Université de Montréal
Faculté des arts et des sciences

Ce mémoire intitulé:

Deep Active Localization

présenté par:

Sai Krishna G.V.

a été évalué par un jury composé des personnes suivantes:

Ioannis Mitliagkas,	président-rapporteur
Liam Paull,	directeur de recherche
Stefan Monnier,	membre du jury

Mémoire accepté le: .....

Sommaire

Des progrès considérables ont été réalisés en robotique mobile au cours des dernières décennies et ces robots sont maintenant capables d'effectuer des tâches qu'on croyait auparavant impossibles. Un facteur critique qui a permis aux robots d'accomplir ces diverses tâches difficiles est leur capacité à déterminer où ils se trouvent dans un environnement donné (localisation). On parvient à une automatisation plus poussée en laissant le robot choisir ses propres actions au lieu de faire appel à un téléopérateur humain. Cependant, la détermination précise de la pose (position + orientation) du robot et l'adaptation de cette capacité à des environnements plus vastes constituent depuis longtemps un défi dans le domaine de la robotique mobile. Les approches traditionnelles à cette tâche de " localisation active " utilisent un critère théorique de l'information pour la sélection des actions ainsi que des modèles perceptuels faits à la main.

Avec une augmentation constante des capacités de calcul disponibles au cours des trois dernières décennies, l'algorithme *back-propagation* a trouvé son utilisation dans des réseaux neuronaux beaucoup plus profonds et dans de nombreuses applications. En l'absence de données labellisées, le paradigme de l'apprentissage par le renforcement (RL) a récemment connu beaucoup de succès en ce qu'il apprend en interagissant avec l'environnement. Cependant, il n'est pas pratique pour un algorithme RL d'apprendre raisonnablement bien à partir de l'expérience limitée du monde réel. C'est pourquoi il est courant d'entraîner l'agent dans un simulateur puis de transférer efficacement l'apprentissage dans de vrais robots.

Dans cette thèse, nous proposons une méthode différentiable de bout en bout afin d'apprendre à choisir des mesures informatives pour la localisation de robots, qui peut être entraînée entièrement en simulation et ensuite transférée sur le robot réel sans aucun ajustement. Pour ce faire, on s'appuie sur les progrès récents de l'apprentissage profond et des paradigmes d'apprentissage de renforcement, combinés aux techniques de randomisation des

domaine. Le système est composé de deux modules d'apprentissage : un réseau neuronal convolutionnel pour la perception, et un module de planification utilisant l'apprentissage profond par renforcement. Nous utilisons une approche multi-échelles dans le modèle perceptuel puisque la sélection d'action à l'aide de l'apprentissage par renforcement nécessite une précision de la position inférieure à la précision nécessaire au contrôle du robot. Nous démontrons que le système résultant surpasse les approches traditionnelles, en termes de perception et de planification. Nous démontrons également la robustesse de notre approche vis-à-vis différentes configurations de cartes et d'autres facteurs de nuisance par l'utilisation de la randomisation de domaine au cours de l'entraînement. Le code a été publié : https://github.com/montrealrobotics/dal et est compatible avec le framework OpenAI gym, ainsi qu'avec le simulateur Gazebo.

Mots clés : Localisation, Apprentissage profond, Apprentissage par renforcement, Localisation active, Robotique

Summary

Mobile robots have made significant advances in recent decades and are now able to perform tasks that were once thought to be impossible. One critical factor that has enabled robots to perform these various challenging tasks is their ability to determine where they are located in a given environment (localization). Further automation is achieved by letting the robot choose its own actions instead of a human teleoperating it. However, determining its pose (position + orientation) precisely and scaling this capability to larger environments has been a long-standing challenge in the field of mobile robotics. Traditional approaches to this task of *active localization* use an information-theoretic criterion for action selection and hand-crafted perceptual models.

With a steady rise in available computation in the last three decades, the *back-propagation* algorithm found its use in much deeper neural networks and in numerous applications. When labelled data is not available, the paradigm of reinforcement learning (RL) is used, where it learns by interacting with the environment. However, it is impractical for most RL algorithms to learn reasonably well from just the limited real world experience. Hence, it is common practice to train the RL based models in a simulator and efficiently transfer (without any significant loss of performance) these trained models into real robots.

In this thesis, we propose an end-to-end differentiable method for learning to take informative actions for robot localization that is trainable entirely in simulation and then transferable onto real robot hardware with zero refinement. This is achieved by leveraging recent advancements in deep learning and reinforcement learning combined with domain randomization techniques. 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 approaches 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.

Key words: Localization, Deep Learning, Reinforcement Learning, Active Localization, Robotics

Table des matières

Liste des tableaux

Liste des figures

Remerciements

I would first like to thank my supervisor Dr. Liam Paull for his constant supervision since the time I joined in his group. It was through the weekly meetings with him that the project has developed to this point in the last 22 months and his support for turning me into a better writer and researcher. He also thoroughly guided me in learning the classical localization and SLAM formulations which played a crucial role in this project. He, along with his Ph.D students Manfred Diaz and Vincent Mai helped me get a head start in the experiments with turtlebot and Gazebo.

I express my deepest gratitude to Dr. Keehong Seo who played a key role by contributing to the results on JAY, development of a custom simulator, code development, timely documentation and most of the brainstorming sessions.

I thank all members of the DAL-team (Keehong Seo, Dhaivat Bhatt, Vincent Mai, Krishna Murthy, Liam Paull) who are involved in our "Deep Active Localization" project (which was accepted to RA-L journal), which is the major focus of this thesis. Dhaivat and Vincent contributed to the experiments on turtlebot and in polishing the manuscript. Krishna and Liam contributed to the ideation phase, brainstorming sessions and manuscript. I also thank Bhairav Mehta, Maxime CB, Florian Golemo for their help in training the policy model.

I thank all my friends at Mila for discussions on various related ideas, reading group sessions, solving my pytorch issues within minutes etc.. Special thanks to Dr.Yoshua Bengio and his team for creating this highly collaborative and productive environment at Mila. I also thank DIRO and UdeM for providing a cool research environment and the physical space for robot experiments.

I thank my parents for their various sacrifices and making me pursue my passions, whatever those passions are.

Chapitre 1

Introduction

Mobile robots have the capability to advance humanity by automating various tasks, including some tasks that humans cannot do. Examples of mobile robots range from a simple automated floor cleaning robot to a Mars rover, from a remote controlled toy car to a fully autonomous vehicle. At the core of this entire field of mobile robotics is the task of localization [**73**]: the robot should know its current location (relative to the environment it is operating upon) to be able to perform any downstream task. For example, for an autonomous car to navigate to its destination, it should precisely know its location to further plan the path to its destination. This localization task is performed by the robot by collecting sensor data and comparing it against its model of the environment (usually called a map). A robot's ability to localize is therefore dependent on *where* in the environment that sensor data is taken, since some parts might be more unique and easy to disambiguate. Active localization [**55**], which is the focus of this thesis, deals with choosing actions (where should the robot move next?) to be able to collect the data to allow it to accurately determine its current location.

The traditional methods for a robot to determine its location involve probabilistically fusing sequences of sensor data over time [**15**]. This is typically done in a recursive filtering framework which requires a motion model (model of the movement of the robot in space given actuation commands) and a measurement model (model of how the sensor data obtained relates to the robot state being estimated). This framework of fusing motion and measurement models in an iterative fashion is called Markov Localization [**15**]

Traditional methods for localization are "*passive*" (agnostic to how actions are selected). They provide a recursive framework for updating an approximation of the posterior of the

state belief as new measurements arrive. In the case of traditional Markov Localization (ML), this is usually done by some type of discretization of the state space, such as a fixed grid [**16**], or by maintaining a set of *particles* (state hypotheses) as is done in Adaptive Monte Carlo Localization (AMCL) [**74**]. 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 [**11**] with the laser scanner since the measurement likelihood can be computed efficiently and in closed form with scan matching [**54**]. 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.

Machine Learning (and Deep Learning (DL) in particular) approaches are able to *learn from available data*, and consequently often have better generalization capabilities [**52**] as they do not over-fit to any set of rigid rules. However, the goal of this work is more ambitious than just learning from data: we do not want to perform any training on the real robots (because it is expensive and very time consuming). Domain randomization (DR) [**75**] deals with adding various noises to a simulator to help a learned model (deep neural network) generalize to real environments when just trained on a simulator. This ability to transfer models from a simulator to real environments is one of the core components of transfer learning [**56**]. It is referred to as zero-shot transfer when no further training is done in real environments. In this thesis, we leverage recent advances in DL combined with DR techniques (sec-4.2.4) to mitigate the issues of rigidity inherent in the classical methods (sec-3.3). We demonstrate the effectiveness (in terms of localization performance and robustness: chapter-5) of our method with thorough experimental results on a simulator and on real robots.

While these techniques help us in solving the *passive localization* problem (we are not selecting the actions that are taken by the robot), our overall goal in this thesis is to solve the problem of *active localization* (where we are selecting the actions). Active variants of ML and AMCL are classical ways to solve the active localization problem [**7, 72, 3, 14, 39**]. These methods typically leverage an information-theoretic [**67**] measure to evaluate the benefit of visiting unseen poses. We describe each of these methods in detail in sec-2.3. Again,

these hand-tuned heuristics tend to lead to over-fitting of a particular algorithm to a specific robot-sensor-environment setup.

Labelled data (datasets containing best actions for a given state) are not available in this case which prevents us from using DL to overcome the problems associated with these hand tuned heuristics. However, the paradigm of reinforcement learning (RL) enables us to *learn* actions by interacting with the environment (either in simulator or in real world) and by taking rewards from the environment after every action. Deep Reinforcement Learning (DRL) (which uses deeper neural networks in RL paradigm, and hence it is sometimes considered a particular form of DL) has seen an incredible recent success on some robotics tasks such as manipulation [**42**] and navigation [**86**], 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 needs to be small (simulator should be very realistic, i.e, models trained on a simulator should directly work in real environments) 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 reduces the burden on completeness and thoroughness of the simulator 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 true localization) that are only available in the simulator. In this work, we apply the framework of DRL to the task of active localization and use it to train policies that transfer to a robot in real-world indoor environments.

To summarize, in this work we leverage deep supervised learning and deep reinforcement learning to build an end-to-end trainable system for active localization. A likelihood model is pre-trained via supervised learning to tell us where the robot is likely to be present in the environment given current observation and map of the environment. We use this likelihood to update the robot's belief about its current location in the environment. Furthermore, we use this belief, current observation and map of the environment to train the robot (via DRL) to take actions that help in minimizing the localization error. All these components are differentiable, thus making both the likelihood and policy models trainable together. To control the motion errors, we train a hierarchical network for likelihood prediction which

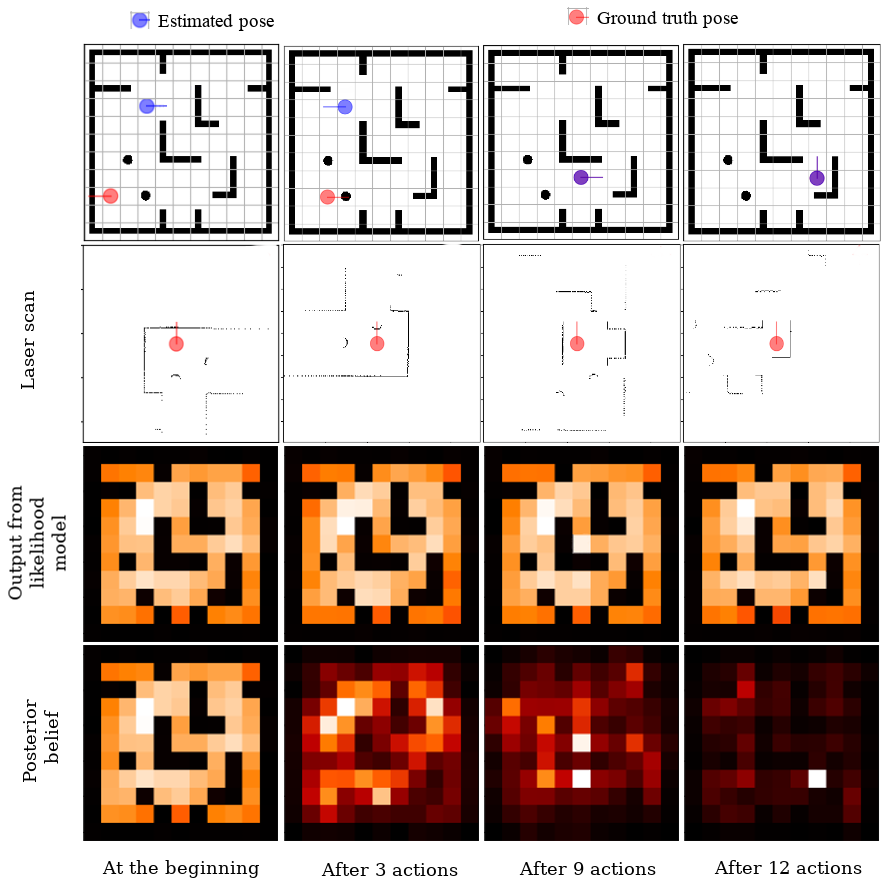


Fig. 1.1. Deep active localization (DAL) demonstrated in simulation. The first row gives a birds eye view of the environment along with the robot's true location and its believed location. The red dot indicates the true pose of the robot and the blue dot indicates the pose of where the robot believes it is. The second row shows the observation (scan image, obtained from laser scan) from the robot's true pose. The third row shows the output of the likelihood model, which tells us where the robot is likely to be present given the current observation and map of the environment. The fourth row shows the robot's posterior belief i.e, where the robot believes it is. The brighter white color denotes higher probability and darker red color denote lesser probability of robot being in that location. 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.

gives us the likely poses of the robot at a much finer resolution thus enabling us to correct for the robot's inevitable drift as it traverses the path towards its goal in the next time step. While more thorough notations and details are given in chapter-2, figure-1.1 provides a rough overview of how our method helps the robot in active localization and increases the robot's confidence over its location.

