GPU Acceleration of Joint Multi-Agent Trajectory Optimization

Dipanwita Guhathakurta[1], Fatemeh Rastgar[2], M Aditya Sharma[1], Madhava Krishna[1] and Arun Kumar Singh[2]

[1] International Institute of Information Technology, Gachibowli, Hyderabad, India

[2] Institute of Science and Technology, University of Tartu, Tartu, Estonia

IROS KYOTO 2022
IEEE/RSJ International Conference on Intelligent Robots and Systems

ABSTRACT

Joint multi-agent trajectory optimization is conventionally considered intractable due to the exponential scaling of the number of collision avoidance constraints and linear increase in the number of variables by increasing the number of agents. On the other hand, the joint formulation allows access to more feasible space leading to better coordination maneuvers. Here, we try to improve the scalability of joint multi-agent trajectory optimization. Our core idea involves breaking the joint problem into several decoupled smaller Quadratic Programming (QP) problems and parallelizing them over GPUs. We compare the performance of our optimizer with the state of the arts in terms of trajectory quality including smoothness cost and arc length and computation time.

OBJECTIVE

- Breaking the joint multi-agent trajectory optimization into several smaller distributed decoupled problems.
- Reducing the decoupled sub-problems in the form of special QP problems.
- Showing that all the QPs associated with the decoupled sub-problems have the same matrices, and only the vector part of the QP are changing across the problem instances.
- Demonstrating that the solution process of such special QPs can be easily parallelized over GPUs.
- Comparison with the state-of-the-art [1,2] in terms of computation time and trajectory qualities

METHOD

Algorithm 1 Distributed Batch Optimizer Algorithm for the i^{th} Agent

1: *Initialize* ${}^{k}\boldsymbol{\xi}_{2,i}$, ${}^{k}\boldsymbol{\xi}_{3,i}$ *and* ${}^{k}\boldsymbol{\xi}_{4,i}$ *at iteration* $k=0$

2: **while** $k \leq$ max iteration or till norm of the residuals are below some threshold **do**

3: *Update* ${}^{k+1}\boldsymbol{\xi}_{1,i}$ *through*

$$ {}^{k+1}\boldsymbol{\xi}_{1,i} = \min_{\boldsymbol{\xi}_{1,i}} \Big(\frac{1}{2}\boldsymbol{\xi}_{1,i}^T\mathbf{Q}\boldsymbol{\xi}_{1,l} - \langle {}^{k}\boldsymbol{\lambda}_i, \boldsymbol{\xi}_{1,i}\rangle + \frac{\rho}{2}\Big\|\mathbf{F}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k}\boldsymbol{\xi}_{2,i}, {}^{k}\boldsymbol{\xi}_{3,i}, {}^{k}\boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) \; st. \mathbf{A}_{eq}\boldsymbol{\xi}_1 = \mathbf{b}_{eq} $$

4: *Update* ${}^{k+1}\boldsymbol{\xi}_{2,i}$ *through*

$$ {}^{k+1}\boldsymbol{\xi}_{2,i} = \min_{\boldsymbol{\xi}_{2,i}} \Big(\frac{\rho}{2}\Big\|\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i(\boldsymbol{\xi}_{2,i}, {}^{k}\boldsymbol{\xi}_{3,i}, {}^{k}\boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) $$

5: *Update* ${}^{k+1}\boldsymbol{\xi}_{3,i}$ *through*

$$ {}^{k+1}\boldsymbol{\xi}_{3,i} = \min_{\boldsymbol{\xi}_{3,i}} \Big(\frac{\rho}{2}\Big\|\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k+1}\boldsymbol{\xi}_{2,i}, \boldsymbol{\xi}_{3,i}, {}^{k}\boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) $$

6: *Update* ${}^{k+1}\boldsymbol{\xi}_{4,i}$ *through*

$$ {}^{k+1}\boldsymbol{\xi}_{4,i} = \min_{\boldsymbol{\xi}_{4,i}} \Big(\frac{\rho}{2}\Big\|\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k+1}\boldsymbol{\xi}_{2,i}, {}^{k+1}\boldsymbol{\xi}_{3,i}, \boldsymbol{\xi}_{4,i})\Big\|_2^2\Big) $$

