Implementation and Evaluation of Semantic-Object SLAM Algorithm

Ziqi Han, ziqihan@umich.edu; Zhewei Ye, yezhewei@umich.edu; Tien-Li Lin, tienli@umich.edu; Yi-Cheng Liu, liuyiche@umich.edu; Shubh Agrawal, shbhgrwl@umich.edu

***Abstract*—Object SLAM enhances indoor environment understanding by incorporating objects into Simultaneous Localization and Mapping (SLAM). We present a monocular Semantic-Object SLAM (SO-SLAM) implementation from scratch, utilizing bounding box, semantic scale, plane-supporting, and symmetry constraints for single-frame initialization and orientation optimization within the GTSAM framework. To evaluate our implementation, we reconstruct point clouds and compare the resulting ellipsoid with three hand-annotated segments from the TUM dataset. Our project demonstrates substantial improvements of SO-SLAM over previous algorithms. The code is available at: https://github.com/MRHan-426/SOSLAM**

I. Introduction

An important problem in robotics is the ability for a robot to accurately determine its location and map its environment in an indoor space. Examples of such locations include an office, home, or school. Traditional SLAM approaches typically represent environmental features using points, lines, or planes. An innovation over this is the Object SLAM framework, which attempts to map the objects in the environment, considering their position, orientation and occupied space. Existing Object SLAM approaches miss the opportunity to use information that we can reliably infer about an object based on semantic labels. Semantic-Object SLAM (SO-SLAM), proposed by Liao et al. in 2021, aims to make use of this information [1]. However, the original author's code is currently unavailable for use: https://github.com/XunshanMan/SoSLAM.

Based on this situation, we make the following contributions:

- Provide an open-source reference implementation for SO-SLAM, which has thus far not been released
- Evaluate the effectiveness using three scenes from the open-source TUM RGB-D dataset

II. Framework

As a SLAM algorithm, our goal is to obtain an optimized estimate for both the robot poses and the pose and size of objects in the environment. We take odometry data and 2D RGB images as inputs.

A. Object Representation

Early Object SLAM algorithms, such as CubeSLAM [2] represent objects as cuboids. An innovation proposed by Nicholson et al[3]. is to instead represent objects as ellipsoids. Ellipsoids allow for a compact quadratic representation. Additionally, they can be easily projected into the plane as ellipses. The ability to project into the plane is a key feature in the implementation of some semantic factors.

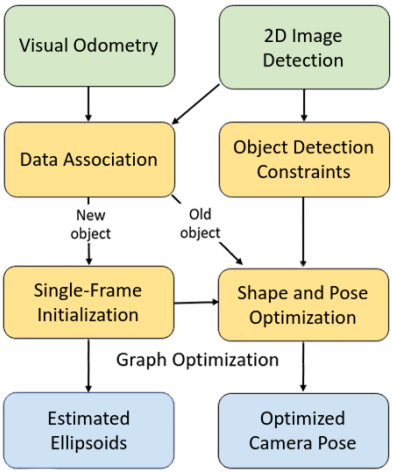


Fig. 1: Flowchart of SO-SLAM algorithm. Green nodes are inputs, blue nodes are outputs, and yellow nodes represent the main components of the algorithm

B. Algorithm Overview

At each given time step, the system receives an RGB image and updated pose from odometry. The image is processed through an image detection algorithm to find 2D object detections, providing a 2D bounding box and semantic label for each detected object. These detections are then carefully associated with odometry data to decide whether a given detection represents a new object (i.e. one that hasn't been seen before) or an existing one in the environment.

If a new object is detected, the Bounding Box Factor, Semantic Scale Factor, and Plane-Supporting Factor are employed to initialize the ellipsoid representing the object. Conversely, if the object has already been detected, the additional information from the new frame is utilized to add extra constraints to refine the object's representation. One such constraint is the Symmetry Factor.

Taking into account all the poses, detected objects, and constraints on these objects, we construct a factor graph. By optimizing this graph, we obtain our final camera poses and ellipsoid poses and shapes, accurately reflecting the environment. This algorithm is summarized by the flowchart in Fig.1.

C. Implementation Overview

Our implementation uses YOLOv8 for image detection. YOLO allows us to easily and reliably detect all objects in a given image (see Fig. 2). For odometry data, we use both ground truth data from the dataset and estimated poses using the ORB-SLAM2 algorithm. Data association was done by hand. Graph optimization was done using the GTSAM library (this is different from [1], which used G2O).