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 integrate this model into an end-to-end deep RL system that does 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 develop an automated processes for map generation, domain randomization [**76**] 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.

We demonstrate the effectiveness of our methods via various sets of experiments in chapter-5. After detailing the dataset generation and experimental setup, we compare our algorithm with traditional approaches (cosine similarity, Markov localization) and other learning based approaches like ANL[**9**]. We also provide experimental results on robustness of our approach (fig-5.5) and effectiveness of hierarchical models (fig-5.7). We also provide a timing comparison of our approach with other approaches (table-5.1).

1.1. Prologue

This thesis is based on my paper "Deep Active Localization" [**19**] which was accepted to the RA-L journal. Here, I summarize the contributions of every author in this work: - I developed the neural network models, simulator on gazebo, custom gym environment, interface with all the reinforcement learning algorithms we used, code for experiments on turtlebot. I contributed to the automated dataset generation, ideation phase, brainstorming sessions, manuscript and experiments on turtlebot.

- Dr. Keehong Seo played a key role by contributing to the results on JAY, development of a custom simulator, code development on gazebo simulator and for real robot experiments, densenet and resnet models, automated dataset generation, timely documentation and most of the brainstorming sessions.

- Dhaivat and Vincent contributed to the experiments on turtlebot and in polishing the manuscript.

- Krishna and Liam contributed to the ideation phase, brainstorming sessions and manuscript.

Chapitre 2

Robot Localization

In this chapter, we define the notations used and classical formulations of the robot localization and active localization problems.

2.1. Localization

The problem of localization corresponds to determining the pose of a mobile robot in an environment given the map of the environment and observations. In 2D, the pose (state), x is defined as the position (**x-y** coordinates) and orientation (θ) of the robot w.r.t to a fixed frame. The map $\mathcal{M}$ represents the position of obstacles or landmarks in the environment the robot is operating in. The observation z is the sensor measurement (for example: scan measurements from a Light Detection And Ranging "LiDAR" (LiDAR) sensor) taken from its current pose x. There are various considerations in localization problems:

- **Local vs Global vs Kidnapped:** If we know the initial pose of robot, it is classified as a local localization problem. If we do not make any assumptions about its initial pose, it is classified as global localization. Kidnapped localization is when a robot is removed from its current pose and dropped somewhere else in the environment during operation. This is a very hard problem to solve [**73**].
- **Passive vs Active:** In passive localization, the robot is tele-operated whereas in active localization, the robot has to choose its own actions to minimize the localization error.
- **Static vs Dynamic Environments:** Depending on whether the obstacles in the environment are fixed or moving, the localization problem can be classified into static obstacle localization or dynamic obstacle localization respectively.

- **Single Robot vs Multi-Robot:** In a single-robot case, the robot updates the belief of its pose based on its own observations. When there are multiple robots in the environment, each robot can also benefit from the information shared/communicated by other robots.

In this thesis, we will focus on active global localization for single robots in static environment although our algorithm is extendable to other paradigms like local or passive localization or multi-robot setting. So, we assume that the obstacles in our environment are static and we do not assume any knowledge of our robot's initial pose. Thus, the goal is to choose actions that minimize the error in localization. Even if the overall task is different (for example, to navigate to a particular location in the environment), the robot is first tasked with active localization to be able to efficiently solve any overall task [**6**]. Sometimes it is possible to simultaneously solve multiple tasks like active localization and mapping [**8**]. Markov localization is a well known method used for solving passive localization, which can also be used in active localization framework.

2.2. Markov Localization

Markov localization is a direct implementation of the recursive Bayes filter. The Bayes filter is used for updating the pose of the robot based on new measurements. We first introduce the notations used and then discuss the framework of recursive Bayes filter being used for Markov Localization.

A state is any information about the robot and its environment. It is denoted by x. The state at time t is denoted as x_t. In the context of localization, the information about the map $\mathcal{M}$ of the environment is fixed and known prior to us. Henceforth, the state x_t refers to the information about pose ($\mathbf{x}$, $\mathbf{y}$, θ) of the robot w.r.t given map of the environment. An observation (or measurement) is what the robot actually sees (through its sensors) from its current pose. It can be a 2D scan or a 3D scan or a image etc. An observation at time step t is denoted as z_t. Control commands (or motion commands) at time step t are denoted as u_t. They could either be angular velocities of the wheels or high level commands like "go forward", "turn left" etc..

The state x is considered to be complete i.e, it is a sufficient summary of all that happened in the previous time steps. So, the state at time step t is only a function of the state at time

step $t-1$ and the motion command u_t. Mathematically,

$$p(x_t|x_{0:t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t)$$

Similarly, the current observation z_t only depends on the current pose of the robot x_t. These completeness *assumptions* are also referred to as *Markov assumptions* or *Markov properties*. Hence, the name *Markov Localization*.

$$p(z_t|x_{0:t}, z_{1:t-1}, u_{1:t}) = p(z_t|x_t)$$

The probability distribution $p(x_t|x_{t-1}, u_t)$ is the state transition probability. $p(z_t|x_t)$ is called the measurement probability. Models of these distributions are also called motion model and measurement model respectively. A belief distribution is a posterior probability over state variables conditioned on the available data. Belief over a state variable x_t is denoted by $bel(x_t)$.

$$bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$$

The belief before incorporating the measurement z_t is usually called as prediction (in our context of Bayes filtering) and is denoted by $\overline{bel}(x_t)$.

$$\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t}, x_0)$$

where, x_0 is the initial pose of the robot. Calculating $bel(x_t)$ from $\overline{bel}(x_t)$ is called measurement or the correction update. Given the belief at previous time step $bel(x_{t-1})$, the motion command u_t and the measurement z_t (taken from x_t), the belief at current time step $bel(x_t)$ can be computed using the Bayes filter as follows: for all x_t, we first predict the belief by incorporating the motion command:

$$\overline{bel}\,(x_t) = \int p\,(x_t|u_t, x_{t-1})\ \text{bel}\ \ (x_{t-1})\,dx$$

Then, the posterior belief is computed by fusing the measurement model based on the latest measurement z_t and the predicted belief $\overline{bel}(x_t)$ as follows:

$$\text{bel}\,(x_t) = \eta p\,(z_t|x_t)\,\overline{bel}\,(x_t)$$

This process of iteratively updating the belief by incorporating the motion model and measurement model at every time step is referred to as *recursive implementation*

of Bayes filter or *Markov Localization*. Please refer to fig-2.1 for a demonstration of Markov localization incorporating all this theory in a 1-D case. However, the integral $\int p\left(x_t|u_t, x_{t-1}\right)$ bel $\left(x_{t-1}\right) dx$ is intractable because of its continuous nature and hence we approximate this *recursive bayes filter* algorithm via discretization (Sec-2.2.1) or by using particles (Sec-2.2.2).

2.2.1. Grid Based Markov Localization

In a grid based localization technique, the belief over the entire space is represented as a 3D tensor. The values in the 3D tensor represent the probability of the robot being in that pose ($\mathbf{x}$, $\mathbf{y}$, θ). The posterior belief is approximated as a *histogram filter* over this discrete representation:

$$bel(x_t) = p_{k,t}$$

where each probability $p_{k,t}$ is defined over a grid cell x_k. Thus, the recursive Bayes filter can be extended to this discrete case as follows: For all k,

$$\overline{p}_{k,t} = \textstyle\sum_i p\left(X_t = x_k|u_t, X_{t-1} = x_i\right) p_{i,t-1}$$
$$p_{k,t} = \eta p\left(z_t|X_t = x_k\right) \overline{p}_{k,t}$$

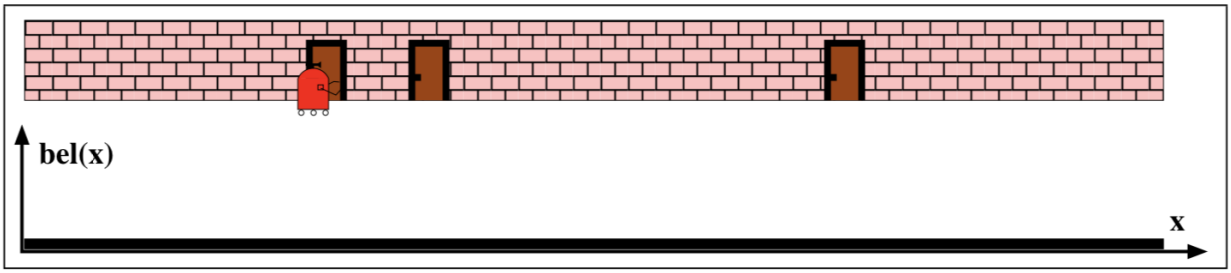
This framework of discretized recursive Bayes filter is the primary motivation for this thesis. We will go on to show later that learning the measurement model $p(z_t|X_t = x_k)$ via domain randomization can enable us to accurately localize despite noises in both the motion and measurement models.

2.2.2. Particle Filter Based Markov Localization

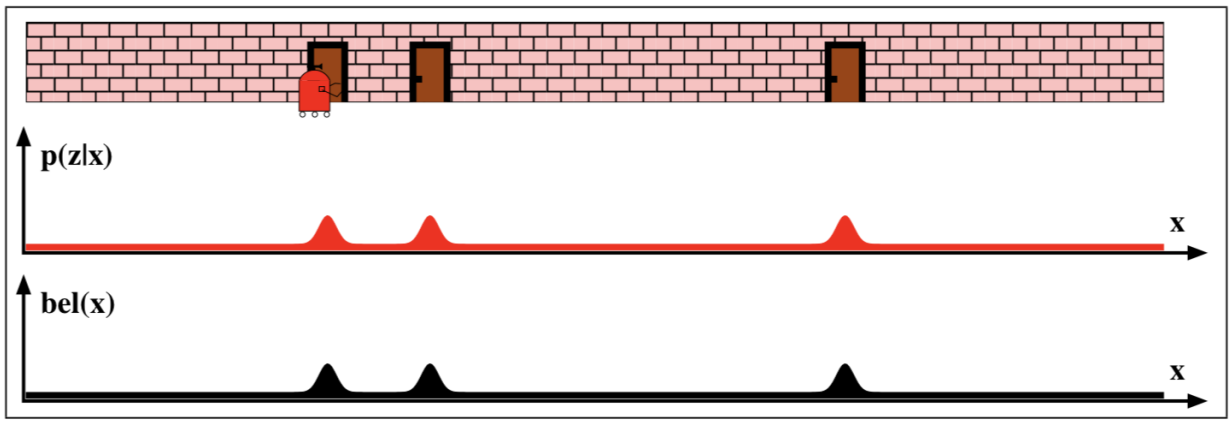
Particle filter based localization is a non-parametric approximation of the Bayes filter **[74]**. The goal is to represent the posterior belief $bel(x_t)$ with a set of random state samples drawn from the posterior. These samples are called particles. A particle set is represented with X_t and each particle by $x_t^{[i]}$. Mathematically, for m particles:

$$X_t = x_t^{[1]}, x_t^{[2]}, ...x_t^{[m]}$$

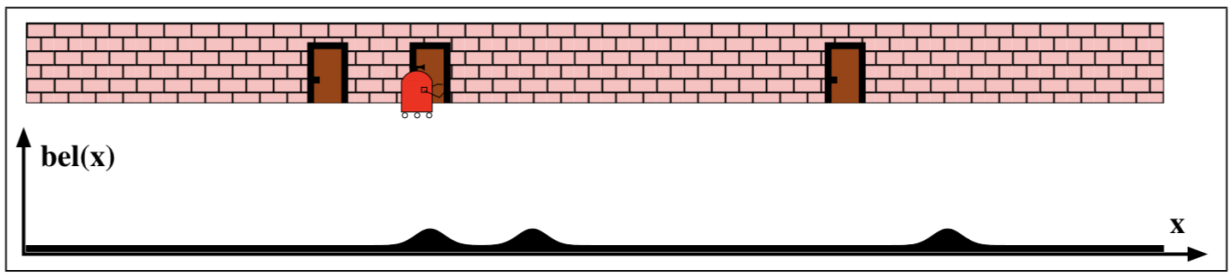
Each particle represents a full state. For the case of localization, a full state would just mean its pose ($\mathbf{x}$,$\mathbf{y}$,θ). The goal is to make these samples represent a true posterior belief.



(a) In a global localization problem, we don't know the initial pose of the robot. So, the belief is initialized as a uniform distribution.



