXIT_FAILURE);
40  }
41 }
```


[1] https://opencv-python.readthedocs.io/en/latest/doc/20.imageHistogramEqualization/imageHistogramEqualization.html
[2] https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/include/openpose_private/tracking/pyramidalLK.hpp
[3] Bouguet, J.-Y. "Pyramidal implementation of the lucas kanade feature tracker." (1999). http://robots.stanford.edu/cs223b04/algo_tracking.pdf

Feature Tracking 1. Optical Flow

- void TrackKLT::feed_new_camera()
  - 에러 검출 / print 문 / 쓰레드 제외
  - Histogram equalization
    - 이미지의 균일한 밝기를 위해 OpenCV 함수 사용.
    - 정해 이미지를 구현할 수는 기본적인 방식

parallel_for 은 OpenCV 에서 제공하는 병렬처리함수.
Cmake 옵션에 따라 제공하는 몇가지 최적화 방식을 선택적으로 사용 가능.

https://docs.opencv.org/3.4/d7/dff/tutorial_how_to_use_OpenCV_parallel_for_.html

  - 카메라 종류, 개수에 따라 함수사용
    - monocular / stereo
    - binocular: 서로 다른 카메라

```
1 void TrackKLT::feed_new_camera(const CameraData &message) {
2   size_t num_images = message.images.size();
3   for (size_t msg_id = 0; msg_id < num_images; msg_id++) {
4     // Histogram equalize
5     cv::Mat img;
6     if (histogram_method == HistogramMethod::HISTOGRAM) {
7       cv::equalizeHist(message.images.at(msg_id), img);
8     } else if (histogram_method == HistogramMethod::CLAHE) {
9       double eq_clip_limit = 10.0;
10      cv::Size eq_win_size = cv::Size(8, 8);
11      cv::Ptr<cv::CLAHE> clahe = cv::createCLAHE(eq_clip_limit, eq_win_size);
12      clahe->apply(message.images.at(msg_id), img);
13    } else {
14      img = message.images.at(msg_id);
15    }
16
17    // Extract image pyramid
18    std::vector<cv::Mat> imgpyr;
19    cv::buildOpticalFlowPyramid(img, imgpyr, win_size, pyr_levels);
20
21    // Save!
22    img_curr[cam_id] = img;
23    img_pyramid_curr[cam_id] = imgpyr;
24  }
25
26  // Either call our stereo or monocular version
27  // If we are doing binocular tracking, then we should parallize our tracking
28  if (num_images == 1) {
29    feed_monocular(message, 0);
30  } else if (num_images == 2 && use_stereo) {
31    feed_stereo(message, 0, 1);
32  } else if (!use_stereo) {
33    parallel_for_(cv::Range(0, (int)num_images), LambdaBody([&](const cv::Range &range) {
34                  for (int i = range.start; i < range.end; i++) {
35                    feed_monocular(message, i);
36                  }
37    }));
38  } else {
39    std::exit(EXIT_FAILURE);
40  }
41 }
```


[1] https://opencv-python.readthedocs.io/en/latest/doc/20.imageHistogramEqualization/imageHistogramEqualization.html
[2] https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/include/openpose_private/tracking/pyramidalLK.hpp
[3] Bouguet, J.-Y. "Pyramidal implementation of the lucas kanade feature tracker." (1999). http://robots.stanford.edu/cs223b04/algo_tracking.pdf

Feature Tracking 1. Optical Flow

- void TrackKLT::feed_monocular()
  - feed_stereo() 와 left right 인덱싱해서 처리하는 것만 상이하므로 monocular 로 코드리뷰.
  - 쓰레드 / print문 제외.
  - Good feature 가 없는 경우 처음 extraction 한 것으로 초기화.
  - perform_detection_<cameraType>() (추후설명) 으로 feature extraction.
  - perform_matching() (추후설명) 으로 tracking.
    - calcOpticalFlowPyrLK 가 여기 안에서 호출.
  - 얻어낸 mask 를 이용. 모든 이미지를 iteration 하여 해당하는 Keypoint 를 찾아 FeatureDatabase 업데이트

