The State of Robot Motion Generation

Kostas E. Bekris, Joe Doerr, Patrick Meng, Sumanth Tangirala

Computer Science Dept., Rutgers University, New Brunswick NJ 08901, USA
kostas.bekris@cs.rutgers.edu

Abstract. This paper reviews the large spectrum of methods for generating robot motion proposed over the 50 years of robotics research culminating in recent developments. It crosses the boundaries of methodologies, typically not surveyed together, from those that operate over explicit models to those that learn implicit ones. The paper discusses the current state-of-the-art as well as properties of varying methodologies, highlighting opportunities for integration.

Keywords: robot motion generation, task and motion planning, control, imitation learning, reinforcement learning, foundation models

1 Introduction

The robotics community is grappling with a critical question. Will the emerging set of data-driven methods for generating robot motion supersede the traditional techniques as access to robot motion data increases?

In this context, this paper reviews methods for robot motion generation, which are classified into those that operate given an explicit model vs. those that implicitly learn one from data. Explicit models can correspond to analytical expressions for the world geometry and dynamics or an explainable, numerical approximation in the form of a simulator. Motion generation given explicit models is rather mature and methods are being deployed on real systems, such as autonomous vehicles and industrial manipulators. At the same time, there is increasing excitement for data-driven methods, which have been demonstrated to perform complex tasks, such as dexterous manipulation and unstructured locomotion. These methods often do not depend on explicit models. Instead, they learn implicit representations, which are stored in the internal parameters of machine learning models.

Given the challenge of comprehensively reviewing the vast amount of work in this area across disciplinary boundaries, the focus is on breadth rather than diving deeply into specific methodologies. Similarly, the focus is on principles that are applicable across robotic platforms instead of techniques that are specific to certain hardware configurations.

The paper concludes with a discussion regarding the properties of the various robot motion generation methods. It argues that integrative approaches and closer interactions between different sub-communities can help develop more robust, safe solutions that can be reliably deployed at reasonable costs and human engineering effort.

2 Motion Generation given Explicit Models

Figure 1 classifies methods that operate over an explicit model. **Motion planning** methods generate safe nominal paths/trajectories to a goal given a fully-observable world model. **Task and motion planning** extends the principle

to tasks that require sequencing multiple goals. Such solutions can be executed open-loop if the underlying model is accurate. Robot models, however, are imperfect, resulting in failures upon execution. Given this challenge, **planning under uncertainty** methods aim to compute robust policies to disturbances that can be modeled. Alternatively, **control and feedback-based planning** methods tightly integrate perception and motion generation so that the robot dynamically reacts to deviations from desired behavior given observations.

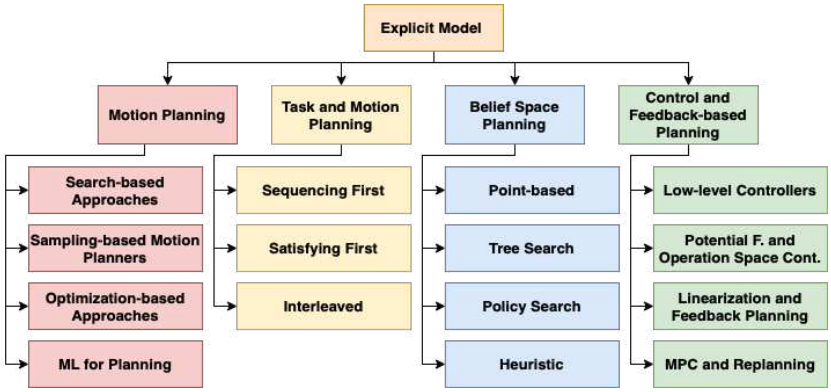

Fig. 1. Robot motion generation methods that operate over an explicit model.

2.1 Motion Planning

Motion planning aims to identify a path with minimal cost, such as the shortest path or the fastest trajectory that brings a robot to a desirable goal state, without undesirable collisions, given a fully-observable world model.

Search-based Approaches: Uninformed search methods, such as Uniform Cost Search (UCS) or Dijkstra's algorithm [1], can compute optimal paths over a discrete representation of the state space in the form of a grid or a graph given a cost function. Informed alternatives, such as A* [2], utilize heuristic cost-to-go estimates from each state to accelerate solution discovery and can still return the optimum solution over the discrete representation for an admissible/consistent heuristic. Due to the comprehensive nature of search and the discrete representation, search methods suffer from the curse of dimensionality, i.e., the possible states to be explored grow exponentially with dimensions, rendering them computationally infeasible for many robotics problems given naïve discretizations. There have been many successful applications of search methods in robotics, however, such as planning for autonomous vehicles [3] and single or dual arm planning [4].

Sampling-based Motion Planners (SBMPs): Sampling provides graph-based representations for searching the collision-free subset of a robot's state space in a more scalable manner than grids. The Probabilistic Roadmap Method (PRM) [5] samples collision-free configurations as nodes of a roadmap, and collision-free local paths define the edges. The roadmap can then be used to solve multiple queries via search. For specific queries, the Rapidly Exploring Random Tree (RRT) [6] generates a tree data structure rooted in the robot's start state to quickly explore the free state space until it reaches the goal's

vicinity. RRT does not require a "steering function", i.e., the ability to perfectly connect two robot states, a primitive required by other methods, allowing it to deal with dynamical systems where "steering functions" are unavailable. PRM and RRT are provably suboptimal, and asymptotically optimal variants (PRM*, RRT*) [7] guarantee that the discovered paths converge to optimal ones as sampling progresses. Recent planners [8] [9] also achieve asymptotic optimality for kinodynamic problems. SBMPs have been applied to autonomous driving [10] and manipulation [11]. The Open Motion Planning Library (OMPL) provides software implementations for most SBMPs [12].

