IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

A Survey Of Typical Machine Learning Based Motion Planning Algorithms For Robotics

Datta lohith, student, Department electronics and communication
Hyderabad.

Abstract: The planning of collision-free motions among a set of obstacles is a fundamental task in robotics. Learning-based motion planning techniques have recently demonstrated notable benefits in resolving various planning issues in high-dimensional spaces and challenging environments. This article provides a summary of numerous learning-based approaches that Robot motion planning issues have been addressed using supervised, unsupervised, and reinforcement learning techniques. These learning-based approaches either make use of For some activities, humans have designed rewards, or they can learn from successful planning experiences. The traditional notion of motion planning and definitions linked to learning are presented in this article. The introduction of various learning-based motion planning algorithms and the fusion of traditional motion planning with learning techniques are well described.

1. Introduction

1.1 Robotics Background

In order for a robot to operate autonomously, it must be capable of interacting with its environment in an intelligent way, This implies that an autonomous robot must be able to capture information about the environment and then perform actions based on that information. A hypothetical robotic system can be dissected into four subsystems:

sensing → representation → planning → actuation

Although, the lines between these subsystems are often blurred in practice.

A sensor is the term given to any part of the robotic system that provides data about the state of the environment. Although the definition of a sensor is necessarily broad, the type of information provided by sensors can be broken into three main categories:

1. The world (terrain shape, temperature, color, composition)

2. The system and its relationship to the world (battery charge, location, acceleration)

3. Other concepts of interest (collaborator, adversary, goal, reward).

Any information available to the robot must be provided a priori or obtained on-line through sensor observations.

The representation is the method by which a robot stores and organizes information about the world. Simple representations may consist of a single value—perhaps indicating the output of a particular sensor. Complex representations may include high-level graphical models and/or geometric maps of the environment.

The planning subsystem (or planner) is responsible for deciding how the robot should behave, with respect to a predefined task, given the information in the representation. A planner might calculate anything from a desired speed/direction of travel to an entire sequence of actions.

Actuation is the method by which the robot acts on the environment. This may involve sending power signals to motors, servos, or other devices that can modify the physical relationship between the robot and the environment.

All four system components are interrelated. However, in the context of this paper, the relationship between the representation and planning subsystems is especially important. The structure and content of the representation define what kinds of decisions the planner is capable of making, and ultimately the set of action plans available to the robot. Conversely, a particular planning system may require a specific type of representation in order to function. Most of this chapter is devoted to these two subsystems.

1.1.1 Sensors

In the context of robotics, the term sensor is broad and ambiguous. It can be used to describe any device or module that is capable of capturing information about the world. Instead of trying to define exactly what a sensor is, it is perhaps more helpful to give examples of different kinds of sensors.

Active sensors glean information about the world by sending a signal into the world and then observing how information from that signal propagates back to the sensor. For instance, devices like radar, sonar, lasers, and lidar send a light or sound wave into the world, and then observe how it is reflected by the environment. Tactile sensors probe the environment physically, much like a human feeling their way around a room in the dark.

Passive sensors function by capturing information that already exists in the environment. This includes devices such as thermometers, accelerometers, altimeters, tachometers, microphones, bumper sensors, etc. Devices like cameras, infrared sensors, and GPS receivers are also considered passive sensors—although their assumptions about certain types of information can be violated (e.g. natural light and GPS signals seldom propagate into cavernous environments).

Sensors can sometimes be described as being either ranged or contact sensors. Ranged sensors capture information about the environment from a distance, and include devices like sonar, radar, cameras, and lidar. In contrast, contact sensors require physical contact with the part of the environment they are sensing, and include devices like thermometers, tactile sensor, strain gages, and bumper sensors.

It is also useful to make the distinction between grid-based (or image) sensors, and other types of sensors. Image sensors capture multiple (and often simultaneous) readings about a swath of the environment, while other sensors only capture information about a point or along a directional vector. Cameras are arguably the most common type of grid-based sensor. Each pixel represents a light value associated with a particular ray traveling through the environment. Similarly, a laser imaging sensor known as lidar assembles many individual laser readings into a spatially related collection of depth values. Theoretically, any collection of individual sensors can form an image sensor, as long as the spatial relationships between the individual sensors are known. Images are appealing because they provide an additional level of knowledge beyond an unorganized collection of individual sensor readings.

