OpenVSLAM stereo implementation

Takashi Ohsawa

agenda

- Overview
- Tracking
- Mapping
- Global Optimization
- Initialization
- Relocalization

OVERVIEW

Source code

- https://github.com/MapIV/openvslam/tree/master
  - commit dd8af1cfda9ca112d171f601a87b3af803260708
- 14642 lines in 70 c++ files
- 8234 lines in 82 header files
- Dependencies
  - OpenCV
  - g2o
  - DBoW2 / FBoW

processing flow overview

left_image, right_image, mask
through feed_stereo_frame

Mask can be set. No feature point is taken within the masked regions.

system

tracker_ (main thread)

mapper_ (created thread)

global_optimizer (created thread)

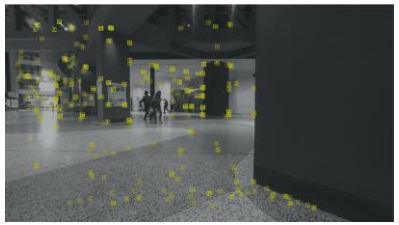
cam_db_

map_db_

bow_db_

frame_publisher_

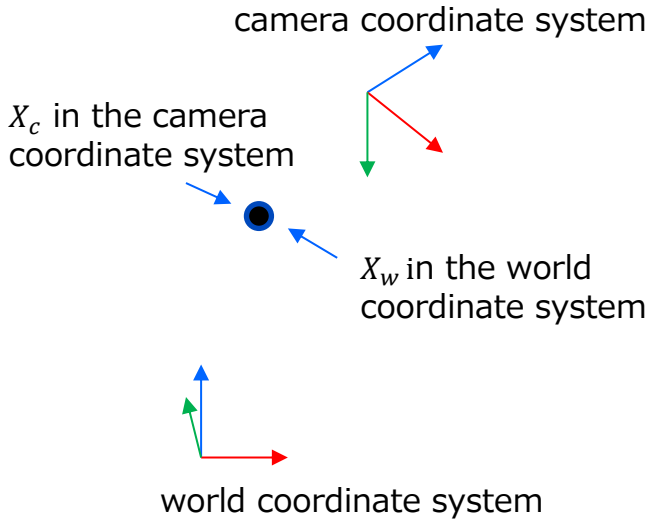
map_publisher_



landmarks, keyframes

cam_pose_cw of Mat44_t type represents the camera pose from the world coordinate system. $\boldsymbol{X_c} = \mathbf{cam_pose_cw} \cdot \boldsymbol{X_w}$ (in other words, transforming the camera coordinate system by cam_pose_cw matches the world coordinate system)

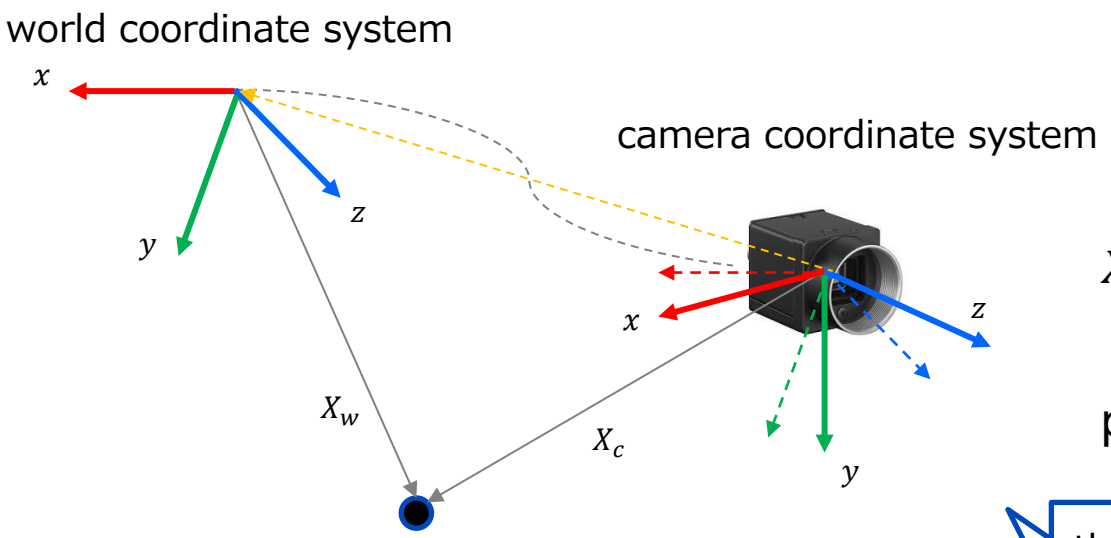
cam_pose_cw
returned by feed_stereo_frame



coordinate system and transforming

the origin of the world coordinate system is the origin of the camera coordinate system at start time

rotating the camera coordinate system by R_{cw} (red, green, blue solid arrows to dotted ones) and then translating it by t_{cw} (dotted orange arrow) matches the world coordinate system.



$X_c = cam_pose_cw \cdot X_w$ where $cam_pose_cw = \begin{pmatrix} R_{cw} & t_{cw} \\ 0 & 1 \end{pmatrix}$, X_c and X_w are 3D points represented by homogeneous coordinate.

the origin of the camera coordinate system is the focal point of the left camera

Map data structure overview

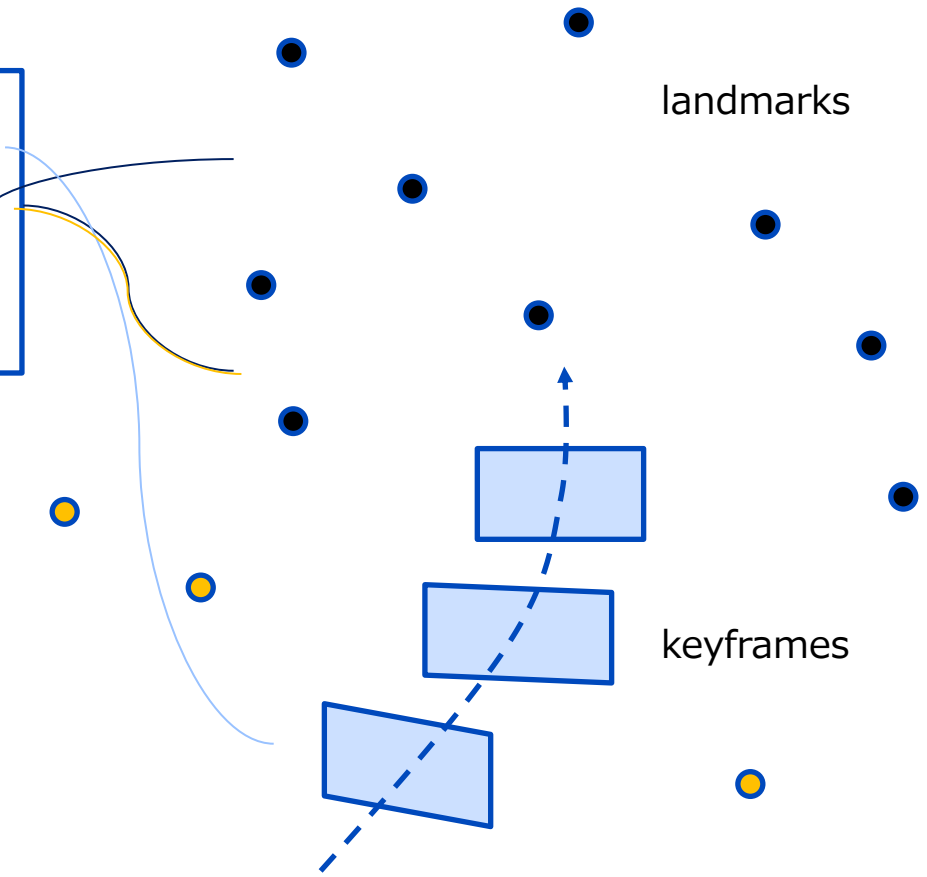
3D points are managed as landmarks with the world coordinate.

map_db_

unordered_map<unsigned int, keyframe*> **keyframes_**

unordered_map<unsigned int, landmark*> **landmarks_**

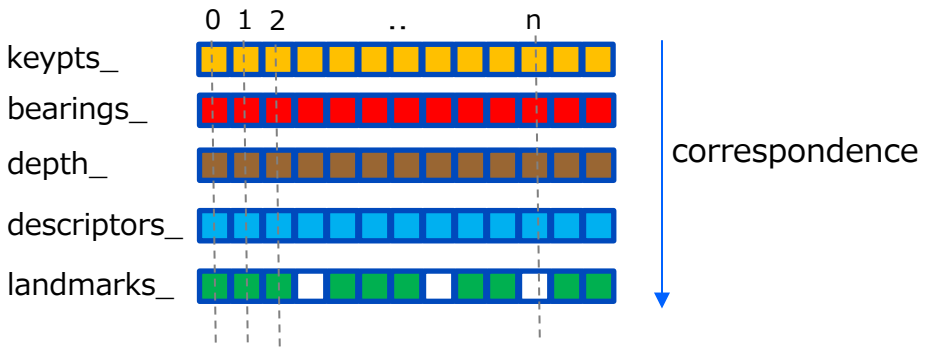
vector<landmark*> **local_landmarks_**

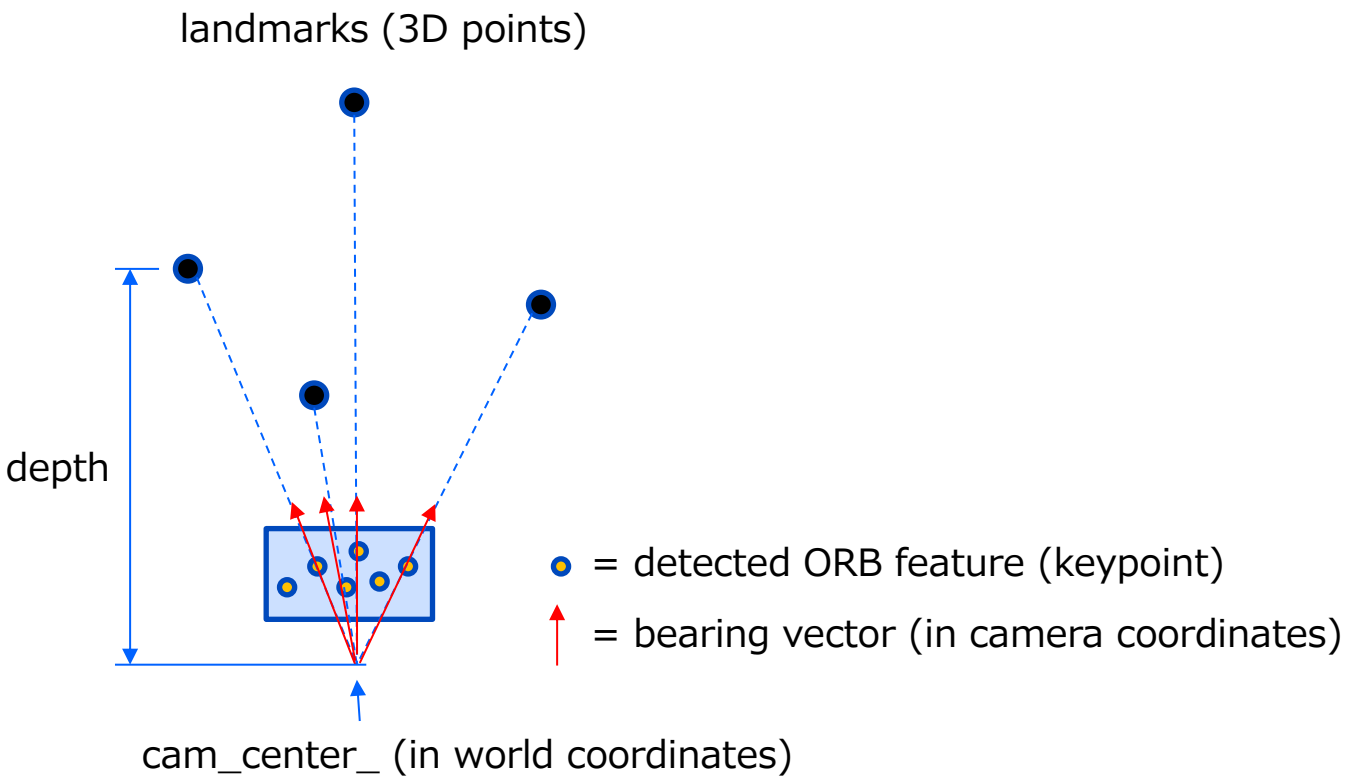


Keyframe data

class keyframe

```
unsigned int id_
unsinged int num_keypts_
vector<cv::KeyPoint> keypts_, undist_keypts_
eigen_alloc_vector<Vec3_t> bearings_
vector<float> depths_
cv::Mat descriptors_
DBoW2::BoWVector bow_vec_
DBoW2::FeatureVector bow_feat_vec_
graph_node graph_node_
Mat44_t cam_pose_cw_, cam_pose_wc_
Vec3_t cam_center_
vector<landmark*> landmarks_
```





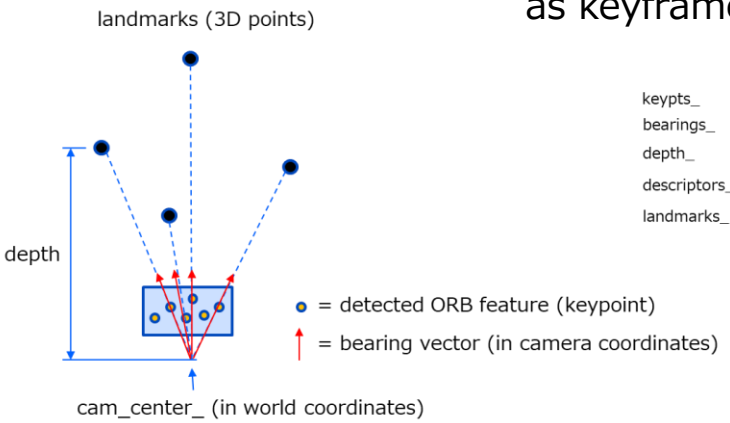
Corresponded keypoint, bearing vector, depth, descriptor and landmark has the same index in keypts_, bearings_, depth_, descriptors_ and landmarks_ respectively.

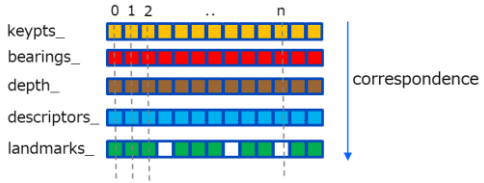
Frame data

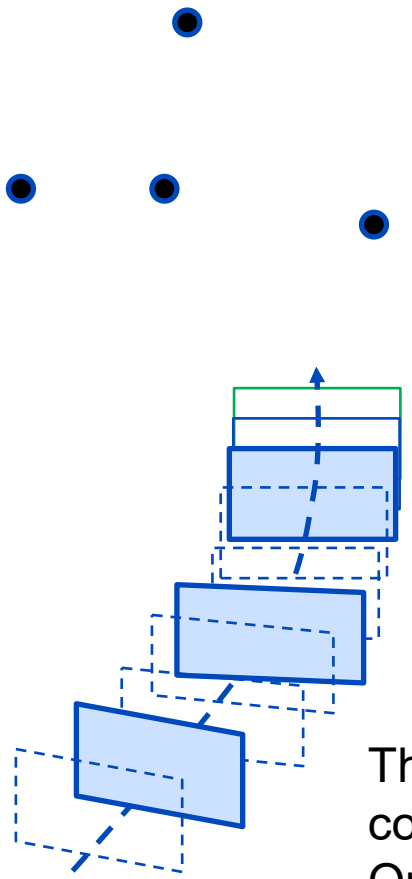
class frame

```
unsigned int id_
unsinged int num_keypts_
vector<cv::KeyPoint> keypts_, undist_keypts_
vector<cv::KeyPoint> keypts_right_
eigen_alloc_vector<Vec3_t> bearings_
vector<float> depths_
cv::Mat descriptors_, descriptors_right_
DBoW2::BoWVector bow_vec_
DBoW2::FeatureVector bow_feat_vec_
Mat44_t cam_pose_cw_
keyframe* ref_keyfrm_
vector<landmark*> landmarks_
```

Similar data structure as keyframe data







= current frame

= last frame

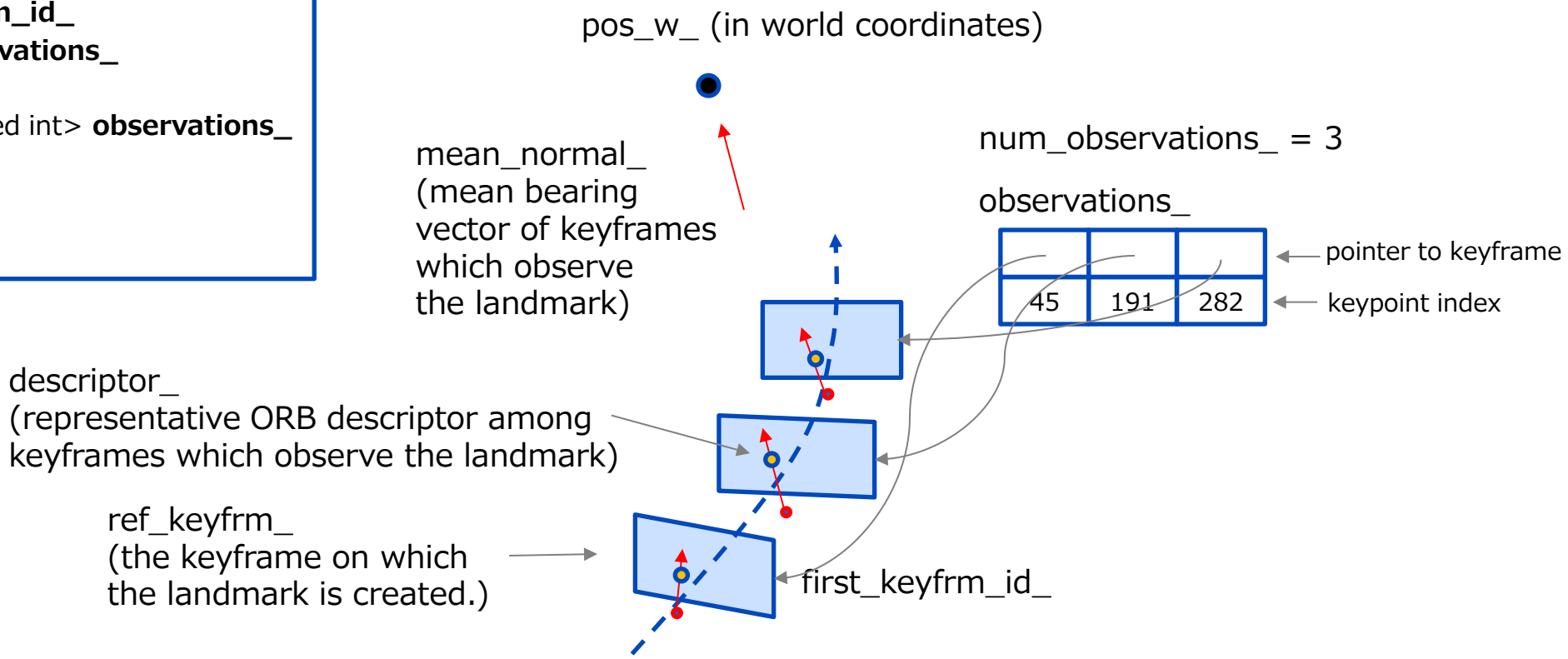
ref_keyfrm_ (which has the highest covisibility with a frame among all keyframes)

There exist frames between keyframes corresponding to every camera image.
Only current and last frames exist at a time.
Past frames are discarded.

Landmark data

class landmark

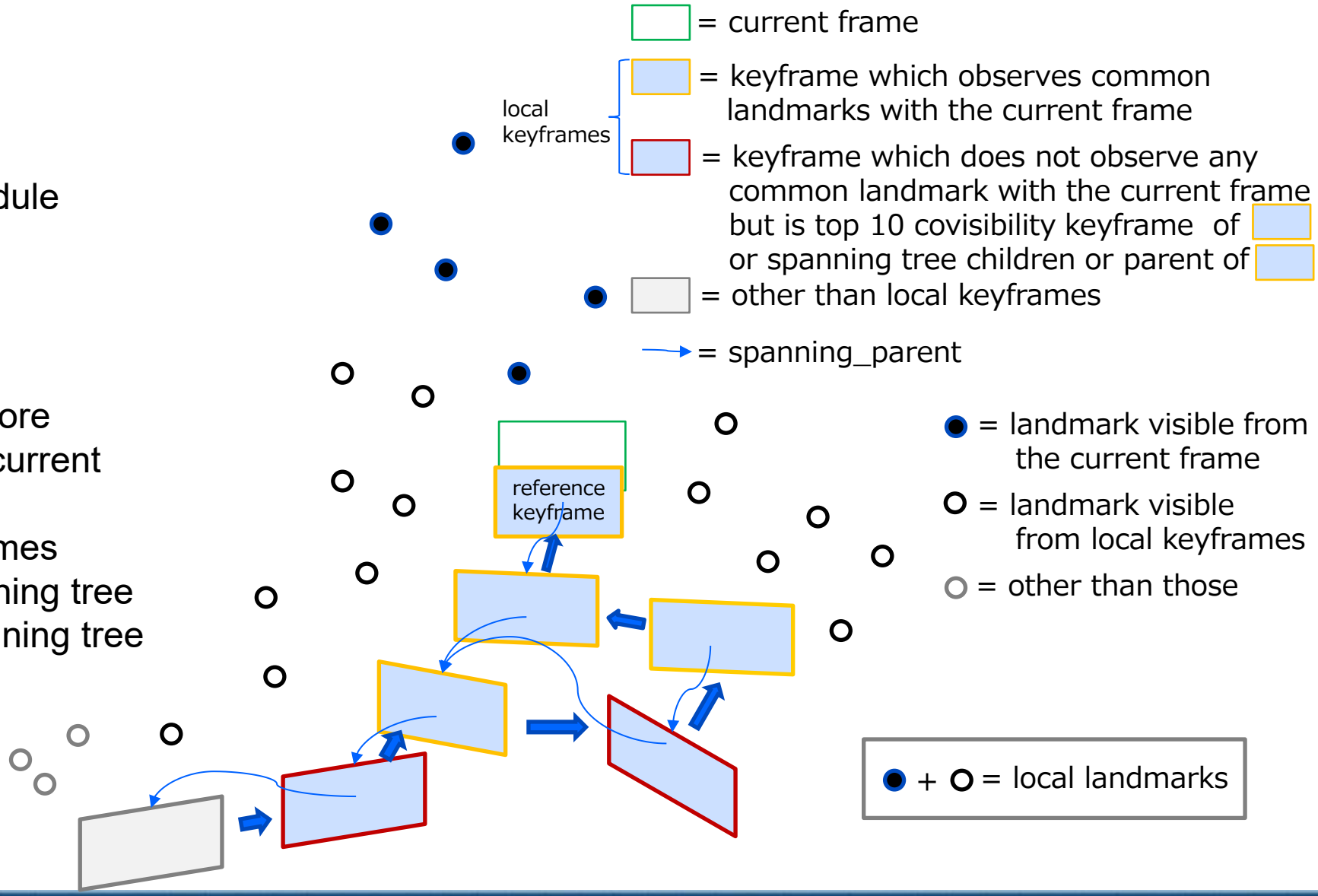
```
unsigned int id_
unsigned int first_keyfrm_id_
unsigned int num_observations_
Vec3_t pos_w_
map<keyframe*, unsigned int> observations_
Vec3_t mean_normal_
cv::Mat descriptor_
keyframe* ref_keyfrm_
```



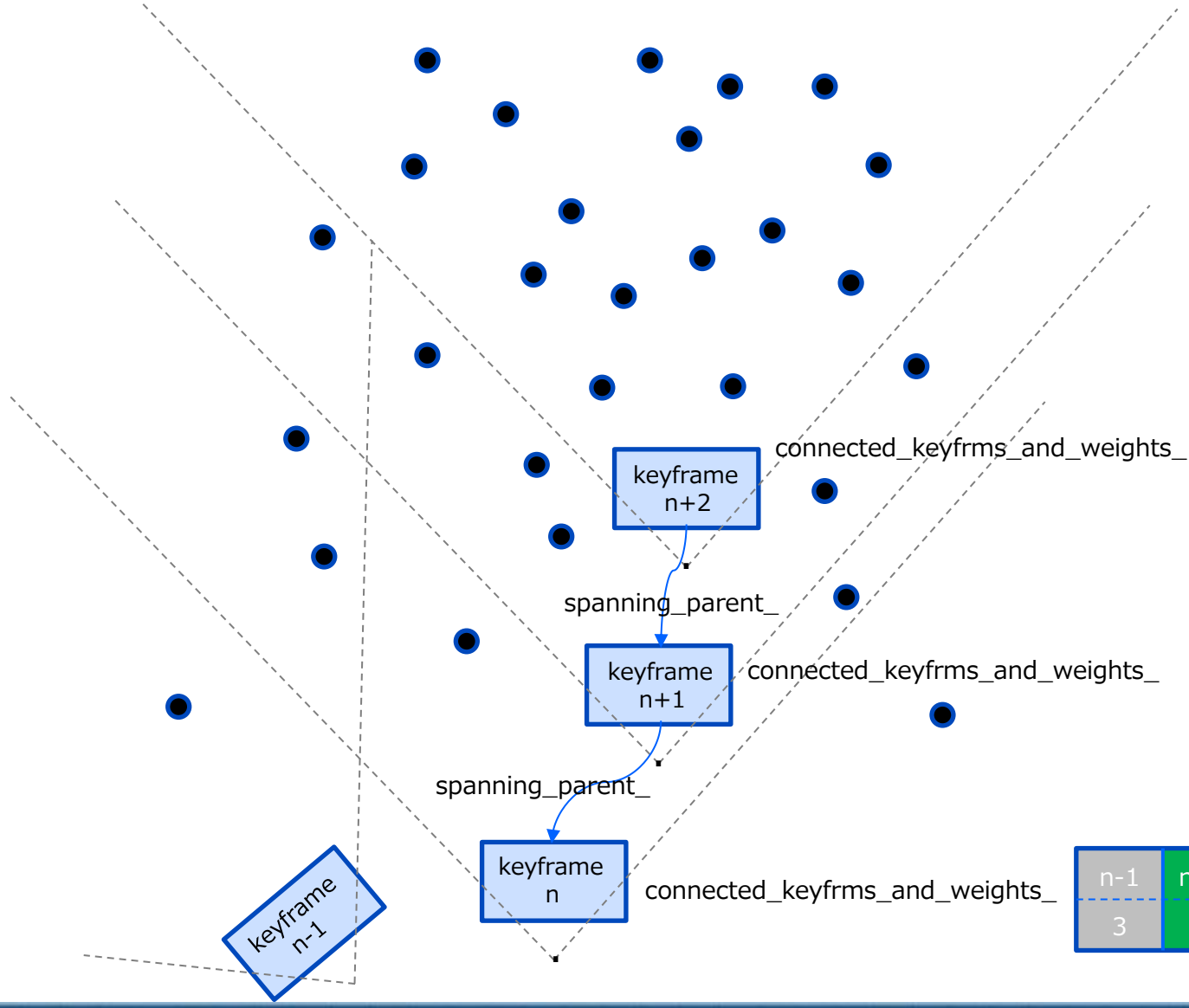
Landmarks and local landmarks

landmarks are:
3D points registered by mapping module

local landmarks are:
landmarks observed currently by:
- local keyframes
  - keyframes which have 1 or more common landmarks with the current frame
  - their top 10 covisibility keyframes
  - their child frames of the spanning tree
  - their parent frame of the spanning tree



Covisibility keyframes



A keyframe which observes same landmarks with concerned keyframe, and the number of the common landmarks is weight_thr_ (=15) or more is called `covisibility keyframe'.
Covisibility keyframes may be called just `covisibilities`.

n	n+1
16	16

keyframe n+2 has common landmarks with keyframe n and n+1 with weights 16 and 16 respectively. The keyframe n and n+1 are covisibility keyframes for the keyframe n+2.

n-1	n	n+2
1	22	16

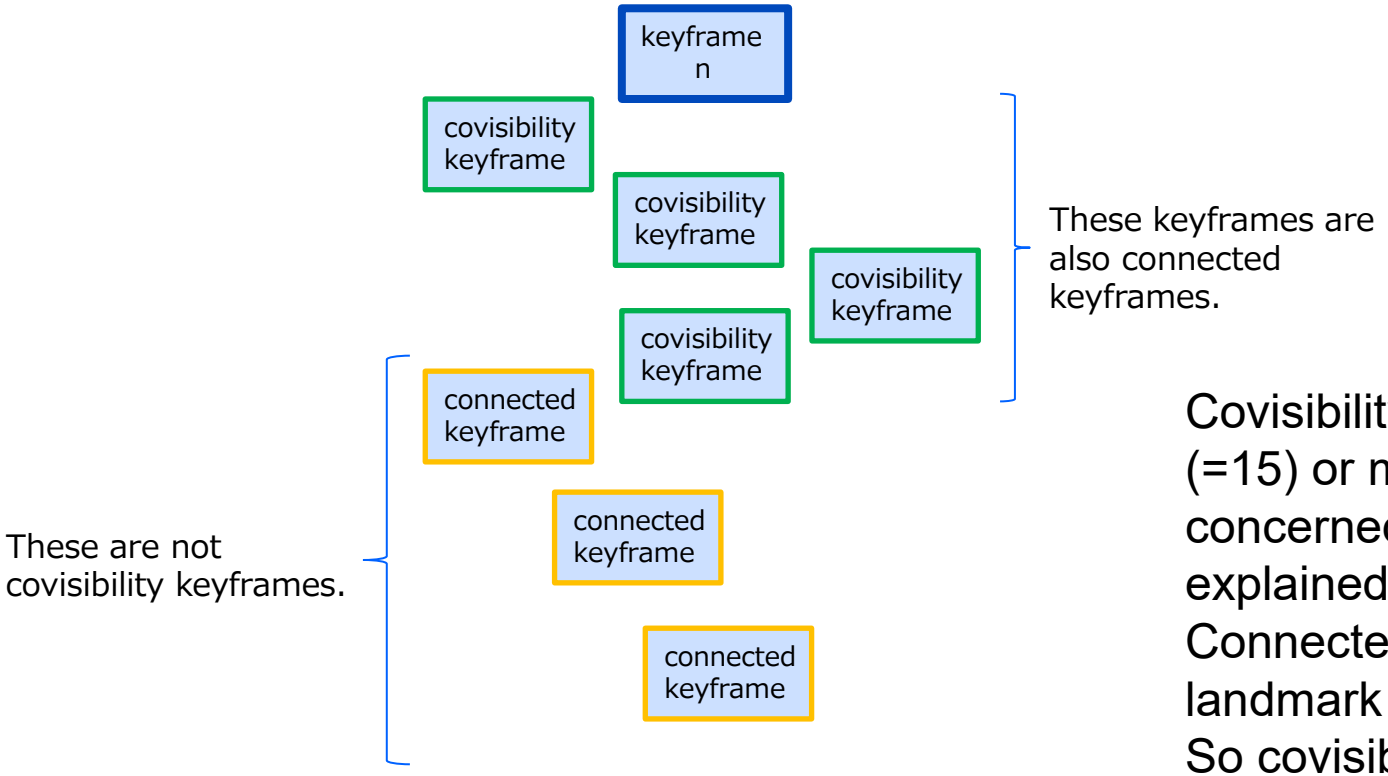
keyframe n+1 has common landmarks with keyframe n-1, n and n+2 with weights 1, 22 and 16 respectively. The keyframe n and n+2 are covisibility keyframes for the keyframe n+1, but the keyframe n-1 is not.

n-1	n+1	n+2
3	22	16

keyframes which have common landmarks with keyframe n

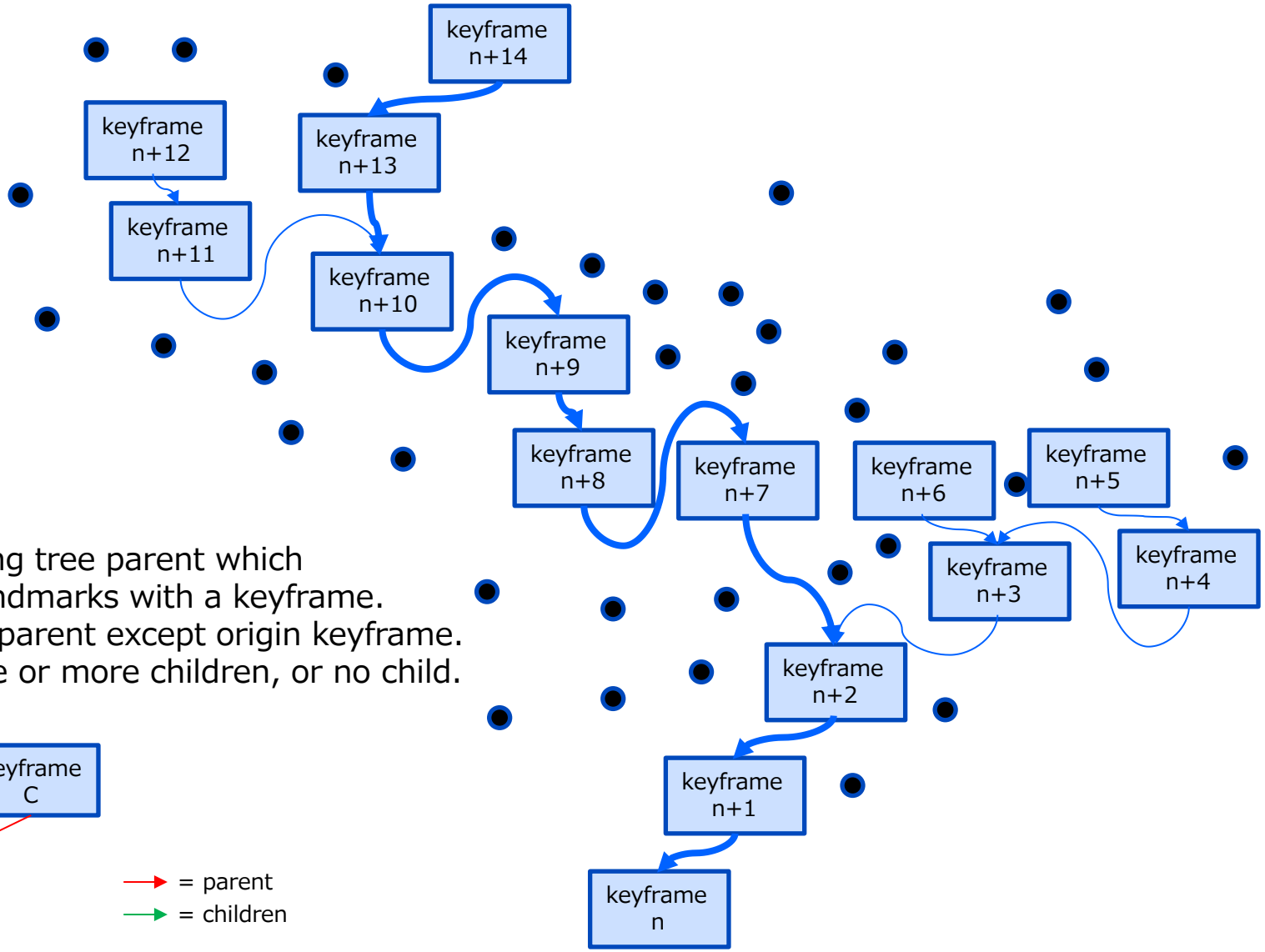
number of common landmarks

Connected keyframes

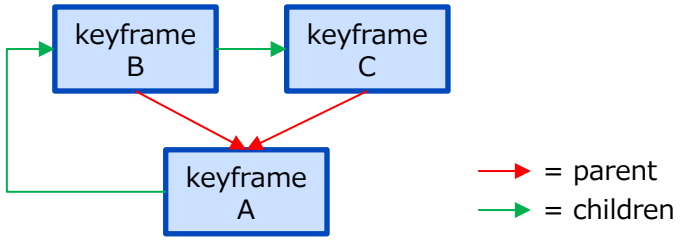


Covisibility keyframe is one observing weight_thr_ (=15) or more landmarks which are observed by concerned keyframe (keyframe n in the left figure) as explained in the previous slide.
Connected keyframe is one observing at least one landmark which are observed by concerned keyframe. So covisibility keyframe is also connected keyframe for concerned keyframe.
Covisibility keyframes are used for tracking, mapping and loop closing, while connected keyframes are used for only loop closing.

Spanning tree

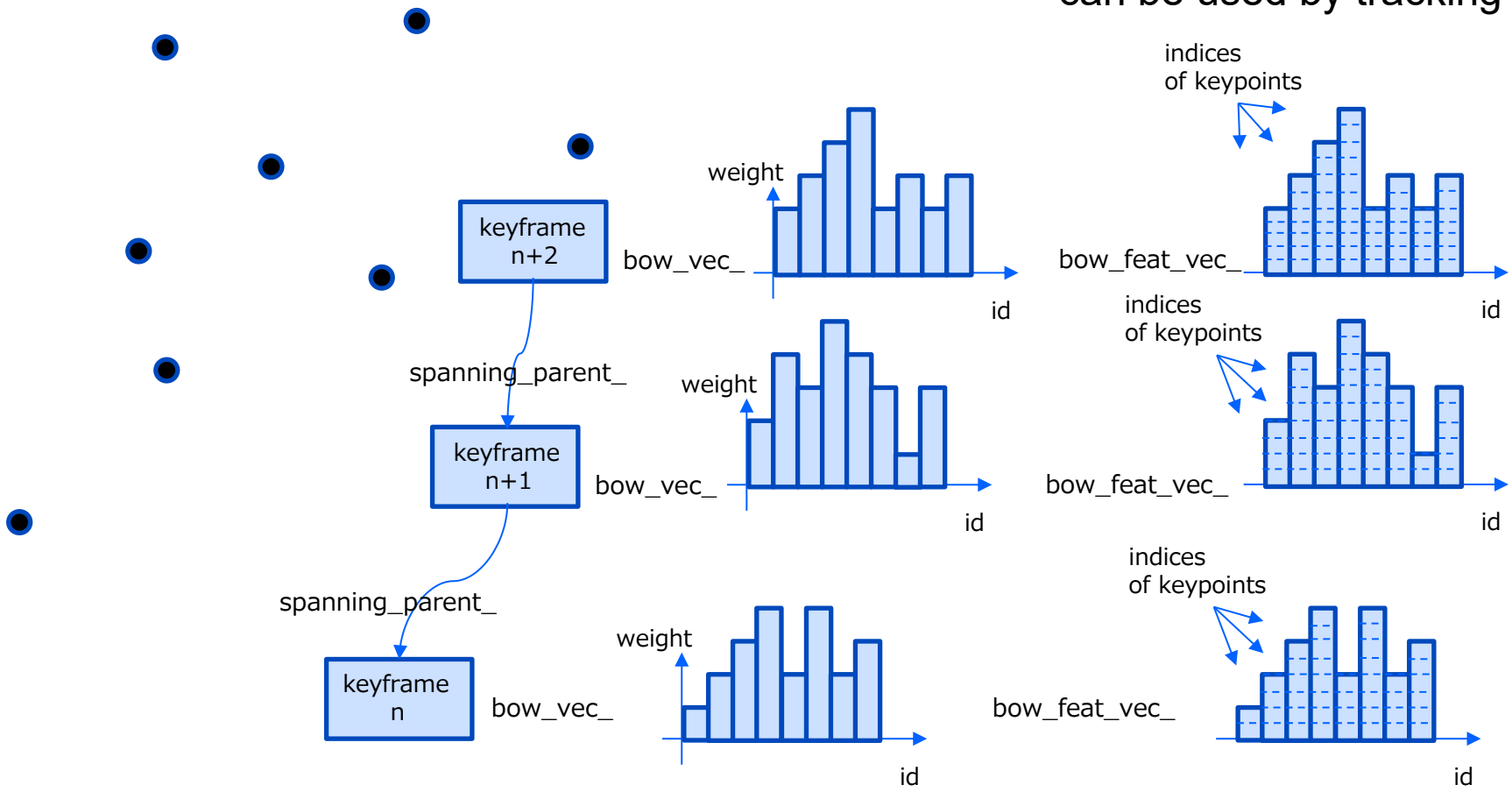


Blue arrow points spanning tree parent which
has the most common landmarks with a keyframe.
A keyframe has only one parent except origin keyframe.
A keyframe may have one or more children, or no child.



Bow data of each keyframe

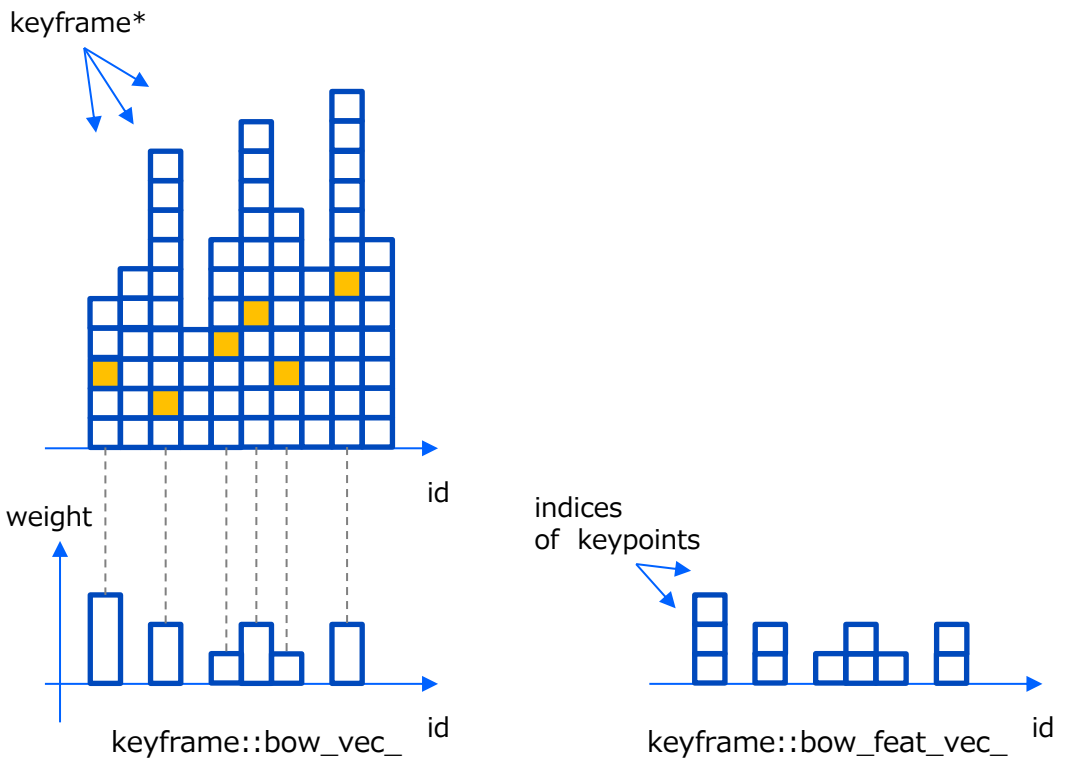
Each keyframe has bow data calculated by DBoW2 or FBoW mainly for the sake of relocalization and detecting loop closure, but also can be used by tracking module.



Bow database

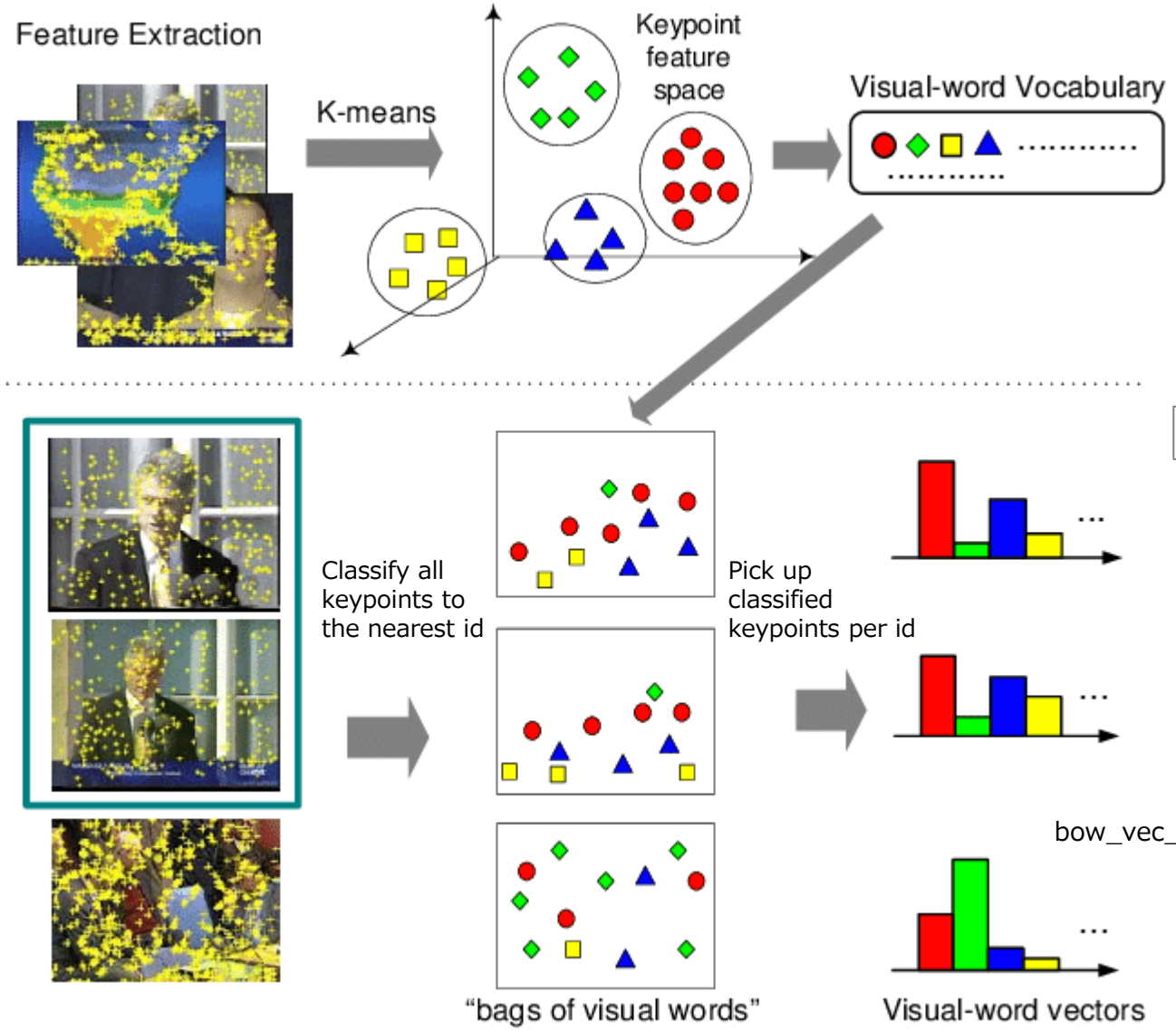
All bow data of all keyframes are registered to keyfrms_in_node_ in bow_database.

bow_database::
unordered_map<unsigned int, list<keyframe*> keyfrms_in_node_



= a keyframe which has bow_vec_ and bow_feat_vec_ above

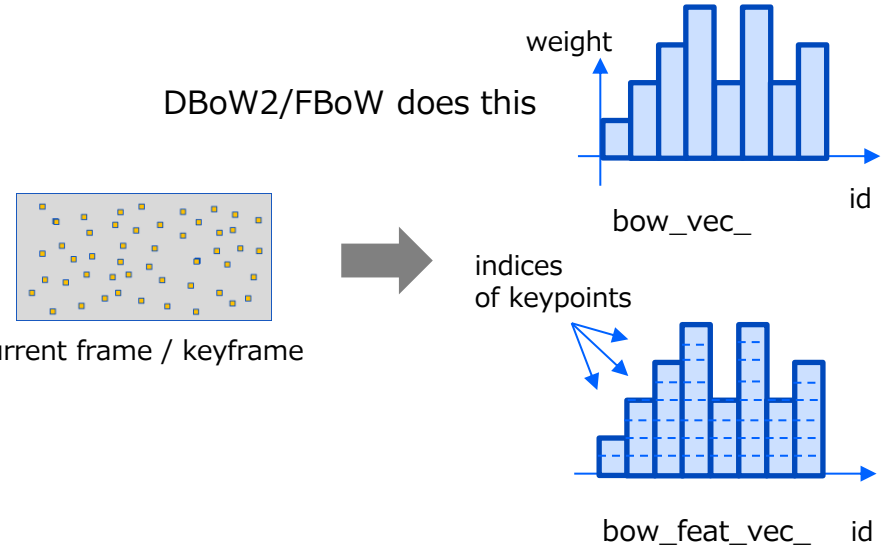
Introduction of BoW



Create visual word vocabulary from training dataset beforehand. Vocabulary file is needed to run OpenVSLAM such as orb_vocab.dbow2.

Compute bow from inputted keypoints.

DBoW2/FBoW does this

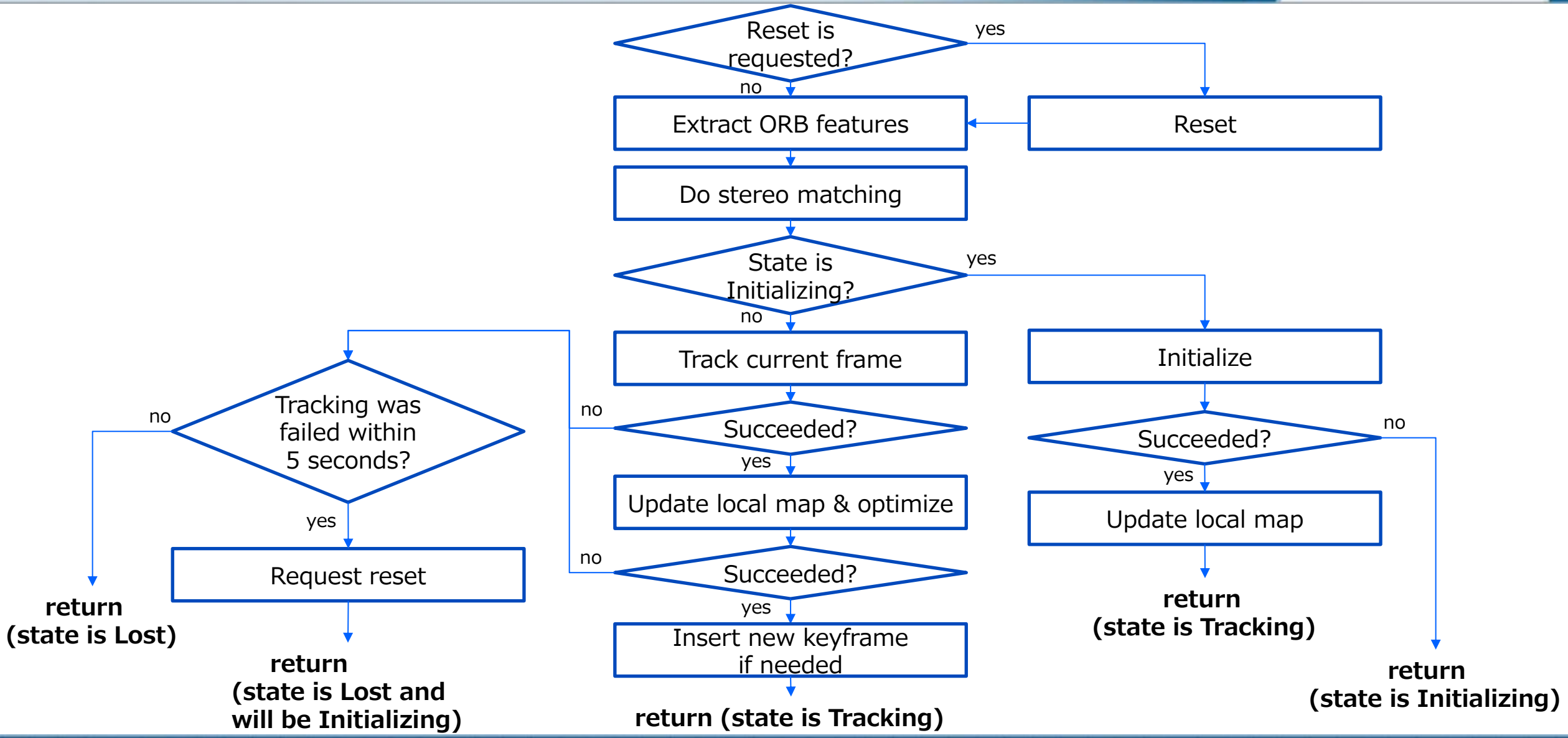


Stereo camera usage

- Left image is used in all cases
  - Tracking, mapping, loop closure
  - a figure of a frame or a keyframe in this document represents left image basically if one frame image is drawn unless it is denoted
- Right image is used supplementarily:
  - Depth calculation by stereo matching
  - Reprojection of a landmark to a frame/keyframe

TRACKING

Tracking overview



feed_stereo_frame

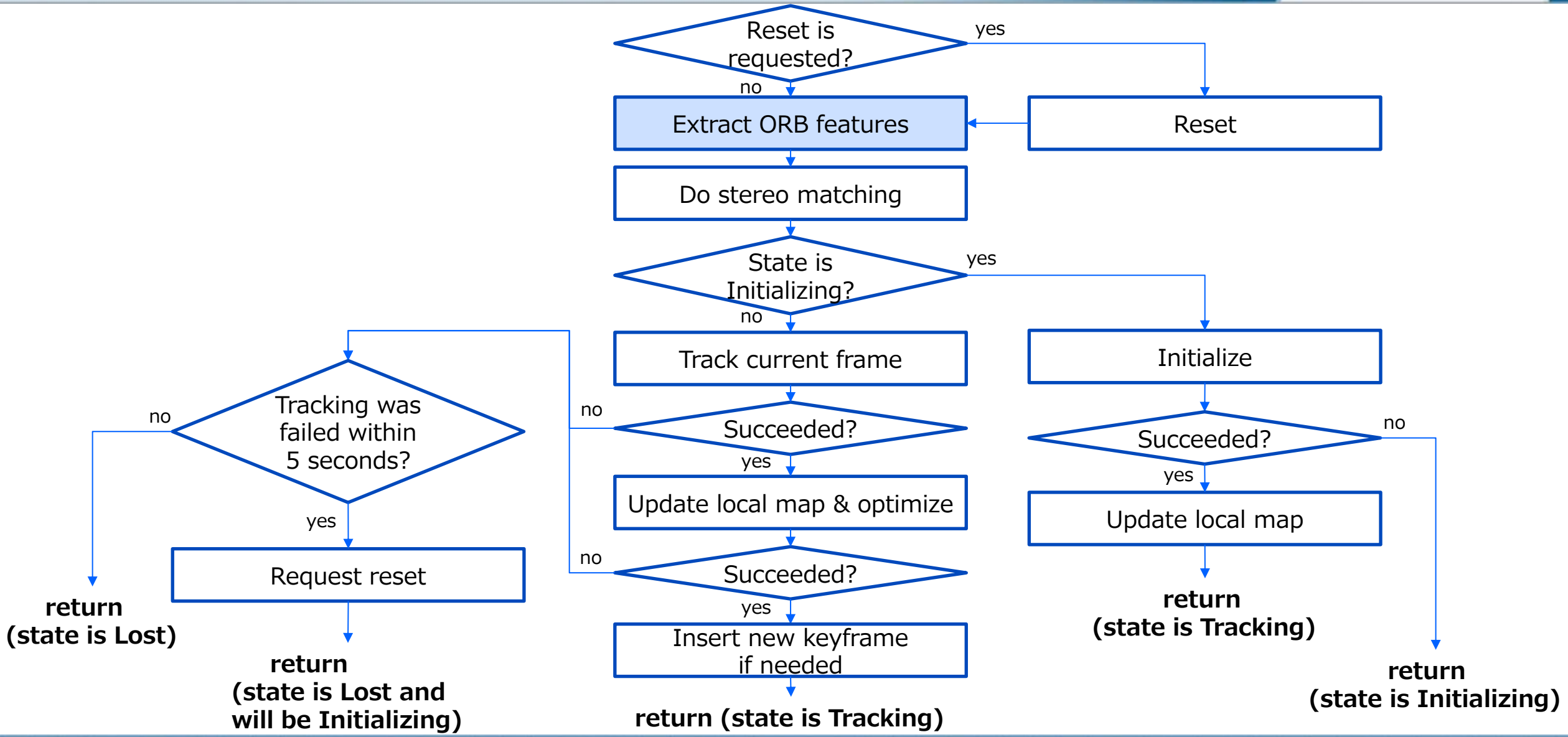
- Called when stereo image received
- Entry point of stereo version of OpenVSLAM
- Call:
  - tracker_->track_stereo_image
    - tracker_ is the instance of the tracking module
    - feature detection and camera pose estimation are done in track_stereo_image
  - frame_publisher_->update
  - map_publisher->set_current_cam_pose

track_stereo_image

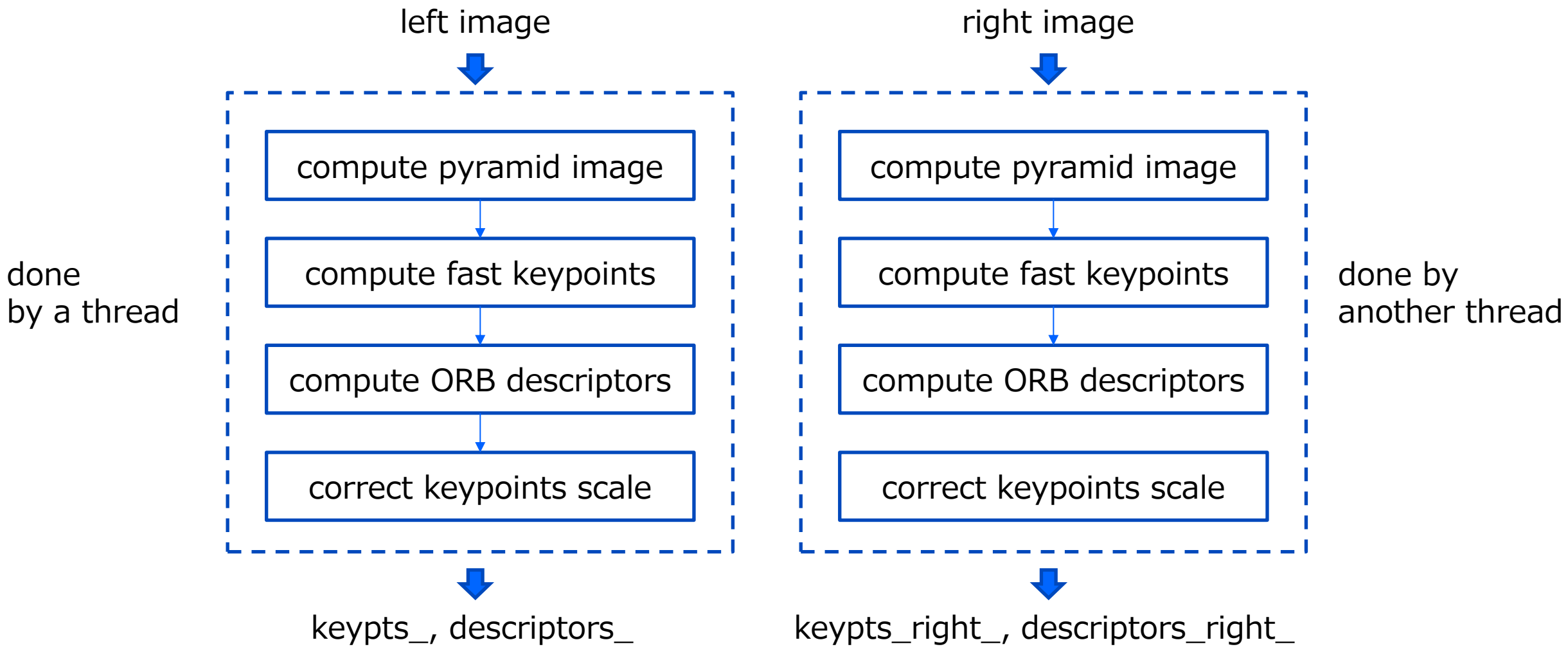
- Convert received image to gray scale
- Call:
  - data::frame
    - extract ORB features (keypoints)
    - estimate the keypoints' depth with stereo match
    - make bearing vectors from the keypoints
  - track
    - match keypoints between current and last frames
    - estimate camera pose by g2o
    - update local map
    - insert keyframe if needed

TRACKING – FEATURE DETECTION

Where we are


Reset is requested?
yes
no
Reset
Extract ORB features
Do stereo matching
State is Initializing?
yes
no
Initialize
Track current frame
Succeeded?
no
yes
Update local map
Succeeded?
no
yes
Update local map & optimize
Succeeded?
no
yes
Insert new keyframe if needed
Tracking was failed within 5 seconds?
no
yes
Request reset
return (state is Lost)
return (state is Lost and will be Initializing)
return (state is Tracking)
return (state is Tracking)
return (state is Initializing)

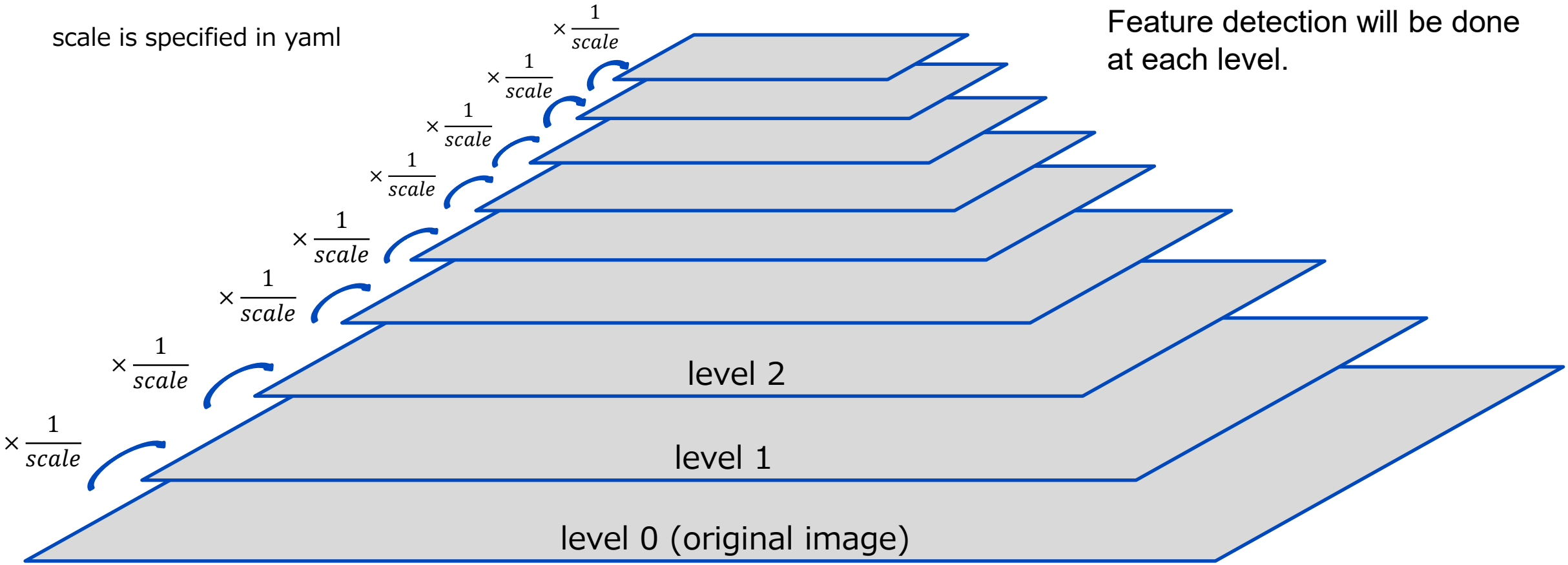
feature detection overview



Note that the left and the right images are not undistorted. Raw images are inputted.

compute_image_pyramid

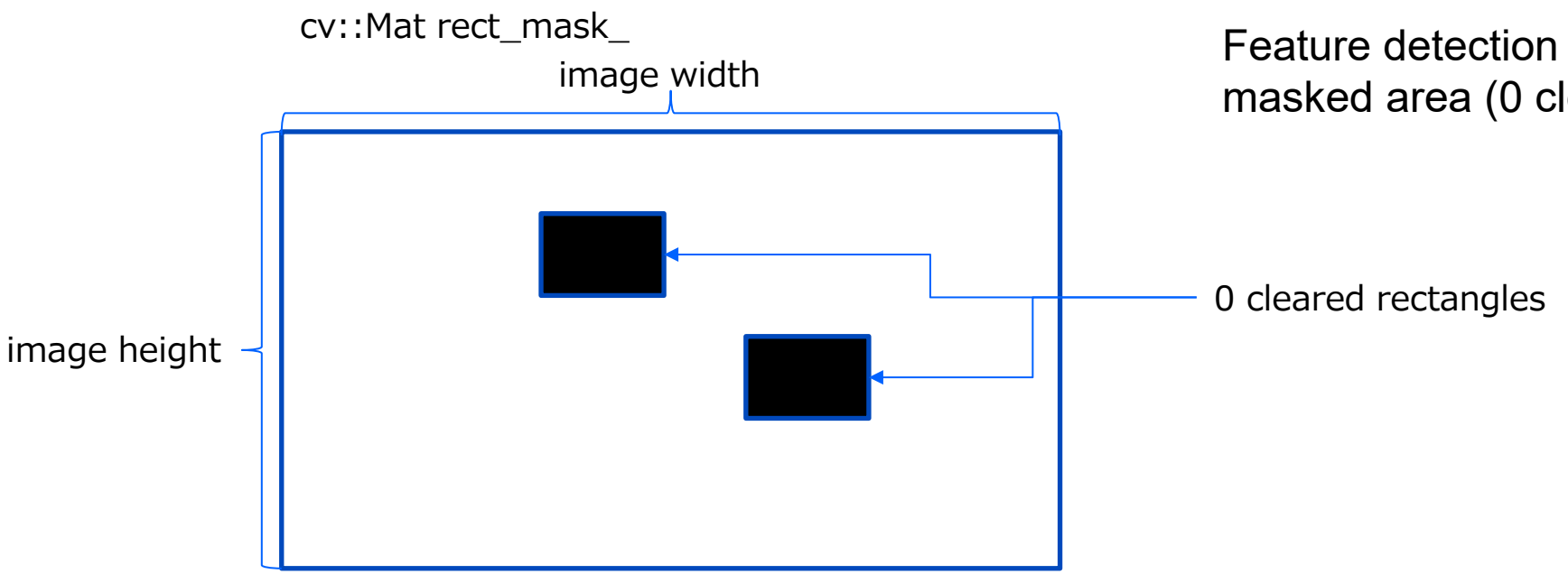
scale is specified in yaml



Feature detection will be done at each level.

create_rectangle_mask

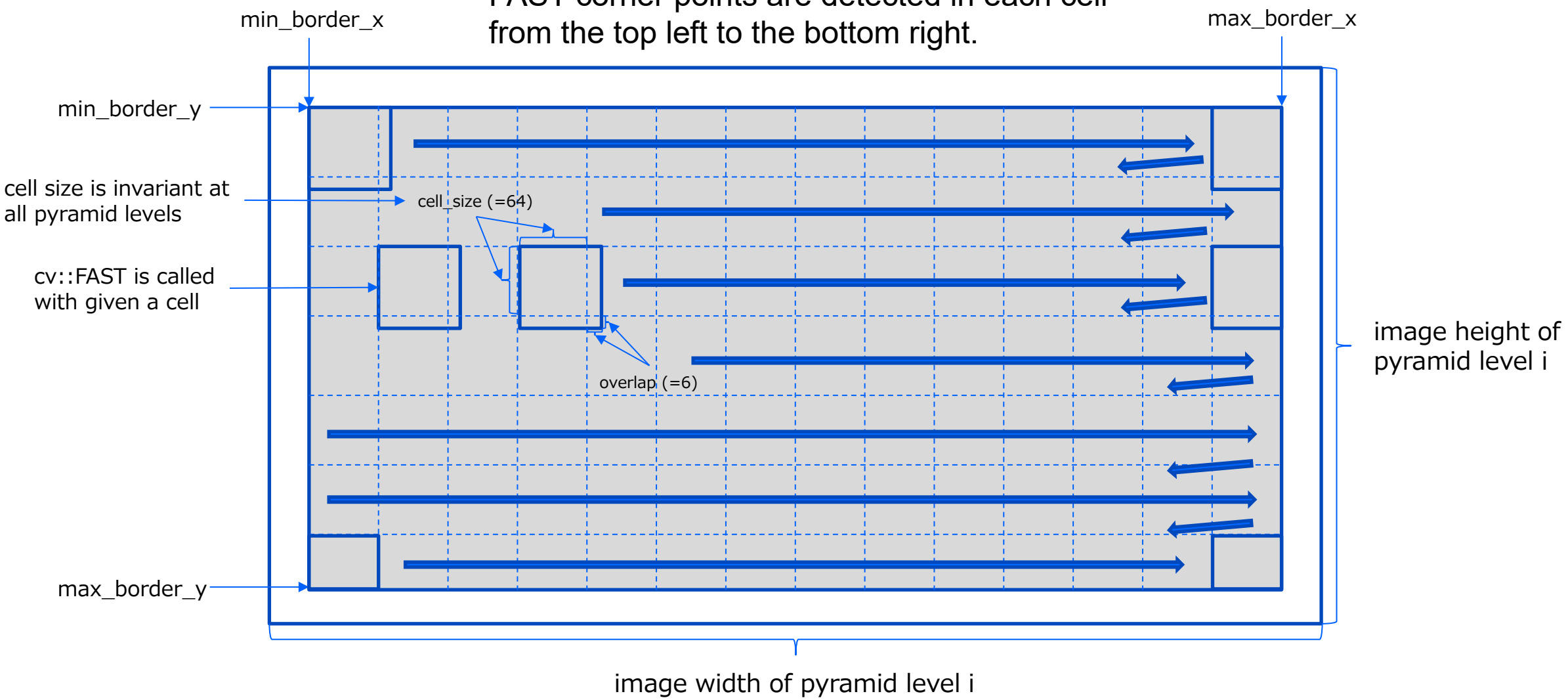
- Create rectangles on local cv::Mat rect_mask_ according to yaml description of Feature.mask_rectangles
- Called only once
- These masks are used when dynamic masks are not given
- If dynamic masks are given, they are used



Feature detection will not be done on masked area (0 cleared rectangles).

compute_fast_keypoints

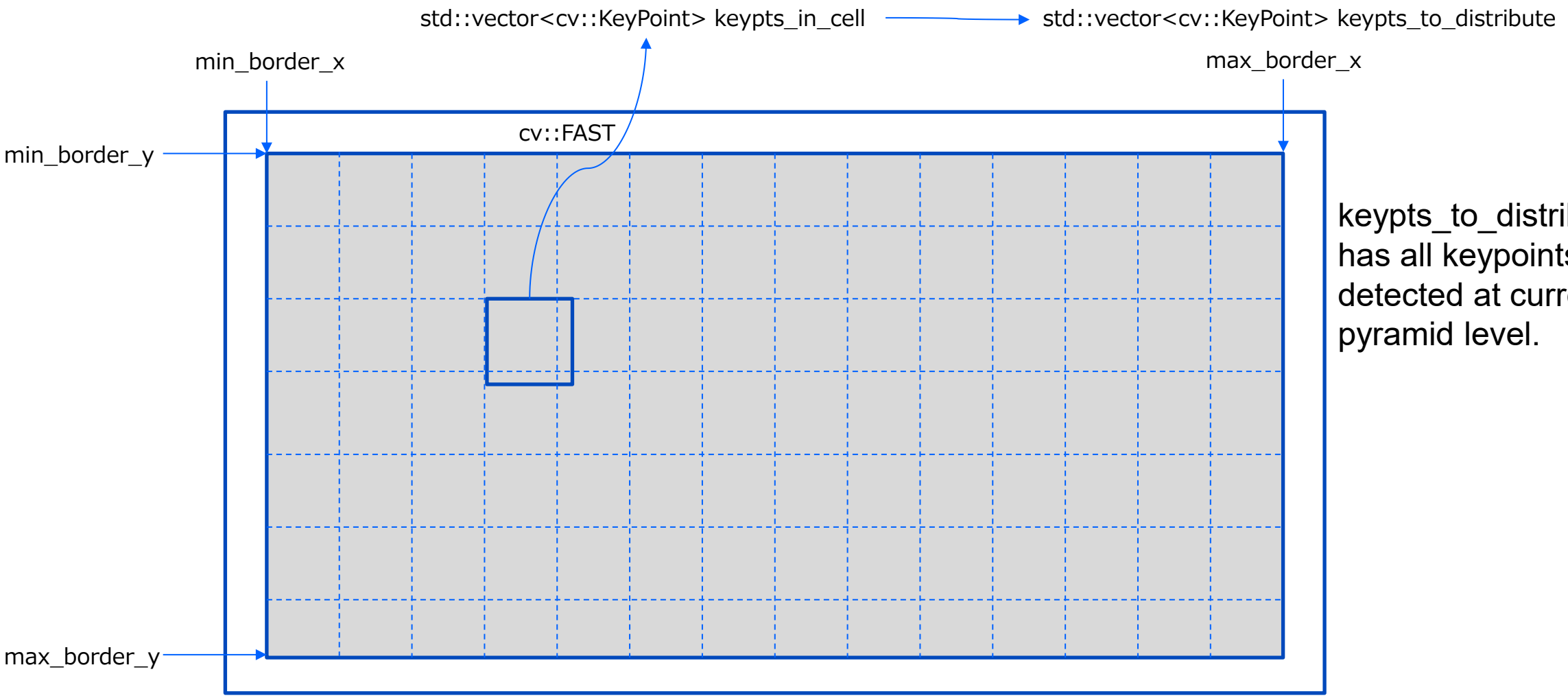
FAST corner points are detected in each cell from the top left to the bottom right.



cv::FAST

- void FAST(const Mat& image, vector<KeyPoint>& keypoints, int threshold, bool nonmaxSupression=true)
  - image: input image on which FAST corners will be detected
    - a cell described in the previous page is passed
  - keypoints: returned vector of cv::KeyPoint
    - local variable of type vector<cv::KeyPoint> is passed
  - threshold: intensity difference between center pixel and around pixels
    - ini_fast_threshold value in yaml is passed at first call
    - min_fast_threshold value in yaml is passed at second call if enough keypoints were not got at the first call
  - nonmaxSupression: if true non-maximum suppression will be applied for detected corners
    - true is passed

collect detected FAST corners

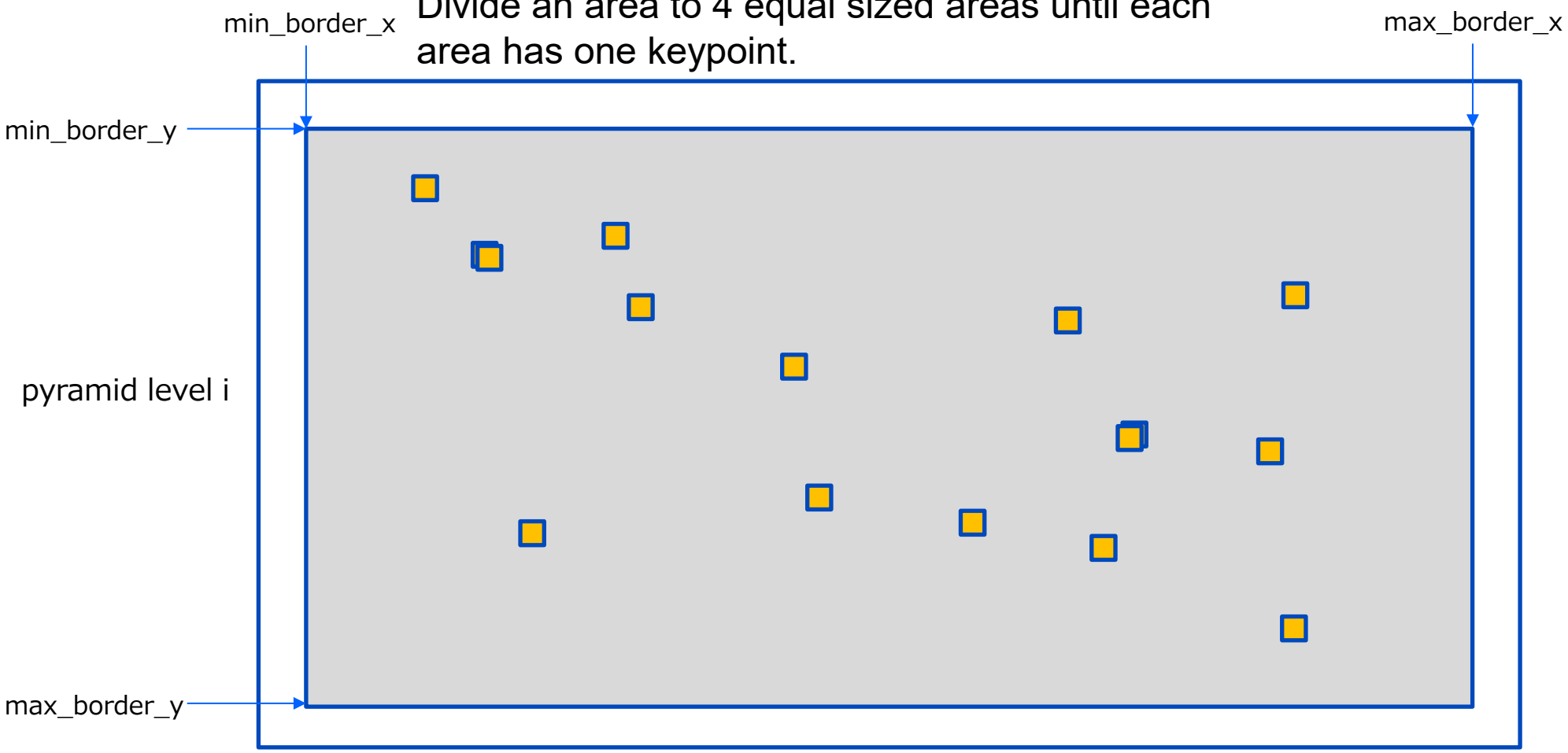


keypts_to_distributes has all keypoints detected at current pyramid level.

pt.x and pt.y of each keypoint in keypts_in_cell are adjusted with min_border_x and min_border_y origin because they are cell-based offset. If the adjusted point is in masked region, it is ignored.