Optimization-based Approaches: The above methods are comprehensive and aim to explore the entire feasible state space, which can lead to increased solution times. Moreover, their basic instances don't utilize gradient information. The alternative is to locally optimize paths or trajectories for robotic systems given an objective function under physical or operational limits, such as collision avoidance and control bounds. Covariant Hamiltonian Optimization for Motion Planning (CHOMP) [13] demonstrated this principle by iteratively optimizing paths by reducing collision and trajectory costs. TrajOpt [14] utilizes sequential convex optimization and progressively ensures that each iteration produces feasible and collision-free trajectories. k-Order Markov Optimization (KOMO) [15] treats trajectory optimization as a sparse nonlinear program and tries to address high-dim. problems by leveraging sparsity in dynamics and constraints. Factor graphs, a graphical optimization tool rooted in state estimation, and least-squares optimization can be also used for trajectory optimization [16]. A recent approach, the Graph of Convex Sets [17], combines optimization and SBMPs. It builds a graphical structure where nodes are convex regions of the free space, and optimization finds a trajectory over the set of nodes that connects the start with the goal. When optimization techniques work, they tend to find high-quality solutions fast. They can suffer, however, from local minima, which arise from the nonlinear and non-convex nature of robotics problems.

Machine Learning (ML) for Planning: ML can be used to improve the computational efficiency of planning [18]. Some approaches focus on planning components, such as effective sampling [19], avoiding collisions [20], or distance metrics [21]. ML can also determine which combination of methods is best suited for a particular problem [22], or when a solution is not feasible [23]. *Neural Motion Planning (NMP)* [24] employs neural networks to approximate a planner's operation typically given data from a simulator. An encoder processes environmental data, like point clouds, to create a compact latent space representation. Then, a planning network predicts the robot's next configuration based on its current state, the goal state and the encoded environment.

2.2 Task and Motion Planning (TAMP)

TAMP methods target long-horizon and multi-step robotic tasks, which may include moving through a sequence of goals or manipulating the environment [25]. TAMP methods typically define low-level operators with motion constraints and high-level logical relationships between the operators. Operators can contain hybrid discrete and continuous parameters, motion constraints, preconditions,

and effects, which make use of manually defined lifted variables. Planning is then performed via search across these lifted states. Possible state transitions are operators with satisfied preconditions, resulting in a sequence of operators called a plan skeleton. The hybrid parameters in the plan skeleton (e.g., start and goals) must be solved along with the low level operator trajectories. TAMP techniques can be categorized by the order they sequence and satisfy operators [25]: **Sequencing first** [26] defines a plan skeleton, i.e., a sequence of operators without satisfying their preconditions; this can cause frequent infeasible plans due to the lifted variables not being descriptive enough of underlying constraints. **Satisfying first** trajectories for operators, then planning with them, can solve this problem, but can spend time creating useless satisfied operators due to not having a plan skeleton to direct search [27]. **Interleaved** sequencing and satisfying methods can provide a mix of both by checking low-level information for feasibility during task planning [28]. TAMP critically relies on engineering to define preconditions and effects that properly characterize operator behavior and expressive lifted variables. Basic TAMP approaches also struggle under partial observability and uncertainty, which is the focus of recent efforts [29].

2.3 Belief Space Planning

The above methods assume a deterministic, perfect world model. Sensing noise and inaccurate execution introduces uncertainty, however, and the need to generate robust motions to such disturbances. If these noise sources can be modeled probabilistically, (Partially Observable) Markov Decision Processes (PO)MDPs provide a formulation to reason about uncertainty. A Markov Decision Process (MDP) consists of a set of states S, set of actions A, transition probabilities between states $T(s_{t+1} \mid s_t, a_t)$, and a reward function $R(s_t, a_t, s_{t+1})$. (PO)MDPs employ belief distributions for the problem representation, i.e., probability distribution over states given a set of observations O. They can be integrated with Bayesian state estimation (e.g., Kalman or particle filters) that return such beliefs. Solutions to (PO)MDPs are not nominal paths but policies that map beliefs to actions. Due to the consideration of uncertainty, computing an exact solution to a (PO)MDP is often computationally intractable [30], especially given the continuous state and action spaces of robotics.

This has motivated approximate solutions. The Successive Approximations of the Reachable Space under Optimal Policies (SARSOP) [31] applies **point-based** value iteration and offline sampling to focus on representative beliefs and determine the best action given the sampled set. Determinized Sparse Partially Observable Trees (DESPOT) [32] employ **tree search** online to compute the optimal action. Online methods can also perform **policy search** by using a controller in a limited search space, increasing scalability but not bounding solution quality [33]. **Heuristic** rules or assumptions can reduce complexity by ignoring long-term consequences and focus on immediate gains or most likely outcomes. For instance, Pre-image Back Chaining constructs state-action sequences (pre-images) leading backward from the goal [34]. Generalized Belief Space (GBS) planning dynamically adapts to ongoing sensing updates [35].

2.4 Control and Feedback-based Planning

Instead of explicitly modeling uncertainty, control tightly integrates state estimation and motion generation in a closed-loop. Thus, it defines policies that are reactive to different possible outcomes that may be observed upon execution.

