Visual SLAM

Generated by Doxygen 1.9.6

Chapter 1

Source content

main.cpp

- Main program execution.

map.hpp

- Map object holds maps (std::map), with pointers (std::shared_ptr) to Frame and Point3D objects as values and corresponding frame ids and point ids as keys. Map also contains the main algorithmic workload

point.hpp

- Point object stores estimated 3D location and map (std::map) with correspondences to imagepoints and features in Frame objects that see this map point.

frame.hpp

- Frame object stores information processed from individual video frames. Also includes FeatureMatcher and Feature Extractor classes.

helper_functions.hpp

- Various helper functions for repetitive procedures like slicing of matrices and conversions to different matrix types for cv::Mat and Eigen.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

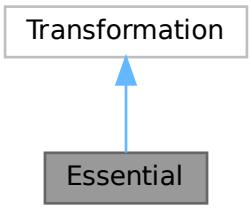
Here is a list of all documented files with brief descriptions:

Chapter 5

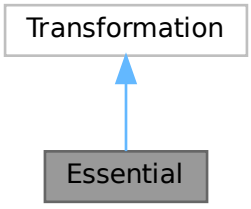
Class Documentation

5.1 Essential Class Reference

Inheritance diagram for Essential:



Collaboration diagram for Essential:



Public Member Functions

- void **Estimate** (const cv::Mat &points1, const cv::Mat &points2, const cv::Mat &K)

Private Attributes

- cv::Mat **triangulatedPoints**

Additional Inherited Members

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/transformation.hpp

5.2 FeatureExtractor Class Reference

Public Member Functions

- std::tuple< cv::Mat, cv::Mat > **compute_features** (const cv::Mat &img)

Private Attributes

- cv::Ptr< cv::FeatureDetector > **detector** = cv::ORB::create(1500)
- cv::Ptr< cv::DescriptorExtractor > **descriptor** = cv::ORB::create(1500)

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/frame.hpp

5.3 FeatureMatcher Class Reference

Public Member Functions

- std::tuple< std::vector< cv::DMatch >, cv::Mat, cv::Mat, cv::Mat, cv::Mat > **match_features** (cv::Mat kp1, cv::Mat desc1, cv::Mat kp2, cv::Mat desc2, float ratio=0.80)

Private Attributes

- cv::BFMatcher **matcher**

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/frame.hpp

5.4 Frame Class Reference

Frame class is used to store extracted information about the video frames.

`#include <frame.hpp>`

Public Member Functions

- Frame ()
- Frame (cv::Mat rgb_img, int id)
- Frame (std::string rgb_path, int id)
- void AddParent (int parent_frame_id, cv::Mat transition)
- std::vector< int > GetParentIDs ()
- cv::Mat GetTransitionWithParentID (int parent_id)
- std::tuple< cv::Mat, cv::Mat > feature_extract (cv::Mat rgb_img, FeatureExtractor feature_extractor)
- std::tuple< cv::Mat, cv::Mat, cv::Mat > process_frame (FeatureExtractor feature_extractor)
- void process (FeatureExtractor feature_extractor)

  method process extracts features and processes the frame
- cv::Mat GetRGB () const

  method GetRGB is getter for the RGB image
- void SetRGB (cv::Mat new_rgb)

  method SetRGB sets RGB image
- void AddPose (cv::Mat init_pose)

  method AddPose adds initial pose
- void UpdatePose (cv::Mat new_pose)

  method UpdatePose does the same as add pose, but for clarity different function name
- cv::Mat GetPose () const

  method GetPose is getter for the 4x4 transformation matrix
- bool IsKeyFrame () const

  method GetPose returns keyframe information
- void **SetAsKeyFrame** ()

  method SetAsKeyFrame is setter for the keyframe, set boolean flag to true if the frame is a keyframe
- void SetKeyPoints (cv::Mat new_points)

  method SetKeyPoints is setter for the image points
- cv::Mat GetKeyPoints () const

  method GetPoGetKeyPointse returns keypoints
- std::vector< cv::KeyPoint > GetKeyPointsAsVector () const

  method GetKeyPointsAsVector returns keypoints as a std::vector, Only visualization needs them in this form
- void SetFeatures (cv::Mat new_features)

  method SetFeatures is setter for the features
- cv::Mat GetFeatures () const

  method SetKeyPoints Getter for the features
- int GetID () const

  method GetID returns frame_id
- void AddID (int new_id)

  method SetFeatures is setter for the frame_id
- cv::Mat **GetCameraCenter** ()

  get camera center

Static Public Member Functions

- static std::tuple< std::vector< cv::DMatch >, cv::Mat, cv::Mat, cv::Mat, cv::Mat > Match2Frames (std::shared_ptr< Frame > prev_frame, std::shared_ptr< Frame > cur_frame, FeatureMatcher feature_matcher)
- static std::vector< cv::KeyPoint > **GetKeyPointsAsVector** (cv::Mat mat_keypoints)

Private Attributes

- cv::Mat **rgb**

  rgb image stored in cv::Mat
- cv::Mat **keypoints**

  extracted keypoints (=imagepoints) stored in Nx2 cv::Mat
- cv::Mat **features**

  extracted descriptors for keypoints stored in Nxfeature_length cv::Mat
- cv::Mat **pose**

  estimated camera pose during the frame
- int **ID**

  unique identifier for the frame when stored in map
- std::map< int, cv::Mat > **parents**

  std::map storing parents of this frame (useful when building a graph)
- bool **keyframe** = false

  boolean flag indicating if the frame is considered keyframe

5.4.1 Detailed Description

Frame class is used to store extracted information about the video frames.

Author

Juuso Korhonen

Date

December 2022

5.4.2 Constructor & Destructor Documentation

5.4.2.1 Frame() [1/3]

```
Frame::Frame ( )  [inline]
```

Empty constructor.

5.4.2.2 Frame() [2/3]

```
Frame::Frame (
            cv::Mat rgb_img,
            int id )  [inline]
```

Constructor

Parameters

rgb_img	- video frame as cv::Mat, for example the output of cv::imread("frame.png")
id	- unique identifier of the frame when added to map

5.4.2.3 Frame() [3/3]

```
Frame::Frame (
            std::string rgb_path,
            int id )  [inline]
```

Constructor

Parameters

rgb_path	- path to video frame file
id	- unique identifier of the frame when added to map

5.4.3 Member Function Documentation

5.4.3.1 AddID()

```
void Frame::AddID (
            int new_id )  [inline]
```

method SetFeatures is setter for the frame_id

Parameters

new_id	

5.4.3.2 AddParent()

```
void Frame::AddParent (
            int parent_frame_id,
            cv::Mat transition )  [inline]
```

method AddParent adds parent frame to the current frame

Parameters

parent_frame↩ _id	int type frame id corresponding to the parent frame
transition	cv::Mat type 4x4 transformation matrix corresponding to the mapping between the frames

5.4.3.3 AddPose()

```
void Frame::AddPose (
            cv::Mat init_pose )  [inline]
```

method AddPose adds initial pose

Parameters

new_rgb	type of cv::Mat corresponding to 4x4 transormation matrix in the world frame

5.4.3.4 feature_extract()

```
std::tuple< cv::Mat, cv::Mat > Frame::feature_extract (
            cv::Mat rgb_img,
            FeatureExtractor feature_extractor )  [inline]
```

method feature_extract extract features from rgb image with helper class FeatureExtractor object

Returns

std::tuple<cv::Mat, cv::Mat> type corresponding to image points and features

5.4.3.5 GetFeatures()

```
cv::Mat Frame::GetFeatures ( ) const  [inline]
```

method SetKeyPoints Getter for the features

Returns

new_features type of cv::mat corresponding to the new features

5.4.3.6 GetID()

```
int Frame::GetID ( ) const  [inline]
```

method GetID returns frame_id

Returns

int type corresponding to frame_id

5.4.3.7 GetKeyPoints()

```
cv::Mat Frame::GetKeyPoints ( ) const  [inline]
```

method GetPoGetKeyPointse returns keypoints

Returns

cv::Mat type corresponding to image points

5.4.3.8 GetKeyPointsAsVector()

```
std::vector< cv::KeyPoint > Frame::GetKeyPointsAsVector ( ) const  [inline]
```

method GetKeyPointsAsVector returns keypoints as a std::vector, Only visualization needs them in this form

Returns

std::vector<cv::KeyPoint> type corresponding to image points

5.4.3.9 GetParentIDs()

```
std::vector< int > Frame::GetParentIDs ( )  [inline]
```

method GetParentIDs returns all the parent ids

Returns

return std::vector<int> type array of frame ids

5.4.3.10 GetPose()

```
cv::Mat Frame::GetPose ( ) const  [inline]
```

method GetPose is getter for the 4x4 transformation matrix

Returns

cv::Mat type corresponding to pose

5.4.3.11 GetRGB()

```
cv::Mat Frame::GetRGB ( ) const  [inline]
```

method GetRGB is getter for the RGB image

Returns

cv::Mat type corresponding to RGB image

5.4.3.12 GetTransitionWithParentID()

```
cv::Mat Frame::GetTransitionWithParentID (
            int parent_id )  [inline]
```

method GetTransitionWithParentID returns 4x4 trasition matrix between the parent frame and the current frame

Returns

std::vector<int> type array of frame ids

5.4.3.13 IsKeyFrame()

```
bool Frame::IsKeyFrame ( ) const  [inline]
```

method GetPose returns keyframe information

Returns

bool type corresponding to if the frame is keyframae

5.4.3.14 Match2Frames()

```
static std::tuple< std::vector< cv::DMatch >, cv::Mat, cv::Mat, cv::Mat, cv::Mat > Frame::↩
Match2Frames (
            std::shared_ptr< Frame > prev_frame,
            std::shared_ptr< Frame > cur_frame,
            FeatureMatcher feature_matcher )  [inline], [static]
```

matches 2 Frame objects

Parameters

prev_frame	shared pointer to previous frame
cur_frame	shared pointer to current frame
feature_matcher	FeatureMatcher object to be used for feature matching

Returns

tuple containing matching indices (for keypoints and descriptors) for previous frame (in col(0)) and current frame (in col(1)), matching keypoints in previous frame, matching descriptors in previous frame, matching keypoints in current frame, matching descriptors in current frame

5.4.3.15 process()

```
void Frame::process (
            FeatureExtractor feature_extractor )  [inline]
```

method process extracts features and processes the frame

Parameters

FeatureExtractor	intance of class FeatureExtractor

5.4.3.16 process_frame()

```
std::tuple< cv::Mat, cv::Mat, cv::Mat > Frame::process_frame (
            FeatureExtractor feature_extractor )  [inline]
```

method process frame and return keypoints, features and the rgb image from where they were found

Parameters

FeatureExtractor	intance of class FeatureExtractor

Returns

std::tuple<cv::Mat, cv::Mat, cv::Mat> type corresponding to keypoints, features and the rgb image

5.4.3.17 SetFeatures()

```
void Frame::SetFeatures (
            cv::Mat new_features )  [inline]
```

method SetFeatures is setter for the features

Parameters

new_features	type of cv::mat corresponding to the new features

5.4.3.18 SetKeyPoints()

```
void Frame::SetKeyPoints (
            cv::Mat new_points )  [inline]
```

method SetKeyPoints is setter for the image points

Parameters

new_points	type of cv::mat corresponding to the image points

5.4.3.19 SetRGB()

```
void Frame::SetRGB (
            cv::Mat new_rgb )  [inline]
```

method SetRGB sets RGB image

Parameters

new_rgb	type of cv::Mat

5.4.3.20 UpdatePose()

```
void Frame::UpdatePose (
            cv::Mat new_pose )  [inline]
```

method UpdatePose does the same as add pose, but for clarity different function name

Parameters

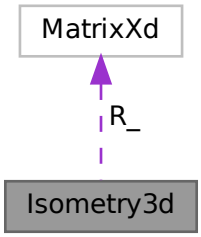
new_rgb	type of cv::Mat corresponding to 4x4 transormation matrix in the world frame

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/frame.hpp

5.5 Isometry3d Class Reference

Collaboration diagram for Isometry3d:



Public Member Functions

- **Isometry3d** (Eigen::MatrixXd R, Eigen::VectorXd t)
- Eigen::MatrixXd **matrix** ()
- Isometry3d **inverse** ()
- Eigen::MatrixXd **orientation** ()
- Eigen::VectorXd **position** ()

Private Attributes

- Eigen::MatrixXd **R_**
- Eigen::VectorXd **t_**

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/isometry3d.hpp

5.6 Map Class Reference

Map class is used to store Frame and Point3D objects.

```
#include <map.hpp>
```

Public Member Functions

- void InitializeMap (std::vector< std::filesystem::path >::iterator &input_video_it, int &id_frame, int &id_point, FeatureExtractor feature_extractor, FeatureMatcher feature_matcher, cv::Mat cameraIntrinsicsMatrix, bool visualize=false)

  Initializes map based on first two keyframes from video (first being the 1st frame and 2nd is being found)
- void localTracking (std::vector< std::filesystem::path >::iterator &input_video_it, int &id_frame, int &id_point, FeatureExtractor feature_extractor, FeatureMatcher feature_matcher, cv::Mat cameraIntrinsicsMatrix, cv::Mat DistCoefficients, PointClouds &clouds, bool visualize=true, bool verbose_optimization=false)

  Tracks map points in consecutive video frames and estimates poses using PnP-algorithm. Breaks when a new keyframe is found:
- void localMapping (int &id_frame, int &id_point, FeatureExtractor feature_extractor, FeatureMatcher feature_matcher, cv::Mat cameraIntrinsicsMatrix, cv::Mat DistCoefficients, int &last_key_frame_id, bool visualize=false)

  Does mapping using the last keyframe and new keyframe: Tries to find new matches between their descriptors and triangulate 3d locations for matches using their estimated poses.
- void **CleanUpBadPoints** ()
- void AddFrame (int frame_id, std::shared_ptr< Frame > frame)

  Adds frame to map.
- void AddPoint3D (int point_id, std::shared_ptr< Point3D > point_3d)

  Adds point to map.
- void AddPoints3D (int &id_point, cv::Mat points_3d, std::shared_ptr< Frame > frame1, cv::Mat uv1, cv::Mat desc1, std::shared_ptr< Frame > frame2, cv::Mat uv2, cv::Mat desc2, cv::Mat inlierMask)

  Adds multiple points to map with one function call.
- std::vector< int > GetPointsVisibleToFrame (int frame_id)

  Gets ids of points that are visible to frame.
- std::vector< int > GetPointsVisibleToFrames (std::vector< int > frame_id_list)

  Gets ids of points that are visible to frames.
- std::tuple< cv::Mat, cv::Mat, cv::Mat, std::vector< int > > GetImagePointsWithFrameID (int frame_id)

  Gets information about the map points that are visible to frame with frame id.
- std::vector< cv::Mat > Get3DPointsWithIDs (std::vector< int > id_list)

  Gets 3d locations of points with a vector of ids.
- std::vector< cv::Mat > GetAll3DPoints ()

  Gets all 3d locations of points in the map.
- std::vector< cv::Mat > GetAllPoses ()

  Gets all poses of Frame objects stored in the map.
- std::vector< cv::Mat > GetAllCameraLocations (bool keyframes_only=false)

  Gets all translation components (xyz camera locations) of poses of Frame objects stored in the map.
- void UpdatePose (cv::Mat new_pose, int frame_id)

  Updates the pose of the frame to new_pose.
- void UpdatePoint3D (cv::Mat new_point, int point_id)

  Updates the 3d location of the point.
- std::shared_ptr< Frame > GetFrame (int frame_id)

  Gets the pointer to frame if the frame id exists in map.
- std::shared_ptr< Point3D > GetPoint (int point_id)

  Gets the pointer to point if the point id exists in map.
- void Store3DPoints (std::map< int, std::shared_ptr< Point3D > > points_map)

  Stores multiple Point3D objects in the map at once.
- void AddParentAndPose (int parent_id, int frame_id, std::shared_ptr< Frame > frame_obj, cv::Mat rel_pose_trans, cv::Mat pose)

  compilation of calls when frame is inserted to map
- void AddPoseNode (int frame_id, std::shared_ptr< Frame > frame_obj, cv::Mat pose, int parent_id, cv::Mat rel_pose_trans)

compilation of calls when frame is inserted to map

- std::vector< int > GetAllFrameIDs ()

  Gets all frame ids in the map.