1.1.2 Representation

The representation subsystem decides what information is relevant to the robot's task, how to organize this data, and how long it is retained. Simple representations may consist if an instantaneous sensor reading, while complex representations may create an entire model of the environment and/or robot. It should be noted that using a complex representation is not a precondition for achieving complexity in the resulting robot behavior. It has been shown that robust and sophisticated behavior can be produced using simple representations of the environment and vise versa [Braitenberg, 1984]. However complex representations may allow a planning system to develop plans that 'think' further into the future. This can be advantageous because knowledge about intended future actions can decrease a system's susceptibility to myopic behavior.

In contrast, model-based (or proactive planners) create relatively detailed action plans. For instance, an entire sequence of movements or a high-level path from the robot's current position to a goal position. In other words, proactive planners assume the robot has enough information to know exactly what it would do in the future, assuming it can forecast all changes in environmental state. Current actions may also be influenced by what the system expects to happen in the future. For instance, a rover might temporarily move away from its goal location, in order to avoid hitting an obstacle in the future. Proactive planners generally require more complex representations such as graphical models of environmental connectivity, maps, etc.

1.1.3 Planning Methods that are not Path-Planning

The main point of this paper is to examine how machine learning is used in conjunction with a particular planning discipline called path-planning. Path-planning algorithms approach the planning problem by attempting to find a sequence of actions that, based on the representation, are likely to move the robot from its current configuration to a goal configuration. Path-planning is not the only planning framework available to a designer, and it is often used alongside other methods. Because a single planning technique is seldom used alone, path- planning is sometimes confused with other planning methods. Therefore, before discussing what path-planning is, it is helpful to outline what path planning is not

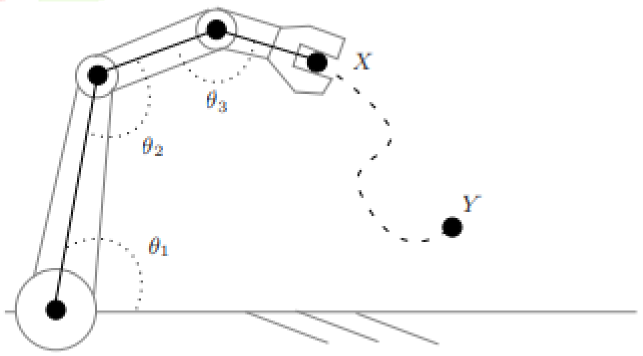

Figure 1: A mechanical arm with gripper at location X is told to move the gripper along the path (dashed line) to location Y . Inverse kinematics is used to calculate the arm angle functions θ1, θ2, and θ3 that accomplish this. Note that the system is under-determined, so there are multiple solutions.

Path-Planning: Algorithm

This section is devoted to path planning-algorithms that do not use machine learning. In order to find a path without resulting to trial-and-error, the representation must contain connectivity information about the different place it represents. This means that a 'recognizable locations' type of map cannot be used. On the other hand, topological maps, metric topological maps, and full metric maps are all valid.

Generic Graph-Search Algorithm

The first family of path-planning algorithms I will discuss are called graph-search algorithms. As the name implies, these algorithms perform path search through a graph. Graph search algorithms can be used directly on topological and metric topological maps, because these representations are essentially graphs. Assuming the world is deterministic (i.e. the same action always produces the same result, given a particular world state), many graph search algorithms will find an optimal path with respect to the representation. An entire subset of graph-search algorithms have been developed for the case when sampling-based methods or combinatorial methods are used in the representation.

