Final Viva

on

Implementation of RGBDSLAM with a Kinect V2

by

Muzzammil Shahe Ansar (RA1911004010665)

dhaRani Vasan (RA1911004010668)

Yashwanth Prasanna (RA1911004010676)

Batch ID - MD43

Under the guidance of

Dr. R. Prithiviraj

Assistant Professor , Department of ECE

INDEX

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.

OBJECTIVE

- To create a robust SLAM system by fusing 3 different kinds of SLAMs and reduce the RMS error to lower than 0.2 m
- To create an RGBDSLAM system that uses conventional 2D image matching.
- To create an RGBDSLAM system that uses 3D geometry matching.
- To create a SLAM system that uses an IMU.

PROBLEM STATEMENT

- RGBDSLAM with a Kinect V2 is an excellent low cost method of implementing a robust SLAM system but it still currently has some limitations.
- To combat the broken link issue (Which is very common when the environment is too volatile, or when the robot is moving too fast for there to be enough overlap between subsequent frames) the proposed SLAM system fuses the data from the Kinect with data from an IMU. This makes for a more robust implementation.
- As for the 2D matching techniques, the SLAM system can use FPH based fast global registration. This is a very different technique compared to 2D matching. This is a purely 3D geometric way of matching two point clouds. Used in combination with other techniques.
- Regarding the backend, according to Juric et al just switching from G2O (by far the most popular backend) to GTSAM can result in some excellent improvements in terms of performance. Therefore this SLAM system will choose to go with the GTSAM backend instead of G2O.

Literature Survey

Paper	Authors	Description
Naudet-Collette, S., Melbouci, K., Gay-Bellile, V., Ait-Aider, O., & Dhome, M. (2021). Constrained RGBD-SLAM. Robotica, 39(2), 277-290. doi:10.1017/S0263574720000363	Sylvie Naudet-Collette, Kathia Melbouci, Vincent Gay-Bellile, Omar Ait-Aider and Michel Dhome	Uses partial knowledge to create a better map
Yan, L.; Hu, X.; Zhao, L.; Chen, Y.; Wei, P.; Xie, H. DGS-SLAM: A Fast and Robust RGBD SLAM in Dynamic Environments Combined by Geometric and Semantic Information. Remote Sens. 2022, 14, 795. https://doi.org/10.3390/rs14030795	Li Yan, Xiao Hu,Leyang Zhao,Yu Chen,Pengcheng Wei and Hong Xie	Robust system for dynamic environments where the static environment assumption does not hold well
Y. Zhou, Y. Wang, F. Poiesi, Q. Qin and Y. Wan, "Loop Closure Detection Using Local 3D Deep Descriptors," in IEEE Robotics and Automation Letters, vol. 7, no. 3, pp. 6335-6342, July 2022, doi: 10.1109/LRA.2022.3156940.	Youjie Zhou; Yiming Wang; Fabio Poiesi; Qi Qin; Yi Wan	Uses 3D matching instead of conventional 2D Image matching for loop closure detection

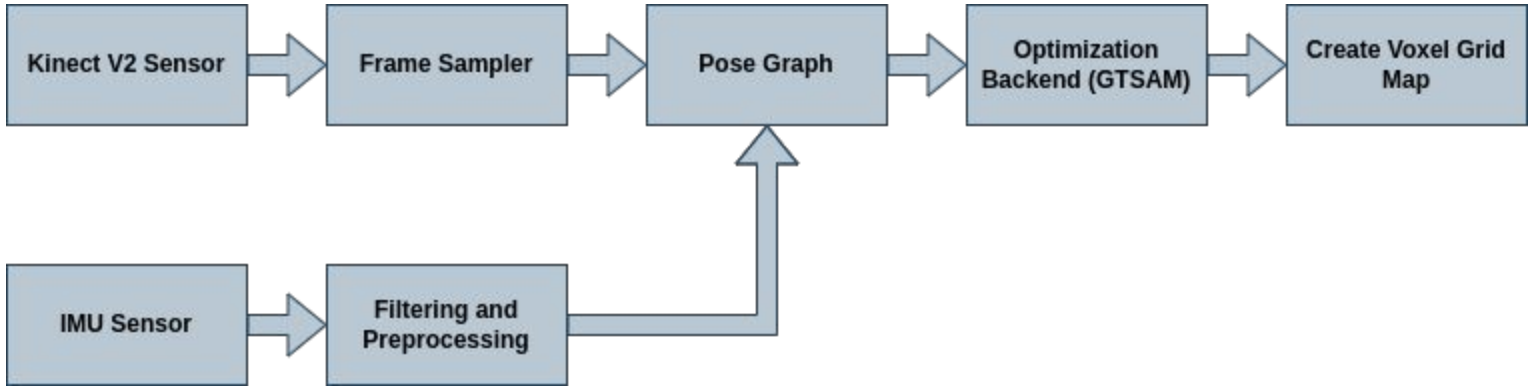
Literature Survey

Yuan, Jing, et al. “ORB-TEDM: An RGB-D SLAM Approach Fusing ORB Triangulation Estimates and Depth Measurements.” *IEEE Transactions on Instrumentation and Measurement*, vol. 71, 2022, pp. 1–15, 10.1109/tim.2022.3154800. Accessed 3 Feb. 2023.	Jing Yuan; Shuhao Zhu; Kaitao Tang; Qinxuan Sun	This implementation of SLAM fuses the 2D ORB matching method with the depth data obtained from the depth sensor in the Kinect V2 Camera.
D. Zhu, G. Xu, X. Wang, X. Liu and D. Tian, "PairCon-SLAM: Distributed, Online, and Real-Time RGBD-SLAM in Large Scenarios," in IEEE Transactions on Instrumentation and Measurement, vol. 70, pp. 1-14, 2021, Art no. 5019114, doi: 10.1109/TIM.2021.3116288.	Donglin Zhu; Guanghui Xu; Xiaoting Wang; Xiaogang Liu; Dewei Tian	Implements a multi system RGBDSLAM system over the network