- std::vector< int > GetAllPointIDs ()

  Gets all point ids in the map.

- void AddPointToFrameCorrespondances (std::vector< int > point_ids, cv::Mat image_points, cv::Mat descriptors, std::shared_ptr< Frame > frame_ptr, cv::Mat inliers)

  Adds point to frame correspondences from points to frames.

- void BundleAdjustement (bool tracking, bool scale=false, bool verbose=false, int n_iterations=10)

  Optimizes poses and 3D locations of points in the map, Leverberg-Marquadt used to minimize the reprojection error. Updates points and poses to the optimized values.

Private Attributes

- std::map< int, std::shared_ptr< Frame > > **frames_**

  frame map container hold unique frame id's as a key and std::shared_ptr<Frame> as a value

- std::map< int, std::shared_ptr< Point3D > > **point_3d_**

  point_3d_ map container hold unique point id's as a key and std::shared_ptr<Point3D> as a value

5.6.1 Detailed Description

Map class is used to store Frame and Point3D objects.

Author

Juuso Korhonen, Jere Knuutinen

Date

December 2022

5.6.2 Member Function Documentation

5.6.2.1 AddFrame()

```
void Map::AddFrame (
            int frame_id,
            std::shared_ptr< Frame > frame )  [inline]
```

Adds frame to map.

Parameters

frame_id	- unique identifier for Frame in the map
frame	- shared pointer pointing to a created Frame object

5.6.2.2 AddParentAndPose()

```
void Map::AddParentAndPose (
            int parent_id,
            int frame_id,
            std::shared_ptr< Frame > frame_obj,
            cv::Mat rel_pose_trans,
            cv::Mat pose )  [inline]
```

compilation of calls when frame is inserted to map

Parameters

parent_id	id of the parent frame
frame_obj	pointer to the Frame object itself
rel_pos_trans	relative pose transformation between parent frame camera pose and frame camera pose
pose	frame camera pose

5.6.2.3 AddPoint3D()

```
void Map::AddPoint3D (
            int point_id,
            std::shared_ptr< Point3D > point_3d )  [inline]
```

Adds point to map.

Parameters

point_id	- unique identifier for Point3D object in the map
point_3d	- shared pointer pointing to a created Point object

5.6.2.4 AddPoints3D()

```
void Map::AddPoints3D (
            int & id_point,
            cv::Mat points_3d,
            std::shared_ptr< Frame > frame1,
            cv::Mat uv1,
            cv::Mat desc1,
            std::shared_ptr< Frame > frame2,
            cv::Mat uv2,
            cv::Mat desc2,
            cv::Mat inlierMask )  [inline]
```

Adds multiple points to map with one function call.

Parameters

id_point	- running indexing of points as reference, gets increased inside the function
point_3d	- shared pointer pointing to a created Point object
frame1	- shared pointer pointing to a created Frame object that sees the point
uv1	- imagepoints in frame 1
desc1	- descriptors in frame 1
frame2	- shared pointer pointing to a created Frame object that sees the point
uv2	- imagepoints in frame 2
desc2	- descriptors in frame 2
inlierMask	- inlierMask containing 1 in the row if the corresponding rows in imagepoints and descriptors should be added

5.6.2.5 AddPointToFrameCorrespondances()

```
void Map::AddPointToFrameCorrespondances (
            std::vector< int > point_ids,
            cv::Mat image_points,
            cv::Mat descriptors,
            std::shared_ptr< Frame > frame_ptr,
            cv::Mat inliers )  [inline]
```

Adds point to frame correspondences from points to frames.

Parameters

point_ids	vector of point ids
image_points	image points in frame for points
descriptors	descriptors for image points in frame for points
frame_ptr	pointer to Frame object
inliers	vector of inlier indices to be used

5.6.2.6 AddPoseNode()

```
void Map::AddPoseNode (
            int frame_id,
            std::shared_ptr< Frame > frame_obj,
            cv::Mat pose,
            int parent_id,
            cv::Mat rel_pose_trans )  [inline]
```

compilation of calls when frame is inserted to map

Parameters

parent_id	id of the parent frame

Parameters

frame_obj	pointer to the Frame object itself
pose	frame camera pose
rel_pos_trans	relative pose transformation between parent frame camera pose and frame camera pose

5.6.2.7 BundleAdjustement()

```
void Map::BundleAdjustement (
            bool tracking,
            bool scale = false,
            bool verbose = false,
            int n_iterations = 10 )  [inline]
```

Optimizes poses and 3D locations of points in the map, Leverberg-Marquadt used to minimize the reprojection error. Updates points and poses to the optimized values.

Parameters

tracking	boolean flag, set true if points are set as fixed (MotionOnly)
scale	boolean flag, if scaling should be done for depth of points. If true, scales points so that median depth for points is 1.
verbose	should we print information about the optimization process
n_iterations	how many iterations to run the optimization algorithm

5.6.2.8 Get3DPointsWithIDs()

```
std::vector< cv::Mat > Map::Get3DPointsWithIDs (
            std::vector< int > id_list )  [inline]
```

Gets 3d locations of points with a vector of ids.

Parameters

id_list	- ids of points

Returns

vector of 3d locations of points

5.6.2.9 GetAll3DPoints()

```
std::vector< cv::Mat > Map::GetAll3DPoints ( )  [inline]
```

Gets all 3d locations of points in the map.

Returns

vector of 3d locations of points

5.6.2.10 GetAllCameraLocations()

```
std::vector< cv::Mat > Map::GetAllCameraLocations (
            bool keyframes_only = false )  [inline]
```

Gets all translation components (xyz camera locations) of poses of Frame objects stored in the map.

Returns

vector of camera locations

5.6.2.11 GetAllFrameIDs()

```
std::vector< int > Map::GetAllFrameIDs ( )  [inline]
```

Gets all frame ids in the map.

Returns

vector of frame ids

5.6.2.12 GetAllPointIDs()

```
std::vector< int > Map::GetAllPointIDs ( )  [inline]
```

Gets all point ids in the map.

Returns

vector of point ids

5.6.2.13 GetAllPoses()

```
std::vector< cv::Mat > Map::GetAllPoses ( )  [inline]
```

Gets all poses of Frame objects stored in the map.

Returns

vector of poses

5.6.2.14 GetFrame()

```
std::shared_ptr< Frame > Map::GetFrame (
            int frame_id )  [inline]
```

Gets the pointer to frame if the frame id exists in map.

Parameters

id	id of the frame

5.6.2.15 GetImagePointsWithFrameID()

```
std::tuple< cv::Mat, cv::Mat, cv::Mat, std::vector< int > > Map::GetImagePointsWithFrameID (
            int frame_id )  [inline]
```

Gets information about the map points that are visible to frame with frame id.

Parameters

frame_id	- id of the frame

Returns

tuple of imagepoints, descriptors, 3d locations, point ids of map points corresponding to the frame

5.6.2.16 GetPoint()

```
std::shared_ptr< Point3D > Map::GetPoint (
            int point_id )  [inline]
```

Gets the pointer to point if the point id exists in map.

Parameters

id	id of the point

5.6.2.17 GetPointsVisibleToFrame()

```
std::vector< int > Map::GetPointsVisibleToFrame (
            int frame_id ) [inline]
```

Gets ids of points that are visible to frame.

Parameters

frame_id	- id of the frame

Returns

vector of point ids

5.6.2.18 GetPointsVisibleToFrames()

```
std::vector< int > Map::GetPointsVisibleToFrames (
            std::vector< int > frame_id_list ) [inline]
```

Gets ids of points that are visible to frames.

Parameters

frame_id	- ids of the frame

Returns

vector of point ids

5.6.2.19 InitializeMap()

```
void Map::InitializeMap (
            std::vector< std::filesystem::path >::iterator & input_video_it,
            int & id_frame,
```

```
            int & id_point,
            FeatureExtractor feature_extractor,
            FeatureMatcher feature_matcher,
            cv::Mat cameraIntrinsicsMatrix,
            bool visualize = false )  [inline]
```

Initializes map based on first two keyframes from video (first being the 1st frame and 2nd is being found)

Parameters

input_video_it	- iterator for going through the video frames (image file paths)
id_frame	- running indexing for Frame objects to be stored in the map
id_point	- running indexing for Point3D objects to be stored in the map
feature_extractor	- an instance of FeatureExtractor class, used for extracting information from the video frames
feature_matcher	- an instance of FeatureMatcher class, used for matching keypoints and features of two video frames
cameraIntrinsicsMatrix	- stores camera parameters in matrix form
visualize	- flag to tell if should visualize the matching process with opencv

5.6.2.20 localMapping()

```
void Map::localMapping (
            int & id_frame,
            int & id_point,
            FeatureExtractor feature_extractor,
            FeatureMatcher feature_matcher,
            cv::Mat cameraIntrinsicsMatrix,
            cv::Mat DistCoefficients,
            int & last_key_frame_id,
            bool visualize = false )  [inline]
```

Does mapping using the last keyframe and new keyframe: Tries to find new matches between their descriptors and triangulate 3d locations for matches using their estimated poses.

Parameters

id_frame	- running indexing for Frame objects to be stored in the map as reference, gets increased inside the map
id_point	- running indexing for Point3D objects to be stored in the map as reference, gets increased inside the map
feature_extractor	- an instance of FeatureExtractor class, used for extracting information from the video frames
feature_matcher	- an instance of FeatureMatcher class, used for matching keypoints and features of two video frames
cameraIntrinsicsMatrix	- stores camera parameters in matrix form
DistCoefficients	- stores camera distortion coefficients in vector form
last_key_frame_id	- index of the last keyframe (store before tracking)
visualize	- flag to tell if should visualize the matching process with opencv
verbose_optimization	- flag to tell if we should print details of the motion only optimization

5.6.2.21 localTracking()

```
void Map::localTracking (
            std::vector< std::filesystem::path >::iterator & input_video_it,
            int & id_frame,
            int & id_point,
            FeatureExtractor feature_extractor,
            FeatureMatcher feature_matcher,
            cv::Mat cameraIntrinsicsMatrix,
            cv::Mat DistCoefficients,
            PointClouds & clouds,
            bool visualize = true,
            bool verbose_optimization = false )  [inline]
```

Tracks map points in consecutive video frames and estimates poses using PnP-algorithm. Breaks when a new keyframe is found:

- 1. at least 20 frames has passed or current frame tracks less than 80 map points
- 2. The map points tracked are fewer than 90% of the map points seen by the last key frame

Parameters

input_video_it	- iterator for going through the video frames (image file paths)
id_frame	- running indexing for Frame objects to be stored in the map as reference, gets increased inside the map
id_point	- running indexing for Point3D objects to be stored in the map as reference, gets increased inside the map
feature_extractor	- an instance of FeatureExtractor class, used for extracting information from the video frames
feature_matcher	- an instance of FeatureMatcher class, used for matching keypoints and features of two video frames
cameraIntrinsicsMatrix	- stores camera parameters in matrix form
DistCoefficients	- stores camera distortion coefficients in vector form
visualize	- flag to tell if should visualize the matching process with opencv
verbose_optimization	- flag to tell if we should print details of the motion only optimization

5.6.2.22 Store3DPoints()

```
void Map::Store3DPoints (
            std::map< int, std::shared_ptr< Point3D > > points_map )  [inline]
```

Stores multiple Point3D objects in the map at once.

Parameters

points_map	std::map with point ids as keys, and Point3D objects as values

5.6.2.23 UpdatePoint3D()

```
void Map::UpdatePoint3D (
            cv::Mat new_point,
            int point_id )  [inline]
```

Updates the 3d location of the point.

Parameters

id	id of the point

5.6.2.24 UpdatePose()

```
void Map::UpdatePose (
            cv::Mat new_pose,
            int frame_id )  [inline]
```

Updates the pose of the frame to new_pose.

Parameters

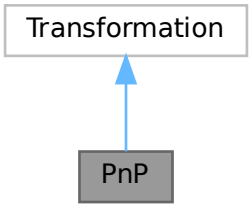
id	id of the frame

The documentation for this class was generated from the following file:

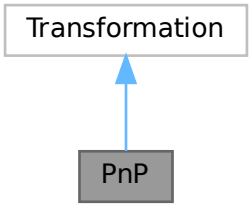
- /home/jere/cpp_visual_slam/src/map.hpp

5.7 PnP Class Reference

Inheritance diagram for PnP:



Collaboration diagram for PnP:



Public Member Functions

- void **Estimate** (cv::Mat matched_3d, cv::Mat curMatchedPoints, cv::Mat cameraIntrinsicsMatrix, cv::Mat DistCoefficients)

Additional Inherited Members

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/transformation.hpp

5.8 Point3D Class Reference

Point class is used to store point id, 3D point locations and corresponding frame information.

