ARTWIN

INDUSTRY & CONSTRUCTION 4.0 SOLUTIONS

Map pivot format specifications

March, 2020

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement no 856994.

MAIN AUTHORS	
Name	**Organisation**
Nam Duong Duong, Jérôme Royan	b-com
Michal Polic	CTU
Nischita Sudharsan	SIEMENS

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement no 856994.

Project Information

TITLE	ARtwin - An AR cloud and digital twins solution for industry and construction 4.0
DURATION	October 2019 – September 2022 (36 months)
WEBSITE	artwin-project.eu
COORDINATOR	Q-PLAN INTERNATIONAL ADVISORS PC
CONTACT PERSON	Anastasia Matonaki, matonaki@qplan-intl.gr
PROJECT OVERVIEW	ARtwin aims at improving productivity and product quality, by deploying an ARCloud platform that will offer a wide variety of cutting-edge AR-based services to industry and construction 4.0.
CONSORTIUM	1. Q-PLAN INTERNATIONAL ADVISORS PC www.qplan-intl.gr – Greece 2. B-COM http://www.b-com.com – France 3. SIEMENS AKTIENGESELLSCHAFT https://new.siemens.com/global/en.html – Germany 4. CESKE VYSOKE UCENI TECHNICKE V PRAZE http://www.cvut.cz/en - Czech Republic 5. NOKIA BELL LABS FRANCE http://www.bell-labs.com/ – France 6. HOLO-INDUSTRIE 4.0 SOFTWARE GMBH http://www.holo-light.com/contact.html - Germany 7. ARTEFACTO SAS http://www.artefacto-ar.com/en/ – France

This project has received funding from the European Union's Horizon 2020 Framework Programme for Research and Innovation under Grant Agreement no 856994.

Table of Contents

Executive Summary

The present document constitutes the Map Pivot Format specification by the ARtwin project, funded by the European Union's Horizon 2020 Research and Innovation programme. The main objective of ARtwin is to provide the European **industry and construction 4.0** with a sovereign **AR Cloud platform** that integrates a set of interactive technologies and services that meet their particular needs. In this respect, the services that are foreseen to be developed include:

- Development and updating of a **unified global map** of the factory/construction site;
- Real-time updating of a 3D **Digital Twin/BIM** of a factory/construction site;
- **Localization service** to track AR devices in order to assist workers in a factory or a construction site; and
- AR **remote rendering service** for displaying complex 3D models on low resources AR devices.

The current document starts by describing the **state of the art** related to 3D Map for **camera relocalization** and **3D dense reconstruction**. Then, the document presents commercial solutions and attempts to extract information on the proprietary format they use.

Then, the current document provides a **full specification format for a 3D map** allowing both camera relocalization and dense 3D reconstruction. It addresses various method from **3D geometric based models** (e.g. Simultaneous Localization And Mapping or Structure From Motion), **retrieval model** (e.g. image retrieval), and **machine learning** based model (e.g. Convolutional Neural Network or Random Forest).

This map pivot format specification will be implemented and used by the services developed during the ARtwin project. Also, this specification will be proposed in standardization at the ETSI Industry Specification Group "Augmented Reality Framework"

1. Introduction

In recent years, Virtual/Augmented/Mixed Reality (VR/AR/MR), robotics, autonomous vehicles (self-driving) have become increasingly trendy in Industry 4.0. In particular, AR is widely used in several sectors and contexts, from consumer applications to manufacturers. AR technology, including smart glasses, provides employees with a field access to the digital twin that perfectly matches the real environment. This improves productivity, quality and safety in the workplace for some tasks.

All AR systems require a localization component in order to estimate the 6 Degree-of-Freedom (DoF) camera pose relative to the scene coordinate system. With the recent rapid development of computer vision, numerous camera pose estimation from vision-based modality have been developed and applied to AR. A common localization system combines camera relocalization and camera tracking to define camera pose. While camera tracking is an iterative process in which each current measurement is based on previous knowledge, camera relocalization is a wholly-new calculation of camera pose based on pre-built map with no temporal constraint. The visual camera relocalization is necessary to retrieve camera pose after tracking lost, rather than restarting the localization from scratch. Moreover, it allows to initialize and conserve location of augmentations that have been setup by users in AR applications. However, it requires a specific map representation which is built from previous knowledge of the scene. This map format depends on different approaches. In this document, we present a generic map pivot format which includes all representations. It is suitable for different relocalization methods.

2. State-of-the-art: Map representation

2.1 Camera relocalization methods

2.1.1 Overview

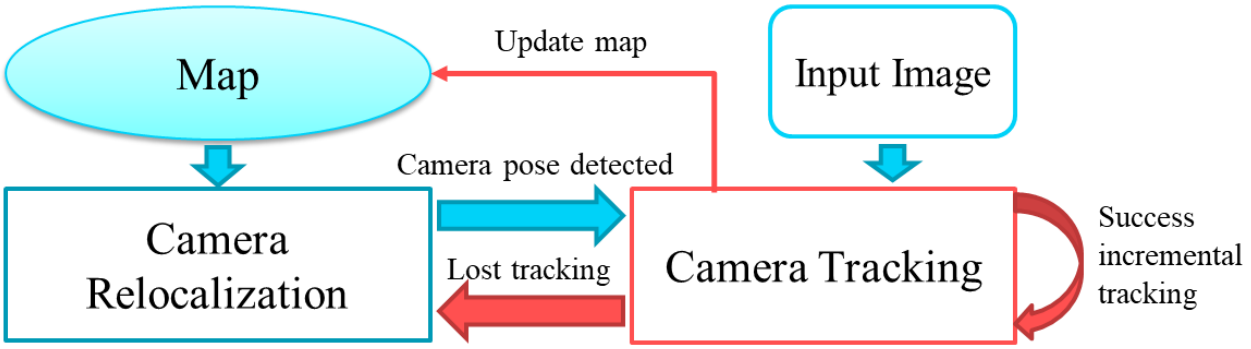


Figure 2-1. A common localization system consists of two components: camera tracking and camera relocalization.

Figure 2-1 presents a common localization system that combines camera relocalization and camera tracking to define camera pose. At beginning, the localization systems start with the camera relocalization to initialize the first camera pose in the scene coordinate system. The camera pose is then tracked by frame-to-frame. Camera relocalization is an important module in localization system. It allows an instant recovery of the camera pose in case of initialization or tracking failures. Camera relocalization leverages the map which is built from known information of a scene in order to infer camera pose from each image independently. Each camera relocalization requires a specific map representation. Therefore, this chapter first presents the state-of-the-art of camera relocalization methods according to four different approaches as shown in Figure 2-2. Then we introduce different types of data representation that is required by camera relocalization approaches. Finally, we give some conclusions and perspectives.

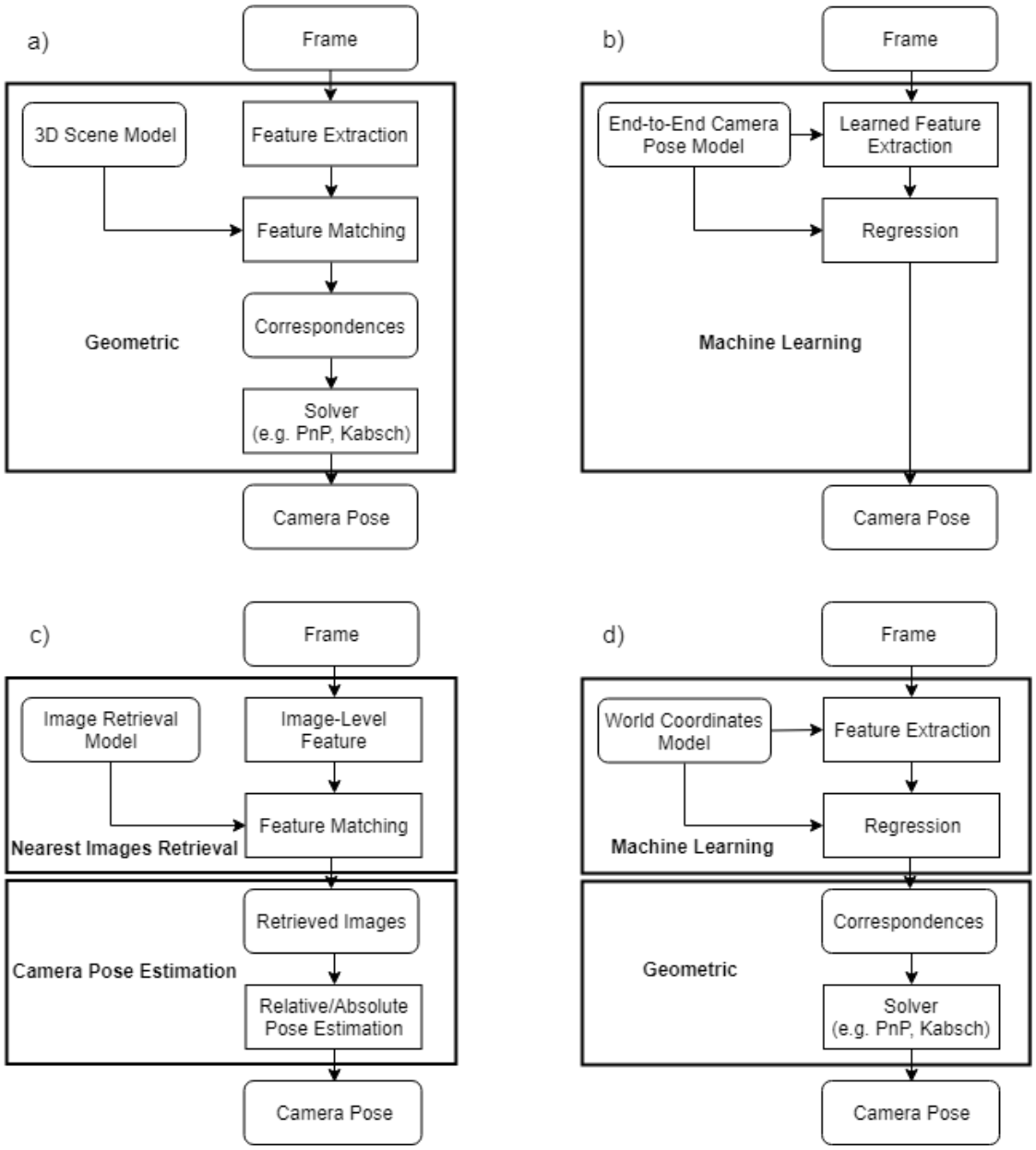


Figure 2-2. The pipelines of the state-of-the-art camera relocalization methods: a) Geometric approach; b) Machine learning approach; c) Image retrieval approach; d) Hybrid approach.

2.1.2 Geometric approaches

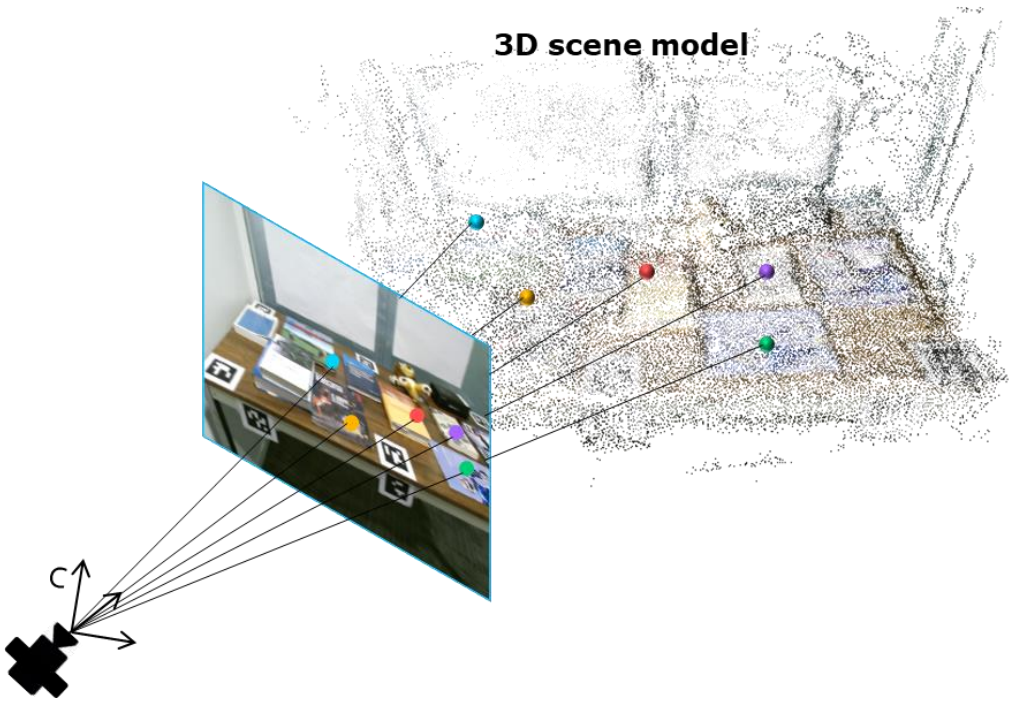


Figure 2-3. Camera relocalization based on 2D-3D point correspondences.

The common pipeline for geometric based methods consists of three principal steps as shown in Figure 2-2-a). The geometric approach for camera relocalization is based on a pre-computed 3D point cloud. Given a 3D model of the scene, the camera pose can be estimated by directly matching 2D image features from the query image to 3D points in the map to define 2D-3D point correspondences for RGB images or 3D-3D point correspondences for RGB-D images. Then, it solves a standard camera absolute pose problem by minimizing reprojection error via PnP [1]–[3] (2D-3D) or Kabsch [4] (3D-3D) algorithms.

- 2D-3D point correspondences: $\mathrm{T} = arg \min_{T \in SE(3)} \sum \left\| p_i - KT^{-1}\widehat{P}_i \right\|^2$
- 3D-3D point correspondences: $\mathrm{T} = arg \min_{T \in SE(3)} \sum \left\| P_i^c - T^{-1}\widehat{P}_i \right\|^2$

Where K is the intrinsic camera matrix. $T = [R\ t]$ is camera pose including 3D rotation and 3D translation. p_i and p_i^c are respectively 2D image coordinate and 3D camera coordinate of each feature. $\widehat{P}_i$ is 3D world coordinate matched to the corresponding feature. If the matches found are contaminated by some small portion of wrong matches (outliers), RANSAC [5] is conventionally applied to clean up the matches. Figure 2-3 demonstrates camera relocalization based on 2D-3D point correspondences.

In the following paragraphs, we present the two main steps of the geometric approach pipeline:

- Build a 3D model of the scene. This model contains a set of 3D points in the world coordinate system associated with feature vectors.
- Match 2D image features of a query image to the 3D model to define 2D-3D point correspondences.

❖ **3D model construction**

A 3D point cloud model is built from a set of images captured by one or more cameras observing a scene. The 3D model can be a sparse or dense point cloud. A dense point cloud is generally build from direct SLAM methods [6]–[15]. By using a depth sensor, a dense point cloud is created from depth maps. Camera pose is estimated by tracking points cloud using Iterative Closest Point (ICP) algorithm [8] and Lucas-Kanade algorithm [16]. However, for the camera relocalization, due to the restriction of 3D feature matching, a query image cannot match its features to the dense model. Thus, we utilize a sparse 3D point cloud associated with 2D features from RGB images, as shown in Figure 2-4. This sparse 3D point cloud is constructed thanks to offline methods or online methods.

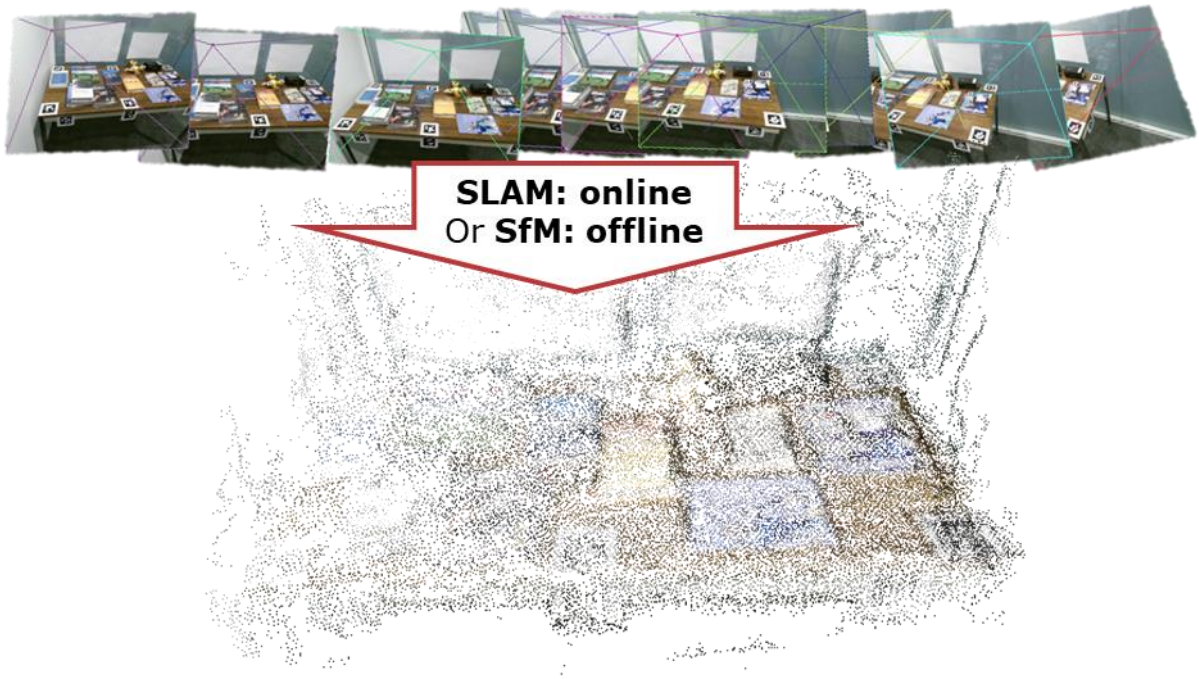


Figure 2-4. A 3D cloud point model attached to feature vectors is constructed by SfM from a set of images

The offline approach is known as Structure from Motion (SfM) [17]–[25]. It allows a complete sequence of images to be analyzed in order to perform a 3D map reconstruction and a camera trajectory estimation. Firstly, feature extraction (e.g. SIFT [26], SURF [27], ORB [28], AKAZE [29]) is performed on images. Then, camera pose is estimated by using features matching amongst pairs of images in the image connectivity graph. Next, 3D points in the map are computed based on estimated camera pose and corresponding points in pairs of images by using triangulation algorithm [30]. As illustrated in Figure 2-5, from each pair of matching keypoints (p_1, p_2) of two RGB images for which the pose (T_1, T_2) is known, a 3D point P is defined by:

$$\begin{cases} p_1 \times (KT_1^{-1}P) = 0 \\ p_2 \times (KT_2^{-1}P) = 0 \end{cases}$$

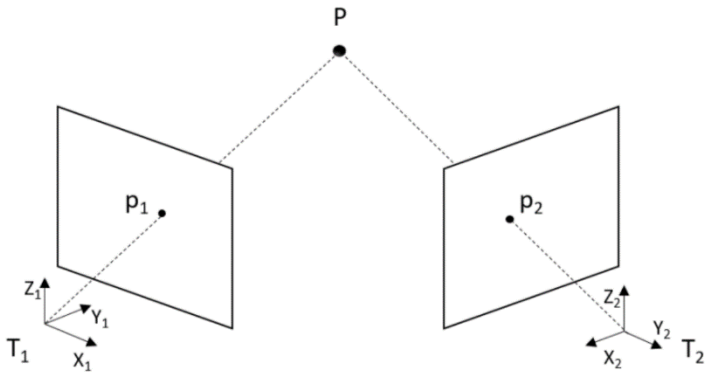


Figure 2-5. 3D point triangulation from a pair of matching points in two images with their estimated camera poses.

And each 3D point is represented by a 2D descriptor. It can be the mean of descriptors obtained from different observations. Finally, an optimization called bundle adjustment [31] is performed to minimize the reprojection error of points.

SfM includes two approaches namely incremental SfM and global SfM. Incremental SfM methods [17], [32] (Bundler), [19] (VisualSfM), [33] (COLMAP) are the standard approach that adds one image at a time to grow the reconstruction. While this method is robust, it is not scalable because it requires to repeat expensive operations such as bundle adjustment. On the contrary, global SfM methods [20]–[22] consider the entire view graph altogether, instead of incrementally adding more and more images to the reconstruction. For this reason, global SfM methods prove themselves to be much faster, with the same or even better accuracy than the incremental SfM approach. It is also much more readily parallelized. The major limitation of global SfM relies on the fact that it only works on an already collected sequence of images.

The most popular online approach is known as indirect SLAM [34]–[36] (Simultaneous Localization And Mapping). The majority of visual SLAM systems are based on camera motion tracking from frame-to-frame through consecutive frames. A tracking system passes through three steps: initialization, prediction, correction. First a few initial landmarks have to be given by using a known object [34] (as a A4 sheet of paper or fiducial marker) or by stereo algorithm [35], [36]. Then the camera pose is predicted and corrected by using Kalman Filter – KF [34] or a particle filter [37], [38]. Next, the 3D point cloud is updated based on a triangulation algorithm knowing the pose of the cameras.

Finally, SLAM methods use local bundle adjustment on selected keyframes [35] or fast global optimization to refine and avoid map duplication (e.g. pose graph) by loop closure as in [36]. However, in large scenes without loop closure, frame-to-frame tracking accumulates error causing drifts. This leads to a 3D point cloud that is not built correctly.

- **Direct 2D-3D point correspondences matching**

Direct 2D-3D point correspondences matching consists in finding 3D points in the world coordinate system corresponding to each 2D feature of a set of keypoints detected in the image coordinate system. This is performed by searching for the nearest neighbors of the keypoint descriptors in the space containing all the descriptors of the 3D point cloud. When the 3D map is very large, for example, covering a wide geographical area of a city, there may have too many 3D points, which raises two major challenges to this approach:

- How to quickly search within a massive database of a very large scene containing millions of 3D points
- How to accurately find correct 2D-3D matches without suffering from ambiguity.

