Dynamic SLAM using Landscape Theory of Aggregation*

Akshit Gandhi[1], Avinash Hemaeshwara Raju[1] and Parv Parkhiya[1]
{akgandhi, ahemaesh, pparkhiy}@andrew.cmu.edu

***Abstract*— Simultaneous Localization and Mapping (SLAM) is an essential part of any mobile robot. While the current state-of-the-art approach solves the problem in a static environment reasonably well, the performance in a dynamic environment is hit or miss. The traditional SLAM method assumes that the number of measurements from static objects would be large enough to dominate measurements from dynamic objects. We advocate for the explicit filtering of measurements from dynamic objects for better localization and mapping performance. We use the landscape theory of aggregation method to form an optimization problem. We observe the measurements for a time-window and compute weights for the optimization. We perform gradient descent to minimize the energy to classify and filter out measurements from dynamic objects. Finally, using ROS's GMapping Package we show improved SLAM output.**

I. INTRODUCTION

The majority of the traditional SLAM problems have been addressed for static environments. The literature on addressing the SLAM problem in dynamic environments is quite limited. With the advent of self-driving cars and factory automation robots, the impact of robot localization in the dynamic environment has become more profound. In this project, we implemented the existing Dynamic SLAM approach described in [1] and improved upon this approach.

The proposed approach utilizes theory originally proposed in a political science journal titled Landscape Theory of Aggregation[2] of all places. While the traditional SLAM algorithm assumes that all the features are static and hopes that the high number of static features will counter the misleading measurements provided by dynamic features, Landscape Theory of Aggregation will allow us to classify the features into the static and the dynamic set. This allows the explicit use of just stationary features in our SLAM pipeline resulting in the improved map and localization output. Also, the structure of the proposed approach allows for integration with any existing SLAM system to get improved results in dynamic environments.

The novelty of the paper is in classifying features into static and dynamic classes. The classification is performed by finding a least-squares optimization solution to the set of correspondences that are to be classified as static or dynamic. Also, the proposed approach requires traditional SLAM once classification is performed to get robot trajectory and map. We used LIDAR, Odometry, and GMapping to work along with the binary classifier to get final SLAM output.

*This work was done as a part of 16833-A Robot Localization And Mapping Course Project Fall-2019 at CMU by Team SLAM Dunk

*Source code can be found at `https://github.com/ahemaesh/Dynamic_SLAM_using_Landscape_Theory_of_Aggregation`

[1]Akshit Gandhi, Avinash Hemaeshwara Raju and Parv Parkhiya are MRSD students at Carnegie Mellon University, Pittsburgh

II. LITERATURE REVIEW

Dynamic SLAM problem is one of the not fully solved critical problems today. While traditionally, SLAM in the dynamic environment has not been explored in-depth, many recent papers have shown promising results using different approaches. The most popular approach, particularly for visual SLAM, is to use semantic segmentation using deep convolutional neural networks. [3][4] CNNs have shown great generalization when trained on the large dataset.

Semantics-based approaches filter the measurement that when projected on the image plane belongs to objects that are likely to be dynamic. For example, Cars, Pedestrians are likely to be moving in the scene and hence measurements corresponding to them are used to filter out. However, we might be losing good features as cars can be stationary in the scene. Also, training these deep semantics networks takes a large amount of data and time and are prone to failure in unexpected cases.

Another approach of solving SLAM in a dynamic environment is to assume some kind of motion model prior on the dynamic objects in the scene. [5][6] While they show promising results, fundamentally they are limited in terms of scalability to various diverse types of dynamic environment. An ideal approach should identify the dynamic feature without worrying about their semantics and prior assumption on its motion model.

The method proposed in [1] is one such method, that uses landscape theory of aggregation to differentiate between dynamic and static features/landmarks without any prior knowledge. While the result may not as impressive as semantic SLAM or SLAM with a prior motion model, it makes up for it with generalized wide applicability instead of the narrow focus of other approaches.

