Underwater Visual Inertial SLAM

Ana Warner, Anja Sheppard, Hersh Vakharia, Ryan Donoghue, Menelik Weatherspoon

***Abstract*— This work improves upon previous implementations of underwater simultaneous localization and mapping by incorporating visual data into the robot state estimation. Previous works primarily rely on sensor measurements and control inputs to acquire an estimate of an ROV's position due to the sparsity of landmarks in an underwater environment; however, problems such as drift commonly occur in these environments which can cause the estimation to become inaccurate. Our method utilizes Graph SLAM coupled with an ORB feature detector for landmark recognition in order to better localize the robot in a contained environment. This was accomplished using the BlueROV2 robot platform. This platform affords the use of an Inertial Measurement Unit (IMU), Doppler Velocity Logger (DVL), Depth sensor, and a stereoscopic camera as part of its sensor package. These sensor readings contribute the factors to our graph solution. The BlueROV2's internal odometry estimates are used as nodes in the factor graph to provide starting point for optimization. The graph SLAM optimization is undertaken using the GTSAM python library and compared against the original odometry data for verification.**

I. INTRODUCTION

Research in Simultaneous Localization and Mapping (SLAM), a central robotic navigation technique, has in recent years turned to graphical approaches in order to achieve robustness and lower uncertainty. For operations in extreme environments, such as underwater, SLAM approaches must be adapted in order to combat unique problems. Underwater SLAM implementations must handle limitations in data quality, visual distortion, and a lack of GPS signal [1]. Therefore a robot must maintain its own robust estimate of location and the surrounding marine environment. In Graph SLAM, a factor graph is used to optimize an initially estimated trajectory, an approach made easier by the open-source software package GTSAM [2]. In this work, we incorporate an underwater feature detector (ORB) to an inertial SLAM factor graph in order to combat odometry drift and achieve accurate localization.

II. BACKGROUND AND RELATED WORKS

A. SLAM

1) Filter-Based Approaches: Before the introduction of graph-based approaches to SLAM, the state-of-the-art approaches to underwater localization were Kalman Filtering and Particle Filtering. In the filtering approach, the probability distribution of the joint posterior density of the robot state given sensor measurements and control inputs is represented by the following:

$$P(x_k, m \mid Z_{0:k}, U_{0:k}, x_0) \tag{1}$$

where X is the set of robot locations, Z is the set of sensor observations, U is the set of control inputs, and m is the landmark [1]. The Extended Kalman Filter (EKF) attempts to approximate the probability distribution by linearizing the robot's motion model.

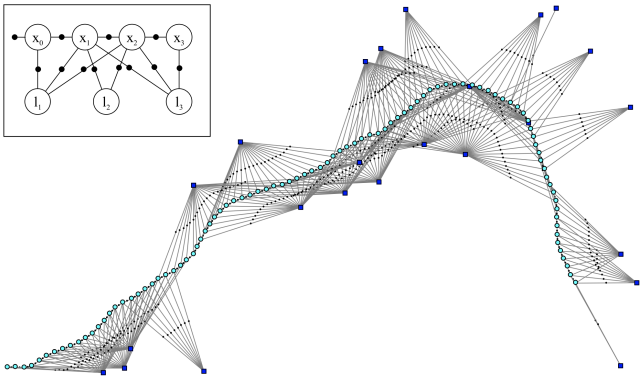

Fig. 1. Factor graph approach to SLAM problem, where the dark blue squares are landmarks, the light blue circles are the poses, and the lines are the constrained edges [3].

Particle Filtering is an alternative to EKF, and it is often preferred due to its ability to represent the joint posterior as a non-Gaussian [1]. Particle filtering represents the posterior distribution as a set of samples, or "particles" with individual weights (w).

$$P(x) = \sum_{i=1}^{N} w^{[i]} \delta_{x[i]}(x) \tag{2}$$

2) Graph SLAM: The invention of graph-based SLAM [2] introduced a new framework for solving the SLAM problem, particularly with large numbers of landmarks. The formulation of the problem in Graph SLAM is fundamentally different than filtering: the factor graph nodes correspond to the poses of the robot at different points in time, and the edges of the graph represent the constraints (aka sensor measurements) on those positions (see Fig. 1).