Classical direct matching approaches use the approximate nearest neighbor search. The most widely used algorithm for the nearest neighbor search is the kd-tree [39] which works well for exacting nearest neighbor in low dimensional data. [40], [41] modify the original kd-tree algorithm to use it for an approximate strategy with high dimension. [42] proposes the use of multiple randomized kd-trees as a means to speed up the approximate nearest neighbor search. On the other hand, [43], [44] use k-means algorithm to compute k-nearest neighbors. [45] proposes FLANN (Fast Library for Approximate Nearest Neighbors) library that contains a collection of algorithms optimized for fast nearest neighbor search in large datasets and for high dimensional features. However, these searching methods become prohibitively expensive in very large and dense feature collections.

Between methods based on prior information, [46] presents a system that recovers the pose of a camera by confronting an image to a subset of 3D points that should be visible in the query. The subset of 3D points is retrieved by using a rough camera pose provided from an external sensor like GPS.

Without a prior camera pose, the 2D-3D matching becomes much more difficult because of having millions of 3D points. Given a set of 2D features $\mathcal{F}$ in a query image and a set of features $\mathcal{P}$ representing the 3D points in the 3D model, two basic matching strategies are considered:

- Feature-to-point matching - F2P, where one takes each feature in the query image, and finds the best matching point in the 3D model.
- Point-to-feature matching - P2F, where one conversely matches points in the 3D model to 2D features in the query image.

Using tree-based approximate search [45], the computational complexity for matching all the features against the 3D points is $\mathcal{O}(|\mathcal{F}|\log|\mathcal{P}|)$. And the one for matching all points against the query image requires time $\mathcal{O}(|\mathcal{P}|\log|\mathcal{F}|)$. For large-scale scenes, there are orders of magnitude more points than query features. Thus, P2F matching will only be more efficient than F2P search if only a small fraction of all points is considered. To accelerate 2D-3D matching for both F2P and P2F strategies, methods attempt to reduce feature search space by using the prioritized feature search algorithm [47]–[49]. [47] proposes a prioritized P2F matching strategy based on co-visibility information. Starting with a set of seed points selected from all parts of the model, they match points against the query image in order to descend priorities. Once a new match is found for a point, P2F increases the priorities of all other points that are visible together with this point in at least one database image. The search is stopped when a fixed number of points has been tried. [50] first proposed the F2P method for camera localization based on the SfM scene representation. To overcome limited viewpoints in the database images, they artificially synthesized novel view image to augment the database. [48] introduces a Vocabulary-based Prioritized Search (VPS) inspired by Bag-of-Words (BoW) matching method. The number of features stored in a visual word therefore gives a good estimate of the matching cost for this particular query feature. To speed up feature matching, they process the features in ascending order of their matching costs, starting with features whose activated visual words contain only few features. The search stops once large enough correspondences have been found. [51] shows that the class of methods introduced in [47], [50] can deal with large environment. They augment the P2F matching with hypothesis of co-occurrence of 3D points present in a close neighborhood. Based on similar spatial observation, [52], [53] consider visibility graph to reject wrong matches. [54] proposed to use binary features

to speed up the search. [55] uses the feature redundancy associated to 3D points to train random ferns on the top of each points. F2P matching time requirement is by the fact greatly reduced. [56] considers F2P matching as a combinatorial optimization problem and design a fast outliers rejection scheme. This promising work have been improved through contribution of [57]. Lowe's ratio test [58] is classically used to reject ambiguous matches.

[49], [59] propose an **Active Search** mechanism based on both F2P and P2F search. This allows them to exploit the distinct advantages of both strategies, while avoiding their weaknesses. This method first considers features more likely to yield F2P matches and to terminate the correspondence search as soon as enough matches have been found. Matches initially lost due to the quantization are efficiently recovered by integrating P2F search with a lower computational complexity.

2.1.3 Machine learning-based approaches

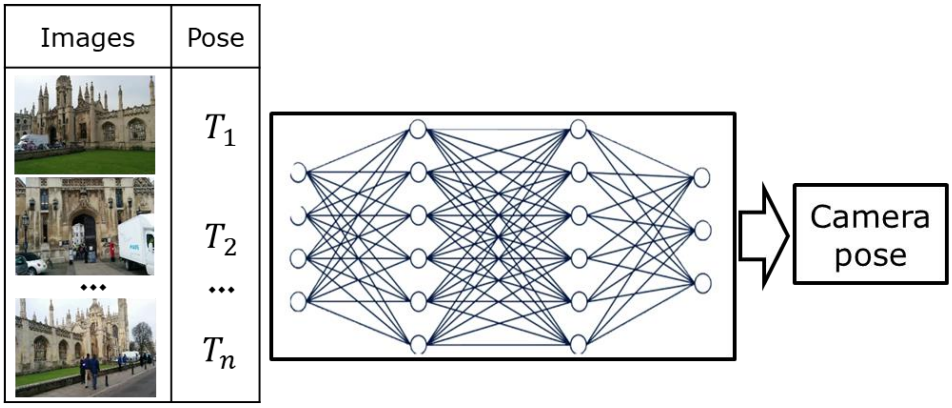


Figure 2-6. End-to-end camera relocalization based on machine learning approaches.

The problematic of camera relocalization can be solved by an end-to-end learning approach. In machine learning, camera relocalization is usually considered as a supervised regression problem. Methods based on this approach follow the same pipeline as described in Figure 2-2-b): Features are extracted by classical feature descriptors or learned features using the known network, e.g. AlexNet [60], GoogLeNet [61], VGG [62], ResNet [63], pretrained on ImageNet dataset [64]. Then, the feature is used to regress the camera pose in the scene. A regression model is learned from the data of the known scene represented as a set of labeled images: the images of the scene are captured from different viewpoints and labelled with theirs camera poses (6-DoF), as illustrated in Figure 2-6. Machine learning based methods are only capable of performing on known scenes and each trained model is uniquely used for the corresponding scene.

❖ **Camera pose regression**

For machine learning approaches, handling camera relocalization is as a regression problem solved by a supervised learning. It is performed based on the information known in advance of each scene. The training phase uses labeled images (images and their corresponding camera poses). The testing phase uses the trained model to relocalize the camera from each unseen image independently.

[65] first propose to address camera pose estimation with the end-to-end deep learning strategy. A CNN is learned from whole images labeled with the camera poses. Then, the trained model is used to directly predict camera pose from every RGB images. [65] presents the way to adapt the GoogLeNet model [61] from

classification to regression by properly modifying their final layers to regress camera pose: adding a fully connected (FC) layer before the two affine regressors. They leverage transfer learning from a pre-trained model of scene recognition by fine-tuning to regress camera pose. The features from the final convolutional layer are a high-level representation of the whole image, providing robustness to various lighting conditions, weather and other dynamic changes in the mainly unchanged scene. Training phase is performed with an objective loss function which is the sum of the translation error and the rotation error:

$$\mathcal{L}(I) = \|t - \hat{t}\|_2 + \beta \left\| q - \frac{\hat{q}}{\|\hat{q}\|} \right\|_2$$

Where (t, q) and $(\hat{t}, \hat{q})$ are ground truth and estimated translation-orientation pairs respectively (orientation being represented by a quaternion). β is a scale factor used to keep both error values to be approximately equal. However, the value of the scale factor is specific to each scene, which makes it remarkably hard to determine for a new scene.

[66] generates a probabilistic pose estimation by using dropout after every convolutional layer as a means of sampling the model weights of PoseNet. The dropout layers in PoseNet do not only play an important role to prevent over-fitting, but also provide an alternative interpretation for CNNs with dropout as a Bayesian model approximation. Instead of adding dropout layers, [67] uses Stochastic Variational Inference (SVI) [68] and Gaussian Process Regression (GPR) [69] as another way to provide the probability distribution for the 6DoF camera pose with one-time inference.

[70] is an improvement of PoseNet's architecture with spatial Long Short-Term Memory (LSTM) added after CNN layers. These features from convolutional layers are considered as an input sequence to a block of four LSTM units operating along four directions (up, down, left, and right) independently. On top of that, there is a regression part which encompasses fully connected layers for predicting the camera pose. [71] also applies LSTMs to predict camera translation only, but using short videos as an input aims at exploiting the temporal information to enhance camera pose estimation. Their method is a bidirectional recurrent neural network (RNN), which captures dependencies between adjacent frames refining accuracy of the global pose. Both of the two architectures lead to an improvement in the accuracy of 6-DoF camera pose, outperforming PoseNet. [72] trains an hourglass network, using skip connections between their encoder and decoder, to directly regress the camera pose. [73] proposes a different method based on a regression forest with hough voting approach to directly regresses the camera pose but it uses the same objective function combining translation error and rotational quaternion error. [74], [75] extend the set of training images with synthetic data.

[76] solves the ambiguity of the scale factor between location error and orientation error in the loss function of [65] by a novel loss function based on the re-projection error.

$$\mathcal{L}_g(I) = \sum_{X \in P'} \left\| KT^{-1}X - K\hat{T}^{-1}X \right\|_2$$

Where P' is a subset of all 3D points in the scene is are visible in the image I. K is the intrinsic calibration matrix of the camera. T and $\hat{T}$ are ground truth and estimated camera pose respectively. However, this loss function requires more time to compute and converges more difficultly.

Rather than using a single image, [77]–[80] propose visual odometry methods based on localizing sequences of images. [77] trains a multi-task network to predict both 6D global pose and the relative 6D

poses between consecutive frames, and report improvements over earlier neural network-based approaches, although their best results rely on using the estimated pose from the previous frame. In very recent work, [78] have added semantics to this approach. [79], [80] are also able to estimate camera pose of a monocular camera and the depth of its view while preserving the scale thanks to the training phase using stereo image pairs.

2.1.4 *Image retrieval approaches*

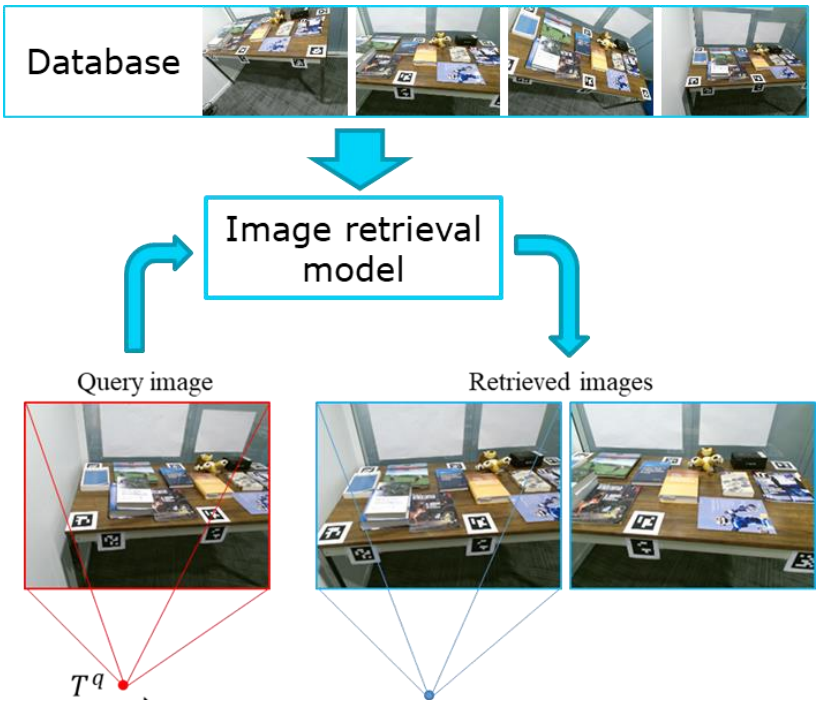


Figure 2-7. Camera relocalization based on the image retrieval approach. The query image passes through the image retrieval model to find nearest images in the database.

The two above approaches directly compute camera pose of a query image based on geometric information correspondences or regression learning. On the contrary, image retrieval approach consists of indirect methods that cast the camera relocalization as an image retrieval problem and provides nearest images and a coarse pose about the query image, as shown in Figure 2-7. The final camera pose is obtained either through estimating the relative pose between the query image and retrieved images, or the absolute pose using a geometric approach. In this section, we introduce the pipeline of the image retrieval approach, as shown in Figure 2-2-c), according to two steps: nearest images retrieval and camera pose estimation.

- **Nearest images retrieval**

The aim of the nearest images retrieval methods is to retrieve a set of images that are in the database and that are similar to an input query image. If the database stores not only a set of images but also the poses of the cameras that have captured them, the pose of the retrieved image provides an information on the possible location of the query image. This image retrieval problem includes two steps: extracting image-level feature for both the query image and the database; searching nearest images in the database based on a feature similarity.

Image-level feature

Extracting image-level feature aims at producing a compact feature representing each image. This can be performed by global feature extraction and local feature aggregation. For global feature extraction, the raw image can serve as a feature, with systematic resizing [81], [82]. Some methods produce this feature on the whole image by using GIST descriptors [83] or several random pixels [84]. With the recent emergence of deep learning, a new class of very efficient global feature have been created [85]–[90] based on deep learning models. Deep learning based methods for camera pose regression need to be trained for a specific scene. In contrast, they have been used for the image retrieval task without special training, exploiting inherent domain transfer capability of neural network. It is also interesting to notice that the most discriminative features are extracted from mid-level convolutional layers instead of fully connected layers. Many approaches leverage classification networks [91], [92], and fine tune them with place recognition datasets.

For local feature aggregation methods, from each single image, a large number of local features are extracted. The feature aggregation is then performed in order to make an efficient image-level feature while reducing the dimensionality of the feature vectors. The aggregation process emphasizes specific features that are more relevant for the localization task. Early feature aggregation techniques for image retrieval rely on BoW representations [44], [93]. It involves counting the number of features associated with each cluster in a large vocabulary and creating a histogram for each set of features from each image. Thus it represents an image in a compact vector. A great example of BoW is FAB-MAP [94] which relies on SURF feature [27]. [95] uses for the first time BoW obtained from BRIEF features [96] along with the very efficient FAST feature detector [97]. It reduces in more than one order of magnitude the time needed for feature extraction, compared to SURF feature. Nevertheless the use of BRIEF is neither rotation nor scale invariant. DBoW2 [98] extends that work using ORB feature [28] which is invariant to rotation and scale. Fisher vector (FV) based methods [99], [100] improve BoW based methods by using Gaussian Mixture Models (GMMs) to generate a probabilistic visual vocabulary. Another advantage of FV is that it can be computed from much smaller vocabularies, and therefore leads to a lower computational cost. Inspired by Fisher Vectors formulation, [101], [102] introduce Vector of Locally Aggregated Descriptors (VLAD) representation for image-based retrieval. VLAD is an extension of BoW. The difference between feature and its closest visual word is assigned to the final feature, instead of the visual word itself. In simpler terms, it first matches a feature to its closest cluster. Then, each cluster stores the sum of the differences of the features assigned to the cluster and the centroid of the cluster. The underlying idea behind VLAD representation have inspired various methods [103], [104]. Another aggregation solution using sum pooling of deep feature maps is presented in [105].

Compared to local feature aggregation, global features are considered less robust in viewpoint changes, occlusion and local variations in the image. Global feature extraction methods using deep learning has been less successful so far in local-level image retrieval. On most retrieval benchmarks, deep methods perform worse than conventional methods that rely on local feature aggregation. However, global features are computationally less intensive to extract and capture a comprehensive feature for each image.

Nearest images search

After extracting image-level feature, each image (query images as well as all the images of the database) is represented by a feature vector. Now, nearest images search can be processed in the same way as the feature matching described in Section 2.1.2. Due to the high dimension of image-level feature, comparison between features (with L2 norm as usually used metric) requires more time than local feature matching.

Thus, when the number of images in the database is very large, the local searching approach cannot be immediately considered. Some works suggest to compress the features to improve the storage requirements and retrieval efficiency. The most common approach is to use unsupervised compression through Principal Component Analysis (PCA) or product quantization [100], [106]. Supervised dimensionality reduction approaches have also been proposed in [107].

There are some other approaches aiming at accelerating nearest images search. [108] considers directly the localization problem as a classification task. The database is classified into discrete camera pose classes. The image feature of the query image passes through the classification model to obtain a set of retrieved images in the same class. On the other hand, by leveraging the aggregation process, DBoW2 [95] which is used in ORB-SLAM [36] builds incrementally a database that contains an invert index. It stores for each visual word in the vocabulary, the keyframes in which it has been seen, so that querying the database can be done very efficiently.

- **Camera pose estimation**

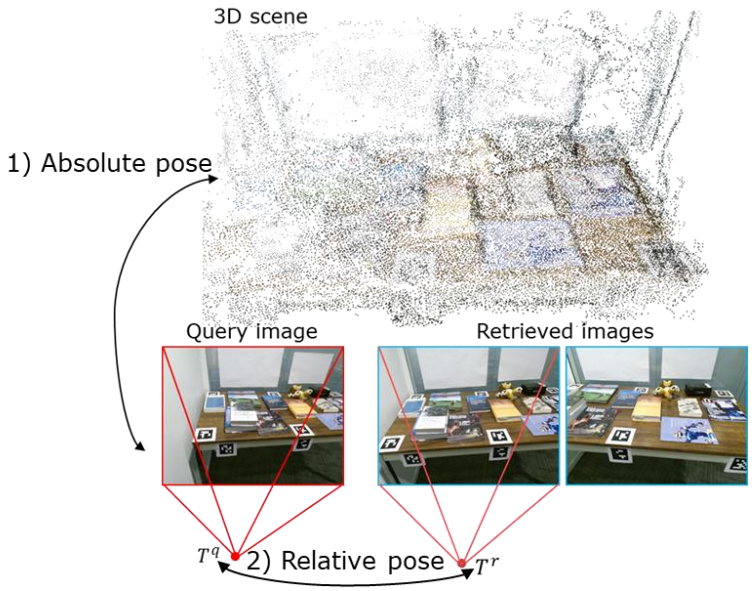


Figure 2-8. Image retrieval approach. Camera relocalization is handled based on nearest image retrieval. The camera pose of the query image can be estimated by two ways: 1) Calculating absolute pose using the geometric approach. 2) Through defining relative pose between the query image and retrieved images.

Figure 2-8 shows two ways to estimate the camera pose of the query image based on information of retrieved images. First way, based on the geometric approach, [36], [52], [109] directly estimate the camera pose by using a part of 3D model of a scene being visible in retrieved images. [82], [84], [89] present the second way is to define camera pose through relative pose between the query image and retrieved images. From the known camera pose of one retrieved image T^r and the transformation matrix from the query image to the retrieved imageT_r^q, the camera pose of the query image T_q is found by:

$$T^q = T_r^q T^r$$

Absolute camera pose estimation

Assume that a 3D point cloud has been built from images in the database. [52] achieves a 3D scene by performing SfM algorithm. On the other hand, 3D scene of [36] is obtained in real-time by mapping frames. Similar to the geometric approach, the 2D-3D point correspondences are required for the absolute camera pose estimation of the query image in the world coordinate system.

A set of sparse features extracted from the query image are matched with keypoints of retrieved images. Note that only keypoints of retrieved images that are associated with 3D points are nominated for matching. From 2D-2D matches, a set of 2D-3D point correspondences is established. The homography transformation matrix can be calculated by 2D-2D matches to filter out wrong correspondences. The precise camera pose is then computed from 2D-3D correspondences based PnP and RANSAC algorithms. [36] attempts to reduce the computational complexity of feature matching by using DBoW2 [95]. DBoW2 reports an additional benefit of the bags of words representation for feature matching. To compute the correspondences between two sets of ORB features, it can constraint the brute force matching only to those features that belong to the same node in the vocabulary tree at a certain level, speeding up the search. [36] also refines the correspondences with an orientation consistency test [98] that discards outliers ensuring a coherent rotation for all correspondences.

Relative camera pose estimation

The correlation information between the query image and retrieved images allows to estimate relative camera pose estimation. [82] presents a first solution in order to improve the camera relocalization in PTAM [35]. They exploit the fact that the SLAM system stores full RGB keyframes, and the process relocalization directly from these. Instead of extracting some forms of interest points and features from keyframes and then matching a novel view against them, they find that keyframes are sufficiently densely distributed so that the full image can be used as a descriptor: for each keyframe added to the map, generating a sub-sampled image, apply a Gaussian blur, and finally subtract the mean image intensity. This zero-mean heavily blurs image forms the keyframe's descriptor. When tracking is lost, each incoming video frame is similarly subsampled, blurred, and mean-normalized. Next, nearest keyframes are found. The camera pose is then set to the position of the nearest keyframe with the lowest image difference. The rotation of the camera is estimated by aligning the requested image with the nearest keyframe by minimizing the square difference of the sum over the whole image. They minimize over the three-parameter group $SE(2)$ in image (pixel) space, allowing ten iterations for convergence. Finally, the resulting 3-DoF image-space transformation is converted to a best-fit 3D camera rotation by considering the motion of a few virtual sample points placed in the image, in a procedure similar to the unscented transform.

[84] provides a camera relocalization solution for KinectFusion system [8] dedicated to only depth sensor. While [82] uses an appearance based method, [84] uses 3D point cloud obtained from depth images to determine the relative transformation between the query image and the nearest keyframes. This transformation can be for instance computed by employing a robust version of the Iterative Closest Point (ICP) algorithm [110]. [88] proposes a Siamese network to generate global features using a continuous metric learning loss based on camera frustum overlap. Given a query image and its nearest retrieved neighbor, their differential pose is defined based on this Siamese network.

