The Foreseeable Future: Self-Supervised Learning to Predict Dynamic Scenes for Indoor Navigation

Hugues Thomas, *Member, IEEE,* Jian Zhang, *Member, IEEE,* Timothy D. Barfoot, *Fellow, IEEE*

***Abstract*—We present a method for generating, predicting, and using Spatiotemporal Occupancy Grid Maps (SOGM), which embed future semantic information of real dynamic scenes. We present an auto-labeling process that creates SOGMs from noisy real navigation data. We use a 3D-2D feedforward architecture, trained to predict the future time steps of SOGMs, given 3D lidar frames as input. Our pipeline is entirely self-supervised, thus enabling lifelong learning for real robots. The network is composed of a 3D back-end that extracts rich features and enables the semantic segmentation of the lidar frames, and a 2D front-end that predicts the future information embedded in the SOGM representation, potentially capturing the complexities and uncertainties of real-world multi-agent interactions. We also design a navigation system that uses these predicted SOGMs within planning, after they have been transformed into Spatiotemporal Risk Maps (SRMs). We verify our navigation system's abilities in simulation, validate it on a real robot, study SOGM predictions on real data in various circumstances, and provide a novel indoor 3D lidar dataset, collected during our experiments, which includes our automated annotations.**

***Index Terms*—Learning and Adaptive Systems, Reactive and Sensor-Based Planning, Deep Learning in Robotics and Automation, Indoor Navigation**

I. Introduction

PREDICTING the future has always fascinated humanity. From the Oracle of Delphi to Paul the Octopus, this curiosity for the unknown has never faded. But we tend to forget that we already predict the future constantly in our daily lives, only it is for a short horizon. Walking in the street, catching a falling object, or driving a car, all these actions require a certain level of anticipation. With practice, humans can become quite good at predicting what might happen for the next few seconds in many situations; what about robots?

We study this question in the context of a concrete example: a robot learning on its own to navigate among humans or dynamic objects in an indoor space. Our approach allows the robot to predict the location of obstacles in a short future horizon (a few seconds), and plan its way around them. A deep neural network predicts these locations as Spatiotemporal Occupancy Grid Maps (SOGMs), which contain occupancy probabilities in space an time, as shown in Figure 1. We use Self-Supervised Learning, which means the training data and annotations are collected automatically. After the robot navigated in a dynamic scene, our annotation pipeline can label 3D lidar points with semantic information and generate past SOGMs, without any human annotation. We supervise the training of our network with this annotated data. Then the robot can navigate with our network prediction integrated in the navigation system, and thus anticipate the movements of dynamic obstacles. In this paper, we provide a detailed description of the collection of algorithms required for these various tasks, for a complete view of the overall approach, as illustrated in Figure 2.

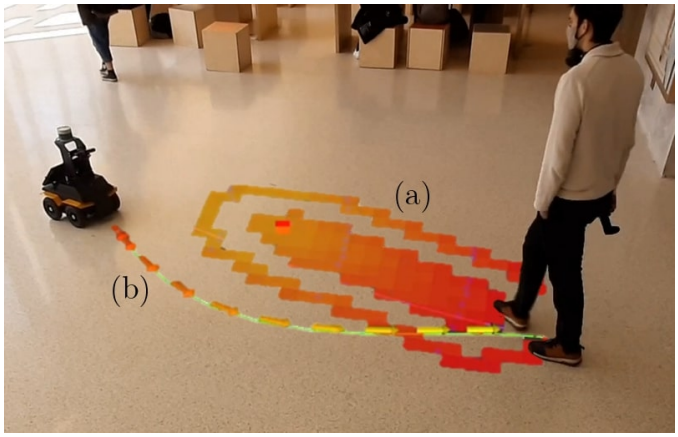


Fig. 1. Our robot navigating in a real dynamic scene. Future occupied locations are predicted as Spatiotemporal Occupancy Grid Maps (a) and the robot plans a trajectory to avoid them (b). Time is represented as a color, from red (now) to yellow (future). The ring and center areas represent low and high occupancy probabilities respectively.

Some of the algorithms we use have already been introduced in two of our previous works. In the first one [1], we described how to automatically annotate 3D lidar points, and train a deep network to predict these 3D labels. In the second one [2], our system learned to predict the future of dynamic scenes as SOGMs. Until now, we only evaluated results in a simulated environment. In this work, we build on these two previous papers and improve our approach with **three novel contributions**:

- a new lidar and SOGM automated annotation pipeline working with real noisy lidar data.
- a new closed-loop navigation system using our network predictions.
- experiments on a real robot, with the data published as an open dataset.

After a literature review in **Section II**, where we highlight the uniqueness of our approach, we define the building blocks used in different parts of our approach in **Section III**. Our localization and mapping method PointMap, our main annotation tool that estimates occupancy probabilities PointRay, and

Hugues Thomas and Jian Zhang are with Apple, Cupertino, USA. Timothy D. Barfoot is with the Institute for Aerospace Studies (UTIAS), University of Toronto, Canada. This work has been conducted while Hugues Thomas was with the Institute for Aerospace Studies (UTIAS), University of Toronto, Canada.

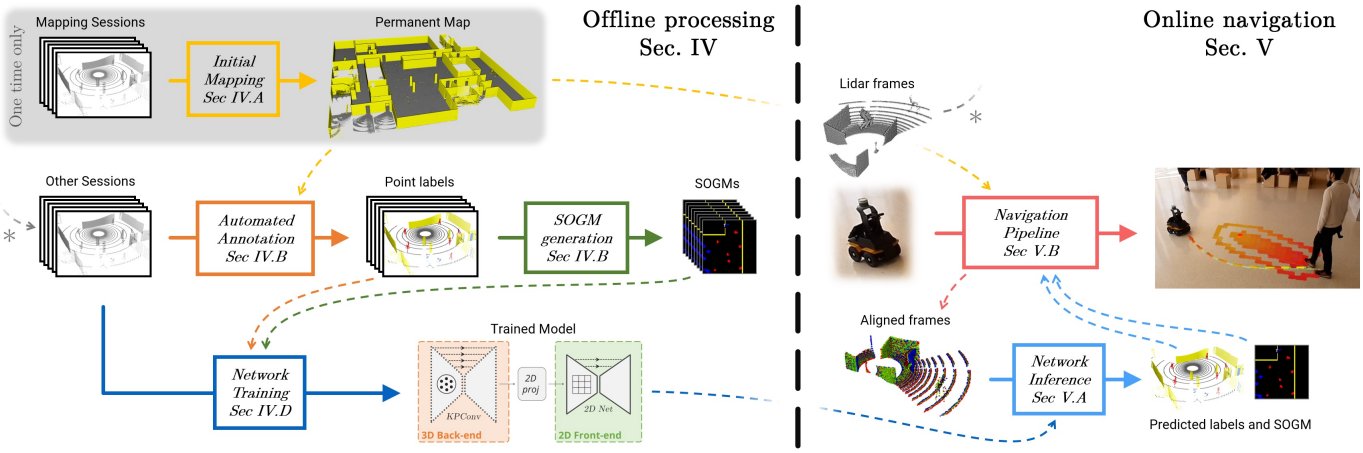


Fig. 2. Our approach aims at robots navigating in the same environment repeatedly. Initially, we create a point cloud map of the environment. Then we alternate offline processing, where a network is trained on the collected data, and online navigation, where the robot collects data (whether it uses the predictions or not).

point-cloud mathematical morphology operators to help reduce the noise in the annotations.

We dedicate **Section IV** to the offline half of our approach. First, a 3D point-cloud map of the environment is created. Then, our lidar annotation algorithm identifies four lidar point labels: *ground*, *permanent* (structures such as walls), *movable* (still but movable obstacles such as chairs), and *dynamic* (moving objects such as people); and generates SOGMs. Finally, this labeled data is used for the training of our 3D-2D feedforward architecture, which only takes three consecutive lidar frames as input, and predicts 3D point labels and the future SOGM. **Section V** focuses on our online navigation system. The network inference is post-processed to obtain Spatiotemporal Risk Maps (SRM) and 3D point labels, that easily connect and interact with the rest of the navigation system.

The simulation experiments, listed in **Section VI**, provide quantitative evaluations of our navigation system, as they allow the use of ground truth, and multiple repetitions. We compare the efficiency and safety of our navigation system when using different types of predictions. The experiments we conduct in the real world (**Section VII**) are crucial to validate that our method generalizes to real applications. We study the predicted SOGMs quantitatively and qualitatively, and provide anecdotal examples of navigation on a real robot.

Our results are best viewed in the supplementary video[1]. In addition, we publish the data collected during our experiments as **a new dataset: UofT-Indoor-3D (UTIn3D)**[2]. It includes the lidar frames, the localization, and the labels provided by our automated annotation approach. We believe it will be valuable to the community given the rarity of indoor 3D lidar datasets with many pedestrians. Along with the dataset, we aim to facilitate the reproduction of our results and encourage research in this direction, with detailed open-source code[3].

[1]Video: https://huguesthomas.github.io/tro_2022_video.html
[2]Dataset: https://github.com/utiasASRL/UTIn3D
[3]Code: https://github.com/utiasASRL/Crystal_Ball_Nav

II. Related Work

Navigation around dynamic obstacles is well studied in robotics. In terms of predicting obstacles' future motions, [3], [4] learned a distribution of possible pedestrian trajectories using inverse optimal control to recover a set of agent preferences consistent with prior demonstration. Following these preliminary works, various solutions to dynamic obstacle forecasting have been explored, that we study in this literature review.

Our work is unique compared to other deep-learning-based approaches for navigation in dynamic scenes. It stands out because of three crucial properties:

- **Self-Supervised**: We use annotation generated automatically and not by humans.
- **End-to-end**: Our predictions do not rely on other algorithms such as object detection and tracking.
- **Pointwise/non-parametric**: the computation cost of our method does not increase with the number of agents in the scene.

A. Mapping, Occupancy Probabilities, and Navigation

ICP-based simultaneous localization and mapping (SLAM) algorithms are widely used in robotics, with many variants [5]–[8]. We designed our PointMap SLAM with a focus on simplicity and efficiency. Similarly to [8], we keep a point cloud with normals as the map, but we update the normals directly on the lidar frames with spherical coordinate neighborhoods. Our approach targets the problem of robots that navigate in the same environment repeatedly. Therefore we prefer to use PointMap as a mapping tool first and then only rely on the frame-to-map alignment for localization.

Computing occupancy probabilities with ray-casting is also a common technique in the literature. Used at first for 2D occupancy grid mapping [9], it was later adapted for 3D mapping [10], [11]. In our case, PointRay computes occupancy probabilities on a point cloud instead of a grid, similarly to [12], and therefore only models free space where points have been measured. Our main addition is the notion of

dynamic and movable labels, which we get by combining multiple sessions. Other minor differences with [12] include a simplification of the probability update rules and the use of frustums instead of cones around lidar rays.

Combining occupancy probabilities and real-time localization is still relatively under-explored. [13] propose to detect short-term and long-term movables similar to our dynamic and movable labels. However, they compute their short-term and long-term features with ray-tracing in a 2D map, while we propose to train a deep network able to predict them directly in the 3D lidar points based on the appearance of the point clouds. Predicting movable points with deep networks was also suggested by [14]. However, they chose a 2D architecture, FastNet, using lidar depth images, and they only predict an objectness score from a human-annotated training set.

B. Self-supervised Learning for Robotics Application

Self-supervised learning is a form of unsupervised learning where the data provides supervision. In robotics, this term usually refers to methods using an automated acquisition of training data [15]–[20]. These approaches often exploit multiple sensors during the robot's operation. In our case, we only use a 3D lidar and an algorithm that provides automated annotation for the data. Deep-learning-based semantic SLAM algorithms have been proposed, using either camera images [21], [22], lidar depth images [23], or lidar point clouds [24], but they always rely on human-annotated datasets, whereas our method learns on its own. To the best of our knowledge, our approach is the first to use multi-session SLAM and ray-tracing as annotation tools for the training of semantic segmentation networks.

C. Object Tracking and Trajectory Prediction

Following the success of recurrent neural networks (RNNs) and in particular long short-term memory networks (LSTMs) [25], [26], the idea of trajectory prediction has received a lot of attention. It requires the obstacles to be isolated as distinct objects and tracked. [27] uses an LSTM-based network to predict obstacle trajectories and plan around them, and [28] exploits a Hidden Markov Model to predict future states from a history of observations. Similarly, [29] detects individual obstacles and predicts their speeds to avoid "freezing zones". Similar object-centric methods are also used in the context of autonomous driving [30], [31]. However, they all rely on detection and tracking predictions and do not easily incorporate multi-modal predictions, problems we do not face with our point-centric approach. Some methods such as [32] do not predict explicit trajectories to navigate around humans but only learn to place waypoints for the high-level planner with respect to the human motion in an RGB image. As they use a close-up camera feed and do not have any explicit future prediction, they are limited to reacting to the humans in front of the robot instead of anticipating the human motion. Closer to our work, [33] predicts future human motions as 2D heat maps, implicitly handling multi-modality, but still relies on object-level predictions and is also limited to 2D inputs, where our method leverages 3D features that are more descriptive. For a more detailed survey of human motion trajectory prediction, we refer to [34].

We propose our approach as an alternative to trajectory prediction methods since it offers several benefits. It utilizes raw sensor input to produce a risk map directly to the local planner, without any intermediary algorithm or object representation. This reduces potential failure points in the system, handles multimodality intrinsically, and has a fixed computational cost regardless of the number of agents. Furthermore, it simplifies annotation provision and enables self-supervised learning without needing to specify objects. We contend that these advantages validate the importance of our contributions, particularly given that any comparison would be challenging due to different evaluation techniques (object-based or pixel-based).

D. Reinforcement Learning for Navigation

Reinforcement Learning has been used extensively in recent years to replace standard motion planning algorithms [35]–[41]. However, standard local planners have proven to be very reliable methods, especially when it comes to producing dynamically feasible trajectories, which most RL methods fail to do. Even when the feasibility is ensured [42], the whole planning algorithm is embedded into a black box end-to-end neural network, which is difficult to tune, debug, and interpret. We chose to keep a standard local planner, with its guarantees and interpretability, and use a self-supervised deep learning method to predict the future motion of dynamic obstacles.

