Accelerating Trajectory Generation for Quadrotors Using Transformers

Srinath Tankasala*   STANKASA@UTEXAS.EDU

Mitch Pryor*   MPRYOR@UTEXAS.EDU

*Cockrell School of Engineering, The University of Texas at Austin, USA

Abstract

In this work, we address the problem of computation time for trajectory generation in quadrotors. Most trajectory generation methods for waypoint navigation of quadrotors, for example minimum snap/jerk and minimum-time, are structured as bi-level optimizations. The first level involves allocating time across all input waypoints and the second step is to minimize the snap/jerk of the trajectory under that time allocation. Such an optimization can be computationally expensive to solve. In our approach we treat trajectory generation as a supervised learning problem between a sequential set of inputs and outputs. We adapt a transformer model to learn the optimal time allocations for a given set of input waypoints, thus making it into a single step optimization. We demonstrate the performance of the transformer model by training it to predict the time allocations for a minimum snap trajectory generator. The trained transformer model is able to predict accurate time allocations with fewer data samples and smaller model size, compared to a feedforward network (FFN), demonstrating that it is able to model the sequential nature of the waypoint navigation problem.

Keywords: Trajectory optimization, Minimum snap, Transformers

1. Introduction

A quadrotor is an extremely agile aerial system that has found widespread use in commercial activities such as inspections, package delivery, cinematography, etc. Multi-waypoint trajectory generation is essential for autonomous systems navigation and is typically done using a local planner. Local planners for quadrotor navigation typically minimize smooth polynomial trajectories based on aggregate snap/jerk. The most widely used formulation in minimum snap/jerk planners is a bi-level optimization where the time allocations across waypoints is computed in the first level and a constrained optimization of polynomial coefficients over different waypoints forms the second level [9, 11]. This strategy of optimizing piecewise time allocations and then optimizing piecewise polynomials between waypoints is also used in minimum-time trajectory generation for drones [20, 5].

For minimum snap trajectory generation, recent works [3, 18] have cleverly reformulated both steps of the bi-level optimizations that are solvable in linear time. In this work, we propose to instead learn the optimal waypoint time allocations using deep learning models. By eliminating the bi-level structure of the optimization problem, we can accelerate the trajectory generation process. This would require the learning-based model to be capable of giving highly accurate predictions of

optimal time allocations for any given sequence of waypoints. In our approach we use sequence models, specifically transformers[16], to learn the optimal time allocations for minimum snap trajectory generation. While the approach here is used to learn parameters in the two level minimum snap optimization, this approach can be extended to learn parameters for other trajectory generation problems such as minimum-time trajectories.

This work has three main contributions: (1) we use transformers to learn optimization parameters that removes the bi-level structure and turns it into a single-level optimization problem making it faster (2) we evaluate the trained transformer model performance, sample efficiency and also show its effectiveness across different settings, including input sizes greater than those of samples in the training data (3) we study the learnt attention patterns to understand the input output relation learnt by the model and the insight it provides about the nature of the time allocation optimization. Overall, this seq2seq modeling approach can be used to learn parameters from offline trajectory data, and can provide good approximations to the time allocations thus speeding up trajectory optimization.

2. Related Work

The differential flatness property of quadrotor dynamics is well known in the literature ([9]) and it allows the user to plan trajectories in the flattened output space of the quadrotor, i.e. position, velocity, acceleration and yaw. Notable approaches for trajectory generation in quadrotors use sequential convex optimization [6] and sampling based kinodynamic planning [7] among others. The most widely used trajectory generation methods for waypoint navigation, however, use polynomial functions as basis/primitives for generating their trajectories[9, 11].

The bi-level optimization structure is also used in minimum-time trajectory generation in the non-linear quadrotor state space [20, 14, 15]. Heuristic functions have been used [5] to speed up time allocation computation, but lead to suboptimal solutions.

Thus in many applications, quadrotor trajectory planning is typically formulated as a bi-level optimization problem with the first level involving time allocations across desired waypoints. For minimum jerk/snap optimizations, the time allocation problem can be solved using a backtracking gradient descent (BGD) until convergence to a local minimum [9, 8]. These methods typically use a starting guess for the time allocation such as a trapezoidal velocity profile (TVP) [8]. This time allocation optimization step has been computationally expensive, until recently where Burke et al [3] and Wang et al [18] have solved it in linear time.