```
1  void TrackKLT::feed_monocular(const CameraData &message, size_t msg_id) {
2    // First we should make that the last images have enough features so we can do KLT
3    // This will "top-off" our number of tracks so always have a constant number
4    int pts_before_detect = (int)pts_last[cam_id].size();
5    auto pts_left_old = pts_last[cam_id];
6    auto ids_left_old = ids_last[cam_id];
7    perform_detection_monocular(img_pyramid_last[cam_id], img_mask_last[cam_id], pts_left_old, ids_left_old);
8
9    // Our return success masks, and predicted new features
10   std::vector<uchar> mask_ll;
11   std::vector<cv::KeyPoint> pts_left_new = pts_left_old;
12
13   // Lets track temporally
14   perform_matching(img_pyramid_last[cam_id], imgpyr, pts_left_old, pts_left_new, cam_id, cam_id, mask_ll);
15   assert(pts_left_new.size() == ids_left_old.size());
16
17   // If any of our mask is empty, that means we didn't have enough to do ransac, so just return
18   if (mask_ll.empty()) {
19     std::lock_guard<std::mutex> lckv(mtx_last_vars);
20     img_last[cam_id] = img;
21     img_pyramid_last[cam_id] = imgpyr;
22     img_mask_last[cam_id] = mask;
23     pts_last[cam_id].clear();
24     ids_last[cam_id].clear();
25     PRINT_ERROR(RED "[KLT-EXTRACTOR]: Failed to get enough points to do RANSAC, resetting.....\n" RESET);
26     return;
27   }
28
29   // Get our "good tracks"
30   std::vector<cv::KeyPoint> good_left;
31   std::vector<size_t> good_ids_left;
32
33   // Loop through all left points
34   for (size_t i = 0; i < pts_left_new.size(); i++) {
35     // Ensure we do not have any bad KLT tracks (i.e., points are negative)
36     if (pts_left_new.at(i).pt.x < 0 || pts_left_new.at(i).pt.y < 0 || (int)pts_left_new.at(i).pt.x >= img.cols ||
37         (int)pts_left_new.at(i).pt.y >= img.rows)
38       continue;
39     // Check if it is in the mask
40     // NOTE: mask has max value of 255 (white) if it should be
41     if ((int)message.masks.at(msg_id).at<uint8_t>((int)pts_left_new.at(i).pt.y, (int)pts_left_new.at(i).pt.x) > 127)
42       continue;
43     // If it is a good track, and also tracked from left to right
44     if (mask_ll[i]) {
45       good_left.push_back(pts_left_new[i]);
46       good_ids_left.push_back(ids_left_old[i]);
47     }
48   }
49
50   // Update our feature database, with theses new observations
51   for (size_t i = 0; i < good_left.size(); i++) {
52     cv::Point2f npt_l = camera_calib.at(cam_id)->undistort_cv(good_left.at(i).pt);
53     database->update_feature(good_ids_left.at(i), message.timestamp, cam_id, good_left.at(i).pt.x, good_left.at(i).pt.y,
   npt_l.x, npt_l.y);
54   }
55
56   // Move forward in time
57   {
58     std::lock_guard<std::mutex> lckv(mtx_last_vars);
59     img_last[cam_id] = img;
60     img_pyramid_last[cam_id] = imgpyr;
61     img_mask_last[cam_id] = mask;
62     pts_last[cam_id] = good_left;
63     ids_last[cam_id] = good_ids_left;
64   }
65 }
```

Feature Tracking 1. Optical Flow

- void TrackKLT::feed_stereo()
  - 쓰레드 / print문 제외.
  - 1. parallel_for 사용. left=0, right=1 이므로 고정되어 있으므로 cv::Range(0, 2)로 하드코딩하고 번갈아가면서 perform_matching() 으로 할당
  - 또한, tracking 을 좌안/우안에 대해 각각 수행.
  - 2. 좌안/우안에서 같은 포인트 서칭. left=i, right=index_right.
  - 3. id, Keypoint 할당할 때,
    - 1. 양 쪽에 다 있는 것들
    - 2. 왼쪽에만 있는 것 (mono feature)
    - 3. 오른쪽에만 있는 것
  - 순서로 넣어줌.

```
std::vector<uchar> mask_ll, mask_rr;
std::vector<cv::KeyPoint> pts_left_new = pts_left_old;
std::vector<cv::KeyPoint> pts_right_new = pts_right_old;

std::vector<cv::KeyPoint> good_left, good_right;
std::vector<size_t> good_ids_left, good_ids_right;
```

```
// ... 선택
// Our return success masks, and predicted new features
std::vector<uchar> mask_ll, mask_rr;
std::vector<cv::KeyPoint> pts_left_new = pts_left_old;
std::vector<cv::KeyPoint> pts_right_new = pts_right_old;

// Lets track temporally
parallel_for_(cv::Range(0, 2), LambdaBody([&](const cv::Range &range) {
                for (int i = range.start; i < range.end; i++) {
                  bool is_left = (i == 0);
                  perform_matching(img_pyramid_last[is_left ? cam_id_left : cam_id_right], is_left ? imgpyr_left :
imgpyr_right,
                                   is_left ? pts_left_old : pts_right_old, is_left ? pts_left_new : pts_right_new,
                                   is_left ? cam_id_left : cam_id_right, is_left ? cam_id_left : cam_id_right,
                                   is_left ? mask_ll : mask_rr);
                }
              }));

// ... 중략
// Loop through all left points
for (size_t i = 0; i < pts_left_new.size(); i++) {
  // Ensure we do not have any bad KLT tracks (i.e., points are negative)
  if (pts_left_new.at(i).pt.x < 0 || pts_left_new.at(i).pt.y < 0 || (int)pts_left_new.at(i).pt.x > img_left.cols ||
      (int)pts_left_new.at(i).pt.y > img_left.rows)
    continue;
  // See if we have the same feature in the right
  bool found_right = false;
  size_t index_right = 0;
  for (size_t n = 0; n < ids_right_old.size(); n++) {
    if (ids_left_old.at(i) == ids_right_old.at(n)) {
      found_right = true;
      index_right = n;
      break;
    }
  }

  // If it is a good track, and also tracked from right to right
  if (mask_ll[i] && found_right && mask_rr[index_right]) {
    // Ensure we do not have any bad KLT tracks (i.e., points are negative)
    if (pts_right_new.at(index_right).pt.x < 0 || pts_right_new.at(index_right).pt.y < 0 ||
        (int)pts_right_new.at(index_right).pt.x >= img_right.cols || (int)pts_right_new.at(index_right).pt.y >=
img_right.rows)
      continue;
    good_left.push_back(pts_left_new.at(i));
    good_right.push_back(pts_right_new.at(index_right));
    good_ids_left.push_back(ids_left_old.at(i));
    good_ids_right.push_back(ids_right_old.at(index_right));
    // PRINT_DEBUG("adding to stereo - %u , %u\n", ids_left_old.at(i), ids_right_old.at(index_right));
  } else if (mask_ll[i]) {
    good_left.push_back(pts_left_new.at(i));
    good_ids_left.push_back(ids_left_old.at(i));
    // PRINT_DEBUG("adding to left - %u \n",ids_left_old.at(i));
  }
}