III. THEORY

A background on Landscape Theory of Aggregation in terms of political science as described in paper[2]

"Aggregation means the organization of elements of a system into patterns that tend to put highly compatible elements together and less compatible elements apart. Landscape theory predicts how aggregation will lead to alignments among actors (such as nations), whose leaders are myopic in their assessments and incremental in their actions. "

The predicted configurations are based upon the attempts of actors to minimize their frustration based upon their

pairwise compatibility. For the implementation of Landscape Theory of Aggregation in SLAM, we see that the frustration score in a set of only static correspondences or in a set of only dynamic correspondences is the least. We calculate a propensity score which gives us the influences of one correspondence on another correspondence. Also, this influence is a measure of the weight of the correspondence, which is analogous to small nation-big nation theory – *"a source of conflict with a small country is not as important for determining alignments as an equivalent source of conflict with a large country"*. Based on the propensity score, our classifier tries to achieve the minimum energy state by accurately classifying correspondences into static or dynamic classes. Once we have classified correspondences, we omit dynamic correspondences and perform SLAM using the static correspondences.

We minimize the following cost function for classification as proposed in [1] which essentially summation of the frustration of each landmark with every other landmark:

$$E(G) = \frac{1}{2}\sum_{i=1}^{n}\sum_{i\neq j=1}^{n}\frac{S_j P_{ij}}{S_i}\sum_{c=1}^{m}(u_{ic}-u_{jc})^2 \tag{1}$$

where,
s_i - weight of the i^{th} feature point
u_{ic} - boolean if i^{th} feature is in c^{th} class
p_{ij} - propensity score between feature point i and j
u_{jc} - boolean if j^{th} feature is in c^{th} class
m - number of class
n - total number of features
k - current timestamp
t - window size
l_{ik} - pose of i^{th} feature at k^{th} time in global frame
σ_x^2 - covariance of x

The size/weight is defined as

$$s_i = \frac{1}{\sigma_x^2} \tag{2}$$

The propensity or compatibility score between two landmarks is defined as

$$p_{ij} = 1 - \frac{1}{t}\sum_{a=1}^{t}||d_{A_i}^{(k)(k+a)} - d_{A_j}^{(k)(k+a)}|| \tag{3}$$

The distance between same landmarks across time is defined as

$$d_{A_i}^{(k)(k+a)} = ||l_i^k - l_i^{k+a}|| \tag{4}$$

IV. DATASET

We initially tested our implementation on the WeanHall dataset Robotdata1.log consisting of odometry and laser-scans that was provided as part of our in-class particle filter assignment. We later collected our own data-set using a SICK LIDAR and an Intel Real-Sense camera mounted on a Clearpath Husky robot as seen in fig 1. In our dataset

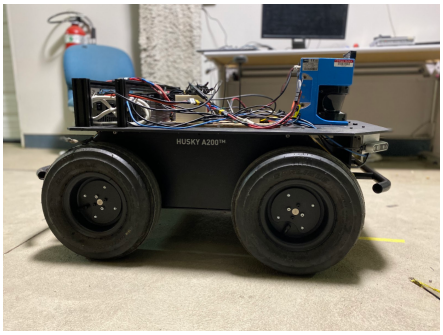

Fig. 1. Husky with SICK LIDAR used for data collection

collection, we recorded the wheel odometry data of the Husky, the laser-scans of the SICK LIDAR and the RGB image frames of the Intel Real-Sense for visualization in a ROSbag. We also recorded the transforms(tf-tree) from the Husky base-link to SICK LIDAR and base-link to Intel Real-Sense camera. We mimicked a dynamic environment by walking in front of a moving Husky.

V. IMPLEMENTATION

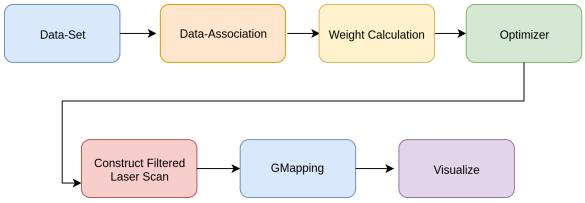

Fig. 2. Final output from the optimizer

We implemented the complete pipeline from getting raw sensor data to getting final SLAM output as shown in figure 2. A brief overview of the pipeline can be found below and is followed by a detailed description of specific parts.

Our dynamic SLAM module takes input in a specific format. We had dataset in 2 different formats, so we wrote a logger module that can listen to ROS topics and create a text file that our module can then process. Once data is loaded, we first perform data association. After that, we compute weights for our optimization problem as mention in the theory section. We run our custom optimizer to get the final classification output of dynamic and static. We construct a new laserscan message with laserscan points from dynamic objects filtered out. Finally, we provide all the necessary interface (tf, odometry, laserscan) to the ROS GMapping package and visualize the output in the RVIZ.

