Structure-Aware Object SLAM

Sachit Mahajan Kevin O'Brien
{sachitma, kobrien}@andrew.cmu.edu

1. Abstract

We propose a novel method for representing objects in an object-based simultaneous localization and mapping (SLAM) system. Tracking objects in a SLAM system offers obvious benefits over traditional SLAM methods that track featurepoints from frame to frame. Object tracking allows for much more efficient computational performance and lower likelihood of false positive matches between frames, as there are many fewer unambiguious objects in a given pair of scenes than there are featurepoints. Our project tackles the front-end piece of the SLAM problem. This entails detecting objects and planes in each frame and parametrizing each object based on its position relative to each plane in the scene. Like other object SLAM representations, ours can allow for estimating odometry between two frames based on the relative poses of static objects in the scene. Where our representation shines, however, is that the object-to-plane linking method allows us to determine whether a given object has noticeably moved within the scene. If an object has moved, we can detect this motion and use this knowledge to discard that object's pose when performing odometry estimation. Using the same method, we are also able to identify repeated objects, symmetric objects, and inaccurate object pose estimations. All of these detections enable us to use as few as one object to perform scene recognition for loop closure detection in the SLAM pipeline. We also develop an algorithm to use this representation and filter our dynamic, repeated and poorly estimated objects in a scene as a proof of concept.
Keywords: Object SLAM, dynamic, Planar SLAM

2. Introduction

2.1. Motivation

Indoor scenes such as homes, offices, restaurants, malls, warehouses, etc. are the target environments for service robots. Mapping and localizing in such environments is not an easy problem and an ongoing field of research. Oftentimes, indoor scenes consist of multiple objects that can be used as landmarks. In addition to objects, indoor scenes also consist of dominant, texture-less planar surfaces such as walls and floors. In most indoor SLAM systems, the entire scene being mapped is presumed to be static. In our project, we aimed to take the best parts of a few different types of SLAM pipelines (namely, object-based and plane-based approaches), and combine them to handle cases for which the scene is not static. In this section, we will discuss the three methods that we considered when formulating our project. At its highest level, our project combines ideas from the methods described in Sections 2.2.2 and 2.2.3.

2.2. Indoor SLAM Methods

2.2.1 Indoor SLAM using featurepoints

Traditional SLAM in indoor environments is a very challenging task. When using detected featurepoints as the reference between successive frames, there is a high likelihood of false positive correspondences throwing a SLAM system off track. This is due to the largely visually repeating nature of indoor scenes. Things like walls, tables, and other large planar surfaces look very similar across many vantage points, making it difficult to consistently find featurepoints and accurately match them between frames. In addition to repetitions across the same scene, indoor environments also tend to repeat themselves in different scenes as well (think of two different hallways with similar wall/floor patterns) that could further corrupt a SLAM system's localization estimate. Given all of these issues, we moved to mitigate them by exploring SLAM using object detections instead of featurepoint detections.

2.2.2 Indoor SLAM using objects

Object SLAM methods provide a good remedy to some of the issues discussed in the previous section. In a given scene, there are many fewer objects than featurepoints, so the process of matching objects between frames becomes much more trivial. The smaller number of points we are trying to match, along with the fact that detected objects will tend to have much larger visual deviations from each other than detected featurepoints would, means that the likelihood of finding false positive object matches between frames decreases drastically.
However, there are still some downsides to this approach. If a given object is dynamic within the scene, matching that object between frames will give an incorrect estimation of odometry, since the object's motion will not align with the camera's motion. Additionally, certain object classes in a scene are not unique and could be repeated, and distinguish-

ing between the different object instances is important to prevent matching the correct object class, but the incorrect object instance (ie, in a scene with two chairs, associating chair 1 in frame 1 to chair 2 in frame 2). Additionally, robustly detecting the correct orientation of symmetric objects is a challenge, since the reported orientation of a 3D bounding box detection could easily be flipped by 180 degrees in any direction but still accurately surround the object in question. More generally, object SLAM methods are also highly sensitive to the accuracy of object pose estimation.

