

# A Survey of Optimization-based Task and Motion Planning: From Classical To Learning Approaches

By: Joshua Pen

# Task and Motion Planning (TAMP)

- Integrates high-level task planning and low-level motion planning
- Enables robots to solve long-horizon, real-world tasks
- Requires multiple levels of reasoning:
  - symbolic (tasks, actions)
  - geometric & physical (motion, contacts)

# Major Approaches to TAMP

- **Constraint-based TAMP** (feasibility)
  - Satisfy predefined constraints
- **Sampling-based TAMP** (exploration)
  - Randomized search (e.g., RRT-style methods)
- **Optimization-based TAMP** (optimality and feasibility)
  - Solve planning as a joint optimization problem

# Why Optimization-Based TAMP?

- Many robotic goals are objective-driven, not state-defined
  - e.g., *maximize height, minimize energy*
- Optimization-based TAMP:
  - Explicitly models plan quality
  - Incorporates task + motion constraints
  - Naturally handles robot dynamics & contacts
- Enables comparison between plans, not just feasibility

# Challenges of Optimization-Based TAMP

- Hybrid optimization is often computationally intractable
- Core tradeoff:
  - richer models  $\leftrightarrow$  higher computational cost
- Key limitations vs. sampling-based methods:
  - Sensitive to initialization
  - Can get stuck in local optima
  - Not complete (cannot prove infeasibility)
- Motivates hierarchical, distributed, and learning-based methods

# Questions to be answered

- Questions:
  - Q1: Why are optimization-based methods important for TAMP? What are the benefits?
  - Q2: How will solutions for each individual component of optimization-based methods inform the strategies to solve integrated TAMP?
  - Q3: What common structures are observed in optimization-based TAMP problems, and which tools and strategies can exploit these structures to efficiently generate long-horizon, dynamic robot plans?
  - Q4: How to leverage machine learning algorithms to enable robust, and generalizable TAMP frameworks?
- Q1–Q2: Explore the key components and critical features of optimization-based TAMP: Problem formulation, Domain representation, Task planning, Motion planning
- Q3–Q4: Examine current strategies and remaining challenges in optimization-based TAMP
- Conclusion: Discuss potential future research directions in TAMP

# Problem Formulation

# Assumptions

- A1 Deterministic transitions: If a symbolic action is applicable to a symbolic state, applying the action brings the deterministic symbolic transition system to a single other symbolic state, similarly for the application of continuous control;
- A2 Known models and objectives: The planner has complete knowledge about the continuous state transition system and the continuous dynamic system, as well as the objective function before the planning process begins.
- A3 Fully observable environments: The planner has complete knowledge about the symbolic and continuous states;
- A4 Sequential plans: the solutions to a TAMP problem are two linearly ordered finite sequences of symbolic actions and continuous controls, respectively;

However, in some extensions of optimization-based TAMP problems, certain assumptions may be relaxed.

# Task and Motion Planning as Joint Optimization

## Core idea

- Optimization-based TAMP is a **joint optimization Task planning (discrete, symbolic) and Motion planning (continuous, geometric), coupled through constraints and cost functions.**

## Task planning domain (discrete)

- Task domain:  $\mathcal{D}^t = \langle \mathcal{S}, \mathcal{A} \rangle$ 
  - $\mathcal{S}$ : symbolic states (discrete variables)
  - $\mathcal{A}$ : actions with transition rule

$$s_{k+1} = \gamma(s_k, a_k)$$

- Task plan (length  $K$ ):

- $s_0 = s^{init}, s_K \in S^{goal}$

$$\langle \mathbf{S}, \mathbf{A} \rangle = \langle s_0, a_0, s_1, \dots, a_{K-1}, s_K \rangle$$

# Task and Motion Planning as Joint Optimization

## Motion planning domain (continuous)

- Motion domain:  $\mathcal{D}^m$
- Robot state at time  $t$ :

$$x_t = [q_t, \dot{q}_t] \in \mathbb{R}^{2n}$$

- Control input:  $u_t \in \mathbb{R}^m$
- Dynamics:

$$x_{t+1} = f(x_t, u_t)$$

- Constraints:  
 $g(x_t, u_t) \leq 0$   
(joint limits, torque limits, collisions)

- Cost:

$$L(x_t, u_t) \in \mathbb{R}$$

# Task and Motion Planning as Joint Optimization

## Task–motion coupling

- Each symbolic state  $s \in \mathcal{S}$  defines a **manifold**  
$$\mathcal{X}^s = M(s)$$
- A symbolic action  $\langle s_k, a_k, s_{k+1} \rangle$  corresponds to a **trajectory segment**
- Constraints:
  - Trajectory stays in the current manifold:  
 $x_t \in \mathcal{X}^{s_k}$
  - Transition occurs at manifold intersection:  
 $x_{T_k} \in \mathcal{X}^{s_k} \cap \mathcal{X}^{s_{k+1}}$