An alternative approach that has gained traction for speeding up optimization computation is the use of data driven methods [12, 2]. While supervised learning models like MLPs (multi-layer perceptron) can be trained to predict the waypoint time allocations [2], they cannot handle variable number of input waypoints and hence are not usable beyond a certain input size. In Ryou et al[12], the authors train an attention based Seq2seq model to predict time allocation. However, it only uses cross attention and doesn't exploit the self-attention mechanism used in Transformers.

Traditional RNNs have be used for trajectory generation [19]. We use transformers [16] as they consider all inputs simultaneously when generating the time allocations and have provided state of the art performance in seq2seq tasks[16]. Their attention function implementations have also been optimized [17, 4] to make them computationally fast. Hence they are suitable for learning the time allocations over the given input waypoints.

3. Approach

In minimum snap planning, we optimize the aggregate snap over the entire trajectory. We denote the 3D trajectory of the UAV as $r(t) = [x(t), y(t), z(t)]$. The minimum-snap trajectory optimization is solved separately along each Cartesian axis (x,y,z), described in [9], and it is formulated as:

$$\min_{r,\underline{t}} J(r, \underline{t})$$

$$\text{s.t.} \quad r(0) = w_0; \quad v(0) = v(T) = 0 \tag{1}$$

$$r_k(\sum_{j=1}^{i} t_j) = w_i; \quad \Sigma t_i = T; \ \forall i = 1, 2, ..., m$$

$$\text{where,} \quad J(r, \underline{t}) = \int_0^T \left\| \frac{d^4 \mathbf{r}}{dt^4} \right\|_2^2 dt; \quad r \in R_T \tag{2}$$

$\underline{t} = [t_1, t_2, ..., t_m]$, is the allocations of total time T over the desired waypoints $W = [w_0, w_1, w_2, w_3, ..., w_m]$, $w_i \in \mathbb{R}^3$. R_T is the set of all feasible trajectories passing through W that can be tracked in time T. In [9], they also include second derivative of yaw in the cost $J(r, \underline{t})$, but in many applications the yaw is planned separately or set to a constant. Hence we do not include yaw as an optimization parameter.

To solve for the minimum snap trajectory optimization, we first reduce the aggregate trajectory snap along each axis for a given $\underline{t}$. The solution is of the form of m polynomials of order n (7 in this case) as shown in eq (3).

$$r_k(t) = \begin{cases} \sum_{j=0}^{n} a_{j,1} t^j, & t_0 \le t \le t_1 \\ \vdots \\ \sum_{j=0}^{n} a_{j,m} t^j, & t_{m-1} \le t \le t_m \end{cases} \tag{3}$$

Given a time allocation $\underline{t}$, the minimum snap trajectory under $\underline{t}$ is a quadratic program given by Optimization 1, refer [9]:

Optimization 1

$$\min_a \quad a^T Q a$$

$$\text{s.t.} \quad A_{eq} a = b_{eq} \tag{4}$$

$$A_{ineq} a \le b_{ineq}$$

---

where $Q \in R^{nm \times nm}$ is a block diagonal matrix containing powers of t, A_{eq} is constructed from Q and the equality constraint enforces the condition of passing through waypoints and continuity of velocities and accelerations. The A_{ineq} and b_{ineq} are typically used to enforce corridor constraints. To solve for $\underline{t}^*$, we iteratively refine our initial guess for the time allocation. The refinement of $\underline{t}$ is also an optimization, shown in Optimization 2:

Optimization 2

$$\min_{\underline{t}\epsilon R^m} J(p(\underline{t}, W), T)$$

$$s.t. \quad \sum_{i=1}^{m} t_i = T \tag{5}$$

where, $p(\underline{t}, W)$ is the optimal trajectory across the waypoints $w_i \in W$ obtained by solving Optimization 1 using time allocation $\underline{t}$. This optimization is typically done by a gradient descent on $\underline{t}$ (see [9]).

