AR Cloud: Towards Collaborative Augmented Reality at a Large-Scale

Nam-Duong Duong[*] IRT b-com
Christophe Cutullic[†] Orange
Jean-Marie Henaff[‡] IRT b-com
Jérôme Royan[§] IRT b-com

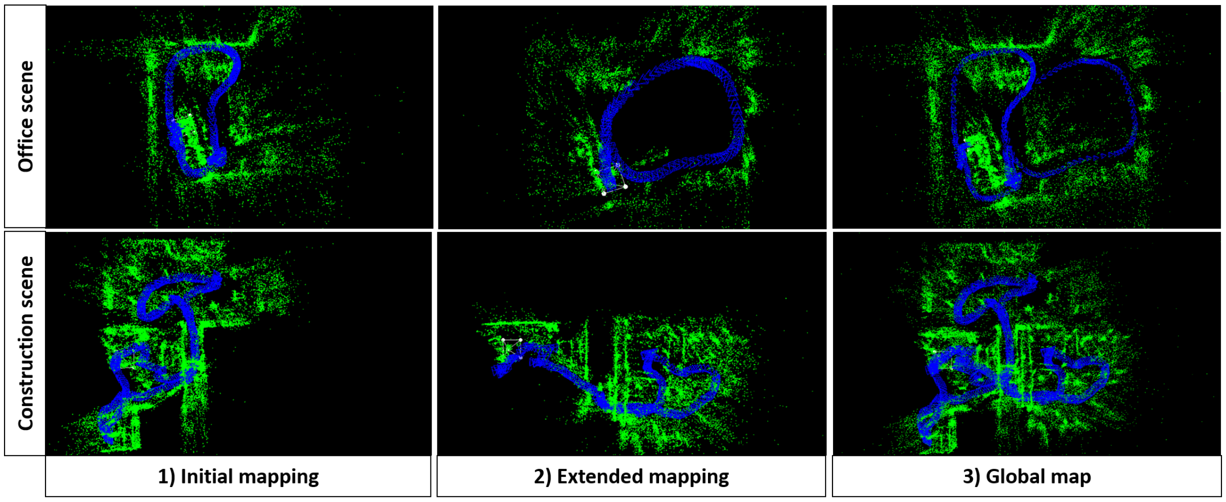

Figure 1: Real-time crowd mapping to create a global map at an office context and a construction site context.

Abstract

Augmented Reality (AR) appears as one of the most promising technologies for the next decades with a lot of applications in Industry 4.0. Unfortunately, most of existing systems are limited to small environments and running on a single device. To cope with these limitations, we present in this paper an AR Cloud solution for real-time crowd mapping. We propose an AR cloud service architecture that is able to 1) continuously relocalize AR devices in relation to the reference coordinate system of a common global 3D map in order to estimate the initial pose of the AR device and to correct tracking drift as soon as possible; 2) locally map in 3D the real environment surrounding each AR device; 3) fuse these 3D local maps with the global 3D map in order to keep it up-to-date, mandatory feature when real environment changes over time. Finally, we show experiments of our AR cloud platform at a large scale.

Index Terms: Augmented Reality—Spatial Computing; Crowd-Mapping—Map Update

1 Introduction

Current back-end of most AR headsets and autonomous agents (robots/cars/drones) implements a Simultaneous Localization And Mapping (SLAM) which aims to jointly estimate the position of an agent (localization) together with a representation of the scene geometry (mapping). This is one of the fundamental problems in computer vision, and there has been tremendous progress in the last two decades. Most existing commercial AR systems (e.g. ARKit, ARCore, or Hololens SDK) provide a highly accurate pose estimation based on a SLAM which performs a robust fusion in real-time of measurements coming from multiple inputs, such as cameras, odometry, depth and inertial sensors. Simultaneously, these AR systems are able to create local maps corresponding to the surrounding area observed by the vision sensors embedded in AR devices. For multi-user AR, the user should also know the state of other users and can be positioned in a unified coordinate system such that 3D virtual contents can be seamlessly displayed at different individual devices. Therefore, the local map created by a user can be shared with other users to offer a collaborative AR experience. However, this local map is limited to small-scale and is not updated in real-time with observations captured by all AR devices.

Indeed, some collaborative SLAM [3, 28] in large environment have been implemented to enable multi-agent AR applications. Nevertheless, local changes (i.e. moving objects) are not detected and updated over time to the global map. Several methods can exploit a semantic knowledge of the scene to deal with this problem and improve the quality of the map [2, 14]. However, these methods mainly focus on the reconstruction from a single device.