# Task and Motion Planning as Joint Optimization

## Joint optimization problem

$$\min_{\langle S, A, X_{1:k}, U_{1:k} \rangle} \sum_{k=0}^{K-1} \sum_{t=0}^{T_k-1} L_{path}(x_{k,t}, u_{k,t}) + L_{goal}(x_{k,T_k})$$

## Subject to:

- Task transitions:  $s_{k+1} = \gamma(s_k, a_k)$
- Robot dynamics and constraints
- Manifold constraints linking task and motion

# Domain Representation

# Domain Representation Overview

- Planning domain representation requires knowledge of:
  - The environment, robot, and objects
  - Their inter-relationships and task goals
  - Continuous-domain information
- This knowledge must be represented in a standardized interface usable by optimization algorithms
- Traditional domain representations come from:
  - AI planning (PDDL)
  - Temporal logic (LTL, STL, MTL)
- A key limitation of these methods is that domain knowledge is hand-specified by experts
- A recent trend is to use learning-based methods to automatically encode domain knowledge for TAMP
- Examples include:
  - Learning symbolic operators to model action preconditions and effects
  - Using LLMs to interpret natural language and encode planning domains in a more intuitive way

# AI Planning

- PDDL (Planning Domain Definition Language) is a standard language used in the AI planning community
- In PDDL, an action is defined by five components:
  - Name: identifier of the action
  - Parameters: discrete and continuous variables used to evaluate conditions
  - Preconditions: predicates that must be satisfied before the action can be applied
  - Effects: predicates that describe how the state changes after the action
  - Cost: a positive scalar representing the action cost
- Classical PDDL assumes a deterministic, discrete, and non-temporal world model
- New versions and extensions of PDDL relax these assumptions to support richer planning scenarios

# Temporal Logic

- Temporal logic provides concise expressions for temporal relationships between symbolic events
- Linear Temporal Logic (LTL) assumes a linear sequence of events and includes:
- Propositional variables
- Boolean operators: negation, conjunction, disjunction
- Temporal operators, such as:
  - Eventually: a condition holds at some point in the future
  - Next: a condition holds at the next time step
  - Always: a condition holds throughout the entire execution
  - Until: one condition holds until another becomes true
  - Release: one condition holds until another is released
- Limitation of LTL: only supports boolean variables and discrete-time evaluation
- Extensions of LTL address these limitations:
  - Metric Temporal Logic (MTL): adds real-time constraints
  - Signal Temporal Logic (STL): supports continuous, real-valued signals
- These extensions enable specification of hybrid planning problems in TAMP

# Learning Operators and State Abstractions

- In Task and Motion Planning, the task planner must search over a large space of symbolic action sequences
- This search becomes combinatorially complex with many objects or long task horizons
- To improve efficiency, learning is used in two main ways:
  - Learning symbolic operators
    - Learn probabilistic transition models for symbolic actions
    - Estimate action feasibility and effects from experience
  - Learning state abstractions
    - Discover intrinsic task structure, such as hierarchies and object relevance
    - Decompose the search space into multiple abstraction levels

# Generating Domain Knowledge by LLMs

- Generating planning domain knowledge, such as action descriptions and goal specifications, traditionally requires manual input from experts using declarative languages like PDDL
- Recent advances in large language models (LLMs) show strong potential for automating this process across planning scenarios

## Generating action descriptions with LLMs

- Two main approaches:
  - Revising existing action descriptions to adapt them to new domains or situations
  - Directly generating new action descriptions for planning
- Major challenge: ensuring practicality and reliability of generated descriptions due to variability in LLM outputs

## Generating goal descriptions with LLMs

- Translate natural language objectives into formal specifications such as PDDL or LTL
- Challenges include:
  - Correctly understanding context
  - Strict adherence to syntax to avoid planning failures
  - Alignment with specific domains and task requirements
- Learning-based methods also translate language into LTL specifications
- AutoTAMP uses LLMs to convert task requirements into goals applicable at both task and motion levels

# Task Planning

# Task Planning Overview

- Task planning focuses on determining sequences of actions to achieve specific goals using symbolic methods
- Traditional approaches include:
  - Classical AI planning, which uses graph search algorithms with specialized heuristics
  - Temporal logic–based planning, especially using LTL, which applies automata theory and reactive synthesis
- A major challenge is combinatorial complexity, which limits scalability and efficiency in large-scale planning problems
- Recent advances aim to bypass this bottleneck by using learned models to guide and accelerate task sequence search