// Loop through all right points
for (size_t i = 0; i < pts_right_new.size(); i++) {
  // Ensure we do not have any bad KLT tracks (i.e., points are negative)
  if (pts_right_new.at(i).pt.x < 0 || pts_right_new.at(i).pt.y < 0 || (int)pts_right_new.at(i).pt.x >= img_right.cols ||
      (int)pts_right_new.at(i).pt.y >= img_right.rows)
    continue;
  // See if we have the same feature in the right
  bool added_already = (std::find(good_ids_right.begin(), good_ids_right.end(), ids_right_old.at(i)) !=
good_ids_right.end());
  // If it has not already been added as a good feature, add it as a mono track
  if (mask_rr[i] && !added_already) {
    good_right.push_back(pts_right_new.at(i));
    good_ids_right.push_back(ids_right_old.at(i));
    // PRINT_DEBUG("adding to right - %u \n", ids_right_old.at(i));
  }
}

// Update our feature database, with theses new observations
for (size_t i = 0; i < good_left.size(); i++) {
  cv::Point2f npt_l = camera_calib.at(cam_id_left)->undistort_cv(good_left.at(i).pt);
  database->update_feature(good_ids_left.at(i), message.timestamp, cam_id_left, good_left.at(i).pt.x,
good_left.at(i).pt.y, npt_l.x,
                           npt_l.y);
}
for (size_t i = 0; i < good_right.size(); i++) {
  cv::Point2f npt_r = camera_calib.at(cam_id_right)->undistort_cv(good_right.at(i).pt);
  database->update_feature(good_ids_right.at(i), message.timestamp, cam_id_right, good_right.at(i).pt.x,
good_right.at(i).pt.y, npt_r.x,
                           npt_r.y);
}
```

1

2

3

Feature Tracking 1. Optical Flow

- void TrackKLT::perform_detection_monocular()
  - 특징점 추출을 위해 사용하지 않는 부분 처리
    - 이미지 상하좌우 10 pixel 부분 제외
    - Downscale 에서 같은 부분에 해당하는 부분 제외
    - 그리드 계산해서 벗어나는 부분 제외
    - 4번에서 x_close, y_close, x_grid, y_grid 를 모두 사용해서 밝기 할당을 하는 것으로 보이는데 잘 이해는 못하였음.
    - 마지막으로 해당하는 이미지에 사각형 그려줌. *(Why?)*

```
void TrackKLT::perform_detection_monocular(const std::vector<cv::Mat> &img0pyr, const cv::Mat &mask0,
std::vector<cv::KeyPoint> &pts0,
                                           std::vector<size_t> &ids0) {

  // Create a 2D occupancy grid for this current image
  // Note that we scale this down, so that each grid point is equal to a set of pixels
  // This means that we will reject points that less than grid_px_size points away then existing features
  cv::Size size_close((int)((float)img0pyr.at(0).cols / (float)min_px_dist),
                      (int)((float)img0pyr.at(0).rows / (float)min_px_dist)); // width x height
  cv::Mat grid_2d_close = cv::Mat::zeros(size_close, CV_8UC1);
  float size_x = (float)img0pyr.at(0).cols / (float)grid_x;
  float size_y = (float)img0pyr.at(0).rows / (float)grid_y;
  cv::Size size_grid(grid_x, grid_y); // width x height
  cv::Mat grid_2d_grid = cv::Mat::zeros(size_grid, CV_8UC1);
  cv::Mat mask0_updated = mask0.clone();
  auto it0 = pts0.begin();
  auto it1 = ids0.begin();
  while (it0 != pts0.end()) {
    // Get current left keypoint, check that it is in bounds
    cv::KeyPoint kpt = *it0;
    int x = (int)kpt.pt.x;
    int y = (int)kpt.pt.y;                                                                    1
    int edge = 10;
    if (x < edge || x >= img0pyr.at(0).cols - edge || y < edge || y >= img0pyr.at(0).rows - edge) {
      it0 = pts0.erase(it0);
      it1 = ids0.erase(it1);
      continue;
    }
    // Calculate mask coordinates for close points
    int x_close = (int)(kpt.pt.x / (float)min_px_dist);
    int y_close = (int)(kpt.pt.y / (float)min_px_dist);                                       2
    if (x_close < 0 || x_close >= size_close.width || y_close < 0 || y_close >= size_close.height) {
      it0 = pts0.erase(it0);
      it1 = ids0.erase(it1);
      continue;
    }
    // Calculate what grid cell this feature is in
    int x_grid = std::floor(kpt.pt.x / size_x);
    int y_grid = std::floor(kpt.pt.y / size_y);                                               3
    if (x_grid < 0 || x_grid >= size_grid.width || y_grid < 0 || y_grid >= size_grid.height) {
      it0 = pts0.erase(it0);
      it1 = ids0.erase(it1);
      continue;
    }
    // Check if this keypoint is near another point
    if (grid_2d_close.at<uint8_t>(y_close, x_close) > 127) {                                  4
      it0 = pts0.erase(it0);
      it1 = ids0.erase(it1);
      continue;
    }
    // Now check if it is in a mask area or not
    // NOTE: mask has max value of 255 (white) if it should be
    if (mask0.at<uint8_t>(y, x) > 127) {
      it0 = pts0.erase(it0);
      it1 = ids0.erase(it1);
      continue;
    }
    // Else we are good, move forward to the next point
    grid_2d_close.at<uint8_t>(y_close, x_close) = 255;
    if (grid_2d_grid.at<uint8_t>(y_grid, x_grid) < 255) {
      grid_2d_grid.at<uint8_t>(y_grid, x_grid) += 1;
    }
    // Append this to the local mask of the image
    if (x - min_px_dist >= 0 && x + min_px_dist < img0pyr.at(0).cols && y - min_px_dist >= 0 && y + min_px_dist
< img0pyr.at(0).rows) {
      cv::Point pt1(x - min_px_dist, y - min_px_dist);
      cv::Point pt2(x + min_px_dist, y + min_px_dist);
      cv::rectangle(mask0_updated, pt1, pt2, cv::Scalar(255), -1);
    }
    it0++;
    it1++;
  }
```

Feature Tracking 1. Optical Flow

- void TrackKLT::perform_detection_monocular()
  - 필요한 feature 수에 따라 알고리즘 진행.
    - 최소 20개와 0.5 * num_features 중 최솟값으로 선택.
    - num_features 는 try_to_initialize() 에서 파라미터 YAML 에 따라 설정
      - VioManagerHelper.cpp LN#124
      - std::floor(num_pts / num_cameras)
    - 그리드에 필요한 feature 수를 가지고 Grider_GRID::perform_griding() 수행
    - *왜 cv::goodFeaturesToTrack() 쓰지 않는지?*
- void TrackKLT::perform_detection_stereo()
  - 전체적인 방식은 동일하여 좌안 -> 우안 순으로 진행.
  - 차이점은 좌안에서 Point 를 찾은 경우, calcOpticalFlowPyrLK() 수행