To overcome above mentioned limitations, we present in this paper an AR cloud platform to offer shared AR experiences anywhere, at anytime in real environment changing over time. By taking advantage of 5G bandwidth and low latency, most of processing is deployed in the cloud or at its edge. In our AR cloud platform, a global map is shared with all AR devices for allowing them to relocalize themselves in the relation to a common global coordinate system, while this global map is also maintained up-to-date based on local maps created in real-time by each AR device. We summarize our contributions to this paper as follows: 1) The SolAR framework, an open source AR cloud platform, allowing rapid creation and deployment to the cloud of spatial computing services dedicated to AR; 2) An AR cloud service architecture for real-time crowd mapping, which consists of three main services: relocalization service, mapping service and map update service; 3) A robust relocalization based on a submap extracted from the global map in order to accelerate the localization of AR devices in relation to the common coordinate system; 4) A real-time local mapping for each AR device

[*]e-mail: nam-duong.duong@b-com.com
[†]e-mail: christophe.cutullic@orange.com
[‡]e-mail: jean-marie.henaff@b-com.com
[§]e-mail: jerome.royan@b-com.com

with drift correction based on the continuous relocalization in relation to the global map, that aims to detect and correct a tracking drift of camera pose immediately; 5) A map update service consisting of merging local maps created by multi-users into the global map and updating the global map according to changes captured by local maps.

2 Related work

Currently, the visual SLAM methods, aided with multi-camera [10, 21] and inertial measurement unit (IMU) sensors [22, 25], have developed rapidly to make a reliable method that achieves highly accurate camera pose estimation and overcomes limitations of the monocular visual SLAM such as pure rotation, texture-less scenes, or lack of the scale information of scenes. This solution has been successfully applied to several commercial products such as Hololens headset to offer individual AR experience.

However, collaborative AR applications request camera pose estimation of all AR devices relative to the common coordinate system in order to enable sharing AR experiences for multi-user. Some simple solutions build an initial map which is shared to other AR devices. This map can be created by running an online SLAM method or an offline Structure From Motion (SFM) [24] on a set of collected images to achieve higher quality. Instead of performing SFM on a single device as in [19], a distributed SFM proposed by [23] applies a divide and conquer strategy to accelerate the map creation by running multiple SFM processes on a CPU server to build a set of submaps corresponding to each subset of images. Even so, the pre-built global map of these methods cannot be updated anymore, reducing its sustainability for environments changing over time, which is relatively common in the real world.

Recently, multi-agent SLAM is a key solution to provide collaborative tracking and mapping tasks based on a distributed computing system that consists of decentralized and centralized architectures. The former creates a local map for each agent instead of maintaining a common global map and allows agents to exchange their summary map to each other [6] in order to avoid double-counting measurements [5]. Another approach [4] performs decentralised mapping using object-based maps enabling co-localization of participating robots and presents a collaborative pose-graph optimization. Even though the decentralized approaches are robust to unreliable network connections, they are much more difficult to deploy than the centralized one. Indeed, due to limited onboard computational resources, they require efficient algorithms to synchronize data between different agents and perform complicated tasks such as bundle adjustment, loop closure detection. Currently, thanks to the development of network systems with low latency and high reliability, most of existing multi-agent SLAM systems adopt a centralized architecture to take advantage of the computing power of central servers. The systems consist of some client-side frontends and one server-side backend. The frontends are usually responsible for the computation of the real-time states of agents that are critical for online applications. The backend takes care of time-consuming tasks concerning optimization, map fusion. Such as in [26], only tracking task based on PTAM [16] is performed on clients, all keyframes are sent to the cloud server in order to compute the local map for each agent, and fuse those maps. However, this approach requires these agents to download regularly the global map from the server for pose estimation and relocalization. Instead, CoSLAM [31] runs both tracking and mapping tasks in the server, and copes with dynamic environments by grouping cameras with scene overlap. In spite of achieving impressive results, the computation power on each agent is totally unused since all processes are fulfilled by the server side. To take advantage of the agent resources, another approach [28] maintains a local map of limited size onboard each agent ensuring basic autonomy of the individual agent. Some other methods [13, 29] exploit RGB-D sensor data for collaborative 3D dense reconstruction. In this paper, instead of rebuilding full visual SLAM, we benefit from the built-in SLAM based on multi-sensor of AR devices developed by ARCore, ARKit, or Hololens SDKs to get directly accurate camera poses. We only implement a crowd mapping in the cloud to create a global map at a large-scale. This allows our system to simply scale to multi-user.

