Real-time goal recognition using approximations in Euclidean space

Douglas Antunes Tesch[b,*], **Leonardo Amado**[a] **and Felipe Meneguzzi**[a, b]

[a]University of Aberdeen
[b]Pontifical Catholic University of Rio Grande do Sul
ORCID (Douglas Antunes Tesch): https://orcid.org/0000-0003-3037-919X, ORCID (Leonardo Amado): https://orcid.org/0000-0001-6119-4601, ORCID (Felipe Meneguzzi): https://orcid.org/0000-0003-3549-6168

Abstract. While recent work on online goal recognition efficiently infers goals under low observability, comparatively less work focuses on online goal recognition that works in both discrete and continuous domains. Online goal recognition approaches often rely on repeated calls to the planner at each new observation, incurring high computational costs. Recognizing goals online in continuous space quickly and reliably is critical for any trajectory planning problem since the real physical world is fast-moving, e.g. robot applications. We develop an efficient method for goal recognition that relies either on a single call to the planner for each possible goal in discrete domains or a simplified motion model that reduces the computational burden in continuous ones. The resulting approach performs the online component of recognition orders of magnitude faster than the current state of the art, making it the first online method effectively usable for robotics applications that require sub-second recognition.

1 Introduction

Goal recognition focuses on predicting an agent's behavior and determining its goal through observing the agent's actions [28, 16, 15]. The ability to anticipate an agent's behavior is critical for autonomous agents working cooperatively and competitively. In cooperative settings, effective goal recognition allows agents to obviate explicit communication to coordinate their joint plans. By contrast, effective goal recognition in non-cooperative settings allows agents to anticipate an opponent's moves and counter them in a timely fashion. In robot football, for instance, this is critical in anticipating the opponent's trajectory and developing a winning counter-strategy. Similarly, for cooperation within a match, team members can choose plans that ease recognition of their goals to others in the same team, minimizing explicit communication [32].

The current state-of-the-art in online goal recognition for continuous and discrete domains stems from approaches from Vered [30, 31]. These methods use an off-the-shelf planner to dynamically compute the probabilities of goal hypotheses as new observations arrive. While these methods use a heuristic to minimize the number of calls to the planner during the online phase, it still needs multiple calls to a full-fledged planner. These calls, however few, can be expensive, making this approach unsuitable for fast recognition under certain conditions. Recent research on goal recognition substantially improves efficiency for both domains [14, 19, 18]. However, these approaches formulate the problem in a discrete space using a planning formalism (typically STRIPS) or rely on a discretization of continuous state space [11] or action space [6]. By contrast, robotics applications where the agent's state includes specific configurations of a robot's degrees of freedom (position, translations, and angular rotation) cannot be trivially discretized. Most approaches for online goal recognition in continuous domains focus on the path-planning problem, disregarding dynamics characteristics of the agents, so the plan consists only of a geometric collision-free path [14, 11]. However, trajectory planning for collision-free paths requiring all dynamics and constraints of the agent is much more complex. Computing the probability of a single observation and hypothesis can take many minutes. Thus, such approaches are unsuitable for robots recognizing the goals of other robots while in motion since relying on calls to a motion planner incurs an unacceptable computational cost [10].

We address these problems through a novel formulation for online goal recognition that works in continuous (trajectory planning problem) and discrete (STRIPS) domains. Our contribution is fourfold. First, we develop an online inference method to compute the probability distribution of the goal hypotheses based on the work of Ramírez [22] and Masters [14] that obviates the need for calls to a planner during run-time. We base our inference method on the Euclidean distance between a pre-computed sub-optimal trajectory and observations of the actual agent. Second, we employ a predefined approximation of the agent's motion model for motion applications that obviates the need for costly computations of sub-optimal paths. The two contributions described above are mainly responsible for reducing the computational burden by orders of magnitude. Third, we adapt our continuous inference to recognize goals in discrete domains using Euclidean distance as measure, overcoming a limitation of previous approaches for goal recognition. Fourth, we extend our inference process so that it generates multiple paths for the same goal to account for different approximately similar alternative paths towards the same goal. We show empirically that this improves the general quality of the recognition process.

2 Online Goal Recognition

We adapt notation from Vered [30], and expand it with the notion of time critical to real applications, i.e., robotics. An online goal recognition (GR) problem is a quintuple $R := \langle W, I, G, M, O \rangle$, where $W : \mathbb{R}^n$ is an n-dimensional continuous state space, e.g., position,

* Corresponding Author. Email: douglas.tesch@edu.pucrs.br.

angles, velocity, acceleration, time, etc.[1] By convention, we always use time as the last dimension and omit time in examples in which time is irrelevant (e.g., initial states). For improved readability, we denote a particular state sampled at a discrete-time t_k as $w[t_k] \in \mathbb{R}^n$. $I \in W$ is the initial state of the agent in the zero-time step. G is a set of partial states where, depending on the goal recognition problem, the goal state may be missing different types of information, which represent goal hypotheses with size $\mathcal{N}$, where $g_n \in G$ is a particular goal state in the set. M is a set comprising all possible trajectories. Finally, $O \subseteq M$ is a discrete set of observations representing snapshots through the real trajectory. As an online problem, the size of the observation set increases as the agent receives new information, where $o[t_k] \in O$ is an observation sampled at time t_k.

A trajectory is a sequence of T discrete timed states as $m_I^{g_n} = [w[0], w[1], \ldots, w[T-1]]$, i.e., it is the set of states $m_I^{g_n} \subset W$ ordered by the last component of each $w \in m_I^{g_n}$. $M_I^{g_n} \subseteq M$ is the set of all possible trajectories starting in the initial state I and finishing in g_n; $m_I^{g_n} \in M_I^{g_n}$ is one particular trajectory to a goal g_n; R is an offline problem when the final discrete time step t_k is known; otherwise, R is an online problem. A solution to the online GR problem R is a hidden goal g_n reachable through a trajectory $m_I^{g_n}$, and which maximizes the conditional probability $P(m_I^{g_n} \mid O)$ of a trajectory given the current observation set O defined in Eq. 1, where $m_I^{g_n\,R}$ is the trajectory that best explains the observations.

$$m_I^{g_n\,R} = \underset{m_I^{g_n} \in M}{\operatorname{argmax}} P(m_I^{g_n} \mid O) \tag{1}$$

This formulation is similar to Ramírez [21]—the main difference is that we search for a trajectory $m_I^{g_n}$ instead of a plan. Instead of computing the trajectory's probability as proportional to the cost differential between an optimal trajectory and the observed one, we compute an $m_I^{g_n}$ that matches the observations and maximizes $P(m_I^{g_n} \mid O)$. This formulation makes three key assumptions: i) agents pursue a single goal that does not change over time; ii) agents always prefer the cheapest cost trajectories; and iii) all goal hypotheses are mutex, i.e., all are different and the agent pursues exactly one.