`#include <point.hpp>`

Public Member Functions

- Point3D (int ID, cv::Mat location_3d)
- int GetID ()

  method GetID returns point_id
- std::shared_ptr< Frame > GetFrame (int frame_id)
- cv::Mat **GetFrame2** (int frame_id)
- cv::Mat GetImagePoint (int frame_id)
- std::map< int, std::tuple< std::shared_ptr< Frame >, cv::Mat, cv::Mat > > **SubsetOfFrames** (int frame_id)
- void AddFrame (std::shared_ptr< Frame > frame, cv::Mat uv, cv::Mat descriptor)
- void UpdatePoint (cv::Mat new_location)
- bool IsVisibleTo (int frame_id)
- std::tuple< cv::Mat, cv::Mat > GetImagePointAndFeature (int frame_id)
- cv::Mat Get3dPoint ()
- int GetNVisibleFrames ()
- void SetFrames (std::map< int, std::tuple< std::shared_ptr< Frame >, cv::Mat, cv::Mat > > subset_of_↩ frames)
- std::map< int, std::tuple< std::shared_ptr< Frame >, cv::Mat, cv::Mat > > & GetFrames ()
- bool IsBad ()

Private Attributes

- int **ID_**

  unique ID that defines the point (Bool)
- cv::Mat **location_3d_**

  3D location of thhe point (cv::Mat)
- std::map< int, std::tuple< std::shared_ptr< Frame >, cv::Mat, cv::Mat > > **frames_**

  map of frames that see this particular point object (std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat>>) // map of frames that see this particular point object

Friends

- std::ostream & **operator<<** (std::ostream &os, const Point3D &p)

5.8.1 Detailed Description

Point class is used to store point id, 3D point locations and corresponding frame information.

Author

Jere Knuutinen

Date

December 2022

5.8.2 Constructor & Destructor Documentation

5.8.2.1 Point3D()

```
Point3D::Point3D (
            int ID,
            cv::Mat location_3d )  [inline]
```

Constructor

Parameters

ID	- Unique identifier of the point when added
*location_3d_*	- 3D location of the point

5.8.3 Member Function Documentation

5.8.3.1 AddFrame()

```
void Point3D::AddFrame (
            std::shared_ptr< Frame > frame,
            cv::Mat uv,
            cv::Mat descriptor )  [inline]
```

Method AddFrame adds frame to frames_ map container

Parameters

frame	std::shared_ptr corresponding to frame object
uv	cv::Mat corresponding to image points

Returns

cv::Mat corresponding to descriptors

5.8.3.2 Get3dPoint()

```
cv::Mat Point3D::Get3dPoint ( )  [inline]
```

Method Get3dPoint getter for getting 3d location

Returns

cv::Mat corresponding to 3d point

5.8.3.3 GetFrame()

```
std::shared_ptr< Frame > Point3D::GetFrame (
            int frame_id )  [inline]
```

Method GetFrame gets frame with frame id

Parameters

frame_id	int corresponding to frame id

Returns

shared pointer corresponding to frame

5.8.3.4 GetFrames()

```
std::map< int, std::tuple< std::shared_ptr< Frame >, cv::Mat, cv::Mat > > & Point3D::Get↩
Frames ( )  [inline]
```

Method GetFrames getter for getting frames

Returns

std::map<int, std::tuple<std::shared_ptr<Frame>

5.8.3.5 GetID()

```
int Point3D::GetID ( )  [inline]
```

method GetID returns point_id

Copy constructor

Parameters

p	constant reference to point object method operator= performs copy assignment
t	constant reference to point object

Returns

reference to t

int type corresponding to frame_id

5.8.3.6 GetImagePoint()

```
cv::Mat Point3D::GetImagePoint (
            int frame_id )  [inline]
```

Method GetImagePoint gets image point with frame_id

Parameters

frame↩ _id	int corresponding to frame id

Returns

cv::Mat image points

5.8.3.7 GetImagePointAndFeature()

```
std::tuple< cv::Mat, cv::Mat > Point3D::GetImagePointAndFeature (
            int frame_id )  [inline]
```

Method GetImagePointAndFeature gets imagepoint and feature with frame id

Parameters

frame↩ _id	id corresponding to frame idetifier

Returns

std::tuple<cv::Mat, cv::Mat> corresponding to image point and descriptors

5.8.3.8 GetNVisibleFrames()

```
int Point3D::GetNVisibleFrames ( )  [inline]
```

Method GetNVisibleFrames getter for getting number of frames that see the point

Returns

cv::Mat corresponding to 3d point

5.8.3.9 IsBad()

```
bool Point3D::IsBad ( )  [inline]
```

Method IsBad for determining if point should be used

Returns

bool if point is seen with less than 3 frames return true, else false

5.8.3.10 IsVisibleTo()

```
bool Point3D::IsVisibleTo (
            int frame_id )  [inline]
```

Method IsVisibleTo checks if the frame with frame_id sees this point

Parameters

frame_id	id corresponding to frame idetifier

Returns

bool true if frame sees that point, else false

5.8.3.11 SetFrames()

```
void Point3D::SetFrames (
            std::map< int, std::tuple< std::shared_ptr< Frame >, cv::Mat, cv::Mat > > subset_of_frames )  [inline]
```

Method SetFrames sets sub set of frames

Parameters

std::map<int,std::tuple<std::shared_ptr<Frame>,cv::Mat,cv::Mat>>	map corresponding to frame information

5.8.3.12 UpdatePoint()

```
void Point3D::UpdatePoint (
            cv::Mat new_location )  [inline]
```

Method UpdatePoint updates 3d point location

Parameters

uv	cv::Mat corresponding to new 3d location

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/point.hpp

5.9 PointClouds Class Reference

The PointClouds class manages the contents of two PointCloud objects, used to keep track of detected points and camera poses. This class can be interfaced to add and remove points from the clouds, as well as to request a screen refresh from the Easy3D viewer.

`#include <screen.hpp>`

Public Member Functions

- **PointClouds** (easy3d::PointCloud *points, easy3d::PointCloud *poses)
- void AddPoint (double x, double y, double z, bool poses=false)

  Adds a single point to either of the point clouds, using xyz coordinates.
- void AddPointMat (cv::Mat point, bool poses=false)

  Adds a single point to either of the point clouds using cv::Mat format.
- void AddPose (double x, double y, double z)

  Shorthand for AddPoint(x, y, z, true);.
- void UpdateView ()

  Manually refresh the viewer associated with the point clouds.
- void Clear (bool poses=false)

  Remove all the points in the selected point cloud.
- void ClearAll ()

  Clears all points in all point clouds.
- void SetPointsMatUpdate (std::vector< cv::Mat > points, bool poses=false)

  Similar to AddPointsMatUpdate(), except this will clear points first.
- void AddPointsMatUpdate (std::vector< cv::Mat > points, bool poses=false)

  Extends the selected point cloud with the contents of the vector. Signals for the viewer to refresh the screen.

Private Attributes

- easy3d::PointCloud * **points_**
- easy3d::PointCloud * **poses_**

5.9.1 Detailed Description

The PointClouds class manages the contents of two PointCloud objects, used to keep track of detected points and camera poses. This class can be interfaced to add and remove points from the clouds, as well as to request a screen refresh from the Easy3D viewer.

Example usage:

PointClouds clouds(points, poses); clouds.AddPoint(0.0, 1.0, -0.5) // Adds to the `points` point cloud clouds.↩ AddPoint(2.0, -1.0, 0.0, true) // Adds to the `poses` point cloud clouds.UpdateView(); clouds.AddPoinstMat↩ Update(<vector of cv::Mat objects>); clouds.ClearAll();

5.9.2 Member Function Documentation

5.9.2.1 AddPoint()

```
void PointClouds::AddPoint (
            double x,
            double y,
            double z,
            bool poses = false )  [inline]
```

Adds a single point to either of the point clouds, using xyz coordinates.

Parameters

x	
y	
z	
poses	If true, the point will be added to the poses point cloud. Otherwise, the points point cloud will be used.

5.9.2.2 AddPointMat()

```
void PointClouds::AddPointMat (
            cv::Mat point,
            bool poses = false )  [inline]
```

Adds a single point to either of the point clouds using cv::Mat format.

Parameters

point	
poses	If true, the point will be added to the poses point cloud. Otherwise, the points point cloud will be used.

5.9.2.3 AddPointsMatUpdate()

```
void PointClouds::AddPointsMatUpdate (
            std::vector< cv::Mat > points,
            bool poses = false )  [inline]
```

Extends the selected point cloud with the contents of the vector. Signals for the viewer to refresh the screen.

Parameters

points	Vector of points in cv::Mat form
poses	If true, add to the POSES point cloud instead of the POINTS one.

5.9.2.4 AddPose()

```
void PointClouds::AddPose (
            double x,
            double y,
            double z )  [inline]
```

Shorthand for AddPoint(x, y, z, true);.

Parameters

x	
y	
z	

5.9.2.5 Clear()

```
void PointClouds::Clear (
            bool poses = false )  [inline]
```

Remove all the points in the selected point cloud.

Parameters

poses	If true, clear the POSES point cloud instead of the POINTS one.

5.9.2.6 ClearAll()

```
void PointClouds::ClearAll ( )  [inline]
```

Clears all points in all point clouds.

5.9.2.7 SetPointsMatUpdate()

```
void PointClouds::SetPointsMatUpdate (
            std::vector< cv::Mat > points,
            bool poses = false )  [inline]
```

Similar to AddPointsMatUpdate(), except this will clear points first.

Parameters

points	points to replace current point cloud with
poses	FALSE if the points point cloud should be replaced, TRUE if the poses point cloud should be replaced

5.9.2.8 UpdateView()

```
void PointClouds::UpdateView ( )  [inline]
```

Manually refresh the viewer associated with the point clouds.

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/screen.hpp

5.10 Screen Class Reference

The screen class is used to interface with the viewer output. It manages the state of the UI, holds handles to the OpenGL buffers used by Easy3D, and other things required to keep the visualization running. The screen class does NOT keep track of the point clouds used by the program. That responsibility is in the PointClouds class.

```
#include <screen.hpp>
```

Public Member Functions

- Screen (std::string title)

  Initialize Easy3D and construct a new Screen object. This will also do some configuration to optimize the output.
- void RegisterPointCloud (easy3d::PointCloud *cloud)

  Registers a point cloud to be rendered using this screen object. Any updates sent to the cloud will now be shown on screen.
- int Run ()

  Blocking call; boots up the visualizer and runs until it completes.

Private Attributes

- easy3d::Viewer * **viewer_**

5.10.1 Detailed Description

The screen class is used to interface with the viewer output. It manages the state of the UI, holds handles to the OpenGL buffers used by Easy3D, and other things required to keep the visualization running. The screen class does NOT keep track of the point clouds used by the program. That responsibility is in the PointClouds class.

5.10.2 Constructor & Destructor Documentation

5.10.2.1 Screen()

```
Screen::Screen (
            std::string title )  [inline], [explicit]
```

Initialize Easy3D and construct a new Screen object. This will also do some configuration to optimize the output.

Parameters

title	The title of the visualizer window

5.10.3 Member Function Documentation

5.10.3.1 RegisterPointCloud()

```
void Screen::RegisterPointCloud (
            easy3d::PointCloud * cloud )  [inline]
```

Registers a point cloud to be rendered using this screen object. Any updates sent to the cloud will now be shown on screen.

Parameters

cloud	

5.10.3.2 Run()

```
int Screen::Run ( )  [inline]
```

Blocking call; boots up the visualizer and runs until it completes.

Returns

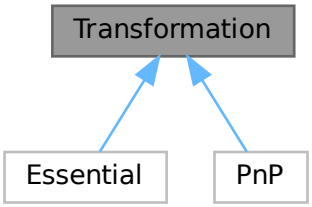
int the return code of the visualizer; 0 if success, nonzero otherwise

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/screen.hpp

5.11 Transformation Class Reference

Inheritance diagram for Transformation:



Public Member Functions

- **Transformation** (cv::Mat &T)
- **Transformation** (cv::Mat R, cv::Mat t)
- cv::Mat **GetTransformation** ()
- Eigen::MatrixXd **GetEigen** ()
- void **SetTransformation** (cv::Mat T)
- void **SetTransformation** (Eigen::MatrixXd T)
- cv::Mat **GetRotation** ()
- cv::Mat **GetTranslation** ()
- cv::Mat **GetInverseTransformation** ()
- double **GetValidFraction** ()
- cv::Mat **GetCVMat** () const
- virtual void **Estimate** ()

Protected Attributes

- cv::Mat **T_** = cv::Mat1d(4,4)
- cv::Mat **inlierMask**

Friends

- Transformation **operator**∗ (const Transformation &op1, const Transformation &op2)

The documentation for this class was generated from the following file:

- /home/jere/cpp_visual_slam/src/transformation.hpp

Chapter 6

File Documentation

6.1 frame.hpp

```
1 #ifndef VISUAL_SLAM_FRAME
2 #define VISUAL_SLAM_FRAME
3
4 #include <iostream>
5 #include <map>
6 #include "helper_functions.hpp"
7 #include <opencv2/core/core.hpp>
8 #include <opencv2/features2d/features2d.hpp>
9 #include <opencv2/highgui/highgui.hpp>
10 //#include <opencv2/xfeatures2d.hpp>
11
12 class FeatureExtractor{
13 public:
14     FeatureExtractor(){};
15
16     // takes video frame as input, outputs vector with keypoints as first element and corresponding
       descriptors as second element
17     std::tuple<cv::Mat, cv::Mat> compute_features(const cv::Mat& img){ //std::tuple<cv::MatrixXd,
       cv::MatrixXd>
18         std::vector<cv::KeyPoint> keypoints;
19         detector->detect ( img,keypoints );
20         cv::Mat descriptors;
21         descriptor->compute ( img, keypoints, descriptors);
22         /*
23         cv::Mat output;
24         cv::drawKeypoints(img, keypoints, output);
25         cv::imwrite("../ORB_result.jpg", output);
26         */
27         return std::tuple(KeyPoint2Mat(keypoints), descriptors);
28     }
29
30 private:
31     cv::Ptr<cv::FeatureDetector> detector = cv::ORB::create(1500);
32     //cv::Ptr<cv::SIFT> detector = cv::SIFT::create(1500);
33     cv::Ptr<cv::DescriptorExtractor> descriptor = cv::ORB::create(1500);
34     //cv::Ptr<cv::SIFT> descriptor = cv::SIFT::create(1500);
35 };
36
37 class FeatureMatcher{
38 public:
39     FeatureMatcher(){
40         matcher = cv::BFMatcher(cv::NORM_L2, false);
41     };
42     std::tuple<std::vector<cv::DMatch>, cv::Mat, cv::Mat, cv::Mat, cv::Mat>
43     match_features(cv::Mat kp1, cv::Mat desc1, cv::Mat kp2, cv::Mat desc2, float ratio = 0.80){
44         std::vector<std::vector< cv::DMatch > > rawMatches;
45         //matcher.match(descriptors1, descriptors2, matches);
46         matcher.knnMatch(desc1, desc2, rawMatches, 2);
47         // perform Lowe's ratio test to get actual matches
48         std::vector<cv::DMatch> matches;
49         cv::Mat pts1;
50         cv::Mat pts2;
51         cv::Mat ft1;
52         cv::Mat ft2;
53         for(auto it = rawMatches.begin(); it != rawMatches.end(); it++){
54             if( (*it)[0].distance < ratio * (*it)[1].distance ){
55                 pts1.push_back( kp1.row((*it)[0].queryIdx) );
56                 pts2.push_back( kp2.row((*it)[0].trainIdx) );
```

```
57                 ft1.push_back( desc1.row((*it)[0].queryIdx) );
58                 ft2.push_back( desc2.row((*it)[0].trainIdx) );
59                 matches.push_back((*it)[0]);
60             }
61         }
62         return std::tuple(matches, pts1, ft1, pts2, ft2);
63     }
64 private:
65     cv::BFMatcher matcher;
66 
67 };
68 
76 class Frame{
77 public:
80     Frame(){};
85     Frame(cv::Mat rgb_img, int id){
86         //std::cout << "Base constructor called " << std::endl;
87         rgb = rgb_img;
88         ID = id;
89         keyframe = false;
90 
91     }
96     Frame(std::string rgb_path, int id){
97         //std::cout << "Base constructor called " << std::endl;
98         rgb = cv::imread(rgb_path);
99         ID = id;
100         keyframe = false;
101 
102     }
103 
104 
105 
106 //     /** Copy constructor
107 //         @param f std::shared_ptr<Frame> smart shared pointer to Frame object
108 //         */
109 //     Frame(const std::shared_ptr<Frame> f){
110 //         //std::cout << "Copy constructor called " << std::endl;
111 //         rgb = f->rgb;
112 //         keypoints = f->keypoints;
113 //         features = f->features;
114 //         pose = f->pose;
115 //         ID = f->ID;
116 //         parents = f->parents;
117 //         keyframe = f->keyframe;
118 
119 //     }
120 
121 
122 //     /** method operator= performs copy assignment
123 //     * @param t constant reference to Frame object
124 //     * @returns reference to t
125 //     */
126 //     Frame& operator=(const Frame& t)
127 //     {
128 //         //std::cout << "Assignment operator called " << std::endl;
129 //         return *this;
130 //     }
131 
132 
141     static std::tuple<std::vector<cv::DMatch>, cv::Mat, cv::Mat, cv::Mat, cv::Mat>
    Match2Frames(std::shared_ptr<Frame> prev_frame, std::shared_ptr<Frame> cur_frame, FeatureMatcher
    feature_matcher){
142         return feature_matcher.match_features(prev_frame->GetKeyPoints(), prev_frame->GetFeatures(),
    cur_frame->GetKeyPoints(), cur_frame->GetFeatures());
143     }
144 
145 
150     void AddParent(int parent_frame_id, cv::Mat transition){
151         parents.insert({parent_frame_id, transition});
152     }
153 
154 
155 
156 
157 
161     std::vector<int> GetParentIDs(){
162         std::vector<int> keys;
163         for(auto it = parents.begin(); it != parents.end(); it++) {
164             keys.push_back(it->first);
165             //std::cout << "Key: " << it->first << std::endl();
166         }
167         return keys;
168     }
169 
173     cv::Mat GetTransitionWithParentID(int parent_id){
174         return parents[parent_id];
175     }
```

```
176
177
181     std::tuple<cv::Mat, cv::Mat>  feature_extract(cv::Mat rgb_img, FeatureExtractor feature_extractor){
182         return feature_extractor.compute_features(rgb_img);
183     }
184
185
190     std::tuple<cv::Mat, cv::Mat, cv::Mat> process_frame(FeatureExtractor feature_extractor){
191         std::tuple<cv::Mat, cv::Mat> ft;
192         ft = this->feature_extract(rgb, feature_extractor);
193         SetKeyPoints(std::get<0>(ft)); // set private vars with setter
194         SetFeatures(std::get<1>(ft));
195         return std::tuple(std::get<0>(ft), std::get<1>(ft), rgb);
196     }
197
202     void process(FeatureExtractor feature_extractor){
203         std::tuple<cv::Mat, cv::Mat> ft;
204         ft = this->feature_extract(rgb, feature_extractor);
205         SetKeyPoints(std::get<0>(ft)); // set private vars with setter
206         SetFeatures(std::get<1>(ft));
207     }
208
209
210
215     cv::Mat GetRGB() const{
216         return rgb;
217     }
222     void SetRGB(cv::Mat new_rgb){
223         rgb = new_rgb;
224     }
225
230     void AddPose(cv::Mat init_pose){
231         pose = init_pose;
232     }
233
238     void UpdatePose(cv::Mat new_pose){
239         pose = new_pose;
240     }
241
246     cv::Mat GetPose() const{
247         return pose;
248     }
253     bool IsKeyFrame() const{
254         return keyframe;
255     }
256
260     void SetAsKeyFrame(){
261         keyframe = true;
262     }
263
268     void SetKeyPoints(cv::Mat new_points){
269         keypoints = new_points;
270     }
271
276     cv::Mat GetKeyPoints() const{
277         return keypoints;
278     }
279
284     std::vector<cv::KeyPoint> GetKeyPointsAsVector() const{
285         std::vector<cv::KeyPoint> vector_of_kp;
286         for(int i = 0; i < keypoints.rows; i++){
287             cv::KeyPoint kp;
288             kp.pt.x = keypoints.at<double>(i,0);
289             kp.pt.y = keypoints.at<double>(i,1);
290             vector_of_kp.push_back(kp);
291         }
292         return vector_of_kp;
293     }
294
295     static std::vector<cv::KeyPoint> GetKeyPointsAsVector(cv::Mat mat_keypoints){
296         std::vector<cv::KeyPoint> vector_of_kp;
297         for(int i = 0; i < mat_keypoints.rows; i++){
298             cv::KeyPoint kp;
299             kp.pt.x = mat_keypoints.at<double>(i,0);
300             kp.pt.y = mat_keypoints.at<double>(i,1);
301             vector_of_kp.push_back(kp);
302         }
303         return vector_of_kp;
304     }
305
310     void SetFeatures(cv::Mat new_features){
311         features = new_features;
312     }
313
318     cv::Mat GetFeatures() const{
319         return features;
320     }
```

```
321
326     int GetID() const{
327         return ID;
328     }
333     void AddID(int new_id){
334         ID = new_id;
335     }
336
341     cv::Mat GetCameraCenter(){
342         return GetTranslation(pose);
343     }
344
345 private:
346     cv::Mat rgb;
347     cv::Mat keypoints;
348     cv::Mat features;
349     cv::Mat pose;
350     int ID;
351     std::map<int, cv::Mat> parents;
352     bool keyframe = false;
353 };
354
355 #endif
```

