Sensor Localization by Few Distance Measurements via the Intersection of Implicit Manifolds

Michael M. Bilevich[1] Steven M. LaValle[2] Dan Halperin[1]

***Abstract*— We present a general approach for determining the unknown (or uncertain) position and orientation of a sensor mounted on a robot in a known environment, using only a few distance measurements (between 2 to 6 typically), which is advantageous, among others, in sensor cost, and storage and information-communication resources. In-between the measurements, the robot can perform predetermined local motions in its workspace, which are useful for narrowing down the candidate poses of the sensor. We demonstrate our approach for planar workspaces, and show that, under mild transversality assumptions, already two measurements are sufficient to reduce the set of possible poses to a set of curves (one-dimensional objects) in the three-dimensional configuration space of the sensor $\mathbb{R}^2 \times \mathbb{S}^1$, and three or more measurements reduce the set of possible poses to a finite collection of points. However, analytically computing these potential poses for non-trivial intermediate motions between measurements raises substantial hardships and thus we resort to numerical approximation. We reduce the localization problem to a carefully tailored procedure of intersecting two or more implicitly defined two-manifolds, which we carry out to any desired accuracy, proving guarantees on the quality of the approximation. We demonstrate the real-time effectiveness of our method even at high accuracy on various scenarios and different allowable intermediate motions. We also present experiments with a physical robot. Our open-source software and supplementary materials are available at https://bitbucket.org/taucgl/vb-fdml-public.**

I. Introduction

Robot localization is the task of a robot to determine its position and orientation inside the environment where it operates using data collected from sensors. As such, localization is a key ingredient in robot navigation which has received increased attention in recent years, since the position and orientation are necessary for planning a path of motion [1]. Localization can be carried out in various ways using a variety of sensors, which may be intrinsic (attached to the robot) or extrinsic (attached to the environment).

A common type of sensor for localization is distance-based, e.g., LiDAR. Many distance measurements do not only allow for localization inside a known environment but are also able to map an unknown environment. This is the task of the intensively investigated Simultaneous Localization And Mapping (SLAM), usually by equipping the robot with either a LiDAR sensor [2] or with a camera [3]. But equipping each robot with a LiDAR sensor or a camera might be expensive both in the price of each sensor and the hardware required to process the information, as well as in the network throughput of sending, receiving, and processing large point clouds.

As we focus in this paper on distance measurement, we will refer to *sensor localization* for short, with the understanding that the sensor is geometrically a point that is rigidly attached to a robot. While our approach is general, we will analyze it in detail for a sensor attached to a mobile robot translating and rotating in the plane. Hence the configuration space of the sensor is three-dimensional and we represent each configuration as the triple (x, y, θ). We will use the robot motions as part of our solution technique.

Note that instead of moving the robot between measurements, one can alternatively fix the pose of the robot and measure simultaneously from a few distinct sensors mounted on the robot—each sensor can be thought of as translating and rotating relative to some fixed point on the robot.

A. Related Work

1) Localization techniques: There are many approaches to localization, using various kinds of sensors and techniques. To localize in an already known environment, one can use particle filters and probabilistic methods [4], [5], [6]. We have already mentioned SLAM, which can be achieved for example by matching LiDAR samples [2] or by using a camera (known as visual SLAM, [3]). Recent works aim to speed up SLAM using GPUs [7], [8], [9]. One can also localize a robot by using RFID tags and sensors [1], or with RSSI signals [10].

2) Methods in meshing and implicit surface intersection: In the paper we deal with the problem of meshing an implicit surface, which is finding an explicit representation for the surface defined as the zeros of some function $f : \mathbb{R}^3 \to \mathbb{R}$, i.e., the set $\{x : f(x) = 0\}$. This can be done in several ways, including the marching-cubes algorithm [11], [12], dual contouring [13], which evaluates the function on some grid, or Delaunay refinement [14], which refines and filters a Delaunay triangulation of a three-dimensional point set. See [15] for a comprehensive survey.

We further deal with the intersection of implicit surfaces, which can also be carried out using a variety of techniques (see [16], [17], [18]). In this paper, we will present methods

[1]Blavatnik School of Computer Science, Tel-Aviv University, Israel. Work by M.B. and D.H. has been supported in part by the Israel Science Foundation (grant no. 1736/19), by NSF/US-Israel-BSF (grant no. 2019754), by the Israel Ministry of Science and Technology (grant no. 103129), by the Blavatnik Computer Science Research Fund, and by the Yandex Machine Learning Initiative for Machine Learning at Tel Aviv University.

