2022 Sixth IEEE International Conference on Robotic Computing (IRC) | 978-1-6654-7260-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IRC55401.2022.00075

Real-time Multi-Objective Trajectory Optimization

Ilya Gukov, Alvis Logins
Huawei Technologies Company Ltd.
{ilya.gukov, alvis.logins}@huawei.com

***Abstract*—In this article, a method is presented for generating a trajectory through predetermined waypoints, with jerk and duration as two conflicting objectives. The method uses a Seq2Seq neural network model to approximate Pareto efficient solutions. It trains on a set of random trajectories optimized by Sequential Quadratic Programming (SQP) with a novel initialization strategy. We consider an example pick-and-place task for a robot manipulator. Based on several metrics, we show that our model generalizes over diverse paths, outperforms a genetic algorithm, SQP with naïve initialization, and scaled time-optimal methods. At the same time, our model features a negligible GPU-accelerated inference time of 5ms that demonstrates applicability of the approach for real-time control.**

***Index Terms*—Trajectory Planning, Multiobjective Optimization, Deep Learning**

I. Introduction

A control system of an industrial robot-manipulator commonly implies a reference input trajectory, represented as a set of joint values per each motor control cycle, and a stabilizing controller per each actuator. *Trajectory planning* (TP) becomes a multi-step procedure, where collision-free path planning is followed by velocity optimization and several subsequent interpolations [1]. The complexity of the pipeline is justified by the slowness of planning algorithms, high frequency of positioning controllers, and limitations of a communication protocol between motors and CPU/GPU.

Recently, Kleff et al. [2] substituted stabilizing controllers with a closed-loop Model Predictive Control scheme running on a single CPU connected to a manipulator by the Fast Robot Interface of the KUKA Sunrise Workbench at 1 kHz. Such frequency was sufficient to notably improve responsiveness of the manipulator. With recent advances in optical network technologies, the Industrial Internet of Things (IIoT) [3] can support connection to a *centralized controller* at 200kHz, which expands capabilities of such simplified control scheme much further, though limited by the insufficient research of the appropriate algorithmic approaches.

We suggest to take advantage of GPU for nonlinear online trajectory optimization in the context of IIoT with centralized control. Given a set of waypoints and pre-computed example trajectories, we are able to train a neural network (NN) that can approximate *the time-jerk optimal trajectory* for all actuators in 5ms with high accuracy. This allows one to rapidly estimate the time-jerk trade-off and adjust aggressiveness of robot movements in real-time.

A significant amount of literature considers jerk as an optimization constraint in Time-Optimal TP (TOTP) [1], [4], [5], or as an objective in Multi-Objective TP (MOTP) [6], [7]. Such *smoothing technique* reduces wear of mechanical parts, and decreases vibrations in the system caused by inertia and the stiffness of the parts [8]. Our solution follows the MOTP line of work, where a path is interpolated by a time-dependent spline, with the parameters optimized in terms of the trajectory duration and jerk simultaneously.

There are two existing methods for MOTP. The first method applies a multi-objective genetic algorithm that iteratively improves the approximation set as a batch [6]. The second method combines the objectives and optimizes their weighted sum [7]. Variation of the weights yields the approximation set. Both methods require at least several seconds to get an estimate, which makes it unpractical in our scenario.

We follow the second method and use Sequential Quadratic Programming (SQP) [7] to optimize random paths over the workspace of a robot-manipulator. The resulting trajectories are used to train a model and evaluate on paths generated by a path planner applied in simulated scenes with obstacles. The metrics used are the ϵ-indicator [9], minimal trajectory duration over the set, and average jerk decrease for several fixed durations. We show that NN has an advantage over SQP with naïve initialization, a genetic algorithm, and two TOPP algorithms adapted to our problem by time scaling. At the same time, the average inference time is low enough to support online TP on a centralized controller.