With the introduction of the Incremental Smoothing and Mapping (ISAM) update, Graph SLAM became a viable online SLAM solution in the early 2000s [4]. It is now considered state-of-the-art.

B. Feature Detection

In order to find the landmarks for the Graph SLAM approach, a common technique is using visual feature extractors and correlating them across frames. Common feature detectors include SIFT [5], SURF [6], BRISK [7], GFTT [8], and more.

Feature detectors that work well with indoor or outdoor environments do not necessarily transfer well to underwater

applications. The additional challenges can stem from differences in light attenuation of water and the often feature-sparse environments that negatively impact Graph SLAM convergence. In this project we aim to use ORB [9], a feature detector that has been shown to perform better in underwater environments relative to other methods. [10].

ORB is an efficient feature corner detector and features are able to be matched between images with low computational cost. Corners are also a relatively simple type of feature to detect which increases robustness in underwater applications.

Detectors such as SIFT and BRISK have scale and rotation invariant features, but they look for unique types of features. In an underwater application features can be subjected to more complex visual transformations which can distort features enough to not be matched. Light reflections, refractions, scattering and absorption can more significantly affect SIFT features compared to ORB [11].

C. Underwater Sensors

In the work by Bagoren et al, a sensor model was designed to incorporate Doppler Velocity Log (DVL) and depth data into the GTSAM factor graph interface for a BlueROV robot. The work's contributions included definitions of the residuals of each sensor and the Jacobians of the sensors. This work is necessary for running graph SLAM with custom factors such as those for an underwater robot.

III. METHODS

Our work builds directly on that of Bagoren et al. by adding landmarks to the SLAM graph detected with ORB. We draw upon the work's Jacobian definition and residual computation for velocity factors to allow us to incorporate DVL data into the graph. It is expected that the addition of visual landmarks should help constrain the graph, rectify drift, and lead to a better result path.

A. Framework

Data was collected by the Field Robotics Group at the University of Michigan Marine Hydrodynamics Lab on April 14, 2023. The robot platform is the BlueROV2, a commercial underwater platform that has been modified to include a Doppler Velocity Log (DVL) and downward facing ZED stereo camera. Data was collected in the form of a Robot Operating System (ROS) bag file, which can be played back in real time for post-processing. Our implementation subscribes to the rosbag's published channels to record measurements. Since the new messages have different frequencies, we synchronize them with the ROS ApproximateTimeSynchronizer, which groups data with similar timestamps.

B. SLAM

A batch solution of graph SLAM was implemented using the GTSAM Python library. The graph was composed of two elements: nodes and factors. Factors are further subdivided into the different sensors available within the BlueROV's sensor package. The graph was created as a Nonlinear Factor Graph and optimized with the Levenberg-Marquardt batch optimizer. The pseudocode for our graph construction can be found in Algorithm 1.

1) Nodes: The nodes within the factor graph are comprised of the robot's 6-dimensional pose and its velocity. The initial estimates for the pose and velocity nodes are constructed using the Kalman-filtered odometry from the onboard DVL sensor as well as the true depth of the robot using the onboard barometer and the following calculation:

$$d_{measured} = \frac{\texttt{absolute pressure} \times 100 - 98250}{997 \times g} \quad (3)$$

where 98250 is the density of water in $\frac{kg}{m^3}$, 997 is the atmospheric pressure of water in millibars, and g is the gravitational acceleration constant, 9.81 $\frac{m}{s^2}$. Absolute pressure is measured by onboard BlueROV pressure sensors and published over the `/BlueROV/pressure2` topic. The final position and velocity are model ed in equations 4 & 5.

$$X_{Final} = [1 \quad 1 \quad 0] * X_{initial} + d_{measured} + \eta_{pose} \quad (4)$$

$$V_{Final} = V_{initial} + \eta_{velocity} \quad (5)$$

Where η represents the isotropic noise used for our sensors estimates of pose and velocity.

2) IMU Factors: IMU factors were used to represent the transformation from one node to the next. Within the GTSAM library, this is accomplished using IMU preintegration techniques allowing for an accumulator variable to summarize several IMU measurements into a single relative motion between two nodes. The accumulator is first initialized using the covariance of the linear acceleration, angular velocity, and integration. Other factors such as second order Coriolis and omega Coriolis were removed from the preintegration parameters or set to 0.