2.1.5 *Hybrid approaches*

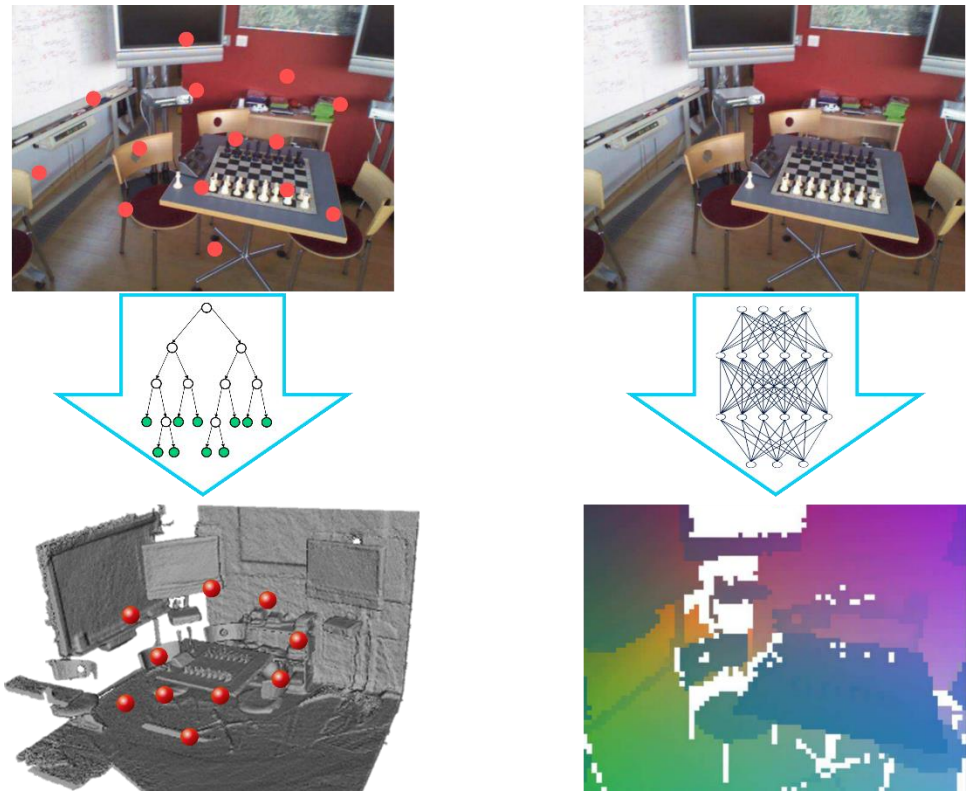


Figure 2-9. Hybrid approach. a) Random forest based methods. b) Deep learning based methods.

Figure 2-2-c shows the pipeline of the hybrid methods. Camera pose is estimated by combining both machine learning approaches and geometric approaches. Machine learning approaches are applied to learn and predict the 3D position of each pixel in world coordinate system. From these correspondences, geometric methods infer camera pose. Instead of directly matching 2D keypoint in a query image to 3D point cloud by feature-based methods, a world coordinate regression model defines rapidly and efficiently 2D-3D point correspondences. We present world coordinate regression models according to two approaches as shown in Figure 2-9: random forest based methods and deep learning based methods.

❖ **Random forest based methods**

The methods are known as voting methods, which have been successfully used in [111] for object detection. Though, in order to obtain a final 6-DoF camera pose (camera translation and rotation), each pixel does not vote directly for a global quantized 6-DoF because of the dimensionality of the camera pose estimation manifold space. Each pixel instead makes a 3D continuous prediction about its own 3D position in the world coordinates system or the camera coordinates system.

The first hybrid method [112] using a random forest proposes predicts directly corresponding 3D points in world space for all pixels in an RGBD image (each pixel in the image effectively denoting a 3D point in camera space). By generating predictions for thousands of pixels, their approach avoids the explicit detection, description and matching of keypoints typically required by traditional 2D-3D correspondences based methods. This makes it simpler and faster to find larger number of correspondences. Moreover, all features

used in [112] are based on simple pixel comparisons [113] and so are extremely fast to evaluate. At testing time, a random subset of pixels from RGB-D image pass through the regression forest to predict 3D world coordinates. The camera pose is then obtained by using Kabsch [4] and RANSAC [5] algorithms to optimize a geometric energy function:

$$T^* = arg\min_T \sum_i \rho(\|\hat{p}_i - Tx_i\|_2)$$

Where x_i is 3D coordinates in the camera coordinate system of a 3D point projected at pixel i. $\hat{p}_i$ is 3D world coordinates prediction. ρ is a top-hat error function.

This initial method have been improved in [114], relying on multiple regression forests to generate a number of camera pose hypotheses. The hypotheses are then clustered, and the mean pose of the cluster minimizing the reconstruction error is selected as the result. [115] introduces mixtures of anisotropic 3D Gaussians to represent the uncertainty associated with the regression forest prediction at leaf nodes and significantly improve the 6-DoF estimation by embedding this information within the full camera pose regression step. However, these methods above are limited by the use of RGB-D images in both training and testing phase.

As an extension of [112], [116] uses an auto-context regression forest from only RGB image patches with lower accuracy. [117] performs RGB relocalization by estimating an initial camera pose using a regression forest, then queries a nearest neighbor keyframe image and refines the initial pose by sparse feature matching between the camera input image and the nearest keyframe. [118] maps parameters between regression forests and neural networks to leverage the performance benefits of neural networks for dense regression while retaining the efficiency of random forests for evaluation. [119] stores a priority queue of non-visited branches whilst passing a feature vector down the forest during testing, and then backtracks to see whether some of those branches might have been better than the one chosen. [120] makes use of both points and line segments (segment feature being based on the features of a points sampling) to achieve more robust relocalization in poorly textured areas and/or in case of motion blur. [121] proposes a new method based on pre-trained regression forest. This method permits to transfer the pre-trained model to a new scene through online adaptation. [122] proposes a sparse feature regression forest learning with a novel split function aiming at accelerating computational time and keeping high accuracy. [123] introduces an adaptive regression forest that is able to update only a subset of uncertain leaves to quickly modify without any interruption the trained model on dynamic scenes such as moving objects.

- **Deep learning based methods**

On the other hand, various related methods have used deep learning approaches to define 2D-3D correspondences. DSAC [124] is the first method using a VGG style architecture for scene coordinate regression. It takes an image patch of 42×42 pixels as input and produces one scene coordinate prediction for the center pixel. This design is not so efficient because the CNN processes neighboring patches independently without reusing computations. They sample patches per image instead of making a dense prediction for all possible patches. DSAC also shows how to replace the RANSAC stage of the conventional pipeline with a probabilistic approach to hypothesis selection that can be differentiated, allowing end-to-end training of the full system. [125] presents a light convolutional neural network regression for scene

coordinate prediction from local patches extracted based on key-point detection, that improves the accuracy and runtime for regressing 3D world coordinates.

[126], [127] use a fully-convolutional encoder decoder network to predict scene coordinates for the whole image at once, thus taking into account the global context. This prevents patch sampling in [124], but needs significant data augmentation to avoid overfitting. [128] that is known as DSAC++ significantly improves the results of DSAC. They use a Fully Convolutional Network (FCN) [129], but without upsampling layers. Their FCN takes a RGB image of 640×480 pixels as input and produces 80×60 scene coordinate predictions. They regress more scene coordinates in less time than their previous work, DSAC. Whilst, they also show how to avoid the need for a 3D model at training time (albeit at a cost in performance). Very recently, [130] has shown how to use an angle-based reprojection loss to remove DSAC++'s need to initialize the scene coordinates with a heuristic when training without a model. However, despite all of these advances, none of these papers remove the need to train on the scene of interest in advance in order to generalize the learn model.

2.1.6 Data representation

In above sections, we present different mechanisms to model scene knowledge. And now, we give data representation requirements corresponding to scene models of different approaches. That allows to implement camera relocalization methods.

Geometric approaches

A geometric model for camera relocalization includes data as follows:

- **Point cloud**. Each point in the point cloud is represented by 3D world coordinates and a descriptor. This descriptor is calculated by combining its descriptors (SIFT, SURF, ORB, AKAZE …) obtained from the reconstruction.
- **Point cloud organization**. The organization of the point cloud is necessary to accelerate 2D-3D matching process. Some solutions used in the state-of-the-art methods:
    - **Kd-tree** [58] based on FLANN library [45]. The point cloud is split by multiple randomized kd-trees.
    - **Hashing table** [112]. It can be applied for binary descriptor (ORB, BRIEF, AKAZE). N-bit descriptor substrings is used as key index in the hashing table.
    - **Visual vocabulary** [48], [49]. Each visual word contains a subset of points in the database. The visual words can be organized by a k-means tree in order to speed up matching.

Machine learning approaches

End-to-end camera relocalization is based on machine learning algorithms: deep learning and random forest. The learned model is trained based on training image annotated with camera poses. Each algorithm requires different data representation.

Deep learning:

- **Network architecture**. It defines input, output, layers in the network
- **Network weights**. They are a set of values of convolutional filters, fully connected neural.

Random forest:

- **Tree architecture**. A forest consists of N trees. Each tree is built by internal nodes and leaf nodes.
- **Tree parameters**. They include split function parameters at internal nodes and predictive models at leaf nodes.

Image retrieval approaches

Image retrieval approaches include two main elements. The first one is the image retrieval model that allows calculating global feature representation for each keyframe. The second one is the association model that allows estimating camera pose of the query image through absolute pose or relative pose:

- Some popular **image retrieval model**:
  - **Visual vocabulary**, for example, FBoW in ORB-SLAM [36].
  - **Randomized ferns,** presented in KinectFusion [84].
  - **Learned model,** used in Inloc method [90]
- Association model:
  - To define the absolute camera pose requires associations between keypoints of keyframes to the point cloud (ORB-SLAM).
  - Or define relative camera pose:
    - A keyframe is sub-sampled as in PTAM [35].
    - A cloud point from depth image for each keyframe in order to determine the relative transformation between the query image and the nearest keyframes based on ICP algorithm (for example, KinectFusion [84]).

Hybrid approaches

Similar to machine learning approaches, these hybrid require to store a learned model that is applied to regress 3D world coordinate from each 2D feature.

- **Deep learning** model [124], [125].
- **Regression forest** model [112], [122].

2.1.7 Conclusion

In this chapter, we presented the state-of-the-art camera relocalization methods according to four approaches: the geometric approach, machine learning approach, image retrieval approach, hybrid approach.

- The geometric approach is an essential solution based on 2D-3D point correspondences. The geometric approach is simple, accurate, and especially useful when the query image has a significant distance to the training images. However, such methods are restricted to a relatively small scene due to the fact that matching cost, depending on the matching scheme employed, can grow exponentially with respect to the number of keypoints. Matching of local features can be noisy and unreliable on scenes with repeated patterns. Besides, to achieve an accurate and effective 3D model, the SfM algorithms take much more time to build and optimize the 3D model.
- The image retrieval approach improves the matching time compared to the geometric approach by finding a coarse pose from the nearest retrieved images. Precise camera pose can be obtained by defining 2D-3D point correspondences between the query image and 3D points seen by the retrieved images or by measuring the similarity of images for relative pose estimation. However, these

methods are often not accurate if the query frame is captured from a viewing pose that is far from those in the database. For the localization system, this approach needs to store a broad set of keyframes. Consequently, memory usage as well as processing time increase concerning the size of models.

- The machine learning approach provides a compact end-to-end camera pose estimation solution. Contrary to the geometric approach, it struggles to generalize beyond their training data. These methods can estimate camera pose in real-time from each image on a GPU, but the training phase takes hours or even days for a small scene. However, significant limitations of these methods lie in their moderate accuracy and the lack of confidence score for each pose estimation. These methods are significantly less accurate than geometric based methods as well as image retrieval based methods. This approach seems more consistent with image retrieval than with camera pose regression.
- The hybrid approach is based on both a deep learning approach and a geometric approach with aiming at benefiting from each. The machine learning part defines the point correspondences faster than geometric based methods. Then geometric part infers camera pose from these correspondences. These methods achieve higher accuracy than basic approaches. However, they are limited scalability on large scale scenes.

Each camera relocalization approach has benefits and limitations, and it constructs a specific map data representation. The map is able to reuse to perform (re)localization in already modeled area. However, building a map that can be updated, expanded or that can work whatever the conditions is not a trivial task. A generic map pivot format for all approaches is impossible, but a format including all representations is preferred. It includes:

- Point cloud with descriptors.
- Point cloud organization: Kd-tree, hashing table, visual vocabulary, etc.
- Dense point cloud.
- Keyframes: image, global feature, keypoints and associations to the point cloud.
- End-to-end camera regression model.
- 2D-3D regression model.

2.2 3D dense reconstruction

Our goal is to maintain a unique real-time updated 3D map of the real environment. Accurate 3D dense map, together with semantic scene understanding of physical products and processes is a critical part of Digital Twin. Concerning our utilization, the Photogrammetric Computer Vision has three main directions of 3D dense map construction: 1) using the standard image-based approaches, i.e., the Multi-view Stereo (MVS), 2) using the fusion of the depth maps streamed from a depth sensor, e.g., Kinect [131] or Intel RealSense [132], 3) employing the 3D laser scan, e.g., Leica BLK360 [133] or FARO FOCUS^S 350 [134]. Note that the MVS utilizes a vast amount of approaches for point cloud calculation [135], as so as calculation of volumetric data [136] and mesh representation of the map [137]. The 3D map can be further extended about semantic scene understanding [138] to classify the moving objects [139] and include virtual CAD models.

Each of these directions requires different equipment leading to different accuracy of the final dense map as so as the acquisition and computational time. Expensive professional laser scans [133], [134] can obtain a high-quality initial map of indoor as so as outdoor environments. However, the acquisition is extremely time-

consuming, which restricts the usage to only a few scanning per long period. The depth fusion approaches as [8], [140] also require specialized hardware. Only a few modern AR devices as Microsoft Hololens [141] or Google Tango are equipped with a depth sensor as a time-of-flight (ToF) camera [142]. These sensors can provide tens of millions of 3D points each second in the form of depth maps [143], [144]. The fusion of these depth maps, usually performed by Iterative Closest Point (ICP) algorithms [8], [140], [145], is computationally challenging because of the large amount of data. In our case, AR devices provide camera tracking, i.e., the camera pose, which can be used to initialize the ICP-like method and speed up the convergence. Running the fusion on GPU units, one can manage the real-time update of the global 3D dense map. Unfortunately, the range and power of such sensors are usually limited and work reliably indoor only. The classical image-based MVS methods [135], [136] calculate the 3D map from a set of images captured from different viewpoints, i.e., these methods can work with any camera device. MVS experience significant progress in recent years using Deep Neural Networks (DNN) [135], [146], [147]. However, still, the image-based approach requires additional computational resources for finding and optimization of all pixel correspondences between an exponential number of subsets of captured images. Therefore, it is challenging to use them in run-time.

The dense representation of the 3D map provides several benefits in comparison with a 2D map. A progressive mesh representation with additional textures to encode geometric details in so-called normal and height maps allows rough visualization of large-scale areas, e.g., factory layout, as so as small detailed parts of individual objects. A dense point cloud using 3D features [148], [149] can describe the world much more invariantly to the viewpoint. For instance, the 2D descriptor of a keypoint, e.g., SIFT [26] or TILDE [150], corresponding to the 3D corner of a desk strongly depends on the viewpoint and perceived background while associated 3D descriptors do not. This benefit leads to more accurate pose estimation and localization [148], [149]. Assuming a static environment, the reconstruction can be iteratively optimized by gradient descendent methods to minimize a reprojection error. Unfortunately, the AR devices are usually used in dynamic environments, e.g., we observe moving robotic hands, with possible occlusions, e.g., moving employees in front of the camera device. To our knowledge, there is no commercial solution that models dynamicity. Only Google Tango has been providing detection of scene changes, but the solution is based on sensor fusion and cannot follow objects over time. The disadvantage of dense representation of the 3D map is that the visualization and further processing require significant hardware resources. Computer Graphics data has to be precomputed in a more compact representation as lightweight mesh smaller number of vertices, and additional textures to encode geometric details. The full resolution mesh cannot be visualized on current computers either on tablets or smartphones and are not ready to be used on Augmented Reality platforms.

2.2.1 Image based approaches

Image-based approaches estimate the most likely 3D shape that explains the input set of images, under the assumptions of known viewpoints (which can be obtained by SfM [33] or SLAM [36]), materials, and lighting. In practice, materials and lighting conditions are not known, i.e., the problem is ill-posed [151]. Generally, most of the algorithms assume the Lambertian textured surfaces, consistent lighting, and more than two images from different viewpoints [135], [146], [147].

Moreover, captured images are typically distorted by the physical properties of the camera lens. There is a vast number of mathematical models either parametrical [25] or non-parametrical [152], which allows us to remove radial distortion if the parameters of distortion were computed beforehand. This process is called

camera calibration. Note that the following algorithms assume undistorted images as input or incorporate the distortion estimation [153], [154]. Some distortions models, e.g., Rolling Shutter models [155], are not related to the physical properties of the camera and have to be estimated for each image separately. It makes the relations between images much more challenging [156] and, therefore, are these models rarely used for dense reconstruction.

There are several approaches, e.g., Shape from Shading [157], [158] and Deep learning depth estimation [159], [160], which focus on judging the depth from one image. Unfortunately, these approaches are not general enough to work well in a wide variety of real environments.

Having exactly two images which observe the one scene, i.e., two-view stereo [161], we can describe the relationships between images by the epipolar geometry [25] and seek the corresponding patches [100] in images using a one-dimensional search. Note that epipolar geometry (known camera poses) constrains each point in one image on a line in the second one. Further, the distance between corresponding patches on the corresponding epipolar lines directly realizes the depth measures, i.e., can be used to compose the depth map. There is a vast literature about so-called photo-consistency measures estimating how likely image domains (image patches) being correct correspondences [162]–[164]. The domain photo-consistency aggregation combines the spatial domain into one measure to increase its robustness. The most common handcrafted photo-consistency aggregations are Sum of Squared Differences (SSD), Sum of Absolute Differences (SAD), Normalized Cross Correlation (NCC), Census [162], and Mutual Information (MI) [163], see more details in [151], [164], [165]. The common photo-consistency aggregations as SAD or SSD are not invariant to illumination changes but often used because of high-speed computation on GPUs [166]–[168]. There are also techniques [169] focused on accuracy, which combine SAD with another measure, such as NCC or Census. The learned photo-consistency aggregation [170] was used to estimate the overall confidence of depth map in [171]. An extension, end to end Deep Neural Network (DNN) confidence estimation from the disparity map was presented in [172], [173], improved by exploiting local consistencies in [172] and also including the input images in [174]. The extension of Neural Networks (NN) working in 2D to MVS is a difficult problem, and the first solution was introduced very recently in [175].

Multi-View Stereo (MVS) methods [135], [175] extend stereo pair by using more than two images that increase the robustness and quality of the reconstructed surfaces. Under the assumption of a rigid scene, the multi-view photo-consistency measures the consistency of illumination, textures, and 3D geometry of the scene being captured from different viewpoints. MVS can be seen as a constrained optimization problem where the multi-view photo-consistency is maximized. With suitable preprocessing, the local optimization methods [176] can be used instead of global optimization methods, e.g., graph cuts [177]. The photo-consistency measure is regularly sensitive to noise [151], and therefore, several handcrafted filtering approaches were developed: 1) local averaging by spatially varying domains [178], 2) the weighted filter concerning the color similarity regarding the reference pixel color as guidance for the selection of weights of the averaged domain is in [179], 3) the anisotropic filters which can be efficiently implemented using a bilateral filter [180], [181], 4) the weighted median filter [182] or 5) directly implement the filter into the photo-consistency measure [169].

The photo-consistency measure describes a similarity of image domains. However, in the general case, we do not know which images see what because the 3D scene itself is unknown, i.e., some parts of the scene may be occluded. This problem is the so-called visibility estimation problem. Well-known methods are Space-

carving [183], [184], usually combined with a smart decomposition of large-scale MVS problem into a sequence of small sub-problems related to each image [143], [185], [186] or view clusters [185]. The reduction of the number of view clusters guaranteeing a "good" reconstruction is optimized in [187]. Another approach assuming an initial coarse reconstruction, i.e., fine-scale visibility estimation, is to select views related to a piece of the 3D map, and iterate visibility estimation and reconstruction [188]–[190]. We can calculate the course reconstruction by, for example, the volumetric fusion approach [191], [192].

The most famous representations of the dense reconstruction are the depth maps [143], [144], 3D point cloud [187], [193], [194], volume scalar field [192], [195], [196], and a mesh [197]–[199]. SOTA approaches focus on particular representation [135], [175], [196] or composes these steps into one pipeline [33] with the texture-mapped mesh as the output.

The depth map is a simple, flexible, and scalable representation that can be calculated for each input image using a few neighboring images for photo-consistency evaluation. The most straightforward, winner-takes-all (WTA) strategy using NCC as the photo-consistency measure and two view stereo was published in [143]. Robustified version [144] calculate WTA for n-images, which are aggregated into one measure using a Parzen window [200]. Simpler but also useful is the aggregation of all WTA's values above a certain threshold [201]. Recently was published [202], [203] local methods for aggregating matching cost using neighboring pixels and utilizing WTA strategy.

The global methods construct and minimize an energy function, i.e., allow smoothing depth for non-reliable pixels [204] while avoiding smoothing color edges. The Markov Random Field (MRF) depth map estimation [205]–[207], assumes labeling of each pixel by discretized depth value in a defined range with enforcing the spatial consistency. MRF can be seen as an NP-hard combinatorial optimization problem. An efficient approximation, so-called alpha-expansion [177], [208], [209], repeatedly solves the max-flow and min-cut problem to improve depth assignments. An extended version of MRF with different, more suitable, label sets for each pixel was published in [210].

Note that calculation of the depth is computationally demanding. The photo-consistency function has to be evaluated for each pixel and hypothesis of the depth over multiple images. The first algorithm running real-time on GPUs is the Plane Sweeping Stereo [168]. The Plane Sweeping Stereo algorithm extracts multiple directions of planes in the observed scene, project images onto multiple parallel planes with these directions using homography, and calculate the depth value at each pixel by WTA strategy. In the end, multiple depth maps are merged to produce the final result. Plane Sweeping Stereo approach is the core of many SOTA approaches [136], [195], [196]. The number of planes defines the depth resolution of the scene volume. Note that the volumetric approaches assume the 3D space divided into regular grids where the voxels are related to the surface. The first learning-based pipeline which utilizes the Plane Sweeping Stereo to precompute the depth voxels is SurfaceNet [147]. The SurfaceNet uses 3D CNN to regularize and classify the voxels which belong to the surface. The most significant disadvantage is that the voxel-based approach requires a large amount of memory to represent the depth. There are several methods [186], [211], [212], which generates dense 3D patches by expanding the confident keypoints. Further, octree space partitionings exploiting the sparsity of the scene were introduced in OctNet [213] and O-CNN [214]. The approaches [147], [215] and their extension DeepMVS [146] apply the divide and conquer strategy to the MVS. Further, the Recurrent NN (RNN) was utilized to decrease the memory requirements in R-MVSNet [196]. Note that the recent publication MVS by Nonparametric Temporal Fusion [216] infer the depth maps based on RNN from

the video stream of the images modeled as a Gaussian Process. And, the TAPA-MVS [217] propose a new global energy function to recover the depth even for textureless areas of the images.