A. Data Association

Data association is a vital part of the pipeline as the weights for the optimization are computed based on how associated laserscan points vary in a common reference frame. We use the nearest neighbor approach for data association. It's important to note here that the particular association done here is not used later in the SLAM pipeline. SLAM pipeline computes its own association based on the occupancy grid

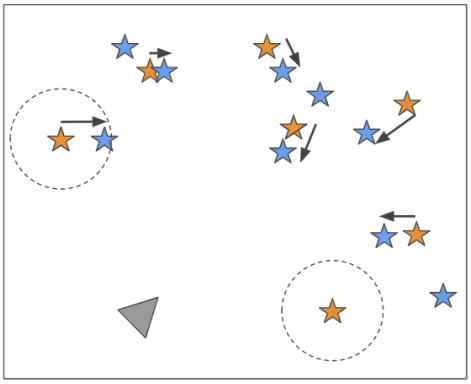

Fig. 3. Nearest Neighbor Association

map it builds in the global reference frame and is separate from common local frame association done here which is used to filter out dynamic points.

Common local frame association is done in the window size of 20 time-steps. Laser scan points are transformed in the reference frame of the first time-step using pure odometry data. Since the local drift of odometry data in a time-window is limited, a simple approach of the nearest neighbor works reasonably well for our purpose. As shown in figure 3, laserscans at two different time-step (shown in yellow and blue) are associated based on Euclidean distance. We also use reject associations which are more than a certain threshold radius. Laserscan points that find association throughout all the time-steps in the window are kept for classification and rest are marked as dynamic.

B. Ceres Based Optimizer

Our first attempt to solve the optimization problem was using Ceres-Solver library[7]. Ceres is a non-linear least-square library written in C++. For the particular optimization of classifying into dynamic and static, we need an optimized parameter to be a binary variable. But Ceres (most optimization library for that matter) only supports continuous variables.

We came with additional cost functions to force the continuous variable into 2 discrete states. Our first choice for two discrete states was 0 and 1 but when we tested our code, we realized that the optimization gradient step does not jump from 0 to 1 since they are far apart. When we choose the two states to be 0 and 0.001, the optimization was able to freely jump from 0 to 0.001 and vice-versa to minimize the cost. This particular binary constraint was added using equation (5).

$$cost : w_1 * \left\|u_i\right\|^2 + w_2 * \left\|u_i - 0.001\right\|^2 \tag{5}$$

The system was also prone to get stuck in local minima where all the variables were assigned the same dynamic or static class. To avoid that, we added additional cost to heavily penalize the assignment of all the variables to the same class as shown in equation (6).

$$cost : \left(\frac{w_1}{\sum_{i=1}^{n} \left|u_i\right|}\right)^2 + \left(\frac{w_2}{\sum_{i=1}^{n} \left|u_i - 0.001\right|}\right)^2 \tag{6}$$

The proposed approach worked on the toy problem we created with just 6 variables but when we tested the optimizer on real data with up to 180 variables, the optimization process became extremely slow to the point of being simply unacceptable. The reason we suspect is the following: Since there exist constraints between every pair of laserscan points, the "A" matrix would be fully dense. Another reason could be searching for a binary gradient on a continuous variable that will lead to many unnecessary gradient computations that are rejected. Therefore, we decided to write our own binary optimizer from scratch which works on explicit Boolean variables.

C. Custom Binary Optimizer

We implemented a custom binary optimizer from scratch that takes an initial guess of dynamic and static classification, minimizes energy function as mention in equation (1) and returns final state vector with dynamic and static classification.

Our binary optimizer idea is quite similar to the traditional gradient descent algorithm with the only difference being the presence of a binary gradient. Since optimizable parameters are Boolean variables and each can only take 2 values, a gradient step at any time can be of either flipping the state or keeping the state of the variable as it is. An example of one such gradient step iteration is shown in figure 4.

Var	u_1	u_2	u_3	...
State	0	0	1	...
ΔE (flipping state)	-45.3	664.2	-345.6	...
New State	1	0	0	...