In collaborative AR systems, all agents are localized in the common coordinate system expressed in the global map. Hence, camera relocalization is necessary to initialize localization of new arrival agents as well as to retrieve camera pose after tracking lost. The camera relocalization leverages the global map which is built from known information of a scene in order to infer camera pose from each image independently. The geometric approaches first build a 3D feature point cloud based on SFM [27] or SLAM [21]. The camera pose can be estimated by directly matching 2D features from the query image to 3D point cloud to define 2D–3D point correspondences, and then running a PnP algorithm [17]. To accelerate the feature matching, a Bag-of-Words (BoW) [11] or a learned-based model [1] are used for retrieving the keyframes which have similar appearance features to the query frame. Even so, the matching of local features is unreliable on scenes with repeated patterns. However, these approaches simply scale to large environments. On the other hand, the machine learning approaches learn an end-to-end regression model [15] from whole images labeled with the camera poses. The trained model then directly predicts camera pose from each RGB image. To improve the accuracy, hybrid approaches [7, 8] compute camera pose by combining both the machine learning approach and the geometric approach. The machine learning part is applied to learn and predict 3D positions of 2D features in the world coordinate system, and the geometric part infers the camera pose from these correspondences. However, the learned model must be retrained from scratch when the global map is extended.

Dynamic environments with moving objects have always been a difficult challenge in SLAM. Most existing approaches handle dynamic region features as outliers and only use static background for mapping [2]. Some other works [14, 18] detect, track, and optimize the trajectory of dynamic objects e.g. people, vehicles, in order to build a complete 3D map. [30] can detect any changes by projecting the map feature into the current view, it then removes out-of-date point cloud corresponding to previous positions of moving objects. [9] proposes a dynamic random forest model, that is able to update gradually predictive model in real-time following changes of a scene.

3 AR Cloud platform

The AR Cloud platform has been developed using the SolAR framework [1]. Indeed, SolAR is an open-source framework under Apache V2 license allowing the creation of spatial computing pipelines dedicated to AR. Based on a C++ interface, it provides an API for more than 50 low and high level components widely used in the field of spatial computing. It also provides numerous component implementations from open-source libraries integrated in modules (e.g. OpenCV, g2o, FBoW, Ceres, OpenGV or PCL). The SolAR framework allows the development of spatial computing pipelines by assembling abstract components. Before starting the spatial computing pipelines, the instantiation and configuration of the components are performed at runtime based on a description file defining the implementation choices and configuration parameters. This provides the ability to test different implementations and component configurations without having to rebuild the pipeline. Finally, the SolAR framework provides a tool to automatically generate, for each SolAR abstract interface, a gRPC client and server. Thus, each component can call another one remotely, which provides the communication layer to easily create services from pipelines. These services are then embedded into docker images, and are hosted in docker repositories.

[1]https://solarframework.github.io/

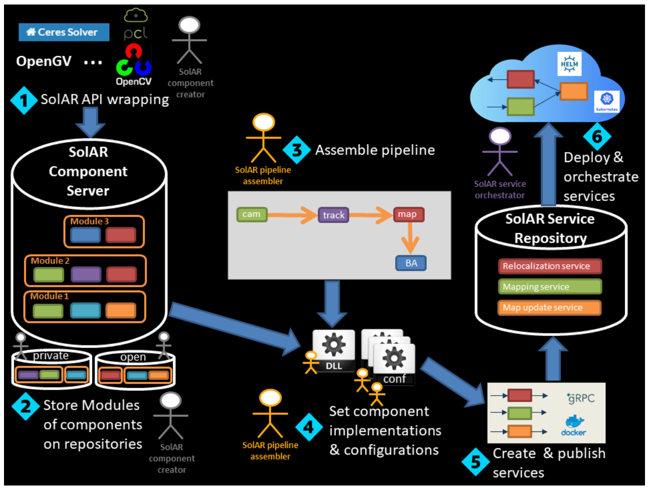

Figure 2: SolAR Framework concept: (1) Create spatial computing components from existing libraries by wrapping SolAR API; (2) Package components into modules and publish them; (3) Assemble and build spatial computing pipelines; (4) Set component implementations and configurations; (5) Create services from pipelines, embed them in docker images and publish them; (6) Deploy services at the edge or in the cloud with Helm Charts.

Kubernetes is used to deploy spatial computing at the edge or in the cloud, the deployment is then managed with helm charts to ease the versioning, sharing, publishing and service upgrades. Fig. 2 presents the concept of SolAR framework.