```
1  // First compute how many more features we need to extract from this image
2  // If we don't need any features, just return
3  double min_feat_percent = 0.50;
4  int num_featsneeded = num_features - (int)pts0.size();
5  if (num_featsneeded < std::min(20, (int)(min_feat_percent * num_features)))
6    return;
7
8  // This is old extraction code that would extract from the whole image
9  // This can be slow as this will recompute extractions for grid areas that we have max features already
10 // std::vector<cv::KeyPoint> pts0_ext;
11 // Grider_FAST::perform_griding(img0pyr.at(0), mask0_updated, pts0_ext, num_features, grid_x, grid_y,
   threshold, true);
12
13 // We also check a downsampled mask such that we don't extract in areas where it is all masked!
14 cv::Mat mask0_grid;
15 cv::resize(mask0, mask0_grid, size_grid, 0.0, 0.0, cv::INTER_NEAREST);
16
17 // Create grids we need to extract from and then extract our features (use fast with griding)
18 int num_features_grid = (int)((double)num_features / (double)(grid_x * grid_y)) + 1;
19 int num_features_grid_req = std::max(1, (int)(min_feat_percent * num_features_grid));
20 std::vector<std::pair<int, int>> valid_locs;
21 for (int x = 0; x < grid_2d_grid.cols; x++) {
22   for (int y = 0; y < grid_2d_grid.rows; y++) {
23     if ((int)grid_2d_grid.at<uint8_t>(y, x) < num_features_grid_req && (int)mask0_grid.at<uint8_t>(y, x) !=
   255) {
24       valid_locs.emplace_back(x, y);
25     }
26   }
27 }
28 std::vector<cv::KeyPoint> pts0_ext;
29 Grider_GRID::perform_griding(img0pyr.at(0), mask0_updated, valid_locs, pts0_ext, num_features, grid_x, grid_y,
   threshold, true);
30
31 // Now, reject features that are close a current feature
32 std::vector<cv::KeyPoint> kpts0_new;
33 std::vector<cv::Point2f> pts0_new;
34 for (auto &kpt : pts0_ext) {
35   // Check that it is in bounds
36   int x_grid = (int)(kpt.pt.x / (float)min_px_dist);
37   int y_grid = (int)(kpt.pt.y / (float)min_px_dist);
38   if (x_grid < 0 || x_grid >= size_close.width || y_grid < 0 || y_grid >= size_close.height)
39     continue;
40   // See if there is a point at this location
41   if (grid_2d_close.at<uint8_t>(y_grid, x_grid) > 127)
42     continue;
43   // Else lets add it!
44   kpts0_new.push_back(kpt);
45   pts0_new.push_back(kpt.pt);
46   grid_2d_close.at<uint8_t>(y_grid, x_grid) = 255;
47 }
48
49 // Loop through and record only ones that are valid
50 // NOTE: if we multi-thread this atomic can cause some randomness due to multiple thread detecting features
51 // NOTE: this is due to the fact that we select update features based on feat id
52 // NOTE: thus the order will matter since we try to select oldest (smallest id) to update with
53 // NOTE: not sure how to remove... maybe a better way?
54 for (size_t i = 0; i < pts0_new.size(); i++) {
55   // update the uv coordinates
56   kpts0_new.at(i).pt = pts0_new.at(i);
57   // append the new uv coordinate
58   pts0.push_back(kpts0_new.at(i));
59   // move id foward and append this new point
60   size_t temp = ++currid;
61   ids0.push_back(temp);
62 }
```

1

2

Feature Tracking 1. Optical Flow

- void TrackKLT::perform_matching()
  - KLT Tracking 이 이루어지는 함수.
  - Point 가 10개 미만인 경우 RANSAC 이 어렵다고 판단하여 return
    - 이전 feed_<cameraType>() 에서도 point 10개 미만으로 인해 perform_matching() 에서 mask 없이 빠져나오면 바로 return 되도록 되어 있음.
  - calcOpticalFlowPyrLK 수행 후, 원본 이미지에서의 Point 로 RANSAC (outlier rejection) 수행.
  - RANSAC 시 2.0 / max_focallength 로 사용하여 pixel 단위로 rejection 이 되도록 함.