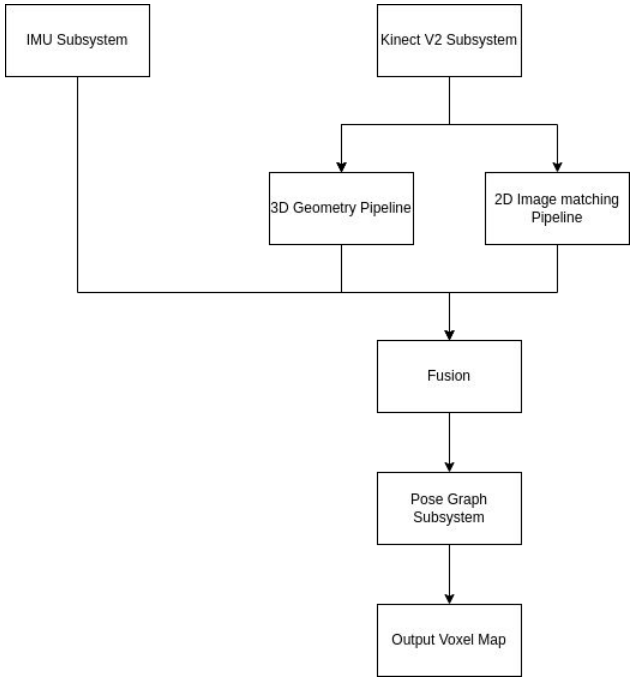
NOVELTY

- This version of SLAM uses FPFH descriptors to match between subsequent point clouds alongside using 2D methods that are traditionally used. The 2D methods are there to augment the geometry methods
- The proposed SLAM system uses sensor fusion with an IMU to obtain better results than just visual odometry and loop closure.
- Compared to most RGBDSLAM implementations available, this implementation uses GTSAM rather than G2O. As can be seen from the literature review before, GTSAM performs much better than G2O in most cases and comes just after SE-Sync.
- The proposed SLAM system also use a Kinect V2 sensor, which is significantly better than the older Kinect V1 sensor generally used for these applications.

System/Architecture diagram (PROPOSED)



METHODOLOGY (FLOW Chart)



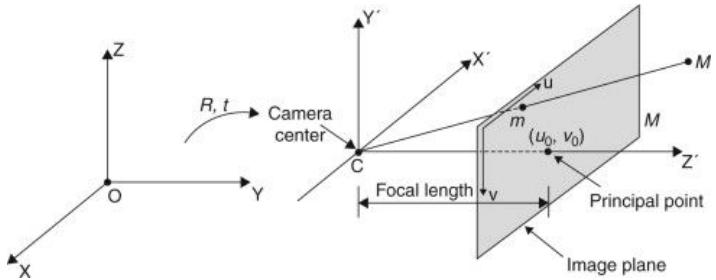
- As can be seen from the flow chart, the data from the kinect and the IMU are piped through different stages.
- Concentrating on the IMU first, it is first put through a digital filter to reduce noise. Once this is done, calibration is required, and for that the proposed SLAM rely on the visual odometry side of things, which occurs every 2-3 seconds.
- Once the IMU is calibrated, the IMU subsystem can then use a Kalman Filter or one of its variations to calculate a short term odometry.
- This odometry cannot be used for more than a few minutes because of the compounding errors of the IMU used (A cheap MEMS based sensor in the interest of keeping costs low).
- This short term odometry, nonetheless, can still be incredibly useful and can be used to make sure that in case of broken visual odometry the system has some idea of where itis. Even if the visual odometry is not broken, it can use both sensors to improve the accuracy further.

Methodology - Continued

- Now coming to the Kinect sensor, it's a combination RGB camera and an IR camera where the latter has a ToF sensor which gives us accurate distance measurements.
- First there is a frame listener which fetches both an RGB and a depth frame. The frames from this are preprocessed for noise reduction and removal of distortion. Following this, the frames are combined together to form an RGBD image.
- Once the system has an RGBD Image as well as the intrinsic parameters of the camera, it can convert it to a point cloud. This point cloud is then downsampled to increase the speed of computation and lower memory usage.
- Following this, the new frame is compared to a buffer of 12 point clouds that were uniformly sampled from the start of the program and the most recent previous frame. Matches result in a loop closure and odometry respectively. They are matched with geometry as well as image based features.
- Of course, it's possible for the comparison to fail and if this happens for the odometry then it loses track of where it is. This can be remedied by injecting data from the IMU so that when the data from the Kinect becomes highly uncertain, the IMU odometry will correct it.
- All of these comparisons are put in a pose graph which every n iterations is optimized. This optimized map is then used to construct an occupancy grid map of the environment.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

2D Image Coordinates | Intrinsic properties (Optical Centre, scaling) | Extrinsic properties (Camera Rotation and translation) | 3D World Coordinates

