LSD-slam and ORB-slam2, a literature based explanation

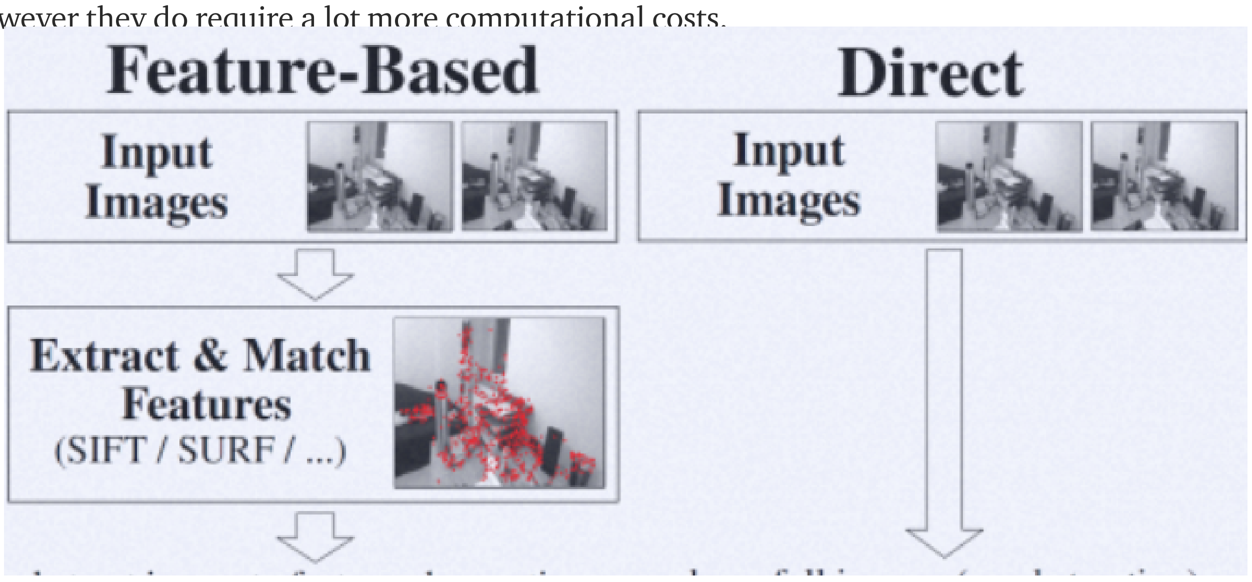
Jeroen Zijlmans Follow
Aug 23, 2017 · 16 min read

Monocular slam

Both algorithms are monocular slam algorithms, where slam means Simultaneous Localization And Mapping, and monocular means that they preform slam based on a rgb image sequence (video) created by 1 camera at each time-instance.
Slam algorithms are algorithms that simultaneously tracks the movement of the camera (usually mounted onto a robot/car/etc.) and create a point cloud map of the surroundings that they passed. They create a map of the surroundings and localize them self within this map.
Monocular slam has has one big characteristic which provides it with a big pro but also a big con, it is scale independent. It cannot estimate the scale of the scenery and thus the precieved scale of the scenery will drift. This often is attempted to be fixed by trying to detect scenery that you already have been (you have traveled in a loop) and then the scale-drift can be estimated and corrected. This does bring the big pro that the algorithms work for big outdoor sceneries, small indoor sceneries and for transitions between these two.

Feature-based vs. Direct

Monocular slam algorithms can be divided into two groups, those who use feature-based methods and those who use direct methods:
Feature-based slam algorithms take the images and within these images, they search for certain features, key-points, (for instance corners) and only use these features to estimate the location and surroundings. This means that they throw away a lot of positional valuable information from the image, but this does simplifies the whole process.
Direct slam algorithms do not search the image for key-points but instead use the image intensities to estimate the location and surroundings. This does mean that they use more information from the images and thus tend to be robuster and create a more detailed map of the surrounding. However they do require a lot more computational costs.