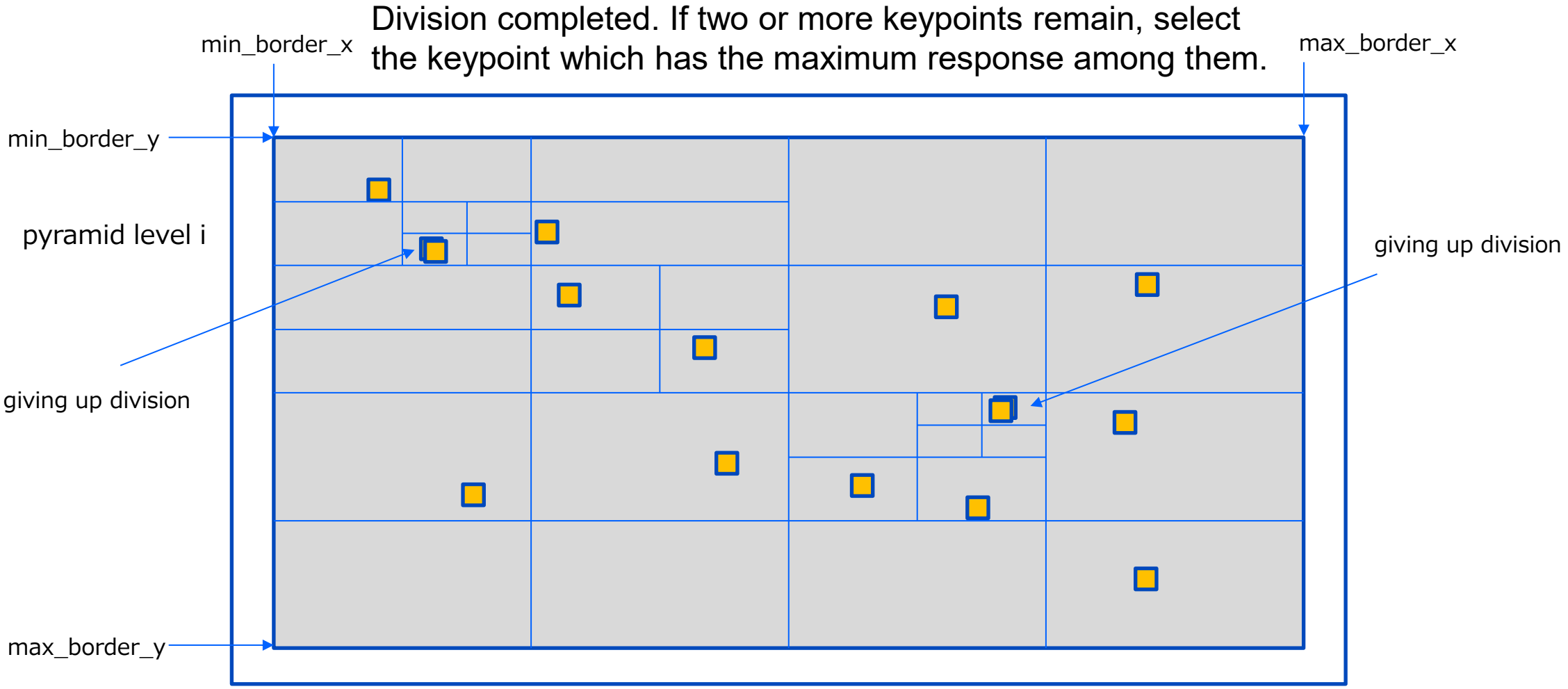
distribute_keypoints_via_tree – 1 of 3

Divide an area to 4 equal sized areas until each area has one keypoint.



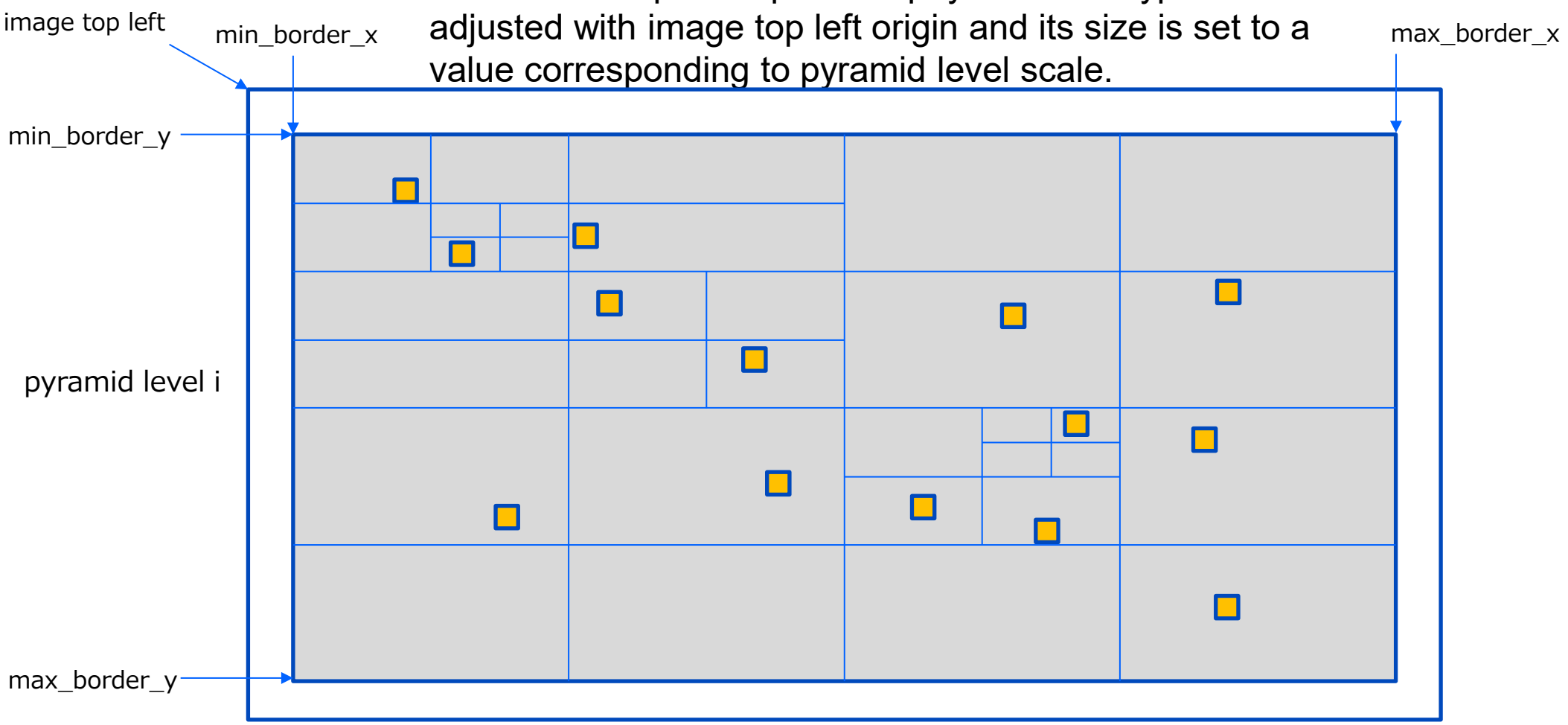
distribute_keypoints_via_tree – 2 of 3

Division completed. If two or more keypoints remain, select the keypoint which has the maximum response among them.



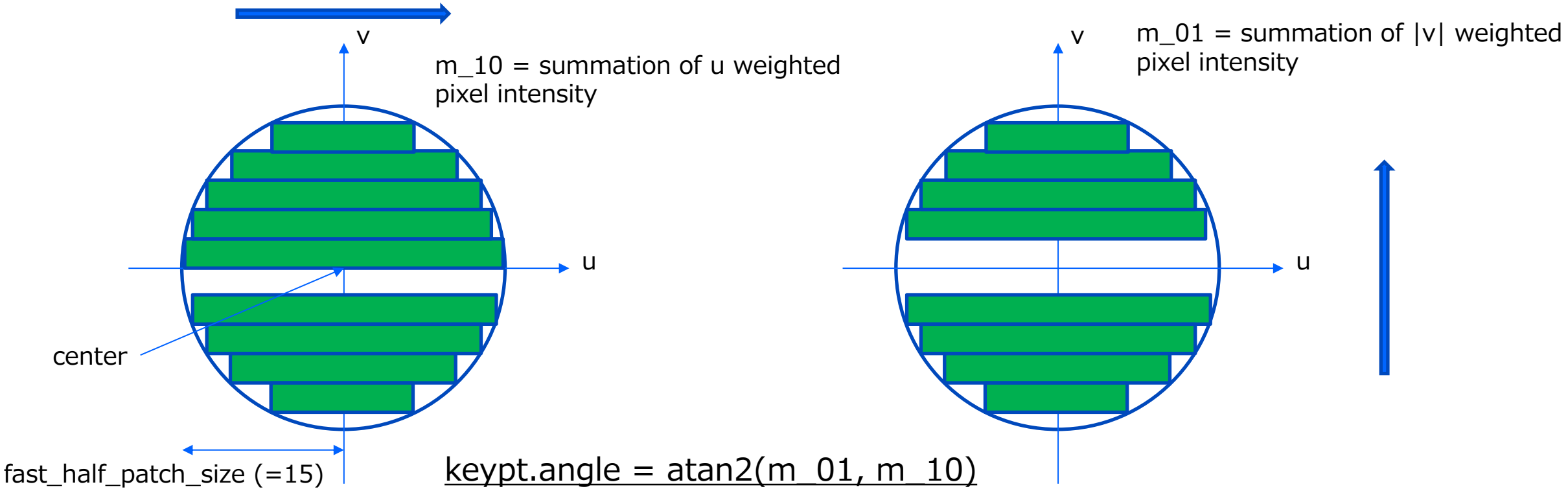
distribute_keypoints_via_tree – 3 of 3

Selection completed. pt.x and pt.y of each keypoint are adjusted with image top left origin and its size is set to a value corresponding to pyramid level scale.



compute_orientation

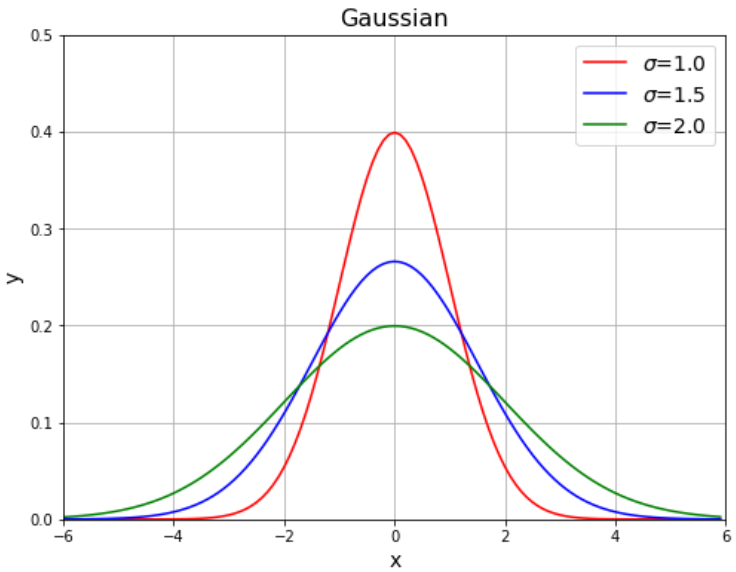
- Calculate orientation of each of all keypoints
- Maybe code copied from opencv
  - https://github.com/barak/opencv/blob/master/modules/features2d/src/orb.cpp#L260
- The orientation will be used in ORB feature matching



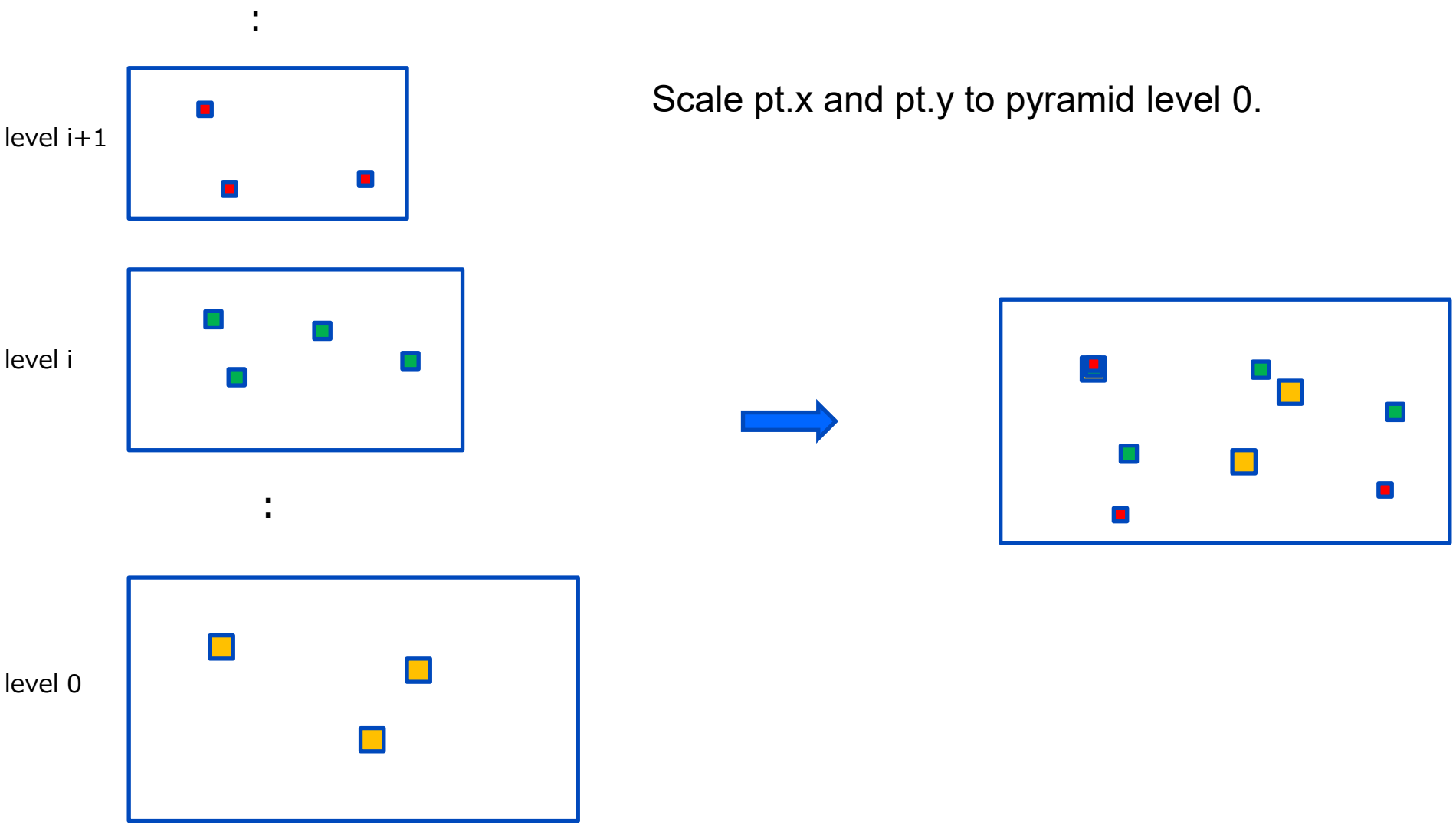
keypt.angle = atan2(m_01, m_10)

compute ORB descriptor

- Compute ORB descriptor of each of all keypoints of each pyramid level
- Each pyramid image is Gaussian blurred before computing ORB descriptors on it
  - kernel size = (7, 7)
  - $\sigma_x = 2, \sigma_y = 2$
- Calculation of ORB descriptor is implemented in compute_orb_descriptor function (OpenCV is not used)



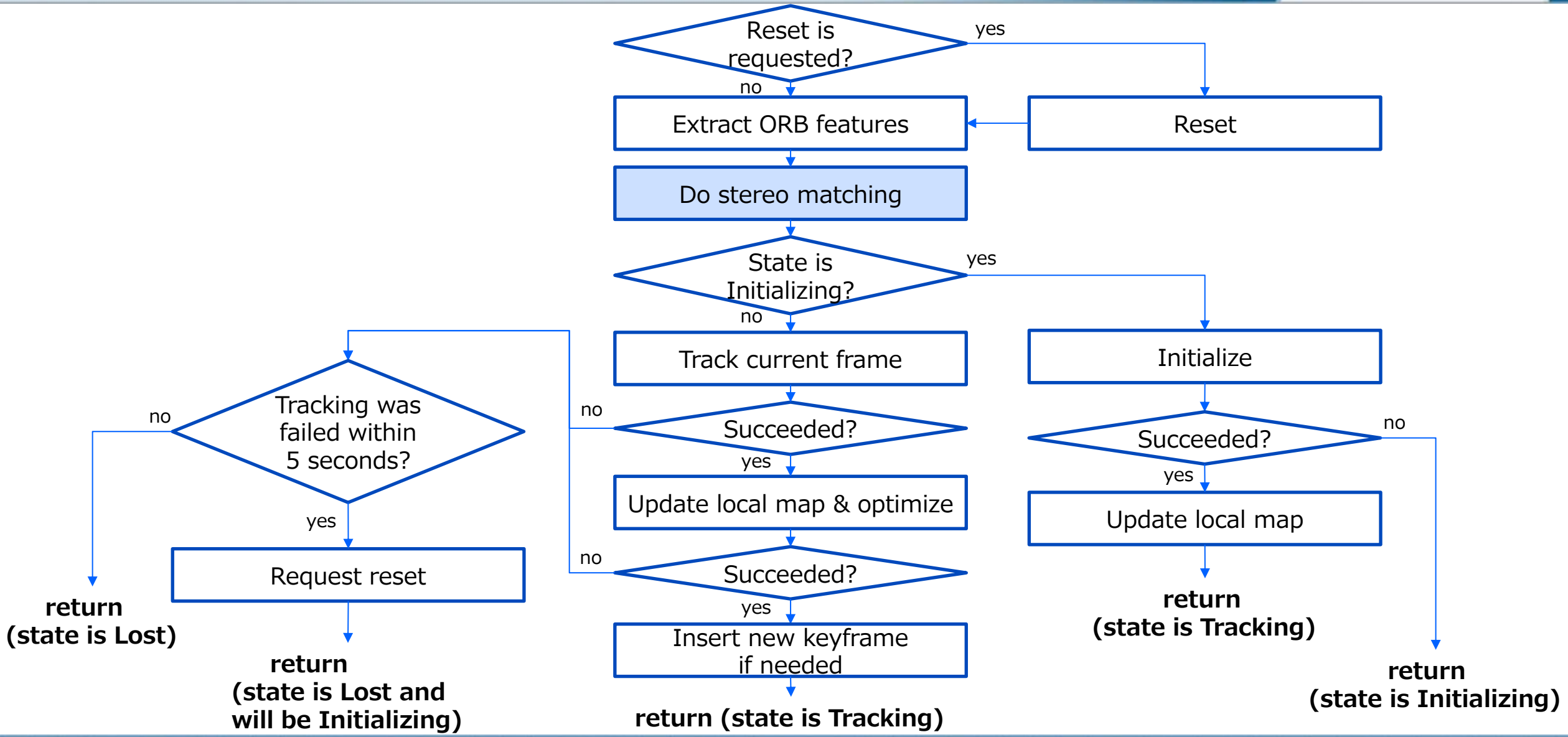
correct_keypoint_scale



Scale pt.x and pt.y to pyramid level 0.

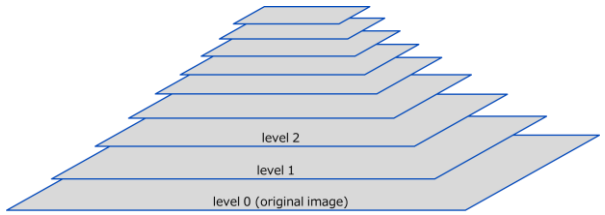
TRACKING – STEREO MATCHING

Where we are

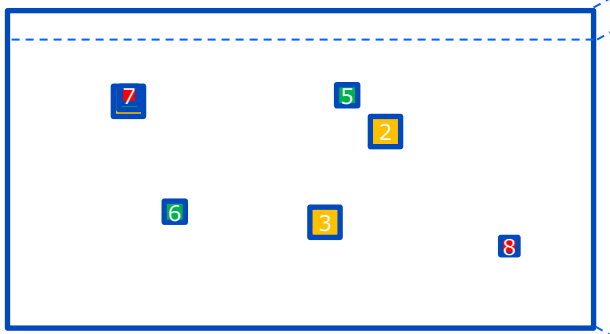

Reset is requested?
yes
no
Reset
Extract ORB features
Do stereo matching
State is Initializing?
yes
no
Initialize
Track current frame
Succeeded?
no
yes
Update local map
Succeeded?
no
yes
Update local map & optimize
Succeeded?
no
yes
Insert new keyframe if needed
Tracking was failed within 5 seconds?
no
yes
Request reset
return (state is Lost)
return (state is Lost and will be Initializing)
return (state is Tracking)
return (state is Tracking)
return (state is Initializing)

stereo::compute – 1 of 5

right pyramid image



keypoints in right image

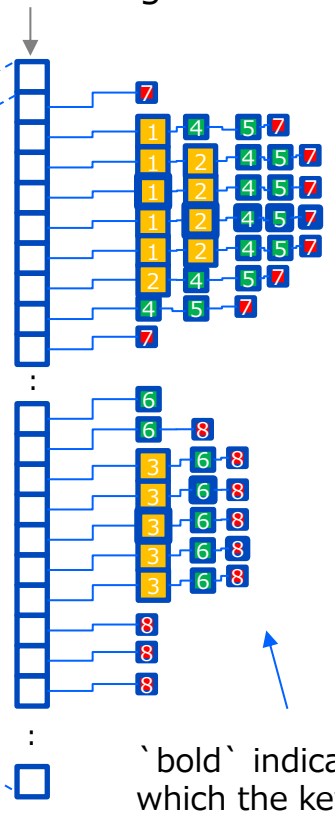


= keypoint at level 0

= keypoint at level i

= keypoint at level i+1

vector<vector<unsigned int>> indices_right_in_row



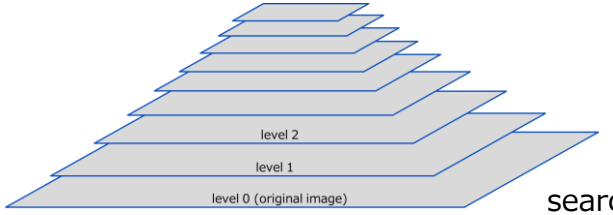
`bold` indicates the row on which the keypoint was detected.

Each keypoint's index is registered to indices_right_in_row spanning several rows both upward and downward from the row on which it is detected. The spanning height depends on the pyramid level on which the keypoint is detected.

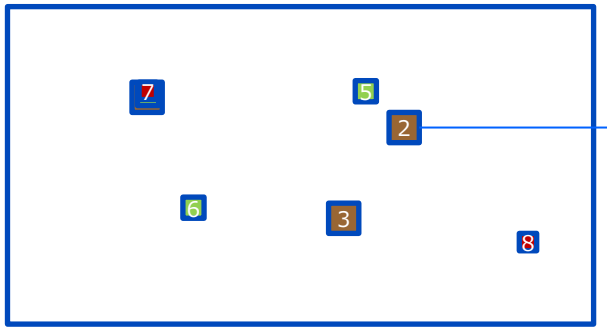
The indices_right_in_row variable is used to match a keypoint in the left image to the right.

stereo::compute – 2 of 5

left pyramid image



keypoints in left image



= keypoint at level 0

= keypoint at level i

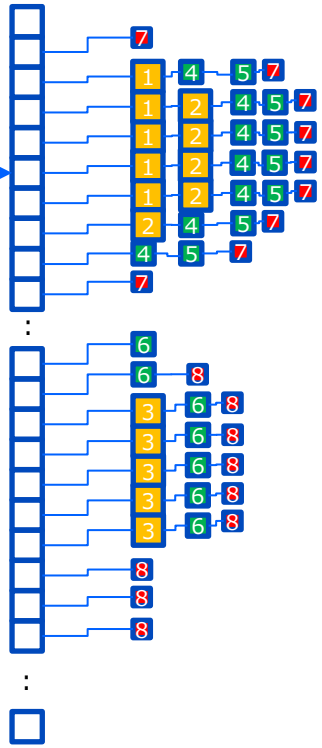
= keypoint at level i+1

vector<vector<unsigned int>> indices_right_in_row

search the best matched keypoint to one on the left image among corresponding row in indices_right_in_row (in this case, from 1 through 2, 4, 5 to 7)

row i

row i

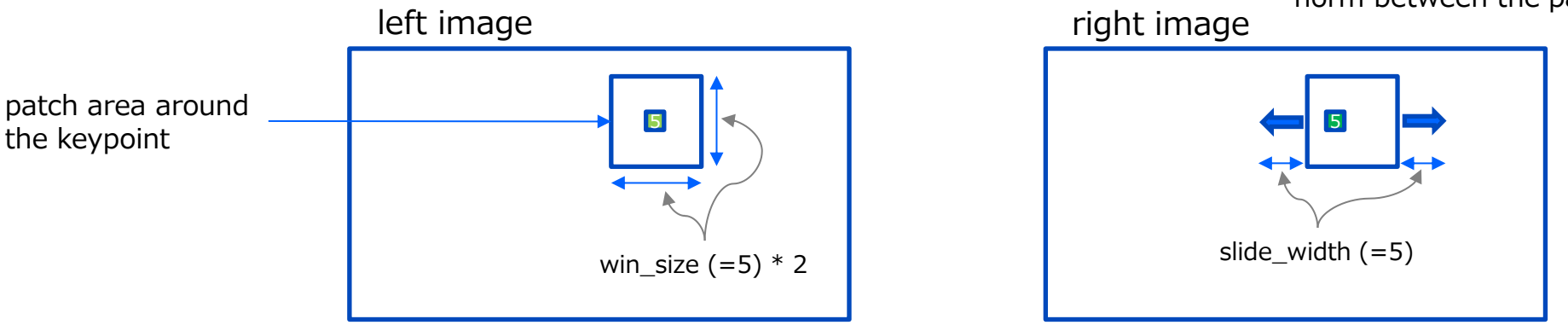


Search a keypoint on the right image that best matches a keypoint on the left image. The criterion is Hamming distance between ORB features. The search applies to all keypoints on the left image.
Thus, the best matched pairs for all keypoints on the left image are obtained.

stereo::compute – 3 of 5

compute_subpixel_disparity

Search best matched position for the left patch area by sliding a patch area one by one pixel and calculating L1 norm between the patches. (L1 norm $=\sum\left|I_{ij}^{left}-I_{ij}^{right}\right|$)

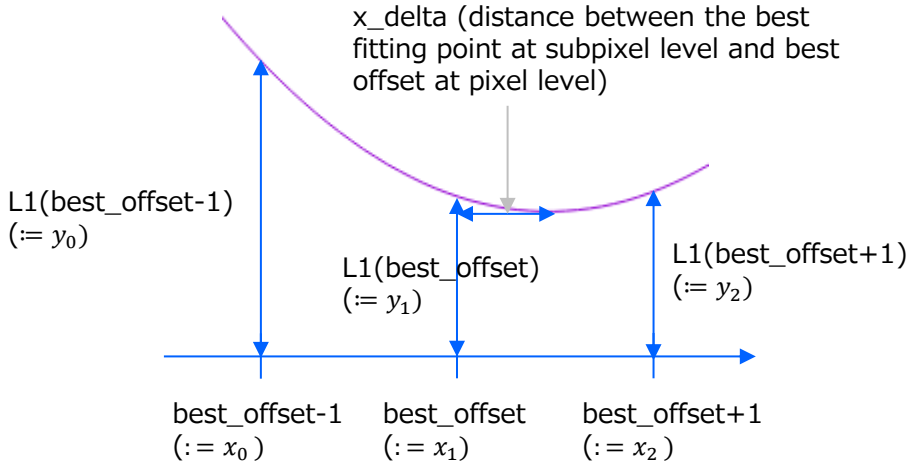


best_offset is obtained.
Then calculate subpixel offset by quadratic curve fitting using best_offset – 1, best_offset, and best_offset + 1.

Assume L1(best_offset-1), L1(best_offset), and L1(best_offset+1) are on a quadratic curve represented by $y = ax^2 + bx + c$. The best fitting point at subpixel level is obtained by calculating x that gives local minimum, that is, $x_{min} = -\frac{b}{2a}$, which is calculated as follows:

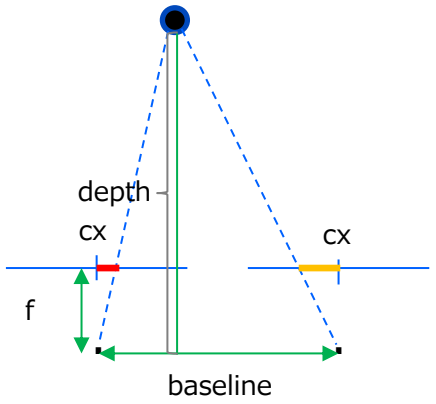
$$y_0 = ax_0^2 + bx_0 + c$$
$$y_1 = ax_1^2 + bx_1 + c$$
$$y_2 = ax_2^2 + bx_2 + c$$
$$y_2 - y_1 = a(x_2^2 - x_1^2) + b(x_2 - x_1) = a(2x_1 + 1) + b$$
$$y_1 - y_0 = a(x_1^2 - x_0^2) + b(x_1 - x_0) = a(2x_1 - 1) + b$$
$$\therefore a = \frac{1}{2}(y_0 - 2y_1 + y_2), \mathrm{b} = -(y_0 - 2y_1 + y_2)x_1 + \frac{1}{2}(y_0 - y_1)$$
$$\therefore x_{min} = -\frac{b}{2a} = x_1 + \frac{y_0 - y_2}{2(y_0 - 2y_1 + y_2)}$$
$$\therefore x_delta = \frac{y_0 - y_2}{2(y_0 - 2y_1 + y_2)}$$

Note:
Stereo computing is done based on distorted keypoint locations. Actually this should be done based on undistorted locations.



stereo::compute – 4 of 5

- best_disp and depth are obtained for each keypoint pair
  - best_disp = best_offset + x_delta
  - depth = focal length * baseline / best_disp



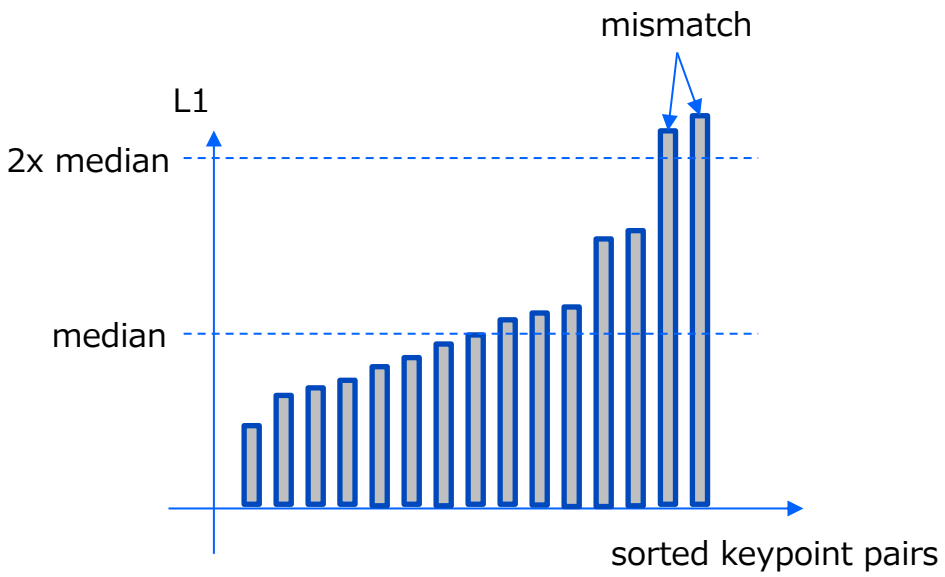
$$\frac{depth}{baseline} = \frac{f}{best_disp}$$

$$\therefore depth = f\,\frac{baseline}{best_disp}$$

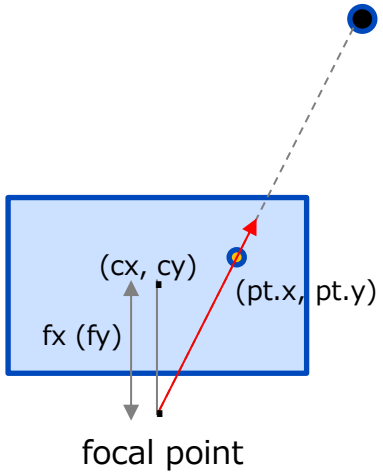
+ = best_disp

stereo::compute – 5 of 5

- L1(best_offset)s of all keypoint pairs (left keypoint and matched right one) are sorted by their size
- A keypoint pair that has greater L1(best_offset) value than 2x median of all L1s are regarded as mismatch and rejected



convert_keypoints_to_bearings



bearing vector

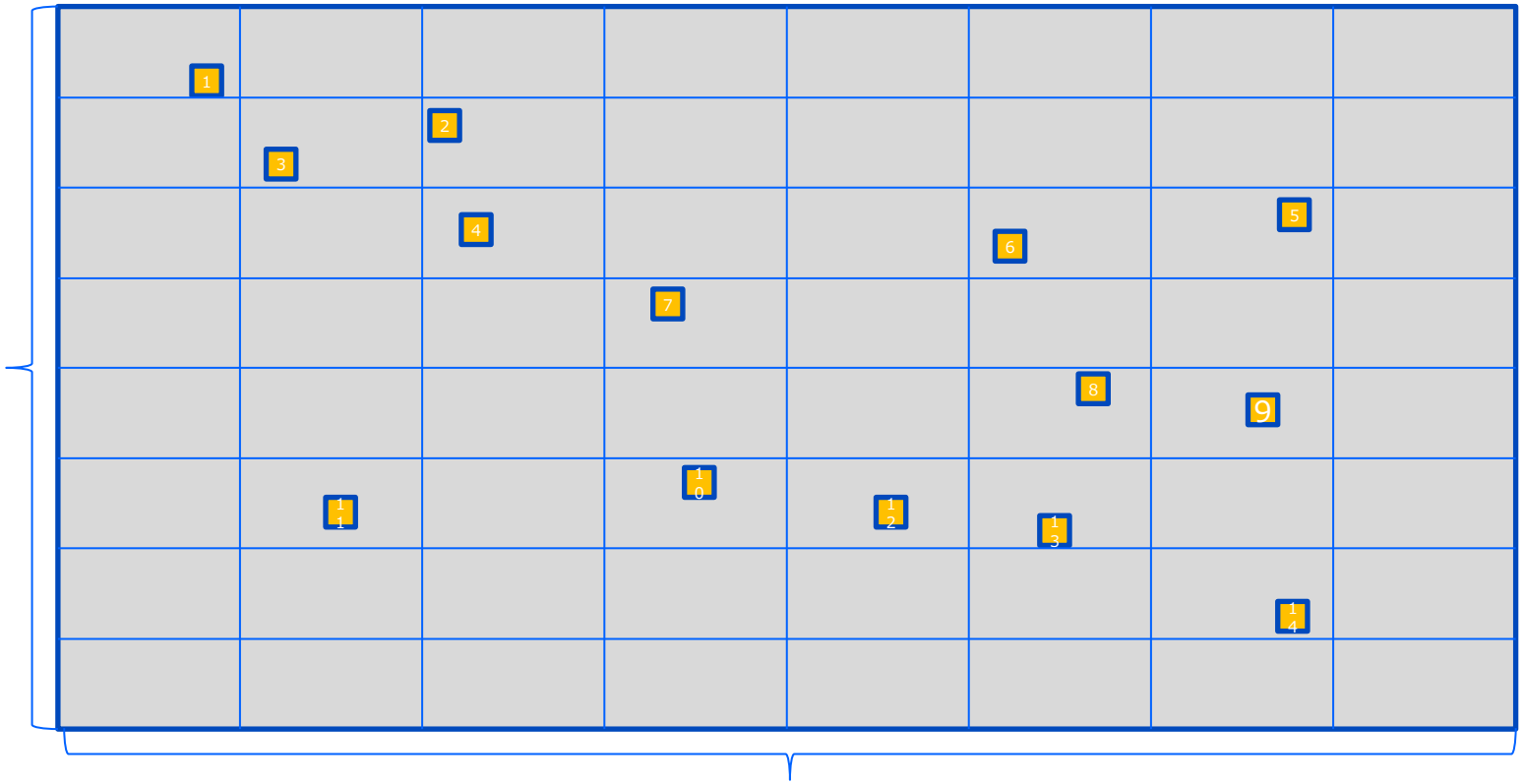
$$= \frac{\left(\frac{pt.x - cx}{fx}, \frac{pt.y - cy}{fy}, 1\right)}{\left|\left(\frac{pt.x - cx}{fx}, \frac{pt.y - cy}{fy}, 1\right)\right|}$$

Calculate bearing vector for each of all keypoints which alive so far.
Note that the calculation is done on the left image and keypoints on the left image are undistorted before the calculation.
The bearing vector will be used in robust_match_based_track and create_new_landmarks.

assign_keypoints_to_grid

vector<vector<vector<unsigned int>>> keypt_indices_in_cells

keypt_indices_in_cells[row][col] has keypoint indices which are accommodated in the cell specified by `row` and `col`.



camera.num_grid_rows_ (48)

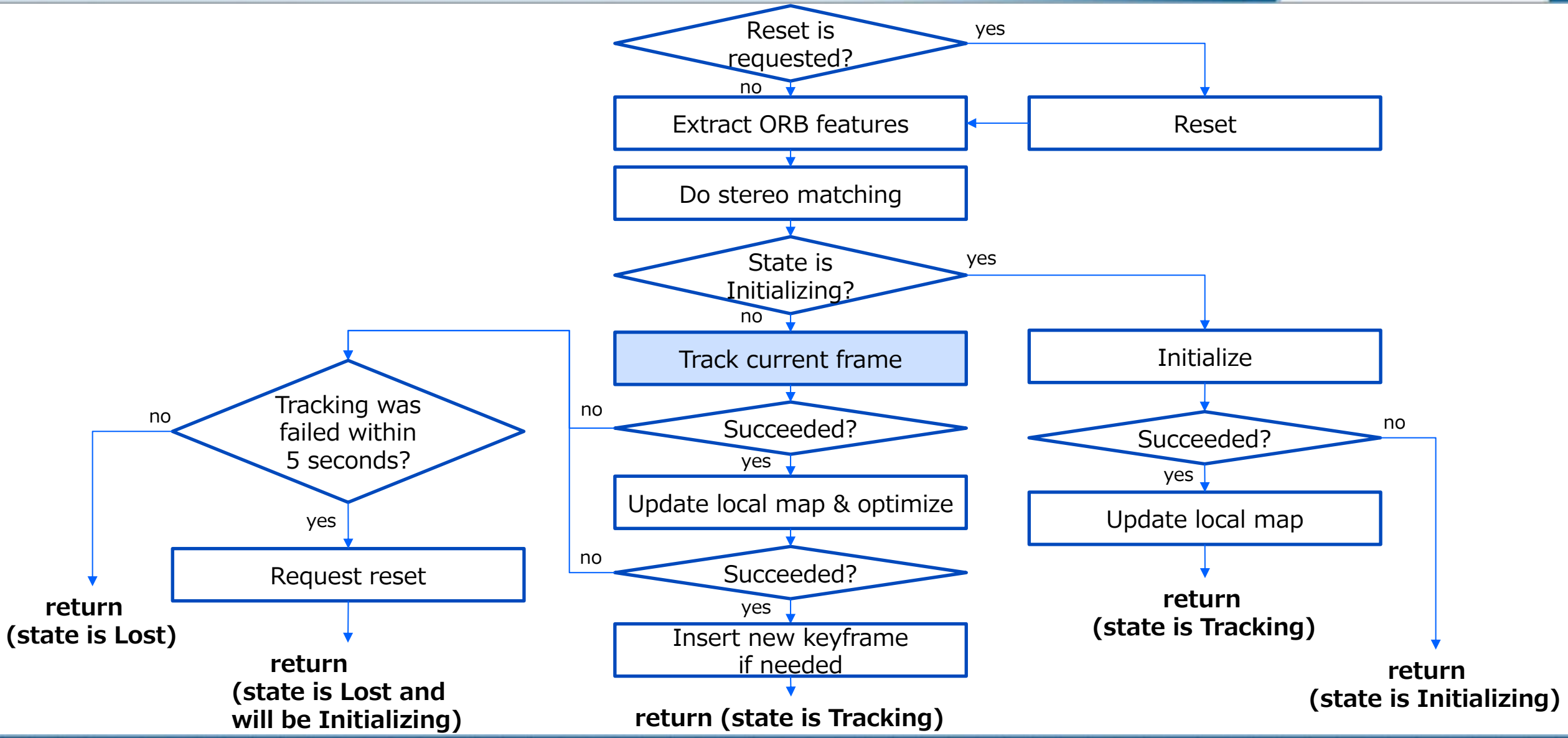
camera.num_grid_cols_ (64)

Note that this is done only on the left image.
This grid data will be used hereafter in searching matched keypoint.

camera.num_grid_cols_ and camera.num_grid_rows_ are specified in src/openvslam/camera/base.h

TRACKING – TRACK

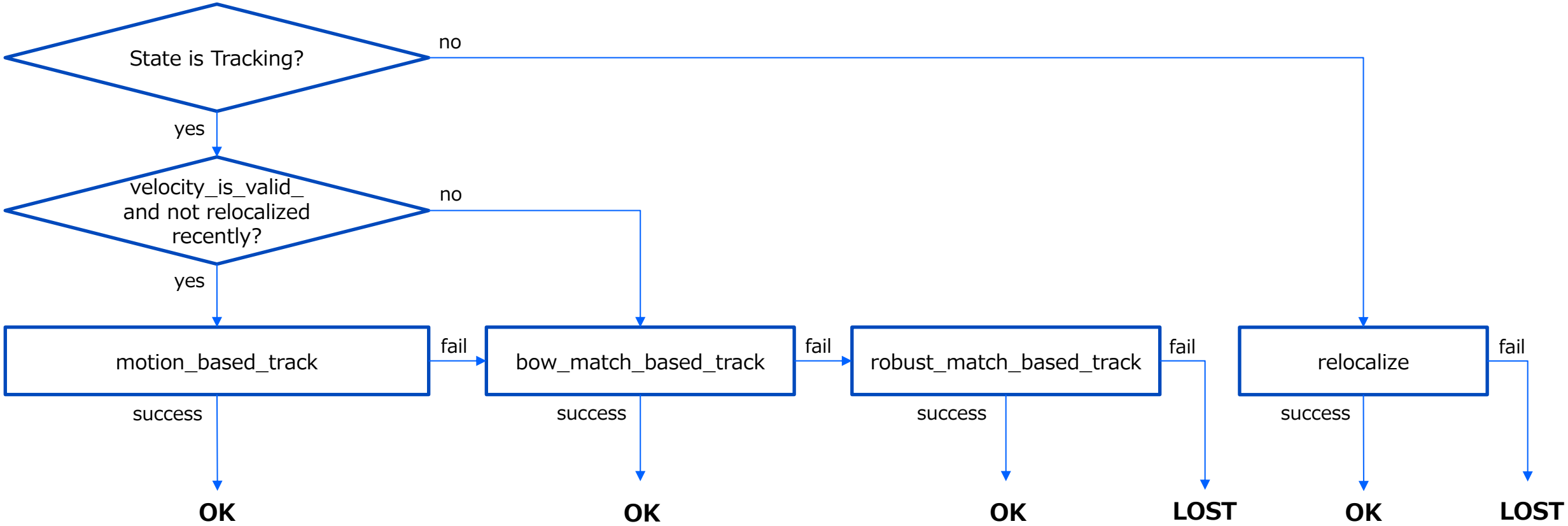
Where we are


Reset is requested?
yes
no
Reset
Extract ORB features
Do stereo matching
State is Initializing?
yes
no
Track current frame
Initialize
Tracking was failed within 5 seconds?
no
yes
Request reset
return (state is Lost)
return (state is Lost and will be Initializing)
Succeeded?
no
yes
Update local map & optimize
Succeeded?
no
yes
Insert new keyframe if needed
return (state is Tracking)
Succeeded?
no
yes
Update local map
return (state is Tracking)
return (state is Initializing)

Before calling track_current_frame

- apply_landmark_replace
  - global optimization and mapping module threads may have replaced landmarks as a result of removal of duplicated landmarks
  - apply_landmark_replace updates landmarks_ vector of the last frame if landmarks are replaced
- update_last_frame
  - last frame's relative pose from its reference keyframe was calculated at previous time
  - the reference keyframe pose may have optimized because of local or global bundle adjustment, so calculate the pose of the last frame at this point of time

track_current_frame



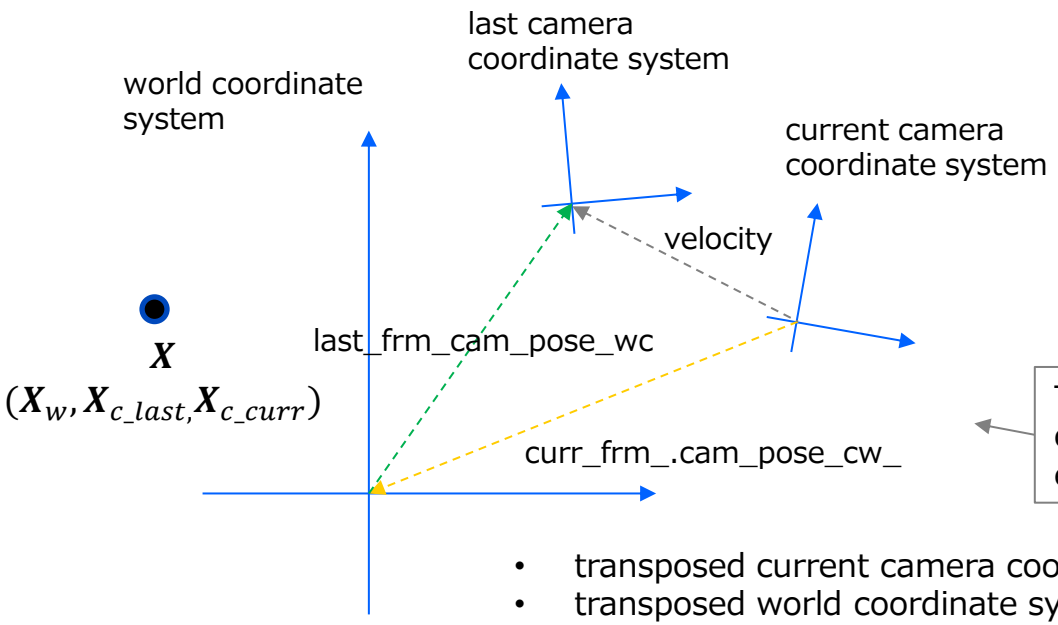
velocity_is_valid_:
basically true except the very first frame or just after relocalization, but not reached to track_current_frame in such a situation. (I guess this is a redundant condition.)

relocalize will be explained in later part

TRACKING – TRACK – TRACK_CURRENT_FRAME – MOTION_BASED_TRACK

motion_based_track I/F

- motion_based_track(data::frame& curr_frm, const data::frame& last_frm, const Mat44_t& velocity)
  - velocity = curr_frm_.cam_pose_cw_ * last_frm_cam_pose_wc
  - note that velocity was calculated at the end of the previous tracking



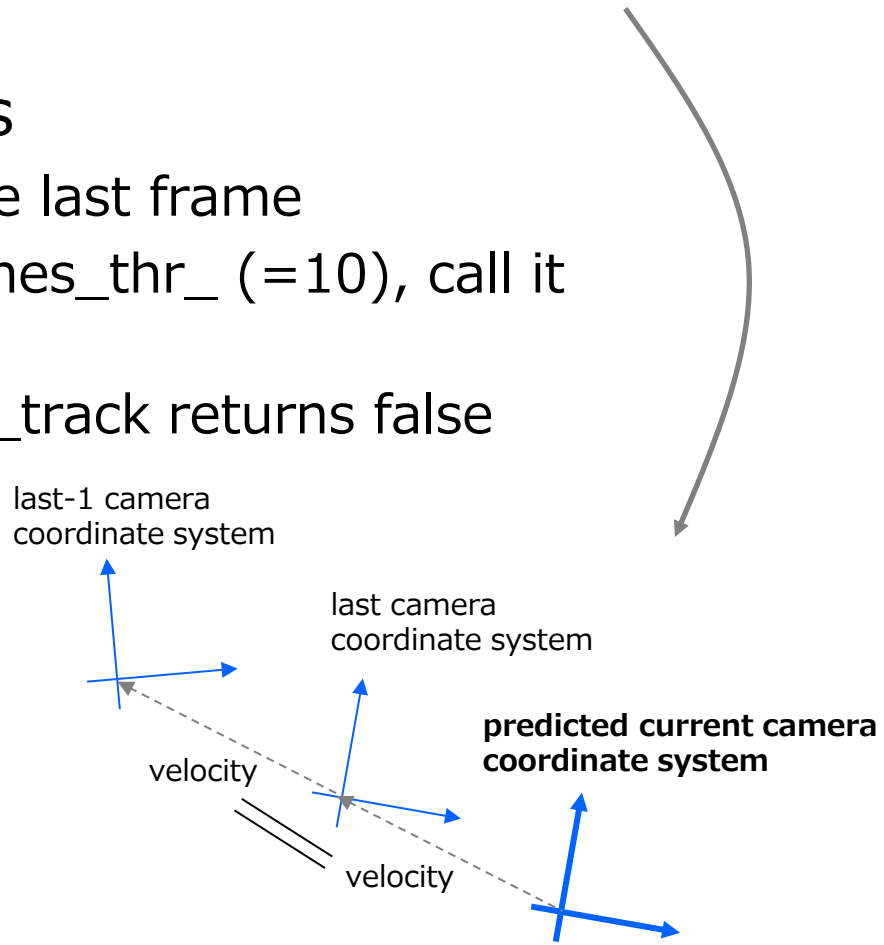
$$\boldsymbol{X}_w = last_frm_cam_pose_wc \cdot \boldsymbol{X}_{c_last}$$
$$\boldsymbol{X}_{c_curr} = curr_frm_.cam_pose_cw_ \cdot \boldsymbol{X}_w$$
$$\therefore \boldsymbol{X}_{c_curr} = curr_frm_.cam_pose_cw_ \cdot last_frm_cam_pose_wc \cdot \boldsymbol{X}_{c_last}$$
$$\boldsymbol{X}_{c_curr} = velocity \cdot \boldsymbol{X}_{c_last}$$

This figure shows the velocity between the current and the last camera pose at the end of the previous tracking, not now.

- transposed current camera coordinate system by curr_frm_.cam_pose_cw_ matches world coordinate system
- transposed world coordinate system by last_frm_cam_pose_wc matches last camera coordinate system
- transposed current camera coordinate system by velocity matches last camera coordinate system

motion_based_track overview

- Predict current camera pose using previous velocity
- Then call:
  - projection::match_current_and_last_frames
    - Search ORB matches between the current and the last frame
    - If the number of matches is less than num_matches_thr_ (=10), call it again with wider margin
    - If still not enough matches (<10), motion_based_track returns false
  - pose_optimizer::optimize
  - discard_outliers
- If the number of inliers is less than num_matches_thr_, return false
- Else return true

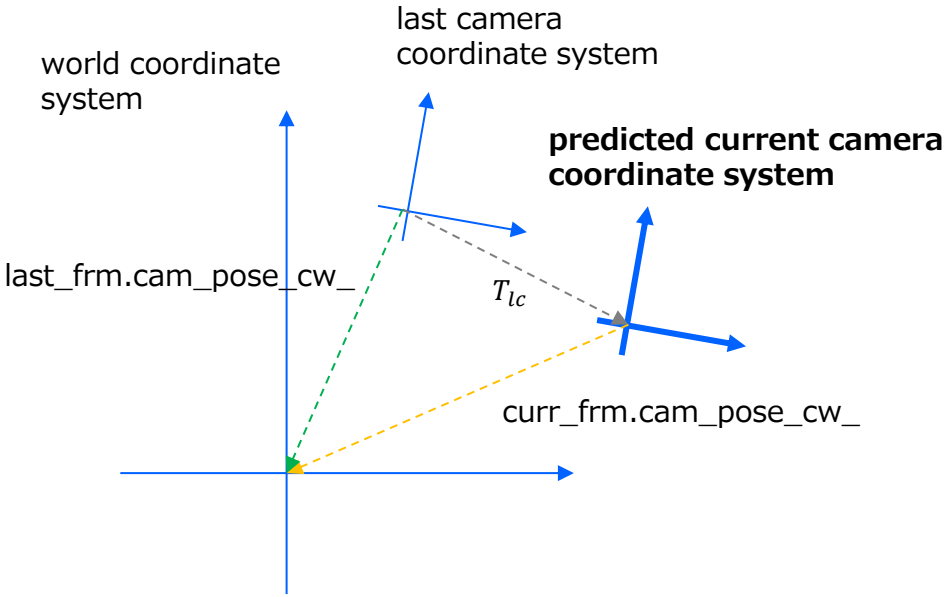


projection::match_current_and_last_frames – 1 of 7

- Determine assume_forward, assume_backward, or neither
- For each of all landmarks in the last frame:
  - Call perspective::reproject_to_image
    - Reproject the landmark to the current frame
    - Fail if reprojected point is outside of the camera screen
  - Call get_keypoints_in_cell
    - Search candidate keypoints using grid cells
    - Fail if no keypoint available
  - Find the best matched keypoint from the candidates
- Finally do angle check to reject outliers

projection::match_current_and_last_frames – 2 of 7

$$
\begin{aligned}
T_{lc} &= last_frm.cam_pose_cw_ \cdot curr_frm.cam_pose_cw_^{-1} \\
&= \begin{pmatrix} R_{last_cw} & \boldsymbol{t}_{last_cw} \\ 0 & 1 \end{pmatrix} \begin{pmatrix} R_{curr_cw}^T & -R_{curr_cw}^T \boldsymbol{t}_{curr_cw} \\ 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} R_{last_cw} R_{curr_cw}^T & -R_{last_cw} R_{curr_cw}^T \boldsymbol{t}_{curr_cw} + \boldsymbol{t}_{last_cw} \\ 0 & 1 \end{pmatrix} \\
\therefore \boldsymbol{t}_{lc} &= -R_{last_cw} R_{curr_cw}^T \boldsymbol{t}_{curr_cw} + \boldsymbol{t}_{last_cw}
\end{aligned}
$$

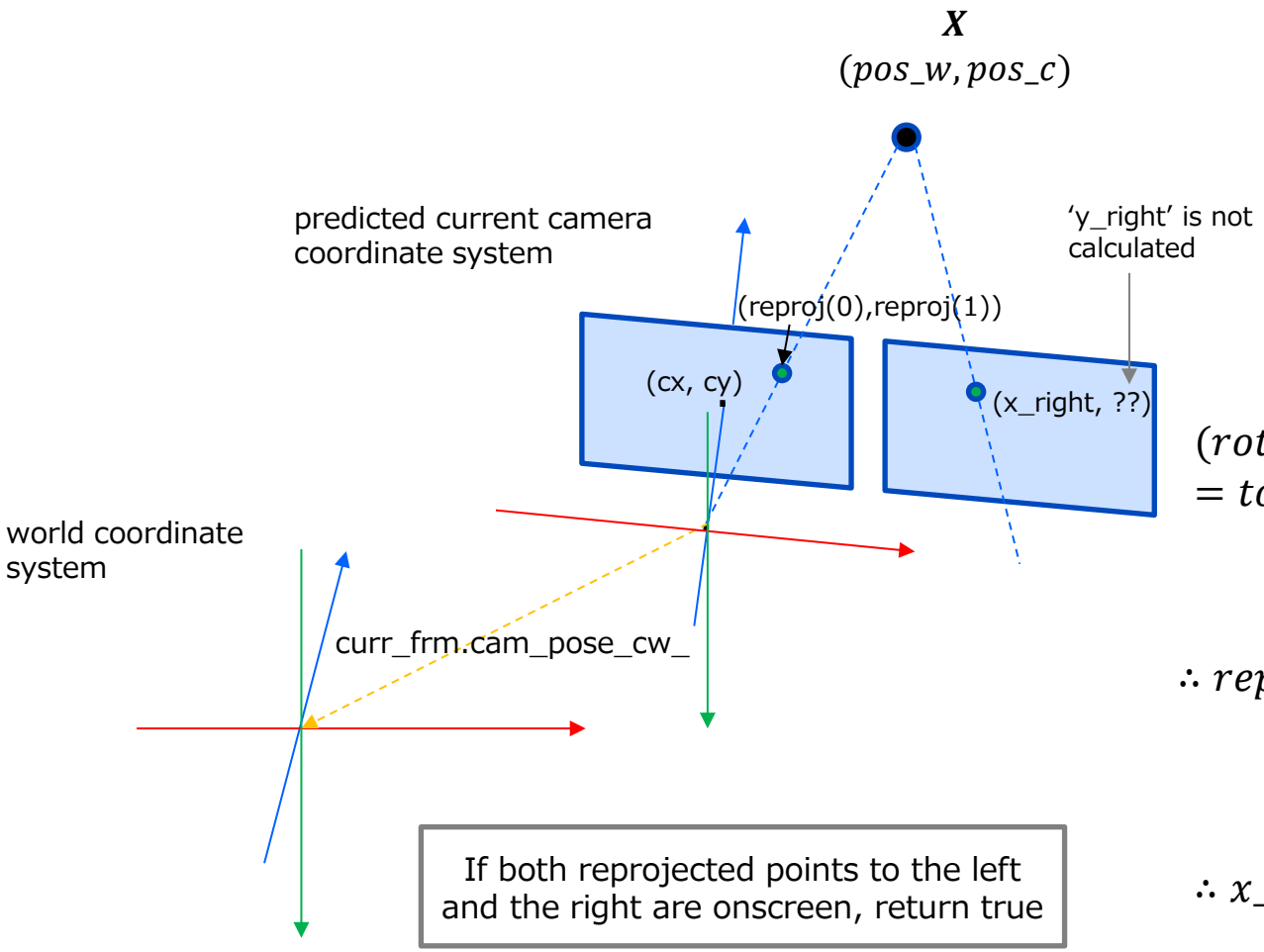


assume_forward:
true if z component of $\boldsymbol{t}_{lc}$ is greater than stereo baseline.
assume_backward:
true if z component of $\boldsymbol{t}_{lc}$ is lower than -1 x (stereo baseline).

assume_forward and assume_backward will be used to restrict scale level of searched keyframes in matching keypoints between the last and the current frames.

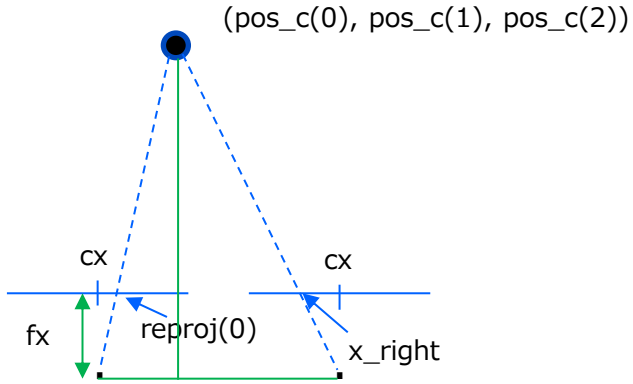
projection::match_current_and_last_frames – 3 of 7

<u>perspective::reproject_to_image</u>



If both reprojected points to the left and the right are onscreen, return true

Reproject landmarks which were observed from the last frame to the current frame of predicted pose.



$$pos_c = rot_cw \cdot pos_w + trans_cw$$

$$(rot_cw = top\ left\ 3x3\ matrix\ of\ curr_frm.cam_pose_cw_, trans_cw = top\ right\ 3x1\ vector\ of\ it)$$

$$\frac{reproj(0) - cx}{fx} = \frac{pos_c(0)}{pos_c(2)}$$

$$\therefore reproj(0) = fx\frac{pos_c(0)}{pos_c(2)} + cx, similary\ reproj(1) = fy\frac{pos_c(1)}{pos_c(2)} + cy$$

$$\frac{cx - x_right}{fx} = \frac{baseline - pos_c(0)}{pos_c(2)}$$

$$\therefore x_right = -fx\frac{baseline - pos_c(0)}{pos_c(2)} + cx = reproj(0) - fx\frac{baseline}{pos_c(2)}$$

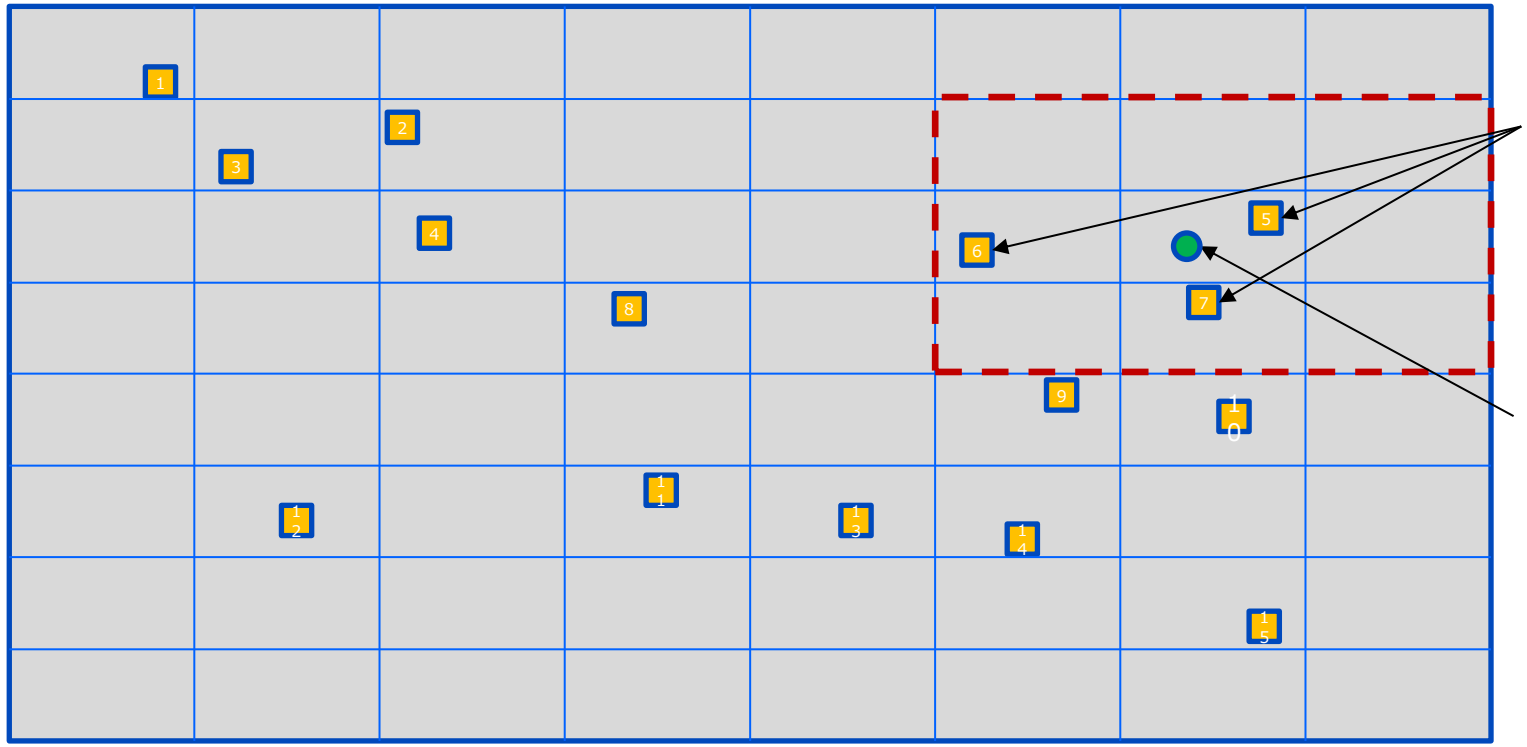
projection::match_current_and_last_frames – 4 of 7

get_keypoints_in_cell

smaller scale level corresponds to lower pyramid level, that is larger image size.

Search corresponding keypoint candidates from an adequate range of region.
If assume_forward is true, keypoints whose scale level is equal to or smaller than that of searching keypoint can be candidates.
If assume_backward is true, keypoints whose scale level is equal to or larger can be candidates.
Else keypoints whose scale level is from that of searching keypoint – 1 to +1 can be candidates.

grid cells of the current frame (see assign_keypoints_to_grid)

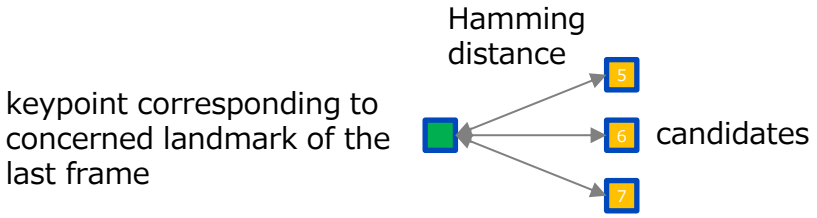


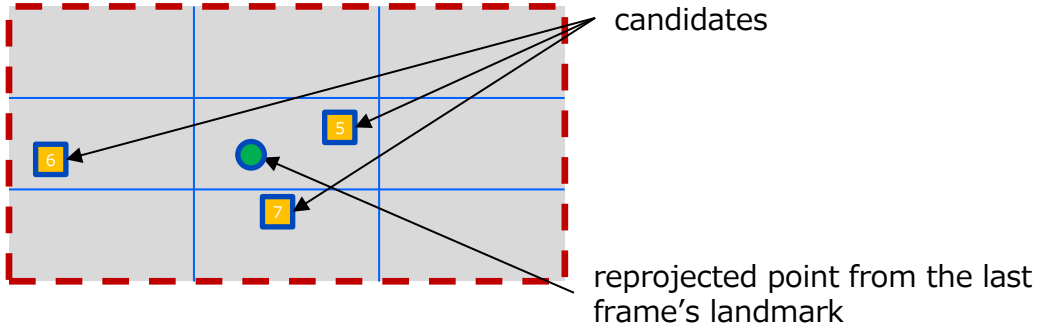
candidate keypoints of the left image of the current frame

reprojected point from landmark observed by the last frame to the current frame of the predicted pose. x,y coordinate is (reproj(0), reproj(1)) obtained in the previous slide.

projection::match_current_and_last_frames – 5 of 7

- Find the best matched keypoint from the candidates obtained in the previous slide
- For each candidate:
  - Reprojection error of x direction on the right image must be within a threshold calculated by using scale factor of the keyframe
  - Calculate Hamming distance between the keypoint corresponding to concerned landmark of the last frame and that of the candidate
  - The distance must be the smallest among all the candidates and below HAMMING_DIST_THR_HIGH

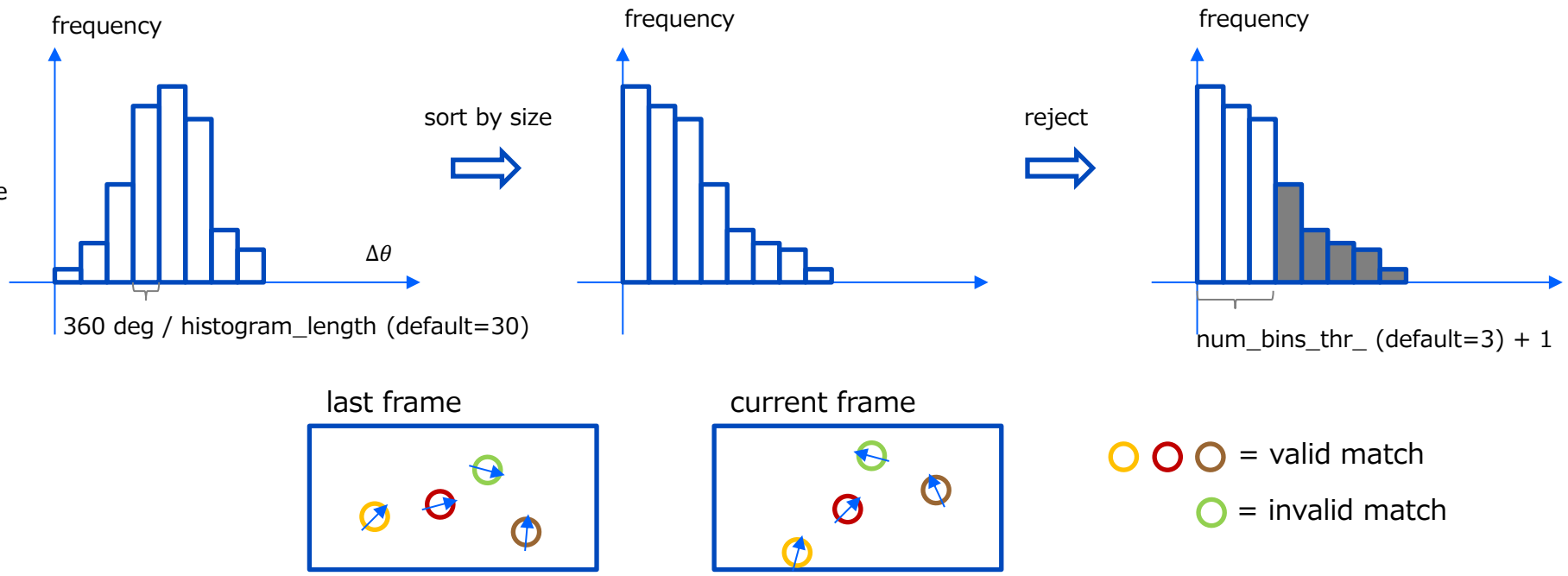




projection::match_current_and_last_frames – 6 of 7

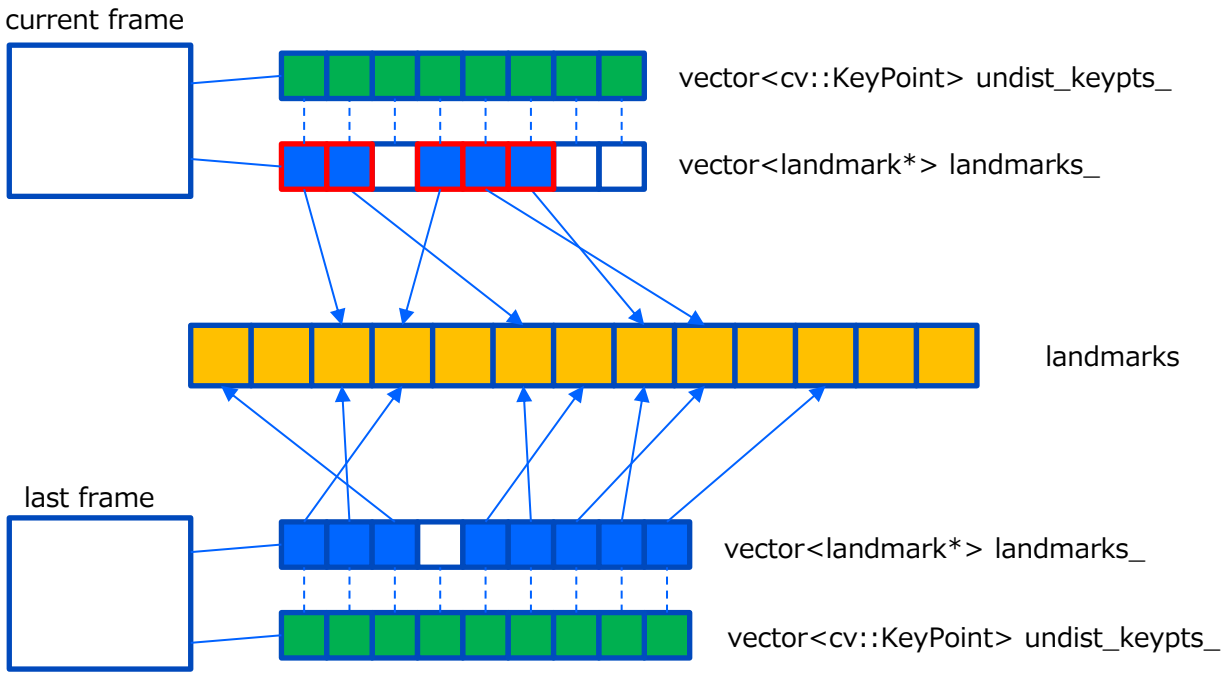
- At this point of time, all candidate keypoints of the current frame corresponding to the landmarks of the last frame have been collected
- Among them, reject keypoints whose angle check are not passed
  - It is expected that validly matched keypoints have similar ORB angle difference between the last and the current frame

$\Delta\theta$=difference of ORB descriptor angle between the keypoint reprojected from landmark in the last frame and its corresponding candidate.



projection::match_current_and_last_frames – 7 of 7

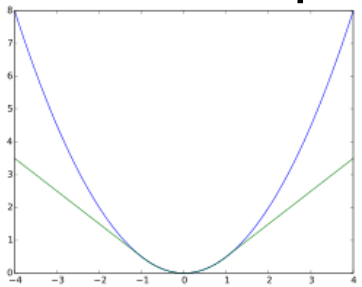
- At this point of time, correspondences of landmarks between the last and the current frame are obtained like below:



Elements of landmarks vector of the current frame whose value are not NULL have correspondence with the last frame (high-lighted with red).