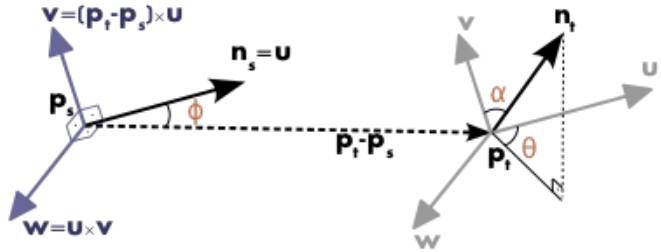


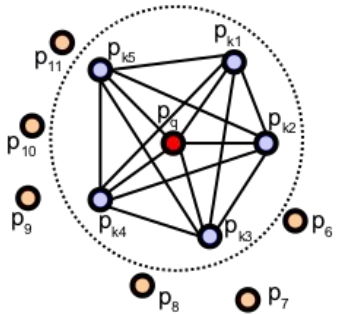
3D Matching Subsystem

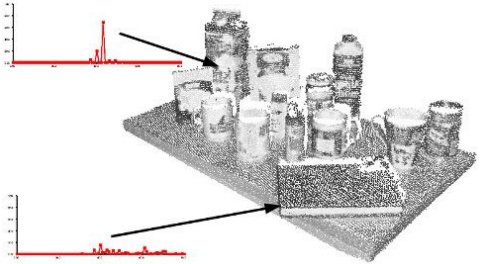
- The 3D matching subsystem compares point clouds using FPFH (Fast Point Feature Histogram) feature descriptors.
- These are similar to SIFT/SURF with 2D images but work in a 3D space.
- The matching of these features are generally more accurate when there is more geometric detail than color, which is very common.
- Once matched, the inliers are selected using RANSAC.
- These inliers are then used to calculate the transformation between two point clouds.
- For the loop closure detection checks have to be put into place to prevent false positives.

Point Feature Histograms

- Point feature histograms are a method of describing features within a point cloud.
- It is calculated in a fairly simple manner. In the K-neighborhood of any point, ie, the set of all points with a distance r < K from the point of interest, select a pair.
- Find the difference between the normals of this pair by considering a relative space as shown in the figure.
- This difference is defined by 4 parameters - α, θ, ϕ and d.
- This is done for all possible pairs in the K-neighborhood.
- The entire range of values that the 4 parameters can take is then converted into discrete bins.
- Every pair that has its values falling in any of the bins causes a count to increment for that bin.
- FPFH is a faster implementation.

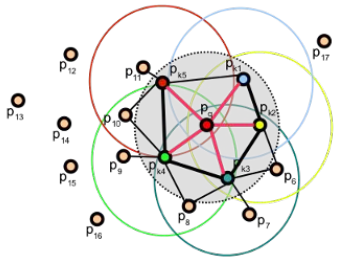






Fast Point Feature Histograms

- FPFHs are used in real time applications where speed is necessary.
- It is a simplified version of the more general PFH.
- In order to calculate the FPFH, first define a simple version of the PFH - one that does not calculate the histogram for all pairs but instead only between the query point and its neighbors. It will be called SPFH
- FPFH is then calculated by adding the SPFH of the query point to the weighted sum of all SPFHs of the points in the K-neighborhood
- It does not capture as much as PFHs, but still does well enough to perform decently.



$$FPFH(\boldsymbol{p}_q) = SPFH(\boldsymbol{p}_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{\omega_i}\cdot SPFH(\boldsymbol{p}_i)$$

FPFH vs Prior method

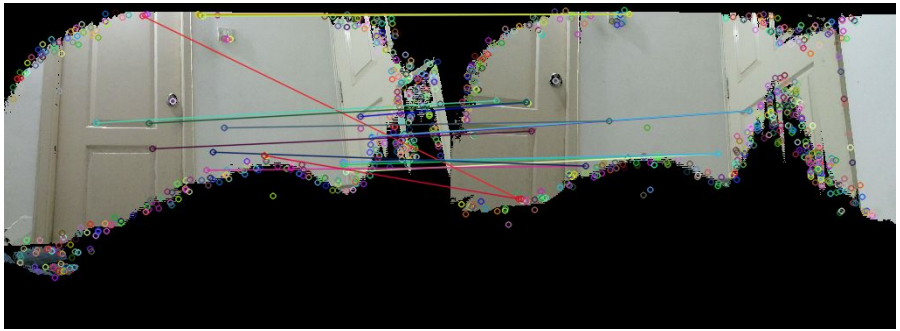
FPFH	**2D Matching**
● Depends on geometry ● Operates on 3D data - much more information ● Fairly fast ● Immune to regular texture, results are accurate ● As long as the geometry is varied, this method provides for a lower error	● Depends on color data ● Operates on 2D data - one dimension of data less ● Faster ● Results are thrown off by regular texture ● This method only provides for a lower error if the colors are irregular

2D Matching Subsystem

- The 2D matching subsystem uses well traditional image feature matching techniques.
- Once these features are detected, like in the 3D case, **RANSAC** is used to select a model with the most inliers.
- Once this model has been selected, the transform can be calculated between the two point clouds using **Umeyama's algorithm.**
- These are more accurate when there is more unique color and textural data than geometric (For example along a corridor which is from a geometric viewpoint uniform, but may have photos and smudges on the wall that work better with image based matching).
- Similar to the 3D method, loop closures should have checks in place to prevent false positives.