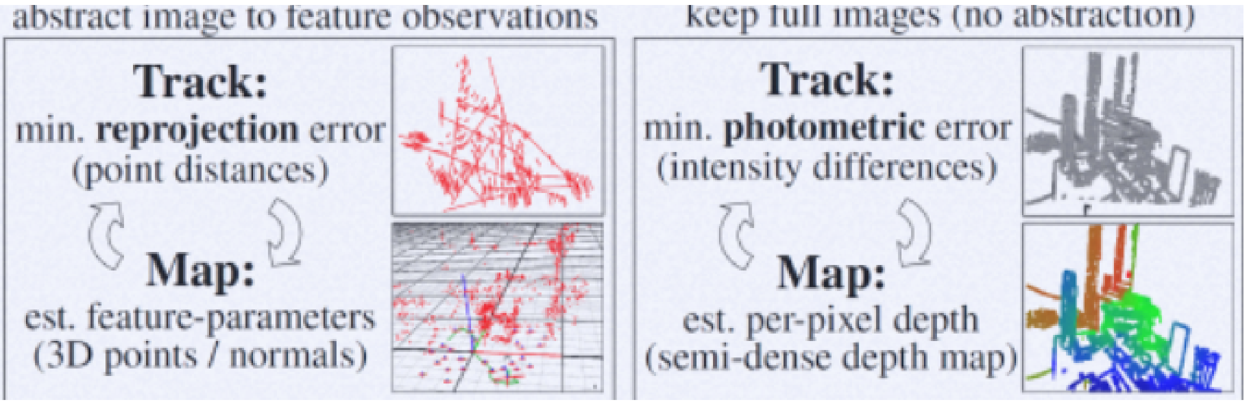


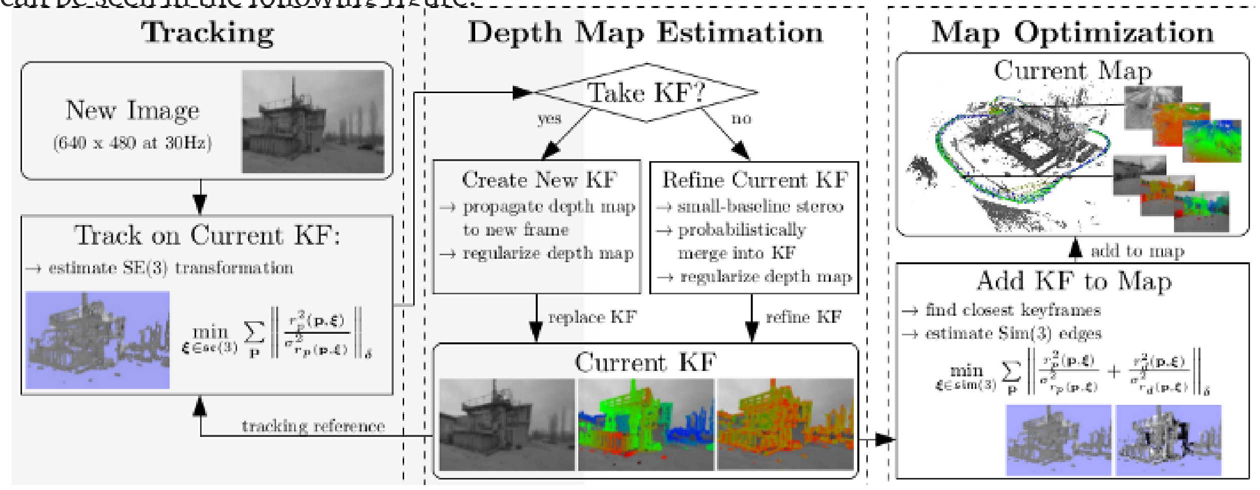
Image showing the difference between Feature-based and Direct slam algorithms.

There are many slam algorithms, many also use other data then Monocular rgb data, such as stereo images (two images taken with two camera's at the same time instance) or rgb-d images (rgb with depth information). Each slam algorithm has its own characteristics and pro's and con's. Here I will discuss two popular slam algorithms: LSD-slam and ORB-slam2, and compare them to each other.

LSD-slam

based on paper: https://vision.in.tum.de/_media/spezial/bib/engel14eccv.pdf

LSD-slam stands for Large-Scale Direct slam and is a monocular slam algorithm. The map of the surrounding is created based on certain key-frames which contain a camera image, an inverse depth map and the variance of the inverse depth map. The depth map and its variance are not created for all the pixels, but only for those in the neighborhood of large image intensity gradients, making them semi-dense. These key-frames are updated with the images that aren't key-frames. They divide the problem into three parts: tracking, depth map estimation and map optimization, as can be seen in the following figure:



Tracking

The tracking part takes the new image and tries to estimate the current camera pose with respect to the current key-frame pose by minimizing the variance-normalized photometric error (for exact formulas, refer to the paper). It uses the previous image pose as initialization.

Depth Map Estimation

Here is first decided if the image is going to be used as a new key-frame, or to refine the current key-frame. This is done based on two weights, the relative distance to the current key-frame and the angle to the current key-frame. Is the weighted sum of these two larger than a certain threshold, a new key-frame is taken. This threshold is relative to the scale of the scene as the key-frames are scaled such that the mean inverse depth is one.