```
1  void TrackKLT::perform_matching(const std::vector<cv::Mat> &img0pyr, const std::vector<cv::Mat> &img1pyr,
   std::vector<cv::KeyPoint> &kpts0,
2                                  std::vector<cv::KeyPoint> &kpts1, size_t id0, size_t id1, std::vector<uchar>
   &mask_out) {
3    // Convert keypoints into points (stupid opencv stuff)
4    std::vector<cv::Point2f> pts0, pts1;
5    for (size_t i = 0; i < kpts0.size(); i++) {
6      pts0.push_back(kpts0.at(i).pt);
7      pts1.push_back(kpts1.at(i).pt);
8    }
9
10   // If we don't have enough points for ransac just return empty
11   // We set the mask to be all zeros since all points failed RANSAC
12   if (pts0.size() < 10) {
13     for (size_t i = 0; i < pts0.size(); i++)
14       mask_out.push_back((uchar)0);
15     return;
16   }
17
18   // Now do KLT tracking to get the valid new points
19   std::vector<uchar> mask_klt;
20   std::vector<float> error;
21   cv::TermCriteria term_crit = cv::TermCriteria(cv::TermCriteria::COUNT | cv::TermCriteria::EPS, 30, 0.01);
22   cv::calcOpticalFlowPyrLK(img0pyr, img1pyr, pts0, pts1, mask_klt, error, win_size, pyr_levels, term_crit,
   cv::OPTFLOW_USE_INITIAL_FLOW);
23
24   // Normalize these points, so we can then do ransac
25   // We don't want to do ransac on distorted image uvs since the mapping is nonlinear
26   std::vector<cv::Point2f> pts0_n, pts1_n;
27   for (size_t i = 0; i < pts0.size(); i++) {
28     pts0_n.push_back(camera_calib.at(id0)->undistort_cv(pts0.at(i)));
29     pts1_n.push_back(camera_calib.at(id1)->undistort_cv(pts1.at(i)));
30   }
31
32   // Do RANSAC outlier rejection (note since we normalized the max pixel error is now in the normalized cords)
33   std::vector<uchar> mask_rsc;
34   double max_focallength_img0 = std::max(camera_calib.at(id0)->get_K()(0, 0), camera_calib.at(id0)->get_K()(1, 1));
35   double max_focallength_img1 = std::max(camera_calib.at(id1)->get_K()(0, 0), camera_calib.at(id1)->get_K()(1, 1));
36   double max_focallength = std::max(max_focallength_img0, max_focallength_img1);
37   cv::findFundamentalMat(pts0_n, pts1_n, cv::FM_RANSAC, 2.0 / max_focallength, 0.999, mask_rsc);
38
39   // Loop through and record only ones that are valid
40   for (size_t i = 0; i < mask_klt.size(); i++) {
41     auto mask = (uchar)((i < mask_klt.size() && mask_klt[i] && i < mask_rsc.size() && mask_rsc[i]) ? 1 : 0);
42     mask_out.push_back(mask);
43   }
44
45   // Copy back the updated positions
46   for (size_t i = 0; i < pts0.size(); i++) {
47     kpts0.at(i).pt = pts0.at(i);
48     kpts1.at(i).pt = pts1.at(i);
49   }
50 }
```

Feature Tracking 2. Descriptor

- void TrackDescriptor::feed_new_camera()
  - Descriptor 방식은 KLT Tracker 와 달리 Pyramid 를 사용하지 않으므로, feed_<cameraType>() 에 바로 넣어서 histogram equalization 수행.
- void TrackDescriptor::feed_<cameraType>()
  - 내부 로직 동일.
  - Descriptor 이므로 cv::Mat desc_new 라는 변수가 perform_detection_monocular(), robust_match() 함수에 모두 사용됨.

```
1 void TrackDescriptor::feed_new_camera(const CameraData &message) {
2   // Either call our stereo or monocular version
3   // If we are doing binocular tracking, then we should parallize our tracking
4   size_t num_images = message.images.size();
5   if (num_images == 1) {
6     feed_monocular(message, 0);
7   } else if (num_images == 2 && use_stereo) {
8     feed_stereo(message, 0, 1);
9   } else if (!use_stereo) {
10    parallel_for_(cv::Range(0, (int)num_images), LambdaBody([&](const cv::Range &range) {
11                    for (int i = range.start; i < range.end; i++) {
12                      feed_monocular(message, i);
13                    }
14                  }));
15  } else {
16    std::exit(EXIT_FAILURE);
17  }
18 }
```

Feature Tracking 2. Descriptor

- void TrackDescriptor::perform_detection_monocular()
  - 마찬가지로 feature extraction 수행.
  - Grider_FAST::perform_griding() 으로 수행.(<-> Grider_GRID::perform_griding())
  - OpenCV 의 ORB extractor 사용.
  - Downscale 하여 feature들끼리 가깝지 않게 처리

- perform_detection_stereo()
  - 로직은 monocular() 와 동일.
  - 다만, 좌안과 우안 사이에 robust_match() 함수로 matching 수행. (KLT 와 마찬가지로)