Learning the optimal time allocations, $\underline{t}^*$, can greatly accelerate the trajectory generation process as it is the most computationally intensive step in many scenarios[9, 20].

We assume that the total flight time T is known. If the total time needs to be minimized, then we scale $T \to \eta T$, $\eta > 0$, along with the optimal allocation $\underline{t}^* \to \eta \underline{t}^*$, until no feasible trajectory $p(\eta \underline{t}^*, W)$ exists (exceeds max velocity or max acceleration constraints for example). This is typically done using a quick line search.

3.1. Modeling methodology

We use the standard transformer model, Vaswani et al. [16], for learning waypoint time allocations. We consider 2D trajectories from the SIGN [1] and ILGDB [10] hand drawing datasets here, but this approach can easily be extended to learn 3D trajectories. We modify the inputs and position function to suit the trajectory generation problem. We normalize the input data to make them invariant to homogeneous transforms (rotation and scaling). This is reasonable as the optimal time allocation ($\underline{t}^*$) scales linearly with total time T and is also independent of the distance scale between waypoints, [2]. Normalizing the inputs reduces the training data needed as the model does not have to explicitly learn that time allocations are invariant to homogeneous transforms, total time.

3.2. Data Preparation

In 2D, a rotation and distance scale invariant representation of the waypoints, W, can be obtained using the range-angle parametrization. The range and angle vectors d, θ are given by:

$$\underline{d} = [l_1, l_2, ..., l_{M-1}]; \quad \theta = [0, \theta_2, \theta_3, ..., \theta_{M-1}] \tag{6}$$

where d_i is the euclidean distance between two consecutive waypoints, i.e. $d_i = ||w_{i+1} - w_i||$. While θ_i is the angle between consecutive waypoints, i.e. $\theta_i = \angle(V_{i-1}, V_i)$, where the vector $V_i = w_i - w_{i-1}$. It should be noted that $-\pi < \theta_i \leq \pi$. We further non-dimensionalize the range vector $\underline{d}$, by scaling all components by a common factor.

$$\underline{d}' = \frac{1}{s} \cdot \underline{d},$$

where, $s > 0$, and $0 < d_i' \leq 1$. This reparametrization gives a representation of the input waypoints that is invariant to homogeneous transformations and is valid as the output $\underline{t}^*$ is unchanged by these transformations. We scale down the optimal time fractions ($\underline{t}^*$, $\Sigma t_i^* = T$) by the total time T to obtain the optimal time allocations of the trajectory. During inference, we scale back the model output $\underline{t}^*$ to match the desired total time T.

To obtain a sample trajectory with a desired number of waypoints (N), we use gestures/curves taken from the handwriting curves dataset (SIGN, ILGDB). For a sample gesture/curve, we preserve the start and end points and arbitrarily collocate the rest of the curve. This provides a sample set of waypoints with the desired length, N. Sample sets of waypoints from the training and test dataset along with their minimum-snap trajectories are shown in Fig 1
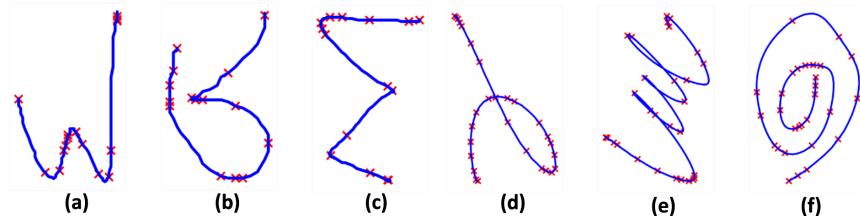

(a)   (b)   (c)   (d)   (e)   (f)

Figure 1: Sample curves collocated to 30 points from the ILGDB dataset ((a) to (c)) and SIGN dataset ((d) to (f))

3.3. Transformer Model

The encoder and decoder of the standard transformer [16] are adapted to learn the time allocation $\underline{t}$ given the input waypoints. The encoder and decoder consist of multi-head self-attention blocks and a position-wise feedforward network (FFN) block.

INPUT ENCODING AND POSITIONAL ENCODING

We use a linear layer to encode sample inputs ($W^{(i)}$) and outputs $T^{(i)}$ into the desired embedding space.

