Deep Active Localization

Sai Krishna Gottipati, Keehong Seo, Dhaivat Bhatt, Vincent Mai, Krishna Murthy, and Liam Paull

***Abstract*—Active localization consists of generating robot actions that allow it to maximally disambiguate its pose within a reference map. Traditional approaches use an information-theoretic criterion for action selection and hand-crafted perceptual models. In this work we propose an end-to-end differentiable method for learning to take informative actions that is trainable entirely in simulation and then transferable to real robot hardware with zero refinement. The system is composed of two learned modules: a convolutional neural network for perception, and a deep reinforcement learned planning module. We leverage a multi-scale approach in the perceptual model since the accuracy needed to take actions using reinforcement learning is much less than the accuracy needed for robot control. We demonstrate that the resulting system outperforms traditional approach for either perception or planning. We also demonstrate our approach's robustness to different map configurations and other nuisance parameters through the use of domain randomization in training. The code has been released: https://github.com/montrealrobotics/dal and is compatible with the OpenAI gym framework, as well as the Gazebo simulator.**

***Index Terms*—Localization, deep learning in robotics and automation, motion and path planning, AI-based methods.**

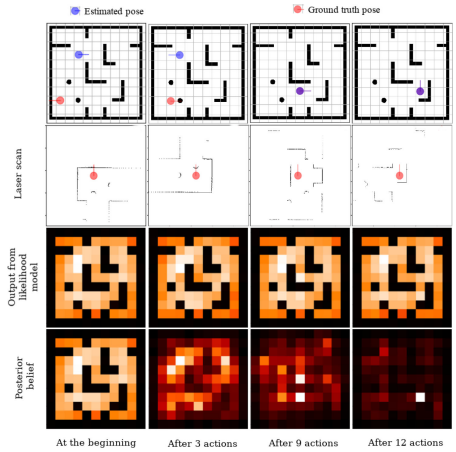


Fig. 1. *Deep active localization (DAL)* demonstrated in simulation. At first, the robot is uncertain of its pose relative to the given map, but as it executes actions it converges onto a true location (see posterior belief improving in the fourth row - lighter color denotes more probability in the posterior).

I. Introduction

LOCALIZATION against a global map, or *global* localization, is a prerequisite for any robotics task where a robot must know where it is (e.g. any task involving navigation). In such settings, it is beneficial for the agent to first select actions in order to disambiguate its location within the environment (map). This is referred to as "*active* localization".

Traditional methods for localization, such as Markov Localization (ML) [1] and Adaptive Monte Carlo Localization (AMCL) [2] are "*passive*" (agnostic to how actions are selected). They provide a recursive framework for updating an approximation of the state belief posterior as new measurements arrive. In the case of ML, this is usually done by some type of discretization of the state space (i.e. a fixed grid) and in AMCL this is achieved by maintaining a set of *particles* (state hypotheses). While these methods have seen widespread success in practice, they are still fundamentally limited since the map representation is hand-engineered and specifically tailored for the given on-board robot sensor. A common example is the pairing of the occupancy grid map [3] with the laser scanner since the measurement likelihood can be computed efficiently and in closed form with scan matching [4]. However, this choice of representation can be sub-optimal and inflexible. Furthermore, sensor parameters such as error covariances tend to be hand-tuned for performance, which is time consuming and error-inducing.

Active variants of ML and AMCL are classical ways to solve the active localization problem [5]–[9]. These methods typically leverage an information-theoretic measure to evaluate the benefit of visiting unseen poses. Again, these hand-tuned heuristics tends to lead to over-fitting of a particular algorithm to a specific robot-sensor-environment setup.

Passive deep learning localization algorithms [10]–[13] are not directly extendable to the active localization domain in the same fashion because they tend not to output calibrated measures of uncertainty. However, other learning-based methods have been built that are dedicated specifically to this task, such as Active Neural Localization (ANL) [14]. For a detailed analysis, please see Section II.

Deep Reinforcement Learning (DRL) has seen an incredible recent success on some robotics tasks such as manipulation [15] and navigation [16], albeit predominantly in simulation. In order for a DRL agent to be trained in simulation and deployed on a real robot either 1) the reality gap is small or is overcome with training techniques such as Domain Randomization (DR) such that no refinement of the policy is needed on the real robot (zero-shot transfer), or 2) the agent policy is fine-tuned on the real robot after primarily training in simulation. The latter option

Manuscript received February 24, 2019; accepted July 12, 2019. Date of publication August 1, 2019; date of current version August 19, 2019. This letter was recommended for publication by Associate Editor N. Sunderhauf and Editor C. Stachniss upon evaluation of the reviewers' comments. This work was supported in part by Samsung Advanced Institute of Technology (SAIT), in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), in part by the Fonds de recherche nature et technologie Québec (FQRNT) and in part by the Canada CIFAR AI Chairs Program. *(Sai Krishna Gottipati and Keehong Seo contributed equally to this work.) (Corresponding author: Sai Krishna Gottipati.)*

S. K. Gottipati, D. Bhatt, V. Mai, K. Murthy, and L. Paull are with the Mila and DIRO, Universite de Montreal, Montreal, QC H3T 1J4, Canada (e-mail: saikrishnagv1996@gmail.com; dhaivat1994@gmail.com; vincent.mai@umontreal.ca; krrish94@gmail.com; paulll@iro.umontreal.ca).

K. Seo is with the SAIT, Samsung Electronics Co. Ltd. Suwon, Gyeonggi-do 16677, South Korea (e-mail: keehong@gmail.com).

Digital Object Identifier 10.1109/LRA.2019.2932575

reduces the burden on the simulation fidelity and training regime, but fine-tuning on the robot can be difficult or impossible if the reward is determined by leveraging ground truth parameters (like localization) that are only available in the simulator. In this work, we show that the framework of DRL holds potential for the task of active localization, and can be used to train policies that transfer to a robot in real-world indoor environments. Borrowing concepts from Bayesian filtering [17], we define a reward as a function of the posterior belief. Specifically, our reward is defined as the probability of posterior belief at ground truth pose.