Fig. 4. An iteration of binary optimizer

Computation of energy function E has a complexity of $O(n^2)$ where n is the number of laserscan points. But we are only interested in minimizing energy. Therefore while computing whether to flip a variable or not, we can marginalize out all the rest and just compute ΔE for flipping which has a complexity of $O(n)$.

We also need an initial guess for the optimization. We use threshold on $mean(d_{A_i})$ from equation (4) to decide the initial guess for the classification of each laserscan point. If the mean is less than the threshold, the initial guess for the laserscan point is considered static else dynamic.

VI. RESULTS

The primary objective of this project was to segment the static and dynamic laserscan points. We first started by running our pipeline on the raw laserscan data (as shown in fig: 5) to get an initial guess of the static and dynamic points (shown in fig:6) and this initialization is passed onto our binary optimizer to get the final segmentation between static and dynamic landmarks (shown in figure 7).

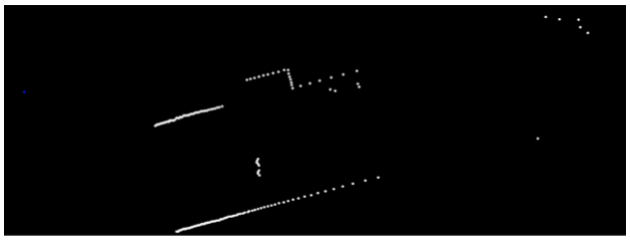

Fig. 5. Raw laserscan input to the pipeline

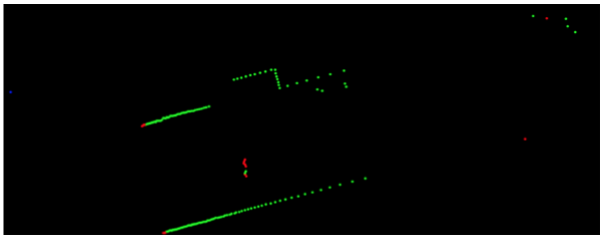

Fig. 6. Initial segmentation for the optimizer

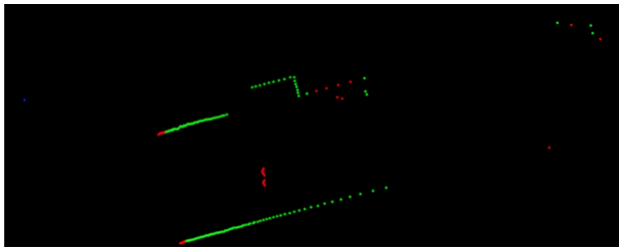

Fig. 7. Final output from the optimizer

The initial guess for the static v/s dynamic segmentation was obtained using the method described in the previous sections. The green points are the laserscan points which are classified as static and the red ones are the dynamic landmarks. The figures shown in 5, 6 & 7 are obtained from a sequence in the robotdata1.log given to us as a part of the Homework 1 on particle filter. The segmentation results clearly show that the person (walking in front of the robot) is a dynamic object and thus the laserscan points which correspond to its legs are classified as dynamic and the rest (which majorly constitute the structure like walls) are classified as static. Since the output images shown are for one timestep we visualized the static and dynamic segmentation over the complete log using RVIZ. The output of which is captured in a video `https://youtu.be/aJXhTr-SyeE`. If one observes the sequence in the video, a natural question spawns about the laserscan points closer and very far away from the LIDAR which are being classified as dynamic. This is because of our conservative approach. For the points that are very close to the LIDAR, there is a high chance that they may not be any associated laserscan point across multiple frames (which is used in our landscape theory) and hence they are classified as dynamic but this doesn't hurt the results because such points may already have been classified as static in the previous time-steps. Also, similar reasoning is true for the very far objects detected by the laserscan. These results can be seen in fig:8.

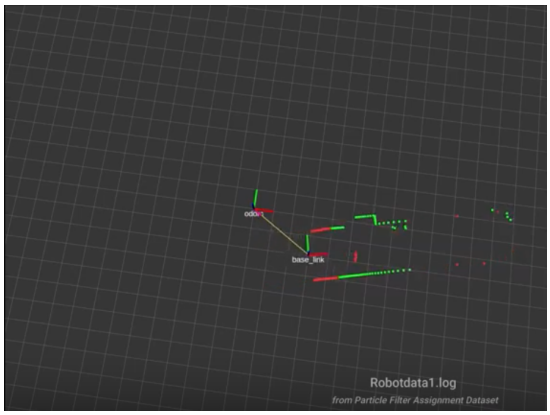

Fig. 8. Laserscan segmentation visualized in RViz

Finally, the output of the SLAM pipeline with and without Dynamic SLAM can be seen in the fig:9 and fig:10. The black points corresponding to trajectory from raw odometry data and red points corresponds to output trajectory by GMapping.