# Classical Task Planning

- Classical task planning addresses planning in deterministic, static, finite, and fully observable state-transition systems with restricted goals and implicit time
- The most common approach is state-space search:
  - Nodes represent states
  - Edges represent action-induced transitions
  - Solutions are sequential paths from an initial state to a goal state
- Key design considerations include:
  - Choice of search space
  - Selection of efficient search algorithms
  - Design of heuristics to guide the search
- Classical AI planning can be viewed as a relaxed search problem
  - Heuristic design involves a trade-off between computational cost and informativeness
- Fast Downward–based planners use hierarchical task decomposition and causal graph heuristics to guide forward search
- Plan-space search methods, such as hierarchical task networks (HTNs), exist but are rarely used in TAMP due to poor integration with motion planners
- For temporal logic–based formulations such as LTL:
  - Automata-based methods are commonly used
  - Reactive synthesis generates controllers that satisfy specifications under all external inputs

# Learning Models for Task Planning in TAMP

- Core bottleneck in TAMP:
  - combinatorial explosion of discrete task plans
  - many expensive motion planning calls
- Key idea:
  - use learning to guide or replace high-level task planning
- Goal:
  - reduce search space
  - improve scalability for long-horizon tasks

# Learning Symbolic Action Models

- Learn action effects and preconditions from experience
- Probabilistic and relational representations:
  - model uncertainty in action outcomes
- Abstract low-level motion into high-level symbols
- Enables:
  - faster task planning
  - reduced reliance on hand-coded models

# Deep Learning-Based Task Planning Models

- Learn task models from large-scale data
- Replace explicit symbolic reasoning with neural models
- Key capabilities:
  - task decomposition
  - long-horizon generalization
  - compositional reasoning
- Examples:
  - Neural task programming: learn task decomposition from demonstrations
  - Neural task graphs: learn action dependencies as graphs
  - Regression planning networks: predict intermediate subgoals
  - Deep RL & affordance foresight: predict action usefulness