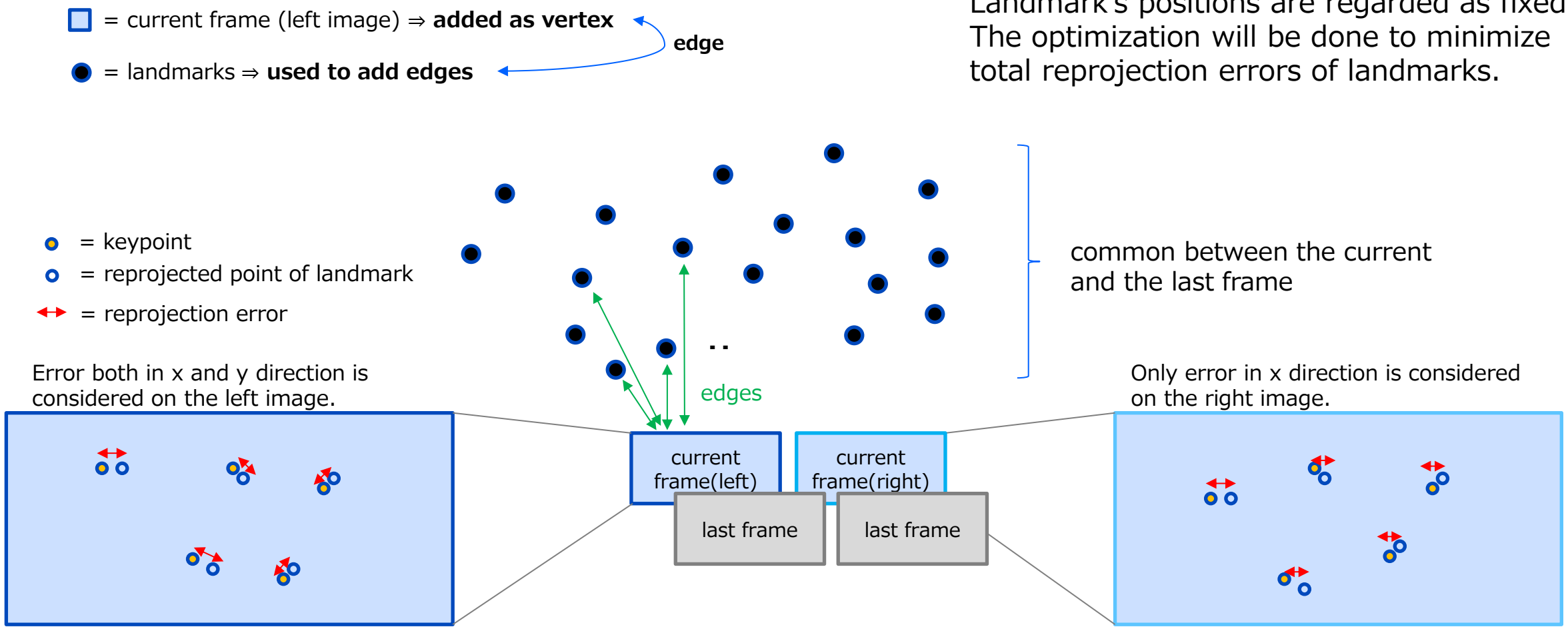
pose_optimizer::optimize – 1 of 9

- Optimize with g2o
  - Linear solver = LinearSolverEigen
  - Block solver = BlockSolver_6_3
  - Algorithm = OptimizationAlgorithmLevenberg
  - Optimizer = SparseOptimizer
- Vertex = current frame pose
- Edges = restriction between current frame pose and reprojected points of landmarks
  - Robust kernel (Huber loss) is set with delta = sqrt of χ^2 value of threshold of significance level of 5%



pose_optimizer::optimize – 2 of 9

The current frame pose will be optimized through g2o optimization.
Landmark's positions are regarded as fixed.
The optimization will be done to minimize total reprojection errors of landmarks.



Note that the error only on the left image is considered if the reprojected point on the right image is out of screen. See next slide in detail.

pose_optimizer::optimize – 3 of 9

= keypoint

= reprojected point of landmark

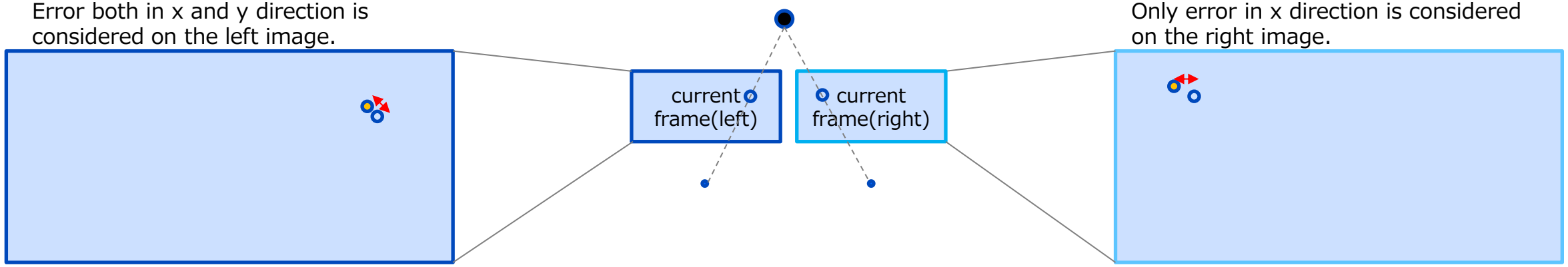
= reprojection error

stereo visible

Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i, x_right_i - reproj_x_right_i)^T$

Error both in x and y direction is considered on the left image.

Only error in x direction is considered on the right image.

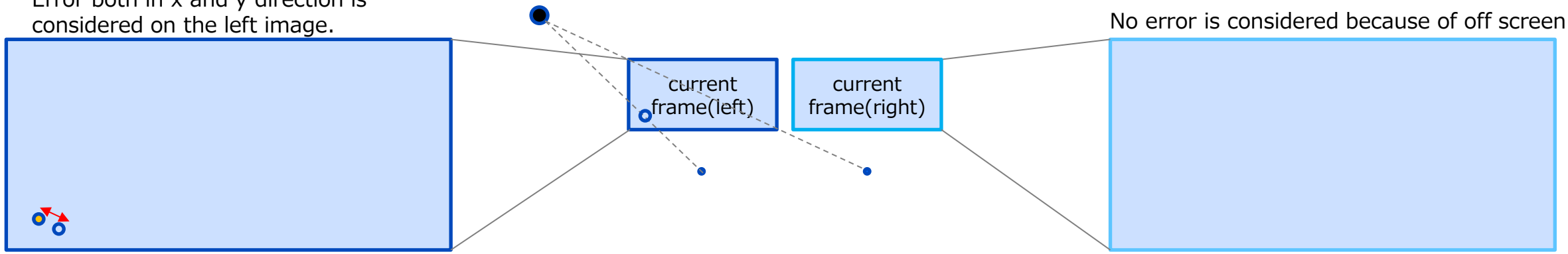


monocular only visible

Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i)^T$

Error both in x and y direction is considered on the left image.

No error is considered because of off screen.



pose_optimizer::optimize – 4 of 9

- Vertex details (g2o::se3::shot_vertex)
  - Derived class of ::g2o::BaseVertex<6, ::g2o::SE3Quat>
  - Predicted current frame pose (cam_pose_cw_) is set through setEstimate at first
  - oplusImpl does setEstimate with $\exp(\Delta) * \mathrm{estimate}()$

pose_optimizer::optimize – 5 of 9

- Edge details (stereo_perspective_pose_opt_edge derived from ::g2o::BaseUnaryEdge<3, Vec3_t, shot_vertex>):

  used when stereo visible

  - Measurement is (x_i, y_i, x_right_i)
  - Information matrix is diagonal matrix whose components value depend on pyramid level of the keypoint
  - Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i, x_right_i - reproj_x_right_i)^T$

$$= \left(x_i - f_x \frac{X_c^i}{Z_c^i} - c_x, y_i - f_y \frac{Y_c^i}{Z_c^i} - c_y, x_right_i - f_x \frac{X_c^i}{Z_c^i} - c_x + f_x \frac{baseline}{Z_c^i}\right)^T$$

written in computeError() of perspective_pose_opt_edge.h

  - Jacobian is $J = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$ *where* $\boldsymbol{p}$ *represents camera pose*

- $$\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} = \begin{pmatrix} -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \\ -\frac{f_x}{Z_c} & 0 & \frac{f_x}{Z_c^2}(X_c - baseline) \end{pmatrix}, \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} 0 & Z_c & -Y_c & 1 & 0 & 0 \\ -Z_c & 0 & X_c & 0 & 1 & 0 \\ Y_c & -X_c & 0 & 0 & 0 & 1 \end{pmatrix}$$

written in linearizeOplus() of perspective_pose_opt_edge.cc (oplus = ⊕)

- $$J = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} \frac{f_x X_c Y_c}{Z_c^2} & -f_x(1 + \frac{X_c^2}{Z_c^2}) & \frac{f_x Y_c}{Z_c} & -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ f_y(1 + \frac{Y_c^2}{Z_c^2}) & -\frac{f_y X_c Y_c}{Z_c^2} & -\frac{f_y X_c}{Z_c} & 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \\ \frac{f_x Y_c}{Z_c^2}(X_c - baseline) & -f_x(1 + \frac{X_c^2}{Z_c^2} - \frac{X_c \cdot baseline}{Z_c^2}) & \frac{f_x Y_c}{Z_c} & -\frac{f_x}{Z_c} & 0 & \frac{f_x}{Z_c^2}(X_c - baseline) \end{pmatrix}$$

pose_optimizer::optimize – 6 of 9

- Edge details (mono_perspective_pose_opt_edge derived from ::g2o::BaseUnaryEdge<2, Vec2_t, shot_vertex>):

  used when only monocular visible

  - Measurement is (x_i, y_i)
  - Information matrix is diagonal matrix whose components value depend on pyramid level of the keypoint
  - Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i)^T$

$$= \left(x_i - f_x \frac{X_c^i}{Z_c^i} - c_x, y_i - f_y \frac{Y_c^i}{Z_c^i} - c_y\right)^T$$

written in computeError() of perspective_pose_opt_edge.h

  - Jacobian is $J = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$ *where* $\boldsymbol{p}$ *represents camera pose*

    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} = \begin{pmatrix} -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \end{pmatrix}, \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} 0 & Z_c & -Y_c & 1 & 0 & 0 \\ -Z_c & 0 & X_c & 0 & 1 & 0 \\ Y_c & -X_c & 0 & 0 & 0 & 1 \end{pmatrix}$

    - $J = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} \frac{f_x X_c Y_c}{Z_c^2} & -f_x(1 + \frac{X_c^2}{Z_c^2}) & \frac{f_x Y_c}{Z_c} & -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ f_y(1 + \frac{Y_c^2}{Z_c^2}) & -\frac{f_y X_c Y_c}{Z_c^2} & -\frac{f_y X_c}{Z_c} & 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \end{pmatrix}$

written in linearizeOplus() of perspective_pose_opt_edge.cc (oplus = $\oplus$)

pose_optimizer::optimize – 7 of 9

Additional explanation for $\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$:

$\boldsymbol{p}$ is camera pose represented by $(\omega_1, \omega_2, \omega_3, t_x, t_y, t_z)$ in which $(\omega_1, \omega_2, \omega_3)$ represents rotation axis and angle, and (t_x, t_y, t_z) represents translation vector described in the camera coordinate.

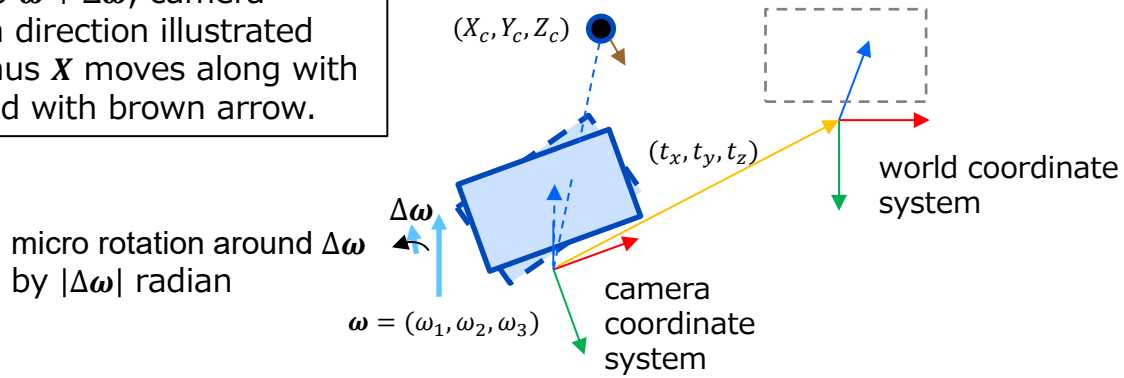
When rotating camera pose by micro rotation of $|\Delta\boldsymbol{\omega}|$ radian around $\Delta\boldsymbol{\omega}$, (X_c, Y_c, Z_c) moves as follows:

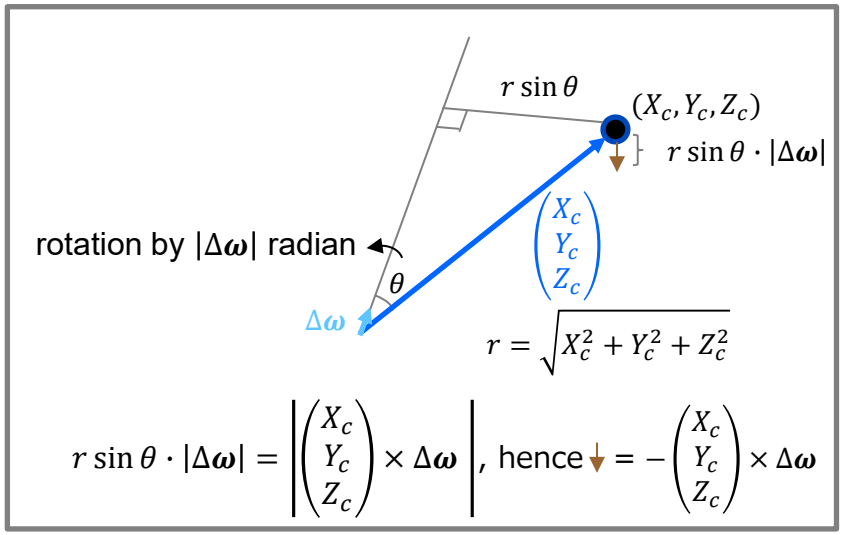
$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \rightarrow \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} - \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \times \Delta\boldsymbol{\omega} \text{ where } \Delta\boldsymbol{\omega} = \begin{pmatrix} \Delta\omega_1 \\ \Delta\omega_2 \\ \Delta\omega_3 \end{pmatrix}$$

$$\therefore \Delta\boldsymbol{X}_c = -\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \times \Delta\boldsymbol{\omega} = -\begin{pmatrix} 0 & -Z_c & Y_c \\ Z_c & 0 & -X_c \\ -Y_c & X_c & 0 \end{pmatrix} \Delta\boldsymbol{\omega} = \begin{pmatrix} 0 & Z_c & -Y_c \\ -Z_c & 0 & X_c \\ Y_c & -X_c & 0 \end{pmatrix} \Delta\boldsymbol{\omega}$$

$$\therefore \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{\omega}} = \begin{pmatrix} 0 & Z_c & -Y_c \\ -Z_c & 0 & X_c \\ Y_c & -X_c & 0 \end{pmatrix}$$

When $\boldsymbol{\omega}$ increases to $\boldsymbol{\omega} + \Delta\boldsymbol{\omega}$, camera rotates along with a direction illustrated with black arrow, thus $\boldsymbol{X}$ moves along with a direction illustrated with brown arrow.





pose_optimizer::optimize – 8 of 9

Additional explanation for $\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$ (continued):

$\boldsymbol{p}$ is camera pose represented by $(\omega_1, \omega_2, \omega_3, t_x, t_y, t_z)$ in which $(\omega_1, \omega_2, \omega_3)$ represents rotation axis and angle, and (t_x, t_y, t_z) represents translation vector described in the camera coordinate.

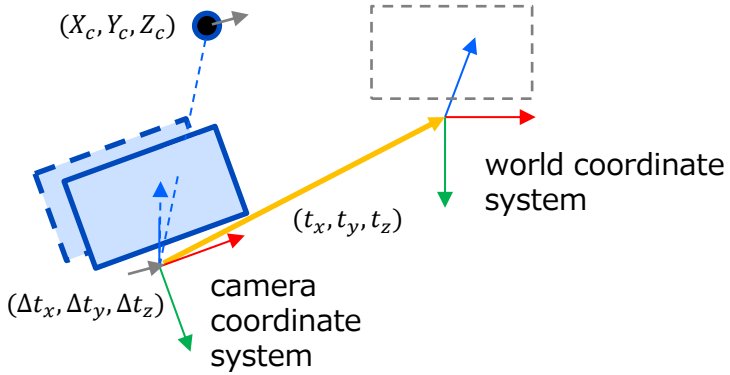
When translating camera pose by micro translation of $(\Delta t_x, \Delta t_y, \Delta t_z)$, (X_C, Y_C, Z_C) moves as follows:

$$\begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \rightarrow \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} + \begin{pmatrix} \Delta t_x \\ \Delta t_y \\ \Delta t_z \end{pmatrix}$$

$$\therefore \Delta \boldsymbol{X}_c = \begin{pmatrix} \Delta t_x \\ \Delta t_y \\ \Delta t_z \end{pmatrix} = \Delta \boldsymbol{t}$$

$$\therefore \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{t}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

When $\boldsymbol{t}$ increases by $\Delta \boldsymbol{t}$, camera moves relatively by $-\Delta \boldsymbol{t}$, thus $\boldsymbol{X}$ moves to $\boldsymbol{X} + \Delta \boldsymbol{t}$.



pose_optimizer::optimize – 9 of 9

- Do optimization
  - Call optimizer.optimize num_trials_ (=4) times if not broken
  - Edges whose χ^2 value is greater than threshold of significance level of 5% are rejected
  - Loop is broken when the number of all edges minus the number of rejected objects is less than 5 (hard coded value)
    - In other words, too many outliers
    - This is failed case
  - Robust kernel is disabled at the last loop
- Set optimized pose to the current frame pose

discard_outliers

- Just count landmarks which were not rejected during optimization
- If the count is below num_matches_thr_ (=10), motion_based_track returns false
  - try bow_match_based_track next
- Else returns true

TRACKING – TRACK – TRACK_CURRENT_FRAME – BOW_MATCH_BASED_TRACK

bow_match_based_track – 1 of 4

- bow_match_based_track(data::frame& curr_frm, const data::frame& last_frm, data::keyframe* ref_keyfrm)
  - Try to match ORB descriptors of the current frame and those of the reference keyframe using Bag of Words

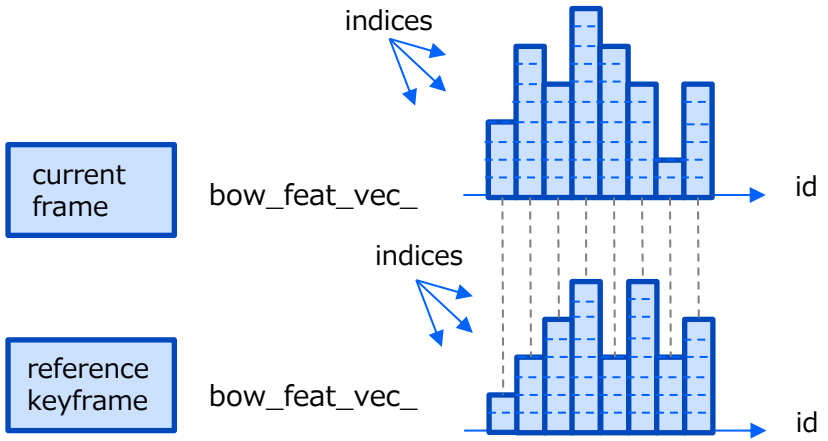
bow_match_based_track – 2 of 4

- Call
  - frame::compute_bow for the current frame
    - bow is calculated by DBoW2 or FBoW
  - bow_tree::match_frame_and_keyframe
    - Search ORB matches between the current frame and the reference keyframe
    - If the number of matches is less than num_matches_thr_ (=10), bow_match_based_track returns false
  - pose_optimizer::optimize
    - Already explained in pose_optimizer::optimize – 1 of 9 to pose_optimizer::optimize – 9 of 9
  - discard_outliers
    - Already explained in discard_outliers
- If the number of inliers is less than num_matches_thr_, return false
- Else return true

bow_match_based_track – 3 of 4

bow_tree::match_frame_and_keyframe

- Use bow_feat_vec_ which has ORB descriptor indices per node id
- For each descriptor index in each node id of the reference keyframe, find the best matched descriptor index from corresponding node id of the current frame using Hamming distance
  - Hamming distance must be the smallest among all candidates and below HAMMING_DIST_THR_LOW and lower than 0.7 * the second best
- Copy the landmark of the reference keyframe corresponding to descriptor index for which the best match with the current frame was found to matched_lms_in_curr
  - Landmarks in matched_lms_in_curr will be used as the current frame's landmarks after returned to caller of bow_match_based_track if succeeded
- At last, do angle check explained in projection::match_current_and_last_frames – 6 of 7



bow_match_based_track – 4 of 4

- set current pose as the same as the last pose
- Call optimizer::optimize
- Then call discard_outliers
  - discard_outliers returns the count of valid matches
- If the count is below num_matches_thr_ (=10), bow_match_based_track returns false
  - try robust_match_based_track next

TRACKING – TRACK – TRACK_CURRENT_FRAME – ROBUST_MATCH_BASED_TRACK

robust_match_based_track – 1 of 8

- robust_match_based_track(data::frame& curr_frm, const data::frame& last_frm, data::keyframe* ref_keyfrm)
  - Try to match ORB descriptors of the current frame and those of the reference keyframe with brute force match

robust_match_based_track – 2 of 8

- Call
  - robust::match_frame_and_keyframe
    - Search ORB matches between the current frame and the reference keyframe
    - If the number of matches is less than num_matches_thr_ (=10), robust_match_based_track returns false
  - pose_optimizer::optimize
    - Already explained in pose_optimizer::optimize – 1 of 9 to pose_optimizer::optimize – 9 of 9
  - discard_outliers
    - Already explained in discard_outliers
- If the number of inliers is less than num_matches_thr_, return false
- Else return true

robust_match_based_track – 3 of 8

robust::match_frame_and_keyframe

- Find the best matched descriptors of the current frame from those of the reference keyframe by brute forth matching using Hamming distance
  - Hamming distance must be the smallest among all candidates and below HAMMING_DIST_THR_LOW and lower than 0.8 * the second best
- Reject outliers by RANSAC using Essential matrix
  - explained in the next few slides
- Count and register landmarks corresponding to descriptors which were not rejected as outliers

robust_match_based_track – 4 of 8

robust::match_frame_and_keyframe

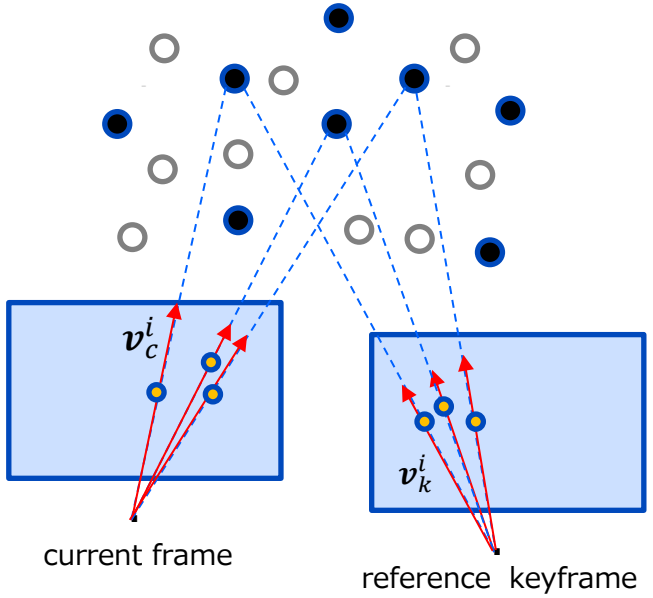
- Reject outliers by RANSAC using Essential matrix
  1. Compute Essential matrix between the current frame and the reference keyframe with 8 points algorithm using randomly selected 8 keypoint pairs
  2. Classify all pairs as inlier or outlier by calculating epipolar constraint using bearing vectors
  3. Go back to 1 until max_num_iter loops
  4. Select the best iteration which scores the most inliers and count them
  5. If the number of inliers is higher or equal to min_set_size (=8), the solution is regarded as success
- Note that Essential matrix is calculated just for the purpose of classifying inliers
  - The current frame pose will be calculated by optimizer::optimize hereafter

robust_match_based_track – 5 of 8

robust::match_frame_and_keyframe

- **Reject outliers by RANSAC using Essential matrix**
  1. Compute Essenti
     points algorithm
  2. Classify all pairs
  3. Go back to 1 unt
  4. Select the best it
  5. If the number of
     success
- Note that essential n
  calculating the current frame pose

Select eight keypoint pairs randomly



Compute Essential matrix

Count inliers among all pairs except the eight

Find the best Essential matrix which scores the most inliers

robust_match_based_track – 6 of 8

robust::match_frame_and_keyframe

- Reject outliers by RANSAC using Essential matrix
  1. **Compute Essential matrix between the current frame and the reference keyframe with 8 points algorithm using randomly selected 8 keypoint pairs**
  2. Classify all pairs as inlier or outlier by calculating epipolar constraint using hearin
  3. Go back to 1 until ma
  4. Select the best iterat
  5. If the number of inlie success

compute_E_21

$$\boldsymbol{v}_k^{i\,T}\begin{pmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{pmatrix} v_c^i = \boldsymbol{v}_k^{i\,T} E v_c^i = 0 \; where \; \boldsymbol{v}_k^i = \begin{pmatrix} x_k^i \\ y_k^i \\ z_k^i \end{pmatrix} \Big/ \left\| \begin{pmatrix} x_k^i \\ y_k^i \\ z_k^i \end{pmatrix} \right\|, \boldsymbol{v}_c^i = \begin{pmatrix} x_c^i \\ y_c^i \\ z_c^i \end{pmatrix} \Big/ \left\| \begin{pmatrix} x_c^i \\ y_c^i \\ z_c^i \end{pmatrix} \right\|$$ for i=0 to 7

$$\therefore \begin{pmatrix} x_k^1 x_c^1 & x_k^1 y_c^1 & x_k^1 z_c^1 & \cdots & z_k^1 z_c^1 \\ x_k^2 x_c^2 & x_k^2 y_c^2 & x_k^2 z_c^2 & & z_k^2 z_c^2 \\ & \vdots & & \ddots & \vdots \\ x_k^7 x_c^7 & x_k^7 y_c^7 & x_k^7 z_c^7 & \cdots & z_k^7 z_c^7 \end{pmatrix} \begin{pmatrix} e_{11} \\ e_{12} \\ e_{13} \\ \vdots \\ e_{33} \end{pmatrix} = \mathrm{A}\boldsymbol{e} = 0$$

$\boldsymbol{e}$ is constant * eigen vector corresponding to minimum eigen value of $A^T\mathrm{A}$.
Applying singular value decomposition to A, we obtain $A\boldsymbol{e} = U\Lambda V^T\boldsymbol{e} = 0$
$\boldsymbol{e}$ is 9th column vector of V.

E is obtained by rearranging $\boldsymbol{e}$ and can be singular value decomposed as $E = U'\Lambda' V'^T \; where \; \Lambda' = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix}$

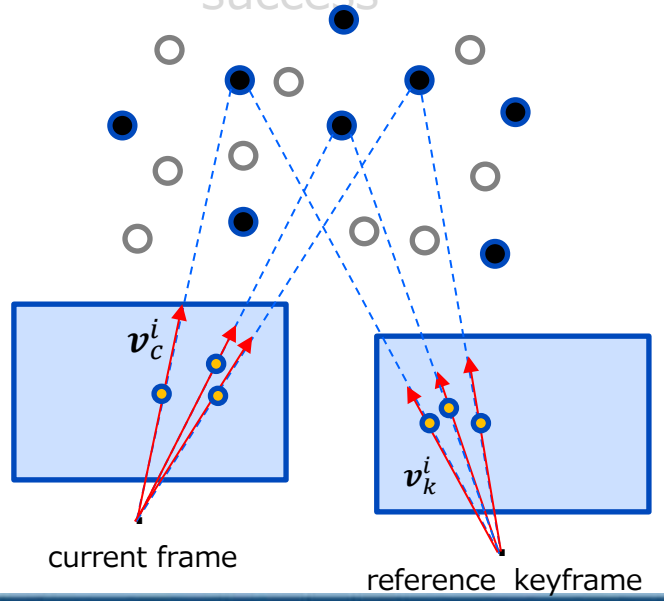
E has 2 eigen value because it is rank2 matrix, so force to set $\lambda_3 = 0$ to make Λ'':

$$\Lambda' = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{pmatrix} \Rightarrow \Lambda'' = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

essential matrix $E = U'\Lambda''V'^T$

$E = [T_\times]R = \begin{pmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{pmatrix} R$, considering $[T_\times]$ is a linear combination of the other two columns, $[T_\times]$ is rank 2 thus E is also rank 2.

1.

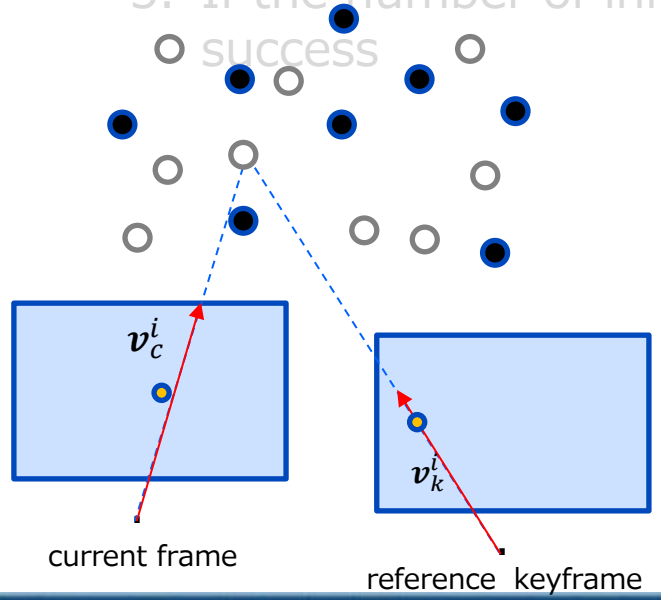


robust_match_based_track – 7 of 8

robust::match_frame_and_keyframe

- Reject outliers by RANSAC using Essential matrix
  1. Compute Essential matrix between the current frame and the reference keyframe with 8 points algorithm using randomly selected 8 keypoint pairs
  2. **Classify all pairs as inlier or outlier by calculating epipolar constraint using bearing vectors**
  3. Go back to 1 until max_num_iter loops
  4. Select the best iterat
  5. If the number of inlie
     success

2.



check_inliers

For each pair not participating to calculate the essential matrix E, if it is correct pair, ${v_k^i}^T E v_c^i \cong 0$ is true.
Consequently, $v_k^i \cdot E v_c^i \cong 0$ and $v_c^i \cdot E^T v_k^i \cong 0$ are also true.
If these are not true, the pair can be regarded as outlier.

Outlier criterion:

if $\frac{|v_k^i \cdot E v_c^i|}{|v_k^i||E v_c^i|} > 0.01745240643\ (\cos(89degree))\ or\ \frac{|v_c^i \cdot E^T v_k^i|}{|v_c^i||E^T v_k^i|} > \cos(89\text{degree})$ is true

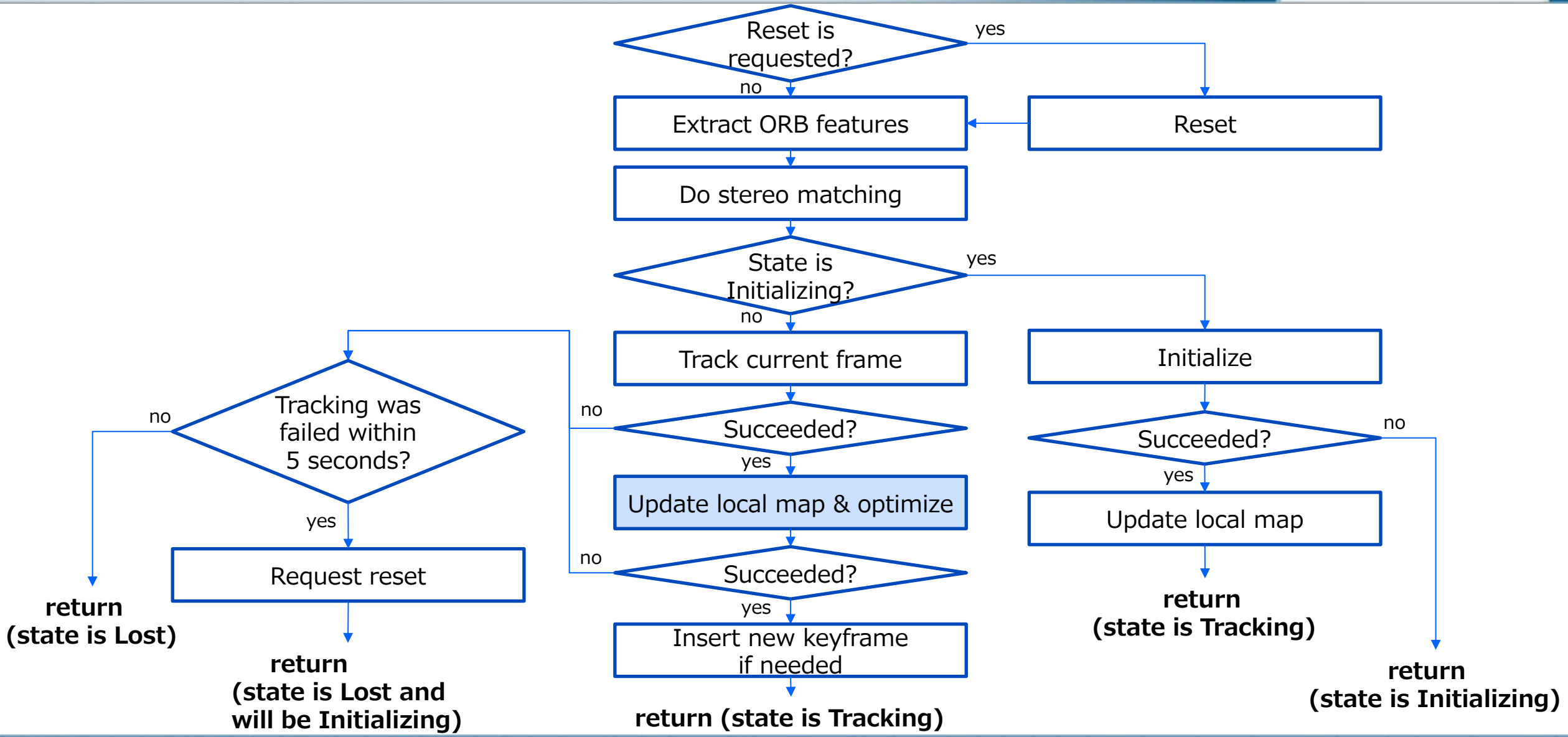
else Inlier

robust_match_based_track – 8 of 8

- set current pose as the same as the last pose
- Call optimizer::optimize
- Then call discard_outliers
  - discard_outliers returns the count of valid matches
- If the count is below num_matches_thr_ (=10), robust_match_based_track returns false
  - Will be LOST

TRACKING – TRACK – UPDATE_LOCAL_MAP

Where we are


Reset is requested?
yes
no
Reset
Extract ORB features
Do stereo matching
State is Initializing?
yes
no
Initialize
Track current frame
Succeeded?
no
yes
Update local map
Succeeded?
no
yes
Update local map & optimize
Succeeded?
no
yes
Insert new keyframe if needed
Tracking was failed within 5 seconds?
no
yes
Request reset
return (state is Lost)
return (state is Lost and will be Initializing)
return (state is Tracking)
return (state is Tracking)
return (state is Initializing)

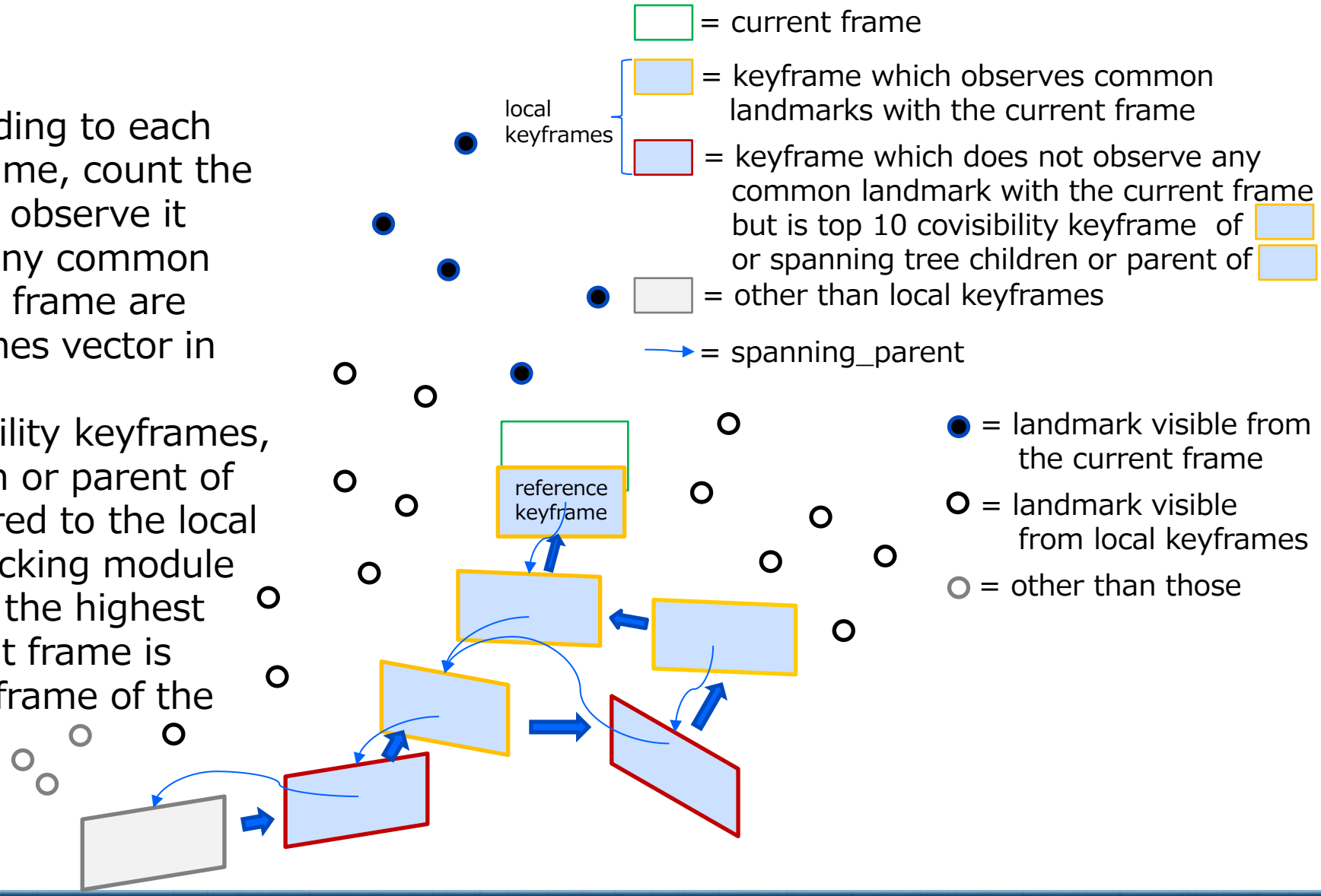
update_local_map – 1 of 3

- Call
  - update_local_keyframes
  - update_local_landmarks
  - map_database::set_local_landmarks
    - Just copy local landmarks vector from the tracking module to map database

update_local_map – 2 of 3

update_local_keyframes

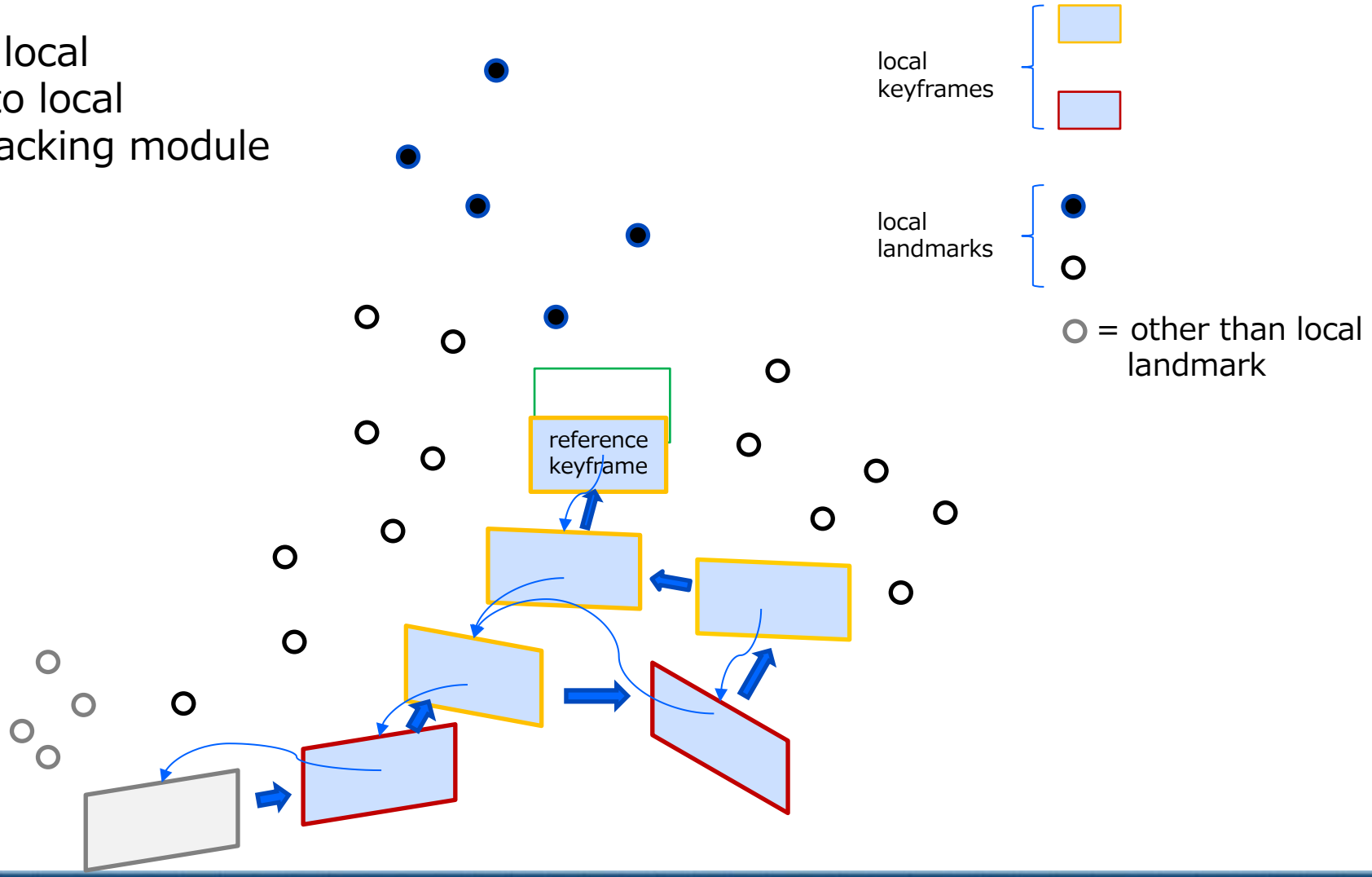
- For a landmark corresponding to each keypoint of the current frame, count the number of keyframes that observe it
- Keyframs which observe any common landmark with the current frame are registered to local keyframes vector in the tracking module
- In addition, top 10 covisibility keyframes, and spanning tree children or parent of the keyframes are registered to the local keyframe vector in the tracking module
- Local keyframe which has the highest covisibility with the current frame is specified as reference keyframe of the tracking module



update_local_map – 3 of 3

update_local_landmarks

- All landmarks visible from local keyframes are registered to local landmarks vector in the tracking module



TRACKING – TRACK – OPTIMIZE_CURRENT_FRAME_WITH_LOCAL_MAP

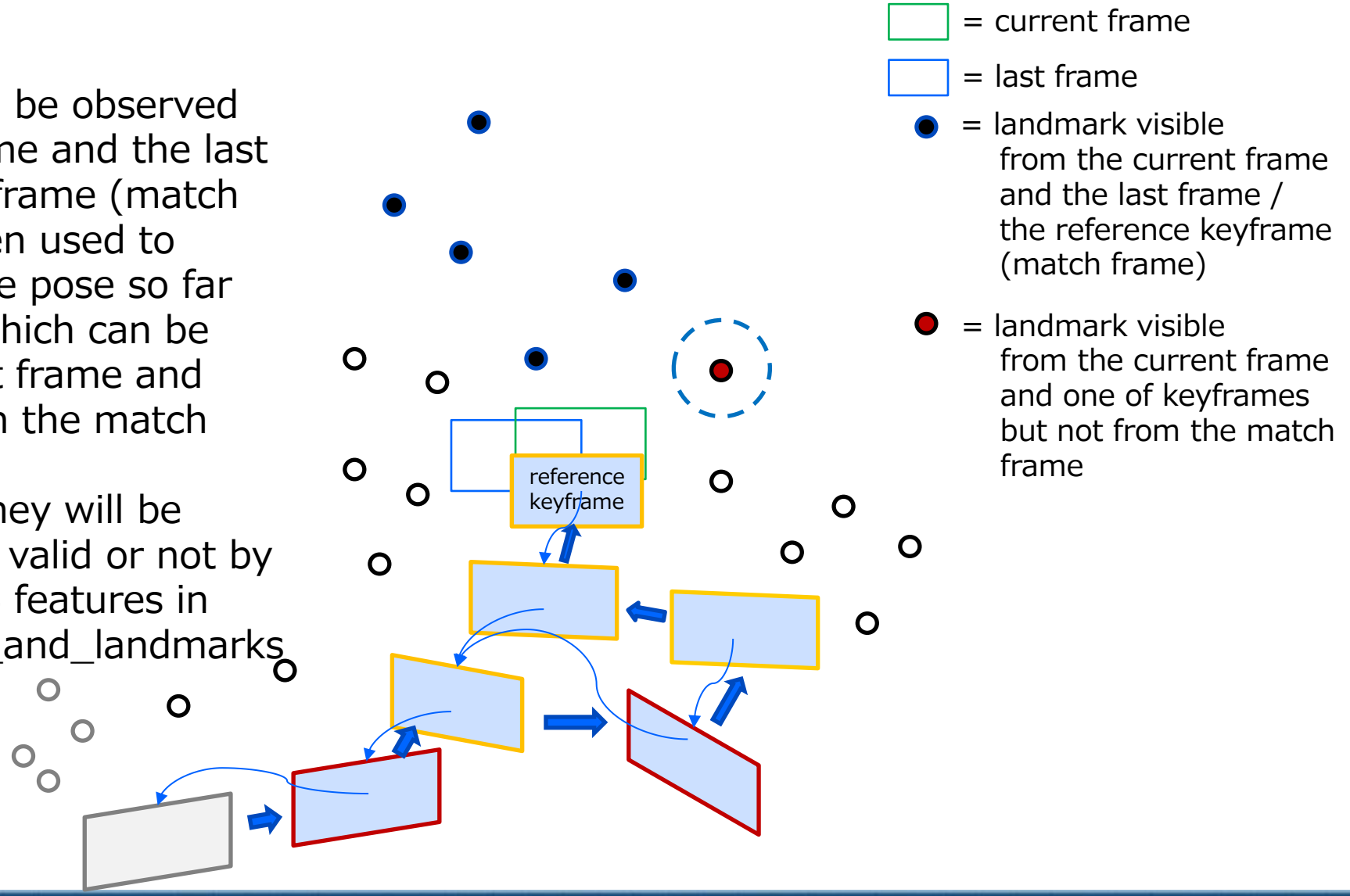
optimize_current_frame_with_local_map – 1 of 5

- Call
  - search_local_landmarks
    - projection::match_frame_and_landmarks
  - pose_optimizer::optimize
    - Call again because current keypoints may have increased by search_local_landmarks
    - Already explained in pose_optimizer::optimize – 1 of 9 to pose_optimizer::optimize – 9 of 9
- Count the number of tracked landmarks
- If the number is below num_tracked_lms_thr (=20), then return false
  - Will be LOST
  - If recently relocalized (within 1 second after previous relocalization), the threshold value of 20 is multiplied by 2
- Else return true

optimize_current_frame_with_local_map – 2 of 5

search_local_landmarks

- Only landmarks which can be observed both from the current frame and the last frame / the reference keyframe (match frame hereafter) have been used to calculate the current frame pose so far
- Search more landmarks which can be observed from the current frame and other local keyframes than the match frame
- If candidates are found, they will be checked whether they are valid or not by Hamming distance of ORB features in projection::match_frame_and_landmarks

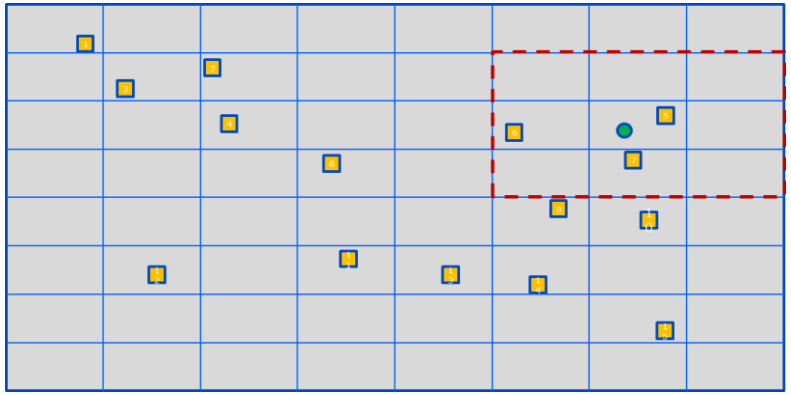


Note:
match frame is:
- the last frame in case tracking was done by motion_based_track
- the reference keyframe in case tracking was done by bow_match_based_track or robust_match_based_track

optimize_current_frame_with_local_map – 3 of 5

projection::match_frame_and_landmarks

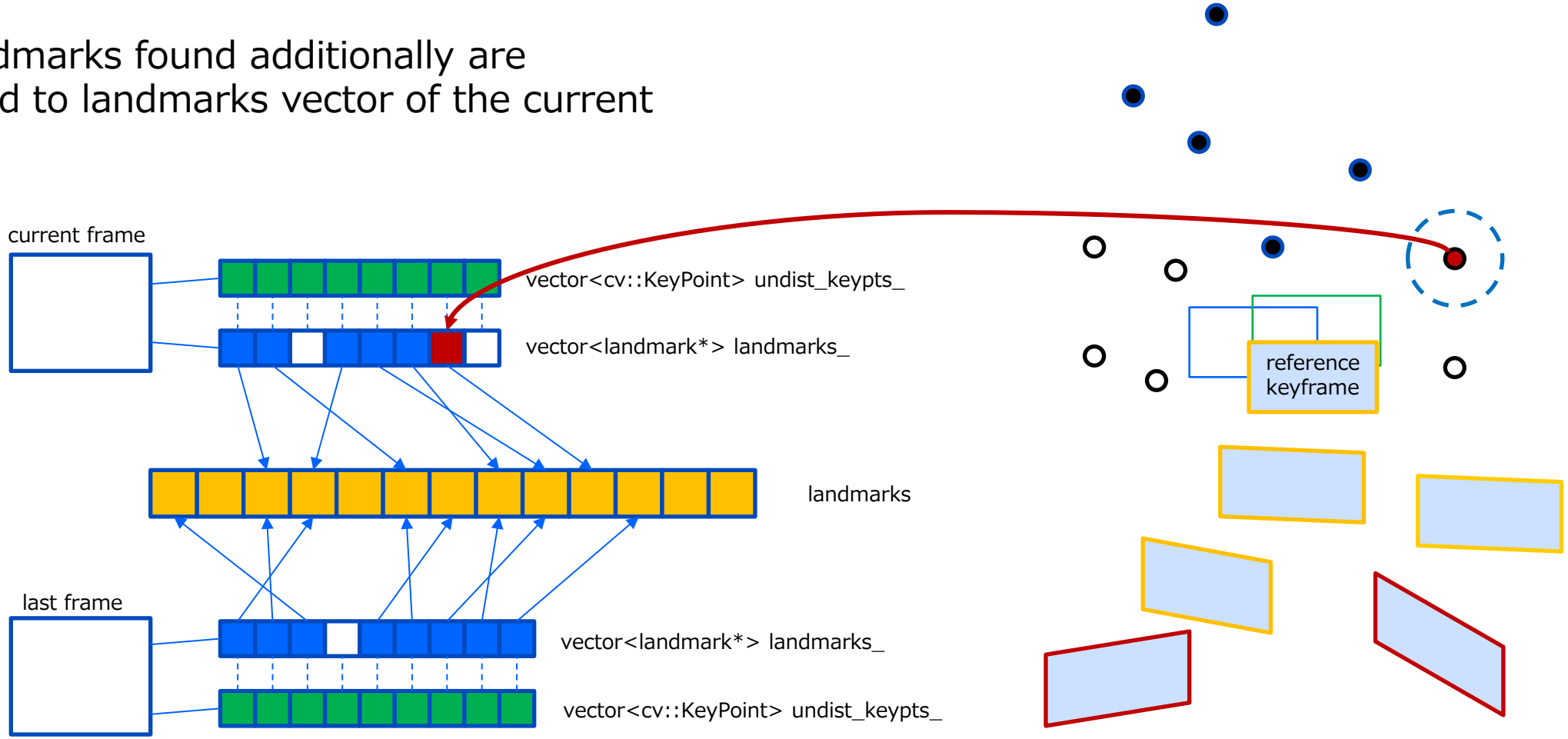
- For each landmark which is observed from local keyframes but not from the match frame:
  - Obtain candidate keypoints within grid cells around the reprojected point of the landmark
    - About grid cell, see assign_keypoints_to_grid.
  - For each candidate:
    - Reprojection error of x direction on the right image must be within a threshold calculated by using scale factor of the keyframe
    - Calculate Hamming distance between the keypoint of the landmark and that of the candidate
    - The distance must be the smallest among all the candidates and below HAMMING_DIST_THR_HIGH and lower than 0.8 * the second best and the scale level is equal to that of the second best
  - valid landmarks are registered to landmarks vector of the current frame



optimize_current_frame_with_local_map – 4 of 5

projection::match_frame_and_landmarks (cont.)

- valid landmarks found additionally are registered to landmarks vector of the current frame

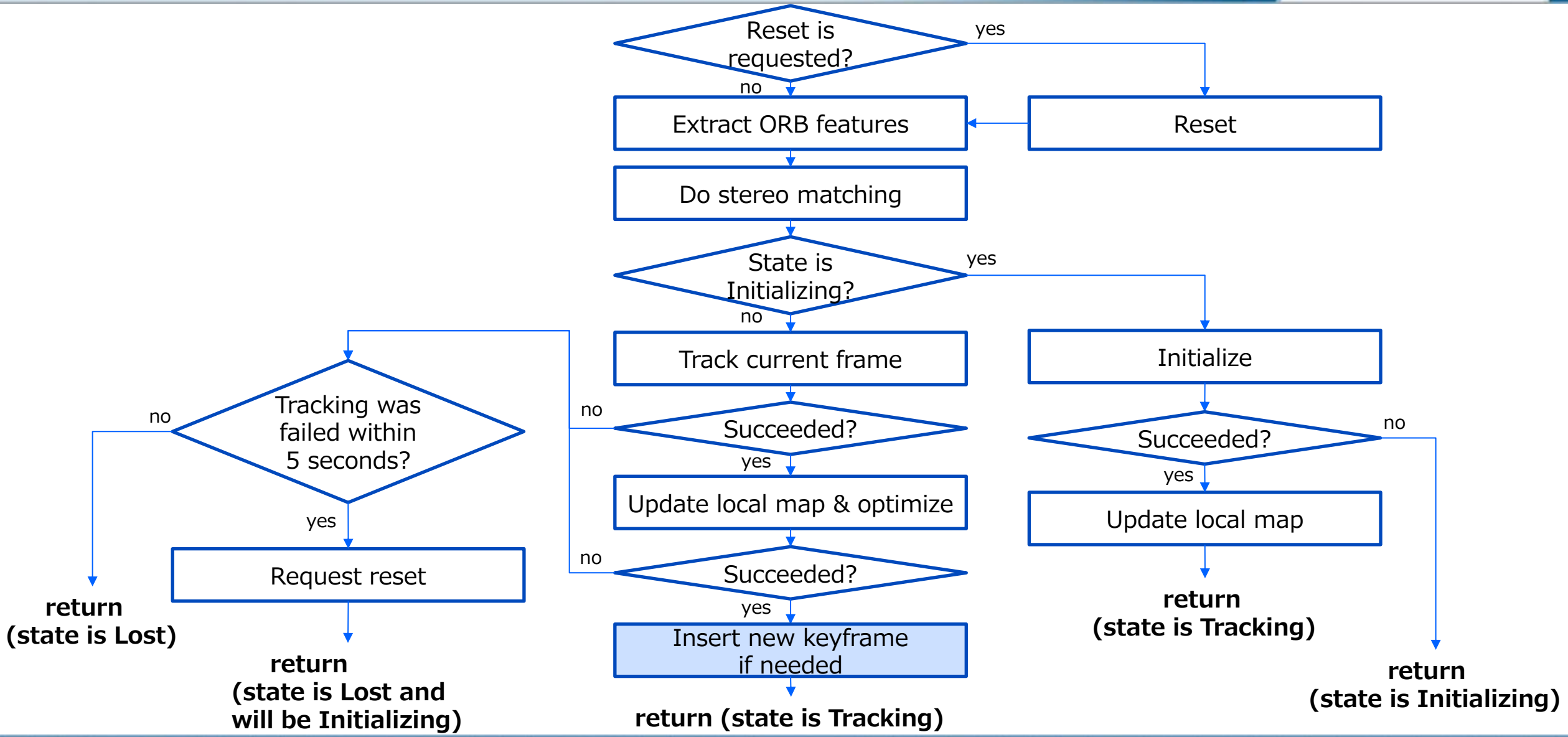


optimize_current_frame_with_local_map – 5 of 5

- Call pose_optimizer::optimize again
  - It is expected that more accurate pose of the current frame can be obtained because the number of landmarks may have increased than before
- Count the number of tracked landmarks which are classified as inliers through pose_optimizer::optimize
  - All landmarks inputted to pose_optimizer::optimize have increased their num_ovservable_ in search_local_landmarks
  - Only landmarks classified as inliers have increased their num_observed_
  - Landmarks classified as outliers keep their num_observed_ unchanged
  - num_observable_ and num_observed_ are information to classify the landmark as redundant or not (used by the mapping module)

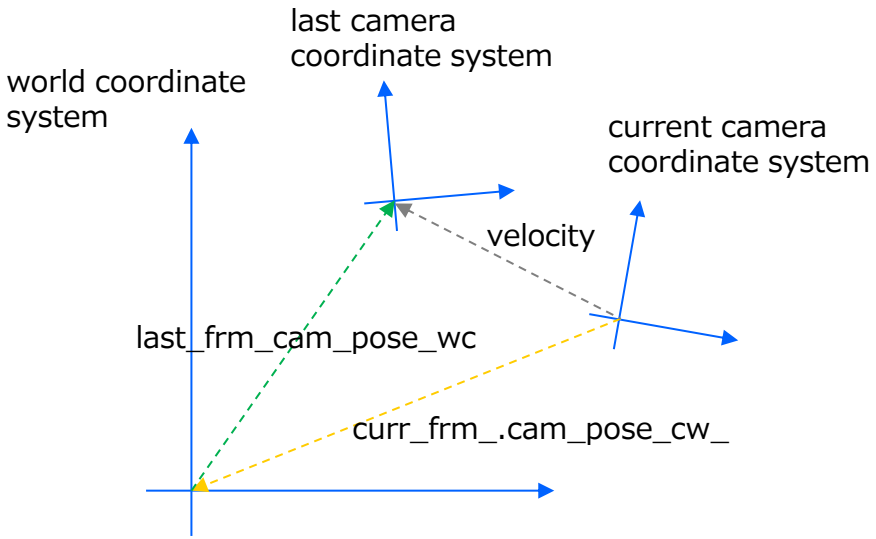
TRACKING – TRACK – INSERT_NEW_KEYFRAME

Where we are


Reset is requested?
yes
no
Reset
Extract ORB features
Do stereo matching
State is Initializing?
yes
no
Initialize
Track current frame
Succeeded?
no
yes
Update local map
Succeeded?
no
yes
Update local map & optimize
Succeeded?
no
yes
Insert new keyframe if needed
Tracking was failed within 5 seconds?
no
yes
Request reset
return (state is Lost)
return (state is Lost and will be Initializing)
return (state is Tracking)
return (state is Tracking)
return (state is Initializing)

Before calling new_keyframe_is_needed

- update_motion_model
  - Calculate velocity for the next frame
  - velocity = curr_frm_.cam_pose_cw_
    * last_frm_cam_pose_wc



- frame_statistics::update_frame_statistics
  - Update below
    - unordered_map<keyframe*, vector<unsigned int>> frm_ids_of_ref_keyfrms_
    - unsigned int num_valid_frms_
    - unordered_map<unsigned int, keyframe*> ref_keyfrms_
    - eigen_alloc_unordered_map<unsigned int, Mat44_t> rel_cam_poses_from_ref_keyfrms_
    - unordered_map<unsigned int, double> timestamps_

new_keyframe_is_needed

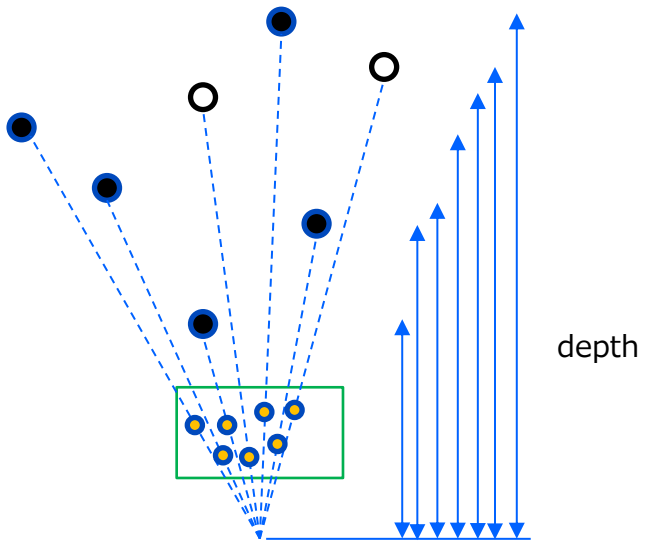
- Return false within a second after relocalization
- Return false if !A1 && !A2 && !A3
  - A1: more than max_num_frms (=camera->fps frames) have been passed (= 1 second passed) since the last keyframe
  - A2: more than min_num_frms (=0) have been passed (= 0 second passed) and mapper is idle
  - A3: num_tracked_lms is below num_reliable_lms * 0.25 (hard coded value)
    - num_tracked_lms = the number of landmarks classified as inliers through pose_optimizer::optimize
    - num_reliable_lms = the number of landmarks which are observed from the reference keyframe and two or more other keyframes
- Return false if condition B is not met
  - B: num_tracked_lms is equal to or above num_tracked_lms_thr (=15) and below num_reliable_lms * lms_ratio_thr (=0.9)
- Return true if queued keyframes to mapper is equal to or less than 2 (hard coded value)
- Else return false

insert_new_keyframe – 1 of 6

- Create a new keyframe instance using the current frame
- Sort keypoints vector by their depth in ascending order
- For each keypoint in the ascending order vector:
  - if the number of newly created landmarks is above min_num_to_create (=100) and its depth is above true_depth_thr_ (written in yaml):
    - Break loop
  - if it is already registered to the current frame's landmarks vector:
    - No need to create a new landmark, skip it
  - Create and register a landmark
- Queue the keyframe instance to mapping module's queue

● = landmark already registered to the current frame's landmarks vector (skipped)

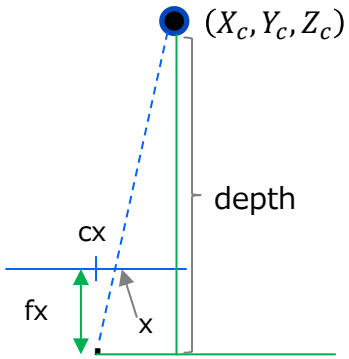
○ = landmark not registered to the current frame's landmarks vector (newly created)



insert_new_keyframe – 2 of 6

Creation of a landmark detail (frame::triangulate_stereo)

- Calculate 3D coordinate in the world coordinate system
- Create a landmark instance with the 3D coordinate



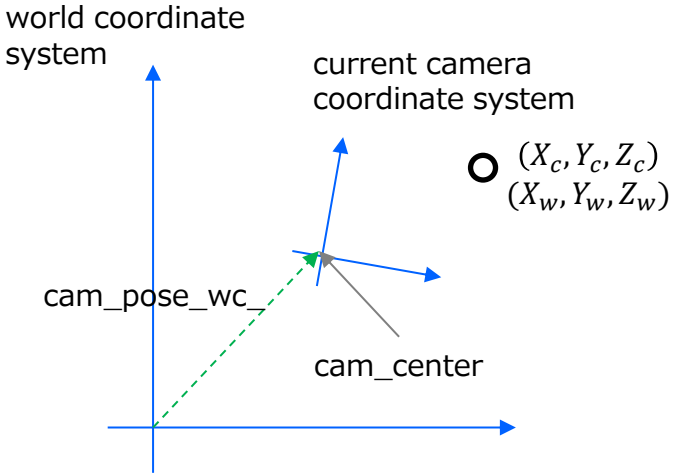
$$\frac{X_c}{depth} = \frac{x - c_x}{f_x}$$

$$\therefore X_c = depth \frac{x - c_x}{f_x}$$

$$Y_c = depth \frac{y - c_y}{f_y}$$

$$Z_c = depth$$

depth has already been calculated by stereo::compute
(see stereo::compute – 4 of 5)



$$\begin{pmatrix} X_w \\ Y_w \\ Z_w \end{pmatrix} = rot_wc_ \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} + cam_center$$

insert_new_keyframe – 3 of 6

<u>registration of a landmark detail</u>

- add_observation for the landmark
- compute_descriptor for the landmark
- update_normal_and_depth for the landmark
- add_landmark for the keyframe
- add_landmark for the map database
- register the landmark to the current frame’s landmarks vector

insert_new_keyframe – 4 of 6

registration of a landmark detail

- **add_observation for the landmark**
- compute_descriptor for the landmark
- update_normal_and_depth for the landmark
- **add_landmark for the keyframe**
- **add_landmark for the map database**
- register the landmark to the current frame's landmarks vector

Add an entry of keyframe object along with index of landmark in the keyframe.

class landmark

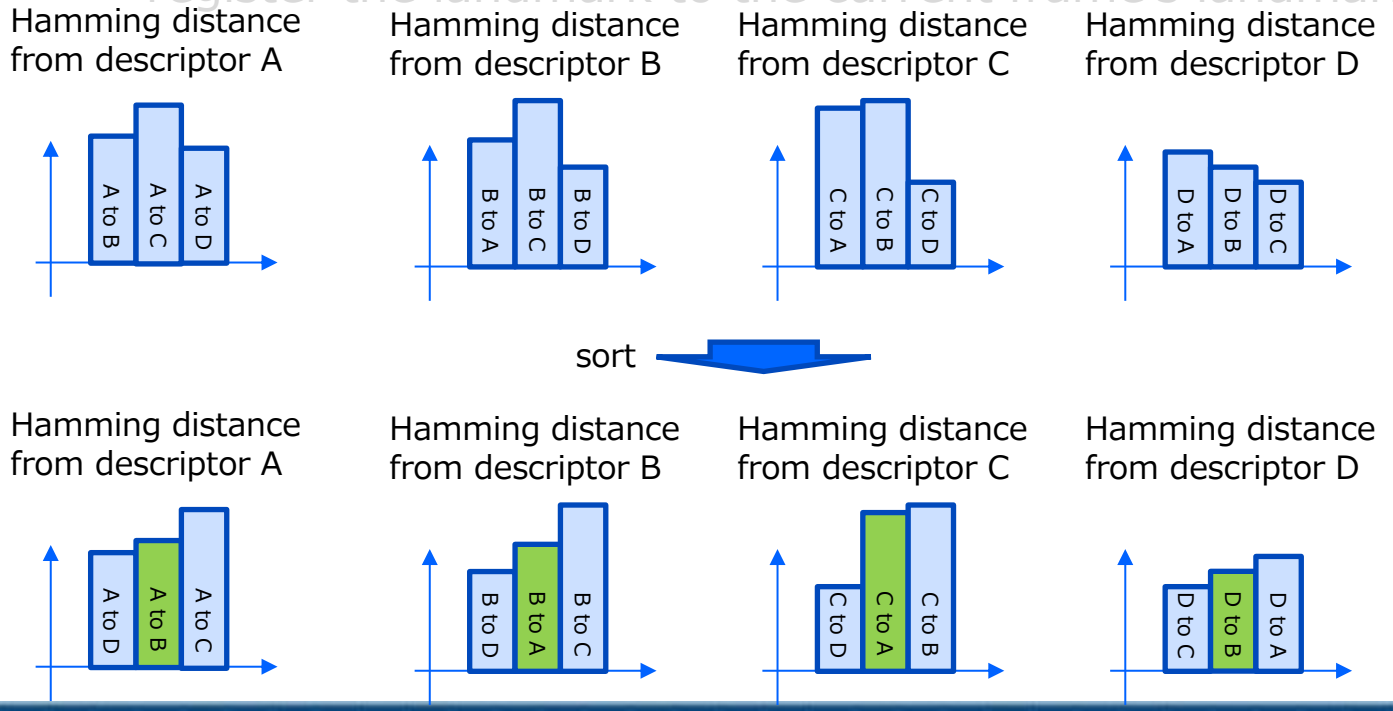
unsigned int **id_**
unsigned int **first_keyfrm_id_**
unsigned int **num_observations_**
Vec3_t **pos_w_**
map<keyframe*, unsigned int> **observations_**
Vec3_t **mean_normal_**
cv::Mat **descriptor_**
keyframe* **ref_keyfrm_**

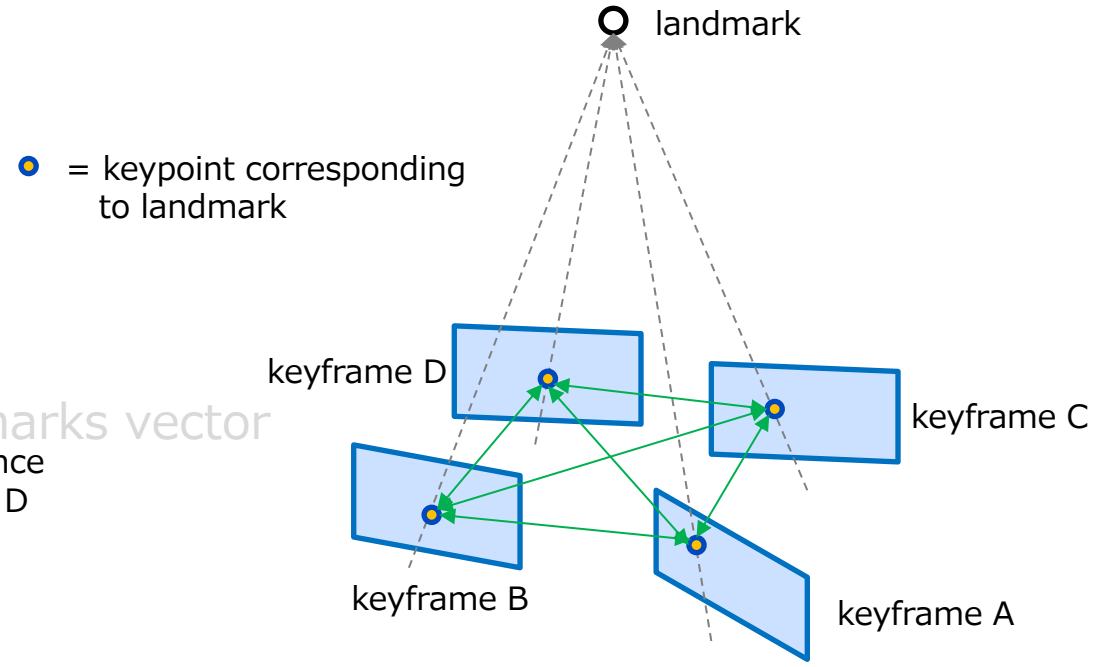
Just add landmark object to vector/unordered_map

insert_new_keyframe – 5 of 6

registration of a landmark detail

- add_observation for the landmark
- **compute_descriptor for the landmark**
- update_normal_and_depth for the landmark
- add_landmark for the keyframe
- add_landmark for the map database
- register the landmark to the current frame's landmarks vector





Select the minimum of median to determine the most representable descriptor among all descriptors. In this case, descriptor D is selected as landmark's descriptor.

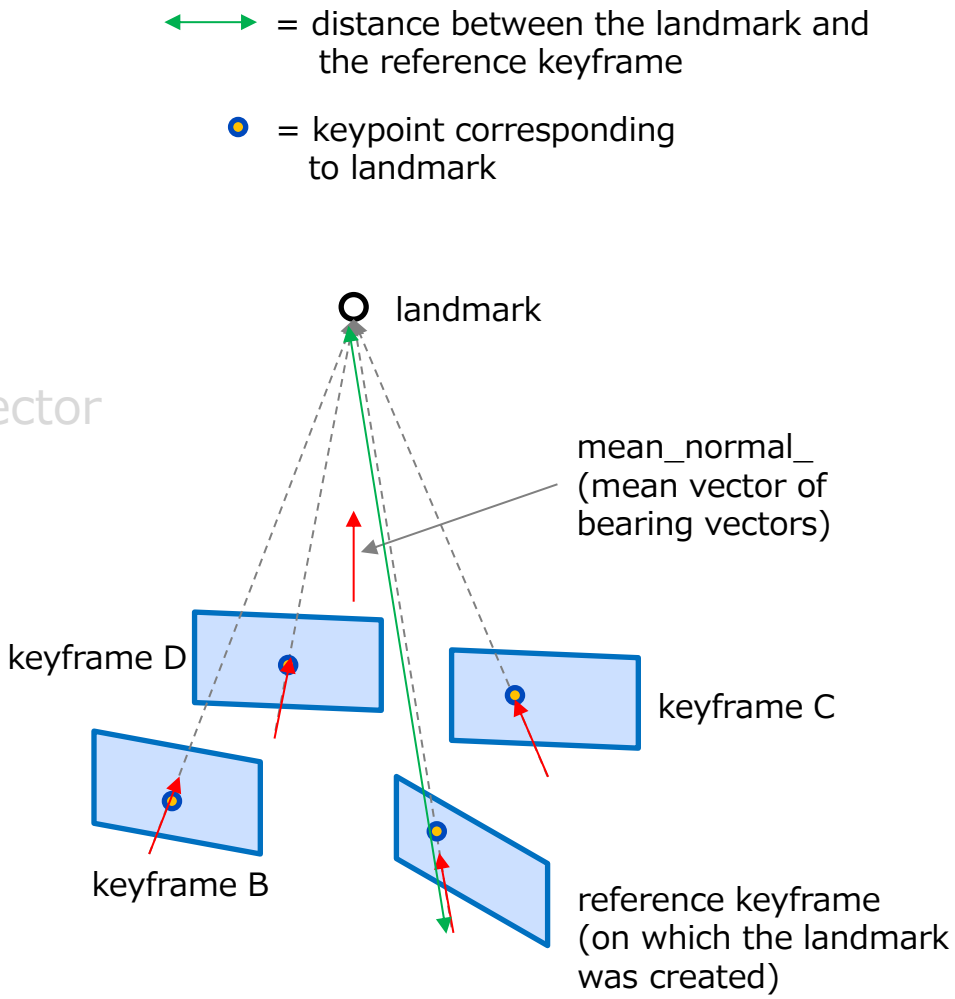
insert_new_keyframe – 6 of 6

registration of a landmark detail

- add_observation for the landmark
- compute_descriptor for the landmark
- **update_normal_and_depth for the landmark**
- add_landmark for the keyframe
- add_landmark for the map database
- register the landmark to the current frame's landmarks vector

Below are attributes of landmark class updated in update_normal_and_depth function:

- mean_normal_
  - mean vector of bearing vectors of keyframes which observe the landmark
- max_valid_dist_
  - (distance between the landmark and the reference keyframe) * (scale factor with which the corresponding keypoint was detected)
- min_valid_dist_
  - max_valid_dist_ / (scale factor at the highest pyramid level)

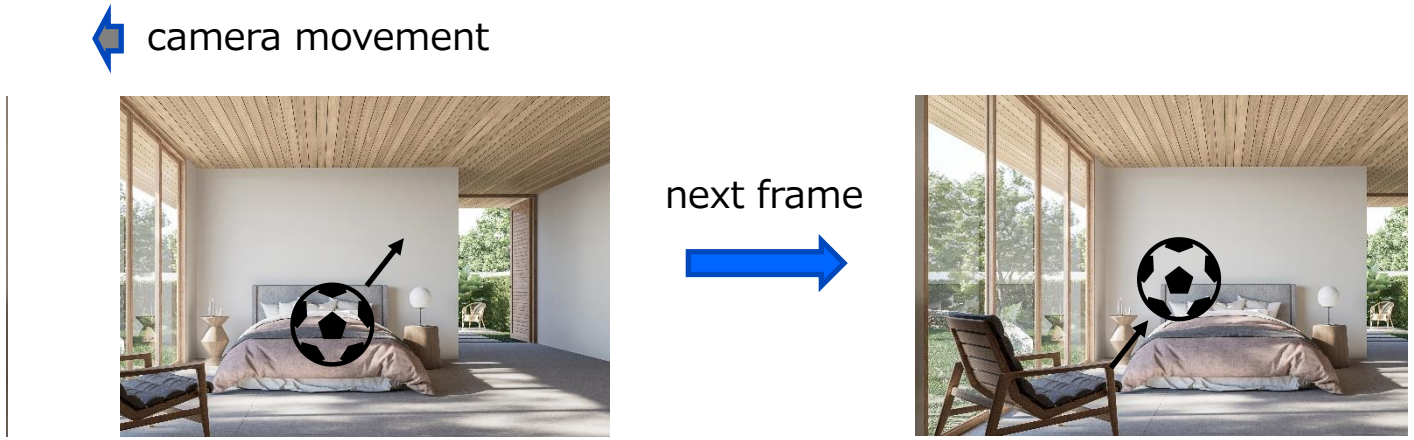


After insert_new_keyframe

- Landmarks associated with the current frame are set to null if they are classified as outliers
  - Newly created landmarks in insert_new_keframe are not classified as outliers
  - All landmarks that are used by optimizer::optimize have been classified as inliers or outliers
  - By doing this, all landmarks associated with the current frame are all valid
- Calculate last_cam_pose_from_ref_keyfrm_ for the next frame
- Copy the current frame instance to that of the last frame

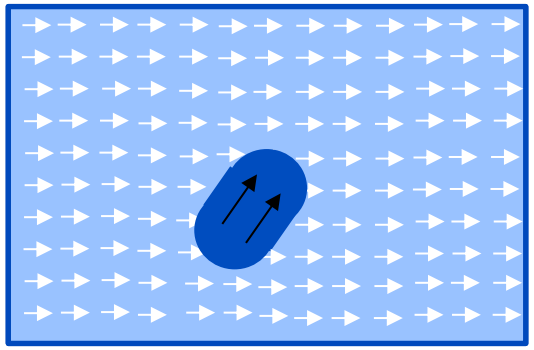
TRACKING – THOUGHT EXPERIMENT

What if a dynamic object crosses?

