Landing Site SLAM for UAVs

Rohit Garg[1] , Pulkit Goyal[2] and Siddhant Jain[3]

***Abstract*— In this work we develop a complete SLAM system for an Unmanned Aerial Vehicle to aid in evaluating potential landing sites for an autonomous landing. We propose a system to fuse Semi-Direct Visual Odometry with a non-linear ICP in a sequential manner to localize the agent in a map and simultaneously generate a dense map of its landing zone. The dense map generated is fed into a separate module that marks different segments as safe to land by evaluating geometric properties such as roughness and slope.**

I. INTRODUCTION

Safety is an important factor in drone autonomy. How does a drone guarantee safety for different scenarios that come up in autonomous flight. Landing is one such scenario that requires the drone to successfully evaluate potential zones that are safe to land on. A drone might have to land autonomously to conserve energy during a mission or might need evaluation of the landing site for dropping payloads. For all such scenarios having a dense map of the ground below, along with have accurate localization with respect to this map becomes critical. Not only is the accuracy important but for this system to be effective it is imperative that the mapping and localization is performed near real-time. In this work we construct and implement one system with the aforementioned objective and constraints for a drone with a stereo camera setup along with inertial sensors on-board.

II. RELATED WORK

Some prior work has been done on this problem, albeit, in the context of larger aerial vehicles, such as helicopters. Scherer et. al [7] presented a work for finding safe landing zones for a full-scale helicopter. They assess landing zones based on the ground conditions and the approach conditions. They use a tilting LIDAR for a dense reconstruction of the ground plane. In addition, they employ a GPS to register the scans. Once a reconstruction is achieved, they perform plane fitting to find the slope estimates of different landing zones. The plane fitting is also used to find the standard deviation along the z direction, which helps in calculating a measure of the roughness. We differ from this approach, as it is not practically feasible to add a LIDAR scanner on a small drone such as the one we are working with. Instead, we generate point clouds using a downward facing stereo setup.

Robotics Institute, Carnegie Mellon University

This work was done as a part of the course 16-833 Robot Localisation and Mapping

[1] rg1@andrew.cmu.edu
[2] pulkitg@andrew.cmu.edu
[3] siddhantjain@cmu.edu

ElasticFusion [8] is one method we considered for performing dense mapping. Similar to many conventional approaches, the work parallelizes tracking and mapping threads to do a dense reconstruction on an RGBD input. While the results are impressive, we couldn't incorporate this method in our system as the system requirements for real time performance were very high (32GB RAM, 3GB GPU memory). Also, this method is not the best for state estimation since its primary focus is on building a dense map and it is not entirely robust to fast camera motion.

It's still an open research problem to be able to do dense mapping while maintaining a precise state estimate on a resource limited computing platform like a micro-UAV. Which is why we decided to shift our focus towards a visual odometry based approach. Forster et. al.[3] devised an interesting system for visual odometry which combines the accuracy of a direct method with the speed of a feature based method. This semi-direct visual odometry method is particularly suited to our problem owing to speed and accuracy for cameras mounted on aerial vehicles. This fact has also been highlighted in the recent work by Delmerico and Scaramuzza [1] that does a performance evaluation of some of the state of the art Visual Inertial Odometry algorithms on real-time embedded platforms.

III. BACKGROUND

Presented below is a short summary behind a few key modules in the system, which will aid in understanding the system as a whole.

A. Semi-Direct Visual Odometry

SVO was initially introduced in 2014 [3] as a fast way to do visual odometry for monocular camera based systems. This work was recently extended to multi-camera, wide FOV systems and includes motion priors that use IMU information [4]. This makes the algorithm suitable to be used on our robot because we have a stereo-visual inertial sensor on it. This algorithm was designed for downward facing cameras which is also the case with our robot. The work, as its name suggests, lies between a feature based and photometric error based tracking system. It introduces a sparse image alignment algorithm which estimates frame to frame motion by minimizing the photometric error of features lying on intensity corners and edges. This makes it faster than a traditional direct method in that it only tracks a fraction of the total number of pixels in an image. It is also more robust than a feature based method because it utilizes features lying on intensity corners and edges. It then builds a sparse map using recursive Bayesian depth estimation and

uses bundle adjustment to refine the structure and camera pose. Which further improves the accuracy. The way it is implemented also makes it more efficient. It extracts features from selected keyframes in a parallel thread, decoupling it from hard real-time constraints. It does not require robust data association because of using the sparse image alignment direct tracking method. Finally, it only needs a sparse map of the environment to work well. All these enhancements mean that SVO takes only 2.5 milliseconds to estimate the pose of a frame on a standard PC. Which makes it ideal to be used on a quadrotor with limited computing resources. The pipeline is illustrated in Fig 1.