If the image is to become a new key-frame, a depth-map of this key-frame needs to be initialized, this is done by projected points from the previous key-frame onto the new key-frame. Then one iteration of spatial regularization and outlier removal is done. (For more information on this, see section 2.3 of www.cv-foundation.org/openaccess/content_iccv_2013/papers/Engel_Semi-dense_Visual_Odometry_2013_ICCV_paper.pdf) The key-frame is finally scaled such that the mean inverse depth is one.
If the image is not used for a new key-frame, it is used to update the depth-map on the current key-frame. This is done by preforming a lot of small-baseline stereo comparisons for the parts of the image where the expected accuracy is large enough and then incorporating this information with the information on the current key-frame.

Map Optimization

Because each key-frame is scaled to have an mean inverse depth of one, each key-frame has a different scale. The key-frames need to be aligned with this scale difference in mind. This is done by minimizing an error function based on the photometric residual and the depth residual, both scaled with their variance, of both images.
After the key-frame is added to the map, a number of key-frames that are close enough to detect a loop (loop-closure) are used to detect loop-closure. This is done by a reciprocal tracking check, which means that the transformation from an original key-frame to the other key-frame is compared with the transformation form the other key-frame to the original key-frame while also taking their uncertainties into account. If these to are similar enough, the algorithm has found a loop-closure.
The map is optimized using pose graph optimization from the g2o package.

Parameters

Parameters change-able from the commandline (see there readme for explanation):

```
bool autoRun = true;
bool autoRunWithinFrame = true;

int debugDisplay = 0;

bool onSceenInfoDisplay = true;
bool displayDepthMap = true;
bool dumpMap = false;
bool doFullReConstraintTrack = false;

// dyn config
bool printPropagationStatistics = false;
bool printFillHolesStatistics = false;
bool printObserveStatistics = false;
bool printObservePurgeStatistics = false;
bool printRegularizeStatistics = false;
bool printLineStereoStatistics = false;
bool printLineStereoFails = false;

bool printTrackingIterationInfo = false;

bool printFrameBuildDebugInfo = false;
bool printMemoryDebugInfo = false;

bool printKeyframeSelectionInfo = false;
bool printConstraintSearchInfo = false;
```

```
    bool printOptimizationInfo = false;
    bool printRelocalizationInfo = false;

    bool printThreadingInfo = false;
    bool printMappingTiming = false;
    bool printOverallTiming = false;

    bool plotTrackingIterationInfo = false;
    bool plotSim3TrackingIterationInfo = false;
    bool plotStereoImages = false;
    bool plotTracking = false;


    float freeDebugParam1 = 1;
    float freeDebugParam2 = 1;
    float freeDebugParam3 = 1;
    float freeDebugParam4 = 1;
    float freeDebugParam5 = 1;

    float KFDistWeight = 4;
    float KFUsageWeight = 3;

    float minUseGrad = 5;
    float cameraPixelNoise2 = 4*4;
    float depthSmoothingFactor = 1;

    bool allowNegativeIdepths = true;
    bool useMotionModel = false;
    bool useSubpixelStereo = true;
    bool multiThreading = true;
    bool useAffineLightningEstimation = true;

    bool useFabMap = false;
    bool doSlam = true;
    bool doKFReActivation = true;
    bool doMapping = true;

    int maxLoopClosureCandidates = 10;
    int maxOptimizationIterations = 100;
    int propagateKeyFrameDepthCount = 0;
    float loopclosureStrictness = 1.5;
    float relocalizationTH = 0.7;


    bool saveKeyframes =  false;
    bool saveAllTracked =  false;
    bool saveLoopClosureImages =  false;
    bool saveAllTrackingStages = false;
    bool saveAllTrackingStagesInternal = false;

    bool continuousPCOutput = false;


    bool fullResetRequested = false;
    bool manualTrackingLossIndicated = false;
```

Key presses:

```
case 'a': case 'A':
//        autoRun = !autoRun;        // disabled... only use for
debugging & if you really, really know what you are doing
        break;
    case 's': case 'S':
//        autoRunWithinFrame = !autoRunWithinFrame;     //
disabled... only use for debugging & if you really, really know what
you are doing
        break;
    case 'd': case 'D':
        debugDisplay = (debugDisplay+1)%6;
        printf("debugDisplay is now: %d\n", debugDisplay);
        break;
    case 'e': case 'E':
        debugDisplay = (debugDisplay-1+6)%6;
        printf("debugDisplay is now: %d\n", debugDisplay);
        break;
    case 'o': case 'O':
        onSceenInfoDisplay = !onSceenInfoDisplay;
        break;
    case 'r': case 'R':
        printf("requested full reset!\n");
        fullResetRequested = true;
        break;
    case 'm': case 'M':
        printf("Dumping Map!\n");
        dumpMap = true;
        break;
    case 'p': case 'P':
        printf("Tracking all Map-Frames again!\n");
        doFullReConstraintTrack = true;
        break;
    case 'l': case 'L':
        printf("Manual Tracking Loss Indicated!\n");
        manualTrackingLossIndicated = true;
        break;
```