4 AR Cloud Services Architecture

A SLAM consists of three main components running in parallel, two related to device localization, and one to mapping. First, for localization, two distinct components are needed. The first one, called tracking, estimates the displacement of the augmented reality device at a high frequency ($\sim 60Hz$). Due to error propagation, the pose of the AR device can drift over time during tracking. The second one, called relocalization, estimates the pose of the AR device at a lower frequency ($\sim 1Hz$) by comparing an observation captured by the AR device with the 3D map previously built by the third component (mapping). This relocalization component is used to estimate the first pose of the AR device, when the tracking is lost, and to correct the drift of the tracking. Finally, the third component maps the real environment in 3D for future relocalizations.

Fig. 3 shows the service architecture of our AR Cloud platform. The tracking is provided by existing AR runtimes (e.g., Hololens SDK, ARCore, or ARKit) and runs on the AR device due to very low latency requirements. However, the relocalization and mapping components are embedded into two distinct services and are deployed at the edge or in the cloud. An instantiation of both components is running for each AR device as a context (the map surrounding the AR device) must be preserved over time. Also, to be able to keep up-to-date a common 3D map of the real environment, a third service, called map update, hosts a shared 3D global map of the real environment and is able to update and extend it by fusing when AR session end the local map produced by each mapping service capturing changes in the real environment such as objects that have moved, been added or removed. This map update service is also able to extract a local map surrounding an AR device, subpart of the global map, for each relocalization service instance in order to relocalize a new AR device connected to the AR cloud platform. This local map will be routed to the corresponding mapping service instance to initialize its local map. Finally, a fourth front-end service routes input and output data between the client, the relocalization and the mapping services. Finally, to ease the integration of the AR Cloud platform to AR applications, a Unity plugin has been developed. This plugin, supporting the Hololens SDK and the AR Foundation framework (for ARKit and ARCore), transmits the images captured by cameras embedded in the AR device and the corresponding poses estimated by the AR runtime (tracking), and receives back an offset to apply to the poses estimated by the AR runtime. Thanks to this AR cloud platform, it is now possible to develop AR applications running continuously at a large scale, handling simultaneously dozen of low resources AR devices, and running over time in high dynamic environments through crowd-sourced updates of the 3D map when changes in the real environment are observed. In the next sections, we introduce each service in more details.

5 Relocalization Service

The relocalization service allows AR devices to localize in the common coordinate system defined by the global map, without prior knowledge of its location. Indeed, each AR device sends to the frond-end service images, their corresponding camera poses in the device coordinate system, T_D^C, and intrinsic parameters, K. Then, the relocalization service handles each requested image to estimate its camera pose in the world coordinate system, T_W^C. Based on both of these camera poses, we can define the transformation from the device coordinate system to the world coordinate system, called the pose offset:

$$T_W^D = T_W^C \cdot (T_D^C)^{-1} \tag{1}$$

This pose offset is sent back to the AR device and applied to the pose estimated at a high frequency by the inboard tracking process of the AR runtime before rendering virtual objects in the world coordinate system. To compute T_W^C, we implement a visual relocalization based on fiducial markers or a sparse map. For the first approach, we define in advance a set of fiducial markers such as ArUco [12] or QR code, with knowing their relative transformation to the world coordinate system. Once these markers are detected in a frame, we can find a set of 2D-3D point correspondences of their corners. For the second approach, this service requires a submap, including 3D feature point cloud and keyframes, extracted from the global map, that will be presented in Sect. 7. At first, each query image is used for detecting nearest keyframes based on DBoW2 [11]. The use of a submap instead of the global map allows us to accelerate this step as well as to reduce data transmission time between the relocalization and the map update services. The 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 used for matching. From these matches, a set of 2D-3D point correspondences is established. In both cases, the camera pose is estimated by minimizing the reprojection error based on EPnP algorithm [17]:

$$\varepsilon = \sum_i \|p_i - K T_C^W P_i^w\|^2 \tag{2}$$

Where p_i and P_i^w are respectively coordinates of a 2D keypoint in the image and a corresponding 3D point in the global map.

In addition, each AR device is continuously relocalized in the global map in order to early detect and correct tracking drift that is used for improving the local map created by the mapping service.