2D feature detection methods

- Multiple different kinds of methods exist for detecting features in 2D images
- Most used ones are - SIFT, SURF, ORB and AKAZE
- Any of the above can be used and all perform well
- SIFT and SURF are patented and thus cannot be used freely
- ORB and AKAZE are free to use as one pleases
- ORB is much faster than AKAZE
- AKAZE results in a lower number of outliers compared to ORB
- The current code runs on SURF descriptors

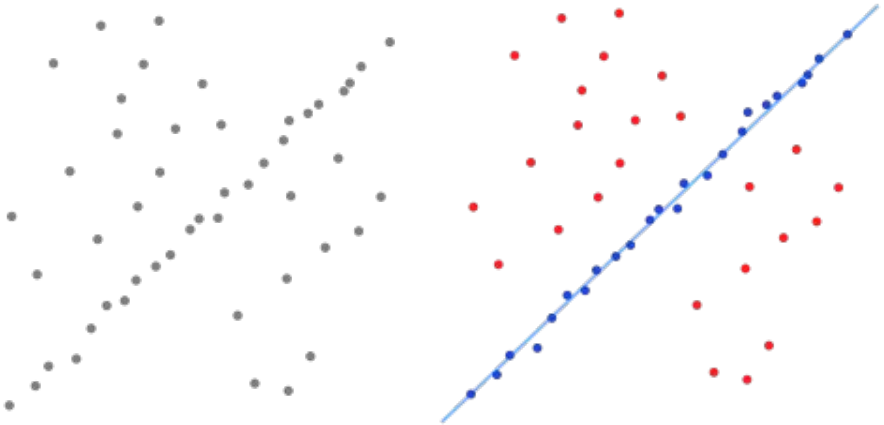


2D Matching - Choosing a model

- In the 2D method, a model is chosen so as to compute transforms between two frames
- To do this, any three matches features are selected
- Since the pair-wise euclidean distances are about equal in both cases, we can say that a valid model is one that passes this test
- This allows us to efficiently filter out all the invalid models much earlier making RANSAC much faster

2D Matching - RANSAC

- RANSAC is an algorithm used to efficiently separate inliers from outliers
- The basic idea of RANSAC is to choose a model that results in the most number of outliers
- This is done by randomly choosing a model from the given dataset, and then finding the number of inliers
- This is continued till a certain number of inliers have been reached or for a certain number of iterations
- The model with the highest number of inliers is then used to model the data and separate the outliers



$$N = \frac{\log(1-p)}{\log(1-(1-\varepsilon)^s)}$$

- s is the number of points from which the model can be instantiated
- ε is the percentage of outliers in the data
- p is the requested probability of success

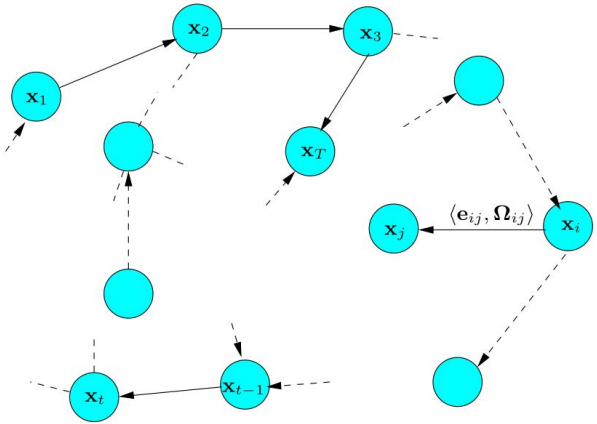
- Example: $p = 99\%$, $s = 5$, $\varepsilon = 50\% \Rightarrow N = 145$

Pose Graph SLAM

- The proposed SLAM system follows the Pose Graph SLAM technique. The idea is to record a series of poses onto a graph, with odometry connecting subsequent poses.
- Edges connecting two poses are also recorded when the same environment is revisited. These are called loop closures.
- Loop closures are highly important. False positives are highly detrimental.
- The total error is calculated from all the edges.
- The overall graph is then optimized to lower the total error. This is usually carried out by the Gauss-Newton or Levenberg–Marquardt algorithm.
- The output is a graph that will match reality closer with a total error that is much lower.

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathbf{F}_{ij}},$$

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \mathbf{F}(\mathbf{x}).$$



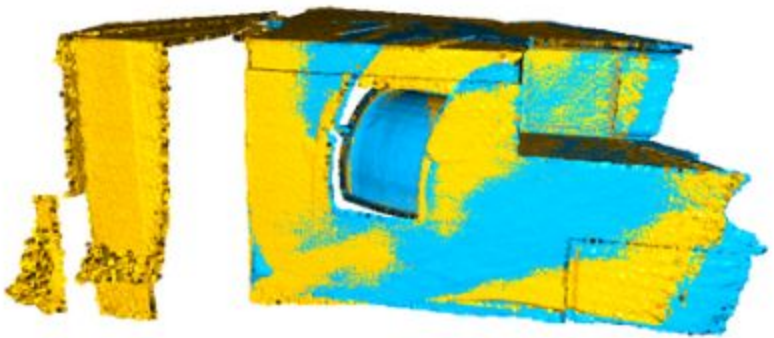
The role of the IMU

- The SLAM system proposed so far uses both 3D geometry and 2D image matching to improve overall accuracy and reduce false positives.
- Still, there are cases where there are no matches at all, which means that the odometry will be broken.
- In practice this is handled with a constant velocity model in most approaches. Some will use a constant and some will calculate it based on prior movement data.
- To remedy this, the proposed SLAM system will use an IMU and fuse the data with the other 2 pipelines to implement a more robust system than currently available.

