# Robotic Vision Scene Understanding Challenge: MSCLab Report

Antyanta Bangunharcana Soohyun Kim Kyung-Soo Kim
Korea Advanced Institute of Science and Technology, Daejeon, Republic of Korea
{antabangun, soohyun, kyungsookim}@kaist.ac.kr

In this report, we describe our method that is based on a combination of existing works to construct a semantic map within a BenchBot simulator. The core of this challenge is to build a semantic map and detect changes in the scene, using the images captured by the robot that can be actively given the action commands to move around the environment.

## 1. Semantic SLAM

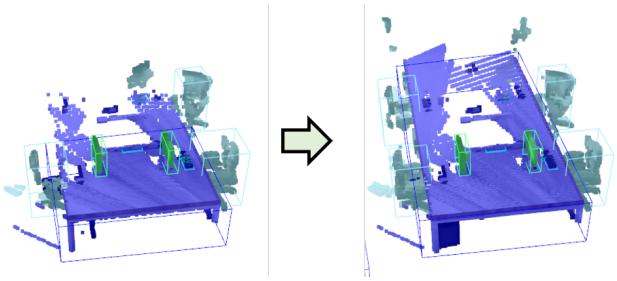

Figure 1. Updating the bounding box of matched table instance.

In the semantic SLAM part of the challenge, our objective is to build a tight-fit bounding box for every detected instance of interest. For this purpose, we define the following dictionary of map variables to represent mapped objects.

1. instances: this variable stores information for each detected instance. The information stored are the current centroid and extent estimates, and the corresponding class ID

2. voxels: When an instance is detected and projected onto 3D, we register the voxels corresponding to the said instance. Each voxel stores information on its 3D coordinates, the number of counts an instance is registered onto said voxel, and the prediction confidence. In our implementation, we set the resolution of the voxels at 2.5$cm$ to limit the memory and computation time.

Given the input RGB and Depth images at each timestep, we first pass the RGB-D point cloud into a 3D detector FCAF3D [3]. This gives as the 3D bounding boxes of objects covered by this model. However, the object classes covered by this model don't include all of the classes of interest in this challenge. So we additionally extract the instance segmentation on the image using QueryInst [2] using the MMDetection implementation [1]. We chose this method as it gives a good balance between accuracy and speed. The points for the detected instances are then reprojected onto 3D and registered to a voxel in 3D, along with the prediction confidence.

For the instances with a corresponding detection from FCAF3D, we simply associate the corresponding 3D bounding boxes. While for the instances without any corresponding 3D prediction, we fit a tight box from a set of predefined anchor boxes. The anchor boxes are chosen based on common knowledge of the approximate sizes of objects. The use of anchor boxes helps to build a better bounding box when the instances are only partially visible. During fitting, we also scale and shift the anchor boxes, similar to an anchor-based object detector.

After we compute the instances' voxels and the 3D bounding boxes, we try to match them with previously detected instances. If the number of intersecting voxels or the IoU of the boxes passes a certain set threshold, then the objects are matched. When the objects are not matched, they are simply registered as a newly detected instance in the dictionary. When the objects are matched, we compute the combined bounding box and add it as another anchor box of that instance. Then we compute a new tight fit box for that instance. Figure 1 illustrates this procedure. The confidence of the matched instances is then updated accordingly. This procedure is repeated at every timestep to iteratively update the map. At the end of the run, only objects with confidence higher than 0.9 are considered.

### 1.1. Current Limitations

In this current implementation, the method is based on rule-based logic by combining multiple existing methods. As such, some scenarios are still unaccounted for. Here, we report some limitations that we encounter in our experiments, which are also illustrated in Figure 2.