Low-level Controllers: *Proportional Integral Derivative (PID)* control and related tools are simple feedback strategies that are robust once tuned. They are ubiquitous for tracking desirable robot controls from higher-level motion generation processes. *Path Tracking Controllers* dynamically select controls given the latest state estimate to minimize path deviation. These controllers, however, are myopic and typically neither reason about the desired goal or obstacles.

Potential Functions and Operational Space Control: *Potential functions* [36] define attractive fields towards the goal and repulsive ones that push away from obstacles. Moving along the negative gradient of the sum of these fields provides the motion vector. Some engineering is needed to tune the parameters of the potentials. In complex environments, the corresponding potential may have multiple minima and goal convergence is not guaranteed. *Navigation functions* [37] are smooth and ensure a single minimum at the goal, but they are more complicated to design and can only be constructed for specific environments (i.e., sphere and star worlds). Such control policies do not have to be defined directly in the robot's state space. *Operational Space Control* [38] defines such control laws in lower-dim. task spaces. For instance, if the task involves the robot's end-effector, a control law is defined to move it towards a desired goal unencumbered by other robot constraints. Additional control laws can be defined so that links avoid collisions. The corresponding motion vectors for the individual links are then mapped and integrated to a state space motion for the robot via the pseudo-inverses of the robot's Jacobian matrix, which relates robot joint velocities to link velocities through a linear transformation parameterized by the joint states. This allows for *multi-level hierarchical control* [39], where a hierarchy can be imposed over constraints, operational tasks, and soft objectives. Then, lower priority objectives are solved in the null space of higher priority ones once projected to the state space. These strategies rely on precise robot models and high-quality sensing to provide accurate feedback on robot state.

Linearization and Feedback-based Planning: Principles of linear control can be applied for robot motion generation. The *Linear Quadratic Regulator (LQR)* provides an optimal solution for linear time-invariant systems given a quadratic cost function as a control law of the form $u(t) = -K \cdot x(t)$, where $x(t)$ and $u(t)$ are respectively the system's state and the control to be applied at time t. Most robots, however, are nonlinear and time-varying, while tasks involve complex, non-quadratic objectives. *Feedback linearization* locally transforms nonlinear systems into equivalent linear ones so that linear control laws can be applied. It can be used for tracking states x_d along a desired trajectory by minimizing the error $e = x(t) - x_d(t)$. Given the linearization, these solutions tend to work in the vicinity of the desired goal/trajectory. They can be ineffective when the feedback-linearized system behaves very differently from the original nonlinear system. To expand the set of initial conditions from which the goal can be reached, *sequential composition* of such feedback policies [40] can

give rise to hybrid controllers that sequentially switch between them. *LQR-Trees* [41] apply sequential composition by combining linear control and SBMPs. They use control verification to evaluate the region of attraction (RoA) of local LQR controllers. They expand a tree backwards from the goal and sample controllers that bring the robot to the RoAs of controllers that lead to the goal. They have been applied on dynamic environments [42] and for systems with dynamics [43].

Model-Predictive Control (MPC) and Replanning: MPC is a simple but powerful feedback strategy that aims to solve fast and repetitively finite horizon optimization problems given the latest state observations [44]. MPC executes the initial portion of the motion and then updates given the latest observation. During each step, MPC uses the system's model to predict future behavior and makes control decisions that minimize a cost function at the end of the finite horizon, while adhering to constraints, such as collision avoidance. Initial MPC approaches used analytical methods to predict future states, and Linear MPC [45] utilizes linear analytical models or approximations. Various Non-Linear MPC (NMPC) variants have been proposed [46]. Shooting MPC is a common NMPC approach that employs numerical methods, such as guessing an initial set of controls and then locally optimizing to minimize the cost function [47]. It is applicable across robotic systems for bridging the model gap. At the same time, shooting MPC involves numerous parameters that require tuning to achieve good performance [48]. It is also possible to replan using longer horizon planners, where it is important to minimize computational costs so as to achieve tight feedback. For instance, D* Lite [49], a replanning variant of A*, updates the path it initially creates to adjust to changes in the workspace. Similarly, replanning versions of SBMPs reuse prior computations to ensure the robot reacts swiftly to changes while maintaining safety [50], including for systems with dynamics [51].

3 Data-driven Motion Generation with Implicit Models

Figure 2 presents a classification of data-driven robot motion generation methods that implicitly learn a model. **Learning from demonstration** methods employ supervision, where a robot is tasked to best replicate the demonstrated behavior. Alternatively, **reinforcement learning** learns to make decisions by experiencing rewards or penalties after interaction with the environment. **Cross-task learning** transfers knowledge from an existing solution to a new task or adapts it dynamically. Finally, **large models**, given access to significant data, allow pre-trained ML models to be fine-tuned and deployed in diverse domains.

3.1 Learning from Demonstrations

Imitation learning mimics the decision-making process given demonstration data.