Point Plane ICP (Algorithm from Review 1)

- Point-plane ICP is a variant of the ICP algorithm used for aligning point clouds.
- It utilizes the normal vectors of the points to estimate the plane of the surface.
- The algorithm aims to minimize the distances between the corresponding points and their respective planes.
- This approach can handle non-rigid transformations and partial overlap between point clouds better than traditional ICP.
- The algorithm involves iteratively computing the transformation that minimizes the distance between the point clouds, using the estimated plane normals to improve alignment accuracy.
- The process continues until convergence is reached, at which point the two point clouds are considered aligned.

$$E(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left((\mathbf{p} - \mathbf{T}\mathbf{q}) \cdot \mathbf{n}_\mathbf{p}\right)^2$$



Colored ICP (Current Algorithm)

- Colored ICP is a variant of the Iterative Closest Point (ICP) algorithm used for aligning point clouds.
- It uses both point coordinates and color information to find correspondences between points in two or more point clouds.
- The algorithm aims to minimize the distances between corresponding points in the two point clouds, taking into account both spatial and color differences.
- Colored ICP can handle point clouds with color information, making it useful in applications where color plays a significant role, such as 3D scanning of objects with textured surfaces.
- The algorithm involves iteratively computing the transformation that minimizes the distance between the point clouds, using color information to refine the alignment.
- The process continues until convergence is reached, at which point the two point clouds are considered aligned.

$$E(\mathbf{T}) = (1-\delta)E_C(\mathbf{T}) + \delta E_G(\mathbf{T})$$

$$E_G(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left((\mathbf{p} - \mathbf{Tq}) \cdot \mathbf{n_p}\right)^2$$

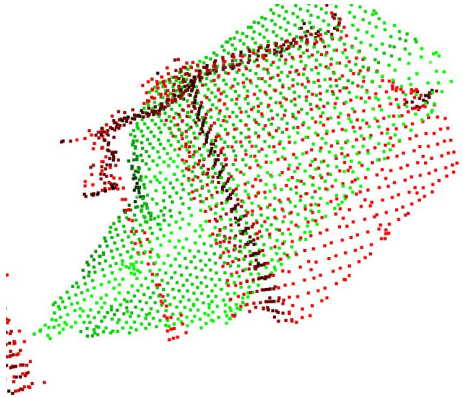
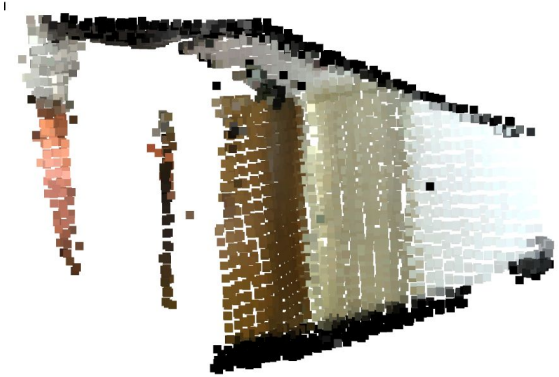
$$E_C(\mathbf{T}) = \sum_{(\mathbf{p},\mathbf{q})\in\mathcal{K}} \left(C_\mathbf{p}(\mathbf{f}(\mathbf{Tq})) - C(\mathbf{q})\right)^2$$

Colored ICP vs Point-Place ICP

- Colored ICP and point-plane ICP are two different variants of the popular ICP algorithm used for aligning point clouds. Colored ICP has several advantages over point-plane ICP
- Incorporating color information: Colored ICP takes advantage of the color information in addition to point coordinates. This can be useful in various applications where color plays a significant role, such as 3D scanning of objects with textured surfaces.
- Enhanced accuracy: The inclusion of color information can lead to more precise alignment since it provides additional data that can help to distinguish between similar-looking surfaces in the point clouds.
- Better visualization: Colored ICP produces visually appealing results by providing color-coded representations of the aligned point clouds, making it easier to interpret and analyze the data.
- The image on the left is point-plane, the one on the right is colored ICP.

$$E(\mathbf{T}) = (1-\delta)E_C(\mathbf{T}) + \delta E_G(\mathbf{T})$$

Colored ICP vs Point-Plane ICP

Point-Plane ICP	Colored ICP
	

Understanding the need for noise reduction

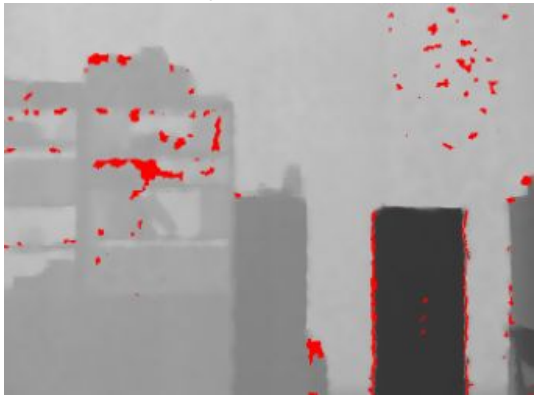
- The Kinect V2 supplies us with accurate depth maps
- However, the depth maps so supplied are riddled with noise
- There are two main sources of noise - pepper noise due to out of range captures and salt noise due to very bright lights
- These noise sources make the corresponding 3D representations (Point Clouds here) noisy as well. The z coordinate will often have ringing of some sorts.
- This causes major issues when trying to match two point clouds

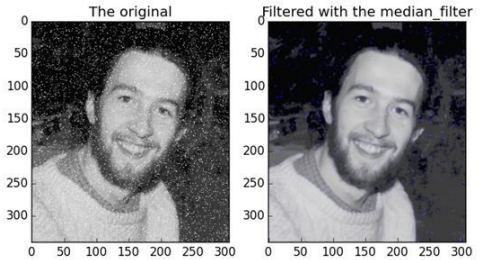
Filtering salt and pepper noise from the depth map

