Preprints of the 21st IFAC World Congress (Virtual)
Berlin, Germany, July 12-17, 2020

Practical method to complete Learning Model Predictive Control with generalization capability

Ferenc Török * **Tamás Péni** *

* *Institue for Computer Science and Control (SZTAKI)*
H-1111, Budapest, Kende u. 13-17, Hungary
E-mails: ferenc.torok@sztaki.mta.hu, peni.tamas@sztaki.mta.hu

Abstract: The paper presents a practical method to complete Learning Model Predictive Control (LMPC) with generalization capability. LMPC has been developed by F. Borrelli and his co-authors for systems performing iterative tasks. The method is based on saving the state trajectories of successful runs and using this database to improve the control performance in the future iterations. When the controller faces a new task, the database is cleared and the learning phase starts over. This paper addresses the question of how a general knowledge base can be built to warm start the learning process. As a potential solution, a practical method is proposed. The algorithm is tailored specifically to the autonomous racing application, but the concept can be extended to a wider class of control problems. The procedure includes the construction of special teaching tracks, on which the trajectory database is generated and a multi-step migration procedure for transferring the learned trajectories onto any new track. The efficiency of the method is demonstrated by numerical simulations.

Keywords: model predictive control, nonlinear control, iterative and repetitive learning control, autonomous racing

1. INTRODUCTION

Designing learning based controllers for autonomous driving is popular and challenging research field. The majority of the papers published in this topic are focusing on the problems of autonomous traffic and consider highway or urban scenarios, where the goal is the safe navigation among the other moving entities Gao et al. (2012); Kuwata et al. (2019). Some researchers however are interested in designing controllers for autonomous racing, where the goal is to operate the vehicle close to its physical limits and achieve higher speed and better lap times on specific racetracks Liniger et al. (2015a) Kapania and Gerdes (2015) Kabzan et al. (2019) Brunner et al. (2017).

Learning Model Predictive Control (LMPC) is one promising control design method in this field. The algorithm was presented by F. Borrelli and his co-authors in Rosolia et al. (2017); Brunner et al. (2017). The method is originally developed for systems performing iterative tasks (Rosolia and Borrelli (2017)); its concept follows the iterative learning control (ILC) scheme Bristow et al. (2006). The procedure is based on saving the trajectories of successful runs and using them to improve the control performance in the future iterations. Technically the saved trajectories are used to construct a local, polytopic terminal set for a short horizon MPC evaluated online at each time instant. The convergence of the learning process is proved in Rosolia and Borrelli (2017).

Though the algorithm has several attractive features, still there is a room for improvement. One possible direction is the generalization of the knowledge base. In the original LMPC framework whenever the controller faces a new task, i.e. the vehicle has to run on a previously unseen track, the database is cleared and the learning phase starts over. In this paper, a practical procedure is proposed for building a generalized knowledge base that can be used to warm start the learning process when a new track is given. First, a set of special teaching tracks have to be constructed on which the learning is performed. Second, a multi-step data migration procedure is proposed that transfers the learned trajectories onto the new track to construct an initial database. It is important to mention that the method also allows to reuse the knowledge acquired through repetitive learning on non-recursive tracks as well hence allowing to use the power of LMPC for non-iterative tasks. The applicability and efficiency of the procedure are demonstrated by numerical simulations.

The paper is organized as follows. Section 2. gives a precise formulation of the control problem to be solved. Section 2. summarizes the key elements of the LMPC algorithm. These two sections are based mainly on Rosolia and Borrelli (2017); Rosolia et al. (2017); Brunner et al. (2017). The novel contributions of the paper are presented in Section 3. Numerical simulations performed with the proposed method are analyzed in Section 4. Finally, the paper closes by drawing the most important conclusions.

2. LEARNING MPC FOR AUTONOMOUS RACING

LMPC is designed for systems performing iterative tasks. It is constructed to learn from the iterations, that is to improve the control performance in each turn. In case of autonomous racing this means the vehicle runs multiple laps on a closed track and improves the driving performance, e.g. decreases the lap time in each run. The detailed description and in-depth theory of the LMPC method is described in Rosolia and Borrelli (2017); Rosolia et al. (2017) while its extension to autonomous racing is discussed in Brunner et al. (2017). This section gives a brief summary of the LMPC framework: only the main elements that are necessary to follow the next sections are recalled.