A. Contributions

In summary we claim the following contributions:

- We propose a multi-layer learned likelihood model which can be trained in simulation from automatically labeled data and then refined in an end-to-end manner,
- We have integrated this into and end-to-end deep RL system that do both high-level action selection and low-level robot control,
- We show that this method works for *zero-shot* transfer onto a real robot and outperforms classical methods.

Furthermore, we have developed automated processes for map generation, domain randomization [18] and likelihood and policy model training, that, in combination with our open-source openAI gym and Gazebo compatible code, allows for easy replication of our method and the baselines.

II. Related Work

Localization is an extremely well-studied robotics problem. We present a treatment of relevant *active* and *passive* approaches to robot localization, drawing parallels and contrasts to our method.

A. Passive Localization

Passive localization methods can broadly be classified into two representative categories: *Bayesian* filtering methods and learning-based methods. Although related, we do not cover visual place recognition approaches as their primary intention is to provide *coarse* location estimates. We refer the interested reader to [19] for an excellent survey.

Bayesian Filtering-Based Methods: Off-the-shelf localization modules such as Extended Kalman Filter (EKF)-based localization [17], Markov localization [1] and Adaptive Monte-Carlo localization (AMCL) [2] are based on a Bayesian filtering viewpoint of localization. All these methods begin with a prior (usually uniform) *belief* of a robot's pose and recursively update the belief as new (noisy) measurements arrive. The representation of the belief is what fundamentally differentiates each approach. EKF-based methods assume a Gaussian distribution to characterize the belief, while Markov localization [1] and AMCL [2] use non-parameteric distributions to characterize the belief (histograms and particles respectively). Consequently, EKF localization lacks the capability to characterize multiple belief hypotheses (owing to the unimodal Gaussian prior), while Markov localization and AMCL suffer computational overhead as the population of the non-parameteric distribution increases.

Learning-Based Methods: Robot localization with learning-based methods has a long history. For example, in the seminal work of Oore *et al.* [20], the authors use a learning-based approach to predict sonar readings from different locations in a map. More recently, approaches such as PoseNet [10] and VLocNet [11] perform *visual* localization by training a convolutional neural network to regress to *scene coordinates*, given an image [10] or a sequence of images [11]. [21] used generative query networks for *implicitly maintaining map representation* to localize. Recently, differentiable particle filters (DPFs) have also been proposed for global localization [12], [13]. However, such approaches need precisely annotated data to train a *PoseNet*, *VLocnet*, or a *DPF* for each new environment that the robot is deployed in. In contrast, DAL transfers essentially *zero-shot* across simulated environments, and from a simulator to the real-world.

B. Active Localization

In terms of *active* localization methods, there are again two broad categories: information-theoretic approaches and learning-based approaches.

Information Theoretic Methods: Burgard *et al.* introduced *active* localization in their seminal work [5]. They demonstrated that, rather than passively driving a robot around, picking actions that reduce the expected localization uncertainty results in a better localization. Using an entropy measure characterized as a mixture-of-Gaussians, they demonstrate that the framework of Markov localization [1] can be extended to action selection. This work was more of a *proof-of-concept*, as the applicability of this method is confined to low-dimensional state-spaces, where entropy computation can be carried out efficiently. Since then, several other approaches [22]–[27] use a similar, information gain maximization cost for the task of active localization or SLAM.

Learning-Based Methods: Active Neural Localization (ANL) [14] is the first known work to employ a learned model for active localization from images and has inspired many of our design choices.

Similar to our approach, ANL comprises two learnable modules, a perceptual model and a policy model, that can be trained end-to-end in a differentiable Bayesian framework. We use a similar representation (grid-based) and make use of the clever Hadamard product to compute the belief posterior. The main benefit of ANL is that the likelihood model can be trained jointly using the rewards from the RL model. This enforces that the likelihood model that is learned is also most useful for the downstream task (active localization).

However, ANL work makes several assumptions that limit its ability to scale to realistic-sized environments and to be transferred to real robot hardware. Specifically, the transition functions in ANL are assumed to be deterministic, an assumption which does not apply well to real robot hardware. In practice, the robot will drift from its grid centroids and at that point the performance of the method will degrade because there is no map data corresponding to off grid locations. In our method (DAL), this issue is handled with a combination of techniques: 1) A non-deterministic transition model is implemented in the simulator, 2) We leverage a hierarchical likelihood estimation method that can estimate at a higher resolution the true location of the robot in a grid cell and therefore compensate for the drift over time, 3) We have a low-level non-RL based feedback controller that executes these reference actions, and 4) We leverage domain randomization in the simulator to account for the possibility that our simulated models are incorrect. The issues of scalability in ANL is related to the fact that a fixed resolution grid must be overlayed over the entire map, similar to the original Markov localization [1]. As the environment increases, either