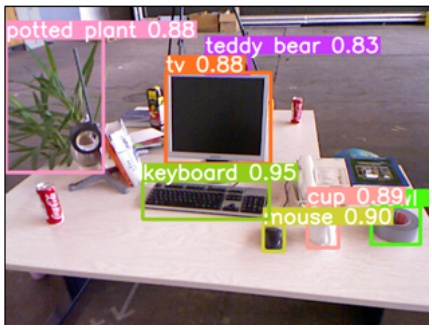


Fig. 2: Bounding boxes produced by YOLOv8 on the fr2 desk dataset

III. Single Frame Initialization

We employ an innovative technique, as outlined in [1], to initialize a three-dimensional ellipsoid for each identified object within the environment with a single RGB image input. An ellipsoid is characterized by its radius, position, and orientation, which are represented by the vector $\boldsymbol{r}$ and the matrix $\boldsymbol{Z}$.

$$r = \begin{bmatrix} a \\ b \\ c \end{bmatrix}, \quad Z = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{1}$$

To combine these parameters, we introduce a 4x4 symmetric matrix Q, which has 9 degrees of freedom:

$$Q = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} a^2 & 0 & 0 & 0 \\ 0 & b^2 & 0 & 0 \\ 0 & 0 & c^2 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} R^\top & 0 \\ t^\top & 1 \end{bmatrix} \tag{2}$$

As presented in [1], we introduce three spatial structure constraints in our implementation: bounding box constraints, scale proportionality constraints, and plane support constraints. These constraints are sufficient to initialize an ellipsoid using a single frame of data, which significantly reduces the system's dependence on the quantity and diversity of observations.

A. Bounding Box Factor

In a given frame, an object is given a bounding box by the object detection algorithm. For a possible ellipsoid $\boldsymbol{Q}$ representing this object, we can project $\boldsymbol{Q}$ onto the 2D image plane (see Fig. 3(a)) using the camera matrix $\boldsymbol{P}$, giving us a 2D ellipse, $\boldsymbol{C}$:

$$\boldsymbol{C} = \boldsymbol{P} \cdot \boldsymbol{Q} \cdot \boldsymbol{P}^T \tag{3}$$

The tangent lines of this projected ellipse create a minimum bounding rectangle. Ideally, the detected bounding box matches this minimum bounding box exactly, so we compute the error from each tangent, as shown in Fig. 3(b).

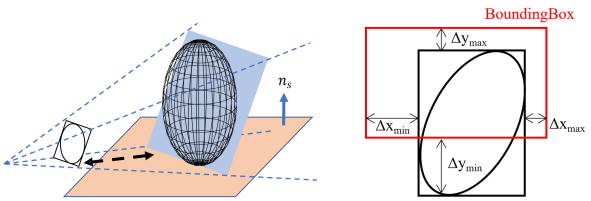


Fig. 3: (a)Bounding box projection. (b)Error between bbox and outer rectangle of ellipse

Hence, the bounding box constraint f_{bbox} constitutes a 4-degree-of-freedom constraint pertaining to the object, which can be defined as:

$$f_{bbox} = \left\| \begin{bmatrix} \Delta x_{\min} \\ \Delta y_{\min} \\ \Delta x_{\max} \\ \Delta y_{\max} \end{bmatrix} \right\|_{\sum_{bbox}} \tag{4}$$

where Σ_{bbox} is the covariance matrix of the bounding box. Following [1], we set $\Sigma_{\text{bbox}} = 10$ in our experiment.

B. Semantic Scale Factor

An essential geometric property of an object is its scale. Because we can expect artificial indoor objects

to follow a certain scale, we can significantly improve the accuracy of object mapping. However, it is quite difficult to determine the absolute scale of an object from 2D images. Instead, we use the system proposed in SO-SLAM[1], called the Scale Proportional Constraint, which constrains an an object's relative proportions rather than its absolute size.