Parameters from settings.h:

```
// validity can take values between 0 and X, where X depends on the
abs. gradient at that location:
// it is calculated as VALIDITY_COUNTER_MAX +
(absGrad/255)*VALIDITY_COUNTER_MAX_VARIABLE
#define VALIDITY_COUNTER_MAX (5.0f)        // validity will never be
higher than this
#define VALIDITY_COUNTER_MAX_VARIABLE (250.0f)        // validity
will never be higher than this

#define VALIDITY_COUNTER_INC 5        // validity is increased by
this on sucessfull stereo
#define VALIDITY_COUNTER_DEC 5        // validity is decreased by
this on failed stereo
#define VALIDITY_COUNTER_INITIAL_OBSERVE 5    // initial validity
for first observations


#define VAL_SUM_MIN_FOR_CREATE (30) // minimal summed validity over
```

```
5x5 region to create a new hypothesis for non-blacklisted pixel
(hole-filling)
#define VAL_SUM_MIN_FOR_KEEP (24) // minimal summed validity over
5x5 region to keep hypothesis (regularization)
#define VAL_SUM_MIN_FOR_UNBLACKLIST (100) // if summed validity
surpasses this, a pixel is un-blacklisted.

#define MIN_BLACKLIST -1    // if blacklist is SMALLER than this,
pixel gets ignored. blacklist starts with 0.




/** ============== Depth Variance Handeling =======================
*/
#define SUCC_VAR_INC_FAC (1.01f) // before an ekf-update, the
variance is increased by this factor.
#define FAIL_VAR_INC_FAC 1.1f // after a failed stereo observation,
the variance is increased by this factor.
#define MAX_VAR (0.5f*0.5f) // initial variance on creation - if
variance becomes larter than this, hypothesis is removed.

#define VAR_GT_INIT_INITIAL 0.01f*0.01f    // initial variance vor
Ground Truth Initialization
#define VAR_RANDOM_INIT_INITIAL (0.5f*MAX_VAR)    // initial
variance vor Random Initialization





// Whether to use the gradients of source and target frame for
tracking,
// or only the target frame gradient
#define USE_ESM_TRACKING 1


#ifdef ANDROID
    // tracking pyramid levels.
    #define MAPPING_THREADS 2
    #define RELOCALIZE_THREADS 4
#else
    // tracking pyramid levels.
    #define MAPPING_THREADS 4
    #define RELOCALIZE_THREADS 6
#endif

#define SE3TRACKING_MIN_LEVEL 1
#define SE3TRACKING_MAX_LEVEL 5

#define SIM3TRACKING_MIN_LEVEL 1
#define SIM3TRACKING_MAX_LEVEL 5

#define QUICK_KF_CHECK_LVL 4

#define PYRAMID_LEVELS (SE3TRACKING_MAX_LEVEL >
SIM3TRACKING_MAX_LEVEL ? SE3TRACKING_MAX_LEVEL :
SIM3TRACKING_MAX_LEVEL)
```

```
// ============== stereo & gradient calculation
======================
#define MIN_DEPTH 0.05f // this is the minimal depth tested for
stereo.

// particularely important for initial pixel.
#define MAX_EPL_LENGTH_CROP 30.0f // maximum length of epl to
search.
#define MIN_EPL_LENGTH_CROP (3.0f) // minimum length of epl to
search.

// this is the distance of the sample points used for the stereo
descriptor.
#define GRADIENT_SAMPLE_DIST 1.0f

// pixel a point needs to be away from border... if too small:
segfaults!
#define SAMPLE_POINT_TO_BORDER 7

// pixels with too big an error are definitely thrown out.
#define MAX_ERROR_STEREO (1300.0f) // maximal photometric error for
stereo to be successful (sum over 5 squared intensity differences)
#define MIN_DISTANCE_ERROR_STEREO (1.5f) // minimal multiplicative
difference to second-best match to not be considered ambiguous.

// defines how large the stereo-search region is. it is [mean] +/-
[std.dev]*STEREO_EPL_VAR_FAC
#define STEREO_EPL_VAR_FAC 2.0f




// ============== Smoothing and regularization
======================
// distance factor for regularization.
// is used as assumed inverse depth variance between neighbouring
pixel.
// basically determines the amount of spacial smoothing (small ->
more smoothing).
#define REG_DIST_VAR
(0.075f*0.075f*depthSmoothingFactor*depthSmoothingFactor)

// define how strict the merge-processes etc. are.
// are multiplied onto the difference, so the larger, the more
restrictive.
#define DIFF_FAC_SMOOTHING (1.0f*1.0f)
#define DIFF_FAC_OBSERVE (1.0f*1.0f)
#define DIFF_FAC_PROP_MERGE (1.0f*1.0f)
#define DIFF_FAC_INCONSISTENT (1.0f * 1.0f)
```

```
// ============== initial stereo pixel selection
======================
#define MIN_EPL_GRAD_SQUARED (2.0f*2.0f)
#define MIN_EPL_LENGTH_SQUARED (1.0f*1.0f)
#define MIN_EPL_ANGLE_SQUARED (0.3f*0.3f)

// abs. grad at that location needs to be larger than this.
#define MIN_ABS_GRAD_CREATE (minUseGrad)
#define MIN_ABS_GRAD_DECREASE (minUseGrad)

// ============== RE-LOCALIZATION, KF-REACTIVATION etc.
======================
// defines the level on which we do the quick tracking-check for
relocalization.



#define MAX_DIFF_CONSTANT (40.0f*40.0f)
#define MAX_DIFF_GRAD_MULT (0.5f*0.5f)

#define MIN_GOODPERGOODBAD_PIXEL (0.5f)
#define MIN_GOODPERALL_PIXEL (0.04f)
#define MIN_GOODPERALL_PIXEL_ABSMIN (0.01f)

#define INITIALIZATION_PHASE_COUNT 5

#define MIN_NUM_MAPPED 5
```