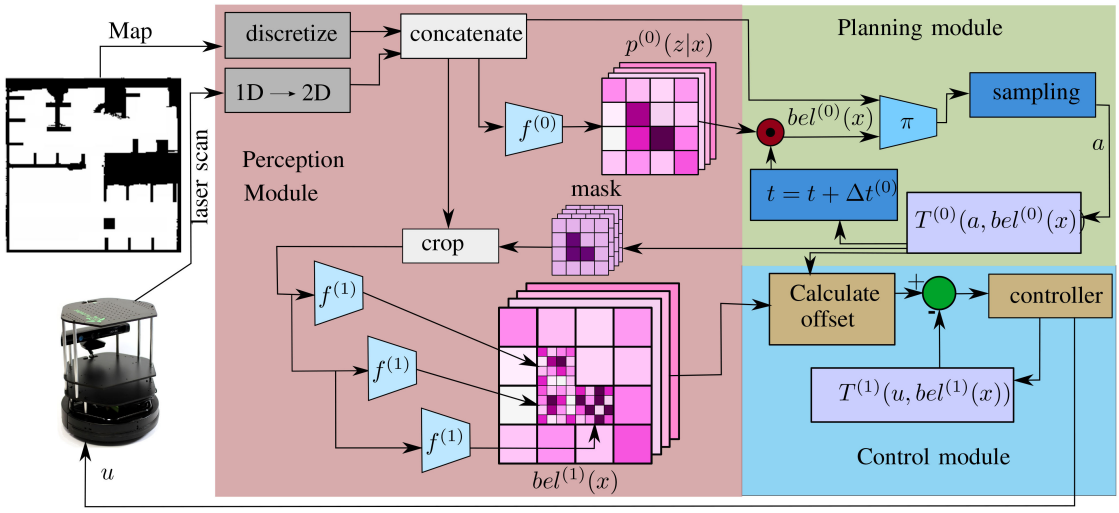


Fig. 2. Deep active localization (DAL) takes a map and sensor data (laser scan) and generates control actions in an end-to-end differentiable framework that includes learned perceptual modules (neural networks) at different scales and a learned policy network that is trained with reinforcement learning (RL). We maintain a coarse low dimensional pose estimate for RL, but then refine it to generate a higher precision pose estimate that is used for robot control.

the grid cells' size increases, or the number of grid cells must increase. For larger cells, the distance that the robot must travel between grid centroids will increase and the problem previously discussed will be exacerbated. In DAL, we propose a hierarchical likelihood estimation approach, which decouples the resolution of the likelihood resolution from the distance that the robot travels during each macro action selected by the RL model.

III. Problem Setup and Overview of Approach

A mobile robot is equipped with a sensor that provides exteroceptive measurements z_t used for localization and proprioceptive measurements used for odometry o_t (this could be the same physical sensor). The robot moves through the map by sending control inputs u_t to its actuators. The agent is provided with a map $\mathcal{M}$.

Problem (**Active Localization**) Assuming that the agent is placed at some random point in the map, find the sequence of control inputs, $u_{1:T}$ that allow it to maximally disambiguate its pose within the map.

Active localization solutions usually take the form:

$$u^*_{1:T} = \underset{u_{1:T}}{\operatorname{argmax}}\, f(bel(x_T), x^*) \tag{1}$$

where the function f quantifies the weight in the state belief posterior at the end of the horizon T, $bel(x_T)$ at the ground truth pose x^*.

Any algorithm used to solve the optimization in (1) must jointly consider both the *perception* task (i.e., how the belief posterior is being updated) as well as the planning and control problem that is used to generate the control actions conditioned on the belief.

Our proposed "deep active localization" approach to solving the active localization problem is summarized in Fig. 2 and Algorithm 1.[1] At each level in the hierarchy, i, we define a grid representation of size $N^{(i)} \times M^{(i)} \times \Theta^{(i)}$ which represents the state space $\mathcal{X}$. The belief posterior is represented as a matrix where $bel(x_t = [n, m, \theta])$ is the probability mass at location $[m, n, \theta]$.

The robot is provided with a map, $\mathcal{M}$ as input. Each time a sensor input, z_t is received, it is converted to a 2D image and combined with $\mathcal{M}$ to form the input to the network at level 0, $f^{(0)}(\mathcal{M}, z_t)$, which produces a coarse measurement likelihood, $p^{(0)}(z_t|x)$. The measurement likelihood is combined with the prior belief (via an element-wise product) to produce the belief posterior. The belief posterior is fed as input to the Reinforcement Learning (RL) model. It is important in order for this procedure to train efficiently that the input is relatively low dimensional. The RL policy is sampled to generate an action, a_t from the set $\{\texttt{fwd}, \texttt{left}, \texttt{right}\}$. This action produces a goal pose that is either the centroid of an adjacent cell at the same orientation, or a pure rotation of $\pm 2\pi/|\Theta^{(0)}|$, where $\Theta^{(0)}$ is the number of uniformly spaced discrete angles being considered at level 0. The action and belief posterior are fed through a noisy transition function to generate the belief prior at the next timestep:

$$\begin{aligned} \bar{bel}^{(0)}(x_{t+1} = [n, m, \theta]) &= T^{(0)}(a_k, bel(x_t)) \\ &= \begin{cases} bel^{(0)}(x_t = [n, m, (\theta + 2\pi/|\Theta^{(0)}|)] & a_k = \text{right} \\ bel^{(0)}(x_t = [n, m, (\theta - 2\pi/|\Theta^{(0)}|)] & a_k = \text{left} \\ bel^{(0)}(x_t = [n + l_n \cos(\theta), m & \\ \qquad + l_m \sin(\theta), \theta]) & a_k = \text{fwd} \end{cases} \end{aligned} \tag{2}$$

where l_n and l_m are the distance between adjacent cell centroids in the $N^{(0)}$ and $M^{(0)}$ directions respectively. For increased robustness, noise is injected into this transition model to represent the fact that the cell transition is not deterministic.

Since each time the robot executes an action it will not arrive exactly at the centroid of the adjacent cell due to dead reckoning error, we must account for the error accrued and compensate for it. We refine our coarse measurement likelihood to predict where *within* the cell our agent actually is to compute a correction.

[1] Conventions: superscripts $x^{(i)}$ denote levels in the hierarchy, subscripts denote time indices, and preceding superscripts ${}^{\{r\}}x$ denote reference frames. In the absence of a frame it is assumed that the variable is in the map fixed (global) frame.