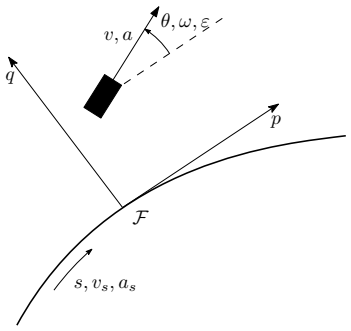


Fig. 1. Coordinate frames, states and inputs.

Note that this version differs at some points from the original algorithm: small improvements and modifications are made to make the procedure more suitable for our purposes.

Vehicle model. The discrete-time nonlinear dynamical model of the vehicle is given in the usual state space form as follows

$$x_{k+1} = f(x_k, u_k), \tag{1}$$

where x_k and u_k collect the states and inputs at the k^{th} time instant:

$$x_k = [p_k\ q_k\ \theta_k\ \omega_k\ v_k\ s_k\ v_{sk}]^T \quad u_k = [a_k\ \varepsilon_k\ a_{sk}]^T. \tag{2}$$

The equations of motion of the vehicle are expressed in a moving frame, attached to the centerline of the track (Fig. 1), so $f(\cdot)$ is given as follows:

$$f(x_k, u_k) = \begin{bmatrix} p_k + T_s\left(-v_{sk}\left(1 - c(s_k)q_k\right) + v_k \cos\theta_k\right) \\ q_k + T_s\left(-c(s_k)v_{sk}p_k + v_k \sin\theta_k\right) \\ \theta_k + T_s\omega_k \\ \omega_k + T_s\left(\varepsilon_k - c(s_k)a_{sk} - g(s_k)v_{sk}^2\right) \\ v_k + T_s a_k \\ s_k + T_s v_{sk} \\ v_{sk} + T_s a_{sk} \end{bmatrix} \tag{3}$$

The state variables p_k, q_k, θ_k and ω_k represent the position, orientation and angular velocity of the vehicle in $\mathcal{F}$ respectively. v_k is the scalar velocity of the vehicle, s_k and v_{sk} are the distance and the velocity of the origin of $\mathcal{F}$ along the centerline of the path. The inputs a_k, ε_k and a_{sk} quantify the scalar acceleration, angular acceleration (in the world frame) and the acceleration of the origin of $\mathcal{F}$ along the centerline respectively. The function $c(s)$ is the curvature along the centerline and $g(s) = \partial c(s)/\partial s$. T_s is the discrete time step. Note that, the vehicle model above is simpler than that is used in Rosolia et al. (2017). This model has been chosen to simplify the presentation of our results. The proposed algorithms can however be easily extended for more complex vehicle dynamics as well.

Receding Horizon Control. To obtain the control inputs at each time instant k, LMPC requires to solve the following finite horizon optimization problem:

$$J^{LMPC,j}(x_t^j) = \min_{u_{t|t},\ldots,u_{t|t+N-1}} \sum_{k=t}^{t+N-1} h(x_{t|k}, u_{t|k}) + \\ + Q^{j-1}(x_{t,t+N}^j) \tag{4a}$$

s.t.

$$x_{t|k+1} = f(x_{t|k}, u_{t|k}),\ \forall k \in \{t, \ldots, t+N-1\} \tag{4b}$$

$$x_{t|t} = x_t^j \tag{4c}$$

$$x_{t|k} \in \mathcal{X},\ u_{t|k} \in \mathcal{U},\ \forall k \in \{t, \ldots, t+N-1\} \tag{4d}$$

$$x_{t|t+N} \in SS^{j-1} \tag{4e}$$

where x_t^j denotes the state at time t of the j^th iteration. Let the optimal solution of (4) be

$$\mathbf{x}_{t|t:t+N}^{*,j} = \left\{x_{t|t}^{*,j}, \ldots x_{t|t+N}^{*,j}\right\} \tag{5a}$$

$$\mathbf{u}_{t|t:t+N}^{*,j} = \left\{u_{t|t}^{*,j}, \ldots u_{t|t+N-1}^{*,j}\right\}. \tag{5b}$$

Then at time t of the j^{th} iteration the first input $u_{t|t}^{*,j}$ of the optimal solution is applied to the plant. The optimization (4) is then solved again at time instance $t+1$, yielding a receding horizon control.