Once initialized, the accumulator can be used to track our change in pose and velocity between sequential nodes. The accumulator calculates changes in pose and velocity by integrating the linear acceleration and angular velocity using Euler's method. The acceleration and angular velocity recorded by the IMU are modelled in equations 6 & 7. $R_{B/W}$ represents the rotation of the world frame into the body frame, $b^{(i)}$ represents the bias term corresponding to variable i, and $\eta^{(i)}$ represents the noise associated with variable i. To obtain our acceleration measurement in the body frame, $R_{B/W}$ was generated using the orientation data extracted from the IMU data package. This matrix was used to transform our gravity vector into the body frame such that we can remove it from $\tilde{a}_B(t)$ to obtain the linear acceleration expected by the accumulator. The accumulator is reset once the IMU factor is created.

$$\tilde{a}_B(t) = a_B(t) + R_{B/W} * g_W + b^a + \eta^a \quad (6)$$

$$\tilde{\omega}_B(t) = \omega_B(t) + b^\omega + \eta^\omega \quad (7)$$

3) DVL Factors: The DVL factor is constructed using GTSAM's CustomFactor class which allows a custom unary factor to be added to the graph. Due to the nonlinear nature of the factor graph, The custom factors require an error function to be defined so that the optimizer has a means of approaching a solution. The definition of the residual and the Jacobian of the DVL model, are from Bagoren et al.

The DVL residual and Jacobian are found below:

$$r_{DVL} = \hat{R}_i \begin{bmatrix} \tilde{v_x} \\ \tilde{v_y} \\ \tilde{v_z} \end{bmatrix} - \begin{bmatrix} \hat{v_x} \\ \hat{v_y} \\ \hat{v_z} \end{bmatrix} \in \mathbb{R}^3 \tag{8}$$

$$H_{DVL} = \hat{R}_i \in SO(3) \tag{9}$$

where $\hat{R}_i$ is the estimated rotation at the current state from the DVL frame to world coordinates, $\tilde{v}$ is the measured velocity from the DVL, and $\hat{v}$ is the estimated velocity in world coordinates.

The custom factor is then constructed using the residual, an isotropic noise model, and the keys of the corresponding node in the factor graph.

During playback of the bag file, data from the DVL, IMU, odometry, and pressure sensor are appended into several arrays. A noteable difference comes during processing of the IMU factor as, due to the accumulator, factors must be created and stored once new odometry data is received for the accumulator to be properly reset. During the subsequent batch solution, IMU and DVL factors get added into a graph variable while the position and velocity values are stored in a Values array, provided by GTSAM. The final graph is illustrated in Figure 3 where the IMU factor serves to link two sequential nodes and the DVL factor acts to provide additional information on its corresponding node. The Levenberg-Marquardt optimizer then acts to minimize the error of all factors by adjusting the values of the graph nodes.

C. Landmarks

Landmarks are extracted from tracked visual features found in the images from our downward-face stereo camera. We used feature tracking provided by the *gtsam_vio* package [12]. This package uses the FAST feature detector by default; we modified it to use ORB. Our SLAM was tested using both detectors, but the resulting paths were marginally different. We opted to use ORB features in our final implementation. An example of the ORB feature detector on an underwater image is seen in Fig. 2.

The `gtsam_vio` package provides a system that returns left and right stereo image pixel pairs of the tracked features. These values are used to compute world pose of the landmark (using standard stereo triangulation for depth), as well as the factors that connect the landmark to other nodes in the graph. Landmarks are added as pose nodes in the graph, as signified by "L1" in Fig. 3. The landmarks are connected to the robot poses that observed that landmark with a GenericStereoFactor3D factor in GTSAM. This factor requires a GTSAM StereoPoint2, a noise model, the index of the robot and landmark node, as well as a GTSAM Cal3_S2Stereo object representing the stereo camera intrinsic and extrinsic parameters.

Landmark detection can improve localization as there are few options underwater to observe direct location. There are no options such as GPS which do not accumulate error. Landmark detection is one of the few ways to get loop closure and correct for drift. However, detecting landmarks underwater also can prove to be difficult due to reflections on the surface and reduced visibility distance in water as compared to air. Improving and adding more robust feature detection methods could prove to improve underwater SLAM.