$$\begin{aligned} P(m_I^{g_n} \mid O) =& \rho P(O \mid m_I^{g_n}) P(m_I^{g_n}) \\ =& \rho P(O \mid m_I^{g_n}) P(m_I^{g_n} \mid g_n) P(g_n) \end{aligned} \tag{2}$$

Thus, we compute the conditional probability in Eq. 2, where $P(g_n)$ is a uniform distribution indicating the probability that the robot is pursuing the goal g_n; ρ is a normalization to avoid computing $P(O)$. To solve Eq. 2, we need to compute $P(m_I^{g_n} \mid g_n)$ and $P(O \mid m_I^{g_n})$. We can compute $P(m_I^{g_n} \mid g_n)$ by synthesizing an optimal trajectory hypothesis $m_I^{g_n *}$ that disregards the observations and aims for the final goal $g_n \in G$. In our case, we assume that if there are multiple optimal paths $m_I^{g_n *}$, then $P(m_I^{g_n *} \mid g_n)$ is uniform (i.e. the agent picks any optimal trajectory at random). Otherwise, the probability $P(m_I^{g_n *} \mid g_n)$ is always one. Computing $P(O \mid m_I^{g_n})$ is a challenging task since we do not have a Probability Density Function (PDF) for the observations. A common approach for this situation is approximating the probability distribution for a parametric function considering an assumption about their characteristics; for example, consider an assumption that the probability $P(O \mid m_I^{g_n})$ increases as the trajectory gets closer to the observations [9]. Thus, Eq. 3 approximates the conditional probability of $P(O \mid m_I^{g_n}$, where $dist(\cdot, \cdot)$ is a spatial distance calculation, e.g., Euclidean distance; $N \in \mathbb{N}$ is the actual number of observations at this moment; $m_I^{g_n *}[t_k]$ and $o[t_k]$ are the state in the optimal trajectory and the observation at the discrete step-time t_k. $\mathcal{T}$ is a set containing the discrete timestamp t_k of all observations, i.e., we can sample states with a synchronized timestamp. Thus, Eq. 3 computes the conditional probability not only under full observability but also under partial observability, using the timestamp set $\mathcal{T}$ to match observations with their respectively optimal trajectory state. The semantics of the timestamps depends on whether the domain is continuous or discrete. In continuous domains, we use a local clock to attach timestamps to each observation as it arrives. Note that continuous domains inherently imply partial observability, since there is an infinite number of possible states between each observation. By contrast, in discrete domains (STRIPS), goal recognition methods do not attach timestamps to observations, and assume a total order between observations. Here, we consider partial observability to be equivalent to the problem of online recognition, that is observations are consecutive and sequentially sampled, with the missing observations corresponding to the suffix of the full plan.

The conditional probability $P(O \mid m_I^{g_n *})$ must increase as the observations O get closer to an optimal trajectory $m_I^{g_n *}$. Thus, the value of $P(O \mid m_I^{g_n *})$ represents the probability that the observations belong to trajectory $m_I^{g_n *}$ and Eq. 3 decays exponentially as the average error increases among all terms in the observations set O and the optimal state trajectory in the same instant of discrete-time t_k.

$$P(O \mid m_I^{g_n *}) := 1 - \mathrm{e}^{\left(-1/\frac{1}{N} \sum_{t_k \in \mathcal{T}} dist(o[t_k], m_I^{g_n *}[t_k])\right)}, \quad n \in [1, \ldots, \mathcal{N}] \tag{3}$$

We can redefine the conditional probability of Eq. 2 as Eq. 4, and the most likely goal hypothesis g_n is that with the highest $P(m_I^{g_n *} \mid O)$ value. An offline stage can compute optimal trajectories for each goal hypothesis, and at each new observation, we compare the observations with such trajectories to recognize the most likely goal hypothesis. We do this by averaging the distances between an optimal trajectory and the observation samples using only the Eqs. 3-4, with no online calls to a planner. Therefore, planner calls in online goal recognition depend only on the number of goal hypotheses in the set G, which differs from previous work that depends on the number of observations O and goals G in the sets.

We note that while Masters [14] shows the last observation is enough to compute the conditional probability for path-planning problems, we average over the entire sequence of observations to improve the method's robustness to outliers and noisy observations.

$$P(m_I^{g_n *} \mid O) = \rho P(O \mid m_I^{g_n *}) P(m_I^{g_n *} \mid g_n) P(g_n), \quad n \in [1, \ldots, \mathcal{N}] \tag{4}$$

Computing the inference step in goal recognition by comparing states, instead of computing a cost function, can lead to problems in continuous domains. Specifically, there is an infinite number of approximately optimal trajectories $m_I^{g_n *}$ an agent can use to navigate between the initial state I and the goal state g_n while avoiding obstacles. In some obstacle configurations, it is even possible that the agent has multiple optimal paths for reaching a goal. To deal with this problem, we based our process of goal inference on an average among a collection of solutions from the same problem. For this, we compute k solution trajectories $m_I^{g_n *}$ for problems comprised the common initial state and each goal hypotheses. Eqs. 5 and 6 summarize the averaging process above; $\mathcal{M}_{g_n}[j]^*$ defines a matrix

[1] For our experiments in Continuous Domains we use $n = 10$.

containing all solutions trajectories $m_I^{g_n *}$ where $j \in [1, \ldots, k]$.

$$P(O \mid {\mathcal{M}_{g_n}}^*) := \frac{1}{k}\sum_{j=1}^{k} 1 - \mathrm{e}^{\left(-1/\frac{1}{N}\sum_{t_k \in \mathcal{T}} dist(o[t_k], {\mathcal{M}_{g_n}}^*[j][t_k])\right)}, \quad n \in [1, \ldots, \mathcal{N}] \tag{5}$$

$$P({\mathcal{M}_{g_n}}^* \mid O) = \rho P(O \mid {\mathcal{M}_{g_n}}^*) P({\mathcal{M}_{g_n}}^* \mid g_n) P(g_n), \quad n \in [1, \ldots, \mathcal{N}] \tag{6}$$

3 Trajectory Planning Approximation

Consider a common problem of robot navigation with obstacle avoidance and dynamic restrictions defined in Cartesian X and Y axes, where the continuous state domain is defined as $W = [x, y, \dot{x}, \dot{y}, t_k]$, i.e., positions, velocity and the discrete-time t_k in both axes. Our online approach to goal recognition needs an optimal trajectory $m_I^{g_n *}[t_k] = [x, y]$ to compute probabilities for the goal hypotheses. However, generating an optimal collision-free trajectory in continuous Cartesian space is a problem of trajectory planning that has a high computational cost [2, 10, 20], which we mitigate by approximating the optimal trajectory $m_I^{g_n *}$ and directly applying it to Eq. 4. This problem becomes more challenging still if the environment is a dynamical system, and we want to compute an optimal trajectory [23]. In this paper, a dynamical system is an environment whose behavior can be described by sequential ordered differential equations [17]. Recall that a trajectory $m_I^{g_n}$ is a sequence of states that describes the agent's movement within such a dynamical system. To navigate such system, the agent needs a policy that applies a correct control input that drives the states to the desired goal [17].[2] Computing a trajectory $m_I^{g_n}$ in any dynamical model requires motion planning to generate such control inputs over time. We avoid running a motion planner by approximating $m_I^{g_n *}$ through a polynomial model, which we compute using a method of trajectory generation from robotics [3], which computes a trajectory using fewer motion parameters, consequently reducing the dimensionality of the optimization problem.