The main components of (4) are detailed as follows.

Parameterization of s. In order to use the LMPC method, the system has to start (at least closely) from the same initial state at each iteration. This can be easily satisfied by most of the state variables, except s_k. Since v_k is nonnegative, s_k is continuously increasing. In order to prevent s_k from growing unboundedly, its value is reset after every lap. The resetting has to be carefully performed in order to avoid jumps in the vicinity of the finish line. For this, s is parameterized to vary in the interval $[-2L_{path}, 0]$, where L_{path} is the length of the centerline. Let s_t^i denote the 6^{th} coordinate of the state vector (2) at time t of the j^{th} iteration. With this parametrization we can determine time $t_j^\star$ when the vehicle crossed the finish line: $t_j^\star$ is the first time instant satisfying the following conditions: $s_{t_j^\star-1}^j \in [-2L_{path}, -L_{path}[$ and $s_{t_j^\star}^j \in [-L_{path}, 0]$. The transformation that maps the last state of an iteration to the initial state of the next iteration can then be defined as follows:

$$x_0^{j+1} = x_{t_j^\star}^j - [0, 0, 0, 0, 0, L_{lap}, 0]^T, \tag{6}$$

This choice of parametrization allows the objective function to decrease as the vehicle travels towards the finish line and avoids jumps in the state trajectory at the end of the iterations.

Objective function. Instead of penalizing only the lap time, as it is done in Rosolia et al. (2017); Brunner et al. (2017), a quadratic cost function is used:

$$h(x_k, u_k) = x_k^T P x_k + u_k^T Q u_k + R^T x_k + S^T u_k \tag{7}$$

where $P, Q \succeq 0$ (positive semidefinite). The quadratic structure offers larger freedom in performance specification. The goal of the controller is to minimize the lap time while keeping ε as small as possible and satisfying the state- and input constraints. In order to fulfill these requirements, the quadratic cost (7) is constructed with the following considerations Lam et al. (2010); Liniger et al. (2015b): a.) The cost of p_k is chosen to be large compared to the other state variable costs in order to guarantee that $\mathcal{F}$ is always placed in the point of the path closest to the vehicle; b.) the costs on the states s_k and ε_k are chosen to be higher then on the rest of the states.

This cost results in maximizing the progress of the origin of $\mathcal{F}$ along the path, while keeping it in the nearest point of the centerline to the vehicle (High cost on p).

Sampled Safe Set. The key element of the LMPC algorithm is the *Sampled Safe Set* of past trajectories. To formally define this set, let

$$\mathbf{x^j} = \left[x_0^j\ x_1^j\ \ldots\ x_{t_j^\star-1}^j\right], \quad \mathbf{u^j} = \left[u_0^j\ u_1^j\ \ldots\ u_{t_j^\star-1}^j\right] \tag{8}$$

collect the control inputs applied to the vehicle model (1) and the resulting states at iteration j. With these vectors, the Sampled Safe Set of the j^{th} iteration is defined as

$$SS^j = \left\{ \bigcup_{i \in M^j} \bigcup_{t=0}^{\infty} x_t^i \right\} \tag{9}$$

where M^j is the set of all iteration indexes, in which iterations the vehicle has successfully crossed the finish line. In this respect, SS^j is technically the collection of all the trajectories along which the vehicle has completed a lap until the j^{th} iteration. At the very first iteration SS^1 is initialized with a trajectory which is calculated as if the car would track the centerline perfectly with a sufficiently small constant velocity, so that all the state and input constraints are fulfilled.