[2]Center for Ubiquitous Computing, Faculty of Information Technology and Electrical Engineering, University of Oulu, Finland. Work by SL has been supported by a European Research Council Advanced Grant (ERC AdG, ILLUSIVE, 101020977) and the Academy of Finland (PERCEPT 322637).

that have provable guarantees on the error and convergence rate, when dealing specifically with robot/sensor localization.

B. Contribution

Our contributions are as follows:

- A general, efficient and robustly implemented method for sensor localization from few distance measurements.
- A conservative theoretical method which achieves a 100% success rate, and a heuristic simplification which achieves a high success rate in experiments.
- Theoretical guarantees on the quality of the approximation, with a linear bound $O(1/n)$ (where n is chosen by the user) on the error, which is the distance between the best pose estimate and the true pose of the robot.
- An open-source code suite which utilizes GPU and CPU multi-core parallelism for computing the few-distance-measurement localization in real time.

II. Problem Statement

The sensor is placed in the interior of a planar workspace $W \subseteq \mathbb{R}^2$. A distance measurement is a mapping

$$h : W \times \mathbb{S}^1 \to \mathbb{R}_+, \tag{1}$$

such that $h(x, y, \theta)$ is the length of the shortest segment connecting the position (x, y) with the boundary of W along a ray emanating from (x, y) in direction θ. The point (x, y) is the position of the sensor and the direction θ is the angle that the ray emanating from the sensor, along which the distance is measured, makes with the positive x-axis.

We are now ready to state the basic version of the problem that we study.

The problem: Given a workspace W, a set $g_1, \ldots, g_k$ of rigid-body transformations, and a set $d_1, \ldots, d_k$ of positive real values, find all the poses (x, y, θ) such that $(x, y) \in W$ and $d_i = h(g_i(x, y, \theta))$ for all $i \in [1, k]$.

We aim to find the configuration (x, y, θ), which is the original pose of the robot. Before each one of the k measurements, the robot moves to another pose or stays put. The pose of the sensor when making the ith distance measurement is $g_i(x, y, \theta)$, where, as just stated, (x, y, θ) is the original pose of the robot, which we aim to find.

As one can see in the figures in Section IV, even for one measurement, the set of all possible poses has a non-trivial shape and at a first glance might be hard to explain. Furthermore, even for simple instances, exact analysis (which is presented in the Supplementary Material) yields an algebraically non-trivial solution. As we wish to allow for more complex transformations (which is beneficial for reducing the candidate poses set) we give up on the exact approach and turn to a general, yet accurate, numerical method, which we describe in he following sections.

In this paper, we will deal with workspaces that have a polygonal boundary, namely polygons or polygons with holes. However, we believe that the techniques that we present here could be used for more involved workspaces for which we can evaluate the distance function $h(X)$, where X is the pose of the sensor.

III. Voxel-Based Approximation: The Method and Its Guarantees

We now reformulate our problem so that it reduces to the intersection of implicit manifolds. We will compute these intersections numerically, by first bounding the manifolds inside sets of voxels.

A. Problem reformulation as an intersection of implicit manifolds

Given some measurement d, we wish to find its preimage [19] $h^{-1}(d)$, namely the set of poses (position and orientation) of a sensor placed inside W from which we can measure a distance d to a wall of W. We will denote this preimage by M_d. Formally:

Definition 3.1: Given some distance measurement d, its *preimage* is defined as

$$M_d = \left\{(x, y, \theta) : h(x, y, \theta) = d, (x, y) \in W, \theta \in \mathbb{S}^1\right\}. \tag{2}$$

Theorem 3.1: For any polygonal workspace and for any $d > 0$, the preimage M_d is the finite (possibly empty) union of manifold patches, namely

$$M_d = \bigcup_j M_d^j \,, \tag{3}$$

where each M_d^j is either a 2-manifold (without a boundary) The proof is given in the Supplementary Material.

Henceforth, we will only deal with the case where the set M_d of manifold patches is not empty, since in our setting if the robot is able to read a measurement d, then its candidate poses consist of at least one valid configuration in M_d.

Observation 3.1: The preimage M_d is the zero set of the following piecewise-continuous function

$$f_d(x, y, \theta) = h(x, y, \theta) - d \,, \tag{4}$$

and thus the preimage has an implicit representation.