Algorithm 1: Deep Active Localization.

procedure DAL $(\mathcal{M}, f^{(0)}, f^{(1)}, \pi)$
  Input:
  $\mathcal{M}$: Map of the environment
  $f^{(0)}$: Trained likelihood model for level-0
  $f^{(1)}$: Trained likelihood model for level-1
  π: Trained policy model
  while True **do**
    Read LiDAR scan z_t
    Obtain high and low res scan images S_h and S_l
    $p^{(0)}(z_t|x) \leftarrow f^{(0)}$
    (M_h, S_h) ▷ Low-res likelihood
    $bel^{(0)} \leftarrow p^{(0)}(z_t|x) \odot \bar{bel}^{(0)}$ ▷ Update belief
    $p(a|s) \leftarrow \pi(bel^{(0)}, M_l, S_l)$ ▷ Get action prob
    $a \leftarrow sample(p(a|s))$ ▷ Sample an action
    $c_t \leftarrow argmax(bel^{(0)})$ ▷ Get current pose
    $g_t \leftarrow$ get-next-pose(c_t, a)
    $\bar{bel}^{(0)} \leftarrow$ transition-belief$(bel^{(0)}, a)$
    find top n cells in $p^{(0)}(z_t|x)$
    for $i \in [0, n]$ **do**
      $M_c, S_c \leftarrow$ crop$(M_h, S_h, (x_i, y_i))$
      $p^{(1)}(z_t|x)[block_i] \leftarrow f^{(1)}(M_c, S_c)$
    $bel^{(1)} \leftarrow p^{(1)}(z_t|x) \odot \bar{bel}^{(1)}$ ▷ High-res belief
    $c_t^{(1)} \leftarrow argmax(bel^{(1)})$ ▷ Pose: max belief
    $u \leftarrow control(c_t^{(1)}, g_t)$

This offset within the cell is used to calculate an offset for the next action so that the error with respect to the grid centroids is *bounded over time*. To achieve this, we chose the most likely cells from the coarse belief $bel^{(0)}(x)$ and refine the measurement likelihood to determine a more precise estimate of the location of the agent within the cell. This is used to calculate an offset that is added to the relative pose transformations between adjacent cells to calculate an actual reference location in the robot frame, r: ${}^{r}x_{ref}$. We use a standard tracking controller to generate control inputs, u_t, which are sent to the robot and a standard kino-dynamic transition model $T^{(1)}(u, bel^{(1)}(x))$ to dead-reckon towards the reference location.

IV. Perception

The objectives of the perception system are:

1) To provide a measurement likelihood to be used by the RL agent, which is trainable in an end-to-end manner,
2) To provide a refined estimate of the pose to the inner loop controller so that the error induced by dead reckoning may be bounded,
3) To be fully trainable in simulation,
4) Not to require any information other than the (possibly noisy) map at test time.

A. Learning Measurement Models

In typical robotics pipelines, the measurement likelihood is constructed using custom metrics, based on models of the sensors in question. Inevitably, some elements of the model are imprecise. For example, covariances in visual odometry models are typically tuned manually since closed-form solutions are difficult to obtain. Additionally, the Gaussian assumption (e.g., in an EKF) is clearly inappropriate for the task of global localization, as evidenced by the prevalence of Monte Carlo based solutions [2].

In our setting, given a map of the environment $\mathcal{M}$ and scan input z_t, we want to *learn* the likelihood of the robot's pose at all candidate points on a multi-resolution grid.

Following the notation defined in Section III, the output of the likelihood model at time t is given by:

$$p^{(0)}(z_t|x) = f_\phi^{(0)}(\mathcal{M}, z_t)$$

where ϕ are the parameters of the neural network model.

1) Data Preprocessing: It is difficult for a neural network to learn from different dimensional inputs. Though we can use different embeddings and concatenate at deeper layers, our experiments revealed that this is not very efficient. So, as a pre-processing step, we convert the scan input z_t into a scan image S_h (h denotes high resolution) of the same size as the grid map. We concatenate both the 2D scan image and the map of the environment and use it as input to our neural network. During the training phase, we obtain the ground truth likelihood $\tilde{p}^{(0)}(z_t|x)$ by taking the cosine similarity or correlation of the current scan with the scans at all other possible positions.

2) Ground Truth Data Generation: We generate and store triplets of (scan-image S_h, map of the environment M_h, ground truth likelihood $\tilde{p}^{(0)}(z_t|x)$) while randomly moving the robot in the simulator and randomly resetting the initial pose after every T time steps and randomly resetting the environment after every e such episodes. We train it on m such environments (maps). Training on these triplets using Resnet-152 [28] or Densenet-121 [29] gave very good results in simulation but did not transfer when these models are transferred to real robot in a real environment.

B. Bayes' Filter

We can now update the belief by taking the element-wise product of the likelihood and the previous belief over all the $N \times M \times O$ grid cells similar to [14].

$$bel^{(0)} = \bar{bel}^{(0)} \odot p^{(0)}(z_t|x)$$

C. Hierarchical Likelihood Model

We estimate the robot pose on a coarse grid, which is sufficient for macro-action selection in RL, but we require a refined estimate for two reasons:

1) It's possible that the coarse estimate is simply not precise enough, particularly in very large maps
2) Without a more precise estimate of the robot's pose within a coarse grid cell, the robot will gradually drift from the grid centroids as a result of dead reckoning when deployed in a real environment.

In hierarchical likelihood estimation (HLE), the likelihood is first estimated at coarse resolution ($N^{(0)} \times M^{(0)} \times O$) and then each of these grid cells is potentially expanded to further finer grid cells ($k \times k$) to get a full resolution map of size ($N^{(1)} \times M^{(1)} \times \Theta$).

For the case of 2-level hierarchy, we have 2 neural networks $f^{(0)}(M_h, S_h)$ and $f^{(1)}(M_c, S_c)$ at levels 0 and 1 respectively. The input for $f^{(0)}$ is the high resolution map M_h (which is a processed version of map $\mathcal{M}$ of the environment)

$(N^0 \times M^0)$, and the scan image S_h which is obtained from the current LiDAR scan z_t $(N^{(0)} \times M^{(0)})$ at the current pose. The likelihood output $p^{(0)}(z_t|x) = f^{(0)}(M_h, S_h)$ is of shape $N^{(0)} \times M^{(0)} \times \Theta^{(0)}$. The parameters of $f^{(0)}$ are optimized by minimizing the mean squared error (MSE) between $p^{(0)}(z_t|x)$ and ground truth likelihood $\tilde{p}^{(0)}(z_t|x)$ (which is at the same resolution of $N^{(0)} \times M^{(0)} \times \Theta^{(0)}$).