In DepthMap::observeDepthRow():

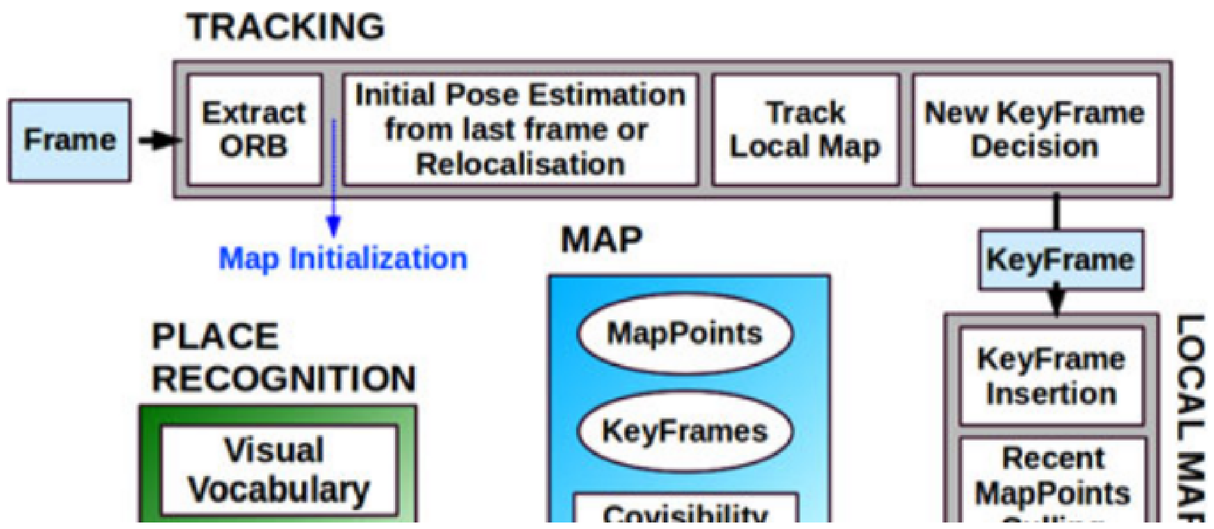
```
The row is being processed for gradients from the 3rd x (column)
till the width of the image-3. Here is also checked if the gradient
is smaller than the MIN_ABS_GRAD_DECREASE. If so, it is skipped,
just as when the gradient is smaller then MIN_ABS_GRAD_CREATE or if
the pixel is blacklisted (when the stereo fails repeatedly on this
pixel.
```