2.2.3 Indoor SLAM using planes

Another family of methods for indoor SLAM utilizes planes for odometry. Planes tend to occupy a large area in indoor scenes, and so are fairly easy to accurately extract. They are also reasonably easy to match between frames, provided that there is small motion between frames. For example, a wall on the left of the camera will remain on the left of the camera if I turn or move slightly in any direction. However, we once again run into the issue of repeating scenes using these methods, as all walls in a given building or room will look the same, so distinguishing between the various walls becomes a problem. Another problem with using planes is that a single plane does not provide enough constraints for 6 degree of freedom (dof) camera pose estimation, so we need at least 3 non-parallel planes to accurately estimate pose.

2.3. Proposed Approach

We propose a novel representation of objects that can be used for object SLAM. This is done by tethering the objects to the Manhattan planes of the scene. First, we find the dominant Manhattan planes in the scene and assume them to be static. Each detected object in the scene is then tethered to each plane. If the object moves between two frames, at least one of the object's plane measurements (plane normal or plane center) will change. This can help in not only detecting if the object is dynamic, but also predicting in which degree of freedom the object moved.

Similarly, if we see multiple instances of the same object class, the measurements of the planes associated with each object instance will be significantly different, and we can detect that we are actually dealing with multiple different objects. This method also allows us to tackle the errors caused by the estimated pose of symmetric objects. It is difficult to accurately determine the orientation of symmetric objects, as multiple different orientation of object look exactly the same. However, if the objects are tethered to planes, the plane measurements will look different at different orientations. Another use case of this representation is that it can be used to detect when an object pose estimation is wrong, assuming that we have at least one prior measurement with correct object pose.

An example can help us to visualize these different cases: say we detect a chair in a room. The room is shaped like a box made of 4 walls, floor and the ceiling, totaling 6 planes. When we first see the chair, we store the distance of the chair from all 6 planes as well as the relative orientations of these planes with respect to the chair. If the chair is moved or rotated in future frames, the distances or relative orientations of the planes with respect to the chair will be different. Since all the planes are assumed to be static, we can use this information to detect if the chair moved. We can use the same method to check if we are looking at a chair in a different room altogether. Our algorithm will not differentiate between these different scenarios but detect if any of these occur for each object and will output the objects that should be discarded if any of the above mentioned condition occurs. Hence it can be used to filter out dynamic, repeated and poorly estimated objects.

When working in the real world it is unlikely that we see all 6 planes in a single frame, however as long as we see 3 non-parallel planes we can accurately detect dynamic, repeated, or symmetric objects, as well as poorly estimated object poses. Even if we are unable to detect all 6 planes in a given pair of frames, we can still detect motion in some degrees of freedom. For example if we see the ground plane in a given frame and a previous frame, we can check the object's roll and pitch, as well as its translation along the scene's vertical axis.

Using all of this information, we finally assign a confidence score to each degree of freedom of each object based on the available constraints. In the example above, if the object's planar measurements with respect to the ground plane match in successive frames, we will assign a high score for the roll, pitch and Z translation degrees of freedom, as we can be sure that the object did not move in those directions. Hence, using this linked object-plane method allows us to reap the individual benefits of each existing method SLAM method discussed in 2.2, while overcoming the limitations of using only one method. We no longer assume a static world with unique objects.

Most importantly, we can use this representation to validate objects between any two frames regardless for the motion between them. This enables a global registration which can be used for scene recognition or loop-closure/relocalization detection. This combined object-plane representation allows us to represent the frame using very few parameters, especially when compared to feature-based or direct methods, while providing sufficient constraints to estimate odometry.

To summarize, we have listed our contributions below:

- A novel representation of object that links the object to planes
- Using this representation to detect if the object is dy-

namic, and to remove ambiguities caused by repeated and symmetric objects

- A novel method for scoring objects by assigning a confidence score in each degree of freedom based on the available constraints and filtering out dynamic, repeated and poorly estimated objects.