```
void TrackDescriptor::perform_detection_monocular(const cv::Mat &img0, const cv::Mat &mask0, std::vector<cv::KeyPoint> &pts0,
                                                  cv::Mat &desc0, std::vector<size_t> &ids0) {
  // Extract our features (use FAST with griding)
  std::vector<cv::KeyPoint> pts0_ext;
  Grider_FAST::perform_griding(img0, mask0, pts0_ext, num_features, grid_x, grid_y, threshold, true);

  // For all new points, extract their descriptors
  cv::Mat desc0_ext;
  this->orb0->compute(img0, pts0_ext, desc0_ext);

  // Create a 2D occupancy grid for this current image
  // Note that we scale this down, so that each grid point is equal to a set of pixels
  // This means that we will reject points that less then grid_px_size points away then existing features
  cv::Size size((int)((float)img0.cols / (float)min_px_dist), (int)((float)img0.rows / (float)min_px_dist));
  cv::Mat grid_2d = cv::Mat::zeros(size, CV_8UC1);

  // For all good matches, lets append to our returned vectors
  // NOTE: if we multi-thread this atomic can cause some randomness due to multiple thread detecting features
  // NOTE: this is due to the fact that we select update features based on feat id
  // NOTE: thus the order will matter since we try to select oldest (smallest id) to update with
  // NOTE: not sure how to remove... maybe a better way?
  for (size_t i = 0; i < pts0_ext.size(); i++) {
    // Get current left keypoint, check that it is in bounds
    cv::KeyPoint kpt = pts0_ext.at(i);
    int x = (int)kpt.pt.x;
    int y = (int)kpt.pt.y;
    int x_grid = (int)(kpt.pt.x / (float)min_px_dist);
    int y_grid = (int)(kpt.pt.y / (float)min_px_dist);
    if (x_grid < 0 || x_grid >= size.width || y_grid < 0 || y_grid >= size.height || x < 0 || x >= img0.cols || y < 0 || y >= img0.rows) {
      continue;
    }
    // Check if this keypoint is near another point
    if (grid_2d.at<uint8_t>(y_grid, x_grid) > 127)
      continue;
    // Else we are good, append our keypoints and descriptors
    pts0.push_back(pts0_ext.at(i));
    desc0.push_back(desc0_ext.row((int)i));
    // Set our IDs to be unique IDs here, will later replace with corrected ones, after temporal matching
    size_t temp = ++currid;
    ids0.push_back(temp);
    grid_2d.at<uint8_t>(y_grid, x_grid) = 255;
  }
}
```

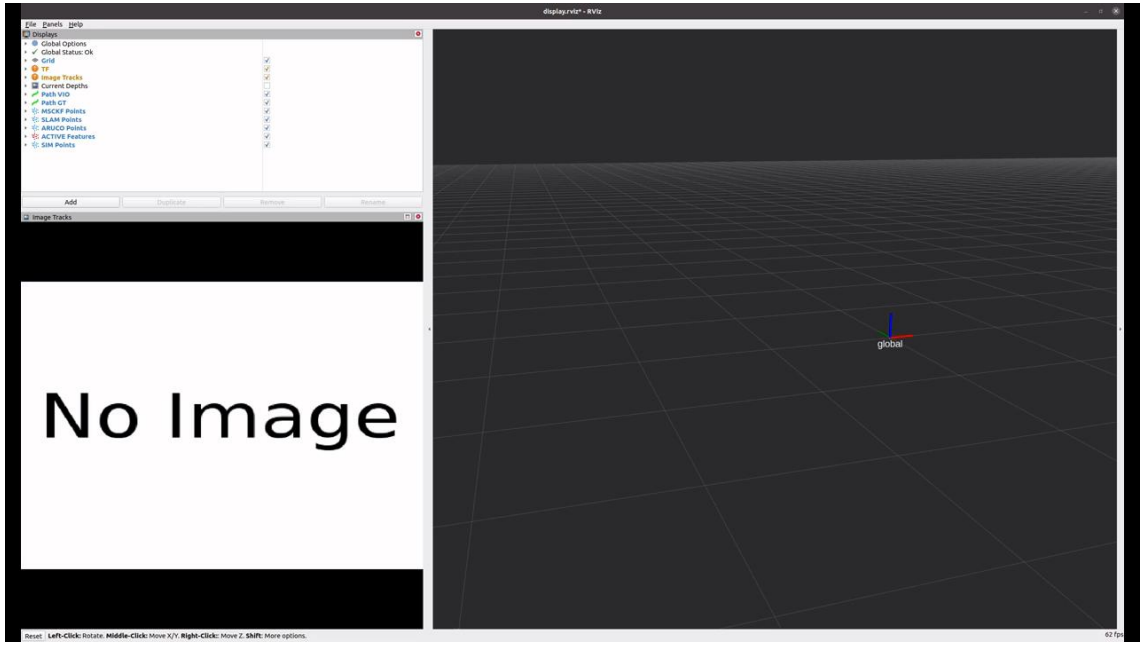
Feature Tracking 2. Descriptor

- void TrackDescriptor::robust_match()
  - 일반적인 descriptor 방식의 tracking.
    - 구체적인 이론내용이나 여타 방식의 구현이 어떤지 공부 필요.
  - 이전 KLT Tracker 와 마찬가지로 RANSAC 기반으로 outlier rejection 수행.
  - 1.0 / focallength 로 사용하여 optical flow 방식보다 보다 타이트하게 outlier rejection 함.

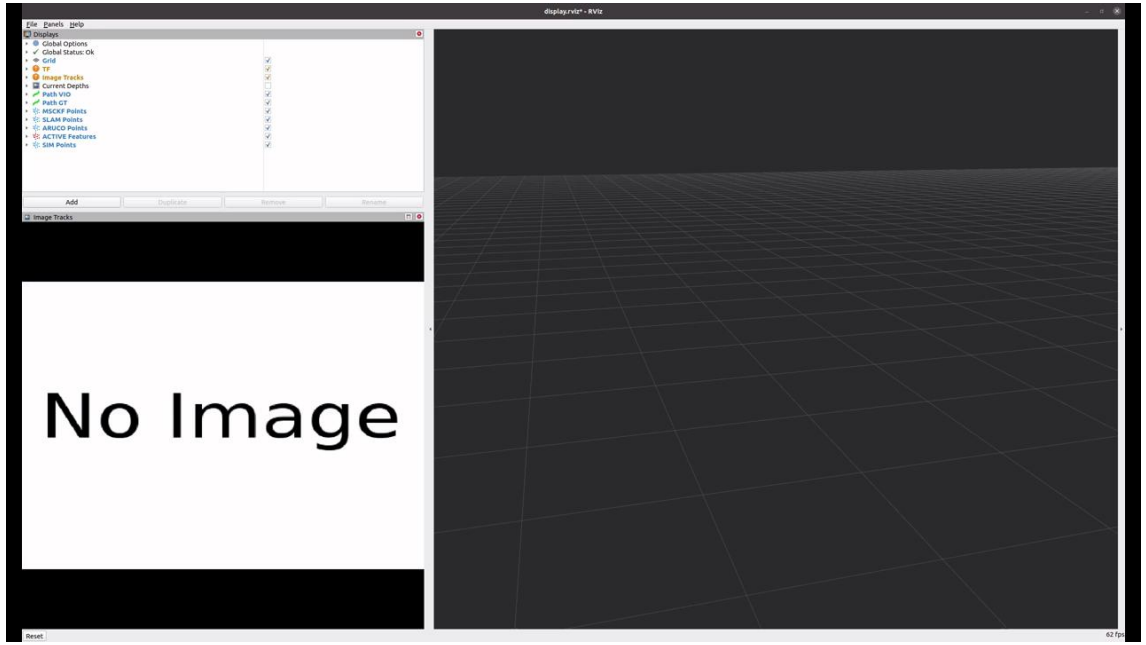
```
void TrackDescriptor::robust_match(const std::vector<cv::KeyPoint> &pts0, const std::vector<cv::KeyPoint> &pts1, const cv::Mat &desc0,
                                   const cv::Mat &desc1, size_t id0, size_t id1, std::vector<cv::DMatch> &matches) {