Iteration Cost. The cost-to-go *value* at time t of the j^{th} iteration is originated from the closed loop trajectory and the corresponding input sequence as follows:

$$J^j_{t \to t^\star_j}(x^j_t) = \sum_{k=t}^{t^\star_j} h(x^j_k, u^j_k) \tag{10}$$

The cost-to-go *function* $Q^j(\cdot)$ is introduced to assign the minimum cost-to-go *value* to the points of SS^j along its trajectories. Formally, $Q^j(\cdot)$ is defined as follows:

$$Q^j(x) = \begin{cases} \min\limits_{(i,t) \in F^j(x)} J^i_{t \to t^\star_j}(x), & \text{if } x \in SS^j \\ \infty, & \text{otherwise} \end{cases} \tag{11}$$

where $F^j(x)$ is defined as:

$$F^j(x) = \left\{ (i,t),\ i \in [0,j], t \geq 0 \text{ with } x = x^i_t,\ \text{for } x^i_t \in SS^j \right\}. \tag{12}$$

Before the very first iteration, Q^1 is initialized using (10) and (11) along SS^1.

SS update step. At the end of a successful iteration, the sampled safe set SS is updated as follows:

$$SS^j = SS^{j-1} \cup \left\{ \mathbf{x}^{j-1}, \tilde{\mathbf{x}}^{j-1} \right\} \tag{13}$$

where $\mathbf{x}^{j-1}$ is the trajectory of the system at the $j-1$-th iteration and $\tilde{\mathbf{x}}^{j-1}$ is a modification of $\mathbf{x}^{j-1}$ for augmenting the SS beyond the finish line:

$$\tilde{\mathbf{x}}^{j-1} = \mathbf{x}^{j-1} + s_{lap} e_6 \mathbf{1}^{1 \times N^{j-1}} \tag{14}$$

where N^{j-1} is the number of trajectory points in $\mathbf{x}^{j-1}$ and $\mathbf{1}^{1 \times N^{j-1}} = [1, ..., 1] \in \mathbb{R}^{1 \times N^{j-1}}$ row vector.

Relaxations. The LMPC problem (4) with the cost function (7) is a quadratic program with integer variables (4e) and nonlinear (4b) constraints. It is therefore numerically demanding. To cast it to a computationally tractable problem, some relaxations on (4) are applied. First, the nonlinear system dynamics (3) is approximated by an LTV system

$$x_{k+1} \cong A_k x_k + B_k u_k + g_k \tag{15}$$

where A_k, B_k and g_k are computed by linearizing (3) around a trajectory obtained at the previous iteration (Lam et al. (2010); Liniger et al. (2015b)).

Second, the terminal state constraint (4e) is relaxed to the convex hull of SS^{i-1} and $Q^{j-1}(\cdot)$ is approximated by using barycentric approximation detailed e.g. in Warren et al. (2007) and Ju et al. (2005).

It is proven in Rosolia and Borrelli (2017) that with these relaxations, the convergence of the LMPC to a local optimum is still guaranteed.

3. BUILDING KNOWLEDGE BASE TO ADD GENERALIZATION CAPABILITY

It is not efficient that every time the controller faces an unknown track, the learning process is re-started from the beginning and the knowledge acquired on the past tracks is not exploited. In this section a practical method is proposed to overcome this limitation.

In our framework, we assume that all tracks the vehicle can face are constructed from a priori known track-primitives. In the first step a set of *teaching-tracks* is constructed from the track-primitives. The controller is then trained on these tracks by LMPC. The SS trajectories learned are saved in a database. When the vehicle faces a new track, called *test-track*, the trajectories of this database are migrated to the new track, hence forming an initial SS set for further learning. Hence the LMPC controller can use the knowledge acquired earlier and there is no need to start from scratch. It is important to note that although the teaching- and the test-tracks have to be built from the same types of track-primitives, the lengths of these primitives do not have to coincide.

Remark: Tracks built from track-primitives can be quite useful in many real scenarios, not limited to the autonomous vehicle racing application. Sufficiently flexible tracks composed of primitives can easily be designed by a high level routing algorithm for example for AGVs, robot arms or laser profiling tools.