6.2 helper_functions.hpp

```
1 #ifndef VISUAL_SLAM_HELPER_FUNCTIONS
2 #define VISUAL_SLAM_HELPER_FUNCTIONS
3
4
5 #include "isometry3d.hpp"
6 #include <iostream>
7 #include <vector>
8 #include <Eigen/Dense>
9
10
11 #include <tuple>
12 #include <map>
13 #include <opencv2/calib3d.hpp>
14 #include <opencv2/core/core.hpp>
15 #include <opencv2/features2d/features2d.hpp>
16 #include <opencv2/core/eigen.hpp>
17 #include <opencv2/highgui/highgui.hpp>
18
19 #ifndef G2O_USE_VENDORED_CERES
20 #define G2O_USE_VENDORED_CERES
21 #endif
22
23 #include "g2o/config.h"
24 #include "g2o/core/block_solver.h"
25 #include "g2o/core/optimization_algorithm_levenberg.h"
26 #include "g2o/solvers/eigen/linear_solver_eigen.h"
27 #include "g2o/types/sba/types_six_dof_expmap.h"
28 #include "g2o/core/robust_kernel_impl.h"
29 #include "g2o/solvers/dense/linear_solver_dense.h"
30 #include "g2o/types/sim3/types_seven_dof_expmap.h"
31 #include "g2o/core/solver.h"
32 #include "g2o/core/sparse_optimizer.h"
33 #include "g2o/solvers/dense/linear_solver_dense.h"
34
35
36
41 cv::Mat MakeHomogeneous(cv::Mat x) {
42     cv::Mat col_of_ones = cv::Mat::ones(x.rows, 1, CV_64F);
43     cv::Mat ret;
44     cv::hconcat(x, col_of_ones, ret);
45     return ret;
46 }
47
48
54 cv::Mat CameraProjectionMatrix2(cv::Mat Pose,cv::Mat K) {
55     return K.t()*Pose(cv::Rect(0,0,4,3));
56 }
57
62 cv::Mat GetRotation(cv::Mat T_) {
63     cv::Mat R = (cv::Mat1d(3,3) <<  T_.at<double>(0,0), T_.at<double>(0,1),
       T_.at<double>(0,2),T_.at<double>(1,0), T_.at<double>(1,1), T_.at<double>(1,2),T_.at<double>(2,0),
       T_.at<double>(2,1), T_.at<double>(2,2));
64     return R;
65 }
70 cv::Mat GetTranslation(cv::Mat T_) {
71     cv::Mat t =  (cv::Mat1d(3,1) << T_.at<double>(0,3), T_.at<double>(1,3), T_.at<double>(2,3));
72     return t;
```

```
73 }
74 /*Taken partly from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
75
85 cv::Mat triangulate(cv::Mat pose1, cv::Mat pose2,cv::Mat pts1,cv::Mat pts2, cv::Mat K, cv::Mat&
       inlierMask) {
86     cv::Mat ret;
87
88     cv::Mat Rcw1 = GetRotation(pose1);
89     cv::Mat Rwc1 = Rcw1.t();
90     cv::Mat tcw1 = GetTranslation(pose1);;
91     cv::Mat Tcw1(3,4,CV_64F);
92     Rcw1.copyTo(Tcw1.colRange(0,3));
93     tcw1.copyTo(Tcw1.col(3));
94
95     //cv::Mat Ow1 = mpCurrentKeyFrame->GetCameraCenter();
96
97     cv::Mat Rcw2 = GetRotation(pose2);
98     cv::Mat Rwc2 = Rcw2.t();
99     cv::Mat tcw2 = GetTranslation(pose2);
100     cv::Mat Tcw2(3,4,CV_64F);
101     Rcw2.copyTo(Tcw2.colRange(0,3));
102     tcw2.copyTo(Tcw2.col(3));
103
104     double fx = 535.4; double fy = 539.2; double cx = 320.1; double cy = 247.6; double invfx = 1.0/fx;
       double invfy = 1.0/fy;
105     for(int i = 0; i < pts1.rows; i++) {
106         cv::Mat xn1 = (cv::Mat_<double>(3,1) << (pts1.at<double>(i, 0)-cx)*invfx, (pts1.at<double>(i,
       1)-cy)*invfy, 1.0);
107         cv::Mat xn2 = (cv::Mat_<double>(3,1) << (pts2.at<double>(i, 0)-cx)*invfx, (pts2.at<double>(i,
       1)-cy)*invfy, 1.0);
108         cv::Mat x3D;
109         cv::Mat A(4,4,CV_64F);
110         A.row(0) = xn1.at<double>(0)*Tcw1.row(2)-Tcw1.row(0);
111         A.row(1) = xn1.at<double>(1)*Tcw1.row(2)-Tcw1.row(1);
112         A.row(2) = xn2.at<double>(0)*Tcw2.row(2)-Tcw2.row(0);
113         A.row(3) = xn2.at<double>(1)*Tcw2.row(2)-Tcw2.row(1);
114         cv::Mat w,u,vt;
115         cv::SVD::compute(A,w,u,vt,cv::SVD::MODIFY_A| cv::SVD::FULL_UV);
116         x3D = vt.row(3).t();
117
118         // Euclidean coordinates
119         x3D = x3D.rowRange(0,3)/x3D.at<double>(3);
120
121         cv::Mat x3Dt = x3D.t();
122         ret.push_back(x3D.t());
123         // Get camera centers
124
125         // NOTICE! THIS IS BASICALLY COPIED FROM ORB-SLAM
126         cv::Mat Ow1 = GetTranslation(pose1);
127         cv::Mat Ow2 = GetTranslation(pose2);
128         //Check triangulation in front of cameras
129         double z1 = Rcw1.row(2).dot(x3Dt)+tcw1.at<double>(2);
130         if(z1<=0){
131             inlierMask.push_back((uchar)0);continue;
132         }
133
134
135         double z2 = Rcw2.row(2).dot(x3Dt)+tcw2.at<double>(2);
136         if(z2<=0){
137             inlierMask.push_back((uchar)0);continue;
138         }
139
140         //Check reprojection error in first keyframe
141         const double &sigmaSquare1 = 1;//mpCurrentKeyFrame->mvLevelSigma2[kp1.octave];
142         const double x1 = Rcw1.row(0).dot(x3Dt)+tcw1.at<double>(0);
143         const double y1 = Rcw1.row(1).dot(x3Dt)+tcw1.at<double>(1);
144         const double invz1 = 1.0/z1;
145
146
147         double u1 = fx*x1*invz1+cx;
148         double v1 = fy*y1*invz1+cy;
149         double errX1 = u1 - pts1.at<double>(i, 0);
150         double errY1 = v1 - pts1.at<double>(i, 1);
151         if((errX1*errX1+errY1*errY1)>5.991*sigmaSquare1){
152             inlierMask.push_back((uchar)0);continue;
153         }
154
155
156         //Check reprojection error in second keyframe
157         const double sigmaSquare2 = 1; //pKF2->mvLevelSigma2[kp2.octave];
158         const double x2 = Rcw2.row(0).dot(x3Dt)+tcw2.at<double>(0);
159         const double y2 = Rcw2.row(1).dot(x3Dt)+tcw2.at<double>(1);
160         const double invz2 = 1.0/z2;
161
162         double u2 = fx*x2*invz2+cx;
163         double v2 = fy*y2*invz2+cy;
164         double errX2 = u2 - pts2.at<double>(i, 0);
```

```
165         double errY2 = v2 - pts2.at<double>(i, 1);
166         if((errX2*errX2+errY2*errY2)>5.991*sigmaSquare2){
167             inlierMask.push_back((uchar)0);continue;
168         }
169 
170 
171         //Check scale consistency
172         cv::Mat normal1 = x3D-Ow1;
173         double dist1 = cv::norm(normal1);
174 
175         cv::Mat normal2 = x3D-Ow2;
176         double dist2 = cv::norm(normal2);
177 
178         if(dist1==0 || dist2==0){
179             inlierMask.push_back((uchar)0);continue;
180         }
181 
182         // NOTICE! THIS IS BASICALLY COPIED FROM ORB-SLAM UP TO HERE
183 
184         // /*const double ratioDist = dist2/dist1;
185         // const double ratioOctave =
    mpCurrentKeyFrame->mvScaleFactors[kp1.octave]/pKF2->mvScaleFactors[kp2.octave];
186 
187         // /*if(fabs(ratioDist-ratioOctave)>ratioFactor)
188         //     continue;*/
189         // if(ratioDist*ratioFactor<ratioOctave || ratioDist>ratioOctave*ratioFactor)
190         //     continue;
191 
192         // */
193         inlierMask.push_back((uchar)1);
194     }
195     return ret;
196 }
197 
198 
204 std::vector<int> GetListDiff(cv::Mat kp1, cv::Mat kp2) {
205     std::vector<int> idx_list;
206     bool found = false;
207     double eps = 1; // up to numerical instabilities
208     for(int i_kp1 = 0; i_kp1 < kp1.rows; i_kp1++){
209         found=false;
210         for(int i_kp2 = 0; i_kp2 < kp2.rows; i_kp2++){
211             if( (std::abs(kp1.at<double>(i_kp1,0) - kp2.at<double>(i_kp2,0)) < eps) &&
    (std::abs((kp1.at<double>(i_kp1,1) - kp2.at<double>(i_kp2,1))) < eps) ) {
212                 found = true;
213             }
214         }
215         if (found==false) {
216             idx_list.push_back(i_kp1);
217         }
218     }
219     return idx_list;
220 }
221 
222 
223 // get rows from m according to queryIdx in matches
229 cv::Mat GetQueryMatches(cv::Mat m, std::vector<cv::DMatch> matches){
230     cv::Mat matched_m;
231     for(auto it = matches.begin(); it != matches.end(); it++){
232         matched_m.push_back( m.row((*it).queryIdx));
233     }
234     return matched_m;
235 }
236 
237 std::vector<int> GetQueryMatches(std::vector<int> point_ids, std::vector<cv::DMatch> matches){
238     std::vector<int> matching_point_ids;
239     for(auto it = matches.begin(); it != matches.end(); it++){
240         matching_point_ids.push_back( point_ids[(*it).queryIdx]);
241     }
242     return matching_point_ids;
243 }
244 
250 cv::Mat GetImagePointsWithIdxList(std::vector<int> idx_list, cv::Mat image_points){
251     cv::Mat new_image_points;
252     for(auto it = idx_list.begin(); it != idx_list.end(); it++){
253         //new_image_points.push_back(image_points.at<double>(*it));
254         new_image_points.push_back(image_points.row(*it));
255     }
256     return new_image_points;
257 }
263 cv::Mat GetImageDescWithIdxList(std::vector<int> idx_list, cv::Mat image_points){
264     cv::Mat new_image_points;
265     for(auto it = idx_list.begin(); it != idx_list.end(); it++){
266         new_image_points.push_back(image_points.row(*it));
267     }
268     return new_image_points;
269 }
```

```
270
271
277 cv::Mat MaskMat(cv::Mat inFrame, cv::Mat mask){
278     cv::Mat outFrame;
279     //inFrame.copyTo(outFrame, mask);
280     //return outFrame;
281     for(int i = 0; i < inFrame.rows; i++){
282         //std::cout << (int)mask.at<uchar>(i) << std::endl;
283         int mask_val = mask.at<uchar>(i);
284         if(mask_val == 1){
285             outFrame.push_back(inFrame.row(i));
286         }
287     }
288     return outFrame;
289
290 }
291
292
293 // returns Nx2 cv::Mat
294
300 cv::Mat KeyPoint2Mat(std::vector<cv::KeyPoint> keypoints){
301     cv::Mat pointmatrix(keypoints.size(), 2, CV_64F);
302     int row = 0;
303     for (auto& kp: keypoints) {
304         pointmatrix.at<double>(row, 0) = kp.pt.x;
305         pointmatrix.at<double>(row, 1) = kp.pt.y;
306         row++;
307     }
308     return pointmatrix;
309 }
310
311
317 cv::Mat KeyPoint2MatUndistord(std::vector<cv::KeyPoint> keypoints, cv::Mat cameraMatrix, cv::Mat
        distCoeffs, bool do_undistord = false){
318     // convert to Point2f
319     std::vector<cv::Point2f> points;
320     cv::KeyPoint::convert(keypoints, points);
321
322     std::vector<cv::Point2f> outputUndistortedPoints;
323     if(do_undistord){
324         cv::undistortPoints(points, outputUndistortedPoints, cameraMatrix, distCoeffs);
325     }else{
326         outputUndistortedPoints = points;
327     }
328     // flatten
329     cv::Mat output = cv::Mat(outputUndistortedPoints.size(), 2, CV_64F, outputUndistortedPoints.data());
330     return output;
331 }
332
341 void EstimateEssential(const cv::Mat& points1, const cv::Mat& points2, const cv::Mat& K, cv::Mat&
        RelativePoseTransformation, cv::Mat& triangulatedPoints, cv::Mat& inlierMask){
342     cv::Mat E = cv::findEssentialMat(points1, points2, K, cv::RANSAC, 0.999, 2, inlierMask);
343     cv::Mat R; // Rotation
344     cv::Mat t; // translation
345     cv::Mat triangulated_points_cv(3, points1.rows, CV_64F); // 3D locations for inlier points estimated
        using triangulation and the poses recovered from essential transform
346     cv::recoverPose(E, points1, points2, K, R, t, 50, inlierMask, triangulated_points_cv);
347     Eigen::MatrixXd R_; // convert to eigen for transformation calculations
348     Eigen::VectorXd t_;
349     cv::cv2eigen(R, R_);
350     cv::cv2eigen(t, t_);
351     Eigen::MatrixXd pos = Isometry3d(R_, t_).inverse().matrix();
352     cv::eigen2cv(pos, RelativePoseTransformation);
353     //triangulatedPoints = triangulated_points_cv.t(); // transpose and return
354     // make euclidean
355     for(int i=0; i < triangulated_points_cv.cols; i++){
356         cv::Mat x3D = triangulated_points_cv.col(i);
357         triangulatedPoints.push_back( (x3D.rowRange(0,3)/x3D.at<double>(3)).t() );
358     }
359
360 }
361
362 cv::Mat segment(cv::Mat mat, int start_idx, int end_idx){
363     cv::Mat segmented_mat;
364     for(int i = 0; i < mat.cols; i++){
365         if(i>=start_idx && i<end_idx){
366             segmented_mat.push_back(mat.col(i));
367         }
368     }
369     return segmented_mat.t();
370 }
371
378 cv::Mat transformMatrix(cv::Mat rvec, cv::Mat tvec) {
379     cv::Mat R;
380     cv::Rodrigues(rvec, R);
381     cv::Mat T_temp;
382     cv::hconcat(R,tvec,T_temp); // horizontal concatenation
```

```
383     cv::Mat z  = (cv::Mat1d(1,4) << 0.0, 0.0, 0.0, 1.0);
384     cv::Mat T;
385     cv::vconcat(T_temp,z,T); // vertical   concatenation
386     return T;
387 }
388
389
390
391 /*Taken from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
392 g2o::SE3Quat toSE3Quat(const cv::Mat &cvT)
393 {
394     Eigen::Matrix<double,3,3> R;
395     R << cvT.at<double>(0,0), cvT.at<double>(0,1), cvT.at<double>(0,2),
396          cvT.at<double>(1,0), cvT.at<double>(1,1), cvT.at<double>(1,2),
397          cvT.at<double>(2,0), cvT.at<double>(2,1), cvT.at<double>(2,2);
398
399     Eigen::Matrix<double,3,1> t(cvT.at<double>(0,3), cvT.at<double>(1,3), cvT.at<double>(2,3));
400
401     return g2o::SE3Quat(R,t);
402 }
403 /*Taken from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
404 Eigen::Matrix<double,3,1> toVector3d(const cv::Mat &cvVector)
405 {
406     Eigen::Matrix<double,3,1> v;
407     v << cvVector.at<double>(0), cvVector.at<double>(1), cvVector.at<double>(2);
408
409     return v;
410 }
411
412
413 /*Taken from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
414 cv::Mat toCvMat(const Eigen::Matrix<double,4,4> &m)
415 {
416     cv::Mat cvMat(4,4,CV_64F);
417     for(int i=0;i<4;i++)
418         for(int j=0; j<4; j++)
419             cvMat.at<double>(i,j)=m(i,j);
420
421     return cvMat.clone();
422 }
423 /*Taken from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
424 cv::Mat toCvMat(const g2o::SE3Quat &SE3)
425 {
426     Eigen::Matrix<double,4,4> eigMat = SE3.to_homogeneous_matrix();
427     return toCvMat(eigMat);
428 }
429 /*Taken from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
430 cv::Mat toCvMat(const Eigen::Matrix3d &m)
431 {
432     cv::Mat cvMat(3,3,CV_64F);
433     for(int i=0;i<3;i++)
434         for(int j=0; j<3; j++)
435             cvMat.at<double>(i,j)=m(i,j);
436
437     return cvMat.clone();
438 }
439 /*Taken from Project ORB_SLAM2 (https://github.com/raulmur/ORB_SLAM2)*/
440 cv::Mat toCvMat(const Eigen::Matrix<double,3,1> &m)
441 {
442     cv::Mat cvMat(3,1,CV_64F);
443     for(int i=0;i<3;i++)
444             cvMat.at<double>(i)=m(i);
445
446     return cvMat.clone();
447 }
448
449
450
456 cv::Mat NormalizeTranslation(cv::Mat P, double median_depth){
457     P.at<double>(0,3) = P.at<double>(0,3)/median_depth;
458     P.at<double>(1,3) = P.at<double>(1,3)/median_depth;
459     P.at<double>(2,3) = P.at<double>(2,3)/median_depth;
460     return P;
461 }
462
463
468 cv::Mat Points2Homogeneous(cv::Mat points3D){
469     cv::Mat points3D_euclidean;
470     for(int i=0; i < points3D.rows; i++){
471         cv::Mat x3D = points3D.row(i).t();
472         x3D = x3D.rowRange(0,3)/x3D.at<double>(3);
473         points3D_euclidean.push_back(x3D.t());
474     }
475     return points3D_euclidean;
476 }
477
478
```

```
479
480 #endif
```

6.3 isometry3d.hpp

```
1 #ifndef VISUAL_SLAM_ISOMETRY3D
2 #define VISUAL_SLAM_ISOMETRY3D
3
4 #include <iostream>
5 #include <vector>
6 #include <Eigen/Dense>
7
8 #include <tuple>
9 #include <map>
10 using Eigen::MatrixXd;
11
12 class Isometry3d{
13 public:
14     Isometry3d(Eigen::MatrixXd R, Eigen::VectorXd t) : R_(R), t_(t) {}
15
16     Eigen::MatrixXd matrix() {
17         Eigen::MatrixXd mat = Eigen::Matrix<double, 4,4>::Identity();
18         //Eigen::Matrix <double, 4, 4> mat;
19         mat.block<3,3>(0,0) = R_;
20         mat.block<3,1>(0,3) = t_;
21         return mat;
22     }
23
24     Isometry3d inverse() {
25         return Isometry3d(this->R_.transpose(), -this->R_.transpose() * this->t_);
26     }
27
28     Eigen::MatrixXd orientation() {
29         return R_;
30     }
31
32     Eigen::VectorXd position() {
33         return t_;
34     }
35
36 private:
37     Eigen::MatrixXd R_;
38     Eigen::VectorXd t_;
39 };
40
41 #endif
```