Grid cells corresponding to a maximum of c values from the coarse likelihood $p^{(0)}(z_t|x)$ are selected. For each of these grid cells, we crop a square patch from the high resolution map and high resolution scan (optional) around the grid cell. Concatenation of the cropped map and cropped scan is used as input for $f^{(1)}$. The network outputs a $k \times k$ block where each cell in the block represents likelihood at finer resolution. Each of the cells in this block is multiplied with the corresponding likelihood of the grid cell at the previous level $p^{(0)}(z_t|x)$. For the cells which are not in the top c values, the likelihood value at level-0 $p^{(0)}(z_t|x)$ is normalized and directly copied to each cell in the corresponding block in $p^{(1)}(z_t|x)$. The parameters of $f^{(1)}$ are optimized by minimizing the MSE between $p^{(1)}(z_t|x)$ and ground truth likelihood $\tilde{p}^{(1)}(z_t|x)$ (which is at the same resolution of $N^{(1)} \times M^{(1)} \times \Theta^{(1)}$).

D. Domain Randomization

Domain randomization has become a popular method to promote generalization, particularly when training an agent in a (necessarily imperfect) simulator, and then deploying it on real hardware [18]. While the LiDAR data modality generally transfers better that vision (camera images) from simulation environments to real ones, it is not without challenges since no sensor model is perfect. Just training the likelihood models without any variability in the simulator results in over-fitting and poor transfer. Since we do not have an exact recreation of the real world environment within our simulator, our learned agent will have to generalize to a new environment configuration while simultaneously bridging the reality gap. Hence, it is important to account for various real world irregularities while training on the simulator. In our data collection pipeline, we randomized the following parameters:

- thickness and length of obstacles to account for different types of obstacles in real environment;
- error in robot pose to account for the possibility that it is not exactly at the centroid of a given cell;
- temperature β for softmax function σ as defined in

$$\sigma(p(x);\beta)_{i,j,k} = \frac{e^{p(x)_{i,j,k}/\beta}}{\sum_{n,m,\theta} e^{p(x)_{n,m,\theta}/\beta}} \tag{3}$$

  which is used to bound the ground truth likelihood and improve learning;
- the LiDAR scan data by adding Gaussian noise to each return and randomly setting some of the incoming data points within the scan to have the value of $+\infty$;
- the map itself through erosion and dilation operators, which was used for training while the simulated LiDAR readings still came from unperturbed maps.

V. Planning and Control

The multi-scale localization estimates are used for planning and control.

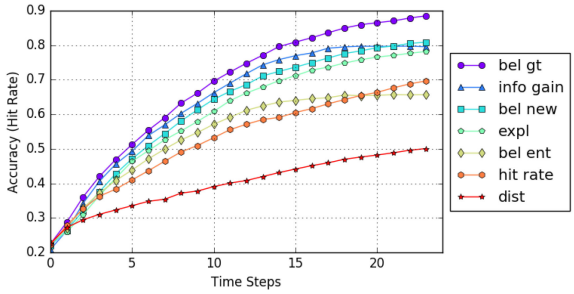


Fig. 3. We trained 7 RL policy models with the following different rewards: 1) probability mass of belief at true pose (bel gt), 2) decrease in the entropy of the belief (info gain), 3) reward of +1 for a new pose in belief (bel new), 4) reward of +1 for a new true pose (expl), 5) negative entropy of belief (bel ent), 6) reward of +1 if Manhattan distance error is equal to 0 (hit rate), 7) Manhattan distance error (dist).

A. Reinforcement Learning

Following the traditional conventions in a Markov Decision Process (MDP) [30], we denote the state $s \in \mathbb{R}^d$, the actions $a \in \mathbb{R}^{d_a}$, the reward function $r(s,a)$ and a transition model $T = p(s'|s,a)$. In our case, the state (input to the RL model) is a concatenation of belief map $bel(x_t)$ $(N^{(0)} \times M^{(0)} \times \Theta^{(0)})$ and the input map of the environment $\mathcal{M}$. We formulate the MDP over high-level actions (`left`, `right`, and `forward`) The goal of any RL algorithm is to optimize its policy $\pi(a|s)$ to maximize the return defined as: $G_t = R_t + \gamma R_{t+1} + ...$ over an initial distribution of states (which is assumed uniform here). We use the advantage actor critic (A2C) [31] algorithm to accomplish this task. The network consists of two convolutional layers and an LSTM unit followed by 2 fully connected heads for actor and critic. There are various choices for a reward function. We used the belief at the true pose as our reward function since it is dense and benefits from the availability of true pose in the simulator. The empirical evidence for the choice is given in Fig. 3.

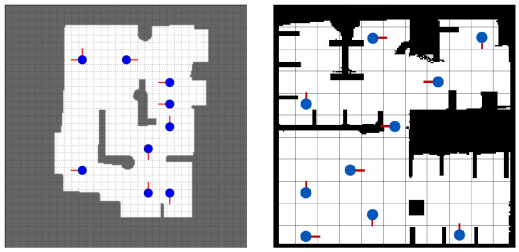
B. Closed-Loop Control

After the high level actions (`left`, `right`, `forward`) are chosen by the RL model, it is important to ensure that the robot reaches its goal position without deviating from the path. Even a minor deviation in each time step results in compounding errors. So, the low level actions (linear and angular velocities of the mobile robot) are given based on the current pose and the goal pose. The current pose is obtained at much finer resolution using our hierarchical likelihood model. See the Section IV-C for more details of hierarchical models. Optionally, the hierarchical model can be used after every fixed number of time steps to correct for the accumulated drift.

VI. Experiments

We demonstrate the efficacy of DAL by conducting several experiments in simulation and on real robots in varying environments. This section describes the experimental setup and presents the results obtained, which demonstrate that DAL outperforms state-of-the-art active localization approaches in terms of localization performance and robustness.