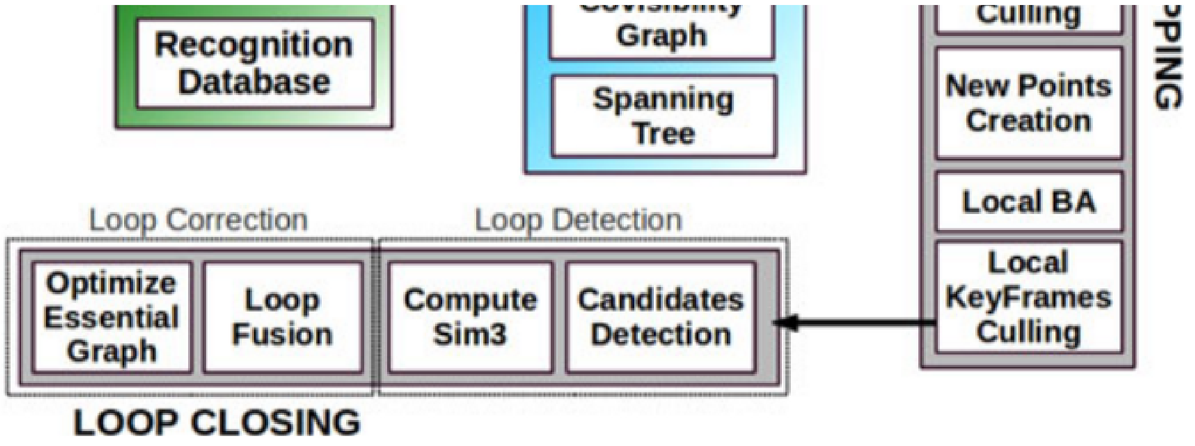
ORB-slam2

based on: http://ieeexplore.ieee.org/document/7219438/?part=1 and

https://arxiv.org/abs/1610.06475
ORB-slam2 is more feature based, and uses ORB features because of the speed in which these can

be extracted from images and there rotational invariance.
An overview of how the algorithm works:





The algorithms works on three threads, a tracking thread, a local mapping thread and a loop closing thread.

Initializing the map

To initialize the map starting by computing the relative pose between two scenes, they compute two geometrical models in parallel, one for a planar scene, a homography and one for non-planar scenes, a fundamental matrix. They then choose one of both based on a relative score of both. Using the selected model they estimate multiple motion hypotheses and en see if one is significantly better then the others, if so, a full bundle adjustment is done, otherwise the initialization starts over.

Tracking

The tracking part localizes the camera and decides when to insert a new keyframe. Features are matched with the previous frame and the pose is optimized using motion-only bundle adjustment. The features extracted are FAST corners. (for res. till 752x480, 1000 corners should be good, for higher (KITTI 1241x376) 2000 corners works). Multiple scale-levels (factor 1.2) are used and each level is divided into a grid in which 5 corners per cell are attempted to be extracted. These FAST corners are then described using ORB. The initial pose is estimated using a constant velocity motion model. If the tracking is lost, the place recognition module kicks in and tries to re-localize itself. When there is an estimation of the pose and feature matches, the co-visibility graph of keyframes, that is maintained by the system, is used to get a local visible map. This local map consists of keyframes that share map point with the current frame, the neighbors of these keyframes and a reference keyframe which share the most map points with the current frame. Through re-projection, matches of the local map are searched on the frame and the camera pose is optimized using these matches. Finally is decided if a new Keyframe needs to be created, new keyframes are inserted very frequently to make tracking more robust. A new keyframe is created when at least 20 frames has passed from the last keyframe, and last global re-localization, the frame tracks at least 50 points of which less then 90% are point from the reference keyframe.

Local mapping

First the new keyframe is inserted into the covisibility graph, the spanning tree linking a keyframe to the keyframe with the most points in common, and a 'bag of words' representation of the keyframe (used for data association for triangulating new points) is created.

New map points are created by triangulating ORB from connected keyframes in the covisibility graph. The unmachted ORB in a keyframe are compared with other unmatched ORB in other keyframes. The match must fulfill the epipolare constraint to be valid. To be a match, the ORB pairs are triangulated and checked if in both frames they have a positive depth, and the parallax,

re projection error and scale consistency is checked. Then the match is projected to other connected keyframes to check if it is also in these.
The new map points first need to go through a test to increase the likelihood of these map points being valid. They need to be found in more than 25 % of the frames in which it is predicted to be visible and it must be observed by at least three keyframes.
Then through local bundle adjustment, the current keyframe, all keyframes connected to it through the co-visibility graph and all the map points seen by these keyframes are optimized using the keyframes that do see the map points but are not connected to the current keyframe.
Finally keyframes that are abundent are discarded to remain a certain simplicity. Keyframes from which more than 90 % of the map points can be seen by three other keyframes in the same scale-level are discarded.