E. Occupancy Grid Maps Predictions

Occupancy Grid Map (OGM) prediction approaches are the closest to our work and can be separated into two groups either using handcrafted or learned features. Handcrafted approaches usually rely on a model for human motion prediction. [43] predicts a Dynamic Risk Density based on the occupancy density and velocity field of the environment. [44] extends this idea with a Gaussian Process regulated risk map of tracked pedestrians and cars. Other recent works focus on adapting the uncertainty of the predictive model [45]–[47], using real-time Bayesian frameworks in closed-loop on real data. These methods either rely on object detection and tracking or are based on instantaneous velocities, and are not able to predict future locations of obstacles accurately. Learned methods, usually based on video frame prediction architectures [48]–[50], are better at predicting complex futures. [51] introduces a difference-learning-based recurrent architecture to extract and incorporate motion information for OGM prediction. [52] presents an LSTM-based encoder-decoder framework to predict the future environment represented as Dynamic Occupancy Grid Maps. They introduce recurrent skip connections into the network to reduce prediction blurriness. Similarly [53] used LSTMs for future OGM predictions, laying the groundwork for [54], which proposed a double-prong network architecture that provides an effective solution to model both static and dynamic objects' spatiotemporal evolution inside OGMs. Concurrently, [55] developed an attention mechanism that can alleviate errors related to dynamic object disappearance from OGM

predictions. Recently, [56] presented a novel approach to OGM prediction using occupancy flow fields.

However, some major differences remain in our approach. First, these methods all take previous OGMs as input, effectively losing valuable information in the shape patterns that common 3D sensors can capture. To our knowledge, we are the first to fill this gap in the literature, by incorporating 3D features in the forecasting of OGMs with the 3D backbone of our network. Second, despite some early attempts at using feedforward architectures [53], [57], recurrent architectures have dominated the space of future OGM prediction. We successfully predict sequences of OGMs without recurrent layers. Eventually, our network can make a distinction between different semantic classes, leveraging interactions between them, when predicting future occupancy.

We argue that feedforward networks have many advantages compared to recurrent networks. Their specific weights for each timestamp make them more adaptable and easier to train, at the cost of being unable to predict arbitrary sequence lengths. However, in real scenarios, this length is often fixed. Furthermore, recurrent networks explicitly model the sequential nature of the spatiotemporal data from the beginning of the input to the end of the output. They don't easily adapt to cases where the nature of the input sequence is different from the nature of the output sequence. This is a major limitation in our case where SOGMs are generated by a long offline process and cannot be obtained in real-time.

III. Algorithms Used in Our Pipeline

Before presenting our approach as a whole, this section introduces the key algorithms that are used throughout our pipeline.

A. PointMap: ICP-based 3D Localization and Mapping

PointMap [1] is our SLAM algorithm, which has two components: an Iterative Closest Point (ICP) localization solution that aligns lidar frames on a point cloud map, and a mapping function that updates the map with the aligned frame. Each function can be used independently or together in a SLAM mode.

For a detailed description of the ICP algorithm, we refer to a previous in-depth review [5]. Following their work, we use the same elements to characterize our ICP: the data filters, the matcher, the outlier rejection, the distance function, and the convergence tests. Our choices for each element are listed in Table I. Most of these are based on previous works [1], [7], [8], taking into account that we use a Velodyne HDL-32E sensor. Some elements, including matching, are simplified for efficiency. We use the latest odometry of the robot as the initial pose to solve the initialization issue common to most ICP solutions. In the following, we define transformations by their rotation and translation components $(\mathbf{R}, \mathbf{t})$. In comparison to [1], we handle the motion-distortion effect of real data within the iterative process of ICP. At each ICP iteration, after estimating the transformation $(\mathbf{R}_1, \mathbf{t}_1)$ at time t_1 (last timestamp of the current lidar frame), and before matching neighbors, we apply motion distortion. We have the poses $(\mathbf{R}_0, \mathbf{t}_0)$ and $(\mathbf{R}_1, \mathbf{t}_1)$ of the lidar at time t_0 and time t_1, therefore for any point stamped with a time $t \in [t_0, t_1]$, its pose $(\mathbf{R}, \mathbf{t})$ is computed as

$$\begin{aligned} \omega &= (t - t_0) \,/\, (t_1 - t_0) \ , \\ \mathbf{t} &= \omega \mathbf{t}_1 + (1 - \omega)\, \mathbf{t}_0 \ , \\ \mathbf{R} &= \mathrm{Slerp}\,(\mathbf{R}_0, \mathbf{R}_1, \omega) \ , \end{aligned} \tag{1}$$

where Slerp is the spherical linear interpolation [58]. Note that during ICP convergence, we chose t_0 as the beginning of the previous frame instead of the end of the previous frame. Otherwise, the smallest mistake made when estimating the previous pose will cause issues and sometimes divergence.

In addition, we use an optional ground plane heuristic in the optimization. This heuristic, particularly useful for indoor scenarios, assumes that the ground is a planar horizontal surface of height $z = 0$. During ICP optimization, any point considered ground can be matched to this plane instead of its nearest neighbor.

The second component of PointMap, the map update function, adds the information from an aligned frame to the map. We use the same update function described in [1], where the map is defined as a sparse voxel grid. The voxel size is $dl_{\mathrm{map}} = 3$ cm and we keep only one point per voxel, with its normal and its score. When using an initial map for localization, we can create a secondary map that we ignore for localization, but keep for further processing, which is particularly useful for the buffer creation step shown in Figure 4 (a).

TABLE I
ICP Configuration for PointMap with a Velodyne HDL-32E.

Elements	Choices
Filters	Subsample frame with a 12cm grid.
Matcher	Sample 600 points at each iteration according to w_{icp}. Match with single nearest neighbor (or ground plane).
Rejection	If pt2pt distance > 2 m. If pt2pl distance > 12 cm (except for the first iteration).
Distance	Optimize point-to-plane distance.
Convergence	Stop at a maximum of 100 iterations. Stop if relative motion goes below 0.01m and 0.001rad.

B. PointRay: Ray-tracing Occupancy Probability

We use PointRay [1] to compute occupancy probability in a point cloud map. The occupancy probability of each point in a map can be found thanks to the data provided by lidar frames. Indeed, each frame provides two kinds of information: occupied space where the points are located, and free space along the lidar rays. If a location exists in the map but gets passed through by multiple rays, it will have a low occupancy probability. We follow the idea from [12] and use the projection of the map in the lidar spherical coordinates frame to model the lidar rays. The occupancy probabilities are deduced from the distance gap between frame points and map points.

PointRay assigns two values to each point of the map x_i, in a voxel i: n_i the number of times this voxel has been

seen, and o_i the number of times this voxel was occupied. For each lidar frame in the list, PointRay first gets the list of occupied voxels, and increments both n_i and o_i for them. For the rest of the points, PointRay verifies if they are seen by a free space frustum in the spherical coordinates (ρ, θ, ϕ). The frustums are defined as the pixels of a 2D grid in the θ and ϕ spherical dimensions. Each pixel (or frustum) stores the smallest point distances to the lidar origin (ρ spherical coordinate). The resolution of the grid is $d\theta = 0.33°$ and $d\phi = 0.5°$, but in the θ dimension, the lidar resolution is variable, so we use nearest-neighbor interpolation to fill the empty pixels along this dimension.

The verification is done by projecting the map in the same frustum grid. To reduce the effect of motion distortion, we cut the lidar frame in $n_{\text{slices}} = 16$ slices along the azimuth and perform the map projection with the median pose of each slice. Given the small slicing angle, the distortion effect is negligible. Although it is slower to compute the occupancy probability one slice at a time, we alleviate the computational cost by only projecting map points that are in the slice area. For every projected map point x_i, PointRay increments n_i (and not o_i) only if the two following conditions are respected:

$$\begin{aligned} \text{cond}_A: &\quad \rho_i < \rho_0 - \text{margin}(\rho_0)\,, \\ \text{cond}_B: &\quad |n_z| > \cos(\beta_{\min}) \quad \text{OR} \quad \alpha < \alpha_{\max}\,, \end{aligned} \tag{2}$$

where ρ_0 and ρ_i are the frustum radius and point radius respectively, $\text{margin}(\rho_0) = \rho_0 \max(d\theta, d\phi)/2$ is the largest half size of the frustum at this particular range, n_z is the vertical component of the point normal, and α the incidence angle of the lidar ray with this normal. $\alpha_{\max} = \frac{5\pi}{12}$ and $\beta_{\min} = \frac{\pi}{3}$ are heuristic thresholds. cond_A ensures that the ray passes through the point and cond_B verifies the incidence angle. We do not update high incidence angles (so that walls are not removed because of frustum width), except for horizontal surfaces because tables are usually nearly parallel to the lidar rays and would never be updated otherwise.

Because of cond_B, ground points are more likely to have low occupancy probabilities, but we take care of that by extracting the ground as a distinct semantic class, unaffected by ray-tracing.

When all the lidar frames of a session have been processed, the final occupancy probability for each point x_i of the map is computed as $p_i = o_i/n_i$. A point has to be seen at least $n_{\min} = 10$ times to be considered valid, otherwise, $p_i = 0.5$.

C. PointMorpho: Pointcloud Mathematical Morphology

In our annotation pipeline, one of the issues we face when dealing with real data is noise. As shown in the middle-bottom picture of Figure 4, the point labels found automatically can be noisy. In this section, we define Mathematical Morphology operators for point clouds to help reduce the noise in the annotations, through spatial smoothing.

Mathematical Morphology regroups techniques for processing geometrical structures and was originally developed for binary images, with four basic morphological operators: erosion, dilation, opening, and closing [59]. Some works have tried to adapt mathematical morphology to point clouds, by considering points as positive elements, and empty space as negative elements [60], [61].

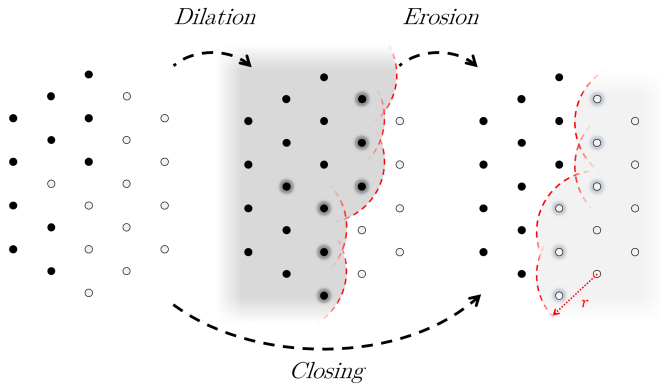


Fig. 3. Illustration of point morphology operations with a radius r. The new black points are the results of a closing of the black points on the white points, while the new white points are the result of an opening of the white points by the black points. Specifically, closing operations merge isolated outliers, whereas opening operations remove isolated inliers.

In this work, we are interested in a simpler problem: applying mathematical morphology on point clouds where some points are considered positives and the other negatives (see Figure 3). We only consider a sub-problem where the structural element is a sphere of radius r, the positive elements are denoted as point cloud A and the negative elements as point cloud B. Therefore, the morphology operations can be described simply:

- **Dilation**: $\mathcal{D}_r(A, B) = A \cup \mathcal{N}_{B,r}(A)$
- **Erosion**: $\mathcal{E}_r(A, B) = A \setminus \mathcal{N}_{B,r}(A)$
- **Closing**: $\mathcal{C}_r(A, B) = \mathcal{E}_r(\mathcal{D}_r(A, B), \mathcal{E}_r(B, A))$
- **Opening**: $\mathcal{O}_r(A, B) = \mathcal{D}_r(\mathcal{E}_r(A, B), \mathcal{D}_r(B, A))$

Where $\mathcal{N}_{B,r}(A) = \{p \in A \mid \exists q \in B : \|p - q\| \leq r\}$ is the subset of A in the neighborhood of B. As we will see in Section IV-B, these tools are particularly efficient for removing noise and undesired artifacts in the annotations.

IV. Offline Processing

A. Initial Mapping

In this work, we assume that our robot will be navigating in the same space for some time, and thus will use a map of this environment for localization. This map will also be used during the annotation process. As explained in our previous works [1], [2], for the annotation, we need the map to contain only *ground* and *permanent* points. It is also convenient for localization, as these points are usually from large planar surfaces, easy to localize against.

The mapping process begins with PointMap in SLAM mode, which generates an initial map of the environment. Next, we apply loop closure using the Open3D library [62], resulting in a point cloud derived from the loop-closed poses. This point cloud is then processed by PointRay to remove dynamic objects. Any space outside the building that may include reflections of indoor space is removed using hard-coded limits. To ensure the map does not contain any movable obstacles, we perform additional runs with the object moved to different

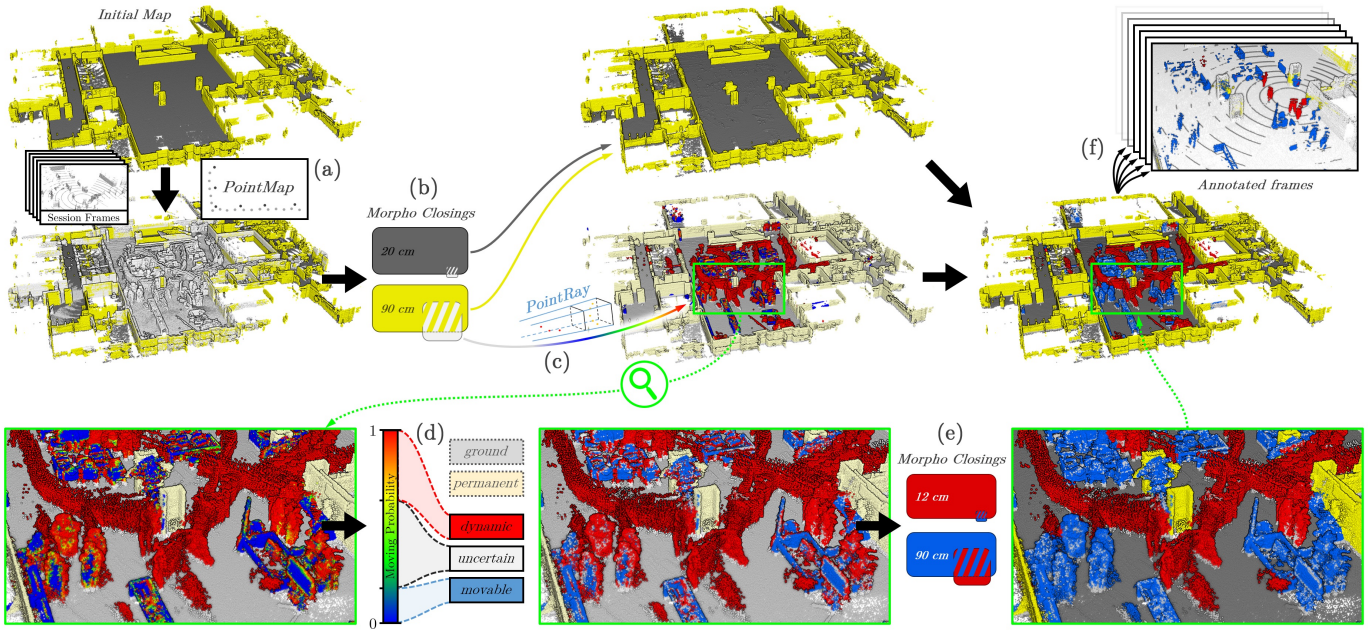


Fig. 4. Illustration of our automated annotation process for real lidar frames. A buffer cloud is created with PointMap (a). The *permanent* and *ground* points are found with morphological closings (b), and the remaining points are annotated as *dynamic* or *movable* by PointRay (c-d). Noise is reduced with morphological closings (e), and the labels are projected back to the frames (f).

places. As it is usually the case in indoor environments, the ground is flat, allowing us to use the flat ground heuristic from PointMap. As a consequence, the ground points can easily be extracted as the horizontal plane at $z = 0$.