Algorithm 1 batch_graph(odom, dvl, imuPreintegration)

Data: $odom$: odometry poses; dvl: Doppler velocity logger twist measurements ; $imuPreintegration$: Preintegrated IMU measurements between poses; $landmarks$: all detected landmarks

Result: $graph$: final factor graph; $initialEstimate$: estimates to be optimized

$graph \leftarrow$ gtsam.NonlinearFactorGraph()
$initialEstimate \leftarrow$ gtsam.Values()
$isam \leftarrow$ gtsam.ISAM2()
$initialEstimate$.insert($B(0), biasNoise$)
for $i = 0$ *to len(odom)* **do**
  $pose \leftarrow odom[i]$
  $velocity \leftarrow$ Vector3(0,0,0)
  if $i == 0$ **then**
    $graph$.add(gtsam.PriorFactorPose3($X(i), pose$, $poseNoise$)
    $graph$.add(gtsam.PriorFactorVector($V(i), velocity$, $velocityNoise$)
    $initialEstimate$.insert($X(i), pose$)
    $initialEstimate$.insert($V(i), velocity$)
  else
    $initialEstimate$.insert($X(i), pose$)
    $initialEstimate$.insert($V(i), velocity$)
    $graph$.add(gtsam.ImuFactor($X(i-1), V(i-1)$, $X(i), V(i), B(0), imuPreintegration$)
    $graph$.add(createDvlFactor($dvl[i]$))
    $imuPreintegration$.resetIntegration()
    for *landmark in landmarks[i]* **do**
      if *initialEstimate.exists(L(landmark.id))* **then**
        $initialEstimate$.insert($L(landmark.id)$, $landmark.pose$)
      end
      $graph$.add(gtsam.GenericStereoFactor3D(
       gtsam.StereoPoint2(...),
       $landmarkNoise$,
       $X(i)$,
       $L(landmark.id)$,
       $cameraCalibration$))
    end
  end
end
return $graph, initialEstimate$

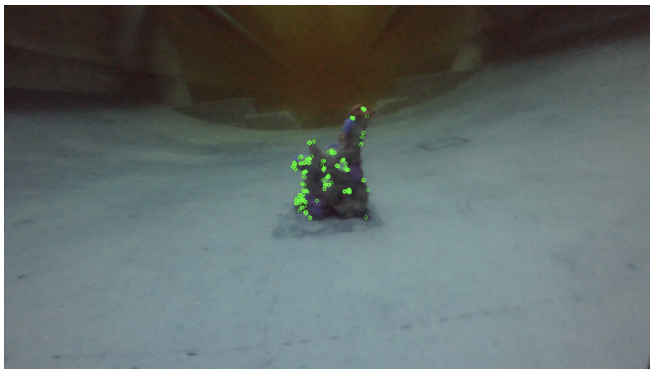

Fig. 2. ORB feature detector on an image from underwater camera feed

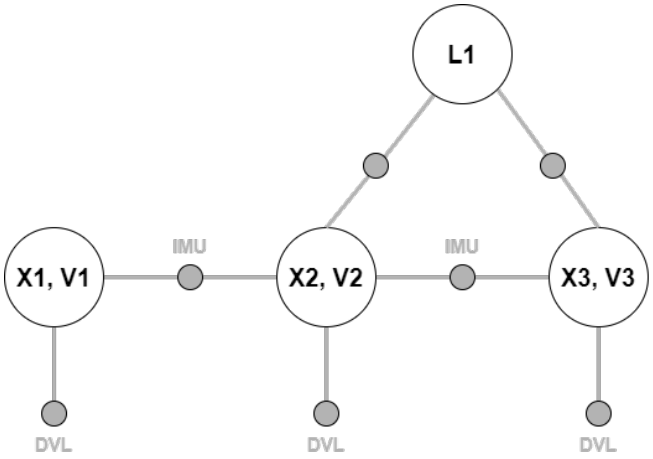

Fig. 3. Snippet of batch factor graph of SLAM implementation incorporating landmarks

IV. RESULTS

Fig. 4 shows the resulting trajectories from the SLAM system.