$W^{(i)} = [[l_1, 0], [l_2, \theta_2], [l_3, \theta_3], ..., [l_{M-1}, \theta_{M-1}]]$ where $W^{(i)}$ is a single sample set of waypoints taken from training dataset. We then solve for the optimal time allocation vector based on minimum snap, using [18], to get $T^{(i)} = [t_1, ..., t_{M-1}]$. We use the standard positional encoding function from [16]. The position encoding simply reflects the order in which the waypoints are visited.

MULTI-HEAD ATTENTION FUNCTION

The queries and keys used for our case are the encodings learnt by the input embedding layers. We use a linear layer that takes the range-angle parameters of the desired waypoints and calculates an embedding of desired size that can be used as the querys, keys and values.

Similarly, the decoder uses a linear layer that converts the time allocations $(0, t_1, t_2, ..., t_{N-1})$ to an appropriate output embedding of the desired size. The time allocation is padded with a zero in the beginning to serve as a starting point for the decoding process. Since we need only $N - 1$ time allocations, the decoder is run for that many steps to get the predicted time allocation vector $\underline{t}_{pred}$. We use the same attention masks for the inputs and outputs as done in [16].

FEEDFORWARD NETWORK

The output head of the transformer is a fully connected Linear layer with a 1 dimensional output (time allocation t_i). We use a cumulative L1 loss between the time allocations predicted by the

5

decoder $\underline{t}_{pred}$ and the true time allocation $\underline{t}^*$. The softmax labels layer from the standard transformer is not used as this is not a classification task, i.e. it is a regression problem. Again for convenience, we normalize the predicted time steps from the decoder $\underline{t}_{pred}$ to sum to 1. Hence the final output is given by eq (7)

$$\underline{t}_{out} = \frac{\underline{t}_{pred}}{||\underline{t}_{pred}||_1} \tag{7}$$

Optimization 1 is a hard non-linear optimization problem compared to Optimization 2 (Quadratic program). The gradient descent process for Optimization 1 may converge to time allocations that are local minimas. However, using an L1 loss provided good results for most trajectories (mean error of 15.7% in test set), as shown in section 4. Training with other loss functions, like in [12] for example, may provide better predictions but are not explored in this work.

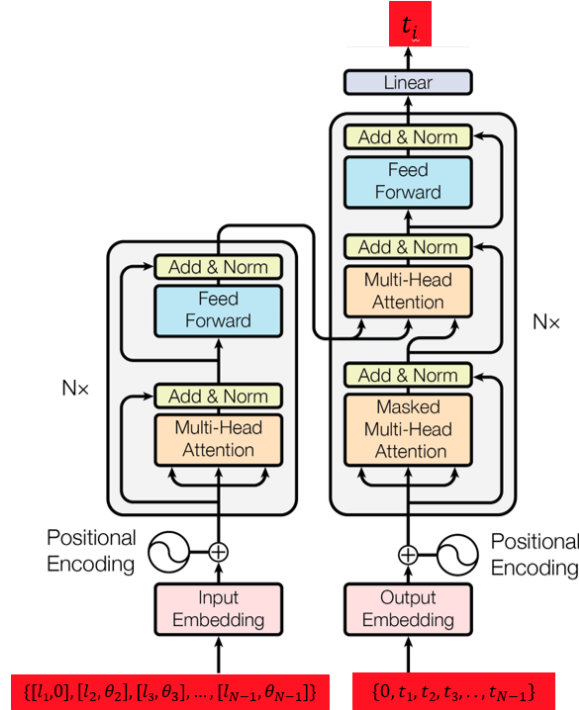The modifications made to the standard transformer for learning time allocations are shown in Fig 2.

Figure 2: Modified inputs and outputs to standard transformer [16] (shown in red) for trajectory generation

4. Results and Analysis

We trained a transformer model using gestures generated from SIGN dataset as described in Section 3.2. We use an embedding dimension of 32 for the inputs and outputs along with 16 attention heads. We use a total of 3 encoder and 3 decoder layers with a feedforward network hidden dimension

of 256 in the transformer model. This results in a total of $139,329$ trainable hyperparameters. To generate the training dataset we collocated every gesture in the SIGN dataset to various resolutions, from 3 to 30 waypoints. This resulted in a total of $493,668$ sets of waypoints. We split the dataset in the ratio of 5:1 for training and validation respectively. For producing the test dataset we collocated every curve in the ILGDB dataset down to 30 waypoints. Gestures that contained less than 30 points were not considered. This resulted in 6289 sets of waypoints.