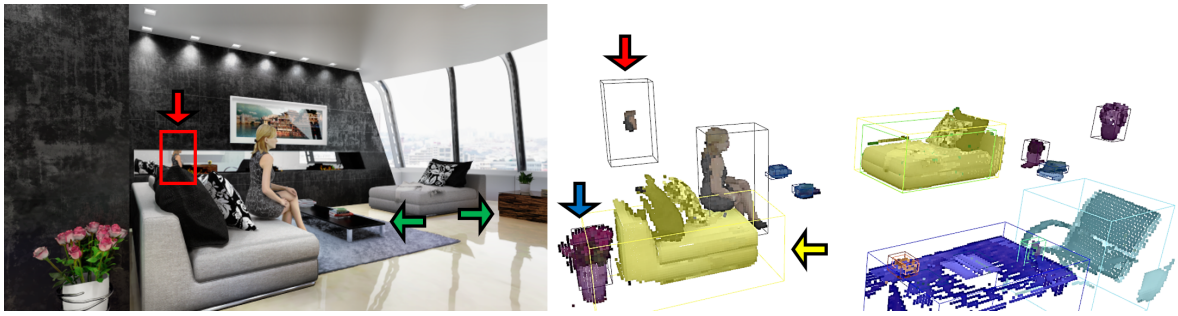

Figure 2. Current limitations of this submission.

1. **Object reflections** are detected as real objects and registered as 3D objects, as shown by the red arrow.
2. The 3D bounding box for **partially occluded objects** may not cover the unseen parts of the object, as shown by the yellow arrow. The use of anchor boxes partially resolves this.
3. However, sometimes the anchor boxes don't cover the whole of the detected objects, as shown by the blue arrow, with the 3D box covering only the top part of the potted plant.
4. Objects which are out of the training distributions may not be detected like the low tables indicated by the green arrows.
5. In addition, we have not implemented any method to remove false positives from detection.

## 2. Scene Change Detection

To perform the scene change detection, we add additional information into the instances variable, namely the scene ID. It registers the ID of the scene in which it was detected when it is detected. Then at the end of the run, this information is used to determine any newly added or removed objects.

## 3. Active actuation

In the active actuation phase, we base our method on the frontier exploration approach [4]. We use the 2D LiDAR data to build a birds eye view map of the environment. At each time step, we mapped only the free space and obstacles that are in front of the object within a range of $6m$ to represent scene areas that are observed by the camera. However, we also observe that the 2D LiDAR sometimes does not represent some of the surrounding obstacles. So we also use the point cloud from the RGB-D image that is between $0.2 \sim 1.2m$ in height to build the BEV map.

Using the constructed BEV map, we can then compute the *frontier* areas that are not yet observed. A frontier point is then chosen based on the distance from the current position. Then A-star algorithm is used to find a path towards this target while avoiding the mapped obstacle. If A-star cannot find a path towards this point, then we delete this frontier point and compute a new one. We then use the computed path to input an action command for the robot to follow this path. Once the robot reaches the target frontier point, we input an action for the robot to rotate 45 deg 7 times to map the surrounding 360 deg scene around the frontier point. Then the process is repeated until no frontier point is left or the timesteps exceed 300.

Figure 3. Matched table .

## References


[1] Kai Chen, Jiaqi Wang, Jiangmiao Pang, Yuhang Cao, Yu Xiong, Xiaoxiao Li, Shuyang Sun, Wansen Feng, Ziwei Liu, Jiarui Xu, Zheng Zhang, Dazhi Cheng, Chenchen Zhu, Tianheng Cheng, Qijie Zhao, Buyu Li, Xin Lu, Rui Zhu, Yue Wu, Jifeng Dai, Jingdong Wang, Jianping Shi, Wanli Ouyang, Chen Change Loy, and Dahua Lin. MMDetection: Open mmlab detection toolbox and benchmark. *arXiv preprint arXiv:1906.07155*, 2019.

[2] Yuxin Fang, Shusheng Yang, Xinggang Wang, Yu Li, Chen Fang, Ying Shan, Bin Feng, and Wenyu Liu. Instances as queries. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6910–6919, 2021.

[3] Danila Rukhovich, Anna Vorontsova, and Anton Konushin. Fcaf3d: Fully convolutional anchor-free 3d object detection. *arXiv preprint arXiv:2112.00322*, 2021.

[4] Brian Yamauchi. A frontier-based approach for autonomous exploration. In *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*, pages 146–151. IEEE, 1997.