To summarize, the best SOTA approaches for dense reconstruction form images are the following. The Cascade Cost Volume for High-Resolution MVS [136] is the so far the fastest and most accurate approach using the cost volume representation, according to SOTA benchmarks [218]. This approach utilizes the MVSNet [146] with coarse to a fine-scale adaptation of plane positions in the scene to estimate the accurately the depth. The PointMVSNet [135] is the so far the best DNN approach targeting point clouds as scene representation. The authors proposed the PointFlow module to estimate the 3D flow based on joint 2D-3D features of points hypothesis. The depth is calculated using the iterative refinement scheme in a coarse-to-fine manner. Finally, the DeepC-MVS [175] propose the fist confidence prediction network for generic MVS-derived depth maps, i.e., using the 2D correspondences only. DeepC-MVS is as so as PointMVSNet not dependent on the resolution of input images, which is the most crucial problem of voxel-based approaches.

2.2.2 Depth sensors

The depth sensor provides a stream of depth data, e.g., depth maps in resolution 1024x1024 with 15 FPS in the case of Hololens [141]. Each depth sensor has different accuracy. The most common depth cameras are 1) Stereo camera, 2) Structure light camera, and 3) Time-of-flight (ToF) camera. The stereo camera captures pairs of images with a known relative pose, e.g., [219]. This camera involves standard MVS-based methods to calculate the depth maps. Structure light camera calculates the depth by illuminating a scene with a specially designed light pattern. This pattern can be modeled as a second camera. The depth if further reconstructed by triangulation between projected beams and observed structures. The most popular, ToF cameras shot a light pulse and record 2D images with an increasing delay from the shutter opening. ToF estimates the 3D information from the time of the response in the 2D images. Unfortunately, the active sensors, i.e., the Structured light and ToF cameras, has limited power and range of the sensor and are not usable in outdoor environments due to lighting conditions.

One of the first publications about the fusion of RGBD images was KinectFusion [8]. The KinectFusion approach assumes a dense volumetric model using the alignment and camera pose estimation by ICP, i.e., the 3D volumetric scene alignment in the memory. Kintinuous [220] extends the KinectFusion about rolling cyclical buffer of volumetric data and loop closure, i.e., if the method recognizes already a visited place, the camera pose trajectory is optimized to remove accumulated errors. Endres et al. [221] published open-source RGBD SLAM, which combines camera pose tracking by feature matching and ICP and simultaneously runs the graph-pose optimization with loop-closure recognition. The DVO-SLAM [222] reformulated optimization function to minimize both the depth error and photometric error. The ElasticFusion [14] proposes applying a non-rigid deformation to the 3D map. The ORB-SLAM2 [109] focused on globally consistent localization and sparse optimization instead of accurate dense map fusion. Note that for accurate reconstruction is necessary calibration [145] of the depth sensor and accurate infra-red images unwrapping [140]. Better memory requirements can be achieved using octree [223] as so as in image-based MVS methods.

2.2.3 LiDAR scans

The laser scans can reconstruct dense point clouds with high-acuracy, i.e., with error in units of millimeters [133], omitting the calculations required by image-based MVS and volumetric fusion of depth maps. The alignment can be done simply by ICP. The disadvantage is long acquisition time and the requirement of the expensive hardware setup, e.g., Leika BLK360 [133] or FARO FOCUSS 350 [134].

2.2.4 Conclusion

We presented different approaches to achieve and maintain the 3D dense reconstruction in this chapter. LiDARs produce high-quality map, in the format of the 3D point cloud, but requires considerable acquisition time. For this reason, the LiDARs are suitable for the 3D map initialization. For the real-time update of the 3D point cloud, the AR devices with ToF cameras are together with accurate calibration [145], accurate IR images unwrapping [140], and camera poses tracking suitable. The state-of-the-art approaches DeepC-MVS [175], PointMVSNet [135] are convenient to calculate the update from keyframes and incrementally increase the accuracy of the 3D map. The moving object has to be segmented using DNN or geometrical consistency constraints and further updated based on CAD models. We propose to maintain the labels for 3D points related to CAD models to have a relationship between virtual and real objects with information about their dynamicity [139]. For the visualization, the progressive lightweight mesh is suitable instead of the dense point cloud. Dense features as corners [148], [149] associated with points in 3D allows more accurate localization. A generic map pivot format to for the dense reincludes:

- dense points cloud
- labels for known CAD models and dynamic objects
- progressive lightweight mesh

3D descriptors associated with dense points

3. Commercial format

3.1 3D dense reconstruction for Industry 4.0

Original scanning data format is. PLY or. E57. These are the required formats for “expert tools” such as NX. Formats used as inputs for the Siemens ProcessSimulate is .POD. The transformation from. PLY to .POD is done currently by Bentley POD creator. The factory personnel can provide sample 3D data for point clouds and also 3D files.

The Navvis scanner is generating a black point cloud by laser scanning which is not visible for human eyes as you cannot identify objects. Navvis is making 360-degree photos every 2 meters to colorize the point cloud by the photos in the photo processing stage. If the scan is done by RealSense or mobile phone cameras, the color integration takes place already in the generating process of the point cloud. Therefore, there are no separate 360-degree photos available.

For the fitting of the 3D models with the 3D scan, NX is used for layouting and Process Simulate for simulation of single robotic cells.

3.2 Camera relocalization

The popular AR commercial systems such as Apple ARKit, Google ARCore, and Microsoft Hololens, are capable of applying in Industry 4.0 by obtaining high accurate camera localization. They use Visual-Inertial Simultaneous Localization and Mapping (VISLAM) that combines RGB cameras or RGB-D cameras with inertial sensor data. Regarding camera relocalization problematic, visual sensors provide significant data in order to rapidly define camera pose when tracking loss or starting the system.

3.2.1 ARKit (Apple)

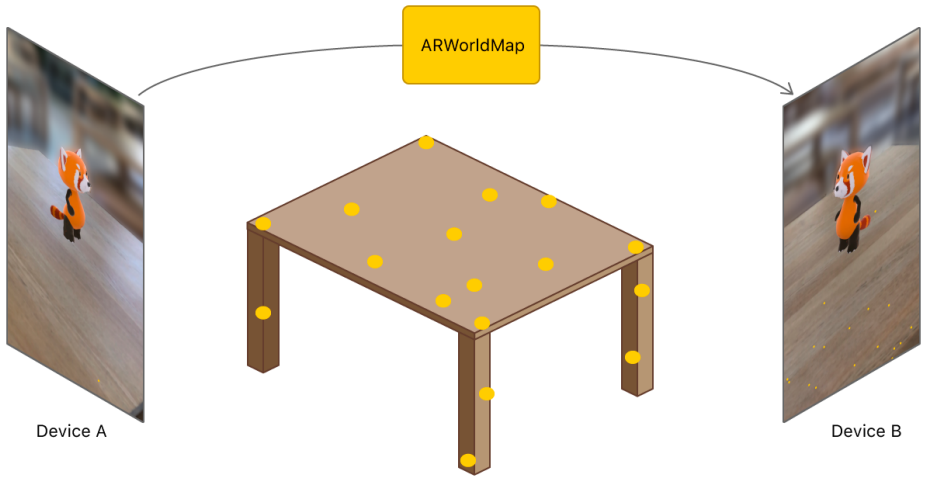


Figure 3-1. A simple shared AR experience for two devices using ARKit.

ARKit has a ARWorldMap class that saves the current world map including the set of anchors recorded, the center point and the size of the world map's space-mapping data (relative to the world coordinate origin of the session the map was recorded in), and the coarse representation corresponding to the point cloud of the space-mapping (a collection of points in the world coordinate including a list of detected 3D points and a list of unique identifiers corresponding to detected feature points). This world map can be saved, loaded, and

shared to another device to make a multiuser AR experience as illustrated in Figure 3-1. When ARKit loses track of the device's position or orientation, ARKit calls relocalization function sessionShouldAttemptRelocalization to try to resume the experience where the user left off instead of immediately restarting system from scratch.

```
Class ARWorldMap: NSObject {
      var anchors: [ARAnchor]
      var center: simd_float3
      var extent: simd_float3
      var rawFeaturePoints: ARPointCloud
}
```

```
Class ARPointCloud: NSObject {
      var points: [vector_float3]
      var identifiers: [UInt64]
}
```

3.2.2 ARCore

ARCore provides an access to the map through its API thanks to the following functions:

- public FloatBuffer getPoints (): Returns a buffer of point coordinates and confidence values. Each point is represented by four consecutive values in the buffer; first the X, Y, Z position coordinates, followed by a confidence value. This is the same format as described in DEPTH_POINT_CLOUD. Point locations are in the world coordinate space, consistent with the camera position for the frame that provided the point cloud.
- public long getTimestamp (): Returns the timestamp in nanoseconds when this point cloud was observed.
- public IntBuffer getIds (): Retrieves a buffer of point cloud point IDs. Each point has a unique identifier (within a session) that is persistent across frames. That is, if a point from point cloud 1 has the same id as the point from point cloud 2, then it represents the same point in space.

Thanks to the ARCore API, developers can access the 3D point cloud with a timestamp corresponding to the last observation as well as a confidence score per point, but does not provide the descriptor attached to each 3D point which is crucial to implement a dedicated relocalization service.

Also, ARCore provides an access to the anchors on which digital content is attached thanks to the following functions:

- public String getCloudAnchorId (): Gets the current cloud anchor ID for this anchor. Returns an empty string if getCloudAnchorState() returns TASK_IN_PROGRESS or NONE.

- public Anchor.CloudAnchorState getCloudAnchorState (): Gets the current cloud anchor state of this anchor (aming ERROR_CLOUD_ID_NOT_FOUND, NONE, SUCCESS, TASK_IN_PROGRESS, etc.).
- public Pose getPose (): Returns the pose of the anchor in the world coordinate space. This pose may change each time update() is called. This pose should only be used for rendering if getTrackingState() returns TRACKING.
- public TrackingState getTrackingState (): Returns the current state of the pose of this anchor. If this state is anything other than TRACKING the pose should not be considered useful.

Thanks to the ARCore API, developers can access to the pose of each anchor related to the world space.

3.2.3 Hololens

Hololens provides an access to a mesh that seems to be generated from the 3D point cloud used for reolocalization, but post-processing for meshing avoid to get back the raw data used for relocalization. Instead Hololens exposes higher level concepts like Anchors, Planes, Images, and so on.

3.2.4 Conclusion

Even if the solution provides access to the original point cloud with descriptors, e.g. ARKit, the code or binary to extract the feature descriptors from the current image is never provided, preventing the reuse of the point cloud to implement a relocation solution of its own. However, we can exploit the efficient tracking solutions offered commercial solution. Indeed, we can capture the images and poses obtained by ARKit, ARCore and Hololens (HoloLensForCV) via their APIs to reconstruct our own 3D map and use this 3D map to develop a dedicated camera relocatization solution.

3.3 Hololayer platform

Hololayer is a Siemens internal platform, currently under development, for persistent geolocated augmented reality. This platform, or parts of it, shall be further developed and used within the ARTwin project. Note that some aspects of Hololayer are currently patent pending or in the process of being submitted for patenting; please treat this information as confidential.

Hololayer consists of an iOS app and a cloud back-end, with a Hololens app in planning. It lets users create AR content in the field, but also allows import of geospatially referenced data e.g. from BIM or 3D asset management systems. It combines GPS, image recognition, OCR for label recognition, and SLAM. It can currently provide a location accuracy of AR augmentations to within approximately 20 cm in most "interesting" parts of industrial plants.

Below are some example images showing how Hololayer is currently being used in a small process facility.

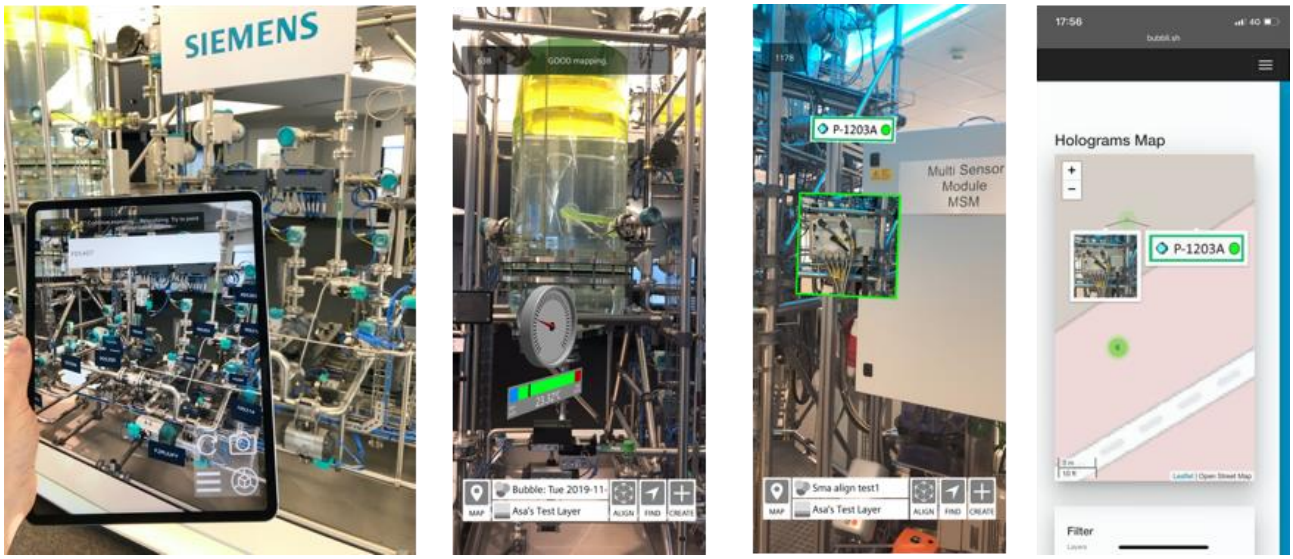


Figure 3-2: Views of Hololayer. From left to right: labels showing the position of devices in a process facility; live data visualization showing temperature values in AR; rendering of 2D images (either taken on site or from a back-end server) in AR; display of these annotations on a 2D map UI.

3.3.1 Overall Architecture - App, Cloud, Web interface

Hololayer consists of:

- An iOS app (Android and Hololens versions will be developed in the future)
- A cloud back-end for image recognition, data storage and access management
- A web interface for rights management and a map-based view of the holograms

3.3.2 Data Model - Bubbles, Layers, Holograms

The basic purpose of the Hololayer app is viewing and editing "holograms". Everything the user can see in augmented reality is a hologram, and has a defined position and orientation in the world. A hologram can be a photograph, a movie, an audio clip, a simple 3D model, a web page or a special graphical control, such as a remote desktop connection or a virtual temperature display showing live OPC data. Holograms can also have metadata, such as a severity (error/warning/info), a textual comment, or the editing history.

Hololayer structures the world logically into "layers", which can be shown and edited one at a time. Each hologram belongs to one layer. For example, there could be a layer "building maintenance", a layer "electrical installation" and a layer "tourist information". Layers can have access rights, so that the information in them is only readable or writable by certain users.

Hololayer divides the world spatially into "bubbles" of about 10 meters in diameter. This roughly corresponds to the size of a normal room. Each bubble has a GPS-referenced geographic location (latitude, longitude, altitude). A bubble may also have identifiers to help to distinguish it from other nearby bubbles, using image recognition - e.g. a room number or a relatively unique object in a room (e.g. a piano). Each bubble also has a friendly name and a preview image. Each hologram belongs to one bubble.

Users can create and edit bubbles and layers in the field, as well as the holograms themselves.

All information on holograms, bubbles and layers are persisted in a cloud backend.

3.3.3 *Localization and Spatial Persistance; Anchors*

To identify the bubble a user is in, the app uses GPS to determine the user's approximate position. Then, it queries the server for a list of all bubbles within a certain distance (200m) of that GPS position. The app also lets the user take a photograph of a "relatively unique" object in the vicinity (e.g. room number or piano), so it can filter the list of potential bubbles by identifiers. If necessary, the user then chooses the correct bubble from a remaining short list.

The app uses ARKit to create a SLAM map of the vicinity of the bubble. It stores this persistently in the backend, associated with the bubble. This allows the app to recognise the position and orientation of the mobile phone to within 20 cm of accuracy, anywhere on the planet.

To support situations where the environment changes frequently, or where SLAM maps do not work well (e.g. process plants), a bubble may additionally have a list of "anchors". An anchor has a position, and an identifier. An anchor can, for example, be a small text label on an electrical outlet, or it can be a small, unique object in the room, such as a cactus. If the app cannot relocalize itself based on the SLAM map, the user simply needs to point the phone at 3 or more such anchors, and the app will re-localize itself correctly, and store an updated SLAM map to the server again.

4. Map specification

4.1 Introduction

A map specification aims to specify the pivot format of the map that will be used to develop the services of the ARtwin cloud platform. Thus, the map specification system needs to be able to represent any map which is created by any relocalization methods. To meet these requirements, the map has adopted the following design objectives.

- Support a variety of encoding formats, including the Extensible Markup Language (XML)
- Provide alternative application programmer interfaces (APIs)
- Allow for the map specification to be implemented at varying relocalization methods.

Figure 4-1 shows a conception of the map specification including map formats, map interfaces and map components. In these following sessions of this chapter, we describe elements of each components in details.

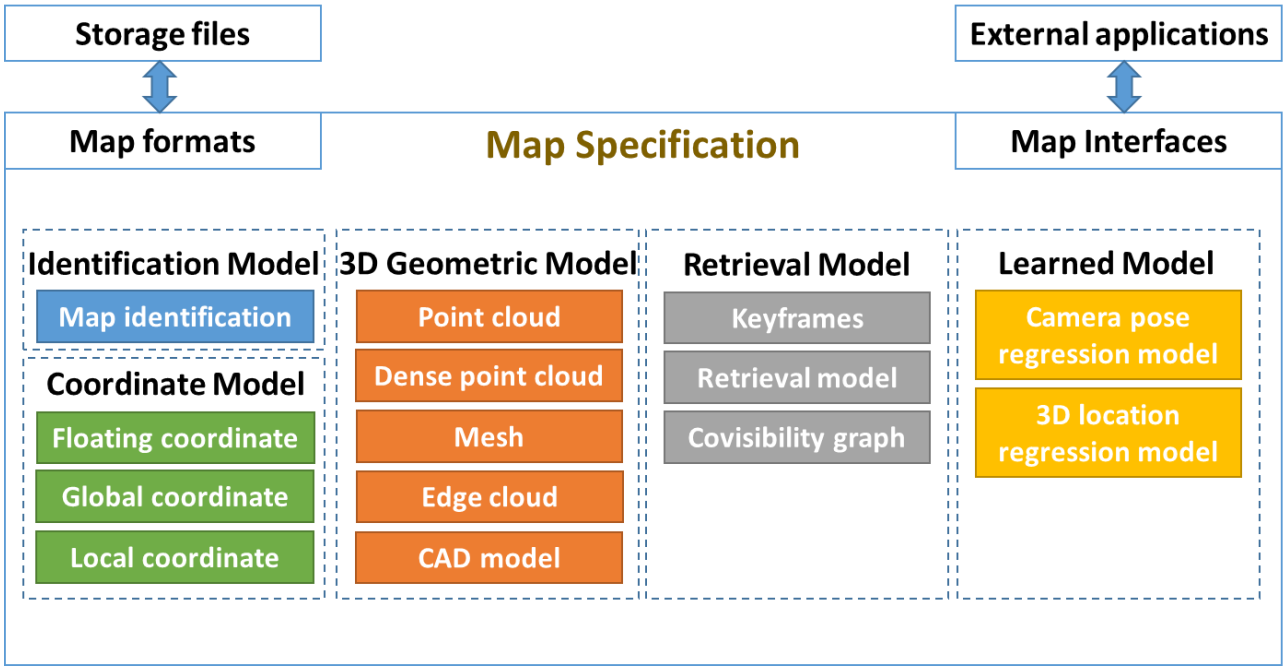


Figure 4-1. Map specification conception

4.2 Data types

First of all, this section describes the syntax and elemental data types used by the map specification.

4.2.1 Type naming

To understand names of field types that you can meet in this document, we present their name formats. They consist of two main fields: class type and variable type.

- **Class type**

Classes have names that begin with MAP. We use a pseudo template in bellow to describe attributes of each class:

```
Class [ClassName] {
      [AttributeType]    [AttributeName]    [DefaultValue] [*]
}
```

- `[ClassName]`: The name of class is corresponding `MAPxxxx`.
- `[AttributeType]`: The type of attribute.
- `[AttributeName]`: The name of the attribute.
- `[DefaultValue]`: The initial value of this attribute.
- [*]: If the attribute is marked with an asterisk, it is optional.

- **Variable type**

  Variable types are formatted by `Mapxxxxyyyy`. Where:

- `xxxx` defines this variable type is single or multiple. If it is null, this variable is a single value. Otherwise, it may be containers for example: `Vector`, `List`, `Set`, `Map`,...
- `yyyy` is the name of variable type.

4.2.2 *Types*

MapByte

The MapByte field specifies one 8-bits signed integer.

MapUInt8

The MapUInt8 field specifies one 8-bit unsigned integer.

MapInt32

The MapInt32 field specifies one 32-bit signed integer.

MapInt64

The MapInt64 field specifies one 64-bit signed integer.

MapFloat

The MapFloat field specifies one single-precision floating point number.

MapDouble

The MapDouble field specifies one double-precision floating point number.

MapUUID

The MapUUID field specifies a Universally Unique IDentifier (UUID) that is a 128-bit number. It is based on Boost UUID library.

MapTime

The MapTime field specifies a time value. It is defined as a double-precision floating point number. The allowable form for a double precision floating point number is defined in the specific encoding. Time values

are specified as the number of milliseconds from a specific time origin. Typically, MapTime fields represent the number of milliseconds since Jan 1, 1970, 00:00:00 GMT.

MapString

The MapString field contains sequences of characters encoded with the UTF-8 universal character set.

MapSPtr

The MapSPtr field is a smart pointer that retains shared ownership of an object through a pointer. Several MapSPtr objects may own the same object.

MapUPtr

The MapUPtr field is a smart pointer that owns and manages another object through a pointer and disposes of that object when the MapUPtr goes out of scope.