Let v_i represent an arbitrary vertex in a graph and let V represent a set of arbitrary vertices. Each node has a unique identifier i. Two vertices v_i and v_j are the same if $i = j$ and different if $i \neq j$. v_{goal} is a set of goal vertices that represent acceptable termination states for the path, and v_{start} is a set of possible starting locations. G is the set of all vertices in the graph. Let $e_{i,j}$ represent an edge that goes from node v_i to node v_j . Edges between nodes can either be directed $e_{i,j} \neq e_{j,i}$ or undirected $e_{i,j} = e_{j,i}$. Most graph-serach algorithms do not specifically require an edge to be directed or undirected, but will only traverse directed edges in one direction (from node vi to node v_j). Without loss of generality, undirected edges can be though of as two directed edges, one going in either direction (the undirected edge $e_{i,j}$ accomplishes the same connectivity as the two directed edges ei,j and $e_{j,i}$). Therefore,

Let e_i be the set of all edges that leave vertex v_i . That is $e_{i,j} \in e_i$ if and only if $e_{i,j}$ exists. Node v_j is considered a neighbor of node v_i if $e_{i,j} \in e_i$. Finally, let E represent the set of all edges in the graph. Graph-search algorithms assume the existence of V, E, v_{goal}, and v_{start}.

During graph-search, an algorithm starts at v_{start} and attempts to find a path to v_{goal}, or vise versa, by exploring from node-to-node via edges. It is known as forward search when the search is conducted from v_{start} to v_{goal}, and backward search when the search is conducted from v_{goal} to v_{start}. Bidirectional search starts at both v_{goal} and v_{start} and attempts to connect the two searches somewhere in the middle. Multi-directional search starts at v_{goal} and v_{start} as well as other random or intuitive locations and attempts to link the searches together in such a way that a path is found between v_{start} and v_{goal}. When an optimal6 forward/backward search algorithm is used to create a bidirectional search algorithm, the resulting algorithm is also optimal. However, this does not extend to multi-directional search.

The relative order in which nodes are expanded can be used to create a tree-representation of the graph-search. This is called a search-tree. In forward or backward search the root(s) of the tree are at v_{start} or v_{goal}, respectively. In practice, it is common to use back-pointers to preserve the structure of the search-tree by creating a back-pointer from each node v_j to the earlier node vi from which v_j was discovered . In bidirectional search, there is a separate search-tree for both the forward and backward directions of the search, and in multi-directional search each search has its own tree. In bidirectional and multi-directional search, two trees are joined when a particular node is included in both search-trees.

Algorithm 1 displays pseudo-code for a generic graph-search algorithm. The algorithm starts by adding the goal nodes to the open-list (line 1). Next, while there are still nodes in the open-list, a node vj is chosen to be expanded (line 3). If vj is a start node, then the algorithm terminates successfully (lines 4-5). Otherwise, vj is closed by adding all of the nodes vi for which vj is a neighbor to the open-list (line 8). A back-pointer is created from vi to vj (line 9) so that the path can be extracted from the search-tree upon successful termination. If the list becomes empty then there are no possible paths from v_{start} to v_{goal}, and the algorithm returns failure at line 10.

1.1.4 Actuation

Once a plan has been created by the planning subsystem, the actuation subsystem is responsible for actually executing the plan. In general, this involves translating the plan into an instruction sequence that is compatible with the various devices (motors, serves, switches, etc.) that the robot uses to act on the environment. As with the sensing subsystem, the actuation subsystem is highly dependent on the specific hardware available to the robot. To ensure safety and overall relevance/usefulness, any plan sent to the actuation subsystem must respect the latter's constraints. Therefore, the constraints of the actuation subsystem must be considered when the planning subsystem is developed.

2.1 Machine Learning in Meta-Sensors

Simple machine learning ideas might be applied to raw sensor data to help improve its accuracy or to help optimize sensing equipment. However, the most common application of machine learning in the sensing subsystem is as part of a meta-sensor. meta-sensor uses software to provide higher-level data than might be expected from a simple sensor . Because a meta-sensor draws on data stored in the representation to help make its decisions, it could alternatively be considered part of the representation subsystem:

$$\text{sensor} \rightarrow (\text{meta}-\text{sensor} \leftrightarrow \text{representation}) \rightarrow \text{planning} \rightarrow \text{actuation}$$

Machine learning has been used in robotics to create meta-sensors from image space data in [Jansen et al., 2005, Happold et al., 2006, Konolige et al., 2006, Thrun et al., 2006, Erkan et al., 2007,Grudic et al., 2007,Ollis et al., 2007,Halatci et al., 2007].

