RGBDSLAM with Kinect V2

Muzzammil Shahe Ansar[a], Dharani Vasan[b], Yashwanth Prasanna[c]

[a]*SRMIST, , Chennai, , ,*
[b]*SRMIST, , Chennai, , ,*
[c]*SRMIST, , Chennai, , ,*

Abstract

The implementation of RGB-D SLAM with a Kinect V2 sensor can provide a robust solution for real-time simultaneous localization and mapping (SLAM) in dynamic environments. The use of a Kinect V2 sensor provides a rich source of information, including RGB images and depth data, which can be used to build a detailed map of the environment and estimate the robot's position.

In this implementation, the RGB and depth data from the Kinect V2 sensor is processed to extract geometric features called fast point feature histograms (FPFH), which are used to align subsequent point clouds and estimate the camera's pose. The resulting data is then incorporated into a pose graph, which is optimized to obtain an estimate of the robot's position and the environment.

The use of a pose graph approach allows for efficient optimization and the incorporation of prior knowledge, such as maps or landmarks, into the SLAM process. This results in a more accurate and robust estimate of the robot's position, even in dynamic environments.

Experimental results have shown that the implementation of RGB-D SLAM with a Kinect V2 sensor provides a real-time and accurate solution for SLAM in dynamic environments. The use of a Kinect V2 sensor provides a rich source of information for building a detailed map of the environment, and the pose graph approach provides an efficient and robust method for estimating the robot's position.

Contents

1. Introduction

Building on previous works on RGBDSLAM, here we propose a system that uses a three-way hybrid approach to create a robust SLAM system that works well in most scenarios encountered in indoor environments. It's a novel method that fuses previously known methods - using FPFH for visual odometry, using SIFT/SURF/ORB for visual odometry, and using an IMU for short-term odometry. All three methods are fused together to provide a particularly robust implementation of SLAM that does not fail often

2. Related Work

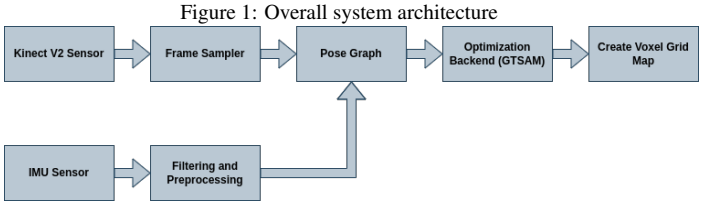
RGBDSLAM has already been implemented with multiple different RGB-D cameras (The Kinect V1, The Xtion Pro, and others). Multiple different approaches have been explored, but the most popular version of RGBDSLAM uses SIFT/SURF/ORB between subsequent frames and uses Umeyama's algorithm to calculate the transformation between the two frames (1). This works well enough but leaves a lot to be desired since only color information is used. Ideally, the extra information from the depth image should also be used for visual odometry. FPFH descriptors (2) are a method of using geometric data as well (Normals to be specific), but again, it is only geometric data. The incredible amount of detail present in the color image will be ignored. There have been attempts at alternative representations using deep learning as well (3). Modern attempts at RGBDSLAM try to better cope with a dynamic environment (4). IMU-based SLAM systems are some of the oldest and some of the most well-explored systems. There are multiple EKF-based systems, which use landmarks for IMU calibration. This paper builds upon both old and new methods to create a method that will be able to cope effectively with real-world usage.

3. Approach

This section gives a detailed look into how this method approaches the problem and how it is implemented. The SLAM system can be broadly divided into two categories - the frontend (Data association) and the backend (Optimization). This particular SLAM system is a Graph-based SLAM, which means that the backend will be one of many popular graph optimization frameworks (G2O, GTSAM etc). The end result will be some

kind of point cloud that is an aggregation of all the previous point clouds after applying a transformation to them to bring them into the proper coordinate space. This point cloud is then voxelized, and the end output is a voxel grid map that can be used directly for navigational purposes. Please refer to the figure on system architecture for more information.