For every set s of 30 waypoints in the ILGDB dataset, we calculated the minimum snap trajectories using the BGD (backtracking gradient descent) algorithm, implemented using [18], to obtain the locally optimal cost $J_{BGD}(s)$. The TVP method from [8] predicts a time allocation for a given set of waypoints, s, by using a trapezoidal velocity profile and we use this method to initialize the time allocations for our BGD algorithm. For the TVP method we use a maximum velocity of $5m/s$ and a maximum acceleration of $2.5m/s^2$. In our method, for any given set of points s we obtain the predicted time allocations $\underline{t}$ from our trained transformer model. We then solve Optimization problem 1 using the time allocations predicted by the transformer to obtain a trajectory with cost $J_T(s)$.

We compare our trained transformer model with prior work in literature, namely Almeida et al [2] and the TVP [8] method used for initialization. In [2] they use an MLP (multi-layer perceptron) that can only be trained for a fixed input-output size. Hence, they train multiple MLPs for different input sizes (3 to 30) to learn the time allocations. The combined models with all 28 MLPs contain $\sim 438,800$ trainable hyper-parameters. The cost/snap of the trajectories generated using the time allocations predicted from the MLP is given by J_{MLP}. For fair comparison, all the costs J_{BGD}, J_T, J_{MLP} and J_{TVP} are calculated using the same final time $\sum t_i = T$ obtained using the TVP method.

We evaluate the optimality of the predicted time allocations from the transformer model based on the total trajectory cost, i.e. total snap. Since the trajectory is a 7th order polynomial (see [9] for further discussion on polynomial order), the total snap of a given trajectory scales with the 7th order of total time T, i.e. $f(\alpha \underline{t}) \sim \alpha^7 f(\underline{t}) \ \forall \ \alpha > 0$ where $f(\underline{t})$ is the snap of the trajectory with time allocations $\underline{t}$. Due to this we normalize the costs by their 7th root, i.e.:

$$J'_{BGD}(s) = J_{BGD}^{1/7}(s); \ J'_T(s) = J_T^{1/7}(s); \ J'_{MLP}(s) = J_{MLP}^{1/7}(s); \ J'_{TVP}(s) = J_{TVP}^{1/7}(s) \quad (8)$$

We use the cost/snap of the trajectory generated from BGD method as our baseline. We compare the relative error ($E(s)$) between trajectories generated from other methods and the BGD method to evaluate their optimality. For a given set of waypoints, s, we define the relative errors for the different methods, i.e. our method (transformers), MLP method [2] and TVP [8] as shown in (9)

$$E_T(s) = \frac{(J'_T(s) - J'_{BGD}(s))}{J'_{BGD}(s))} \cdot 100; \qquad E_{MLP}(s) = \frac{(J'_{MLP}(s) - J'_{BGD}(s))}{J'_{BGD}(s)} \cdot 100;$$
$$E_{TVP}(s) = \frac{(J'_{TVP}(s) - J'_{BGD}(s))}{J'_{BGD}(s)} \cdot 100 \qquad (9)$$

Fig 3 shows the histogram of the relative errors, $E(s)$ of the different methods over all sets of waypoints in the ILGDB test dataset. On average, the cost error in the transformer model is $E_T^{avg} = 15.7\%$ with standard deviation of 14.6% over the BGD baseline. The MLP method [2] has a mean error in cost of $E_{MLP}^{avg} = 21.4\%$ with standard deviation of 13.7%. The TVP initialization