MapVector2Di

The MapVector2D field specifies a two-dimensional (2D) vector. MapVector2D is represented as a pair of single-precision integer values.

MapVector2Df

The MapVector2D field specifies a two-dimensional (2D) vector. MapVector2D is represented as a pair of single-precision floating point values.

MapVector3Di

The MapVector3D field specifies a three-dimensional (3D) vector. MapVector3D is represented as a tuple of single-precision integer values.

MapVector3Df

The MapVector3D field specifies a three-dimensional (3D) vector. MapVector3D is represented as a tuple of single-precision floating point values.

MapMatrix3Df

The MapMatrix3D field specifies a 3×3 matrix of single-precision floating point numbers. MapMatrix3D matrices are organized in row-major fashion. The default value of an uninitialized MapMatrix3D field is the identity matrix [1 0 0 0 1 0 0 0 1].

MapTransform3Df

The MapTransform3D field specifies a 4x4 matrix of single-precision floating point numbers to define a transform in 3D space. MapTransform3D matrices are organized in row-major fashion. The default value of an uninitialized MapTransform3D field is the identity matrix [1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1].

MapURL

A URL (Uniform Resource Locator), described in RFC 1738, is a form of Universal Resource Identifier (URI) that specifies a file located on a particular server and accessed through a specified protocol (e.g. http).

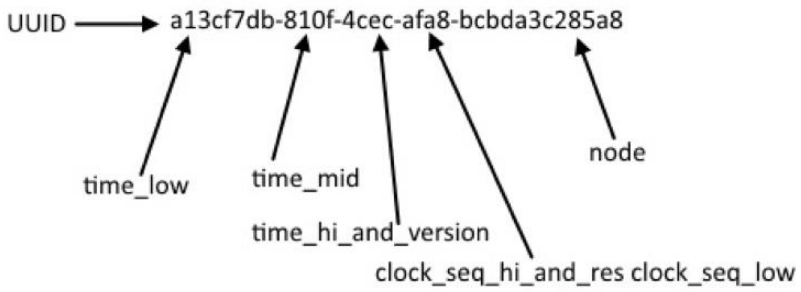
4.3 Map

```
Class Map {
      MapIdentification                  m_id                  null
      MapCoordinateSystem                m_coordSystem         null
      Map3DModels                        m_3DModels            null *
      MapRetrievalModels                 m_retrievalModels     null *
      MapLearnedModels                   m_learnedModels       null *
}
```

4.4 Map identification

```
Class MapIdentification {
      MapUUID          m_uuid                0
      MapString        m_name                ""
      MapString        m_author              ""
      MapTime          m_createdTime         0     *
      MapTime          m_lastObservationTime 0     *
      MAPBBox3Df       m_bbox                null  *
}
```

- `m_uuid`: The UUID to identify the map.
- `m_author`: The author's name of the map.
- `m_name`: Specify the name of the map.
- `m_createdTime`: a timestamp corresponding to the creation of the map.
- `m_lastObservationTime`: a timestamp corresponding to the last observation of the map.
- `m_bbox`: The bounding box of the map.



Name	Length(Hex Digits)
time_low	8
time_mid	4
time_hi_and_version	4
clock_seq_hi_and_res clock_seq_low	4
node	12

Figure 4-2. The UUID record layout

The map identification component is to identify a map that is created by a service client. It includes identification and general information. Each map implementation is identified with a UUID to nearly ensure the uniqueness of a map when the system instantiates it. When generated according to the standard methods, UUIDs are for practical purposes unique. Their uniqueness does not depend on a central registration authority or coordination between the parties generating them, unlike most other numbering schemes. While the probability that a UUID will be duplicated is not zero, it is close enough to zero to be negligible. Figure 4-2 shows an UUID example and the UUID record layout.

Additionally, this component is also identified by its name, author's name. Moreover, it saves the created time and the last updated time. And the final element in this component is a 3D bounding box that covers whole map. We define a bounding box structure below. It consists of the 3D behind left under corner (X_C, Y_C, Z_C) and the size (Width, Height, Depth) as presented in Figure 4-3.

```
Struct MapBBox3Df {
      MapVector3Df     m_behindLeftUnder      (0, 0, 0)
      MapVector3Df     m_size                 (0, 0, 0)
}
```

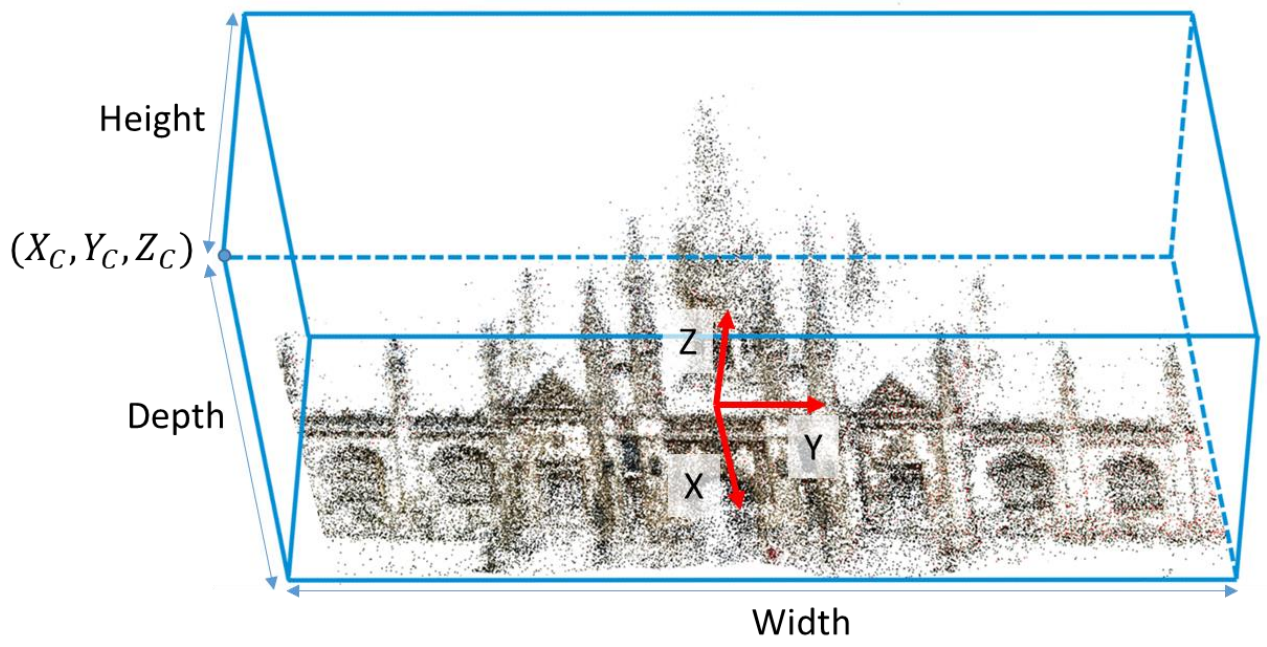


Figure 4-3. The3D bounding box covers the entire map.

4.5 Reference coordinate systems

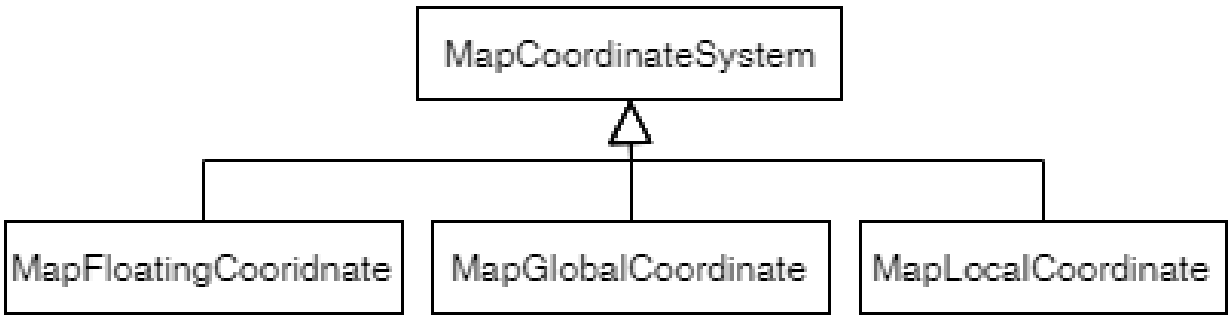


Figure 4-4. Derived classes from MapCoordinateSystem class

Each map is attached to a specific coordinate system. It can be a floating coordinate system or be related to other coordinate systems: global and local reference coordinates systems.

```
Class MapCoordinateSystem {
      MapString                          m_name                ""          *
      MapMap<string, MapTransform3Df>    m_anchors             null        *

}
```

- `m_name`: The name of this coordinate system.
- `m_anchors`: A set of anchors defined by their name and a 3D transform related to the coordinate system.

```
Class MapFloatingCoordinate: public MapCoordinateSystem{
```

```
}
```

The floating coordinate system is that does not have any links to other maps or its coordinates are continuously changing in other references for example coordinate system of a bus running on the road is change over time compared to the earth coordinate system.

```
Class MapGlobalCoordinate: public MapCoordinateSystem{
      MapVector3Df          m_globalPosition          (0, 0, 0)
      MapVector3Df          m_globalRotation          (0, 0, 0)
}
```

- `m_globalPosition`: Locate the position in the geographic coordinate system.
- `m_globalRotation`: Define the rotation in the geographic coordinate system.

The global coordinate system specifies a geographic coordinate of a map in the earth coordinate system based on GeoPose. It is the combination of position (x, y, z or longitude, latitude, elevation) and orientation (pitch, roll, and yaw). The position is represented respectively longitude and latitude in decimal degrees using the WGS84 datum and elevation in meters. Latitudes range from -90 to 90, and longitudes range from -180 to 180. The elevation is its height above or below a fixed reference point, most commonly a reference geoid, and a mathematical model of the Earth's sea level.

```
Class MapLocalCoordinates: public MapCoordinateSystem {
      MapUUID               m_parentID                0
      MapTransform3Df       m_parentTransform         Identity
}
```

- `m_parentID`: The ID of the parent map that this coordinate system is related to.
- `m_parentTransform`: The 3D transformation determines the position and orientation of this coordinate system relative to the parent coordinate system.

And the local reference coordinate system is what this map belongs to. It stores a 3D transformation between this coordinate system and its parent coordinate system. Figure 4-5 illustrates an example of a building coordinate system that is related to a city coordinate system.

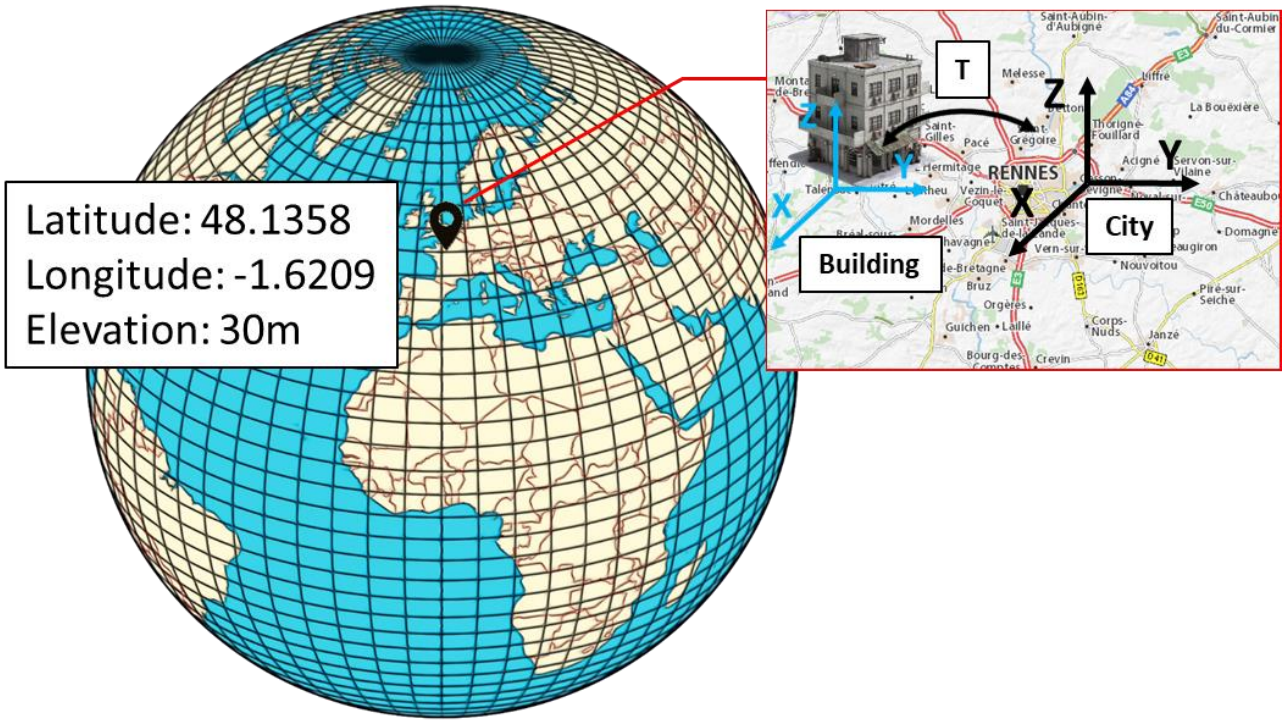


Figure 4-5. A reference coordinate example of a building map. It has a global coordinate in the earth coordinate system and a local coordinate in the city map.

4.6 3D Primitives

```
Class Map3DPrimitive {
      MapFloat                    m_confidence          0.0   *
      MapInt32                    m_usedTimes           0     *
      MapTime                     m_timestamp           0     *
      MapInt                      m_label               0     *
}
```

- `m_confidence`: Confidence score.
- `m_usedTimes`: Number of times used.
- `m_timestamp`: a timestamp corresponding to the last time the primitive has been observed.
- `m_label`: the ID of the label corresponding to an object to which this primitive belongs.

The Map3DPrimitive is an abstract class from which any 3D primitive will inherit. The confidence score on a primitive is higher if the system estimates that the primitive is well present in the real space (for example if this primitive has been observed for a long time by many AR systems without detecting it as an outlier). The used time specified the number of time the primitive has been considered as an inlier by AR systems, and the timestamp is the last time an AR system has consider the primitive as an inlier. It also stores an identification of the object class which a primitive belongs to.

Any 3D primitive inherits from Map3DPrimitive as shown in the following inheriting UML diagram.

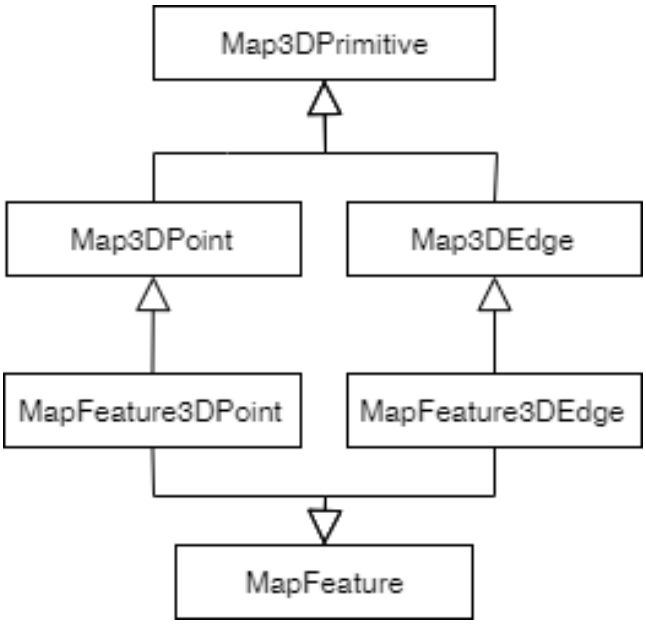


Figure 6: Derived classes from the Map3DPrimitive class.

```
Class Map3Dpoint: public Map3DPrimitive {
      MapVector3Df                    m_position          (0, 0, 0)
      MapVector3Di                    m_color             (0, 0, 0)   *
      MapVector3Df                    m_normal            (0, 0, 0)   *
}
```

- `m_position`: 3D position of a point in the world coordinate system.
- `m_color`: Color of a point in RGB space.
- `m_normal`: Normal direction.

The *Map3Dpoint* class represents a 3D point positioned in the coordinate system of the map with its RGB color and a surface normal vector. Although the surface normal is basically obtained from a mesh, we can use approximations to infer the surface normal from a dense point cloud (not relevant for a sparse point cloud) directly based on neighboring map points. Figure 4-7**Σφάλμα! Το αρχείο προέλευσης της αναφοράς δεν βρέθηκε.** illustrates the surface normal estimation from neighboring pairs of 3D points by exploiting the regular grid structure.

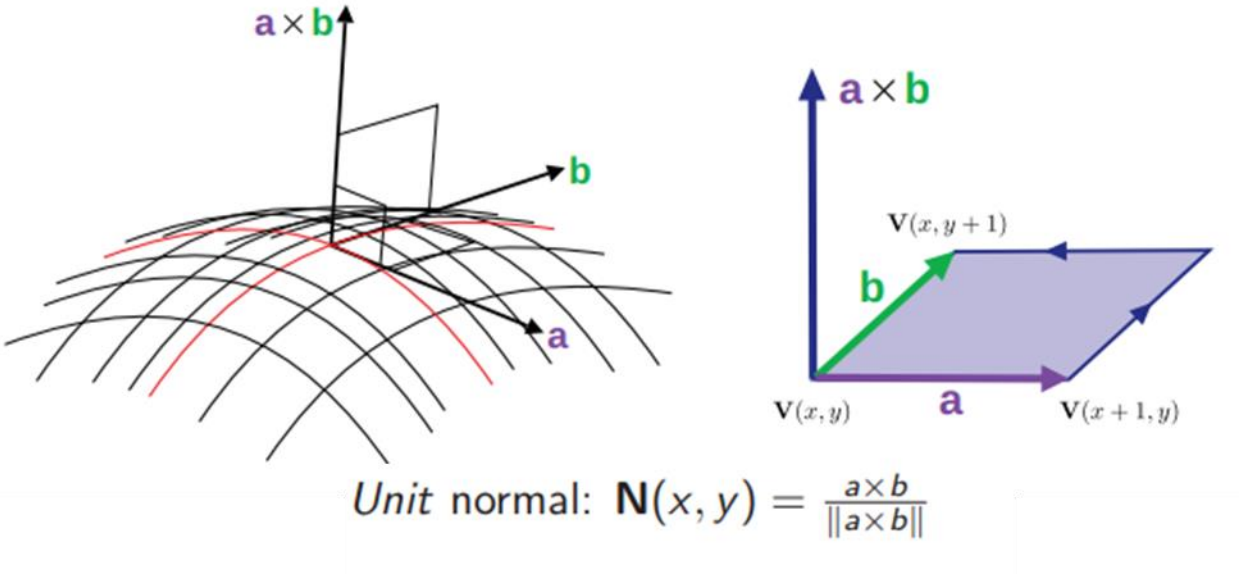


Figure 4-7. Surface normal estimation from neighboring pairs of 3D points.

```
Class Map3DEdge: public MAP3DPrimitive {
      MapVector3Df                    m_startPoint          (0, 0, 0)
      MapVector3Df                    m_endPoint            (0, 0, 0)
}
```

- `m_startPoint`: Starting point.
- `m_endPoint`: End point.

Each 3D edge is specified by a starting 3D point and an end 3D point.

```
Class MapFeature {
      MapSPtr<MapDescriptor>          m_descriptor          null
      MapMap<MapInt32, MapInt32>      m_keyframeVisibility  null       *
}
```

- `m_descriptor`: Description vector (2D or 3D descriptor).
- `m_keyframeVisibility`: Map storing keyframes observing this 3D point, where the first element corresponds to the index of the keyframe, and the second element to the index of the keypoint in this keyframe.

```
Class MapFeature3DPoint: public MapFeature, public Map3Dpoint {
      MapVector3Df                    m_viewingDirection    (0, 0, 0)  *
}
```

- `m_viewingDirection`: Mean viewing direction.

Each feature 3D cloud point obviously contains its 3D position in the world coordinate system of the map, its color which is retrieved by RGB image observations, the viewing direction which is the mean unit vector of

all its viewing directions (the ray that join the point with the optical center of the frame that observe it), and keypoint indices of keyframes observing this 3D point.

Moreover, it is represented by a description vector that allows camera relocalization methods to perform feature matching in order to define point correspondences. This descriptor is a local feature computed from appearance information (e.g. SIFT, SURF, ORB, AKAZE feature) or geometrical information (e.g. FPFH, VFH feature) of local neighborhood of each keypoint as illustrated in Figure 4-8.

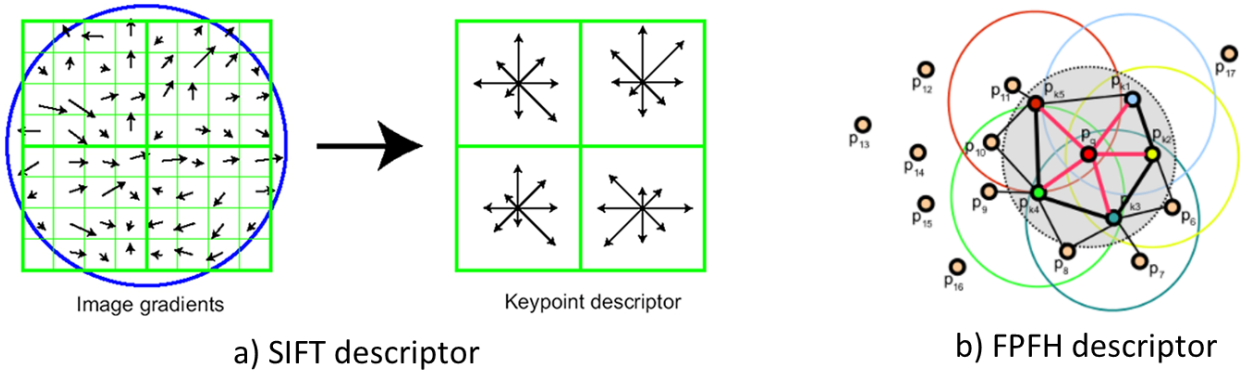


Figure 4-8. Feature descriptors: a) SIFT descriptor computed based on image appearance. b) FPFH descriptor computed based on 3D geometrical points.