A human could intervene at this step to move furniture around between mapping sessions, but it is not required. We assume furniture would be moved at some point in the long-term robot's life, and only do it ourselves for convenience in our short-term experiments. We also note an interesting idea: this strategy could be used for cheap and fast 3D point cloud annotation of any object. For example, only remove the chairs first to annotate them, then remove tables, etc.

B. Automated Lidar Point Annotation and SOGM Generation

We use an automated annotation process to label the 3D lidar points [1] and generate training SOGMs [2], allowing self-supervised learning. Our network can thus learn from new situations encountered throughout the robot's life. We refer to these two papers for detailed explanations of the annotation process on simulated data and only focus here on the specific changes made to handle real data.

The training runs are collected following different controlled or in-the-wild strategies described in Section VII. The collected data consists of sequences of lidar measurements localized in our map with PointMap. We first annotate lidar frames, with the combination of PointMap and PointRay, as described in [1], and shown in Figure 4. As opposed to [1], we are able to handle noisy and imperfect real data. We begin by accumulating the lidar point clouds to get a buffer of points for the session. To annotate the *permanent* points, we perform a point-morphology closing of the map's *permanent* points on the buffer points ($r = 0.9$ m). To annotate the ground points we perform another closing of the map's *ground* points on the buffer points ($r = 0.2$ m). Intuitively, the closing operation can be seen as a more sophisticated distance-based annotation. Distance-based annotation absorbs any point within the defined radius, whereas closing only absorbs points within the radius if they are isolated from other points. For example, the feet of a person will not be annotated as ground with a closing operation. A larger radius means the closing can absorb larger isolated groups of points. These operations leave us with the remaining buffer points that are processed by PointRay, to get their occupancy probabilities, p_i, during this session. Points with $p_i < \tau_s = 0.3$ are labeled as *dynamic*, points with $p_i > \tau_m = 0.6$ are labeled as *movable*, and the remaining points are left *uncertain*. We conduct a noise removal step consisting of a first closing of *dynamics* on *movables* ($r = 0.12$ m) to clean the isolated or tiny groups of *movable* points inside dynamic areas. And then a second larger closing of *movables* on *dynamics* ($r = 0.9$ m) to ensure we only keep large groups of dynamic points. Eventually, the annotations are projected back on the lidar frames, taking into account motion distortion.

Now that we have annotated lidar frames, we can generate SOGMs automatically. Our SOGMs have three channels, one for each obstacle class: *permanent*, *movable*, and *dynamic*. Because we want to be able to augment the training data with rotations, it is better to save our 2D labels as intermediate 2D point cloud structures that can easily be rotated and then transformed into SOGMs during training. An annotated 2D point cloud is computed and saved for each lidar frame by removing non-obstacle points, projecting the remaining points on a horizontal plane, and subsampling them with a grid size of 3 cm. To reduce the noise, we remove isolated dynamic points from these precomputed 2D point clouds, and perform a point morphology opening of the *dynamic* points by the static

points (*permanent* + *movable*), with $r = 0.3$ m, to erase small groups of *dynamic* points too close to static points.

At training time, as shown in Figure 5 we stack the 2D points in a third dimension according to their timestamps, apply data augmentation, and project them to a SOGM structure of spatial resolution $dl_{2D} = 12$ cm and temporal resolution $dt = 0.1$ s. The *permanent* and *movable* occupancies from all time steps of the SOGM are merged because they are not moving. Therefore, in addition to the future locations of dynamic obstacles, our network also learns to complete partially seen static objects.

C. Network Architecture for Lidar Segmentation and SOGM Prediction

Our network architecture (Figure 6) is composed of two parts, a 3D back-end, and a 2D front-end. The 3D back-end is a KPConv network [63] predicting a semantic label for each input point. Predicting 3D labels helps the network training by providing an additional supervisory signal and ensures that rich features are passed to the 2D front-end. We keep the KP-FCNN architecture and parameters of the original paper: a U-Net with five levels, each composed of two ResNet layers, refer to [63] for details. The network input is a point cloud made from $n_f = 3$ lidar frames (covering a time frame of 0.3 s) aligned in the map coordinates and merged. We only keep the points inside a $R_{in} = 8$ m radius, as we are interested in the local vicinity of the robot. Each point is assigned a one-hot n_f-dimensional feature vector, encoding the lidar frame to which it belongs. To help with computational speed for inference on a real robot, the input point density is controlled using a relatively large grid subsampling size ($dl_{3D} = 12$ cm). It is equal to the PointMap subsampling size, allowing us to reuse the subsampled cloud aligned and rectified by PointMap as is.

The 3D point features of dimension $D_{3D} = 64$ are passed to the 2D front-end with a grid projection using the same spatial resolution dl_{2D} as the SOGM. The size of the grid is determined as the inscribed square in the R_{in}-radius circle: $h_{grid} = w_{grid} = 94$. Features from points located in the same cell are averaged to get the new cell features. The features of

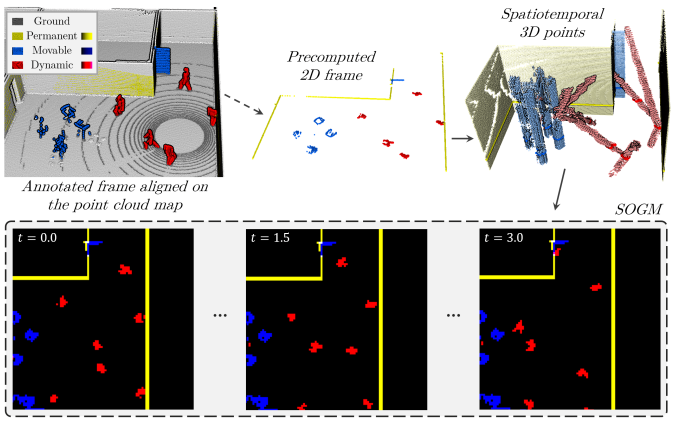


Fig. 5. During pre-processing, every frame is semantically filtered and projected in 2D. During training, the 2D frames are stacked in 3D according to their timestamps and projected to a 3D grid to create the SOGMs.

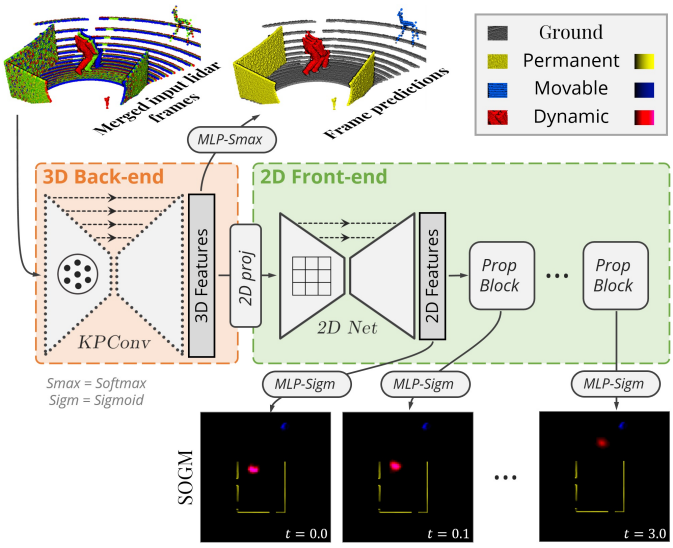


Fig. 6. Illustration of our 3D-2D feedforward architecture. The 3D back-end is a 5-layer KPConv architecture, producing features at the point level. The 2D front-end is composed of a 3-layer 2D U-Net architecture, followed by consecutive convolutional propagation blocks.

the empty cells are set to zero. The obtained 2D feature map is then processed by an image U-Net architecture with three levels, each composed of two ResNet layers, to diffuse the information contained in sparse locations to the whole grid. This dense feature map is used to predict the initial time step of the SOGM. Then, it is processed by successive propagation blocks, each composed of two ResNet layers. The output of each propagation block is used to predict the corresponding time step of the SOGM. We define the final prediction time $T = 4.0$ s, meaning that our SOGMs have $n_T = T/dt + 1 = 41$ time steps in total. More details and hyperparameters can be found in our implementation. Note that the *permanent* and *movable* predictions are redundant but we keep them to help the network to keep the knowledge of their location, and to learn better interactions between the classes further into the future.

D. Network Training

The training loss of our network is a combination of two loss functions. The standard semantic segmentation loss of a KPConv network L^{3D}, and a loss function applied to each SOGM prediction layer L_k^{2D}. We define it as

$$L_{tot} = \lambda_1 L^{3D} + \lambda_2 \sum_{k<n_T} \frac{L_k^{2D}}{n_T}, \tag{3}$$

where $\lambda_1 = 1.0$, $\lambda_2 = 10.0$. L^{3D} is the standard cross entropy loss used in the original KPConv network, and L_k^{2D} is a Binary Cross-Entropy loss applied to layer k of our SOGM predictions:

$$L_k^{2D} = \sum_{i \in M_k} BCE(x_{k,i}, y_{k,i}), \tag{4}$$

where $x_{k,i}$ is the network logit at the pixel i of the time-step layer k in the SOGM, $y_{k,i}$ is its corresponding label and BCE

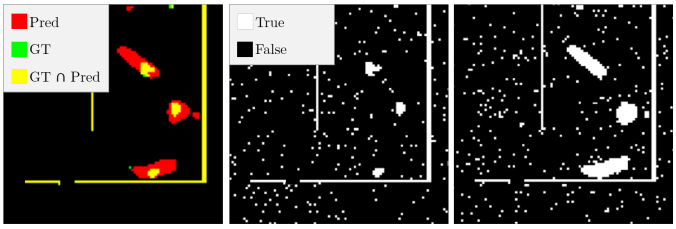


Fig. 7. Loss masks reducing the influence of empty pixels during training.

stands for Binary Cross-Entropy. Note that for clarity, we use a simple index i for 2D pixels. The SOGM loss is thus a masked Binary Cross-Entropy, where the mask M_k is here to help ignore the over-represented empty pixels and focus on the positive examples. We first tried a mask covering the positive label values in addition to some random pixels (GT-Mask), but then improved it to cover the union of positive labels and positive prediction pixels (Active-Mask) to help reduce the false positives (see Figure 7).

Our network is trained with PyTorch, with an SGD optimizer. The initial learning rate is 0.01, and decayed by 0.96 at every epoch (total decay adds up to 0.1 every 60 epochs). In this setup, our input point clouds contain on average 20k points. We use a variable batch size targeting $B = 6$, for an average of 85k points per batch. During training, we only use rotation augmentation around the vertical axis. To cope with unbalanced classes in real data, we implement a sampling strategy targeting input examples containing *dynamic* points. Instead of sampling frames randomly during training, we chose the frames that contain *dynamic* points more often than the rest of the frames (with a ratio of 10:1). We also have the possibility to train a network with a combination of real and simulation examples (with a customizable ratio), a strategy that we evaluate in section VII-C. The rest of the training parameters are kept identical as in the original KPConv paper [63], and more details can be found in our open-source implementation.

V. Online Navigation

A. Network Inference and Post-processing

During navigation, our network receives lidar point clouds sent by PointMap. They are already subsampled to $dl_{3D} = 12$ cm, aligned on the map, and motion rectified. When three consecutive point clouds have been received, the CPU processing kicks in and performs all the necessary operations including merging frames, subsampling layer points, computing neighbors, etc. As soon as the CPU pre-processing is over, the GPU computes a forward pass of the network and gets the predicted SOGM, which contains the future occupancy locations up to 4 seconds after the input lidar timestamp.

We want to use these predictions in the Timed-Elastic-Band (TEB) planner [64], [65], but the original implementation only handles point obstacles. We chose to modify the TEB implementation to be able to handle grid representations (see Figure 9). TEB originally minimizes a linearly decreasing cost function: $\mathcal{C}_{\text{obst}} = \max(0, 1 - d/d_0)$, where d is the distance from the optimized pose to the closest obstacle and d_0 a predefined influence distance. The simple way to handle grid structures is to let TEB minimize the risk value at the current pose (interpolated from the closest grid values). Then we just need the grid values to represent a smooth cost function. In our case, we defined a linearly decreasing risk value similar to the original obstacle cost but with some modifications, as shown in Figure 8. First, we use a threshold $\tau_{\text{risk}} = 0.4$ to extract risky area from the SOGM:

$$\mathrm{R}^1_{k,i} = \mathrm{SOGM}_{k,i} > \tau_{\text{risk}} \,. \tag{5}$$

Then we apply a 2D convolution to sum the polynomial decreasing cost from all pixels:

$$\mathrm{R}^2_{k,i} = \sum_j \left(\mathcal{C}\left(i,j\right)^p \times \mathrm{R}^1_{k,j} \right) , \tag{6}$$

with $\mathcal{C}\left(i,j\right) = \max\left(0, 1 - d\left(i,j\right) \times dl_{2\mathrm{D}}/d_0\right)$, where $d\left(i,j\right)$ is the distance from pixel i to pixel j in the grid space, p is explained later, and d_0 has the same meaning as in the original TEB, the influence distance of obstacles. This convolution diffuses the risk in space, but we also decided to diffuse the risk in time:

$$\mathrm{R}^3_{k,i} = \sum_l \left(\mathcal{C}_{\mathrm{t}}\left(k,l\right)^p \times \mathrm{R}^2_{l,j} \right) , \tag{7}$$

with $\mathcal{C}_{\mathrm{t}}\left(k,l\right) = \max\left(0, 1 - \left\|t_k - t_l\right\| / \Delta_0\right)$, where $t_{k/l}$ is the time at layer k/l, p is explained later, and $\Delta_0 = 1$ s is the influence range in time.