3. Related Work

We took a lot of inspiration for our project from CubeSLAM [15] , which follows a similar idea of obtaining 3D bounding box estimates of objects and using those to inform the SLAM pipeline estimates. CubeSLAM is able to use dynamic objects to help in the odometry estimation, rather than treating those dynamic objects as outliers like we do. The main difference though, is that their system is only shown an outdoor dataset (KITTI), so there is more variance between frames than in an indoor dataset like the one we are using. CubeSLAM does not use objects for loop closure detection due to the problems mentioned above in motivation section. The work [14] extends [15] and adds planes in SLAM formulation. The plane extraction is based on a deep learning method, and mainly used for scene reconstruction. Another work used for reference is [7], this work jointly optimizes on constraints from featurepoints, object and planes. They do not link planes to objects, but do add point-plane constraints in their SLAM formulation. They represent objects as quadrics which allows a compact representation of rough extent. The work [11] uses objects in SLAM formulation and considers objects to be parallel to the ground plane and penalizes objects farther away from ground plane. It also introduces using objects for loop closure, where loop closures are attempted if three or more objects are matched. [11] assumes the world to be static and with unique objects. The works [9][8] [2] discuss on how to formulate the SLAM problem using infinite plane representation. They elaborate on plane matching and using planar constraints to get odometry.

Plane Extraction: We looked into both geometric [5] [2] [4] and deep learning [10] [14] based methods for plane extraction. The geometric methods mostly relied on either RANSAC or region-growing methods. We ended up using [4] a region growing method for real-time plane extraction. For the plane representation we took inspiration from [9]. Another interesting work we took inspiration from was [8] where the authors achieve better plane extraction by adding a filtering step in point cloud pre-processing.

Plane Labelling: The works [6][5][3] use classifiers to assign labels to the planes. Since we already had the plane orientation from the normals, we did not need a classifier. We used the Manhattan assumption followed in [3]. We used the energy functions and mentioned tricks to form heuristics for plane labelling.

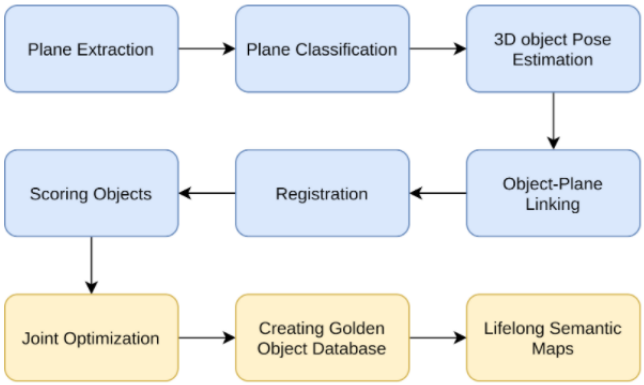

Figure 1. Our algorithm. The blue cells show what we implemented in the project, and the yellow cells show what we see as future work. First we extract planes from the image and classify them as walls, ceiling, or floor. We then perform 3D object pose detection to find the poses of all the objects in the scene. Next, for each object, we find the relative pose of each plane to that object, giving us a solid estimate of where in the world the given object is. In future frames,d we attempt to register each object detected with all the objects detected in the previous frame. We then assign each object a score per degree of freedom of the object. A high score indicates that the object has not moved along the given degree of freedom, while a low score indicates that the object has move along that degree of freedom. We use all of these scores in a heuristic function that accounts for some degree of variability in the object detection network to determine whether a given object has moved, and therefore, whether it should be used or discarded in future odometry estimates

3D Object Pose Estimation: Initially, we pursued using [15] method for detecting 3D bounding boxes of objects given a proposed 2D bounding box on the image plane. This method uses the line segments in the image to give an estimate of the vanishing points in the scene, and tries to generate cuboid proposals that are consistent with these vanishing points. After experimenting with their codebase, we saw decent results, but the generated cuboids seemed to always be skewed from the expected result. We think this has something to do with the fact that their algorithm assumes the world origin to be at the object rather than at the camera center. We eventually abandoned this method in favor of a deep learning based object pose detection method, DOPE [13].

4. Method

In this section we will expand on the major steps that we implemented in our pipeline. A flowchart of the process we implemented is shown in Figure 1.