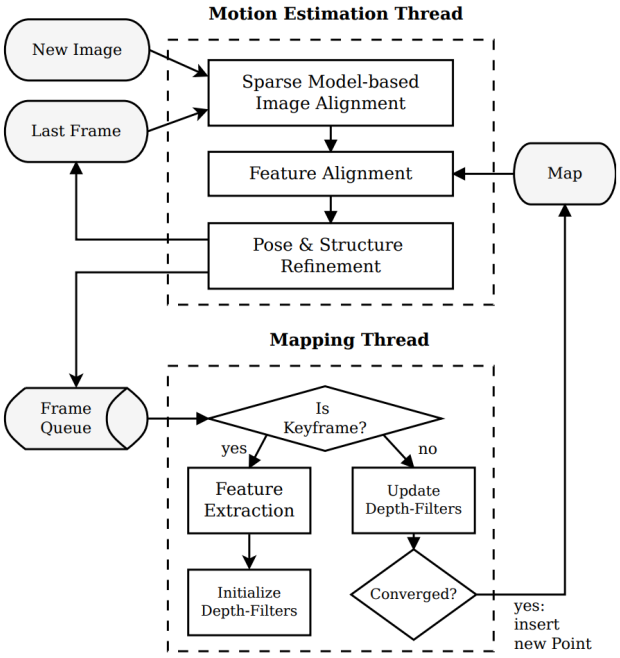

Fig. 1. SVO Tracking and Mapping Pipeline [Credits: Forster et al. [4]]

B. Iterative Closest Point

Iterative Closest Point (ICP) is a popular registration algorithm that is used to transform a source point cloud such that its difference from a target point cloud is minimal. It's an EM style algorithm, which starts with an approximation to the transformation and then iteratively converges to a solution. The steps involved in a typical ICP algorithm are as follows:

- Find correspondences between source point cloud and target point cloud
- Optimize for a transformation matrix (rotation + translation) such that the sum of distances between the correspondences of the point cloud pair is minimized
- Apply this transformation matrix on the source point cloud
- Check for loop breaking conditions, such as max number of iterations, difference in transformation from previous iteration and of course, distance between the correspondences
- Repeat, if conditions are not met.

There are many variants of ICP which basically differ on different aspects of the aforementioned steps. In our case specifically, we use the LM-ICP, which performs a non-linear optimisation of the least squares problem by using the Levenberg-Marquardt Algorithm [5]

IV. System Overview

The basic idea is to combine a fast visual inertial odometry algorithm with a point cloud registration and fusion approach that generates a dense map. A brief system overview is provided in Fig 2. It consists of a downward facing visual inertial sensor system that feeds time synced stereo images and IMU information to the VIO algorithm. We are using the Semi-Direct Visual Odometry method proposed in [4] to give us pose information. In parallel there is a stereo matching node that takes in the stereo images to produce dense point clouds using block matching. There is a pre-processing stage before ICP that uses the pose information to register the generated point cloud. Then ICP is used to align and fuse the incoming point clouds together. Once a large enough map is generated, it can be fed to the landing zone evaluation stage to identify good spots for the drone to land on.

The idea with this pipeline is to use a fast algorithm give us pose estimates that can be used to pre-align point clouds, before giving it to a more robust, but slower algorithm like ICP. If the point clouds are already aligned correctly, ICP will take much lesser time to generate accurate registrations for point cloud pairs.

A. UAV specifications

For this work the robot we used has two 1280x1024 px IDS Imaging Camera Modules with hardware triggering, an Epson IMU and an onboard NVIDIA TX1 compute unit running Ubuntu 14.04 with ROS Indigo. This robot served as our test platform along with a data collection unit. Figure 3 shows the exact setup we had.

B. Programming Environment

All our programming was done in C++. We developed independent ROS nodes that can speak to each other using the publisher subscriber system in ROS. Each block in the system overview from Figure 2 has it's independent node, listening on specific messages.

