ICRA2026 Late Breaking Results Poster presented at
2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Cascading Velocity Modulation for Multi-Agent Path Finding Execution

SeungHyun Park[1], Jaehong Shim[2], and †Gyuho Eoh[3]

***Abstract*—Multi-Agent Path Finding (MAPF) plans are increasingly deployed on real multi-robot fleets, where communication dropouts, actuator faults, and sensor noise routinely cause individual robots to deviate from the planned trajectory. We propose Cascading Velocity Modulation (CVM), a continuous execution controller that maps the temporal margin on each dependency edge into a proportional velocity command and propagates an exponentially attenuated damping signal along the dependency chain. CVM runs a three-step control loop: self-recovery, direct cushioning, and cascade propagation. CVM reduces the makespan by about 25 percent on average compared to a binary baseline, over ten randomized scenarios with 5 to 8 agents, each containing a malfunctioning agent that suffers an unexpected delay. An experiment with eight e-puck2 robots reproduces about a 35 percent reduction under two simultaneously malfunctioning agents.**

I. Introduction

MAPF solvers produce collision-free trajectories under the assumption that every robot reaches every waypoint on time, but on real hardware communication dropouts, actuator faults, and sensor noise routinely violate that assumption, so robust execution needs an online layer that adapts the offline plan to observed deviations.

A widely used online layer, ADG [1], reacts with a binary go/wait controller: as soon as the margin on any dependency drops to zero, the dependent agent fully stops. This is safe but unnecessarily aggressive, turning short intermittent slowdowns into stop-and-go waves that ripple through the team. Temporal plan graphs BTPG [2], STPG [3], and Wink-TPG [4] optimize the *passing order* at contested resources, but none modulate speed continuously.

We propose *continuous velocity modulation on the dependency graph*, which adjusts each agent's speed in proportion to the temporal margin τ on its incoming dependencies and propagates the slowdown along the chain. Fig. 1 shows the concept. In both simulation and an experiment with eight e-puck2 robots, CVM reduces the makespan by about 30 % over a binary baseline.

II. Method

We use a rotation-aware variant of CCBS [5] to obtain a plan $\mathcal{P}$, and extract a dependency graph $G = (V, E)$ from

[1]Department of Mechatronics Engineering, Tech University of Korea
[2]Department of Mechatronics Engineering, Tech University of Korea
[3]Department of Mechatronics Engineering, Tech University of Korea
†For correspondence and questions: gyuho.eoh@tukorea.ac.kr.

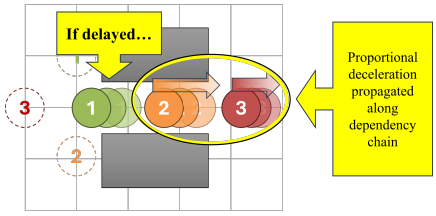

Fig. 1. Concept of proportional delay propagation along a dependency chain.

sequential resource usage. Each cycle, from the observed scheduling delay δ_i of each agent we compute the temporal margin

$$\tau(i \to j) = m + (\delta_j - \delta_i) - \sigma \tag{1}$$

of each edge, where m is the planned margin and σ a safety buffer. The multiplier $\rho(\tau, \theta) := \mathrm{clamp}(\tau/\theta, 0, 1)$ converts a margin into a slowdown factor. Fig. 2 shows both quantities.

CVM runs four steps per cycle (Fig. 3). *Safety Fallback:* whenever the temporal margin of any incoming dependency becomes non-positive, the dependent agent is forced to wait at a full stop to prevent a collision.

Step 1 (Self-Recovery): if the malfunctioning agent i has accumulated delay while every upstream dependency still has slack, i accelerates up to $v_{\max}$ to recover a fraction $1/\gamma_1$ of its delay within the current segment:

$$v_i^{S1} = \min\left(\frac{d^i_{\mathrm{rem}}}{(t_e - t) - \delta_i/\gamma_1},\ v_{\max} \right). \tag{2}$$

Step 2 (Direct Cushioning, $h = 1$): each direct dependent j is slowed in proportion to the tightest incoming margin,

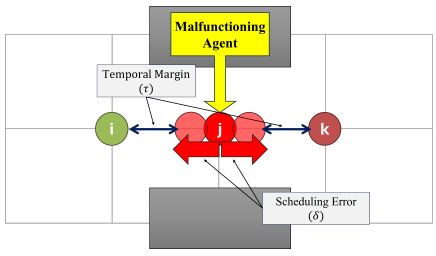

Fig. 2. Temporal margin τ and scheduling error δ induced by a malfunctioning agent.

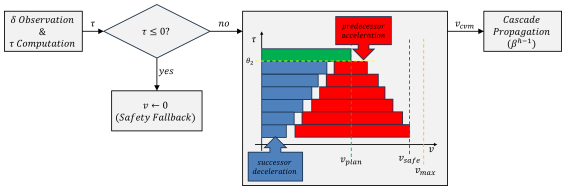

Fig. 3. CVM control pipeline: δ, τ estimation, self-recovery, direct cushioning, and cascade propagation.

so a small loss of margin becomes a small slowdown rather than a full stop:

$$v_j^{S2} = v_j^{S1} \cdot \min_{i \in \mathrm{pred}(j)} \rho(\tau(i \to j), \theta_2)\,. \tag{3}$$

Step 3 (Cascade Propagation, $h \geq 2$): an indirect dependent k at chain depth $h \geq 2$ receives the same proportional slowdown, but attenuated by β^{h-1}, which turns stop-and-go waves into a smooth ripple along the chain:

$$v_k^{S3} = v_k^{S1}\big(1 - \beta^{h-1}(1 - \rho(\tau(j \to k), \theta_3))\big). \tag{4}$$

The final command is $v_i^{\text{final}} = \min(v_i^{S2}, v_i^{S3})$, so the stronger slowdown always wins.