4.1. Plane Extraction and Classification

We assume that we are in a cuboid room. The geometric model of the room layout L will be composed of at least six

Manhattan planes ($\pi_1, \pi_2, \pi_3, \pi_4, \pi_5, \pi_6$), where each plane π_i is parametrized by its plane normal n_i and distance to the origin (plane center) d_i.

To detect all of the planes in a given image, we utilize the RGB-D camera's depth map to generate estimates for large planar surfaces in the scene. To accentuate edges and eliminate noise, we use a bilateral filter to pre-process these depth images. This edge aware filtering enables us smoothen the areas with low gradients while preserving the edges. We then passed these images into the pipeline created in the paper [4]. This method uses normal clustering and region-growing methods to predict the dominant planes in the image.

Once we have all of the planes in a given scene, we classify them into four categories: walls, ceilings, floors, or clutter planes. Inspiration for this method came from [5].

We use the relative position and orientation of planes to first classify the ground and ceiling planes. The assumption we use here is that the ground and ceiling plane visible in a frame will always be anti-parallel. Once we have the ground and ceiling planes classified, we use the assumption of a Manhattan world to find the walls, as we assume all the walls in the scene will be vertical. All vertical planes are filtered using the plane's 2D area, Mean Squared Error, RGB mean, RGB variance, and luminance mean to determine which planes are in fact walls, and which planes are just clutter planes. We define the plane area as the percentage of an image's pixels lying on the plane. If the area of a detected plane is less than 3% of the image's area, the plane is ignored. Once we filter these planes out, we know that the textureless walls will have a similar RGB mean and low RGB variance.

4.2. Object Pose Detection

Next, we use 3D object pose detection to determine the position of all of the objects in the scene relative to the camera's pose.

After experimenting with [15], we eventually settled on using the network DOPE [13], which gives a very accurate results. DOPE's advantage is that it is trained solely on synthetic object data, so it is very easy to generate large training datasets. This abundance of training data leads to state-of-the-art object pose detection results on YCB objects that we were able to add to our simulator. The network generates a belief map of object edges, and uses points along those edges in a PnP algorithm to determine cuboid vertices and subsequently, object poses. We used a set of weights for each object that was trained on the 3D models in the YCB household objects dataset. This network worked fairly robustly, and was able to give us reasonable pose estimates for all of the objects in most frames. Once an object was detected, we manually set the object's roll and pitch to be zero, as we assumed all objects to be oriented parallel to the ground plane, same as in [11]

4.3. Object-Plane Linking

Now equipped with all the object and plane data, we are ready to link everything together. For each object, we determine the relative positions of all of the planes in the object's frame of reference. This gives a relative estimate of where each plane lies relative to each object, effectively "tethering" the object to the planes in the scene. We build up a hierarchical data structure where for each frame, we have a list of object structures. In each object structure, we have that object's pose in the camera frame, as well as all of the poses of all of the planes in the object's frame. We then use this structure to perform registration and scoring between frames for each object.

4.4. Registration and Scoring

After performing this initial data structure setup, we can now go about the process of comparing objects in successive frames. In any pair of frames where the same object instance has been detected in both frames, we compare the relative poses of all the planes in the objects' respective frames. The labeled planes are compared to their counterparts (ie, all wall planes are compared with each other, etc). From these two sets of plane-relative poses, we define a heuristic function to see whether the object has moved relative to the planes. This consists of 3 constraints used to compare two planes π_1 and π_2 with normals n_1 and n_2 and centers d_1 and d_2 respectively:

- Plane orientation: Same planes should be parallel or anti-parallel to each other

$$|n_1 \cdot n_2| = 1 \tag{1}$$

  Wall planes can also be orthogonal to each other, in which case

$$|n_1 \cdot n_2| = 0 \tag{2}$$

- Distance between planes: Planes should be close to each other (ideally the centers should be in the same position relative to the object)

$$(d_2 - d_1) \cdot n_1 = 0 \tag{3}$$

- Luminance check: Same plane would have similar values in the luminance channel. This will be used when we know the frames are continuous and not for scene recognition.