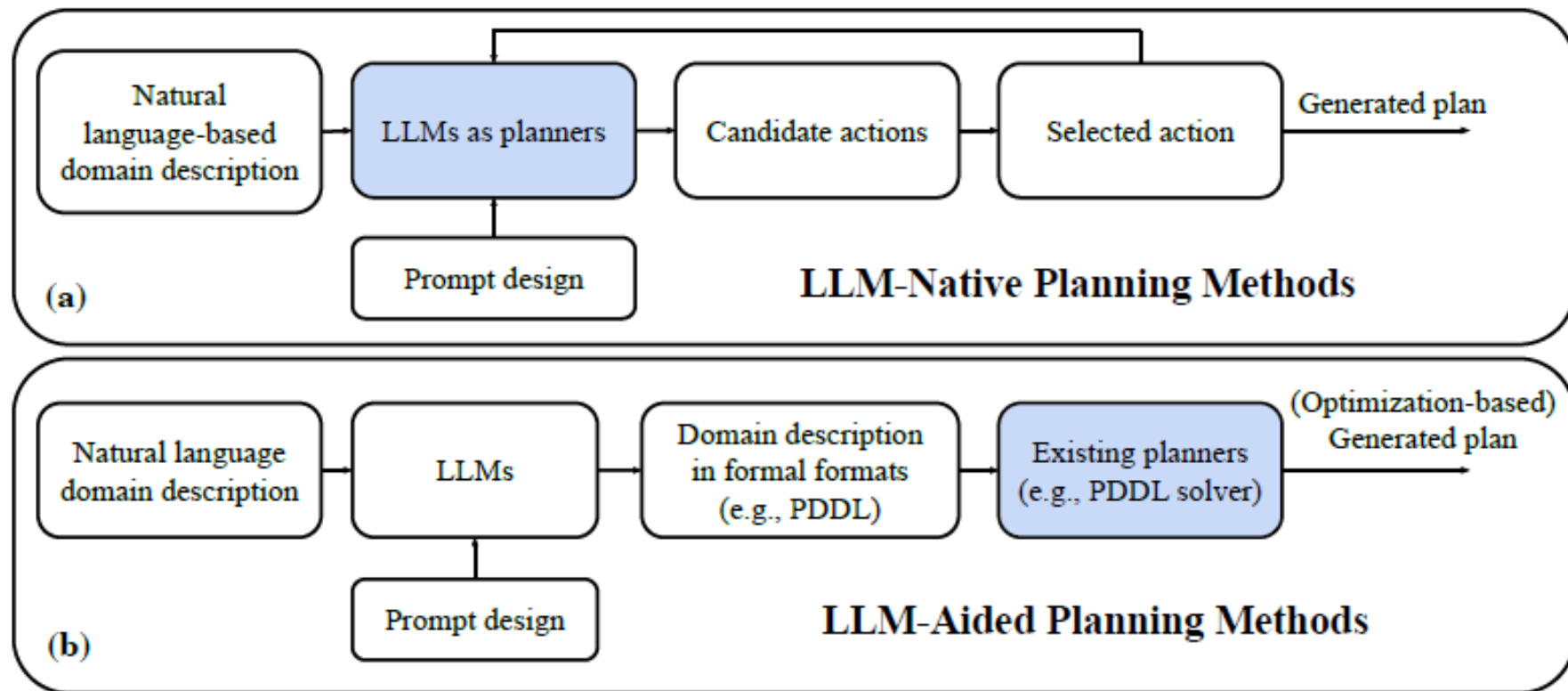
# Takeaways for TAMP

- Learning improves:
  - scalability
  - generalization
  - long-horizon planning
- Common theme:
  - guide or replace **symbolic task search**
- Limitation:
  - learned models still need integration with motion feasibility
- Sets the stage for:
  - learning + optimization-based TAMP

# LLMs for Task Planning

- Traditional task planning:
  - minimize number of actions or total plan cost
- LLMs (ChatGPT, Bard, LLaMA):
  - introduce **commonsense** and **world knowledge**
  - strong at **language understanding & symbolic reasoning**
- Key benefit:
  - reason about tasks **without prior robot interaction**
- Limitation:
  - weak at **numerical optimization**

# Two LLM-Based Planning Paradigms



# LLM-Native Planning

- LLM generates task plans:
  - one-shot or iterative prompting
- Focus on:
  - prompt design
  - grounding language to robot actions
- Often augmented with:
  - affordance models
  - reinforcement learning
  - environment feedback
- Example systems:
  - SayCan: combines LLMs with affordance scoring
  - Inner Monologue: replanning using feedback
  - ProgPrompt: programmatic prompts + failure recovery

# LLM-Aided Planning

- LLMs enhance **classical task planners**
- Common integration strategies:
  - Natural language → PDDL / temporal logic
  - Extract commonsense knowledge for actions
  - Guide search heuristics (e.g., MCTS)
- Optimization remains:
  - explicit
  - interpretable
  - efficient

# Optimization in LLM-Based Planning

- LLM-Native methods
  - optimization is *implicit*
  - occurs through prompting & interaction
- LLM-Aided methods
  - optimization is explicit
  - handled by classical planners
- Both approaches:
  - compatible with optimization-based TAMP
  - improve scalability and generalization

# Optimization-Based Motion Planning

# Optimization-Based Motion Planning Overview

- Aims to generate a continuous robot motion trajectory and control sequence
- Optimizes an objective function subject to kinematic and or dynamic constraints
- Common trajectory optimization techniques include:
  - Direct methods, which transcribe the problem into nonlinear programs (NLPs)
  - Indirect methods, which leverage optimality conditions
- To address increasing environmental and task complexity:
  - Distributed optimization techniques are introduced
  - Consensus-based methods such as ADMM improve scalability
- Recent advances combine model-based trajectory optimization with data-driven methods
  - Learn from offline optimized trajectories
  - Enable predictive generation of motion plans through imitation

# Trajectory Optimization

**Given:** Motion planning domain  $D_m$

Initial state  $x_{\text{init}}$

Goal state  $x_G$

A motion plan is a trajectory:

$$\langle X, U \rangle = \langle x_0, u_0, x_1, u_1, \dots, x_T \rangle$$

with

$$x_0 = x_{\text{init}}, \quad x_T = x_G$$

**Optimization formulation**

$$\min_{X, U} \sum_{t=0}^{T-1} L_{\text{path}}(x_t, u_t) + L_{\text{goal}}(x_T)$$

subject to

$$x_{t+1} = f(x_t, u_t)$$

$$x_0 = x_{\text{init}}, \quad x_T = x_G$$

$$g(x_t, u_t) \leq 0$$

# Trajectory Optimization

- Direct collocation converts a continuous motion planning problem into a finite-dimensional optimization problem
- Instead of solving differential equations directly, it:
  - Discretizes time into a set of knot points
  - Treats robot states and control inputs at each knot point as decision variables
  - Enforces robot dynamics as constraints between neighboring points
- Enables the use of general-purpose nonlinear programming solvers, such as IPOPT and SNOPT
- To improve real-time performance:
  - Differential Dynamic Programming (DDP) is a shooting method that exploits problem structure
  - Uses Riccati recursion to efficiently handle nonlinear dynamics
  - Classical DDP is limited to unconstrained trajectory optimization
  - Recent variants extend DDP to handle state and control constraints