II. Problem Statement

Let us consider a path consisting of N waypoints, where the i-th waypoint $\mathbf{q}_i$ is a vector of M joint values. We aim to build a three-times differentiable function $\mathbf{q}(t)$ that interpolates the waypoints. The function is *a trajectory* that depends on time parameter $t \geq 0$. We find $\mathbf{q}(t)$ in the form of 5-th order B-splines [6]. Let T be the timestamp of the last point in the path: $\mathbf{q}(T) = \mathbf{q}_N$. Then, $\mathbf{q}(t)$ should satisfy a set kinematic constraints (Eq. 1), and zero boundary conditions on velocity $\dot{\mathbf{q}}(t)$ and acceleration $\ddot{\mathbf{q}}(t)$ (Eq. 2).

$$\begin{cases} |\mathbf{q}(t)| < \mathbf{q}_{\max} \\ |\dot{\mathbf{q}}(t)| < \dot{\mathbf{q}}_{\max} \\ |\ddot{\mathbf{q}}(t)| < \ddot{\mathbf{q}}_{\max} \end{cases}, \forall t \in [0, T] \tag{1}$$

$$\dot{\mathbf{q}}(0) = \dot{\mathbf{q}}(T) = \ddot{\mathbf{q}}(0) = \ddot{\mathbf{q}}(T) = 0 \tag{2}$$

We formulate the *Multi-Objective Trajectory Planning* (MOTP) problem as $\min_{\mathbf{h}}(J, T)$, subject to Eqs. 1, 2, where J is the *jerk* defined as

$$J = \sum_{j=1}^{M} \sqrt{\int_0^T \dddot{q}_j(t)^2 dt / T} \tag{3}$$

A solution to MOTP is *an approximation set* $H = \{\mathbf{h}_k\}$, where for any $\mathbf{h}_{k_1} \in H$ there is no another $\mathbf{h}_{k_2} \in H$ which results in smaller J and T simultaneously. In other words, $\mathbf{h}_{k_1}$ does not *dominate* any other solution in H.

As a comparison metric of approximation sets, we use *the binary ϵ-indicator* I_ϵ [9]. Let J_k and T_k be the objectives that correspond to $\mathbf{h}_k$. $\mathbf{h}_{k_1}$ is said to ϵ-dominate $\mathbf{h}_{k_2}$ ($\mathbf{h}_{k_1} \succ \mathbf{h}_{k_2}$) if and only if $J_{k_1} \leq \epsilon \cdot J_{k_2}$ and $T_{k_1} \leq \epsilon \cdot T_{k_2}$ for a given $\epsilon > 0$. The binary ϵ-indicator is defined as

$$I_\epsilon(H_1, H_2) = \inf_\epsilon \{\forall \mathbf{h}_{k_1} \in H_1 \;\; \exists \mathbf{h}_{k_2} : \mathbf{h}_{k_1} \succ \mathbf{h}_{k_1}\} \quad (4)$$

I_ϵ shows how small ϵ can be so that, after the multiplication of the objectives in H_2 by ϵ, all the solutions in H_1 dominate H_2. We say H_1 dominates H_2 if and only if $I_\epsilon < 1$. The approximation set H_1 is *the Pareto optimum* if there is no any H_2 that dominates H_1.

III. Methods

A. A Neural Network based approach

Consider a sequence $D = [\mathbf{d}_i]_i$ of relative shifts between the waypoints, $\mathbf{d}_i = \mathbf{q}_{i+1} - \mathbf{q}_i$. We derive an approximation set H by estimating $\tilde{\mathbf{h}}(D, \lambda)$ given by Eq. 5 for different values of parameter $\lambda \in [0, 1]$:

$$\tilde{\mathbf{h}}(D, \lambda) = \arg\min_{\mathbf{h}} \lambda J(D, \mathbf{h}) + (1 - \lambda) T(D, \mathbf{h}) \quad (5)$$

We use D instead of $[\mathbf{q}_i]_i$ as a parameter because it shrinks the input domain of $\tilde{\mathbf{h}}$, and D is sufficient to derive $\mathbf{q}(t)$ because J and T are invariant to translation of the curve.

We use SQP to solve the problem in Eq. 5 w.r.t. the kinematic constraints (Eqs. 1, 2). SQP has two major disadvantages. First, it requires an initial estimate $\tilde{\mathbf{h}}_0$, and the result is sensitive to how good the estimate is. Second, obtaining a single solution takes up to 2s, which is too slow to try different $\tilde{\mathbf{h}}_0$ or λ values. Therefore, we propose to *precompute* a set of solutions using SQP, and use this dataset to train NN that *predicts* $\tilde{\mathbf{h}}$ for any D and λ within some input domain.