We will use the straightforward mapping $\phi : \mathbb{S}^1 \to [-\pi, \pi)$ and embed our configuration space in $\mathbb{R}^3$, to simplify further discussions and analysis. We are now ready to cast our problem in terms of *implicit manifold intersection*: Given a workspace W, a set $g_1, \ldots, g_k$ of rigid-body transformations $g_i : \mathbb{R}^3 \to \mathbb{R}^3$, and a set $d_1, \ldots, d_k$ of non-negative real values, find the intersection:

$$\bigcap_{i=1}^{k} M_{d_i} \,, \tag{5}$$

where M_{d_i} is the implicit manifold derived, as above, for measurement d_i after applying a transformation g_i with respect to the original pose (position and orientation) of the robot:

$$M_{d_i} = (f_{d_i} \circ g_i)^{-1}(\{0\}) \,. \tag{6}$$

B. Overview of the method

We assume that we are given a set of measurements d_i with corresponding rigid-body transformations $g_i : \mathbb{R}^3 \to \mathbb{R}^3$. We are also given a parameter n, which determines the resolution and accuracy of our computation: we split a bounding cube of the configuration space into $n \times n \times n$ voxels. Our method comprises the following steps:

1) For each measurement d_i, we bound the two-dimensional preimage M_{d_i} inside a set of (three-dimensional) voxels, which we will refer to as a *voxel cloud*, i.e., a union of pairwise disjoint voxels, with an edge length of $\frac{1}{n}$. We define voxel clouds formally in Section III-C below.
2) We intersect two or more voxel clouds by keeping only the voxels that appear in all of the clouds.
3) If there are more than two preimages, then the resulting intersection should be a union of dot-like clusters of voxels (where each is purportedly representing a single point of intersection of manifolds), from which we can extract estimations for possible poses of the robot.

We will also show that this method is complete, in the sense that the true pose of the robot is contained in the voxel-cloud approximation. Additionally, we will show that the L_2 distance between the true pose of the robot and the closest estimation returned by the algorithm is bounded by $O\left(\frac{1}{n}\right)$.

In the following subsections, we will present our method in detail. We note that most operations presented are local; hence, parallelization (e.g., using GPUs) can be applied, as we demonstrate below in the experiments section.

C. Definitions of voxel grids and clouds

We now formally define voxels, voxel grids (and their representation as a graph) and voxel clouds. We assume below that the hyper-parameter n (and its corresponding edge length $\frac{1}{n}$) are fixed.

Definition 3.2: Let $(i,j,k) \in \mathbb{Z}^3$. We define the *voxel* in location (i,j,k) to be the subset $C_{1/n}(i,j,k) \subseteq \mathbb{R}^3$:

$$C_{1/n}(i,j,k) \coloneqq \left[\frac{i}{n}, \frac{i+1}{n}\right) \times \left[\frac{j}{n}, \frac{j+1}{n}\right) \times \left[\frac{k}{n}, \frac{k+1}{n}\right) . \tag{7}$$

Note that for different coordinates $(i,j,k) \neq (i',j',k') \in \mathbb{Z}^3$, $C_{1/n}(i,j,k) \cap C_{1/n}(i',j',k') = \emptyset$, and they form a partition of $\mathbb{R}^3$. For a point $p \in \mathbb{R}^3$, denote the unique voxel containing it by $C_{1/n}(p)$.

Definition 3.3: Two distinct voxels $C_{1/n}(i,j,k)$, $C_{1/n}(i',j',k')$ are said to be *neighbors* if:

$$(i,j,k) - (i',j',k') \in \{-1,0,1\}^3 . \tag{8}$$

Henceforth, we will identify each voxel with its corresponding coordinates in $\mathbb{Z}^3$. Denote the set of all voxels by $\mathcal{C}_{1/n}$, and thus we can identify $\mathcal{C}_{1/n} \simeq \mathbb{Z}^3$. The relation of neighboring voxels also gives rise to a graph structure on the voxels:

Definition 3.4: The $\frac{1}{n}$*-voxel grid graph* is a graph whose vertices correspond to the voxels $\mathcal{C}_{1/n}$ and an edge in the graph between two vertices exists if their corresponding voxels are neighbors.

Definition 3.5: A *voxel cloud* is a union of voxels $V \subseteq \mathcal{C}_{1/n}$.

D. Step I: Bounding preimages within voxel clouds

Assume that our domain is an axis-aligned box $B \subseteq \mathbb{R}^3$, e.g., $B_{xy} \times [-\pi, \pi]$ where B_{xy} is an axis-aligned bounding box for the workspace $W \subseteq \mathbb{R}^2$. In this section, for simplicity, we assume that B is the unit cube $[0,1] \times [0,1] \times [0,1] \subseteq \mathbb{R}^3$.

Our goal in this step is to find a voxel cloud $\hat{M}_{d_i,n}$ which approximates the preimage M_{d_i}.

Recall that we overlay the domain B with an $n \times n \times n$ voxel grid. For each measurement d_i, compute the distance function h (composed with the associated rigid-body transformation g_i) at each vertex $v \in C_{1/n}$ of each cube in the grid. Then if for a given measurement d_i we have two vertices v_1, v_2 of the same voxel $C_{1/n}(i,j,k)$ with $h(v_1) \geq d_i$ and $h(v_2) \leq d_i$, we add that $C_{1/n}(i,j,k)$ to a voxel cloud $\hat{M}'_{d_i,n}$. If we are in a continuous sub-domain of h (see below for more details), from the intermediate value theorem we know that the manifold should intersect that voxel and hence we add it to the voxel cloud $\hat{M}_{d_i,n}$.