6.4 map.hpp

```
1 #ifndef VISUAL_SLAM_MAP
2 #define VISUAL_SLAM_MAP
3
4 #include "screen.hpp"
5 #include "isometry3d.hpp"
6 #include "frame.hpp"
7 #include "point.hpp"
8 #include "helper_functions.hpp"
9 #include "transformation.hpp"
10
11 #include <iostream>
12 #include <vector>
13
14 #include <chrono>
15 #include <Eigen/Dense>
16
17 #include <tuple>
18 #include <map>
19 #include <opencv2/core/eigen.hpp>
20 #include <opencv2/videoio.hpp>
21
22 #ifndef G2O_USE_VENDORED_CERES
23 #define G2O_USE_VENDORED_CERES
24 #endif
25
26 #include "g2o/config.h"
27 #include "g2o/core/block_solver.h"
28 #include "g2o/core/optimization_algorithm_levenberg.h"
29 #include "g2o/solvers/eigen/linear_solver_eigen.h"
30 #include "g2o/types/sba/types_six_dof_expmap.h"
```

```
31 #include "g2o/core/robust_kernel_impl.h"
32 #include "g2o/solvers/dense/linear_solver_dense.h"
33 #include "g2o/types/sim3/types_seven_dof_expmap.h"
34 #include "g2o/core/solver.h"
35 #include "g2o/core/sparse_optimizer.h"
36 #include "g2o/solvers/dense/linear_solver_dense.h"
37
38 //#include "helper_functions.hpp"
39
47 class Map {
48     public:
58         void InitializeMap(std::vector<std::filesystem::path>::iterator& input_video_it, int& id_frame,
      int& id_point, FeatureExtractor feature_extractor, FeatureMatcher feature_matcher, cv::Mat
      cameraIntrinsicsMatrix, bool visualize = false){
59             // store first frame as prev_frame
60             cv::Mat img, dispImg;
61             img = cv::imread(*input_video_it);
62             //std::shared_ptr<Frame> prev_frame(new Frame(img, id_frame)); // create frame object out of
      image
63             std::shared_ptr<Frame> prev_frame = std::make_shared<Frame>(img, id_frame);
64             prev_frame->process(feature_extractor);
65             prev_frame->SetAsKeyFrame();
66             prev_frame->AddPose(cv::Mat::eye(4,4,CV_64F)); // add Identity as initial pose
67             (*this).AddFrame(id_frame, prev_frame); // add to map
68             id_frame++;
69             // read next image
70             input_video_it++;
71             cv::Mat image;
72             image = cv::imread(*input_video_it);
73             while(!img.empty()){
74                 //std::shared_ptr<Frame> cur_frame(new Frame(image, id_frame)); // create frame object
      out of image
75                 std::shared_ptr<Frame> cur_frame = std::make_shared<Frame>(image, id_frame);
76                 image = cv::imread(*input_video_it);
77                 input_video_it++;
78                 cur_frame->process(feature_extractor);
79                 std::vector<cv::DMatch> matches; cv::Mat preMatchedPoints; cv::Mat preMatchedFeatures;
      cv::Mat curMatchedPoints; cv::Mat curMatchedFeatures;
80                 std::tuple<std::vector<cv::DMatch>, cv::Mat , cv::Mat, cv::Mat , cv::Mat> match_info
81                      = Frame::Match2Frames(prev_frame, cur_frame, feature_matcher);
82                 // parse tuple to objects
83                 matches = std::get<0>(match_info); preMatchedPoints = std::get<1>(match_info);
      preMatchedFeatures = std::get<2>(match_info);
84                 curMatchedPoints = std::get<3>(match_info); curMatchedFeatures = std::get<4>(match_info);
85                 // draw matches
86                 if(visualize){
87                     cv::drawMatches(prev_frame->GetRGB(), prev_frame->GetKeyPointsAsVector(),
88                     cur_frame->GetRGB(), cur_frame->GetKeyPointsAsVector(), matches, dispImg);
89                     cv::imshow("Display Image", dispImg);
90                     cv::waitKey(1);
91                 }
92                 if(matches.size() < 100){
94                     continue;
95                 }
96                 //Essential transformation = Essential();
97                 //transformation.Estimate(preMatchedPoints, curMatchedPoints, cameraIntrinsicsMatrix);
99
100                 cv::Mat inlierMask;
101                 cv::Mat RelativePoseTransformation, TriangulatedPoints;
102                 EstimateEssential(preMatchedPoints, curMatchedPoints, cameraIntrinsicsMatrix,
      RelativePoseTransformation, TriangulatedPoints, inlierMask);
103                 if(cv::sum(inlierMask)[0]/preMatchedPoints.rows < 0.9 ){
105                     continue;
106                 }
107
108                 // new keyframe is found
109                 cur_frame->SetAsKeyFrame();
110                 cv::Mat cur_pose = RelativePoseTransformation; // shortcut as previous pose is identity,
      inverse because opencv treats transformation as "where the points move" instead of "where the camera
      moves"
111                 // Adds cur frame to map with estimated pose, parent frame, and relative pose
      transformation between parent and frame
112
113                 (*this).AddPoseNode(id_frame, cur_frame, cur_pose, id_frame - 1,
      RelativePoseTransformation);
114                 id_frame++;
115                 // get inliers and turn to eigen matrices
116                 (*this).AddPoints3D(id_point, TriangulatedPoints, prev_frame, preMatchedPoints,
      preMatchedFeatures, cur_frame, curMatchedPoints, curMatchedFeatures, inlierMask);
117                 // at end of loop
118                 break;
119             }
120         }
121
136         void localTracking(std::vector<std::filesystem::path>::iterator& input_video_it, int& id_frame,
      int& id_point, FeatureExtractor feature_extractor, FeatureMatcher feature_matcher, cv::Mat
      cameraIntrinsicsMatrix, cv::Mat DistCoefficients, PointClouds& clouds, bool visualize = true, bool
```

```
    verbose_optimization = false){
137             //Get the map points that the last keyframe sees
138             int lastkeyframe_idx = id_frame-1;
139             std::tuple<cv::Mat, cv::Mat, cv::Mat, std::vector<int» map_points =
    GetImagePointsWithFrameID(lastkeyframe_idx); // get information of the points the last keyframe sees
140             // start tracking
141             input_video_it++;
142             cv::Mat image;
143             image = cv::imread(*input_video_it);
144             int trackFrameCount = 0;
145             while(!image.empty()){
146                 // create Frame object from video frame and increase videoframe iterator
147                 //std::shared_ptr<Frame> cur_frame(new Frame(image, id_frame)); // create frame object
    out of image
148                 std::shared_ptr<Frame> cur_frame = std::make_shared<Frame>(image, id_frame);
149                 image = cv::imread(*input_video_it);
150                 input_video_it++;
151                 cur_frame->process(feature_extractor);
152                 // pass imagepoints of map points (last keyframe), descriptors of map points (last
    keyframe), current frame imagepoints, and current frame for feature matching
153                 std::vector<cv::DMatch> matches; cv::Mat preMatchedPoints; cv::Mat preMatchedFeatures;
    cv::Mat curMatchedPoints; cv::Mat curMatchedFeatures;
154                 std::tuple<std::vector<cv::DMatch>, cv::Mat, cv::Mat, cv::Mat, cv::Mat> match_info =
    feature_matcher.match_features(std::get<0>(map_points), std::get<1>(map_points),
    cur_frame->GetKeyPoints(), cur_frame->GetFeatures());
155                 // parse tuple to objects
156                 matches = std::get<0>(match_info); preMatchedPoints = std::get<1>(match_info);
    preMatchedFeatures = std::get<2>(match_info); curMatchedPoints = std::get<3>(match_info);
    curMatchedFeatures = std::get<4>(match_info);
157                 // get 3d locations of matching imagepoints and corresponding point ids
158                 // matched_3d are 3d point locations of those map points that we are able to match in
    the current frame
159                 cv::Mat matched_3d = GetQueryMatches(std::get<2>(map_points), matches);
160                 std::cout « "TRACKING " « matched_3d.rows « " POINTS" « std::endl;
161                 // corresponding_point_ids are point ids of those map points that we are able to match
    in the current frame
162                 std::vector<int> corresponding_point_ids = GetQueryMatches(std::get<3>(map_points),
    matches);
163                 cv::Mat rvec, tvec;
164                 cv::Mat inliers;
165                 cv::solvePnPRansac(matched_3d, curMatchedPoints, cameraIntrinsicsMatrix,
    DistCoefficients, rvec, tvec, false, 200, 3.0F, 0.95, inliers);
166                 if(inliers.rows<10){
167                     continue;
168                 }
169                 //std::cout « "Inliers passing solvePnPRansack: " « inliers.rows « "/" «
    curMatchedPoints.rows « std::endl;
170                 cv::Mat T = transformMatrix(rvec,tvec);
171                 cv::Mat W_T_curr = T.inv(); // From w to curr frame W_T_curr
172                 //PnP transformation = PnP();
173                 //transformation.Estimate(matched_3d, curMatchedPoints, cameraIntrinsicsMatrix,
    DistCoefficients);
174                 cv::Mat prev_T_W = (GetFrame(id_frame-1)->GetPose()).inv(); // From prev to world frame
    W_T_curr
175                 cv::Mat Relative_pose_trans = prev_T_W * W_T_curr;
176                 AddParentAndPose(id_frame-1, id_frame, cur_frame, Relative_pose_trans, W_T_curr);
177                 id_frame++;
178                 AddPointToFrameCorrespondances(corresponding_point_ids, curMatchedPoints,
    curMatchedFeatures, cur_frame, inliers);
179                 BundleAdjustement(true, false, verbose_optimization); // Do motion only (=points are
    fixed) bundleadjustement by setting tracking to true
180                 if(visualize){
181                     cv::Mat dispImg;
182                     cv::drawMatches(GetFrame(lastkeyframe_idx)->GetRGB(),
    Frame::GetKeyPointsAsVector(std::get<0>(map_points)), cur_frame->GetRGB(),
    cur_frame->GetKeyPointsAsVector(), matches, dispImg);
183                     cv::imshow("Display Image", dispImg);
184                     cv::waitKey(1);
185                 }
186                 // Check if current frame is a key frame:
187                 // 1. at least 20 frames has passed or current frame tracks less than 80 map points
188                 // 2. The map points tracked are fewer than 90% of the map points seen by the last key
    frame
189                 std::cout « ((double)inliers.rows) / ((double)std::get<0>(map_points).rows) « std::endl;
190                 if( (trackFrameCount > 20 || inliers.rows < 80) && ( (((double)inliers.rows) /
    ((double)std::get<0>(map_points).rows)) < 0.9) ){ // || ( (((double)inliers.rows) /
    ((double)std::get<0>(map_points).rows)) < 0.9) ) { //|| (inliers.rows / std::get<0>(map_points).rows
    < 0.9)){
191                 //if( (trackFrameCount > 15 && inliers.rows < 120)  ){
192                     std::cout«"New keyframe found" « std::endl;
193                     break;
194                 }
195                 trackFrameCount++;
196                 // visualize all points
197                 std::vector<cv::Mat> created_points = GetAll3DPoints();
198                 std::vector<cv::Mat> camera_locs = GetAllCameraLocations();
199                 clouds.Clear();
```

```
200                 clouds.AddPointsMatUpdate(created_points, false);
201                 clouds.AddPointsMatUpdate(camera_locs, true);
202             }
203 
204         }
205 
218         void localMapping(int& id_frame, int& id_point, FeatureExtractor feature_extractor,
    FeatureMatcher feature_matcher, cv::Mat cameraIntrinsicsMatrix, cv::Mat DistCoefficients,int&
    last_key_frame_id, bool visualize = false){
219             GetFrame(id_frame-1)->SetAsKeyFrame(); // Set lastly added frame to be a new keyframe
220             //# Get last keyframe pose from global map (from world to previous keyframe)
221             cv::Mat W_T_prev_key = GetFrame(last_key_frame_id)->GetPose();
222             cv::Mat prev_key_T_W = W_T_prev_key.inv(); // Get pose from previous keyframe to world
223             cv::Mat W_T_cur_key = GetFrame(id_frame-1)->GetPose(); // Get pose from W to curr keyframe
224             // (int parent_frame_id, cv::Mat transition)
225             GetFrame(id_frame-1)->AddParent(last_key_frame_id, prev_key_T_W*W_T_cur_key);
226             cv::Mat image_points_already_in_map =
    std::get<0>(GetImagePointsWithFrameID(last_key_frame_id));
227             cv::Mat kp1 = GetFrame(last_key_frame_id)->GetKeyPoints(); // this contains all the image
    points (matched and unmatched)
228             cv::Mat desc1 = GetFrame(last_key_frame_id)->GetFeatures();
229             std::vector<int> idx_list = GetListDiff(kp1, image_points_already_in_map); // THIS DOES NOT
    WORK PERFECTLY CORREC; BUT SUFFICIENT FOR NOW
230             // GetImagePointsWithIdxList
231             cv::Mat unmatched_kp1 = GetImagePointsWithIdxList(idx_list, kp1);
232             cv::Mat unmatched_desc1 = GetImageDescWithIdxList(idx_list, desc1);
233             std::tuple<std::vector<cv::DMatch>, cv::Mat, cv::Mat, cv::Mat, cv::Mat> match_info =
    feature_matcher.match_features(unmatched_kp1, unmatched_desc1, GetFrame(id_frame-1)->GetKeyPoints(),
    GetFrame(id_frame-1)->GetFeatures());
234             std::vector<cv::DMatch> matches; cv::Mat last_keyframe_points; cv::Mat
    last_keyframe_features; cv::Mat cur_keyframe_points; cv::Mat cur_keyframe_features;
235             matches = std::get<0>(match_info); last_keyframe_points = std::get<1>(match_info);
    last_keyframe_features = std::get<2>(match_info); cur_keyframe_points = std::get<3>(match_info);
    cur_keyframe_features = std::get<4>(match_info);
236             if(visualize){
237                 cv::Mat dispImg;
238                 cv::drawMatches(GetFrame(last_key_frame_id)->GetRGB(),
    Frame::GetKeyPointsAsVector(unmatched_kp1), GetFrame(id_frame-1)->GetRGB(),
    GetFrame(id_frame-1)->GetKeyPointsAsVector(), matches, dispImg);
239                 cv::imshow("Display Image", dispImg);
240                 cv::waitKey(1);
241             }
242             cv::Mat Proj1 = CameraProjectionMatrix2(GetFrame(last_key_frame_id)->GetPose(),
    cameraIntrinsicsMatrix);
243             cv::Mat Proj2 = CameraProjectionMatrix2(GetFrame(id_frame-1)->GetPose(),
    cameraIntrinsicsMatrix);
244             cv::Mat inlierMask;
245             cv::Mat new_triagulated_points = triangulate(GetFrame(last_key_frame_id)->GetPose(),
    GetFrame(id_frame-1)->GetPose(), last_keyframe_points, cur_keyframe_points, cameraIntrinsicsMatrix,
    inlierMask);
246 
248             std::cout << "ADDING " << cv::sum(inlierMask)[0] << " NEW MAP POINTS" << std::endl;
249             // cleanup bad points from map (seen by less than 3 frames)
250             CleanUpBadPoints();
251 
252             AddPoints3D(id_point, new_triagulated_points, GetFrame(last_key_frame_id),
    last_keyframe_points, last_keyframe_features, GetFrame(id_frame-1), cur_keyframe_points,
    cur_keyframe_features, inlierMask);
253         }
254 
255         void CleanUpBadPoints(){
256             for (auto it = point_3d_.cbegin(); it != point_3d_.cend() /* not hoisted */; /* no increment
    */)
257             {
258             if (it->second->IsBad())
259             {
260                 point_3d_.erase(it++);    // or "it = m.erase(it)" since C++11
261             }
262             else
263             {
264                 ++it;
265             }
266             }
267 
268         }
269 
270 
276         void AddFrame(int frame_id, std::shared_ptr<Frame> frame) {
277             // TODO: add warning for the duplicate
278             frames_[frame_id] = frame;
279         }
280 
285         void AddPoint3D(int point_id, std::shared_ptr<Point3D> point_3d) {
286             // TODO: add warning for the duplicate
287             if(point_3d_.find(point_id)!=point_3d_.end()){
288                 throw std::invalid_argument("Duplicate point_id in AddPoint3D");
289             }
```

```
290             point_3d_[point_id] = point_3d;
291         }
292 
293         // Adds multiple points to map with one function call, gets running indexing of points
    (id_point), which is increased inside the function,
294         // matrix of 3D point locations (Nx3), pointers to frames (1&2) that see the point
307         void AddPoints3D(int& id_point, cv::Mat points_3d, std::shared_ptr<Frame> frame1, cv::Mat uv1,
    cv::Mat desc1, std::shared_ptr<Frame> frame2, cv::Mat uv2, cv::Mat desc2, cv::Mat inlierMask){

308             for (int i = 0; i < points_3d.rows; i++) {
309                 int mask_val = inlierMask.at<uchar>(i);
310                 //std::cout « "Mask value: "« mask_val « ", z location: " « (points_3d.at<double>(i,2))
    « std::endl;
311                 if( (mask_val == 1)){
312                     // make sure it is normalized, use helper segment to make euclidean
313                     // cv::Mat location_3D = segment(points_3d.row(i) / points_3d.at<double>(i,3), 0,
    3);
314                     cv::Mat location_3D = points_3d.row(i);
315                     std::shared_ptr<Point3D> pt_object(new Point3D(id_point, location_3D));
316                     pt_object->AddFrame(frame1, uv1.row(i), desc1.row(i));
317                     pt_object->AddFrame(frame2, uv2.row(i), desc2.row(i));
318                     (*this).AddPoint3D(id_point, pt_object);
319                     id_point++;;
320                 }
321             }
322         }
323 
329         std::vector<int> GetPointsVisibleToFrame(int frame_id) {
330             std::vector<int> point_id_list;
331             for(auto it = point_3d_.begin(); it != point_3d_.end(); ++it) {
332                 if(it->second->IsVisibleTo(frame_id)){
333                     point_id_list.push_back(it->first);
334                 }
335             }
336             return point_id_list;
337         }
338 
344         std::vector<int> GetPointsVisibleToFrames(std::vector<int> frame_id_list) {
345             std::vector<int> point_id_list;
346             for(auto it = point_3d_.begin(); it != point_3d_.end(); ++it) {
347                 for (auto it2 = frame_id_list.begin(); it2 != frame_id_list.end(); ++it2) {
348                     // it contrain key values, take value which is pointer to object instance which has
    Is visible function
349                     continue;
350                     //visibility.push_back(((*it).second)->IsVisibleTo(*it2));
351                 }
352             }
353             return point_id_list;
354         }
355 
361         std::tuple<cv::Mat, cv::Mat, cv::Mat, std::vector<int» GetImagePointsWithFrameID(int frame_id) {
362             //std::vector<std::tuple<cv::Mat, cv::Mat, int» ret; // return vector of tuples
    image points, desctiptors and point 3d eigen vectors
363             std::tuple<cv::Mat, cv::Mat, cv::Mat, std::vector<int» ret; // return tuple image points,
    descriptors and point 3d vectors
364             for(auto ptr_point_obj = point_3d_.begin(); ptr_point_obj != point_3d_.end();
    ++ptr_point_obj) {
365                 if(ptr_point_obj->second->IsVisibleTo(frame_id)){
366                     // Get the imagepoint and feature corresponding to the frame
367                     std::tuple<cv::Mat, cv::Mat>  imgpoint_and_feature =
    ptr_point_obj->second->GetImagePointAndFeature(frame_id);
368                     cv::Mat location_3d = ptr_point_obj->second->Get3dPoint();
369                     int point_id = ptr_point_obj->second->GetID();
370                     std::get<0>(ret).push_back(std::get<0>(imgpoint_and_feature));
    std::get<1>(ret).push_back(std::get<1>(imgpoint_and_feature));
    std::get<2>(ret).push_back(location_3d); std::get<3>(ret).push_back(point_id);                    }
371             }
372             return ret;
373 
374         }
375 
376 
382         std::vector<cv::Mat> Get3DPointsWithIDs(std::vector<int> id_list) {
383             std::vector<cv::Mat> points;
384             for (auto it = id_list.begin(); it != id_list.end(); ++it) {
385                 points.push_back(point_3d_[*it]->Get3dPoint());
386             }
387             return points;
388         }
389 
394         std::vector<cv::Mat> GetAll3DPoints() {
395             std::vector<cv::Mat> all_points;
396             for (auto it = point_3d_.begin(); it != point_3d_.end(); ++it) {
397                 all_points.push_back(it->second->Get3dPoint());
398             }
399             return all_points;
400         }
```

```
401
405         std::vector<cv::Mat> GetAllPoses() {
406             std::vector<cv::Mat> allposes;
407             for(auto it = frames_.begin(); it != frames_.end(); ++it) {
408                 cv::Mat temp = it->second->GetPose();
409                 allposes.push_back(temp);
410             }
411             return allposes;
412         }
413
417         std::vector<cv::Mat> GetAllCameraLocations(bool keyframes_only = false) {
418                 std::vector<cv::Mat> all_cam_locs;
419                 for(auto it = frames_.begin(); it != frames_.end(); ++it) {
420                     if(keyframes_only && it->second->IsKeyFrame()){
421                         cv::Mat temp = -GetRotation(it->second->GetPose()).inv() *
    GetTranslation(it->second->GetPose());
422                         all_cam_locs.push_back(temp.t());
423                     }else if(!keyframes_only){
424                         cv::Mat temp = -GetRotation(it->second->GetPose()).inv() *
    GetTranslation(it->second->GetPose());
425                         all_cam_locs.push_back(temp.t());
426                     }
427
428                 }
429                 return all_cam_locs;
430         }
434         void UpdatePose(cv::Mat new_pose, int frame_id) {
435             // TODO check if frame with frame_id even exist yet
436             if(frames_.find(frame_id)==frames_.end()){
437                 throw std::invalid_argument("No such frame_id exists in map in UpdatePose: " +
    frame_id);
438             }
439             frames_[frame_id]->UpdatePose(new_pose); // TODO: function to convert eigen pose to open cv
    mat and wise versa
440         }
441
445         void UpdatePoint3D(cv::Mat new_point, int point_id) {
446             if(point_3d_.find(point_id)==point_3d_.end()){
447                 throw std::invalid_argument("No such point_id exists in map in UpdatePoint3D: " +
    point_id);
448             }
449             point_3d_[point_id]->UpdatePoint(new_point);
450         }
451
455         std::shared_ptr<Frame> GetFrame(int frame_id) {
456             if (frames_.find(frame_id) == frames_.end()) {
457                 std::cout « "Trying to Get non-existent frame" « std::endl;
458                 return nullptr;
459             } else {
460                 return frames_[frame_id];
461             }
462         }
466         std::shared_ptr<Point3D> GetPoint(int point_id) {
467             if (point_3d_.find(point_id) == point_3d_.end()) {
468             // not found
469             throw std::invalid_argument("No such point_id exists in map in GetPoint: " + point_id);
470             } else {
471             // found
472             return point_3d_[point_id];
473             }
474         }
475
479         void Store3DPoints(std::map<int, std::shared_ptr<Point3D» points_map) {
480             point_3d_.insert(points_map.begin(), points_map.end());
481         }
482
489         void AddParentAndPose(int parent_id, int frame_id, std::shared_ptr<Frame> frame_obj, cv::Mat
    rel_pose_trans, cv::Mat pose) {
490             frame_obj->AddParent(parent_id, rel_pose_trans);
491             frame_obj->AddPose(pose);
492             frame_obj->AddID(frame_id);
493             this->AddFrame(frame_id, frame_obj);
494         }
495
503         void AddPoseNode(int frame_id, std::shared_ptr<Frame> frame_obj, cv::Mat pose, int parent_id,
    cv::Mat rel_pose_trans) {
504             frame_obj->AddParent(parent_id, rel_pose_trans);
505             frame_obj->AddPose(pose);
506             frame_obj->AddID(frame_id);
507             this->AddFrame(frame_id, frame_obj);
508         }
509
514         std::vector<int> GetAllFrameIDs() {
515             std::vector<int> frame_id_list;
516             for(auto it = frames_.begin(); it != frames_.end(); ++it) {
517                 int frame_id = it->first;
518                 frame_id_list.push_back(frame_id);
```

```
519             }
520             return frame_id_list;
521 
522         }
523 
527         std::vector<int> GetAllPointIDs() {
528             std::vector<int> point_id_list;
529             for(auto it = point_3d_.begin(); it != point_3d_.end(); ++it) {
530                 int point_id = it->first;
531                 point_id_list.push_back(point_id);
532             }
533             return point_id_list;
534         }
542         void AddPointToFrameCorrespondances(std::vector<int> point_ids, cv::Mat image_points, cv::Mat
    descriptors, std::shared_ptr<Frame> frame_ptr, cv::Mat inliers){
543             for(int i = 0; i < inliers.rows; i++){
544                 int inlier_idx = inliers.at<int>(i,0);
545                 GetPoint(point_ids[inlier_idx])->AddFrame(frame_ptr, image_points.row(inlier_idx),
    descriptors.row(inlier_idx));
546             }
547         }
554         void BundleAdjustement(bool tracking, bool scale=false, bool verbose = false, int n_iterations =
    10){
555             double fx = 535.4; double fy = 539.2; double cx = 320.1; double cy = 247.6;
556             // set up BA solver
557             typedef g2o::BlockSolver< g2o::BlockSolverTraits<6,3> > Block;
558             std::unique_ptr<Block::LinearSolverType> linearSolver (new
    g2o::LinearSolverEigen<Block::PoseMatrixType>());
559             std::unique_ptr<Block> solver_ptr( new Block(std::move(linearSolver)));
560             g2o::OptimizationAlgorithmLevenberg* solver = new g2o::OptimizationAlgorithmLevenberg (
    std::move(solver_ptr) );
561             g2o::SparseOptimizer optimizer;
562             optimizer.setAlgorithm ( solver );
563 
564             std::vector<int> frame_id_list = this->GetAllFrameIDs();
565             std::vector<int> point_id_list = this->GetAllPointIDs();
566 
567             // loop trough all the frames
568             for(auto it = frame_id_list.begin(); it != frame_id_list.end(); ++it) {
569                 int frame_id = *it;
570                 if( this->GetFrame(frame_id)->IsKeyFrame() && !tracking){
571                     cv::Mat pose_cv = this->GetFrame(frame_id)->GetPose();
572                     g2o::VertexSE3Expmap* vSE3 = new g2o::VertexSE3Expmap();
573                     vSE3->setEstimate(toSE3Quat(pose_cv));
574                     vSE3->setId((*it)*2);
575                     vSE3->setFixed(*it==0);
576                     optimizer.addVertex(vSE3);
577                     //std::cout « "Adding to graph pose: " « frame_id « std::endl;
578                 }else if(tracking){
579                     cv::Mat pose_cv = this->GetFrame(frame_id)->GetPose();
580                     g2o::VertexSE3Expmap* vSE3 = new g2o::VertexSE3Expmap();
581                     vSE3->setEstimate(toSE3Quat(pose_cv));
582                     vSE3->setId((*it)*2);
583                     vSE3->setFixed(*it==0);
584                     optimizer.addVertex(vSE3);
585                 }
586 
587             }
588 
589             const float thHuber2D = sqrt(5.99);
590             // loop through all points and (inside) create edges to frames that see the point
591             for(auto it = point_id_list.begin(); it != point_id_list.end(); ++it) {
592                 int point_id = *it;
593                 g2o::VertexPointXYZ* vPoint = new g2o::VertexPointXYZ();
594                 vPoint->setEstimate(toVector3d((this->GetPoint(point_id)->Get3dPoint()).t()));
595                 vPoint->setId(point_id*2+1);
596                 vPoint->setMarginalized(true);
597                 vPoint->setFixed(tracking);
598                 optimizer.addVertex(vPoint);
599                 // GetFrames() returns std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat,
    cv::Mat» frames_; // map of frames that see this  particular point object
600                 auto it2_start = this->GetPoint(point_id)->GetFrames().begin();
601                 auto it2_end = this->GetPoint(point_id)->GetFrames().end();
602                 for(auto it2 = it2_start; it2 != it2_end; it2++){
603 
604                     if(optimizer.vertex((it2->first)*2) == NULL){
605                         continue;
606                     }
607 
608                     Eigen::MatrixXd uv;
609                     cv::cv2eigen(std::get<1>(it2->second).t(), uv);
611                     g2o::EdgeSE3ProjectXYZ* e = new g2o::EdgeSE3ProjectXYZ();
612                     e->setVertex(0,
    dynamic_cast<g2o::OptimizableGraph::Vertex*>(optimizer.vertex(point_id*2+1)));
613                     e->setVertex(1,
    dynamic_cast<g2o::OptimizableGraph::Vertex*>(optimizer.vertex((it2->first)*2)));
614                     e->setMeasurement(uv);
```

```
615                     e->setId(point_id+100000);
616                     e->setInformation(Eigen::Matrix2d::Identity());
617                     g2o::RobustKernelHuber* rk = new g2o::RobustKernelHuber;
618                     e->setRobustKernel(rk);
619                     rk->setDelta(thHuber2D);
620                     e->fx = fx;
621                     e->fy = fy;
622                     e->cx = cx;
623                     e->cy = cy;
624                     optimizer.addEdge(e);
625                 }
626             }
627             //optimizer.save("beforeopt.g2o");
628             optimizer.initializeOptimization();
629             optimizer.setVerbose(verbose);
630             optimizer.optimize(n_iterations);
631             //optimizer.save("afteropt.g2o");
632 
633             double median_depth = 1;
634             if(scale){
635                 int num_points = 0;
636                 std::vector<double> median_vec;
637                 for(auto it = point_id_list.begin(); it != point_id_list.end(); ++it) {
638                     int point_id = *it;
639                     g2o::VertexPointXYZ* vPoint =
    static_cast<g2o::VertexPointXYZ*>(optimizer.vertex(point_id*2+1));
641                     median_vec.push_back( cv::norm(toCvMat(vPoint->estimate()).t()) );
642                 }
643                 std::sort(median_vec.begin(), median_vec.end()); // sort so that median depth is middle
    element
644                 median_depth = median_vec[median_vec.size()/2];
645             }
646             for(auto it = frame_id_list.begin(); it != frame_id_list.end(); ++it) {
647                 int frame_id = *it;
648                 if(optimizer.vertex(frame_id*2) == NULL){
649                     continue;
650                 }
651                 g2o::VertexSE3Expmap* vSE3 =
    static_cast<g2o::VertexSE3Expmap*>(optimizer.vertex(frame_id*2));
652                 g2o::SE3Quat SE3quat = vSE3->estimate();
654                 GetFrame(frame_id)->UpdatePose(NormalizeTranslation(toCvMat(SE3quat), median_depth));
655             }
656             for(auto it = point_id_list.begin(); it != point_id_list.end(); ++it) {
657                 int point_id = *it;
658                 g2o::VertexPointXYZ* vPoint =
    static_cast<g2o::VertexPointXYZ*>(optimizer.vertex(point_id*2+1));
660                 GetPoint(point_id)->UpdatePoint(toCvMat(vPoint->estimate()).t()/median_depth);
661             }
662         }
663 
664      private:
665         std::map<int, std::shared_ptr<Frame» frames_;
666         std::map<int, std::shared_ptr<Point3D» point_3d_;
667 };
668 
669 #endif
```