The motion parameters are the agent's desired dynamics characteristics or state at a specific time used to compute a full continuous trajectory between two points [13]. These motion parameters depend on the type of trajectory to be computed, e.g., linear, trapezoidal, and polynomial. In this paper, we use a polynomial trajectory that takes the desired time duration and position, velocity, and acceleration as motion parameters in the initial and final states of a trajectory. Algorithm 1 describes the general framework of using a vector representation and a trajectory approximation in a continuous goal recognition process, which we detail in the following subsections.

3.1 Polynomial Trajectory

In an obstacle-free environment, we only need a single trajectory to describe the movement of an agent between two points. However, in an environment with obstacles, in most cases, a single low-polynomial trajectory cannot reach the goal without violating some restrictions. A common approach to generating such trajectories while keeping the polynomial degree low is to compute separate sub-trajectories that, when sequenced, form a complete trajectory between the initial and desired final state [13]. To compute each sub-trajectory, we use the concept of a via point, which is a vector that includes some motion parameters, as shown in Eq. 7, where $i \in [1, 2, \ldots, q]$; $q \in \mathbb{N}^*$ is the number of via points in the trajectory; the terms in the vector v_i are the position, velocity, acceleration in the respectively Cartesian axes per via point i, and td_i is the time duration of a single sub-trajectory between via points i and $i+1$. Two via points have all the parameters to compute a single trajectory. We define a sequence of via points to generate a complete trajectory, starting with the desired initial state I and ending with the final goal state g, which we define in Eq. 8.

$$v_i = \begin{bmatrix} x_i & y_i & \dot{x}_i & \dot{y}_i & \ddot{x}_i & \ddot{y}_i & td_i \end{bmatrix}^T \tag{7}$$

$$V|_I^{g_n} = \begin{bmatrix} v_1 & v_2 & \ldots & v_q \end{bmatrix} \tag{8}$$

Next, we compute a trajectory between each via point of Eq. 8 using a model that approximates the agent dynamics. We chose a fifth-degree polynomial for two key reasons. First, this type of trajectory can handle the agent's dynamic constraints, such as position, velocity, and acceleration. Second, certain fifth-degree polynomials are the most complex polynomials without discontinuities that have computationally cheap analytical solutions. Eqs. 9-14 [3, Ch. 7] define the resulting model, where $x|_i^{i+1}(t)$ and $y|_i^{i+1}(t)$ are the positions, $\dot{x}|_i^{i+1}(t)$ and $\dot{y}|_i^{i+1}(t)$ are the velocities, and $\ddot{x}|_i^{i+1}(t)$ and $\ddot{y}|_i^{i+1}(t)$ are the accelerations in instant of time t in X and Y Cartesian axes; a_{fi} and b_{fi} are unknown coefficients where $f \in [1, 2, \ldots, 5]$. $x|_i^{i+1}(t)$ refers to the trajectory on the X axis (and respectively $y|_i^{i+1}(t)$ on the Y axis) with initial in v_i and final in v_{i+1} via points.

$$x|_i^{i+1}(t) = a_{5i}t^5 + a_{4i}t^4 + a_{3i}t^3 + a_{2i}t^2 + a_{1i}t + a_{0i} \tag{9}$$

$$y|_i^{i+1}(t) = b_{5i}t^5 + b_{4i}t^4 + b_{3i}t^3 + b_{2i}t^2 + b_{1i}t + b_{0i} \tag{10}$$

$$\dot{x}|_i^{i+1}(t) = 5a_{5i}t^4 + 4a_{4i}t^3 + 3a_{3i}t^2 + 2a_{2i}t + a_{1i} \tag{11}$$

$$\dot{y}|_i^{i+1}(t) = 5b_{5i}t^4 + 4b_{4i}t^3 + 3b_{3i}t^2 + 2b_{2i}t + b_{1i} \tag{12}$$

$$\ddot{x}|_i^{i+1}(t) = 20a_{5i}t^3 + 12a_{4i}t^2 + 6a_{3i}t + 2a_{2i} \tag{13}$$

$$\ddot{y}|_i^{i+1}(t) = 20b_{5i}t^3 + 12b_{4i}t^2 + 6b_{3i}t + 2b_{2i} \tag{14}$$

The coefficients a_{wi} and b_{wi} in the model trajectory of Eqs. 9-14 can be computed analytically by Eqs. 3.1-18, where x_i and x_{i+1} are the position; $\dot{x}_i$ and $\dot{x}_{i+1}$ are the velocity; $\ddot{x}_i$ and $\ddot{x}_{i+1}$ are the acceleration in via points v_i and v_{i+1}, respectively. For space, we omit the computation of the coefficients b_{wi} of Eqs. 10, 12, and 14, as they are exactly as above, but changing the work axes to Y.

$$a_{5i} = \frac{1}{2{td_i}^5}\left[td_i\left[(\ddot{x}_{i+1} - \ddot{x}_i)td_i - 6(\dot{x}_{i+1} + \dot{x}_i)\right], +12(x_{i+1} - x_i)\right], \tag{15}$$

$$a_{4i} = \frac{1}{2{td_i}^4}\left[td_i\left(16\dot{x}_i + 14\dot{x}_{i+1} + (3\ddot{x}_i - 2\ddot{x}_{i+1})td_i\right) +30(x_i - x_{i+1})\right], \tag{16}$$

$$a_{3i} = \frac{1}{2{td_i}^3}\left[td_i\left((\ddot{x}_{i+1} - 3\ddot{x}_i)td_i - 8\dot{x}_{i+1} - 12\dot{x}_i\right) +20(x_{i+1} - x_i)\right], \tag{17}$$

$$a_{2i} = \frac{\ddot{x}_i}{2}, \quad a_{1i} = \dot{x}_i, \quad a_{0i} = x_i. \tag{18}$$

After computing each trajectory of Eqs. 9 and 10 using the via points sequence of Eq. 8, we can concatenate all of them to synthesize a full path between initial and goal states as shown in Eqs. 19 and 20, where $\otimes$ is the concatenation operator; $X|_i^{i+1}$ and $Y|_i^{i+1}$ are the vectors from Eqs. 9 and 10, respectively. Thus, we derive the approximate

[2] In dynamical systems, a set of goal states is called a reference.