- There are multiple ways of filtering this map
- Median Filter: A median filter is a common method for removing salt and pepper noise from depth maps. The filter replaces each pixel with the median value of its surrounding pixels, which can smooth out the noise without significantly affecting the edges and details in the image.
- Gaussian Filter: A Gaussian filter can also be used to remove salt and pepper noise by blurring the image. The filter replaces each pixel with a weighted average of its surrounding pixels, with the weights determined by a Gaussian distribution.
- Bilateral Filter: A bilateral filter is a nonlinear filter that can remove salt and pepper noise while preserving the edges in the depth map. The filter takes into account both the spatial distance and the difference in intensity between neighboring pixels.
- Morphological Operations: Morphological operations, such as erosion and dilation, can be used to remove salt and pepper noise from a depth map. These operations work by smoothing the image and removing isolated pixels or small clusters of pixels.

Median Filtering

- For phase 2, we will be using median filters with some basic peak detection for salt and pepper noise reduction
- This reduction also results in a slight decrease in depth map resolution - since it is essentially a blur
- However, the output point clouds will be much cleaner and will not result in broken edges - which is far more important
- The image below on the left is the noisy depth map - the parts colored in red are noise
- The image on the right is the median filtered depth map. The salt and pepper noise have been effectively eliminated

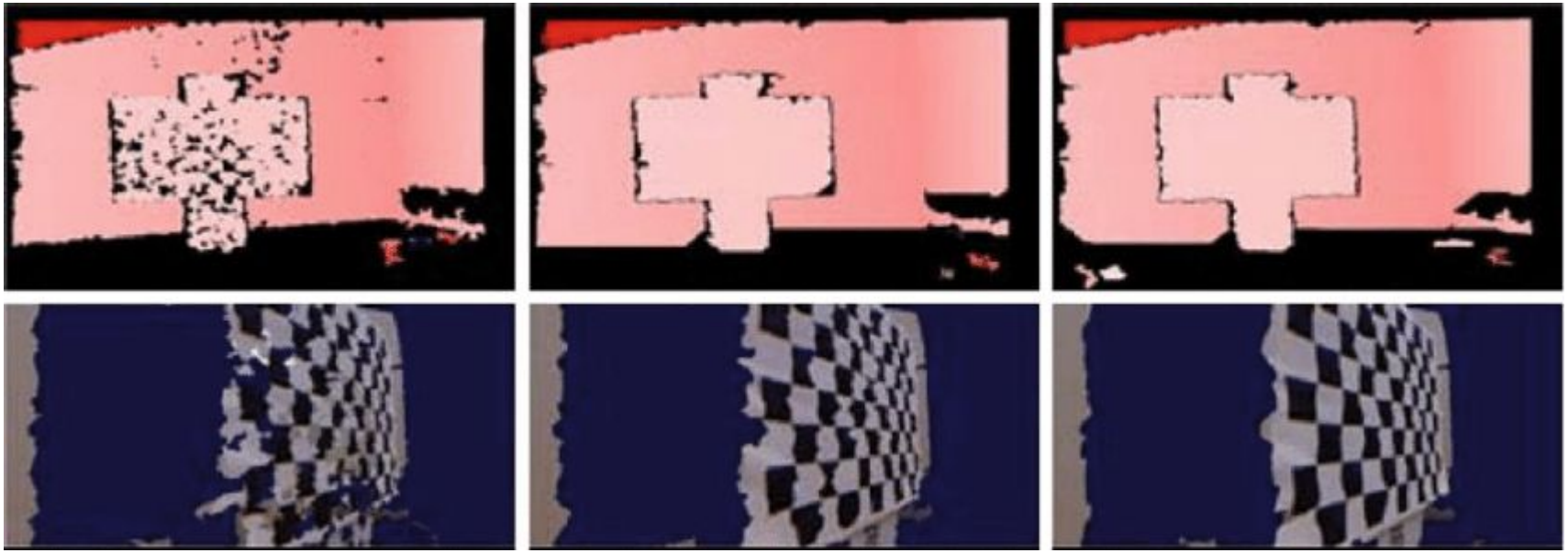




Issues of salt and pepper noise with point clouds

- Inaccurate depth measurements: Salt and pepper noise can cause individual pixels or groups of pixels to have inaccurate depth values. This can result in a significant distortion of the corresponding point clouds, leading to misalignment and decreased accuracy.
- Missing or erroneous points: Salt and pepper noise can cause individual pixels or groups of pixels to be misinterpreted or lost altogether, resulting in missing or erroneous points in the corresponding point clouds.
- Decreased precision: The presence of salt and pepper noise in the depth images can lead to a decrease in the precision of the corresponding point clouds. This is because the noise can cause a loss of detail and accuracy in the point clouds, making it more difficult to distinguish between different surfaces and features.
- Increased processing time: When processing depth images with salt and pepper noise, additional processing steps may be required to filter out the noise before the corresponding point clouds can be generated. This can increase processing time and computational requirements, potentially slowing down the overall system.