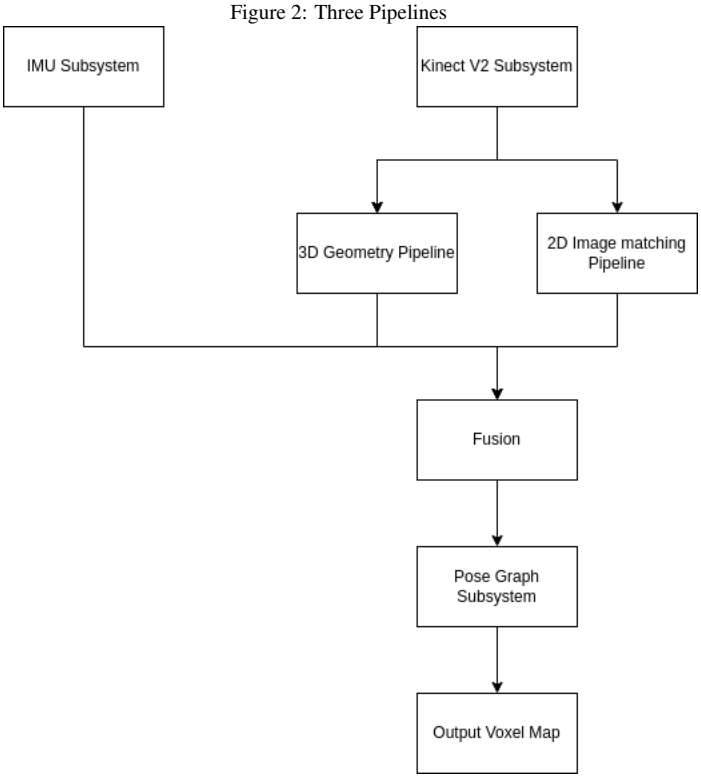
Figure 1: Overall system architecture



3.1. Frontend

The frontend is the most interesting part of any SLAM system and performs data association. The end result of the frontend is a pose graph with odometry and loop closures. The frontend performs well if it is able to, with decent accuracy, measure odometry and loop closures. More than anything, false positives need to be avoided. There are three different methods working simultaneously, as mentioned before. We will start with the FPFH method. The FPFH method starts with the Kinect. The RGBD Images are fetched using the Freenect 2 SDK. Once fetched they are filtered for noise (A simple median filter is being used) and then converted into point clouds. These point clouds are then compared to the previously fetched point cloud using standard methods (Rough FPFH-based global registration followed by colored point-place ICP registration). We use colored point-plane ICP registration here because it allows us to use color data as well, which the Kinect V2 offers in abundance. This odometry can fail or succeed. Assuming it succeeds we have a new pose connected to a previous one. If it fails we now have to decide how to represent the graph. We can use constant model methods or assume that the edge itself is broken. We use the latter method where we assume the odometry is discontinued and the edge representing that has a very high level of uncertainty. This gets optimized out by the backend. This is purely for odometry. For loop closures, a set of 12 uniformly sampled point clouds are kept in memory. Every new point cloud is compared to each of these 12 and if a suitable match is found it is inserted into the pose graph, with an uncertainty higher than that of odometry. It is imperative that false positives are avoided at all costs since they can completely ruin the optimized output. Of course, since we are using geometric data here, there are cases where the system fails - when there is not enough unique geometric data. This is rare in indoor environments since it has multiple edges, corners, objects, etc, but long stretches of corridors for example are not ideal since they look the same geometrically. There is a chance that there is some variance in the color data, so the exact same method is replicated, but instead of using FPFH-based global registration, we use SIFT/SURF/ORB-based global registration - this is the traditional method. When both these methods fail, we fall back to an IMU for odometry. The IMU itself is not ideal for long-term odometry since the integration error builds up and makes the end result unusable but on a relatively shorter time scale, it can be an effective method of tracking the movement of any robot through three-dimensional space. This three-pronged approach allows this system to perform admirably even when two of the three pipelines fail to function well. Refer to the figure below.

Figure 2: Three Pipelines



3.2. Backend

The backend of any pose graph slam system is a graph optimization framework. The GTSAM framework is used here. The framework works with factor graphs instead of pose graphs, but the conversion between the two is trivial. Any pose graph is just a set of poses connected to each other with edges. These edges will have an associated uncertainty as well. The goal is to optimize and find a minimum in the error function of the graph. The error function of the graph is simple the sum of the squared errors of the edges. It is in the end, a non linear least squares problem. The following equation gives us the error function -

$$E(\mathbf{x}) = \sum_{i,j \in C} \mathbf{e}(x_i, x_j, z_{ij})^T \mathbf{\Omega}_{ij} \mathbf{e}(x_i, x_j, z_{ij}) \tag{1}$$

$$\mathbf{x}_{optimized} = argmin\, E(\mathbf{x}) \tag{2}$$

Here $\mathbf{x}$ is the set of all poses, or as it is generally referred to, the state. x_i is just the i^{th} pose. The function $\mathbf{e(x_i, x_j, z_{ij})}$ is the vector error function of a single edge relating poses x_i and x_j under