6 Mapping Service

Once the relocalization service has successfully determined the pose offset, the front-end service forwards this pose offset, the image and its associated pose from the AR device to the mapping service in order to create a sparse local map for each AR device. Our mapping service is based on ORB-SLAM [21] without the tracking step. The camera pose in the world coordinate system is calculated by using the pose offset T_W^D and the camera pose T_D^C in the device coordinate

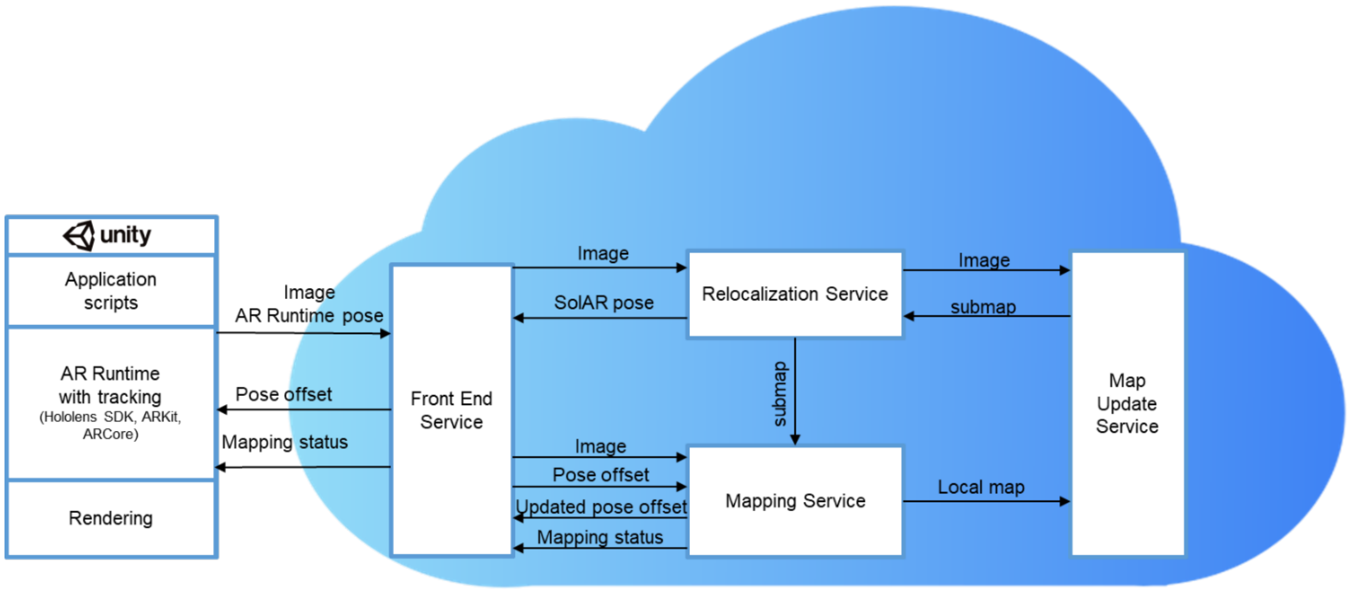

Figure 3: Service architecture of the AR Cloud platform.

system as follows:

$$T_W^C = T_W^D \cdot T_D^C \tag{3}$$

If the relocalization service estimates the pose of the device based on the markers, a 3D sparse map is initialized from scratch. However, if the relocalization service is able to estimate a pose from the submap returned by the map update service, this submap is sent back to the mapping service to initialize the local map. Then this service gradually adds new keyframes and triangulates new 3D points to extend the local map. However, instead of inserting keyframes as fast as possible to make the tracking more robust in ORB-SLAM, we add a new condition comparing the candidate keyframe to neighbor keyframes of the reference keyframe to avoid creating redundant keyframes.

Although AR devices provide highly accurate camera pose, drift error is still accumulated over time. Therefore, similar to [21], we integrate the loop closure detection and correction into our mapping service for the drift correction. Simultaneously, this allows this service to refine the pose offset and to return the updated pose offset to the AR device to improve the virtual objects registration. Moreover, with the goal of detecting and correcting the drift as soon as possible, we take into account a series of the pose offsets estimated by the continuous relocalization. Basically, if no tracking drift is observed, we achieve the similar offsets from Equation 1. However, assuming at the time t, the tracking is drifting by Δ_t, we will obtain a drifted pose $\widetilde{T_D^C} = \Delta_t T_D^C$. This drift transformation can be detected based on two consecutive pose offsets as follows:

$$\Delta_t = (T_W^D)_{t-1} \cdot (T_W^D)_t^{-1} \tag{4}$$

Then, we apply the inverse of this drift transformation to current neighbor keyframes and local point cloud. Finally, we run a local bundle adjustment only on the keyframes and the point cloud created between two times of these pose offsets. This process allows us to immediately correct the effect of this drift error on the mapping in real-time. At the end of the AR session, the mapping service transfers the local maps to the map update service.

7 Map Update Service

The map update service sequentially processes each local map sent by the mapping service, in order to extend and update the global map. This process consists of three steps: the map overlap detection, the map fusion and the map update.