Summing risk this way means larger risk areas will have higher risk values. To even out the risk value for any risk area, we apply the same convolution but on normalized risk:

$$\mathrm{R}^4_{k,i} = \sum_l \sum_j \mathcal{C}_{\mathrm{t}}\left(k,l\right)^p \left(\mathcal{C}\left(i,j\right)^p \left(\mathrm{R}^1_{l,j} / \mathrm{R}^3_{l,j} \right) \right) . \tag{8}$$

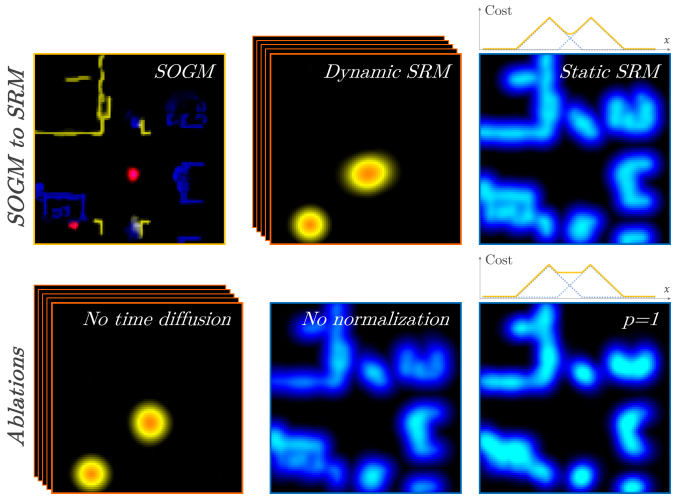


Fig. 8. Conversion of SOGMs into decoupled static and dynamic SRMs. We show the impact of time diffusion, normalization, and parameter $p = 1$ ($p = 3$ for static SRM). We show the effect of parameter p in a small graph on top of the static SRM.

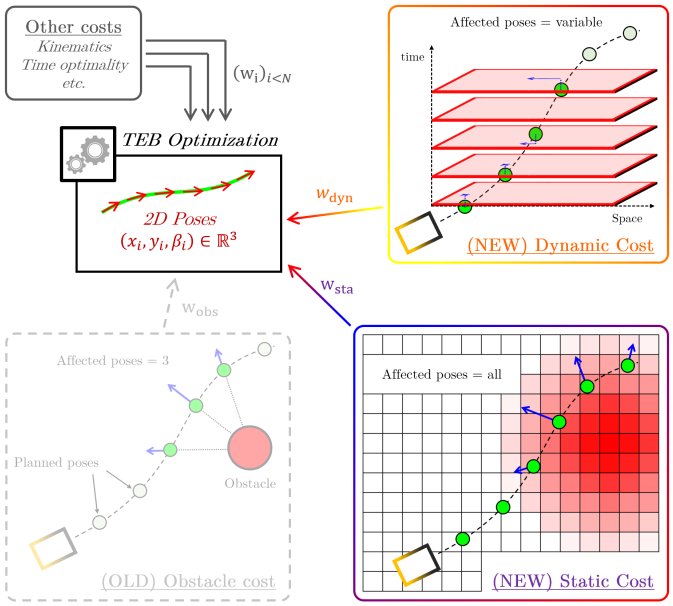


Fig. 9. Illustration of TEB optimization costs. To replace the obstacle cost, we define a static cost and a dynamic cost. Our new costs use risk maps that directly define the cost value and its gradient with bilinear interpolation.

We put the linearly decreasing cost value to the power p, but we can retrieve a linearly decreasing diffusion by taking the power $1/p$ of this final cost:

$$\mathrm{SRM}_{k,i} = \left(\mathrm{R}^4_{k,i}\right)^{(1/p)} . \tag{9}$$

This risk value behaves like a p-norm (see the small graphs in Figure 8), the higher p is, the closer it is to the maximum value of the linear influence of each surrounding pixel. We use $p = 3$ in the following.

In addition to this new definition of SRM, we decouple the *static* risk and *dynamic* risk. The *dynamic* risk is computed from the *dynamic* channel of the SOGM, while the *static* risk comes from the *permanent* and *movable* SOGM channels. Because the *static* risk is the same at any time, it can be stored in a single 2D risk map. For convenience, we store it in the first layer of the SRM, while the rest of the SRM layers only contain the *dynamic* risk. This decoupling allows us to have separate distance and weight parameters for both risks, and keep better control of the navigation behavior. The *dynamic* risk is diffused in space and time defined as above with $d_0^{dyn} = 1.2$ m, and $\Delta_0 = 1$ s and the static risk is diffused only in space with $d_0^{sta} = 0.9$ m.

The planner only accounts for dynamic obstacles within the prediction horizon, and poses beyond this horizon are assumed to have zero dynamic risk. TEB also allows the optimization of multiple trajectories for different homotopy classes. We keep this feature by creating estimated point obstacles at local maxima in the SOGM, ignored by the trajectory optimizer, but used for homotopy class computation.

In simulation, we have access to high computing power, with an Nvidia RTX 3090 GPU and an Intel i9-10980XE CPU @ 3.00GHz. In addition, we can slow the simulation time factor to reduce delay to virtually zero if we need. On the real robot, we are limited to a laptop configuration, with a much smaller GPU (Nvidia T1000) and a much slower CPU (i7-11800H @ 2.30GHz). However, with our current implementation and parameters (especially $dl_{3D} = 12$ cm), we can run everything in real-time. The input frames arrive from PointMap with a delay of approximately 45 ms. The CPU pre-processing takes on average 47 ms and the GPU network computations 74 ms. Finally, 53 ms are used for the SRM conversion. The total delay to get a new prediction varies between 230 ms (10^{th} percentile) and 320 ms (90^{th} percentile), with an average of 259 ms, which is sufficient given the 4 s horizon or our predictions. It means only the first few layers of the predictions are obsolete. To remain closer to the real configuration in our simulation experiments, we simulate this delay by waiting before publishing the network

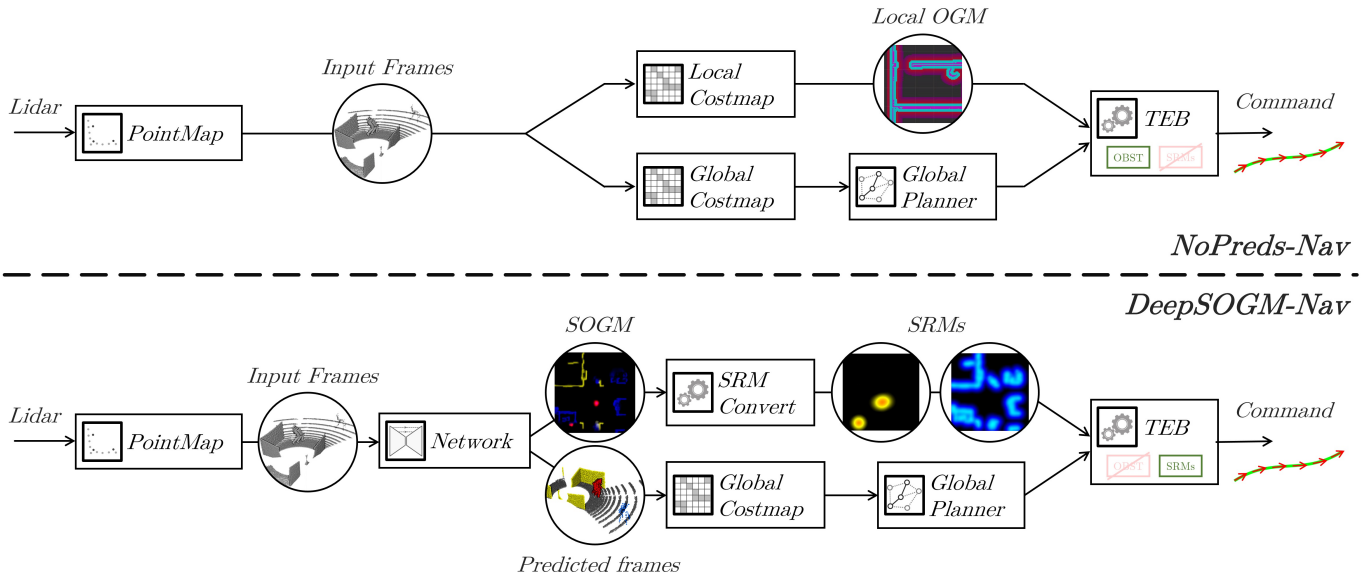


Fig. 10. Illustration of the two navigation systems implemented on the simulated and the real robot. *NoPreds-Nav* is a standard ROS navigation system using the regular TEB planner without predictions. *DeepSOGM-Nav* integrates the network predictions.

results to the rest of the pipeline.

B. Standard Navigation System and Prediction Integration

Our network predictions can easily be plugged into a standard navigation system. We use original ROS plugins for most of the navigation except for localization, which is performed with PointMap (adapted as a ROS plugin), and the local planner, which is our modified version of TEB. As shown in Figure 10, in the standard navigation pipeline, lidar frames are processed by PointMap to get the current robot pose. Then local and global costmaps are computed with the *move_base* ROS node. The global planner finds the optimal path to the goal and TEB follows this path while avoiding obstacles in the local costmap.

When using deep predictions, the subsampled and aligned frame from PointMap is sent to the network to be labeled, and to produce the SOGM, immediately converted into SRMs. The global costmap is computed with the labeled points and ignores dynamic obstacles. TEB tries to follow the global plan while avoiding high-risk areas in the SRMs. Note that we use the raw lidar frame for localization as opposed to [1], where we used predicted frames, because our network needs three aligned input frames to be able to extract the current speed of dynamic points.

VI. Simulation Experiments

In [2], we evaluated our Deep SOGM predictions on simulated data. We showed that our predictions could generalize to different types of actors, compared them to the predictions of other methods, and provided ablation studies. In this section, we complete these experiments with an evaluation of our navigation system. In particular, we compare the efficiency and safety of the TEB planner when using different types of SOGM predictions, or no prediction at all. We choose to conduct these experiments in simulation to allow large-scale testing, true metrics, and a repeatable controlled scenario for comparable results.

A. Simulation Setup

We use the same Gazebo simulated environment as in [1], [2] for our experiments. In this case, we designed a controlled experiment that could be repeated many times to get reliable results and fair comparisons between all methods. Tables and chairs are generated randomly in the space and a fixed number of Flow Followers [2] are moving between a set of goals that we chose around the robot path, to force a lot of encounters. The robot is always asked to follow the same path, consisting of going across the main atrium a few times. See Figure 11 for a visualization of this setup. Note that during training, we use the data collected in [2], which differs from the setup presented in Figure 11. The number of agents is randomized, more goals are dispatched throughout the environment, and the robot navigates various routes. Despite this, we postulate that when navigating through the large central space, the general movement patterns are likely consistent, with agents typically moving toward doors or corridors. We believe that they align with real-world behavior. Rather than aimlessly wandering, people typically walk toward specific locations, such as doors or corridors.

In our experiments, we use metrics that are simple and intuitive. To measure the efficiency of the planner, we use the **Time to Reach the Final Goal** (T_f) $\downarrow$ in seconds. To measure the safety of the planner, we measure the distance from the center of the robot to the center of the closest dynamic actor (whose position is given by the simulator), and derive two metrics from it: the **Collision Ratio** (%C) $\downarrow$, measuring the percentage of the total time during which the robot is in collision with an actor (distance smaller than $d_c = 0.4m$); and the **Risk Ratio** (%R) $\downarrow$, measuring the proportion of the session during which the robot is in a risky area (distance smaller than $d_r = 1.0m$), which indicates when the robot is dangerously close to an actor. In addition to these main metrics, we provide four additional metrics providing more insight into the results:

- average absolute speed (AAS) $\uparrow$
- percentage of time stopped (%S) $\downarrow$
- average linear speed (ALS) $\uparrow$
- percentage of time going backwards (%B) $\downarrow$

The AAS measures the robot's absolute speed in the horizontal (x, y) plane, which is always positive, regardless of the direction, and is averaged across the session. On the contrary, ALS measures the speed with respect to the robot heading direction, which can be negative. The %S and %B metrics are measured with absolute speed and linear speed respectively. The %S is the proportion of time when the absolute speed is inferior to 0.1 m/s, while the %B is the proportion of time when the linear speed is inferior to -0.1 m/s.

B. Planner Comparison Using Different Types of Predictions

In our first experiment, we verify the benefit of our Deep-SOGM predictions for navigation. We thus keep the TEB planner and its parameters fixed and compare its performances when using:

- *NoPreds*: original version of the TEB ROS package, using local costmap pixels as obstacles.

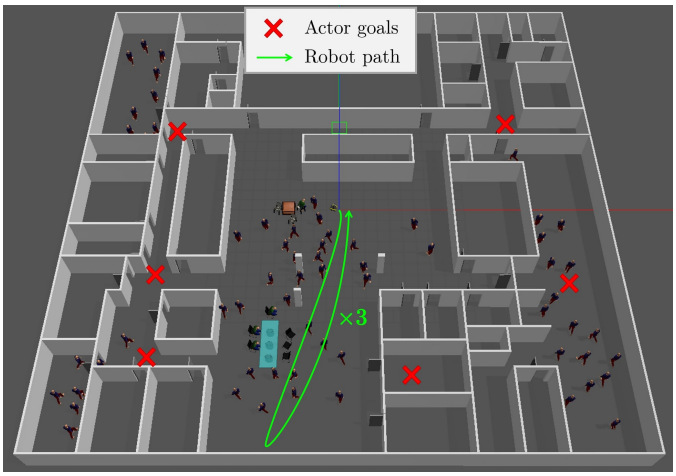


Fig. 11. Simulation setup for our experiments. Actors are walking towards a goal randomly selected among the possible red cross locations. The robot navigates back and forth in the main atrium.

- *IgnoreDyn*: idea from [1], ignoring the dynamic obstacles, only using the static SOGM.
- *LinSOGM*: use actor current speeds (provided by the simulator), and extrapolate their positions linearly.
- *DeepSOGM*: our deep SOGM predictions. We use the network trained on Flow Followers from [2], and use it for inference here.
- *GtSOGM*: precompute actors' movements in advance, to get the ground truth future SOGM when navigating.

For a fair comparison, we enforce the same 250ms delay for the methods using SOGMs, to reflect what the real robot would be able to achieve. Note that for the *GtSOGM* method, the actors will not react to the robot, as their movements are computed in advance. We find that this slight difference does not affect the results, because the FlowFollowers only try to avoid the robot if it is nearly colliding with them. For each method, we repeat the experiments 20 times to get a reliable average, standard deviation (std), and box plot. The results are compiled in Figure 12 and Figure 13 completes these results with additional metrics.