In [Jansen et al., 2005] Gaussian mixture models are used to classify terrain as 'sand,' 'gravel,' 'grass,' 'foliage,' or 'sky' in image-space. Image features are pixel-color planes that have been adjusted to account for gamma correction in the camera. Training example labels are provided off-line by a human, and cross validation is used to determine the optimal number of Gaussians per model. A separate GMM is built for each type of terrain the robot is expected to encounter (e.g. desert, forest, marshland), and an additional meta-gausian-model is used to determine which environment the robot is currently in.

In [Happold et al., 2006] histogram methods are used on-line in image-space to learn a mapping from color to geometric classes. Class labels are provided from stereo disparity. More recent work by the same authors [Ollis et al., 2007] focuses on learning the probability that pixels are associated with the 'obstacle' class.

[Konolige et al., 2006] explore two different self-supervised frameworks in image-space. The first is a path detection technique that uses a Gaussian mixture model in conjunction with AdaBoost over decision stumps. At start up, the system makes the optimistic assumption that it is on a path. The terrain in front of the robot is sampled and used to create a GMM in image space to distinguish between 'path' from 'not path.' Next, all pixels in the image are labeled according to this classification. If the resulting label pattern has a path-like shape, then the robot concludes that it is, in fact, on a path. Finally, this labeling is used with AdaBoost to create a decision stump that determines 'path' vs. 'not path' for subsequent images. The system repeats this process after every meter of movement. The second method presented in [Konolige et al., 2006] is similar to the first, except that near-field stereo is used to provide AdaBoost with 'traversable' vs. 'obstacle' examples. AdaBoost creates a decision stump that is used to classify the remaining image and/or subsequent images.

[Thrun et al., 2006] outlines the system that won the DARPA Grand Challenge in 2005. The system uses a Gaussian mixture model to learn a mapping from color (red, green, blue) to traversability. Class labels are provided by on-board laser sensors. As new training examples are provided, new 'local' Gaussians are created and then added to the 'global' model. This is done by either modifying the model's previous set of Gaussians, or discarding them in favor of the new 'local' Gaussians. The former option is chosen over the latter if the mahalanobis distance between a new and old Gaussian is less than a predefined threshold.

In [Erkan et al., 2007] a neural net is trained off-line from log-files34 and used to extract low dimensional texture features from normalized YUV space image patches. A scaled image pyramid is also used to normalize the image-space appearance of near- and far-field information.

A hierarchical Gaussian mixture model is used in [Angelova et al., 2007] to classify terrain as belonging to one of many different classes (e.g. 'sand,' 'soil,' 'mud,' etc.). Features used include: average RGB colors, color histograms, and texture filters. Classification is performed in image space, and separate meta-classifiers are used for the near- and far-field. Training occurs off-line.

2.2 Machine Learning in the Representation

When a cell-based metric map is cast as a connected graph for the purposes of graph-search path-planning, it is expected that each edge has a cost associated with it.

$$\text{sensor} \rightarrow \text{map}-\text{features} \rightarrow \text{cost} \rightarrow \text{path}-\text{search}$$

In the previous section, I examined several techniques in which meta-sensors output class labels such as 'obstacle' vs. 'traversable terrain.' For graph search, these correspond to cost values of 1 and ∞, respectively. Meta-sensors may also output classes labels like 'lake,' 'field,' and 'road,' for which a notion of cost is less obvious. In this section, I examine how machine learning has been used to obtain cost from the latter type of class labels (as well as other map features).

$$\text{sensor} \rightarrow \text{features} \rightarrow \text{labels} \rightarrow \text{cost} \rightarrow \text{path}-\text{search}$$