7: *Update Lagrange multiplier coefficient through*

$$ {}^{k+1}\boldsymbol{\lambda}_i = {}^{k}\boldsymbol{\lambda}_i - \rho(\mathbf{F}^{k+1}\boldsymbol{\xi}_{1,i} - \mathbf{g}_i({}^{k+1}\boldsymbol{\xi}_{2,i}, {}^{k+1}\boldsymbol{\xi}_{3,i}, {}^{k+1}\boldsymbol{\xi}_{4,i}))\mathbf{F} $$

8: **end while**

9: *Return* ${}^{k+1}\boldsymbol{\xi}_{1,i}, {}^{k+1}\boldsymbol{\xi}_{2,i}, {}^{k+1}\boldsymbol{\xi}_{3,i}, {}^{k+1}\boldsymbol{\xi}_{4,i}$

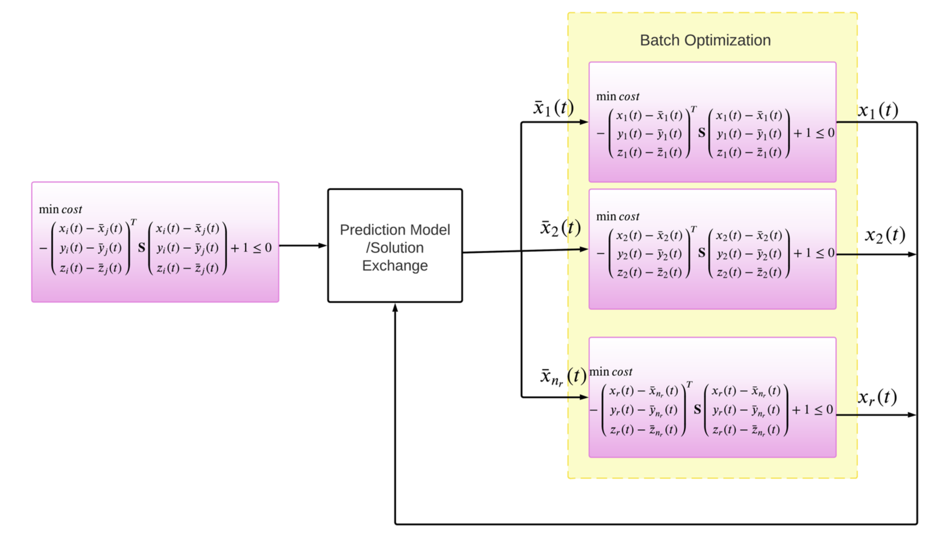

Fig.1: All agents communicate their current trajectories. In the next iteration, each agent uses this prior communicated trajectories to form the collision avoidance constraints at the next planning cycle. This in turn allows each agent to act independently. In other words, the communication strategy takes a joint trajectory optimization problem (first block on the left) and converts it into n_r decoupled problems. Our approach is GPU accelerated parallelized solution of the decoupled sub-problems

Overview

we present a special class of QPs and how their solution can be accelerated over GPUs. consider

$$ \min_{\boldsymbol{\xi}_i} \Big(\frac{1}{2}\boldsymbol{\xi}_i^T\overline{\mathbf{Q}}\boldsymbol{\xi}_i + \overline{\mathbf{q}}_i^T\boldsymbol{\xi}_i\Big), \quad \text{st: } \overline{\mathbf{A}}\boldsymbol{\xi}_i = \overline{\mathbf{b}}_i, \; i \in \{1,2,...,n_r\} \qquad (1) $$

where ξ_i is the optimization variable required to be solved for n_r different QP problems.

$$ \begin{bmatrix} \overline{\mathbf{Q}} & \overline{\mathbf{A}}^T \\ \overline{\mathbf{A}} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\xi}_i \\ \boldsymbol{\mu}_i \end{bmatrix} = \begin{bmatrix} \overline{\mathbf{q}}_i \\ \overline{\mathbf{b}}_i \end{bmatrix}, \; \forall i \in \{1,2,...,n_r\} \qquad (2) $$