The input paths may have different lengths. Therefore, we consider a class of *sequence-to-sequence* (Seq2Seq) NN models, where we use λ as a control parameter. Such models are commonly used in Natural Language Processing [10]. In particular, we implement an encoder/decoder architecture illustrated on Fig. 1. The encoder and decoder consist of two-layer bi-directional long short-term memory (LSTM) networks [11]. The encoder produces a hidden state X, $|X| = 32$. We concatenate X with λ and pass it trough a Fully Connected (FC) layer with ReLU activation and a LayerNorm (LN) layer [12]. The resulting vector goes through LSTM and another FC layer to get the final prediction $\tilde{\mathbf{h}}$.

For each (D, λ) we try ten $\tilde{\mathbf{h}}_0$ values and select the one that corresponds to the minimal objective. Let

$$r^i_{\min} = \max_j \frac{q^{i+1}_j - q^i_j}{(\dot{\mathbf{q}}_{\max})_j}; \; r^i_{\max} = C \cdot r^i_{\min}, \quad (6)$$

where C is a parameter, j joint index, and i waypoint index. We define i-th vector component of $\tilde{\mathbf{h}}_0$ as $(\tilde{\mathbf{h}}_0)_i = r^i_{max}$ for $C \in [1..10]$.

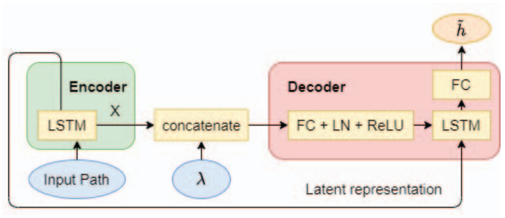


Fig. 1: Network architecture

NN may produce a trajectory that violates the constraints. Similarly to [1], time scaling is applied to the output in this case. Torque bounds and other constraints can also be integrated into this post-processing step. To reduce the risk of constraint violation, we use a modified L1 loss function that encourages the model to predict larger values:

$$\mathcal{L} = \frac{1}{N-1} \sum_{j=1}^{N-1} (|\tilde{h}_j - h_j| + \alpha \max(h_j - \tilde{h}_j, 0)) \quad (7)$$

where $\alpha = 0.01$.

Once the network is trained for the first time, we use it to make a prediction on all the *training* dataset. Then, we use these predictions as a new initial estimate for SQP. This improves SQP solutions and increases the number of paths for which SQP converges. Thus, we recompute the dataset, train NN anew, and repeat the procedure 10 times to get the final model.

B. Baselines

We compare with the multi-objective genetic optimization algorithm *NSGA-II* [6]. It starts with some feasible approximation set H_0 of size ℓ, then iteratively improves the set as follows. Solutions in H_i are first sorted by non-domination, i.e. a solution $\mathbf{h}$ is evaluated by the number of other solutions that dominate $\mathbf{h}$. The top k solutions are selected, $k < \ell$. Ties are resolved by choosing solutions that are further from neighbors w.r.t. Euclidean norm in the (J, T) plane. A crossover and mutation operations are applied to the selected solutions, we use similar operations as in [6].

A crossover generates a new solution $\mathbf{h}'$ based on a sampled pair $\mathbf{h}^A$ and $\mathbf{h}^B$ according to some elitist strategy. In our implementation, it generates a random set of vector component indices $\mathbb{I}$, and sets $h'_i = h^B_i$ for $i \in \mathbb{I}$ or $h'_i = h^A_i$ otherwise. A mutation copies and randomly changes a single solution with probability p_m. We change the solution by generating a random set of indexes $\mathbb{J}$ and setting $\tilde{h}_i$ to a random value $r_i \in [r_{\min}, r_{\max}]$ for $i \in \mathbb{J}$, $C = 75$.

Next, we check if the resulting spline satisfies the velocity and acceleration constraints. If the constraints are satisfied, we add it to a set H'. We repeat the operations until $|H'| = 2k$. Finally, ℓ solutions are selected from H' for the next generation set H_{i+1}, according to the same sorting strategy. In our experiments, we have found that the best result is achieved when $|\mathbb{I}| = 1$ and $|\mathbb{J}| = |\mathbf{h}|$.

