Ambarella-To-PatchMatchNet Algorithm

Marco Orsingher

1 Introduction

The Structure-from-Motion (SfM) module produces a (possibly very large) set of calibrated images $\mathcal{I} = \{I_i\}_{i=1}^{N}$, each with sparse features $\mathcal{F}_i = \{F_j\}_{j=1}^{M}$ and poses $\mathcal{P} = \{\mathbf{K}_i, \mathbf{R}_i, \mathbf{t}_i\}_{i=1}^{N}$, as well as a set of sparse 3D points in the world coordinate frame $\mathcal{X} = \{\mathbf{X}_i\}_{i=1}^{P}$. The following Multi-View Stereo (MVS) module aims at estimating a depth map for each frame, in order to merge them in later stages for a full 3D reconstruction of the environment. The implicit assumption here is that the output of SfM is already well verified, meaning that matching features, poses and geometrical relationships can be considered as sufficiently good. A preprocessing step could be adopted in order to filter out 3D points with high reprojection error and features in the images that do not correspond to a a 3D point in the input point cloud.

2 Keyframe Selection

The first step is to select a subset of keyframes $\mathcal{K} \subset \mathcal{I}$ that are representative enough for the 3D reconstruction of the scene, since it is both impractical and often useless to consider every single frame in the video input. Borrowing ideas from visual SLAM literature, the method currently implemented starts from $\mathcal{K} = \{I_0\}$ and adds the current image as a keyframe if it has sufficient spatial distance with respect to the previous keyframe, in order to ensure a good baseline for MVS. There are two possible ways of improving this:

1. Two keyframes should not be redundant in terms of the visual content. This is trickier to quantify and it can be based either on the the number of sparse features detected in the current frame or on distance measures between all the pixels.

2. A keyframe should ideally contain the least possible number of moving objects, since they are sources of occlusions for MVS. This can be estimated either using optical flow or by computing the amount of pixels corresponding to classes like people and vehicles, if a semantic segmentation of the image is available. Also, the EmptyCities project could be used as a preprocessing step to remove dynamic objects [1].

Note that some other works, such as [2], also require that two keyframes are not too temporally distant. However, this makes little sense in our case (e.g. consider a car standing still before a semaphore: a single frame could be representative of hundreds of frames).

3 Depth Range Computation

For each selected keyframe, the depth range is computed simply by transforming the 3D point in the world coordinate frame to the local frame of the current image and taking the minimum (and maximum) value on the z-coordinate.

4 Camera Clustering Algorithm

The camera clustering algorithm aims at building a set of (possibly overlapping) clusters of cameras and points that can be processed in parallel by the MVS stage. There are a few works in literature, such as [3] [4] [5] [6], but their major drawback is that clustering is done by testing each camera agains each other, which makes little sense if both points and cameras are (almost) uniformly distributed in space. Our solution is to cluster the 3D space in independent blocks and assign the best cameras to each block. Given a vehicle that moves in the (x, z) plane, the algorithm works as follows:

1. Divide the world in blocks with size (x_b, z_b). The total number of blocks will be:

$$B = B_x \cdot B_z = \left(\frac{||x_{max} - x_{min}||}{x_b} \right) \cdot \left(\frac{||z_{max} - z_{min}||}{z_b} \right) \tag{1}$$

   with (x_{min}, x_{max}) and (z_{min}, z_{max}) the range of the point cloud. These blocks are organized in a 2D grid with indices $i \in [0, B_x - 1]$ and $j \in [0, B_z - 1]$.

2. Cluster the point cloud by assigning each point to the corresponding block. This is very easy to do, just loop over the points and and assign the current point (P_x, P_y, P_z) to the block $(i, j) = (P_x/x_b, P_z/z_b)$.

3. Remove blocks without points. Then, for blocks with a few number of points, cluster them together with the smallest block among adjacent ones.

4. For each block, assign to it all the cameras that can see the associated points at a sufficient distance. If needed, this could be improved by selecting only a fixed number of *best* cameras, given some definition of how a camera is *better* than another one. Also, this can be done in parallel.

5. Remove blocks without cameras. Then, for blocks with a few number of cameras, cluster them together with the adjacent block with the least number of cameras associated.

The output of the algorithm is a set of clusters with points and cameras associated to each cluster.

5 View Selection Algorithm

In the MVS stage, each cluster is processed independently and each *reference* image in a cluster needs a set of *source* images. The view selection algorithm assigns this set of source images to each reference image in the cluster. Currently, the most naive approach is implemented, in which a covisibility score is computed between two images I_i and I_j as:

$$w_{ij} = \sum_{f \in \mathcal{F}_i \cap \mathcal{F}_j} e^{-\frac{(\alpha(f) - \mu_\alpha)^2}{2\sigma_\alpha^2}} \tag{2}$$

where the triangulation angle corresponding to a feature $\alpha(f)$ is computed as:

$$\alpha(f) = \arccos \left(\frac{\mathbf{v}_{ref} \cdot \mathbf{v}_i}{||\mathbf{v}_{ref}|| \cdot ||\mathbf{v}_i||} \right) \tag{3}$$

with $\mathbf{v}_{ref}$ and $\mathbf{v}_i$ the view rays of the reference and the current image, respectively, associated to the feature f. The parameter μ_α is a target triangulation angle, while the parameter σ_α controls the smooth transition of the weight between 0 and 1. Then, the best M images for each I_i are selected as the source set. This implicitly favors image pairs with a high number of shared features and it is the method implemented also in [7]. There are three directions that can be taken in order to improve this algorithm:

1. When computing the covisibility score, two images should be favored if the 3D point associated to a common feature is equally spaced from the two camera centers:

$$w_{ij} = w_{ij} \cdot \sum_{f \in \mathcal{F}_i \cap \mathcal{F}_j} e^{-\frac{(||\mathbf{c}_i - \mathbf{X}(f)|| - ||\mathbf{c}_i - \mathbf{X}(f)||)^2}{2\sigma_d^2}} \tag{4}$$

2. Good source images might be in adjacent clusters.

3. A good set of neighbors should guarantee that the covisibility score is sufficiently high among *all* pairs in the source set, not just between the reference and each other image. However, it is also true that by selecting keyframes, we already avoid to pick images that are too similar, therefore this contribution should be negligible.

6 Conclusion

Finally, results are stored in the format required by PatchMatchNet [8], in order to be processed by the MVS algorithm.

References

[1] Bescos et al., *Empty Cities: a Dynamic-Object-Invariant Space for Visual SLAM*, IEEE Transactions on Robotics 2020

[2] Mur-Artal et al., *ORB-SLAM: a Versatile and Accurate Monocular SLAM System*, IEEE Transactions on Robotics 2015

[3] Furukawa et al., *Towards Internet-scale Multi-view Stereo*, CVPR 2010

[4] Zhang et al. *Joint Camera Clustering and Surface Segmentation for Large-scale Multi-view Stereo*, ICCV 2015

[5] Mauro et al, *Overlapping Camera Clustering Through Dominant Sets for Scalable 3D Reconstruction*

[6] Ladikos et al., *Spectral Camera Clustering*, ICCV 2009

[7] Yao et al., *MVSNet: Depth Inference for Unstructured Multi-view Stereo*, ECCV 2018

[8] Fangjinhua et al., *PatchmatchNet: Learned Multi-View Patchmatch Stereo*, ArXiV 2020