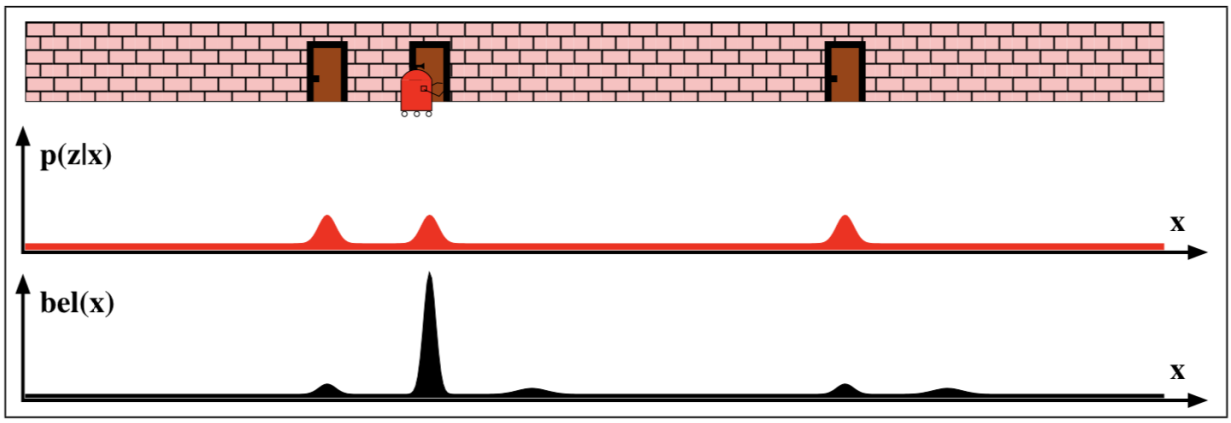
(b) The robot observes a door. So, the likelihood model $p(z|x)$ predicts that the robot is likely to be present infront of any door. Multiplying this likelihood with our prior belief (which is uniform) gives us the posterior belief $bel(x)$. So, the robot now believes that it is infront of a door, but is not sure of which door it is.



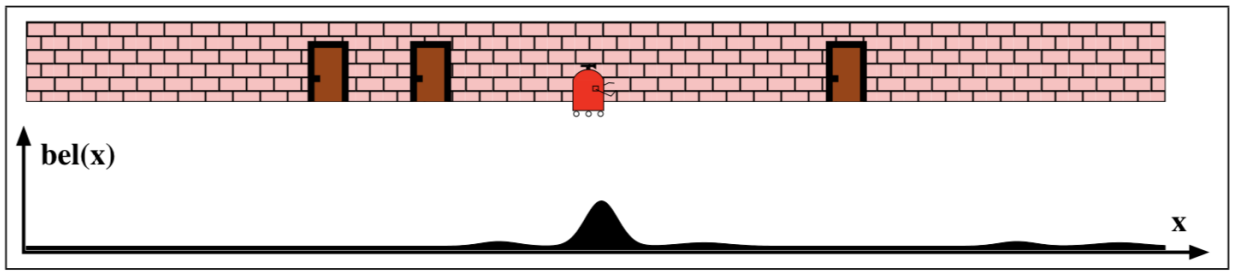
(c) When the robot moves towards right, due to noises in the motion model $p(x_t|x_{t-1},u_t)$, the variance of the belief $bel(x)$ slightly increases and the peak reduces. In the context of Markov localization, this is also referred to as predicted belief $\overline{bel}(x)$.

Fig. 2.1. Demonstration of Markov Localization, adapted from **[73]**

$$x_t^{[m]} \sim p(x_t|z_{1:t}, u_{1:t}, x_0)$$



(d) The robot now takes an observation z again and it observes the door again. So, similar to fig-1(b), the output of the likelihood model p(z|x) will be same, where it predicts that the robot is likely to be present infront of any door. However, if we now multiply this with the predicted belief obtained in fig-1(c), we get a corrected belief which has a peak infront of the second door, where the robot actually is.



(e) When the robot moves again, the robot becomes slightly inconfident of its pose again due to noise in the motion model, but, the maximum likelihood estimate of its belief $bel(x)$ is same as its true location which indicates that the robot has localizaed well.

Fig. 2.1. Demonstration of Markov localization, adapted from **[73]**

Particles from time step $t-1$ to time step t can be transformed using a Bayes filter. At first, a proposal distribution is obtained by transforming every particle $x_{t-1}^{[i]}$ using the state transition distribution:

$$x_t^{[i]} \sim p(x_t|u_t, x_{t-1}^{[i]})$$

This is similar to $\bar{bel}(x_t)$. Now, to obtain the posterior distribution $bel(x_t)$, each of these particles has to be resampled (importance sampling **[30]**) with replacement **[59]**. The importance factor of every particle is computed as the likelihood of the current observation z_t based on its believed pose $x_t^{[i]}$:

$$w_t^{[i]} = p(z_t|x_t^{[i]})$$

2.3. Active Localization

In the Markov localization methods discussed so far, we assumed that the control commands u_t are given to the robot. The control commands for a mobile robot usually refer to the angular velocities of the two front wheels. The problem of active localization, which is the main focus of this thesis, deals with finding these control commands u_t such that the localization error is minimized. More precisely, we define the problem statement as follows:

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:

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

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^*.

Burgard et al. introduced *active* localization in their seminal work [**7**]. They demonstrated that, rather than passively driving a robot around, picking actions that reduce the expected localization uncertainty results in better localization. Using an entropy measure characterized as a mixture-of-Gaussians, they demonstrate that the framework of Markov localization [**15**] 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 [**12, 61, 46, 79, 13, 34**] use a similar, information gain maximization cost for the task of active localization or SLAM (Simultaneous Localization And Mapping [**10, 5**]).

Feder et al [**12**] chose the action which maximizes the information gain. Roy et al [**61**] also argued that going along the high information paths helps in better localization even though the time taken or distance traveled might be higher. Mariottini et al [**46**] also used an entropy-based planning algorithm after computing SIFT (Scale Invariant Feature Transform [**45**]) features (and disambiguate them using visual bag of words [**43**] and vocabulary tree techniques). Valencia et al [**79**] tackled the problem of active SLAM in the pose SLAM setting based on the information gain of a link (edge) in the pose graph.

Kollar et al [**36**] used optimization methods to find the robot locations and then applied reinforcement learning techniques to generate control commands. However, their approach is limited to planning to explore with reinforcement learning given the belief of robot's pose.

Forster et al [**13**] chose the motion trajectories that minimize the perceptual ambiguities by deriving an expression for information gain in terms of camera parameters and other terms. The framework introduced by Kim et al [**34**] accounts for SLAM localization uncertainty, area coverage performance, and the identification of good candidate regions in the environment for visual perception.

Chapitre 3

Machine Learning Methods for Localization

3.1. Review of Deep Learning

Deep Learning is currently being used in a wide range of applications like recognizing hand written digits [**41**], machine translation [**4**], language generation [**60**], playing board games like chess [**78**], go [**64**], Hex [**2**], playing video games like Atari [**50**], starcraft [**80**], generating realistic images [**17**], generating fake videos [**35**], facial recognition [**81**], object tracking [**24**], predicting stock prices [**51**] and many others. Though most of these applications were previously tackled with other statistical methods (like Support Vector Machines [**77**], decision trees [**58**]) or domain-specific methods [**45**], deep learning has helped improve the accuracy, has better generalization abilities [**52**] and has removed the dependency on domain specific knowledge.

Deep learning, especially in the context of robotics, typically refers to the the process of training deep neural networks to minimize an objective function corresponding to the given training (input, target) data. The network is also expected to perform equally well on the test set, which is not used during the training procedure. This is achieved by modifying the parameters of the neural network based on the gradients of the objective function w.r.t these parameters. This procedure is called *gradient descent*.

3.1.1. Mathematics of Deep Learning

A fully connected deep neural network (figure-3.2) consists of layers of neurons. The first layer takes the input and is called the input layer, the final layer is the output layer and all the other layers in between are called hidden layers. Each neuron (figure-3.1) takes

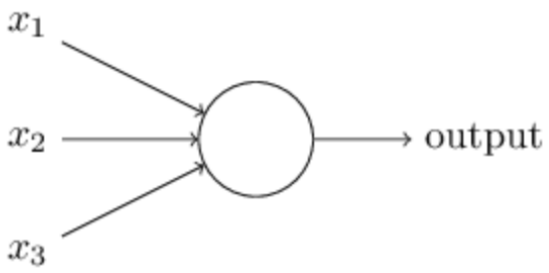


Fig. 3.1. A neuron, adapted from[**53**]

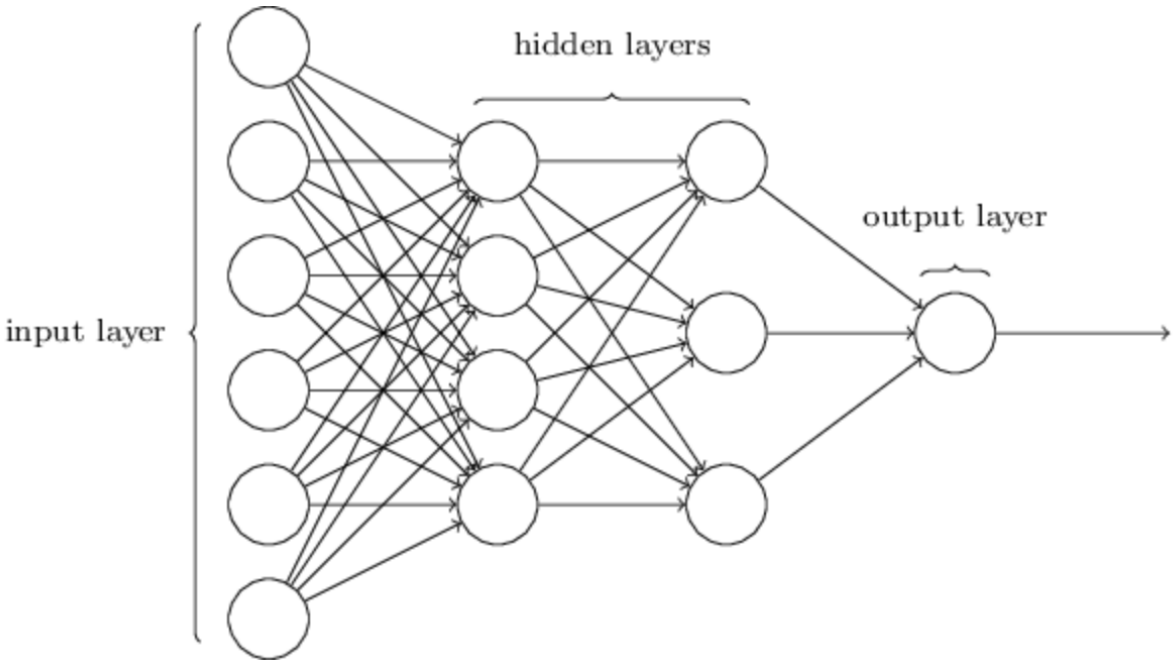


Fig. 3.2. A fully connected neural network with two hidden layers, adapted from [**53**]

in multiple inputs (which are outputs of the previous layer) and sums them by multiplying each of its inputs with their corresponding weights. These weights are the parameters of our neural network which adjust themselves to minimize the error function, which is typically a function (like mean squared error) of the output of this neural network and the target output over all the input samples.

Adopting the notation from [**53**], let w^l_{jk} be the weight for the connection from the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer. Similarly, let b^l_j be the bias of j^{th} neuron in the l^{th} layer. Let a^l_j be the activation of j^{th} neuron in the l^{th} layer. Let σ be

(a) (b)

Fig. 3.3. Notations used

the sigmoid activation. Then, the output at a^l_j is:

$$a^l_j = \sigma\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

This can be compactly written in vector form as:

$$a^l = \sigma\left(w^l a^{l-1} + b^l\right)$$

Let $z^l \equiv w^l a^{l-1} + b^l$ and the cost (error) function be denoted as C. Mathematically, a quadratic cost function can be written as:

$$C = \frac{1}{2n}\sum_x \left\|f(x) - a^L(x)\right\|^2$$

where x are the inputs, $f(x)$ are the corresponding target outputs and a^L_x are the corresponding neural network outputs. and our goal is to reduce this cost function, which is typically done by a method called *gradient descent*.

Let δ^l denote the error of neurons at layer l. $\delta^l_j \equiv \frac{\partial C}{\partial z^l_j}$. Then, the error at the output layer will be:

$$\delta^L_j = \frac{\partial C}{\partial a^L_j}\sigma'\left(z^L_j\right)$$

which can be further simplified as follows:

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$$

$$\delta^L = \left(a^L - y\right) \odot \sigma'\left(z^L\right)$$

where, $\odot$ is the element-wise product.

Now, the errors for the previous layer can be obtained using:

$$\delta^l = \left(\left(w^{l+1}\right)^T \delta^{l+1}\right) \odot \sigma'\left(z^l\right)$$

With these error terms, weights and biases of the network can be modified as follows:

$$\frac{\partial C}{\partial b_j^L} = \delta_j^l$$
$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l$$

While fully connected layers can solve lot of easy problems like MNIST image classification, they fail to work for larger datasets as they can't capture spatial information and hence break when the images are slightly enlarged or rotated. Convolutional layers help us in overcoming these problems by using *learnable filters* which are slid across the entire input example to capture spatial relations or to identify a particular feature. For example, a convolution equation can be written as: $a^1 = \sigma\left(b + w * a^0\right)$ where a^1 denotes the set of output activations from one feature map, a^0 is the set of input activations, and * is called a convolution operation. Convolution layers are often used in conjunction with pooling layers [**53**].

One of the contributions of this work is to train likelihood model $p(z|x)$ where the input is the map $\mathcal{M}$, observation (2D scan image) z_t and the output is $p(z|x)$ which is a 3D tensor. So, we only use convolutional layers for this purpose. However standard convolutional neural networks (CNNs) failed to give good performance, prompting us to use more robust networks like Resnet and DenseNet which we detail below.