The first step tries to detect common areas between the local map and the global map based on the place recognition algorithm [11] and then find 3D points matching together between the two maps. If the overlap area is sufficient to compute the transformation between these two maps, the local map is transformed to the world coordinate system defined by the global map. However, in our proposed pipeline, each local map is initialized by a submap of the global one and expressed in this common coordinate system. This step is hence skipped.

The second step allows to merge the local map into the global map. Firstly, to avoid duplication of 3D points in the map, given each 3D point of the local map, we try to find candidate 3D points of the global map in a radius around this point using the FLANN library [20] to speed up the search. We then compare their features to verify a correct match. All duplicated points are merged by moving the visibilities of local map points to global map points. Secondly, we merge keyframes and update the covisibility graph. Each keyframe of the local map is respectively added to the global map and is assigned to a new identification. Simultaneously, new nodes corresponding to these keyframes are inserted in the covisibility graph of the global map while retaining weights of edges from the local map. Furthermore, based on the duplicated points identified above, we define new edges between new keyframes and existing keyframes in the global covisibility graph. All BoW features of new keyframes are also put in the global retrieval model.

The last step prunes outdated 3D points to keep the global map up-to-date according to the local map created from recent observations. Firstly, we define a confidence score model for each 3D point, that is calculated by the inverse of the number of consecutive times a 3D point is considered as an outlier, N_{out}^{-1}. Once, this point is determined as an inlier, this score is reset to 1. If this score is less than a threshold τ ($\tau = 0.3$ in our experiments), we remove this 3D point because it is no longer accurate. To determine a 3D point is an outlier or an inlier, inspired by [30], we project respectively every existing 3D point of the global map to each new keyframes, and we evaluate if this 3D point is projected into the keyframe and if the angle formed by its normal vector and the view direction from the keyframe is less than α ($\alpha = 30°$ in our experiments). In this case, if there exists a feature of the keyframe at the projected position matching with the feature of the 3D point, this point is defined as an inlier. Otherwise, it is an outlier. To prune obsolete and redundant keyframes, if a keyframe has the number of common features shared with other keyframes greater than 90% of its total number of map point visibilities, we remove this keyframe. Finally, we run a global bundle adjustment after merging to optimize the whole global map.

In addition, the map update service can extract a submap concerning the active zone of a user to transfer it to the relocalization service. For each query image, we define a set of retrieved keyframes based on [11] and use feature matching verification to find the nearest keyframe. Then we extract a submap including local point cloud and neighbor keyframes of the nearest keyframe.

8 Experiments

Table 1: Mapping results of local and global maps for the two scenes.

	Office scene		Construction scene	
Sequence	Seq-01	Seq-02	Seq-01	Seq-02
#Images	1092	911	2258	1794
#Keyframes	200	165	349	270
#3D points	20447	11707	35760	23629
Global map	355 keyframes 28776 points		598 keyframes 56349 points	

We evaluated our AR Cloud platform in an office scene ($\sim 100m^2$) and a construction site scene ($\sim 200m^2$). For each scene, we captured two data sequences. The first one is to initialize a global map and the second one aims to extend the global map. The services have been deployed on an all-in-one edge infrastructure packaged in a flight-case including the spatial computing server as well as the 5G RAN and Core servers, offering an autonomous solution easy to deploy in various environments. A Microsoft Hololens 2 was used for the experimentation. It was connected to a Wi-Fi/5G gateway to transfer images and camera poses to our services. The origin of the world coordinate system is initialized at the center of the fiducial marker that is attached to each scene, as shown in Fig. 4.

8.1 Robust mapping using the drift correction

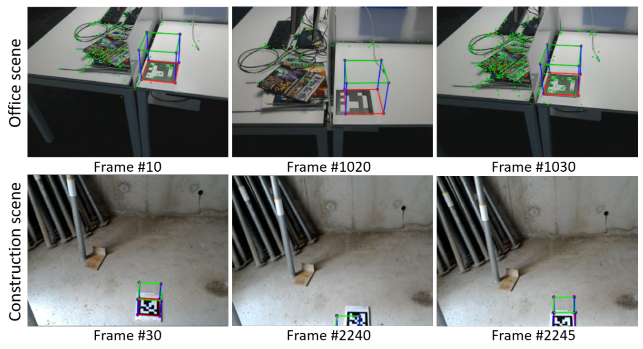

Figure 4: The drift correction for improving tracking and mapping.