So as we can clearly see from the figure 9-10 and figure 11-12 that if we don't use dynamic slam, the occupancy map which is built has many holes/other artifacts which map to uncertain regions in the map and hence after filtering through our algorithm the map which is obtained is free of any such holes or artifacts. The full runs of SLAM for the robotdata1.log and our captured log can be seen in these videos: `https://youtu.be/Am5gR6rEqZA` & `https://youtu.be/_B_P1TQPGAs`

VII. CHALLENGES

The primary challenge we faced was with binary optimization where the output of our optimizer/classifier was constrained in static or dynamic (like 0 or 1). We used ceres solver which outputs a continuous variable. We tried different approaches to constrain the output of the ceres solver, but given the size of the search space, it was very slow at optimizing values for 180 laserscan points. The optimizer was very slow it took upwards of 60 mins. The optimizer works really well for sample inputs of size 6 but the time complexity scaled exponentially for the actual inputs.

The other challenge was with dataset integration with our pipeline. As our software was developed using the txt/log file provided in Assignment 1, it became necessary to modify

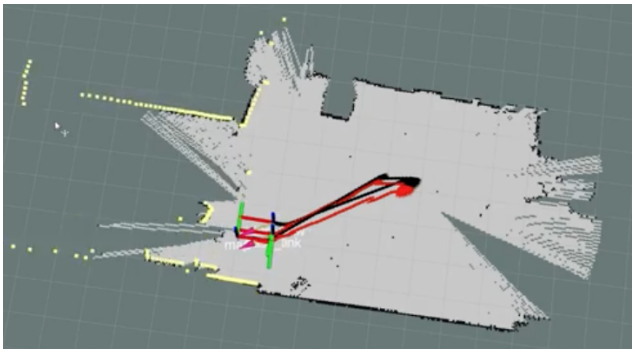

Fig. 9. Traditional SLAM on collected dataset

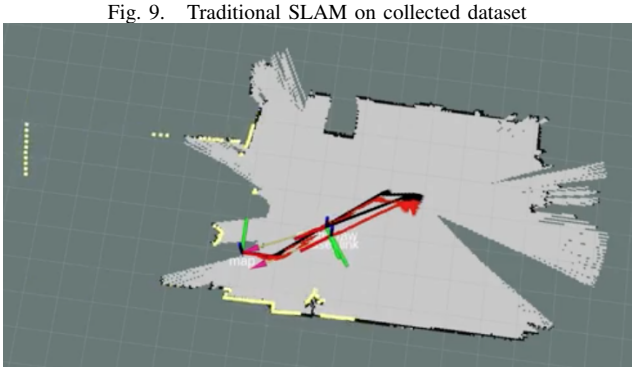

Fig. 10. Dynamic SLAM on collected dataset

our collected dataset into this format. The dataset was in the form of a ROSbag and thus we had to write a log writer tool that converted the ROSbag into a txt file as expected by the pipeline. After running the entire pipeline we found that the map being built was very very small in scale. After analyzing the logs we realized that the HW1 logs had the scale in centimeters and our ROSbag was collected on a metric scale. Finally, we found the issue and changed the log writer to account for the conversion. Debugging this issue took quite some time.

Other secondary challenges were in developing interfaces. Our classified output was a vector of 180 points but to feed it into a GMapping ROS package, we had to convert them into a laserscan message, publish the odometry and the odometry TF, etc. Also, we had to write our own visualizer tool from scratch to see the static and dynamic segmentation since we could not use something like Matplotlib as we were working with C++. The visualizer plots the points on an empty black image shows them to the user and also publishes the images on a ROS topic.

VIII. FUTURE WORK

In terms of future work, it would be interesting to explore more into the below regions:

- Testing the system on a diverse dataset with ground truth for quantitative analysis
- Adapting the system for 3D LIDAR / image-feature based landmarks
- Extending to multi-class classification where each non-static class corresponds to feature points belonging to a particular dynamic object. It could help with dynamic object segmentation and tracking also and thus is a very interesting idea. It would also enable modeling motion of various dynamic objects in the scene.
- Apart from these it would be interesting to see the qualitative results where the dynamic object occupies a major chunk of the laserscan data. We believe that the traditional method will fail miserably so it would be interesting to see the comparison.