# Distributed Optimization

## Distributed Structures in Trajectory Optimization

- Distributed structures are often solved using alternating optimization methods, such as ADMM, to improve the efficiency of trajectory optimization (TO)
- In many cases, TO problems must be reformulated explicitly to expose their distributed structure
- Consensus constraints are commonly introduced to reveal and enforce this structure

## Consensus ADMM Formulation

- Consider an optimization problem where the objective is the sum of multiple components:
  - Minimize the sum of  $N$  cost functions
- This problem can be reformulated in a consensus ADMM form:
  - Each subproblem optimizes its own local variables
  - All local variables are constrained to agree with a shared global variable
- Achieving good consensus performance depends on:
  - Proper selection of ADMM parameters
  - Techniques such as over-relaxation, adaptive penalty parameters, and acceleration methods

# Distributed Optimization

## Exploiting Problem Structure

- **Spatial structure**
  - Subsystems have separable dynamics
  - Variables may be coupled through objectives or constraints, such as collision avoidance between robots
- **Temporal structure**
  - TO is typically discretized over time
  - Costs and constraints are often local to each timestep
  - Dynamics constraints couple states and controls across consecutive timesteps
- **System structure**
  - The system consists of multiple interacting subsystems with different dynamics
  - ADMM separates the full problem into smaller subproblems, each corresponding to a subsystem

# Learning Methods for Motion Planning

## Learning to Facilitate Trajectory Optimization (TO)

- Achieving real-time trajectory optimization remains challenging in many applications
- Learning methods are widely explored to support motion generation by:
  - Learning objectives and constraints to guide TO
  - Learning physical models for integration into TO
  - Learning end-to-end policies that imitate TO solutions

## Learned Objectives and Constraints for TO

- Cost functions can be recovered from trajectory demonstrations
- Guided Cost Learning
  - Iteratively samples trajectories using TO
  - Uses policy optimization to recover costs explaining expert behavior
- Inverse KKT
  - Learns both cost functions and constraints
  - Explicitly models optimality conditions of constrained optimization

# Learning Methods for Motion Planning

## Learned Physical Models for TO

- **Challenge**
  - Physical contact introduces discontinuities (impacts, stick–slip)
  - Leads to non-smooth gradients and numerical instability in TO
- **Motivation**
  - Replace hard contact models with smooth, differentiable approximations
  - Enable stable gradient-based optimization
- **Learned contact models**
  - ContactNets
    - Learn inter-body distances and contact Jacobians with smooth implicit representations
  - Residual contact dynamics
    - Jointly learn continuous and contact dynamics via residual networks
  - Object-centric models
    - Learn object geometry and dynamics with differentiable contact behavior
  - Signed Distance Fields (SDFs)
    - Provide smooth distance and gradient information for optimization-based planning
- **Limitation**
  - Accurately modeling stiff contacts remains challenging