V. Data Collected

We have collected several datasets of the drone flying over varied surfaces such as low vegetation (grass, small plants), rocks, and walkways. The datasets are in the form of rosbags that consist of the following information that is time-synced:

- Right Camera Raw Image
- Right Camera Information
- Left Camera Raw Image
- Left Camera Information
- IMU Data (Accelerations and Velocities)

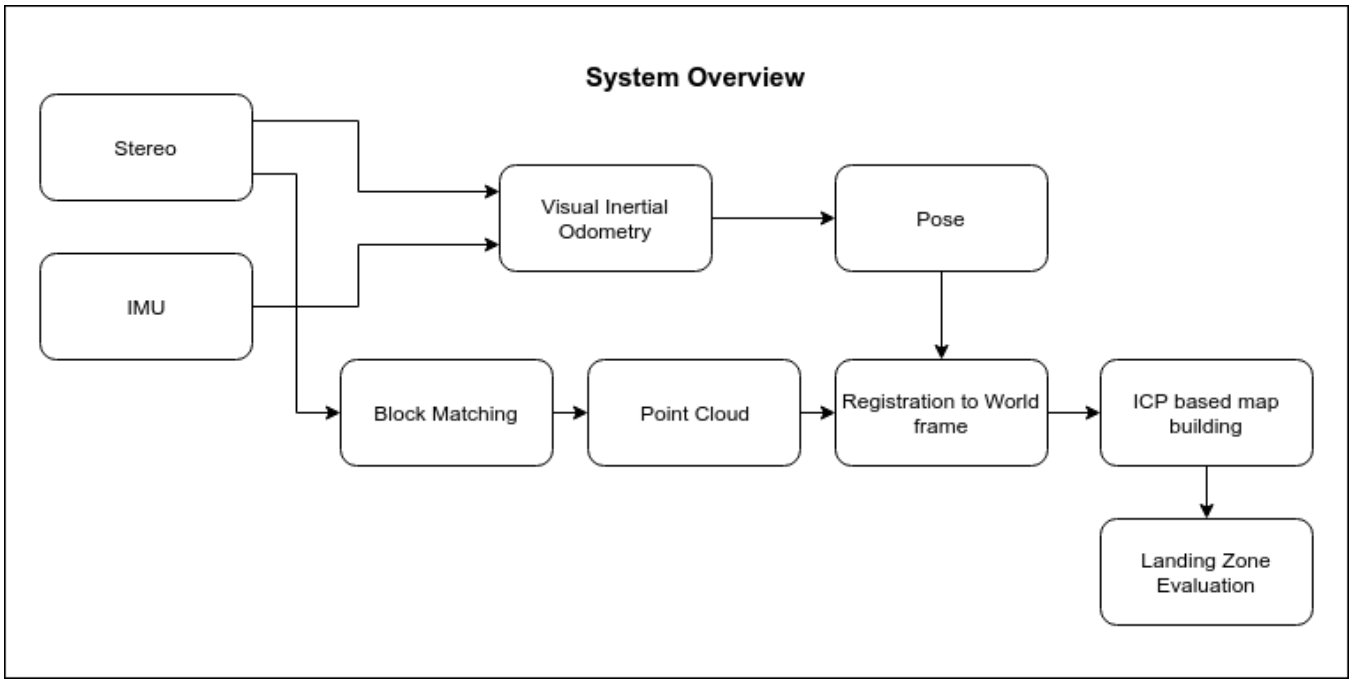

Fig. 2. System Overview: Two independent streams for generating the final map, flowing into the evaluation module

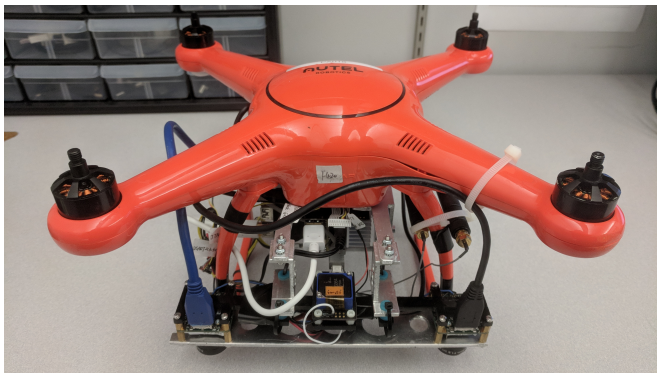

Fig. 3. Drone Setup with a downward looking stereo camera and an IMU

Most of this data was collected around the Newell Simon Hall at Carnegie Mellon University, Pittsburgh using the drone as described in Fig. 3

VI. Code Overview

A. Point Cloud Generation

This is a standard OpenCV implementation of the block matching algorithm that produces a disparity map and projects it to a point cloud using camera intrinsics.