We also compare with two TOPP algorithms provided by the MoveIt framework [1], the Iterative Parabolic Time

Parametrization (IPTP) and Time-Optimal Trajectory Generation (TOTG) [13] algorithms. IPTP begins with time intervals such that the minimum velocity is equal to the most constraining value over the joints. Then, it iteratively increases the time intervals until all the acceleration constraints are satisfied. TOTG applies circular blends to non-smooth paths, finds the maximal velocity profile in the phase plane under the assumption of zero acceleration at nondifferentialble points, and uses numerical integration to derive time parametrization. Notably, these algorithm do not estimate H and do not consider jerk and zero acceleration boundary condition. Nevertheless, we interpolate the solutions with 5th order B-splines with zero boundary condition up to the 3rd order, and derive H by scaling the trajectories in the time domain.

C. Dataset

For experiments, we used a 6DOF robot manipulator model (Figure 2), and 3 types of paths. The first type are random paths of lengths $N \in [3, 20]$. Waypoints were sampled uniformly in Cartesian space within the workspace of the robot, and inverse kinematics was applied to get the corresponding joint values. The second and third types are paths generated for two different pick-and-place scenes with obstacles (Fig. 2) using the standard Move!It path planner RRTConnect [1]. It generates two random trees in the joint space until a collision-free path between source and destination is found. The resulting path is contracted by removing the waypoints that keep the path collision-free after removal.

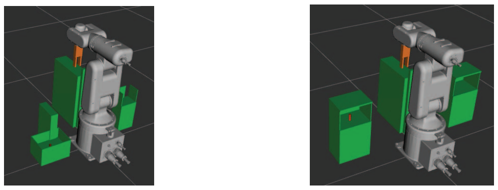


Fig. 2: The first (left) and second (right) scenes for planning.

The integral in Eq. 3 was estimated numerically on a regular time grid $\Delta t = 2$ms. Our experiments have been conducted on Intel(R) Xeon(R) Gold 6154 CPU @ 3.00GHz. Neural networks have been run on Tesla V100-SXM2 GPU. For training, we use the Adam optimizer, and a total of $2 \cdot 10^5$ random paths. We use a single trained model to interpolate all types of paths.

IV. Results

Figure 3 illustrates the approximation sets for two random paths of length 6 and 11, and two paths obtained from the path planners of length 6 and 8. NN demonstrates better H than IPTP and TOTG. SQP with naïve initialization and NSGA-II demonstrate similar H, but they require significantly more computational time. NN takes 5 ± 1.5ms per point, while SQP needs 2.6$\pm$0.9s per point. NSGA-II does not allow to calculate H pointwise. For $\ell = 100$, NSGA-II takes 77 ± 17s. In this experiment, we have calculated 10 SQP and 10^3 NN points by uniformly varying $\lambda \in [0, 1]$. In many cases SQP does not converge, so there are only few SQP points in Figures 3a,3b, and 3c. NSGA-II always produces ℓ points, but they are often highly clustered, as for Figures 3b and 3c. Our algorithm brings a wider coverage of H. However, some solutions might violate constraints and become non-dominant after scaling, e.g. solutions with $J > 11$ on Figure 3a.

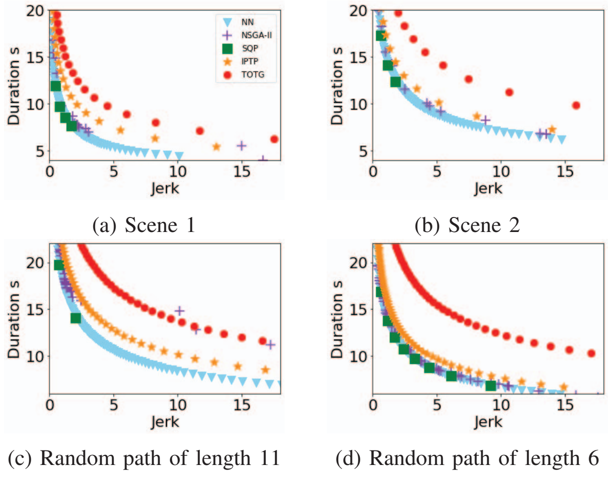


Fig. 3: Model performance on random and pre-planned paths

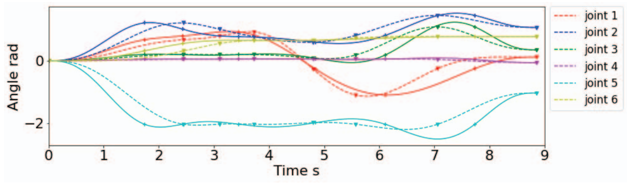


Fig. 4: $\mathbf{q}(t)$ produced by IPTP (solid) and NN (dashed) on the same path and having same T. Markers represent waypoints. $N = 8, \eta = 0.36$.