had a mean cost error of $E_{TVP}^{avg} = 50.7\%$. The figure shows that the time allocations predicted by the transformer model are better than the time allocations predicted by the MLP method. This is because the transformer model is able to use data over all input sizes and model the sequential relation across the input waypoints. We observe that the transformer model is able to guess better time allocations than the baseline BGD method itself. This is represented by the region of the histogram (Fig 3) with $E_T < 0$ and corresponds to $\sim 10.7\%$ of test dataset. The model is able to predict better because the BGD method can get stuck at local minimas leading to sub-optimal time allocations. This was also observed in the MLP model which was able to make better predictions than the baseline ($E_{MLP} < 0$) and corresponds to $\sim 3.1\%$ of the test dataset.
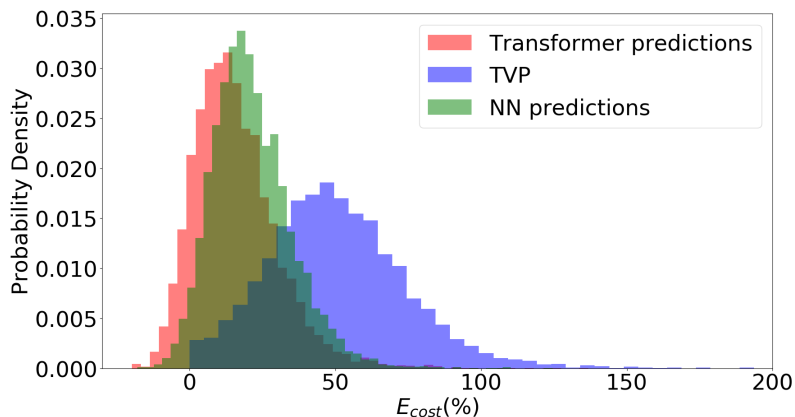

Figure 3: Histograms of error in cost E_{cost} between using transformer and a MLP

To demonstrate the ability of our approach to model the sequential nature of the problem, we test the transformer model on input sizes outside its training dataset. The gestures in the training dataset were collocated to sizes within $[3, 30]$. So we generate a test set by collocating all gestures in the ILGDB dataset to 40 waypoints. Fig 4 shows the error distribution between the predictions of the trained model and their BGD baselines for trajectories with 40 waypoints. This result shows that transformer models can generalize to input sizes larger than the training dataset. As seen in Fig 4, the transformer predictions are close to the costs obtained by using the BGD method. The average cost error in model predictions for 40 waypoints is $E_T^{avg} = 42.7\%$ over the BGD baseline.

All the above trajectory computations had an average computation time in the order of $10^{-3}s$ (Intel-9750H processor and NVIDIA RTX 3060 graphics processor). For real time inference over large inputs ($> 10^2$), we must reduce the context window size of the transformer and modify self-attention functions such as [17, 4]. Additionally, the attention computation across all context windows can be parallelized to make it faster. These introduce more model parameters that require careful selection. These selection methods have been studied extensively in the literature, such as in speech recognition [13] where thousands of audio packets are processed per second. These modifications are not performed in our work and we only evaluate the performance on the standard transformer.
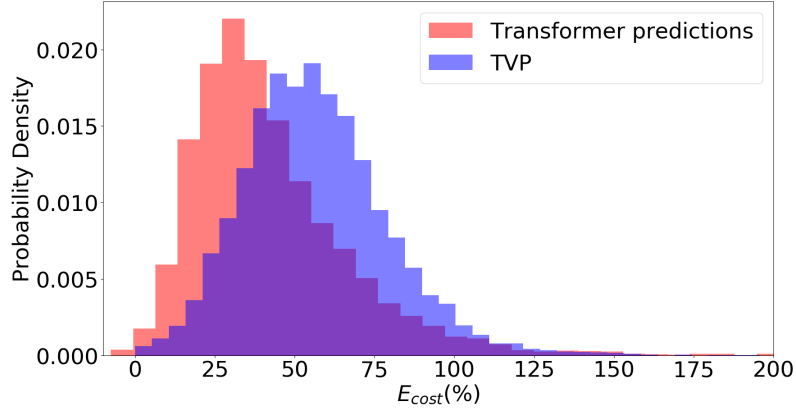
Figure 4: Error histogram in cost E_{cost} from transformer predictions on trajectories with 40 waypoints (out of distribution)