$$ \begin{bmatrix} \boldsymbol{\xi}_1 & | & \cdots & | & \boldsymbol{\xi}_{n_r} \\ \boldsymbol{\mu}_1 & | & \cdots & | & \boldsymbol{\mu}_{n_r} \end{bmatrix} = \overbrace{\left(\begin{bmatrix} \overline{\mathbf{Q}} & \overline{\mathbf{A}}^T \\ \overline{\mathbf{A}} & \mathbf{0} \end{bmatrix}^{-1}\right)}^{matrix} \overbrace{\begin{bmatrix} \overline{\mathbf{q}}_1 & | & \overline{\mathbf{q}}_2 & | & \cdots & \overline{\mathbf{q}}_{n_r} \\ \overline{\mathbf{b}}_1 & | & \overline{\mathbf{b}}_2 & | & \cdots & \overline{\mathbf{b}}_{n_r} \end{bmatrix}}^{\text{stacked vectors}}, \qquad (3) $$

where μ_i are the dual optimization variables. Since the matrix in (2) does not depend on the batch index i, the optimization variables for all batches can be computed in parallel through (3).

Distributed Optimization Problem

Considering collision avoidance constraints in the polar form (see [2,3]), the i[th] decoupled sub-problem shown in Fig. 1 can be formulated in the following manner.

$$ \min_{x_i(t),y_i(t),z_i(t)} \sum_{t,i} \Big(\ddot{x}_i^2(t) + \ddot{y}_i^2(t) + \ddot{z}_i^2(t)\Big), \qquad (4a) $$

$$ \big[x_i(t),\dot{x}_i(t),\ddot{x}_i(t),y_i(t),\dot{y}_i(t),\ddot{y}_i(t),z_i(t),\dot{z}_i(t),\ddot{z}_i(t)\big]_{t=t_0} = \mathbf{b}_{o,i}, \qquad (4b) $$

$$ \big[x_i(t),\dot{x}_i(t),\ddot{x}_i(t),y_i(t),\dot{y}_i(t),\ddot{y}_i(t),z_i(t),\dot{z}_i(t),\ddot{z}_i(t)\big]_{t=t_f} = \mathbf{b}_{f,i}, \qquad (4c) $$

$$ -\frac{(x_i(t)-x_j(t))^2}{a^2} - \frac{(y_i(t)-y_j(t))^2}{a^2} - \frac{(z_i(t)-z_j(t))^2}{b^2} + 1 \leq 0, \qquad (4d) $$

$$ \forall t, \{i,j \in \{1,2,...,n_r\}, j \neq i\}, \qquad (4e) $$

where, $(x_i(t), y_i(t), z_i(t))$ represents the position of the i[th] agent at timestamp t. The cost function(4a) minimizes the acceleration along each axis at each time instant for all the agents. Initial and final boundary conditions are shown in (4b) and (4c). The unknown parameters, $\alpha_{ij}(t), \beta_{ij}(t)$, $d_{ij}(t)$ in collision avoidance constraints (4d) and (4e) need to be computed. Now, we parametrize x, y, and z, using time-dependent polynomial basis functions, and stack their coefficients as $\xi_{1,i}$. Then, we define $\xi_{2,i} = \boldsymbol{\alpha}_{ij}$ and $\xi_{3,i} = \boldsymbol{\beta}_{ij}$ and, $\xi_{4,i} = \mathbf{d}_{ij}$ and utilize augmented Lagrangian method to rewrite the optimization problem as:

$$ \min_{\xi_{1,i},\xi_{2,i},\xi_{3,i},\xi_{4,i}} \Big(\frac{1}{2}\xi_{1,i}^T\mathbf{Q}\xi_{1,i} - \langle\boldsymbol{\lambda}_i, \xi_{1,i}\rangle + \frac{\rho}{2}\|\mathbf{F}\xi_{1,i} - \mathbf{g}_i(\xi_{2,i}, \xi_{3,i}, \xi_{4,i})\|_2^2\Big) \qquad (5a) $$

$$ \mathbf{A}_{eq}\boldsymbol{\xi}_{1,i} = \mathbf{b}_{eq}, \qquad (5c) $$

$$ \boldsymbol{\xi}_{4,i} \geq \mathbf{1}, \qquad (5b) $$

Where λ_i is Lagrange multiplier, $\mathbf{A}_{eq}$ and $\mathbf{b}_{eq}$ are based on equality constraints and $\mathbf{F}_o$ is generated by vertically stacking polynomial functions.