6.5 point.hpp

```
1 #ifndef VISUAL_SLAM_POINT
2 #define VISUAL_SLAM_POINT
3 
4 #include <iostream>
5 #include <vector>
6 #include <Eigen/Dense>
7 #include "frame.hpp"
8 #include <tuple>
9 #include <map>
10 #include <iostream>
11 
17 class Point3D{
18 public:
23     Point3D(int ID, cv::Mat location_3d) : ID_(ID), location_3d_(location_3d) {
24         //std::cout « "Point Base constructor called " « std::endl;
25         ID_ = ID;
26         location_3d_ = location_3d;
27     }
28 
29 
33     // Point3D(const Point3D& p) {
34     //     //std::cout « "Point Copy constructor called " « std::endl;
35     //     ID_ = p.ID_;
36     //     location_3d_ = p.location_3d_;
```

```
37      //      frames_ = p.frames_;
38      // }
39
40
45      //Point3D& operator=(const Point3D& t)
46      //{
47          //std::cout << "Point Assignment operator called " << std::endl;
48         // return *this;
49      //}
50
51      // destructor
52      //~Point3D();
53
58      int GetID() {
59          return ID_;
60      }
61
66      std::shared_ptr<Frame> GetFrame(int frame_id) {
67          if (frames_.find(frame_id) == frames_.end()) {
68          // not found
69          throw std::invalid_argument("Tried to accessing non-existing frame in point to fetch frame
        pointer");
70          return nullptr;
71          } else {
72          // found
73              auto it = frames_.find(frame_id);
74              // return second element from map. ie. return value. not all values but first value that is
        frame pointer
75              return std::get<0>(it->second);
76          }
77      }
78
79      cv::Mat GetFrame2(int frame_id) {
80          if (frames_.find(frame_id) == frames_.end()) {
81          // not found
82          throw std::invalid_argument("Tried to accessing non-existing frame in point to fetch image point
        (ver2)");
83          cv::Mat ret;
84          return ret;
85          } else {
86          // found
87              auto it = frames_.find(frame_id);
88              // return second element from map. ie. return value. not all values but second value that is
        image point
89              return std::get<1>(it->second);
90          }
91      }
92
93
98      cv::Mat GetImagePoint(int frame_id) {
99          if (frames_.find(frame_id) == frames_.end()) {
100          // not found
101          throw std::invalid_argument("Tried to access non-existing frame in point to fetch image point");
102          cv::Mat ret;
103          return ret;
104          } else {
105          // found
106              auto it = frames_.find(frame_id);
107              // return second element from map. ie. return value. not all values but second value that is
        image point
108              return std::get<1>(it->second);
109          }
110      }
111
112
113      std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat>> SubsetOfFrames(int frame_id) {
114          std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat>> ret;
115          for(auto it = frames_.begin(); it != frames_.end(); ++it) {
116              if (frame_id == it->first) {
117                  ret[frame_id] = std::tuple(std::get<0>(it->second), std::get<1>(it->second),
        std::get<2>(it->second));
118                  return ret;
119              }
120          }
121      }
122
128      void AddFrame(std::shared_ptr<Frame> frame, cv::Mat uv, cv::Mat descriptor) {
129          frames_[frame->GetID()] = std::tuple(frame, uv, descriptor);
130      }
131
135      void UpdatePoint(cv::Mat new_location) {
136          location_3d_ = new_location;
137      }
138
143      bool IsVisibleTo(int frame_id) {
144          for(auto it = frames_.begin(); it != frames_.end(); ++it) {
145              if (frame_id == it->first) {
```

```
146                 return true;
147             }
148         }
149         return false;
150     }
151 
156     // get imagepoint and feature with frame id (1x)
157     std::tuple<cv::Mat, cv::Mat> GetImagePointAndFeature(int frame_id) {
158         if(frames_.find(frame_id)==frames_.end()){
159             std::cout << "Unknown frame: " << std::to_string(frame_id) << " to point: " <<
      std::to_string(ID_) << std::endl;
160             std::tuple<cv::Mat, cv::Mat> ret;
161             return ret; // return empty
162         }
163         std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat> temp = frames_[frame_id];
164         std::tuple<cv::Mat, cv::Mat> ret = std::make_tuple(std::get<1>(temp), std::get<2>(temp));
165         return ret;
166     }
167 
171     cv::Mat Get3dPoint() {
172         return location_3d_;
173     }
174 
178     int GetNVisibleFrames() {
179         return frames_.size();
180     }
184     void SetFrames(std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat» subset_of_frames)
      {
185         frames_ = subset_of_frames;
186     }
187 
191     std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat»& GetFrames(){
192         return frames_;
193     }
197     bool IsBad() {
198         if (frames_.size() < 3) {
199             return true;
200         }
201         else {
202             return false;
203         }
204     }
205 
206     friend std::ostream& operator«(std::ostream& os, const Point3D& p);
207 
208 private:
209     int ID_;
210     cv::Mat location_3d_;
211     // 3D location of thhe point
212     // map(key=Frame_ID, values = frame, image_point, descriptrors)
213     std::map<int, std::tuple<std::shared_ptr<Frame>, cv::Mat, cv::Mat» frames_;
214 };
215 
216 std::ostream& operator«(std::ostream& os, const Point3D& p){
217     os << "Point with ID: " << p.ID_ << ", 3d location: " << p.location_3d_ << ", And Point->Frame
      correspondences:\n";
218     for(auto it2 = p.frames_.begin(); it2 != p.frames_.end(); ++it2){
219         os << "Frame id: " << it2->first << std::flush << ", uv: " << std::get<1>(it2->second) << "\n" <<
      std::flush;
220     }
221     return os;
222 };
223 
224 
225 #endif
```