Note that checking whether $h \circ g_i \geq d_i$ is equivalent to checking whether $f_{d_i} \circ g_i \geq 0$ by definition of f_{d_i}, and the same is true for $\leq$. The pseudo-code for this step is presented in the Supplementary Material.

This method, which we call *marching voxels*, can be viewed as a variant of the celebrated marching-cubes algorithm [20]. The current method ignores the topology of the manifold and instead takes the entire bounding voxel, as opposed to the marching-cubes algorithm, which for each voxel computes a triangle soup that represents the intersection of the manifold with that voxel. In our case, we wish to get a set containing the preimage, hence we consider entire voxels.

Assuming that we take n to be sufficiently large, our voxel cloud approximation $\hat{M}_{d_i,n}$ of the preimage M_{d_i} is both complete and geometrically close to the actual preimage, as we assert in the following Theorems 3.2 and 3.3. Although the algorithm is simple, proving its correctness is non-trivial and is deferred to the Supplementary Material.

Theorem 3.2: (Completeness) For a sufficiently large n, the voxel cloud $\hat{M}_{d_i,n} \subseteq \mathbb{R}^3$ returned by the marching voxels algorithm contains the preimage M_{d_i}, namely

$$M_{d_i} \subseteq \hat{M}_{d_i,n} . \tag{9}$$

Theorem 3.3: (Proximity) For a sufficiently large n, the Hausdorff distance H between the preimage M_{d_i} and its voxel cloud approximation $\hat{M}_{d_i}$ satisfies

$$H\left(M_{d_i}, \hat{M}_{d_i}\right) \leq \frac{\sqrt{3}}{n} . \tag{10}$$

The proofs of the theorems rely on our knowledge of the algebra of the underlying manifolds that we approximate. However, working with the exact algebraic representations becomes impractical when we proceed to intersect two or more manifolds. In practice, we allow for the existence

of a relatively small set $A_{n,W}$ of voxels (which depends on n and the workspace W, but not on the measurement d_i), which we call "blind spots" and for which the two theorems above do not hold. Namely, we may miss a solution configuration if it is contained in $A_{n,W}$, and we cannot guarantee that points in $A_{n,W}$ are Hausdorff close to the actual preimage. In the Supplementary Material we show how to theoretically guarantee completeness also for the voxels in $A_{n,W}$. Furthermore, we suggest a simple-to-implement and computationally fast heuristic, for which we show in experiments that we obtain 100% success rate despite ignoring the blind spots: we append the neighbors of each voxel returned by the marching voxel algorithm to the output cloud.

E. Step II: Intersecting voxel clouds

We use the same voxel grid for approximating each of the preimages $\hat{M}_{d_i}$. Our goal now is to approximate their intersection. Here as well the approximation will consist of a voxel cloud that contains the intersection. It suffices to intersect the approximating voxel clouds from the previous step: we return a voxel cloud approximation of the intersection as a collection of all voxels that appear in all of the preimage voxel approximations, i.e., $\bigcap_{d_i} \hat{M}_{d_i}$.

Like in the previous step, we also have similar claims for completeness and proximity. More details and proofs are given in the Supplementary Material.

Theorem 3.4: (Completeness) The intersection of preimages is contained in our voxel cloud approximation of the intersection:

$$\bigcap_{d_i} M_{d_i} \subseteq \bigcap_{d_i} \hat{M}_{d_i}. \tag{11}$$

Theorem 3.5: (Proximity) The Hausdorff distance between the intersection of preimages and its voxel approximation is $O\left(\frac{1}{n}\right)$:

$$H\left(\bigcap_{d_i} M_{d_i}, \bigcap_{d_i} \hat{M}_{d_i}\right) \leq \frac{\sqrt{3}}{n}. \tag{12}$$

F. Step III: Cleaning up the estimation

We assume that we have three or more measurements ($i \geq 3$). After we get a voxel approximation $\bigcap_{d_i} \hat{M}_{d_i}$ of a point set, we wish to report points in $\mathbb{R}^3$ (which represent $\mathbb{R}^2 \times \mathbb{S}^1$ as position and orientation), which will be our estimates for the robot's pose. We can then compute the connected components of the voxel approximation $\bigcap_{d_i} \hat{M}_{d_i}$ using the graph representation defined above, and for each connected component we compute the *center of mass* to get a point set $S \subseteq \mathbb{R}^3$.