Together, the Safety Fallback and Steps 1–3 absorb a delayed agent into a smooth slowdown along the chain while keeping the team collision-free.

III. Experiments

A. Simulation

We generate ten randomized CCBS scenarios on a 5×5 grid with 5–8 agents, and ten paired delay seeds per scenario (100 runs total). Each seed injects one agent with a linear ramp slowdown to 10 % of the planned speed. We compare three execution policies on the same plan and the same anomaly: *Binary* (stop-and-go), *CVM-decel* (Self-Recovery and Direct Cushioning only, i.e. Step 1+2), and *CVM-full* (Step 1+2+3 with cascade propagation, $\beta = 0.95$). The self-recovery threshold is set to $\theta_1 = 0.05$, the direct-cushioning threshold to $\theta_2 = 0.5$, and the cascade threshold to $\theta_3 = 0.8$, with recovery aggressiveness $\gamma_1 = 2.0$ and safety buffer $\sigma = 0.2$.

Table I summarizes the results. CVM-decel reduces the mean makespan by 26.4 % and CVM-full by 24.6 %, and remains collision-free. Improvement grows with team density. CVM-decel slightly leads CVM-full here because the key benefit of CVM-full is that its preemptive slowdown gently absorbs the stop-and-go restart penalty, and the simulator has no such penalty to offset.

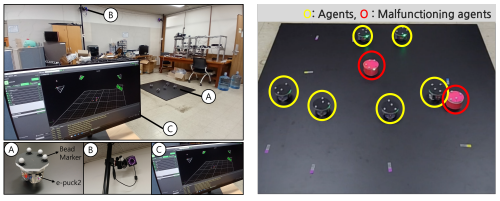

Fig. 4. Real-world environment.

TABLE I
Simulation makespan [s] over 10 scenarios, 10 seeds each.

Agents	Binary	CVM-decel	CVM-full
5	23.94 ± 1.02	17.79 ± 2.10 (+25.8 %)	17.79 ± 2.10 (+25.8 %)
5	22.11 ± 1.86	19.80 ± 0.48 (+10.4 %)	19.80 ± 0.48 (+10.4 %)
5	25.08 ± 1.17	19.38 ± 3.27 (+22.7 %)	20.34 ± 4.23 (+18.9 %)
6	34.59 ± 2.01	24.78 ± 2.46 (+28.4 %)	25.17 ± 3.21 (+27.2 %)
6	39.33 ± 1.74	30.18 ± 0.06 (+23.3 %)	30.18 ± 0.06 (+23.3 %)
6	33.66 ± 1.56	24.81 ± 1.95 (+26.3 %)	26.34 ± 3.09 (+21.8 %)
7	31.80 ± 1.65	21.75 ± 1.83 (+31.6 %)	21.75 ± 1.83 (+31.6 %)
7	43.77 ± 1.59	30.39 ± 4.53 (+30.6 %)	31.50 ± 6.06 (+28.1 %)
8	43.68 ± 1.59	28.83 ± 4.11 (+34.0 %)	29.73 ± 4.08 (+31.9 %)
8	47.55 ± 2.25	32.82 ± 5.85 (+31.0 %)	35.04 ± 8.94 (+26.3 %)
mean	**34.55**	**25.05** (+26.4 %)	**25.76** (+24.6 %)

TABLE II
Real-world makespan [s] over two 8-agent trials.

Agents	Binary	CVM-full
8	41.52	29.92 (+27.9 %)
8	46.21	27.10 (+41.3 %)

B. Real-world

Fig. 4 shows the real-world setup: eight e-puck2 robots on a 5×5 floor grid with two malfunctioning agents. Table II reports the makespan of both trials. The real-robot makespans track the simulation results, and CVM-full reduced the makespan by about 35 % over the Binary baseline, with no inter-robot collision.

IV. Conclusion

We presented CVM, a continuous execution controller that turns the temporal margin on each dependency edge into a proportional, exponentially damped slowdown along the chain. Across simulation and an experiment with eight e-puck2 robots, CVM consistently reduces the makespan compared to a binary baseline while keeping the team collision-free. A current limitation is that variable-speed coordination may induce cyclic deadlocks under complex dependency structures, which is left for future work.

Acknowledgment

This research was supported by a grant (D2403003) from Gyeonggi Technology Development Program funded by Gyeonggi Province.

References

[1] W. Hönig, S. Kiesel, A. Tinka, J. W. Durham, and N. Ayanian, "Persistent and robust execution of MAPF schedules in warehouses," IEEE Robot. Autom. Lett., vol. 4, no. 2, pp. 1125–1131, Apr. 2019.
[2] J. Li, Y. Chen, D. Harabor, P. J. Stuckey, and S. Koenig, "Bidirectional temporal plan graph for fast execution of MAPF plans," in Proc. AAAI, 2024.
[3] H. Jiang, X. Zhang, and J. Li, "Speedup techniques for switchable temporal plan graph optimization," in Proc. AAAI, 2025.
[4] J. Yan et al., "WinkTPG: A temporally accelerated plan executor for multi-agent path finding," arXiv preprint arXiv:2508.01495, 2025.
[5] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in Proc. IJCAI, 2019, pp. 39–45.