Algorithm 1 Continuous Online Goal Recognition in Vector Representation

1: **function** CONTINUOUSVECINFERENCE(W, I, G, V_{max}, k)
2: **for all** $g_n \in \mathcal{G}$ **do** ▷ Precompute approximate trajectories
3: $\mathcal{M}_{g_n} \leftarrow \emptyset$ ▷ Vector to save all solution trajectories
4: $\mathcal{V} \leftarrow$ generate k position parameters $RRT^*(W, I, g_n)$
5: **for each** trajectory $V|_I^{g_n} \in \mathcal{V}$ indexed by i **do**
6: $q \leftarrow |V|_I^{g_n}|$
7: $u \leftarrow$ OPTIMIZATION($V|_I^{g_n}, W, V_{max}$)
8: **for all** $j \in 0, \ldots, q-1$ **do**
9: $V|_I^{g_n}[j] \leftarrow$ insert remaining parameters $u[j]$
10: $\hat{m}_I^{g_n} \leftarrow$ COMPUTEPOLYTRAJECTORY($V|_I^{g_n}$)
11: $\mathcal{M}_{g_n}[i] \leftarrow \hat{m}_I^{g_n}$
12: **while** $New\, o_k \in O$ is available **do** ▷ Online recognition
13: **for all** $g_n \in \mathcal{G}$ **do**
14: $P(\mathcal{M}_{g_n} \mid O) \leftarrow$ COMPUTEPR($O, \mathcal{M}_{g_n}$)
15: **return** $\text{argmax}_{g_n} P(\mathcal{M}_{g_n} \mid O)$

trajectory by sequencing the vectors as Eq. 21.

$$X|_1^q = X|_1^2 \otimes X|_2^3 \otimes \ldots \otimes X|_{q-1}^q \tag{19}$$

$$Y|_1^q = Y|_1^2 \otimes Y|_2^3 \otimes \ldots \otimes Y|_{q-1}^q \tag{20}$$

$$\hat{m}_I^{g_n} = \begin{bmatrix} X|_1^q & Y|_1^q \end{bmatrix} \tag{21}$$

3.2 Finding via point parameters

Algorithm 1 brings together the computations necessary to compose full trajectories from via points. While we present the algorithm as a single function, in practice, the loop in Lines 2–11 takes place offline, whereas only the inference of Lines 12–14 takes place at recognition time. The algorithm starts by generating k different sequences of via point position parameters for all goals in Lines 2–4. To compose the trajectory using the via points from Eq. 8, we need to find all motion parameters of each via point in the sequence $V|_I^{g_n}$, i.e., position, velocity, acceleration, and time duration. The position can be obtained through an optimal geometric planner such as Rapidly-exploring Random Trees (RRT*) [12], Batch Informed Trees (BIT*) [7], and Sparse Roadmap Spanner (SPARS) [4], for example. Given the initial position I and the goal position g_n, a geometric planner can produce a sequence of via point position parameters, even in an environment with obstacles. As our approach uses multiple solutions in the inference process, Line 4 in Algorithm 1 generates k different sequences of via point position parameters for the same goal hypothesis and store them in a vector $\mathcal{V}$.

Line 7 in Algorithm 1 computes the remaining via point parameters of $V|_I^{g_n}$ by an RL-based optimization defined by Eqs. 22-24 that enforces the agent's dynamic constraints [1], where $h(s_i,\, u_i,\, s_{i+1}) = td_i$ is the cost function; V_{max} is a maximum velocity vector for the trajectory; s_i and u_i are states and actions set defined by the Eqs. 25-26, where $s_i,\, u_i \in \mathbb{R}^4$. Here, with one set of states s_i, actions u_i, and position via points in v_i and v_{i+1}, it is possible to compute a single trajectory of $X|_i^{i+1}$ and $Y|_i^{i+1}$ from Eqs. 19-20.

$$u^* = \underset{u \in U}{\text{argmin}} \left(J_u(s_i) \right), \tag{22}$$

$$J_\mu(s_i) = h(s_i, u_i, s_{i+1}) + J_\mu(s_{i+1}) \;\; i = 1,\, 2, \ldots,\, q-1. \tag{23}$$

$$subj.\ to : || \begin{bmatrix} \dot{x}|_i^{i+1}(t) & \dot{y}|_i^{i+1}(t) \end{bmatrix} || \leq V_{max},\; \forall t \in (0,\; td_i] \tag{24}$$

$$s_i = \begin{bmatrix} \dot{x}_i & \dot{y}_i & \ddot{x}_i & \ddot{y}_i \end{bmatrix}^T, \tag{25}$$

$$u_i = \begin{bmatrix} \dot{x}_{i+1} & \dot{y}_{i+1} & \ddot{x}_{i+1} & \ddot{y}_{i+1} & td_i \end{bmatrix}^T \tag{26}$$

We optimize to find high velocities while penalizing violation of its maximal constraint and to minimize the trajectories' overall time duration td_i. The optimization process from Eqs. 22-24 is often done incrementally. This requirement is due to the continuous states from the formulation. The resulting iterative optimization process finds the velocities, accelerations, and time duration terms of Eq. 8 for each v_i. We can use them to compose each via point vector v_i of Eq. 7 and finally build the $V|_I^{g_n}$ matrix from Eq. 8 that contains all via points necessary to compute a complete trajectory. Lines 8 and 9 of Algorithm 1 corresponds to this process. Using the $V|_I^{g_n}$ matrix, Algorithm 1 computes each trajectory sequence of Eqs. 19-20 and composes the full trajectory $\hat{m}_I^{g_n}$ of Eq. 21 as function COMPUTEPOLYTRAJECTORY in Line 10. Vector $\mathcal{M}_{g_n}$ stores the full trajectory $\hat{m}_I^{g_n}$ in Line 11. Lines 12-14 constitute the goal inference step at run-time with each new observation using the COMPUTEPR function, which implements Eqs. 5-6 for each goal hypothesis, i.e., it computes the average conditional probability among all trajectories for each hypothesis.

4 Experiments in Continuous Domains

We use a simple but realistic simulation to compare our method with the state-of-the-art in online goal recognition for continuous domains. Our experiments use 29 benchmark scenarios from Moving-AI [26] based on Starcraft maps[3]. All scenarios comprise a 512x512 pixel map and the group of points in axes X and Y used as goal hypotheses. Further details are available in the online supplement [29]. We conducted the experiments using a 2.2GHz six-core Intel i7 CPU with 24GB RAM, running Ubuntu 22.04.