  // Our 1to2 and 2to1 match vectors
  std::vector<std::vector<cv::DMatch>> matches0to1, matches1to0;

  // Match descriptors (return 2 nearest neighbours)
  matcher->knnMatch(desc0, desc1, matches0to1, 2);
  matcher->knnMatch(desc1, desc0, matches1to0, 2);

  // Do a ratio test for both matches
  robust_ratio_test(matches0to1);
  robust_ratio_test(matches1to0);

  // Finally do a symmetry test
  std::vector<cv::DMatch> matches_good;
  robust_symmetry_test(matches0to1, matches1to0, matches_good);

  // Convert points into points for RANSAC
  std::vector<cv::Point2f> pts0_rsc, pts1_rsc;
  for (size_t i = 0; i < matches_good.size(); i++) {
    // Get our ids
    int index_pt0 = matches_good.at(i).queryIdx;
    int index_pt1 = matches_good.at(i).trainIdx;
    // Push back just the 2d point
    pts0_rsc.push_back(pts0[index_pt0].pt);
    pts1_rsc.push_back(pts1[index_pt1].pt);
  }

  // If we don't have enough points for ransac just return empty
  if (pts0_rsc.size() < 10)
    return;

  // Normalize these points, so we can then do ransac
  // We don't want to do ransac on distorted image uvs since the mapping is nonlinear
  std::vector<cv::Point2f> pts0_n, pts1_n;
  for (size_t i = 0; i < pts0_rsc.size(); i++) {
    pts0_n.push_back(camera_calib.at(id0)->undistort_cv(pts0_rsc.at(i)));
    pts1_n.push_back(camera_calib.at(id1)->undistort_cv(pts1_rsc.at(i)));
  }

  // Do RANSAC outlier rejection (note since we normalized the max pixel error is now in the normalized cords)
  std::vector<uchar> mask_rsc;
  double max_focallength_img0 = std::max(camera_calib.at(id0)->get_K()(0, 0), camera_calib.at(id0)->get_K()(1, 1));
  double max_focallength_img1 = std::max(camera_calib.at(id1)->get_K()(0, 0), camera_calib.at(id1)->get_K()(1, 1));
  double max_focallength = std::max(max_focallength_img0, max_focallength_img1);
  cv::findFundamentalMat(pts0_n, pts1_n, cv::FM_RANSAC, 1 / max_focallength, 0.999, mask_rsc);