Because scenes are always dynamic by illumination change, appearance change, occlusion and moving objects, the above information of each cloud point can be no longer accurate. Therefore, the information about confidence score, number of times a point used, last updated time are stored in each cloud point.

```
Class MapFeature3DEdge: public MapFeature, public Map3DEdge {


}
```

The *MapFeature3DEdge* class represent a 3D edge with an associated descriptor.

4.7 3D geometric model

```
Class Map3DModels {
      MapInt                                m_size              0
      MapList<MapSPtr<Map3DModel>>          m_models            null
}
```

- `m_size`: Number of 3D models in the map.
- `m_ models`: A list of 3D models.

A map can be represented by a set of 3D models such as: 3D feature point clouds, 3D non-feature point clouds, 3D edge clouds, meshes, CAD models. The set of models of a given map are grouped together in this class. The different kind of models are necessary to adapt to various camera relocalization methods as presented in Section 2.1.2.

```
Class Map3DModel {
      MapString                          m_name              ""
      MapString                          m_source            ""    *
      MapMap<MapInt, MapString>          m_labels            null  *
}
```

- `m_size`: the name of the 3D model.
- `m_source`: Source of the 3D model.
- `m_labels`: A list of labels attached to the 3D primitives of the map with their associated ID and name.

This class is abstract, and represent any 3D model. A name and a source can be specified for each 3D model. The source of the map can help to estimate the confidence level that can be given to this model (e.g.: is the map come from a certified organism, or is it a crowd sourced map). Also, a MapMap of labels with their associated ID can be provided, these ID will be referenced by each 3D primitive stored into the map.

Hence, this 3D geometric model component contains a list of possible 3D object. They are inherited from the 3D Model class `Map3DModel` as described in the below diagram .

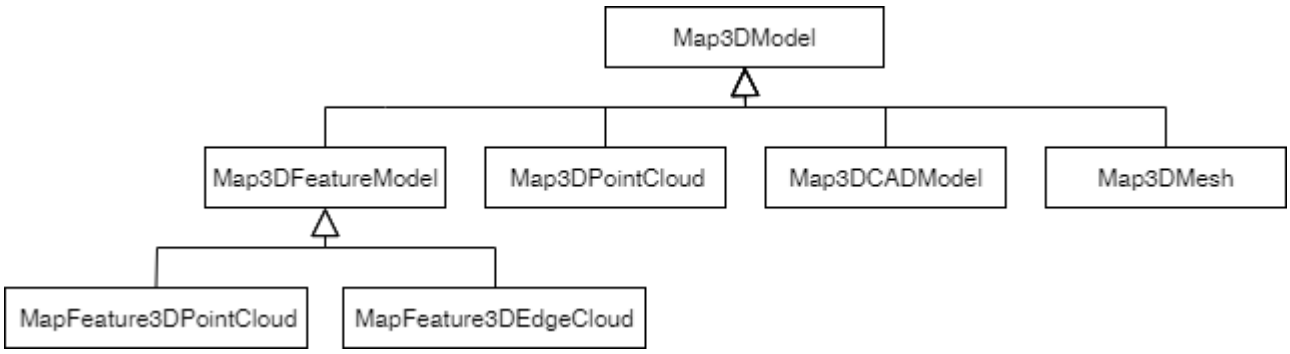


Figure 4-9. Derived class from MAP3DModel class

```
Class MapFeature3DModel {
      MapEnumKeypointDetectorType        m_detectorType      0     *
      MapEnumKeypointDescriptorType      m_descriptorType    0
}
```

- `m_detectorType`: Feature detection type used by the feature model.
- `m_descriptorType`: Feature extraction type used by the feature model.

The MapFeature3DModel class is abstract and represent any 3D model represented by a set of 3D primitives (e.g. points or edges) for which a feature detector and descriptor is associated (e.g. SIFT, SURF, ORB, or LSD). The feature detector and descriptor are essential if we want to match the 3D primitives of the model with the primitives extracted in a given image. For example, a feature 3D point cloud model is shown in Figure 2-4 that is constructed by SfM from a set of RGB images based on SURF feature detection and extraction.

4.7.1 3D feature point cloud

```
Class MapFeature3DPointCloud: public MapFeature3DModel {
      MapList<MapSPtr<MapFeature3DPoint>>    m_points         null
}
```

- `m_points`: A list of points with the corresponding feature descriptors.

The feature 3D point cloud is the most popular feature 3D geometric model used for relocalization. It is used in the majority of SLAM methods. A 3D feature point cloud consists of a set of 3D points in which each 3D point is attached with a description vector. This component includes a detector type and a descriptor type inherited from `MapFeature3DModel` to specify what kind of feature is used to construct the point cloud.

4.7.2 3D point cloud (without feature)

```
Class Map3DpointCloud: public Map3DModel {
      MapList<MapSPtr<Map3Dpoint>>           m_points         null
}
```

- `m_points`: A list of cloud points.

Figure 4-10. A 3D non-feature point cloud is constructed by RealSense camera.

Similar to 3D feature point cloud, a 3D point cloud consists of a set of 3D points. However, this point cloud is normally denser than one of the feature 3D model. They are often rapidly created by depth sensors. Figure 4-10 shows a 3D point cloud (without features) fused by using some depth images of RealSense camera. The main difference between feature 3D point cloud and simple 3D point cloud concerns elements of each cloud point. Whereas a feature 3D point is represented by a descriptor vector, a simple 3D point has no attached descriptor. A cloud point in the 3D point cloud is described in the `Map3Dpoint` class.

4.7.3 3D edge cloud

```
Class MapFeature3DEdgeCloud: public MapFeature3DModel {
      MapList<MapSPtr<MapFeature3DEdge>>     m_edges          null
```

```
}
```

- `m_edges`: A list of edges.

Edge-based approach is well-known methods for (re)localization in texture-less scenes. A 3D edge cloud consists of a set of 3D edges in which each 3D point is attached with a description vector. This component includes a detector type and a descriptor type inherited from the *MapFeature3DModel* class to specify what kind of feature is used to construct the 3D edge cloud. Figure 4-11 shows a 3D constructed edge cloud based on LSD line detection. For more detail elements contained in a 3D edge, see the `Map3DEdge` class description.

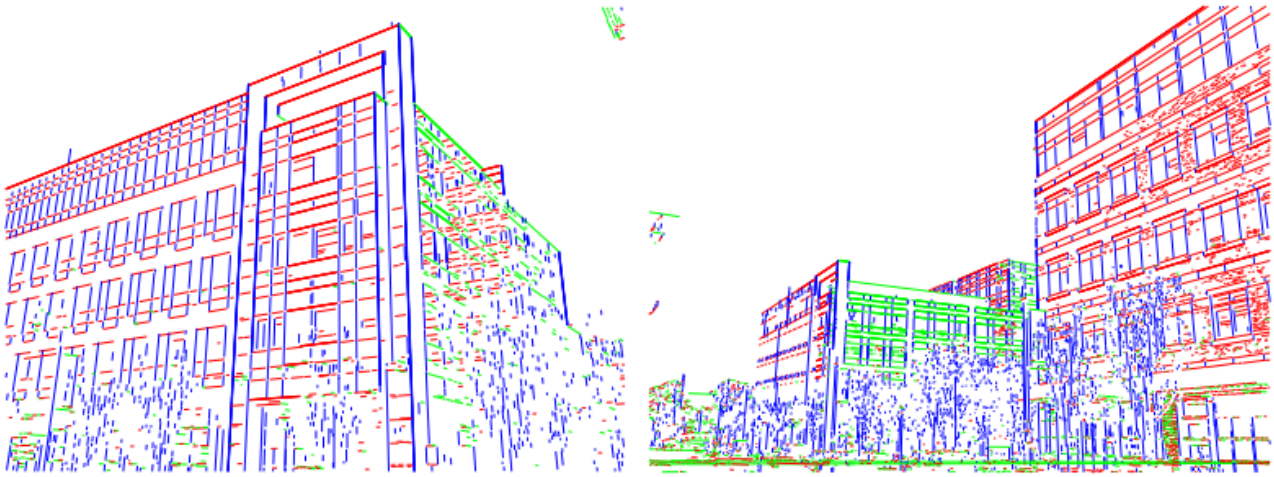


Figure 4-11. 3D edges reconstruction

4.7.4 Mesh

```
Class Map3DMesh: Map3DModel {
      MapVector<MapVertex>                    m_vertices          null
      MapVector<MapTexture>                   m_textures          null  *
      MapVector<MapPolygon>                   m_polygons          null
}
```

- `m_vertices`: A set of vertices.
- `m_textures`: A set of textures.
- `m_polygons`: A set of polygons.

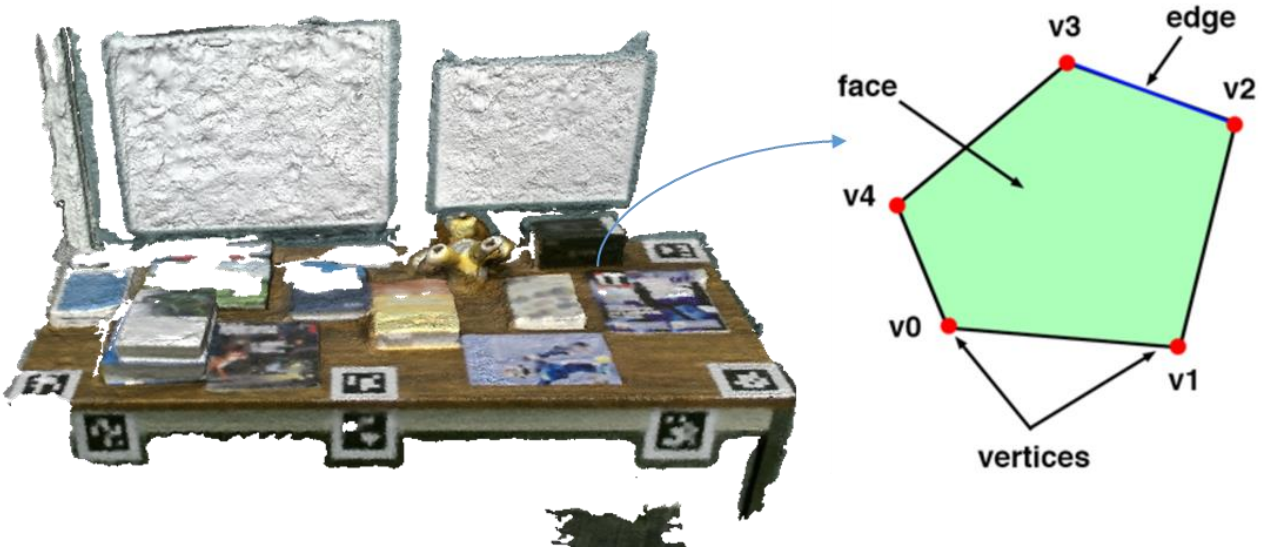


Figure 4-12. A mesh sample and principal elements in a mesh.

A typical mesh should at least need a set of vertices where each vertex contains a position vector, a normal vector and a texture coordinate vector. A mesh should also contain indices for indexed drawing and material data in the form of textures (ambient, diffuse, specular colors).

Now that we set the minimal requirements for a mesh class we can define a vertex:

```
Struct MapVertex {
      MapVector3Df                    m_position              (0, 0, 0)
      MapVector3Df                    m_normal                (0, 0, 0)   *
      MapVector2Df                    m_textureCoordinate     (0, 0)      *
}
```

- `m_position`: 3D position.
- `m_normal`: 3D normal.
- `m_textureCoordinate`: Texture coordinate vector.

We also want to organize the texture data in a texture struct:

```
Struct MapTexture {
      MapString                                 m_format              null
      MapURL                                    m_url                 null
      }
```

- `m_format`: A string describing the format use to encode the texture.
- `m_url`: The URL of the texture.

And we define a polygon struct that contains a set of indices of vertices concerning a face.

```
Struct MapPolygon {
      MapVectorInt                m_indices        null
```

```
}
```

- `m_indices`: A set of indices.

4.7.5 CAD model

Computer-aided design (CAD) is the use of computers (or workstations) to aid in the creation, modification, analysis, or optimization of a design. CAD software is used to increase the productivity of the designer, improve the quality of design, improve communications through documentation, and to create a database for manufacturing. CAD output is often in the form of electronic files for print, machining, or other manufacturing operations.

```
Class Map3DCADModel: Map3DModel {
      MapString                              m_format              null
      MapURL                                 m_url                 null
}
```

- `m_format`: A string describing the format use to encode the CAD model.
- `m_url`: The URL of the CAD model.

An example of CAD model format is the Jupiter Tesselation (JT format). It is an efficient, industry-focused and flexible ISO-standardized 3D data format developed by Siemens PLM Software which allows to represent a CAD model with a mesh. Mechanical CAD domains of Aerospace, automotive industry, and Heavy Equipment use JT as their most leading 3D visualization format. JT format is a scene graph that supports the attributes and nodes that are CAD specific. Sophisticated compression techniques are used to store facet data (triangles). This format is structured to support visual attributes, product and manufacturing information (PMI), and Metadata.

But other CAD model formats are using a Boundary Representation (e.g. BREP) which represent a solid as a collection of connected surface elements, the boundary between solid and non-solid. The elements are not described with a mesh, but with primitive objects on which Boolean operations are applied (Constructive Solid Geometry), or by mathematical functions (e.g. Bezier curve, B-Spline, or NURBS). This model as the advantage of being more accurate, but can require more computation to extract relevant features compared to mesh models.

4.8 Retrieval model

```
Class MapRetrievalModels {
      MapInt                                 m_size                0
      MapList<MapSPtr<MapRetrievalModel>>    m_listRetrievalModels null
}
```

- `m_size`: Number of retrieval models in the map.
- `m_listRetrievalModels`: A list of retrieval models.

Image retrieval approach performs camera relocalization through nearest images retrieval that is well-known as place recognition problem. The final camera pose is achieved based on either estimating the relative pose between the query image and retrieved images, or the absolute pose using a geometric approach.

This component specifies retrieval models for camera relocalization that consist of various methods as presented in Section 2.1.4. Therefore, this component contains a list of possible retrieval model objects. They are inherited from the `MapRetrievalModel` class.

```
Class MapRetrievalModel {
      MapString                                m_source              ""    *
}
```

- `m_source`: Source of this retrieval model.

4.8.1 Keyframe retrieval

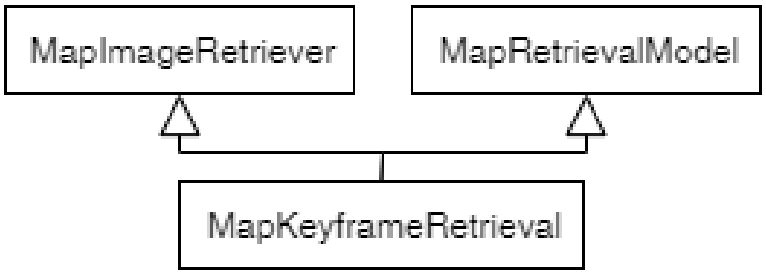


Figure 4-13. Inheritance diagram for MapKeyframeRetrieval

```
Class MAPKeyframeRetrieval: public MapRetrievalModel, public
MapImageRetriever {
      MapList<MapSPtr<MapKeyframe>>          m_keyframes           null
      MapSPtr<MapCovisibilityGraph>          m_covisGraph          null  *
}
```

- `m_keyframes`: A list of keyframes.
- `m_covisGraph`: Covisibility information between keyframes.

```
Class MapImageRetriever {
      MapEnumRetrieverDescriptorType         m_descriptorType      0
      MapList<MapSPtr<MapDescriptor>>        m_descriptors         null
}
```

- `m_descriptorType`: Descriptor type.
- `m_descriptors`: A list of global-level descriptors for keyframes.

Keyframe retrieval model aims at retrieving a set of nearest images that are the most similar to a query image. This model stores a list of keyframes that are captured from different viewpoints, and their level-image feature. These features are specified by retrieval methods such as local feature aggregation, global learned feature. This allows to accelerate matching process.

In addition, this model has also a covisibility graph providing information between keyframes. We describe more details information of keyframe and covisibility graph in the next sections.

4.8.2 Keyframe

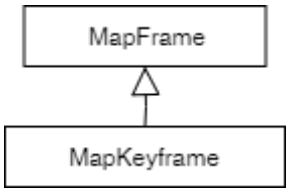


Figure 4-14. Inheritance diagram for MapKeyframe

```
Class MapKeyframe: public MapFrame {
      MapInt32                          m_id                0
      MapFloat                          m_confidence        0.0         *
      MapTime                           m_updatedTime       0           *
      MapMap<string, mapTrasform3Df>    m_anchors           null        *
}
```

- `m_id`: Identification of each keyframe.
- `m_confidence`: Confidence score.
- `m_updatedTime`: Last updated time.
- `m_anchors`: A set of anchors defined by their name and a 3D transform related to the keyframe pose.

Anchors can be attached to a keyframe. It is useful as the anchors will move with the keyframe when this last is updated for instance by a bundle adjustment during a loop closure.

```
Class MapFrame {
      MapTransform3Df             m_pose                      Identity
      SPtr<MapImage>              m_image                     null        *
      MapDescritor                m_imageDescriptor           null
      MapMatrix3Df                m_intrinsic                 Identity    *
      MapVector<MapKeypoint>      m_keypoints                 null        *
      MapVector<MapDescriptor>    m_keypointdescriptors       null        *
      MapMap<MapInt32, SPtr<Map3DFeaturePoint>> m_visibility  null        *
}
```

- `m_pose`: Camera pose of the frame.
- `m_imageDescriptor`: A global descriptor extracted from the image of the frame.
- `m_image`: Raw image is stored in the frame.
- `m_intrinsic`: camera intrinsic matrix.
- `m_keypoints`: A vector of detected keypoints in the frame.
- `m_keypointDescriptors`: A vector of corresponding extracted features from keypoints.

- `m_visibility`: Map storing the 3D point visibilities, where the first element corresponds to the index of the keypoint of the frame, and the second element to the index of the corresponding cloud point.

```
Class MapImage {
      MapEnumImageType                m_type                0
      MapVector2Di                    m_size                (0, 0)
      MapInt32                        m_nbChannels          0
      MapVector<MapUInt8>             m_data                null
}
```

- `m_type`: Image type for example: grey image, RGB image, depth image.
- `m_size`: Width and height of image.
- `m_nbChannels`: Number of channels.
- `m_data`: Data buffer.

```
Class MapKeypoint {
      MapVector2Df                    m_xy                  (0, 0)
      MapFloat                        m_size                0          *
      MapFloat                        m_response            0          *
      MapFloat                        m_angle               0          *
      MapByte                         m_octave              0          *
}
```

- `m_xy`: Coordinates of keypoint.
- `m_size`: Diameter of the meaningful keypoint neighborhood.
- `m_reponse`: Response by which the strongest keypoints have been selected.
- `m_octave`: Octave (pyramid layer) from which the keypoint has been extracted.

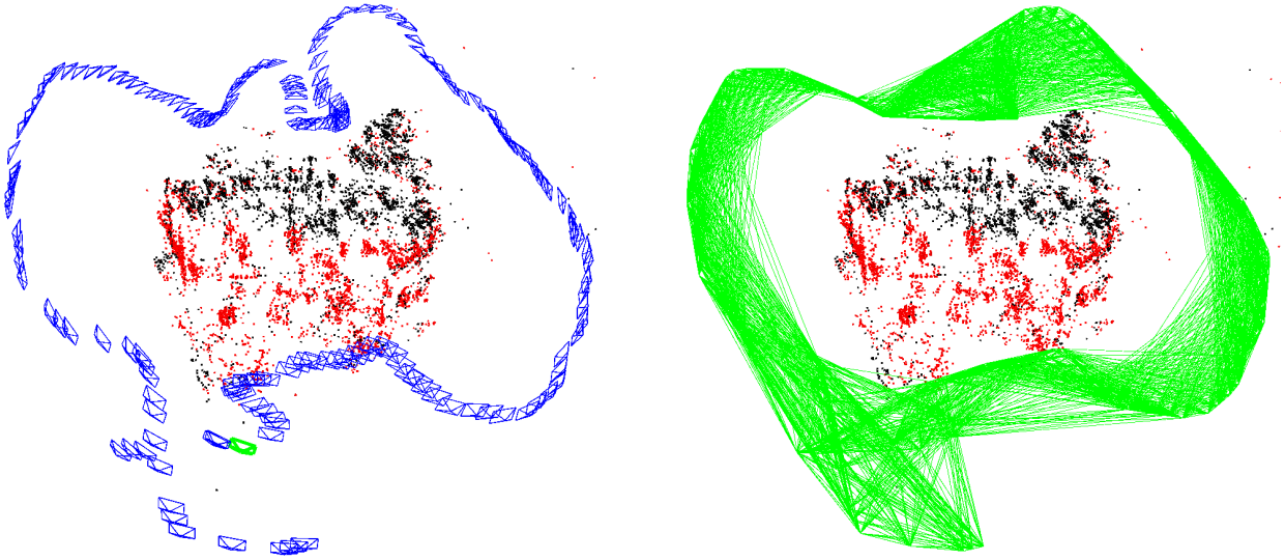


Figure 4-15. Left image: A set of keyframes (blue). Right image: covisibility graph.

Keyframes are selected from some of all images captured around a scene. They are required to approximately cover the entire space and at the same time minimize the content redundancy amongst the selected frames. Each keyframe stores the camera pose which is a rigid transformation between its camera coordinate and the world coordinate. Figure 4-15 (left image) shows a set of keyframes and their camera poses.

Moreover, the camera intrinsic matrix that specifies parameters of camera is stored in each keyframe as well.

Depend on camera relocalization methods, each keyframe also stores a raw image (RGB or depth), features extracted in this frame, associations to 3D map points. This information is to define the relative or global camera pose of a query image via its nearest keyframe.

4.8.3 Covisibility graph

```
Class MapCovisibilityGraph {
      MapSet<MapInt32>                          m_nodes              null
      MapMap<MapInt32, MapSet<MapInt32>>        m_edges              null
      MapMap<MapInt64, MapFloat>                m_weights            null
}
```

- `m_nodes`: A set of nodes
- `m_edges`: A set of nodes links for each node.
- `m_weights`: A map of edge-weights.