3.1.2. Resnet

For very deep networks (dozens or hundreds of layers), the gradient of the objective function cannot be easily propagated to the initial layers of the network. It faces the problem of vanishing gradients because of repeated multiplication of the gradient across every layer. That is, the gradient value becomes infinitesimally small making it inconsequential in the parameter update equation. The residual network (Resnet [**21**]) is first proposed in 2015 to counter the vanishing gradient problem encountered during training of very deep networks.

Instead of fitting the parameters for $F(x)$, they introduce "identity shortcut connection" through which they fit the parameters for $F(x) + x$ which solves the vanishing gradient

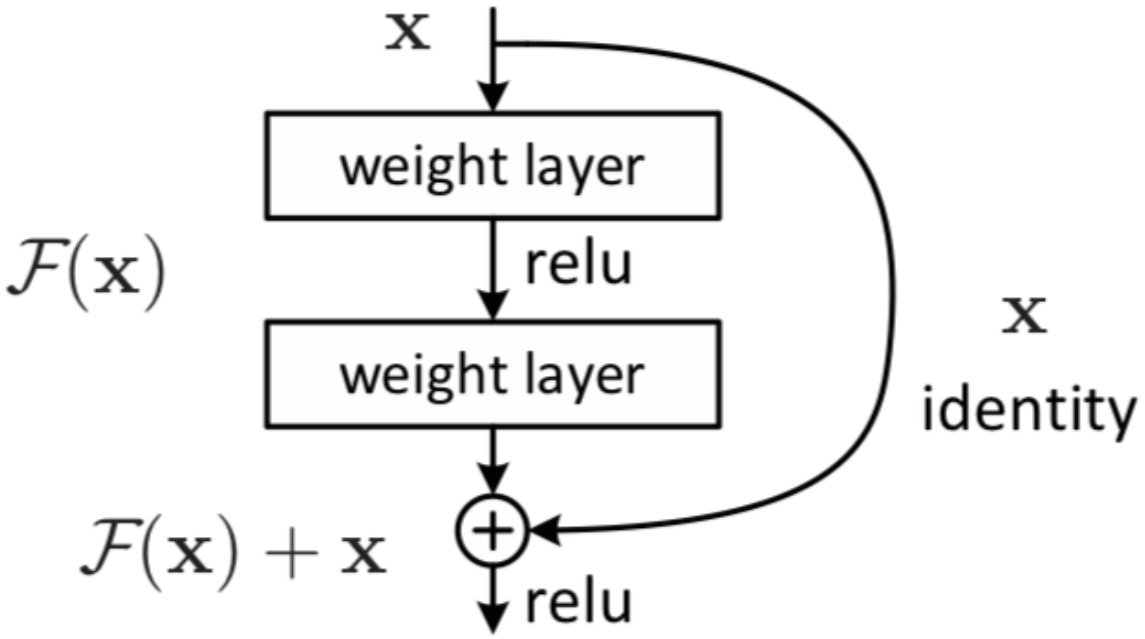


Fig. 3.4. A Resnet block, adapted from [**22**]

problem by just outputting an identity mapping from the Resnet block in the worst case scenario. Highway networks [**65**],[**66**] and LSTM [**25**] also explore a similar idea of gating "information" to counter the vanishing gradient problem. Results demonstrate that Resnet performed better in static (non-time series) supervision tasks like image classification. It was further improved upon by [**23**] where the gradients are allowed to flow to any other earlier layers through the shortcut connections, and it was used successfully to train a 1001-layered network. ResneXt [**84**] demonstrated further improvement by using split-transform-merge paradigm similar to inception network [**71**].

3.1.3. DenseNet

To achieve better training accuracies, one of the key goals of deep networks is to facilitate a better flow of information and gradients between the initial layers and final layers of the network. Stochastic depth network [**27**] tries to achieve this by randomly dropping layers during training. Fractal network [**40**] repeatedly combines several parallel layer sequences with a different number of convolutional blocks to obtain a large nominal depth. However, all these approaches just create a short path from initial layers to final layers. On the other

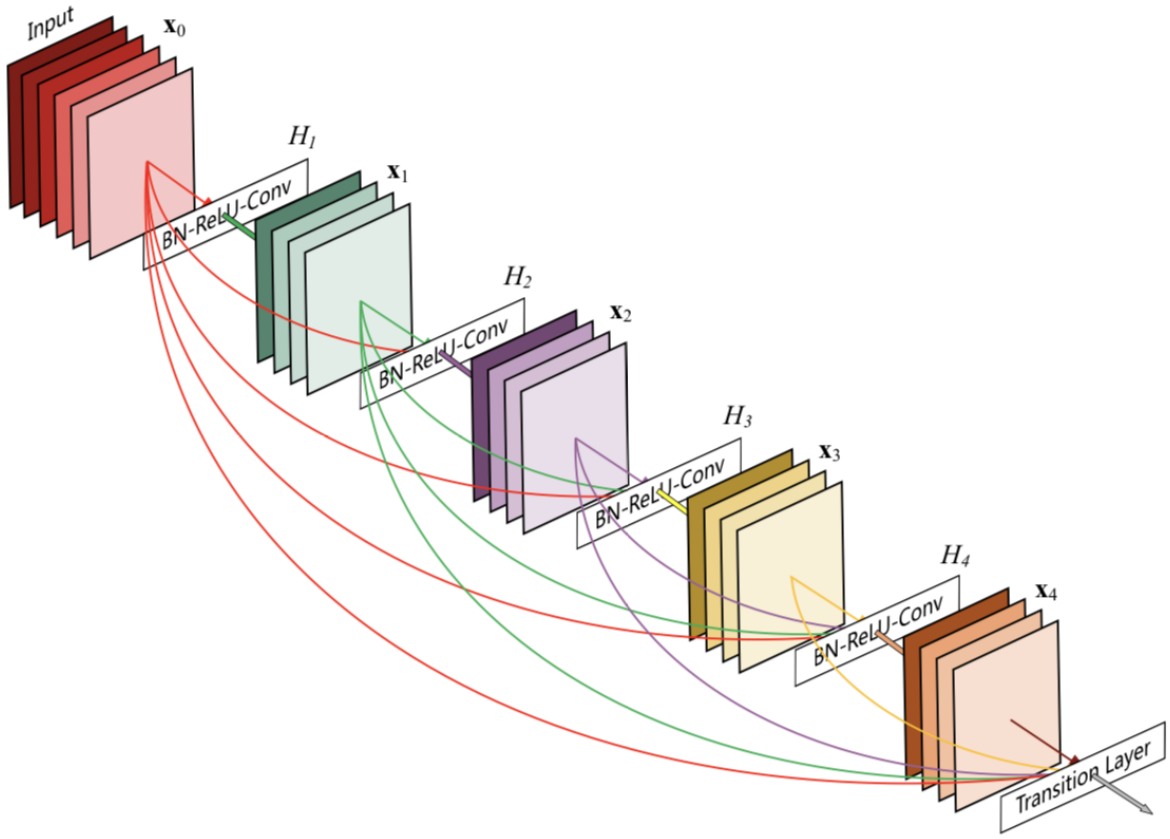


Fig. 3.5. A sample densenet, adapted from [**26**]

hand, Densenet [**26**](fig-3.5) connects all the layers directly with each other. One important difference to note here is that while Resnet (and its variants) combine features by adding them, densenet combines features by concatenating them.

3.2. Deep Reinforcement Learning

Supervised learning applies only to the settings where labelled data are available and the networks only need to learn a mapping from inputs to targets. Human brains do not usually function this way. Humans learn by interacting with the environment. Reinforcement learning (RL) deals exactly with this framework of "learning by interacting" where it learns to map the situations to actions to achieve a predefined goal or to maximize a given reward function. The rewards are often delayed which makes RL all the more challenging. Instead of optimizing the immediate reward or a random trial and error search, the algorithm needs to be intelligent enough to optimize the long-term rewards.

Recent advances in reinforcement learning have enabled its widespread use in various applications including various game playing agents **[78]** **[80]** **[64]**, autonomous cars **[33]**, health care **[18]** among others. We leverage some of these advancements in this thesis, specifically the algorithms related to policy gradients.

The Markov Decision Process (MDP)**[57]** is a mathematical framework for describing the underlying reinforcement learning task. It is characterized by the state s_t, action a_t, transition function $P(s_{t+1}|s_t, a_t)$ and reward r_t at every time step. Given a discount factor $\gamma \in [0,1)$, 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.

The RL algorithms can be classified into many types **[69]**: model-based vs model-free, on-policy vs off-policy. More importantly, they can be classified based on the update rule: policy-based **[70]** (or, policy gradient methods) which directly update the policy and value based methods. We have used policy gradient methods to train our policy model and here, we briefly explain the various policy gradient methods.

The objective is to choose a policy parameterized by θ that maximizes the objective function $J(\theta)$ defined as:

$$J(\boldsymbol{\theta}) \doteq v_{\pi_{\boldsymbol{\theta}}}(s_0)$$

where, v is the value function corresponding to policy π_θ at the initial state s_0. Policy gradient methods involve updating a policy based on the reward obtained at every time step or at the end of an episode. The goal is to update the current policy in such a way that the overall reward is maximized. This is achieved by updating the policy in the direction of the actions which give maximum reward. If the policy π is characterized by parameters θ then the policy gradient update rule can be written as:

$$\theta_{t+1} = \theta_t + \alpha \frac{\hat{Q}(s,a) \nabla \pi_{\theta_t}(a|s)}{\pi_{\theta_t}(a|s)}$$

where, α can be seen as a learning rate parameter, and Q is the action-value function, or simply the Q-function, which gives the total discounted return for taking action a from state s. This can be re-written as:

$$\theta_{t+1} = \theta_t + \alpha \hat{Q}(s,a) \log \nabla \pi_{\theta_t}(a|s)$$

3.2.1. REINFORCE

REINFORCE [**82**] is the simplest form of policy gradient algorithm, where the update rule is exactly the same except that we need to use empirical estimates of $Q(s,a)$ (obtained via Monte-Carlo rollouts) instead of the actual Q-function which is not available to us. An improvised version, also called "REINFORCE with baseline" uses the "advantage function" $A(s,a)$ instead of Q-function. The advantage function is defined as: $A(s,a) = Q(s,a) - V(s)$. Thus, the update rule becomes:

$$\theta_{t+1} = \theta_t + \alpha \hat{A}(s,a) \log \nabla \pi_{\theta_t}(a|s)$$

3.2.2. Actor Critic

Instead of using Monte Carlo rollouts for the Q-function (which only works in episodic environments and are sample inefficient), Actor critic methods [**37**] use another neural network (critic) to learn the Q-function. In most cases, actor and critic are just two different fully connected heads of a same convolutional neural network.

3.2.3. Advantage Actor Critic

Actor critic still suffers from high variance of the gradient estimates. In order to reduce the variance, a baseline function (independent of action; value function is the usual choice) is subtracted from the Q-function, and the resulting function is called advantage function [**48**] $A(s,a) = Q(s,a) - V(s)$. Note that this is still unbiased since the expectation of the advantage function will still give us the Q-function.

3.2.4. Trust Region Policy Optimization

The methods we have discussed so far suffer from either too large (catastrophic) or too small (no improvement) gradient steps. Trust region policy optimization (TRPO)[**62**] addresses this problem. While these other policy gradient methods are based on line search optimization (gradient descent), TRPO is a trust region based optimization where maximum allowed step size is first determined and then an optimal point is located within this trust region. Mathematically, the parameters are updated as follows:

$$\theta_{k+1} = \arg\max_\theta \mathcal{L}(\theta_k, \theta)$$
$$\text{s.t. } \overline{D}_{KL}(\theta \| \theta_k) \leq \delta$$

where, $\mathcal{L}(\theta_k, \theta)$ measures how the current policy π_θ performs relative to the old policy π_{θ_k} on the data from the old policy:

$$\mathcal{L}(\theta_k, \theta) = \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}}\left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a)\right]$$

and, $\overline{D}_{KL}(\theta \| \theta_k)$ measures the KL divergence between the current policy and the old policy on the data from the old policy

$$\overline{D}_{KL}(\theta \| \theta_k) = \mathop{\mathrm{E}}_{s\sim\pi_{\theta_k}}\left[D_{KL}(\pi_\theta(\cdot|s) \| \pi_{\theta_k}(\cdot|s))\right]$$

where, the KL divergence between two probability distributions $p(x)$ and $q(x)$ is defined as: $D_{KL}(p(x)\|q(x)) = \sum_{x\in X} p(x) \ln \frac{p(x)}{q(x)}$

3.2.5. ACKTR

Actor Critic based Kronecker-Factored Trust Region (ACKTR)[**83**] optimizes both the actor and critic networks via the trust region method, but is much faster than TRPO because it computes Kronecker-factored approximate curvature (K-FAC)[**47**] instead of expensive conjugate gradients. On the other hand, it has much better sample complexity than actor-critic because of its large gradient updates using natural gradients.

3.2.6. Proximal Policy Optimization

While TRPO and ACKTR solve the problem of size of gradient steps, they both use complex second order updates which are computationally expensive. Proximal policy optimization (PPO)[**63**] addresses the same problem while still making first order gradient updates. It does so by adding a penalty term to the objective function instead of using it as a constraint. So, the parameter update equation is:

$$\theta_{k+1} = \arg\max_\theta \max_{s,a\sim\pi_{\theta_k}} \left[L(s, a, \theta_k, \theta)\right]$$

where,

$$L\left(s,a,\theta_k,\theta\right)=\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}A^{\pi_{\theta_k}}(s,a),\quad \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)},1-\epsilon,1+\epsilon\right)A^{\pi_{\theta_k}}(s,a)\right)$$

Even though it appears complex, it is essentially making sure that the current policy is not very different from the old policy.