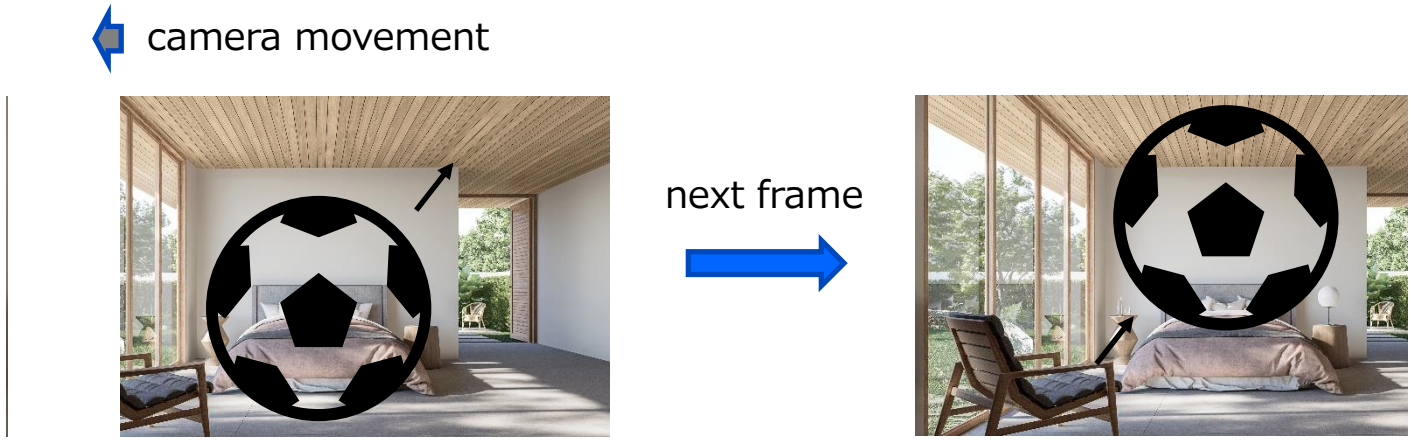


Assumed that a dynamic object is small, it is expected that landmarks on the object are rejected as outliers because they are minor compared to landmarks on the background scene.
True camera movement will be recognized by SLAM.

keypoints movement between the two frames

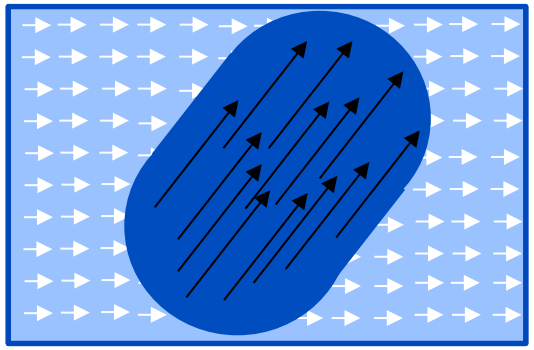


What if a large dynamic object crosses?



Assumed that a dynamic object is large, it may not be true that keypoints on the dynamic object is minor. SLAM may not recognize true camera movement.

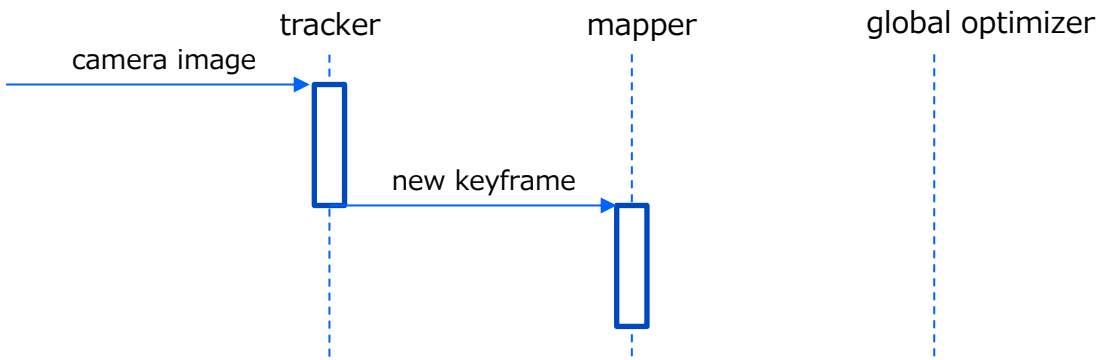
keypoints movement between the two frames



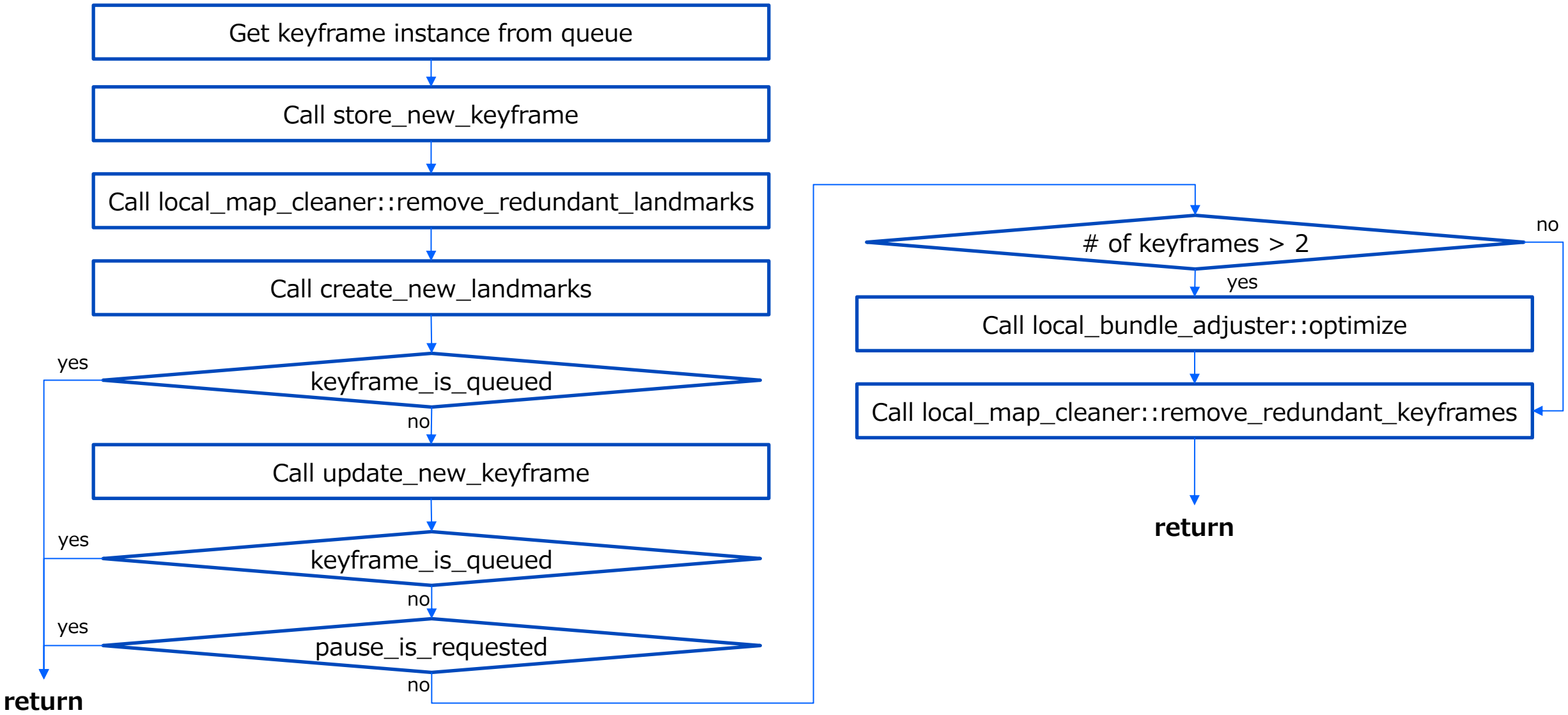
MAPPING

mapping module thread

- Mapping module thread is created at initialization
- It is waiting for some messages:
  - Terminate
  - Pause
  - Reset
  - Keyframe insertion
    - Call mapping_with_new_keyframe
    - Then call global_optimization_module::queue_keyframe



mapping_with_new_keyframe



store_new_keyframe – 1 of 7

- Call keyframe::compute_bow
  - bow_vec_ and bow_feat_vec_ are calculated by DBoW2 / FBoW
  - See Bow data of each keyframe about bow_vec_ and bow_feat_vec_
- For all landmarks observed from the new keyframe:
  - If it is created on the new keyframe,
    - local_map_cleaner::add_fresh_landmark
  - else
    - landmark::add_observation
      - Already explained in insert_new_keyframe – 4 of 6
    - landmark::update_normal_and_depth
      - Already explained in insert_new_keyframe – 6 of 6
    - landmark::compute_descriptor
      - Already explained in insert_new_keyframe – 5 of 6

add_observation, update_normal_and_depth, and compute_descriptor for them have been already called in insert_new_keyframe

store_new_keyframe – 2 of 7

graph_node::update_connections – 1 of 5

- **Update connection and covisibility data**
- Update spanning tree

Update covisibility keyframes and connected keyframes.
See Covisibility keyframes and Connected keyframes

connected keyframes

connected_keyfrms_and_weights_

A	B	C	D	E
6	20	83	11	139

← keyframes which have common landmarks with this keyframe

← number of common landmarks

covisibility keyframes

ordered_covisibilities_

E	C	B

ordered_weights_

139	83	20

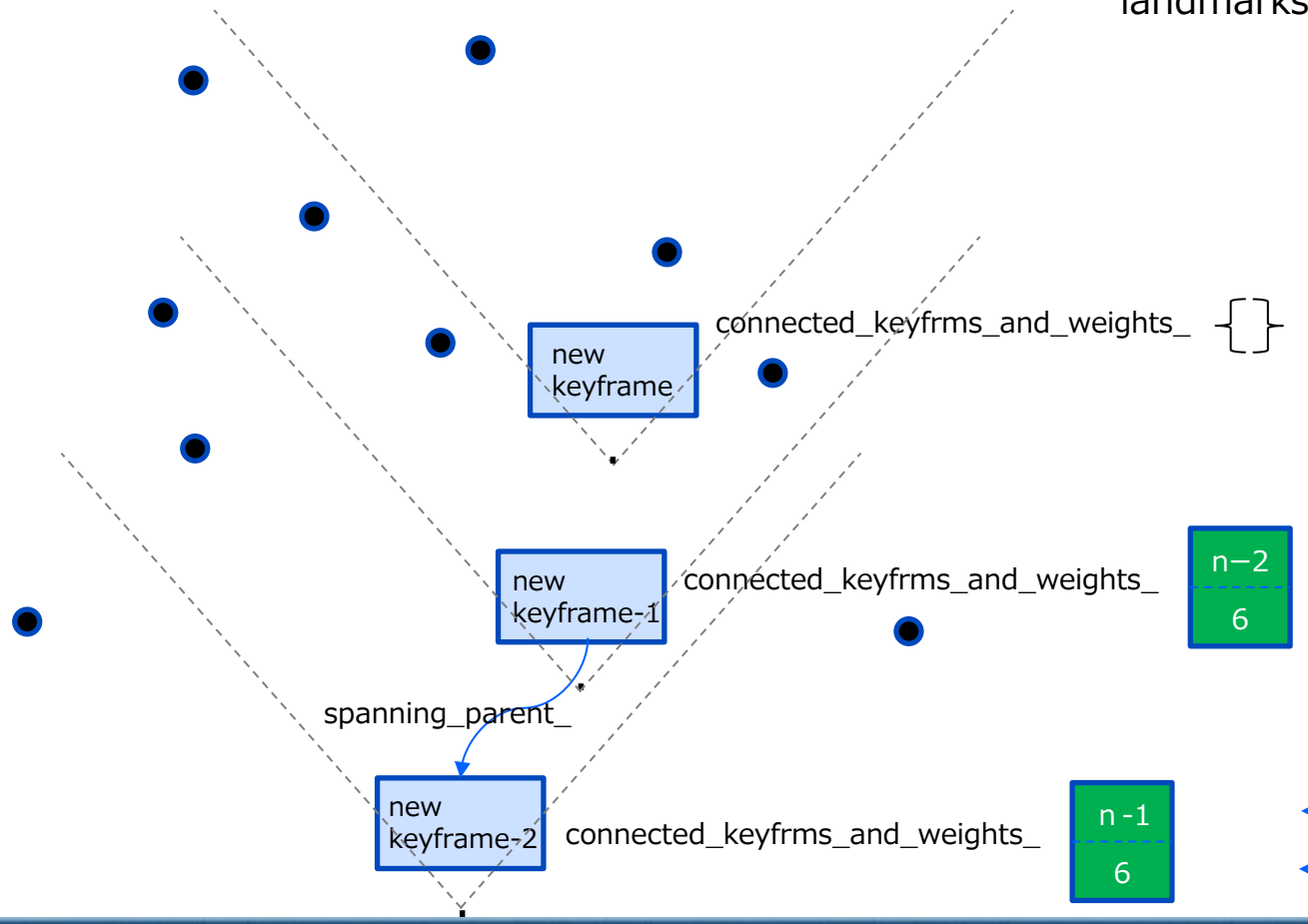
covisibility keyframes are connected keyframes which have weight_thr_ (=15) or more common landmarks with this keyframe.
They are sorted in descending order by weights

store_new_keyframe – 3 of 7

graph_node::update_connections – 2 of 5

- **Update connection and covisibility data**
- Update spanning tree

Look over keyframes which have common landmarks with the new keyframe by referring to observations_ of the landmarks. The keyframes are classified as `covisibility keyframe` or `connected keyframe` depending on the number of common landmarks.



Note that ordered_covisibities_ and ordered_weight_ are omitted in the figure here and in the next few slides. They will be updated as well as connected_keyfrms_and_weights_.

← keyframes which have common landmarks with this keyframe

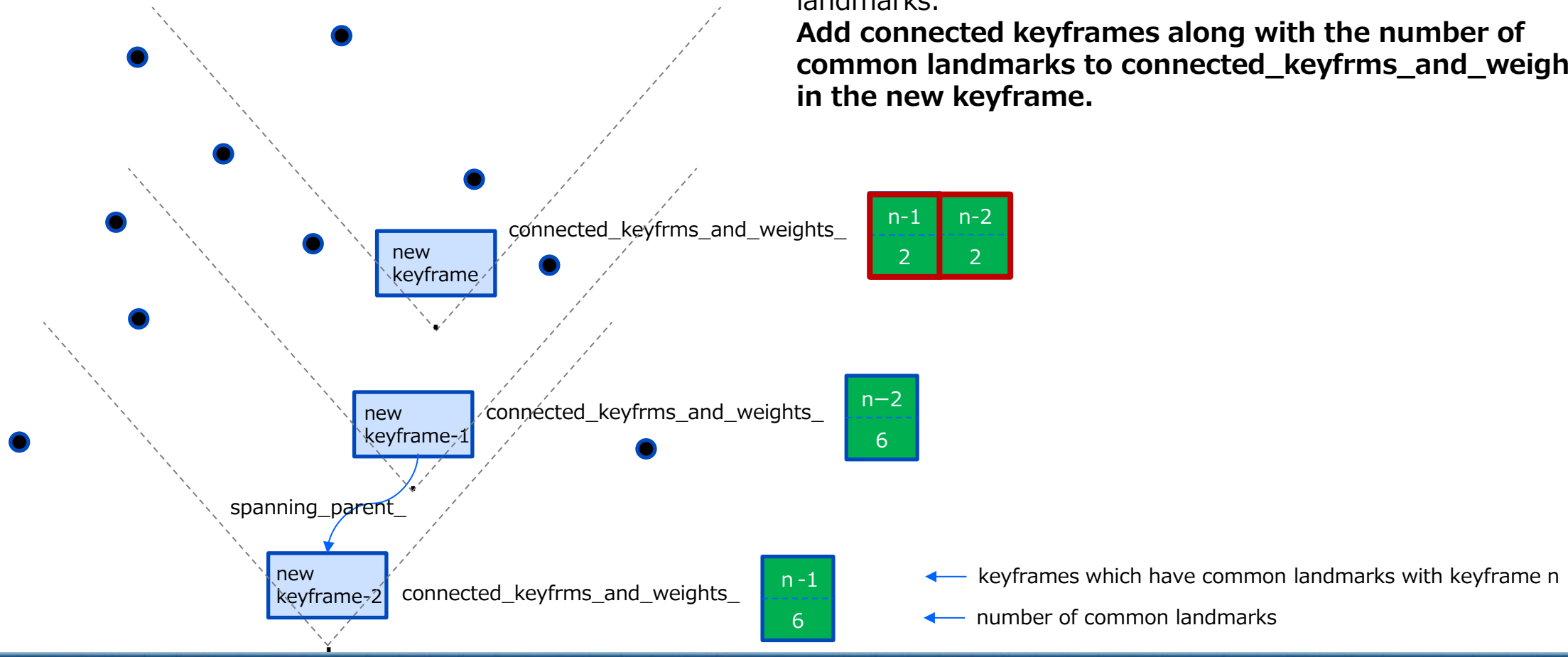
← number of common landmarks

store_new_keyframe – 4 of 7

graph_node::update_connections – 3 of 5

- **Update connection and covisibility data**
- Update spanning tree

Look over keyframes which have common landmarks with the new keyframe by referring to observations_ of the landmarks. The keyframes are classified as `covisibility keyframe` or `connected keyframe` depending on the number of common landmarks.
Add connected keyframes along with the number of common landmarks to connected_keyfrms_and_weights_ in the new keyframe.

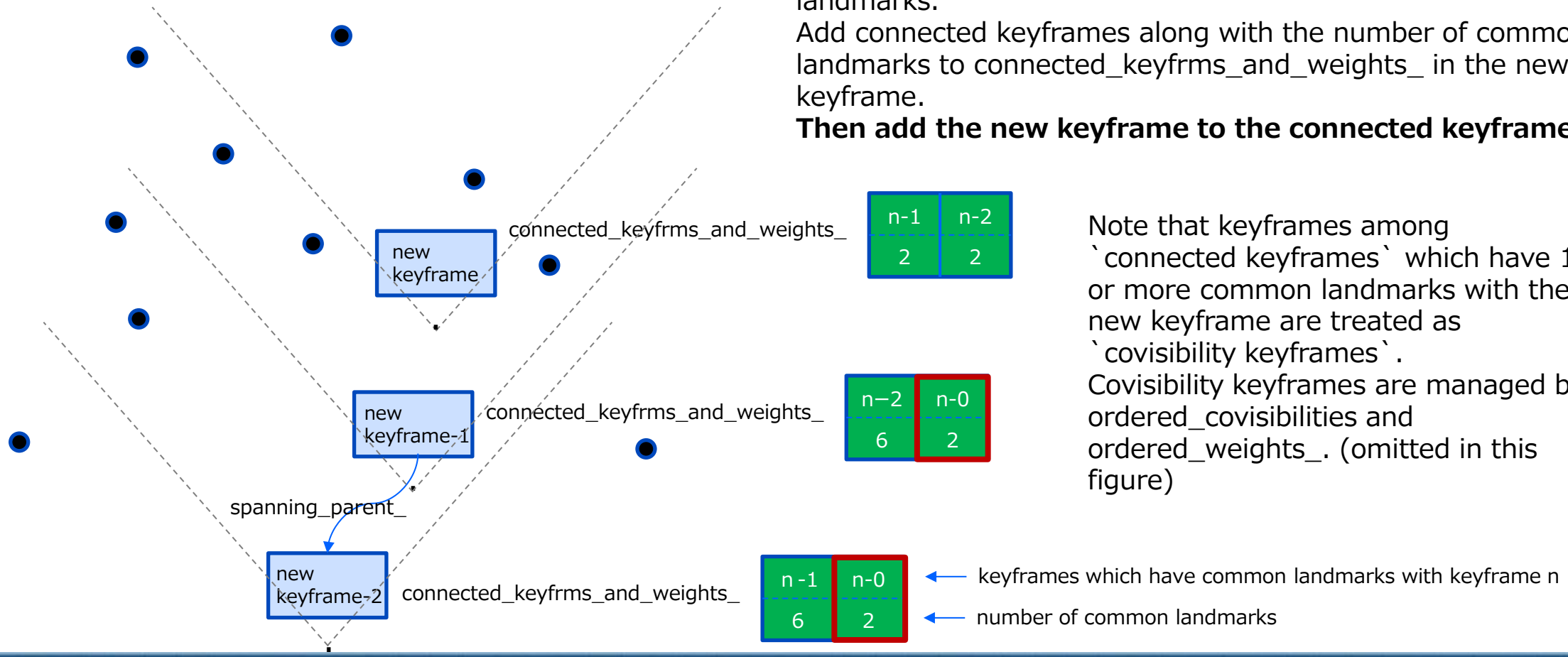


store_new_keyframe – 5 of 7

graph_node::update_connections – 4 of 5

- **Update connection and covisibility data**
- Update spanning tree

Look over keyframes which have common landmarks with the new keyframe by referring to observations_ of the landmarks. The keyframes are classified as `covisibility keyframe` or `connected keyframe` depending on the number of common landmarks.
Add connected keyframes along with the number of common landmarks to connected_keyfrms_and_weights_ in the new keyframe.
Then add the new keyframe to the connected keyframes.



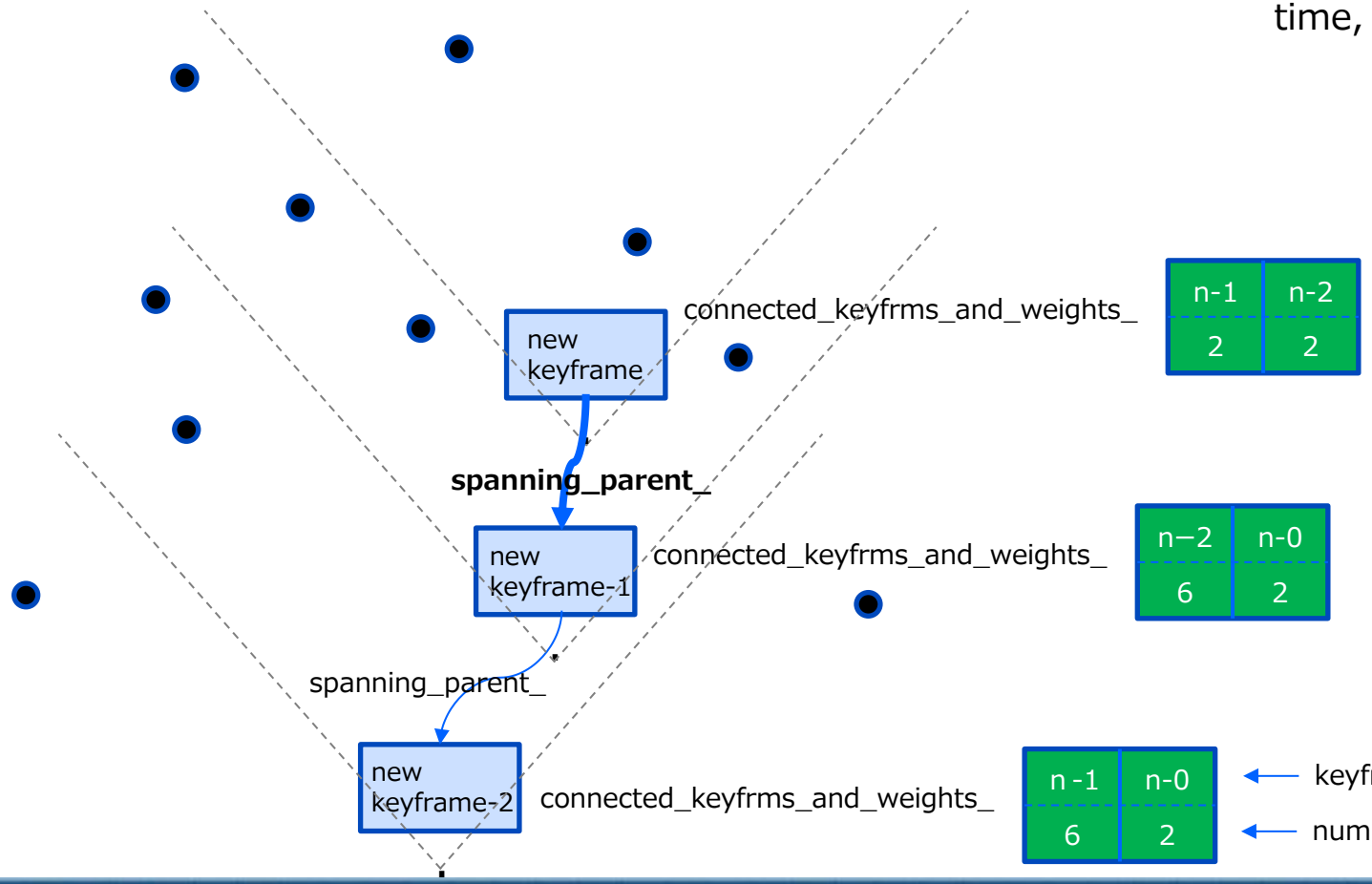
Note that keyframes among `connected keyframes` which have 15 or more common landmarks with the new keyframe are treated as `covisibility keyframes`.
Covisibility keyframes are managed by ordered_covisibilities and ordered_weights_. (omitted in this figure)

store_new_keyframe – 6 of 7

graph_node::update_connections – 5 of 5

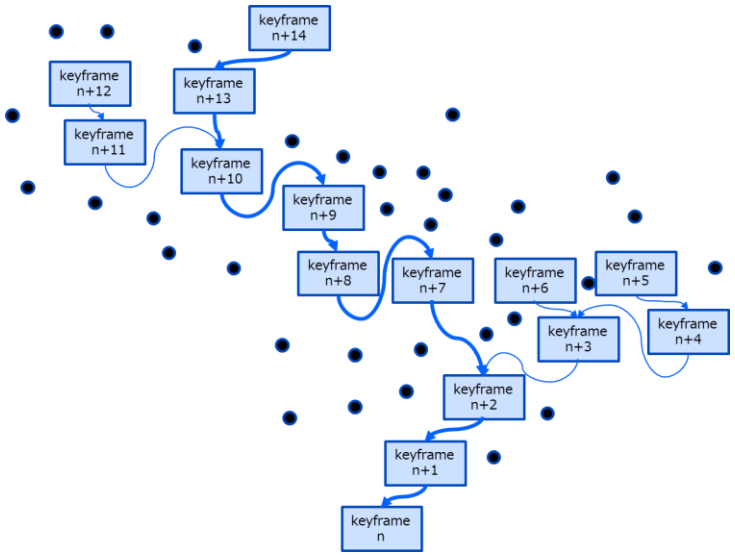
- Update connection and covisibility data
- **Update spanning tree**

Connect spanning tree to set the keyframe which has the most common landmarks with the new keyframe as its parent. At the same time, set itself to the parent's children.



keyframes which have common landmarks with keyframe n

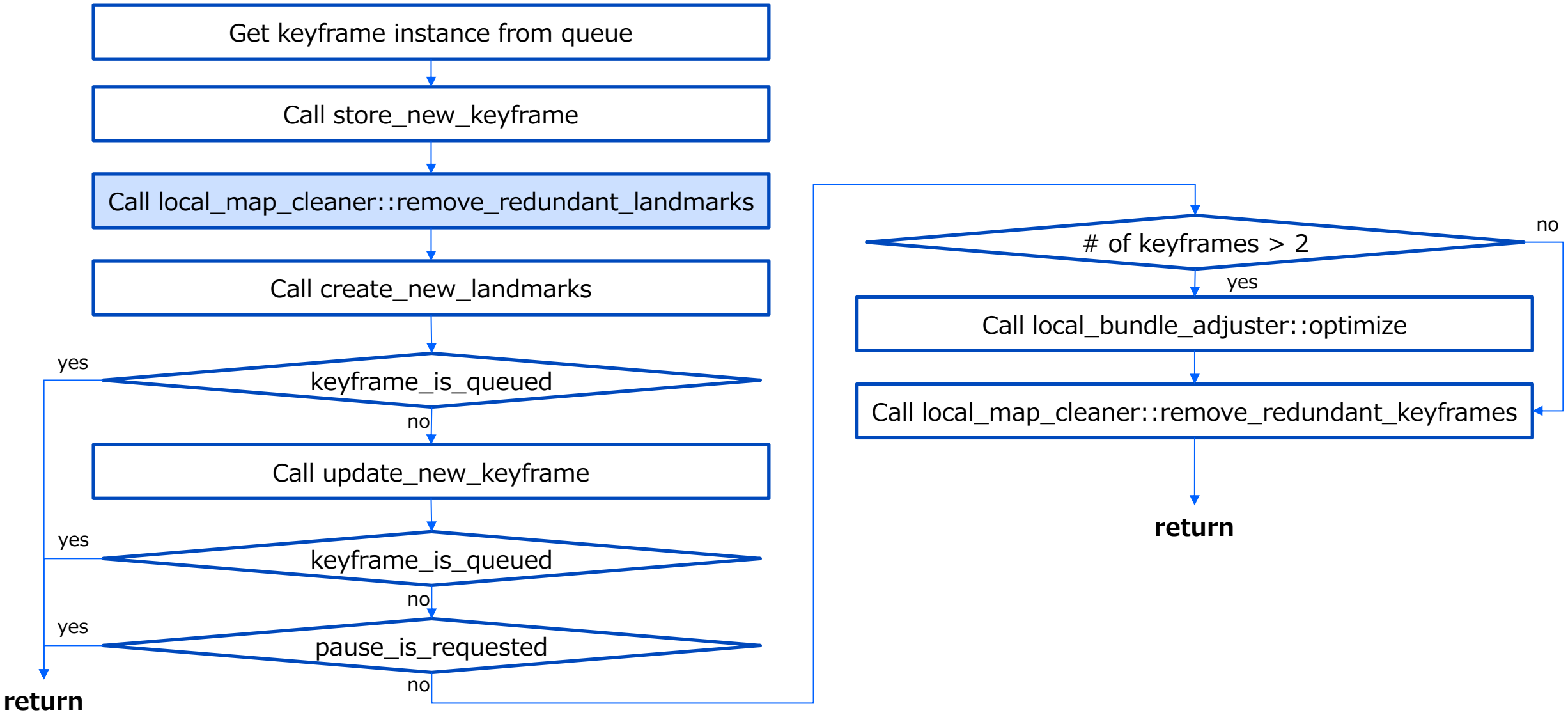
number of common landmarks



store_new_keyframe – 7 of 7

- Call map_database::add_keyframe
  - Just add the current keyframe to map database

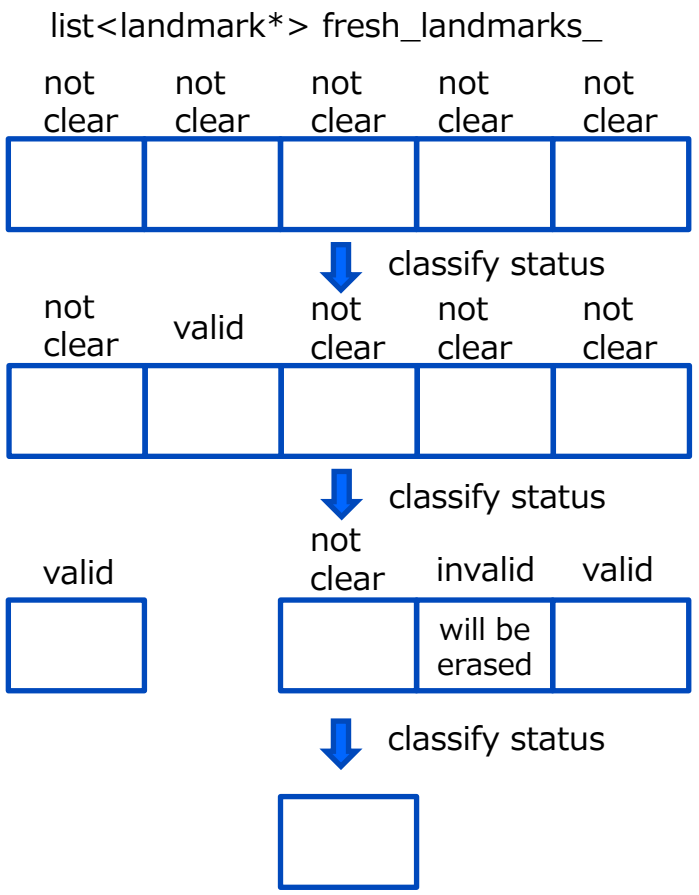
Where we are


Get keyframe instance from queue
Call store_new_keyframe
Call local_map_cleaner::remove_redundant_landmarks
Call create_new_landmarks
keyframe_is_queued
yes
no
Call update_new_keyframe
keyframe_is_queued
yes
no
pause_is_requested
yes
no
return
# of keyframes > 2
no
yes
Call local_bundle_adjuster::optimize
Call local_map_cleaner::remove_redundant_keyframes
return

local_map_cleaner::remove_redundant_landmarks

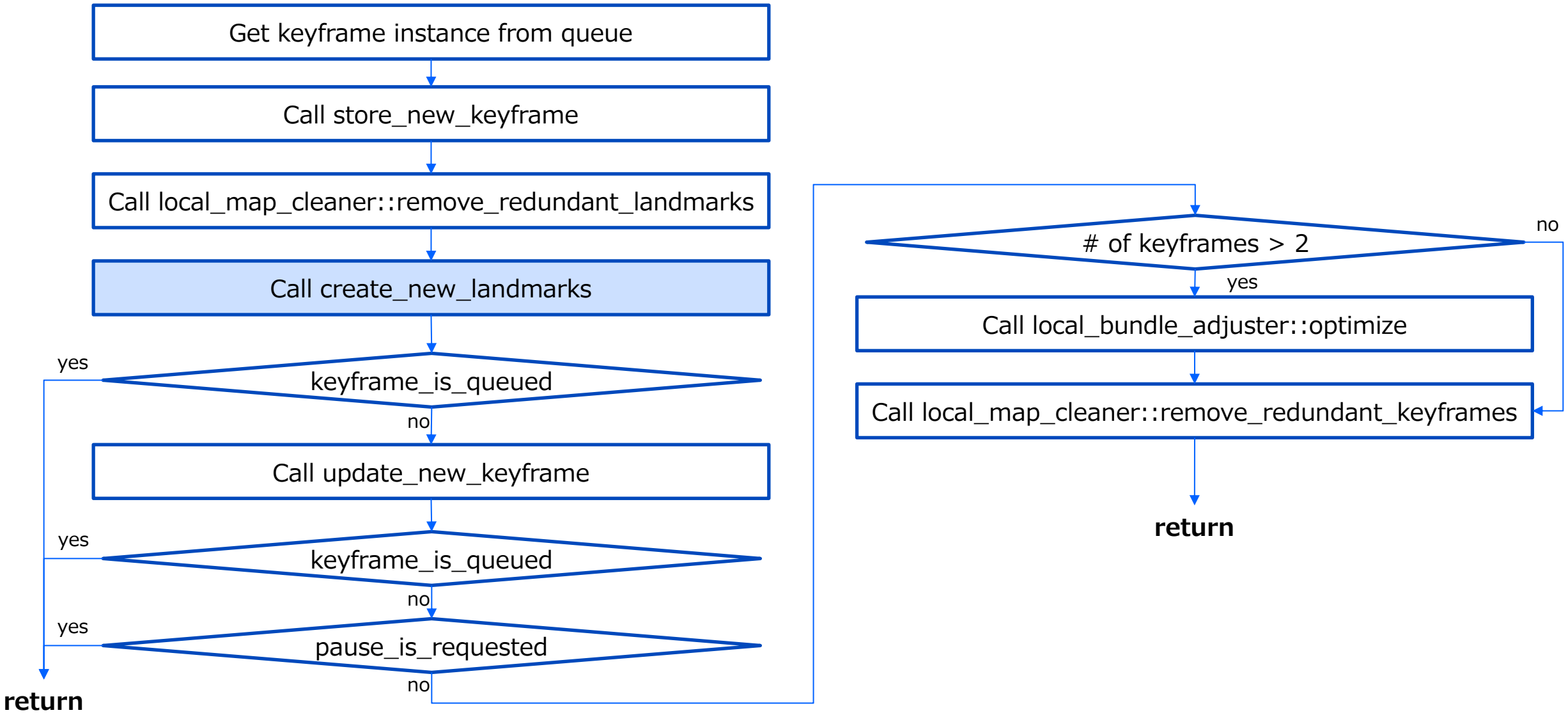
- For each landmark in fresh_landmarks_ list, determine its status:
  - `invalid` if 1 or 2 below is met:
    1. num_observed / num_observable < observed_ratio_thr (=0.3)
       - num_observed: the number which is incremented when the landmark is classified as inliers in tracking process
       - num_observable: the number which is incremented when the landmark is used in tracking process (don't care inlier or outlier)
    2. the number of keyframes which observe the landmark is smaller than or equal to num_obs_thr (=3) after num_reliable_keyfrms (=2) keyframes insertion
    - `invalid` landmarks are removed from fresh_landmarks_ list and erased by landmark::prepare_for_erasing call
  - `valid` else if 1 below is met:
    1. The number of keyframes which observe the landmark is larger than num_obs_thr (=3) after num_reliable_keyfrms+1 (=3) keyframes insertion
    - `valid` landmarks are removed from fresh_landmarks_ list
  - else `not clear`
    - `not clear` landmarks keep staying in fresh_landmarks_ list

Newly created landmarks have been registered in fresh_landmarks_ list in store_new_keyframe.



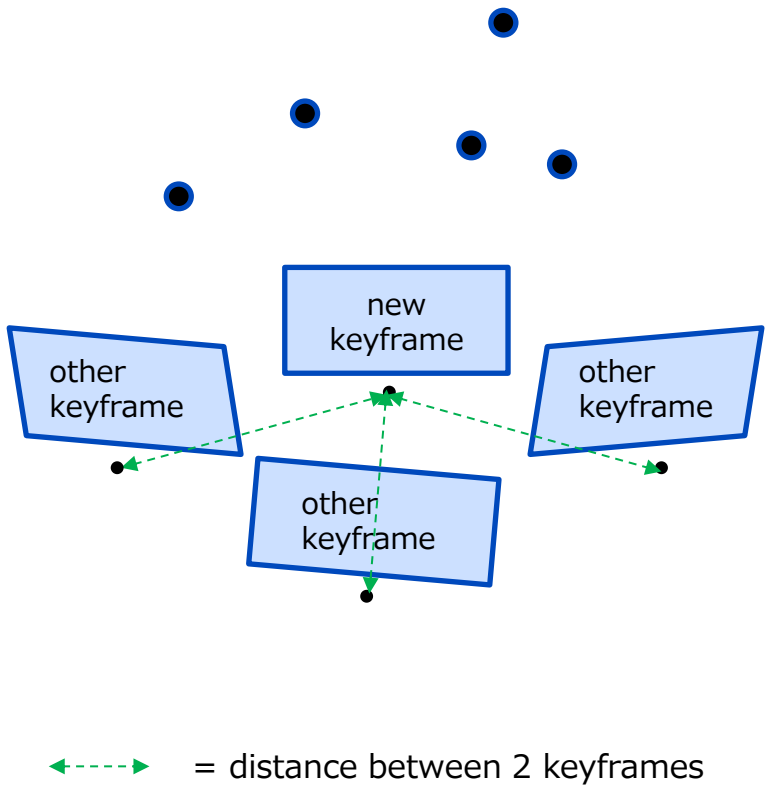
Only `not clear` stay in fresh_landmark_ list and will be classified again next time

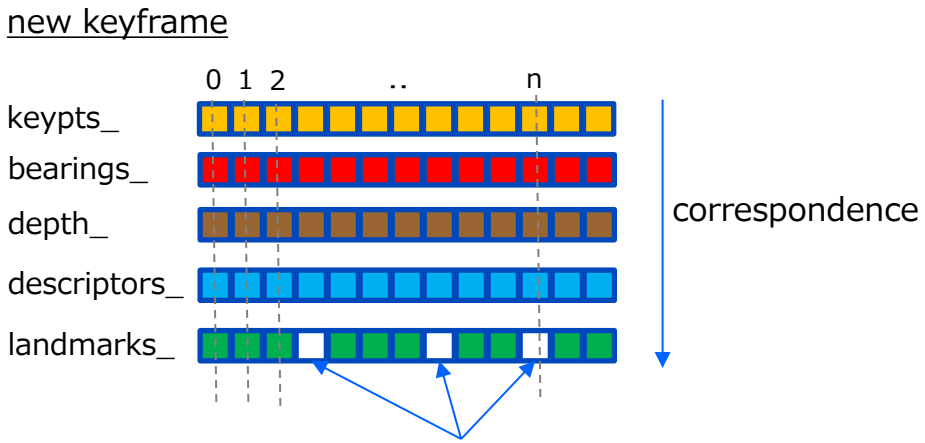
Where we are


Get keyframe instance from queue
Call store_new_keyframe
Call local_map_cleaner::remove_redundant_landmarks
Call create_new_landmarks
yes
keyframe_is_queued
no
Call update_new_keyframe
yes
keyframe_is_queued
no
yes
pause_is_requested
no
return
# of keyframes > 2
no
yes
Call local_bundle_adjuster::optimize
Call local_map_cleaner::remove_redundant_keyframes
return

create_new_landmarks – 1 of 8

- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes:
  - If the distance between the new keyframe and it is below stereo baseline length, skip it
  - Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)
  - Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks
  - Call triangulate_with_two_keyframes to add landmarks

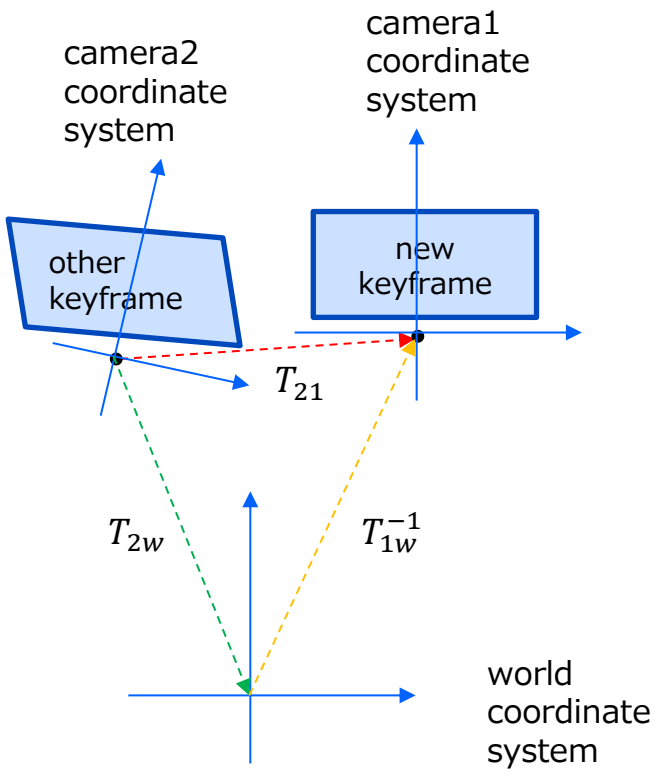




try to correspond keypoint pairs between the current frame and its top 10 covisibility keyframes and then create landmarks

create_new_landmarks – 2 of 8

- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes :
  - If baseline distance between the new keyframe and it is below stereo baseline length, skip it
  - **Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)**
  - Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks
  - Call triangulate_with_two_keyframes to add landmarks



$$T_{21} = T_{2w}T_{1w}^{-1} = \begin{pmatrix} R_{2w} & \boldsymbol{t}_{2w} \\ 0 & 1 \end{pmatrix}\begin{pmatrix} R_{1w}^T & -R_{1w}^T\boldsymbol{t}_{1w} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} R_{2w}R_{1w}^T & -R_{2w}R_{1w}^T\boldsymbol{t}_{1w} + \boldsymbol{t}_{2w} \\ 0 & 1 \end{pmatrix}$$

$$\therefore R_{21} = R_{2w}R_{1w}^T, \boldsymbol{t}_{21} = -R_{2w}R_{1w}^T\boldsymbol{t}_{1w} + \boldsymbol{t}_{2w}$$

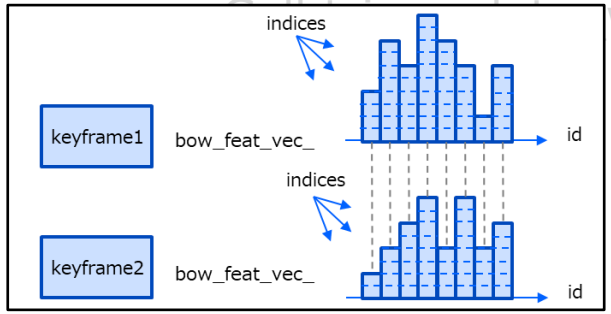
$$E = \boldsymbol{t}_{21} \times R_{21} = \begin{pmatrix} 0 & -t_{21_z} & t_{21_y} \\ t_{21_z} & 0 & -t_{21_x} \\ -t_{21_y} & t_{21_x} & 0 \end{pmatrix} R_{21} \; where \; \boldsymbol{x}_2^T E \boldsymbol{x}_1 = 0.$$

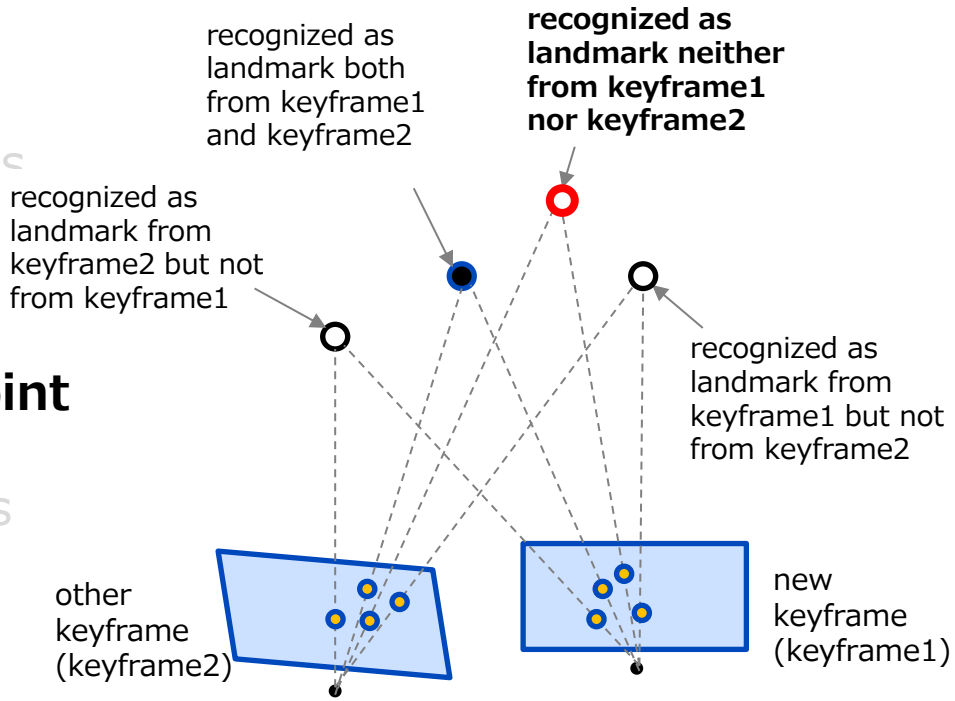
($\boldsymbol{x}_1$ *and* $\boldsymbol{x}_2$ *is bearing vector* of camera1 and camera2 respectively)

E will be used in check_epipolar_constraint called from robust::match_for_triangulation

create_new_landmarks – 3 of 8

- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes :
  - If baseline distance between the new keyframe and it is below stereo baseline length, skip it
  - Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)
  - **Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks**
  - Call triangulate_with_two_keyframes to add landmarks



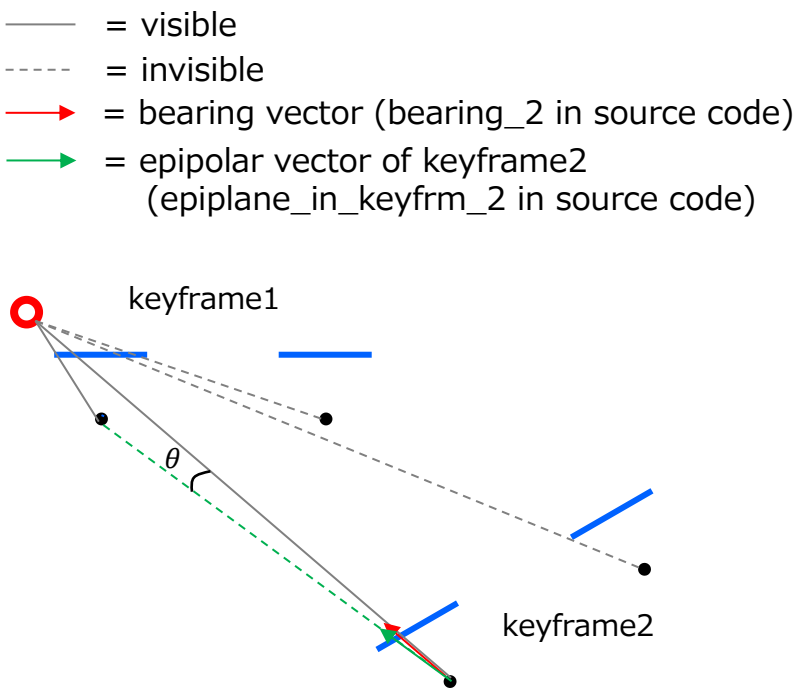


- For each keypoint of keyframe1:
  - If the keypoint is corresponded to a landmark (= already registered to landmarks vector), skip it
  - For each keypoint of keyframe2 whose bow has the same bow id with the keypoint of keyframe1:
    1. If the keypoint of keyframe2 is corresponded to a landmark, skip it
    2. If Hamming distance between the keypoints is above the best or HAMMING_DIST_THR_LOW, skip it

    (to be continued)

create_new_landmarks – 4 of 8

- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes :
  - If baseline distance between the new keyframe and it is below stereo baseline length, skip it
  - Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)
  - **Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks**
  - Call triangulate_with_two_keyframes to add landmarks

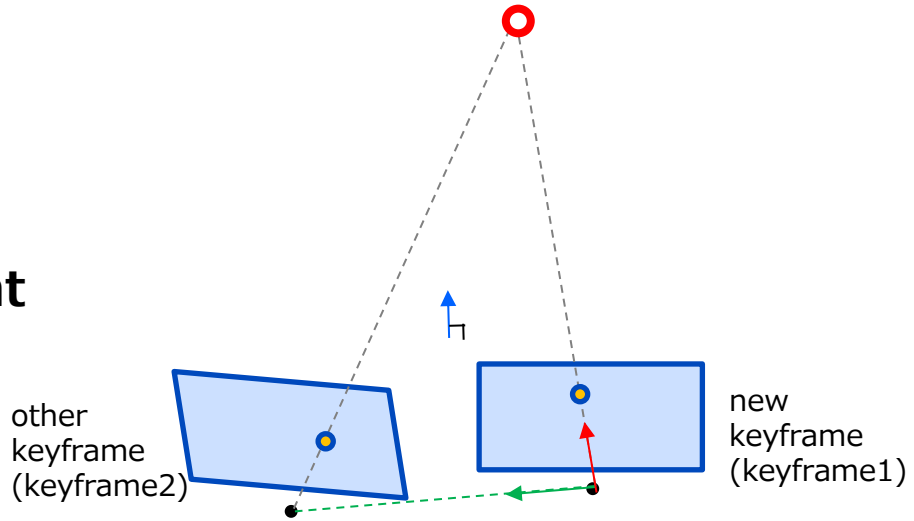


3. The pair is checked whether it is valid or not:
   If the keypoints of keyframe1 and keyframe2 exist only on left image
   - And if the angle θ in the right figure is below 3.0 degree
   - Then the keypoint pair is classified as invalid
   - In other words, reject if the corresponding landmark is near the epipolar line

   (to be continued)

create_new_landmarks – 5 of 8

- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes :
  - If baseline distance between the new keyframe and it is below stereo baseline length, skip it
  - Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)
  - **Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks**
  - Call triangulate_with_two_keyframes to add landmarks



4. The pair is further checked whether it is valid or not (check_epipolar_constraint):
   - If the angle θ between bearing vector of keyframe 1 and normal vector of epiplane is near 90 degree
     - Concretely if 90 degree – abs(θ) < 0.2 degree * scale_factor
   - Then the keypoint pair is classified as valid
- Do step 1 to 4 for all keypoints of keyframe1 and keyframe2
- Do angle check explained in [projection::match_current_and_last_frames – 6 of 7]
- At last, matched pairs are all stored in 'matches_idx_pairs' with their keypoint indices as below

→ = bearing vector (bearing_1 in source code)
→ = epipolar vector of keyframe1
→ = normal vector of epiplane (epiplane_in_1 in source code)

vector<pair<unsigned int, unsigned int>> matches_idx_pairs

pair 0	pair 1	pair 2		pair n	
124	1820	493	· ·	81	keypoint indices in new keyframe
349	13	930	· ·	587	keypoint indices in other keyframe

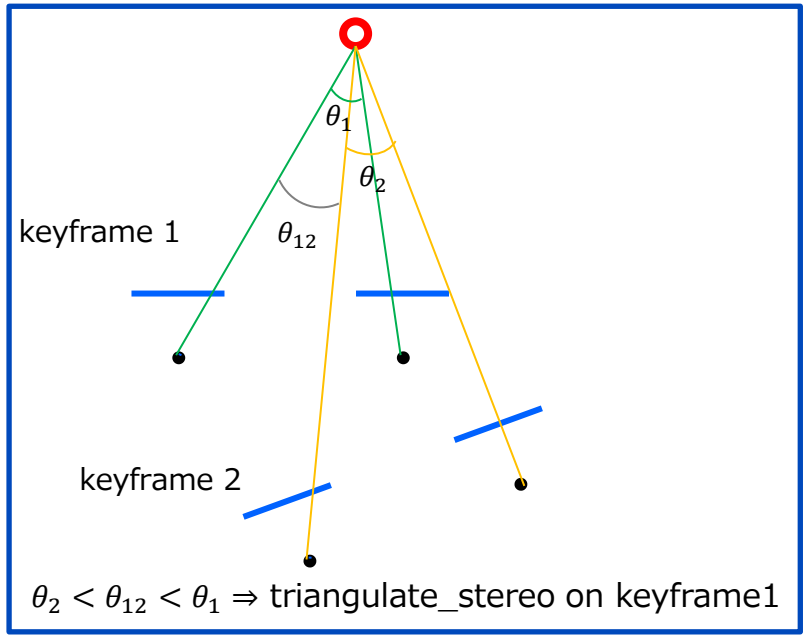
create_new_landmarks – 6 of 8

- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes :
  - If baseline distance between the new keyframe and it is below stereo baseline length, skip it
  - Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)
  - Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks
  - **Call triangulate_with_two_keyframes to add landmarks**

- For each keypoint pair found by robust::match_for_triangulation, calculate 3D world coordinate of the corresponding landmark using **two_view_triangulator::triangulate** (detailed in the next slide)
- If succeeded, then:
  - create a landmark instance with the calculated 3D world coordinate
  - **add_observation** for the landmark with each of two keyframes
  - **add_landmark** for each of two keyframes with the landmark
  - **compute_descritptor for** the landmark (insert_new_keyframe – 5 of 6)
  - **update_normal_and_depth** for the landmark (insert_new_keyframe – 6 of 6)
  - **add_landmark** for map database with the landmark
  - **local_map_cleaner::add_fresh_landmark** with the landmark

create_new_landmarks – 7 of 8

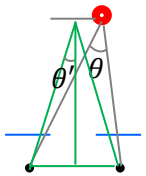
- Get top 10 covisibility keyframes of the new keyframe
- For each keyframe of the covisibility keyframes :
  - If baseline distance between the new keyframe and it is below stereo baseline length, skip it
  - Calculate essential matrix between the new keyframe and it (essential_solver::create_E_21)
  - Call robust::match_for_triangulation to find keypoint pairs which are not corresponded to landmarks
  - **Call triangulate_with_two_keyframes to add landmarks**

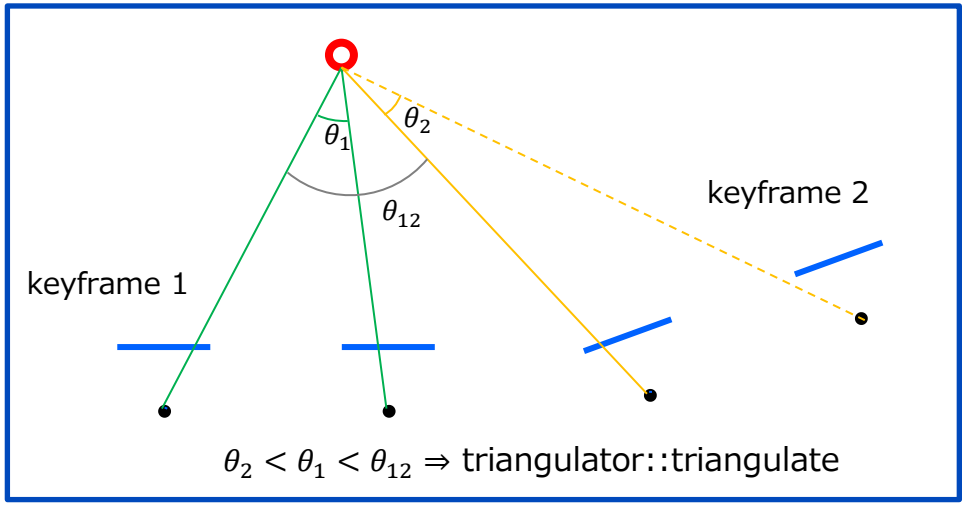


two_view_triangulator::triangulate detail:

- Triangulate with two frames or with stereo depending on parallax:
  - If the angle between [keyframe1 to landmark] and [keyframe2 to landmark] (θ_{12}) is greater than stereo angles (θ_1, θ_1), call **triangulator::triangulate** (detailed in the next page)
  - Else call **triangulate_stereo** on either keyframe on which the stereo angle is greater (see insert_new_keyframe – 2 of 6 about triangulate_stereo)

  Note: stereo angle θ is approximately calculated by 2 $*\theta'$ as below
- The landmark is checked with its depth, reprojection error and scale factors both on keyframe1 and keyframe2
- If passed, a new landmark will be created.





create_new_landmarks – 8 of 8

triangulator::triangulate detail

$$C\boldsymbol{x}_1 = \boldsymbol{X}_1 \;\therefore \boldsymbol{x}_1 \times \boldsymbol{X}_1 = \boldsymbol{0}$$

$$\boldsymbol{X}_1 = (R_{1w} \quad \boldsymbol{t}_{1w})\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = \begin{pmatrix}P_{1w}^1 \\ P_{1w}^2 \\ P_{1w}^3\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix}$$

$$\boldsymbol{x}_1 \times \boldsymbol{X}_1 = \boldsymbol{x}_1 \times \begin{pmatrix}P_{1w}^1 \\ P_{1w}^2 \\ P_{1w}^3\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = \begin{pmatrix}y_1P_{1w}^3 - z_1P_{1w}^2 \\ z_1P_{1w}^1 - x_1P_{1w}^3 \\ x_1P_{1w}^2 - y_1P_{1w}^1\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = \boldsymbol{0}$$

Two of three equations are independent, so $\begin{pmatrix}x_1P_{1w}^2 - y_1P_{1w}^1 \\ y_1P_{1w}^3 - z_1P_{1w}^2\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = \boldsymbol{0}$

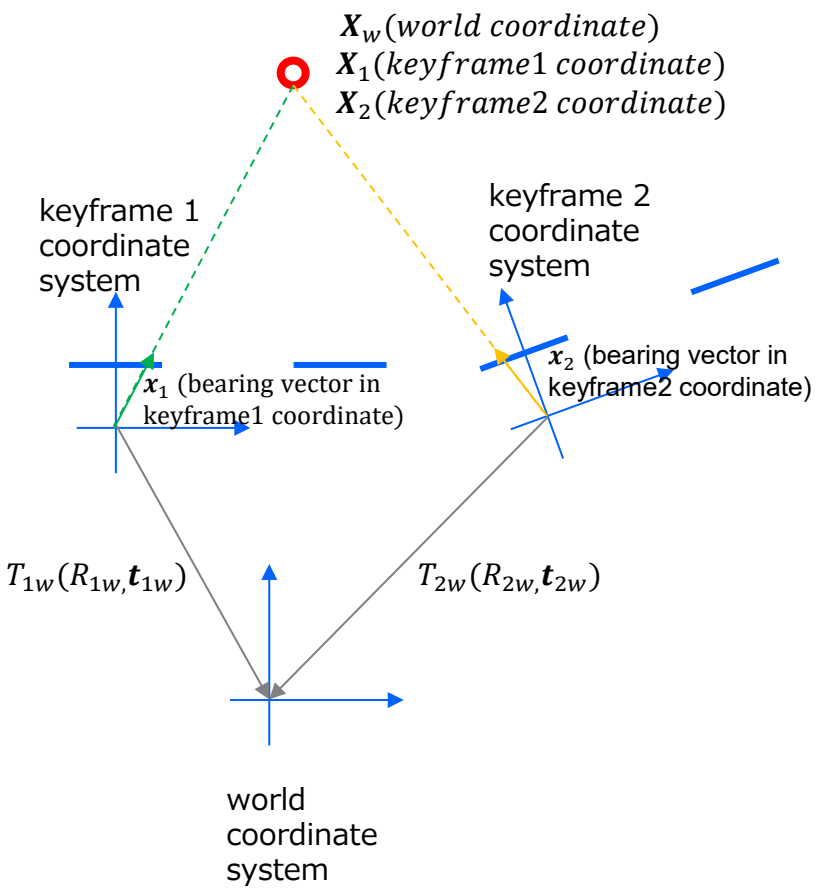
Similarly, $\begin{pmatrix}x_2P_{2w}^2 - y_2P_{2w}^1 \\ y_2P_{2w}^3 - z_2P_{2w}^2\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = \boldsymbol{0} \;\therefore \begin{pmatrix}x_1P_{1w}^2 - y_1P_{1w}^1 \\ y_1P_{1w}^3 - z_1P_{1w}^2 \\ x_2P_{2w}^2 - y_2P_{2w}^1 \\ y_2P_{2w}^3 - z_2P_{2w}^2\end{pmatrix}\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = A\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = U\Lambda V^T\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = \boldsymbol{0}$

$\begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix}$ is constant * eigen vector corresponding to minimum eigen value of $U\Lambda V^T$.

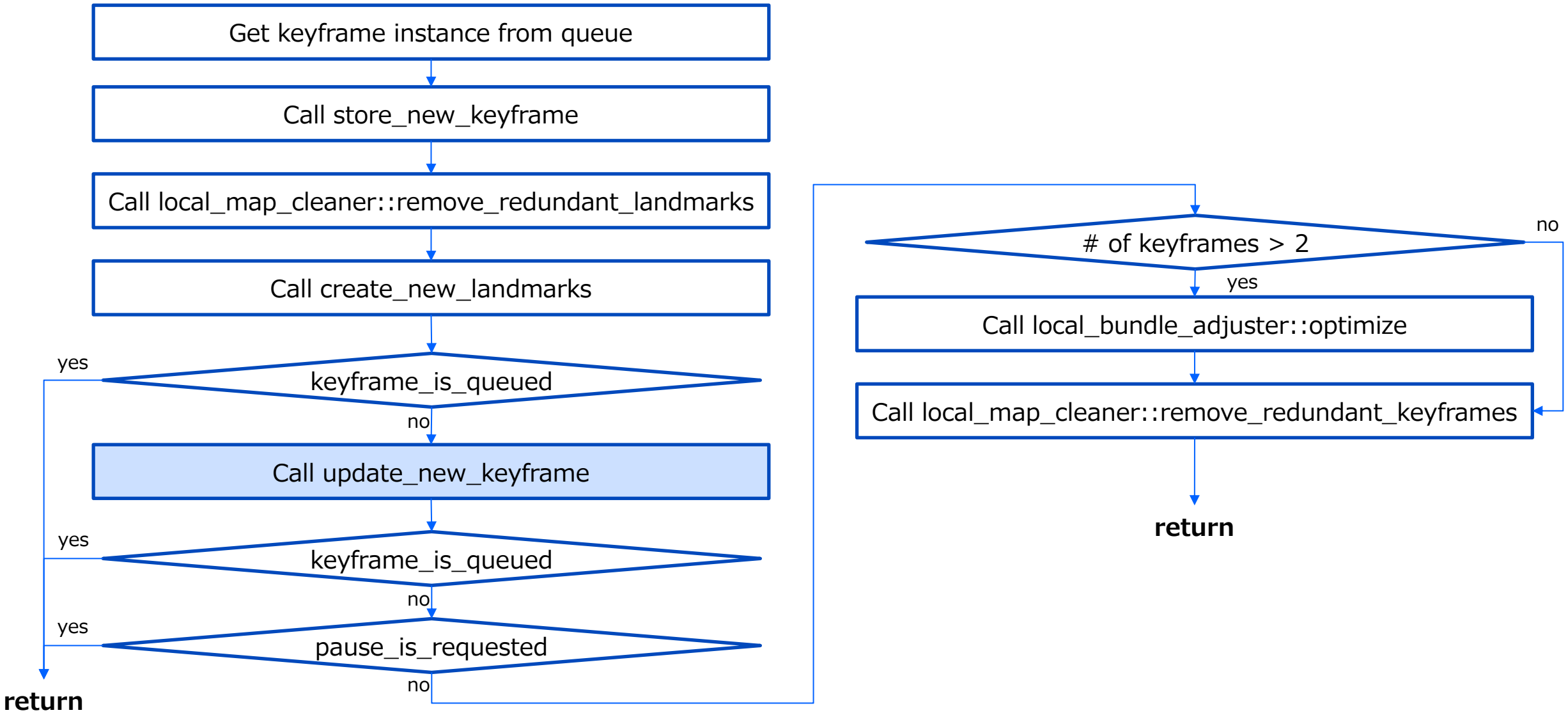
$$\therefore \begin{pmatrix}\boldsymbol{X}_w \\ 1\end{pmatrix} = C\begin{pmatrix}v_{14} \\ v_{24} \\ v_{34} \\ v_{44}\end{pmatrix} \; where \begin{pmatrix}v_{14} \\ v_{24} \\ v_{34} \\ v_{44}\end{pmatrix} is\ 4th\ column\ vector\ of\ matrix\ V$$

and also the eigen vector corresponding to the minimum eigen value.

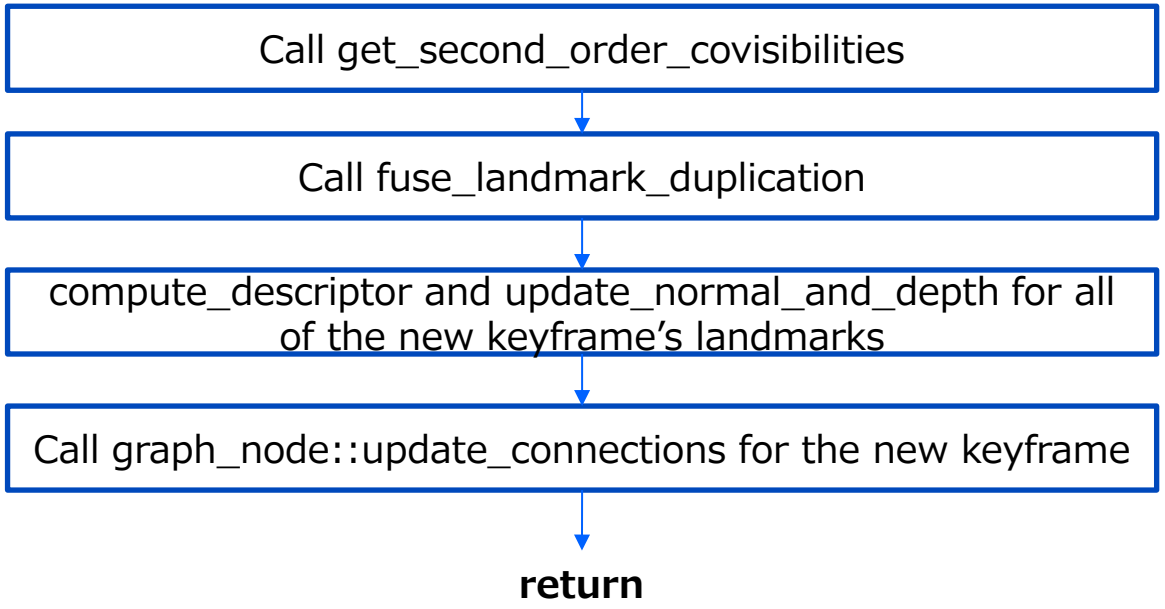
$$\therefore \boldsymbol{X}_w = \begin{pmatrix}v_{14}/v_{44} \\ v_{24}/v_{44} \\ v_{34}/v_{44}\end{pmatrix}$$



Where we are


Get keyframe instance from queue
Call store_new_keyframe
Call local_map_cleaner::remove_redundant_landmarks
Call create_new_landmarks
yes
keyframe_is_queued
no
Call update_new_keyframe
yes
keyframe_is_queued
no
yes
pause_is_requested
no
return
# of keyframes > 2
no
yes
Call local_bundle_adjuster::optimize
Call local_map_cleaner::remove_redundant_keyframes
return

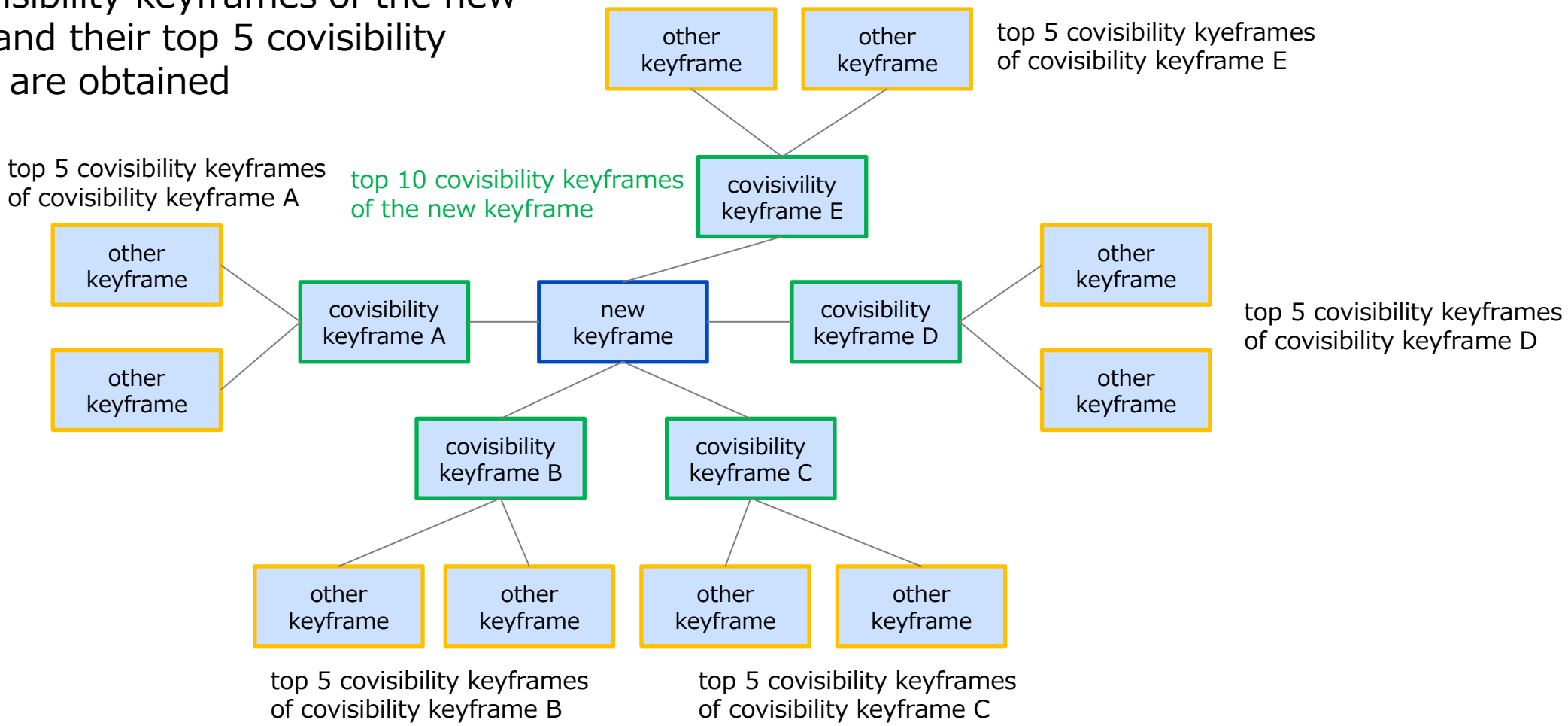
update_new_keyframe – 1 of 10



update_new_keyframe – 2 of 10

get_second_order_covisibilities

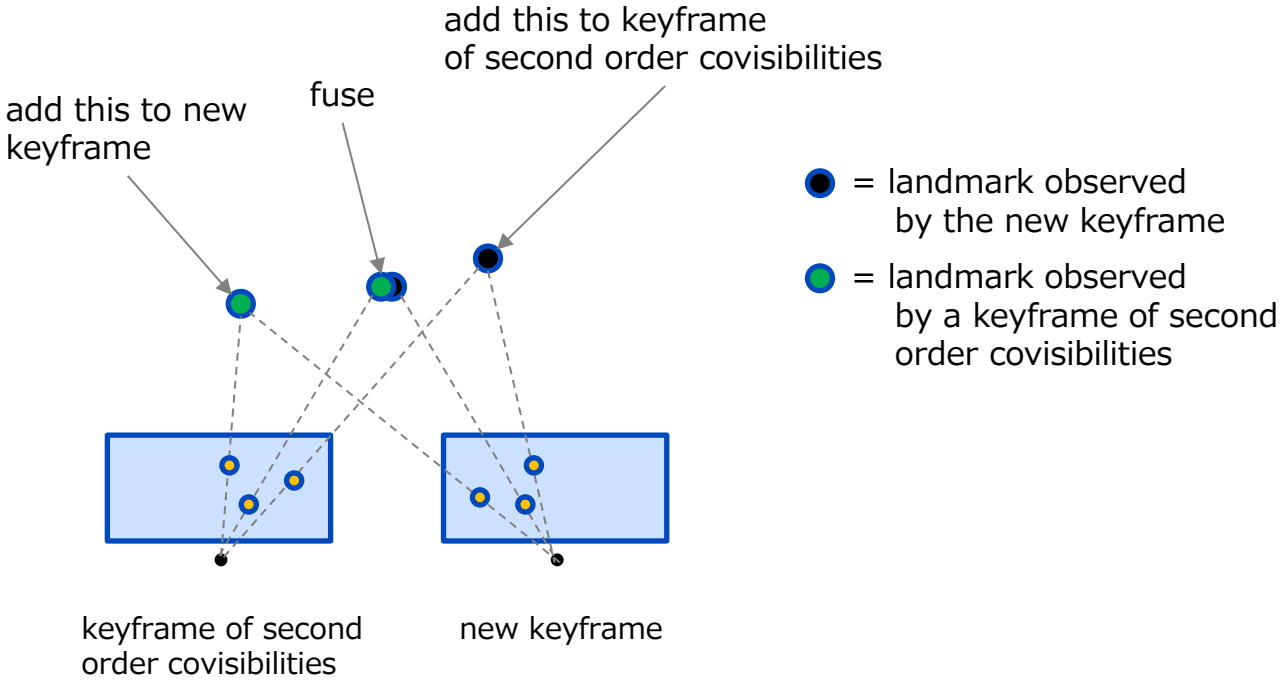
- top 10 covisibility keyframes of the new keyframe and their top 5 covisibility keyframes are obtained



update_new_keyframe – 3 of 10

fuse_landmark_duplication

- Between the new keyframe and the second order covisibilities:
  - Fuse landmarks if they are separately registered but recognized as the same
  - Add landmarks which are observed only from either keyframe to another keyframe
- Do this by calling fuse::replace_duplication

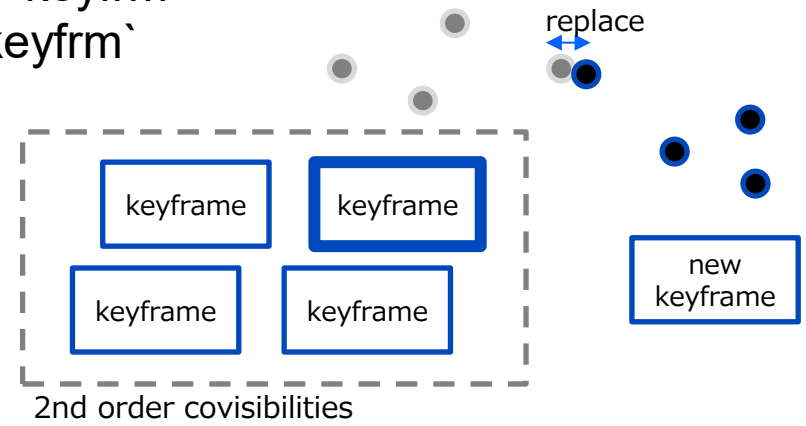


update_new_keyframe – 4 of 10

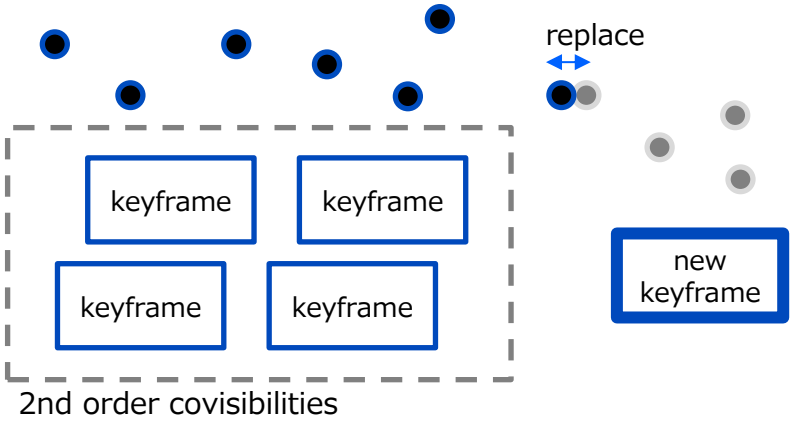
fuse::replace_duplication(keyframe* keyfrm, const T& landmarks_to_check, const float margin)

- If duplicated, do either below depending on observation count:
  - Replace a landmark of `keyfrm` with a landmark in `landmarks_to_check`
  - Replace a landmark in `landmarks_to_check` with a landmark of `keyfrm`
- Add a landmark in `landmarks_to_check` to `keyfrm` if not found in `keyfrm`

For each keyframe of 2nd order covisibilities:
- Call fuse::replace_duplication(the 2nd order covisibility,
  landmarks of the new keyframe)



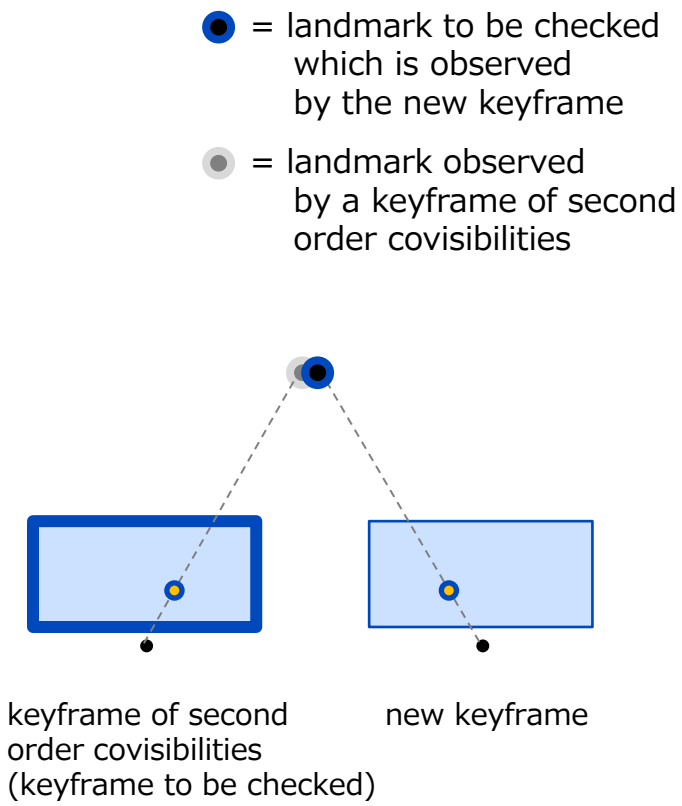
- Call fuse::replace_duplication(the new keyframe,
  all landmarks of 2nd order covisibilities)



update_new_keyframe – 5 of 10

fuse::replace_duplication

- Reproject landmark to be checked to keyframe to be checked
- Search the best matched keypoint with the reprojected point by calculating reprojection error between the keypoint and the reprojected point and by Hamming distance
- Obtain a landmark corresponding to the best matched keypoint
- If obtained:
  - Replace either landmark with another landmark which has more observations
- Else:
  - Add the landmark to keyframe to be checked
  - Add observation to the landmark



update_new_keyframe – 6 of 10

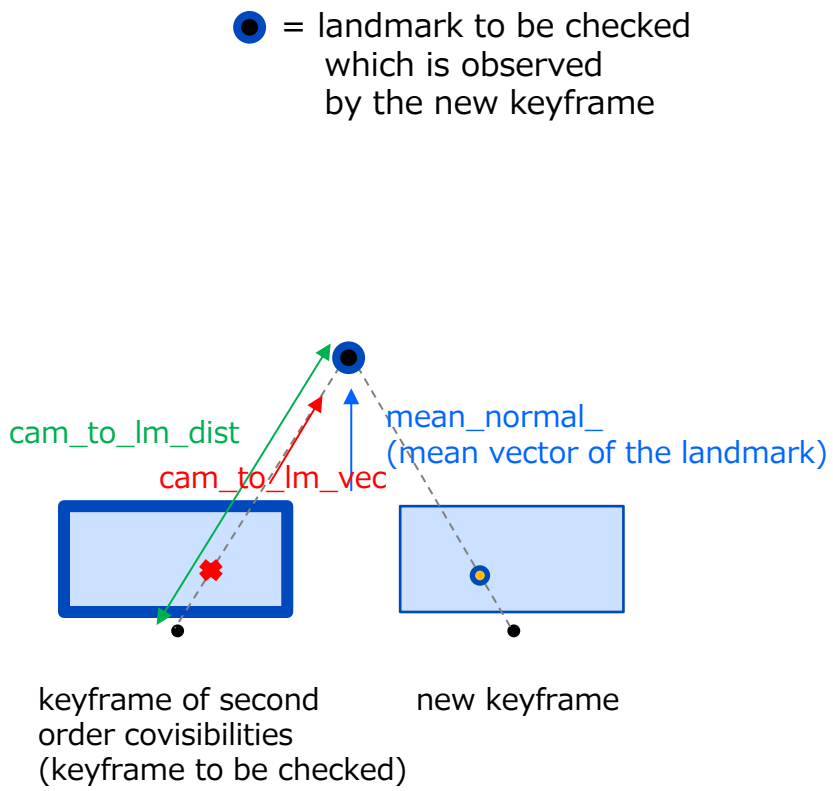
fuse::replace_duplication

- **Reproject landmark to be checked to keyframe to be checked**
- Search the best matched keypoint with the

See projection::match_current_and_last_frames – 3 of 7 about reprojection.