The mapping between sensor/map data and cost often involves two or three steps—for instance, depth and color data from image space are mapped to height and vegetation data in Cartesian space, which are then mapped to cost either directly or indirectly via class labels.

$$\text{sensor} \rightarrow \text{image}-\text{features} \rightarrow \text{image}-\text{classes} \rightarrow \text{cartesian}-\text{features} \rightarrow \text{cartesian}-\text{classes} \rightarrow \text{cost} \rightarrow \text{path}-\text{search}$$

The particular information stored in a map is method dependent. It may include anything from raw data (e.g. color, height, slope) to derived features (e.g. texture, obstacle probability). All of the methods described in this section assume that map features are stored in a feature vector at each map grid. Depending on the method, training and test sets (used for supervised learning) can either be constructed directly from image-space features, or by projecting and accumulating this data in Cartesian space.

2.3 Learning the Representation

Learning the representation (i.e. the representation itself) is a different problem than using machine learning in the representation (previous subsection). In this subsection I will briefly outline the former. In my opinion, most of this topic consists of the research body known as simultaneous localization and mapping or SLAM. The basic idea can be summarized as follows: in the absence of robust localization sensor (e.g. GPS), a robot in unknown terrain must relay on other observations to both: (1) construct a model of the environment and (2) localize itself within this model. Most SLAM algorithms are applied in either an outdoor field-robotic setting or on indoor mobile robots. In either case, the relative positions of landmarks, with respect to each other and/or the robot, are used to recover the organization of the world.

2.4 Machine Learning in the Planning Subsystem

Given that the name of this paper is 'machine learning applied to robotic path-planning,' one might expect this to be the largest section of the paper. This is not the case (although it not the smallest either). Part of the reason for this, the most direct way to modify the behavior of a planning system is to alter the data in the representation. For example, it is relatively easy to apply black-box machine learning algorithms to the representation subsystem in order to generate more intelligent map features—and thus facilitate better overall performance. In contrast, it is difficult to improve the behavior of tried-and-true graph-search algorithms—especially when the latter have theoretical guarantees on optimality, completeness, and convergence.

3.1 Unsupervised Learning Based Motion Planning

In contrast to enormous supervised learning-based motion- planning algorithms, there exist few unsupervised learning frameworks for motion-planning problems. Sarker et al. [61] proposed a novel motion prediction network called PROM-Net, which learns to make visual predictions for robot motions from raw video frames in a completely unsupervised manner. Compared with supervised learning-based motion planners, the PROM-Net is lightweight and can be easily implemented, especially for platforms with limited computing memory. Inspired by RL, an unsupervised learning path planning algorithm is introduced, called Plan2vec . Plan2vec uses near- neighbour distances to construct a weighted graph and distills path-integral to extrapolate local metric for global embedding. Experimental results reveal that it can significantly amortize the planning cost and enhance reactive planning.

3.2 Reinforcement Learning Based Motion Planning

RL originated from optimal control and animal psychology inspired trial-and-error search [63]. There are mainly three types of RL approaches, value-based, policy-based, and actor-critic (AC). AC derives from policy-based approaches, which uses a critic to estimate the action-value function. For convenience and clarity, this section is divided into two subsections, motion- planning with value-based RL method and motion-planning with policy-based RL method. Many strategies improving the performance of the motion planning framework by RL have been realized in recent years. According to the role of RL playing in motion-planning, these methods can be roughly divided into two categories, End-to-end Solution and Module Solution. the motion-planning algorithm can be formulated as an MDP problem; thus, some methods map the motion-planning problems to MDPs and solve the MDPs directly. RL policy and the motion-planning algorithm interact with each other and work as a whole as a motion planner. When the RL policy works as a motion planner, it is classified as an End-to-end Solution. The other RL methods replace one or two of the components of the motion planner with RL policy, which is classified as a Module Solution.