Covisibility graph information between keyframes is represented as an undirected weighted graph as in illustrated in Figure 4-15 (right image). Each node is a keyframe and an edge between two keyframes exists if they share observations of the same map points, being the weight of the edge the number of common map points. This graph is necessary to AR cloud because it allows to define a sub-map which associates to the current camera pose. Indeed, Figure 4-15 (left image) shows a current camera view (green) and current local map points (red).

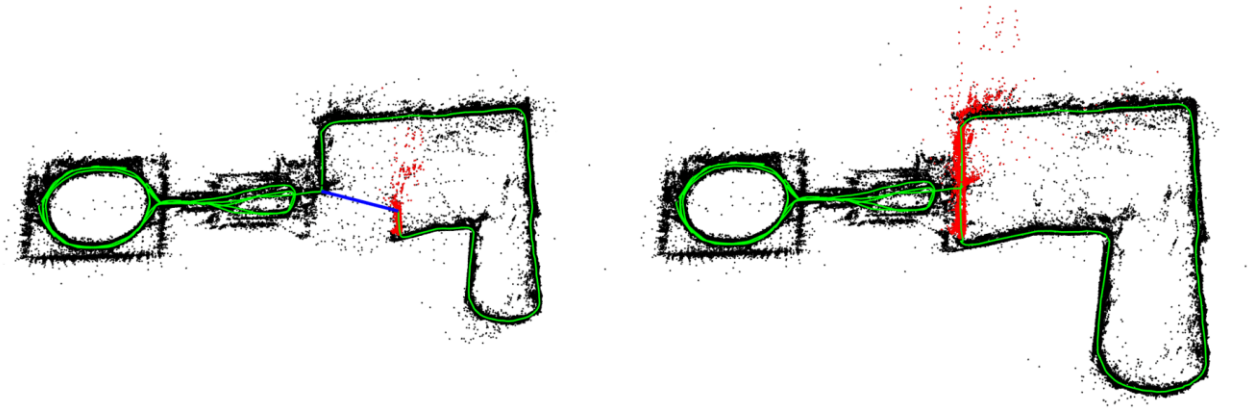


Figure 4-16. Map before and after a loop closure. The loop closure match is drawn in blue, the trajectory in green, and the local map for the tracking at that moment in red.

Moreover, this graph is also to optimize the map by applying a bundle adjustment or pose graph optimization to a loop closure detection. Figure 4-16 illustrates the reconstructed map before and after a loop closure optimization.

4.9 Machine learning model

```
Class MapLearnedModels {
      MapInt                                m_size                 0
      MapList<MapSPtr<MapLearnedModel>>     m_learnedModels        null
}
```

- `m_size`: Number of learned models in the map.
- `m_learnedModels`: A list of learned models.

This component specifies learned models of a map for machine learning based camera relocalization methods presented in Section 2.1.3 and Section 2.1.5. This component contains a list of possible learned models that consists of two principal types: camera pose regression and 3D location regression. They are inherited from the `MapLearnedModel` class as described in Figure 4-17. Each learned model stores an identification, source information as well as its model type that indicates what kind of machine learning method is applied (e.g. deep learning, random forest, …).

```
Class MapLearnedModel {
      MapInt32                              m_id                   0
      MapURL                                m_url                  ""
      MapEnumModelType                      m_type                 0
      MapString                             m_format               ""
      MapFloat                        m_confidence          0.0          *
      MapTime                         m_updatedTime         0            *
}
```

- `m_id`: Identification of the model.

- `m_url`: The url where to download the learned model.
- `m_type`: The type of the model.
- `m_format`: The format used to encode the learned model (e.g ONNX, CAFFE, or Tensorflow).
- `m_confidence`: Confidence score.
- `m_updatedTime`: Last updated time.

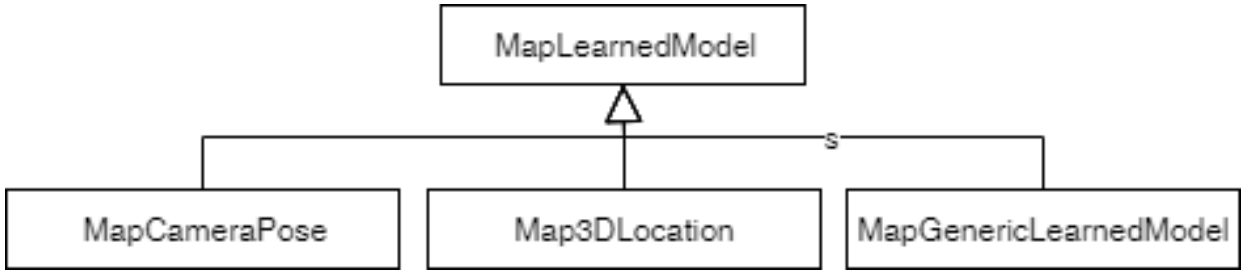


Figure 4-17. Derived classes from `MapLearnedModel` class

4.9.1 Camera pose regression

```
Class MapCameraPoseRegression: public MapLearnedModel {
      MapEnumDataType             m_inputType                RGB_IMAGE
}
```

- `m_inputType`: Input data type of the model.

Camera pose regression is an end-to-end approach for camera relocalization. It regresses directly camera pose from each input data. The model is trained specifically for each scene by using a set of training data and its camera pose ground-truth. Because the type of input data is various (for example: RGB image, depth image, stereo images), this component stores `m_inputType` aiming at defining what kind of input data used for this model. It is inherited from `MapLearnedModel` components.

4.9.2 3D location regression

```
Class Map3DLocationRegression: public MapLearnedModel {
      MapEnumFeatureType          m_featureType              0
}
```

- `m_featureType`: Feature type of the model.

Different from camera pose regression, this component predicts 3D locations corresponding to feature extracted around keypoints of each input image in order to define 2D-3D or 3D-3D point correspondences. The camera pose is then estimated by applying geometric algorithms such as PnP, Ransac, Kabsch. The 3D location regression component specifies the feature extraction type that related to size of feature, number of channels, and type of data. It is also inherited from `MapLearnedModel` components.

4.9.3 3D Generic learned model

```
Class MapGenericLearnedModel: public MapLearnedModel {
      MapEnumDataType             m_inputType            RGB_IMAGE
      MapEnumDataType             m_outputType           CAMERA_POSE
```

```
}
```

- `m_inputType`: Input data type of the model.
- `m_outputType`: Output data type of the model.

Many non-end-to-end approaches have been proposed by the research community. The generic learned model aims at covering the various kind of learned models used in computer vision pipelines. It defines the type of the input data which will feed the model, and the output data inferred by the model.

4.9.4 Feature Type

```
Enum MapEnumFeatureType {
      #2D Keypoints
      UNKNOWN = 0
      SIFT = 1
      SURF = 2
      ORB = 3
      KAZE = 4
      AKAZE = 5
      BRIEF = 6
      BRISK = 7
      FAST = 8
      LIFT = 9
      MICROSOFT = 10
      GOOGLE = 11
      APPLE = 12
      MAGICLEAP = 13
      NREAL = 14
      #2D lines
      LSD = 40
      MSLD = 41
      LBD = 42
      BRLD = 43
      #3D points
      CRH = 80
      CVFH = 81
      FPFH = 82
```

```
      GASD = 83
      GFPFH = 84
      GRSD = 85
      NARF = 86
      PFH = 87
      RIFT = 88
      RSD = 89
      SHOT = 90
      SHOT_COLOR = 91
      SPIN = 92
      UNIQUE_SHAPE = 93
      VFH = 94
}
```

These feature types aim at covering most of feature descriptors used by the computer vision community, but cannot be exhaustive. This enumeration could be extended in a future version of the specification if required.

4.9.5 Learned Model Data Type

```
Enum MapEnumDataType {
      RGB_IMAGE = 0x00000001
      DEPTH_IMAGE = 0x00000002
      RGBD_IMAGE = 0x00000003
      STEREO_RGB_IMAGE = 0x00000004
      STEREO_DEPTH_IMAGE = 0x00000008
      STEREO_RGBD_IMAGE = 0x0000000C
      KEYPOINT = 0x00000010
      KEYPOINTS = 0x00000020
      2D_EDGE = 0x00000040
      2D_EDGES = 0x00000080
      RGB_PATCHE = 0x00000100
      DEPTH_PATCHE = 0x00000200
      CAMERA_LOCATION = 0x00000400
      CAMERA_ORIENTATION = 0x00000800
      CAMERA_POSE = 0x00000C00
      CAMERA_CALIBRATION = 0x00001000
```

```
      OBJECT_LOCATION = 0x00002000
      OBJECT_ORIENTATION = 0x00004000
      OBJECT_POSE = 0x00006000
      OBJECT_LABEL = 0x00008000
      POINT_3D_LOCATION = 0x00010000
      EDGE_3D_LOCATION = 0x00020000
      2D_2D_POINT_MATCHES = 0x00040000
      2D_3D_POINT_CORRESPONDANCES = 0x00080000
      3D_3D_POINT_CORRESPONDANCES = 0x00100000
}
```

The type of data are stored on 32 bits, allowing to combine them if the model takes a multimodal input or infers a multimodal output. These data types aims at covering any input or output data of a learned model, and could be extended in a future version of the specification if required.

4.10 Format

Map specification requires encoding formats including the Extensible Markup Language (XML) to store map data representation. XML supports structuring data, is similar to HTML, and is readable by systems and humans. It defines the structure of the map: hierarchical relationships among objects, initial values for objects, and dataflow connections between objects.

XML architecture consists of a header and component tags. The header is a single line of UTF-8 text identifying the file as an XML file, followed by the XML declaration that identifies the validating XML DTD.

```
<?xml version="1.0" encoding="UTF-8"?>
```

The component tags defines elements of a map. They are organised by nodes and child nodes. Each tag begins and ends with angle brackets < and >, respectively. A node's body is enclosed by a pair of matching open and closing tags, where a closing tag has a slash / prepended to the element name. The following illustrates the use of node and field syntax:

```
<component name="COMPONENT_NAME">
      <property name="ATTRIBUTE_NAME" value="VALUE"/>
</component>
```

The value of elements can be URL linking to compression files of point cloud, keyframes, retrieval model, machine learning model.

4.11 Conclusion

In this section, we presented our conception for the map pivot specification. The map specification system is able to represent any maps to cover all relocalization methods including geometric approach, image retrieval approach, machine learning approach, hybrid approach. This map specification can be adapted to commercial solutions such as ARKit, ARCore, HoloLens. Moreover, we define identification and reference

coordinates components of a map aiming at defining relationship of a set of maps. This map specification describes elements of a map that are specified by the classes and their inherited diagrams. These elements are saved in storage files and can be distributed onto AR cloud. Finally, this specification allows us to determine APIs and storage component implementation and develop the services of the ARtwin cloud platform.

References

[1] R. M. Haralick, D. Lee, K. Ottenburg, and M. Nolle, "Analysis and solutions of the three point perspective pose estimation problem," in *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991, pp. 592–598.

[2] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 25, no. 8, pp. 930–943, 2003.

[3] V. Lepetit, F. Moreno-Noguer, and P. Fua, "Epnp: An accurate o (n) solution to the pnp problem," *Int. J. Comput. Vis.*, vol. 81, no. 2, p. 155, 2009.

[4] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallogr. Sect. A Cryst. Physics, Diffraction, Theor. Gen. Crystallogr.*, vol. 32, no. 5, pp. 922–923, 1976.

[5] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[6] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," in *2011 international conference on computer vision*, 2011, pp. 2320–2327.

[7] S. Lieberknecht, A. Huber, S. Ilic, and S. Benhimane, "RGB-D camera-based parallel tracking and meshing," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, 2011, pp. 147–155.

[8] R. A. Newcombe *et al.*, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, 2011, pp. 127–136.

[9] K. F. Whelan, M. Kaess, M. F. Fallon, H. Johannsson, J. J. Leonard, and J. McDonald, "Kintinuous: Spatially Extended KinectFusion," in *AAAI 2012*, 2012.

[10] R. A. Newcombe, D. Fox, and S. M. Seitz, "Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 343–352.

[11] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *European Conference on Computer Vision*, 2014, pp. 834–849.

[12] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2017.

[13] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph.," in *Robotics: science and systems*, 2015, vol. 11.

[14] T. Whelan, R. F. Salas-Moreno, B. Glocker, A. J. Davison, and S. Leutenegger, "ElasticFusion: Real-time dense SLAM and light source estimation," *Int. J. Rob. Res.*, p. 0278364916669237, 2016.

[15] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM: Real-Time Dense Monocular SLAM With Learned Depth Prediction," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[16] B. Peasley and S. Birchfield, "RGBD Point Cloud Alignment Using Lucas--Kanade Data Association and Automatic Error Metric Selection," *IEEE Trans. Robot.*, vol. 31, no. 6, pp. 1548–1554, 2015.

[17] N. Snavely, S. M. Seitz, and R. Szeliski, "Photo tourism: exploring photo collections in 3D," in *ACM transactions on graphics (TOG)*, 2006, vol. 25, pp. 835–846.

[18] N. Snavely, S. M. Seitz, and R. Szeliski, “Modeling the world from internet photo collections,” *Int. J. Comput. Vis.*, vol. 80, no. 2, pp. 189–210, 2008.

[19] C. Wu, “Towards linear-time incremental structure from motion,” in *3DTV-Conference, 2013 International Conference on*, 2013, pp. 127–134.

[20] N. Jiang, Z. Cui, and P. Tan, “A global linear method for camera pose registration,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 481–488.

[21] P. Moulon, P. Monasse, and R. Marlet, “Global fusion of relative motions for robust, accurate and scalable structure from motion,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 3248–3255.

[22] K. Wilson and N. Snavely, “Robust global translations with 1dsfm,” in *European Conference on Computer Vision*, 2014, pp. 61–75.

[23] S. Agarwal, Y. Furukawa, N. Snavely, B. Curless, S. M. Seitz, and R. Szeliski, “Reconstructing rome,” *Computer (Long. Beach. Calif).*, vol. 43, no. 6, pp. 40–47, 2010.

[24] M. Pollefeys *et al.*, “Visual modeling with a hand-held camera,” *Int. J. Comput. Vis.*, vol. 59, no. 3, pp. 207–232, 2004.

[25] R. Hartley and A. Zisserman, “Multiple view geometry in computer vision,” *Robotica*, vol. 23, no. 2, p. 271, 2005.

[26] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, 1999, vol. 2, pp. 1150–1157.

[27] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *European conference on computer vision*, 2006, pp. 404–417.

[28] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, “ORB: An efficient alternative to SIFT or SURF.,” in *ICCV*, 2011, p. 2.

[29] P. F. Alcantarilla, A. Bartoli, and A. J. Davison, “KAZE features,” in *European Conference on Computer Vision*, 2012, pp. 214–227.

[30] R. I. Hartley and P. Sturm, “Triangulation,” *Comput. Vis. image Underst.*, vol. 68, no. 2, pp. 146–157, 1997.

[31] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, “Bundle adjustment—a modern synthesis,” in *International workshop on vision algorithms*, 1999, pp. 298–372.

[32] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, “Building rome in a day,” in *2009 IEEE 12th international conference on computer vision*, 2009, pp. 72–79.

[33] J. L. Schönberger and J.-M. Frahm, “Structure-from-Motion Revisited,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[34] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, “MonoSLAM: Real-time single camera SLAM,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 6, pp. 1052–1067, 2007.

[35] G. Klein and D. Murray, “Parallel tracking and mapping for small AR workspaces,” in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, 2007, pp. 225–234.

[36] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos, “ORB-SLAM: a versatile and accurate monocular SLAM system,” *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, 2015.

[37] M. Pupilli and A. Calway, “Real-Time Camera Tracking Using a Particle Filter.,” in *BMVC*, 2005.

[38] J. Kwon and K. M. Lee, "Visual tracking decomposition," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 1269–1276, 2010, doi: 10.1109/CVPR.2010.5539821.

[39] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977.

[40] J. S. Beis and D. G. Lowe, "Shape indexing using approximate nearest-neighbour search in high-dimensional spaces," in *cvpr*, 1997, vol. 97, p. 1000.

[41] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu, "An optimal algorithm for approximate nearest neighbor searching fixed dimensions," *J. ACM*, vol. 45, no. 6, pp. 891–923, 1998.

[42] C. Silpa-Anan and R. Hartley, "Optimised KD-trees for fast image descriptor matching," in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[43] T. Liu, A. W. Moore, K. Yang, and A. G. Gray, "An investigation of practical approximate nearest neighbor algorithms," in *Advances in neural information processing systems*, 2005, pp. 825–832.

[44] D. Nister and H. Stewenius, "Scalable recognition with a vocabulary tree," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006, vol. 2, pp. 2161–2168.

[45] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration.," *VISAPP (1)*, vol. 2, no. 331–340, p. 2, 2009.

[46] C. Arth, D. Wagner, M. Klopschitz, A. Irschara, and D. Schmalstieg, "Wide area localization on mobile phones," in *2009 8th ieee international symposium on mixed and augmented reality*, 2009, pp. 73–82.

[47] Y. Li, N. Snavely, and D. P. Huttenlocher, "Location recognition using prioritized feature matching," in *European Conference on Computer Vision*, 2010, pp. 791–804.

[48] T. Sattler, B. Leibe, and L. Kobbelt, "Fast image-based localization using direct 2d-to-3d matching," in *2011 International Conference on Computer Vision*, 2011, pp. 667–674.

[49] T. Sattler, B. Leibe, and L. Kobbelt, "Efficient & effective prioritized matching for large-scale image-based localization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 9, pp. 1744–1756, 2017.

[50] A. Irschara, C. Zach, J.-M. Frahm, and H. Bischof, "From structure-from-motion point clouds to fast location recognition," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 2009, pp. 2599–2606.

[51] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua, "Worldwide pose estimation using 3d point clouds," in *European conference on computer vision*, 2012, pp. 15–29.

[52] T. Sattler, M. Havlena, F. Radenovic, K. Schindler, and M. Pollefeys, "Hyperpoints and fine vocabularies for large-scale location recognition," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2102–2110.

[53] L. Liu, H. Li, and Y. Dai, "Efficient global 2d-3d matching for camera localization in a large-scale 3d map," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2372–2381.

[54] Y. Feng, L. Fan, and Y. Wu, "Fast localization in large-scale environments using supervised indexing of binary features," *IEEE Trans. Image Process.*, vol. 25, no. 1, pp. 343–358, 2016.

[55] M. Donoser and D. Schmalstieg, "Discriminative feature-to-point matching in image-based localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 516–523.

[56] L. Svärm, O. Enqvist, F. Kahl, and M. Oskarsson, "City-scale localization for cameras with known

vertical direction," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 7, pp. 1455–1461, 2017.

[57] B. Zeisl, T. Sattler, and M. Pollefeys, "Camera pose voting for large-scale image-based localization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2704–2712.

[58] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vis.*, vol. 60, no. 2, pp. 91–110, 2004.

[59] T. Sattler, B. Leibe, and L. Kobbelt, "Improving image-based localization by active correspondence search," in *European conference on computer vision*, 2012, pp. 752–765.

[60] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012, doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[61] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1–9.

[62] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," in *International Conference on Learning Representations*, 2015.

[63] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[64] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.

[65] A. Kendall, M. Grimes, and R. Cipolla, "PoseNet: A convolutional network for real-time 6-DOF camera relocalization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2938–2946.

[66] A. Kendall and R. Cipolla, "Modelling Uncertainty in Deep Learning for Camera Relocalization," *Proc. Int. Conf. Robot. Autom.*, 2016.

[67] M. Cai, C. Shen, and I. D. Reid, "A Hybrid Probabilistic Model for Camera Relocalization," in *BMVC*, 2018.

[68] J. Hensman, N. Fusi, and N. D. Lawrence, "Gaussian Processes for Big Data," in *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence*, 2013, pp. 282–290.

[69] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer School on Machine Learning*, 2003, pp. 63–71.

[70] F. Walch, C. Hazirbas, L. Leal-Taixe, T. Sattler, S. Hilsenbeck, and D. Cremers, "Image-Based Localization Using LSTMs for Structured Feature Correlation," in *The IEEE International Conference on Computer Vision (ICCV)*, 2017.

[71] R. Clark, S. Wang, A. Markham, N. Trigoni, and H. Wen, "VidLoc: A Deep Spatio-Temporal Model for 6-DoF Video-Clip Relocalization," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[72] I. Melekhov, J. Ylioinas, J. Kannala, and E. Rahtu, "Image-based localization using hourglass networks," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 879–886.

[73] A. Kacete, T. Wentz, and J. Royan, "[POSTER] Decision Forest For Efficient and Robust Camera Relocalization," in *Mixed and Augmented Reality (ISMAR-Adjunct), 2017 IEEE International Symposium on*, 2017, pp. 20–24.

[74] T. Naseer and W. Burgard, "Deep regression for monocular camera-based 6-dof global localization in

outdoor environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 1525–1530.

[75] J. Wu, L. Ma, and X. Hu, "Delving deeper into convolutional neural networks for camera relocalization," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5644–5651.

[76] A. Kendall and R. Cipolla, "Geometric loss functions for camera pose regression with deep learning," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017.

[77] A. Valada, N. Radwan, and W. Burgard, "Deep auxiliary learning for visual localization and odometry," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6939–6946.

[78] N. Radwan, A. Valada, and W. Burgard, "Vlocnet++: Deep multitask learning for semantic visual localization and odometry," *IEEE Robot. Autom. Lett.*, vol. 3, no. 4, pp. 4407–4414, 2018.

[79] R. Li, S. Wang, Z. Long, and D. Gu, "Undeepvo: Monocular visual odometry through unsupervised deep learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7286–7291.

[80] S. Brahmbhatt, J. Gu, K. Kim, J. Hays, and J. Kautz, "Geometry-aware learning of maps for camera localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2616–2625.

[81] J. Hays and A. A. Efros, "IM2GPS: estimating geographic information from a single image," in *2008 ieee conference on computer vision and pattern recognition*, 2008, pp. 1–8.

[82] G. Klein and D. Murray, "Improving the agility of keyframe-based SLAM," in *European Conference on Computer Vision*, 2008, pp. 802–815.

[83] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *Int. J. Comput. Vis.*, vol. 42, no. 3, pp. 145–175, 2001.

[84] B. Glocker, J. Shotton, A. Criminisi, and S. Izadi, "Real-time rgb-d camera relocalization via randomized ferns for keyframe encoding," *IEEE Trans. Vis. Comput. Graph.*, vol. 21, no. 5, pp. 571–583, 2015.