Effect of filtering on point clouds produced

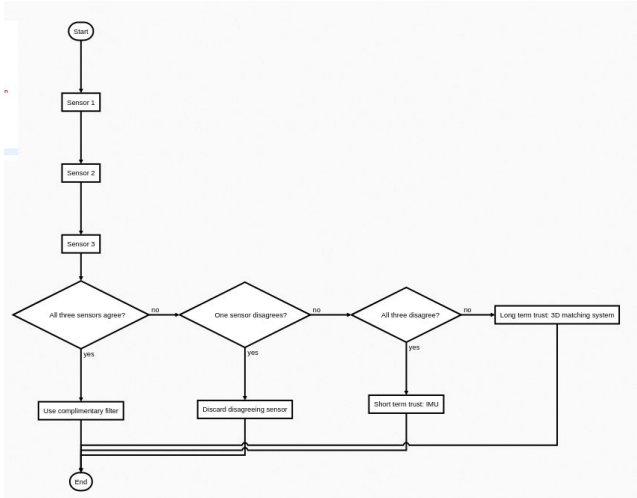


Output Fusion

- We have output from all 3 methods that need to be put together
- Sometimes we need to choose between the three methods in case there are any disagreements
- We could use traditional methods of sensor fusion like EKFs but unfortunately we have massively different sample rates
- IMU Sample rate : 1000Hz
- 2D Matching Sample rate : 3Hz
- 3D Matching Sample rate : 0.3Hz
- We need to keep that in mind when combining the output

Output Fusion - Trust

- We need to trust the appropriate sensor at the appropriate time
- If all 3 sensors agree to some extent we can simply use a gh filter to choose an output that is between all 3 with weights that are chosen according to each method’s accuracy
- If one of the outputs disagree, what do we do?
- If the other two agree reasonably well we discard the one that does not agree and move on
- If all three disagree, then we reach a conundrum - which method should we trust? On the short term, the IMU is the most trusted, partially due to its high sample rate. On the long term, assuming enough feature points are available, the 3D matching system fares better over the 2D matching system



Improving Performance

- The performance when the project was completed was about 30s of processing per frame
- It is at about 1.5-2.5 seconds per frame
- The massive speed increases required a lot of work for optimization
- Switching from Python to C++ for the 2D matching subsystem helped massively
- Down sampling of the input point cloud was made more aggressive
- The 3D feature point matching was separated into two phases - rough and precise alignment. Rough alignment convergence distance was kept much larger than before to help with performance
- RANSAC for 2D matching was rewritten from scratch for improvements

HARDWARE/SOFTWARE SPECIFICATION

Hardware Used:

- Kinect V2 Sensor
- Kinect V2 adapter for Windows
- MPU6050
- Raspberry Pi Pico
- 3D printer
- Filament

Software Used:

- Python
- Open3D (Point cloud computations and visualization)
- Freenect2 (Kinect 2 SDK)
- Numpy (General mathematical computations)
- Transforms3d (Converting between forms of SE(3) rotations)
- GTSAM (Pose Graph Optimization)
- OpenCV (Image processing)

RESEARCH PROGRESS

- Kinect V2 Only SLAM with FPFH has been implemented.
  - Kinect V2 Frame Listener has been implemented
  - Frame Preprocessing has been implemented
  - Uniform Frame Sampling has been implemented
  - Data serialization and deserialization for the pose graph has been implemented
  - Decomposition of transforms into translation and rotation has been added
  - Voxelization has been implemented
- G2O was replaced by GTSAM.
- Multiple edge types and related error models have been added and tweaked (Odometry, Loop Closure, Broken)
- One stage FPFH based registration was replaced for a 2 stage rough and precise registration (FPFH based global registration and ICP based local registration).
- Performance was improved from 6 seconds per frame to 2.5 seconds per frame
- IMU support has been implemented
- The point to place ICP algorithm has been replaced with colored ICP
- Median filtering has been implemented for the depth image
- Original quality point clouds are now stored in files and retrieved when necessary
- Localization support has been implemented
- Performance degradation from switching to the colored ICP algorithm has been improved to the previous status quo

OUTPUT - Odometry Only Scenario

Raw Output	Optimized Output	Voxel Map
		

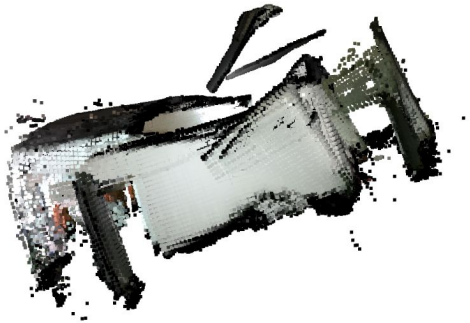
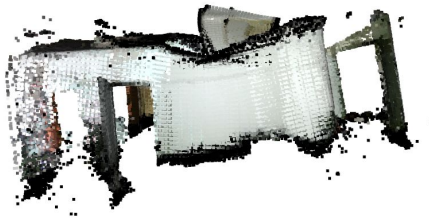
Output - Odometry, Loop Closure and Broken Odometry Scenario

Unoptimized Output	Optimized Output	Voxel Map
 		

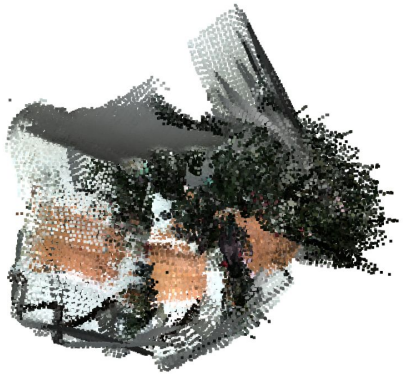
Output - Comparing 2D matching and 3D matching