3.2.7. Intrinsic reward setting

While all these metrics make intuitive sense, there is no empirical or theoretical evidence to prove that one of these metrics is better than the other for a learning algorithm. So, we let the algorithm learn it's own reward function, adopting the approach from **[85]**. We train a "intrinsic reward" network to output rewards at every time step. The parameters of this network are optimized based on a standard policy loss used in any policy gradient method. We have observed that this boosts the performance in some experiments.

θ: policy parameters

η: intrinsic reward parameters

r^{ex} = extrinsic reward from the environment

$r^{in}_\eta = r^{in}_\eta$ (s,a): intrinsic reward estimated by η

$G^{ex}(s_t,a_t)=\sum_{i=t}^{\infty}\gamma^{i-t}r_i^{ex}$

$G^{in}(s_t,a_t)=\sum_{i=t}^{\infty}\gamma^{i-t}r_\eta^{in}(s,a)$

$G^{ex+in}(s_t,a_t)=\sum_{i=t}^{\infty}\gamma^{i-t}(r_i^{ex}+\lambda r_\eta^{in}(s,a))$

$J^{ex}=E_\theta[\sum_{t=0}^{\infty}\gamma^t r_t^{ex}]$

$J^{in}=E_\theta[\sum_{t=0}^{\infty}\gamma^t r_\eta^{it}(s_t,a_t)]$

$J^{ex+in}=E_\theta[\sum_{t=0}^{\infty}\gamma^t(r_t^{ex}+\lambda r_\eta^{it}(s_t,a_t))]$

λ: relative weight of intrinsic reward

And, the gradient updates are as follows:

$$\Delta_\theta=\alpha\nabla_\theta J^{ex+in}(\theta)$$

$$\boldsymbol{\Delta}_\eta=\beta G^{ex}(s_t,a_t)\frac{\nabla_{\theta'}\pi_{\theta'}(a_t|s_t)}{\pi_\theta(a_t|s_t)}\alpha\lambda\sum_{i=t}^{\infty}\gamma^{i-t}\nabla_\eta r_\eta^{in}(s_i,a_i)\nabla_\theta log\pi_\theta(a_t|s_t)$$

The reader can refer to **[85]** for the derivation of these gradient updates.

3.3. Deep Learning and Reinforcement Learning for Localization

So far, we have seen formulations of classical localization (sec-2.1) and a brief overview of deep learning (sec-3.1) and reinforcement learning (sec-3.2). In this section, we will see how DL has been used for passive localization (sec-3.3.1) and RL for end-to-end active localization (sec-3.3.2) in some of the previous works and compare them with our approach.

3.3.1. Passive Localization

Robot localization with learning-based methods has a long history. For example, in the seminal work of Oore et. al. [**55**], the authors use a learning-based approach to predict sonar readings from different locations in a map. More recently, approaches such as PoseNet [**32**] and VLocNet [**1**] perform *visual* localization by training a convolutional neural network to regress to *scene coordinates*, given an image [**32**] or a sequence of images [**1**]. Recently, differentiable particle filters (DPFs) have also been proposed for global localization [**29, 31**]. 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, in our approach, the likelihood model $p(z|x)$ and the policy model $\pi(a|s)$ transfers essentially *zero-shot* across simulated environments, and from a simulator to the real-world.

3.3.2. Active Localization

Passive deep learning localization algorithms 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) [**9**].

ANL is the first known work to employ a learned model for active localization from images [**9**]. Similar to our approach, their approach comprises two modules, a perceptual model and a policy model, and can be trained end-to-end in a Bayesian framework. However, the ANL work makes several assumptions that limit its ability: 1) to scale to realistic-sized environments and 2) 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 scalability issue of ANL is related to the fact that a fixed resolution grid must be overlayed over the entire map. As a result, as the map size increases then the number of candidate grid locations as function of area will decrease and the distance that the robot must travel between grid centroids will increase. In DAL, we address this issue again with the 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.

Also of note is that in ANL, the perceptual model is not completely learned. The mapping from sensor input to a latent embedding is learned, but the measurement likelihood is generated by computing the cosine similarity of these latent representations. This is inefficient in-terms of memory and computation, as every memory image has to be passed through this perceptual model and stored even if the requirement is to compute likelihood only at very few locations in the environment. In our approach, the entire likelihood model is learned, and the input is just current observation and map of the environment.

Chapitre 4

Learning to Actively Localize

In this chapter we present our algorithm for active localization (selecting actions for a robot to be able to disambiguate its pose in a known environment). We develop an end-to-end differentiable system that is able to learn how to localize passively (through supervised learning) and simultaneously able to learn to take good actions for active localization (through reinforcement learning). We represent the free space as a grid of candidate points for localization and high-level actions consist of moving from the centroid of a grid cell to a cell infront of it, or to turn left or right (a discrete action space). Note that the space has been divided into discrete grids to enable us to obtain the posterior belief using recursive Bayes filter. However, we leverage a lower-level continuous feedback controller to ensure that the robot does not drift from the fixed grid over time.

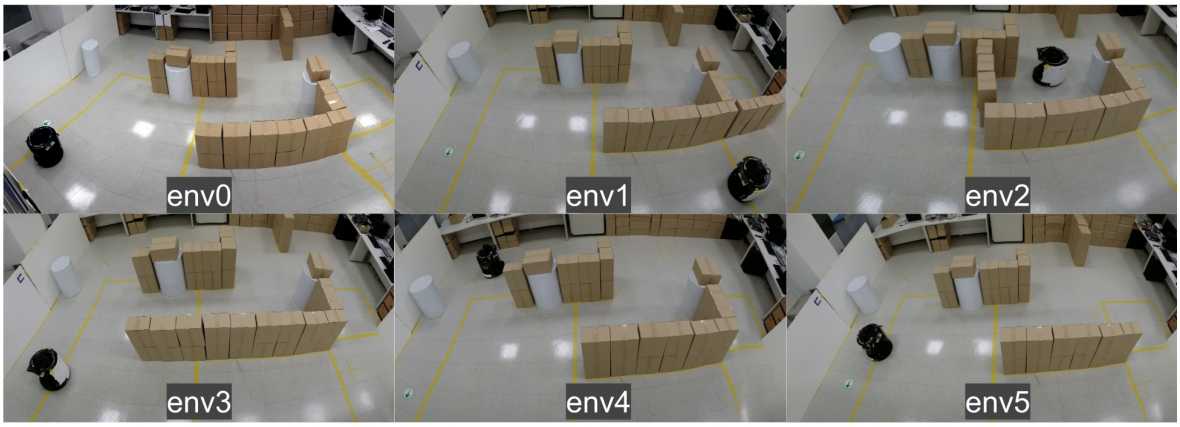


Fig. 4.1. JAY robot being tested in various environments

In the following subsections, we first give a detailed overview of the entire pipeline and algorithm. We will then describe the data generation and training procedures for the likelihood model and the hierarchical model with domain randomization techniques and then we elaborate further on the reinforcement learning algorithm used. Further, in chapter-5, we dive into details of dataset generation and experimental setup and compare our algorithm with traditional approaches (cosine similarity, Markov localization) and other learning based approaches like ANL [**9**]. We also provide experimental results on the robustness of our approach (fig-5.5) and effectiveness of hierarchical models (fig-5.7).

4.1. System Overview

Our approach to solving the active localization problem is summarized in Fig. 4.2 and Algorithm 1 [1]. For simplicity, we have shown two hierarchical levels in Fig. 4.2 since it is appropriate for our setup, but further levels of hierarchy could be added following the same approach(i.e., by providing: a measurement likelihood model, a planner, and a transition function) 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 $[m,n,\theta]$.

In Fig. 4.2, we assume that the neural network models are already trained, for a description of the training procedure for the measurement likelihood models see Sec. 4.2.1 and for the Reinforcement Learning (RL) policy see Sec. 4.3.1.

In this subsection we will define our problem precisely and outline the structure of our proposed solution.

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 the $\mathcal{M}$ 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 element-wise product) to produce the belief posterior. The belief posterior is fed as input to the RL model. The other inputs to the RL model are low dimensional map and low dimensional scan. It is important for this procedure, in order to train efficiently that the input is relatively low dimensional.

[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.

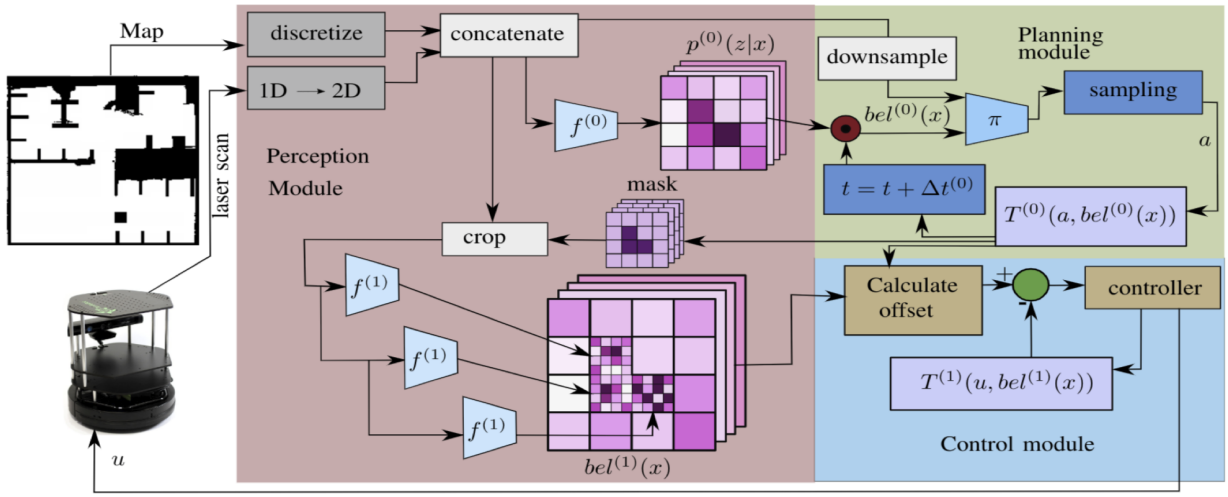


Fig. 4.2. Deep Active Localization: Our proposed method takes a map and sensor data (laser scan) and generates control actions in an end-to-end differentiable framework that includes learned perceptual modules (neural networks) at different scales and a learned policy network that is trained with reinforcement 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 RL policy is sampled to generate an action, a_t from the set { `left`, `right`, `forward` }. 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:

$$\begin{aligned} \bar{bel}^{(0)}(x_{t+1} = [n,m,\theta]) &= T^{(0)}(a_k, bel(x_t)) \\ &= \begin{cases} bel^{(0)}(x_t = [n,m,(\theta - 2\pi/|\Theta^{(0)}|)] & a_k = left \\ bel^{(0)}(x_t = [n,m,(\theta + 2\pi/|\Theta^{(0)}|)] & a_k = right \\ bel^{(0)}(x_t = [n + l_n \cos(\theta), m + l_m \sin(\theta), \theta]) & a_k = fwd \end{cases} \end{aligned} \tag{4.1.1}$$

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

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 s cells in $p^{(0)}(z_t|x)$
    for $i \in$ [0,s] **do**
      M_c, $S_c \leftarrow$ crop$(M_h, S_h, (n_i, m_i))$
      $p^{(1)}(z_t|x)[block_i] \leftarrow f^{(1)}(M_c, S_c)$
    $bel^{(1)} \leftarrow p^{(1)}(z_t|x) \odot \bar{bel}^{(1)}$ ▷ High-res belief
    $c_t^{(1)} \leftarrow argmax(bel^{(1)})$ ▷ Pose: max belief
    $u \leftarrow control(c_t^{(1)}, g_t)$

cell our agent actually is to compute a correction. 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 "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 to add 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..

4.2. Perception

The objectives of the perception system are:

(1) To provide a measurement likelihood to be used by RL agent, which is trainable in an end-to-end manner,

(2) To provide a refined estimate of 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.

4.2.1. 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 extended kalman filter [**28**]) is clearly inappropriate for the task of global localization, as evidenced by the prevalence of Monte Carlo based solutions [**74**].

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 Sec 4.1, the output of the likelihood model at time step t is given by:

$$p^{(0)}(z_t|x) = f_{\phi}^{(0)}(\mathcal{M},z_t) \tag{4.2.1}$$

where ϕ are the parameters of the neural network model.

4.2.1.1. *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.

4.2.1.2. *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 **[22]** or Densenet-121 **[26]** gave very good results in simulation but did not transfer when these models are transferred to real robot in a real environment. This demonstrates the need for domain randomization to achieve zero-shot transfer. See 4.2.4 for further details.

4.2.2. Bayes' Filter

We can now update the belief by taking element wise product of likelihood and the previous belief over all the $N \times M \times O$ grid cells

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

4.2.3. Hierarchical Likelihood Model

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

(1) It is 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.

Traditional approaches try to solve this problem using the cosine similarity technique (by taking dot product of the current scan with the scan at multiple poses and then normalizing the resultant tensor), but it has the drawback of being computationally very expensive as the current scan information has to be compared with scan information at multiple poses.

Recent approaches like ANL**[9]** try to overcome this problem by using neural networks to predict the likelihood in a 3D grid of dimensions $N^{(1)} \times M^{(1)} \times \Theta^{(1)}$, where the whole environment is divided into $N^{(1)}$ rows, $M^{(1)}$ columns and $\Theta^{(1)}$ orientations. Each of these grid cells represent the likelihood of robot in that pose. But, this doesn't alleviate the problem of

decoupling the localization precision from the size of the map. With input being $N \times M \times 2$, and the output being $N \times M \times \Theta$, it is usually difficult and inefficient for a CNN to learn this mapping.