Behavior Cloning (BC) trains a model in a supervised manner given a dataset of demonstrations to provide a motion policy. Successful applications of BC have enabled robots to learn complex behaviors [52][53]. The process starts with data collection, which includes capturing sensory inputs and corresponding control actions for a target task. An appropriate ML model architecture (e.g.,

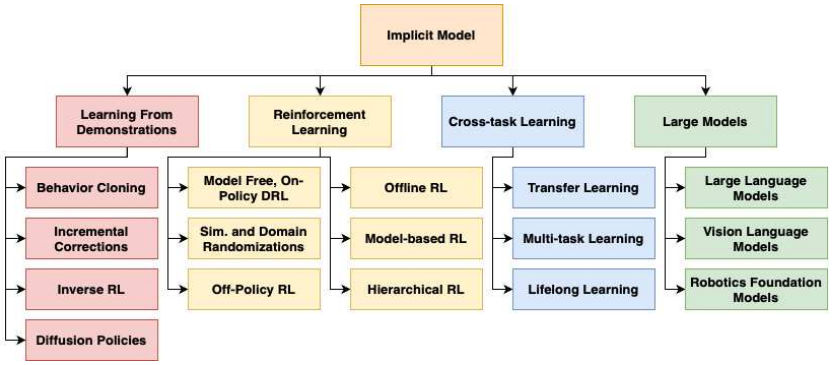

Fig. 2. Robot motion generation methods that operate over an implicit model.

Convolutional Neural Networks (CNNs) for visual data or Recurrent Neural Networks (RNNs) for sequential data) is trained using supervision to minimize a loss function, such as mean squared error. Gaps between the demonstration and the execution setup, as well as noise and stochasticity, can cause a compounding distributional shift between training and testing [54]. Noise modulates the perception of the true state and can cause erroneous actions from the BC policy. Stochastic state transitions can move the process from an in-distribution state to an out-of-distribution (OOD) one. These effects can compound and make the policy increasingly generate improper actions bringing the robot's state further OOD. Furthermore, BC is sensitive to engineering decisions, such as observation space, hyperparameters, and recurrent information [55]. These shortcomings motivate more sophisticated approaches.

Incremental Corrections: The DAgger algorithm [56] focuses on decision making in OOD states given the available demonstrations. A dataset of rollouts is collected by executing the BC policy. Then, an expert, such as a human or a planner, is queried to provide the proper actions at the states encountered in this dataset, and the policy is retrained. This process is iterative, as new OOD states will be encountered after each retraining. DAgger heavily relies on an expert to provide the corrections. SafeDAgger provided a more query-efficient extension [57]. It first uses a safety policy to predict the error of the learned policy. This safety policy selects only a small subset of training examples to be collected.

Inverse Reinforcement Learning (IRL) focuses on extracting the problem's underlying dense reward function given demonstrations to achieve improved generalization relative to BC [58]. This reward function can then be used to extract the policy provided an MDP problem formulation. Initial works [59] modeled the reward as a linear combination of input features, and the weights of this linear combination are obtained by solving an optimization problem. A significant challenge of IRL, however, is the difficulty in scaling to high-dim. systems, which recent approaches attempt to mitigate [60].

Diffusion Policies: Demonstrations commonly include multi-modal behaviors, which multi-layer perceptrons (MLP) may find difficult to express since they are trained via mean squared error (MSE) that averages the demonstrated actions resulting in erroneous choices [61]. In compuater vision, diffusion models

have become popular for generating images by expressing multi-modal distributions [62]. This motivates diffusion processes as implicit models for BC, where Diffusion Policy [63] has outperformed alternatives, such as LSTM-GMM [64] in learning from demonstrations. Inference speed of diffusion policies is a challenge for robot motion generations [65] and is an active area of research.

3.2 Reinforcement Learning (RL)

One area of robotics that has garnered interest [66] is Reinforcement Learning (RL), where a machine learning model predicts the Q function of an MDP and the corresponding policy. It has been applied across tasks, e.g., grasping [67], locomotion [68], and assembly [69].

Model-free, On-Policy RL: Standard RL gathers data from interaction with the environment, where Policy Gradient (PG) methods generate a batch of full trajectory data, compute the gradient of each state transition, and scale each gradient by the respective discounted return. PGs update the policy and repeat the process towards maximizing the expected cumulative reward. Proximal Policy Optimization (PPO) [70] is a PG method that leverages a value function, which is the average discounted return from a given state. These are on-policy methods, i.e., they only use data collected from the most recent policy. Exploration can be achieved through the periodic execution of random actions (epsilon greedy), injecting Gaussian noise to policy outputs [71], or using a stochastic policy for entropy maximization [70].

RL faces *multiple challenges* in robotics: (i) sample inefficiency: Policies need a lot of data to train and take significant training time. Real robot data, however, can be slow, expensive, and unsafe to collect [72]. (ii) instability: Policies can be inconsistent across training sessions due to poorly designed rewards, exploration strategies, learning rate parameters or learning error from neural networks. (iii) reward engineering: Simple rewards are sparse, i.e., they assign a 0 reward everywhere except at the goal. While desirable, sparse rewards often do not find a successful behavior in reasonable time [72]. This motivates dense rewards to guide exploration, which require manual engineering. While dense rewards can improve training time and stability, they can also lead to wrong behaviors, if they do not match the true task objective. (iv) long horizon tasks: The state space to be explored for such tasks is even larger making the propagation of rewards across subproblems difficult.

The above challenges have motivated multiple RL variations in robotics:

Simulation and Domain Randomization: Simulators, which are explicit world models, are useful for addressing sample inefficiency by generating data without real-world experiments. They suffer, however, from the model gap issue. Even so, it may still be possible to create good policies via *domain randomization (DR)* [73], which can partially compensate for imperfect models during sim-to-real transfer. DR results in behaviors that have the highest expected reward across a variety of underlying dynamics and perception errors.