Each scenario consists of a map and a number of points we can use as goal and initial states to define goal recognition problems. Scenario maps are $10m \times 10m$ Cartesian X and Y spaces with walls acting as obstacles. Each scenario contains eight randomly spread spatial points $p_n \in \mathbb{R}^3$ where $n \in [1, \ldots, 8]$, each of which containing x and y coordinates plus an orientation in radians. We select random points until we have eight that comply with two constraints to avoid trivial goal recognition settings: they must be at least $23cm$ away from any wall and $2m$ away from every other point.

We use one scenario with eight points deliberately distributed around the map in Figure 1a as a working example. White represents traversable space, and marked colored points represent potential initial and goal position points. The experiment samples observations from a simulation of a common robot with a two-wheeled motor and a unidirectional wheel defined by Eq. 27, where $\alpha(t)$ is the velocity control and $\omega(t)$ is the angular control rate. $x_r(t)$ and $y_r(t)$ are the positions in Cartesian axes and θ_r is the orientation in radians. We use a sampling period of 0.1 seconds and disregard dynamics such as wheel friction, motor dynamics, and elastic deformations.

$$\begin{aligned} \dot{x}_r(t) &= \alpha(t) cos(\theta(t)), \\ \dot{y}_r(t) &= \alpha(t) sin(\theta(t)), \\ \dot{\theta}_r(t) &= \omega(t). \end{aligned} \tag{27}$$

We generate recognition problems in each map using all combinations of the points in the scenario, with the remaining points being goal hypotheses. Thus, we have one problem where the ground truth is a trajectory from p_1 to p_2 (whose x, y coordinates we call g_2),

[3] http://movingai.com/benchmarks/sc1/index.html

with p_2 to p_8 being goal hypotheses, another one with p_8 to p_7 (*q.v.* g_7) with p_1 to p_7 as goal hypotheses, and so on. This yields 56 goal recognition problems per map using the Cartesian positions x and y of each point p_n from Figure 1a.

We assume agents optimize total motion time following the dynamical robot model of Eq. 27 defined by Eqs. 28-32, where tf is the total simulation time; ω_{lim} is the maximal angular velocity; g_n is a goal in Cartesian position x, y; $wall(x_r(t), y_r(t))$ is a function that measures the Euclidean distance from the robot position to its nearest wall obstacle at time t; $wall_{lim}$ is the minimum separation between the robot and an obstacle. In our experiments $\omega_{lim} = 3$ rad/s, g_n is sampled from p_n with free angular orientation, and $wall_{lim}$ is 0.01 meters for all experiments. We represent the complete observation from the initial state I to the goal state g_n as $O_I^{g_n}$. Figure 1b exemplifies the robot's optimal trajectory obtained through optimization in the simulated environment. In this example, the robot is pursuing the goal point g_2 from the initial point p_1 and the recognition process has full observability so that $O_I^{g_n} = m_{p_1}^{g_2 *}$.

$$\alpha^*, \omega^* = \underset{\alpha,\ \omega}{\operatorname{argmin}}\ tf \tag{28}$$

$$subj.\ to: \quad \begin{bmatrix} x_r(tf) & y_r(tf) \end{bmatrix} = g_n \tag{29}$$

$$|\alpha(t)| \leq V_{max}, \tag{30}$$

$$|\omega(t)| \leq \omega_{lim}, \tag{31}$$

$$wall(x_r(t), y_r(t)) \geq wall_{lim}\ \forall\ t \in [0,\ tf] \tag{32}$$

4.1 Computing the Via Point Parameters

We use the Open Motion Planning Library (OMPL) [27] with the RRT^* geometric planning algorithm to compute all the position parameters of Eq. 8. This off-the-shelf planner provides a cost-minimal sequence of via point positions from an initial position to a goal. Calls to RRT^* have a 5-second time limit and use distance as the cost function so that the via points are part of one shortest path to the goal. Figure 1c shows an example of an output provided for the RRT^* planner, where the initial and goal states are p_1 and g_2, respectively. Circles are the via points positions from RRT^*, and the dashed line connects them in sequence

Next, we need to find the velocity parameters of each via point. We implement the optimization from Eqs. 22-24 in SciPy in its default configuration. The optimization settings are: maximum velocity vector of $V_{max} = 1$; random initial actions u; all acceleration terms in the via points being zero.

We use the via points to compute the approximate trajectory $\hat{m}_I^g$ from Eq. 21. Figure 1d shows the trajectory difference between an estimated trajectory $\hat{m}_{p_1}^{g_2}$ and its respectively observation $O_{p_1}^{g_2}$. The final stage of our method is to compute the conditional probability of Eq. 4 using the estimate trajectories $\hat{m}_I^{g_n}$ instead of the optimal for each goal at each new observation, allowing us to infer the most likely goal hypotheses. Figure 2 illustrates, in our working example, the conditional probability values $P(\hat{m}_{p_1}^{g_n} \mid O_{p_1}^{g_2})$ of Eq. 4 for all goal points in the set over time.

4.2 Results

Table 1 compares our method (Vector) with the Mirroring of Kaminka [11] (Mirroring) and the Recompute plus Prune (R+P) of Vered [30], in all goal recognition problems for all 28 scenarios. Mirroring requires $(|O|+1)|G|$ planner calls, whereas R+P requires between $|G|$ and $(|O|+1)|G|$ by heuristically deciding when to use the planner. The table shows average values over all scenarios and reports the positive predictive value (PPV) of the correct goal in percentage, the accuracy (ACC) of the predictions, the spread of goals in the output (SPR), the number of planner calls (PC) required to infer each goal, and the runtime of each method. For each metric, we report their mean throughout the experiments and their standard deviation in parentheses. We separate the algorithm's online and offline parts to highlight the key advantage of our method. Table 1 also shows the results of our method changing the number of solution trajectories k used in inference. The supplementary material breaks down these results for each scenario.

	PPV (%)	ACC (%)	SPR	PC	Online Time(s)	Offline Time(s)
Mirr.	41.1(9.3)	85.2(2.3)	1.1(0)	49(0)	1.0e4(5.7e3)	2.4e3(1.6e3)
R+P	44.3(9.0)	85.9(2.2)	1.1(0.04)	29.5(2.7)	4.8e3(3.0e3)	2.3e3(1.5e3)
Vec k=1	41.6(7.8)	85.4(1.9)	1.0(0)	7.0(0)	8.9e-2(1.0e-2)	1.0e2(40.2)
Vec k=5	47.7(9.2)	86.9(2.3)	1.0(0)	7.0(0)	3.8e-2(1.3e-3)	1.7e2(53.6)
Vec k=10	48.5(9.3)	87.1(2.3)	1.0(0)	7.0(0)	4.1e-2(1.4e-3)	2.1e2(40.3)
Vec k=15	49.7(8.6)	87.4(2.1)	1.0(0)	7.0(0)	3.7e-2(8.2e-4)	2.2e2(41.2)
Vec k=20	49.8(9.3)	87.4(2.3)	1.0(0)	7.0(0)	3.8e-2(8.0e-4)	2.4e2(45.8)

Table 1: Comparison among online goal recognition methods for continuous domains.