Loop closing

To detect possible loops, they check bag of words vectors of the current keyframe and its neighbors in the covisibitlity graph. The min. simularity of these bag of words vectors is taken as a benchmark and from all the keyframes with a bag of words simulatrity to the current key frame that is greater that this benchmark, all the keyframes that are allready connected to the current keyframe are removed. If three loop canditates that are consistant are detected consecutively, this loop is regarded as a serious candiddate.
For these loops, the similarity transformation is calculated (7DOF, 3 trans, 3 rot, 1 scale) RANSAC itterations are prformed to find them and these are then optimized after which more correspondences are searched and then again an optimization is preformed. If the similarity is supported by having enough inlier's, the loop is accepted.
The current keyframe pose in then adjusted and this is propagated to its neighbors and the corresponding map-points are fused. Finally a pose graph optimization is preformed over the essential graph to take out the loop closure created errors along the graph. This also corrects for scale drift.

Parameters

Parameters from the settingfile:

```
intrinsic camera parameters: fx fy cx cy as K matrix
                             k1 k2 p1 p2 k3 as D matrix

fps: frames per second, default: 30
rgb = color order (RGB,1, or RBG,0,)
Orb parameters:
nFeatures : number of features per image
scaleFactor: Scale factor between levels in the scale pyramid
nLevels : Number of levels in the scale pyramid
Image is divided in a grid. At each cell FAST are extracted imposing
a minimum response.Firstly we impose iniThFAST. If no corners are
detected we impose a lower value minThFAST You can lower these
values if your images have low contrast
IniThFAST :
MinThFast :
```

Parameters from Frame.cc:

```
The sigma squared of the frame is obtained from mpORBextractorLeft-
>GetScaleSigmaSquares();
in Frame::ComputeStereoMatches():
   There is searched in the right image for the keypoints of the
   left image in a limit of: minZ = mb = stereo baseline, minD = 0
   and maxD = mbf/minZ = stereo baseline times the focal length
   devided by minZ = focallength.
```

Parameters from Initializer.cc:

```
in Initializer::Initialize():
   The choice between using the homography or fundamental method
   is made by calculating both in seperate threads and calculating
   a ratio based on the scores: RH=SH/(SH+SF), if this is greater
   than 0.4, the homography is used, otherwise, the fundamental is
   used.
```

Parameters from LocalMapping.cc:

```
in LocalMapping::CreateNewMapPoint():
   Uses the 20 (10 if not monocular) nearest keyframes in the
   covisibility graph and of these only the ones with a large
   enough baseline. and a ORB matcher with nnratio of 0.6.

   The reprojection error due to triangulation is checked to be
   smaller than 5.991 (7.8 in case of stereo) times the sigma
   squared.
in LocalMapping::SearchInNeighbors():
   Uses the 20 (10 if not monocular) nearest keyframes in the
   covisibility graph.
in LocalMapping::KeyFrameCulling():
   Check redundant keyframes (only local keyframes)
   A keyframe is considered redundant if the 90% of the MapPoints
   it sees, are seen in at least other 3 keyframes (in the same or
   finer scale)
```

Parameters from LoopClosing.cc

```
in LoopClosing::LoopClosing():
   the CovisibilityConsistency Threshold is set to 3.
in LoopClosing::DetectLoop():
   if a loop is detected within the last 10 keyframes or 10
   keyframes have not been created yet, the function immediately
   returns false.
in LoopClosing::ComputeSim3():
   An ORB matcher is used of a nn ratio of 0.75. If there are more
   than 20 matches between a the keyframe and the candidate, a
   Sim3solver is used with the ransacparamters of a probability of
   0.99, 20 minimal inliers and 300 as max iterations.
   A loop is defined as a loop when more than 40 matches have been
   found.
```

Parameters from Optimizer.cc:

```
in Optimizer::BundleAdjustment():
   a 2D Huber threshold is set as sqrt(5.99) and a 3D Huber
   threshold is set as sqrt(7.815).
in Optimizer::OptimizeEssentialGraph():
   An Levenburg algorithm is used with an initial Lambda of 1e-16.
   The amount of covisible KF is set to 100.
   The optimization is done for 20 iterations.
```

```
in Optimizer::OptimizeSim3():
   Here the optimization is done for 5 iterations. After which the
   optimezation is done again with only the inliers. (for 10
   iterations if there are outliers, otherwise 5 iterations, or if
   there are less than 10 inliers left, the whole optimization
   fails).
```

Parameters from ORBextractor.cc:

```
patch_size = 31 , half_patch_size = 15 , edge_threshold = 19
```

Parameters from ORBmatcher.cc:

```
TH_HIGH = 100 , TH_LOW = 50 , HISTO_LENGTH = 30
NOT FINISHED YET
```