Since real data is noisy, we tune thresholds for the tests, for example in Equation (1) we say that two planes are parallel if $|n_1 \cdot n_2| > thr_p$ where thr_p is the parallel threshold.

We then assign a score to each possible degree of freedom of the object. Based on how similar the plane poses are

between the two frames, we are able to assign a high confidence score if the object did not move in a given degree of freedom, and a low confidence score if the object did move in that degree of freedom or if we don't have enough constraints along that degree of freedom. We define a heuristic function to use all of these six confidence scores to determine which objects to discard in our odometry estimate between the two frames, and which objects to keep.

We assign a high confidence score to a given degree of freedom if the corresponding plane is

- Parallel or anti-parallel to its counterpart
- Close to its counterpart in position
- Perpendicular to other planes

If we have sufficient planes that can be used to detect motion in a dof, and that dof still has a low confidence score, our algorithm suggests that we discard the object while determining the odometry estimate or scene recognition. In some cases objects, that are dynamic are not discarded as we do not have sufficient planes to make a decision. Even in these cases we will still assign a low confidence score.

An example: Say we are comparing frame 1 and frame 2. We detect two walls, one ground plane, and a chair in each frame. If a wall (in frame 1's chair's frame of reference) in frame 1 is neither parallel nor perpendicular to the two wall planes of frame 2 (seen by the chair in frame 2) and that wall is not close in position to either of the frame 1 planes, it will be suggested that we discard the chair.

Continuing with the same example, if only one wall and the ground plane is detected in the second frame, we cannot accurately estimate the motion using only two planes as the problem is under-constrained. Instead, we will just check that the wall plane of frame 2 is still perpendicular to the ground plane of frame 1 and assign a confidence to the object's roll and pitch accordingly. Additionally, we will assign confidence to the object's yaw if frame 2's wall is parallel/perpendicular to the walls in frame 1, and add confidence to the object's X and Y direction if the wall planes are close in position. The Z direction and yaw will be assigned confidence if ground planes match in position and orientation respectively.

5. Experiments

5.1. Data Collection

For data collection, we used the Habitat-Sim platform[12] to generate a sample room and render new objects into the scene. Usng this simulator instead of available datasets provided us more freedom and customizability to conduct our experiments. We used the cracker (Cheezit), sugar, soup, mustard, and gelatin (Jello) 3D object models from the YCB Model Dataset [1] as our scene objects to be detected, as we already had access to pretrained weights for the DOPE network using these objects. We can see a sample output of Habitat-Sim in Figure (2)

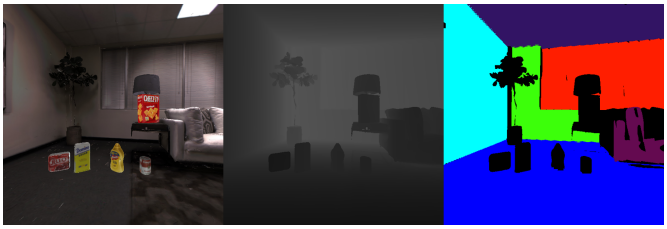

Figure 2. The image on the left shows the Habitat-Sim RGB frame, the middle one is the depth map and the one on the right shows the extracted planes

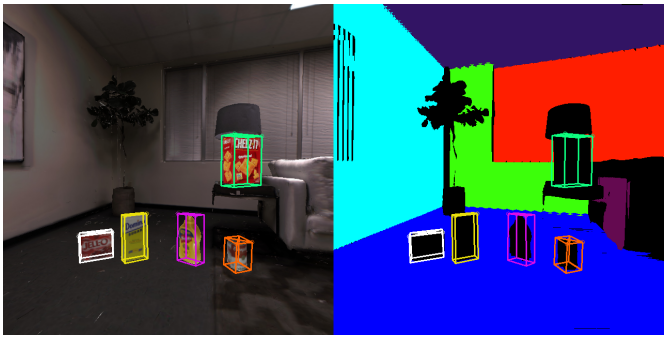

Figure 3. The image on the left shows the detected objects, the one on the right shows the linked object-planes representation. This frame will be referred to as Frame 0