To simplify the presentation, we restrict our attention to tracks composed of straight sections and 90° turns. We present the main steps of our algorithm on this specific problem. The procedures, however, can be easily extended to more complex tracks built from a larger set of track-primitives.

Track-primitives and teaching-tracks. Since the directions of two successive turns have effect on the optimal trajectory between them, we choose straight sections with 45° turns at their both ends for track-primitives. The turns are created using quintic splines Piazzi et al. (2002) for sufficient differentiability. Depending on the direction of the turns, this would make 4 track-primitives, but due to the symmetricity it is sufficient to distinguish only two: one with two same turns at its ends and one with different turns (see Fig. 2).

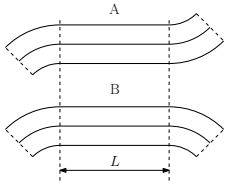
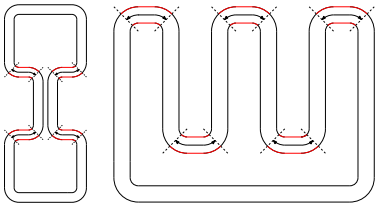


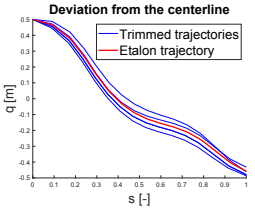
Fig. 2. The track-primitives.

Each teaching-track focuses on a single type of primitive with a specific length L. Several tracks with different length L are constructed. A track contains several pieces of these primitives isolated with sufficiently long straight sections, so that the successive primitives do not influence each other. Examples of teaching-tracks can be seen in Fig. 3.

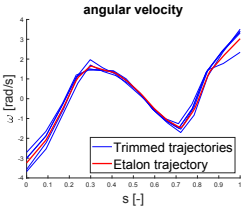


(a) A-type (b) B-type

Fig. 3. Schematic drawings of A and B type teaching-tracks. The track-primitives are marked on the figures.

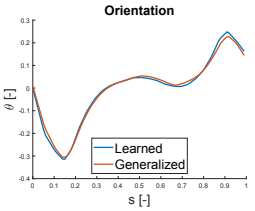


(a) Path deviation profiles.

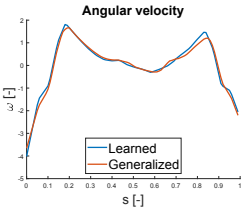


(b) Angular velocity profiles.

Fig. 4. State profiles of the trimmed and the etalon trajectories along an A-type track-primitive with $L = 2\,[m]$.



(a) Orientation profiles.EF



(b) Angular velocity profiles.

Fig. 5. State profile of the learned etalon trajectory and the generalized B-type primitive with $L = 5.5\,[m]$.

3.1 Post-processing of the learned SS

Recall that our goal is to use the knowledge acquired on teaching-tracks on previously unknown tracks by migrating the learned SS set onto it. The SS migration is a series of post-processing steps carried out on the SS trajectories learned on teaching-tracks. These post processing steps include the *trimming* of the relevant sections from the learned SS trajectories; *resampling* the points of the trimmed sections; creating *etalon trajectories* from the resampled trajectories; interpolating between etalon trajectories for reaching a specified length primitive; *pruning* the interpolated (generalized) trajectories and creating *bridge trajectories* between the successive track-primitives. The steps, that are detailed in this section have to be carried out on each SS trajectory but are only shown for the j^{th} one. In this way, the index j of the iteration number, is omitted in this and the next subsections.