6.6 screen.hpp

```
1 #ifndef VISUAL_SLAM_SCREEN
2 #define VISUAL_SLAM_SCREEN
3 
4 #include <opencv2/core/core.hpp>
5 
6 #include <easy3d/viewer/viewer.h>
7 #include <easy3d/renderer/camera.h>
8 #include <easy3d/renderer/renderer.h>
9 #include <easy3d/renderer/camera.h>
10 #include <easy3d/renderer/drawable_points.h>
11 #include <easy3d/renderer/drawable_lines.h>
12 #include <easy3d/renderer/drawable_triangles.h>
13 #include <easy3d/core/types.h>
14 #include <easy3d/core/point_cloud.h>
15 #include <easy3d/util/timer.h>
```

```
16 #include <easy3d/util/logging.h>
17 #include <easy3d/util/resource.h>
18 #include <easy3d/util/initializer.h>
19
20 //using namespace easy3d;
21
22 #include <vector>
23 #include <thread>
24
33 class Screen {
34 public:
41     explicit Screen(std::string title) {
42         viewer_ = new easy3d::Viewer(title);
43         // Initialize some parameters to make the output easier to navigate
44         viewer_->camera()->setViewDirection(easy3d::vec3(1, 0, 0));
45         viewer_->camera()->setUpVector(easy3d::vec3(1, 0, 0));
46     }
53     void RegisterPointCloud(easy3d::PointCloud* cloud) {
54         viewer_->add_model(cloud);
55     }
61     int Run() {
62         std::this_thread::sleep_for(std::chrono::milliseconds(1000));
63         return viewer_->run();
64     }
65 private:
66     easy3d::Viewer * viewer_;
67 };
68
82 void ConfigurePointCloud(
83     easy3d::PointCloud * cloud,
84     float point_size = 1.0f,
85     bool plain_style = false,
86     easy3d::vec4 color = easy3d::vec4(1.0, 1.0, 1.0, 1.0)
87 ) {
88     auto drawable = cloud->renderer()->get_points_drawable("vertices");
89     drawable->set_point_size(point_size);
90     if (plain_style) {
91         drawable->set_impostor_type(easy3d::PointsDrawable::PLAIN);
92     }
93     else {
94         drawable->set_impostor_type(easy3d::PointsDrawable::SPHERE);
95     }
96     drawable->set_color(color);
97
98 }
99
108 void SpawnWorkerThread(std::function<void ()> callback) {
109     easy3d::Timer<>::single_shot(0, [&](){
110         callback();
111     });
112 }
113
129 class PointClouds {
130 public:
131     explicit PointClouds(easy3d::PointCloud* points, easy3d::PointCloud* poses) : points_(points),
    poses_(poses) {}
141     void AddPoint(double x, double y, double z, bool poses = false) {
142         // It would be possible to use the same point cloud for both points and poses,
143         // but this solution also works and requires less extra configuration.
144         if (poses) {
145             poses_->add_vertex(easy3d::vec3(x, y, z));
146         }
147         else {
148             points_->add_vertex(easy3d::vec3(x, y, z));
149         }
150     }
151
159     void AddPointMat(cv::Mat point, bool poses = false) {
160         AddPoint(point.at<double>(0), point.at<double>(1), point.at<double>(2), poses);
161     }
169     void AddPose(double x, double y, double z) {
170         AddPoint(x, y, z, true);
171     }
176     void UpdateView() {
177         points_->renderer()->update();
178         poses_->renderer()->update();
179     }
180
186     void Clear(bool poses = false) {
187         if (poses) {
188             poses_->clear();
189         }
190         else {
191             points_->clear();
192         }
193     }
194
```

```
199     void ClearAll() {
200         Clear(true);
201         Clear(false);
202     }
203 
211     void SetPointsMatUpdate(std::vector<cv::Mat> points, bool poses = false) {
212         Clear(poses);
213         AddPointsMatUpdate(points, poses);
214     }
215 
223     void AddPointsMatUpdate(std::vector<cv::Mat> points, bool poses = false) {
224         for (auto point : points) {
225             AddPointMat(point, poses);
226         }
227         UpdateView();
228     }
229 private:
230     easy3d::PointCloud * points_;
231     easy3d::PointCloud * poses_;
232 };
233 
234 #endif
```

6.7 transformation.hpp

```
1 
2 #ifndef VISUAL_SLAM_TRANSFORMATION
3 #define VISUAL_SLAM_TRANSFORMATION
4 
5 #include "isometry3d.hpp"
6 #include "frame.hpp"
7 #include "point.hpp"
8 #include "helper_functions.hpp"
9 
10 #include <iostream>
11 #include <vector>
12 
13 #include <chrono>
14 #include <Eigen/Dense>
15 
16 #include <tuple>
17 #include <map>
18 #include <opencv2/core/eigen.hpp>
19 #include <opencv2/videoio.hpp>
20 
21 class Transformation {
22     public:
23         //empty constructor
24         Transformation(){}
25         //base constructor
26         Transformation(cv::Mat& T): T_(T){}
27         //alternative constructor
28         Transformation(cv::Mat R, cv::Mat t){
29             T_(cv::Rect(0,0,3,3)) = R;
30             T_(cv::Rect(0,3,1,3)) = t;
31         }
32 
33         cv::Mat GetTransformation() {
34             return T_;
35         }
36 
37         Eigen::MatrixXd GetEigen(){
38             Eigen::MatrixXd eigen_mat;
39             cv::cv2eigen(T_, eigen_mat);
40             return eigen_mat;
41         }
42 
43 
44 
45         void SetTransformation(cv::Mat T){
46             T_ = T;
47         }
48 
49         void SetTransformation(Eigen::MatrixXd T){
50             cv::eigen2cv(T, T_);
51         }
52 
53 
54         cv::Mat GetRotation() {
55             cv::Mat R = (cv::Mat1d(3,3) <<  T_.at<double>(0,0), T_.at<double>(0,1),
       T_.at<double>(0,2),T_.at<double>(1,0), T_.at<double>(1,1), T_.at<double>(1,2),T_.at<double>(2,0),
       T_.at<double>(2,1), T_.at<double>(2,2));
56             return R;
```

```
57          }
58
59          cv::Mat GetTranslation() {
60              cv::Mat t =  (cv::Mat1d(3,3) << T_.at<double>(0,3), T_.at<double>(1,3), T_.at<double>(2,3));
61              return t;
62          }
63
64          cv::Mat GetInverseTransformation() {
65              return T_.inv();
66          }
67          // returns the fraction N_inlierPoints / N_Points, where N_inlierPoints: number of points fitting
        the estimated transformation, N_points
68          double GetValidFraction(){
69              double fraction = 0;
70              for(int i = 0; i < inlierMask.rows; i++){
71                  //std::cout << (int)mask.at<uchar>(i) << std::endl;
72                  int mask_val = inlierMask.at<uchar>(i);
73                  if(mask_val == 1){
74                      fraction += 1;
75                  }
76              }
77              fraction /= inlierMask.rows;
78              return fraction;
79          }
80
81          cv::Mat GetCVMat() const{
82              return T_;
83          }
84
85          friend Transformation operator*(const Transformation& op1, const Transformation& op2);
86
87
88          virtual void Estimate(){
89              std::cout << "Virtual Estimate() called, usually an error" << std::endl;
90          }
91
92      protected:
93          cv::Mat T_ = cv::Mat1d(4,4);
94          cv::Mat inlierMask; // needed to assess the success of transform estimation
95  };
96
97  Transformation operator*(const Transformation& op1, const Transformation& op2){
98      Transformation t = Transformation(op1.GetCVMat(), op2.GetCVMat());
99      return t;
100 }
101
102
103 class Essential : public Transformation {
104     public:
105     void Estimate(const cv::Mat& points1, const cv::Mat& points2, const cv::Mat& K){
106         cv::Mat E = cv::findEssentialMat(points1, points2, K, cv::RANSAC, 0.999, 2, inlierMask);
107         cv::Mat R; // Rotation
108         cv::Mat t; // translation
109         cv::Mat triangulated_points_cv(3, points1.rows, CV_64F); // 3D locations for inlier points
        estimated using triangulation and the poses recovered from essential transform
110         cv::recoverPose(E, points1, points2, K, R, t, 50, inlierMask, triangulated_points_cv);
111         Eigen::MatrixXd R_; // convert to eigen for transformation calculations
112         Eigen::VectorXd t_;
113         cv::cv2eigen(R, R_);
114         cv::cv2eigen(t, t_);
115         Eigen::MatrixXd pos = Isometry3d(R_, t_).matrix().inverse();
116         cv::eigen2cv(pos, T_);
117         triangulatedPoints = triangulated_points_cv.t(); // transpose and store to private member
118     }
119
120     private:
121         cv::Mat triangulatedPoints; // recoverPose also estimates 3D locations for inlier points
122
123 };
124
125 class PnP : public Transformation {
126     public:
127     void Estimate(cv::Mat matched_3d, cv::Mat curMatchedPoints, cv::Mat cameraIntrinsicsMatrix, cv::Mat
        DistCoefficients) {
128         cv::Mat rvec, tvec, inliers;
129         cv::solvePnPRansac(matched_3d, curMatchedPoints, cameraIntrinsicsMatrix, DistCoefficients, rvec,
        tvec, false, 100, 8.0F, 0.9899999999999999911, inliers);
130         T_ = transformMatrix(rvec,tvec);
131         //store inlier indices to mask
132         inlierMask = cv::Mat::zeros(curMatchedPoints.rows, 1, CV_64F);
133         for(int i = 0; i < inliers.rows; i++){
134             int inlier_idx = inliers.at<int>(i);
135             inlierMask.at<int>(inlier_idx) = 1; // conversion to uchar from int
136         }
137     }
138 };
139
```

```
140 #endif
```

Index