# Learning Methods for Motion Planning

## End-to-End Policy Learning Guided by TO

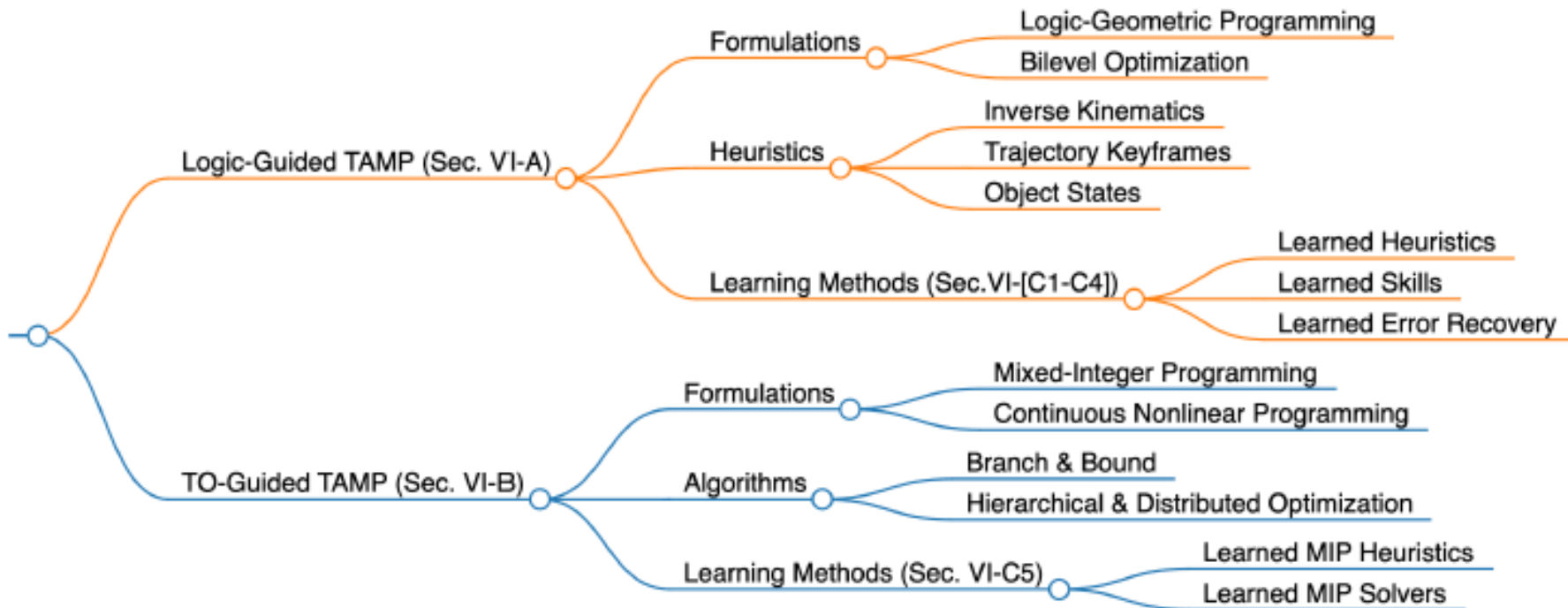
- Motivation
  - TO is accurate but:
    - Computationally slow
    - Sensitive to initialization
    - Unsuitable for real-time execution
  - Solution: learn neural policies that imitate offline TO solutions
- Policy learning from TO
  - Guided Policy Search (GPS) iterative policy training using DDP-generated trajectories
  - ADMM-based methods enforce consistency between policy outputs and TO trajectories
- Sequential and task-aware motion generation
  - OracleNet uses RNNs to reconstruct motion plans step-by-step
  - CoMPNet encodes task descriptions and environments into latent space and generates intermediate configurations conditioned on start and goal
- Sequence modeling approaches
  - Transformer policies
    - Autoregressive action generation trained in simulation
  - Diffusion policies
    - Iterative denoising for high-dimensional, multimodal action sequences

# Integrated Task and Motion Planning

# Integrated Task and Motion Planning Overview

- Task planning and motion planning are interdependent
- Plans must be:
  - feasible and
  - near-optimal
- Optimization-based TAMP:
  - jointly reasons over **discrete decisions** and **continuous trajectories**
- Discrete variables:
  - symbolic actions, task order
- Continuous variables:
  - robot trajectories, forces, timings
- Core challenge:
  - combinatorial search × numerical optimization