4.1. Attention patterns

The mean of the attention maps(across all heads) in layer 1 and layer 2 of the encoder-decoder attention block during inference is shown in Fig 5. The encoder-decoder attention maps have learnt to attend to the neighbouring prior and future input waypoints. The block diagonal structure indicates that there is a "horizon" over which input waypoints influence the time allocation of a given segment. Learning the influence of adjacent waypoints, including the future waypoints helps the transformer make better predictions on the optimal time allocation. Given the block diagonal nature of the attention maps, we can choose a smaller size for the context window. However, we have not used context windows to limit the attention span in this work.
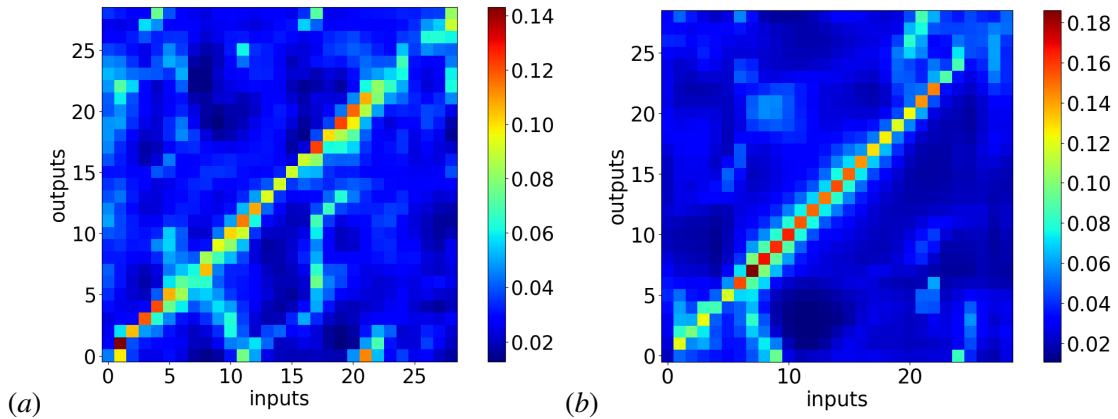

Figure 5: Averaged attention maps across all heads of encoder-decoder attention block (a) average attention across all heads in layer 1 (b) average attention in layer 2 of encoder-decoder block

4.2. Sample efficiency

We study the sensitivity of the model performance with the size of the training dataset. We select random subsets of the training data and measure the performance of the corresponding trained model (E_T^{mean}) on the test data (ILGDB). In Fig 6 we see that the transformer model is able to perform just as well well as the MLP model using only 10% of the training data ($E_T^{mean} = 22.6\%$). This means that the model can provide good predictions using fewer training samples for a given input size, demonstrating its sample efficiency.
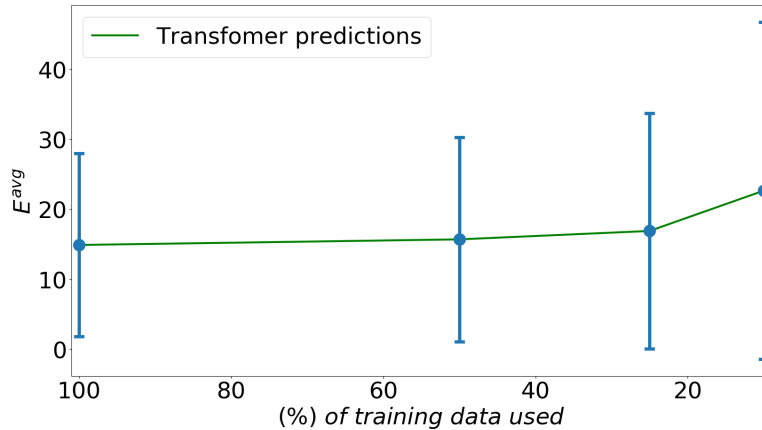

Figure 6: Average cost error over test data vs amount of training data used

5. Conclusion

The experiments shown here demonstrate that a Transformer can successfully predict near optimal time allocations and even provide good starting guesses for input sizes greater than the training data. The transformer model is able to accomplish this with fewer model hyperparameters ($< 1/3^{rd}$) and training data samples ($1/10^{th}$) compared to using an MLP. The learnt attention patterns between input and output embeddings indicate a "horizon" of waypoints (before and after) which influence the time allocation at a given waypoint.