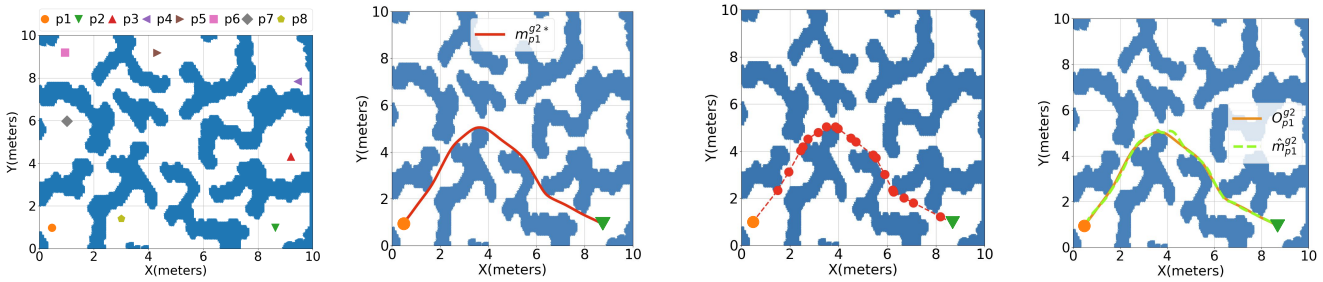
To compare online recognition performance over time, we divide each of the 56 observable trajectories (all combinations of initial states and goals) into six points equally spaced in time, named test observations. Thus, each problem has six sampled points used as observations ($\{o_1, \ldots, o_6\}$, omitting the final observation that indicates the goal), which we can use to measure online recognition accuracy (convergence to the correct goal) over time. To illustrate this, we go back to our working example with points deliberately distributed in the scenario from Figure 1a. Figure 3a shows the six sample observations (red dots) over a complete trajectory (dashed line) with initial and goal states as p_1 and g_2. Figure 3b shows the average goal recognition PPV (and its margin of error with a confidence level of 95%) at each of the six sample observations for each method. Results indicate that our method has better positive predictive value and accuracy within the standard deviation for all fractions of observations.

To better illustrate a case where the motion problem has more than one trajectory as a solution, we contrive a scenario where many combinations of initial states and goals will lead to two optimal, completely distinct trajectories. We compare our method Vector with Mirroring and R+P to show how each method deals with this challenging recognition problem. While the results for these problems are present in the aggregated results in this section, we provide further detail about them in the supplementary material [29].

In conclusion, our approach using $k = 1$ is competitive with the state of the art, with a marginal advantage in accuracy and positive predictive value within the standard deviation. Importantly, our approach offers a substantial speed-up. While the offline computations have almost an order of magnitude speed up, the online computations improve by six orders of magnitude. Besides the improvements for the basic case of our algorithm, the addition of multiple solutions (i.e., increasing $k > 1$) in the inference process brings an increase in accuracy and positive predictive value. The results show that with $k = 5$, we widen the performance gap; increasing the k value further away only brings a minor increase in performance, and we can see a stabilization around $k = 15$.

5 Discrete Domains

Our approach so far works exclusively in continuous Euclidean state spaces represented as numeric vectors. While applying it directly to discrete domains is not trivial, we now show how converting the dis-



(a) Starcraft's BigGameHunters map. Marks represent potential positions.

(b) Robot optimal trajectory obtained thought optimization from RRT^* considering initial point p_1 to goal point g_2.

(c) Example of output from initial and goal states as p_1 and g_2.

(d) Comparison between the approximated trajectory $\hat{m}_{p1}^{g2}$ and the observation O_{p1}^{g2}.

Figure 1: Example of trajectories generated in different stages of the experimental scenario.

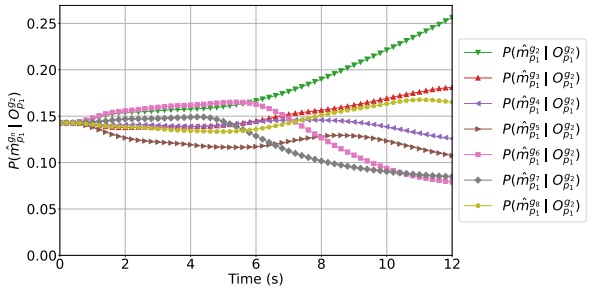


Figure 2: Conditional probabilities $P(\hat{m}_{p1}^{g_n} \mid O_{p1}^{g_2})$ for all goals in problems starting at p_1 and goal point at g_2.

crete state into a vectorial continuous state space representation allows us to apply this inference in such domains. STRIPS-style domains are the largest goal recognition datasets openly available [18].

We consider a discrete domain to be a STRIPS-style PDDL (Planning Domain Definition Language) description with the same semantics of Fikes et al. [5] as $\mathcal{D}$, with typed objects set Z and a set of typed predicates $\mathcal{R}$. A classical planning problem can be described as $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$, where: $\mathcal{F}$ is the set of facts (instantiated predicates from Z); $\mathcal{A}$ is the set actions with objects from Z; $\mathcal{I} \in 2^{\mathcal{F}}$ is the initial state; and $\mathcal{G} \subseteq 2^{\mathcal{F}}$ is the goal state. A discrete goal recognition problem is a tuple $\mathcal{L} = \langle \mathcal{D}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$, where: $\mathcal{D}$ is the domain defined above; $\mathcal{O} \subseteq \mathcal{F}$ is a set of observations. A trajectory in a discrete domain is a sequence of ordered states, while a plan π is a sequence of ordered actions. Plans here match exactly their definition from classical planning [8].

While not having the same problem of infinite possible approximately optimal plans (for non-zero action costs), discrete domains often have multiple optimal plans to achieve a goal from a given initial state. Therefore, we apply the same approach used in continuous domain inference in the discrete domain, i.e., we consider multiple solutions with the same goal hypotheses in our inference process. Our approach for discrete domains takes inspiration from Sohrabi et al. [24] to use a Top-k planner to compute multiple plans. Top-k planners compute up to k plans that are a solution to a problem P, if P has more than k possible solutions [25].

Algorithm 2 summarizes our approach in discrete domains. As with Algorithm 1, we condense both the offline and online part in the same pseudocode, while in practice these are computed separately. Lines 2–7 comprise the offline part of the algorithm, whereas Lines 8–11 are the online part. The offline part of the algorithm starts by using the Top-k planner to generate k plans π_{g_n}, for a goal hypothesis g_n and store them into vector Π (Line 4). The algorithm then rolls out each solution plan π_{g_n} from Π, generating a trajectory $m_I^{g_n}$ which it stores in vector $\mathcal{M}_{g_n}$. Lines 6–7 represent this process. In the online inference, at each new observation, the algorithm computes the average conditional probability among all trajectories for each goal hypothesis following the Bayesian formulation of goal recognition. Like in the continuous domain, function ComputePr (Line 10) implements Eqs. 5-6 for each goal hypothesis.