The scale of an object is $\boldsymbol{S} = [a, b, c]^T$, where a, b, c is the half scale of its X, Y, Z axes. The scale ratio $\boldsymbol{r} = [r_{sx}, r_{sy}]^T$ is defined as:

$$r_{sx} = \frac{a}{c}, \quad r_{sy} = \frac{b}{c} \tag{5}$$

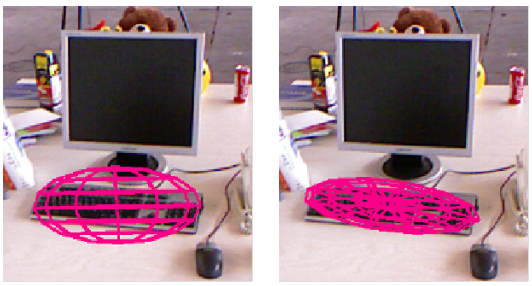
We then construct a table mapping semantic labels to expected scale ratios, which can be obtained by averaging the scale ratios of different objects in the semantic category. Given a prior semnatic label l (from object detection), we can compare a candidate ellipse $\boldsymbol{Q}$ which its corresponding semantic scale ratio $\boldsymbol{r_s}(l)$. The semantic scale error term is then defined as:

$$f_{ssc}(l) = \left\| \begin{bmatrix} \frac{a}{c} - r_{sx}(l) \\ \frac{b}{c} - r_{sy}(l) \end{bmatrix} \right\|_{\Sigma_{ssc}} \tag{6}$$

where Σ_{ssc} is the scale variance. Following [1], we set $\Sigma_{ssc} = 1$ in our experiment. The constraint error becomes the smallest when the scale value of the object $\boldsymbol{Q}$ is consistent with its semantic scale prior value $\boldsymbol{r}_s$.

For example, since a keyboard is relatively flat, we set its ratio as:

$$r_{sx}(\text{keyboard}) = 10, r_{sy}(\text{keyboard}) = 20 \tag{7}$$



(a) (b)

Fig. 4: (a) Keyboard without semantic scale constraint. (b) Keyboard with semantic scale constraint

The results of our implementation demonstrate its effectiveness in handling scale ambiguity in object mapping. However, some limitations persist, such as the need for a comprehensive scale ratio table of common objects. Future directions for research include refining the scale ratio table and exploring ways to automatically learn scale priors from data to enhance adaptability.

C. Plane-Supporting Factor

One factor we are trying to infer about the ellipsoid is its orientation in 3D space. Most objects in indoor environments are resting on some plane (e.g. a cup on a table). To exploit information about this common case, SO-SLAM[1] proposes a method called the Plane-Supporting Factor.

The stability of the object depends on the orientation of its axes with respect to the normal vector of the supporting plane. If the object's z-axis is upward (opposite the direction of gravity), then its x- and y-axes must be perpendicular to the normal vector of the supporting plane, denoted by n_s. This ensures that the object is stable and does not tip over.

Mathematically, the plane support constraints can be expressed as follows:

$$f_{\text{psc}}(Q) = \left\| \begin{bmatrix} \text{Rot}_x(Q) \cdot n_s \\ \text{Rot}_y(Q) \cdot n_s \end{bmatrix} \right\|_{\Sigma_{\text{psc}}} \tag{8}$$

This means the orientation of the ellipsoid Q should converge such that its z axis is perpendicular to the supporting plane, as shown in Fig. 5

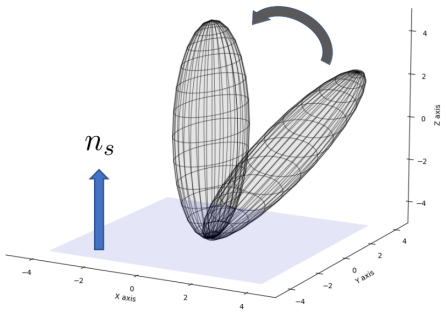


Fig. 5: Quadric with plane-supporting constraint which is tangent to the supporting plane n_s

SO-SLAM[1] proposes the additional constraint that the ellipsoid be tangent to its supporting plane. However, we believe that it is infeasible to effectively extract accurate ground information from a single RGB image. Additionally, manually annotating ground based on point cloud data does not significantly contribute to the algorithm's applicability in real-world scenarios. Therefore, we have not introduced constraints regarding the tangential relationship between the ground and ellipsoid in our implementation.

D. Solving the single frame initialization

Due to the various types of constraints involved, obtaining an analytical solution directly is challenging. To address this, we use a nonlinear optimization technique

as mentioned in SO-SLAM[1] that uses the Levenberg-Marquardt algorithm to search for the optimal ellipsoid, $\widehat{\boldsymbol{Q}}$. The objective function is formulated to incorporate three constraints: the bounding box constraint f_{bbox}, the semantic scale constraint f_{ssc}, and the plane supporting constraint f_{psc}, shown as follows:

$$\widehat{\boldsymbol{Q}} = \underset{Q}{\operatorname{argmin}} \left(f_{\mathrm{bbox}} + f_{\mathrm{ssc}} + f_{\mathrm{psc}} \right) \tag{9}$$

The optimized ellipsoid is then used as an initial estimate for the object.