B. Point Cloud Registration

This module registers the point cloud using the pose information given by the VIO algorithm. The registration is done to transform the point to a fixed reference frame and align its z-axis with the gravity vector. The alignment between z-axis and g vector is important because it influences the slope computation accuracy in the landing zone evaluation phase.

C. Point Cloud Fusion

We are performing point cloud fusion by using the ICP algorithm. The code we wrote uses the implementation in the point cloud library(PCL)[6] which does a non-linear optimisation using the Levenberg-Marquardt Algorithm . It currently takes a set of N clouds and fuses all the point clouds in the frame of the first point cloud. So, it starts with registering point cloud 2 with point cloud 1. After transforming and fusing the two together, it repeats the process with point cloud 3 and the fused point cloud. In this sequential manner, we generate a final fused point cloud comprising of all the N point clouds. This continuous approach of fusing point clouds leads to a large point cloud size with time as redundant points might get added repetitively. To remedy this issue, we run a voxel grid filter every 10 iterations through the point cloud to keep the map size under limits.

Visual Inertial Odometry

The visual odometry has been set up for using the stereo and IMU sensor feedback. It is the implementation of SVO as found on its official github repository [2]. Some changes have been made to integrate it with the rest of the code-base.

Map to Landing Zone Evaluation

This is glue code that connects the landing zone evaluation pipeline to the rest of the code so that the evaluation can be carried out on the generated map as opposed to running directly using the stereo and IMU.

Landing Zone Evaluation

This is an existing algorithm being used as the final portion of the pipeline. It is code that was written by Rohit as part of another project he has been working on. This piece of

code was not written as part of this project. But has been mentioned for reference. Also, it must be noted that it will not be possible to release this module as part of sharing the codebase at the end of the project since it is not open for sharing.

VII. Results

Here we show the results are different stages in our pipeline. The results are best seen via linked videos to the paper, but for illustration we are providing some screenshots as well.

A. Short Trajectory

We tested our algorithm on a short trajectory first. SVO is a fast algorithm, but struggles with accuracy on long runs. Therefore, we wanted to use it for short runs first to see how accurate the final output can look like. Fig 4 shows the trajectory used for registration of the point cloud. Fig 5 shows the output from the ICP module. The final fused point cloud qualitatively looks pretty accurate. The landing zone evaluation post processing on the map is shown in Fig 6. Which also looks about right. The tables are clearly segmented out as being unsafe, and the ground is segmented as safe.

B. Long Trajectory

The same flight data was used, but for a longer 20s duration as opposed to 9s in the short trajectory case. Analysis of the trajectory shown in Fig 7 shows that the odometry is not very accurate in the later half with keyframes spaced further apart. We predicted that this would probably add a fair amount of error to the registration process and hoped that ICP would be able to reduce it. But the output from the ICP module as shown in Fig 8 shows that there are multiple planes at angles to each other. This represents a failure case for the entire pipeline and shows that this technique may not be robust for long trajectories.

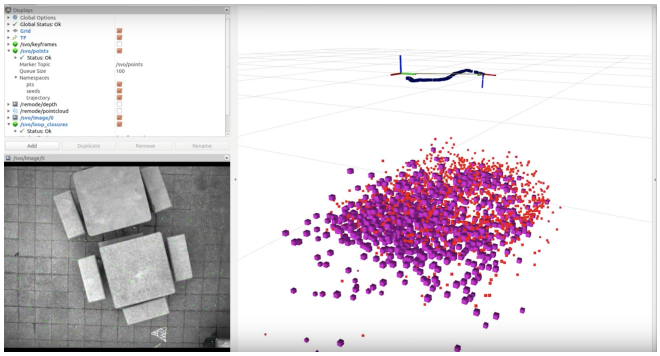

Fig. 4. SVO Output: The result from the SVO algorithm on a sample from the dataset we collected. Full Video: https://youtu.be/xs2hG6Y-YyU

VIII. Technical Challenges Faced

One of the most time consuming aspect of the implementation effort in this work was with calibrating the stereo camera and IMU together so that it can be used