For each estimated point $q \in S$, we compute the distance measurements resulting from placing the sensor in the estimated pose q and applying the corresponding rigid-body transformations and get a new measurements $\hat{d}_i(q)$. We compute the mean square error for the point q:

$$\mathcal{L}(q) = \sum_i \left(d_i - \hat{d}_i(q)\right)^2. \tag{13}$$

Finally, we return the best points (with $\mathcal{L}(q)$ as a metric) from S.

Note that our upper bound on the error (Theorem 3.5) is now changed, as the center of mass we return might not be in the same voxel where the correct solution lies. Hence this bound is modified as follows:

Lemma 3.6: Let $p \in \mathbb{R}^3$ be the pose of the robot. Then, if for some $q \in S$, the point p lies inside the volume of the corresponding connected component V, then:

$$||p - q|| \leq \operatorname{diam}(V) = \sup_{a,b \in V} ||a - b||. \tag{14}$$

We show the following bounds on the diameter:

Lemma 3.7: The diameter of a connected component V with ρ voxels is bounded as follows:

$$2\left(\frac{3}{4\pi}\rho\right)^{\frac{1}{3}} \cdot \frac{1}{n} \leq \operatorname{diam}(V) \leq \rho\sqrt{3} \cdot \frac{1}{n}. \tag{15}$$

Depending on the value of ρ the upper bound might be large (although when ρ is $O(1)$, both upper bounds are still $O\left(\frac{1}{n}\right)$). But as we demonstrate in the experiments below, the actual distance is closer to the lower bound on $\operatorname{diam}(V)$ in Equation 15. The proofs can be found in the Supplementary Material.

G. Dealing with Measurement Uncertainty

In practice, we deal with inaccuracies in both the mapping of the workspace and in the distance returned by the sensor. We will account for the latter, as the former can be viewed as another error in the distance measurement. Formally, assume that we get a noisy distance measurement function $\hat{h}$ instead of the actual distance h from a wall of the room.

Assume that we are given an upper bound ε_M on the measurement error such that for any $(p_x, p_y, p_\theta) \in W \times \mathbb{S}^1$:

$$\left|\hat{h}(p_x, p_y, p_\theta) - h(p_x, p_y, p_\theta)\right| \leq \varepsilon_M \ . \tag{16}$$

We can then show that if the sensor has sufficient clearance (of ε_M units) from the walls of the room, our estimate for the localization has an error of at most:

$$\frac{\sqrt{3}}{n} + \varepsilon_M. \tag{17}$$

The proof of his bound is presented in the Supplementary Material.

We do not discuss here the case of a stochastic error. While we hypothesize that the analysis is similar, it is left for future work.

IV. Experiments and Results

A. Implementation Details

The code is written as a C++ library and has Python bindings. We used OpenCL [21] to compute the marching voxels and the manifold intersection in parallel. The code was run on a macOS machine with Intel Core i5 CPU and Intel HD Graphics 6000 GPU, on a macOS machine with Apple M1 Pro and on a Windows machine with Intel Core i9-10850K CPU and NVIDIA RTX 3090 GPU. Running times

for all systems are presented below. The parallelization of the marching voxels using a GPU was inspired by [22].

We demonstrate the performance of our algorithm on the following test scenes:

- `square-room`: A square room with dimensions of $2[m] \times 2[m]$.
- `lab`: A polygon based on a LiDAR scan of our lab. Axis-parallel bounding box dimensions: $6.5[m] \times 5[m]$.
- `floor-plan`: Plan of the floor where our lab is located. Axis-parallel bounding box dimensions: $40[m] \times 20[m]$.
- `random`: Randomly generated non-convex polygons. Axis-parallel bounding box dimensions: $2[m] \times 2[m]$.

B. Examples for 2 distance measurements

For each scene, we assume the robot has sampled a distance of d_1, rotated in place π radians counter-clockwise, and then sampled a distance of d_2.

In the 2D images in Figure 1, black lines represent the room walls, red lines represent the query box and blue curves are the projection of the preimage intersection curves.

In Figures 1a, 1c and 1f we visualize the results of the preimage intersection, projected onto the workspace. Figures 1b, 1e and 1j depict the embedding of the preimages and their intersection in $\mathbb{R}^3$. The plane parallel to the view is the workspace (xy-plane) and the depth of the image (the blue axis) is the angle θ. The preimage M_{d_1} is drawn in red, M_{d_2} in blue and their intersection $M_{d_1} \cap M_{d_2}$ in purple. In Figure 1d we zoom in on a particular portion of the workspace `floor-plan` to obtain more accurate results, as the same value of n for the resolution yields smaller voxels in the workspace.