The real environments used for testing were simple and reconfigurable. Using a SLAM software (gmapping [32]) we obtained maps as in Fig. 4.



(a) Sait map (b) Mila map

Fig. 4. The maps of the real environments in SAIT and Mila were generated by using a SLAM algorithm, *gmapping* [32]. The map size was adjusted to 224 × 224 pixels each of size 0.04 m × 0.04 m. We randomly selected 10 poses on the map and used them repeatedly as the starting poses for the 10 episodes of each test.

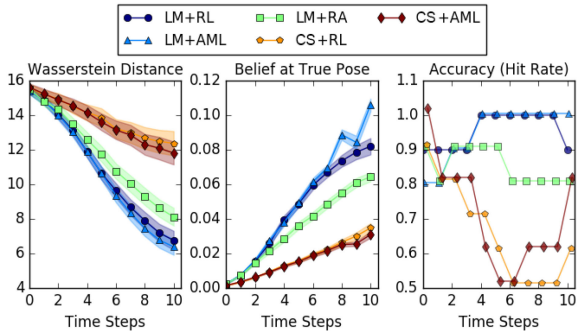


Fig. 5. Localization performance was evaluated in 8 headings and 33 × 33 grids with JAY in the real environment. We compared our proposed likelihood model for 33 × 33 (LM), cosine similarity (CS), active Markov localization (AML), random action (RA), and the trained policy π with reinforcement learning (RL). Offsets are added to some of the accuracy values in the plot to better identify the overlapping markers.

A. Dataset Generation

We generated a dataset to train the likelihood model to estimate the measurement likelihood $p(z|x)$. To do so, we generated random maps, sensor data at some random poses as well as the target distributions at those poses. The procedure had the following steps:

1) We use Kruskal's algorithm to generate random maze-like environments at low resolution. The output is a square matrix with 1s representing obstacles and 0 s representing free cells. The algorithm guarantees that all the open spaces are connected.
2) Based on the grid map at low resolution generated above, we place obstacles (of different sizes) at the cells in the high resolution. The algorithm then randomly dilates and erodes the surfaces of obstacles in the map so that its texture becomes more realistic.
3) We place the robot at all the grid centers of the low resolution map at all $\Theta^{(0)}$ directions. From each of those locations, we simulate a laser scan. As a result, we obtain $M^{(0)} \times N^{(0)} \times \Theta^{(0)}$ laser scans. We call this the scan matrix, which has dimension $M^{(0)} \times N^{(0)} \times \Theta^{(0)} \times 360$. Given a map, we precompute a range vector at each location with respect to the low-resolution grids. A range vector in our experiment has the length of 360, representing distances at each of 360 degrees.
4) We now spawn the robot at 10 random locations and compute the cosine similarity of the scan at the current pose with the scan at all possible poses to get a likelihood map. With each of those likelihoods, we form a triplet of map, current scan, likelihood and use these as the training set.

The procedure was repeated for 10000 different maps.

B. Training the Likelihood and Policy Models

We first describe how we trained our likelihood model (LM) which was used in our real robot experiments. We generated a dataset containing 10,000 maps with 10 scan inputs for each map by following the procedure explained above. The dimension of ground truth likelihood (GTL) was set to $33 \times 33 \times 8$, i.e., 8 headings with 33*33 x and y locations.

The likelihood model was trained with a densenet201 model on this dataset. We applied the following randomization for training LM:

- the temperature for softmax function was manually decreased over epochs from 1.0 to 0.1,
- the input scan was randomly rotated in the range of $\pm 2^\circ$,
- 100 randomly selected pixels were flipped ($0 \leftrightarrow 1$),
- and drop-out was applied in densenet with the rate of 0.1.

The likelihood model was trained first with 100,000 data instances over 16 epochs and adapted to the map of the real environment in a simulator for 10,000 inputs.

The policy model π was trained on a simulator with 2,000 randomly generated maps, with 20 episodes (each of length 24) for each map. Reward (equal to probability mass of belief at true pose) was given at each time step.

C. Real Robot Setup

To demonstrate the feasibility and robustness of our approach, we tested our trained likelihood and policy models on two mobile robots: JAY and Turtlebot, and in two different environments.

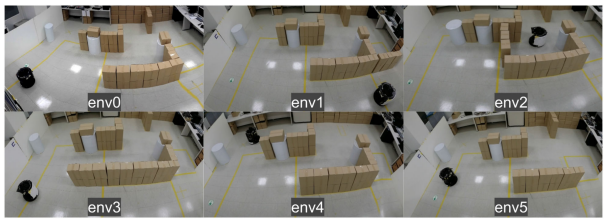
1) Experiments on JAY: JAY (Rainbow Robotics) is cylinder-shaped with the radius of 0.30 m, the height of 0.50 m and the weight of 46 kg before modification, whose CPU is IntelCo re i7-8809G Processor. We mounted an extra LiDAR A2M8 (Slamtec) on the top of the robot in the center to secure 360-degree view. The LiDAR provides 0 to 360 degree angular range, and 0.15 to 8.0 m distance range, with 0.9 degree angular resolution, at the rate of 10 Hz.

The ROS navigation package including AMCL was manually initialized and used for ground truth localization. DAL was running on a separate server equipped with 4 Nvidia GTX-1080Ti GPUs and an Intel CPU E5-2630 v4 (2.1 GHz, 10 cores) to send velocity commands to JAY over a 2.4 GHz WIFI channel.

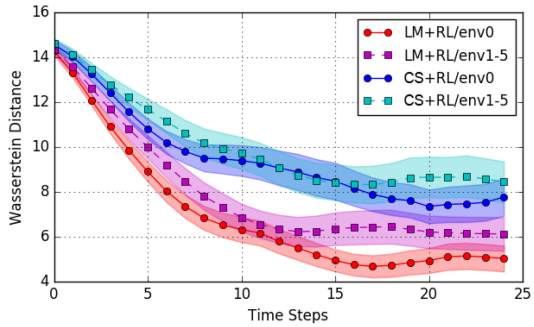
With JAY deployed in the indoor environment shown as 'env0' in Fig. 6(a), we tested our likelihood model. To evaluate the performance of our method, we compared both the localization model and the RL-based planning model with standard approaches.