First, we look at TEB without predictions, *NoPreds*, and see that it nearly never collides (%C < 0.5). However, it often gets into risky situations and has to stop, therefore, it ends up being inefficient, with a longer time to finish. The first idea, *IgnoreDyn*, proposed in [1], is to ignore dynamic obstacles, hoping for the fact that they will avoid the robot on their own. Even if the Flow Followers are implemented to avoid the robot, they end up colliding often with it (2%C on average). We argue that people would be better at avoiding the robot, but it would be risky to rely solely on people's reactions, and it would probably lead to many collisions. We notice that this planner is the fastest ($T_f < 140$ s on average), and rarely stops, because it goes straight to the goal, without avoiding the dynamic obstacles. Then we evaluate a planner using the linear extrapolation of the actor's current speeds, *LinSOGM*. This method gives the robot a sense of the future movement of the actors, which leads to a reduced time in risky and collision areas. But it is not ideal yet, because the robot anticipates on an approximate linear prediction, and has to readjust many times. It particularly affects efficiency (time to finish) and is even worse than the regular TEB. Our version of TEB, *DeepSOGM*, performs well compared to the other methods, with close to zero collision (%C < 0.5), the shortest time in risk areas (5%R on average), and a relatively fast finishing time. Eventually, we compare the performances to the SOGMs using the actual ground truth future provided by the simulator, and this is probably the most impressive result. Our performances are extremely close to the performances of a robot that could actually predict the future. Compared to *GtSOGM*, *DeepSOGM* actually yielded the lowest number of collisions, plausibly due to the prediction of larger risk regions that account for the model's inherent prediction uncertainty.

We believe this result is more useful than a comparison to other OGM prediction methods, which would be complex to implement in our pipeline without modifying several other components. It lets us believe that our prediction method, in itself, is strong enough to provide SOGMs of sufficient quality for navigation and that further improvements would probably be achieved by upgrading other components (SRM conversion, planner, etc.), or finding ways to prolong the prediction time horizon.

In our **supplementary video**, we show the robot navigating with different types of prediction, first in rviz view, where predictions can be visualized, and then in a schematic bird-eye view where the difference in trajectories between *NoPreds* and *DeepSOGM* can be seen.

C. Ablation Studies of the Planner Using Deep-SOGM

In this second experiment, we use the same protocol to compare three versions of our Planner using our Deep-SOGM predictions. We show how some of the key choices made for the SOGM-to-SRM conversion affect the performances in terms of safety and efficiency in Figure 14. First, we measure the performances when computing SRM with $p = 1$. The navigation is riskier and less efficient because the risk function is not well defined in the space between obstacles. Then we remove the diffusion of the risk in the time dimension, one of the additions made in this work. In that case, the robot can plan trajectories closer to the back or the front of moving actors, which naturally increases the time spent in risky areas. It allows the robot to go faster in some cases, but also means it

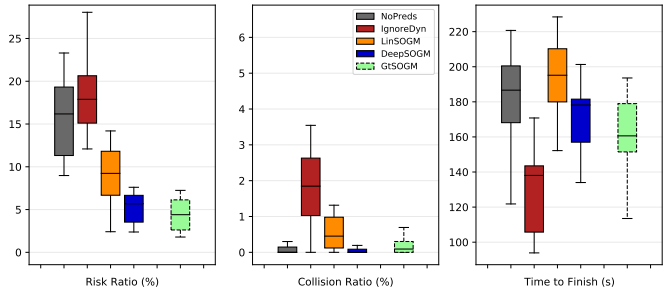


Fig. 12. Evaluation of the Safety and Efficiency of the TEB planner with different types of predictions. For each type of prediction, results are collected over 20 different sessions.

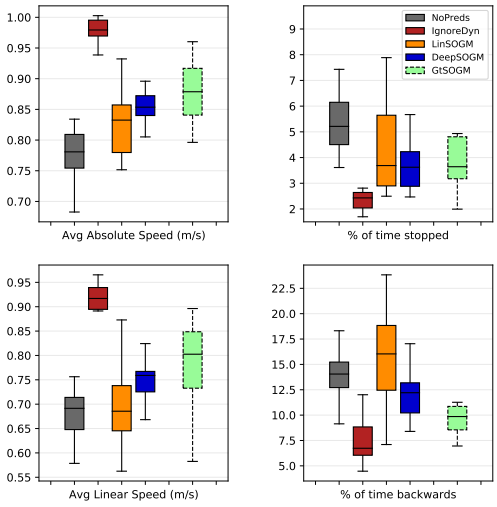


Fig. 13. Additional metrics for the evaluation of the Safety and Efficiency of the TEB planner with different types of predictions.

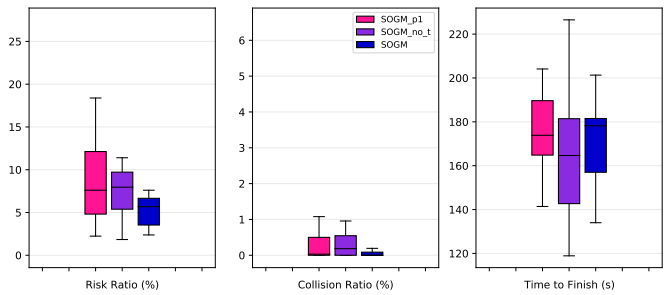


Fig. 14. Evaluation of the Safety and Efficiency of our DeepSOGM TEB planner without some key components for SOGM-SRM conversion. For each ablation study, results are collected over 20 different sessions.

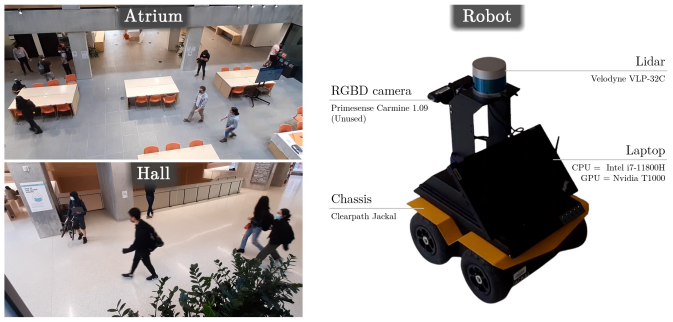


Fig. 15. Our real robot setup and experiment spaces. In this work, we only use the lidar sensor and perform most of the computations on a laptop fixed to the robot.

will end up more often in collision situations, where it has to stop and reverse. Therefore, the distribution of time to finish is more spread out for this method. In both cases, our final version of the planner has better performance.

VII. Real-world Experiments

A main goal of this paper is to validate that our algorithms generalize to real-world indoor navigation. In this section, we analyze our network predictions and our navigation system using real data. First, we can study the network predictions on their own, in a similar fashion as [2], by comparing the network predictions to the data annotated by our automated pipeline. We evaluate how the predictions can improve over time, in a lifelong learning manner, and how adding simulated data to the training set impacts the results. Then we analyze our navigation system. In the real world, it is hard to reproduce multiple navigation experiments as we could do in simulation, but we show, with anecdotal examples, that the conclusions from simulated experiments generalize to real data. Eventually, we compile the data collected during our experiments to provide a new 3D lidar dataset with indoor pedestrians for the robotics community.

A. Real-world Setup

For the real-world experiments, we use a Clearpath Jackal robot, shown in Figure 15. It is a small field robotics research platform, with an onboard computer, GPS, and IMU fully integrated with ROS. In this work, we use a single 3D sensor: a Velodyne VLP-32C lidar sensor. An RGBD camera is mounted on the robot, but we do not use it for our experiments. To this platform, we add a laptop computer with an Intel CPU (i7-11800H @ 2.30GHz) and an Nvidia GPU (Nvidia T1000 4GB). Most of the computations (localization, planning, inference) are performed on the laptop. Only basic tasks (Velodyne processing, low-level control) are performed on the onboard Jackal computer.

In the real world, it is hard to reproduce multiple experiments as we could do in simulation. On the one hand, if you choose to navigate "in-the-wild" in a space where people are not told to behave in a particular way around the robot, the navigation conditions from one session to another will be totally different depending on how people react to the robot or try to confuse it. On the other hand, if you choose to have a controlled experiment, where people are told to act in a particular way, you can assume that the navigation conditions will be roughly similar, but you limit yourself to over-simplified situations and behaviors, and cannot be sure that the results will generalize well to any circumstances.

Having verified our navigation system performances in simulation, another thorough study on our real-world platform is not crucial, and we decided to conduct both controlled and in-the-wild experiments to validate that our robots behave as intended and to confirm the results from Section VI. This is why we perform experiments in two different spaces of the same building. The **Atrium** has several tables and chairs that are often moved and configured differently depending on the occasion. In this space, students usually come to work and thus dynamic obstacles are not very common, except during specific events. This space was used to conduct more controlled experiments. The main **Hall** of the building has a big entrance, stairs, and elevator that lead to classrooms, and large open spaces without tables, where students come and go depending on where they are heading. During peak hours, this space can be crowded with dynamic obstacles, which was ideal for our in-the-wild experiments. Pictures of both spaces

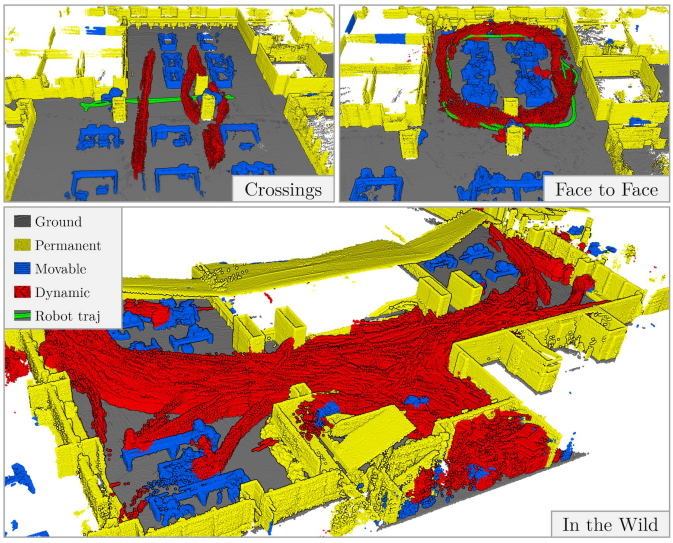


Fig. 16. Different parts of our 3D lidar point cloud dataset, annotated by our automated pipeline. We see controlled scenarios with the robot trajectory in green, and the *dynamic* points in red. A crowded in-the-wild session is also given as an example.

TABLE II
DESCRIPTION OF ALL THE SESSIONS IN UTIN3D-ATRIUM DATASET. FOR EACH SESSION WE SPECIFY THE TIME, THE NUMBER OF FRAMES (N_f), THE NUMBER OF POINTS (IN MILLIONS), THE PERCENTAGE OF FRAMES CONTAINING DYNAMIC POINTS (DynF), AND THE PERCENTAGE OF DYNAMIC POINTS (DynP). WE HIGHLIGHT CROWDED SESSIONS IN BOLD WHEN DynF $> 50\%$ OR DynP $> 5\%$.

UTIn3D-Atrium

	Date	Tr/Va		Time	N_f	Mpts	DynF	DynP
UTIn3D-A1	2021-12-10_12-53-37	✓		0:04:21	2610	271.0	43.3%	1.9%
	2021-12-10_13-06-09		✓	0:03:37	2166	224.8	6.8%	.5%
	2021-12-10_13-17-29		✓	0:04:22	2617	277.1	.0%	.1%
	2021-12-10_13-26-07	✓		0:02:58	1782	189.3	21.3%	1.2%
	2021-12-10_13-32-10	✓		0:04:12	2519	267.9	39.2%	3.8%
	2021-12-13_18-16-27	✓		0:02:54	1743	183.0	29.4%	1.2%
	2021-12-13_18-22-11	✓		0:04:17	2568	275.8	19.9%	1.0%
	2021-12-15_19-09-57	✓		0:02:31	1511	159.9	17.6%	1.1%
	2021-12-15_19-13-03	✓		0:03:43	2229	238.3	22.5%	1.0%
UTIn3D-A2	2022-01-18_10-38-28	✓		0:03:35	2151	228.2	49.9%	3.5%
	2022-01-18_10-42-54	✓		0:03:38	2180	231.1	30.7%	2.0%
	2022-01-18_10-47-07	✓		0:01:06	664	68.5	22.1%	1.1%
	2022-01-18_10-48-42	✓		0:03:51	2306	244.6	21.2%	1.2%
	2022-01-18_10-53-28	✓		0:03:43	2232	236.8	50.6%	4.0%
	2022-01-18_10-58-05	✓		0:03:42	2216	235.1	37.9%	3.6%
	2022-01-18_11-02-28	✓		0:03:36	2161	229.4	25.4%	1.4%
	2022-01-18_11-11-03	✓		0:03:53	2333	247.4	47.8%	3.6%
	2022-01-18_11-15-40	✓		0:03:32	2126	225.5	31.0%	1.8%
	2022-01-18_11-20-21	✓		0:04:02	2422	257.5	35.3%	1.8%
UTIn3D-A3	2022-02-25_18-19-12	✓		0:03:11	1909	201.2	27.3%	1.4%
	2022-02-25_18-24-30	✓		0:03:07	1869	197.9	28.0%	1.2%
	2022-02-25_18-29-18	✓		0:03:24	2041	215.9	23.1%	1.0%
	2022-03-01_22-01-13	✓		0:03:08	1879	199.5	57.2%	2.6%
	2022-03-01_22-06-28	✓		0:03:42	2222	236.2	27.7%	1.3%
	2022-03-01_22-25-19	✓		0:03:44	2243	238.8	43.2%	2.8%
UTIn3D-A4	2022-05-20_12-47-48		✓	0:03:02	1820	190.3	**69.0%**	**7.7%**
	2022-05-20_12-54-23	✓		0:03:01	1815	191.0	**67.0%**	**5.2%**
	2022-05-20_12-58-26	✓		0:03:37	2171	226.3	**59.8%**	**8.3%**
	2022-05-31_14-45-53	✓		0:02:16	1363	143.5	44.5%	3.5%
	2022-05-31_16-25-23	✓		0:02:53	1736	184.3	47.1%	3.0%
	2022-05-31_16-29-56	✓		0:02:52	1717	182.4	60.6%	4.5%
	2022-05-31_16-35-32	✓		0:01:49	1094	115.5	**67.8%**	**5.8%**
	2022-05-31_16-38-34	✓		0:02:00	1196	126.6	31.5%	2.3%
	2022-05-31_18-33-02	✓		0:02:01	1215	129.0	11.0%	.6%
	2022-05-31_19-34-18		✓	0:01:48	1082	114.9	33.3%	1.2%
	2022-05-31_19-37-08	✓		0:02:27	1467	155.4	77.8%	4.9%
	2022-05-31_19-40-52		✓	0:02:50	1702	180.6	54.1%	2.8%
	2022-05-31_19-44-52	✓		0:01:58	1177	124.8	51.3%	3.0%
	2022-05-31_19-47-52	✓		0:01:59	1194	127.0	50.6%	2.6%
	2022-05-31_19-51-14	✓		0:01:58	1184	125.5	24.9%	1.3%
	Total	35	5	2:04:19	74632	7897.9	37.7%	2.6%

TABLE III
DESCRIPTION OF ALL THE SESSIONS IN UTIN3D-HALL DATASET. FOR EACH SESSION WE SPECIFY THE TIME, THE NUMBER OF FRAMES (N_f), THE NUMBER OF POINTS (IN MILLIONS), THE PERCENTAGE OF FRAMES CONTAINING DYNAMIC POINTS (DynF), AND THE PERCENTAGE OF DYNAMIC POINTS (DynP). WE HIGHLIGHT CROWDED SESSIONS IN BOLD WHEN DynF $> 50\%$ OR DynP $> 5\%$.