C. Examples for 3, 4 and 5 distance measurements

Figures 1g through 1i are examples of pose estimations for the scene `lab` after performing $k = 3, 4, 5$ measurements. The robot is placed at the bottom left corner of the room, represented as a green dot, and we took 5 measurements, in increments of $\pi/4$. Each estimation is presented as an orange arrow whose base indicates the estimated position of the robot in the workspace and the direction is the estimated orientation of the robot. In all figures we see that the true position of the robot is indeed in the estimated set returned for each k.

D. Error and Success Rate in Simulations

To test the error and success rate of our method, we performed the following simulation. Sample a random point p in some workspace $W \subseteq \mathbb{R}^2$ and a random orientation $\theta \in \mathbb{S}^1$. We call this pose the "ground truth". We then measure the distance to the walls from the point p with orientations $\theta + (i-1) \cdot \pi/4$ for $i = 1, \ldots, 6$ and random uniform offsets in x and y coordinates, to get measurement $d_1, \ldots, d_6$, and run our method to get a set of pose estimations S. We return the point $s_{\mathcal{L}}$ in S with the lowest value of $\mathcal{L}$, as detailed in Section III-F. We also return the point $s_{\mathcal{W}}$ in S whose Euclidean distance in $\mathbb{R}^3$ to the ground truth pose is minimal. If the set S is empty or for the returned pose the Euclidean distance in $\mathbb{R}^3$ to the ground truth is farther than twice the upper bound we present in Lemma 3.7, we consider this as a failure. Otherwise, we measure the distance between the ground truth pose and the best estimation to get the approximation error.

We performed this experiment on the workspaces `lab`, `floor-plan` and `random`, for grid resolutions of $n \in \{50, 75, 100, 125, 150, 175, 200\}$. For each workspace and for each n, results were averaged over 100 trials. We measure the success rate for the estimation achieving the lowest value of $\mathcal{L}$, which we call the $\mathcal{L}$-success rate, and the success rate and error for the estimation with lowest Euclidean distance in $\mathbb{R}^3$ from the ground truth, which we call the $\mathcal{W}$-success rate and the $\mathcal{W}$-error rate respectively. The $\mathcal{L}$-success rate is the performance of our proposed method in practice, while the $\mathcal{W}$-success rate tests the completeness of the returned set of predictions. Error rate is the Euclidean distance in $\mathbb{R}^3$ to the ground truth, when the workspace W and $\mathbb{S}^1$ are both normalized to the unit cube $[0, 1] \times [0, 1] \times [0, 1] \subseteq \mathbb{R}^3$ (so that the units are comparable).

Table I shows the average success rate for each workspace for both predictions, as well as the average number of pose estimates generated by the algorithm. The number of pose estimates is a combination of symmetry in the polygons and error incurred by our post-processing heuristic. As we can see, the consistent 100% $\mathcal{W}$-success rate suggests that our algorithm indeed satisfies completeness in the sense that the returned set of estimations also contains the ground truth estimation, justifying our post-processing heuristic. Note that the workspace `floor-plan` is highly symmetrical since many rooms are identical to each other, hence lower success rate is expected for this scene.

Figure 2a shows the $\mathcal{W}$-error rate for each workspace as a function of n. For reference, we also present a crude upper bound, denoted by $\mathrm{UB}(n)$, which is based on the lower estimate in Equation 15 when taking a value of $\rho = 510$. The choice of this value of ρ is based on an experiment which is described in detail in the Supplementary Material. As we can see, $\mathrm{UB}(n)$ is an upper bound for the $\mathcal{W}$-error rate, and the error rates is $O\left(\frac{1}{n}\right)$. Figure 2b compares the running time for the `lab` workspace, for different values of n on the two different hardware systems. Running time for the other workspaces is similar.

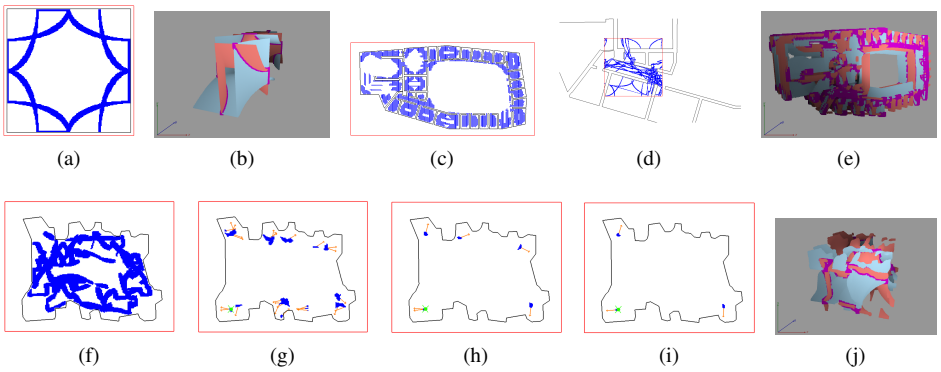
E. Physical Robot Experiments

1) Error in localization: As a proof of concept, we also evaluated our method on a physical robot. We used a DJI RoboMaster EP Core, equipped with a distance sensor in

TABLE I

SUCCESS RATES FOR $k = 6$, FOR DIFFERENT WORKSPACES.