Skip matching if:
- cam_to_lm_dist in the right figure is above max_valid_dist_ of the landmark
- cam_to_lm_dist is below min_valid_dist_ of the landmark
- Angle between cam_to_lm_vec and mean_normal_ in the right figure is above 60 degree

See insert_new_keyframe – 6 of 6 about mean_normal_, max_valid_dist_ and min_valid_list_.



update_new_keyframe – 7 of 10

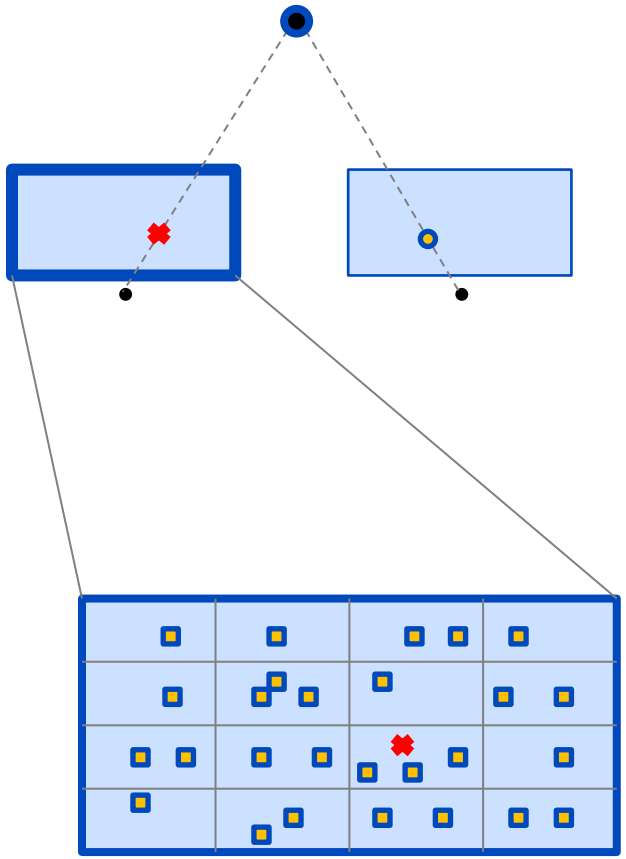
fuse::replace_duplication

- Reprojection landmark to be checked to keyframe to be checked
- **Search the best matched keypoint with the reprojected point by calculating reprojection error between the keypoint and the reprojected point and by Hamming distance**
- Obtain a landmark corresponding to the best matched keypoint

Obtain keypoints within grid cells around the reprojected point as candidates.
About grid cell, see assign_keypoints_to_grid.

For each keypoint of the candidates, calculate reprojection error from the reprojected point and evaluate it by χ^2 significance level of 5%.
If passed, calculate Hamming distance.

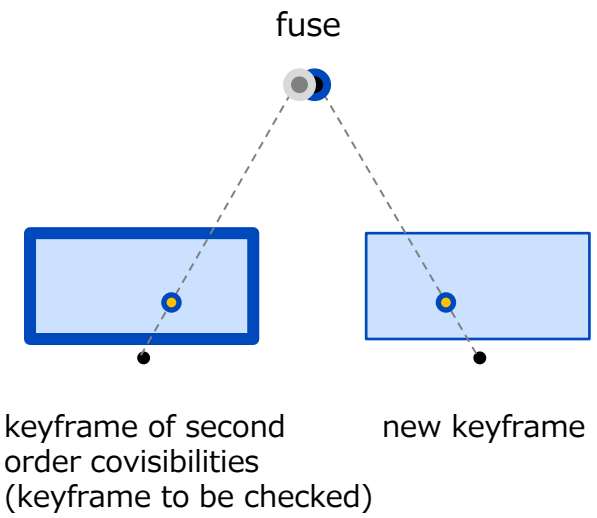
If Hamming distance is below HAMMING_DIST_THR_LOW and is the minimum among all, it is the one to be sought.



update_new_keyframe – 8 of 10

fuse::replace_duplication

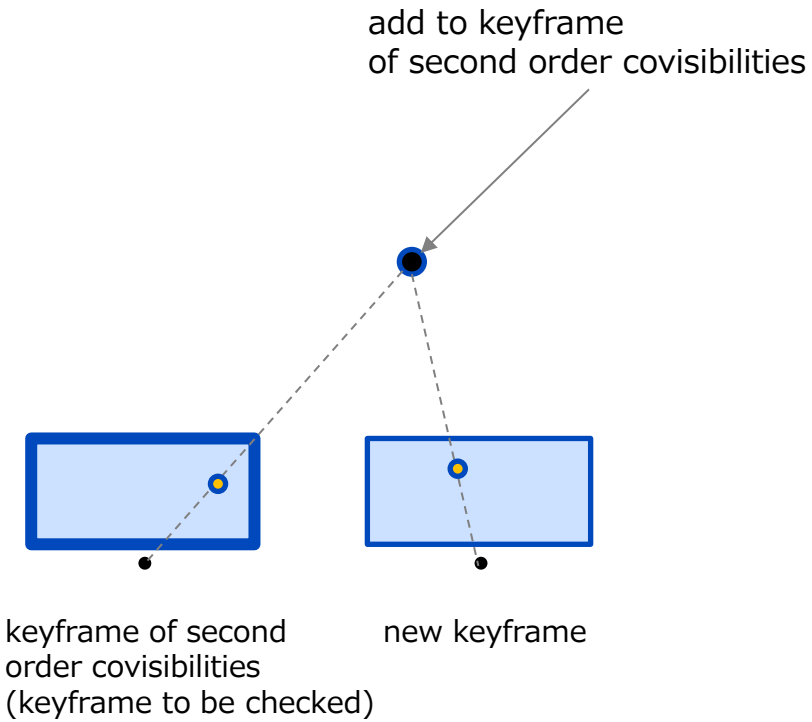
- Reproject landmark to be checked to keyframe to be checked
- Search the best matched keypoint with the reprojected point by calculating reprojection error between the keypoint and the reprojected point and by Hamming distance
- **Obtain a landmark corresponding to the best matched keypoint**
- **If obtained:**
  - **Replace either landmark with another landmark which has more observations**
- Else:
  - Add the landmark to keyframe to be checked
  - Add observation to the landmark



update_new_keyframe – 9 of 10

fuse::replace_duplication

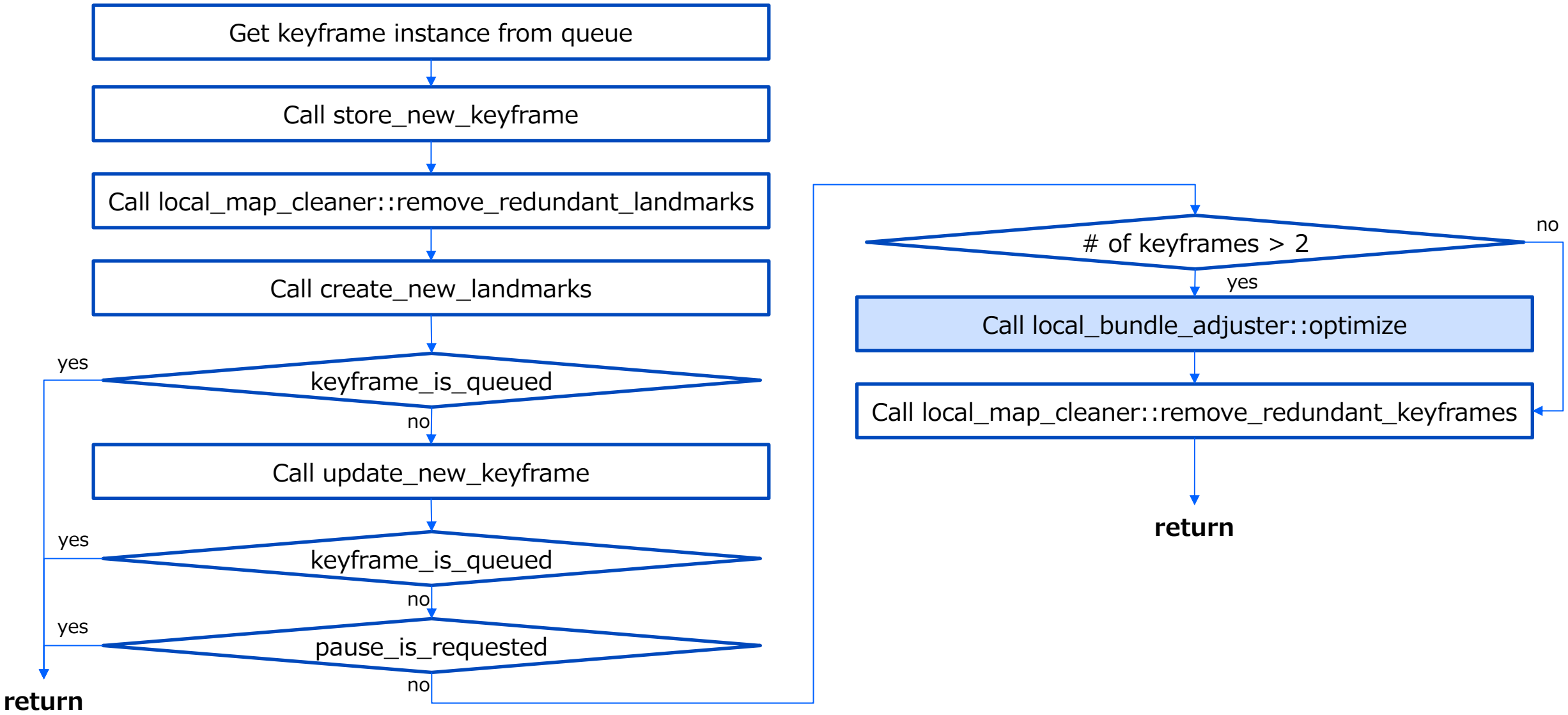
- Reproject landmark to be checked to keyframe to be checked
- Search the best matched keypoint with the reprojected point by calculating reprojection error between the keypoint and the reprojected point and by Hamming distance
- **Obtain a landmark corresponding to the best matched keypoint**
- If obtained:
  - Replace either landmark with another landmark which has more observations
- **Else:**
  - **Add the landmark to keyframe to be checked**
  - **Add observation to the landmark**



update_new_keyframe – 10 of 10

- For each of all landmarks of the new keyframe:
  - Call compute_descriptor
    - Already explained in insert_new_keyframe – 5 of 6
  - Call update_normal_and_depth
    - Already explained in insert_new_keyframe – 6 of 6
- Update graph by calling graph_node::update_connections
  - Already explained in store_new_keyframe – 2 of 7 to store_new_keyframe – 6 of 7
- These are needed because the landmarks observed from keyframes may have changed through fuse_landmark_duplication

Where we are


Get keyframe instance from queue
Call store_new_keyframe
Call local_map_cleaner::remove_redundant_landmarks
Call create_new_landmarks
yes
keyframe_is_queued
no
Call update_new_keyframe
yes
keyframe_is_queued
no
yes
pause_is_requested
no
return
# of keyframes > 2
no
yes
Call local_bundle_adjuster::optimize
Call local_map_cleaner::remove_redundant_keyframes
return

local_bundle_adjuster::optimize – 1 of 9

- Optimize with g2o
  - Linear solver = LinearSolverCSparse
  - Block solver = BlockSolver_6_3
  - Algorithm = OptimizationAlgorithmLevenberg
  - Optimizer = SparseOptimizer
- Vertices = local keyframe poses and local landmark positions
- Edges = restriction between local keyframes and reprojected points of local landmarks
  - Robust kernel (Huber loss) is set with delta = sqrt of χ^2 value of threshold of significance level of 5%

local_bundle_adjuster::optimize – 2 of 9

□ + □ = local keyframes ⇒ **added as vertices with setFixed(false) [if keyframe of id=0, setFixed(true)]**

● + ○ = local landmarks ⇒ **added as vertices**

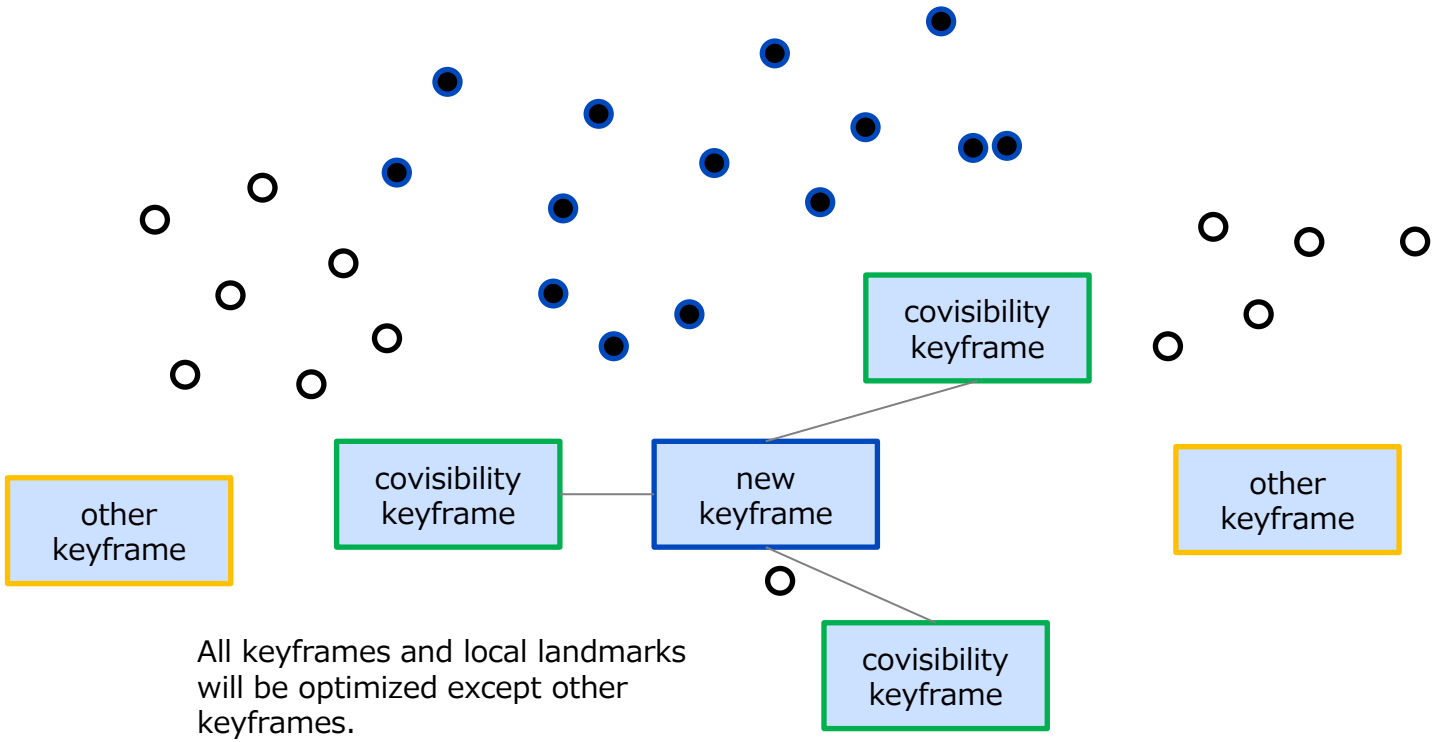
edge

edge

□ = keyframes which observe local landmarks but has no common one with the current frame ⇒ **added as vertices with setFixed(true)**

● = landmarks observed from the new keyframe

○ = local landmarks observed from local keyframes but not from the new keyframe



All keyframes and local landmarks will be optimized except other keyframes.

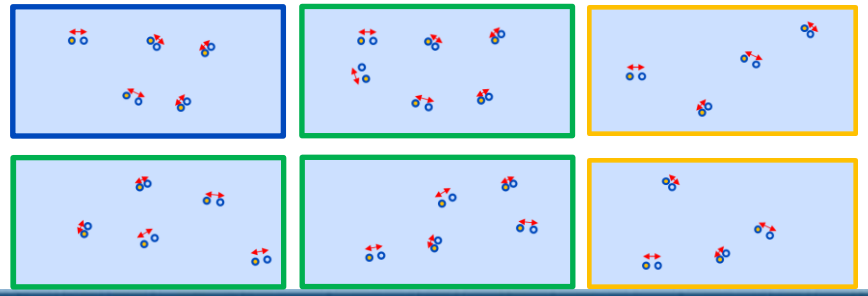
The local keyframe poses including the new keyframe's one, and local landmark positions will be optimized through g2o optimization. The optimization will be done to minimize total reprojection errors of all the keyframes added as vertices.

● = keypoint

○ = reprojected point of landmark

↔ = reprojection error

Note that reprojection error is considered both on the left and right images when stereo visible as in pose_optimizer::optimize – 3 of 9.



local_bundle_adjuster::optimize – 3 of 9

= keypoint

= reprojected point of landmark

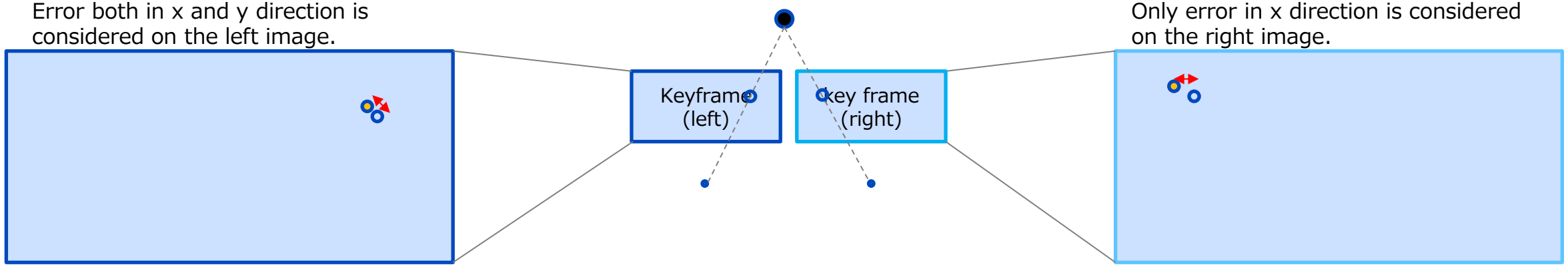
= reprojection error

stereo visible

Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i, x_right_i - reproj_x_right_i)^T$

Error both in x and y direction is considered on the left image.

Only error in x direction is considered on the right image.

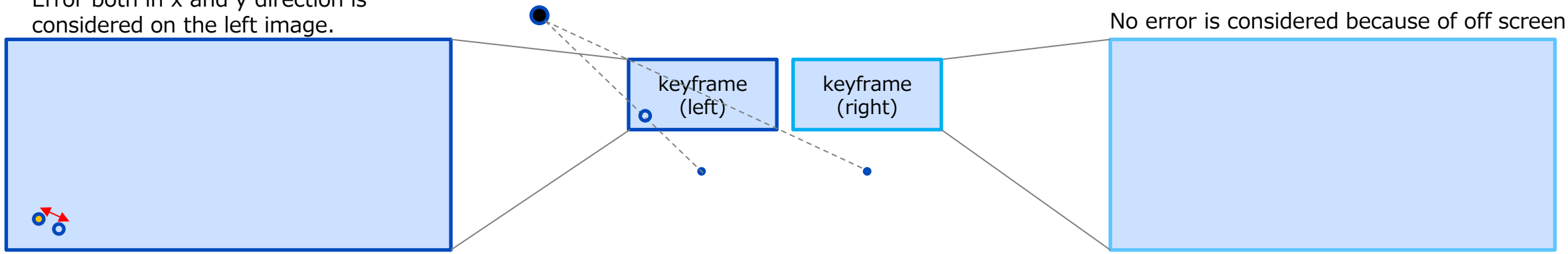


monocular only visible

Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i)^T$

Error both in x and y direction is considered on the left image.

No error is considered because of off screen.



local_bundle_adjuster::optimize – 4 of 9

- Vertices details:
  - keyframe vertex (g2o::se3::shot_vertex):
    - Derived class of ::g2o::BaseVertex<6, ::g2o::SE3Quat>
    - Keyframe poses (cam_pose_cw) are set through setEstimate at first
    - oplusImpl does setEstimate with $\exp(\Delta) * \text{estimate}()$
  - landmark vertex (g2o::landmark_vertex):
    - Derived class of ::g2o::BaseVertex<3, Vec3_t>
    - Positions (pos_w) are set through setEstimate at first
    - oplusImple does setEstimate with _estimate + Δ

local_bundle_adjuster::optimize – 5 of 9

- Edge details (stereo_perspective_reproj_edge derived from ::g2o::BaseBinaryEdge<3, Vec3_t, landmark_vertex, shot_vertex>):

  used when stereo visible

  - Measurement is (x_i, y_i, x_right_i)
  - Information matrix is diagonal matrix whose components value depend on pyramid level of the keypoint
  - Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i, x_right_i - reproj_x_right_i)^T$

$$= \left(x_i - f_x \frac{X_c^i}{Z_c^i} - c_x, y_i - f_y \frac{Y_c^i}{Z_c^i} - c_y, x_{right_i} - f_x \frac{X_c^i}{Z_c^i} - c_x + f_x \frac{baseline}{Z_c^i}\right)^T$$

  written in computeError() of perspective_reproj_edge.h

  - Jacobian for $\boldsymbol{X}_w$ is $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_w} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{X}_w}$

    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} = \begin{pmatrix} -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \\ -\frac{f_x}{Z_c} & 0 & \frac{f_x}{Z_c^2}(X_c - baseline) \end{pmatrix}, \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{X}_w} = R_{cw} = \begin{pmatrix} R_{cw}^{11} & R_{cw}^{12} & R_{cw}^{13} \\ R_{cw}^{21} & R_{cw}^{22} & R_{cw}^{23} \\ R_{cw}^{31} & R_{cw}^{32} & R_{cw}^{33} \end{pmatrix}$

    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_w} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{X}_w} =$

      written as _jacobianOplusXi in linearizeOplus() of perspective_reproj_edge.cc

$$\begin{pmatrix} -\frac{f_x}{Z_c} R_{cw}^{11} + \frac{f_x X_c}{Z_c^2} R_{cw}^{31} & -\frac{f_x}{Z_c} R_{cw}^{12} + \frac{f_x X_c}{Z_c^2} R_{cw}^{32} & -\frac{f_x}{Z_c} R_{cw}^{13} + \frac{f_x X_c}{Z_c^2} R_{cw}^{33} \\ -\frac{f_y}{Z_c} R_{cw}^{21} + \frac{f_y Y_c}{Z_c^2} R_{cw}^{31} & -\frac{f_y}{Z_c} R_{cw}^{22} + \frac{f_y Y_c}{Z_c^2} R_{cw}^{32} & -\frac{f_y}{Z_c} R_{cw}^{23} + \frac{f_y Y_c}{Z_c^2} R_{cw}^{33} \\ -\frac{f_x}{Z_c} R_{cw}^{11} + \frac{f_x}{Z_c^2}(X_c - baseline) R_{cw}^{31} & -\frac{f_x}{Z_c} R_{cw}^{12} + \frac{f_x}{Z_c^2}(X_c - baseline) R_{cw}^{32} & -\frac{f_x}{Z_c} R_{cw}^{13} + \frac{f_x}{Z_c^2}(X_c - baseline) R_{cw}^{33} \end{pmatrix}$$

local_bundle_adjuster::optimize – 6 of 9

- Edge details (stereo_perspective_reproj_edge , continued)
  - Jacobian for $\boldsymbol{p}$ is $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c}\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$

- $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} = \begin{pmatrix} -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \\ -\frac{f_x}{Z_c} & 0 & \frac{f_x}{Z_c^2}(X_c - baseline) \end{pmatrix}, \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} 0 & Z_c & -Y_c & 1 & 0 & 0 \\ -Z_c & 0 & X_c & 0 & 1 & 0 \\ Y_c & -X_c & 0 & 0 & 0 & 1 \end{pmatrix}$

written as _jacobianOplusXj in linearizeOplus() of perspective_reproj_edge.cc

- $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c}\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} \frac{f_x X_c Y_c}{Z_c^2} & -f_x(1+\frac{X_c^2}{Z_c^2}) & \frac{f_x Y_c}{Z_c} & -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ f_y(1+\frac{Y_c^2}{Z_c^2}) & -\frac{f_y X_c Y_c}{Z_c^2} & -\frac{f_y X_c}{Z_c} & 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \\ \frac{f_x Y_c}{Z_c^2}(X_c - baseline) & -f_x(1+\frac{X_c^2}{Z_c^2} - \frac{X_c \cdot baseline}{Z_c^2}) & \frac{f_x Y_c}{Z_c} & -\frac{f_x}{Z_c} & 0 & \frac{f_x}{Z_c^2}(X_c - baseline) \end{pmatrix}$

About derivation of $\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$, see pose_optimizer::optimize – 7 of 9 and pose_optimizer::optimize – 8 of 9

local_bundle_adjuster::optimize – 7 of 9

- Edge details (mono_perspective_reproj_edge derived from ::g2o::BaseBinaryEdge<2, Vec2_t, landmark_vertex, shot_vertex>):

  used when only monocular visible

  - Measurement is (x_i, y_i)
  - Information matrix is diagonal matrix whose components value depend on pyramid level of the keypoint
  - Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i)^T$

$$= \left(x_i - f_x \frac{X_c^i}{Z_c^i} - c_x, y_i - f_y \frac{Y_c^i}{Z_c^i} - c_y\right)^T$$

  written in computeError() of perspective_reproj_edge.h

  - Jacobian for $\boldsymbol{X}_w$ is $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_w} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{X}_w}$
    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} = \begin{pmatrix} -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \end{pmatrix}, \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{X}_w} = R_{cw} = \begin{pmatrix} R_{cw}^{11} & R_{cw}^{12} & R_{cw}^{13} \\ R_{cw}^{21} & R_{cw}^{22} & R_{cw}^{23} \\ R_{cw}^{31} & R_{cw}^{32} & R_{cw}^{33} \end{pmatrix}$
    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_w} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{X}_w} = \begin{pmatrix} -\frac{f_x}{Z_c} R_{cw}^{11} + \frac{f_x X_c}{Z_c^2} R_{cw}^{31} & -\frac{f_x}{Z_c} R_{cw}^{12} + \frac{f_x X_c}{Z_c^2} R_{cw}^{32} & -\frac{f_x}{Z_c} R_{cw}^{13} + \frac{f_x X_c}{Z_c^2} R_{cw}^{33} \\ -\frac{f_y}{Z_c} R_{cw}^{21} + \frac{f_y Y_c}{Z_c^2} R_{cw}^{31} & -\frac{f_y}{Z_c} R_{cw}^{22} + \frac{f_y Y_c}{Z_c^2} R_{cw}^{32} & -\frac{f_y}{Z_c} R_{cw}^{23} + \frac{f_y Y_c}{Z_c^2} R_{cw}^{33} \end{pmatrix}$

  written as _jacobianOplusXi in linearizeOplus() of perspective_reproj_edge.cc

local_bundle_adjuster::optimize – 8 of 9

- Edge details (mono_perspective_reproj_edge , continued)
  - Jacobian for $\boldsymbol{p}$ is $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c}\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$
    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c} = \begin{pmatrix} -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \end{pmatrix}, \frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} 0 & Z_c & -Y_c & 1 & 0 & 0 \\ -Z_c & 0 & X_c & 0 & 1 & 0 \\ Y_c & -X_c & 0 & 0 & 0 & 1 \end{pmatrix}$
    - $\frac{\partial \boldsymbol{E}}{\partial \boldsymbol{p}} = \frac{\partial \boldsymbol{E}}{\partial \boldsymbol{X}_c}\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}} = \begin{pmatrix} \frac{f_x X_c Y_c}{Z_c^2} & -f_x(1+\frac{X_c^2}{Z_c^2}) & \frac{f_x Y_c}{Z_c} & -\frac{f_x}{Z_c} & 0 & \frac{f_x X_c}{Z_c^2} \\ f_y(1+\frac{Y_c^2}{Z_c^2}) & -\frac{f_y X_c Y_c}{Z_c^2} & -\frac{f_y X_c}{Z_c} & 0 & -\frac{f_y}{Z_c} & \frac{f_y Y_c}{Z_c^2} \end{pmatrix}$

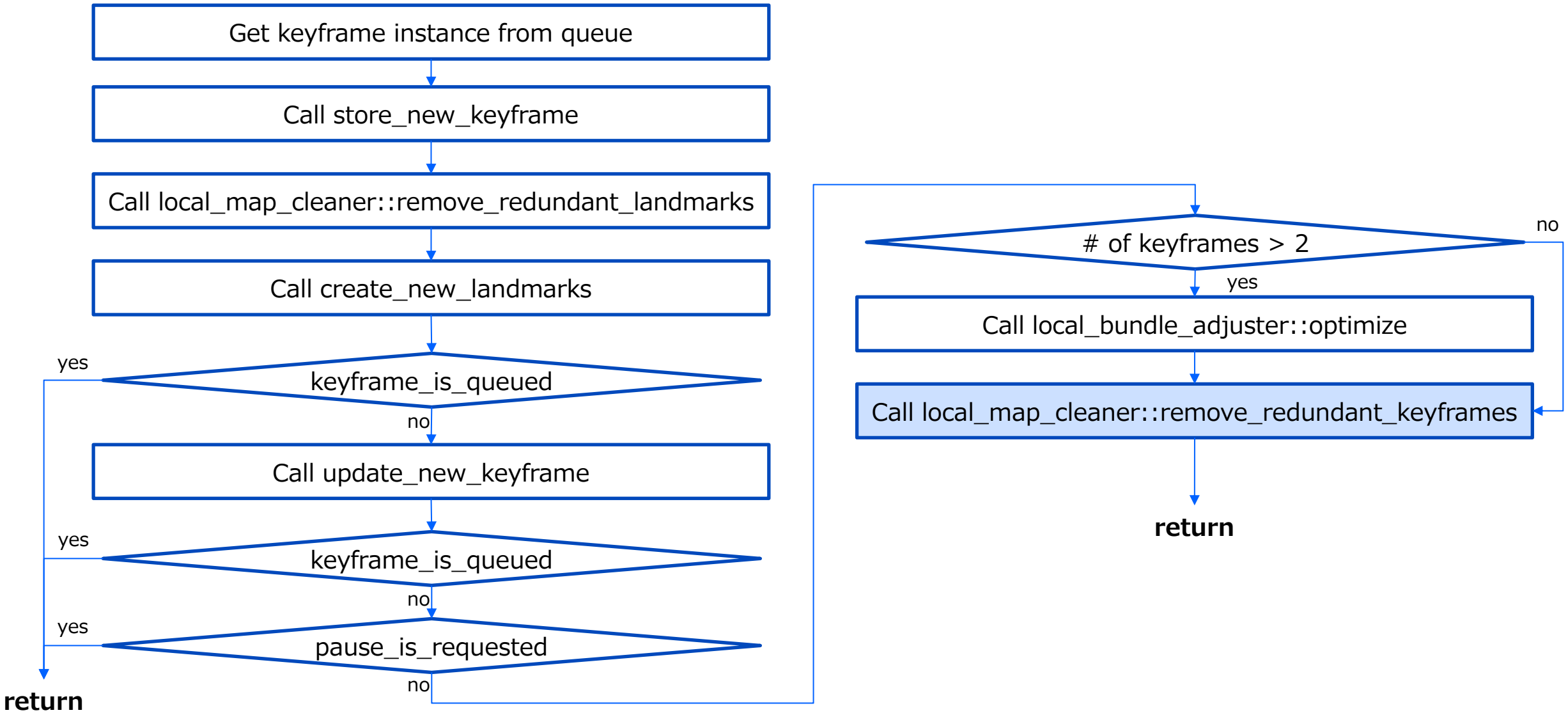
written as _jacobianOplusXj in linearizeOplus() of perspective_reproj_edge.cc

About derivation of $\frac{\partial \boldsymbol{X}_c}{\partial \boldsymbol{p}}$, see pose_optimizer::optimize – 7 of 9 and pose_optimizer::optimize – 8 of 9

local_bundle_adjuster::optimize – 9 of 9

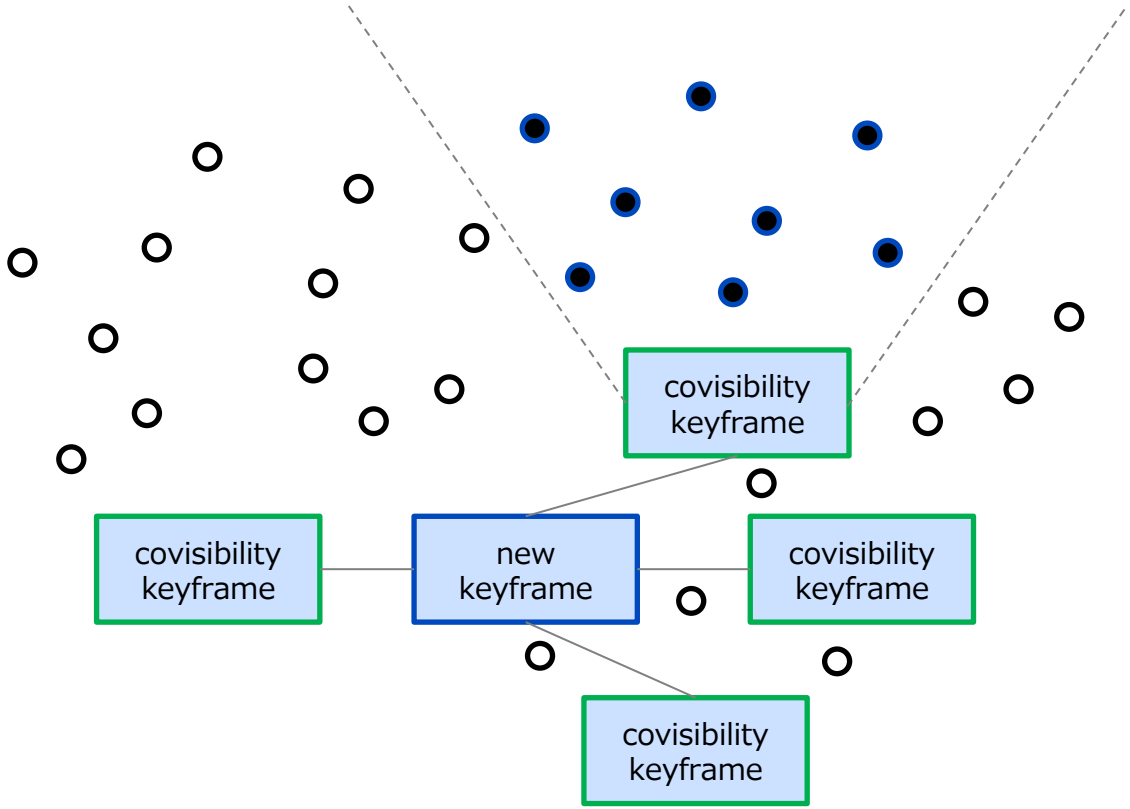
- Do optimization
  - Call optimizer.optimize once
  - Reject Edges whose χ^2 value is greater than threshold of significance level of 5%
  - Call optimizer.optimize again
- After optimization
  - Outliers of landmarks are removed from keyframes' landmarks vector and do erase_observation of the keyframes
  - Set optimized poses to keyframes
  - Set optimized position to landmarks and call update_normal_and_depth for each landmark

Where we are


Get keyframe instance from queue
Call store_new_keyframe
Call local_map_cleaner::remove_redundant_landmarks
Call create_new_landmarks
yes
keyframe_is_queued
no
Call update_new_keyframe
yes
keyframe_is_queued
no
yes
pause_is_requested
no
return
# of keyframes > 2
no
yes
Call local_bundle_adjuster::optimize
Call local_map_cleaner::remove_redundant_keyframes
return

local_map_cleaner::remove_redundant_keyframes – 1 of 12

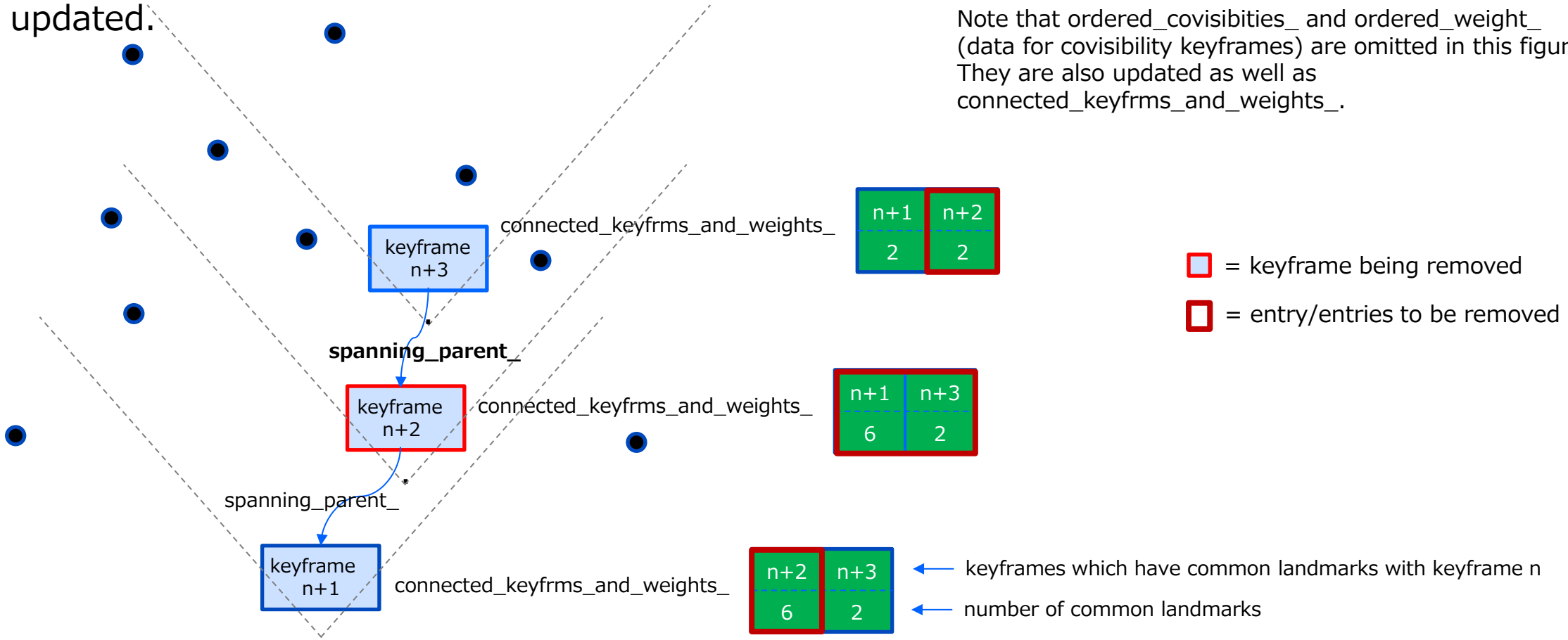
- For each keyframe of the current frame's covisibility keyframes, do followings:
  - For each landmark observed from the keyframe:
    - it is regarded as redundantly observed from the keyframe if following condition is met:
      - 3 or more keyframes have reliable scale level of the keypoint corresponding to that of the concerned keyframe (reliable scale level = equal to or less than the scale level of the keypoint on the concerned keyframe plus 1)
  - Count the number of redundant observation and the number of all landmarks
  - If the ratio of the number of redundant observation to the number of all landmarks is greater or equal to redundant_obs_ratio_thr (=0.9), the keyframe is regarded as redundant
  - The redundant keyframe is erased by calling **keyframe::prepare_for_erasing**



keyframe::prepare_for_erasing
(graph_node::erase_all_connections)

Relationship of connected and covisibility keyframes are updated.

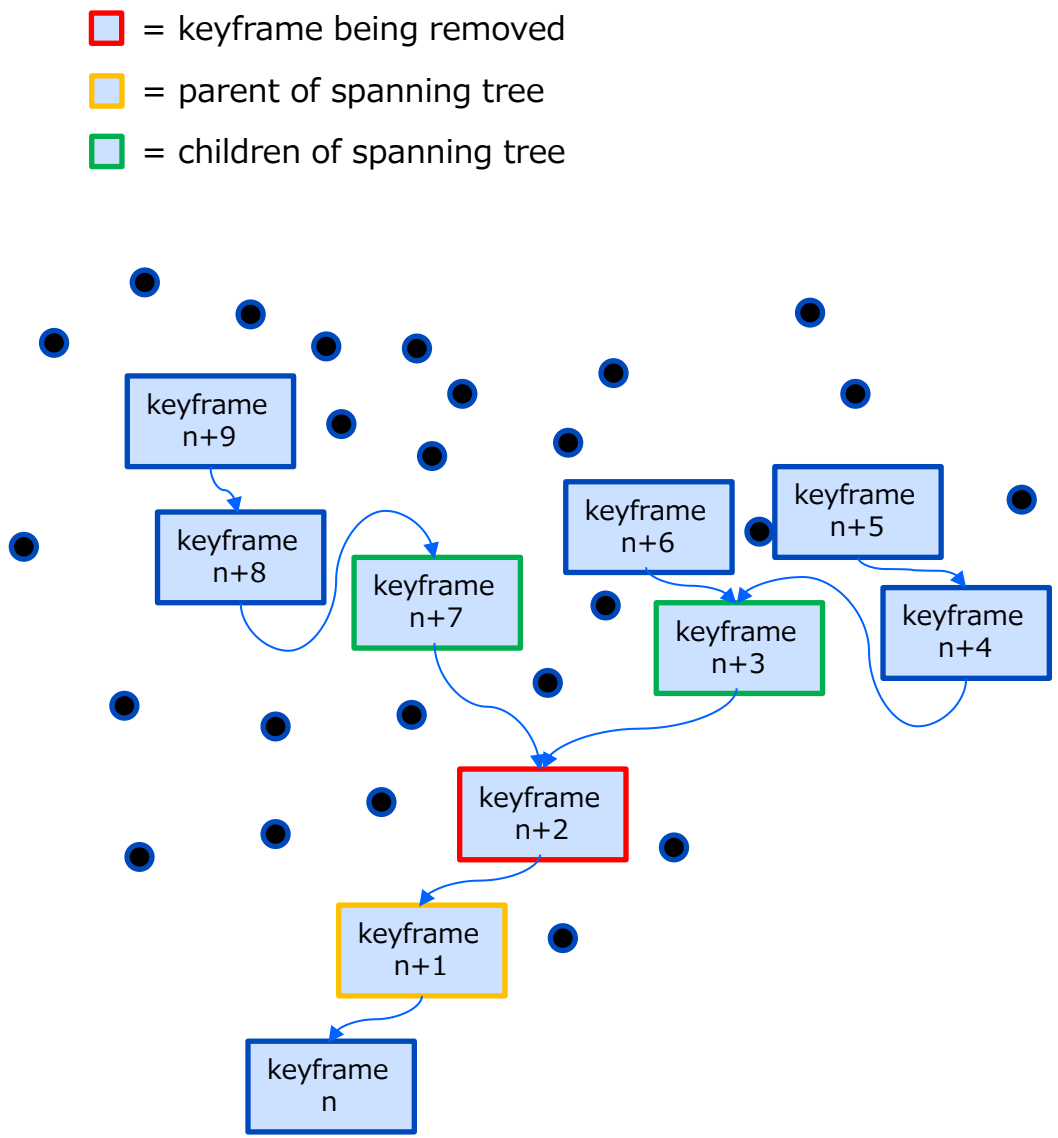
Note that ordered_covisibities_ and ordered_weight_ (data for covisibility keyframes) are omitted in this figure. They are also updated as well as connected_keyfrms_and_weights_.



local_map_cleaner::remove_redundant_keyframes – 3 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 1/8)

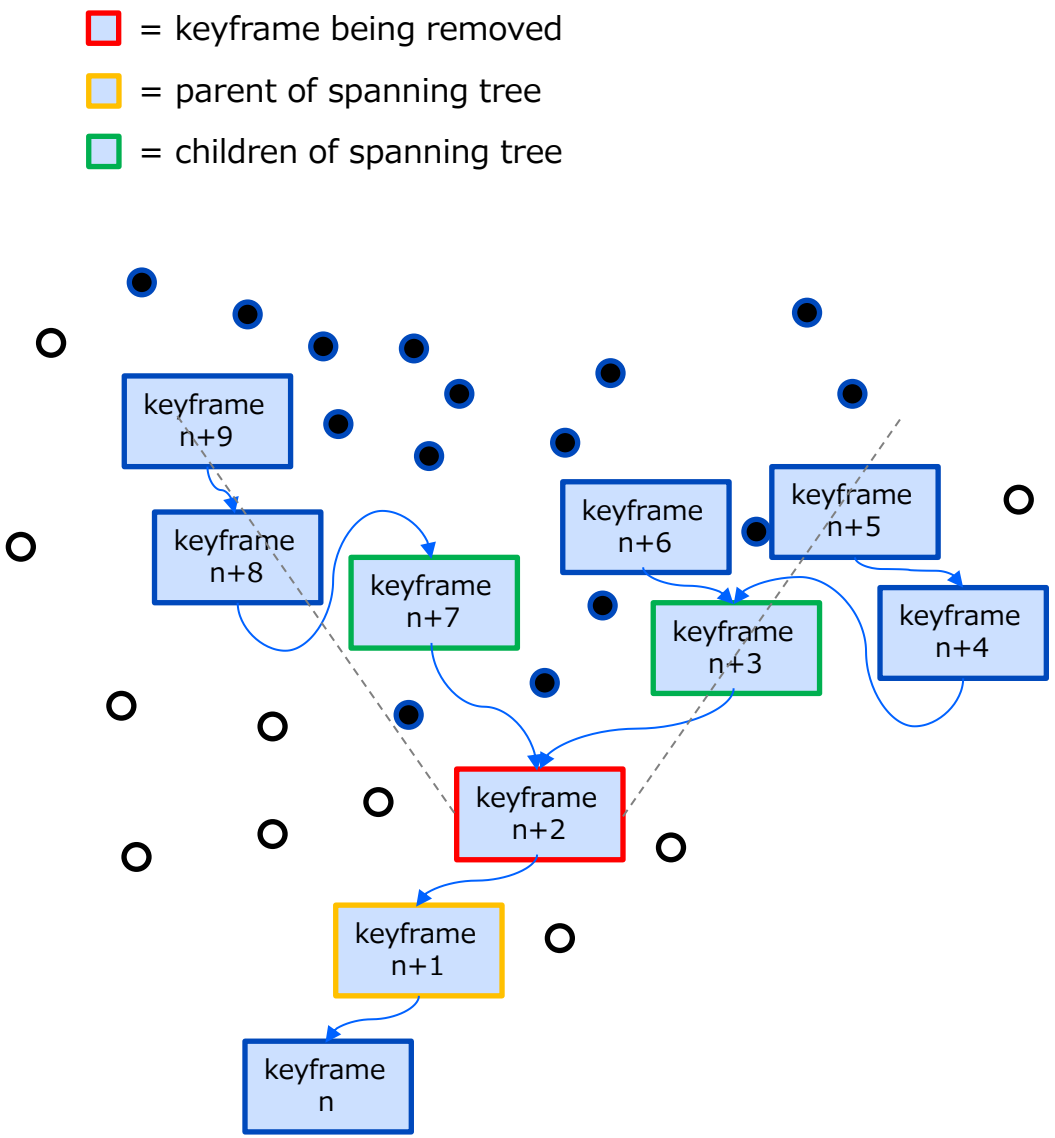
1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 1) If found, change spanning tree parent of the found child and then regard it as parent candidate
   2) If not found, the parent become their spanning tree parent (then skip 4 and 5)
4. Go back to 2 and find a keyframe which has the most common landmarks with the spanning tree parent or the parent candidate
   - Among all parent and parent candidates, the keyframe which has the most common landmarks with the child becomes its spanning tree parent
5. Do this for all spanning children



local_map_cleaner::remove_redundant_keyframes – 4 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 2/8)

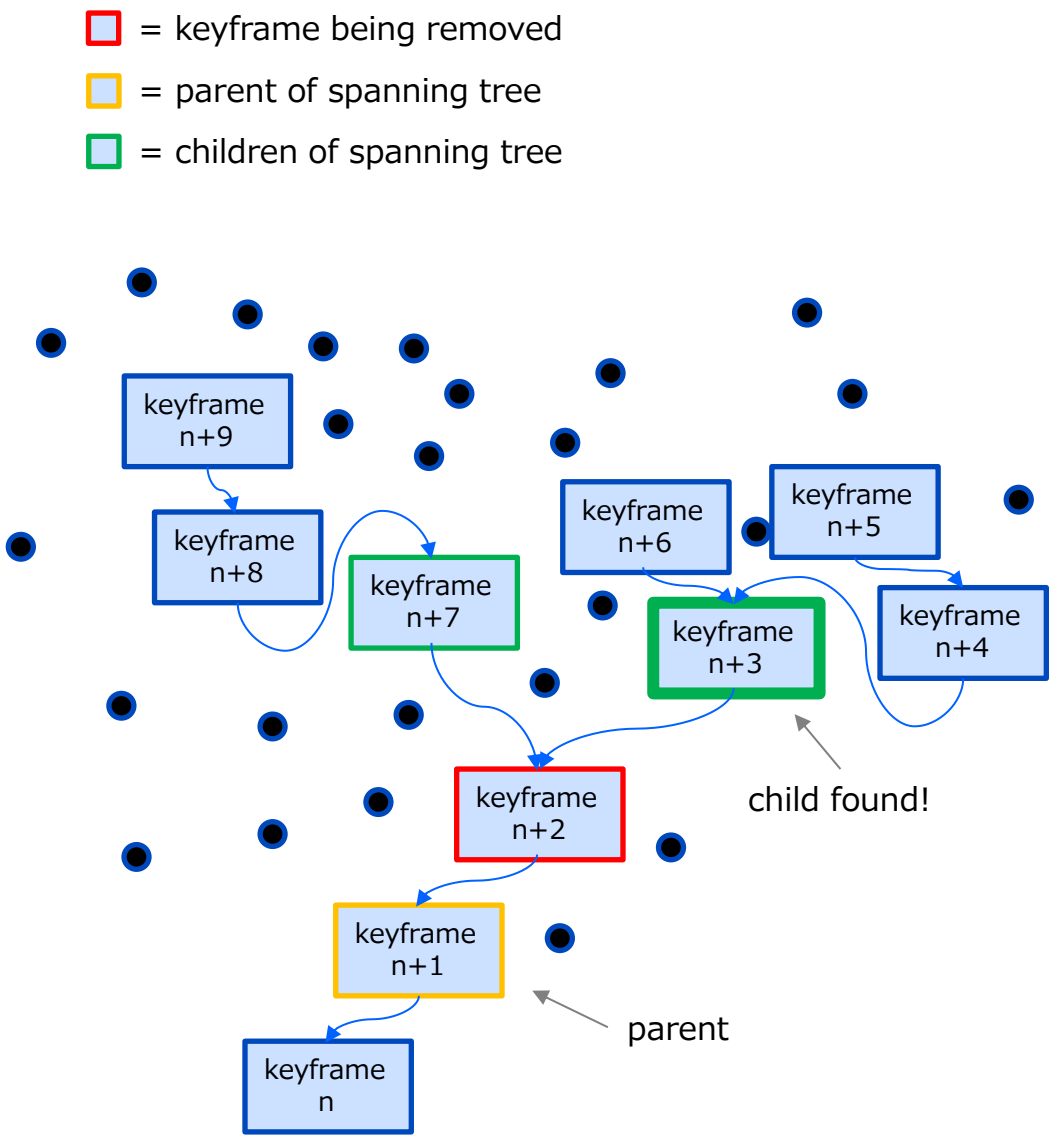
1. As for the keyframe being removed, erase_observation from all landmarks



local_map_cleaner::remove_redundant_keyframes – 5 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 3/8)

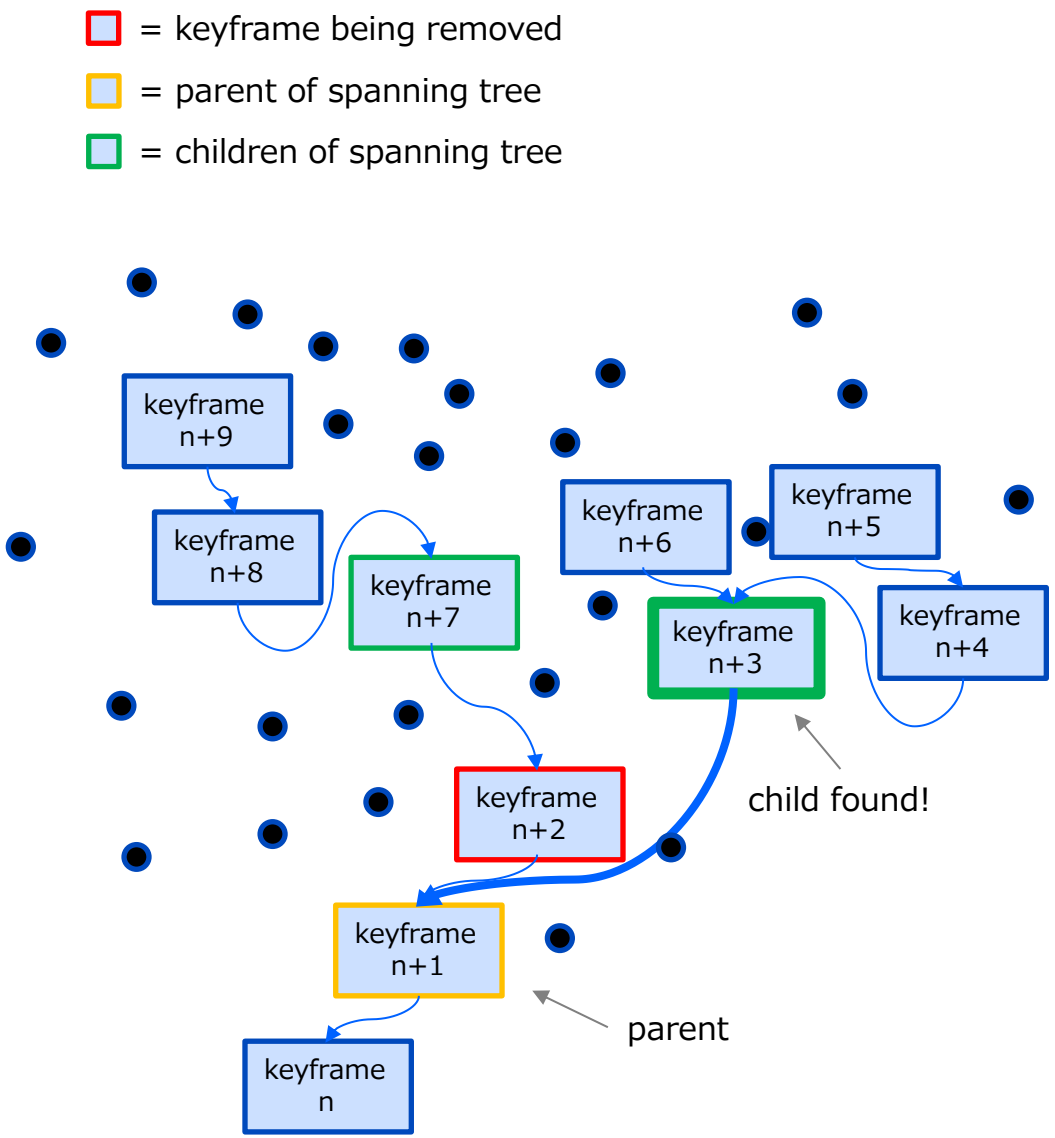
1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)



local_map_cleaner::remove_redundant_keyframes – 6 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 4/8)

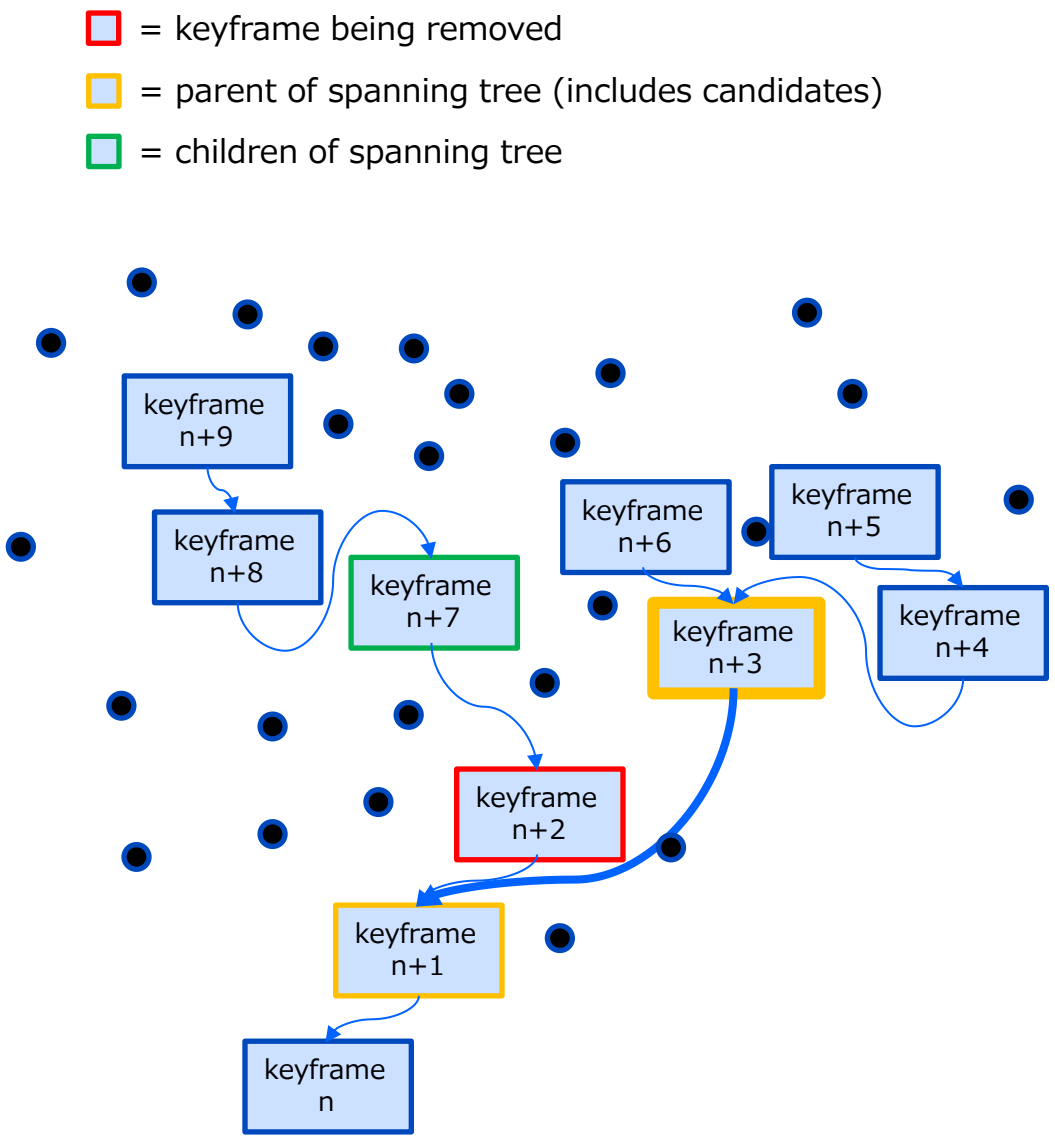
1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 1) If found, change spanning tree parent of the found child



local_map_cleaner::remove_redundant_keyframes – 7 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 5/8)

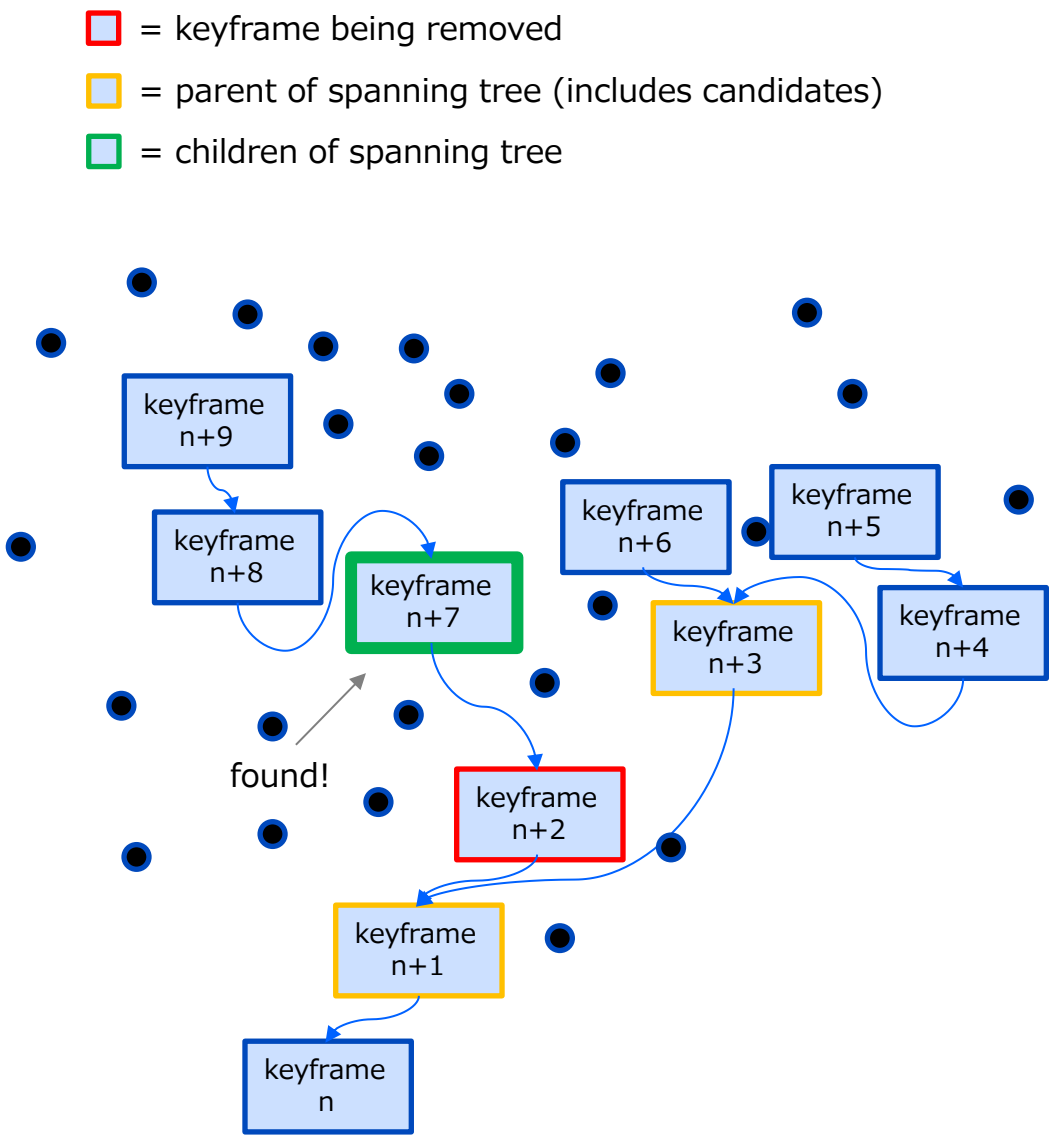
1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 1) If found, change spanning tree parent of the found child and then regard it as parent candidate



local_map_cleaner::remove_redundant_keyframes – 8 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 6/8)

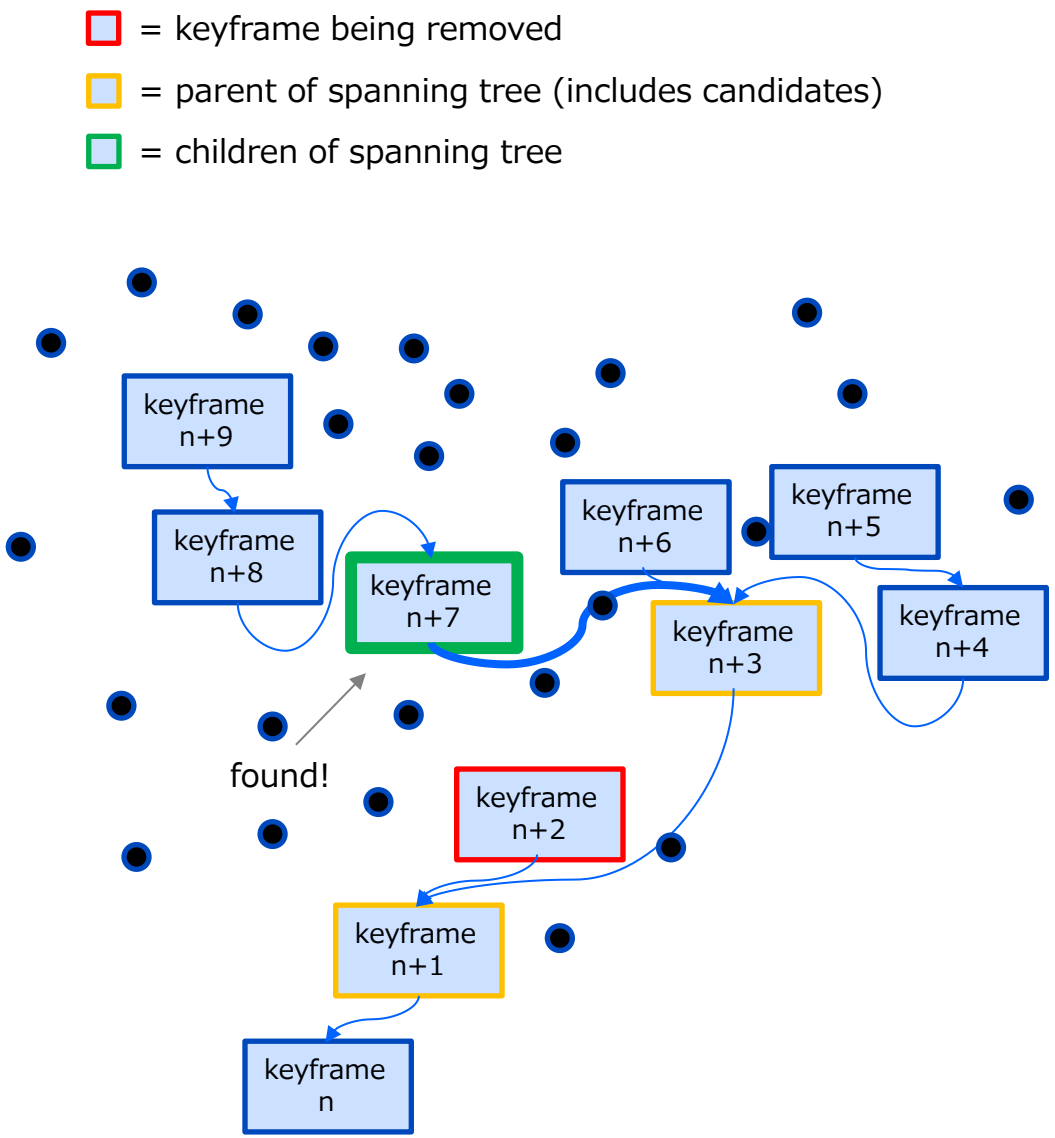
1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 1) If found, change spanning tree parent of the found child and then regard it as parent candidate
4. Go back to 2 and find a keyframe which has the most common landmarks with the spanning tree parent or the parent candidate



local_map_cleaner::remove_redundant_keyframes – 9 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 7/8)

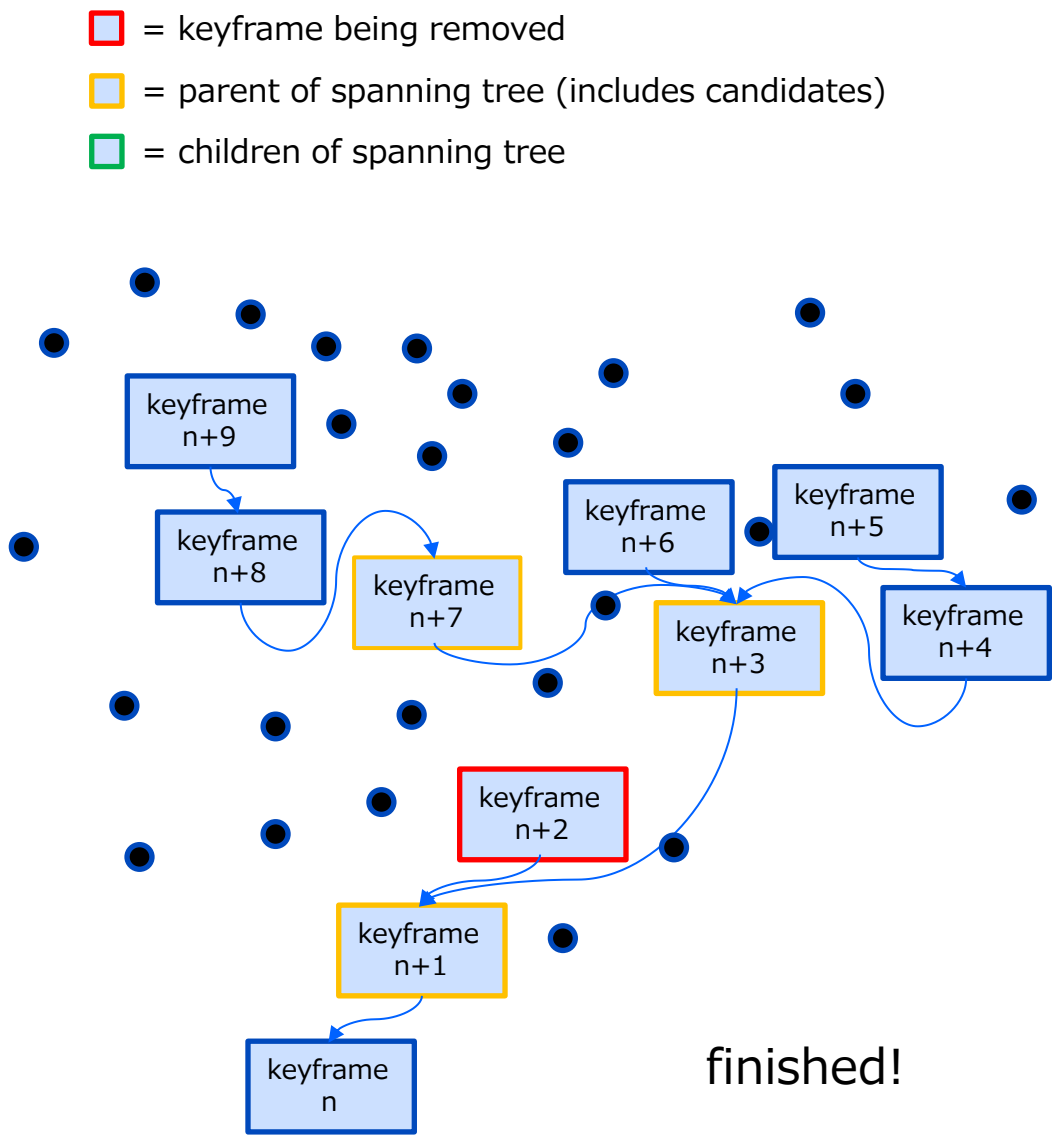
1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 1) If found, change spanning tree parent of the found child and then regard it as parent candidate
4. Go back to 2 and find a keyframe which has the most common landmarks with the spanning tree parent or the parent candidate
   - Among all parent and parent candidates, the keyframe which has the most common landmarks with the child becomes its spanning tree parent



local_map_cleaner::remove_redundant_keyframes – 10 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 7/8)

1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 1) If found, change spanning tree parent of the found child and then regard it as parent candidate
4. Go back to 2 and find a keyframe which has the most common landmarks with the spanning tree parent or the parent candidate
   - Among all parent and parent candidates, the keyframe which has the most common landmarks with the child becomes its spanning tree parent
5. Do this for all spanning children

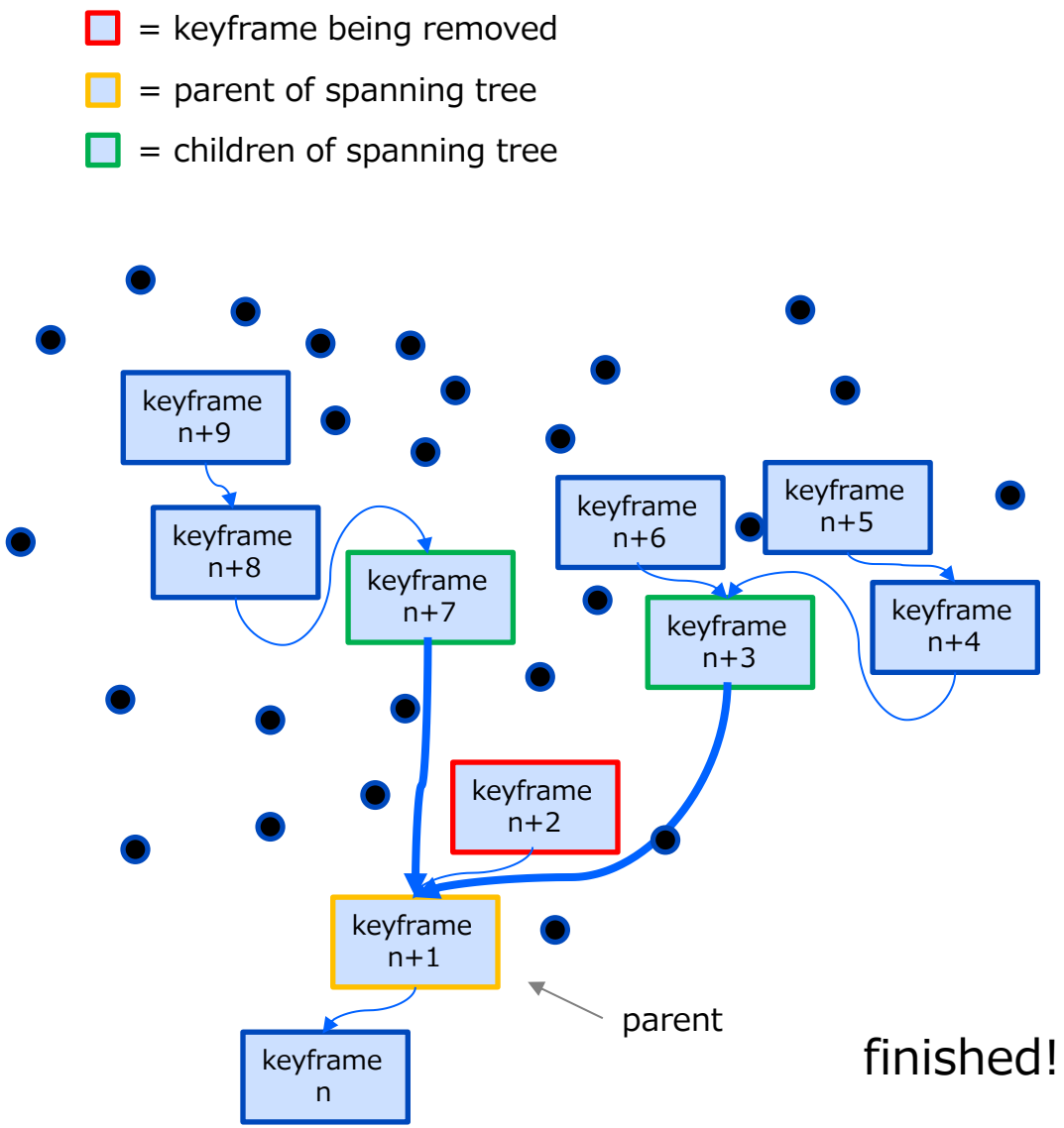


local_map_cleaner::remove_redundant_keyframes – 11 of 12

keyframe::prepare_for_erasing
(graph_node::recover_spanning_connections – 8/8)

1. As for the keyframe being removed, erase_observation from all landmarks
2. Among all spanning tree children (□), find the keyframe which has the most common landmarks with the spanning tree parent (□)
3. 2) If not found, the parent become their spanning tree parent (then skip 4 and 5)

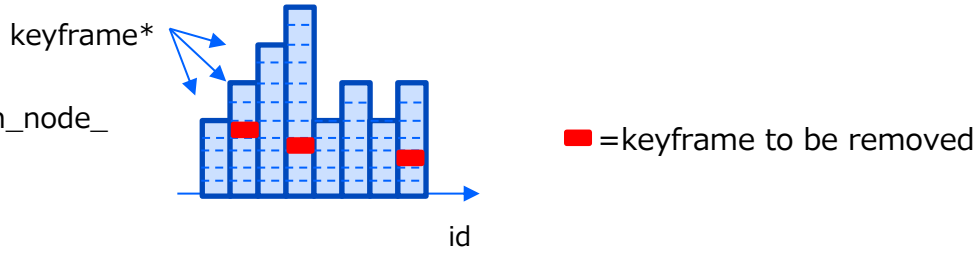
No common landmark was found among the children and the parent in this case.



finished!