We compare two different measurement likelihood techniques:

- Trained likelihood model (LM): use the trained model for the sensor measurement likelihood at each pose,
- Cosine similarity (CS): use cosine similarity between the precomputed scan matrix and the input scan for the sensor measurement likelihood,



(a) The original and the five modified environments



(b) Wasserstein distance between belief and true belief

Fig. 6. LM is tested in modified environments env1 to env5 and compared with the error from the original environment env0. Error is measured in terms of Wasserstein distance between belief and ground truth.

in combination with three different planning approaches:

- Reinforcement Learning (RL): use the trained policy model π for sampling next action.
- Active Markov Localization (AML): the robot virtually goes one step ahead along each of the possible actions and select the one with the largest decrease in pose belief entropy.
- Random Action (RA): sample next action randomly with uniform probability.

The work that is most related to our own is ANL [14]. ANL was developed for visual inputs where the perceptual model generates feature vectors which are compared via cosine similarity with reference feature vectors generated from the map to build the measurement likelihood. This is achieved in simulation through synthesizing views of the environment to compare against. These images are included as part of the "map design". It is much more laborious to manually generate these views in a real environment, it would require collecting an image from every sample grid location in the map and storing them in a database. Therefore, direct application of this method to a real robot setup is not possible.

We argue that the fairest comparison with ANL is to consider the LiDAR returns as the output of the perceptual model and to use cosine similarity to generate the measurement likelihood. So in this context, we consider the CS + RL case to be the closest approximation of the ANL method and is used for comparison.

Each test condition ran for 10 episodes of length 11 steps. Fig. 4(a) shows the 10 initial poses that were randomly sampled to be applied to all the tests.

2) Results From JAY Experiment: We evaluated the test results using the following metrics:

i) Earth Mover's or Wasserstein distance W, which quantifies the error between our belief map and true belief map (which has probability 1.0 at the true pose and 0 everywhere else)

$$W = \sum_{x} p(x) D(x, x^*), \tag{4}$$

where D is the Manhattan distance between two poses.

ii) Estimated belief at true pose $p(x^*)$ (this is also the reward that was used for training)

iii) "Hit rate", which is defined as the number of times the maximum likelihood estimate of the belief is the same as the true pose.

Each test consists of 10 episodes. The mean and standard deviation over the 10 episodes were then computed for each of the 11 steps as plotted in 5 to illustrate how the metrics changed during an episode in average.

From the results plotted in Fig. 5, we can see that our trained likelihood model performed better than cosine similarity in all of the metrics. This could be attributed to the susceptibility of CS to noise such as an inaccurate map and the robot deviating from center of a grid cell. Since we accounted for all these sources of error during training, our LM proved to be robust. Our RL model performed as well as AML. In terms of time complexity, our RL model is however multiple times faster than AML as we show in the time complexity analysis (cf. Table I). Even though hit rate is initially high and appears to have solved the localization problem, the belief at ground truth pose is much less (on the order 1e-3) as seen in the corresponding wasserstein distance and belief at true pose plots.

To further justify our robustness claims of our trained LM, we modified the environment by adding or moving some obstacles and ran the same experiment again. We tested it for LM + RL versus CS + RL on five differently modified environments from 'env1' to 'env5' shown in Fig. 6(a). We also tested again at the original environment 'env0'. Each test was done with the same 10 initial poses as above while the length of an episode was increased to 25. The number of headings was fixed at four.

3) Experiments on Turtlebot: We used a Hukoyo Laser UST-20LX which outputs 1040 measurement ranges with detection angle of 260 and angular resolution of 0.25. The netbook runs on Intel Core i3-4010U processor with 4 GB RAM. Our test environment is of size 9×9 meters^2 and is combination of two rooms as shown in 4(b). Similar results as that of JAY were observed (omitted for brevity).

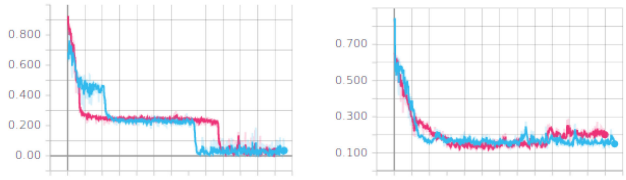
We have released our OpenAI gym compatible code in the hope that this will act as a testbed for active localization research. Once trained on our simulator, the policies and the likelihood model learned are directly transferable to a real robot. We have also integrated the environment with various state of the art RL algorithms [33] like PPO [34], A2C [31], ACKTR [35].

TABLE I
TIME COMPLEXITY ANALYSIS (IN SECONDS)

	CS	LM	RL	AML(+LM)
4x11x11	0.202	0.0531	0.00122	1.23
4x33x33	0.316	0.0518	0.00172	1.29
8x33x33	0.492	0.0866	0.00203	1.38
24x33x33	0.770	0.123	0.00194	1.47

D. Hierarchical Likelihood Models

We present the experimental results of our 2 novel hierarchical likelihood models. From Fig. 7(a), we can observe that both weighted, where we multiply the likelihood of every cell at higher level with the corresponding likelihood at



(a) loss at level-0 (b) loss at level-1

Fig. 7. Blue: weighted HLE. red: unweighted HLE.