A single NN-optimized trajectory in comparison to IPTP is shown on Figure 4. Lower amplitudes and increased smoothness of dashed lines are noticeable for all joint except joint 1, that lead to the overall jerk reduction in 1.57 times.

Next, we consider approximation sets for 700 random paths, and 200 paths from Scene 2 where the source position is randomized. We compared H in terms of the ϵ-index $I_\epsilon(H_1, H_2)$ (Eq. 4). Table I shows the mean ϵ-index and the fraction of paths where $I_\epsilon < 1$, i.e. the cases where H_1 dominates H_2. We see that NN dominates NSGA-II in 10% of cases for Scene 2, 5% for random paths, does not dominate SQP, dominates IPTP in 64% and 2% of cases correspondingly, and in all cases for TOTG. We drop TOTG from further consideration due to poor performance. Notably, the baselines never dominate our solution. The mean values $\bar{I}_\epsilon$ are close to 1, while the mean value of the ϵ-index with the reversed arguments $\bar{I}'_\epsilon$ is significantly higher. This suggests that our model is close to domination and gives better solutions on average. In order to support this statement further, we present additional statistics in the following.

TABLE I: $I_\epsilon = I_\epsilon(H_{\text{NN}}, H)$, $I'_\epsilon = I_\epsilon(H, H_{\text{NN}})$, where H_{NN} is a set by NN and H is a set by the method defined in the second column. The percentages is the fraction of paths where ϵ-index is smaller than 1.

Paths	Methods	$\bar{I}_\epsilon$	$I_\epsilon < 1$	$\bar{I}'_\epsilon$	$I'_\epsilon < 1$
SCENE 2	NSGA-II	1.05	10%	2	0%
	SQP	1.06	0%	8.4	
	IPTP	1.005	64%	1.26	
	TOTG	0.98	100%	1.95	
RANDOM	NSGA-II	1.07	5%	1.31	0%
	SQP	1.01	0%	6.94	
	IPTP	1.09	2%	1.1	
	TOTG	0.97	100%	1.87	

First, we consider extreme values in approximation sets when $J \to \infty$ and the problem boils down to TOPP. Let us define τ as a normalized increase in the minimal T over H, relative to our solution, $\tau = (T^{\min} - T^{\min}_{\text{NN}})/T^{\min}_{\text{NN}}$. Averaged τ is presented in Table II. All values are positive, implying that the NN-based time-optimal trajectories are faster on average.

TABLE II: τ value, averaged over Scene 2 and random paths.

Methods	τ, Scene 2	τ, Random
NSGA-II	1.18 ± 0.44	0.38 ± 0.32
SQP	0.15 ± 0.27	0.18 ± 0.26
IPTP	0.01 ± 0.14	0.07 ± 0.17

Second, we collect statistics on how much our solution reduces jerk for a fixed trajectory duration. Let us denote jerk reduction as $\eta(T) = \frac{J(T) - J_{\text{NN}}(T)}{J(T)}$, where $J(T)$ and $J_{\text{NN}}(T)$ are jerk values of a path of duration T. SQP and NSGA-II does not allow custom, so we compare only with IPTP, where such solution is obtained by time scaling. We run NN for different λ and select the duration of resulting trajectory as the target T for comparison. Figure 5 shows η for $\lambda \in \{0.25, 0.5, 0.75, 1\}$. The distributions are similar for all λ, with $\eta > 0$ in most cases, implying that NN decreases jerk.

Figure 6 represents the distribution of η over path lengths for different λ. The jerk reduction η is the highest for path length of around 8-13 waypoints, and particularly low values for very short paths (4 waypoints). This, however, strongly depends on the training dataset. With a more complex and effective SQP initialization policy or by using another optimization method, our NN-based approach may support longer paths.