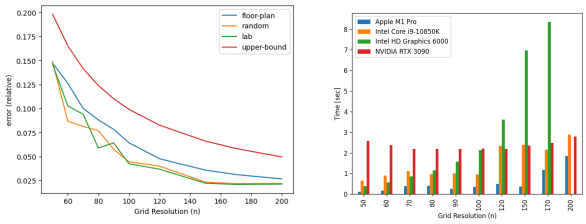
Workspace	$\mathcal{L}$-Success Rate	$\mathcal{W}$-Success Rate	# Estimates
`random`	90.14%	100%	31.92
`lab`	95.28%	100%	10.73
`floor-plan`	80.43%	100%	88.77



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

Fig. 1. Various screenshots from our software, in 2D (projected preimages) and in 3D space. Figures 1a and 1b correspond to the scene `square-room`, figures 1c, 1d and 1e correspond to `floor-plan` and figures 1f, 1g, 1h, 1i and 1j correspond to `lab`. See text for detailed explanations.



(a) (b)

Fig. 2. On the left: $\mathcal{W}$-Error rate as function of n, for different workspaces. The theoretical upper bound on the error $\mathrm{UB}(n)$ is also drawn for a reference. On the right: running time (in seconds) for different values of n for the `lab` workspace.

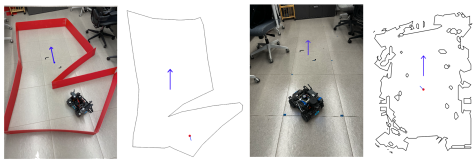


Fig. 3. Two examples of localization results in real life. Left: localization in a polygonal room bounded by cardboard (red) walls. Right: localization in the laboratory itself. In both examples the left column is a picture of the robot in the respective room, and the right column is a sketch of the corresponding best pose estimate returned by the algorithm. A blue arrow is drawn as a reference guide. The origin is marked with black tape on the floor.

the front and back. At first we conducted an automatic test: in the `lab` workspace (which is also a LiDAR scan of our lab), we chose a random point and asked the robot to advance to that point. Then we carried out in the physical setting a similar experiment as described in Section IV-C, i.e., $k = 3$ measurements with rotation of $\pi/4$ and taking measurements from the front and back of the robot, yielding overall 6 distance measurements. We compare the estimate returned by our algorithm which minimized the Euclidean distance to the ground truth pose. Averaging over 50 experiments, we achieve an average error of $0.0649[m]$ in the workspace (i.e., the Euclidean distance between the position (x, y coordinates) of the estimate and the ground truth), and an average error of 0.0676 radians in orientation. We note that the LiDAR that we used to create the map has an error of up to $0.05[m]$.

We also created simpler polygonal rooms out of cardboard, placed the robot at random points and carefully measured its distance with respect to the origin. After averaging over 20 experiments, we achieve an average error of $0.026[m]$ in the workspace and 0.107 radians in orientation.

For both experiments we chose a resolution value of $n = 200$, which leads to about $0.041[m]$ of error for the x, y coordinates and 0.0314 radians of error.

See Figure 3 for examples of robot placement in these experiments and the pose estimates returned by the algorithm.

2) Comparison against SLAM: As a comparison, we carried out the following experiment: we placed the robot in pre-determined landmarks in the room for which we measured their location manually. We then queried the pose from our algorithm and the pose estimate resulting from the SLAM computation of Google Cartographer [23]. Finally, we measured the Euclidean distance between each pose estimate from the ground truth (the true location of the landmark), and the Euclidean distance between both estimates as well. Averaging on 10 experiments, the Google Cartographer had an error of 0.02377[m] and our method had an error of 0.06[m]. The average Euclidean distance between both pose estimates is 0.05744[m].

We note that both methods incorporate the robot's odometry, which is error prone. However, Google Cartographer also used the robot's IMU.

V. Conclusions and Further Work

In this work we have presented a numerical method for estimating the unknown pose of a robot inside a known environment using only a few (2 to 6) distance measurements. We show that our method returns conservative results, and can be applied in real-life situations.