# Logic-Guided TAMP vs TO-Guided TAMP



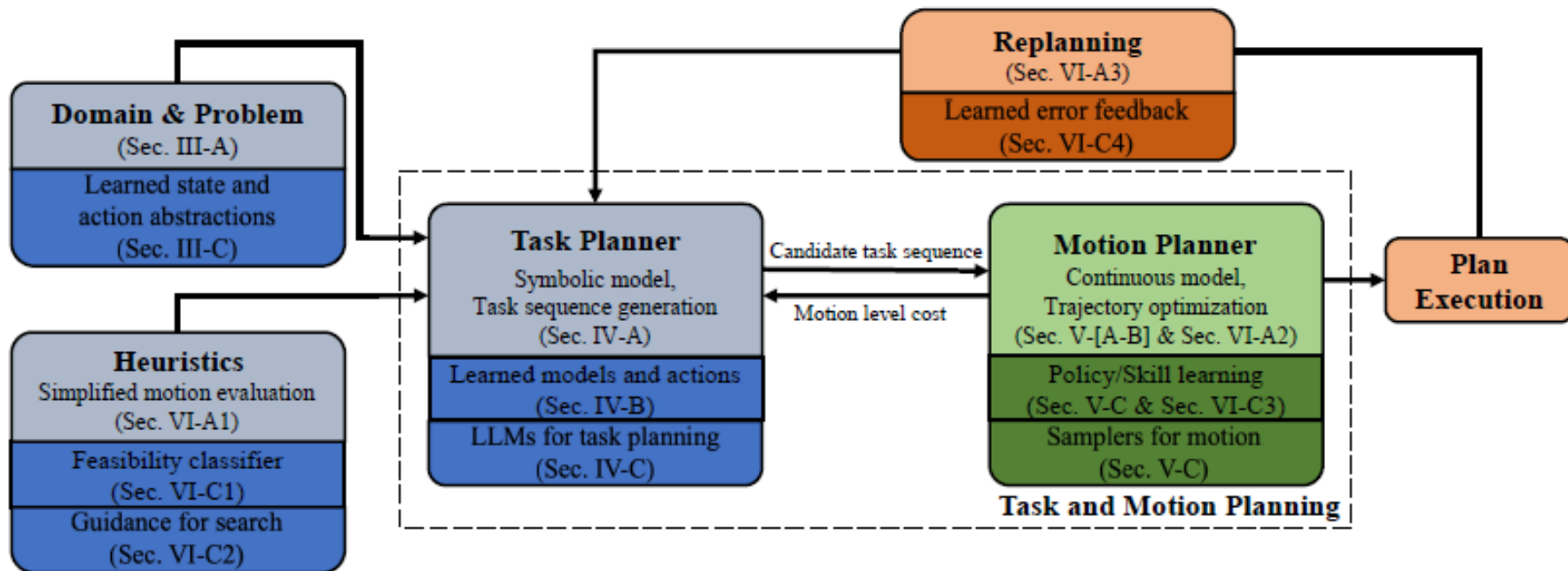
# Logic-Guided TAMP

- Symbolic planner (e.g., PDDL) searches task space
- Motion planning:
  - embedded as refinement
- Typical structure:
  - task planning ↔ motion feasibility checks

The core tradeoff: How much to intervene?

- Strict hierarchy:
  - task plan first → motion later
  - often infeasible
- Fully intertwined:
  - motion planning at every node
  - computationally intractable
- Research question:
  - How to balance feasibility checks vs cost?

# Schematic overview of logic-guided TAMP



# Making Logic-Guided TAMP Practical

- Heuristics approximate motion feasibility:
  - IK checks
  - object-centric distances
- Multi-stage refinement:
  - coarse checks → full trajectory optimization
- Multi-modal motion planning:
  - symbolic switches → trajectory segments

# Receding Horizon Logic-Guided TAMP

- Real-world execution is noisy
- Receding horizon TAMP:
  - plan → execute → replan
- Analogous to MPC, but over:
  - hybrid symbolic + continuous domains