Trimming. The relevant sections of the teaching-tracks (marked red in Fig. 3) have to be cut out from the total trajectory. The i^{th} relevant section on the teaching-track $\hat{\mathbf{x}}^{\pi}_{L,i}$ is selected by using the curvilinear abscissa as follows:

$$\hat{\mathbf{x}}^{\pi}_{L,i} = \left\{ x \in \mathbf{x}^{\pi}_{L} : x^T e_6 \in [\underline{s}^{\pi}_{i}, \overline{s}^{\pi}_{i}] \right\}, \ \forall i = 1, ..., M \quad (16)$$

where $\pi \in \{A, B\}$ and $L \in \mathcal{L}_{\pi}$ are the type and length of the learned primitive. $\mathcal{L}_{\pi}$ is the set of learned lengths from primitive π. $i \in 1, ..., M$ is the index of the learned primitive on the teaching-track; M is the number of learned primitives from type π and length L. $\underline{s}^{\pi}_{i}, \overline{s}^{\pi}_{i}$ are the lower and upper boundaries of the curvilinear abscissa, between which the i^{th} learned primitive ("relevant" section) lies on the teaching-track. $\mathbf{x}^{\pi}_{L}$ is a complete trajectory on the teaching-track, from which the "relevant" sections $\hat{\mathbf{x}}^{\pi}_{L,i}$ $i = 1, ..., M$ are trimmed out.

Resampling and etalon trajectories. In this step, the etalon trajectory $\bar{\mathbf{x}}^{\pi}_{L}$ of the track-primitive of type π and length L is created from the trimmed trajectories $\hat{\mathbf{x}}^{\pi}_{L,i}$, $i \in \{1, \ldots, M\}$. For this, all $\hat{\mathbf{x}}^{\pi}_{L,i}$ trajectory segments are normalized in the s coordinate. Then, they are resampled (interpolated and evaluated) at K equidistantly placed points at $s = [0, 1, ...K-1]/(K-1)$. The resampling ensures that all trajectories are represented by the same number of points placed at same locations. (K is typically chosen to be a few times multiple of the number of samples representing the longest learned primitive before resampling.)

After $\hat{\mathbf{x}}^{\pi}_{L,i}$ is normalized and resampled, the etalon trajectory $\bar{\mathbf{x}}^{\pi}_{L}$ of a track-primitive of type π and length L is obtained by taking the average of the learned trajectories:

$$\bar{\mathbf{x}}^{\pi}_{L} = \bigcup_{j=1}^{K} \left\{ \sum_{i=1}^{M} \frac{1}{M} \hat{x}^{\pi}_{L,i,j} \right\} \quad (17)$$

where $\hat{x}^{\pi}_{L,i,j}$ is the j^{th} point of $\hat{\mathbf{x}}^{\pi}_{L,i}$. As an example, the angular velocity and path deviation profile of the trimmed trajectories and the etalon trajectory of an A-type turn with $L = 2\,[m]$ can be seen in Fig. 4.

The etalon trajectories represent the knowledge acquired on the teaching-tracks. They are stored in a database and recalled when the vehicle faces a new (previously unseen) track.

3.2 SS migration

Interpolation. Assume a database storing a set of etalon trajectories is available and a new track is given. The goal is to construct an initial sampled safe set by using the knowledge stored in the database. For this, suppose the new track contains a track-primitive of type π of length L_{des}. The trajectory segment $\tilde{\mathbf{x}}^{\pi}_{L_{des}}$ corresponding to this track-primitive can be constructed from the etalon trajectories of type π by linear interpolation as follows:

$$\tilde{\mathbf{x}}^{\pi}_{L_{des}} = \bigcup_{j=1}^{K} \left\{ \frac{\overline{L} - L_{des}}{\overline{L} - \underline{L}} \bar{x}^{\pi}_{\underline{L},j} + \frac{L_{des} - \underline{L}}{\overline{L} - \underline{L}} \bar{x}^{\pi}_{\overline{L},j} \right\} \quad (18)$$

where $\bar{x}^{\pi}_{L,j}$ is the j^{th} point of the *etalon trajectory* of the primitive of type π and length L and

$$\begin{aligned} \underline{L} &= \sup \{ L \in \mathcal{L}_{\pi} : L \leq L_{des} \} \\ \overline{L} &= \inf \{ L \in \mathcal{L}_{\pi} : L > L_{des} \}. \end{aligned} \quad (19)$$

By performing several experiments we have found that linear interpolation provides a suitable initial guess for the optimal trajectory so it can be used to initialize the safe set.