$$ \mathbf{g}_{y,i} = \overline{\mathbf{y}}_j + a\mathbf{d}_{ij}\sin\boldsymbol{\beta}_{ij}\sin\boldsymbol{\alpha}_{ij}, \forall j, \quad \mathbf{g}_{z,i} = \overline{\mathbf{z}}_j + b\mathbf{d}_{ij}\cos\boldsymbol{\beta}_{ij}, \forall j \quad \mathbf{g}_{x,i} = \overline{\mathbf{x}}_j + a\mathbf{d}_{ij}\sin\boldsymbol{\beta}_{ij}\cos\boldsymbol{\alpha}_{ij}, \forall j, $$

$$ \mathbf{F} = \begin{bmatrix} \mathbf{F}_o & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{F}_o & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{F}_o \end{bmatrix} \quad \mathbf{g}_i = \begin{bmatrix} \mathbf{g}_{x,i}(\xi_{2,i}, \xi_{3,i}, \xi_{4,i}) \\ \mathbf{g}_{y,i}(\xi_{2,i}, \xi_{3,i}, \xi_{4,i}) \\ \mathbf{g}_{z,i}(\xi_{2,i}, \xi_{3,i}, \xi_{4,i}) \end{bmatrix} \qquad (6) $$

Using Alternating Minimization (AM) method, our optimization problem (5a)-(5d) can be solved through Algorithm 1.

BENCHMARKS

Implementation Details:

- A desktop computer with 32 GB RAM and RTX 2080 NVIDIA GPU.
- Using JAX [4] in python to accelerate linear computations

SCAN ME

Benchmarks:

- The agents' start and goal positions are sampled along the circumference of a circle.
- The agents are initially located on a grid and are tasked to converge to a line formation.

Qualitative Results

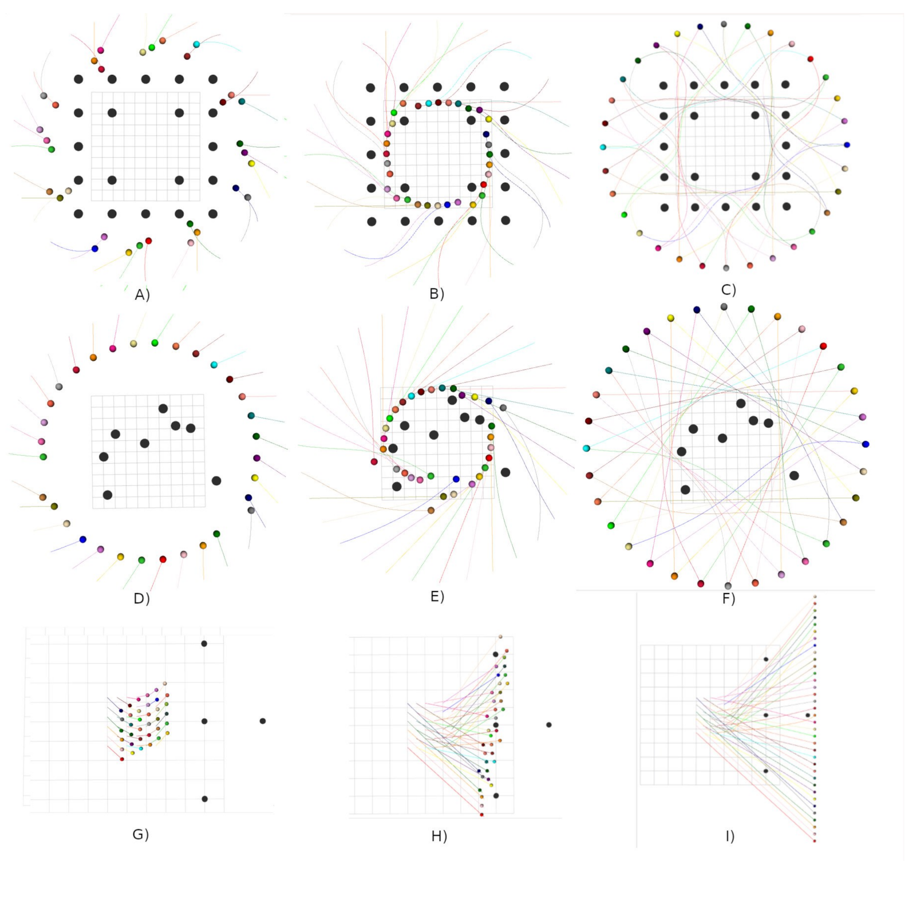

Fig.2: Trajectory snapshots for (A-C) 32 agents, with radius 0.3m and 20 obstacles of radius 0.4m, (D-F) 32 agents, with radius 0.3m and 8 randomly placed obstacles of radius 0.4m, (G-I) 36 agents with radius 0.1m arranged in a grid configuration are required to move to a line formation. Also, there are 4 static obstacles with radius 0.15m.