In future work, we would like to improve the accuracy and success rate in practice, deal with stochastic errors and errors in odometry, and extend our method to higher dimensions. A related problem that may be solved using similar techniques is finding the optimal motions that the robot needs to perform to reduce the volume of the possible poses.

REFERENCES

[1] A. Motroni, A. Buffi, and P. Nepa, "A survey on indoor vehicle localization through RFID technology," *IEEE Access*, vol. 9, pp. 17 921–17 942, 2021.

[2] Q. Zou, Q. Sun, L. Chen, B. Nie, and Q. Li, "A comparative analysis of LiDAR SLAM-based indoor navigation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6907–6921, 2022.

[3] M. Servières, V. Renaudin, A. Dupuis, and N. Antigny, "Visual and visual-inertial SLAM: State of the art, classification, and experimental benchmarking," *Journal of Sensors*, vol. 2021, 2021.

[4] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, "Monte carlo localization for mobile robots," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 2, 1999, pp. 1322–1328 vol.2.

[5] M. Speekenbrink, "A tutorial on particle filters," *Journal of Mathematical Psychology*, vol. 73, pp. 140–152, 2016.

[6] X. Wang, T. Li, S. Sun, and J. M. Corchado, "A survey of recent advances in particle filters and remaining challenges for multitarget tracking," *Sensors*, vol. 17, no. 12, 2017. [Online]. Available: https://www.mdpi.com/1424-8220/17/12/2707

[7] T. Ma, N. Bai, W. Shi, X. Wu, L. Wang, T. Wu, and C. Zhao, "Research on the application of visual SLAM in embedded GPU," *Wireless Communications and Mobile Computing*, vol. 2021, 2021.

[8] M. Abouzahir, R. Latif, M. Ramzi, and M. Sbihi, "OpenCL and OpenGL implementation of simultaneous localization and mapping algorithm using high-end GPU," in *ITM Web of Conferences*, vol. 46. EDP Sciences, 2022, p. 04001.

[9] R. Mittal, V. Pathak, and A. Mithal, "A novel approach to optimize SLAM using GP-GPU," in *Proceedings of International Conference on Data Science and Applications*. Springer, 2021, pp. 273–280.

[10] S. R. Jondhale, R. Maheswar, and J. Lloret, "Survey of existing RSSI-Based L&T systems," in *Received Signal Strength Based Target Localization and Tracking Using Wireless Sensor Networks*. Springer, 2022, pp. 49–64.

[11] M. Botsch, L. Kobbelt, M. Pauly, P. Alliez, and B. Levy, *Polygon Mesh Processing*. CRC Press, 2010. [Online]. Available: https://books.google.co.il/books?id=AuXqBgAAQBAJ

[12] E. Chernyaev, "Marching cubes 33: Construction of topologically correct isosurfaces," Tech. Rep., 1995.

[13] T. Ju, F. Losasso, S. Schaefer, and J. Warren, "Dual contouring of hermite data," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 2002, pp. 339–346.

[14] J.-D. Boissonnat and S. Oudot, "Provably good sampling and meshing of surfaces," *Graphical Models*, vol. 67, no. 5, pp. 405–451, 2005, solid Modeling and Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1524070305000056

[15] J.-D. Boissonnat, D. Cohen-Steiner, B. Mourrain, G. Rote, and G. Vegter, *Meshing of Surfaces*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 181–229. [Online]. Available: https://doi.org/10.1007/978-3-540-33259-6_5

[16] S. Tanaka, Y. Fukuda, and H. Yamamoto, "Stochastic algorithm for detecting intersection of implicit surfaces," *Computers & Graphics*, vol. 24, no. 4, pp. 523–528, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0097849300000558

[17] O. Caprani, L. Hvidegaard, M. Mortensen, and T. Schneider, "Robust and efficient ray intersection of implicit surfaces," *Reliable Computing*, vol. 6, no. 1, pp. 9–21, 2000.

[18] X. Li, H. Jiang, S. Chen, and X. Wang, "An efficient surface–surface intersection algorithm based on geometry characteristics," *Computers & Graphics*, vol. 28, no. 4, pp. 527–537, 2004.

[19] S. M. LaValle *et al.*, "Sensing and filtering: A fresh perspective based on preimages and information spaces," *Foundations and Trends® in Robotics*, vol. 1, no. 4, pp. 253–372, 2012.

[20] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer graphics (New York, N.Y.)*, vol. 21, no. 4, pp. 163–169, 1987.

[21] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Computing in Science & Engineering*, vol. 12, no. 3, pp. 66–73, 2010.

[22] J. Lei and K. Jia, "Analytic marching: An analytic meshing solution from deep implicit surface networks," 2020. [Online]. Available: https://arxiv.org/abs/2002.06597

[23] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LIDAR SLAM," in *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2016, pp. 1271–1278.