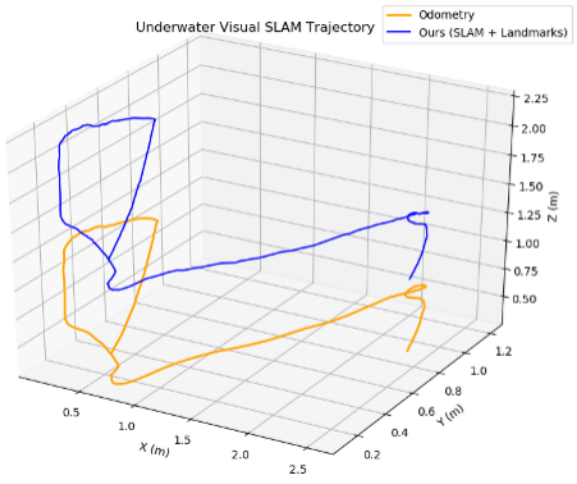

Fig. 4. SLAM-optimized solution accounting for depth, in blue, and DVL-filtered odometry in orange

We compare the DVL Kalman-filtered odometry to the SLAM solution using mean-squared error (MSE). The equation for MSE is as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n} ||Y_i - (\hat{Y}_i - z_d)||^2$$

where Y represents the filtered odometry pose estimate, $\hat{Y}$ represents the SLAM pose, and z_d represents the depth offset that we had added to the SLAM poses. Thus the MSE gives an average norm error between an odometry point and its optimized counterpart with depth equalized.

The MSE for our SLAM implementation was 0.81.

The code for this project can be found at `https://github.com/hvak/visual-underwater-slam`. A video summarizing the project can be found at `https://youtu.be/LLFDb0DvM18`.

V. DISCUSSION AND CONCLUSION

A. Analysis of Results

As seen in the figure and the MSE value, our SLAM solution looks very similar to the DVL Kalman filter with an offset, despite incorporating landmarks. This is because the feature detector was not able to find many features that extended across multiple frames, and so the SLAM system was not able to correct for drift. Fig. 5 shows examples of good and bad feature tracking in our dataset. We can conclude that without good landmarks, a graph SLAM system performs on par with a well-tuned Kalman filter.

B. Challenges

The primary challenge of this project revolved around the GTSAM SLAM library. Our initial solution attempts made use of the ISAM iterative solver since we intended to implement the code on the BlueROV2 robotics platform; however, we consistently encountered indeterminant linear system errors. Due to the nature of the ISAM solver, it is very difficult to narrow down what could be causing this problem and in our case, we were unsuccessful in trying to solve it. As a result, we defaulted to solving the factor graph as a batch solution with the intention of returning to an iterative solver once we successfully found a solution. The batch solver greatly deepened our understanding of the underlying processes of the GTSAM library and posed its own set of unique challenges. The primary issue we encountered revolved around the nature of IMU factors within GTSAM. Documentation on the underlying operations of GTSAM is sparse; additionally, there are few examples to demonstrate the proper use of an IMU in a GTSAM factor graph. Our group, therefore, had to dedicate a large amount of time and energy to gaining a deeper understanding of how these factors can be created and effectively implemented into our graph. We eventually decided to stick with a batch implementation, with ISAM being a goal of future work.

Our dataset was recorded from a real test, but the U-M Marine Hydrodynamics Lab does not have a motion capture system, so we did not have a ground truth trajectory. Since

we have not been working directly with the robot, we also do not have the ability to evaluate whether the sensors were properly calibrated, what the ground truth should look like (other than through a video stream), and other factors that may impact the quality of the test.

Further, the dataset we used was filmed in a pool with few macro-scale visual landmarks or features. The pool contained one large rock which limited the potential of the feature detection system, to which we attribute the lack of significant results improvement. Additionally, better tuning of ORB would have likely improved results.