In HLE (Hierarchical Likelihood Estimation), likelihood is first estimated at coarse resolution ($N^{(0)} \times M^{(0)} \times O$) and then each of these grid cells is 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 high the resolution map M_h (which is a processed version of map $\mathcal{M}$ of the environment) ($N^0 \times M^0$), and scan image S_h ($N^0 \times M^0$) which is obtained from the current LiDAR scan z_t 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 corresponding likelihood of the grid cell at 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 $p^{(0)}(z_t|x)$. The parameters of $f^{(1)}$ are optimized by minimizing the mean squared error (MSE) between $p^{(1)}(z_t|x)$ and ground truth likelihood $\tilde{p}^{(1)}(z_t|x)$ (which is at the same resolution of $N^{(1)} \times M^{(1)} \times \Theta^{(1)}$).

During the training phase, we choose c to be close to the 0.5 - 1.0 times the total cells ($N \cdot M \cdot O$) because otherwise small errors in $p^{(0)}(z_k|x)$ leads to compounding or irrecoverable errors in $p^{(1)}(z_k|x)$ and the entire model would collapse. During the testing phase, it is sufficient to just consider top one percent of the cells or even lesser as we are mostly concerned about finer likelihoods only in the most likely cells from $p^{(0)}(z_k|x)$

4.2.4. Domain Randomization

DR is the process of varying the parameters in a simulation environment during model training in the hope that the resulting policy becomes more robust to errors in the simulation (the reality gap) [**76**]. The hope is that when the agent is transferred from the simulator to the real robot environment that the real world appears as if it is just another simulator variation.

While the LiDAR sensor modality generally transfers better than vision (camera images) from simulation to the real world scenario 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 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 of softmax. It is important to normalize the ground truth likelihood to keep it bounded for a machine learning system to be able to learn. Dividing each element in $p^{(0)}(z_t|x)$ with the sum of all elements in $p^{(0)}(z_t|x)$ is a common way to normalize, but that gives us mostly uniform output. Hence, we use softmax with temperature which is defined as:

$$\sigma(p^{(0)}(z_t|x))_{n,m,\theta} = \frac{e^{\beta p^{(0)}(z_t|x)_{n,m,\theta}}}{\sum_{n,m,\theta} e^{\beta p^{(0)}(z_t|x)_{n,m,\theta}}} \tag{4.2.2}$$

  We vary the temperature parameter β randomly within limits to make sure that it is not biased towards overly-uniform or overly-sharp likelihood outputs
- Noise in the LiDAR scan: We train our network by adding Gaussian noise to every LiDAR scan point. Additionally, for every scan, we randomly set some of the incoming data points within the scan to have the value of $+\infty$.

- There are errors in the actual map of the environment created using gmapping [**20**]. This map is preprocessed and used as an input to the likelihood model and RL model for experiments on the real robot. Hence, it is important to ensure that some noise (more erosion and dilation of the map) is added to the map during training phase (on simulation) as well. However, note that the LiDAR readings will still come from the unperturbed map.

4.3. Planning and Control

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

4.3.1. Reinforcement Learning

Following the traditional conventions in a Markov Decision Process [**68**], we denote the state $s \in \mathbb{R}^d$ and the actions $a \in \mathbb{R}^{d_a}$, the reward function $r(s,a)$ and a deterministic 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 it's policy $\pi(a|s)$ to maximize the discounted return defined as: $G_t = R_t + \gamma R_{t+1} + ...$ over initial distribution of states (which is assumed uniform here). We use advantage actor critic (A2C) [**49**] 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 belief at true pose as our reward function since it is dense, well-behaved, and benefits from the availability of true pose in the simulator. The empirical evidence for the choice is given in Fig-4.3. We also benchmark the performance of Proximal Policy Optimization (PPO) [**63**] and ACKTR [**83**] in our deep active localization framework.

4.3.2. 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

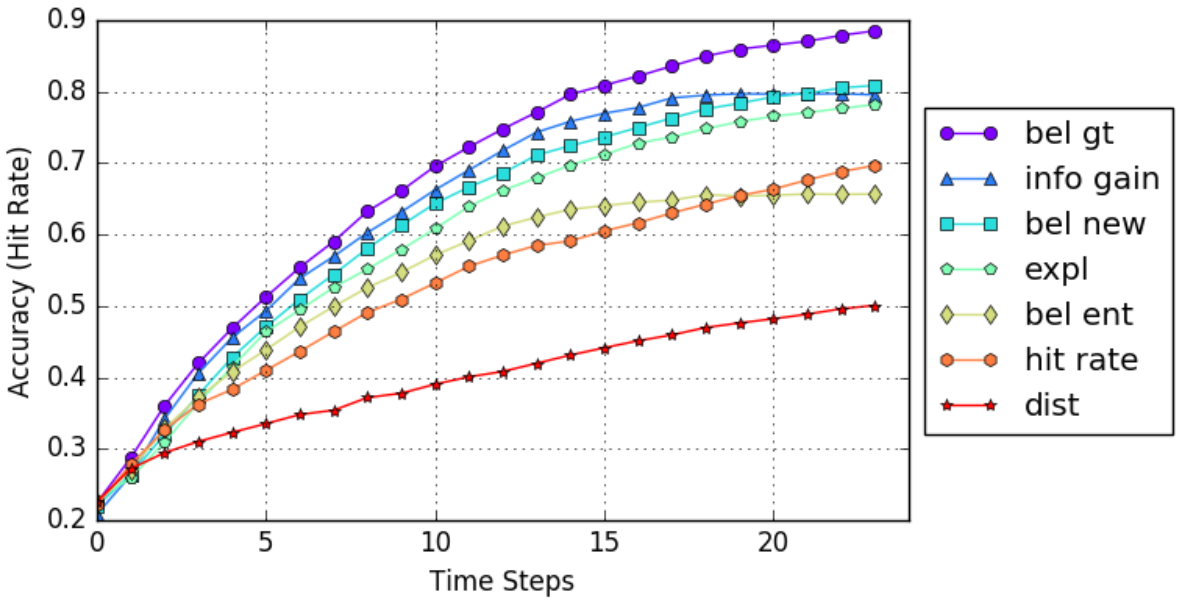


Fig. 4.3. We trained 7 RL policy models with the following different rewards: 1) probability mass of belief at true pose (bel gt), 2) decrease in the entropy of the belief (info gain), 3) reward of +1 for a new pose in belief (bel new), 4) reward of +1 for a new true pose (expl), 5) negative entropy of belief (bel ent), 6) reward of +1 if Manhattan distance error is equal to 0 (hit rate), 7) Manhattan distance error (dist).

hierarchical likelihood model. See the Sec. 4.2.3 for more details of hierarchical models. Optionally, the HLE model can be used after every fixed number of time steps to correct for the accumulated drift.

4.3.3. Zero-shot Transfer

The transferability of the perceptual model is enabled through the use of domain randomization (DR) as discussed in Sec. 4.2.4. Another common use of DR is to randomize over physical properties of the robot (i.e., dynamics). This is not necessary in our case since we are performing RL at the *planning* level of abstraction and using a more traditional feedback controller to execute the plans.

Chapitre 5

Experimental Setup and Results

We demonstrate the efficacy of DAL by conducting several experiments in simulation and on real robots in a diversity of 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 re-configurable. Using a SLAM software (gmapping **[20]**) we then obtained maps as shown in Fig. 5.3.

5.1. Dataset Generation

Using the following algorithm, we generated a dataset to train the likelihood model to estimate the measurement likelihood $p(z|x)$. The algorithm generates random maps, sensor data at some random poses as well as the target distributions at those poses.

(1) We use Kruskal's algorithm to generate random maze-like environments at low resolution. The output is a square matrix with ones representing obstacles and zeros 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 low resolution map at all $\Theta^{(0)}$ directions. From each of those locations, we simulate a laser scan. So, we obtain $M^{(0)} \times N^{(0)} \times \Theta^{(0)}$ laser scans. We call this as scan matrix and it's of dimension $M^{(0)} \times N^{(0)} \times \Theta^{(0)} \times 360$.

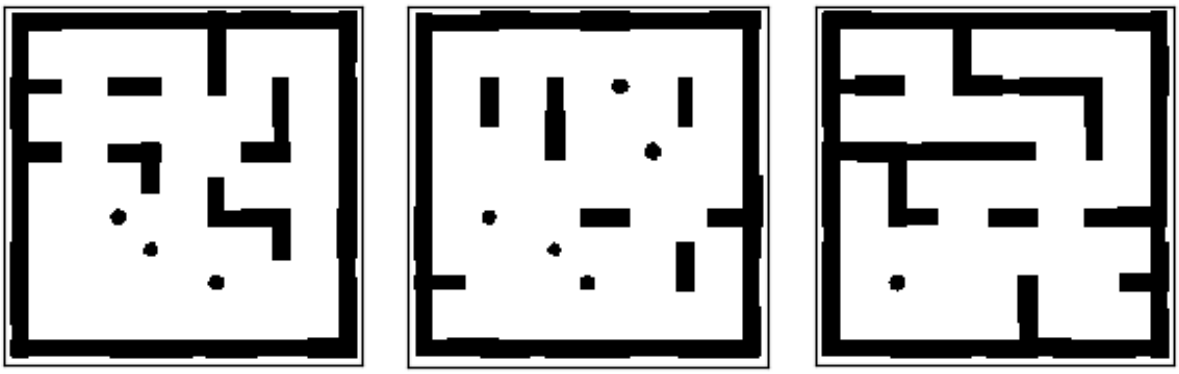


Fig. 5.1. Random maps generated from Kruskal's maze algorithm with some of the walls pruned

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. The noise added to this process is described in 4.2.4 and 5.2

5.2. Training 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 $33 \cdot 33$ x and y locations.

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,
- 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.

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 randomly generated 2,000 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.

5.3. Experimental Setup

To demonstrate the feasibility and robustness of our approach, we tested our trained likelihood and policy models on 2 mobile robots: JAY and Turtlebot and successfully localized in two different environments.

5.3.1. Experiments on JAY

JAY (Rainbow Robotics) is cylinder-shaped with the radius of 0.30m, the height of 0.50m and the weight of 46 kg before modification, whose CPU is IntelCore 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. 5(a), we tested our likelihood model. To compare the localization performance with some of existing methods that run on grids, the experiments included the following methods.

- Trained likelihood model (LM): use the trained model for the sensor measurement likelihood at each pose.

(a) JAY with top-mounted 360-degree view LiDAR

(b) Turtlebot with top-mounted 260-degree view LiDAR

Fig. 5.2. Robots used in our experiments

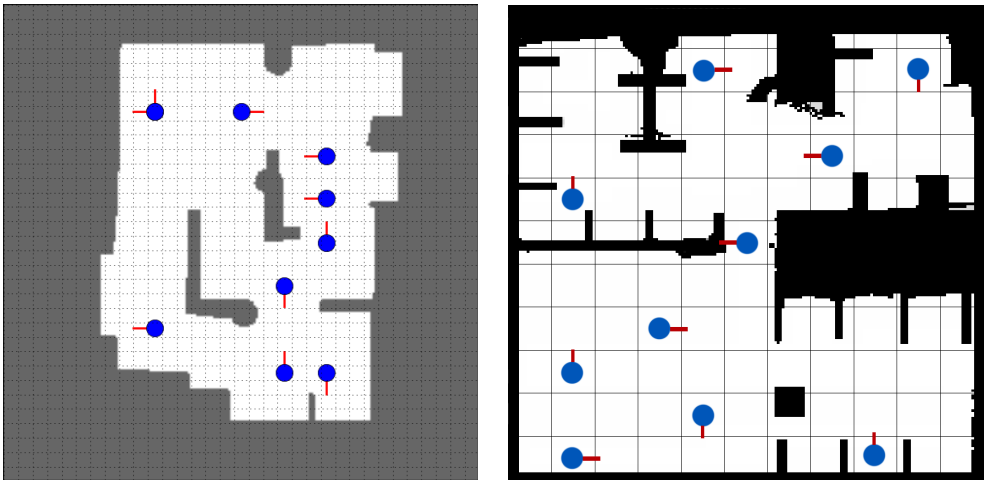
- Cosine similarity (CS): use cosine similarity between the precomputed scan matrix and the input scan for the sensor measurement likelihood.
- Reinforcement Learning (RL): use the trained policy model π for sampling next action.
- Active Markov Localization (AML): the robot virtually goes one step ahead along each of the possible actions and select the one with the largest reward, defined as the decrease in entropy.
- Random Action (RA): sample next action randomly with uniform probability.

The work that is most related to our own is ANL [**9**]. 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.

For each test condition, it ran for 10 episodes of length 11. Fig. 3(a) shows the 10 initial poses that were randomly sampled to be applied to all the tests.

A custom control law computed velocity command at each step to move JAY to the next target pose by using the odometry feedback, where the target pose was computed from the believed pose $belx_t$ and the action sampled from the policy.



(a) Sait map (b) Mila map

Fig. 5.3. The maps of the real environments in SAIT and Mila were generated by using a SLAM algorithm, *gmapping* [**20**]. The map size was adjusted to 224×224 pixels each of size $0.04m \times 0.04m$. We randomly selected 10 poses on the map and used them repeatedly as the starting poses for the 10 episodes of each test.

5.3.2. Results from JAY Experiment

We evaluated the test results using the following metrics:

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

$$W = \sum_{x} p(x) D(x,x^*), \tag{5.3.1}$$

where we define D to be the Manhattan distance between two poses.