IV. Shape and Pose Optimization

After initialization, we add additional factors to the factor graph to further constrain the object.

A. Symmetry Factor

Because most man-made objects are symmetric, the shape and pose of the ellipsoid is constrained by a symmetry property.

1) Preliminaries: The symmetry property states that, for any 3D point $v_0 \in V$, there must be a point $v_0^S \in V$ which is symmetric to v_0 across some symmetry plane, π_{sym}. We express reflection accross this plane by the linear transformation $v_0^S = \mathcal{S}(v_0, Q)$

For a given object, Q, let $V_Q = \{v_0, v_1, ...\}$ denote the set of points on the object's surface in the world coordinates and $U_Q = \{u_0, u_1, ...\}$ denote the set of points on the surface in image coordinates. We have that U_Q is the image of V_Q under the camera projection transformation $\mathcal{P}$. Further, we must also have that $v \in V_Q$ are on the surface of the ellipsoid Q. This means any $v \in V_Q$ satisfies:

$$\begin{cases} u = \mathcal{P}(v) = P \cdot v \\ v^T Q v = 0 \\ v \text{ is visible by camera} \end{cases} \tag{10}$$

Based on the above constraints, we can derive an equation for the symmetry point u_0^S of an image coordinate u_0 as:

$$u_0^S = (\mathcal{P} \circ \mathcal{S} \circ \mathcal{P}^\dagger)(u_0, Q) = \mathcal{P}(\mathcal{S}(\mathcal{P}^\dagger(u_0, Q), Q)) \tag{11}$$

This equation says that a 2D point u_0 should be back-projected to 3D ellipsoid surface point v_0, then reflectied across the symmetry plane to get v_0^S, and finally projected onto image point u_0^S.

We now need to find a invariant descriptor β which can be used in the cost function to optimize objects towards being symmetric. That is, we want a function β satisfying that, for a symmetric pair u_0, u_0^S:

$$\beta(u_0) = \beta(u_0^S) \tag{12}$$

We will now describe various invariant descriptors presented by [1]:

2) 2D descriptor: One way we can describe symmetric points are as points which are the same distance away from edges on the object. The β_{2D} descriptor finds the edge points and calculates the distance between the sampled point, u_0 and its nearest edge point u_0^E in image coordinates. This can efficiently compute the cost for every pixel. However, when projecting a 3D ellipsoid to a 2D ellipse, distortion will occur, which may cause the symmetry property to no longer hold. Fig. 6 shows an instance where, even though $\|v_0 - v_0^E\| = \|v_0^S - (v_0^S)^E\|$ in world coordinates, distortion causes $\|u_0 - u_0^E\| \neq \|u_0^S - (u_0^S)^E\|$ in image coordinates.

3) 3D descriptor: The β_{3D} descriptor first back-projects the point to 2D coordinates, then measures the distance between that 3D point and its nearest edge point. This descriptor does satisfy that $\beta_{3D}(u_0) = \beta_{3D}(u_0^S)$ since $\|v_0 - v_0^E\| = \|v_0^S - (v_0^S)^E\|$. However, finding the nearest edge point in 3D space requires iterating over all points in 3D space, making it intractable in practice.

These two ways are not good enough in practice, so $\beta_{IDT}(\cdot)$ is proposed and has an assumption to make it efficient and well describe symmetry property. $\beta_{IDT}(\cdot)$ assumes that the nearest edge point in 2D will also be the nearest edge point in 3D, so it doesn't iterate all point in 3D but only finds the nearest edge points in 2D then projects it back to 3D space instead. In this way, v_0^E can be express by u_0^E as follow:

$$\begin{cases} u_0^E = argmin_{u \in U_E} \|u - u_0\| \\ v_0^E = argmin_{v \in V_E} \|v - v_0\| = \mathcal{P}^\dagger(u_0^E) \end{cases} \tag{13}$$