[85] R. Arandjelovic, P. Gronat, A. Torii, T. Pajdla, and J. Sivic, "NetVLAD: CNN architecture for weakly supervised place recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5297–5307.

[86] A. Gordo, J. Almazan, J. Revaud, and D. Larlus, "End-to-end learning of deep visual representations for image retrieval," *Int. J. Comput. Vis.*, vol. 124, no. 2, pp. 237–254, 2017.

[87] H. J. Kim, E. Dunn, and J.-M. Frahm, "Learned contextual feature reweighting for image geo-localization," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 3251–3260.

[88] V. Balntas, S. Li, and V. Prisacariu, "Relocnet: Continuous metric learning relocalisation using neural nets," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 751–767.

[89] Z. Laskar, I. Melekhov, S. Kalia, and J. Kannala, "Camera relocalization by computing pairwise relative poses using convolutional neural network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 929–938.

[90] H. Taira *et al.*, "InLoc: Indoor visual localization with dense matching and view synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7199–7209.

[91] Y. Gong, L. Wang, R. Guo, and S. Lazebnik, "Multi-scale orderless pooling of deep convolutional

activation features," in *European conference on computer vision*, 2014, pp. 392–407.

[92] A. Sharif Razavian, J. Sullivan, A. Maki, and S. Carlsson, "A baseline for visual instance retrieval with deep convolutional networks," in *International Conference on Learning Representations, May 7-9, 2015, San Diego, CA*, 2015.

[93] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," in *null*, 2003, p. 1470.

[94] M. Cummins and P. Newman, "Appearance-only SLAM at large scale with FAB-MAP 2.0," *Int. J. Rob. Res.*, vol. 30, no. 9, pp. 1100–1123, 2011.

[95] D. Gálvez-López and J. D. Tardós, "Bags of Binary Words for Fast Place Recognition in Image Sequences," *IEEE Trans. Robot.*, vol. 28, no. 5, pp. 1188–1197, Oct. 2012, doi: 10.1109/TRO.2012.2197158.

[96] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *European conference on computer vision*, 2010, pp. 778–792.

[97] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European conference on computer vision*, 2006, pp. 430–443.

[98] R. Mur-Artal and J. D. Tardós, "Fast relocalisation and loop closing in keyframe-based SLAM," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 846–853.

[99] F. Perronnin, J. Sánchez, and T. Mensink, "Improving the Fisher kernel for large-scale image classification," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6314 LNCS, no. PART 4, pp. 143–156, 2010, doi: 10.1007/978-3-642-15561-1_11.

[100] H. Jegou, F. Perronnin, M. Douze, J. Sánchez, P. Perez, and C. Schmid, "Aggregating local image descriptors into compact codes," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 9, pp. 1704–1716, 2012.

[101] H. Jégou, M. Douze, C. Schmid, and P. Pérez, "Aggregating local descriptors into a compact image representation," in *CVPR 2010-23rd IEEE Conference on Computer Vision & Pattern Recognition*, 2010, pp. 3304–3311.

[102] R. Arandjelovic and A. Zisserman, "All about VLAD," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2013, pp. 1578–1585.

[103] H. Jin Kim, E. Dunn, and J.-M. Frahm, "Predicting good features for image geo-localization using per-bundle vlad," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1170–1178.

[104] A. Torii, R. Arandjelovic, J. Sivic, M. Okutomi, and T. Pajdla, "24/7 place recognition by view synthesis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1808–1817.

[105] A. Babenko and V. Lempitsky, "Aggregating Local Deep Features for Image Retrieval," in *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[106] F. Perronnin, Y. Liu, J. Sánchez, and H. Poirier, "Large-scale image retrieval with compressed fisher vectors," in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 3384–3391.

[107] A. Gordoa, J. A. Rodr\'\iguez-Serrano, F. Perronnin, and E. Valveny, "Leveraging category-level labels for instance-level image retrieval," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3045–3052.

[108] T. Weyand, I. Kostrikov, and J. Philbin, “Planet-photo geolocation with convolutional neural networks,” in *European Conference on Computer Vision*, 2016, pp. 37–55.

[109] R. Mur-Artal and J. D. Tardós, “Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras,” *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, 2017.

[110] S. Rusinkiewicz and M. Levoy, “Efficient variants of the ICP algorithm.,” in *3dim*, 2001, vol. 1, pp. 145–152.

[111] J. Gall and V. Lempitsky, “Class-specific hough forests for object detection,” in *Decision forests for computer vision and medical image analysis*, Springer, 2013, pp. 143–157.

[112] J. Shotton, B. Glocker, C. Zach, S. Izadi, A. Criminisi, and A. Fitzgibbon, “Scene coordinate regression forests for camera relocalization in RGB-D images,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 2930–2937.

[113] V. Lepetit and P. Fua, “Keypoint recognition using randomized trees,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 9, pp. 1465–1479, 2006.

[114] A. Guzman-Rivera *et al.*, “Multi-output learning for camera relocalization,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1114–1121.

[115] J. Valentin, M. Nießner, J. Shotton, A. Fitzgibbon, S. Izadi, and P. H. S. Torr, “Exploiting uncertainty in regression forests for accurate camera relocalization,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4400–4408.

[116] E. Brachmann, F. Michel, A. Krull, M. Ying Yang, S. Gumhold, and others, “Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 3364–3372.

[117] L. Meng, J. Chen, F. Tung, J. J. Little, and C. W. de Silva, “Exploiting Random RGB and Sparse Features for Camera Pose Estimation.,” in *BMVC*, 2016.

[118] D. Massiceti, A. Krull, E. Brachmann, C. Rother, and P. H. S. Torr, “Random forests versus Neural Networks—What’s best for camera localization?,” in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, 2017, pp. 5118–5125.

[119] L. Meng, J. Chen, F. Tung, J. Little J., J. Valentin, and C. Silva, “Backtracking Regression Forests for Accurate Camera Relocalization,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2017)*, 2017.

[120] L. Meng, F. Tung, J. J. Little, J. Valentin, and C. W. de Silva, “Exploiting Points and Lines in Regression Forests for RGB-D Camera Relocalization,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 6827–6834.

[121] T. Cavallari, S. Golodetz, N. A. Lord, J. Valentin, L. Di Stefano, and P. H. S. Torr, “On-The-Fly Adaptation of Regression Forests for Online Camera Relocalisation,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[122] N.-D. Duong, A. Kacete, C. Soladie, P.-Y. Richard, and J. Royan, “Accurate Sparse Feature Regression Forest Learning for Real-Time Camera Relocalization,” in *2018 International Conference on 3D Vision (3DV)*, 2018, pp. 643–652.

[123] N.-D. Duong, A. Kacete, C. Soladie, P.-Y. Richard, and J. Royan, “DynaLoc: Real-Time Camera Relocalization from a Single RGB Image in Dynamic Scenes based on an Adaptive Regression Forest,” in *15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2020*, 2020.

[124] E. Brachmann *et al.*, "DSAC - Differentiable RANSAC for Camera Localization," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[125] N.-D. Duong, A. Kacete, C. Sodalie, P.-Y. Richard, and J. Royan, "xyzNet: Towards Machine Learning Camera Relocalization by Using a Scene Coordinate Prediction Network," in *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, 2019, pp. 258–263.

[126] X. Li, J. Ylioinas, and J. Kannala, "Full-Frame Scene Coordinate Regression for Image-Based Localization," in *RSS*, 2018.

[127] M. Bui, S. Albarqouni, S. Ilic, and N. Navab, "Scene Coordinate and Correspondence Learning for Image-Based Localization," in *BMVC*, 2018, p. 3.

[128] E. Brachmann and C. Rother, "Learning less is more-6d camera localization via 3d surface regression," in *Proc. CVPR*, 2018, vol. 8.

[129] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[130] X. Li, J. Ylioinas, J. Verbeek, and J. Kannala, "Scene Coordinate Regression with Angle-Based Reprojection Loss for Camera Relocalization," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, p. 0.

[131] "Azure Kinect DK – Develop AI Models: Microsoft Azure," – *Develop AI Models | Microsoft Azure*. Microsoft.

[132] "Intel® RealSense™ Depth and Tracking Cameras," *Intel® RealSense™ Depth and Tracking Cameras*. Intel.

[133] "Leica BLK360," *Leica Geosystems*. .

[134] "FARO Focus 3D Laser Scanner," *FARO*. FARO Technologies, Inc.

[135] R. Chen, S. Han, J. Xu, and H. Su, "Point-based multi-view stereo network," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 1538–1547.

[136] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, "Cascade Cost Volume for High-Resolution Multi-View Stereo and Stereo Matching," *arXiv Prepr. arXiv1912.06378*, 2019.

[137] P. Labatut, J.-P. Pons, and R. Keriven, "Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts," in *2007 IEEE 11th international conference on computer vision*, 2007, pp. 1–8.

[138] H. Zhu, F. Meng, J. Cai, and S. Lu, "Beyond pixels: A comprehensive survey from bottom-up to semantic image segmentation and cosegmentation," *J. Vis. Commun. Image Represent.*, vol. 34, pp. 12–27, 2016.

[139] A. Rosinol, A. Gupta, M. Abate, J. Shi, and L. Carlone, "3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans," *arXiv Prepr. arXiv2002.06289*, 2020.

[140] F. J. Lawin, P.-E. Forssén, and H. Ovrén, "Efficient multi-frequency phase unwrapping using kernel density estimation," in *European Conference on Computer Vision*, 2016, pp. 170–185.

[141] "Microsoft HoloLens: Mixed Reality Technology for Business," *Microsoft HoloLens | Mixed Reality Technology for Business*. .

[142] M. Hansard, S. Lee, O. Choi, and R. P. Horaud, *Time-of-flight cameras: principles, methods and applications*. Springer Science & Business Media, 2012.

[143] C. H. Esteban and F. Schmitt, "Silhouette and stereo fusion for 3D object modeling," *Comput. Vis.*

Image Underst., vol. 96, no. 3, pp. 367–392, 2004.

[144] G. Vogiatzis, C. H. Esteban, P. H. S. Torr, and R. Cipolla, "Multiview stereo via volumetric graph-cuts and occlusion robust photo-consistency," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 12, pp. 2241–2246, 2007.

[145] D. Pagliari, F. Menna, R. Roncella, F. Remondino, and L. Pinto, "Kinect Fusion improvement using depth camera calibration," *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. 40, no. 5, p. 479, 2014.

[146] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang, "Deepmvs: Learning multi-view stereopsis," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2821–2830.

[147] M. Ji, J. Gall, H. Zheng, Y. Liu, and L. Fang, "Surfacenet: An end-to-end 3d neural network for multiview stereopsis," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2307–2315.

[148] F. Tombari, S. Salti, and L. Di Stefano, "A combined texture-shape descriptor for enhanced 3D feature matching," in *2011 18th IEEE international conference on image processing*, 2011, pp. 809–812.

[149] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," *ACM Trans. Graph.*, vol. 38, no. 5, pp. 1–12, 2019.

[150] Y. Verdie, K. Yi, P. Fua, and V. Lepetit, "Tilde: A temporally invariant learned detector," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5279–5288.

[151] Y. Furukawa, C. Hernández, and others, "Multi-view stereo: A tutorial," *Found. Trends®in Comput. Graph. Vis.*, vol. 9, no. 1–2, pp. 1–148, 2015.

[152] T. Schöps, V. Larsson, M. Pollefeys, and T. Sattler, "Why Having 10,000 Parameters in Your Camera Model is Better Than Twelve," *arXiv Prepr. arXiv1912.02908*, 2019.

[153] Z. Kukelova, M. Bujnak, and T. Pajdla, "Real-time solution to the absolute pose problem with unknown radial distortion and focal length," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 2816–2823.

[154] Z. Kukelova, J. Heller, and A. Fitzgibbon, "Efficient intersection of three quadrics and applications in computer vision," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 1799–1808.

[155] C. Albl, Z. Kukelova, and T. Pajdla, "R6p-rolling shutter absolute camera pose," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 2292–2300.

[156] C. Albl, A. Sugimoto, and T. Pajdla, "Degeneracies in rolling shutter sfm," in *European Conference on Computer Vision*, 2016, pp. 36–51.

[157] R. Zhang, P.-S. Tsai, J. E. Cryer, and M. Shah, "Shape-from-shading: a survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 8, pp. 690–706, 1999.

[158] D. Yang and J. Deng, "Shape from shading through shape evolution," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3781–3790.

[159] C. Godard, O. Mac Aodha, and G. J. Brostow, "Unsupervised monocular depth estimation with left-right consistency," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 270–279.

[160] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, "Deep ordinal regression network for monocular depth estimation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern*

Recognition, 2018, pp. 2002–2011.

[161] D. Scharstein, T. Taniai, and S. N. Sinha, “Semi-global stereo matching with surface orientation priors,” in *2017 International Conference on 3D Vision (3DV)*, 2017, pp. 215–224.

[162] R. Zabih and J. Woodfill, “Non-parametric local transforms for computing visual correspondence,” in *European conference on computer vision*, 1994, pp. 151–158.

[163] P. Viola and W. M. Wells III, “Alignment by maximization of mutual information,” *Int. J. Comput. Vis.*, vol. 24, no. 2, pp. 137–154, 1997.

[164] H. Hirschmuller and D. Scharstein, “Evaluation of stereo matching costs on images with radiometric differences,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 31, no. 9, pp. 1582–1599, 2008.

[165] A. Spyropoulos, N. Komodakis, and P. Mordohai, “Learning to detect ground control points for improving the accuracy of stereo matching,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1621–1628.

[166] R. Yang and M. Pollefeys, “Multi-resolution real-time stereo on commodity graphics hardware,” in *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.*, 2003, vol. 1, pp. I--I.

[167] A. Hornung and L. Kobbelt, “Robust and efficient photo-consistency estimation for volumetric 3d reconstruction,” in *European Conference on Computer Vision*, 2006, pp. 179–190.

[168] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys, “Real-time plane-sweeping stereo with multiple sweeping directions,” in *2007 IEEE Conference on Computer Vision and Pattern Recognition*, 2007, pp. 1–8.

[169] X. Mei, X. Sun, M. Zhou, S. Jiao, H. Wang, and X. Zhang, “On building an accurate stereo matching system on graphics hardware,” in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011, pp. 467–474.

[170] X. Han, T. Leung, Y. Jia, R. Sukthankar, and A. C. Berg, “Matchnet: Unifying feature and metric learning for patch-based matching,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 3279–3286.

[171] A. Seki and M. Pollefeys, “Patch Based Confidence Prediction for Dense Disparity Map.,” in *BMVC*, 2016, vol. 2, no. 3, p. 4.

[172] M. Poggi, F. Tosi, and S. Mattoccia, “Quantitative evaluation of confidence measures in a machine learning world,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5228–5237.

[173] P. Knobelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock, “End-to-end training of hybrid CNN-CRF models for stereo,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2339–2348.

[174] F. Tosi, M. Poggi, A. Benincasa, and S. Mattoccia, “Beyond local reasoning for stereo confidence estimation with deep learning,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 319–334.

[175] A. Kuhn, C. Sormann, M. Rossi, O. Erdler, and F. Fraundorfer, “DeepC-MVS: Deep Confidence Prediction for Multi-View Stereo Reconstruction,” *arXiv Prepr. arXiv1912.00439*, 2019.

[176] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *Int. J. Comput. Vis.*, vol. 47, no. 1–3, pp. 7–42, 2002.

[177] V. Kolmogorov and R. Zabin, “What energy functions can be minimized via graph cuts?,” *IEEE Trans.*

Pattern Anal. Mach. Intell., vol. 26, no. 2, pp. 147–159, 2004.

[178] K.-J. Yoon and I. S. Kweon, "Adaptive support-weight approach for correspondence search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 4, pp. 650–656, 2006.

[179] C. Richardt, D. Orr, I. Davies, A. Criminisi, and N. A. Dodgson, "Real-time spatiotemporal stereo matching using the dual-cross-bilateral grid," in *European conference on Computer vision*, 2010, pp. 510–523.

[180] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *Sixth international conference on computer vision (IEEE Cat. No. 98CH36271)*, 1998, pp. 839–846.

[181] A. Hosni, C. Rhemann, M. Bleyer, C. Rother, and M. Gelautz, "Fast cost-volume filtering for visual correspondence and beyond," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 2, pp. 504–511, 2012.

[182] Z. Ma, K. He, Y. Wei, J. Sun, and E. Wu, "Constant time weighted median filtering for stereo matching and beyond," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 49–56.

[183] S. M. Seitz and C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *Int. J. Comput. Vis.*, vol. 35, no. 2, pp. 151–173, 1999.

[184] K. N. Kutulakos and S. M. Seitz, "A theory of shape by space carving," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999, vol. 1, pp. 307–314.

[185] M. Goesele, N. Snavely, B. Curless, H. Hoppe, and S. M. Seitz, "Multi-view stereo for community photo collections," in *2007 IEEE 11th International Conference on Computer Vision*, 2007, pp. 1–8.

[186] Y. Furukawa and J. Ponce, "Accurate, dense, and robust multiview stereopsis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 8, pp. 1362–1376, 2009.

[187] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski, "Towards internet-scale multi-view stereo," in *2010 IEEE computer society conference on computer vision and pattern recognition*, 2010, pp. 1434–1441.

[188] O. Faugeras and R. Keriven, *Variational principles, surface evolution, PDE's, level set methods and the stereo problem*. IEEE, 2002.

[189] V. Gool and others, "Dense matching of multiple wide-baseline views," in *Proceedings Ninth IEEE International Conference on Computer Vision*, 2003, pp. 1194–1201.

[190] J.-P. Pons, R. Keriven, and O. Faugeras, "Multi-view stereo reconstruction and scene flow estimation with a global image-based matching score," *Int. J. Comput. Vis.*, vol. 72, no. 2, pp. 179–193, 2007.

[191] H.-H. Vu, P. Labatut, J.-P. Pons, and R. Keriven, "High accuracy and visibility-consistent dense multiview stereo," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 5, pp. 889–901, 2011.

[192] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 1996, pp. 303–312.

[193] C. Kim, H. Zimmer, Y. Pritch, A. Sorkine-Hornung, and M. H. Gross, "Scene reconstruction from high spatio-angular resolution light fields.," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 71–73, 2013.

[194] A. Hornung and L. Kobbelt, "Interactive pixel-accurate free viewpoint rendering from images with silhouette aware sampling," in *Computer Graphics Forum*, 2009, vol. 28, no. 8, pp. 2090–2103.

[195] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "Mvsnet: Depth inference for unstructured multi-view stereo," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 767–783.

[196] Y. Yao, Z. Luo, S. Li, T. Shen, T. Fang, and L. Quan, "Recurrent mvsnet for high-resolution multi-view stereo depth inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 5525–5534.

[197] "Maps," *Apple*. .

[198] "No Title." .

[199] "Bing Maps," *Bing*. Microsoft.

[200] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Stat.*, vol. 33, no. 3, pp. 1065–1076, 1962.

[201] M. Goesele, B. Curless, and S. M. Seitz, "Multi-view stereo revisited," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, 2006, vol. 2, pp. 2402–2409.

[202] X. Mei, X. Sun, W. Dong, H. Wang, and X. Zhang, "Segment-tree based cost aggregation for stereo matching," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 313–320.

[203] Q. Yang, "A non-local cost aggregation method for stereo matching," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 1402–1409.

[204] F. Tosi, M. Poggi, and S. Mattoccia, "Leveraging confident points for accurate depth refinement on embedded systems," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2019, p. 0.

[205] V. Kolmogorov and R. Zabih, "Computing visual correspondence with occlusions using graph cuts," in *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, 2001, vol. 2, pp. 508–515.

[206] J. Kappes *et al.*, "A comparative study of modern inference techniques for discrete energy minimization problems," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 1328–1335.

[207] R. Szeliski *et al.*, "A comparative study of energy minimization methods for markov random fields with smoothness-based priors," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 6, pp. 1068–1080, 2008.

[208] Y. Boykov and V. Kolmogorov, "An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 26, no. 9, pp. 1124–1137, 2004.

[209] Y. Boykov, O. Veksler, and R. Zabih, "Fast approximate energy minimization via graph cuts," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 11, pp. 1222–1239, 2001.

[210] N. D. F. Campbell, G. Vogiatzis, C. Hernández, and R. Cipolla, "Using multiple hypotheses to improve depth-maps for multi-view stereo," in *European Conference on Computer Vision*, 2008, pp. 766–779.

[211] M. Lhuillier and L. Quan, "A quasi-dense approach to surface reconstruction from uncalibrated images," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 3, pp. 418–433, 2005.

[212] A. Owens, J. Xiao, A. Torralba, and W. Freeman, "Shape anchors for data-driven multi-view reconstruction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 33–40.

[213] G. Riegler, A. Osman Ulusoy, and A. Geiger, "Octnet: Learning deep 3d representations at high resolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 3577–3586.

[214] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong, "O-cnn: Octree-based convolutional neural networks for 3d shape analysis," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–11, 2017.

[215] A. Kar, C. Häne, and J. Malik, "Learning a multi-view stereo machine," in *Advances in neural information processing systems*, 2017, pp. 365–376.

[216] Y. Hou, J. Kannala, and A. Solin, "Multi-View Stereo by Temporal Nonparametric Fusion," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 2651–2660.

[217] A. Romanoni and M. Matteucci, "Tapa-mvs: Textureless-aware patchmatch multi-view stereo," in *Proceedings of the IEEE International Conference on Computer Vision*, 2019, pp. 10413–10422.

[218] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, "Tanks and temples: Benchmarking large-scale scene reconstruction," *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–13, 2017.

[219] "Zed 2," *STEREOLABS*. .

[220] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald, "Real-time large-scale dense RGB-D SLAM with volumetric fusion," *Int. J. Rob. Res.*, vol. 34, no. 4–5, pp. 598–626, 2015.

[221] openMVG, "openMVG," *GitHub*. 2020.

[222] C. Kerl, J. Sturm, and D. Cremers, "Dense visual SLAM for RGB-D cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 2100–2106.

[223] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Auton. Robots*, vol. 34, no. 3, pp. 189–206, 2013.