Algorithm 2 Discrete Online Goal Recognition in Vector Representation

Require: $\mathcal{P} = \langle \mathcal{F}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle, k$

1: **function** OnlineVectorInference($\mathcal{P}, k$)
2: **for all** $g_n \in \mathcal{G}$ **do** ▷ Precompute optimal plans
3: $\mathcal{M}_{g_n} \leftarrow \emptyset$ ▷ Vector to save all solution trajectories
4: $\Pi \leftarrow$ Planner($\mathcal{P}, g_n, k$) ▷ Generate k plans
5: **for each** plan $\pi_{g_n} \in \Pi$ indexed by i **do**
6: $m_I^{g_n} \leftarrow$ rollout($\mathcal{P}, \pi_{g_n}$)
7: $\mathcal{M}_{g_n}[i] \leftarrow m_I^{g_n}$
8: **while** $New\ o_k \in \mathcal{O}$ is available **do** ▷ Online recognition
9: **for all** $g_n \in \mathcal{G}$ **do**
10: $P(\mathcal{M}_{g_n} \mid O) \leftarrow$ ComputePr($\mathcal{O}, \mathcal{M}_{g_n}$)
11: **return** $\text{argmax}_{g_n}\ P(\mathcal{M}_{g_n} \mid O)$

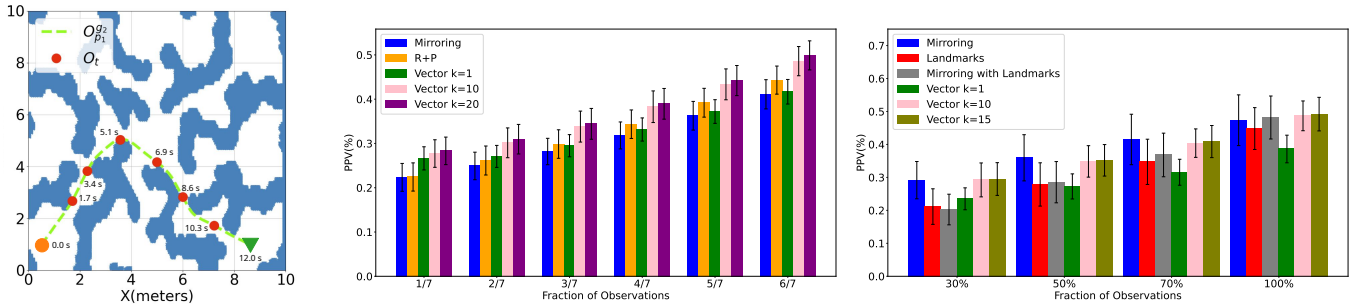
The difference from the continuous version of our approach is in the computation of the Euclidean distance of Eq. 5. To compute the distance between two sets of predicates (the observation and the computed state of a trajectory), we use the function of Eq. 33. This computes the Euclidean distance between the two states directly from the STRIPS format of these states. Note that the expression within the modulo operator is the Symmetric difference.

$$dist = \sqrt{|\ (o - m_I^{g_n}) \cup (m_I^{g_n} - o))\ |} \tag{33}$$

6 Experiments in Discrete Domains

We evaluate our inference method empirically against three online goal recognition methods. The first *Mirroring* by Kaminka [11] (Mirr.). The second and third are *Goal Recognition with Landmarks* (Land.) and *Goal Mirroring with Landmarks* both by Vered [31] (GM+L), where the last one is the current state-of-the-art. Our experiments use an openly available goal and plan recognition dataset [18], which contains thousands of recognition problems comprising large and non-trivial planning problems in the STRIPS fragment of PDDL (with optimal and sub-optimal plans as observations), including domains and problems from datasets from Ramırez and Geffner 2009. Domains include realistic applications (e.g., DWR, ROVERS, LOGISTICS), and hard artificial domains (e.g., SOKOBAN).

Table 2 shows the result of our empirical evaluation of these methods against our online goal recognition formulation for discrete do-



(a) Trajectory example for O^{g2}_{p1} with sampled observations.

(b) PPV percent and margin of error comparison from continuous domains over observations.

(c) PPV percent and margin of error comparison from discrete domains over observations.

Figure 3: Comparison between on-line goal recognition methods.

	PPV (%)	ACC (%)	SPR	PC	Online Time(s)	Offline Time(s)
Mirr.	47.3(13.6)	81.8(8.1)	2.5(1.2)	124.8(57.4)	80.4(117.6)	4.6(5.3)
Land.	44.8(11.2)	82.7(5.2)	1.6(0.51)	0.0(0)	5.7e-1(6.1e-1)	1.3e-1(2.6e-1)
GM+L	48.2(11.5)	84.7(5.0)	1.3(0.47)	29.9(7.9)	12.5(12.5)	4.8(5.8)
Vec k=1	38.6(7.4)	76.3(7.6)	2.4(0.52)	8.2(3.9)	4.9e-3(8.4e-3)	6.8(6.9)
Vec k=5	45.9(7.6)	81.7(6.7)	1.9(0.44)	8.2(3.9)	2.1e-2(4.1e-2)	14.5(21.9)
Vec k=10	48.7(7.9)	83.7(5.9)	1.7(0.45)	8.2(3.9)	4.4e-2(9.2e-2)	23.5(41.1)
Vec k=15	49.2(8.9)	84.2(5.7)	1.7(0.44)	8.2(3.9)	6.4e-2(1.3e-1)	33.66(63.7)

Table 2: Comparison among online goal recognition methods for discrete domains.

mains. All results are averages over all problems in every experimented domain, and report positive predictive value (PPV); overall accuracy (ACC) for each experiment; spread (SPR) is the size of the hypothesis solution set chosen by recognition method and (PC) is the number of planner calls during the whole recognition process. The online supplement [29] breaks the results down for each domain. Table 2 also shows results when our method uses k solution plans during inference for $k \in [1, 5, 10, 15]$. These values correspond respectively to 3.0%, 15.2%, 30.4%, and 45.6% of the total number of optimal solution plans on average over all problems. Our implementation uses the SymK[4] planner with the forward-search option. SymK is a state-of-the-art optimal Top-k planner based on symbolic search that extends Fast Downward [25].

The results show that our approach can be competitive with the state-of-the-art while being much faster in inference (online) time, trading off in offline processing time. Likewise, the results in the continuous domain show that with $k = 5$, our performance improves substantially; whereas increasing the k value further only brings a minor increase in performance, and we can see a stabilization around $k = 10$. The parameterization of k does not consider whether all k plans are indeed optimal. Indeed, increasing k might not necessarily improve performance when the number of optimal plans for a problem is fewer than k. Additional experiments show that filtering the k plans for optimality improves PPV by 4.5%.