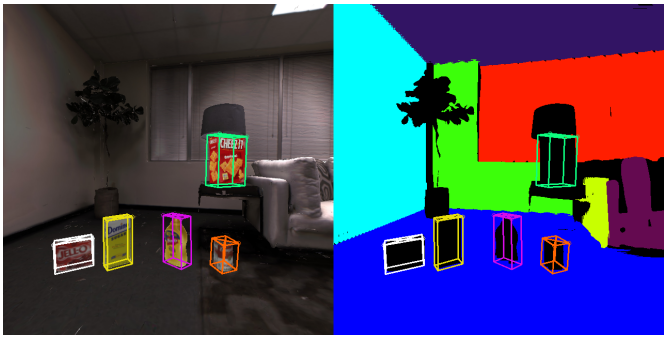

Figure 4. Frame with small motion as compared to Frame 0

We collected around 100 different frames of the scene with all objects present, moving some objects around in the scene to try and see if we could properly detect when a given object has moved between frames. We tried to simulate the scenarios discussed in above sections to demonstrate that our method can properly identify these scenarios. We start with a canonical frame, detect all the objects, and link the planes to the objects. This frame and representation can be seen in Figure (3). This frame is called Frame 0 and we compare the different scenarios this Frame 0.

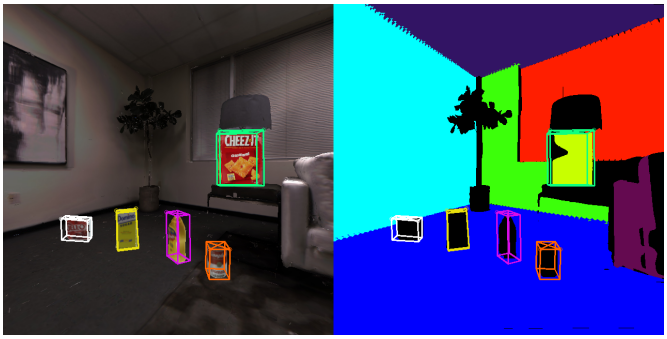

Figure 5. Frame with different viewpoint as compared to Frame 0

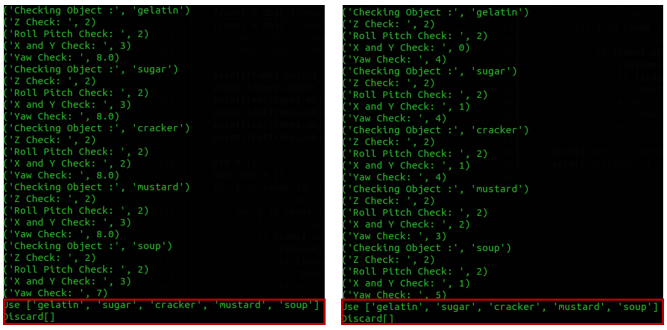

Figure 6. Left image shows results and scored for frame in Image (4) when compared to (3). The one on the right does the same for frame in Image (5) when compared to (3)

5.2. Scenes with Static Objects

In this experiment we compare Frame 0 with two frames, one with small motion (next consecutive frame) seen in Figure (4) and one at a different viewing point seen in (5). In both the scenarios, our system can detect that all the objects have not moved and assigns a high confidence score, so no objects are filtered out.

5.3. Scenes with Dynamic Objects

In this experiment, we moved some of the objects seen in Frame 0 to test whether our algorithm can detect detect this change and suggest that we discard the moved objects. We move the sugar and cracker box and produce the scene as shown in Figure (7) and compared it to Frame 0.