For each sequence, as soon as the device is localized in the global coordinate system, the front-end service forwards images, camera poses and the pose offset to the mapping service to create a local map. Details of the number of processed images, 3D points and keyframes for each map are reported in Table 1. The first two columns of Fig. 1 show the visualization of achieved local maps consisting of 3D point cloud (green points) and keyframes (blue pyramids). In the first column of Fig. 4, we also visualize a simple virtual cube at the origin of the world coordinate system to verify the tracking accuracy. Although the Hololens SDK achieves highly accurate camera pose, after a few minutes of experimentation, the drift error is accumulated leading to the virtual cube being shifted about $5cm$ for the office scene and $20cm$ for the construction scene. This is presented in the second column of Fig. 4. However, thanks to our continuous relocalization and loop closure detection, these drift errors are detected and corrected. Indeed, the third column of Fig. 4 shows images with the cube displayed precisely on the marker.

8.2 Map fusion and map update

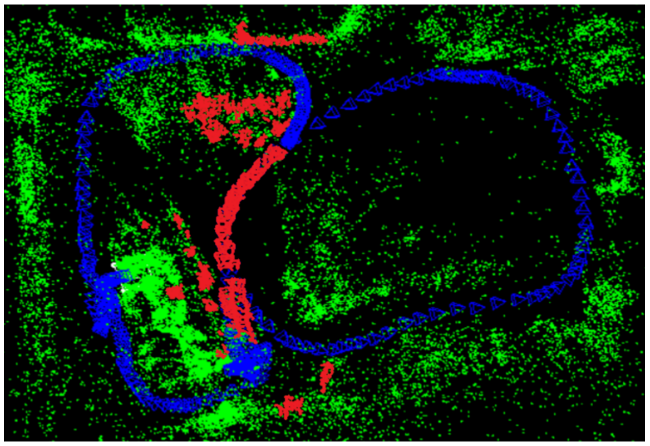

Figure 5: The point cloud and keyframes overlap detection.

For each scene, we started the experiment by resetting the global map from scratch. At the beginning, the first client scans a part of the environment, then this local map is sent to the map update service to initialize the global map. After that, the second client arrives on this scene and requests the relocalization service to be localized in the global coordinate system and to create an extended local map. At the end of the experiment, this second local map is transferred to the map update service for merging and updating the global map. This process is illustrated in Fig. 1. To achieve a compact global map, the map update service rapidly detects 3D points and keyframes overlap (e.g. the red ones in Fig. 5) between the local map and the current global map to merge them. The result of the final global maps is presented in Table 1.

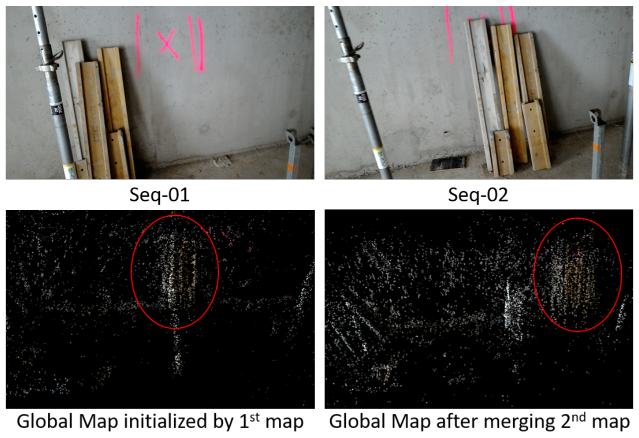
Seq-01
Seq-02
Global Map initialized by 1st map
Global Map after merging 2nd map

Figure 6: Map update. Removing outlier 3D points of moving objects and adding new 3D points corresponding to their new position.

Moreover, the map update service is able to keep the global map up-to-date by taking into account changes of the real environment captured in local maps. For example, in the construction scene, there are some wooden planks being shifted to the right in the second sequence. This change creates 3D points corresponding to new positions of these planks, while the old 3D points are detected as outliers and removed. Fig. 6 shows results of the map update process. The removed and added 3D points are marked by red ellipses.

9 Conclusion

We presented in this paper an AR Cloud platform for offering collaborative AR applications at the large scale on any AR devices. By moving spatial computing services, including relocalization, mapping and map update, at the edge or in the cloud, our AR Cloud platform is able to maintain an up-to-date global 3D map of the real environment. It also supports a crowd-sourced mode, when the 3D map is shared with multiple AR devices for allowing them to localize themselves in real-time based on observations captured by their vision sensors. We will continue the development and experimentation of the AR Cloud platform in order to optimize the localization performance and to address additional AR use cases.

Acknowledgments

Part of this work has been funded by the European Commission through the H2020 project ARtwin (Grant Agreement no. 856994).