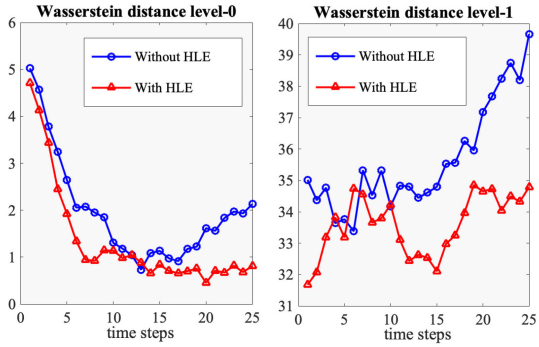


Fig. 8. Wasserstein distance between DAL's belief and ground truth belief at level-0 and level-1 when HLE (hierarchical likelihood estimate) is used (red) and when it is not used (blue). We can see that HLE (based on cosine similarity in this figure) indeed helps in controlling the motion errors.

lower level, and un-weighted variants of hierarchical likelihood model performed equally well and obtained convergence. In these experiments, we used $N^{(0)} = 11$, $M^{(0)} = 11$, $N^{(1)} = 88$, $M^{(1)} = 88$, $\Theta^{(0)} = 4$, $\Theta^{(1)} = 4$, $k = 8$. However, higher levels of hierarchy are very sensitive to hyper-parameters because they are dependent on the performance of previous layers. The robustification studies of the hierarchical model, a thorough empirical evaluation and making them insensible to hyper parameters are possible avenues for future research.

VII. Conclusion

We have presented a learning-based approach to active localization from a known map. We propose multi-scale learned perceptual models that are connected with an RL planner and inner loop controller in an end-to-end fashion. We have demonstrated the effectiveness of the approach on real robot settings in two completely different setups.

In future work, we would like to explore the removal of the reliance of the system on a high fidelity map such as is generated by *gmapping*. Much more appealing would be, for example, if the robot could localize on a hand drawn sketch or some much more easily obtained representation.

References

[1] W. Burgard, D. Fox, and S. Thrun, "Markov localization for mobile robots in dynamic environments," *J. Artif. Intell. Res.*, vol. 11, pp. 391–427, 1999.
[2] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Monte carlo localization for mobile robots," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1999, pp. 1322–1328.
[3] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.
[4] E. B. Olson, "Real-time correlative scan matching," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009.
[5] W. Burgard, D. Fox, and S. Thrun, "Active mobile robot localization," in *Proc. Int. Joint Conf. Artif. Intell.*, 1997.
[6] S. Thrun, "Finding landmarks for mobile robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1998.
[7] T. Arbel and F. P. Ferrie, "Viewpoint selection by navigation through entropy maps," in *Proc. Int. Conf. Comput. Vis.*, 1999.
[8] D. Fox, W. Burgard, and S. Thrun, "Active Markov localization for mobile robots," *Robot. Auton. Syst.*, vol. 25, pp. 195–207, 1998.
[9] R. Kmmerle, R. Triebel, P. Pfaff, and W. Burgard, "Monte carlo localization in outdoor terrains using multi-level surface maps," *J. Field Robot.*, vol. 25, pp. 346–359, 2008.
[10] A. Kendall, M. Grimes, and R. Cipolla, "Convolutional networks for real-time 6-dof camera relocalization," in *Proc. Int. Conf. Comput. Vis.*, 2015.
[11] N. R. Abhinav Valada and W. Burgard, "Deep auxiliary learning for visual localization and odometry," in *Proc. IEEE Int. Conf. Robot. Autom.*, May 2018.
[12] R. Jonschkowski, D. Rastogi, and O. Brock, "Differentiable particle filters: End-to-end learning with algorithmic priors," in *Proc. Robot., Sci. Syst.*, Pittsburgh, PA, USA, 2018.
[13] P. Karkus, D. Hsu, and W. S. Lee, "Particle filter networks: End-to-end probabilistic localization from visual observations," *IEEE Int. Conf. Robot. Autom.*, 2018.
[14] D. S. Chaplot, E. Parisotto, and R. Salakhutdinov, "Active neural localization," in *Proc. Int. Conf. Learn. Representations*, 2018.
[15] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 1334–1373, 2016.
[16] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2017.
[17] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. Cambridge, MA, USA: MIT Press, 2005.
[18] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. Int. Conf. Intell. Robots Syst.*, 2017.
[19] S. Lowry *et al.*, "Visual place recognition: A survey," *IEEE Trans. Robot.*, vol. 32, no. 1, pp. 1–19, Feb. 2016.
[20] S. Oore, G. E. Hinton, and G. Dudek, "A mobile robot that learns its place," *Neural Comput.*, vol. 9, pp. 683–699, 1997.
[21] D. Rosenbaum, F. Besse, F. Viola, D. J. Rezende, and S. M. A. Eslami, "Learning models for visual 3D localization with implicit mapping," to be published.
[22] H. J. S. Feder, J. J. Leonard, and C. M. Smith, "Adaptive mobile robot navigation and mapping," *Int. J. Robot. Res.*, vol. 18, 1999.
[23] N. Roy, W. Burgard, D. Fox, and S. Thrun, "Coastal navigation-mobile robot navigation with uncertainty in dynamic environments," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1999.
[24] G. L. Mariottini and S. I. Roumeliotis, "Active vision-based robot localization and navigation in a visual memory," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011.
[25] R. Valencia, J. V. Mir, G. Dissanayake, and J. Andrade-Cetto, "Active pose slam," in *Proc. Int. Conf. Intell. Robots Syst.*, 2012.
[26] C. Forster, M. Pizzoli, and D. Scaramuzza, "Appearance-based active, monocular, dense reconstruction for micro aerial vehicle," *Robotics: Science and Systems*, X. Berkeley, CA, USA: Univ. California, 2014.
[27] A. Kim and R. M. Eustice, "Active visual slam for robotic area coverage: Theory and experiment," *Int. J. Robot. Res.*, vol. 34, 2015.
[28] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016.
[29] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017.
[30] R. S. Sutton, "On the significance of Markov decision processes," in *Proc. Int. Conf. Artif. Neural Netw.*, 1997.
[31] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2016.
[32] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, Feb. 2007.
[33] I. Kostrikov, "Pytorch implementations of reinforcement learning algorithms," GitHub repository, 2018.
[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[35] Y. Wu, E. Mansimov, S. Liao, R. B. Grosse, and J. Ba, "Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation," in *Proc. 31st Conf. Neural Inf. Process. Syst.*, 2017.