Off-policy RL Sample inefficiency motivates data reuse. Off-policy RL stores the expected discounted future reward of any in-distribution state-action tuple on a Q function, which is used to train the policy π. To update the parameters

θ of the Q function for a given state transition (s, a, r, s') (state, action, reward, next state), the immediate reward r and future rewards $Q(s', \pi_\phi(s'))$ are used to define the loss function: $L(\theta) = E(Q_\theta(s, a) - r + \gamma \cdot Q_\theta(s', \pi_\phi(s')))^2$. This allows off-policy RL algorithms, such as TD3 [74] and SAC [75], to be more sample efficient than online RL. Off-policy RL, however, is complicated by the instability of neural network learning, especially because errors compound while learning Q functions [76]. A way to mitigate instability is to use target Q functions to query for future rewards [74][75][76]. The target Q functions average their weights slowly with the learned Q function's weights. Given that the policy is defined as $\pi^*(s) = \arg\max_a Q(s, a)$, errors in Q function learning, which overvalue actions, are propagated rapidly. This is called an overestimation bias [74]. One way to mitigate overestimation bias to is use a clipped, double Q function, an ensemble of two Q networks coupled with target Q networks, that take the lower value of the two when queried for future rewards to underestimate Q values when uncertain [74][75]. Hindsight Experience Replay (HER) [77] aims for sample-efficiency in multi-goal problems given only sparse rewards. It relabels the end or intermediate states of executed trajectories as the desired goal allowing to train over all available experiences.

Offline RL Offline RL is a subcase of off-policy RL, which doesn't interact online with the environment to modulate the replay buffer. Instead it uses offline data, focusing on sample efficiency. Offline RL aims to improve the optimality of suboptimal demonstrations by patching together data towards the most optimal solution rather than imitating the data [78]. Additionally, it can handle multi-modal demonstrations as it can solve for the single most optimal action for each state. The policy π used for updating the Q function is parameterized via a function approximator. This can cause π to choose actions that are out-of-distribution (OOD). The Q function queried with the OOD state-action pair will output an untrained value, which can be an erroneous large reward that can propagate and degrade performance [79]. In online RL, when an OOD state-action pair is overvalued, π will favor the action and gather data on the OOD state-action pair, which will remedy the error. IQL [80], which is an offline RL method, mitigates distributional shift by using only state-action pairs from the dataset. Instead, CQL [81] employs a conservative estimate on Q values.

Model-based RL learns a model of the environment, which is then used to roll out the policy to generate training data offline. Relative to a simulator, the learned model can operate over a more compact state representation, run faster, and can be queried for any given state. The major drawback is that RL may exploit errors in the learned model. MOPO [82] aims to mitigate these effects by detecting when the query to the model is out of distribution.

Hierarchical RL (HRL) focuses on long horizon tasks, similar to TAMP (Section 2.2). It temporally abstracts high-level actions that correspond to low-level learned policy. The high-level actions can target exploration to better cover the space and more easily assign rewards [83]. Using random actions to search is not effective in long horizon tasks and guidance for their generation is critical. Long horizon tasks remain rather challenging for RL approaches.

3.3 Cross-task Learning

Demonstrations from related or earlier tasks can help bootstrap or guide RL.

Transfer Learning methods exhibit high variability. One approach bootstraps RL by using an adaptively-weighted auxiliary term in the loss function of PPO to increase action similarity for the learned policy against the demonstrated one from a task with similar MDP [84]. It is also possible to pre-train multiple tasks with offline RL given a single task-conditioned policy [85]. Online, the approach fine-tunes both the conditioning parameter and policy to automatically reset tasks for autonomous real-world training. For transferring information across heterogeneous MDPs, researchers have mapped states and actions across MDPs [86] and transfered useful representations across domains [87].

Multi-Task Learning trains many tasks in parallel while sharing information across tasks to accelerate training and improve generalization [88]. Methods focus on how to select parameters to sharing so as to effectively transfer learned representations between tasks [89].

Lifelong Learning emphasizes learning a new task in a sequence by transferring information from previously learned ones without forgetting how to solve them. Retaining previous task information can be done by mixing previous data with new ones during retraining [90]. Functionally composed modular lightweight networks have been proposed to learn a large variety of combinatorially related tasks to solve novel combination tasks in a zero-shot manner [91].

3.4 Large Models

Large Language Models (LLMs) pretrained on internet-scale data have opened new avenues for robot motion generation. LLMs are capable of semantic reasoning and planning, making them candidates for extracting task specifications and creating action models for task planning [92]. LLMs have also been used to improve existing action models to handle failure cases [93]. These methods often do not generate constraints for low level motion in the operators as in TAMP. Furthermore, they have been used in conjunction with evolutionary optimization as code generators for defining rewards, which can be used to acquire complex skills via reinforcement learning [94].

Vision Language Models (VLMs) integrate information from visual input and language. Saycan [95] integrates a visual affordance model, which evaluates possible robot actions, with a language model that interprets the user's commands and generates high-level action plans. VLMs have also been used for autonomous generation of demonstration data over diverse tasks [96].

This lead to vision-language-action models (VLAs) or **Robotics Foundation Models**, i.e., large pretrained models that map actions to sensing data and language specifications. They can be fine-tuned for specific tasks to provide improvements in training time and generalization over trainingfrom scratch. OpenVLA [97] is such a model trained on a large robotic manipulation dataset [98] and utilizes large pretrained vision encoders and language models. It has been argued that it performs well on in-distribution tasks and robotic embodiments, while it can be fine-tuned for novel tasks and robotic embodiments.