These results confirm that our method can be easily applied to discrete domains while retaining its strengths. Figure 3c compares results among the methods at four points during observation (and its margin of error with a confidence level of 95%), at 30%, 50%, 70%, and 100% of their respective full observation. All results shown in Figure 3c are averages over the problems of each domain, *e.g.*, partial observation problems (30%, 50%, and 70%) have an average of 12 problems for each domain, and the full observation has an average length of 12 actions. Results indicate that our approach is faster across the board in the online phase, using substantially fewer calls to the planner than all other approaches. Importantly, our approach provides superior accuracy and PPV, with a marginally higher spread.

[4] `https://github.com/speckdavid/symk`

7 Related Work

Recent work in goal recognition follow the general approaches developed by Kaminka [11] and Masters [14], which focuses on comparing the cost functions between an optimal trajectory and a trajectory following the observations. Fitzpatrick [6] has a different approach applied in a scenario without obstacles; the inference measures the error between a computed trajectory and the observations using Euclidean distance. Ignoring obstacles, however, is a major limitation for realistic (and widespread) ground navigation scenarios. Obstacle avoidance, however, creates additional challenges in goal recognition, for instance multiple sub-solutions, necessitating a goal recognizer to reason about different possible solutions available to an agent. This paper addresses such issues by reasoning over top-k plans.

8 Conclusion

We introduce an approach for online goal recognition suitable in continuous and discrete domains. Our approach is suitable for recognizing agent motion in continuous environments with obstacles. If we know the observed agent's potential initial states and a set of possible goals a priori, we can execute the costliest computations in an offline stage. This allows computation of the inference process through a simple equation in milliseconds at each new observation, providing the probability distribution over the goals. We develop a mechanism for discrete domains to convert STRIPS-style problems into vectors amenable to our function approximation.

Empirical evaluation shows our method is six orders of magnitude faster than the state-of-the-art in the online stage and four times faster than the state-of-the-art in the preprocessing stage for the continuous case. This advantage in execution time is due to our method using fewer planner calls ($|G|$), replacing most of the original planner calls with an approximate motion model. While our offline preprocessing time is higher than the fastest methods for discrete domains, it is the only approach with a similar runtime that works for both types of domains. The major drawback of using an approximation is trading speed for accuracy. However, any method that takes minutes to perform online goal recognition in a moving robot is impractical. Thus, this paper sets us up to a new class of goal recognition methods suitable for applications in continuous domains where recognition must happen in milliseconds.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) - Finance Code 001.

References

[1] D. Bertsekas. *Reinforcement learning and optimal control*. Athena Scientific, 2019.

[2] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, and W. Burgard. *Principles of robot motion: theory, algorithms, and implementations*. MIT press, 2005.

[3] J. J. Craig. *Introduction to robotics: mechanics and control*. Pearson Education, 2005.

[4] A. Dobson and K. E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research*, 33(1):18–47, 2014.

[5] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2 (3-4):189–208, 1971.

[6] G. Fitzpatrick, N. Lipovetzky, M. Papasimeon, M. Ramirez, and M. Vered. Behaviour recognition with kinodynamic planning over continuous domains. *Frontiers in Artificial Intelligence*, 4, 2021.

[7] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa. Batch Informed Trees (BIT*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*, 39(5):543–567, 2020.

[8] M. Ghallab, D. S. Nau, and P. Traverso. *Automated planning - theory and practice*. Elsevier, 2004.

[9] R. V. Hogg, J. W. McKean, and A. T. Craig. *Introduction to mathematical statistics*. Pearson Education, 2019.

[10] A. Jain, L. Chan, D. S. Brown, and A. D. Dragan. Optimal cost design for model predictive control. In *Learning for Dynamics and Control*, pages 1205–1217. Proceedings of Machine Learning Research, 2021.

[11] G. A. Kaminka, M. Vered, and N. Agmon. Plan recognition in continuous domains. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2018.

[12] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30 (7):846–894, 2011.

[13] K. M. Lynch and F. C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[14] P. Masters and S. Sardina. Cost-based goal recognition for path-planning. In *Sixteenth Conference on Autonomous Agents and Multi-Agent Systems*, pages 750–758, 2017.

[15] F. Meneguzzi and R. F. Pereira. A survey on goal recognition as planning. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2021.

[16] R. Mirsky, S. Keren, and C. Geib. Introduction to symbolic plan and goal recognition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 16(1):1–190, 2021.

[17] S. B. Niku. *Introduction to robotics: analysis, control, applications*. John Wiley & Sons, 2020.

[18] R. Pereira, N. Oren, and F. Meneguzzi. Landmark-based heuristics for goal recognition. In *Thirty-First AAAI Conference on Artificial Intelligence*, volume 31, 2017.

[19] R. Pereira, N. Oren, and F. Meneguzzi. Landmark-based approaches for goal recognition as planning. *Artificial Intelligence*, 279:103217, 12 2019.

[20] M. Pivtoraiko, R. A. Knepper, and A. Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[21] M. Ramırez and H. Geffner. Plan recognition as planning. In *Twenty-First International Joint Conference on Artificial Intelligence*, pages 1778–1783. Citeseer, 2009.

[22] M. Ramírez and H. Geffner. Probabilistic plan recognition using off-the-shelf classical planners. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[23] W. Schwarting, J. Alonso-Mora, and D. Rus. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, 1(1):187–210, 2018.

[24] S. Sohrabi, A. V. Riabov, and O. Udrea. Plan recognition as planning revisited. In *Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 3258–3264, 2016.

[25] D. Speck, R. Mattmüller, and B. Nebel. Symbolic top-k planning. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 9967–9974, 2020.

[26] N. R. Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[27] I. A. Şucan, M. Moll, and L. E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012.

[28] G. Sukthankar, C. Geib, H. H. Bui, D. Pynadath, and R. P. Goldman. *Plan, activity, and intent recognition: Theory and practice*. Elsevier, 2014.

[29] D. Tesch, L. R. Amado, and F. Meneguzzi. Real-time goal recognition using approximations in euclidean space. *arXiv preprint arXiv:2307.07876*, 2023. Extended version of this paper.

[30] M. Vered and G. A. Kaminka. Heuristic online goal recognition in continuous domains. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*, pages 4447–4454, 2017.

[31] M. Vered, R. F. Pereira, G. Kaminka, and F. R. Meneguzzi. Towards online goal recognition combining goal mirroring and landmarks. In *Seventeenth International Conference on Autonomous Agents and MultiAgent Systems*, page 10–15, 2018.

[32] Y. Zhang, S. Sreedharan, A. Kulkarni, T. Chakraborti, H. H. Zhuo, and S. Kambhampati. Plan explicability and predictability for robot task planning. In *IEEE International Conference on Robotics and Automation*, pages 1313–1320, 2017.