UTIn3D-Hall

	Date	Tr/Va		Time	N_f	Mpts	DynF	DynP
UTIn3D-H	2022-03-08_21-02-28	✓		0:04:04	2445	257.2	**93%**	**13.3%**
	2022-03-08_21-08-04	✓		0:02:24	1446	152.5	**59%**	**5.1%**
	2022-03-08_22-19-08	✓		0:03:27	2070	219.9	54%	4.1%
	2022-03-08_22-24-22	✓		0:02:28	1481	156.3	44%	3.8%
	2022-03-09_15-55-10	✓		0:02:40	1600	168.9	50%	3.8%
	2022-03-09_15-58-56		✓	0:03:24	2042	217.1	30%	1.7%
	2022-03-09_16-03-21		✓	0:02:51	1715	180.6	**72%**	**12.6%**
	2022-03-09_16-07-11	✓		0:02:29	1492	157.1	**92%**	**7.3%**
	2022-03-16_16-05-29	✓		0:02:57	1773	186.4	**92%**	**20.0%**
	2022-03-16_20-05-22	✓		0:02:57	1772	185.9	**93%**	**14.8%**
	2022-03-16_20-13-08	✓		0:03:17	1968	206.5	**86%**	**8.8%**
	2022-03-16_21-21-35	✓		0:03:30	2097	222.9	46%	2.7%
	2022-03-16_21-28-09	✓		0:02:02	1226	129.4	33%	2.8%
	2022-03-22_14-04-53	✓		0:01:15	745	78.4	**81%**	**10.2%**
	2022-03-22_14-07-26	✓		0:02:37	1572	166.5	**93%**	**8.9%**
	2022-03-22_14-12-20		✓	0:02:42	1625	172.1	**54%**	**5.6%**
	2022-03-22_15-05-20	✓		0:02:54	1737	183.7	**92%**	**12.2%**
	2022-03-22_15-09-02	✓		0:02:34	1539	163.1	**57%**	**6.0%**
	2022-03-22_15-12-23	✓		0:02:29	1492	158.2	57%	4.9%
	2022-03-22_16-04-06	✓		0:02:59	1789	188.4	**100%**	**18.7%**
	2022-03-22_16-08-09		✓	0:02:42	1622	171.3	**95%**	**11.3%**
	2022-03-28_14-53-33		✓	0:02:30	1502	159.4	25%	1.5%
	2022-03-28_14-57-17	✓		0:02:43	1635	173.2	53%	4.1%
	2022-03-28_15-00-42	✓		0:02:40	1603	169.8	**66%**	**5.4%**
	2022-03-28_15-04-24	✓		0:02:32	1520	161.0	42%	3.9%
	2022-03-28_16-56-52		✓	0:00:55	554	58.5	58%	3.7%
	2022-03-28_17-03-29	✓		0:01:36	959	100.5	**100%**	**35.7%**
	2022-03-28_17-07-19	✓		0:01:42	1025	108.5	**83%**	**9.0%**
	2022-03-28_17-10-13	✓		0:02:36	1564	165.5	**53%**	**5.5%**
	2022-03-28_21-57-36	✓		0:02:35	1550	164.1	**71%**	**5.5%**
	2022-03-28_22-02-15	✓		0:02:50	1700	179.8	**78%**	**7.8%**
	2022-04-01_14-00-06		✓	0:02:19	1387	146.6	59%	5.0%
	2022-04-01_14-03-50	✓		0:02:48	1681	177.3	**94%**	**13.5%**
	2022-04-01_14-53-42	✓		0:01:29	891	94.5	75%	3.5%
	2022-04-01_14-57-35		✓	0:02:17	1376	145.8	44%	2.3%
	2022-04-01_15-01-18	✓		0:04:33	2734	288.3	**99%**	**29.1%**
	2022-04-01_15-06-55		✓	0:03:21	2008	211.3	**92%**	**15.1%**
	2022-04-01_15-11-29		✓	0:02:36	1566	165.5	**54%**	**5.1%**
	Total	28	10	1:40:47	60503	6391.8	**69%**	**8.8%**

are shown in Figure 15.

B. Real Data Collection and Automated Annotation

Our first real experiments were conducted in the Atrium because it is ideal for controlled scenarios. With low traffic, and people mostly sitting for long periods, it is less likely that unexpected circumstances arise there. Following our automated annotation pipeline, we first conducted a mapping session, obtaining the initial map shown in Figure 4. Mapping and refinement were done by driving the robot manually for convenience, but could also have been done with the *NoPreds-Nav*, with PointMap in SLAM mode. We started by collecting a first batch of 9 sessions in the space without interfering. Therefore, only a handful of dynamic obstacles were encountered. A second batch of 10 sessions was collected in a controlled manner, where a person was asked to cross the robot's path perpendicularly at three different predefined points (shown in Figure 16). The third batch of 6 sessions was collected with a focus on face-to-face encounters. The robot was asked to navigate along a looping trajectory and a person was told to walk along the same loop in the opposite direction (See Figure 16). Finally, we collected 15 additional sessions during a conference that was organized in the building. For these sessions, the layout of the Atrium was different and more people were present in the space without any instructions on how to behave around the robot.

For more "in-the-wild" results, we also collected data in the main hall of the building. After the initial mapping and the refinement runs, we collected 38 sessions, at different hours on different days, alternating between crowded times (See Figure 16) and more calm moments. The sessions collected in this space are not organized in a particular order, because they all were collected without any instructions given to the people

in the space. In these sessions (and also in the last batch of sessions in the atrium), we noticed several people trying to mess with the robot by acting in unexpected ways and sometimes had to stop the robot ourselves to avoid collisions. Ideally, we would like the robot to be able to predict any kind of behavior, even the disrupting ones, which is why we keep these sessions in the dataset. Most of the sessions were collected using the *NoPreds-Nav*, and some later sessions were collected with the *DeepSOGM-Nav*, using a network trained on earlier sessions. From a data collection and annotation point of view, this does not really matter. A different robot's behavior may induce different reactions from the people around it, but these cases rarely happen. The collected sessions are listed with details in Tables II and III. Our open dataset is called **UofT-Indoor-3D (UTIn3D)** and is available for the community to use. We share the lidar frames, the trajectories computed by PointMap, and our annotations. 3D lidar datasets with crowds of indoor pedestrians are not very common, and we hope that UTIn3D will be beneficial for the community.

We do not have quantitative measurements of the quality of the annotation on real data, but we can observe the results qualitatively. For the most part, the annotation quality is very good. The different classes are quite well split, with only a few leaks from one class to another, for example where people are close to tables and then moving away, as we can see in Figure 16. This type of mistake does not affect our navigation system so much as most encounters are happening far away from static objects like tables. Examples of annotated SOGMs can be seen in Figure 17.

C. SOGM Predictions in Real Scenarios

In this section, we focus on the evaluation of the network SOGM predictions. Similarly to [2], we compare the predicted SOGMs to labeled SOGMs annotated by our automated pipeline, using the same metrics, and considering only the *dynamic* class. In our first experiments shown in Table IV, we compare the performances of our network when trained on more and more data. To assess performance at different future horizons, mean average precision is computed for the 10th, 20th, and 30th layers of SOGMs at 1, 2, and 3 seconds ahead, respectively. We also measure the total Average precision on the whole SOGM. The relatively low values in this table are explained by the complexity of the task. The future movements in the scene are never written in advance and can be multimodal (several possible trajectories are always possible). Therefore, the predictions incorporate this uncertainty and become blurry as time advances. It means lower precision scores, which reduces the values for our metric. The actual performances of our predictions can be judged with the qualitative visualizations we provide.

First, in a lifelong learning manner, we use the UTIn3D-Atrium dataset, which has been split into 4 parts, and we train networks on increasing amounts of the dataset. We test on both UTIn3D validation sets, but we value the validation more in UTIn3D-Hall, as it contains a lot of "in-the-wild"

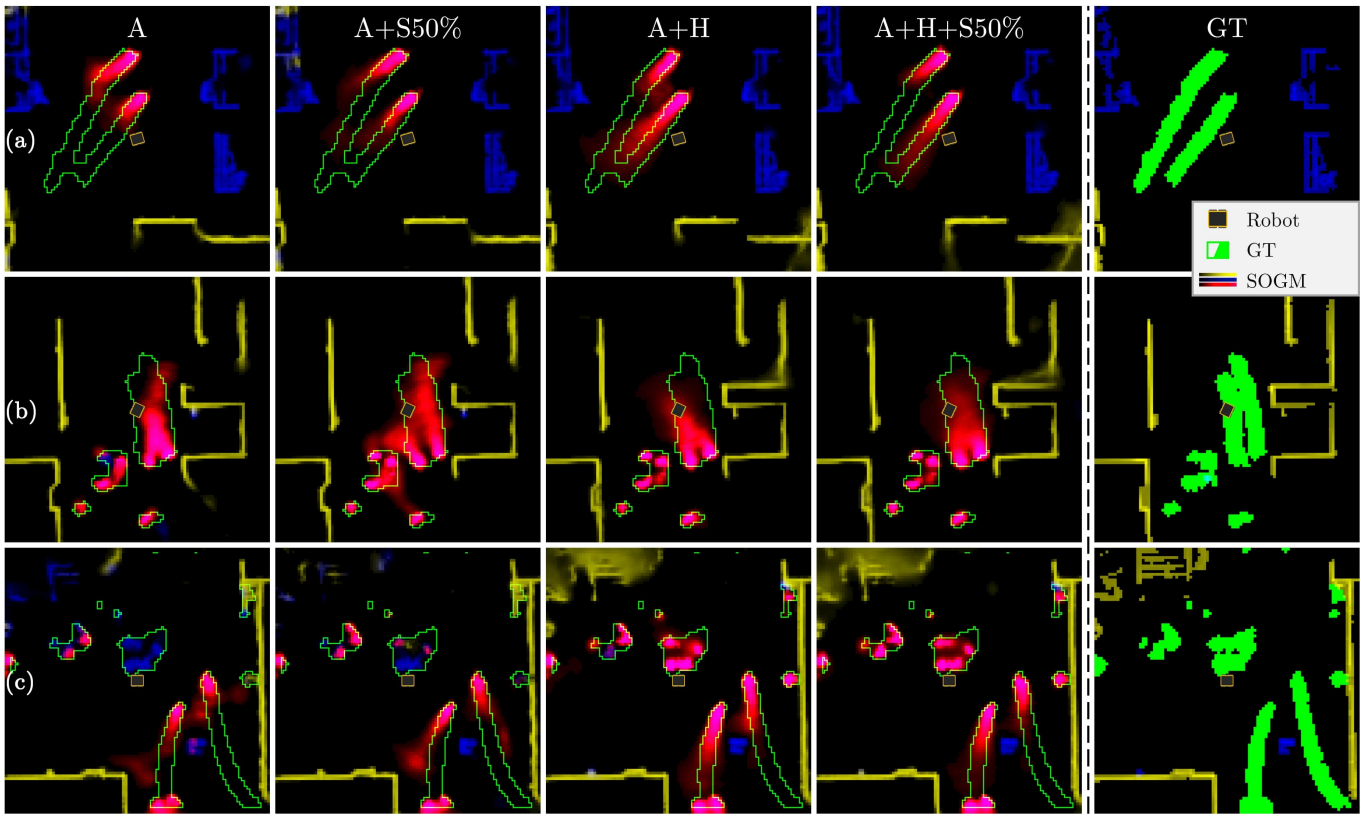


Fig. 17. Qualitative comparison of SOGMs predicted with networks trained on an increasing amount of data. On all these examples from the UTIn3D-Hall dataset, we see that the more data we add to the training set, the better the predictions get. Our best network is trained on a training set combining UTIn3D-Atrium, UTIn3D-Hall, and simulated data.

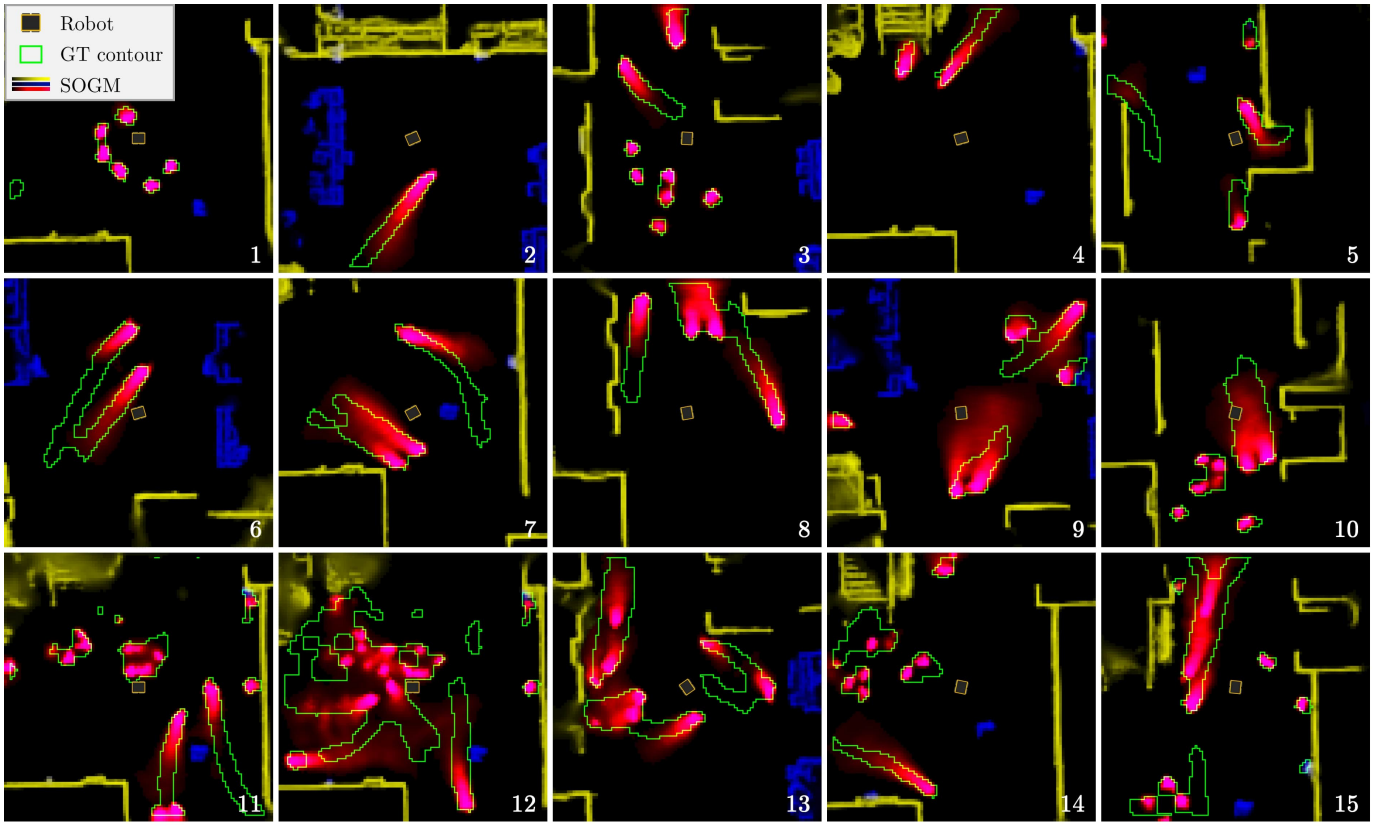


Fig. 18. Examples of in-the-wild SOGM predictions with our best network (trained on A+H+S50%), chosen in the UTIn3D-Hall validation set. Our network can handle various circumstances, from simple behaviors, like people standing around the robot, to extremely crowded and dynamic scenes.