4 Perspectives on Robot Motion Generation Research

Promises and Pitfalls of the Explicit Model Approach: These methods have a long history and can reliably solve various challenges today. For instance, planning collision-free motions for industrial arms is reliably addressed today at high speeds. Integration with state estimation allows mobile robots to execute navigation in semi-structured domains reliably. Challenges arise where model or state estimation reliability is low. Examples of such setups often involve the presence of complex contacts and partial observability, such as the manipulation of previously unknown objects in clutter, locomotion over uneven terrains, navigation at high speeds or in dynamic, unstructured environments.

For long-horizon tasks, TAMP methods require an engineer to encode possible pathways and concepts before symbolic reasoning, which often involves expensive combinatorial reasoning. These methods are limited to the variables defined and cannot easily anticipate changes from what has been programmed. While belief-space planning targets noise and partial observability, it requires significant computation and access to accurate models of uncertainty, which are not always available. At the same time, feedback-based solutions do find applicability in real-world domains, either via MPC or hierarchical control strategies but fully understanding the conditions under which a controller can solve a problem is an active challenge.

Promises and Pitfalls of the Implicit Model Approach: The progress in ML promises effective data-driven motion generation methods that can access prior experience and do not require an engineered model or even accurate state estimation. They have unblocked perception tasks, such as object detection and estimation, which are often prerequisites for robust motion generation. Learning from demonstration, especially with diffusion processes that allow reflecting multi-modal distributions, can achieve impressive results exactly in tasks that the traditional, explicit model approaches face challenges with, such as dexterous manipulation and locomotion. This is true, however, as long as the setup upon execution resembles the demonstration setup, thus limiting generalization. The various versions of reinforcement learning bring the promise of broader generalization, and there have been many successful demonstrations of learning skills for robotic tasks via RL, though sample inefficiency still remains a bottleneck for achieving highly accurate solutions across a wide set of initial conditions.

These limitations have motivated the robotics community to pursue the direction of collecting more data for robotics problems in lab environments, which may be diverse across embodiments and tasks [98], towards the objective of mimicking the success of foundation models in language and vision challenges. While this direction should be pursued, it is not clear that it is possible to collect internet-scale demonstration data that will allow learning robust enough policies that cover the space of possible tasks that robots need to solve in novel, unstructured, and human environments. Furthermore, predicting when the resulting solutions will be successful is challenging, which is a significant concern, as failures in robotics can cause physical harm.

Integrative Directions: There is promise in the integration of solutions. For instance, data-driven methods training can benefit from simulation, where the explicit model approaches act as the demonstrators under full observability. The data-driven methods can learn to map sensing data to the actions demonstrated by the planners. This requires accurate enough models for the policies to be transferable to real systems, as well as strategies that makes these policies robust to varying conditions.

Upon deployment, data-driven methods must be part of architectures that provide safety and verification, where they can benefit from explicit model methods. For instance, describing when a learned controller is successful, similar to control verification, can allow safe deployment. It can also assist in controller composition for long-horizon tasks, where high-level symbolic reasoning can be beneficial. Such task planning needs to be adaptive and allow a robot to dynamically define pre/postconditions, and discover new skills for its task. It should also be accompanied by failure explanation to identify why a problem is not solvable and guide data collection or reasoning for addressing similar challenges in the future. For explainability, maintaining an internal world model or a simulation (e.g., a "cognitive, physical engine") that is learned from data and first principles can be useful. There are already successful instances of integrating explicit and implicit models. There is work on natural-langauge-task-specification TAMP problems on large scenes where task plans from an LLM are verified using a model and pose-level planning is performed by a classical planner [99]. Another integration approach queries a VLM to solve a TAMP problem on the fly by creating 3D key points that resultin in a task plan as a sequence of path and goal constraints that guide motion planning [100].

A gap towards integrative solutions is the lack of common interfaces, software components and benchmarks that would allow to easily switch and experiment with components from different methodologies. Most existing instances of software infrastructure either support one set of methods or the other, requiring the ad hoc tool composition for a novel integrative approach.

References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Numerische Mathematik*, 1959.
[2] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE TSSC*, 1968.
[3] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *IJRR*, 2009.
[4] B. Cohen, S. Chitta, and M. Likhachev, "Single- and dual-arm motion planning with heuristic search," *IJRR*, 2014.
[5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *TRO*, 1996.
[6] S. LaValle and J. Kuffner, "Randomized kinodynamic planning," in *ICRA*, 1999.
[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *IJRR*, 2011.