Traditional Method	Three Way Method
	

Output - Odometry, Loop Closure and Broken Odometry 360 degree scenario

Unoptimized Output	Optimized Output	Voxel Map
		

Output - Three way approach

Unoptimized Point Cloud	Optimized Point Cloud	Voxel Grid Map
		

Output - Parameters

Parameter	**Traditional Method**	**Three Way Method**
Average time to process one frame	321ms	2.21s
Average RMSE over multiple scenarios	0.212	0.019
Average chance of failing SLAM catastrophically over multiple scenarios	48%	14%

WORK PLAN (TIMELINE)

Till 1st review	Till 2nd review	Till 3rd review
● Kinect V2 Only SLAM has been implemented. ● G2O was replaced by GTSAM. ● Multiple edge types and related error models have been added and tweaked (Odometry, Loop Closure, Broken) ● One stage FPFH based registration was replaced for a 2 stage rough and precise registration (FPFH based global registration and ICP based local registration). ● Performance was improved from 6 seconds per frame to 2.5 seconds per frame	● Switch over to colored ICP ● Add in partial IMU support ● Add more noise reduction for the depth image ● Add support for stored original quality point clouds ● Add localization visualization ● Refactor code base	● Add complete support for the IMU and combine both the IMU and Kinect readings ● Add traditional 2D based matching in C/C++ with RANSAC to augment the robustness ● (Maybe) Switch the 3D geometry matching to PROSAC ● (Maybe) Convert most math code to CUDA

BUDGET

Total Budget: < 7K (Variable)

- Kinect V2: Rs 2500
- Kinect V2 Adapter: Rs 1500
- Raspberry Pi Pico: Rs 350
- MPU6050: Rs 150
- Filament 1KG PLA: Rs 1000

APPLICATION

Simultaneous Localization and Mapping (SLAM) is a computer vision and robotics technique that enables real-time mapping and localization of environments. SLAM has a wide range of applications across various industries, including:

1. Robotics: SLAM is used for navigation and mapping by autonomous robots, allowing them to create maps of unknown environments and determine their location within them.
2. Augmented Reality (AR): SLAM is used in AR applications for real-time mapping and tracking of the physical world, providing a more immersive and interactive experience.
3. Self-driving Cars: SLAM is used in self-driving cars for real-time mapping and localization in dynamic environments, enabling safe and efficient navigation.
4. Drones: SLAM is used for navigation and mapping in aerial environments by drones, helping them to fly safely and efficiently in unknown environments.
5. Industrial Inspection: SLAM is used for mapping and inspection of large-scale industrial facilities, allowing for more efficient and thorough inspections.
6. Gaming: SLAM is used in gaming to create realistic and immersive gaming environments, providing a more engaging experience for players.
7. Agriculture: SLAM is used for mapping and monitoring of crops and fields, allowing for more efficient and effective agricultural practices.
8. Geography: SLAM is used for mapping and surveying of geographical areas, providing accurate and detailed maps for various applications.

In conclusion, SLAM has numerous applications across various industries, including robotics, AR, self-driving cars, drones, industrial inspection, gaming, agriculture, and geography. Its ability to simultaneously localize and map environments in real-time provides valuable insights and benefits for these industries.

FUTURE WORK

In terms of things I would like to do after the 3rd review:

- Look into RANSAC alternatives like MLESAC and PROSAC
- Look into implementing plan detection and segmentation
- Look into reimplementing most numerical calculations on the GPU
- Look into using multiple threads on the CPU to improve performance
- Look into augmenting the SLAM process with a motion based parallax based depth estimation

REFERENCES

- Juric, Andela, et al. “A Comparison of Graph Optimization Approaches for Pose Estimation in Slam.” *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*, 2021, https://doi.org/10.23919/mipro52101.2021.9596721.
- Khan, Muhammad Shahzad, et al. “Multi-Sensor Slam for Efficient Navigation of a Mobile Robot.” *2021 4th International Conference on Computing & Information Sciences (ICCIS)*, 2021, https://doi.org/10.1109/iccis54243.2021.9676374.
- Gong, Xieping, et al. “Review of Visual Slam.” *Third International Conference on Artificial Intelligence and Electromechanical Automation (AIEA 2022)*, 2022, https://doi.org/10.1117/12.2646825.
- Xin, Guan-xi, et al. “A RGBD Slam Algorithm Combining Orb with PROSAC for Indoor Mobile Robot.” *2015 4th International Conference on Computer Science and Network Technology (ICCSNT)*, 2015, https://doi.org/10.1109/iccsnt.2015.7490710.
- Ligocki, Adam, and Aleš Jelínek. “Fusing the RGBD Slam with Wheel Odometry.” *IFAC-PapersOnLine*, vol. 52, no. 27, 2019, pp. 7–12., https://doi.org/10.1016/j.ifacol.2019.12.724.
- Yuan, Jing, et al. “ORB-TEDM: An RGB-D SLAM Approach Fusing ORB Triangulation Estimates and Depth Measurements.” *IEEE Transactions on Instrumentation and Measurement*, vol. 71, 2022, pp. 1–15, 10.1109/tim.2022.3154800. Accessed 3 Feb. 2023.
- Endres, Felix, et al. “An Evaluation of the RGB-D SLAM System.” *2012 IEEE International Conference on Robotics and Automation*, May 2012, https://doi.org/10.1109/icra.2012.6225199. Accessed 3 Feb. 2023.
- Grisetti, G., et al. “A Tutorial on Graph-Based SLAM.” IEEE Intelligent Transportation Systems Magazine, vol. 2, no. 4, 2010, pp. 31–43, https://doi.org/10.1109/mits.2010.939925.