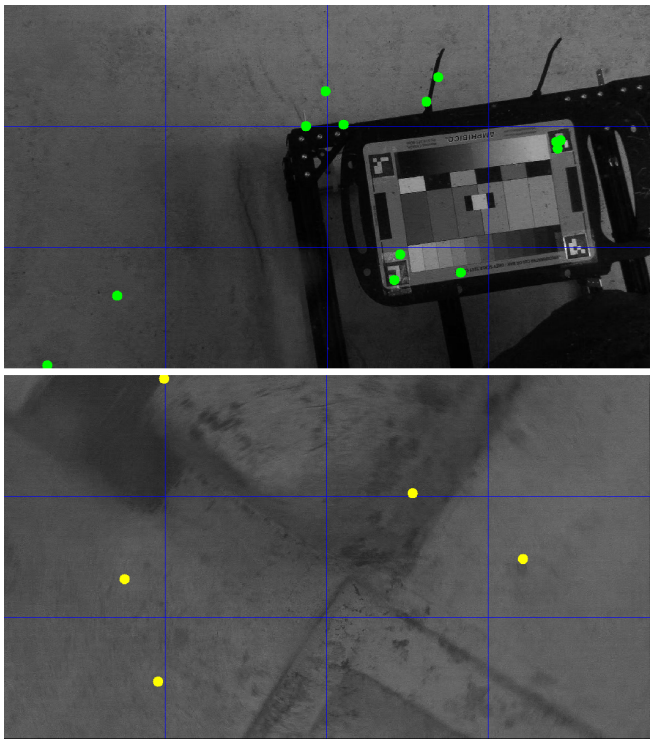

Fig. 5. Top image shows good feature tracking, where green dots are features that are being tracked. The bottom image shows bad feature tracking, where yellow dots are newly detected features.

VI. FUTURE WORK

We would like to adapt this system into an incremental solver so that it will be more useful for AUVs exploring new environments by running online. Additionally, using a dataset that has more features and a ground truth would mitigate the consequences of our dataset limitations.

Ultimately, the feature detectors that we used in this project were not able to locate good enough visual features to account for drift. Tuning these feature detectors or developing a better underwater feature detector would drastically improve these results. Many current approaches are utilizing learning to track landmarks across frames, and this is an interesting direction for future work.

In addition, due to the different properties of water and how it absorbs light more than air, new methods should take this into account. Perhaps instead of using visible light to perform visual slam underwater, methods that use higher frequencies of light may be able to better visualize underwater environments. Being able to achieve better visibility with these wavelengths in combination with improved feature detection methods may prove to make underwater SLAM more feasible.

ACKNOWLEDGMENT

We would like to thank Professor Maani Ghaffari, the instructor of ROB 530. Additionally, we extend great thanks to Onur Bagoren, who was a great help in collecting and processing data. We gratefully acknowledge the University of Michigan Field Robotics Group for the use of their datasets.

REFERENCES

[1] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part i," *IEEE Robotics Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006. DOI: 10.1109/MRA.2006.1638022.

[2] S. Thrun and M. Montemerlo, "The graph slam algorithm with applications to large-scale mapping of urban structures," *The International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[3] F. Dellaert and M. Kaess, "Square root sam: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006. DOI: 10.1177/0278364906072768. eprint: https://doi.org/10.1177/0278364906072768. [Online]. Available: https://doi.org/10.1177/0278364906072768.

[4] M. Kaess, A. Ranganathan, and F. Dellaert, "Isam: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008. DOI: 10.1109/TRO.2008.2006706.

[5] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[6] H. Bay, T. Tuytelaars, and L. Van Gool, "Surf: Speeded up robust features," in *Computer Vision – ECCV 2006*, A. Leonardis, H. Bischof, and A. Pinz, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417, ISBN: 978-3-540-33833-8.

[7] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *2011 International Conference on Computer Vision*, 2011, pp. 2548–2555. DOI: 10.1109/ICCV.2011.6126542.

[8] M. Karpushin, G. Valenzise, and F. Dufaux, "Good features to track for rgbd images," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 1832–1836. DOI: 10.1109/ICASSP.2017.7952473.

[9] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski, "Orb: An efficient alternative to sift or surf," *2011 International Conference on Computer Vision*, pp. 2564–2571, 2011.

[10] F. Hidalgo and T. Bräunl, "Review of underwater slam techniques," in *2015 6th International Conference on Automation, Robotics and Applications (ICARA)*, 2015, pp. 306–311. DOI: 10.1109/ICARA.2015.7081165.

[11] P. Tueller, R. Kastner, and R. Diamant, "A comparison of feature detectors for underwater sonar imagery," in *OCEANS 2018 MTS/IEEE Charleston*, 2018, p. 3. DOI: 10.1109/oceans.2018.8604786.

[12] V. Kopli, *Gtsam$_v$io*, 2020. [Online]. Available: https://github.com/vkopli/gtsam_vio.