dynamic obstacles. For this experiment, we see that increasing the amount and the diversity of data helps the network achieve better performances, which validates our assumption that a robot using our algorithm would be able to improve itself throughout its life. Because for other experiments, we used simulated data, we have a good opportunity to test the ability of our network to generalize to combinations of real-world and simulated data. We first notice that using only simulated data, the predictions are useless. Even if our simulated space is a copy of the UTIn3D-Atrium space, Sim-to-Real transfer is a very complex problem that we do not expect to solve here. However, when combining simulation data with real data in the training set, we see that the results are improved. The more we add simulated data to UTIn3D-Atrium, the better the performances are. It means, that without seeing any real data, the network is not able to generalize to this new unseen modality, but when given a few examples of the real data, the network can leverage the diversity of dynamic obstacles in the simulated data, at the same time as the specificity of the real data, to improve its performances. We believe that this is due to the fact that UTIn3D-Atrium does not contain a lot of dynamic obstacles (as shown in Table II). The network only needs some frames of the dataset to adapt to the particularity of real data and get better results when it relies more on the movements seen in the simulated data, with many more actors.

Then we also add UTIn3D-Hall to the training data. We notice a big step up in the performances on both validation sets. Following our analysis of the combined results with simulation, it makes sense, because UTIn3D-Hall contains a lot of dynamic obstacles (as shown in Table III), with diverse behaviors. We notice that by combining both real training datasets, we achieve much better results than by combining one real dataset and simulated data. We could expect this, because, like the simulated data, UTIn3D-Hall contains a lot of dynamic obstacles, but it is not very different in nature from UTIn3D-Atrium. It is interesting to note that the gap between simulation and reality affects the performance more than the gap between two different spaces with different room configurations. This result confirms that our network does not overfit the training data and that its predicted SOGMs can generalize well to multiple different spaces.

Eventually, we combine everything, achieving the best results of all on both validation sets. This final result finally confirms that our network has the ability to generalize to combinations of diverse real-world and simulated spaces. The more data we provide, the better the results become, which is exactly the goal of our approach, as the robot should be able to collect this data on its own throughout its life. Interestingly, when we add more simulated data to the training set, we see the opposite as before: a reduction in the performance. In this case, when the real dataset is large enough, we think that adding some examples of simulated data helps by providing more diversity, but relying too much on it makes the network worse on real validation.

We complete these quantitative results with qualitative examples of SOGM prediction for different training sets in

TABLE IV
EVALUATION OF THE SOGM PREDICTIONS WITH AN INCREASING AMOUNT OF DATA, AND WITH COMBINATIONS OF DATA COLLECTED IN THE REAL WORLD AND SIMULATED SPACES. WE PROVIDE THE MEAN AVERAGE PRECISION (%) AT GIVEN FUTURE TIMES (1S, 2S, AND 3S) AND ON THE WHOLE SOGM (TOTAL). BEST RESULTS ARE HIGHLIGHTED IN **BOLD** AND RESULTS WITH 10% OF THE BEST ONES ARE UNDERLINED.

	UTIn3D-A-val				UTIn3D-H-val			
Training data	1s	2s	3s	**Total**	1s	2s	3s	**Total**
only-S	3.0	0.4	0.1	4.2	8.7	1.2	0.4	8.5
A (1)	11.9	5.7	3.3	9.8	23.3	11.2	8.5	20.3
A (12)	13.9	5.9	3.9	10.6	29.9	11.5	6.9	21.9
A (123)	16.3	6.2	3.0	11.1	33.8	10.7	4.9	22.0
A (1234)	17.3	6.3	3.5	11.6	35.8	14.1	6.6	23.6
A+S20%	18.4	7.0	3.6	12.1	40.5	17.0	7.8	26.2
A+S50%	18.7	8.2	5.0	13.4	37.6	14.8	7.5	24.8
A+S80%	20.3	10.2	7.0	15.3	37.9	19.2	12.6	27.7
A+H	24.2	11.7	7.4	16.5	**56.9**	34.7	24.5	41.7
A+H+S20%	**26.7**	**14.8**	**9.5**	**19.1**	56.4	34.9	24.7	41.8
A+H+S50%	24.7	13.6	8.6	18.0	56.5	**35.6**	**25.6**	**42.3**
A+H+S80%	21.3	10.6	7.0	15.6	54.1	33.8	24.8	40.9

TABLE V
3D BACK-END ABLATION STUDY AND COMPARISON WITH A CLOSE SOTA METHOD, ON A+H+S20% TRAINING SET. METHODS WITH * USE THE ANNOTATED SOGMS AS INPUT AND ARE NOT USABLE BY OUR ROBOT IN REAL-TIME.

	UTIn3D-A-val				UTIn3D-H-val			
Method	1s	2s	3s	**Total**	1s	2s	3s	**Total**
Ours 3D+2D	26.7	14.8	9.5	19.1	**56.4**	**34.9**	**24.7**	**41.8**
Ours only2D	22.8	16.2	13.1	18.9	41.2	25.4	18.7	31.9
Ours only2D*	**45.8**	**32.5**	**25.8**	**36.9**	52.8	31.9	23.0	39.7
Lange et al. [55]*	30.9	21.5	15.9	27.0	18.9	14.4	11.4	21.8

Figure 17. Similarly to [2], we use a merged representation of the SOGM, where *dynamic* predictions from all layers are colored in red, with the corresponding labeled SOGM superimposed as a green contour. The better performances of the network trained on A+H+S50% can be visualized in all three examples. In example (a), we see two people walking in relatively free space. Using only A or A+S50% data, the network is not able to predict the trajectories very well, as it has never seen the Hall. When using A+H, the trajectory gets better and takes the shape of a banana distribution, a phenomenon we previously saw in simulation [2]. Finally, adding simulation with A+H+S50%, helps refine the predictions, with a banana shape much closer to the green contour. In example (b), we notice that without simulation (A and A+H), the predictions tend to merge for groups like these two persons. With simulation data (A+S50% and A+H+S50%), which contains a lot of examples with multiple actors, the predictions for groups get better. Eventually, in example (c) we notice groups of standing people that are classified as movable objects by the first networks and eventually classified as dynamic with A+H+S50%. We do not consider this as an improvement, but more as an open question: should standing people be classified as movable or dynamic? Here this is decided by the network itself, depending on the data it has seen. In UTIn3D-Atrium data, people are standing for long periods, as they are discussing in the context of a conference, in UTIn3D-Hall, with a lot of passage, people are usually stopping only briefly, but then moving again, which explains the difference in the predictions.

In addition, more examples of predictions from A+H+S50% are shown in 18 to visualize the capabilities of our best network on real data. We show simple examples with only a few people (1-5), multiple people walking together as groups (6-10), and complex crowded scenes (11-15). Among these examples, we can find many interesting predictions. First, we notice the banana shape distribution (better seen as animated SOGMs in the video) for one person but also groups (2, 4, 6-10, 14). We also see predictions of people fading when they get into elevators or stairs (4, 5, 13), or predictions expecting people to avoid the robot (5, 7, 9, 10).

Eventually, we provide a new ablation study that was missing from [2], where we compare the performances of our network when using the 3D back-end or not. We also compare to another 2D-based prediction method [55]. The only2D version of our network is implemented by replacing the 3D output features of the 3D back-end network with its 3D input features. The rest of the architecture remains unchanged. In this way, we avoid having to create a new input preprocessing method and simply leverage our 2D projection module for extracting 2D features from the 3D lidar input. We had to adapt the opensource code provided by [55] to our problem by modifying their input pipeline and mirroring our setup with 3 input frames and 40 output frames. Due to the sequential nature of their method, we had to feed annotated SOGMs as input to their network. Despite this significant advantage, Table V and Figure 19 show that this network struggles to predict SOGMs. Multiple factors potentially led to these poor performances. First, they originally use a different setup, where the predictions are made in the vehicle's ego-motion. In this scenario, even static objects appear to move in future images. Additionally, their network was designed with a recurrent architecture, which is not adapted to a short input sequence and long output sequence. It was also not optimized for strongly imbalanced data.

The results, shown in Table V and Figure 19, confirm the superiority of our 3D+2D network compared to its only2D counterpart. We even see that the 3D+2D network outperforms all the 2D-based networks on UTIn3D-H, which is the dataset containing the largest number of people moving in the space. We hypothesize that the human pose, which is only visible in 3D, is a strong indicator of future motion, explaining why the 2D networks struggle compared to our 3D+2D network.

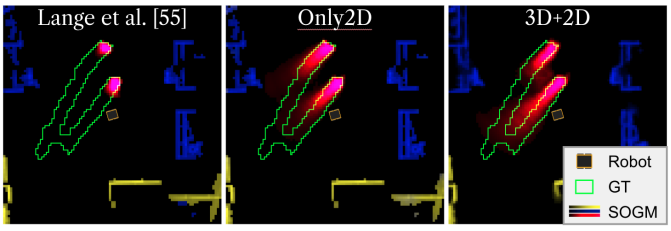


Fig. 19. Qualitative example of the ablation study presented in Table V.

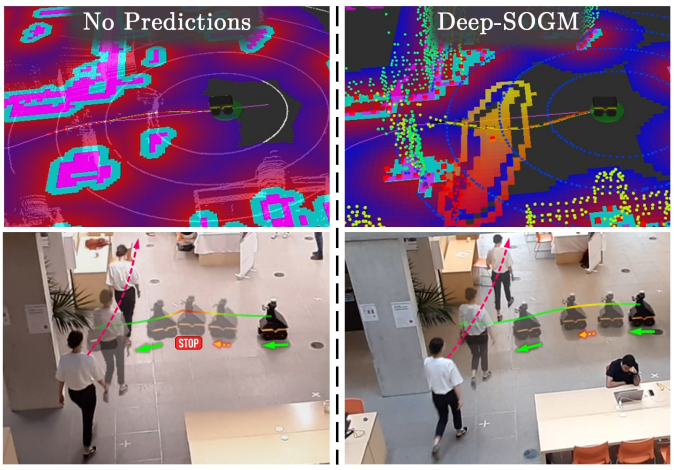


Fig. 20. Anecdotal example showing the benefit of our Deep-SOGM predictions on a real robot. Without predictions, the robot sees an obstacle on its left and plans to go on the right. But the person is walking in that direction, forcing the robot to stop. With our predictions, the robot anticipates the person's movement and makes a plan to yield, without having to stop.

D. Real Robot Navigation

The final step in our work is to test our navigation system in the real world, to validate the results we had in Section VI in simulation. In this section, we collect qualitative results and anecdotal examples, because we cannot reproduce a fair and accurate experimental process matching the one we have in simulation. We do not have ground truth information that the simulator would provide, and more importantly, it is very hard to reproduce multiple experiments with the same conditions to compare different methods.

First, we conduct a controlled experiment where people are asked to cross the path of the robot perpendicularly and compare the reactions of the robot when using the standard *NoPreds-Nav* or when using our *DeepSOGM-Nav* with and without time diffusion. For this experiment, we repeated the session two times for each system and collected the data. After annotating it, we could collect the distance to the closest dynamic obstacle and compute similar metrics to the ones used in the simulated experiments in Section VI. We use different thresholds for the risk ratios, adapted to the real setup we defined: a Low-Risk Ratio measuring the proportion of the session during which the distance is smaller than $1.5m$), and a High-Risk Ratio measuring the proportion of the session during which the distance is smaller than $0.6m$). We also add two more metrics based on encounter statistics. For this, we segment out each encounter between the robot and a dynamic obstacle as the period when the distance is smaller than $1.5m$. For each encounter we measure the duration and the minimum distance, then we average these values per session.

In Table VI, we find that *DeepSOGM-Nav* has the best performance in terms of safety and efficiency. It is very good at avoiding high-risk areas and keeping a higher minimum distance when crossing the path of people. We notice that without time diffusion for the SRM, *DeepSOGM-Nav* is faster but a lot riskier. Overall, even though we cannot ensure the exact same conditions every time, get enough repetitions for a good statistical evaluation, and get ground truth distance measurements; we still obtain similar results as in the simulation experiment.

TABLE VI
EVALUATION OF THE SAFETY AND EFFICIENCY OF OUR NAVIGATION SYSTEMS WITH DIFFERENT PREDICTIONS. WE CONDUCT TWO SESSIONS WITH EACH SYSTEM. BEST RESULTS ARE HIGHLIGHTED IN **BOLD** AND RESULTS WITH 10% OF THE BEST ONES ARE UNDERLINED.

Nav System	Low-Risk Ratio (<0.6 m)	High-Risk Ratio (<1.5 m)	Time to Finish (s)	Encounter min-dist (m, avg)	Encounter duration (s, avg)
DeepSOGM-Nav	17.2%	**0.2%**	91.7	**0.84**	**1.99**
	15.6%	1.9%	90.5	0.71	2.03
NoPreds-Nav	16.9%	2.8%	105.1	0.58	2.24
	16.3%	4.3%	103.4	0.51	2.43
DeepSOGM-Nav (no time diff)	21.8%	5.4%	88.7	0.51	2.42
	20.4%	3.4%	**86.4**	0.53	2.21

We also show what happens qualitatively during these experiments in Figure 20. We see that without predictions, the planner sees an object coming from its left, and plans to avoid it by turning to the right. The closer this object gets, the further to the right the planned trajectory is "pushed", until the point where the robot has to stop and readjust its trajectory to pass on the left behind the person. By doing this the robot gets into a risky situation. On the contrary, with predictions, the planner anticipates that the person is going to be on its right in a few seconds, and, from the beginning, plans a trajectory that passes behind the person, which is much safer and more efficient. The reactions of the robot when using predictions are much more similar to what normal persons would do when crossing each others' paths.