By learning the optimal time allocations for any given sequence of waypoints, our method removes the bi-level structure of the optimization problem. This training approach can be extended to other bi-level optimization problems (minimum jerk and minimum time [20] for example). The major bottleneck for running the above standard transformer model is computing the self-attention functions in real-time for large inputs. This requires several modifications to the transformer model that are discussed in section 4 and is a direction for future work.

6. Acknowledgements

References

[1] Abdullah Almaksour et al. "Personalizable pen-based interface using lifelong learning". In: *2010 12th International Conference on Frontiers in Handwriting Recognition*. IEEE. 2010, pp. 188–193.

[2] Marcelino M de Almeida, Rahul Moghe, and Maruthi Akella. "Real-time minimum snap trajectory generation for quadcopters: Algorithm speed-up through machine learning". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 683–689.

[3] Declan Burke, Airlie Chapman, and Iman Shames. "Fast Spline Trajectory Planning: Minimum Snap and Beyond". In: *arXiv preprint arXiv:2105.01788* (2021).

[4] Krzysztof Choromanski et al. "Rethinking attention with performers". In: *arXiv preprint arXiv:2009.14794* (2020).

[5] Fei Gao et al. "Online safe trajectory generation for quadrotors using fast marching method and bernstein basis polynomial". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 344–351.

[6] Charles R Hargraves and Stephen W Paris. "Direct trajectory optimization using nonlinear programming and collocation". In: *Journal of guidance, control, and dynamics* 10.4 (1987), pp. 338–342.

[7] Steven M LaValle and James J Kuffner Jr. "Randomized kinodynamic planning". In: *The international journal of robotics research* 20.5 (2001), pp. 378–400.

[8] Sikang Liu et al. "High speed navigation for quadrotors with limited onboard sensing". In: *2016 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2016, pp. 1484–1491.

[9] Daniel Mellinger and Vijay Kumar. "Minimum snap trajectory generation and control for quadrotors". In: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 2520–2525.

[10] Ney Renau-Ferrer et al. "The ILGDB database of realistic pen-based gestural commands". In: *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*. IEEE. 2012, pp. 3741–3744.

[11] Charles Richter, Adam Bry, and Nicholas Roy. "Polynomial trajectory planning for aggressive quadrotor flight in dense indoor environments". In: *Robotics research*. Springer, 2016, pp. 649–666.

[12] Gilhyun Ryou, Ezra Tal, and Sertac Karaman. "Real-Time Generation of Time-Optimal Quadrotor Trajectories with Semi-Supervised Seq2Seq Learning". In: *6th Annual Conference on Robot Learning*. 2022. URL: https://openreview.net/forum?id=ZJYbW8Cwys.

[13] Sainbayar Sukhbaatar et al. "Adaptive attention span in transformers". In: *arXiv preprint arXiv:1905.07799* (2019).

[14] Srinath Tankasala, Can Pehlivanturk, and Mitch Pryor. "Near minimum time trajectory planning for surveying using UAVs". In: *2022 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2022, pp. 873–880. DOI: 10.1109/ICUAS54217.2022.9836132.

[15] Srinath Tankasala et al. "Generating Smooth Near Time-Optimal Trajectories for Steering Drones". In: *2022 European Control Conference (ECC)*. 2022, pp. 1484–1490. DOI: 10.23919/ECC55457.2022.9838451.

[16] Ashish Vaswani et al. "Attention is all you need". In: *Advances in neural information processing systems*. 2017, pp. 5998–6008.

[17] Sinong Wang et al. "Linformer: Self-attention with linear complexity". In: *arXiv preprint arXiv:2006.04768* (2020).

[18] Zhepei Wang et al. "Generating large-scale trajectories efficiently using double descriptions of polynomials". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2021, pp. 7436–7442.

[19] Yifan Zhang et al. "Recurrent LSTM-based UAV Trajectory Prediction with ADS-B Information". In: *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE. 2022, pp. 1–6.

[20] Boyu Zhou et al. "Robust and efficient quadrotor trajectory generation for fast autonomous flight". In: *IEEE Robotics and Automation Letters* 4.4 (2019), pp. 3529–3536.