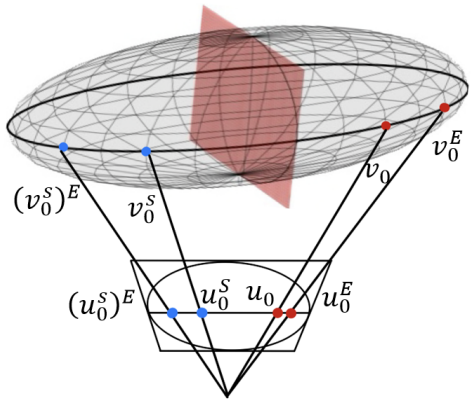


Fig. 6: Projection between image and world coordinate

Using the above equation to get the loss function:

$$f_{sym}(Q) = \Sigma_{u_i} (\|\mathcal{P}^\dagger(u_i^E) - v_i\| - \|\mathcal{P}^\dagger((u_i^S)^E)\|)^2 \tag{14}$$

In implement process, the Canny edge detection algorithm helps us to get objection edge features in the

bounding box, then the uniform sample point method will sample 25 points in each bounding box. As stated in SO-SLAM, making a query to record the corresponding nearest edge points before the optimization process is important to reduce the computational cost, and we can get the nearest edge point u_0^E in 2D rapidly. Furthermore, project the sampled point u_0 to a ray in 3D space and use equation 10 to get v_0 then use the symmetry plane to derive v_0^S. Finally, project v_0^S to image coordinate and find the nearest edge point $(u_0^S)^E$ by query and project it to 3D space $(v_0^S)^E$. After this process, we can calculate the error $f_{sym}(Q)$ for this ellipsoid Q and optimize its direction.

V. Experiment

We are unable to locate any TUM dataset with associated data online (i.e., each object possesses its own unique identifier), nor can we find ground truth information on object location, size, and orientation for object SLAM. Consequently, we manually annotate three segments of the dataset and obtain the actual location and size of the objects by reconstructing the point cloud. We make these datasets available online for others to use, which we believe can be considered a contribution. The three scenes include Fr1 Desk, Fr2 Desk, and Fr2 Dishes.The red ellipsoids are the ground truth labeled by hand, as shown in Fig.7.

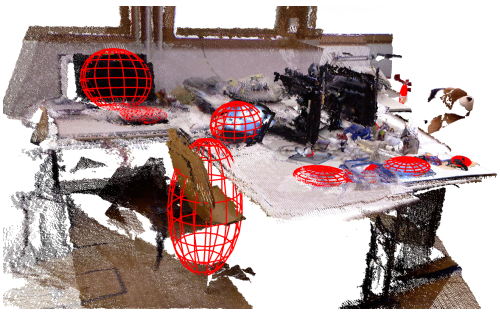


Fig. 7: Ground truth ellipsoids based on point cloud

The ellipsoids beyond the red one are the optimized ellipsoids used to represent the objects in the environment, as shown in Fig.8. By comparing the differences in position, size, and orientation between the red ellipsoids and ellipsoids of other colors, we can further assess the effectiveness of our algorithm.

Utilizing performance metrics Intersection over Union (IoU) and Rotation (Rot, in degrees), we comprehensively assess the efficacy of mapping methodologies. IoU quantifies the degree of overlap between the circumscribed cubes of the predicted object and the ground-truth object. In the case of objects

Fig. 8: Optimized ellipsoids comparing with ground truth

exhibiting symmetry, Rot (in degrees) measures the minimal rotational angle necessitated for aligning the three rotation axes of the estimated object with any axis of the ground-truth object, such that they form a straight line. For trajectory evaluation, the aforementioned metrics represent the mean values derived from the assessment of all objects.

For certain objects that appear within the camera's field of view and subsequently exit the viewing area temporarily, we quantify their presence by counting the number of frames in which they appear on the screen and calculating the mean values of two distinct metrics. We carry out experiments in three self-calibrated experimental scenarios: In Scenario 1, we calibrate seven objects; in Scenario 2, we calibrate an unspecified number of objects; and in Scenario 3, we calibrate six objects. We measure the Rotation (Rot) and Intersection over Union (IoU) of these objects. Please note that due to the absence of publicly available ground truth, there may be deviations in the size, orientation, and position of the objects we calibrate.

We conduct a comparative analysis between our experimental data and the data presented in the original paper. However, due to the limited number of scenarios tested (only three), a comprehensive comparison is not feasible. Our results reveal that the performance of our implementation is roughly on par with that of the original paper. The slight improvement observed in our case might be attributed to differences in the ground truth data employed, as well as variations in the types and quantities of objects used, which were not explicitly specified in the original paper.