Scaling and resampling. The trajectory segments constructed by (18) are still normalized in s. Therefore, they have to be rescaled to fit to the actual length of the corresponding track-primitives. Note also that these trajectory

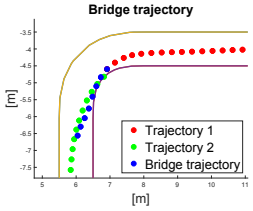


Fig. 6. Bridge trajectory between two successive primitives.

segments still contain K points. This can be unnecessary large, making it difficult to properly evaluate $J(\cdot)$ and $Q(\cdot)$. Therefore, the trajectory segments are "downsampled" such that the distance between any two successive points be smaller than the distance the vehicle can travel in one discrete time step.

Bridge trajectories. At this point we have trajectory segments for every track-primitive of the new track. However, the continuity of the state trajectories between two successive primitives is not guarantied. To remedy this issue and generate continuous state trajectories bridge trajectories are constructed to connect the successive track-primitives. These trajectories can be obtained by solving the LMPC problem (4) from the last point of each migrated trajectory section with the following modification: In (4e) use the already migrated trajectory segment points instead of SS^{j-1}. The solution of the LMPC problem defines a feasible state trajectory that starts from the last state of a track-primitive and ends among the states of the primitives following it. An example of a bridge trajectory can be seen on Fig. 6. The migrated SS set is composed from the migrated trajectory segments together with the bridge trajectories.

4. CASE STUDY

In this section a numerical example is presented to demonstrate the efficiency and applicability of the proposed method.

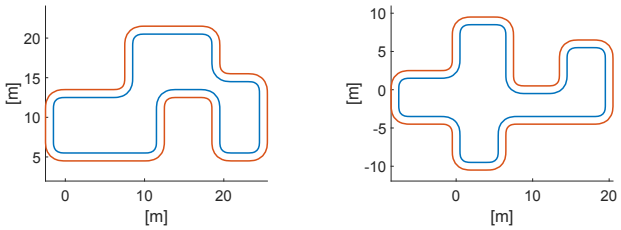
4.1 Teaching and test tracks

We considered race-tracks, built from track primitives A and B that are depicted in Fig. 2. For learning, 10 different teaching tracks were constructed with length $\mathcal{L}_\pi = \{1, 2, 3, 5, 10\}\,[m]$ where $\pi = \{A, B\}$. The structure of the teaching-tracks are illustrated in Fig. 3. The time step of the simulation was $T_s = 0.05\,[s]$ and the prediction horizon during the teaching process was $N = 15$. The training was run until the iteration cost converged to a steady state (minimal) value.

The performance of the LMPC controller with migrated SS was then analyzed on two test tracks that can be seen in Fig. 7. These tracks contain A and B-type primitives of lengths different from the learned ones. Results of simulations on these tracks with and without migrated SS set are presented next. We have achieved very similar results on both test tracks, regarding the performance of the controller with SS migration. Due to the shortage of space, we only detail the results of the first track.

4.2 Performance on unknown tracks

The learning process took an average of 8 laps on each teaching-track and lasted for approximately $80\,[min]$ on the 10 tracks. (The relatively long, $N = 15$, horizon allows



(a) Test track 1. (b) Test track 2.

Fig. 7. The test tracks on which the SS migration was tested.

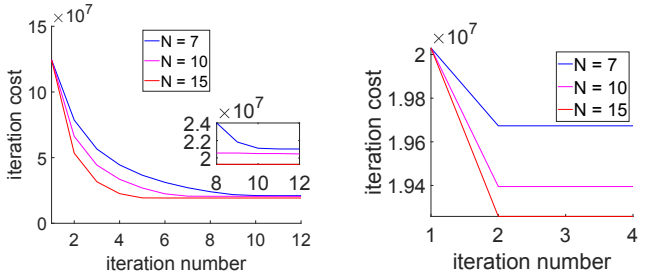
the controller to converge to the optimum in only a few iterations.) After the learning was completed, we have carried out the post processing steps of the SS migration to reach the etalon trajectories which were then saved. After these, obtaining the migrated SS from the saved etalon trajectories took approximately $2.7\,[s]$ on the test-tracks.