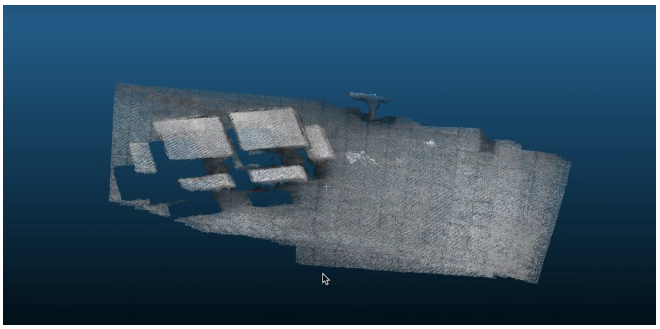

Fig. 5. Reconstructed Map: One view of the point cloud generated for the map of the ground. Full Video: Full Video: https://youtu.be/8d3sL-cS4DA

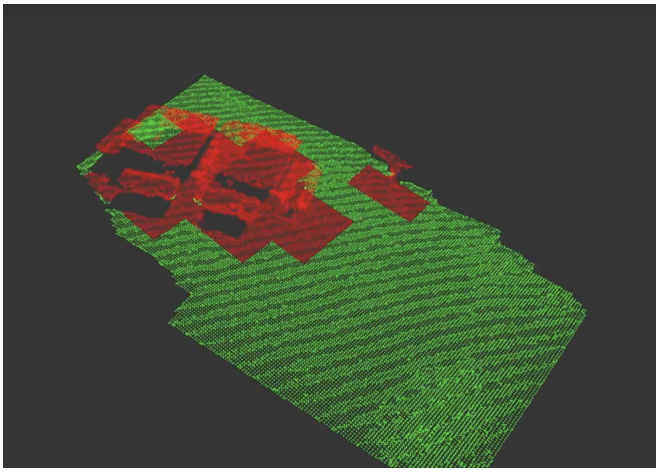

Fig. 6. Landing Zone Evaluations: Reds suggest unsafe to land and greens suggest that the spot is safe to land on. Full Video: https://youtu.be/-I2iNvp8sqk

in the SVO implementation. Even after several attempts, the Kalibr Toolbox was not able to successfully reduce the calibration re-projection errors to an acceptable range. There was non-trivial effort involved in gathering better calibration videos for the toolbox to work with.

Another significant technical challenge was to get SVO to work accurately in our constrained setup. The original paper works with images coming in at 70 fps, where with the hardware setup on our drone we were limited to an input of 10 fps. Also, initializing the algorithm could only be done in-air and not during take-off. During take-off the robot sees very few features, leading to a poor initialization of the algorithm which in-turn affects the entire pipeline downstream, leading to poor results.

The last major challenge was to speed-up the ICP module. Currently, this is our rate determining step, stopping us from the running the system in real time. From our experiments, we are close to 1 fps speed with ICP which is very slow. We tried reducing the max number of iterations ICP runs, but observed that it's the first iteration which is the most time consuming and hence the number of iterations wouldn't help our case as much. The graph in Fig. 9 plots our the average time taken to align a pair of point clouds, given the max

number of iterations.