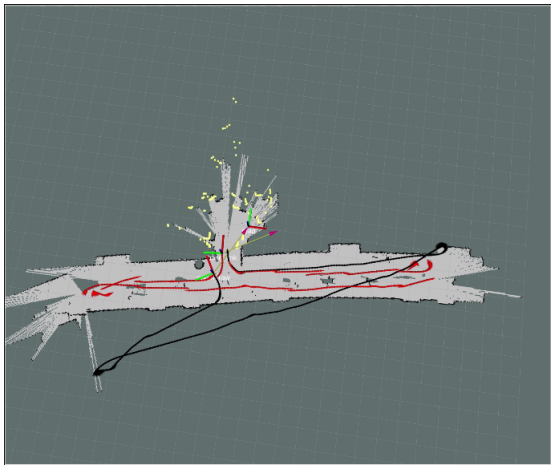

Fig. 11. Traditional SLAM on Robotdata1.log

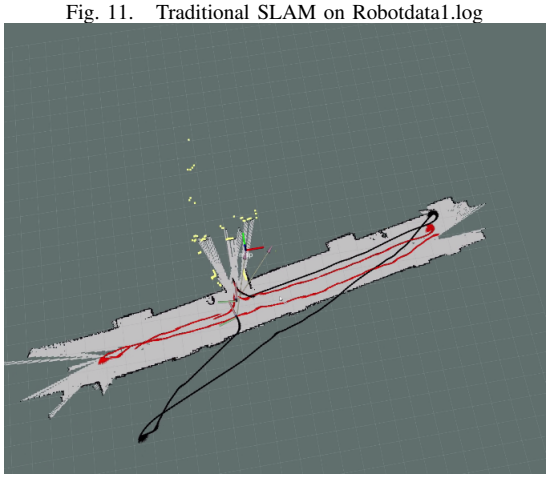

Fig. 12. Dynamic SLAM on Robotdata1.log

IX. CONCLUSIONS

To summarize as a part of this project:

- We successfully implemented the idea mentioned in the "Landscape Theory of Aggregation" paper and shown qualitative improvements in the generated map
- We designed our own binary optimizer
- A tool to convert ROS bags into the desired log formats, which makes it very generalize-able across different projects
- Custom visualizer for C++ which plots the static and dynamic laserscan points
- Collected our own dataset to demonstrate the working of the pipeline

REFERENCES

[1] Hua, C., Dou, L., Fang, H. et al. J. Cent. South Univ. (2016) : A novel algorithm for SLAM in dynamic environments using landscape theory of aggregation https://doi.org/10.1007/s11771-016-3320-9

[2] Robert Axelrod and D. Scott Bennett(1993): A Landscape Theory of Aggregation. British Journal of Political Scienc http://www-personal.umich.edu/~axe/Ax%20Bennett%20Landscape%20BJPS%201993.pdf

[3] Chao Yu, Zuxin Liu, Xinjun Liu, Fugui Xie, Yi Yang, Qi Wei, Qiao Fei. DS-SLAM: A Semantic Visual SLAM towards Dynamic Environments. https://arxiv.org/abs/1809.08379

[4] Linhui Xiao, Jinge Wang, Xiaosong Qiu, Zheng Rong, Xudong Zou: Dynamic-SLAM: Semantic monocular visual localization and mapping based on deep learning in dynamic environment https://www.sciencedirect.com/science/article/pii/S0921889018308029

[5] Huijing Zhao, Masaki Chiba, Ryosuke Shibasaki, Xiaowei Shao, Jinshi Cui, Hongbin Zha: SLAM in a Dynamic Large Outdoor Environment using a Laser Scanners https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=4543407

[6] Mina Henein, Gerard Kennedy, Robert Mahony and Viorela Ila: Exploiting Rigid Body Motion for SLAM in Dynamic Environments https://natanaso.github.io/rcw-icra18/assets/ref/ICRA-MRP18_paper_13.pdf

[7] Sameer Agarwal and Keir Mierle and Others: Ceres Solver http://ceres-solver.org/