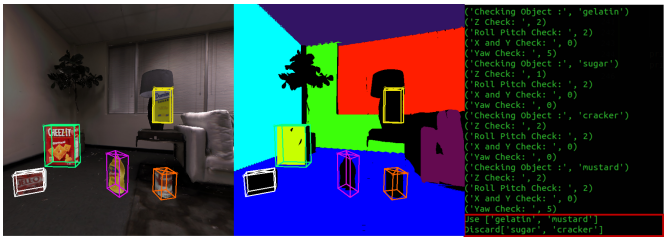

Figure 7. Frame with sugar and cracker boxes moved as compared to Frame 0, the image on the right is our result

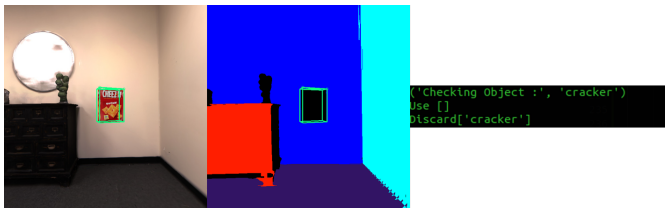

Figure 8. Frame with cracker boxes in a different room as compared to Frame 0, the image on the right is our result

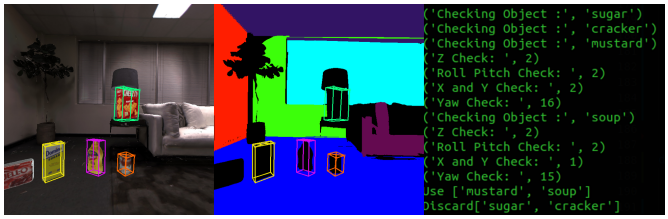

Figure 9. Frame with cracker box having error in pose estimation

We can see that our algorithm detects that sugar and cracker box were dynamic and asks us to discard these objects and use the remaining objects.

5.4. Different Scenes with Repeated Objects

The objective of this experiment is to place the same object in two entirely different scenes and test if our algorithm suggests us to discard the object. This has applications in scenarios where the same object class is repeated in a house, for example, and can cause false loop closures.

We place the cracker box in a different room and compare it to Frame 0. This can be seen in Figure (8). In the same figure you can see that our algorithm asks us to discard the cracker box, since it interprets this cracker box as a new cracker box. From this result, we can infer that this is a different scene altogether as compared to Frame 0.

5.5. Scenes with Poor Object Pose Estimation

We demonstrate that we can use our method to filter out objects whose pose has been inaccurately estimated. For this experiment we had tightened the thresholds in Section 4.4. We used a frame in which the cracker box was poorly estimated, this is shown in Figure (9).

Since the thresholds are tightened, we are more sensitive to noise. That can explain why our algorithm suggests that we discard the sugar box even though it looks visually correct.

5.6. Failure Scenario

In some experiments, the algorithm was unable to identify the dynamic objects. However, even in these cases the confidence score for motion in dof's where motion actually occurred was very low, which shows us that our algorithm is partially correct. One example of a failure is shown in Fig-

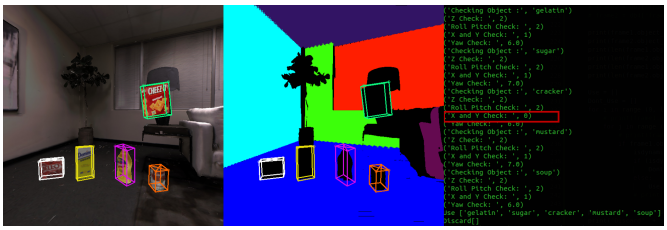

Figure 10. Frame with cracker box moved as compared to Frame 0

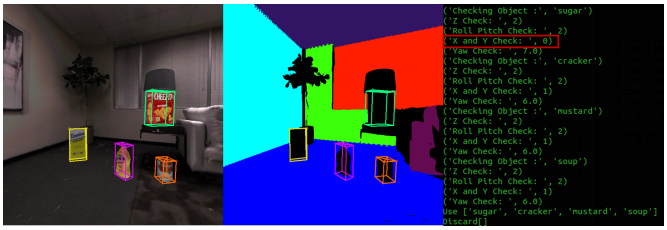

Figure 11. Frame with sugar boxes moved as compared to Frame 0

ure (10) where the cracker box was moved and rotated. One possible explanation of this failure can be the fact that our object pose estimation method hard-codes all object's roll and pitch as 0 as we assume objects to parallel to ground plane.

Another failure case can be seen in Image (11) where the sugar object was moved in the X and Y axis. We are unable to identify it as dynamic but still assign a 0 confidence score in those axes. This can be fixed by spending more time on tuning the thresholds to detect small object motion.