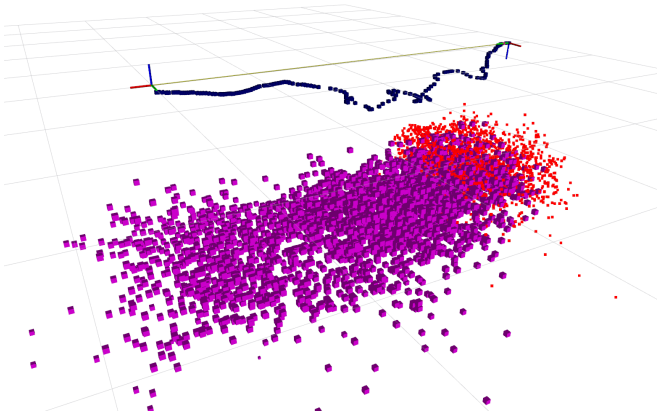

Fig. 7. SVO Output Trajectory for a 20s flight

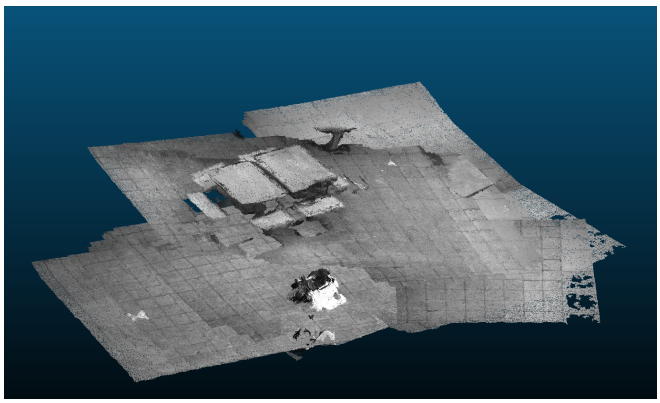

Fig. 8. Failure Case: Long 20s Trajectory

IX. Time-line discussion

We managed to finish most things we had proposed in the stipulated time, eventually. There were a few hiccups initially with respect to calibration as discussed in section VII which delayed us a little but finally we could get the entire pipeline in place. That said, we would have liked to run the system on the drone directly for an autonomous landing aided by this system, but we couldn't get enough time to run the system on the drone as well.

X. Conclusions

To conclude, this work provides a proof of concept for a unmanned aerial vehicle to perform landing site evaluation. We experimented with a number of different algorithms for pose tracking and for map fusion giving us a solid experience of building our own SLAM systems. For the future work on this project, we hope to speed up our pipeline to shoot for the near real-time goal that we always had. In addition, we felt that having objective evaluation metrics would have been very beneficial for our experiments. Hence, we would like to set-up datasets and corresponding metrics that can help us experiment more scientifically, by observing the performance of our experiments on these established evaluation metrics.

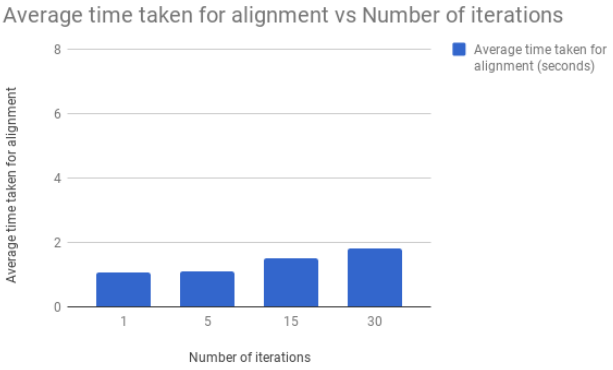

Fig. 9. Time taken for ICP

We would like experiment with removing/replacing different modules in the proposed system. We also want to observe the effects of relaxing constraints such as in-flight initialization, small flight durations and highly dense reconstructions on our landing zone evaluation module. Currently we are observing that some times we don't do as well in a longer flight as the errors from SVO add-up, leading to incorrect transformations, which eventually impact the final map built.

References

[1] J Delmerico and D Scaramuzza. "A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots". In: *IEEE International Conference on Robotics and Automation (ICRA)*. CONF. 2018.

[2] Christian Forster. *Semi-direct Visual Odometry*. 2017. URL: `https://github.com/uzh-rpg/rpg_svo` (visited on 04/01/2018).

[3] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. "SVO: Fast semi-direct monocular visual odometry". In: *Robotics and Automation (ICRA), 2014 IEEE International Conference on*. IEEE. 2014, pp. 15–22.

[4] Christian Forster et al. "Svo: Semidirect visual odometry for monocular and multicamera systems". In: *IEEE Transactions on Robotics* 33.2 (2017), pp. 249–265.

[5] DONALD W. MARQUARDT. "AN ALGORITHM FOR LEAST-SQUARES ESTIMATION OF NONLINEAR PARAMETERS*". In: *Journal of the Society for Industrial and Applied Mathematics* 2.2 (1963).

[6] Radu Bogdan Rusu and Michael Dixon. *Iterative Closest Point*. URL: `http://docs.pointclouds.org/trunk/classpcl_1_1_iterative_closest_point.html`.

[7] Sebastian Scherer, Lyle Chamberlain, and Sanjiv Singh. "Autonomous landing at unprepared sites by a full-scale helicopter". In: *Robotics and Autonomous Systems* 60.12 (2012), pp. 1545–1562.

[8] Thomas Whelan et al. "ElasticFusion: Dense SLAM without a pose graph". In: Robotics: Science and Systems. 2015.