Dynamic Programming Approach for Motion Planning with Arrival Requirements

Ty Nguyen and **Tsz-Chiu Au**
School of Electrical and Computer Engineering
Ulsan National Institute of Science and Technology
{tynguyen,chiu}@unist.ac.kr

Abstract

Motion planning with predictable timing and velocity is crucial in certain tasks in multi-robot coordination, such as role assignment and formation positioning in robot soccer and autonomous intersection management. This paper introduces a dynamic programming algorithm that is polynomial-time in finding the set of feasible setpoint schedules for a mobile robot to reach a specific position on a trajectory at a specific time and a specific velocity. Although the running time of the algorithm is longer than that of the bisection method we proposed previously, the amortized cost of plan generations is much smaller, making it suitable for real time update of setpoint schedules to cope with control and sensing errors.

Introduction

We consider the problem of controlling a mobile robot such as an autonomous vehicle to arrive at a specific position on a one dimensional trajectory at a given arrival time and a given arrival velocity. This motion control is fundamental in a number of multi-robot systems, in particular autonomous intersection management (AIM) which coordinates vehicles to enter an intersection in unison, leading to a much lower traffic delay than traffic signals and stop signs (Dresner and Stone 2008). In some sport games such as robot soccer, the question of whether a player can move to a target position at certain time and at certain velocity to hit a ball is important in role assignment and formation positioning (MacAlpine, Price, and Stone 2015). Some general-purpose motion planning algorithms are capable of satisfying both the arrival time and arrival velocity requirements, but few of them are efficient, complete algorithms. Johnson and Hauser (2012) devised a complete planner that computes collision-free, time-optimal, longitudinal control sequences for meeting arrival time and velocity requirements. However, this problem is different from ours as it focuses on computing time-optimal trajectories. Recently, Au et al. introduced a bisection method to construct a setpoint schedule, based on vehicles' acceleration and deceleration profiling, to control a vehicle to arrive at a destination at a given time and velocity (Au, Quinlan, and Stone 2012). In this paper, we will show that this problem can be also solved by dynamic programming with a proper discretization of the continuous configuration space.

Dynamic programming (DP) has been adopted to solve a wide variety of planning problems. This paper compares the bisection method with the DP algorithm in motion planning with arrival requirements. First, we present the DP algorithm for solving the decision version of the problem (a.k.a. the validation problem in (Au, Quinlan, and Stone 2012)). Second, we conduct simulation experiments to compare the precision and running times of the two approaches. In the experimental results, we identify a trade off between the execution time of the DP algorithm and the precision in arrival time and velocity: to achieve a high precision, the DP algorithm needs a much longer setup time than the bisection method. However, the plan extraction step takes much less time than the bisection method. Given that the procedure is expected to run repeatedly, as described in (Au, Quinlan, and Stone 2012), in order to cope with the sensing and control errors in vehicle control, the DP algorithm has a lower amortized running time than the bisection method.

Computing the Feasible Set

Given the specification $(D, T^{\mathsf{stable}}, D^{\mathsf{stable}})$ of a road ρ and the starting velocity v_0, a configuration (t, v) is *feasible* for ρ if a robot can reach the end of ρ at time t and at velocity v. Here we want to find the set of all feasible configurations for a road with any given starting velocity v_0. We denote that set by $\mathbb{F}(v_0, D)$, and call $\mathbb{F}$ the *feasible set table* of ρ.

Computing the feasible set is not straightforward, because T^{stable} and D^{stable} can be arbitrary functions. We cope with this problem by introducing a DP algorithm for computing the feasible set, based on a discretization of time, velocity, and distance. Let $\mathbb{V} = \{v_0, v_1, \ldots, v_{m_v}\}$, $\mathbb{T} = \{t_0, t_1, \ldots, t_{m_t}\}$, and $\mathbb{D} = \{d_0, d_1, \ldots, d_{m_d}\}$ be a finite set of velocities, times, and lengths, respectively, so that $\{v \times t : \forall v \in \mathbb{V} \text{ and } \forall t \in \mathbb{T}\} \subseteq \mathbb{D}$.

Let $\mathbb{F}$ be a $|\mathbb{V}| \times |\mathbb{D}|$ table such that $\mathbb{F}(v_0, d)$ is the feasible set if the starting velocity is v_0 and the length of ρ is d. $\mathbb{F}$ can be defined recursively as follows. When $d = 0$, $\mathbb{F}(v_0, 0) = \{(0, v_0)\}$. When $d > 0$,

$$\mathbb{F}(v_0, d) = \left[\bigcup_{v_{\mathsf{int}} \in V \setminus \{v_0\}} \mathbb{F}^1(v_0, d, v_{\mathsf{int}}) \right] \cup \mathbb{F}^2(v_0, d) \qquad (1)$$

$$\mathbb{F}^1(v_0, d, v_{\mathsf{int}}) = \begin{cases} \textsc{AddTime}(\mathbb{F}(v_{\mathsf{int}}, d - D^{\mathsf{stable}}(v_0, v_{\mathsf{int}})), \\ \qquad T^{\mathsf{stable}}(v_0, v_{\mathsf{int}})) \quad \text{if } D^{\mathsf{stable}}(v_0, v_{\mathsf{int}}) \le d, \\ \emptyset \qquad\qquad \text{otherwise,} \end{cases}$$

$$\mathbb{F}^2(v_0, d) = \begin{cases} \textsc{AddTime}(\mathbb{F}(v_0, d - \delta d_{v_0}), 1) & \text{if } \delta d_{v_0} \le d, \\ \emptyset & \text{otherwise,} \end{cases}$$

where $\textsc{AddTime}(F, \delta t) = \{ (t+\delta t, v) : \forall (t,v) \in F \}$ and $\delta d_{v_0} = v_0 \times \delta t$, in which δt is the time interval in discretization. Based on Equation 1, we can devise a DP algorithm to compute $\mathbb{F}$. We omit the pseudo-code of the algorithm due to the space limitation. The running time of the algorithm is $O(|\mathbb{T}| \times |\mathbb{V}|^3 \times |\mathbb{D}|)$.