Figure 3: Before Optimization

Figure 4: After Optimization

the constraint z_{ij}. Here z_{ij} is simply the measured transformation. The error function is zero when x_i and x_j are separated by a transformation exactly equal to z_{ij}. Multiple graph optimization algorithms are available, the choice is almost overwhelming. The SLAM system proposed here uses GTSAM, with a Levenberg-Marquadt optimizer. The end result $\mathbf{x}_{\mathbf{optimized}}$ is the state which results in the minimum value of the error function stated above.

4. Output

We will go through multiple scenarios of outputs possible from this system. The first is an odometry-only scenario. Here all loop closure detection from visual aid was disabled. We can see that the split ceiling and ineligible ceiling fans are now a single ceiling and legible fans. The voxelized version can then be seen. This is the version of the map that will be used for actual navigation purposes. Refer to Figures 3-5. The second part involves all pipelines functioning, we have odometry, loop closures as well as broken odometry, as can be seen from the output. The broken link causes parts of the map to create a small map in a different area. After optimization, these errors are optimized out resulting in a good map of the environment. The output is a cross-section of a room and the camera was moved in a manner that would result in an output like that. This was done to provide visibility. Refer to figures 6-8. And now finally, we have an output of the same room, but the camera

Figure 5: After Voxelization

Figure 6: Before Optimization

Figure 7: After Optimization

Figure 8: After Voxelization

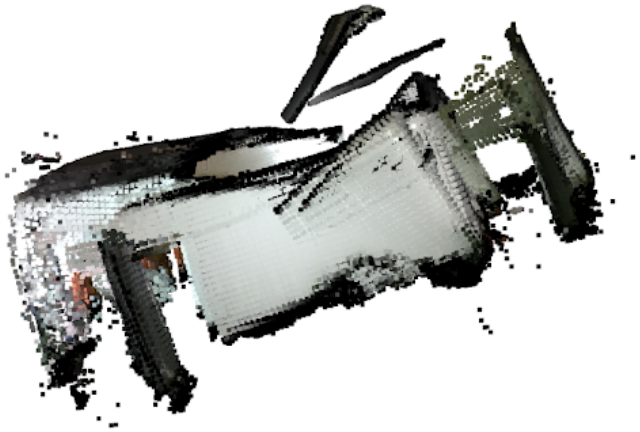


Figure 9: Before Optimization

Figure 10: After Optimization

was moved as it would be for actual rooms, so the output is not a cross section. It is more representative of the kinds of output that robots would deal with. Refer to figures 9-11.

5. Summary and conclusions

As can be seen from our output, we have a well functioning SLAM system that is more robust than any single method alone. The only difference between traditional methods of RGBDSLAM and this one is the presence of an IMU. In most robots, IMUs will be present regardless for a multitude of reasons, therefore using the output form one to improve the SLAM further adds no cost in most cases. The implementation is certainly more complicated than a single ended approach, but the results are well worth the cost. The chances of this SLAM system failing catastrophically is very low compared to traditional methods.

Acknowledgements

Thanks to Dr. R. Prithiviraj for guiding me through the project and helping me with multiple practicalities of publications. This would not have been possible without him.

Figure 11: After Voxelization

References

[1] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the rgb-d slam system," in *2012 IEEE International Conference on Robotics and Automation*, pp. 1691–1696, 2012.

[2] R. B. Rusu, N. Blodow, and M. Beetz, "Fast point feature histograms (fpfh) for 3d registration," in *2009 IEEE International Conference on Robotics and Automation*, pp. 3212–3217, 2009.

[3] Y. Zhou, Y. Wang, F. Poiesi, Q. Qin, and Y. Wan, "Loop closure detection using local 3d deep descriptors," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 6335–6342, 2022.

[4] L. Yan, X. Hu, L. Zhao, Y. Chen, P. Wei, and H. Xie, "Dgs-slam: A fast and robust rgbd slam in dynamic environments combined by geometric and semantic information," *Remote Sensing*, vol. 14, no. 3, 2022.