6. Conclusion

We have proposed a novel object representation for use in indoor SLAM systems that leverages the relative poses between the objects and planes in a scene. This linked object-plane representation can be used to perform frame-to-frame registration between static objects, as well as identify and filter out dynamic objects, repeating objects, and symmetric objects in the scene. By collecting our own data in the Habitat environment, and performing our own experiments, we have shown a proof-of-concept system demonstrating the validity of our object representation as a registration method between frames in a SLAM pipeline.

7. Future Work

The next step is using this representation in a SLAM pipeline where we discard the suggested objects and scale the uncertainty based on confidence scores. We can use the additional constraints provided by planes to jointly optimize for both odometry and object pose estimation. We can define the cost function from planes as

$$C = (1 - |n_1 \cdot A_{21} n_2|) + ((A_{21} d_2 - d_2) \cdot n_1) \quad (4)$$

where A_{21} is the odometry between the frames. This can be combines with the the cost function of object poses.

Our representation has application in the loop closure detection and relocalization part of the pipeline using objects. We can detect faulty loop closures as the objects will either have low confidence scores or would be discarded. Instead of the conventional Bag of Words file, we suggest that we form a 'golden object file' for each object, continuously updating and adding the planes which are co-visible. Then, along with matching two frames, we can also try to match the objects with the 'golden objects' and detect accurate loop closures.

Another way to extend this work would be forming compressed semantic rich planar-object maps. This representations allows us to store the scene information in a few parameters, it can be extended to lightweight maps with planes and objects.

8. Acknowledgement

This work was carried out as the final project for 16-887 Special Topics in Geometry-based Methods in Vision (Fall 2020) at Carnegie Mellon University. We thank the Professors and our TA's for their advise and guidance.

References

[1] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set. *IEEE Robotics Automation Magazine*, 22(3):36–52, Sep 2015. 5

[2] A. Concha and J. Civera. Using superpixels in monocular slam. 05 2014. 3

[3] A. Concha, W. Hussain, L. Montano, and J. Civera. Manhattan and piecewise-planar constraints for dense monocular mapping. 07 2014. 3

[4] C. Feng, Y. Taguchi, and V. R. Kamat. Fast plane extraction in organized point clouds using agglomerative hierarchical clustering. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6218–6225, 2014. 3, 4

[5] V. Hedau, D. Hoiem, and D. Forsyth. Recovering the spatial layout of cluttered rooms. In *2009 IEEE 12th International Conference on Computer Vision*, pages 1849–1856, 2009. 3, 4

[6] D. Hoiem, A. A. Efros, and M. Hebert. Recovering surface layout from an image. *Int. J. Comput. Vision*, 75(1):151–172, Oct. 2007. 3

[7] M. Hosseinzadeh, Y. Latif, T. Pham, N. Sünderhauf, and I. D. Reid. Towards semantic SLAM: points, planes and objects. *CoRR*, abs/1804.09111, 2018. 3

[8] M. Hsiao, E. Westman, G. Zhang, and M. Kaess. Keyframe-based dense planar slam. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5110–5117, 2017. 3

[9] M. Kaess. Simultaneous localization and mapping with infinite planes. *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4605–4611, 2015. 3

[10] C. Liu, K. Kim, J. Gu, Y. Furukawa, and J. Kautz. Planercnn: 3d plane detection and reconstruction from a single image, 2019. 3

[11] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison. Slam++: Simultaneous localisation and mapping at the level of objects. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, 2013. 3, 4

[12] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra. Habitat: A platform for embodied ai research, 2019. 5

[13] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox, and S. Birchfield. Deep object pose estimation for semantic robotic grasping of household objects. 2018. 3, 4

[14] S. Yang and S. Scherer. Monocular object and plane slam in structured environments. *IEEE Robotics and Automation Letters*, 4(4):3145–3152, Oct 2019. 3

[15] S. Yang and S. A. Scherer. Cubeslam: Monocular 3d object detection and SLAM without prior models. *CoRR*, abs/1806.00557, 2018. 3, 4