Experimental Evaluation

We compared the performance of the DP algorithm with that of the bisection method in a vehicle simulator implemented using the PyGame library. External conditions such as road friction and air resistance were taken into account in the simulation. We implemented a PID controller and two setpoint schedulers, one uses the DP algorithm and the other uses the bisection method, to control a vehicle to reach a position on a road. We measured the running time of algorithms as well as the errors in arrival times and arrival velocity.

Given a road distance d, an experimental trial consists of the following steps: 1) randomly choose a starting velocity v_0, an arrival time t_{end}, and an arrival velocity v_{end}; 2) generate two setpoint schedules, one by the DP algorithm and the other by the bisection method; 3) If both methods fail to generate a setpoint schedule, we go back to Step 1 and choose another problem to solve; and 4) execute the setpoint schedules in the vehicle simulator to control the vehicle to arrive at the destination.

In our experiment, we chose three different values of d: $50m$, $70m$, and $100m$. For each d, we did 30 trials using the bisection method, and another 30 trials using the DP algorithm. The latter is divided into three groups, each has a different discretization. The discretization is controlled by a parameter called the *step size*. The smaller the step size is, the finer the discretization is. For instance, if the step size is 0.1, one second will be discretized into 10 time steps with a time interval of 0.1s. If the step size is 0.2, one second will be discretized into 5 time steps with a time interval of 0.2s. Similarly, if the step size is 0.1, 1m will be discretized into 10 distance steps with a distance interval of 0.1m. For each trials, we measured the running times of the algorithms. For the DP algorithm, we measured the running times of the computation of $\mathbb{F}$ and plan extraction separately. We also measured the errors between the arrival time and velocity and the proposed arrival time and velocity. The results are shown in Tables 1 and 2.

As can be seen, the setpoint schedules generated by the DP algorithm have a higher accuracy in arrival time. while the errors in the arrival velocity are quite similar. However, the DP algorithm takes more time to compute the feasible set table than the bisection method. But once the feasible set table is computed, we can extract a solution plan and construct a setpoint schedule quickly. The DP algorithm will

Table 1: Errors and the running time of the DP algorithm.

Step	Error		Running Time (s)	
Size	Time Error (s)	Velocity Error (m/s)	$\mathbb{F}$ Creation	Plan Extraction
0.1	1.194 ± 0.433	1.953 ± 3.235	309.691	0.004467
0.2	1.215 ± 0.357	1.964 ± 2.207	74.68	0.002712
0.4	1.309 ± 0.742	2.229 ± 4.364	29.861	0.002103

Table 2: Errors and Running Time of Bisection Method

Time Error (s)	Velocity Error (m/s)	Running Time (s)
-3.246 ± 0.385	2.045 ± 1.574	0.17607

be much slower if we use it once only. However, in practice a vehicle will run its setpoint scheduler repeatedly since it suffers from sensing and control errors and needs to update its setpoint schedule from time to time. For instance, suppose the update frequency is 10Hz (i.e., 10 updates per second). If a vehicle runs for n seconds, the total running time of the DP algorithm with step size 0.4 is $29.9 + 0.021n$ on average, whereas the total running time of the bisection method is $1.7n$. Hence, when $n \ge 17.65s$, the DP algorithm has a smaller running time. However, using finer discretization to reduce the time and velocity errors will also increase the time it takes to break even with the bisection method.

Conclusions and Future Work

Motion planning with predictable timing and velocity will enable a number of interesting applications in multi-robot systems. In this paper, we proposed a novel, polynomial-time algorithm for setpoint scheduling with guarantees on arrival time and velocity, and compared it with the bisection method we previously proposed. According to our experiments, to achieve a high precision in arrival time and velocity, the dynamic programing approach needs a fine discretization of a continuous configuration space, resulting in a high setup time. However, once the algorithm generates the solution table, extracting a setpoint schedule from the table can take much less time than the bisection method, making the dynamic programing approach more suitable for repeated use. In the future, we intend to improve the running time of the DP algorithm.

References

[Au, Quinlan, and Stone 2012] Au, T.-C.; Quinlan, M.; and Stone, P. 2012. Setpoint scheduling for autonomous vehicle controllers. In *ICRA*, 2055–2060.

[Dresner and Stone 2008] Dresner, K., and Stone, P. 2008. A multiagent approach to autonomous intersection management. *Journal of Artificial Intelligence Research (JAIR)*.

[Johnson and Hauser 2012] Johnson, J., and Hauser, K. 2012. Optimal acceleration-bounded trajectory planning in dynamic environments along a specified path. In *ICRA*, 2035–2041.

[MacAlpine, Price, and Stone 2015] MacAlpine, P.; Price, E.; and Stone, P. 2015. Scram: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning. In *AAAI*.