- keyframe::prepare_for_erasing (remaining)
  - Call frame_statistics::replace_reference_keyframe
    - Update frm_ids_of_ref_keyfrms_, ref_keyfrms_ and rel_cam_poses_from_ref_keyfrms_ by replacing the removed keyframe with its spanning tree parent
  - Call map_database::erase_keyframe
    - Just erase from keyframes unordered_map
  - Call bow_database::erase_keyframe
    - Update keyfrms_in_node_ by removing the keyframe

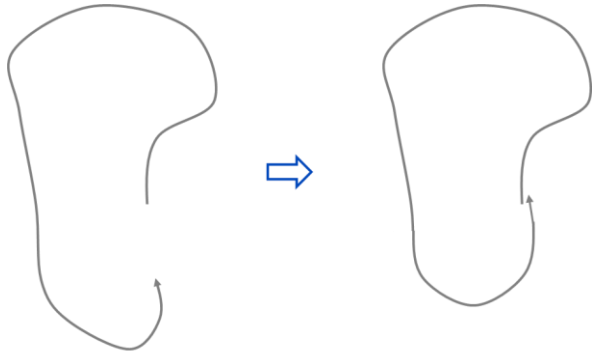
unordered_map<unsigned int, list<keyframe*>> keyfrms_in_node_

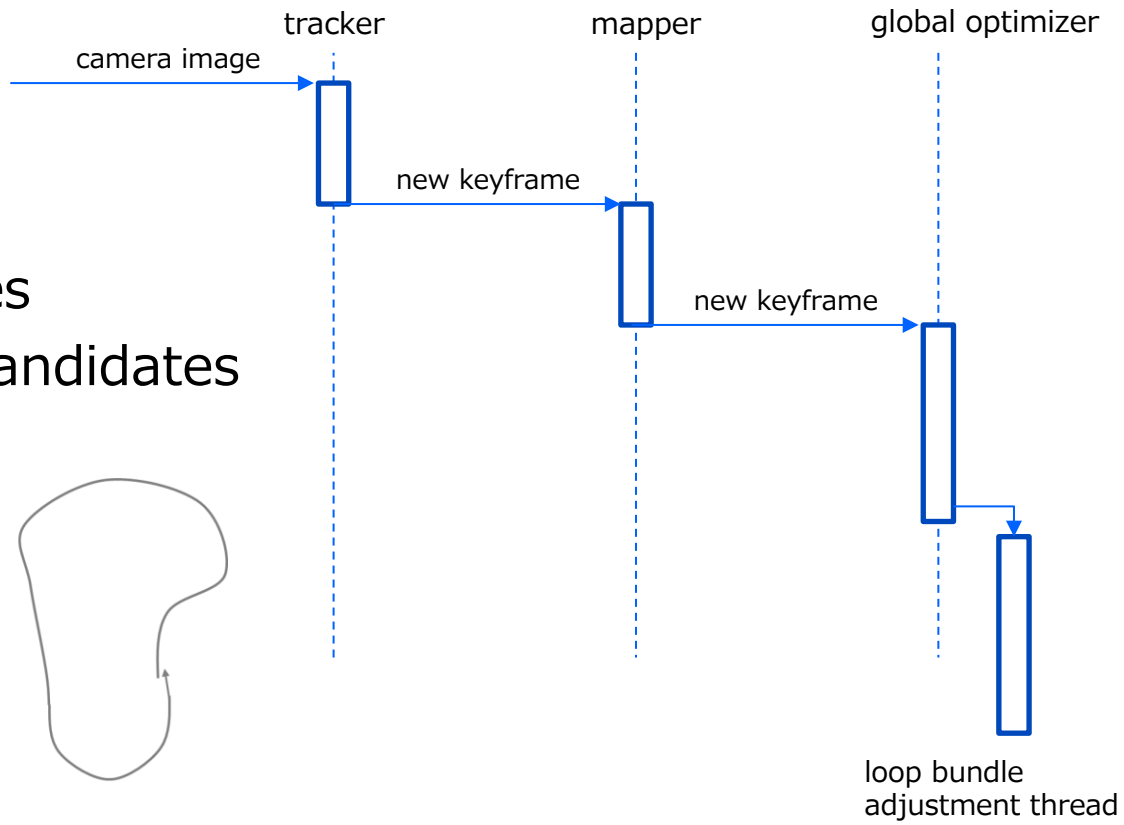


GLOBAL OPTIMIZATION

Global optimization module thread

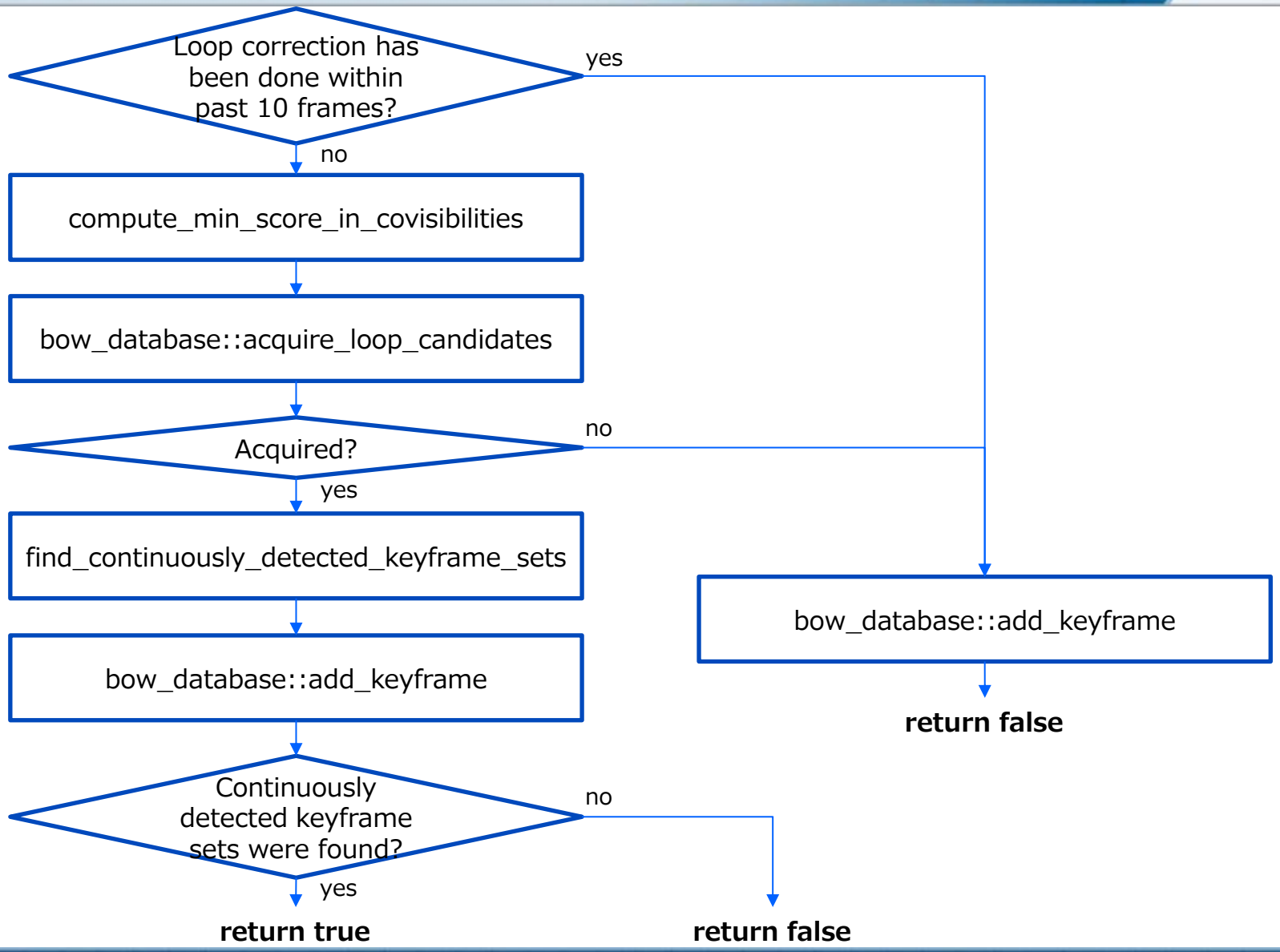
- Global optimization module thread is created at initialization
- It is waiting for some messages:
  - Terminate
  - Pause
  - Reset
  - Keyframe insertion
    - Call loop_detector::detect_loop_candidates
    - If detected, call loop_detector::validate_candidates
    - If passed, call correct_loop





GLOBAL_OPTIMIZATION – LOOP_DETECTOR::DETECT_LOOP_CANDIDATES

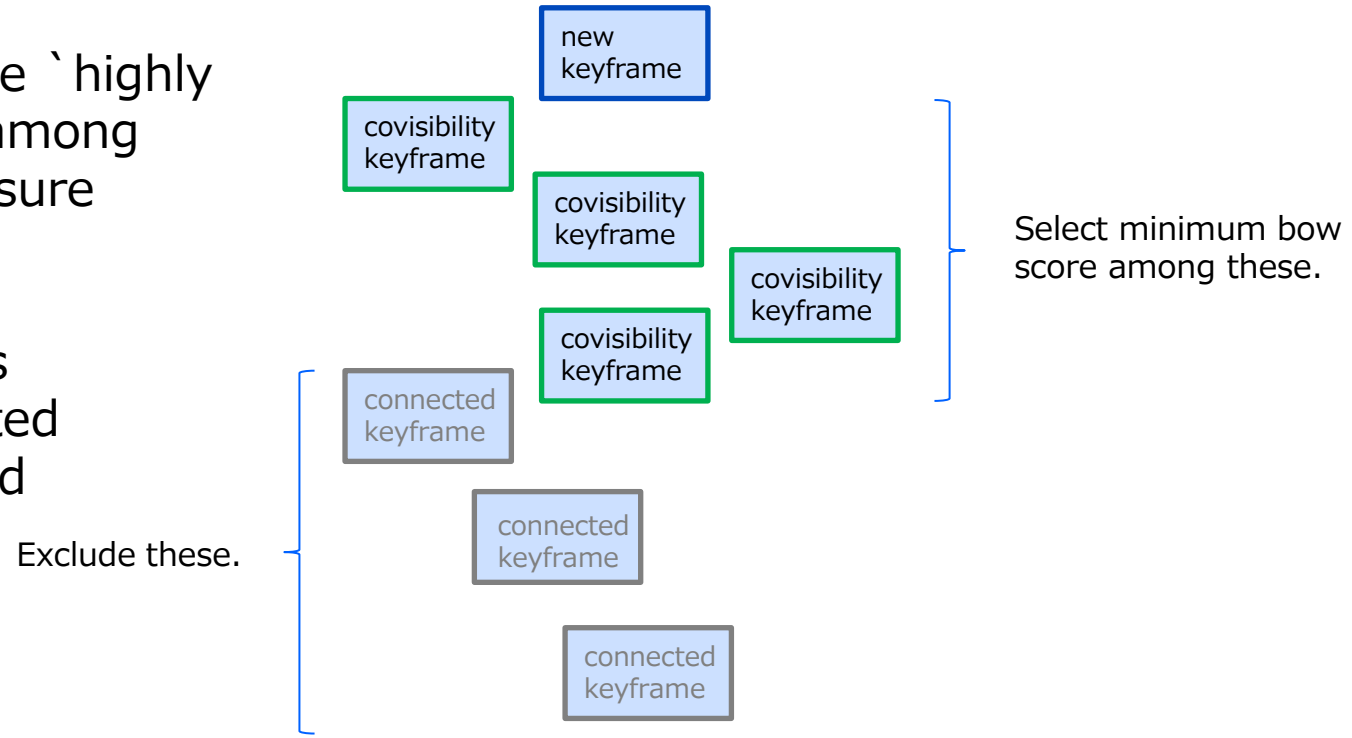
loop_detector::detect_loop_candidates – 1 of 2



loop_detector::detect_loop_candidates – 2 of 2

compute_min_score_in_covisibilities

- Compute bow scores between the new keyframe and its covisibility keyframes using DBoW2/FBoW
  - connected keyframes are not included
  - the scores represent a degree of resemblance of the images of the two keyframes
  - It is expected that the scores indicate `highly resemble` and the minimum score among them is a criterion to detect loop closure
- Among the scores, select the minimum
- The score will be used in bow_database::acquire_loop_candidates
- About covisibility keyframes and connected keyframes, see Covisibility keyframes and Connected keyframes



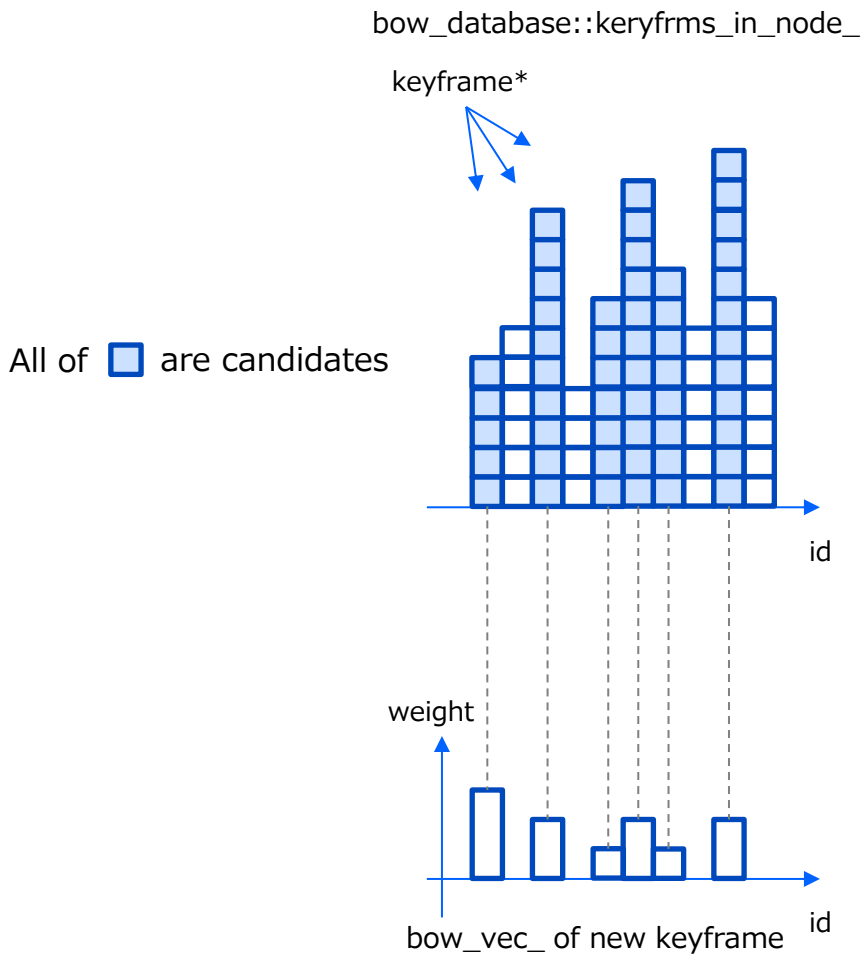
bow_database::acquire_loop_candidates – 1 of 6

- Call set_candidates_sharing_words to find keyframes from bow_database which share any visual word with the new keyframe
- Call compute_scores to reject keyframes which have lower common visual words than 80% of the maximum common visual words
  - The number of common visual words are also obtained by set_candidates_sharing_words
- Call align_scores_and_keyframes to reject keyframes which have lower common visual words than minimum score obtained by compute_min_score_in_covisibilities explained in the previous slide
- Call align_total_scores_and_keyframes to compute scores considering the similarity between bow of the new keyframe and those of candidate keyframes together with their covisibility keyframes
- Reject more candidate keyframes considering a score returned by align_total_scores_and_keyframes

bow_database::acquire_loop_candidates – 2 of 6

- **Call set_candidates_sharing_words to find keyframes from bow_database which share any visual word with the new keyframe**
- Call compute_scores to reject keyframes which have lo
- C
- C
- R

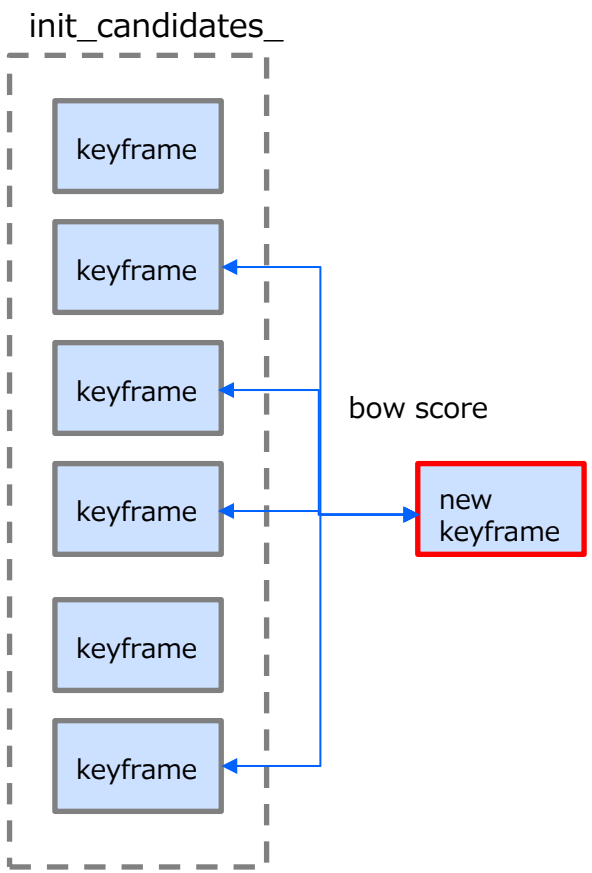
  - Obtain keyframes from bow_database::keyfrms_in_node_ which have the same bow id as the new keyframe's one
    - Exclude keyframes which are listed in keyfrms_to_reject
    - In keyfrms_to_reject there exist connected keyframes of the new keyframe which are expected to be bow similar
    - Note that we are searching bow similar keyframes except neighbor keyframes of the new keyframe
  - Obtained keyframes are copied to init_candidates_
  - Also count the number of common visual words with the new keyframe
    - Do NOT exclude keyframes which are listed in keyfrms_to_reject
    - Bow similar keyframes we are searching are expected to have common visual words to the same degree as connected keyframes
  - The number is copied to num_common_words_ together with the keyframe which will be used hereafter



bow_database::acquire_loop_candidates – 3 of 6

- Call set_candidates_sharing_words to find keyframes from bow_database which share any visual word with the new keyframe
- **Call compute_scores to reject keyframes which have lower common visual words than 80% of the maximum common visual words**
  - **The number of common visual words are also obtained by set_candidates_sharing_words**
- Call align_scores_and_keyframes to reject keyframes w
  sc
  ex
- Ca
  sc
  keyframe and those of candidate keyframes together with their covisibility keyframes
- Reject more keyframe candidates considering a score returned by align_total_scores_and_keyframes

- For each keyframe in init_candidates_ if it has more common visual words than the threshold above, compute bow score between it and the new keyframe using DBoW2 or FBoW
- The score are copied to scores_ together with the keyframe instance which will be used in align_scores_and_keyframes explained in the next slide



bow_database::acquire_loop_candidates – 4 of 6

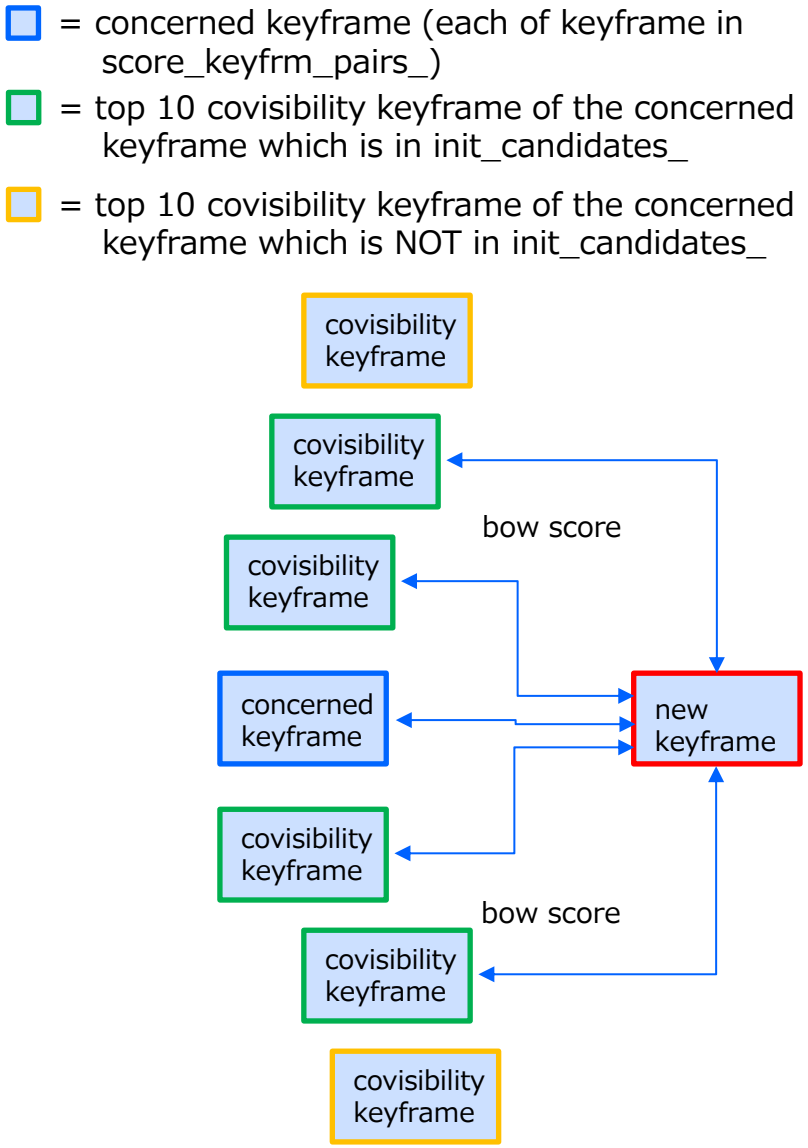
- Call set_candidates_sharing_words to find keyframes from bow_database which share any visual word with the new keyframe
- Call compute_scores to reject keyframes which have lower common visual words than 80% of the maximum common visual words
  - The number of common visual words are also obtained by set_candidates_sharing_words
- **Call align_scores_and_keyframes to reject keyframes which have lower common visual words than minimum score obtained by compute_min_score_in_covisibilities explained in the previous slide**
- Ca
  co
  ar
  co
- Re
  re

- For each keyframe in init_candidates_:
  - If it has more common visual words than 80% of the maximum common bow
  - And if its bow score is higher than the minimum score obtained by compute_min_score_in_covisibilities
  - Then it is copied to score_keyfrm_pairs_ together with its score which will be used in align_total_scores_and_keyframes explained in the next slide

bow_database::acquire_loop_candidates – 5 of 6

- Call set_candidates_sharing_words to find keyframes from bow_database which share any visual word with th
- Ca
  lo
  co
- Ca
  w
  sc
  explained in the previous slide

  - For each keyframe in score_keyfrm_pairs_
    - set bow score of the keyframe to total_score
    - For each top 10 covibility keyframe of the keyframe
      - If it is in init_candidate_ and has more common visual words than 80% of the maximum, sum up bow score to total_score
    - total_score is copied to total_score_keyfrm_pairs_ together with the best keyframe whose bow score is highest among them
    - total_score_keyfrm_pairs will be used in the next slide
  - The best total score among all is returned, which will be used in the next slide
- **Call align_total_scores_and_keyframes to compute scores considering the similarity between bow of the keyframe and those of candidate keyframes together with their covisibility keyframes**
- Reject more keyframe candidates considering a score returned by align_total_scores_and_keyframes

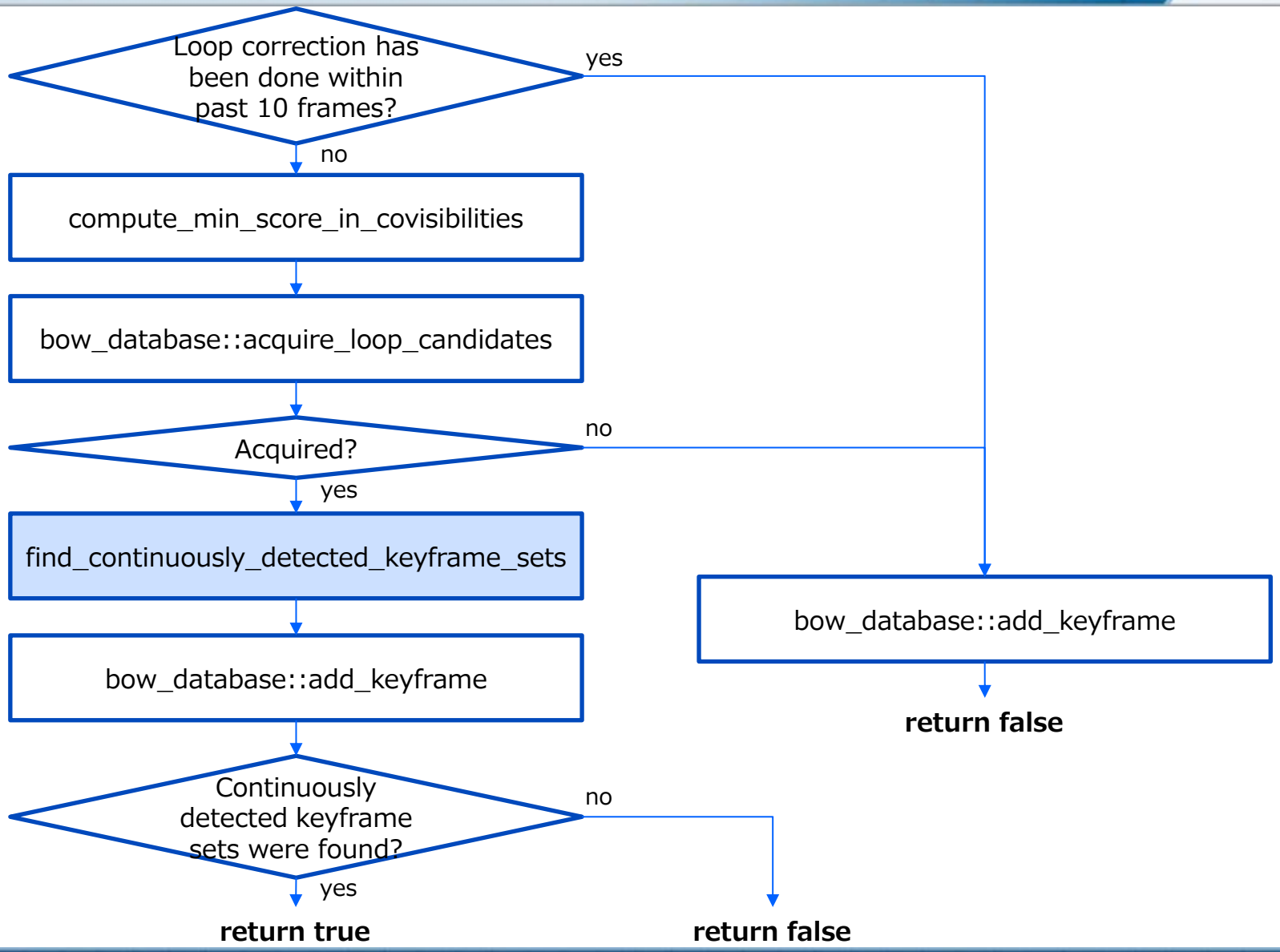


bow_database::acquire_loop_candidates – 6 of 6

- Call set_candidates_sharing_words to find keyframes from bow_database which share any bow with the new keyframe
- Call compute_scores to reject keyframes which have lower common visual words than 80% of the maximum common visual words
  - The number of common visual words are also obtained by set_candidates_sharing_words
- Call align_scores_and_keyframes to reject keyframes which have lower common visual words than minimum score obtained by compute_min_score_in_covisibilities
- Ca
  considering the similarity between bow of the keyframe and those of candidate keyframes together with their covisibility keyframes
- **Reject more keyframe candidates considering a score returned by align_total_scores_and_keyframes**

- For each keyframe in total_score_keyfrm_pairs_:
  - If total score is below 75% of the best total score, it is rejected
  - Else it becomes one of final candidates

Where we are

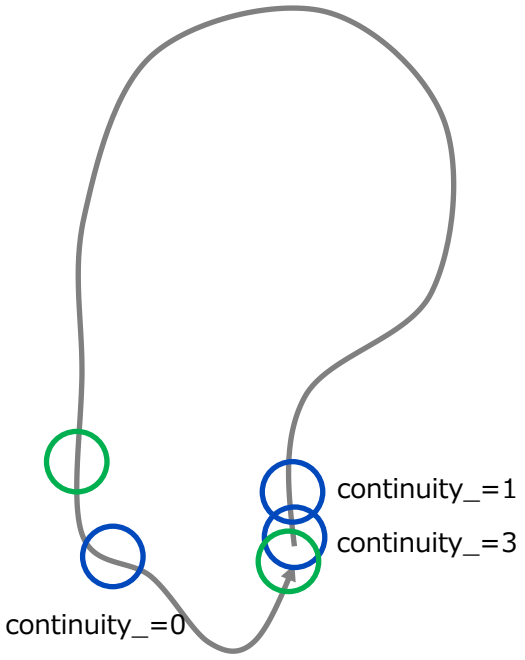

Loop correction has been done within past 10 frames?
yes
no
compute_min_score_in_covisibilities
bow_database::acquire_loop_candidates
Acquired?
no
yes
find_continuously_detected_keyframe_sets
bow_database::add_keyframe
bow_database::add_keyframe
return false
Continuously detected keyframe sets were found?
no
yes
return true
return false

find_continuously_detected_keyframe_sets – 1 of 3

keyframe set is used to validate loop detection candidate keyframes more robustly.
But what is keyframe set?

keyframe set is a group of keyframes:

- It has a leader keyframe and its connected keyframes
  - When a new keyframe set is created, a keyframe in init_loop_candidates becomes the leader (init_loop_candidates = return value of bow_database::acquire_loop_candidates)
- It has a continuity count (continuity_ in source code)
  - When a new keyframe set is created, it is set to 0
  - It is incremented the next time a new keyframe is inserted if any keyframe in the keyframe set is the same as a keyframe in init_loop_candidates_ or one of its connected keyframes
  - Else the keyframe set is not kept
- Current keyframe sets are copied to curr_cont_detected_keyfrm_sets and then to cont_detected_keyfrm_sets_



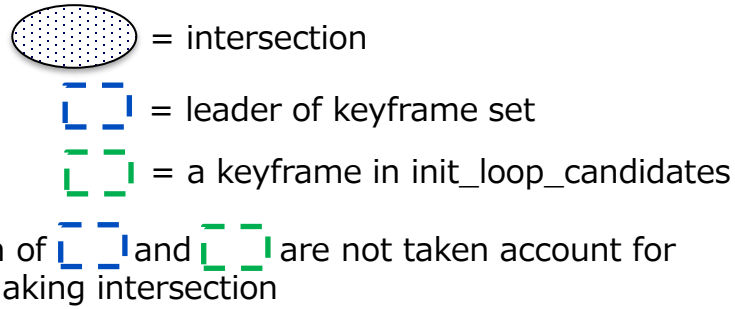
= camera trajectory

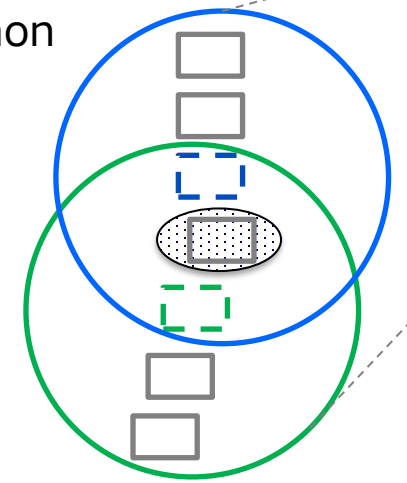
= keyframe set at the previous keyframe insertion

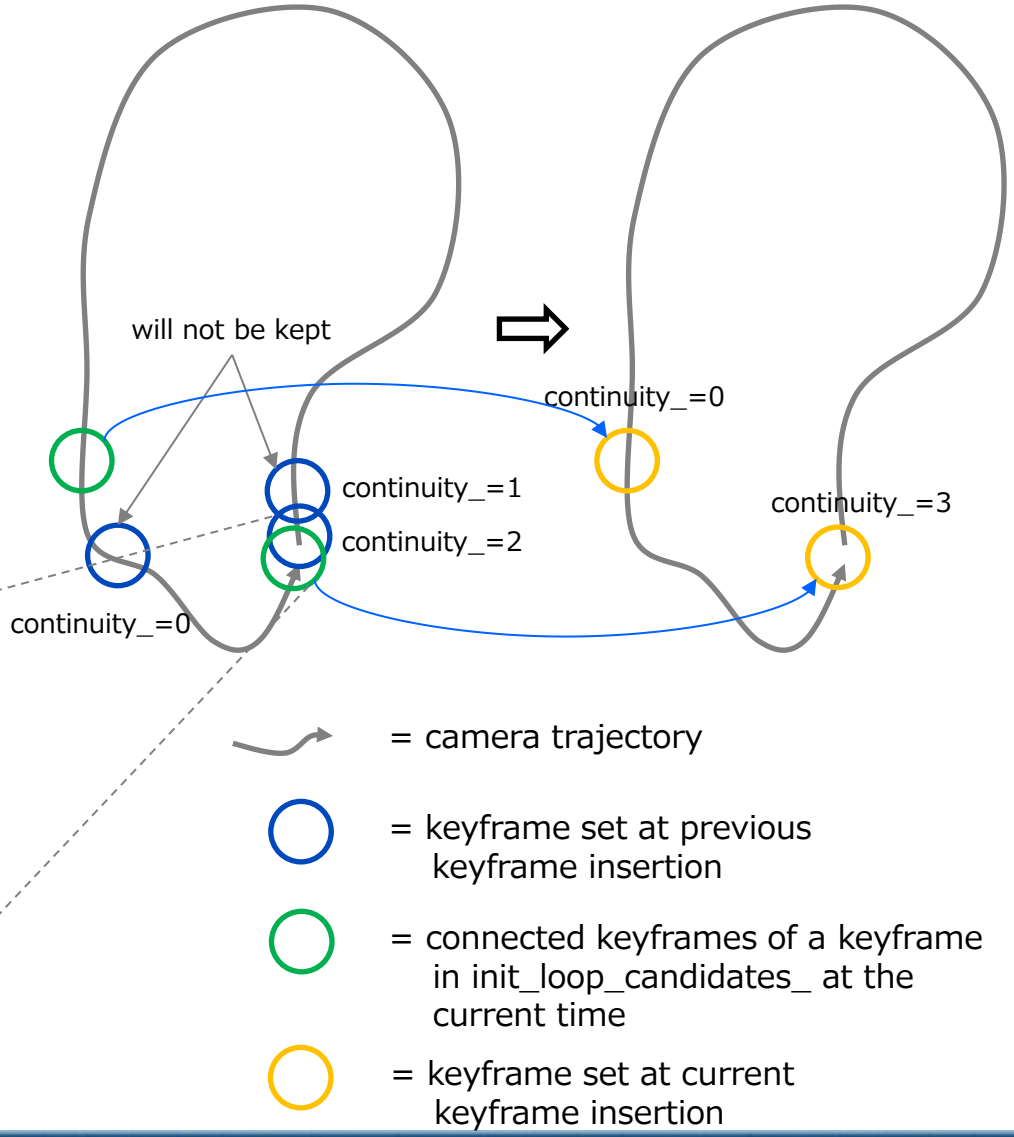
= a keyframe in init_loop_candidates_ and its connected keyframes at the current time

find_continuously_detected_keyframe_sets – 2 of 3

- For each keyframe in init_loop_candidates_:
  - Obtain its connected keyframes
  - For each keyframe set which exists at this time:
    - Find keyframes both in the connected keyframes and the keyframe set (using intersection_is_empty)
    - If found, keyframe set is reborn with the keyframe in init_loop_candidates_ and its connected keyframes, and its continuity count is incremented. Then copy the keyframe set to curr_cont_detected_keyfrm_sets.
  - If none of the keyframe sets does not have the same keyframe with the connected keyframes, the keyframe in init_loop_candidates_ and its connected keyframes becomes a new keyframe set
- Existing keyframe sets which have no common keyframe does not keep remained







find_continuously_detected_keyframe_sets – 3 of 3

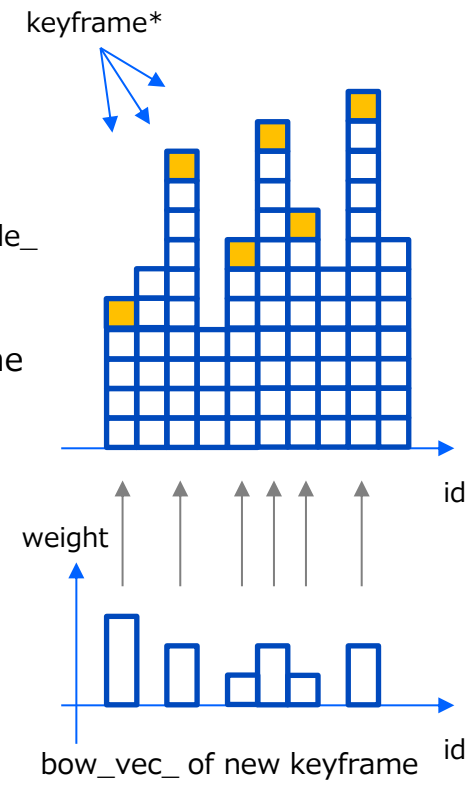
- After calling find_continuously_detected_keyframe_sets, the continuity count is checked
- If the continuity count of a keyframe set is above or equal to min_continuity_ (=3), the leader keyframe of the keyframe set is copied to loop_candidates_to_validate_
- Keyframes in loop_candidates_to_validate_ will be validated more strictly in loop_detector::validate_candidates hereafter

bow_database::add_keyframe

- Add current keyframe to each list of the bow id to which the current keyframe's bow corresponds

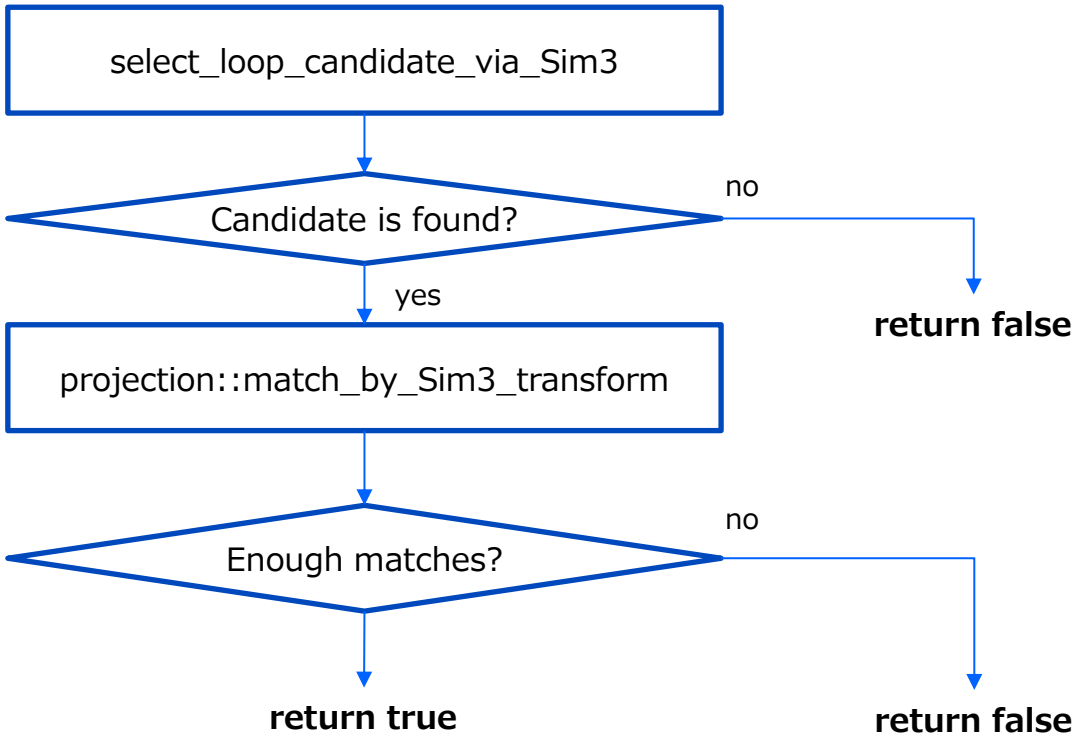
unordered_map<unsigned int, list<keyframe*>> keryfrms_in_node_

= new keyframe

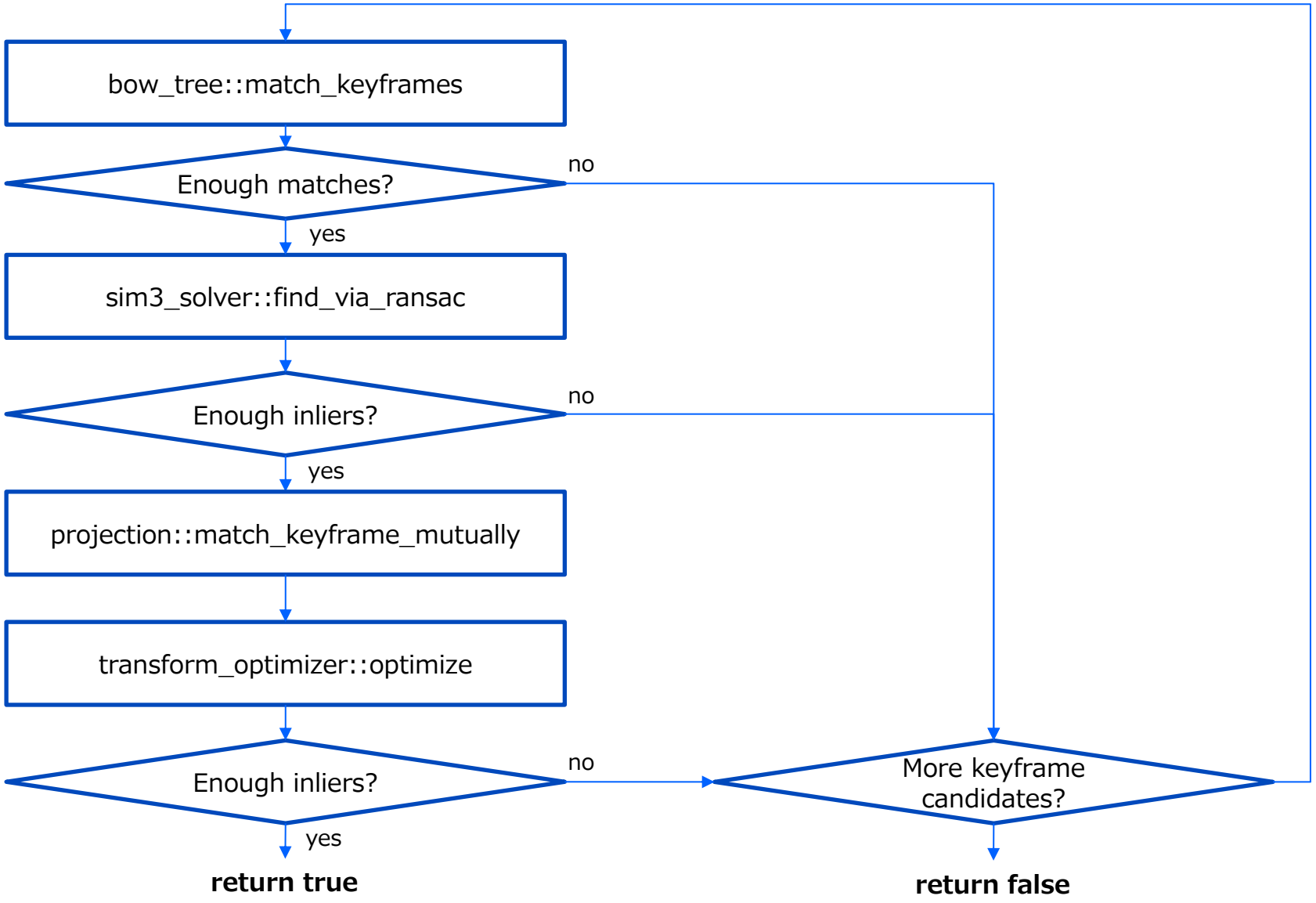


GLOBAL_OPTIMIZATION – LOOP_DETECTOR::VALIDATE_CANDIDATES

loop_detector::validate_candidates



select_loop_candidate_via_Sim3 – 1 of 23



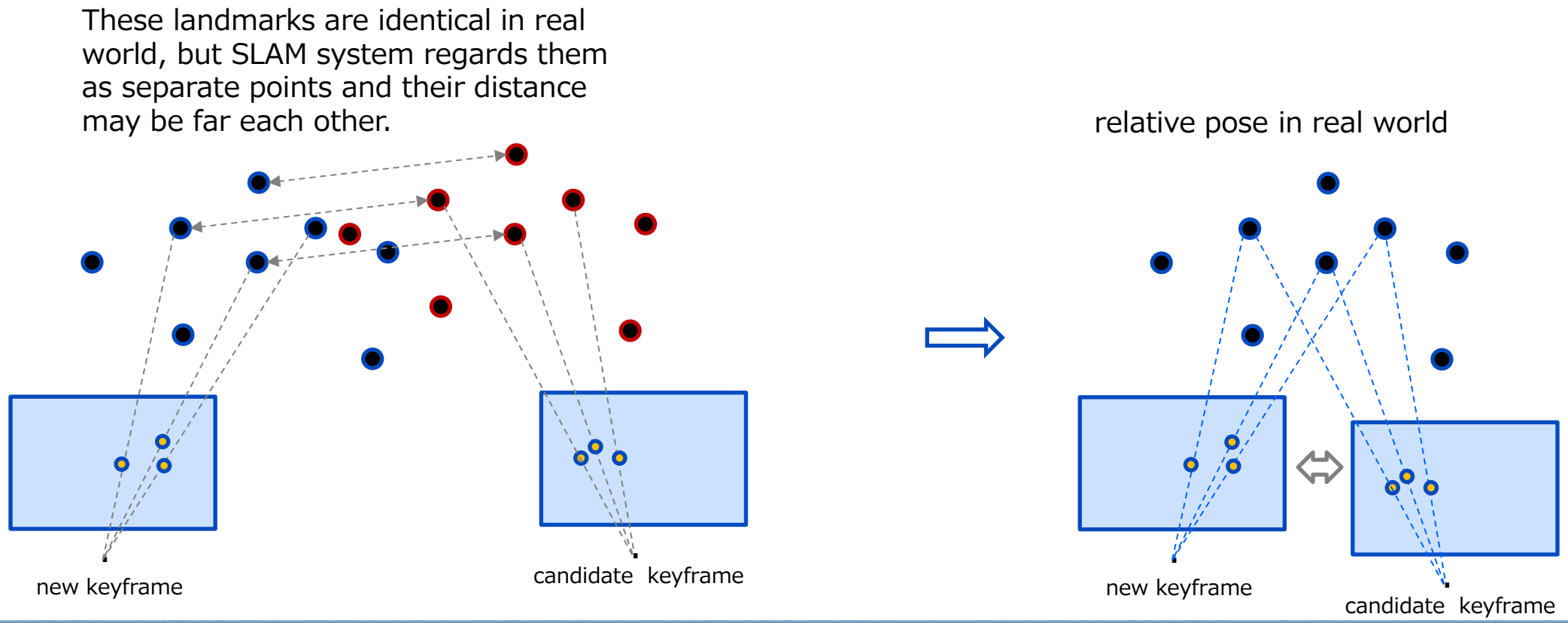
select_loop_candidate_via_Sim3 – 2 of 23

- Terminology
  - SE3 = 3D rigid transformations
    - 6 Degree of Freedom (DoF) in SE3
  - Sim3 = 3D similarity transformations
    = 3D rigid transformations + scale
    - 7 DoF in Sim3

select_loop_candidate_via_Sim3 – 3 of 23

Current problem is as below:

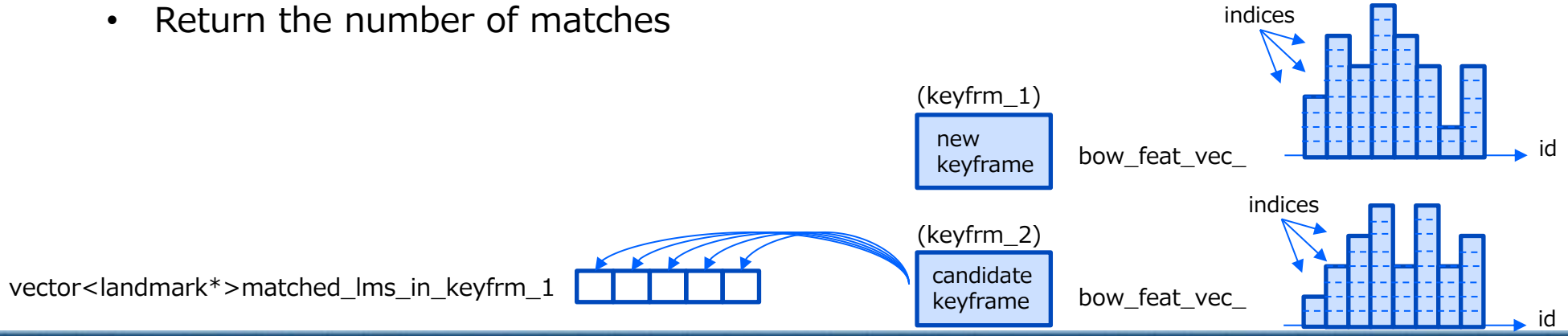
- Need to obtain relative pose in real world between the new keyframe and the candidate keyframe considering landmarks' correspondences which are identical in real world



select_loop_candidate_via_Sim3 – 4 of 23

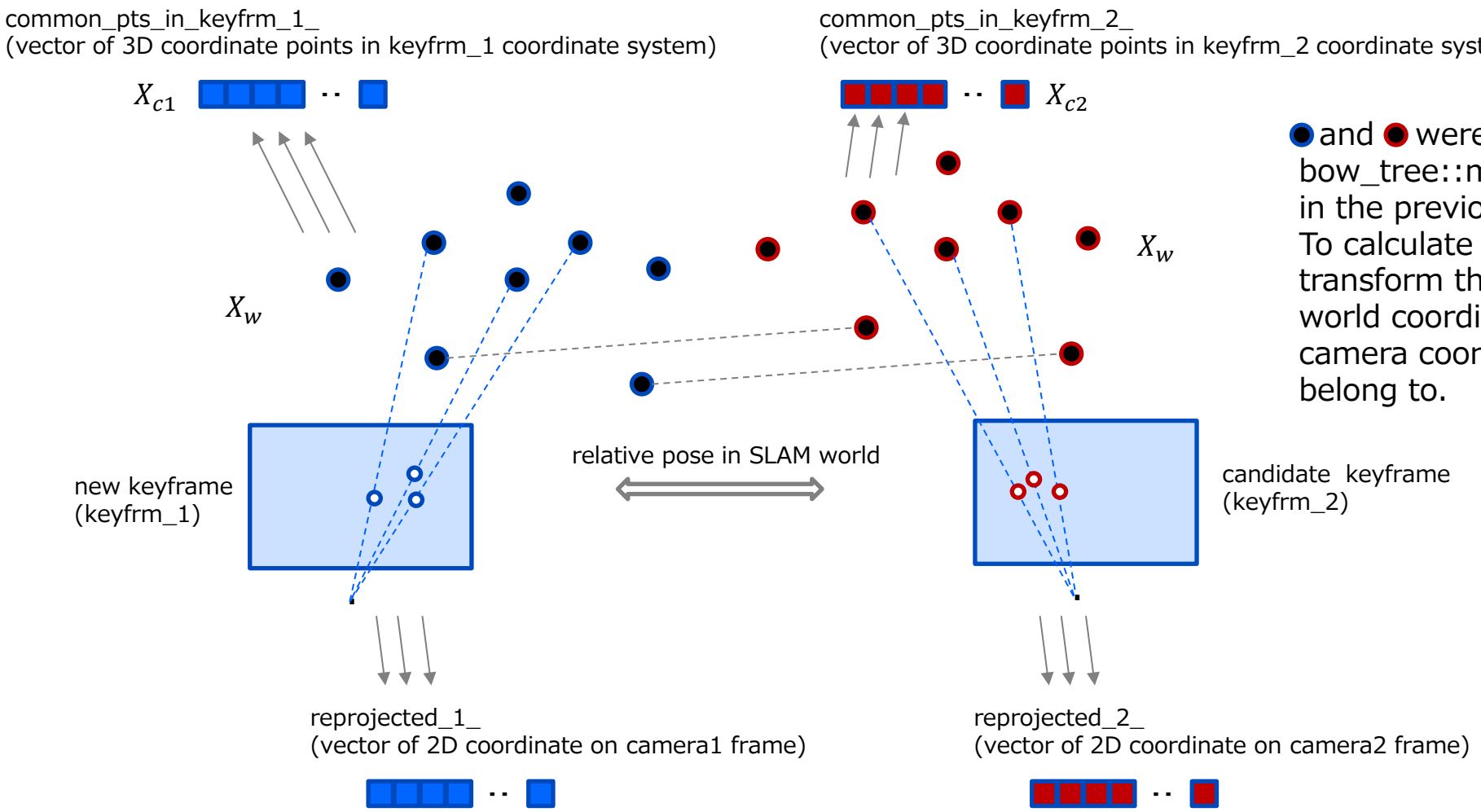
bow_tree::match_keyframes (very similar to bow_tree::match_frame_and_keyframe - bow_match_based_track – 3 of 4)

- Use bow_feat_vec_ which has ORB descriptor indices per node id
- For each descriptor index of each node id of the new keyframe, find the best matched descriptor index from corresponding node id of the candidate keyframe using Hamming distance
  - Hamming distance must be the smallest and below HAMMING_DIST_THR_LOW and lower than 0.75 * the second best
- Copy the landmark of the candidate keyframe corresponding to descriptor index for which the best match with the new keyframe was found to matched_lms_in_keyfrm_1
  - Note that landmarks in matched_lms_in_keyfrm_1 are of the candidate keyframe and not of the new keyframe
  - They will be used hereafter for pose alignment
- At last, do angle check explained in projection::match_current_and_last_frames – 6 of 7
- Return the number of matches



select_loop_candidate_via_Sim3 – 5 of 23

sim3_solver::sim3_solver



● and ● were corresponded by bow_tree::match_keyframes in the previous slide.
To calculate relative pose, transform them from the world coordinate to the camera coordinate which they belong to.

Relative pose calculation

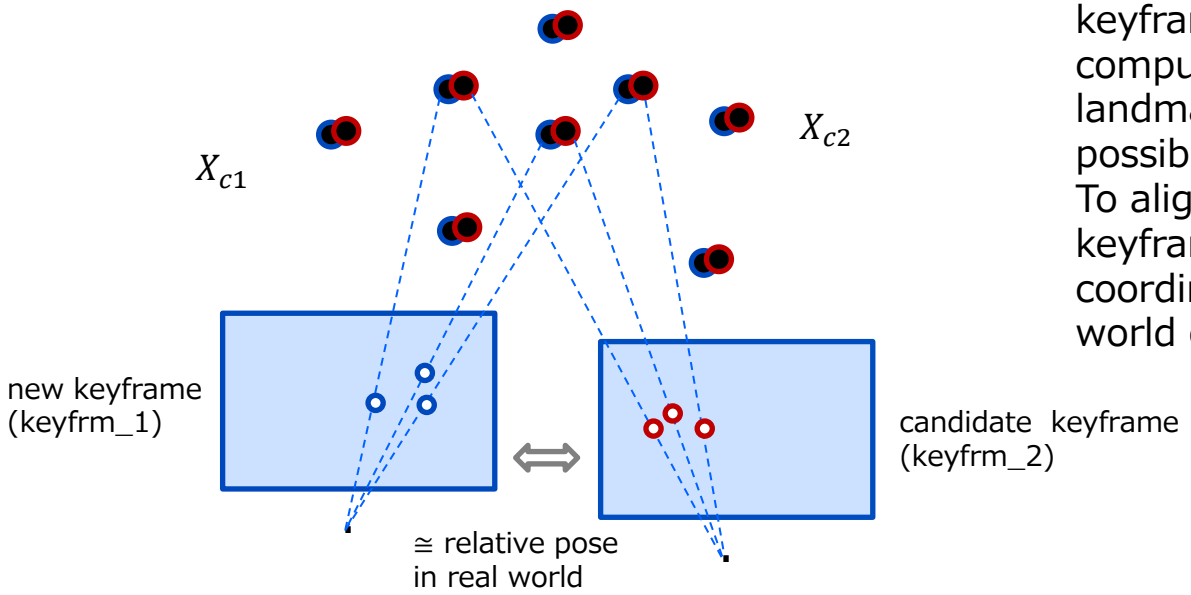
common_pts_in_keyfrm_1_
(vector of 3D coordinate points in keyfrm_1 coordinate system)

X_{c1}

common_pts_in_keyfrm_2_
(vector of 3D coordinate points in keyfrm_2 coordinate system)

X_{c2}

align by compute_Sim3



Align the relative pose between the new keyframe and the candidate by compute_Sim3 for corresponded landmarks to be as identical points as possible.
To align the relative pose, we need to use keyframe coordinates (camera coordinates) for landmarks instead of the world coordinate.

reprojected_1_
(vector of 2D coordinate on camera1 frame)

reprojected_2_
(vector of 2D coordinate on camera2 frame)

select_loop_candidate_via_Sim3 – 7 of 23

sim3_solver::find_via_ransac

Overview:

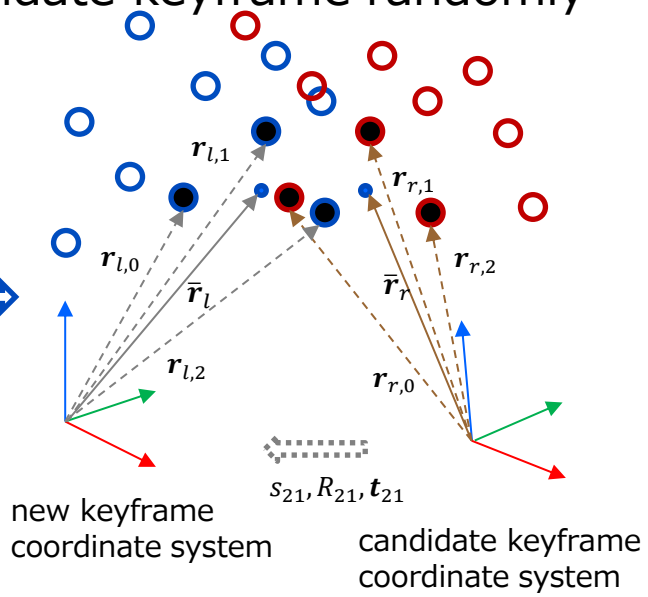
- Calculate relative pose between the new keyframe and the candidate keyframe using:
  - Landmarks of the candidate keyframe found by bow_tree::match_keyframes in the previous slide (in candidate keyframe coordinate)
  - Landmarks of the new keyframe corresponding to those of the candidate keyframe (in new keyframe coordinate)
- The relative pose is calculated by the manner explained in Absolute_Orientation.pdf (mit.edu)
  - See 2.B to 2.E and 4.A to 4.B in the paper
- The calculated pose is tested via RANSAC with 200 iterations
- The pose which achieves the best score is chosen
- Determine succeed or not depending on the number of inliers with the best pose

select_loop_candidate_via_Sim3 – 8 of 23

sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- Calculate relative pose by calling compute_Sim3:
- Count inliers using the calculated relative pose
  - Find the best relative pose during 200 iterations

Select three landmark pairs randomly



Compute relative pose

Count inliers among all landmarks except the three

Find the best relative pose which has the most inliers

select_loop_candidate_via_Sim3 – 9 of 23

sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- **Calculate relative pose by calling compute_Sim3 – 1 of 3**
- Count inliers using the calculated relative pose
  - Find the best relative pose during 200 iterations

$$\bar{\boldsymbol{r}}_l = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{r}_{l,i}, \bar{\boldsymbol{r}}_r = \frac{1}{n}\sum_{i=1}^{n} \boldsymbol{r}_{r,i}$$

$$\boldsymbol{r}'_{l,i} = \boldsymbol{r}_{l,i} - \bar{\boldsymbol{r}}_l, \boldsymbol{r}'_{r,i} = \boldsymbol{r}_{r,i} - \bar{\boldsymbol{r}}_r$$

R_{21} is sought by maximizing $\sum_{i=1}^{n} \boldsymbol{r}'_{r,i} \cdot R_{21}\boldsymbol{r}'_{l,i}$

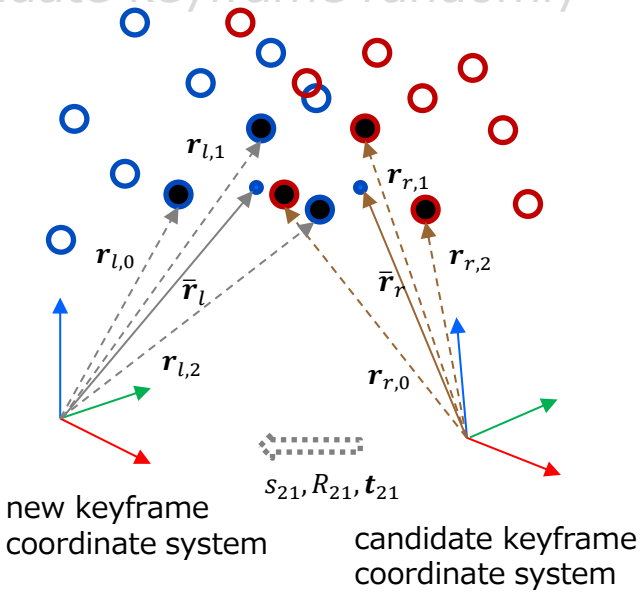
this is derived by minimizing the sum of squared errors between landmarks of the new keyframe and those of the candidate keyframe

$\sum_{i=1}^{n} \boldsymbol{r}'_{r,i} \cdot R_{21}\boldsymbol{r}'_{l,i}$ is represented by using quaternion as follows:

$$\sum_{i=1}^{n} \dot{\boldsymbol{r}}'_{r,i} \cdot (\dot{\boldsymbol{q}}\dot{\boldsymbol{r}}'_{l,i}\dot{\boldsymbol{q}}^*) \text{ where } \dot{\boldsymbol{r}}'_{s,i} = 0 + ix'_{s,i} + jy'_{s,i} + kz'_{s,i},$$

$$\dot{\boldsymbol{q}} = q_0 + iq_1 + jq_2 + kq_3 \; representing \; rotation, \dot{\boldsymbol{q}}^* = q_0 - iq_1 - jq_2 - kq_3$$

$$\sum_{i=1}^{n} \dot{\boldsymbol{r}}'_{r,i} \cdot (\dot{\boldsymbol{q}}\dot{\boldsymbol{r}}'_{l,i}\dot{\boldsymbol{q}}^*) = \sum_{i=1}^{n} (\dot{\boldsymbol{r}}'_{r,i}\dot{\boldsymbol{q}}) \cdot (\dot{\boldsymbol{q}}\dot{\boldsymbol{r}}'_{l,i}) = \sum_{i=1}^{n} (\dot{\boldsymbol{q}}\dot{\boldsymbol{r}}'_{l,i}) \cdot (\dot{\boldsymbol{r}}'_{r,i}\dot{\boldsymbol{q}}) \; (to \; be \; continued)$$



⇩

Compute relative pose

select_loop_candidate_via_Sim3 – 10 of 23

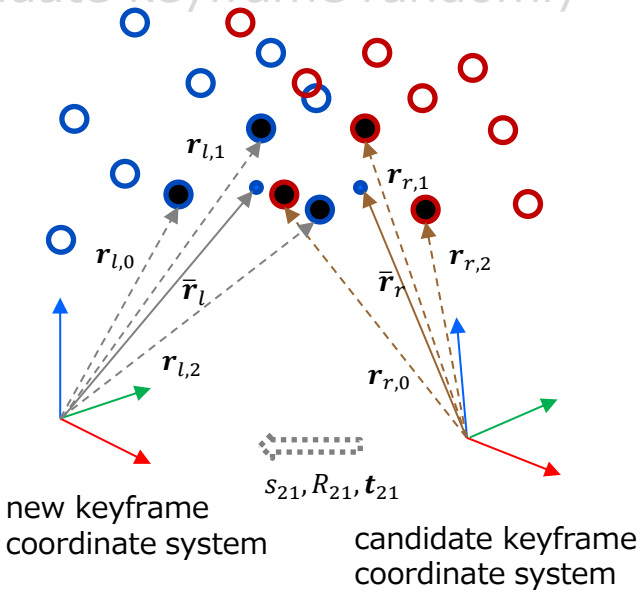
sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- **Calculate relative pose by calling compute_Sim3 – 2 of 3**
- Count inliers using the calculated relative pose
  - Find the best relative pose during 200 iterations

$$\sum_{i=1}^{n}(\dot{\boldsymbol{q}}\dot{\boldsymbol{r}}'_{l,i})\cdot(\dot{\boldsymbol{r}}'_{r,i}\dot{\boldsymbol{q}}) = \sum_{i=1}^{n}\begin{pmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & z'_{r,i} & -y'_{r,i} \\ y'_{r,i} & -z'_{r,i} & 0 & x'_{r,i} \\ z'_{r,i} & y'_{r,i} & -x'_{r,i} & 0 \end{pmatrix}\dot{\boldsymbol{q}}\cdot\begin{pmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & -z'_{r,i} & y'_{r,i} \\ y'_{r,i} & z'_{r,i} & 0 & -x'_{r,i} \\ z'_{r,i} & -y'_{r,i} & x'_{r,i} & 0 \end{pmatrix}\dot{\boldsymbol{q}}$$

$$= \sum_{i=1}^{n}\dot{\boldsymbol{q}}^{T}\begin{pmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & z'_{r,i} & -y'_{r,i} \\ y'_{r,i} & -z'_{r,i} & 0 & x'_{r,i} \\ z'_{r,i} & y'_{r,i} & -x'_{r,i} & 0 \end{pmatrix}^{T}\begin{pmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & -z'_{r,i} & y'_{r,i} \\ y'_{r,i} & z'_{r,i} & 0 & -x'_{r,i} \\ z'_{r,i} & -y'_{r,i} & x'_{r,i} & 0 \end{pmatrix}\dot{\boldsymbol{q}}$$

$$= \dot{\boldsymbol{q}}^{T}\sum_{i=1}^{n}\begin{pmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & z'_{r,i} & -y'_{r,i} \\ y'_{r,i} & -z'_{r,i} & 0 & x'_{r,i} \\ z'_{r,i} & y'_{r,i} & -x'_{r,i} & 0 \end{pmatrix}^{T}\begin{pmatrix} 0 & -x'_{r,i} & -y'_{r,i} & -z'_{r,i} \\ x'_{r,i} & 0 & -z'_{r,i} & y'_{r,i} \\ y'_{r,i} & z'_{r,i} & 0 & -x'_{r,i} \\ z'_{r,i} & -y'_{r,i} & x'_{r,i} & 0 \end{pmatrix}\dot{\boldsymbol{q}} = \dot{\boldsymbol{q}}^{T}N\dot{\boldsymbol{q}}$$



⇩

Compute relative pose

The problem of maximizing $\sum_{i=1}^{n}\boldsymbol{r}'_{r,i}\cdot R_{21}\boldsymbol{r}'_{l,i}$ can be replaced by maximizing $\dot{\boldsymbol{q}}^{T}N\dot{\boldsymbol{q}}$

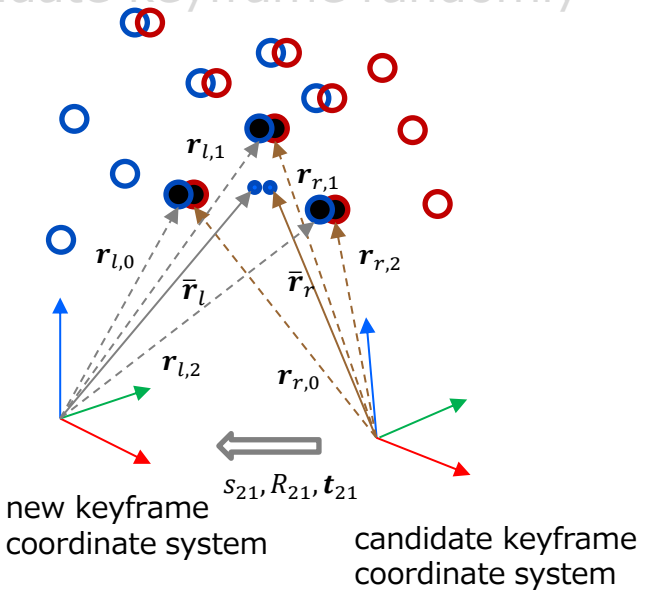
sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- **Calculate relative pose by calling compute_Sim3 – 3 of 3**
- Count inliers using the calculated relative pose
  - Find the best relative pose during 200 iterations

N is sought as follows:

Calculate $M = \begin{pmatrix} S_{xx} & S_{xy} & S_{xz} \\ S_{yx} & S_{yy} & S_{yz} \\ S_{zx} & S_{zy} & S_{zz} \end{pmatrix}$, $where\ S_{xx} = \sum_{i=1}^{n} x'_{l,i} x'_{r,i}\ , S_{xy} = \sum_{i=1}^{n} x'_{l,i} y'_{r,i}\ , and\ so\ on.$

$$N = \begin{pmatrix} (S_{xx}+S_{yy}+S_{zz}) & S_{yz}-S_{xz} & S_{zx}-S_{xz} & S_{xy}-S_{yz} \\ S_{yz}-S_{xz} & (S_{xx}-S_{yy}-S_{zz}) & S_{xy}+S_{yx} & S_{zx}+S_{xz} \\ S_{zx}-S_{xz} & S_{xy}+S_{yx} & (-S_{xx}+S_{yy}-S_{zz}) & S_{yz}+S_{zy} \\ S_{xy}-S_{yz} & S_{zx}+S_{xz} & S_{yz}+S_{zy} & (-S_{xx}-S_{yy}+S_{zz}) \end{pmatrix} by\ using\ M$$



The eigen vector of N corresponding to the maximum eigen value maximizes $\dot{\boldsymbol{q}}^T N \dot{\boldsymbol{q}}$, so it is rotation quaternion to be sought. R_{21} is sought by converting the rotation quaternion.