  // Loop through all good matches, and only append ones that have passed RANSAC
  for (size_t i = 0; i < matches_good.size(); i++) {
    // Skip if bad ransac id
    if (mask_rsc[i] != 1)
      continue;
    // Else, lets append this match to the return array!
    matches.push_back(matches_good.at(i));
  }
}
```

Visualization

- use_klt: true / false
- num_pts: 150 -> 200
- 이외 파라미터 모두 동일



KLT Tracker



Descriptor

Discussion / Future Works

- 세미나 도중 나온 질문
  - TODO for next week seminar
- extraction 에서 index 처리 하는 부분
- Grider_GRID, Grider_FAST 방식
- Optical flow 방식과 Descriptor 방식의 성능 차이
  - 처음 initialization 이 적게 걸리고, 전반적인 feature 수가 차이남.
  - ov_eval 을 같이 활용해서 성능 평가를 해보아도 좋아보임.
- RViZ 시각화에서 feature tracking 시각화하는 파트

Thanks for your kind attention

chanjoon.park@kaist.ac.kr

Appendix

- void TrackDescriptor::perform_detection_stereo()

```cpp
void TrackDescriptor::perform_detection_stereo(const cv::Mat &img0, const cv::Mat &img1, const cv::Mat &mask0, const cv::Mat &mask1,
                                               std::vector<cv::KeyPoint> &pts0, std::vector<cv::KeyPoint> &pts1, cv::Mat &desc0,
                                               cv::Mat &desc1, size_t cam_id0, size_t cam_id1, std::vector<size_t> &ids0,
                                               std::vector<size_t> &ids1) {

  // Assert that we need features
  assert(pts0.empty());
  assert(pts1.empty());

  // Extract our features (use FAST with griding), and their descriptors
  std::vector<cv::KeyPoint> pts0_ext, pts1_ext;
  cv::Mat desc0_ext, desc1_ext;
  parallel_for_(cv::Range(0, 2), LambdaBody([&](const cv::Range &range) {
                  for (int i = range.start; i < range.end; i++) {
                    bool is_left = (i == 0);
                    Grider_FAST::perform_griding(is_left ? img0 : img1, is_left ? mask0 : mask1, is_left ? pts0_ext : pts1_ext,
                                                 num_features, grid_x, grid_y, threshold, true);
                    (is_left ? orb0 : orb1)->compute(is_left ? img0 : img1, is_left ? pts0_ext : pts1_ext, is_left ? desc0_ext : desc1_ext);
                  }
                }));

  // Do matching from the left to the right image
  std::vector<cv::DMatch> matches;
  robust_match(pts0_ext, pts1_ext, desc0_ext, desc1_ext, cam_id0, cam_id1, matches);

  // Create a 2D occupancy grid for this current image
  // Note that we scale this down, so that each grid point is equal to a set of pixels
  // This means that we will reject points that less then grid_px_size points away then existing features
  cv::Size size0((int)((float)img0.cols / (float)min_px_dist), (int)((float)img0.rows / (float)min_px_dist));
  cv::Mat grid_2d_0 = cv::Mat::zeros(size0, CV_8UC1);
  cv::Size size1((int)((float)img1.cols / (float)min_px_dist), (int)((float)img1.rows / (float)min_px_dist));
  cv::Mat grid_2d_1 = cv::Mat::zeros(size1, CV_8UC1);

  // For all good matches, lets append to our returned vectors
  for (size_t i = 0; i < matches.size(); i++) {

    // Get our ids
    int index_pt0 = matches.at(i).queryIdx;
    int index_pt1 = matches.at(i).trainIdx;

    // Get current left/right keypoint, check that it is in bounds
    cv::KeyPoint kpt0 = pts0_ext.at(index_pt0);
    cv::KeyPoint kpt1 = pts1_ext.at(index_pt1);
    int x0 = (int)kpt0.pt.x;
    int y0 = (int)kpt0.pt.y;
    int x0_grid = (int)(kpt0.pt.x / (float)min_px_dist);
    int y0_grid = (int)(kpt0.pt.y / (float)min_px_dist);
    if (x0_grid < 0 || x0_grid >= size0.width || y0_grid < 0 || y0_grid >= size0.height || x0 < 0 || x0 >= img0.cols || y0 < 0 ||
        y0 >= img0.rows) {
      continue;
    }
    int x1 = (int)kpt1.pt.x;
    int y1 = (int)kpt1.pt.y;
    int x1_grid = (int)(kpt1.pt.x / (float)min_px_dist);
    int y1_grid = (int)(kpt1.pt.y / (float)min_px_dist);
    if (x1_grid < 0 || x1_grid >= size1.width || y1_grid < 0 || y1_grid >= size1.height || x1 < 0 || x1 >= img0.cols || y1 < 0 ||
        y1 >= img0.rows) {
      continue;
    }

    // Check if this keypoint is near another point
    if (grid_2d_0.at<uint8_t>(y0_grid, x0_grid) > 127 || grid_2d_1.at<uint8_t>(y1_grid, x1_grid) > 127)
      continue;

    // Append our keypoints and descriptors
    pts0.push_back(pts0_ext[index_pt0]);
    pts1.push_back(pts1_ext[index_pt1]);
    desc0.push_back(desc0_ext.row(index_pt0));
    desc1.push_back(desc1_ext.row(index_pt1));

    // Set our IDs to be unique IDs here, will later replace with corrected ones, after temporal matching
    size_t temp = ++currid;
    ids0.push_back(temp);
    ids1.push_back(temp);
  }
}
```