References

[1] 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*, pp. 5297–5307, 2016.

[2] B. Bescos, J. M. Fácil, J. Civera, and J. Neira. Dynaslam: Tracking, mapping, and inpainting in dynamic scenes. *IEEE Robotics and Automation Letters*, 3(4):4076–4083, 2018.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós. Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam. *IEEE Transactions on Robotics*, 37(6):1874–1890, 2021.

[4] S. Choudhary, L. Carlone, C. Nieto, J. Rogers, H. I. Christensen, and F. Dellaert. Distributed mapping with privacy and communication constraints: Lightweight algorithms and object-based models. *The International Journal of Robotics Research*, 36(12):1286–1311, 2017.

[5] A. Cunningham, V. Indelman, and F. Dellaert. Ddf-sam 2.0: Consistent distributed smoothing and mapping. In *2013 IEEE international conference on robotics and automation*, pp. 5220–5227. IEEE, 2013.

[6] A. Cunningham, M. Paluri, and F. Dellaert. Ddf-sam: Fully distributed slam using constrained factor graphs. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3025–3030. IEEE, 2010.

[7] 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)*, pp. 258–263. IEEE, 2018.

[8] 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)*, pp. 643–652. IEEE, 2018.

[9] 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.

[10] J. Engel, J. Stückler, and D. Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1935–1942. IEEE, 2015.

[11] D. Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012. doi: 10.1109/TRO.2012.2197158

[12] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[13] S. Golodetz, T. Cavallari, N. A. Lord, V. A. Prisacariu, D. W. Murray, and P. H. Torr. Collaborative large-scale dense 3d reconstruction with online inter-agent pose optimisation. *IEEE transactions on visualization and computer graphics*, 24(11):2895–2905, 2018.

[14] M. Gonzalez, E. Marchand, A. Kacete, and J. Royan. Twistslam: Constrained slam in dynamic environment. *arXiv preprint arXiv:2202.12384*, 2022.

[15] 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*, pp. 2938–2946, 2015.

[16] G. Klein and D. Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pp. 225–234. IEEE, 2007.

[17] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155–166, 2009.

[18] P. Li, T. Qin, et al. Stereo vision-based semantic 3d object and egomotion tracking for autonomous driving. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 646–661, 2018.

[19] S. Lynen, T. Sattler, M. Bosse, J. A. Hesch, M. Pollefeys, and R. Siegwart. Get out of my lab: Large-scale, real-time visual-inertial localization. In *Robotics: Science and Systems*, vol. 1, p. 1, 2015.

[20] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[21] R. Mur-Artal and J. D. Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE transactions on robotics*, 33(5):1255–1262, 2017.

[22] R. Mur-Artal and J. D. Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters*, 2(2):796–803, 2017.

[23] L. Platinsky, M. Szabados, F. Hlasek, R. Hemsley, L. Del Pero, A. Pancik, B. Baum, H. Grimmett, and P. Ondruska. Collaborative augmented reality on smartphones via life-long city-scale maps. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 533–541. IEEE, 2020.

[24] F. Poiesi, A. Locher, P. Chippendale, E. Nocerino, F. Remondino, and L. Van Gool. Cloud-based collaborative 3d reconstruction using smartphones. In *Proceedings of the 14th European Conference on Visual Media Production (CVMP 2017)*, pp. 1–9, 2017.

[25] T. Qin, P. Li, and S. Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Transactions on Robotics*, 34(4):1004–1020, 2018.

[26] L. Riazuelo, J. Civera, and J. M. Montiel. C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.

[27] T. Sattler, B. Leibe, and L. Kobbelt. Efficient & effective prioritized matching for large-scale image-based localization. *IEEE transactions on pattern analysis and machine intelligence*, 39(9):1744–1756, 2016.

[28] P. Schmuck, T. Ziegler, M. Karrer, J. Perraudin, and M. Chli. Covins: Visual-inertial slam for centralized collaboration. In *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pp. 171–176. IEEE, 2021.

[29] P. Stotko, S. Krumpen, M. B. Hullin, M. Weinmann, and R. Klein. Slamcast: Large-scale, real-time 3d reconstruction and streaming for immersive multi-client live telepresence. *IEEE transactions on visualization and computer graphics*, 25(5):2102–2112, 2019.

[30] W. Tan, H. Liu, Z. Dong, G. Zhang, and H. Bao. Robust monocular slam in dynamic environments. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 209–218. IEEE, 2013.

[31] D. Zou and P. Tan. Coslam: Collaborative visual slam in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):354–366, 2012.