$\boldsymbol{t}_{21}$ is easily calculated as follows:

$$\boldsymbol{t}_{21} = \bar{\boldsymbol{r}}_r - s_{21} R_{21} \bar{\boldsymbol{r}}_l$$

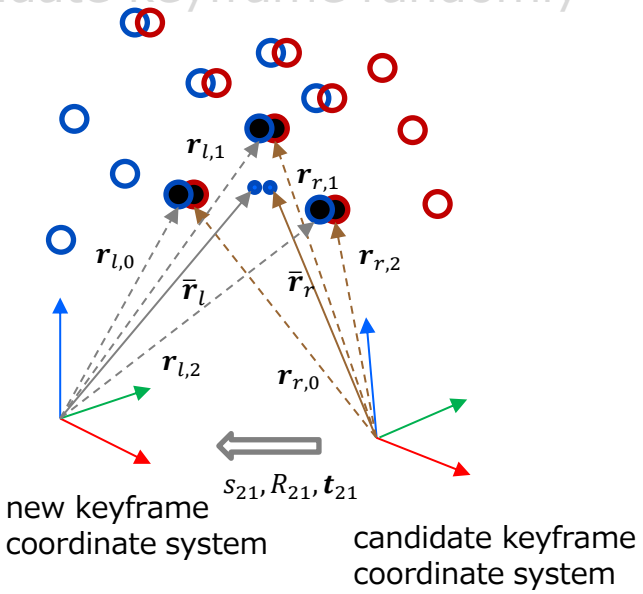
s_{21} is 1.0 in stereo version.

Compute relative pose

select_loop_candidate_via_Sim3 – 12 of 23

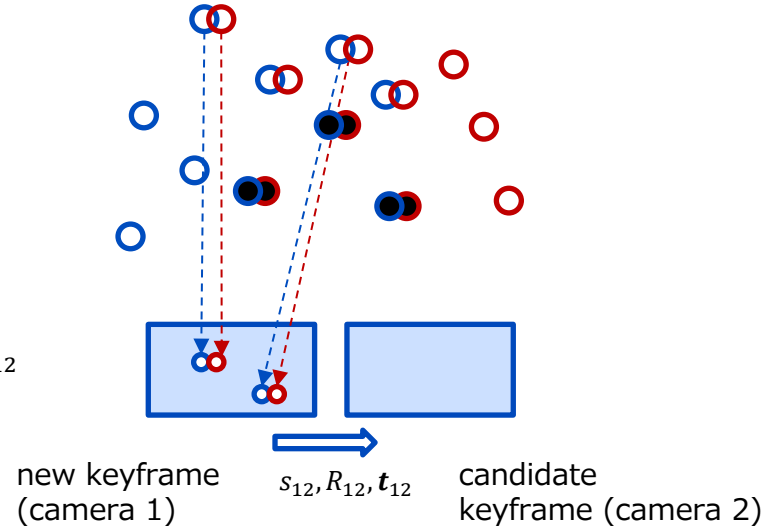
sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- Calculate relative pose by calling compute_Sim3:
- **Count inliers using the calculated relative pose**
  - Find the best relative pose during 200 iterations



Each landmark pair except the three which were used to calculate relative pose is classified as inlier or outlier.
Inliers criterion:

- Reprojection error both on the new keyframe and the candidate keyframe are lower than χ^2 threshold of significance level of 1% (looser than 5%)

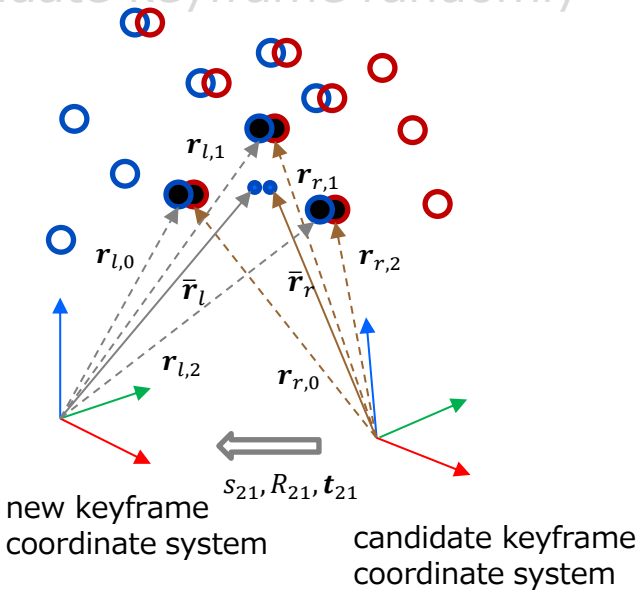


Compute relative pose

select_loop_candidate_via_Sim3 – 13 of 23

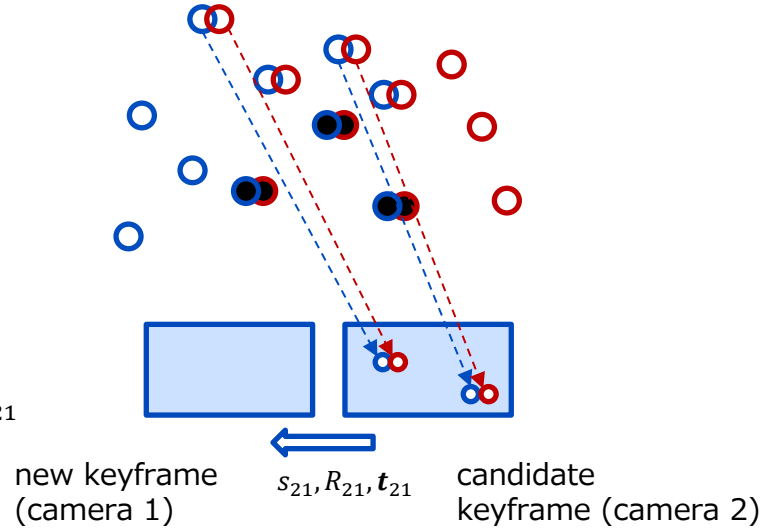
sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- Calculate relative pose by calling compute_Sim3:
- **Count inliers using the calculated relative pose**
  - Find the best relative pose during 200 iterations



Each landmark pair except the three which were used to calculate relative pose is classified as inlier or outlier.
Inliers criterion:

- Reprojection error both on the new keyframe and the candidate keyframe are lower than χ^2 threshold of significance level of 1% (looser than 5%)

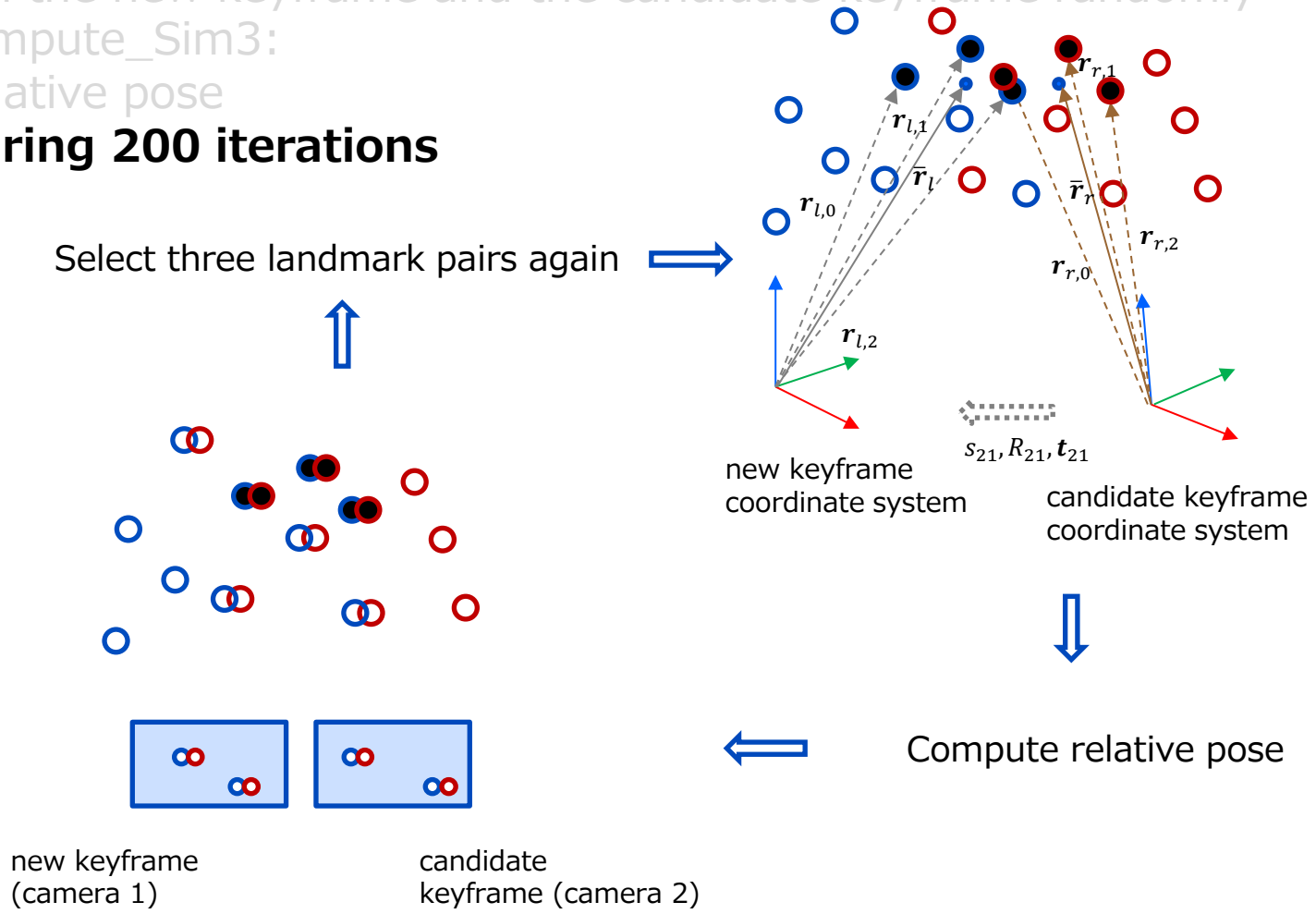


Compute relative pose

sim3_solver::find_via_ransac

- Select three landmark pairs between the new keyframe and the candidate keyframe randomly
- Calculate relative pose by calling compute_Sim3:
- Count inliers using the calculated relative pose
  - **Find the best relative pose during 200 iterations**

If the maximum inliers count is below min_num_inliers (=20), find_via_ransac is regarded as failed.
The pose which achieves the best score is chosen and will be used as the initial relative pose in transform_optimizer.

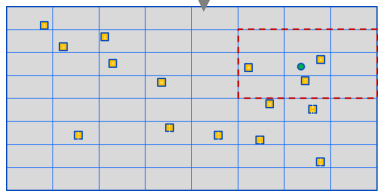


select_loop_candidate_via_Sim3 – 15 of 23

projection::match_keyframes_mutually

Try to correspond landmarks between the new keyframe and the candidate keyframe further. Note that landmarks in lms_in_keyfrm_1 are skipped because they are already corresponded before calling this function in bow_tree::match_keyframes.

- For each landmark of camera1 (the new keyframe):
  - Reproject it to camera2 (the candidate keyframe) using $R_{w1}, t_{w1}, s_{21}, R_{21} and\ \boldsymbol{t}_{21}$
  - Skip matching if:
    - cam_to_lm_dist is below min_cam_to_lm_dist
    - cam_to_lm_dist is above max_cam_to_lm_dist
  - Obtain keypoints within grid cells around the reprojected point
    - About grid cell, see assign_keypoints_to_grid.
  - find the best matched keypoint among them by Hamming distance:
    - Hamming distance must be the smallest among the keypoint candidates and below HAMMING_DIST_THR_HIGH
  - The best matched keypoint is registered to matched_indices_2_in_keyfrm_1
- Do above as for camera2 (the candidate keyframe)
  - The best matched keypoint is registered to matched_indices_1_in_keyfrm_2
- If the best matched keypoints are both in matched_indices_2_in_keyfrm_1 and matched_indices_1_in_keyfrm_2, and mutually link each other, they are regarded as corresponded



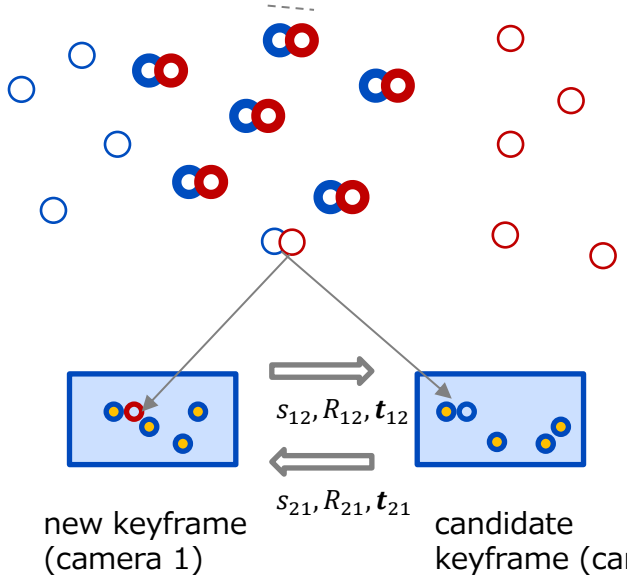
○ = landmark of the new keyframe (landmarks_1 in source code)

○ = landmark of the candidate keyframe (landmarks_2 in source code)

○ = landmark of the candidate keyframe which were corresponded by bow_database::match_keyframes (matched_lms_in_keyfrm_1 in source code)

○ = landmark of the new keyframe which corresponds to ○

○ is skipped because it is already corresponded.



select_loop_candidate_via_Sim3 – 16 of 23

<u>projection::match_keyframes_mutually</u>

Try to correspond landmarks between the new keyframe and the candidate keyframe further. Note that landmarks in matched_lms_in_keyrm_1 are skipped because they are already corresponded before calling this function.

- For each landmark of camera1 (the new keyframe):
  - Reproject it to camera2 (the candidate keyframe) using $R_{w1}, t_{w1}, s_{21}, R_{21} and\ \boldsymbol{t}_{21}$
  - Skip matching if:
    - cam_to_lm_dist is below min_cam_to_lm_dist
- Do
- If the best matched keypoints are both in matched_indices_2_in_keyfrm_1 and matched_indices_1_in_keyfrm_2, and mutually link each other, they are regarded as corresponded

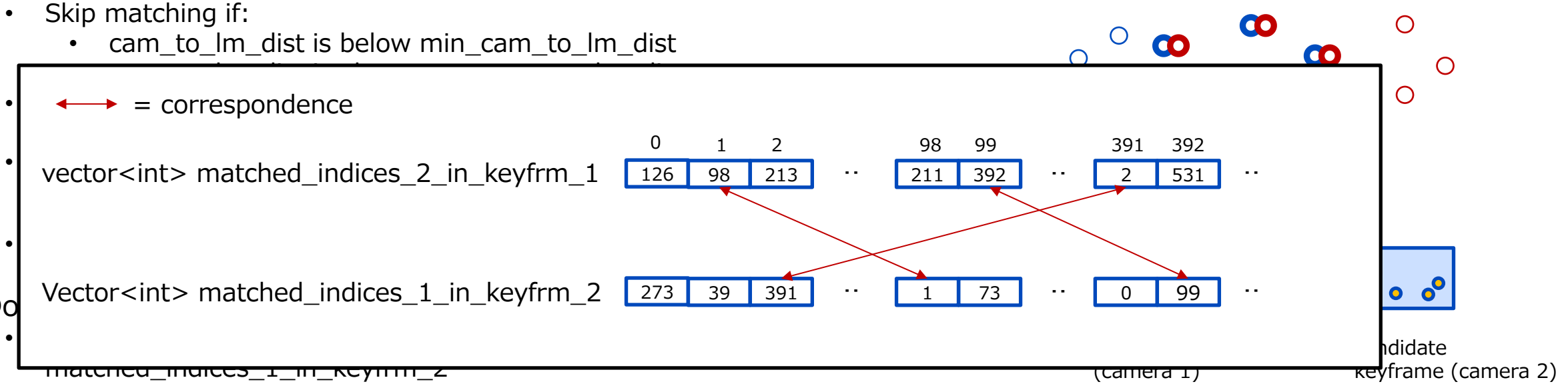
○ = landmark of the new keyframe (landmarks_1 in source code)

○ = landmark of the candidate keyframe (landmarks_2 in source code)

○ = landmark of the candidate keyframe which can be observed from the new keyframe, and were corresponded by bow_database::match_keyframes (matched_lms_in_keyfrm_1 in source code)

○ = landmark of the new keyframe which corresponds to ○

○ is skipped because it is already corresponded.



select_loop_candidate_via_Sim3 – 17 of 23

<u>transform_optimizer::optimize</u>

- Optimize with g2o
  - Linear solver = LinearSolverEigen
  - Block solver = BlockSolver_X (essentially it is BlockSolver_6_3 in stereo version and is BlockSolver_7_3 in monocular version)
  - Algorithm = OptimizationAlgorithmLevenberg
  - Optimizer = SparseOptimizer
- Vertex = relative pose with scale between the new keyframe and the candidate keyframe
- Edges = restriction between the relative pose with scale and reprojected points of landmarks onto the new keyframe and the candidate
  - Robust kernel (Huber loss) is set with delta = sqrt of χ^2 value larger than threshold of significance level of 5% (=10 instead of threshold of significance level of 5% = 7.81473)

transform_optimizer::optimize

Note that reprojection error is considered only on the left image. This is different in pose_optimizer::optimize – 3 of 9 or local_bundle_adjuster::optimize – 3 of 9 where reprojection error is considered both on the left and right images.

= relative pose between camera 1 and 2 ⇒ **added as vertex**

= landmarks of camera1 ⇒ **used to add edges**

= landmarks of camera2 ⇒ **used to add edges**

edge

= reprojection of landmark of camera2 onto camera1 (forward reprojection)

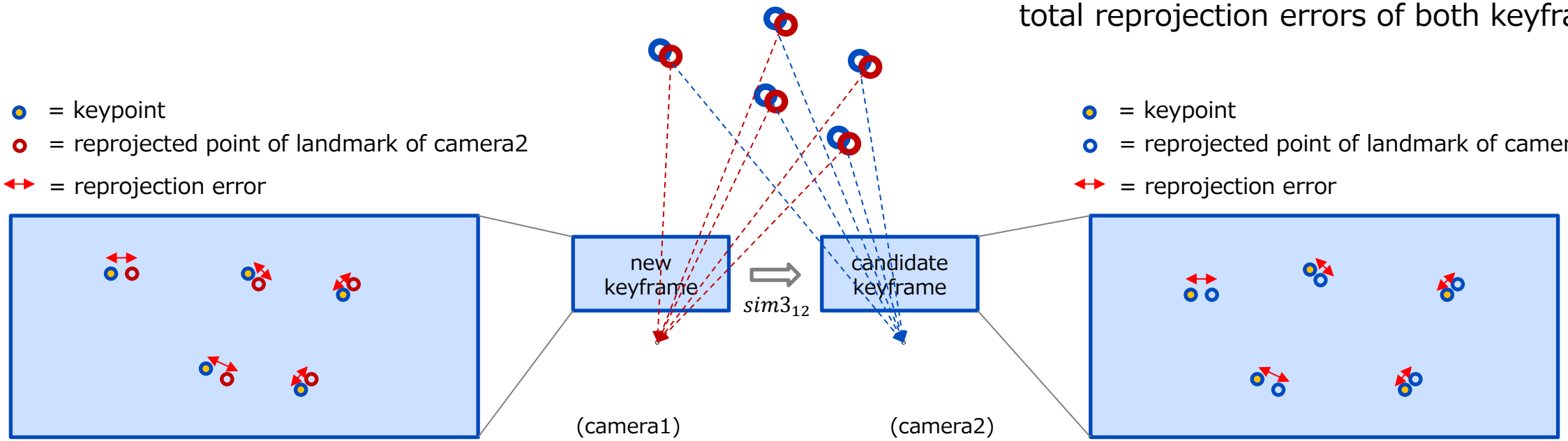
= reprojection of landmark of camera1 onto camera2 (backward reprojection)

The relative pose will be optimized through g2o optimization.
Landmark positions are regarded as fixed.
The optimization will be done to minimize total reprojection errors of both keyframes.

= keypoint

= reprojected point of landmark of camera2

= reprojection error



= keypoint

= reprojected point of landmark of camera1

= reprojection error

select_loop_candidate_via_Sim3 – 19 of 23

<u>transform_optimizer::optimize</u>

Vertex details (g2o::sim3::transformer_vertex)

- Derived class of ::g2o::BaseVertex<7, ::g2o::Sim3>
- relative pose (g2o_sim3_12) is set through setEstimate at first
- oplusImpl does setEstimate with $\exp(\Delta) * \mathrm{estimate}()$

select_loop_candidate_via_Sim3 – 20 of 23

transform_optimizer::optimize

Forward reprojection error detail (perspective_forward_reproj_edge):

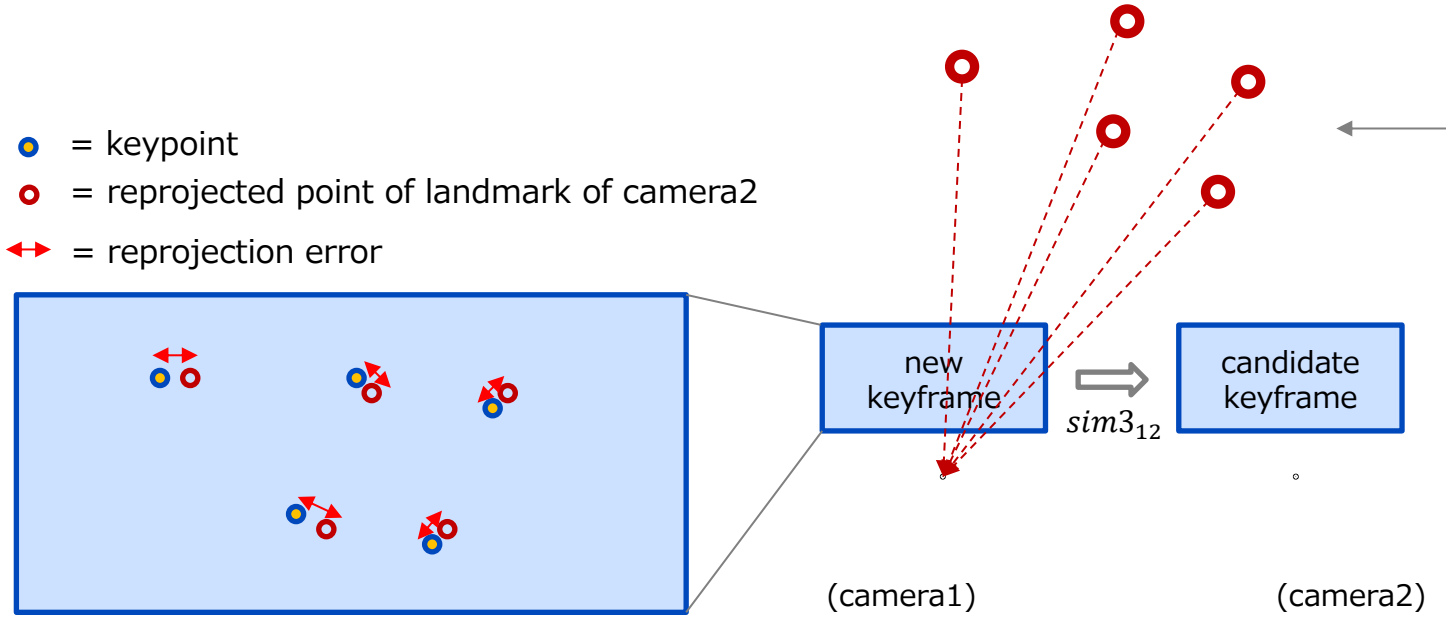
- Information matrix is diagonal matrix whose components value depend on pyramid level of the keypoint
- $X_2 = T_{2w}X_w, X_1 = sim3_{12}X_2$ *where* $sim3_{12}$ *is relative pose from camera1 to camera2*
- Reproject X_1 onto camera1
- Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i, x_right_i - reproj_x_right_i)^T$

$$= (x_i - f_x \frac{X_1^i}{Z_1^i} - c_x, y_i - f_y \frac{Y_1^i}{Z_1^i} - c_y, x_right_i - f_x \frac{X_1^i}{Z_1^i} - c_x + f_x \frac{baseline}{Z_1^i})^T$$

written in computeError() of forward_reproj_edge.h

T_{2w}
3D world coordinate → 3D camera2 coordinate

$sim3_{12}$
→ 3D camera1 coordinate



linearlizeOplus method is not overwritten.
Maybe numerical differentiation is used.

select_loop_candidate_via_Sim3 – 21 of 23

<u>transform_optimizer::optimize</u>

Backward reprojection error detail (perspective_backward_reproj_edge):

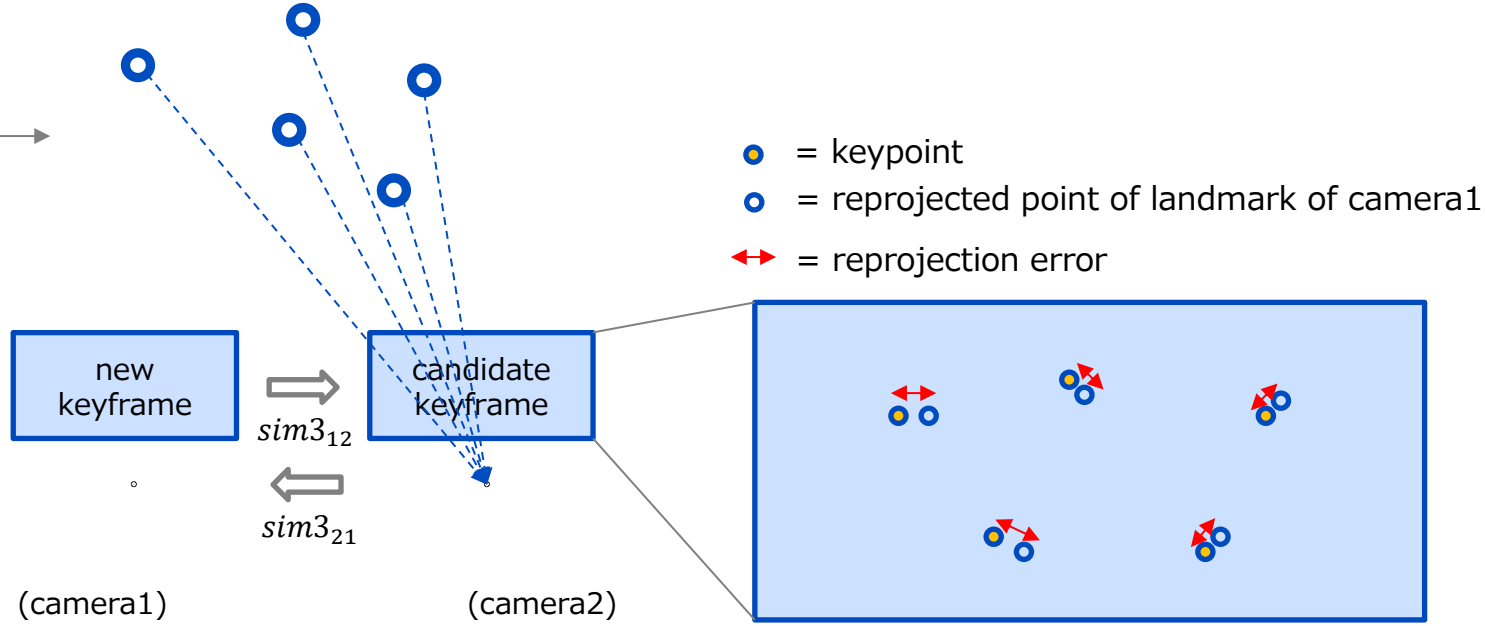
- Information matrix is diagonal matrix whose components value depend on pyramid level of the keypoint
- $X_1 = T_{1w}X_w, sim3_{21} = sim3_{12}^{-1}, X_2 = sim3_{21}X_1$ $where\ sim3_{21}\ is\ relative\ pose\ from\ camera2\ to\ camera1$
- Reproject X_2 onto camera2
- Residual error is $\boldsymbol{E} = (x_i - reproj_x_i, y_i - reproj_y_i, x_right_i - reproj_x_right_i)^T$

$$= (x_i - f_x\frac{X_2^i}{Z_2^i} - c_x, y_i - f_y\frac{Y_2^i}{Z_2^i} - c_y, x_right_i - f_x\frac{X_2^i}{Z_2^i} - c_x + f_x\frac{baseline}{Z_2^i})^T$$

written in computeError() of backward_reproj_edge.h

T_{1w}
3D world coordinate → 3D camera1 coordinate

$sim3_{21}$
→ 3D camera2 coordinate



linearlizeOplus method is not overwritten.
Maybe numerical differentiation is used.

select_loop_candidate_via_Sim3 – 22 of 23

<u>transform_optimizer::optimize</u>

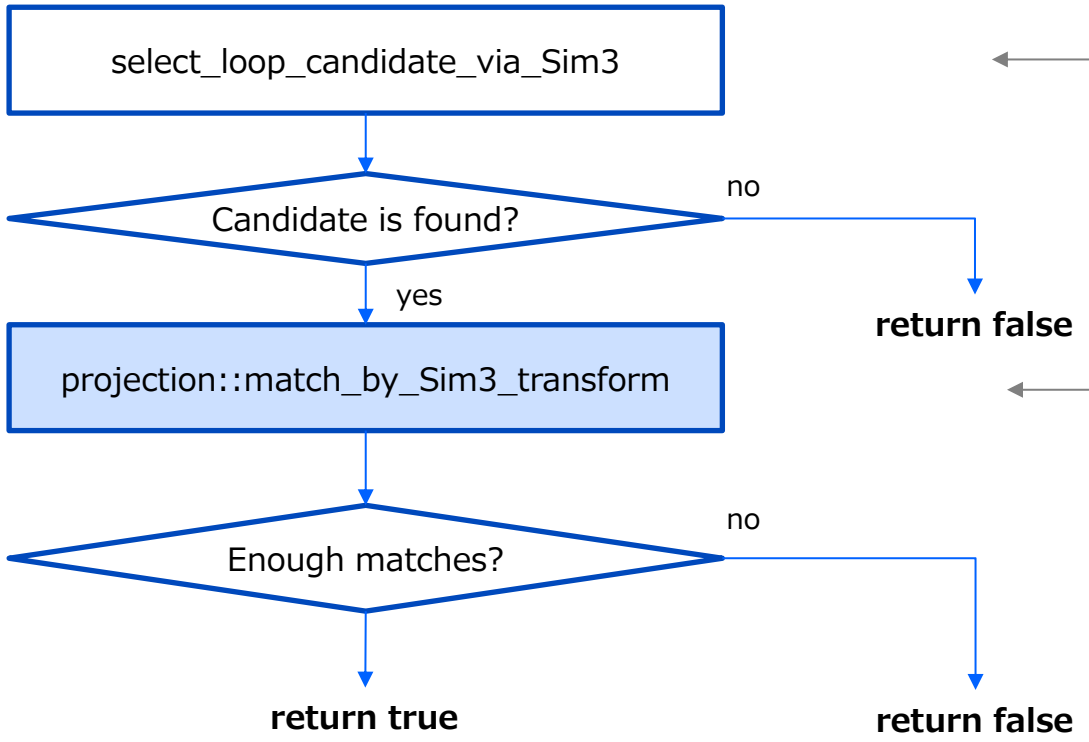
- Do optimization
  - Call optimizer.optimize once
  - Reject Edges as outliers whose χ^2 value is greater than or equal to 10
  - Call optimizer.optimize again
  - Count inliers whose χ^2 value is less than 10
- After optimization
  - Optimized $sim3_{12}$ is written to reference parameter of transform_optimizer::optimize
  - The number of inliers is returned

Note that the optimization result is not applied to the new keyframe pose here because it is in validation phase now. The optimization result is used later in correct_loop.

select_loop_candidate_via_Sim3 – 23 of 23

- If the number of inliers returned by transform_optimizer::optimize is above or equal to 20 (hard coded value), the optimization is regarded as succeeded
  - Optimized $sim3_{12}$ (g2o_sim3_cand_to_curr in source code) is converted to $sim3_{1w}$ (g2o_Sim3_world_to_curr)
    - $sim3_{12}$ = transformation from the new keyframe coordinate system to the candidate keyframe coordinate system
    - $sim3_{1w}$ = transformation from the new keyframe coordinate system to the world coordinate system
    - g2o_Sim3_world_to_curr will be used in global_optimization_module::correct_loop
  - selected keyframe is returned by reference parameter of select_loop_candidate_via_Sim3
  - Landmarks of the selected keyframe which are observed from the new keyframe have already been copied to curr_match_lms_observed_in_cand
  - return true
- If all of candidates failed, return false

loop_detector::validate_candidates so far



Validation by using the new keyframe and candiate:

- First validate by bow-based match
- Next validate by RANSAC with calculating Sim3 relative pose
- Further match with reprojection to each other
- Finally validate by counting inliers after optimizing relative Sim3 pose

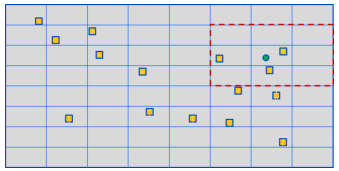
Next:

- Try to match more landmarks by using covisibility keyframes of the candidate

projection::match_by_Sim3_transform

Try to correspond landmarks between the new keyframe and covisibility keyframes of the candidate keyframe for robustness.

- Reproject a landmark of the candidate keyframe or its covisibilitiy keyframes to the new keyframe using the optimized pose obtained by select_loop_candidate_via_Sim3
  - See projection::match_current_and_last_frames – 3 of 7 about reprojection.
- Skip matching if:
  - cam_to_lm_dist is above max_valid_dist_
  - cam_to_lm_dist is below min_valid_dist_
  - Angle between cam_to_lm_vec and mean_normal_ is above 60 degree
  - See insert_new_keyframe – 6 of 6 about mean_normal_, max_valid_dist_ and min_valid_list_.
- Obtain keypoints within grid cells around the reprojected point
  - About grid cell, see assign_keypoints_to_grid.
- find the best matched keypoint among them by Hamming distance:
  - Hamming distance must be the smallest among the keypoint candidates and below HAMMING_DIST_THR_LOW
  - The best matched keypoint is registered to matched_lms_in_keyfrm (= curr_match_lms_observed_in_cand)



○ = landmark of the new keyframe

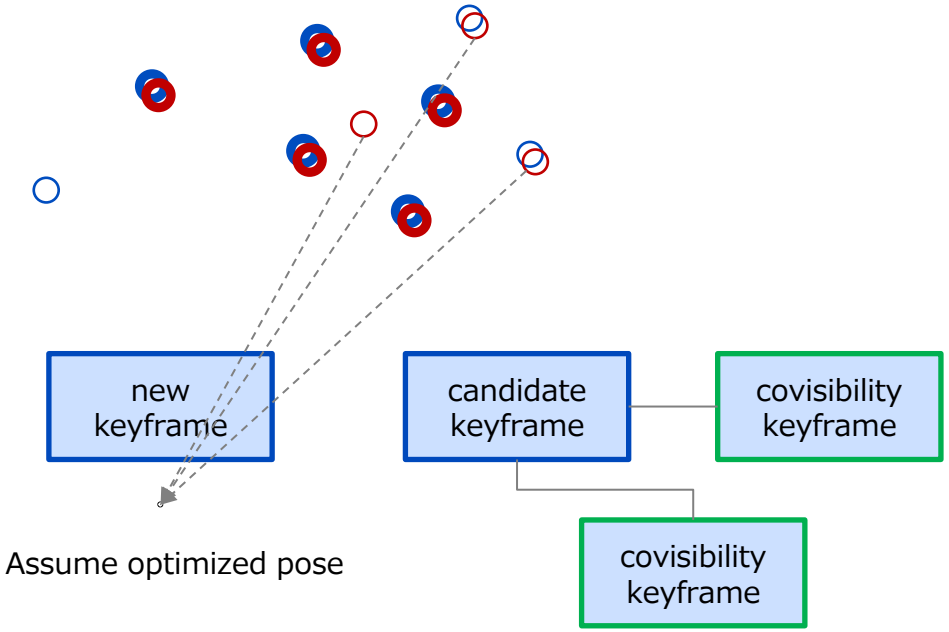
○ = landmark of the candidate or its covisibility keyframes

○ = landmark of the candidate keyframe which are corresponded to that of the new keyframe

○ = landmark of the new keyframe which corresponds to ○

- - - → = reprojection to new keyframe using optimized pose

○ is skipped because it is already corresponded.



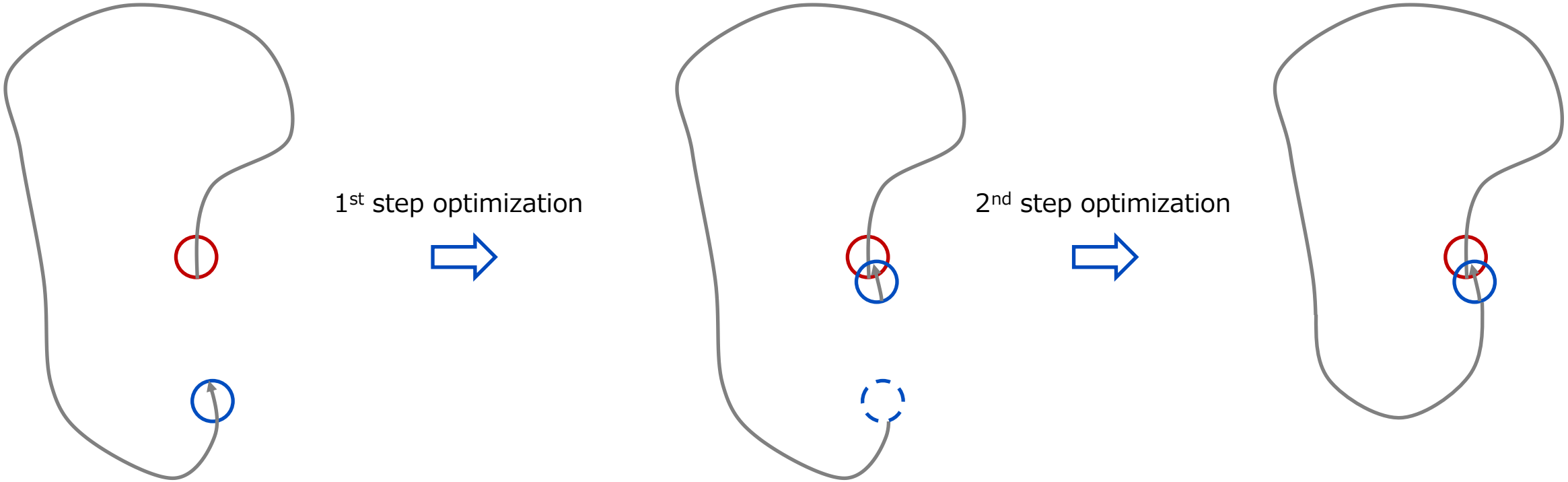
Return from loop_detector::validate_candidates

- If the number of corresponded landmarks between the new keyframe and the candidate keyframe with its covisibility keyframes is above or equal to num_final_matches_thr (=40), return true
- Else return false

- Note that the optimized pose obtained by transform_optimizer::optimize is not applied
  - The application will be done in global_optimization_module::correct_loop using this optimization result

GLOBAL_OPTIMIZATION – CORRECT_LOOP

correct_loop overview

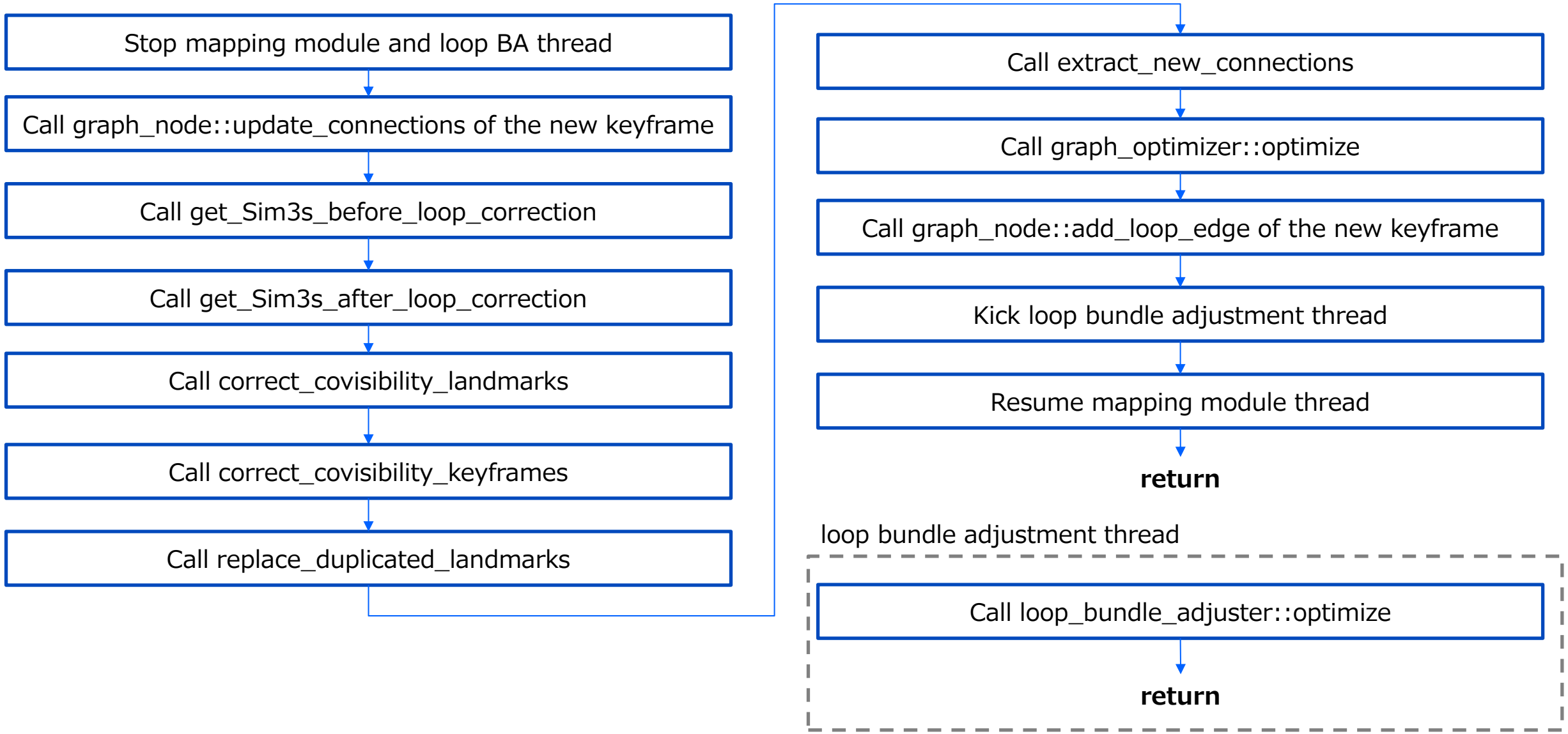


= the new keyframe and its covisibility keyframes

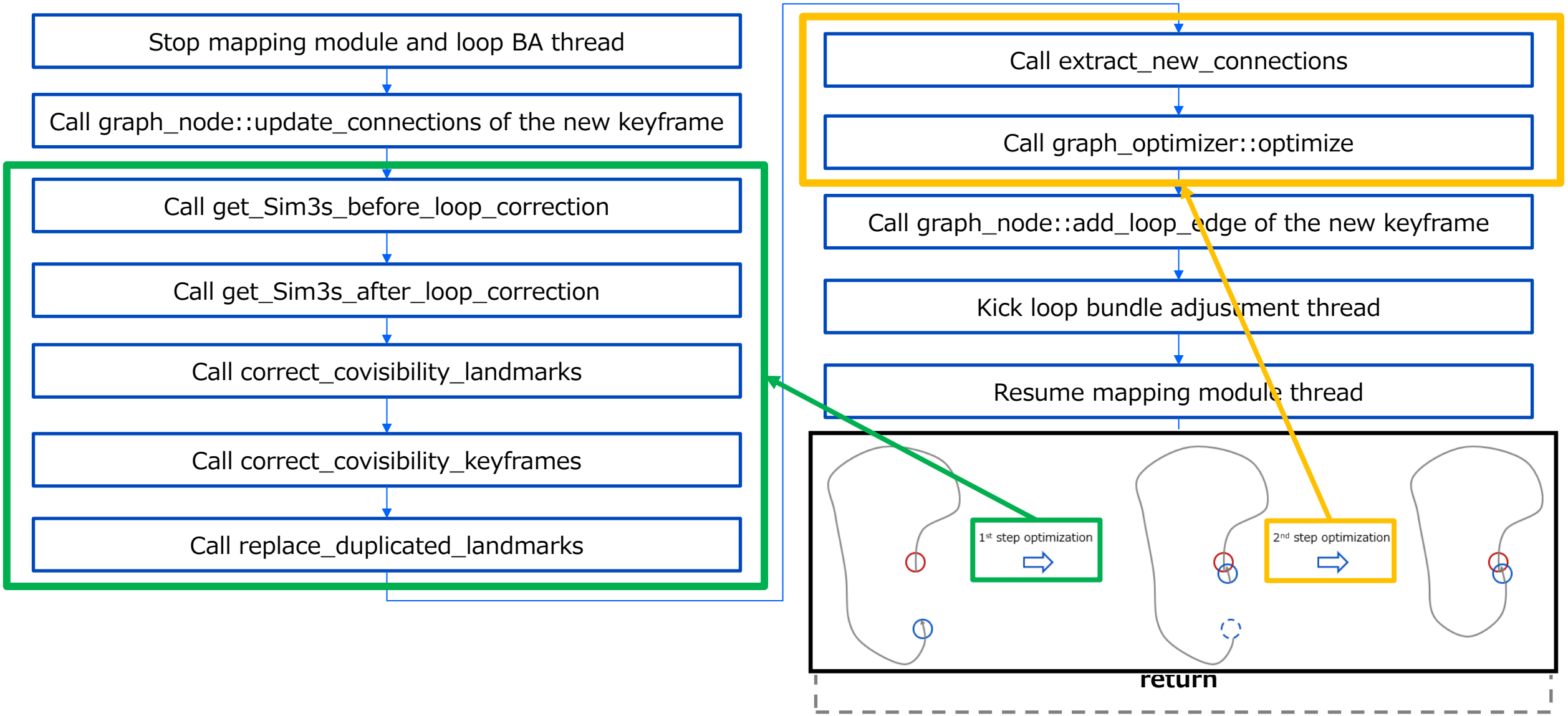
= the final candidate keyframe and its covisibility keyframes

3rd step optimization
by loop bundle adjustment

global_optimization_module::correct_loop – 1 of 2



global_optimization_module::correct_loop – 2 of 2



Stop mapping module and loop BA thread

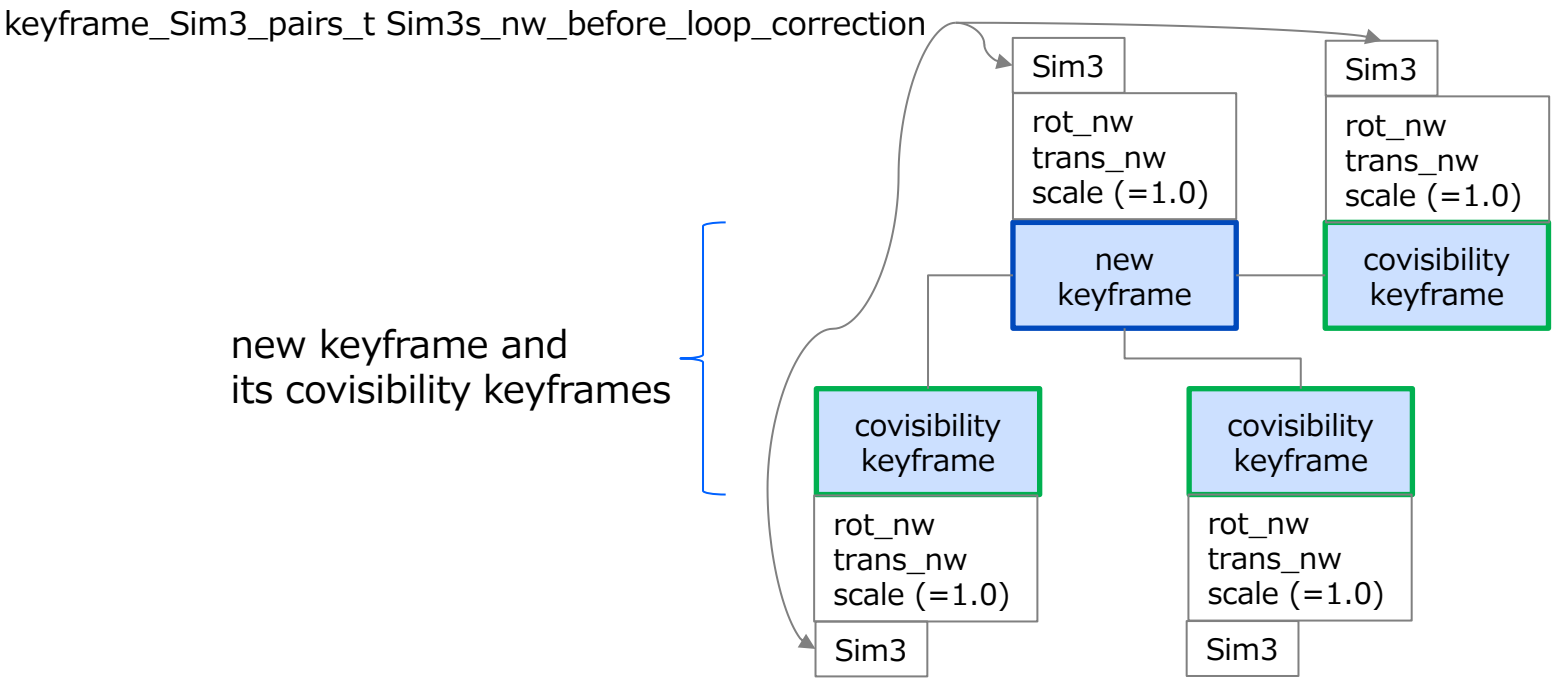
- Global bundle adjustment thread is going to optimize all landmark positions and keyframe poses
- Let mapping module thread wait till the optimization is finished for exclusive access
- If previous loop bundle adjustment has been running, it is no meaning because its optimization will be soon overwritten, so stop it

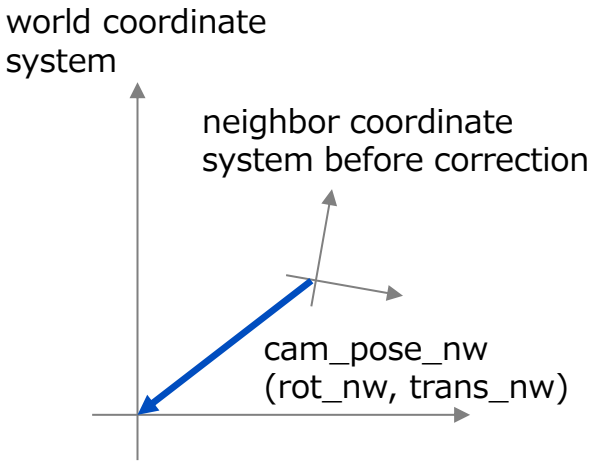
graph_node::update_connections

- Already explained in store_new_keyframe – 2 of 7 to store_new_keyframe – 6 of 7
- It has been called once in mapping_module::store_new_keyframe by mapping module thread
- Call it again because landmarks of the new keyframe may have changed since then, and thus connections for the new keyframe may change
  - Tracking and Mapping threads may have changed landmarks because they can run in parallel

get_Sim3s_before_loop_correction

Record poses of the new keyframe and its covisibility keyframes before loop correction.



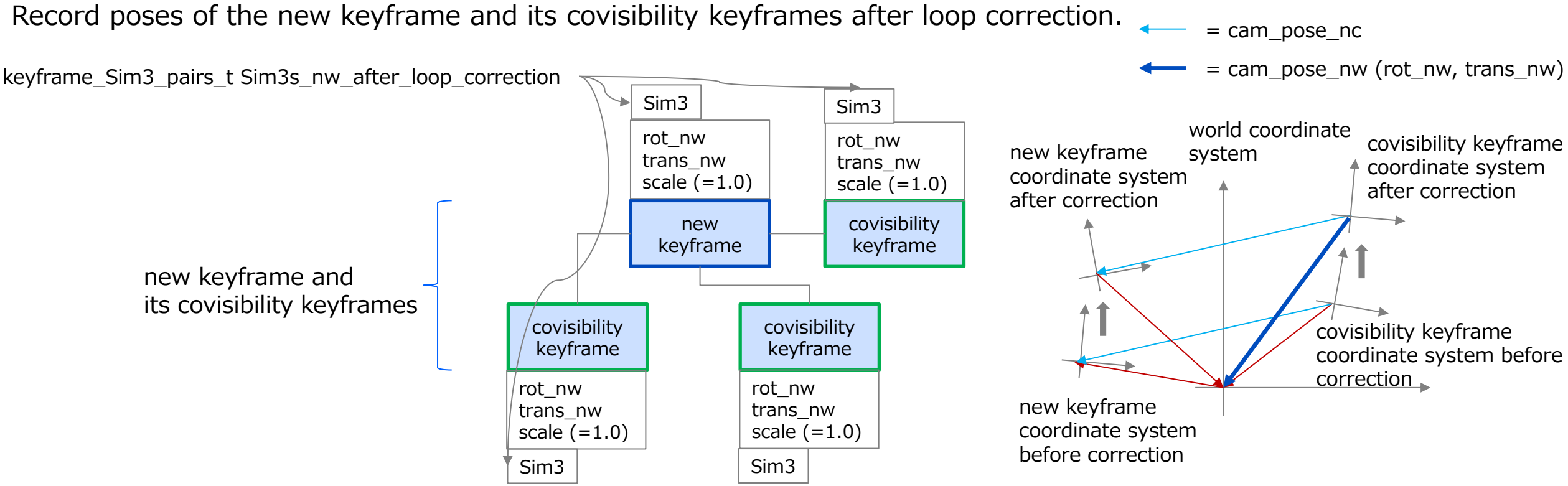


- rot_nw = rotation matrix from a keyframe coordinate system to the world coordinate system
- trans_nw = translation vector from a keyframe coordinate system to the world coordinate system
- scale = 1.0 (fixed)

Note that any keyframe coordinate system has not been aligned with optimization

get_Sim3s_after_loop_correction

Record poses of the new keyframe and its covisibility keyframes after loop correction.



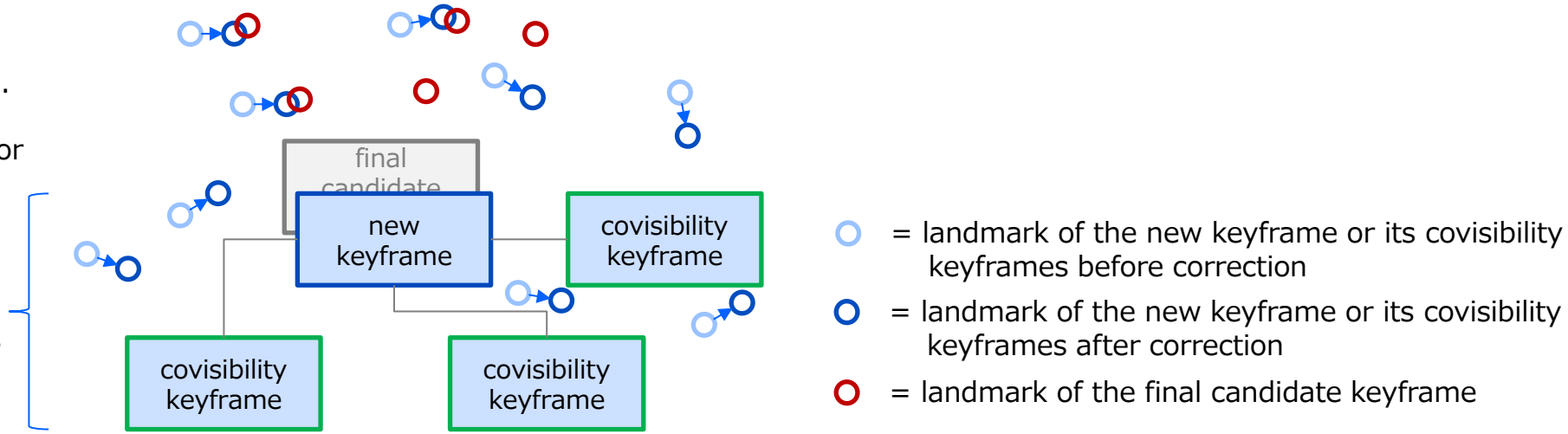
- rot_nw = rotation matrix from a keyframe coordinate system **aligned with the transform optimization** to the world coordinate system
- trans_nw = translation vector from a keyframe coordinate system **aligned with the transform optimization** to the world coordinate system
- scale = 1.0 (fixed)

Optimized result by transform_optimizer (select_loop_candidate_via_Sim3 – 17 of 23 to select_loop_candidate_via_Sim3 – 22 of 23) is used here.

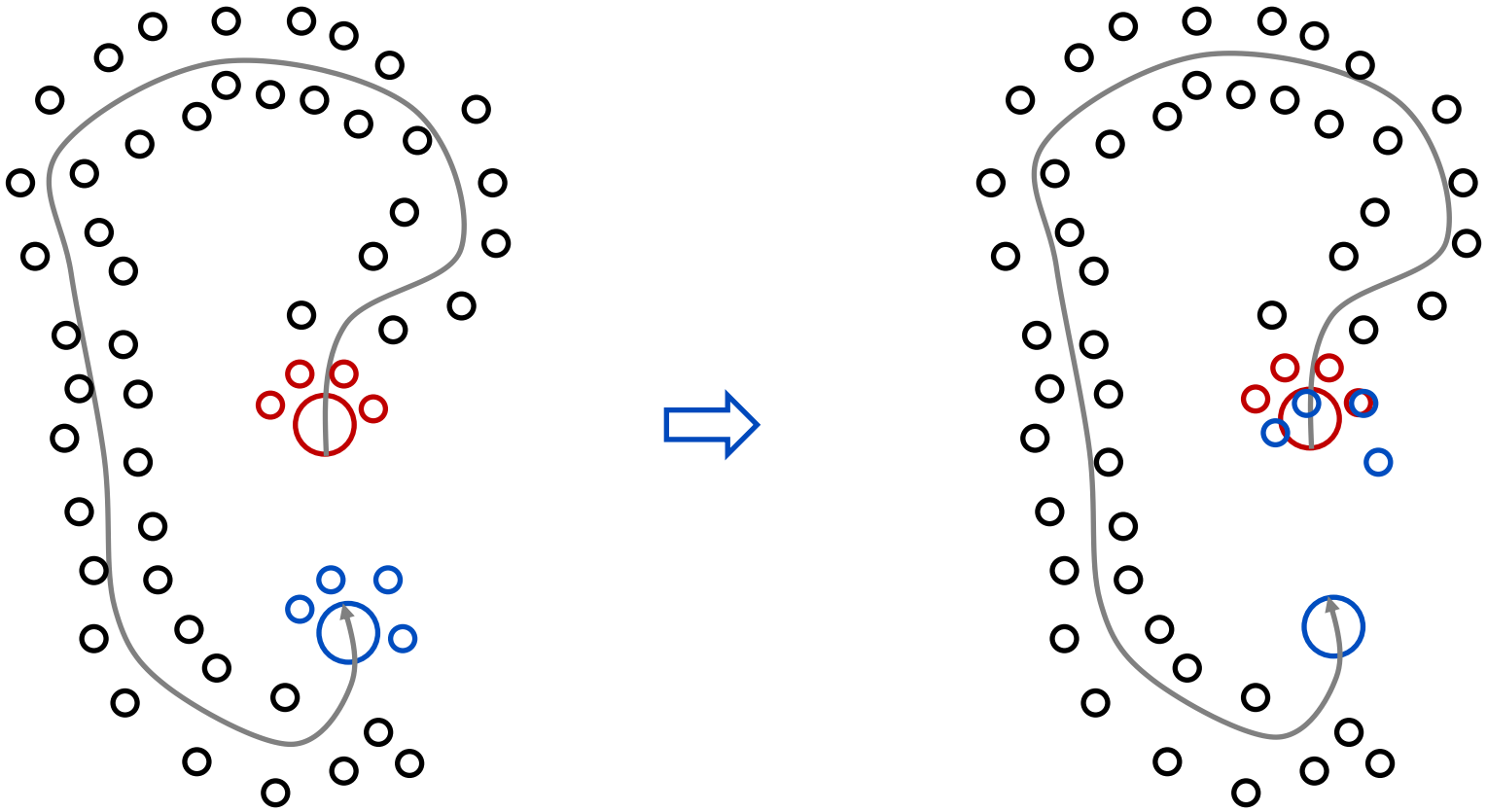
correct_covisibility_landmarks

- For each keyframe of the new keyframe and its covisibility keyframes:
  - For each landmark of the keyframe:
    - Correct the landmark position using Sim3s_nw_before_loop_correction and Sim3s_nw_after_loop_correction ($T_wn_after_correction \cdot T_nw_before_correction \cdot X_w$)
      - Optimization result of transform optimizer has been applied to Sim3s_nw_after_loop_correction
    - If already corrected, skip it
    - update_normal_and_depth for the landmark because of geometry change
      - see insert_new_keyframe – 6 of 6 about update_normal_and_depth
    - Set the keyframe id to its ref_keyfrm_id_in_loop_fusion_

The landmark correction does not care which keyframe is used at all. For example, neither care the landmark's reference keyframe, nor its mean normal.



Result of correct_covisibility_landmarks



= the new keyframe and its covisibility keyframes

= the final candidate keyframe and its covisibility keyframes

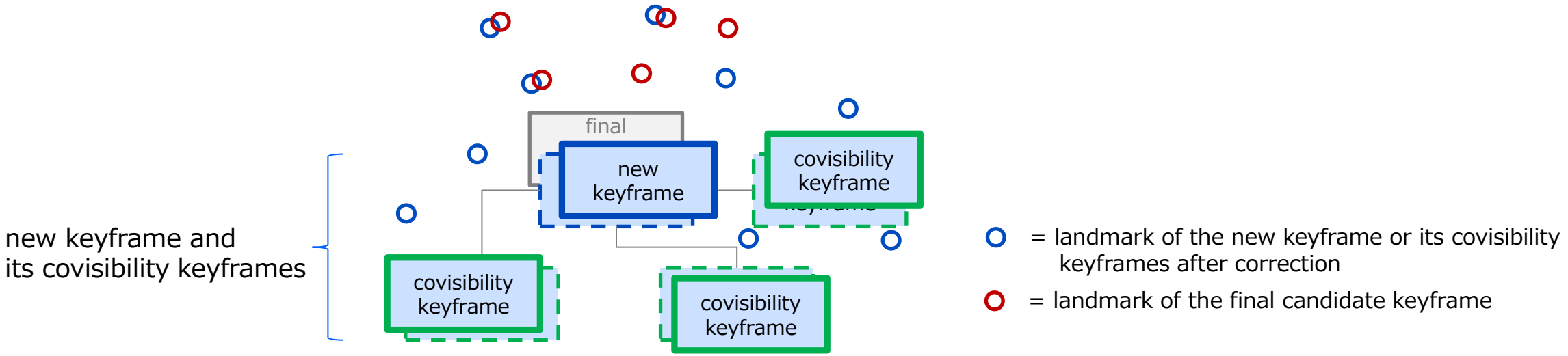
= landmarks of

= landmarks of

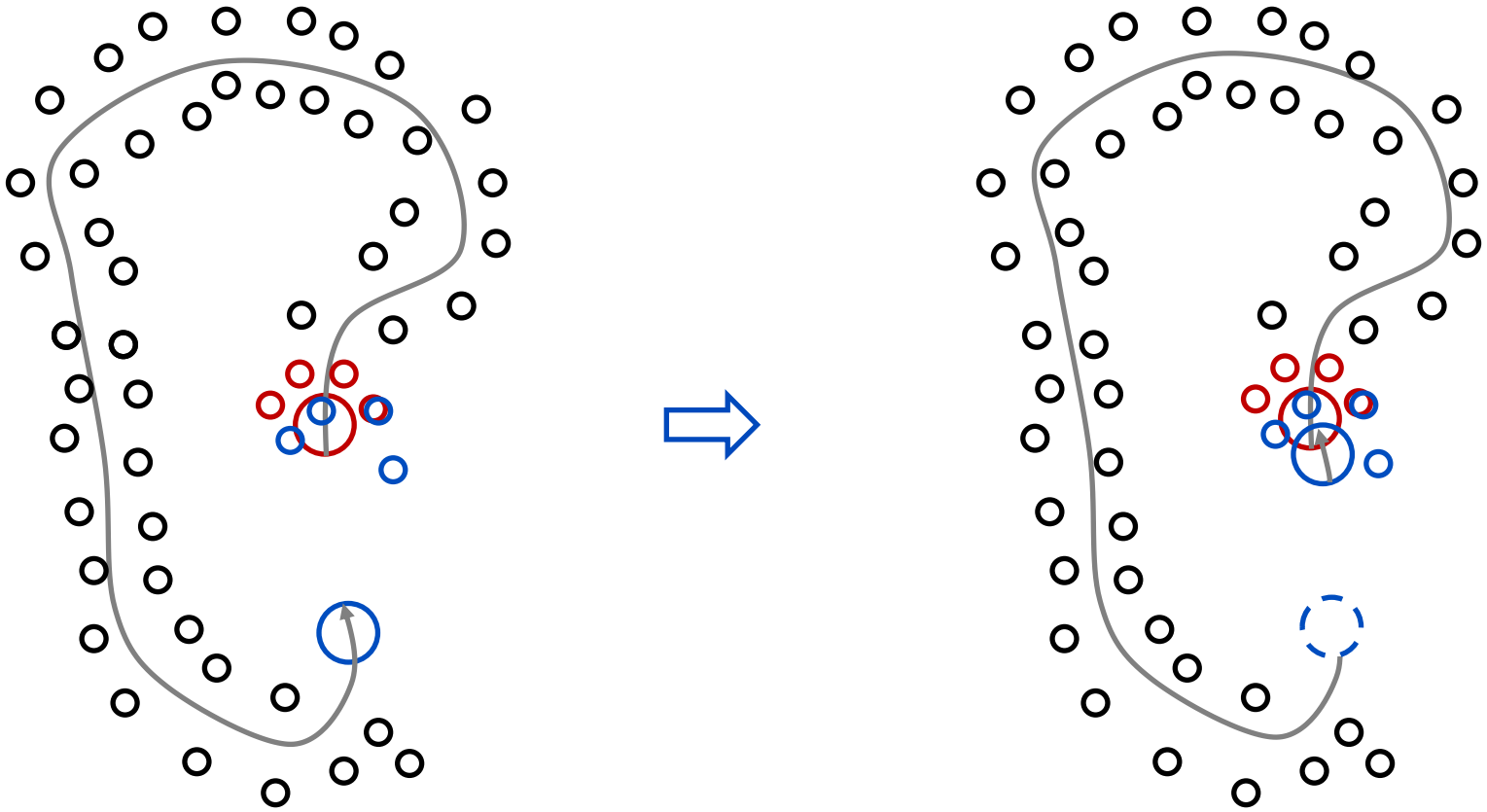
= other landmarks

correct_covisibility_keyframes

- For each keyframe of the new keyframe and its covisibility keyframes:
  - Correct the keyframe pose using Sim3s_nw_after_loop_correction
    - Optimization result of transform optimizer has been applied to Sim3s_nw_after_loop_correction
  - update_connections for the keyframe
    - see store_new_keyframe – 2 of 7 to store_new_keyframe – 6 of 7



Result of correct_covisibility_keyframes



= the new keyframe and its covisibility keyframes

= the final candidate keyframe and its covisibility keyframes

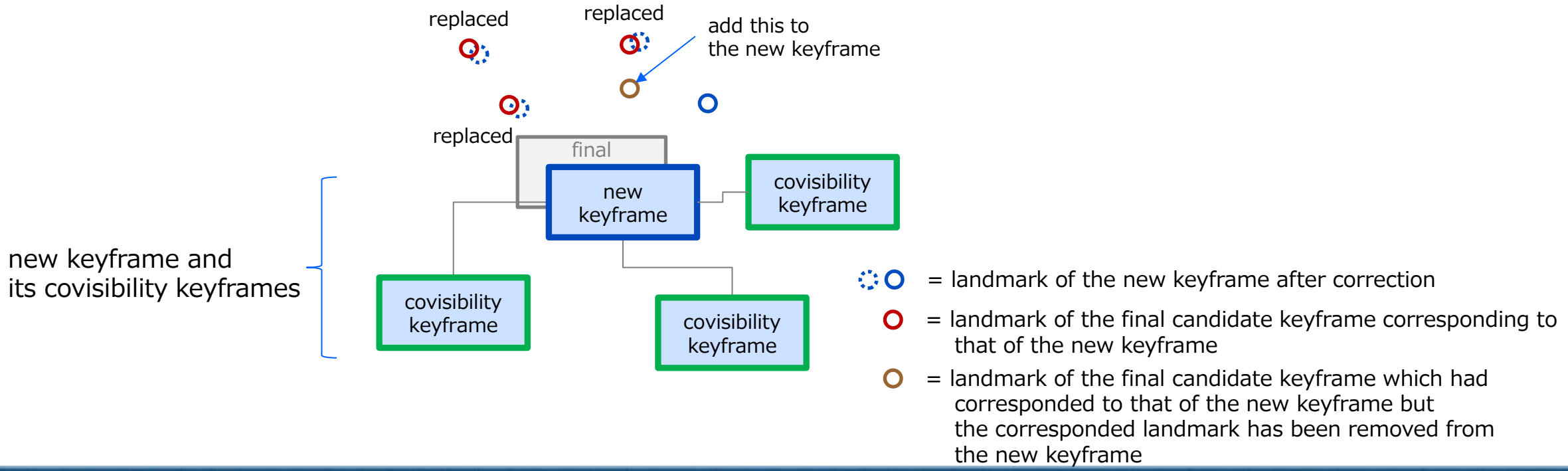
= landmarks of

= landmarks of

= other landmarks

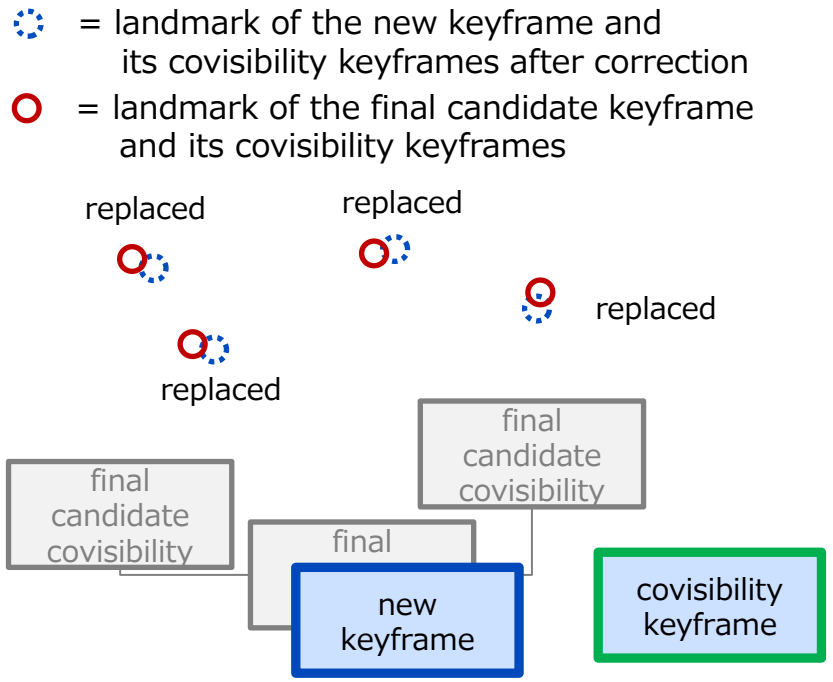
replace_duplicated_landmarks – 1 of 2

- For each landmark of the final candidate keyframe which corresponds to that of the new keyframe:
  - If corresponded landmark of the new keyframe exists:
    - Replace the corresponded landmark of the new keyframe with the final candidate keyframe's one
  - Else:
    - Add it to the new keyframe

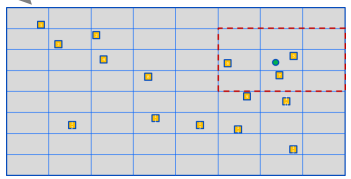


replace_duplicated_landmarks – 2 of 2

Try to detect duplicated landmarks between the new keyframe with its covibility keyframes and the final candidate keyframe with its covisibility keyframes. (fuse::detect_duplication)

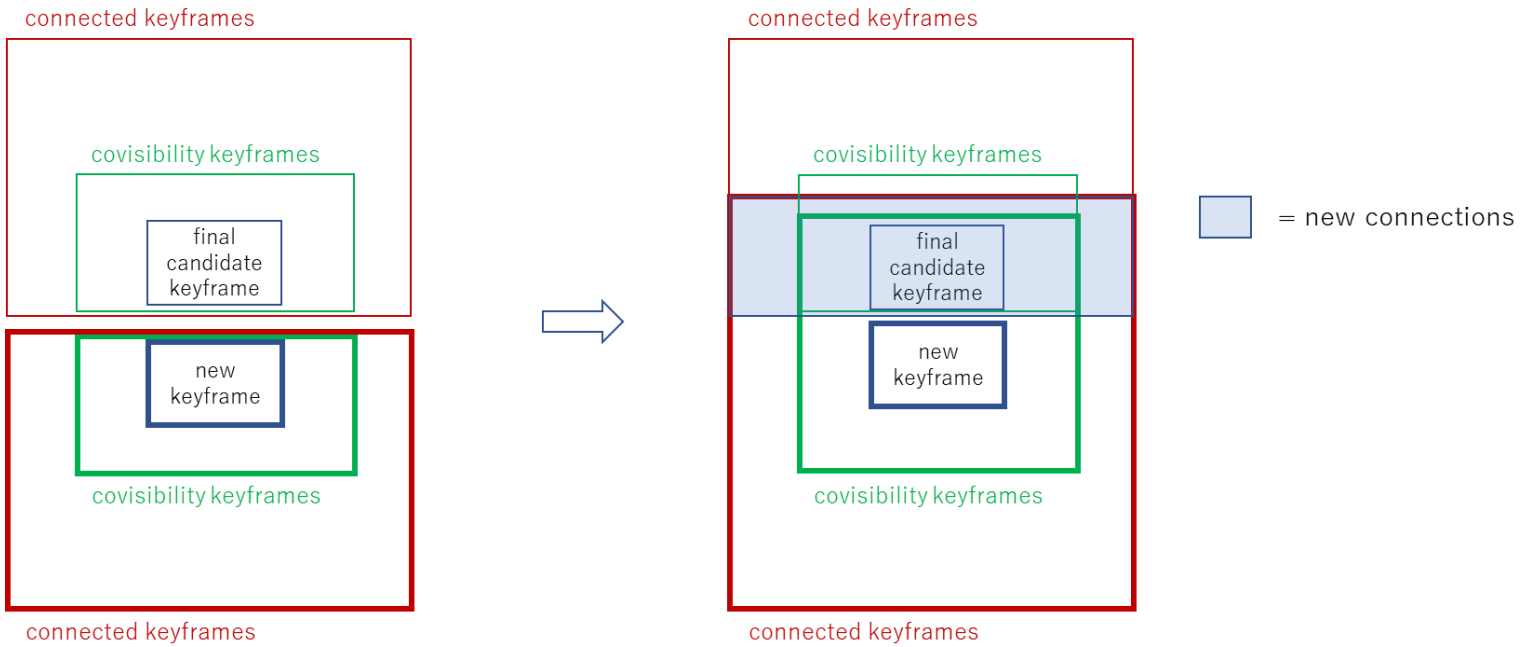


- Reproject landmark of the final candidate keyframe or its covisibility keyframe to the new keyframe or its covisibility keyframe using the optimized relative pose
  - See projection::match_current_and_last_frames – 3 of 7 about reprojection.
- Skip matching if:
  - cam_to_lm_dist is above max_valid_dist_
  - cam_to_lm_dist is below min_valid_dist_
  - Angle between cam_to_lm_vec and mean_normal_ is above 60 degree
  - See insert_new_keyframe – 6 of 6 about mean_normal_, max_valid_dist_ and min_valid_list_.
- Obtain keypoints within grid cells around the reprojected point
  - About grid cell, see assign_keypoints_to_grid.
- find the best matched keypoint among them by Hamming distance:
  - Hamming distance must be the smallest among the keypoint candidates and below HAMMING_DIST_THR_LOW
- If a landmark corresponding to the best matched keypoint exists, it is registered to duplicated_lms_in_keyfrm, else add it as a new landmark (landmarks in duplicated_lm_in_keyfrm are replaced with landmarks of the final candidate keyframe or its covisibility keyframes after returning to replace_duplicatd_landmarks)

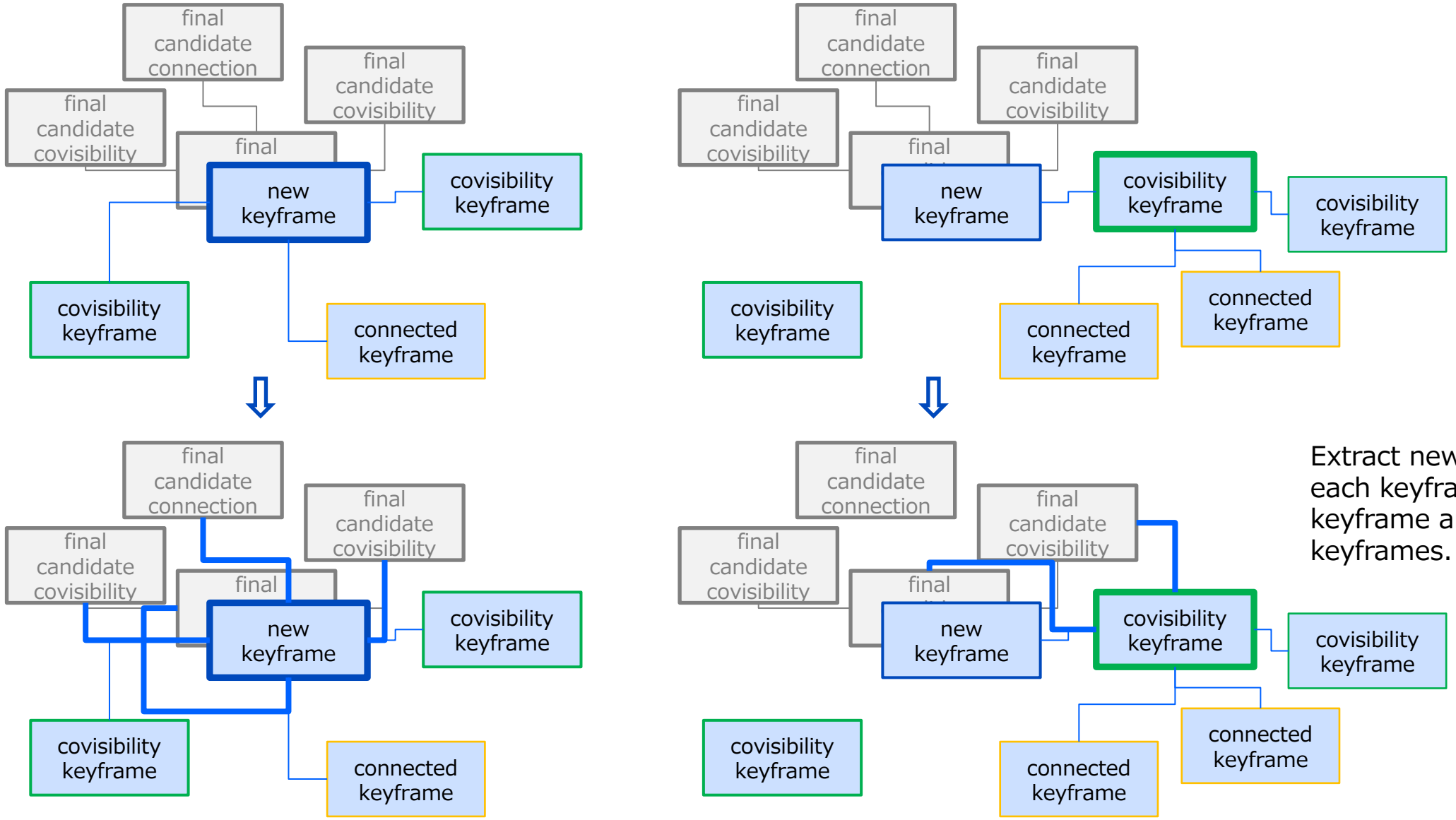


extract_new_connections – 1 of 3

- For each keyframe of the new keyframe and its covisibility keyframes:
  - update_connections
    - Connections are expected to change because landmarks of the keyframe come to be shared with the final candidate keyframe and its connected keyframes because of replace_duplicated_landmarks.
    - see store_new_keyframe – 2 of 7 to store_new_keyframe – 6 of 7 for update_connections
  - Remove old covisibility and connected keyframes of the new keyframe from the updated connections



extract_new_connections – 2 of 3

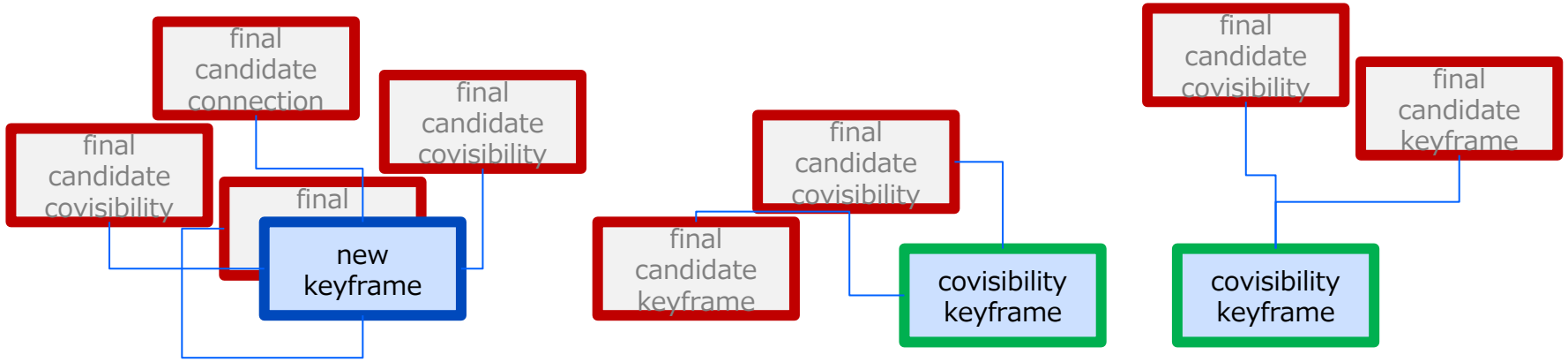


..