Parameters from Tracking.cc:

```
Max/Min Frames to insert keyframes and to check relocalisation
mMinFrames = 0;
mMaxFrames = fps;
Threshold close/far points
Points seen as close by the stereo/RGBD sensor are considered
reliable and inserted from just one frame. Far points require a
match in two keyframes.
mbf = camera.bf: some kind of scale
mThDepth = mbf*(float)fSettings["ThDepth"]/fx;
For RGB-D inputs only. For some datasets (e.g. TUM) the depthmap
values are scaled.
mDepthMapFactor = fSettings["DepthMapFactor"];
if(fabs(mDepthMapFactor)<1e-5)
  mDepthMapFactor=1;
else
  mDepthMapFactor = 1.0f/mDepthMapFactor;
in Tracking::StereoInitialization():
  hardcoded: if the amount of keypoints
     (mCurrentFrame.N) is less than 500,it is not
      initialized. on line 511
in Tracking::MonocularInitialization():
  hardcoded: if the amount of keypoints
     (mCurrentFrame.mvKeys.size()) is less than 100,it is not
      initialized. on line 590 and 603
in Tracking::CreateInitialMapMonocular():
   If the number of cepoints on the current keyframe is less than
   100, Then the initializatoin has gone rong and the map resets.
   on line 694
In Tracking::TrackReferenceKeyFrame():
   An ORBMatcher is created with a nnratio of 0.7. line 764
   if the amount of orb matches is less then 15, the function
   returns False (line 769) Then it removes outliers and returns
   the result of the comparison if the number of matches is more
   or equal then 10.
In Tracking::UpdateLastFrame():
   All the points closer than mThDepth (see above) are inserted
   unless these are less than 100, then the 100 closest are used.
   line 862
In Tracking::TrackWithMotionModel():
   An ORBMatcher is created with a nnratio of 0.9. line 764
```

```
    If stereo tracking is done, th is set to 15, else to 7. th is
    passed on to matcher.SearchByProjection to determine the
    searchingradius (after being scaled). If this results in less
    then 20 matches, twice the th is used. If still not 20 matches
    are found, it fails.
    it removes outliers and returns
    the result of the comparison if the number of matches is more
    or equal then 10.
  In Tracking::TrackLocalMap():
    if the amount of found matches are less then 30, it return
    false, else it returns true. (50 if there have been less then
    nMaxFrames since the last relocalization)
  In Tracking::CreateNewKeyFrame():
    All the points closer than mThDepth (see above) are inserted
    unless these are less than 100, then the 100 closest are used.
    line 1129
  In Tracking::SearchLocalPoints():
    The mappoints are projected into the frame and if the cosine
    between the last en current viewing angle is smaller then 0.5
    (line 1175) then it is used. Then an orbmatcher is used with an
    nnratio of 0.8, and again a th is set (3 for RGBD, 5 if the
    camera has recently been relocalised and otherwise 1) for
    matcher.SearchByProjection (see under TrackWithMotionModel()
  In Tracking::UpdateLocalKeyframe():
    If less than 80 keyframes that observe a map point are included
    in the local map, then it is filled up to 80 using keyframes
    that are neigbors to already-included keyframes.
  In Tracking::Relocalization():
    Here a ORBMatcher is used with an nnratio of 0.75. If more than
    15 matches are found between the local frame and one of the
    keyframes, this keyframe is considered a candidate and a PNP
    solver is run using the ransac parameters: probability of 0.99,
    minInliers of 10, maxIterations of 300, minSEt of 4, epsilon of
    0.5, th2 of 5.991.
    Then some iterations of P4P RANSAC is done until a match is
    found or there are no candidates anymore. For each candidate, 5
    ransac itterations are done, and if ransac reaches the max.
    iterations, the candidate is discarded. If not, the pose is
    optimized: If the amount of correspondences is less then 10, it
    is skipped. If it is less then 50, there is searched by
    projection (in a coarse window) with a threshold of 3 and a
    ORBDist of 100, and then it is optimized again if it returns
    more than 50 total correspondences. If after this optimization,
    between the 30 and 50 correspondeces are left over, there is
    again searched by projection, this time with a threshold of 3
    and a ORBDist of 64, and it is optimized again. If more than 50
    correspondeces where found, then the  correct frame has been
    found, otherwise, it returns false.
```

103 claps

WRITTEN BY

Jeroen Zijlmans

Follow

See responses (1)

More From Medium

Related reads

Computer Vision: Keep a Sharp Eye on the Road

Intellias...
Feb 14, 2018 · 13...

Related reads

Depth Estimation Using Encoder-Decoder Networks and Self-Supervised Learning

NeuroHive in Go...
Jun 8, 2018 · 5 mi...

Related reads

The Camera IS The Lidar

Angus Pacala in...
Sep 1, 2018 · 5 mi...