[8] J. D. Gammell and M. P. Strub, "Asymptotically optimal sampling-based motion planning methods," *Annual Review of Control, Robotics, and Autonomous Systems*, 2021.
[9] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state–cost space," *TRO*, 2016.
[10] J. hwan Jeon, R. V. Cowlagi, S. C. Peters, *et al.*, "Optimal motion planning with the half-car dynamical model for autonomous high-speed driving," in *ACC*, 2013.
[11] A. Perez, S. Karaman, A. Shkolnik, E. Frazzoli, S. Teller, and M. R. Walter, "Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms," in *IROS*, 2011.
[12] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *RAM*, 2012.
[13] M. Zucker, N. Ratliff, A. Dragan, *et al.*, "CHOMP: Covariant hamiltonian optimization for motion planning," *IJRR*, 2013.
[14] J. Schulman, J. Ho, A. X. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *RSS*, 2013.
[15] M. Toussaint, "Newton methods for k-order Markov Constrained Motion Problems," arXiv:1407.0414, Tech. Rep., 2014.
[16] M. Mukadam, J. Dong, F. Dellaert, and B. Boots, "Steap: Simultaneous trajectory estimation and planning," *Autonomous Robots*, 2019.
[17] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIOPT*, 2024.
[18] T. McMahon, A. Sivaramakrishnan, E. Granados, and K. E. Bekris, "A survey on the integration of machine learning with sampling-based motion planning," *FTR*, 2022.
[19] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," in *IROS*, 2015.
[20] B. Burns and O. Brock, "Sampling-based motion planning using predictive models," in *ICRA*, 2005.
[21] H.-T. L. Chiang, A. Faust, S. Sugaya, and L. Tapia, "Fast swept volume estimation with deep learning," in *WAFR*, 2020.
[22] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato, "A machine learning approach for feature-sensitive motion planning," in *WAFR*. 2005.
[23] S. Li and N. Dantam, "Learning proofs of motion planning infeasibility," in *RSS*, 2021.
[24] A. H. Qureshi, A. Simeonov, M. J. Bency, and M. C. Yip, "Motion planning networks," in *ICRA*, 2019.
[25] C. R. Garrett, R. Chitnis, R. Holladay, *et al.*, "Integrated task and motion planning," *Annual Review of Control, Robotics, & Autonomous Systems*, 2021.
[26] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *ICRA*, 2014.
[27] K. Hauser and V. Ng-Thow-Hing, "Randomized multi-modal motion planning for a humanoid robot manipulation task," *IJRR*, 2011.
[28] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," *Towards Service Robots for Everyday Environments*, 2012.

[29] A. Curtis, G. Matheos, N. Gothoskar, *et al.*, "Partially observable task and motion planning with uncertainty and risk awareness," *RSS*, 2024.
[30] H. Kurniawati, "Partially observable markov decision processes (pomdps) and robotics," *CoRR*, 2021.
[31] J. Pineau, G. Gordon, and S. Thrun, "Point-based value iteration: An anytime algorithm for pomdps," in *IJCAI*, 2003.
[32] N. Ye, A. Somani, D. Hsu, and W. S. Lee, "Despot: Online pomdp planning with regularization," *JAIR*, 2017.
[33] H. Bai, D. Hsu, W. Lee, and N. Vien, "Monte carlo value iteration for continuous-state pomdps," in *WAFR*, 2010.
[34] L. P. Kaelbling and T. Lozano-Pérez, "Pre-image backchaining in belief space for mobile manipulation," in *ISRR*. 2017.
[35] V. Indelman, L. Carlone, and F. Dellaert, "Planning in the continuous domain: A generalized belief space approach for autonomous navigation in unknown environments," *IJRR*, 2015.
[36] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *ICRA*, 1985.
[37] E. Rimon and D. Koditschek, "Exact Robot Navigation Using Artificial Potential Functions," in *TRO*, 1992.
[38] O. Khatib, "A unified approach for motion and force control of robot manipulators: The operational space formulation," *JRA*, 1987.
[39] L. Sentis and O. Khatib, "Synthesis of whole-body behaviors through hierarchical control of behavioral primitives," in *IJHR*, 2005.
[40] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek, "Sequential composition of dynamically dexterous robot behaviors," *IJRR*, 1999.
[41] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts, "Lqr-trees: Feedback motion planning via sums-of-squares verification," *IJRR*, 2010.
[42] M. K. M. Jaffar and M. Otte, "Pip-x: Funnel-based online feedback motion planning/replanning in dynamic environments," in *WAFR*, 2022.
[43] C. K. Verginis, D. V. Dimarogonas, and L. E. Kavraki, "Kdf: Kinodynamic motion planning via geometric sampling-based algorithms and funnel control," *TRO*, 2022.
[44] E. Camacho and C. Alba, *Model Predictive Control*. 2013.
[45] "Constrained optimal control of linear and hybrid systems," *IEEE Transactions on Automatic Control*, 2005.
[46] A. Zheng, "A computationally efficient nonlinear mpc algorithm," in *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*, 1997.
[47] A. Richards and J. P. How, "Real-Time Model Predictive Control: A Framework for Optimal Guidance and Control in Aerospace Systems," *JGCD*, 2004.
[48] W. Edwards, G. Tang, G. Mamakoukas, T. Murphey, and K. Hauser, "Automatic tuning for data-driven model predictive control," in *ICRA*, 2021.
[49] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *ICRA*, 2002.
[50] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *IJRR*, 2002.
[51] K. E. Bekris and L. E. Kavraki, "Greedy but safe replanning under kinodynamic constraints," in *ICRA*, 2007.
[52] A. Zeng, S. Song, J. Lee, A. Rodriguez, and T. Funkhouser, "Tossingbot: Learning to throw arbitrary objects with residual physics," *TRO*, 2020.