Extract new connections for each keyframe of the new keyframe and its covisibility keyframes.

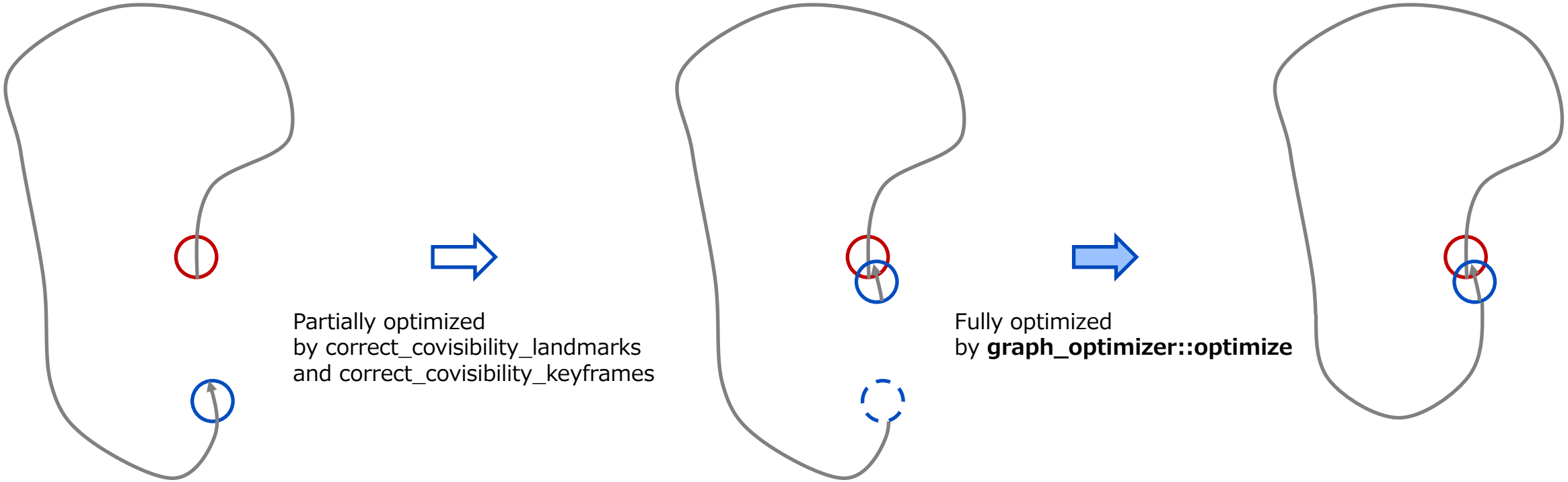
extract_new_connections – 3 of 3

Extracted new connections are as follows:



new_connections will be used in graph_optimizer::optimize for edges.

graph_optimizer::optimize – 1 of 7



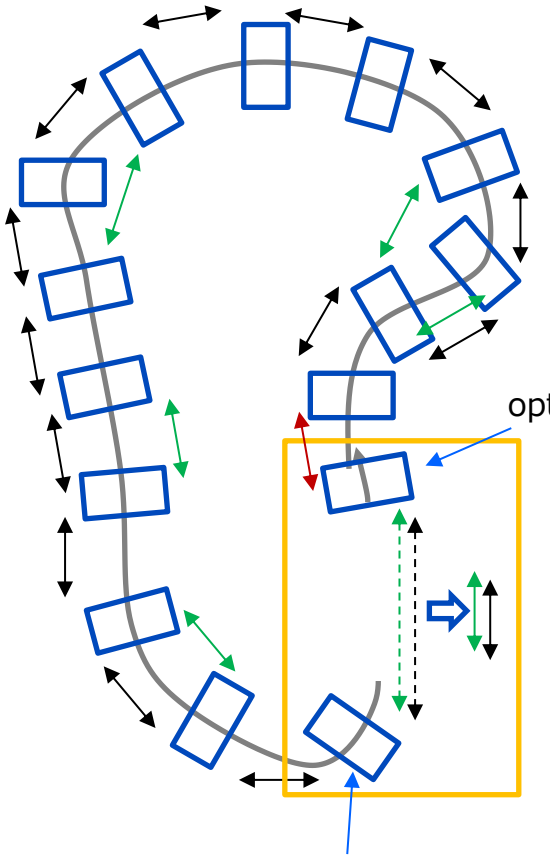
= the new keyframe and its covisibility keyframes

= the final candidate keyframe and its covisibility keyframes

graph_optimizer::optimize – 2 of 7

- Optimize with g2o
  - Linear solver = LinearSolverCSparse
  - Block solver = BlockSolver_7_3
  - Algorithm = OptimizationAlgorithmLevenberg
  - Optimizer = SparseOptimizer
- Vertices = all keyframe poses
- Edges = restriction between keyframes:
  - Between keyframes in new connections
  - Between each of all keyframes and its spanning tree parent
  - Between loop edges
  - Between each of all keyframes and its covisibility keyframes
  - No robust kernel is set in any case

graph_optimizer::optimize – 3 of 7



= keyframe ⇒ **added as vertex**

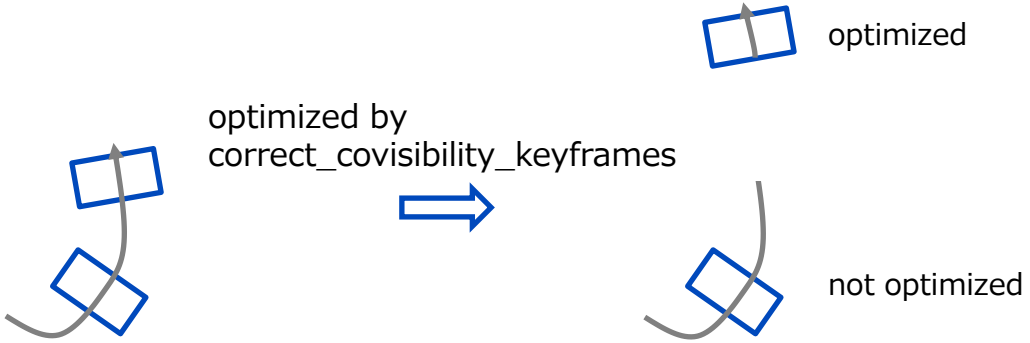
= relative pose between keyframes in new connections ⇒ **used to add edge**

= relative pose between parent and child of spanning tree ⇒ **used to add edge**

= relative pose between keyframes which have100 or more common landmarks ⇒ **used to add edge**

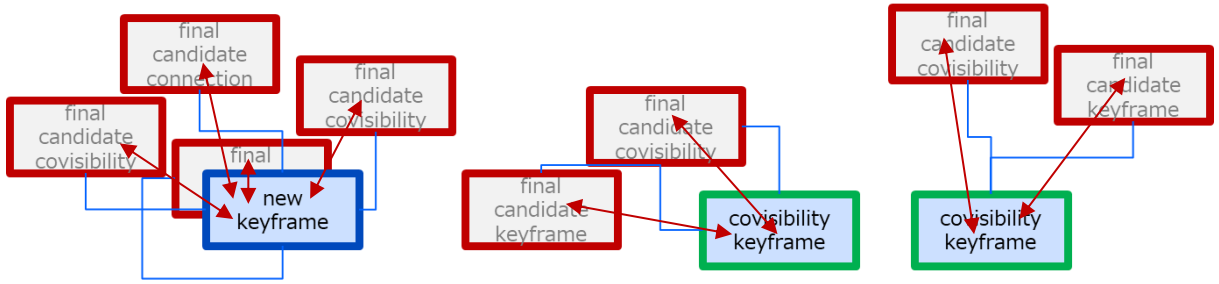
landmark positions are not optimized, and not used for the optimization.

These keyframes were closer each other before correct_covisibility_keyframes. So setMeasurement for this edge with original relative pose at first. By doing this, it is forced to become close after optimization.

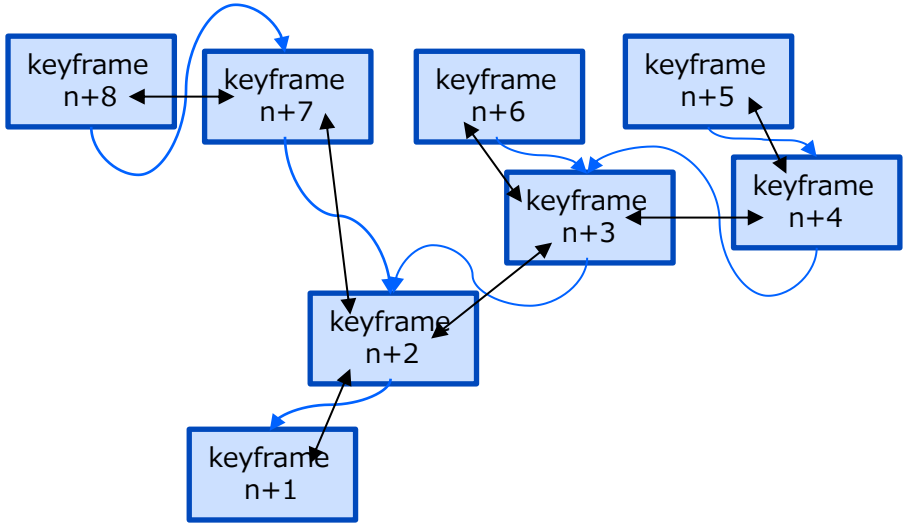


graph_optimizer::optimize – 4 of 7

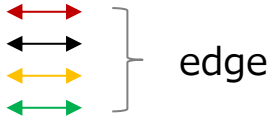
Edges between keyframes in new connections
(applied only with min_weight (=100) or more common landmarks, but edges between the new keyframe and the final candidate keyframe is always used without the condition.)

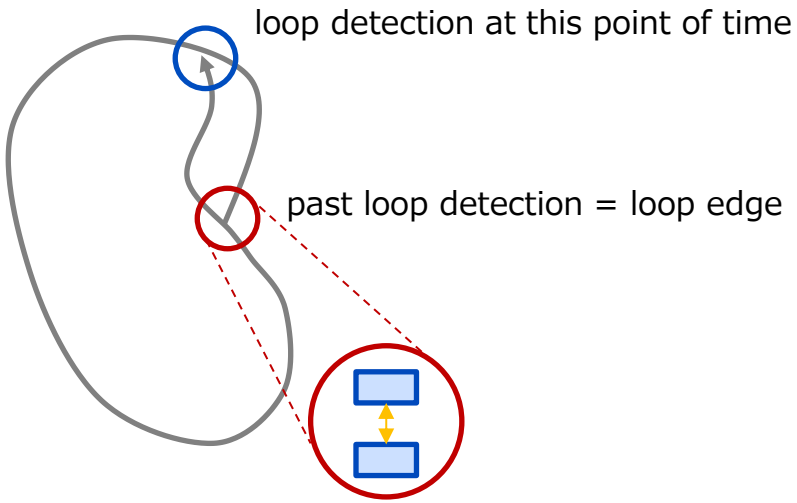


Edges between each of all keyframes and its spanning tree parent

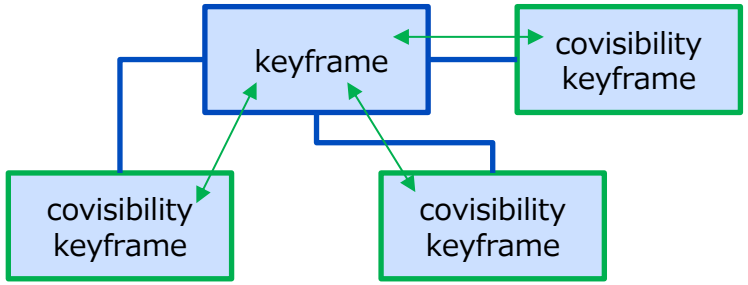


Edges between loop edges





Edges between each of all keyframes and its covisibility keyframes
(applied only with min_weight (=100) or more common landmarks)

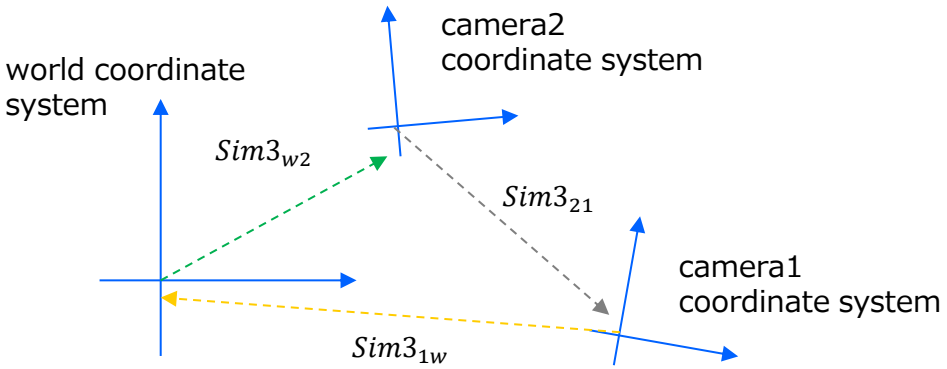


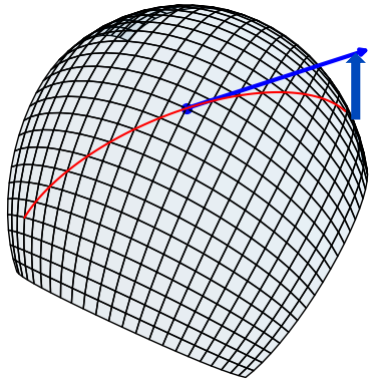
graph_optimizer::optimize – 5 of 7

- Vertex details (g2o::sim3::shot_vertex)
  - Derived class of ::g2o::BaseVertex<7, ::g2o::Sim3>
  - all keyframe poses (Sim3_cw) are set through setEstimate at first
  - oplusImpl does setEstimate with $\exp(\Delta) * \mathrm{estimate}()$

graph_optimizer::optimize – 6 of 7

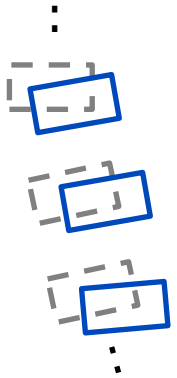
- Edge details (graph_opt_edge derived from ::g2o::Sim3, shot_vertex, shot_vertex>):
  - Measurement is Sim3_21 (relative pose from camera2 coordinate system to camera1 coordinate system)
    - Note that if edges are added between pre-optimized keyframe and not-pre-optimized keyframe, relative pose to be set is prior optimized one (see graph_optimizer::optimize – 3 of 7)
  - Information matrix is identity matrix (no use of weighting)
  - Residual error is logarithm map of $Sim3_{21}Sim3_{1w}Sim3_{w2}$
    - $Sim3_{21}Sim3_{1w}Sim3_{w2} = I$ if no error
    - $E = Log(Sim3_{21}Sim3_{1w}Sim3_{w2}) - Log(I) = Log(Sim3_{21}Sim3_{1w}Sim3_{w2}) \because Log(I) = \mathbf{0}$
    - logarithm map implementation is there: g2o/sim3.h at master · RainerKuemmerle/g2o · GitHub
  - linearizeOplus method is not overwritten (maybe numerical diffferentiation is used)

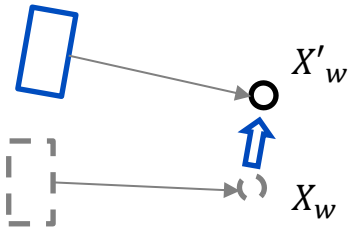




graph_optimizer::optimize – 7 of 7

- Do optimization
  - Call optimizer.optimize once
  - Do not apply χ^2 test
- After optimization
  - Optimization result is applied to:
    - All keyframe poses
  - All landmark positions are also optimized with the following keyframe
    - with their reference keyframe basically
    - with their ref_keyfrm_id_in_loop_fusion_ if they were optimized in correct_covisibility_landmarks – see correct_covisibility_landmarks





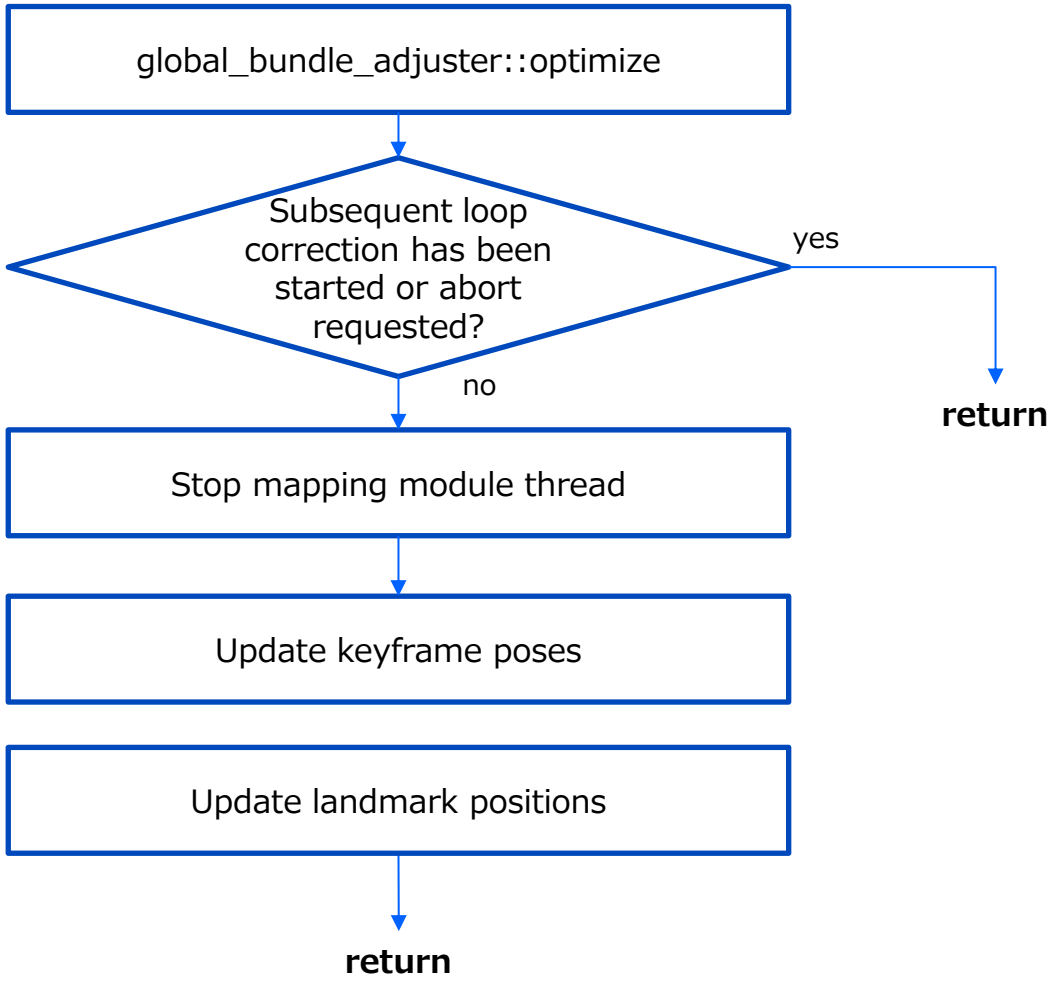
$$X'_w = Sim3_wc_after_correction \cdot Sim3_cw_before_correction \cdot X_w$$

After graph_optimizer::optimize

- Kick loop bundle adjustment thread
  - It will take long time for the thread to finish global bundle adjustment
  - Do not wait for its completion
- Resume mapping module thread
- Set loop_corrector::prev_loop_correct_keyfrm_id to the new keyframe
  - Just for next loop detection
    - See `Loop correction has been done within past 10 frames?` in loop_detector::detect_loop_candidates – 1 of 2

GLOBAL_OPTIMIZATION – LOOP_BUNDLE_ADJUSTER::OPTIMIZE

loop_bundle_adjuster::optimize



global_bundle_adjuster::optimize – 1 of 5

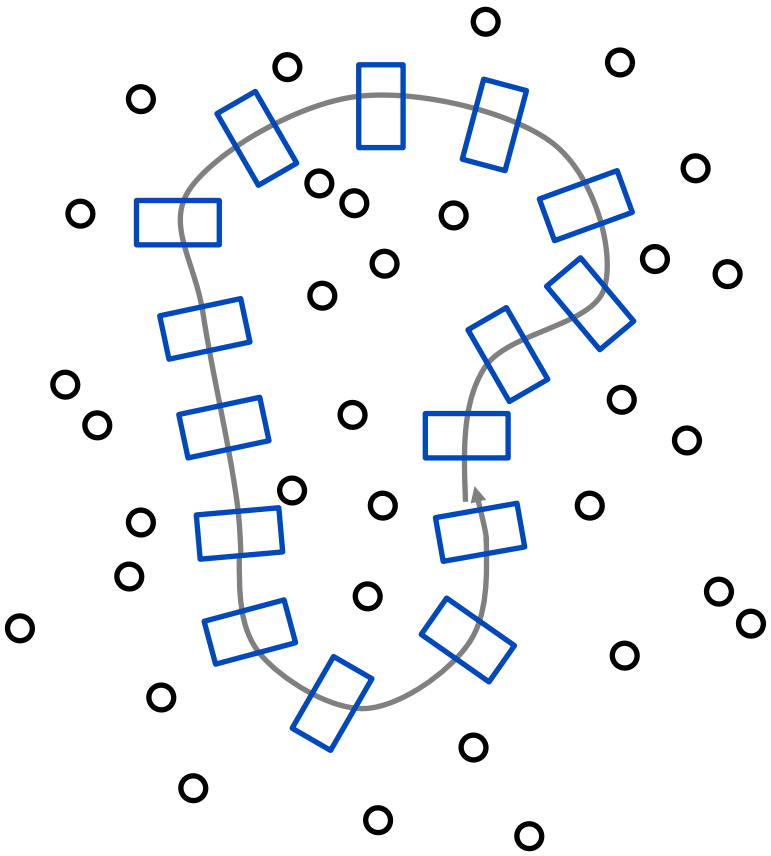
- Optimize with g2o
  - Linear solver = LinearSolverCSparse
  - Block solver = BlockSolver_6_3
  - Algorithm = OptimizationAlgorithmLevenberg
  - Optimizer = SparseOptimizer
- Vertices = all keyframe poses and all landmark positions
- Edges = restriction between keyframes and reprojected points of landmarks
  - No robust kernel is set
- Similar to local bundle adjustment – see local_bundle_adjuster::optimize – 1 of 9 to local_bundle_adjuster::optimize – 9 of 9

global_bundle_adjuster::optimize – 2 of 5

□ = all keyframes ⇒ **added as vertices with setFixed(false) [if keyframe of id=0, setFixed(true)]**

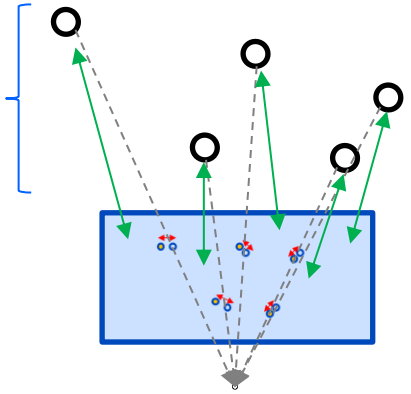
○ = all landmarks ⇒ **added as vertices**

edge



All keyframe poses and all landmark positions will be optimized through g2o optimization.
The optimization will be done to minimize total reprojection errors of all keyframes.

All landmarks observed from a keyframe



For each keyframe of all keyframes, edges between the keyframe and all landmarks which it observes are added.

= edge

= keypoint

= reprojected point of landmark

= reprojection error

Note that reprojection error is considered both on the left and right images when stereo visible as in pose_optimizer::optimize – 3 of 9.

global_bundle_adjuster::optimize – 3 of 5

- Vertices details:
  - keyframe vertex (g2o::se3::shot_vertex):
    - Derived class of ::g2o::BaseVertex<6, ::g2o::SE3Quat>
    - Keyframe poses (cam_pose_cw) are set through setEstimate at first
    - oplusImpl does setEstimate with $\exp(\Delta) * \text{estimate}()$
  - landmark vertex (g2o::landmark_vertex):
    - Derived class of ::g2o::BaseVertex<3, Vec3_t>
    - Positions (pos_w) are set through setEstimate at first
    - oplusImpl does setEstimate with _estimate + Δ
- Same as in local bundle adjustment (local_bundle_adjuster::optimize – 4 of 9)

global_bundle_adjuster::optimize – 4 of 5

- Edges details:
  - stereo_perspective_reproj_edge and mono_perspective_reproj_edge
  - Same as in local bundle adjustment
  - See local_bundle_adjuster::optimize – 5 of 9 - local_bundle_adjuster::optimize – 8 of 9

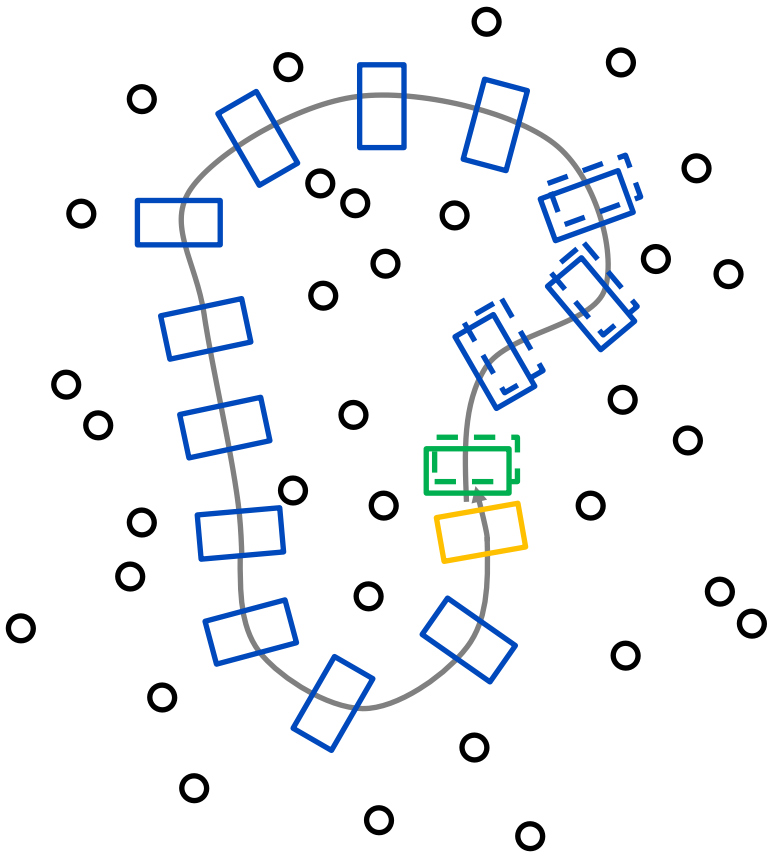
global_bundle_adjuster::optimize – 5 of 5

- Do optimization
  - Call optimizer.optimize once
  - Do not apply χ^2 test
- After opitmization
  - If the new keyframe id = 0:
    - Optimized poses and positions are applied to keyframes and landmarks
    - This is the case at initialization
  - Else:
    - Record optimized poses and positions of each of keyframes and landmarks respectively
      - Optimized pose to keyfrm->cam_pose_cw_after_loop_BA_ for each keyframe
      - Optimized position to lm->pos_w_after_global_BA_ for each landmark
    - Optimized result will be applied after global_bundle_adjuster::optimize is finished with exclusive lock obtained

After global_bundle_adjuster::optimize

- If subsequent loop correction has been started or aborting loop BA is requested, do not apply the optimization result and soon return
- else stop mapping module for exclusive access and then apply the optimization result

Update keyframe poses

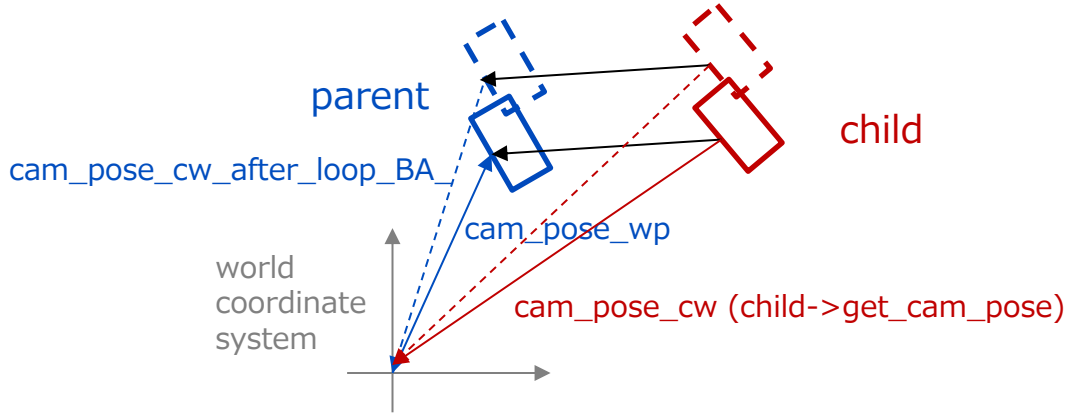


Align keyframe poses with the optimized result along with spanning tree:

- Starting from origin keyframe
- Proceeding to children
- Ending with a parent who has no child (= the new keyframe)

If a keyframe that was not optimized is found, also optimize its pose with using its parent's optimized pose as in figure below:

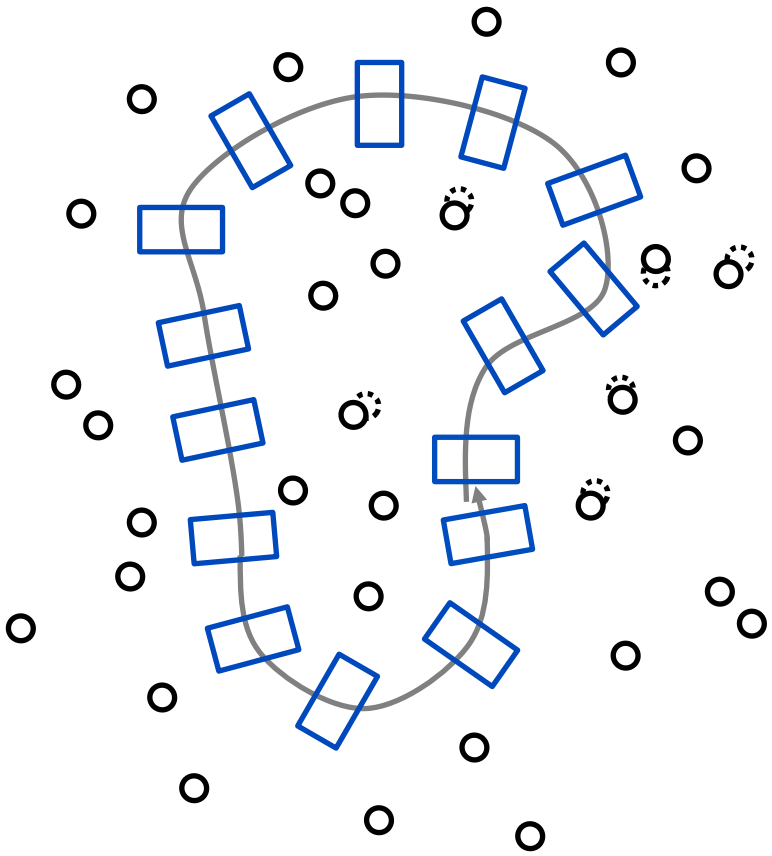
- Basically, all keyframes have been optimized
- A newly created keyframe after the optimization is the case



= · = cam_pose_cw·cam_pose_wp

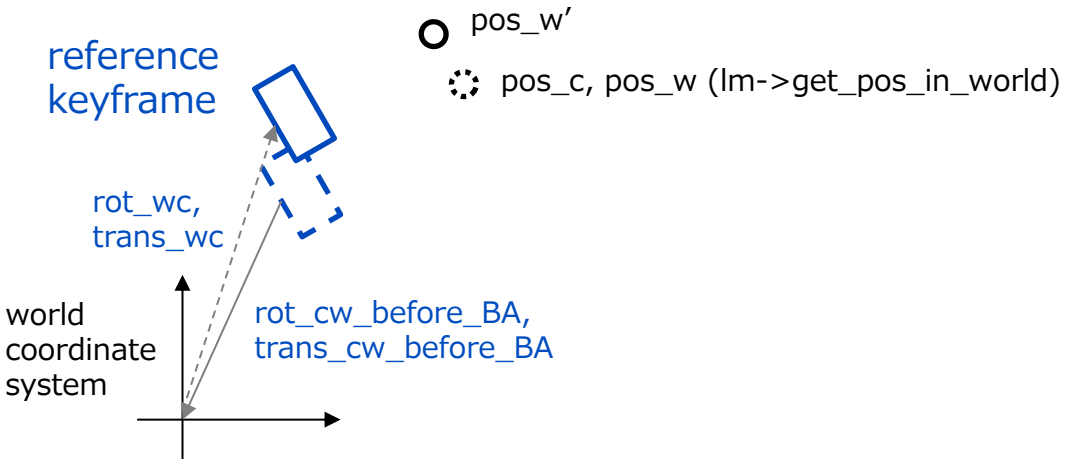
optimized pose = = · = cam_pose_cw·cam_pose_wp·cam_pose_cw_after_loop_BA_

Update landmark positions



Align all landmark positions in the map database.
If a landmark that was not optimized is found, also optimize its position with using its reference keyframe pose as in figure below:

- Basically, all landmarks have been optimized
- A newly created landmark after the optimization is the case

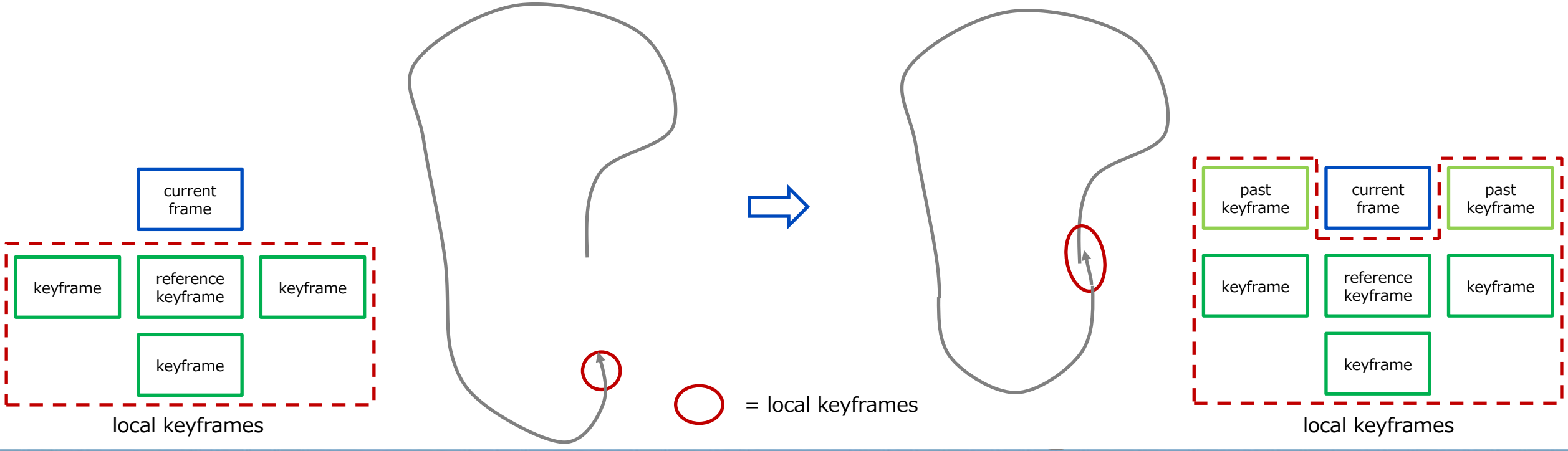


pos_c = rot_cw_before_BA·pos_w + trans_cw_before_BA
optimized position = pos_w' = rot_wc·pos_c + trans_wc

GLOBAL_OPTIMIZATION – THOUGHT EXPERIMENT

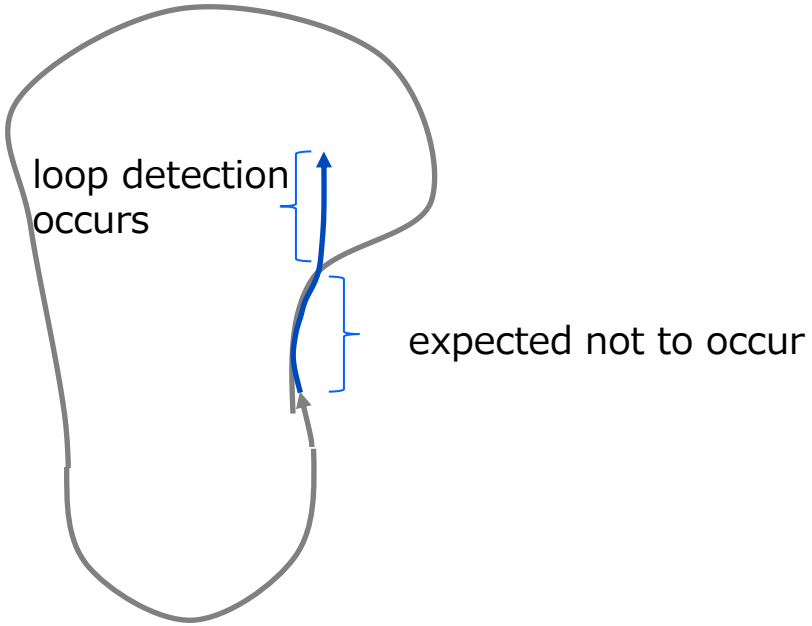
Past keyframes are reused as reference keyframe after loop detection?

- Maybe yes
  - reference keyframe of the current frame comes to have connections with past keyframes
  - It is possible for past keyframes to become reference keyframe of current frame if they are local keyframes (see update_local_map – 2 of 3)
  - whether past keyframes are reused or a new keyframe is created may depend on algorithm in new_keyframe_is_needed (see new_keyframe_is_needed)



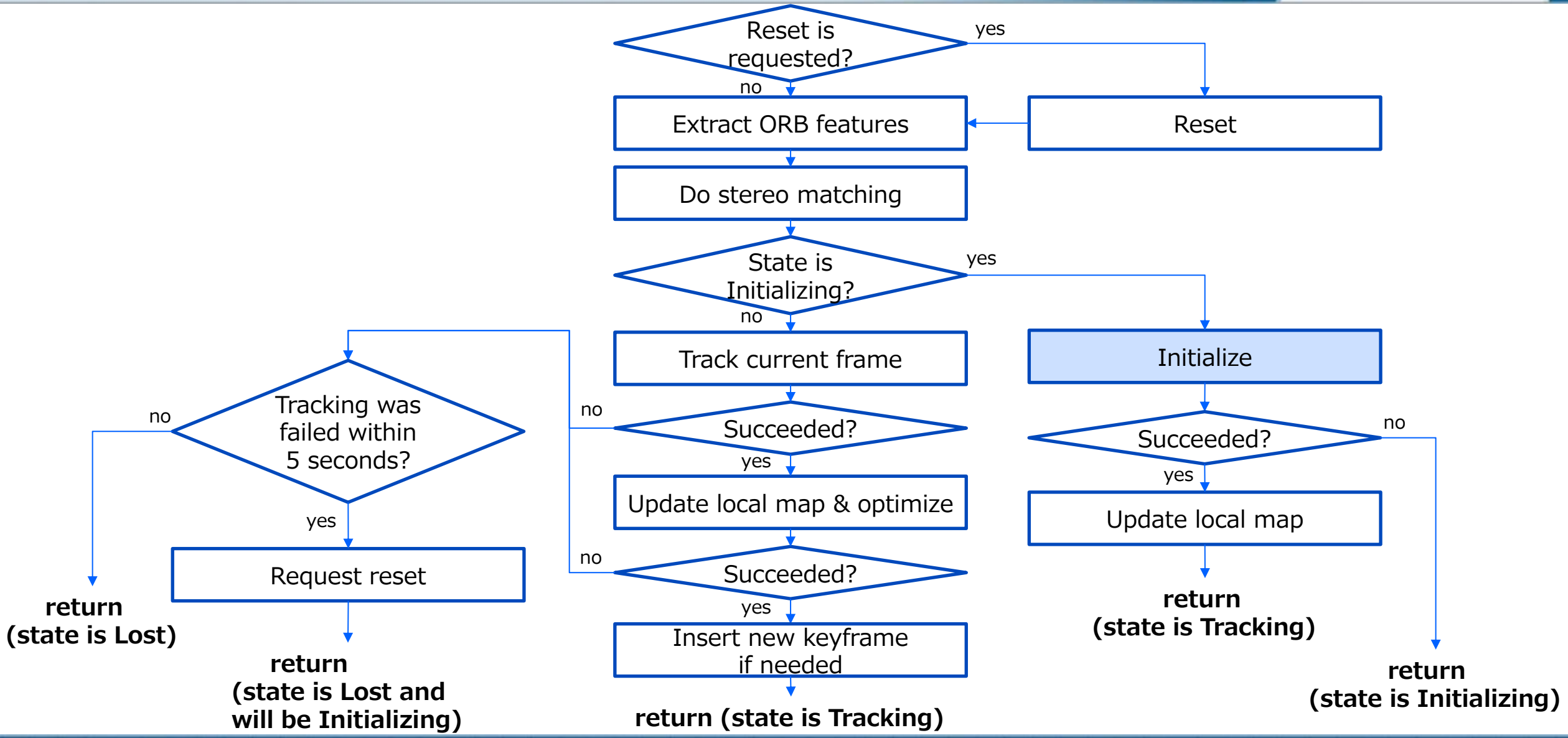
Loop detection occurs continuously?

- As long as no keyframe is inserted, loop detection will not occur
  - If past keyframes are reused well as reference keyframe, loop detection will be suppressed



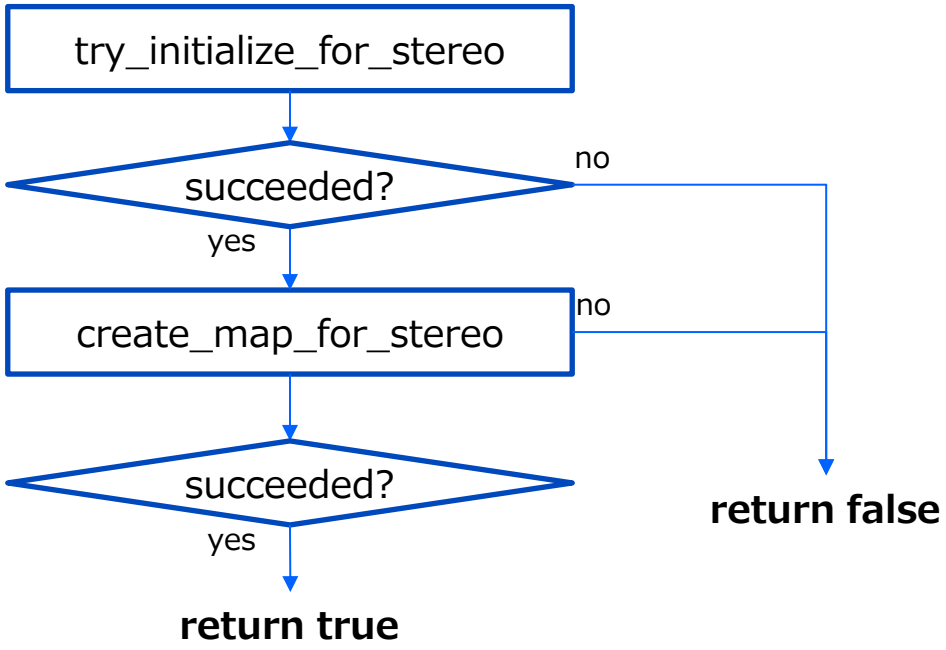
INITIALIZATION

When initialized?



initializer::initialize

Succeeded if the number of detected keypoints is greater or equal to a threshold (min_num_triangulated_ in yaml)



create_map_for_stereo

- Set the current frame pose with Mat44_t::Identity
- Create a new keyframe with using the current frame
  - The new keyframe pose is also Mat44_t::Identity
- with the new keyframe:
  - compute_bow (store_new_keyframe – 1 of 7)
  - add_keyframe to map_database (store_new_keyframe – 7 of 7)
  - set map_database::origin_keyfrm_
- For each keypoint detected in the current frame:
  - Create a landmark using frame::triangulate_stereo (insert_new_keyframe – 2 of 6)
  - add_observation to the landmark with the new keyframe
  - add_landmark to the new keyframe
  - compute_descriptor of the landmark (insert_new_keyframe – 5 of 6)
  - update_normal_and_depth for the landmark (insert_new_keyframe – 6 of 6)
  - add the landmark to the current frame
  - add_landmark to map_database

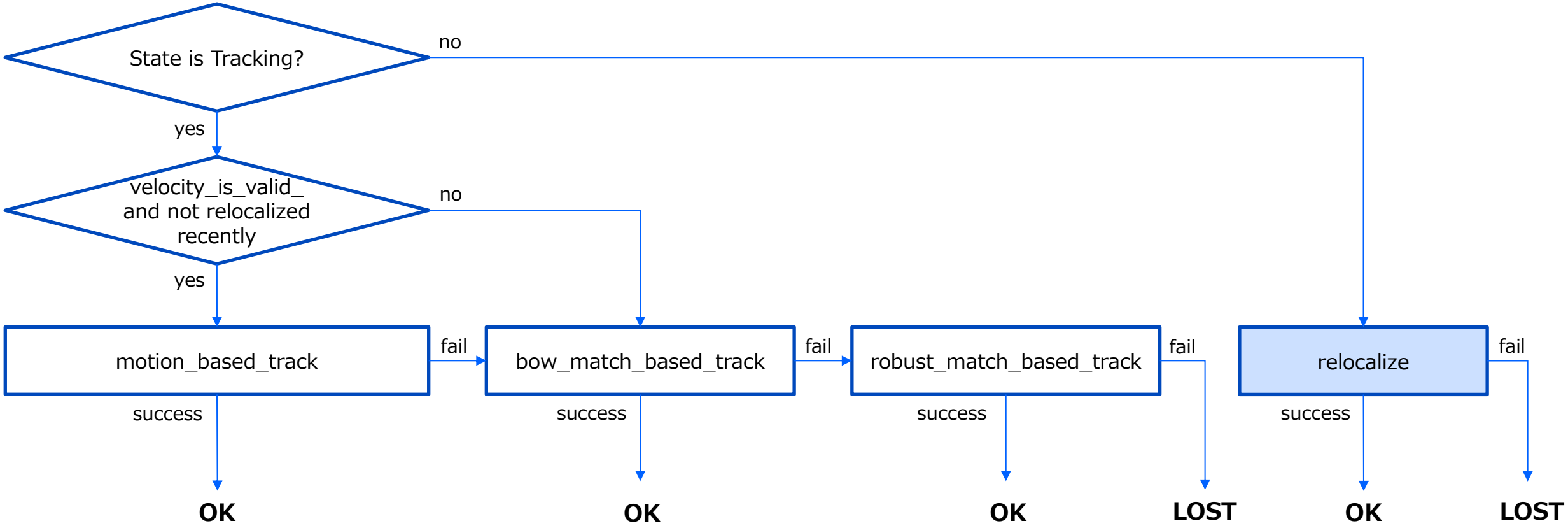
update_local_map

- Already explained
  - See update_local_map – 1 of 3, update_local_map – 2 of 3 and update_local_map – 3 of 3
- When called from initialization, just add the new keyframe and its landmarks to local keyframes and local landmarks vector of the tracking module respectively

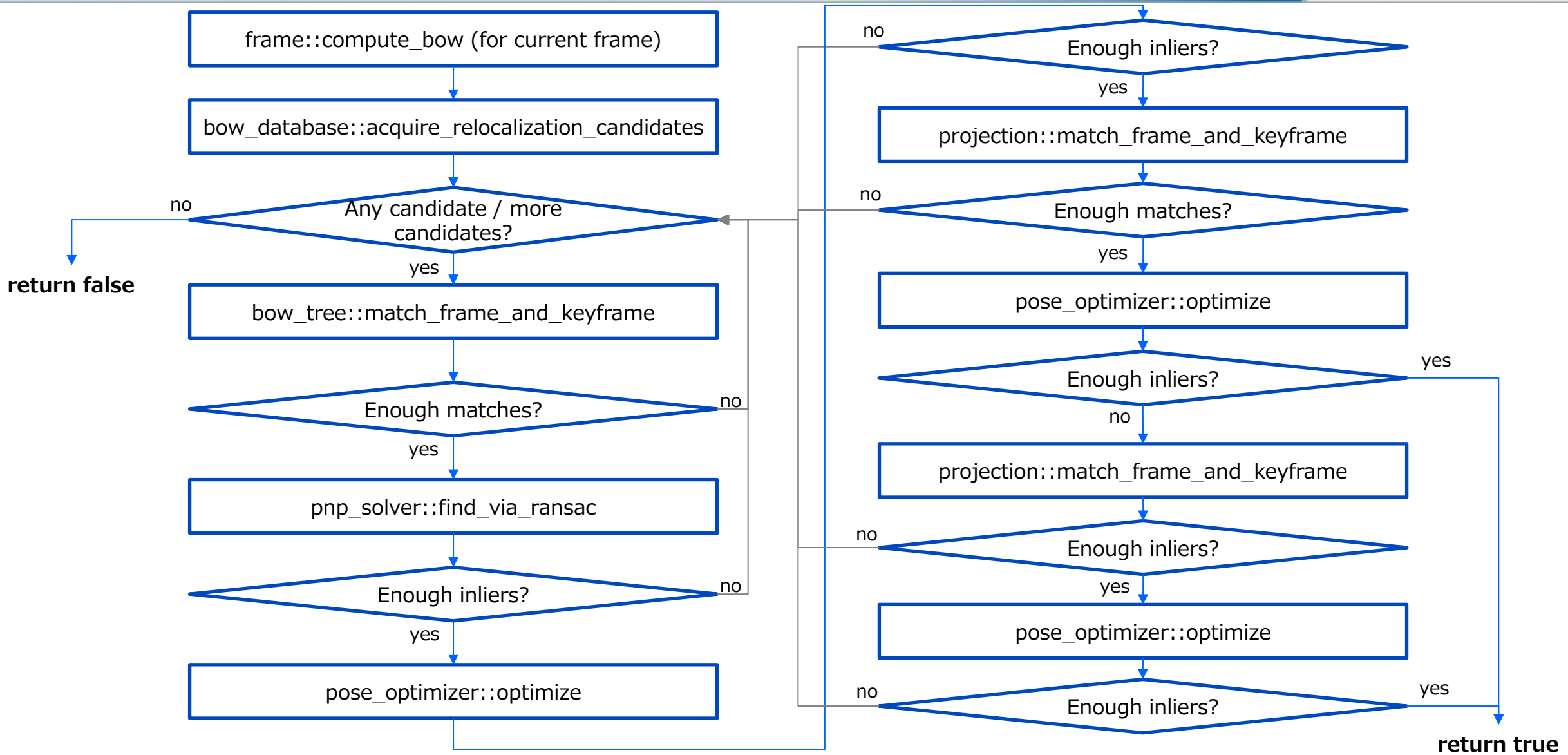
RELOCALIZATION

When relocalized?

track_current_frame
See Tracking overview about when track_current_frame is called

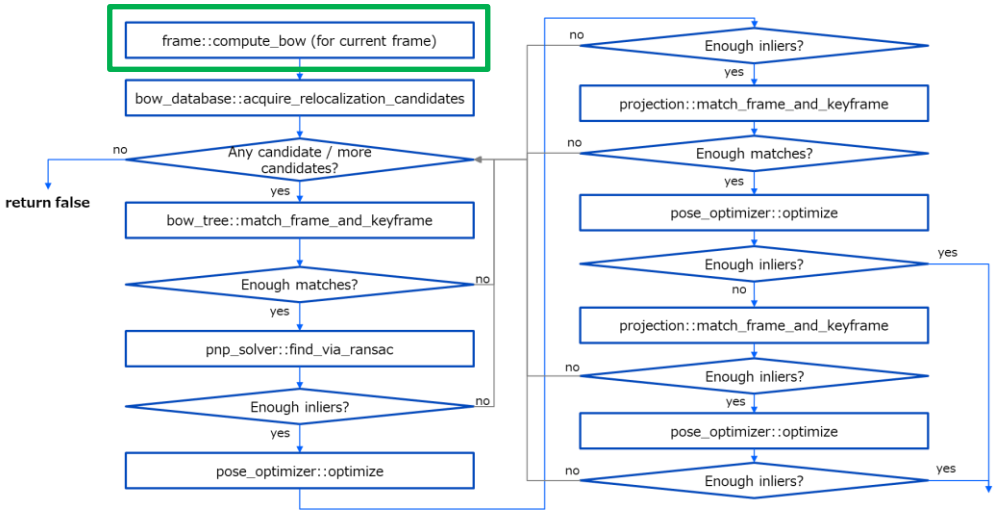


relocalizer::relocalize



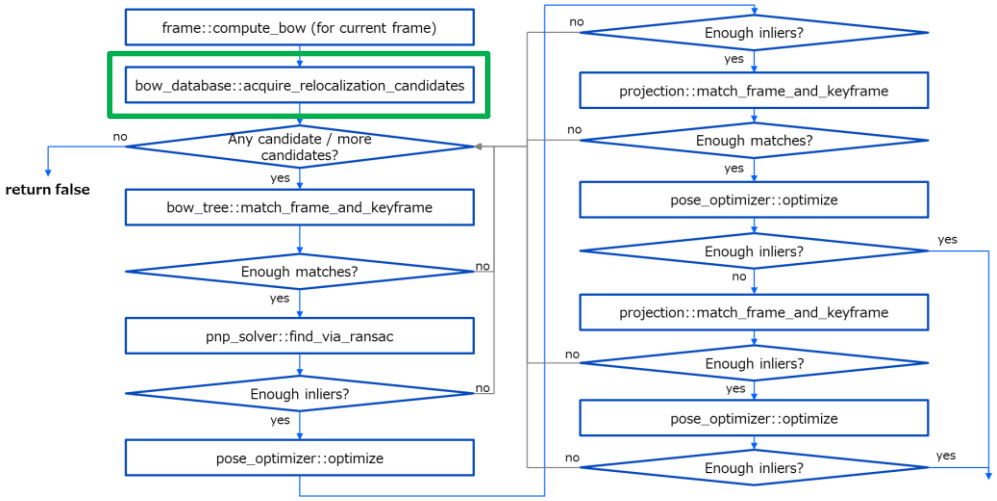
frame::compute_bow

- Compute bow of the current frame using DBoW2/FBoW
  - See store_new_keyframe – 1 of 7



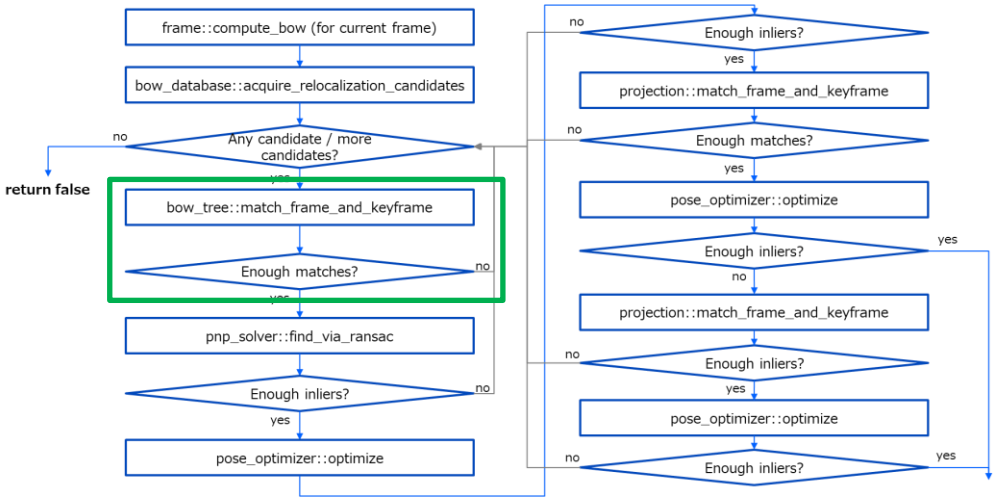
bow_database::acquire_relocalization_candidates

- Very similar to bow_database::acquire_loop_candidates
  - See bow_database::acquire_loop_candidates – 1 of 6 - bow_database::acquire_loop_candidates – 6 of 6
- Difference is argument of set_candidates_sharing_words
  - keyfrms_to_reject is empty when called from acquire_relocalization_candidates because the current frame pose is lost
  - keyfrms_to_reject is used when called from acquire_loop_candidates to reject keyframes near the current frame



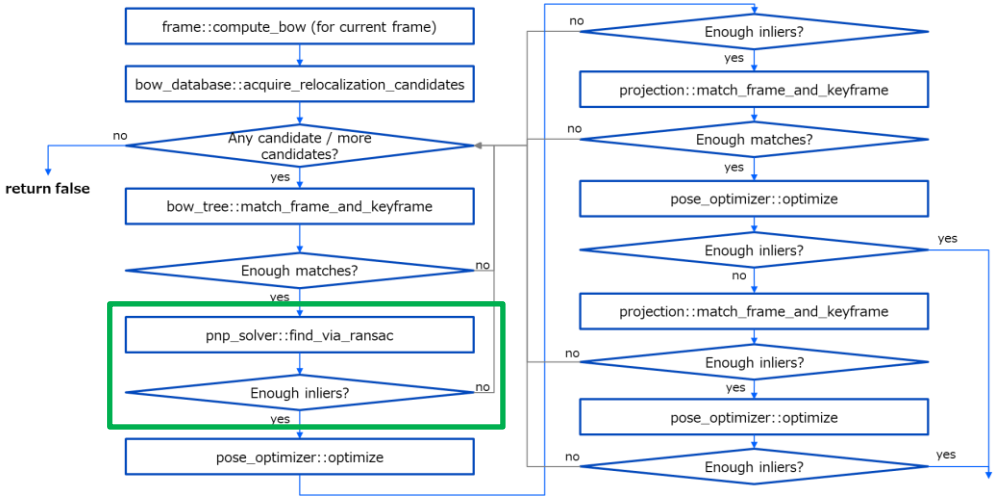
bow_tree::match_frame_and_keyframe

- Bow based match between the current frame and the candidate keyframe
  - Already explained in bow_match_based_track – 3 of 4
- If the number of matched keypoints is less than min_num_bow_matches (=20), regarded as failed



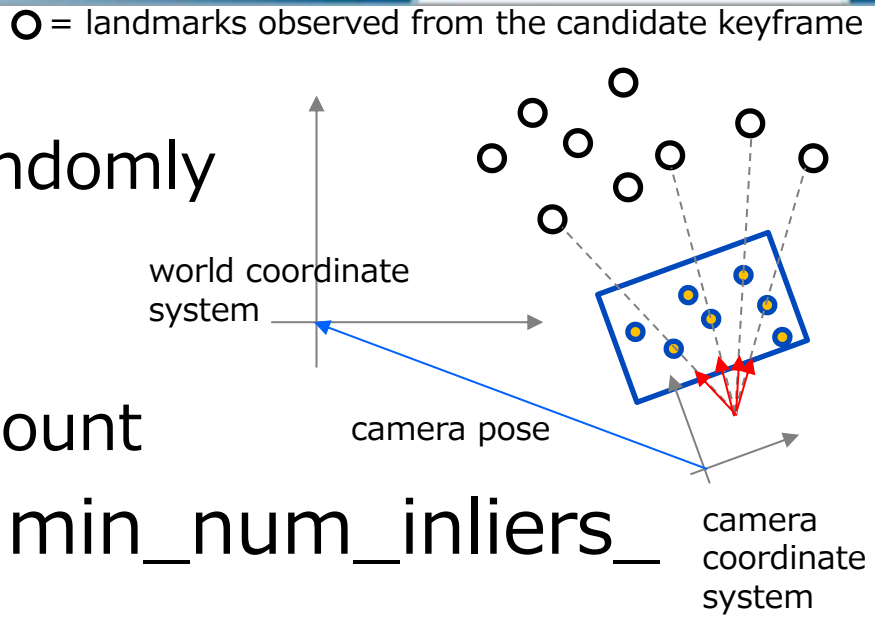
pnp_solver::find_via_ransac – 1 of 2

- PnP solver implementation based on Efficient PnP
  - See https://github.com/cvlab-epfl/EPnP
  - Obviously solve/pnp_solver.cc is based on https://github.com/cvlab-epfl/EPnP/blob/master/cpp/epnp.cpp



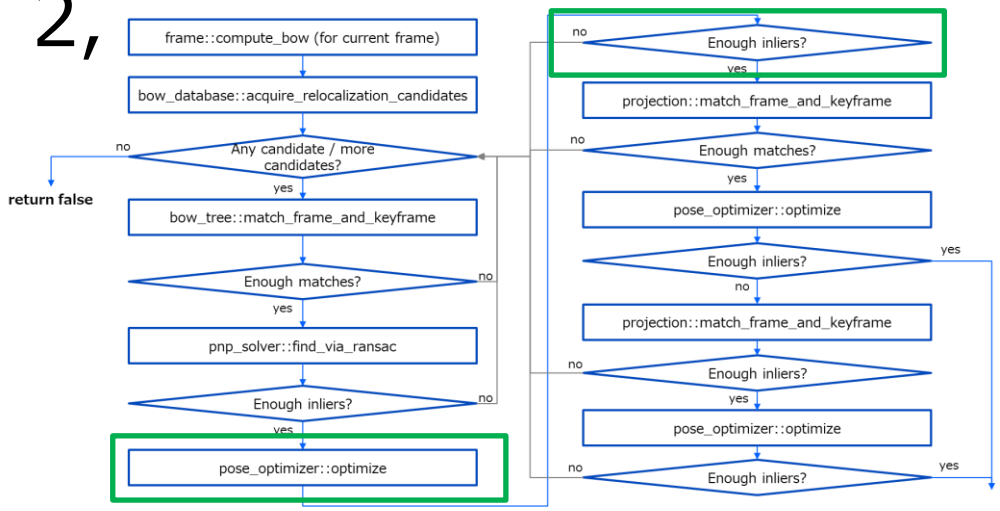
pnp_solver::find_via_ransac – 2 of 2

- For 200 loops:
  - Select 4 landmark and bearing vector pairs randomly
  - compute_pose
    - Estimate the current camera pose using EPnP
  - Record the best matched pose and its inliers count
- If the best inliers count is less or equal to min_num_inliers_ (=10):
  - return (with failed)
- Else:
  - compute_pose with all inlier points
  - return (with succeeded)



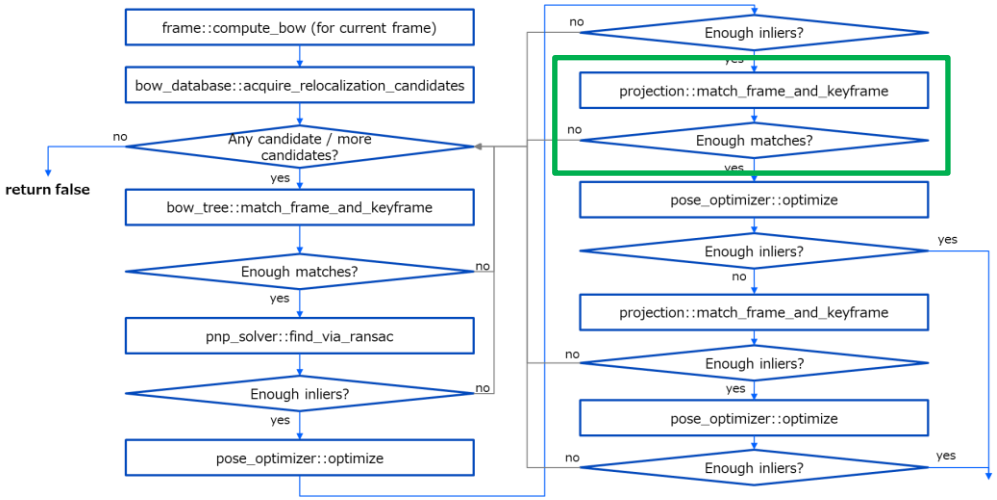
pose_optimizer::optimize

- Optimize the current pose with using landmarks
  - Already explained in pose_optimizer::optimize – 1 of 9 - pose_optimizer::optimize – 9 of 9
- Before calling pose_optimizer::optimize, the current frame pose is set with the pose obtained by pnp_solover::find_via_ransac
- After pose_optimizer::optimize, if the number of inliers is less than min_num_bow_matches (=20) / 2, regarded as failed

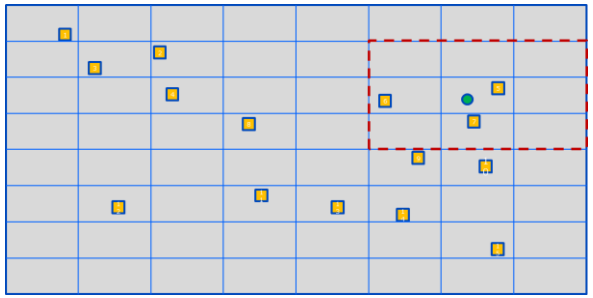


projection::match_frame_and_keyframe

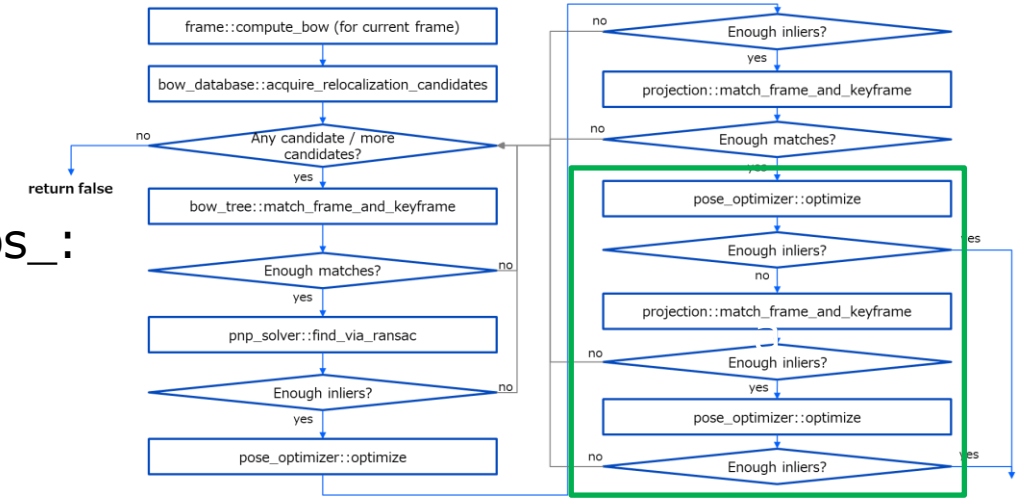
Try to correspond landmarks between the current frame and the candidate keyframe. (similar to projection::match_current_and_last_frames)



- Reproject the candidate keyframe's landmark to the current frame
  - See projection::match_current_and_last_frames – 3 of 7 about reprojection.
- Skip matching if:
  - cam_to_lm_dist is above max_valid_dist_
  - cam_to_lm_dist is below min_valid_dist_
  - See insert_new_keyframe – 6 of 6 about max_valid_dist_ and min_valid_list_.
- Obtain keypoints within grid cells around the reprojected point
  - About grid cell, see assign_keypoints_to_grid.
- find the best matched keypoint among them by Hamming distance:
  - Hamming distance must be the smallest among the keypoint candidates and below hamm_dist_thr (=64)
- At last, do angle check explained in projection::match_current_and_last_frames – 6 of 7
- The best matched keypoint is registered to the current frame



- If the number of correspondences are less than 50, regarded as failed

pose_optimizer::optimize again

- The number of correspondences may have increased because of projection::match_frame_and_keyframe
- It is expected that more accurate pose of the current frame is obtained by calling pose_optimizer::optimize again
- If inliers count is above or equal to min_num_valid_obs_ (=50):
  - Succeeded
- Else:
  - projection::match_frame_and_keyframe again
    - More correspondences may be found because of the previous pose_optimizer::optimize call
  - pose_optimizer::optimize again
  - If inliers count is above or equal to min_num_valid_obs_:
    - Succeeded
  - Else:
    - Failed