Taking into account both the experimental outcomes and the visual quality of the 3D reconstructions, we believe that our replication of the code has been largely

TABLE I: TUM RGBD Fr1 Desk

Object	**Yaw** (°)	**Pitch** (°)	**IoU (%)**
Ass Book	9.6	2.0	67.30
Black Mouse	1.7	0.0	36.78
Blue Book	1.7	0.0	17.33
Bottle	58.5	0.0	9.91
Chair	26.3	0.0	54.74
Cube Book	6.0	0.0	9.80
Cup	12.1	20.0	61.31
Game Remote	9.6	2.0	59.81
Keyboard	58.5	0.0	0.00
Laptop	45.60	0.0	7.17
Laptop2	16.11	20.0	9.39
Monitor	10.17	0.0	46.55
White Book	12.1	20.0	39.16
White Mouse	6.0	0.0	12.61

TABLE II: Result Comparison Fr1 Desk

	#Obj	**Rot** (°)	**IoU (%)**
SO-SLAM	13	11.72	11.00
Quadric-SLAM	13	45.01	6.60
Ours	14	12.07	30.84

TABLE III: TUM RGBD Fr2 Desk

Object	**Yaw** (°)	**Pitch** (°)	**IoU (%)**
Monitor	8.82	14.99	42.68
Keyboard	4.04	0.05	40.37
Mouse	2.53	0.00	57.2
Book	9.6	2.0	44.82
Cup	58.5	0.0	37.6
Tape	26.3	0.0	59.12
Chair	9.1	21.5	11.32

TABLE IV: Result Comparison Fr2 Desk

	#Obj	**Rot** (°)	**IoU (%)**
SO-SLAM	12	8.8	19.80
Quadric-SLAM	12	44.67	13.00
Ours	7	10.60	41.87

TABLE V: TUM RGBD Fr2 Dishes

Object	**Yaw** (°)	**Pitch** (°)	**IoU (%)**
Cup	12.1	23.58	53.84
Bigger Bowl	6.0	0.0	56.73
Smaller bowl	1.7	0.0	66.98
Laptop	77.07	2.0	30.26
Monitor	40.70	0.0	34.94
Chair	50.7	0.0	9.27

TABLE VI: Result Comparison Fr2 Dishes

	#Obj	**Rot** (°)	**IoU (%)**
SO-SLAM	4	-	31.20
Quadric-SLAM	4	-	11.80
Ours	6	18.47	42.00

successful. It is important to note that during the implementation process, we prioritize accuracy over real-time performance. Consequently, we do not replicate the frame rate and real-time aspects mentioned in the original paper.

VI. Conclusion

We present an open-source Semantic-Object SLAM (SO-SLAM) implementation, addressing the unavailability of the original author's code. Our work prioritizes accuracy and utilizes various constraints within the GTSAM framework. We evaluated our implementation on three manually annotated segments of the TUM RGB-D dataset, comparing it with other methods such as Quadric-SLAM. Our results show that our SO-SLAM implementation performs on par with the original paper, with slight improvements due to differences in ground truth data and object variations. By providing detailed performance metrics, we demonstrate that our SO-SLAM outperforms other algorithms in terms of Intersection over Union (IoU) and Rotation (Rot). While our implementation prioritizes accuracy over real-time performance, our experimental results and 3D reconstruction quality indicate a successful replication of the SO-SLAM code. Our project contributes an open-source reference implementation for the research community, enabling further development and understanding of SLAM techniques that incorporate object semantics. YouTube video. https://www.youtube.com/watch?v=_yUy5nOtfMM.

REFERENCES

[1] Z. Liao, Y. Hu, J. Zhang, X. Qi, X. Zhang, and W. Wang, "So-slam: Semantic object slam with scale proportional and symmetrical texture constraints," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4008–4015, 2022.

[2] S. Yang and S. Scherer, "Cubeslam: Monocular 3-d object slam," *IEEE Transactions on Robotics*, vol. 35, no. 4, pp. 925–938, 2019.

[3] L. Nicholson, M. Milford, and N. Sünderhauf, "Quadricslam: Dual quadrics from object detections as landmarks in object-oriented slam," *IEEE Robotics and Automation Letters*, vol. 4, no. 1, pp. 1–8, 2018.

[4] K. Ok, K. Liu, K. Frey, J. P. How, and N. Roy, "Robust object-based slam for high-speed autonomous navigation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 669–675.