V. Conclusions

We have proposed an NN-based supervised method for constructing an approximation set for the time-jerk-optimal trajectory planning problem, where trajectories are B-spline curves with zero boundary conditions up to the third order. We have generated a dataset using SQP with iterative refinement of the initial solution, and trained a Seq2Seq model to predict the parameters of B-splines in an efficient way. The generalization ability of the model has been illustrated experimentally by comparison with NSGA-II, SQP with naïve initialization, and two scaled TOPP solutions, applied to random paths, and paths generated by a Move!It path planner in a pick-and-place task using a 6DOF manipulator model.

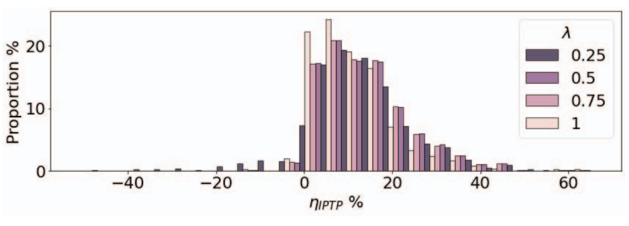
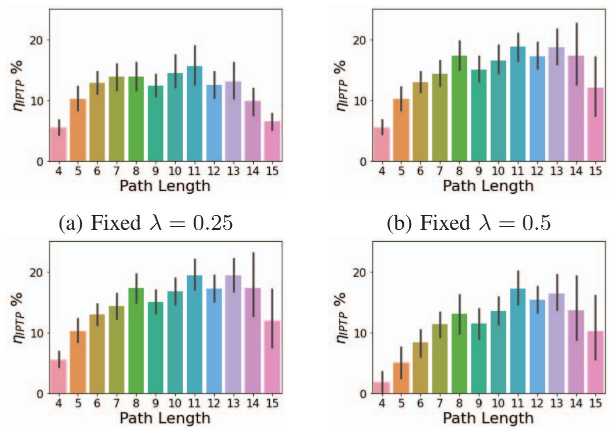


Fig. 5: η comparing NN to IPTP (fixed T). 10^3 random paths.



(a) Fixed $\lambda = 0.25$ (b) Fixed $\lambda = 0.5$

(c) Fixed $\lambda = 0.75$ (d) Fixed $\lambda = 1$

Fig. 6: η dependence on N. 10^3 random paths.

References

[1] I. A. Sucan and S. Chitta, "Moveit." [Online]. Available: https://moveit.ros.org/

[2] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-Frequency Nonlinear Model Predictive Control of a Manipulator," in *IEEE ICRA*, 2021, pp. 7330–7336.

[3] K. Christodoulopoulos, S. Bidkar, T. Pfeiffer, and R. Bonk, "Proof-of-concept demonstration of time critical periodic traffic in industry-grade passive optical networks," in *Optical Fiber Communications Conference and Exhibition*, 2022, pp. 1–3.

[4] H. Hayati, D. Eager, A.-M. Pendrill, and H. Alberg, "Jerk within the context of science and engineering—a systematic review," *Vibration*, vol. 3, no. 4, pp. 371–409, 2020.

[5] D. Kaserer, H. Gattringer, and A. Müller, "Nearly optimal path following with jerk and torque rate limits using dynamic programming," *IEEE Transactions on Robotics*, vol. 35, no. 2, pp. 521–528, 2019.

[6] J. Huang, P. Hu, K. Wu, and M. Zeng, "Optimal time-jerk trajectory planning for industrial robots," *Mechanism and Machine Theory*, vol. 121, pp. 530–544, 2018.

[7] Y. Zhang, Z. Sun, Q. Sun, Y. Wang, X. Li, and J. Yang, "Time-jerk optimal trajectory planning of hydraulic robotic excavator," *Advances in Mechanical Engineering*, vol. 13, no. 7, 2021.

[8] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Trajectory planning in robotics," *Mathematics in Computer Science*, vol. 6, no. 3, pp. 269–279, 2012.

[9] E. Zitzler, L. Thiele, M. Laumanns, C. Fonseca, and V. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[10] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *CoRR*, vol. abs/1409.3215, 2014.

[11] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[12] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," 2016.

[13] T. Kunz and M. Stilman, "Time-optimal trajectory generation for path following with bounded acceleration and velocity," in *Robotics: Science and Systems VIII*, 2012.