3.3 Motion planning with policy-based RL method

Policy-based RL approaches are proposed to address the optimization problem under the context of POMDP. They optimise the stochastic policies in the form of probability function mapping state to action directly, which is different from selecting deterministic actions according to the value- action function learnt by a value-based RL method. This means that they are naturally applicable to exploration in high- dimensional or continuous action spaces because only a set of parameters of the policy needs to be learnt instead of the value function to express the entire action space. Thus, they have better convergence properties than those of the value-based method. As a research focus of policy-based approaches, the AC method normally comprises actor and critic networks. The actor learns the policy parameters to generate actions, in the direction given by the critic, that update the value function parameters to evaluate the reward of the actions. In policy optimization, the objective function is to maximise

$$J(\theta) = E_{\tau\sim\pi_\theta}[R(\tau)]$$

where πθ is the policy parameterised by θ and τ is the trajectory sampled according to πθ. R(τ) is the total reward of τ. The policy gradient is

$$\nabla_\theta J(\theta) = E_\tau\left[\sum_{t=0}^{T-1} G_t \cdot \nabla_\theta log\pi_\theta(a_t|s_t)\right]$$

where $G_t = \sum_{t'=t}^{T-1} r_{t'}$ is the return for a Monte-Carlo trajectory. Gt is the unbiased but noisy estimate of $Q^{\pi_\theta}(a_t, s_t)$. In AC, G_t is replaced by critic $Q_w(a_t, s_t)$, which is parameterised by w. Then the gradient function becomes.

$$\nabla_\theta J(\theta) = E_\tau\left[\sum_{t=0}^{T-1} Q_w(a_t, s_t) \cdot \nabla_\theta log\pi_\theta(a_t|s_t)\right]$$

where $\pi_\theta(a_t|s_t)$ is the actor policy and $Q_w(a_t, s_t)$ is the critic value.

Training time for the imitation task for motion planning can be reduced by applying the Generative Adversarial Imitation Learning (GAIL) method incorporated DDPG into the GAIL approach, thus generating expert demonstration trajectories. The RL method is utilised to learn search heuristics as a part of the planning algorithms on the basis of expert demonstrations or solved instances. The learning setting is provably capable of improving the efficiency of motion planner in highly dynamic environments. However, because of inadequate training data distribution near obstacles, training neural motion planning with imitation learning in high-dimensional domains may suffer from low precision and success rate. Therefore, RL becomes a promising tool to carry WANG ET AL. - 309 out active and sufficient exploration for finding a collision free trajectory. A DDPG method is proposed for motion planning, which is modified with reduced variance in the actor update. By utilising a known system transition function to estimate expected discounted future rewards, the actor network experiences low estimation errors in the policy gradient process. Meanwhile, learning from previous experiences, the agent will gradually become 'smart' to try recorded successful actions rather than randomly starting from scratch. Thus, the training time is shortened, and a success rate of near 1.0 is reached with the neural motion planner trained by DDPG motion-planning algorithm, inspiring researchers to combine deep RL-based methods with prevailing approaches for motion planning.

4. CONCLUSIONS

In this article, the learning-based robot motion-planning algorithms are discussed from four aspects: classical definition and learning-related definition, motion planning with supervised learning method, motion-planning with unsupervised learning method, and motion planning with the reinforcement learning method.

It serves as a survey for readers to understand the learning-based robot motion-planning algorithms conveniently. In the future, the development of learning-based motion- planning algorithms with high generalisation ability is an important research direction. Specifically, most of the current learning-based motion-planning algorithms are restricted to environments similar to those of the training data, thereby performing poorly when transferred to more complex conditions, or ones with larger differences.

Additionally, many methods cannot deal with large-scale environment maps. Therefore, it is necessary to develop algorithms that can work in different scale environments. In addition, as far as we know, the main difficulty that supervised learning-based methods face is the collection of fullscale data, while RL-based methods can be inefficient because of the necessity for robots to interact with environments. It is an interesting topic to combine the advantages of these two methods to reduce the negative aspects of each method.