VIII. CONCLUSION

In this paper, we presented a self-supervised approach that provides a robot with the ability to anticipate future movements in a dynamic scene. It can be seen as an imperfect but efficient crystal ball that does not rely on any human annotation. To gain this ability, a robot only needs to navigate in dynamic scenes, and our automated annotation pipeline will create a training set from the collected data. Our network architecture can then be trained to predict Spatiotemporal Occupancy Grid Maps, which contain information about the future of dynamic scenes. Finally, the robot can use this network within a simple navigation system. It gets this information in real-time, transform it into Spatiotemporal Risk Maps, and uses it in a local planner able to avoid high-risk areas in space and time.

Adapted from our previous work that was only tested on simulated data, our pipeline has been heavily improved, and thoroughly tested on real data. We compared our navigation pipeline with different kinds of predictions in simulation, validated the results on a real robot, and showed compelling SOGM prediction results in various circumstances. In addition, we provide a new 3D lidar dataset with indoor pedestrians, which contains lidar frames, with our annotations computed automatically. This dataset can be used to reproduce our results or explore new potential methods for future predictions in dynamic scenes.

REFERENCES

[1] H. Thomas, B. Agro, M. Gridseth, J. Zhang, and T. D. Barfoot, "Self-supervised learning of lidar segmentation for autonomous indoor navigation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.

[2] H. Thomas, M. G. d. S. Aurin, J. Zhang, and T. D. Barfoot, "Learning spatiotemporal occupancy grid maps for lifelong navigation in dynamic scenes," in *2022 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2022.

[3] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2009, pp. 3931–3936.

[4] K. M. Kitani, B. D. Ziebart, J. A. Bagnell, and M. Hebert, "Activity forecasting," in *European Conference on Computer Vision*. Springer, 2012, pp. 201–214.

[5] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing icp variants on real-world data sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[6] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9, 2014.

[7] E. Mendes, P. Koch, and S. Lacroix, "Icp-based pose-graph slam," in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2016, pp. 195–200.

[8] J.-E. Deschaud, "Imls-slam: scan-to-model matching based on 3d data," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2480–2485.

[9] H. Moravec and A. Elfes, "High resolution maps from wide angle sonar," in *Proceedings. 1985 IEEE international conference on robotics and automation*, vol. 2. IEEE, 1985, pp. 116–121.

[10] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison *et al.*, "Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*, 2011, pp. 559–568.

[11] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[12] F. Pomerleau, P. Krüsi, F. Colas, P. Furgale, and R. Siegwart, "Long-term 3d map maintenance in dynamic environments," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3712–3719.

[13] J. Biswas and M. Veloso, "Episodic non-markov localization: Reasoning about short-term and long-term features," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3969–3974.

[14] A. Dewan, G. L. Oliveira, and W. Burgard, "Deep semantic classification for 3d lidar data," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 3544–3549.

[15] B. Sofman, E. Lin, J. A. Bagnell, J. Cole, N. Vandapel, and A. Stentz, "Improving robot navigation through self-supervised online learning," *Journal of Field Robotics*, vol. 23, no. 11-12, pp. 1059–1075, 2006.

[16] A. Lookingbill, J. Rogers, D. Lieb, J. Curry, and S. Thrun, "Reverse optical flow for self-supervised adaptive autonomous robot navigation," *International Journal of Computer Vision*, vol. 74, no. 3, pp. 287–302, 2007.

[17] R. Hadsell, P. Sermanet, J. Ben, A. Erkan, M. Scoffier, K. Kavukcuoglu, U. Muller, and Y. LeCun, "Learning long-range vision for autonomous off-road driving," *Journal of Field Robotics*, vol. 26, no. 2, pp. 120–144, 2009.

[18] C. A. Brooks and K. Iagnemma, "Self-supervised terrain classification for planetary surface exploration rovers," *Journal of Field Robotics*, vol. 29, no. 3, pp. 445–468, 2012.

[19] B. Ridge, A. Leonardis, A. Ude, M. Deniša, and D. Skočaj, "Self-supervised online learning of basic object push affordances," *International Journal of Advanced Robotic Systems*, vol. 12, no. 3, p. 24, 2015.

[20] M. Nava, J. Guzzi, R. O. Chavez-Garcia, L. M. Gambardella, and A. Giusti, "Learning long-range perception using self-supervision from short-range sensors and odometry," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1279–1286, 2019.

[21] L. Zhang, L. Wei, P. Shen, W. Wei, G. Zhu, and J. Song, "Semantic slam based on object detection and improved octomap," *IEEE Access*, vol. 6, pp. 75 545–75 559, 2018.

[22] K. Wang, Y. Lin, L. Wang, L. Han, M. Hua, X. Wang, S. Lian, and B. Huang, "A unified framework for mutual improvement of slam and semantic segmentation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5224–5230.

[23] X. Chen, A. Milioto, E. Palazzolo, P. Giguère, J. Behley, and C. Stachniss, "Suma++: Efficient lidar-based semantic slam," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 4530–4537.

[24] L. Sun, Z. Yan, A. Zaganidis, C. Zhao, and T. Duckett, "Recurrent-octomap: Learning state-based map refinement for long-term semantic mapping with 3-d-lidar data," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3749–3756, 2018.

[25] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 961–971.

[26] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2255–2264.

[27] K. D. Katyal, G. D. Hager, and C.-M. Huang, "Intent-aware pedestrian prediction for adaptive crowd navigation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3277–3283.

[28] R. Peddi, C. Di Franco, S. Gao, and N. Bezzo, "A data-driven framework for proactive intention-aware motion planning of a robot in a human environment," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5738–5744.

[29] A. J. Sathyamoorthy, U. Patel, T. Guan, and D. Manocha, "Frozone: Freezing-free, pedestrian-friendly navigation in human crowds," *IEEE Robotics and Automation Letters*, vol. 5, no. 3, pp. 4352–4359, 2020.

[30] W. Luo, B. Yang, and R. Urtasun, "Fast and furious: Real time end-to-end 3d detection, tracking and motion forecasting with a single convolutional net," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018, pp. 3569–3577.

[31] S. Casas, W. Luo, and R. Urtasun, "Intentnet: Learning to predict intention from raw sensor data," in *Conference on Robot Learning*. PMLR, 2018, pp. 947–956.

[32] V. Tolani, S. Bansal, A. Faust, and C. Tomlin, "Visual navigation among humans with optimal control as a supervisor," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2288–2295, 2021.

[33] A. Jain, S. Casas, R. Liao, Y. Xiong, S. Feng, S. Segal, and R. Urtasun, "Discrete residual flow for probabilistic pedestrian behavior prediction," in *Conference on Robot Learning*. PMLR, 2020, pp. 407–419.

[34] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, "Human motion trajectory prediction: A survey," *The International Journal of Robotics Research*, vol. 39, no. 8, pp. 895–935, 2020.

[35] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1343–1350.

[36] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6252–6259.

[37] J. Liang, U. Patel, A. Sathyamoorthy, and D. Manocha, "Crowd-steer: Realtime smooth and collision-free robot navigation in densely crowded scenarios trained using high-fidelity simulation," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI*, vol. 2020, 2020, pp. 4221–4228.

[38] A. J. Sathyamoorthy, J. Liang, U. Patel, T. Guan, R. Chandra, and D. Manocha, "Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 11 345–11 352.

[39] M. Everett, Y. F. Chen, and J. P. How, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," *IEEE Access*, vol. 9, pp. 10 357–10 377, 2021.

[40] R. Strudel, R. Garcia, J. Carpentier, J.-P. Laumond, I. Laptev, and C. Schmid, "Learning obstacle representations for neural motion planning," 2020.

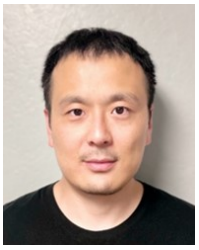
[41] L. Liu, D. Dugas, G. Cesari, R. Siegwart, and R. Dubé, "Robot navigation in crowded environments using deep reinforcement learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5671–5677.

[42] U. Patel, N. K. S. Kumar, A. J. Sathyamoorthy, and D. Manocha, "Dwa-rl: Dynamically feasible deep reinforcement learning policy for robot navigation among mobile obstacles," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020.

[43] A. Pierson, C.-I. Vasile, A. Gandhi, W. Schwarting, S. Karaman, and D. Rus, "Dynamic risk density for autonomous navigation in cluttered environments without object detection," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 5807–5814.
[44] Z. Huang, W. Schwarting, A. Pierson, H. Guo, M. Ang, and D. Rus, "Safe path planning with multi-model risk level sets," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 6268–6275.
[45] J. F. Fisac, A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, S. Wang, C. J. Tomlin, and A. D. Dragan, "Probabilistically safe robot planning with confidence-based human predictions," *arXiv preprint arXiv:1806.00109*, 2018.
[46] A. Bajcsy, S. L. Herbert, D. Fridovich-Keil, J. F. Fisac, S. Deglurkar, A. D. Dragan, and C. J. Tomlin, "A scalable framework for real-time multi-robot, multi-human collision avoidance," in *2019 international conference on robotics and automation (ICRA)*. IEEE, 2019, pp. 936–943.
[47] S. Bansal, A. Bajcsy, E. Ratner, A. D. Dragan, and C. J. Tomlin, "A hamilton-jacobi reachability-based framework for predicting and analyzing human motion for safe planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7149–7155.
[48] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *arXiv preprint arXiv:1605.08104*, 2016.
[49] Y. Wang, Z. Gao, M. Long, J. Wang, and S. Y. Philip, "Predrnn++: Towards a resolution of the deep-in-time dilemma in spatiotemporal predictive learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 5123–5132.
[50] Y. Wang, J. Zhang, H. Zhu, M. Long, J. Wang, and P. S. Yu, "Memory in memory: A predictive neural network for learning higher-order non-stationarity from spatiotemporal dynamics," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9154–9162.
[51] N. Mohajerin and M. Rohani, "Multi-step prediction of occupancy grid maps with recurrent neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 600–10 608.
[52] M. Schreiber, V. Belagiannis, C. Gläser, and K. Dietmayer, "Motion estimation in occupancy grid maps in stationary settings using recurrent neural networks," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 8587–8593.
[53] M. Itkina, K. Driggs-Campbell, and M. J. Kochenderfer, "Dynamic environment prediction in urban scenes using recurrent representation learning," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019, pp. 2052–2059.
[54] M. Toyungyernsub, M. Itkina, R. Senanayake, and M. J. Kochenderfer, "Double-prong convlstm for spatiotemporal occupancy prediction in dynamic environments," in *2021 International Conference on Robotics and Automation (ICRA)*. IEEE, 2021.
[55] B. Lange, M. Itkina, and M. J. Kochenderfer, "Attention augmented convlstm for environment prediction," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1346–1353.
[56] R. Mahjourian, J. Kim, Y. Chai, M. Tan, B. Sapp, and D. Anguelov, "Occupancy flow fields for motion forecasting in autonomous driving," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 5639–5646, 2022.
[57] S. Hoermann, M. Bach, and K. Dietmayer, "Dynamic occupancy grid prediction for urban autonomous driving: A deep learning approach with fully automatic labeling," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2056–2063.
[58] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, 1985, pp. 245–254.
[59] G. Matheron and J. Serra, "The birth of mathematical morphology," in *Proc. 6th Intl. Symp. Mathematical Morphology*. Sydney, Australia, 2002, pp. 1–16.
[60] S. Calderon and T. Boubekeur, "Point morphology," *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, pp. 1–13, 2014.
[61] J. Balado, P. Van Oosterom, L. Díaz-Vilariño, and M. Meijers, "Mathematical morphology directly applied to point cloud data," *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 168, pp. 208–220, 2020.
[62] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.
[63] H. Thomas, C. R. Qi, J.-E. Deschaud, B. Marcotegui, F. Goulette, and L. J. Guibas, "Kpconv: Flexible and deformable convolution for point clouds," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 6411–6420.
[64] C. Rösmann, F. Hoffmann, and T. Bertram, "Planning of multiple robot trajectories in distinctive topologies," in *2015 European Conference on Mobile Robots (ECMR)*. IEEE, 2015, pp. 1–6.
[65] ——, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017.

Hugues Thomas (Member, IEEE) received the Ph.D. degree from Mines Paristech, Université Paris Sciences et Lettres (PSL), Paris, France, in 2019. He is currently a Postdoctoral Researcher with the Autonomous Space Robotics Lab (ASRL), University of Toronto, which develops methods to allow mobile robots to operate in large-scale, unstructured, 3-D environments, using rich onboard sensing (e.g., cameras and laser rangefinders) and computation.

His research interests focus on deep learning, 3D point clouds and robotics. He has developed a novel convolutional operator for 3D point clouds, KPConv, designed deep network architectures for 3D object classification, object part segmentation and semantic scene parsing, explored the problems of rotation invariance and equivariance in 3D convolutions, and studied the application of self-supervised learning to robotics applications.



Jian Zhang (Member, IEEE) received the B.S. degree in mechatronics from Zhejiang University, Hangzhou, China, in 2010, and the Ph.D. degree in mechanical engineering with robotics specialty from Purdue University, West Lafayette, IN, USA, in 2015. He is currently an R&D manager at Apple Inc., USA. His research interests are robotics, autonomous systems, deep learning, and embodied artificial intelligence.

Timothy D. Barfoot (Fellow, IEEE) received the Ph.D. degree from University of Toronto, Toronto, ON, Canada, in 2002. He currently leads the Autonomous Space Robotics Lab (ASRL), University of Toronto, and works on the area of autonomy for mobile robots targeting a variety of applications.

He held a Canada Research Chair (Tier 2) for the full 10 years and was an Early Researcher Awardee in the Province of Ontario. Timothy was also a Visiting Professor at the University of Oxford in 2013 and recently completed a leave as Director of Autonomous Systems at Apple in California in 2017-9. He is currently the Chair of the UofT Engineering Science Robotics Major, Associate Director of the UofT Robotics Institute, Faculty Affiliate of the Vector Institute, and co-Faculty Advisor of UofT's self-driving car team that won the SAE Autodrive competition five years in a row. He sits on the Editorial Boards of the International Journal of Robotics Research (IJRR) and Field Robotics (FR), the Foundation Board of Robotics: Science and Systems (RSS), and served as the General Chair of Field and Service Robotics (FSR) 2015, which was held in Toronto. He is the author of a book, "State Estimation for Robotics".