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

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

Each test consists of 10 episodes; the mean and standard deviation over the 10 episodes were then computed for each of 11 steps as plotted in 5.4 to illustrate how the metrics changed during an episode in average.

From the results plotted in Fig. 5.4, we can clearly see that our trained likelihood model performed much better than scan matching in all the metrics. This could be attributed to the susceptibility of SM to noises such as inaccurate map and the robot being off the center of a grid cell. Since we accounted for all these noises during training, our LM proved to be robust. Our RL model performed as good 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 5.1). 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.

	SM	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

Tab. 5.1. Time complexity analysis (in seconds)

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 5 differently modified environments from 'env1' to 'env5' shown in Fig. 5(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. Number of headings was fixed at 4.

5.3.3. Experiments on Turtlebot

We used 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 4GB RAM. Our test environment is of size 9×9 meters^2 and is combination

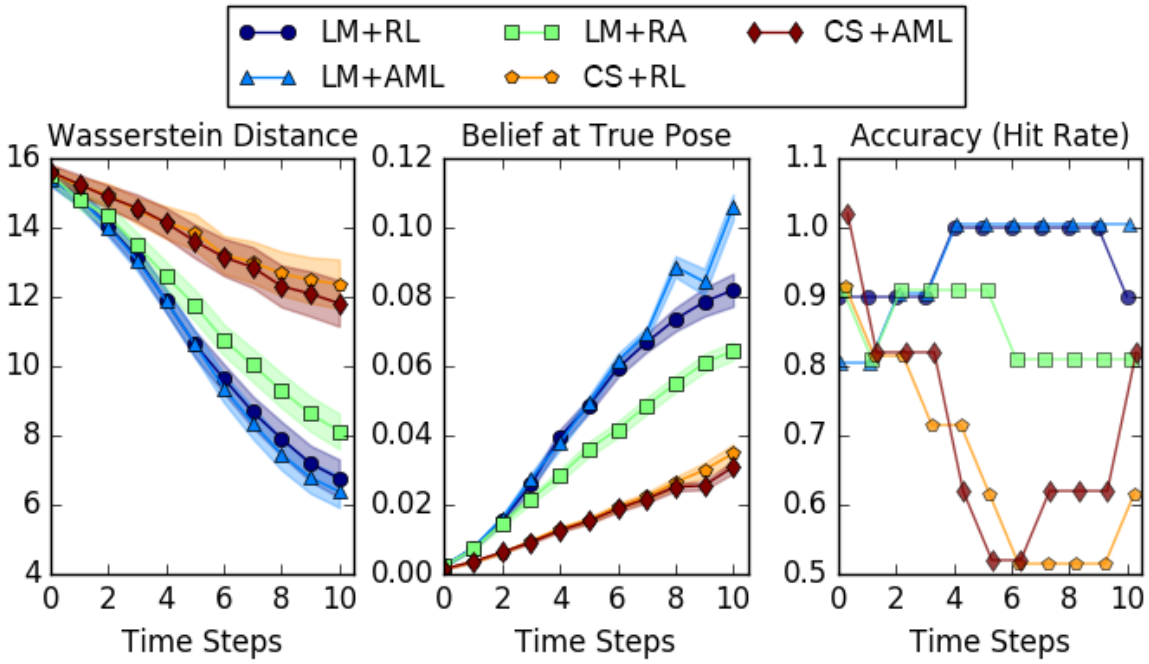


Fig. 5.4. Localization performance was evaluated in 8 headings and 33×33 grids with JAY in the real environment. We compared our proposed likelihood model for 33×33 (LM), cosine similarity (CS), active Markov localization (AML), random action (RA), and the trained policy π with reinforcement learning (RL).

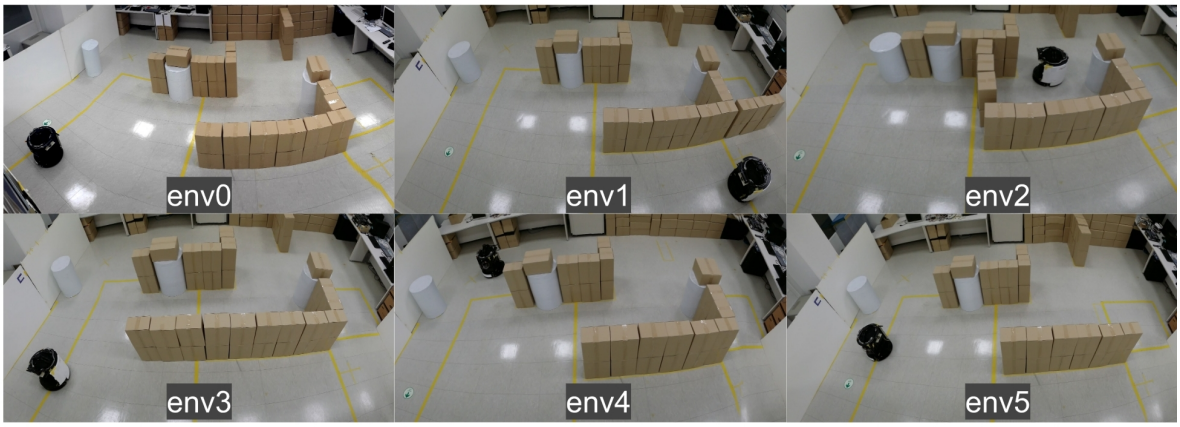
of 2 rooms as shown in 3(b). Similar results as that of JAY were observed (omitted for brevity).

5.4. gym-dal

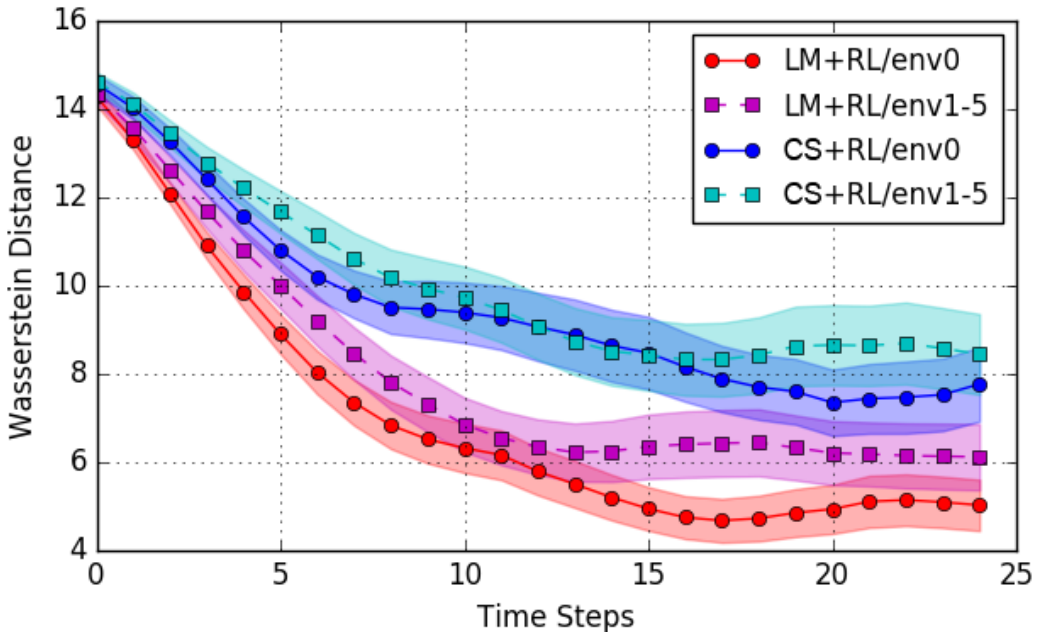
Most of the robotics research involves the extensive use of ROS and Gazebo environments which are convenient but they are not ideal platform for implementing learning algorithms especially reinforcement learning. To alleviate these problems, we are open sourcing our gym environment. We hope that this 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 real robot. We have also integrated the environment with various state of the art RL algorithms [**38**] like PPO [**63**], A2C[**49**], ACKTR [**83**].

5.5. Hierarchical Likelihood models

We present the experimental results of our 2 novel hierarchical likelihood models. From Fig. 6(a), we can observe that both weighted (where we multiply the likelihood of every cell



(a) The original and the 5 modified environments

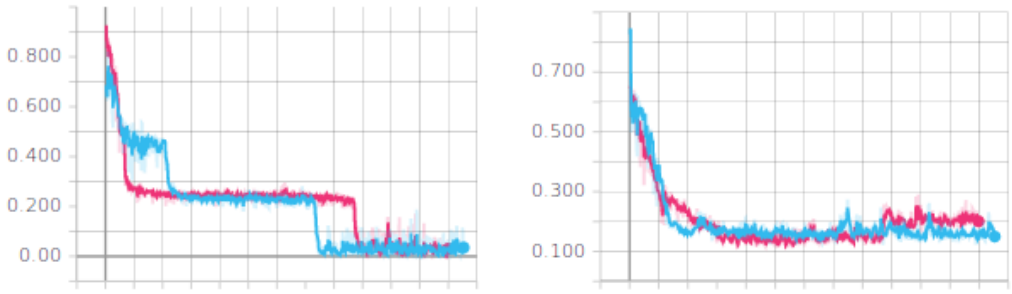


(b) Wasserstein distance between belief and true belief

Fig. 5.5. 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 true belief.

at higher level with the corresponding likelihood at 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 insensitive to hyper parameters are possible avenues for future research.



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

Fig. 5.6. blue: weighted HLE. red: unweighted HLE

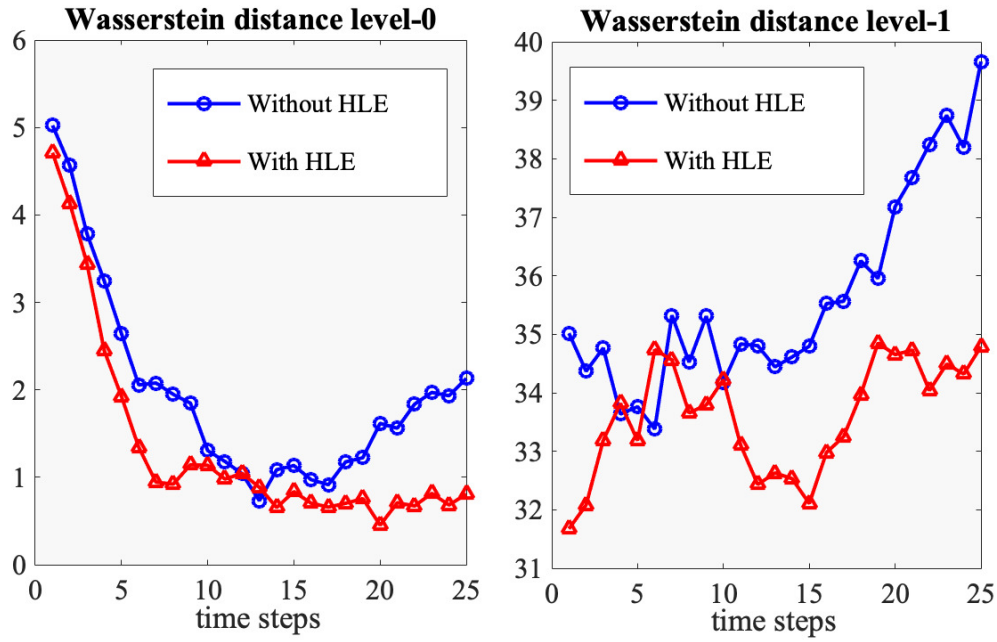


Fig. 5.7. Wasserstein distance between DAL's belief and ground truth belief at level-0 and level-1 when HLE (hierarchical likelihood estimate) is used (red) and when it is not used (blue). We can see that HLE (based on cosine similarity in this figure) indeed helps in controlling the motion errors.

Chapitre 6

Conclusion

In this work, we have started by formally defining the problem statements of localization and active localization and discussed the traditional approaches to these problems. After giving a thorough introduction to deep learning and reinforcement learning frameworks, we have presented a learning-based approach to active localization from a known map within the framework of the traditional method of Markov localization. Furthermore, to overcome the problem of accumulating motion errors, we proposed 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. We are also exploring to extend our approach from active localization to active SLAM setting, where the goal is also to construct the map the robot is operating upon than to just localize itself. The other interesting avenue is to extend it to particle filter setting instead of limiting it to grid based setting and further apply stein-based optimization techniques[**44**] for learning the likelihood model.

Bibliographie

[1] Noha Radwan Abhinav Valada et Wolfram Burgard : Deep auxiliary learning for visual localization and odometry. *In ICRA*, May 2018.

[2] Thomas Anthony, Zheng Tian et David Barber : Thinking fast and slow with deep learning and tree search. *NIPS*, abs/1705.08439, 2017.

[3] Tal Arbel et Frank P Ferrie : Viewpoint selection by navigation through entropy maps. *In ICCV*, 1999.

[4] Dzmitry Bahdanau, Kyunghyun Cho et Yoshua Bengio : Neural machine translation by jointly learning to align and translate. *In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[5] Tim Bailey et Hugh Durrant-Whyte : Simultaneous localisation and mapping (slam): Part ii state of the art.

[6] Joydeep Biswas et Manuela M. Veloso : Localization and navigation of the cobots over long-term deployments. *The International Journal of Robotics Research*, 32(14):1679–1694, 2013.

[7] Wolfram Burgard, Dieter Fox et Sebastian Thrun : Active mobile robot localization. *In IJCAI*, 1997.