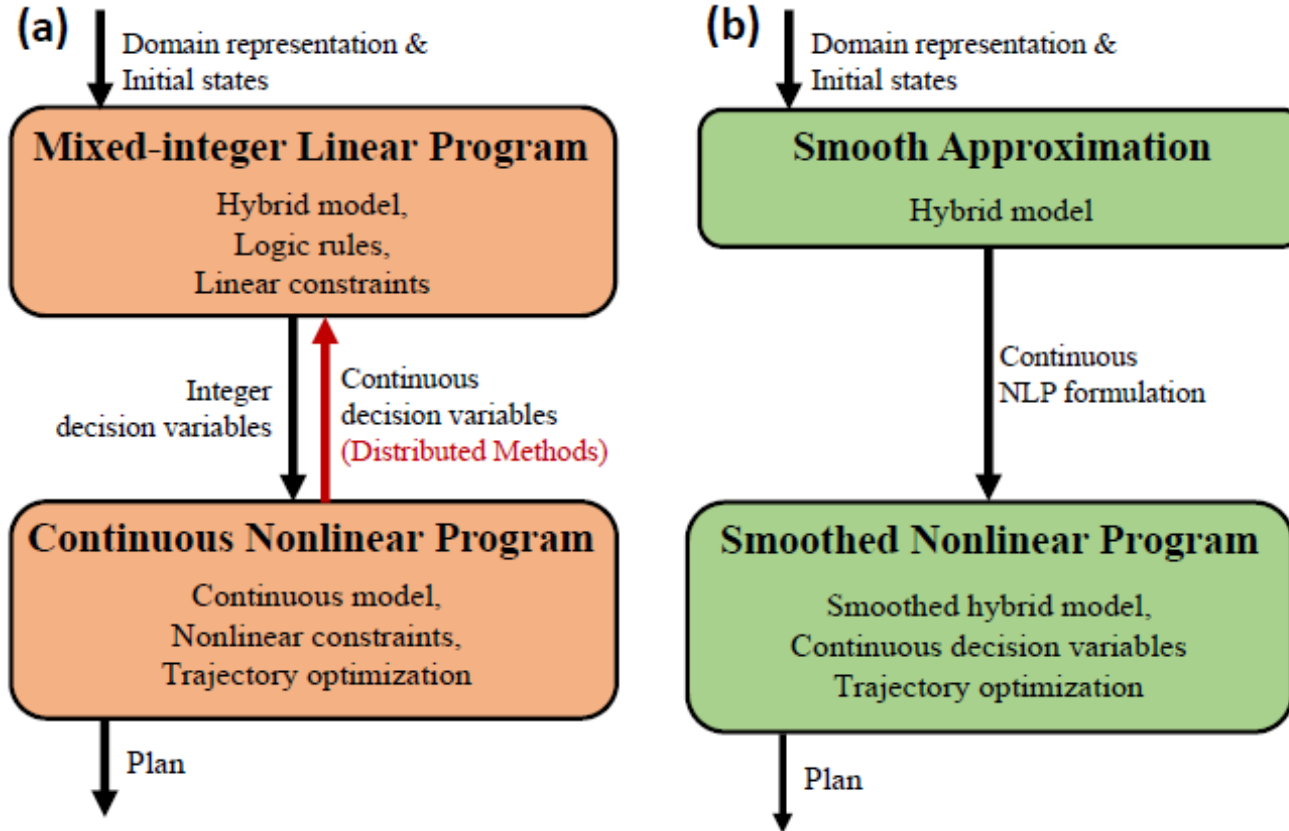
# TO-Guided TAMP

- Single optimization problem
- Discrete decisions → binary variables
- Formulated as:
  - MILP / MICP / MINLP
- No explicit task planner

## Some algorithms:

- **Branch-and-Bound:**
  - global optimality
  - poor scalability
- **Hierarchical & distributed methods:**
  - split discrete and continuous parts
- **Smooth approximations:**
  - relax discreteness → continuous NLP

# Example algorithm structures for TO-guided TAMP



# Learning for Integrated TAMP

- Classical TAMP:
  - expensive search
  - hand-engineered skills
- Learning helps:
  - prune infeasible plans
  - guide search
  - reuse motion skills

## Roles of Learning in Integrated TAMP:

1. Feasibility classifiers
2. Search guidance
3. Skill policies
4. Error recovery
5. Learning for MIP solvers

# Integrated TAMP — Key Takeaways

- Logic-Guided:
  - interpretable
  - heuristic-driven
- TO-Guided:
  - unified optimization
  - computationally heavy
- Learning:
  - bridges scalability gaps
- Open challenge:
  - optimality × robustness × efficiency

# Future Challenges and Opportunity

# Foundation and Diffusion Models for TAMP

- Foundation Models for TAMP
  - Robot planning requires breaking complex task specifications into environment-specific, actionable steps
  - LLMs and VLMs often produce overly abstract plans, ignoring physical and practical constraints
  - Limited short-term memory can cause information loss in multi-stage or sequential tasks
  - This impacts planning coherence and efficiency
  - Despite these challenges, promising progress in LLM- and VLM-based planning methods is emerging
- Diffusion Models for TAMP
  - In TAMP, diffusion models can act as trajectory samplers for individual skills
  - They generate diverse and feasible motion trajectories, improving robustness
  - Generative skill chaining learns short-horizon, skill-centric diffusion models
  - A compositional framework combines skills to generate long-horizon plans from a plan skeleton
  - Future direction: integrate LLMs and diffusion models to build generative, multi-task, end-to-end TAMP frameworks

# Multi-Modal Sensing and Policy Learning for TAMP

- Multi-Modal Sensing for TAMP
  - Most current TAMP systems rely primarily on visual sensing
  - Future opportunities include integrating multi-modal sensing, such as: Vision, force–torque, tactile, and proprioceptive sensors
  - Requires advanced sensor fusion methods Improved contact information representation, extraction, and utilization is critical for robust planning
- Policy Learning in and for TAMP
  - Reinforcement learning–based policies face key challenges: Sample inefficiency due to costly trial-and-error learning
  - Dependence on carefully designed dense reward functions
  - These issues are amplified in long-horizon, complex tasks
  - Training often occurs in simulation, creating additional challenges for sim-to-real transfer
  - Leveraging real-world robot teleoperation data can improve generalization
    - Requires careful system design to ensure: Seamless human–robot handover Efficient and scalable data collection

# TAMP Beyond Manipulation: Emerging Application Domains

- Locomotion and Manipulation

- Manipulation is typically object-centric with lower-frequency contact switching
- Locomotion is robot-centric with high-frequency contact changes
- Often handled hierarchically: contact planning → trajectory generation (e.g., centroidal planning)
- Open challenge: developing a unified TAMP framework for loco-manipulation Requires balancing dexterous grasping and uneven-terrain locomotion

- Human–Robot Collaboration (HRC)

- Human behavior introduces uncertainty and non-determinism
- Robot must reason about: Human intentions, Timing, and safety constraints
- Emerging directions: Novel communication modalities, Integration of natural language, and conversational interaction into planning

- Real-World Deployment

- Real environments are open, dynamic, and uncertain
- Key challenges: Robustness to environmental variability Sensor noise and calibration errors Reliable low-level control under imperfect feedback
- Requires improved scene understanding capturing both spatial and semantic relationships

Thank You!