Optimizer Convergence Validation

We conceptually validate the convergence of our optimizer by plotting the constraints residual over iterations Fig. (3). If these residuals have a decreasing trend over iterations and converge to zero, trajectories are collision-free. Since this trend is satisfied in Fig. (3), the trajectories returned by our optimizer ensure the agents do not collide with each other and obstacles.

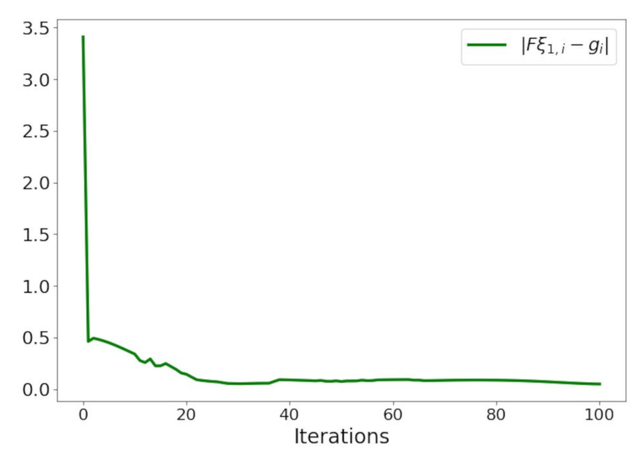

Fig. 3: Validating optimizer convergence empirically

Comparisons

		16 agents					32 agents		
		2 Obs	4 Obs	8 Obs	12 Obs	24 Obs	12 Obs	16 Obs	20 Obs
Comp. time	[2]	0.34	0.37	0.70	0.79	1.49	1.68	1.752	1.80
	ours	**0.15**	**0.16**	**0.16**	**0.17**	**0.17**	**0.19**	**0.20**	**0.20**
	[1]	0.62	0.70	0.68	0.66	0.79	12.50	12.42	11.82
Arc-length	[2]	13.48	14.25	15.53	15.93	24.19	23.85	24.04	24.14
	ours	**9.99**	11.69	11.11	11.19	10.24	**22.59**	**22.03**	**23.15**
	[1]	10.30	**10.67**	**10.74**	**10.60**	15.79	26.21	26.15	26.29
Smoothness	[2]	0.10	0.11	0.14	0.15	0.16	0.15	0.16	**0.17**
	ours	**0.048**	**0.093**	**0.08**	**0.106**	**0.06**	**0.13**	**0.12**	0.21
	[1]	0.11	0.11	0.11	0.11	0.3	0.36	0.36	0.36

Table 1: Comparison of our optimizer with [1,2] in terms of computation time, arc-length and smoothness}

CONCLUSION

By leveraging mathematical reformulations and GPU-based parallelization, our optimizer computes trajectories for tens of agents in cluttered environments within a fraction of a second. In comparison with state-of-the-art baseline approaches, we achieve improvement in terms of not only the computation time, but also trajectory quality.

REFERENCES

[1] J. Park, J. Kim, I. Jang, and H. J. Kim, "Efficient multi-agent trajectory planning with feasibility guarantee using relative Bernstein polynomial," in 2020 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2020, pp. 434–440

[2] F. Rastgar, H. Masnavi, K. Kruusamae, A. Aabloo, and A. K. Singh, "Gpu accelerated convex approximations for fast multi- agent trajectory optimization," IEEE Robotics and Automation Letters, vol. 6, no. 2, pp. 3303–3310, 2021

[3] F. Rastgar, A. K. Singh, H. Masnavi, K. Kruusamae, and A. Aabloo, "A novel trajectory optimization for affine systems: Beyond convex- concave procedure," in 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020, pp. 1308–1315

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, et al., "Jax: composable transformations of python+ num programs," Version 0.2, vol. 5, pp. 14–24, 2018.