[8] Luca Carlone, Jingjing Du, Miguel Efrain Kaouk Ng, Basilio Bona et Marina Indri : Active slam and exploration with particle filters using kullback-leibler divergence. *Journal of Intelligent Robotic Systems*, 75:291–311, 2014.

[9] Devendra Singh Chaplot, Emilio Parisotto et Ruslan Salakhutdinov : Active neural localization. *CoRR*, 2018.

[10] Hugh Durrant-Whyte et Tim Bailey : Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006.

[11] A. Elfes : Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57, 1989.

[12] Hans Jacob S. Feder, John J. Leonard et Christopher M. Smith : Adaptive mobile robot navigation and mapping. *IJRR*, 1999.

[13] C. Forster, M. Pizzoli et D. Scaramuzza : Appearance-based active, monocular, dense reconstruction for micro aerial vehicle. *RSS*, 2014.

[14] Dieter Fox, Wolfram Burgard et Sebastian Thrun : Active markov localization for mobile robots. *Robotics and Autonomous Systems*, 1998.

[15] Dieter Fox, Wolfram Burgard et Sebastian Thrun : Markov localization for mobile robots in dynamic environments. *J. Artif. Int. Res.*, 11(1):391–427, juillet 1999.

[16] Dieter Fox, Wolfram Burgard et Sebastian Thrun : Markov localization for reliable robot navigation and people detection. *In Sensor Based Intelligent Robots*, pages 1–20. Springer, 1999.

[17] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville et Yoshua Bengio : Generative adversarial nets. *In* Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence et K. Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[18] Omer Gottesman, Fredrik D. Johansson, Joshua Meier, Jack Dent, Donghun Lee, Srivatsan Srinivasan, Linying Zhang, Yi Ding, David Wihl, Xuefeng Peng, Jiayu Yao, Isaac Lage, Christopher Mosch, Li-Wei H. Lehman, Matthieu Komorowski, Aldo Faisal, Leo Anthony Celi, David Sontag et Finale Doshi-Velez : Evaluating reinforcement learning algorithms in observational health settings. *CoRR*, abs/1805.12298, 2018.

[19] Sai Krishna Gottipati, Keehong Seo, Dhaivat Bhatt, Vincent Mai, Krishna Murthy et Liam Paull : Deep active localization. *IEEE Robotics and Automation Letters*, 4(4):4394–4401, 2019.

[20] G. Grisetti, C. Stachniss et W. Burgard : Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, Feb 2007.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Deep residual learning for image recognition. *CoRR*, 2015.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren et Jian Sun : Identity mappings in deep residual networks. *In ECCV*, 2016.

[24] David Held, Sebastian Thrun et Silvio Savarese : Learning to track at 100 FPS with deep regression networks. *ECCV*, abs/1604.01802, 2016.

[25] Sepp Hochreiter et Jürgen Schmidhuber : Long short-term memory. *Neural Comput.*, 9(8):1735–1780, novembre 1997.

[26] Gao Huang, Zhuang Liu et Kilian Q. Weinberger : Densely connected convolutional networks. *CoRR*, 2016.

[27] Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra et Kilian Q. Weinberger : Deep networks with stochastic depth. *ECCV*, abs/1603.09382, 2016.

[28] Leopoldo Jetto et Sauro Longhi : Development and experimental validation of an adaptive extended kalman filter for the localization of mobile robots. *In IEEE Trans on robotics and automation, vol.15, No.2*, 1999.

[29] Rico Jonschkowski, Divyam Rastogi et Oliver Brock : Differentiable particle filters: End-to-end learning with algorithmic priors. *CoRR*, 2018.

[30] H. Kahn et T. E. Harris : Estimation of particle transmission by random sampling. *In Monte Carlo Method*, volume 12 de *Applied Mathematics Series*, pages 27–30. National Bureau of Standards, 1949.

[31] Péter Karkus, David Hsu et Wee Sun Lee : Particle filter networks: End-to-end probabilistic localization from visual observations. *CoRR*, 2018.

[32] Alex Kendall, Matthew Grimes et Roberto Cipolla : Convolutional networks for real-time 6-dof camera relocalization. *ICCV*, 2015.

[33] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley et Amar Shah : Learning to drive in a day. *CoRR*, abs/1807.00412, 2018.

[34] Ayoung Kim et Ryan M. Eustice : Active visual slam for robotic area coverage: Theory and experiment. *IJRR*, 2015.

[35] Hyeongwoo Kim, Pablo Garrido, Ayush Tewari, Weipeng Xu, Justus Thies, Matthias Niessner, Patrick Pérez, Christian Richardt, Michael Zollhöfer et Christian Theobalt : Deep video portraits. *ACM Trans. Graph.*, 37(4):163:1–163:14, juillet 2018.

[36] Thomas Kollar et Nicholas Roy : Trajectory optimization using reinforcement learning for map exploration. *IJRR*, 2008.

[37] Vijay R. Konda et John N. Tsitsiklis : On actor-critic algorithms. *SIAM J. Control Optim.*, 2003.

[38] Ilya Kostrikov : Pytorch implementations of reinforcement learning algorithms. GitHub repository, 2018.

[39] Rainer Kümmerle, Rudolph Triebel, Patrick Pfaff et Wolfram Burgard : Monte carlo localization in outdoor terrains using multi-level surface maps. *J. Field Robotics*, 2008.

[40] Gustav Larsson, Michael Maire et Gregory Shakhnarovich : Fractalnet: Ultra-deep neural networks without residuals. *In ICLR*, 2017.

[41] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard et Lawrence D. Jackel : Handwritten digit recognition with a back-propagation network. pages 396–404, 1990.

[42] Sergey Levine, Chelsea Finn, Trevor Darrell et Pieter Abbeel : End-to-end training of deep visuomotor policies. *CoRR*, 2015.

[43] Li Liu, Jie Chen, Paul W. Fieguth, Guoying Zhao, Rama Chellappa et Matti Pietikäinen : A survey of recent advances in texture representation. *CoRR*, abs/1801.10324, 2018.

[44] Qiang Liu et Dilin Wang : Stein variational gradient descent: A general purpose bayesian inference algorithm. *In Advances In Neural Information Processing Systems*, pages 2378–2386, 2016.

[45] David G. Lowe : Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, novembre 2004.

[46] G. L. Mariottini et S. I. Roumeliotis : Active vision-based robot localization and navigation in a visual memory. *In ICRA*, 2011.

[47] James Martens et Roger Grosse : Optimizing neural networks with kronecker-factored approximate curvature. *In* Francis Bach et David Blei, éditeurs : *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 de *Proceedings of Machine Learning Research*, pages 2408–2417, Lille, France, 07–09 Jul 2015. PMLR.

[48] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver et Koray Kavukcuoglu : Asynchronous methods for deep reinforcement learning. *In* Maria Florina Balcan et Kilian Q. Weinberger, éditeurs : *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 de *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[49] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver et Koray Kavukcuoglu : Asynchronous methods for deep reinforcement learning. *CoRR*, 2016.

[50] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra et Martin A. Riedmiller : Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[51] Ralph Neuneier : Enhancing q-learning for optimal asset allocation. *In* M. I. Jordan, M. J. Kearns et S. A. Solla, éditeurs : *Advances in Neural Information Processing Systems 10*, pages 936–942. MIT Press, 1998.

[52] Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester et Nati Srebro : Exploring generalization in deep learning. *In* I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan et R. Garnett, éditeurs : *Advances in Neural Information Processing Systems 30*, pages 5947–5956. Curran Associates, Inc., 2017.

[53] Michael Nielsen : Neural networks and deep learning. *Determination Press*, 2015.

[54] Edwin B. Olson : Real-time correlative scan matching. *In ICRA*, 2009.

[55] Sageev Oore, Geoffrey E. Hinton et Gregory Dudek : A mobile robot that learns its place. *Neural Computation*, 1997.

[56] Sinno Jialin Pan et Qiang Yang : A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, octobre 2010.

[57] Martin L. Puterman : *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1st édition, 1994.

[58] J. R. Quinlan : Induction of decision trees. *Mach. Learn.*, 1(1):81–106, mars 1986.

[59] Des Raj et Salem H. Khamis : Some remarks on sampling with replacement. *Ann. Math. Statist.*, 29(2):550–557, 06 1958.

[60] Sai Rajeswar, Sandeep Subramanian, Francis Dutil, Christopher Joseph Pal et Aaron C. Courville : Adversarial generation of natural language. *CoRR*, abs/1705.10929, 2017.

[61] N. Roy, W. Burgard, D. Fox et S. Thrun : Coastal navigation-mobile robot navigation with uncertainty in dynamic environments. *In ICRA*, 1999.

[62] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan et Philipp Moritz : Trust region policy optimization. *In* Francis Bach et David Blei, éditeurs : *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 de *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[63] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford et Oleg Klimov : Proximal policy optimization algorithms. *CoRR*, 2017.

[64] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan et Demis Hassabis : Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *Nature*, abs/1712.01815, 2017.

[65] Rupesh Kumar Srivastava, Klaus Greff et Jürgen Schmidhuber : Highway networks. *CoRR*, abs/1505.00387, 2015.

[66] Rupesh Kumar Srivastava, Klaus Greff et Jürgen Schmidhuber : Training very deep networks. *In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, pages 2377–2385, Cambridge, MA, USA, 2015. MIT Press.

[67] C. Stachniss, G. Grisetti et W. Burgard : Information gain-based exploration using rao-blackwellized particle filters. *In Proceedings of Robotics: Science and Systems (RSS)*, Cambridge, MA, USA, 2005.

[68] Richard S. Sutton : On the significance of markov decision processes. *In ICANN*, 1997.

[69] Richard S. Sutton et Andrew G. Barto : *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st édition, 1998.

[70] Richard S. Sutton, David McAllester, Satinder Singh et Yishay Mansour : Policy gradient methods for reinforcement learning with function approximation. *In Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[71] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke et Andrew Rabinovich : Going deeper with convolutions. *CVPR*, abs/1409.4842, 2015.

[72] Sebastian Thrun : Finding landmarks for mobile robot navigation. *In ICRA*, 1998.

[73] Sebastian Thrun, Wolfram Burgard et Dieter Fox : *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[74] Sebastian Thrun, Dieter Fox, Wolfram Burgard et Frank Dellaert : Monte carlo localization for mobile robots. *In ICRA*, 1999.

[75] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba et Pieter Abbeel : Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017.

[76] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba et Pieter Abbeel : Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, 2017.

[77] Simon Tong et Daphne Koller : Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, mars 2002.

[78] Sai Krishna G. V., Kyle Goyette, Ahmad Chamseddine et Breandan Considine : Deep pepper: Expert iteration based chess agent in the reinforcement learning setting. *CoRR*, abs/1806.00683, 2018.

[79] R. Valencia, J. Valls Miró, G. Dissanayake et J. Andrade-Cetto : Active pose slam. *In IROS*, 2012.

[80] Oriol Vinyals, Igor Babuschkin, Junyoung Chung, Michael Mathieu, Max Jaderberg, Wojciech M. Czarnecki, Andrew Dudzik, Aja Huang, Petko Georgiev, Richard Powell, Timo Ewalds, Dan Horgan, Manuel Kroiss, Ivo Danihelka, John Agapiou, Junhyuk Oh, Valentin Dalibard, David Choi, Laurent Sifre, Yury Sulsky, Sasha Vezhnevets, James Molloy, Trevor Cai, David Budden, Tom Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Toby Pohlen, Yuhuai Wu, Dani Yogatama, Julia Cohen, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Chris Apps, Koray Kavukcuoglu, Demis Hassabis et David Silver : AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. `https://deepmind.com/blog/alphastar-mastering-real-time-strategy-game-starcraft-ii/`, 2019.

[81] Mei Wang et Weihong Deng : Deep face recognition: A survey. *arxiv*, abs/1804.06655, 2018.

[82] Ronald J. Williams : Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, mai 1992.

[83] Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse et Jimmy Ba : Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, 2017.

[84] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu et Kaiming He : Aggregated residual transformations for deep neural networks. *CVPR*, abs/1611.05431, 2016.

[85] Zeyu Zheng, Junhyuk Oh et Satinder Singh : On learning intrinsic rewards for policy gradient methods. *CoRR*, 2018.

[86] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei et Ali Farhadi : Target-driven visual navigation in indoor scenes using deep reinforcement learning. *In ICRA*, 2017.

Annexe A

Les différentes parties et leur ordre d'apparition

In chapter-2, we introduce the notations used and classical formulations of passive and active localization problems. More specifically, in sec-2.1, we introduce the precise definition of the type of localization problem we are solving in this work. We then continued to develop notations and markov localization framework in sec-2.2 and then defined the active localization problem in sec-2.3 and the extension of markov localization to active setting in sec-2.3.

In chapter-3, we introduced the relevant concepts in deep learning and reinforcement learning. In sec-3.1, we gave a brief introduction to deep learning followed by its detailed working in sec-3.1.1 and relevant network architectures Resnet (sec-3.1.2) and densenet (sec-3.1.3). We then introduced the paradigm of reinforcement learning and MDP in sec-3.2 and explained the policy gradient methods (sec-3.2.1,sec-3.2.2,sec-3.2.3,sec-3.2.4,sec-3.2.5,sec-3.2.6) and intrinsic reward formulation (sec-3.2.7). We then discuss how DL and DRL can be used for passive localization (sec-3.3.1) and active localization (sec-3.3.2).

In chapter-4, we dive into our method building upon the formulations built in earlier chapters. In sec-4.1, we give an overview of our entire system followed by describing each of the individual parts: perceptual system (sec-4.2) and policy model (sec-4.3) including the application of domain randomization. Further, in chapter-5, we explain the experimental setup and discuss the results