[53] T. Zhang, Z. McCarthy, O. Jow, *et al.*, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *ICRA*, 2018.
[54] M. Laskey, J. Lee, R. Fox, A. Dragan, and K. Goldberg, "Dart: Noise injection for robust imitation learning," in *CoRL*, 2017.
[55] A. Mandlekar, D. Xu, J. Wong, *et al.*, "What matters in learning from offline human demonstrations for robot manipulation," *CoRL*, 2021.
[56] S. Ross, G. Gordon, and D. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in *AISTATS*, 2011.
[57] J. Zhang and K. Cho, "Query-efficient imitation learning for end-to-end simulated driving," in *AAAI*, 2017.
[58] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *AIJ*,
[59] A. Y. Ng and S. J. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000.
[60] Y. Xu, W. Gao, and D. Hsu, "Receding horizon inverse reinforcement learning," in *NeurIPS*, 2022.
[61] T. Pearce, T. Rashid, A. Kanervisto, *et al.*, "Imitating human behaviour with diffusion models," *ICLR*, 2023.
[62] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *NeurIPS*, 2020.
[63] C. Chi, Z. Xu, S. Feng, *et al.*, "Diffusion policy: Visuomotor policy learning via action diffusion," in *RSS*, 2023.
[64] A. Graves, *Generating sequences with recurrent neural networks*, 2014.
[65] B. Kang, X. Ma, C. Du, T. Pang, and S. Yan, "Efficient diffusion policies for offline reinforcement learning," in *NIPS*, 2024.
[66] L. Brunke, M. Greeff, A. W. Hall, *et al.*, "Safe learning in robotics: From learning-based control to safe reinforcement learning," *Annual Review of Control, Robotics, and Autonomous Systems*, 2022.
[67] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *IJRR*, 2018.
[68] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne, "Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning," *TOG*, 2017.
[69] T. Inoue, G. De Magistris, A. Munawar, T. Yokoya, and R. Tachibana, "Deep reinforcement learning for high precision assembly tasks," in *IROS*, 2017.
[70] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv:1707.06347*, 2017.
[71] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous control with deep reinforcement learning," *CoRR*, 2015.
[72] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: Lessons we have learned," *IJRR*, 2021.
[73] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *IROS*, 2017.
[74] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *ICML*, 2018.
[75] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *ICML*, 2018.

[76] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, 2015.
[77] M. Andrychowicz, F. Wolski, A. Ray, *et al.*, "Hindsight experience replay," *NeurIPS*, 2017.
[78] A. Kumar, J. Hong, A. Singh, and S. Levine, "When should we prefer offline reinforcement learning over behavioral cloning?" *ICLR*, 2022.
[79] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," *arXiv:2005.01643*, 2020.
[80] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," *arXiv:2110.06169*, 2021.
[81] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *NeurIPS*, 2020.
[82] T. Yu, G. Thomas, L. Yu, *et al.*, "Mopo: Model-based offline policy optimization," *NeurIPS*, 2020.
[83] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, "Hierarchical reinforcement learning: A comprehensive survey," *CSUR*, 2021.
[84] S. Schmitt, J. J. Hudson, A. Zidek, *et al.*, "Kickstarting deep reinforcement learning," *arXiv:1803.03835*, 2018.
[85] H. R. Walke, J. H. Yang, A. Yu, *et al.*, "Don't start from scratch: Leveraging prior data to automate robotic reinforcement learning," in *CoRL*, 2023.
[86] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine, "Learning invariant feature spaces to transfer skills with reinforcement learning," *ICLR*, 2017.
[87] J. Yang, C. Glossop, A. Bhorkar, *et al.*, "Pushing the limits of cross-embodiment learning for manipulation and navigation," *RSS*, 2024.
[88] Y. Zhang and Q. Yang, "A survey on multi-task learning," *TKDE*, 2021.
[89] X. Sun, R. Panda, R. Feris, and K. Saenko, "Adashare: Learning what to share for efficient deep multi-task learning," *NeurIPS*, 2020.
[90] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, "Experience replay for continual learning," *NeurIPS*, 2019.
[91] J. A. Mendez, H. van Seijen, and E. Eaton, "Modular lifelong reinforcement learning via neural composition," *arXiv:2207.00429*, 2022.
[92] M. Han, Y. Zhu, S.-C. Zhu, Y. N. Wu, and Y. Zhu, "Interpret: Interactive predicate learning from language feedback for task planning," *RSS*, 2024.
[93] G. Chen, L. Yang, R. Jia, *et al.*, "Language-augmented symbolic planner for open-world task planning," *RSS*, 2024.
[94] Y. J. Ma, W. Liang, G. Wang, *et al.*, "Eureka: Human-Level Reward Design via Coding Large Language Models," *arxiv-2310.12931*, 2023.
[95] A. Brohan, Y. Chebotar, C. Finn, *et al.*, "Do as i can, not as i say: Grounding language in robotic affordances," in *CoRL*, 2023.
[96] J. Duan, W. Yuan, W. Pumacay, *et al.*, "Manipulate-anything: Automating real-world robots using vision-language models," *arXiv:2406.18915*, 2024.
[97] M. J. Kim, K. Pertsch, S. Karamcheti, *et al.*, "OpenVLA: An Open-Source Vision-Language-Action Model," *arXiv:2406.09246*, 2024.
[98] A. Padalkar, A. Pooley, A. Jain, *et al.*, "Open X-embodiment: Robotic learning datasets and RT-X models," *arXiv preprint arXiv:2310.08864*, 2023.
[99] K. Rana, J. Haviland, S. Garg, J. Abou-Chakra, I. D. Reid, and N. Suenderhauf, "Sayplan: Grounding large language models using 3d scene graphs for scalable task planning," *CoRR*, 2023.
[100] W. Huang, C. Wang, Y. Li, R. Zhang, and L. Fei-Fei, "Rekep: Spatio-temporal reasoning of relational keypoint constraints for robotic manipulation," 2024.