Machine learning can be applied at many different places in the robot system, and there is no standard way it is used for path-planning. Although machine learning has been used inside path-planning algorithms themselves, this idea has not yet been widely embraced. Much of the reason for this is that many graph-search algorithms provide optimal or nearoptimal solutions with respect to the representation.

Thus, as the representation becomes more accurate with respect to the environment, graph search can produce solutions that are optimal or near-optimal with respect to the real-world. In practice, most designers have opted to embrace graph-search, and then use machine learning to make the representation and sensing subsystems more intelligent—indirectly improving path-planning with respect to the real-world. Machine learning has been used in the representation and sensing subsystems to create metasensors that provide the system with more useful information about the environment than raw sensor data and/or map features.

Other ideas have focused on training a graph-search cost-function from expert provided examples. Finally, the accuracy of the representation itself can be improved by calculating the most likely organization, given sensor observations.

References

1. Cheng, R., Shankar, K., Burdick, J.W.: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). In: Learning an Optimal Sampling Distribution for Efficient Motion Planning. IEEE, Las Vegas (2020)
2. Pérez-Higueras, N., Caballero, F., Merino, L.: Learning human-aware path planning with fully convolutional networks. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 1–5. IEEE, Brisbane (2018)
3. Ariki, Y., Narihira, T.: Fully Convolutional Search Heuristic Learning for Rapid Path Planners. arXiv preprint arXiv:1908.03343 (2019)
4. Ichter, B., Harrison, J., Pavone, M.: Learning sampling distributions for robot motion planning. In: Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 7087–7094. IEEE, Brisbane (2018)

5. Kumar, R., et al.: Lego: Leveraging Experience in Roadmap Generation for Sampling-based Planning. arXiv preprint arXiv:1907.09574 (2019)
6. Takahashi, T., et al.: Learning heuristic functions for mobile robot path planning using deep neural networks. In: Proceedings of the International Conference on Automated Planning and Scheduling, vol. 29, pp. 764–772. Berkeley (2019)
7. Aizerman, M., Braverman, E., and Rozonoer, L. (1964). Theoretical foundations of the potential function method in pattern recognition learning. Automation and Remote Control, 25:821–837.
8. Albus, J. S. (1975). A new approach to manipulator control: The cerebellar model articulation controller (cmac). Dynamic Systems, Measurement and Control, pages 220–227.
9. Angelova, A., Matthies, L., Helmick, D., and Perona, P. (2007). Fast terrain classification using variable-length representation for autonomous navigation. In Computer Vision and Pattern Recognition.
10. Bagnell, J. A., Ratliff, N. D., and Zinkevich, M. A. (2006). Maximum margin planning. In International Conference on Machine Learning.
11. Bagnell, J. A. and Schneider, J. G. (2001). Autonomous helicopter control using reinforcement learning policy search methods. In Proceedings of International Conference on Intelligent Robotics and Automation.
12. Bajracharya, M., Tang, B., Howard, A., Turmon, M., and Mathies, L. (2008). Learning long-range terrain classification for autonomous navigation. In International Conference on Intelligent Robots and Automation, pages 4018–4024.
13. Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. Annals of Mathematics and Statistics, 37:1554–1563.
14. Bellman, R. (1957). A markovian decision process. Journal of Mathematics and Mechanics,
15. Bellotto, N. and Hu, H. (2009). Multisensor-based human detection and tracking for mobile service robots. IEEE Transactions on Systems, Man, and Cybernetics, 39:167–181.
16. Billard, A., Calinon, S., Dillmann, R., and Schaal, S. (2007). Handbook of Robotics: Chapter 59, Robot Programming By Demonstration. MIT Press, Cambridge, MA.
17. Braitenberg, V. (1984). Vehicles: Experiments in Synthetic Psychology. MIT Press, Cambridge, MA.
18. Burges, C. J. C. (1998). A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery, 2:121–167.
19. Calinon, S. and Billard, A. (2007). Learning of gestures by imitation in a humanoid robot. In Imitation and Social Learning in Robots, Humans, and Animals:Behavioural, Social and Communicative Dimensions, pages 153–177.