Then, we run the LMPC controller on the test-tracks with and without migrated SS. In both cases we run the controller with $N = 7, 10$ and 15 and logged the iteration costs. These iteration costs on test-track 1 without and with migrated SS can be seen in Fig. 8. In general, it can be said that the final performance of the LMPC controller is approximately the same in all cases. The major difference between the controllers with and without migrated SS is the number of iterations required for the iteration cost to converge. With migrated SS the controller reaches the local optimal solution much faster and already starts from a near optimal iteration cost. In these cases, the costs of the very first iterations differed from the minimal value by less than 5%. The times required for convergence in the different setups can be examined in Table 1.

Table 1. Convergence times and optimal values reached with and without SS migration with different horizon lengths.

	Without SS migration		**With SS migration**	
	Conv. time	Minimal cost	Conv. time	Minimal cost
$N = 7$	$374.5[s]$	$2.18 \cdot 10^7$	$45.6[s]$	$1.97 \cdot 10^7$
$N = 10$	$276.1[s]$	$2.12 \cdot 10^7$	$45.3[s]$	$1.94 \cdot 10^7$
$N = 15$	$220.8[s]$	$1.93 \cdot 10^7$	$44.6[s]$	$1.93 \cdot 10^7$

Apart from the convergence time, SS migration also has an effect on the minimal value of the iteration cost. This is in connection with the fact that the final locally optimal control law slightly depends on the length of the prediction horizon as it is visualized in the subfigure of Fig. 8. The longer the prediction horizon is, the faster the iteration cost converges and the smaller minimal iteration cost is achieved. One of the appealing features of SS migration is that the teaching can be carried out with a long prediction horizon and then, after SS migration, the controller can run with a much shorter one which is computationally cheaper. After a long horizon teaching, a controller with a short prediction horizon can reach smaller iteration costs than one with the same horizon length but without the migrated SS. (It also reaches this minimum much faster, as it was detailed previously.) This feature of the SS migration is illustrated in Table 1, from which one can read that the controller with $N = 7, 10$ converges to a lower iteration cost if SS migration is applied. On the other hand, the costs to which the controllers with $N = 15$ converge differ from each other with $6.4 \cdot 10^{-3}$ % in the two



(a) Learning from scratch.

(b) Learning after SS migration.

Fig. 8. The iteration costs of the LMPC controller on teaching-track 1. without and with SS migration.

cases. This is due to the fact, that the teaching process was also carried out with $N = 15$.

To summarize the experiences of the case study, we can say that an LMPC controller with migrated SS converges to a minimal cost much faster than the same controller without SS migration. Furthermore, if the controller has a shorter horizon than that was used for teaching, the controller will reach a better final performance if SS migration is used. Also, if the teaching and the testing horizon length is the same, the final performance is identical with and without SS migration.

5. CONCLUSION

A practical procedure for generalizing the knowledge acquired by learning model predictive control has been presented. The method is tailored specifically to the autonomous racing application. It has been shown if the race-tracks are built from a-priori known track-primitives, a set of teaching tracks can be constructed on which the optimal motion profile (the SS set) for each track primitive can be learned. An efficient, multi-step strategy is proposed to migrate the SS trajectories learned on teaching tracks to a different, *a priori* unknown track, to initialize the further learning. This initialization allows the LMPC controller to make generalizations from its acquired knowledge during training. The proposed strategy has several attractive features. First, the number of iterations required for learning the optimal control on a track largely depends on the prediction horizon N of the LMPC control. As N increases, the number of iterations until optimality decreases meanwhile the computational costs heavily grow. This leads to a compromise between learning rate and computational expenses, which are mightily limited in real time applications. With SS migration however, this issue is remedied. Although it is practical to use a long LMPC horizon during the learning process, it is sufficient to use a much shorter one on the new tracks. Moreover, it has been shown that the controller with shorter horizon outperforms the same controller without migrated SS.

Although the learning process may take considerable time, as the simulation results of section 6 show, a good performance can be achieved in the very first lap on any new track by using only the migrated trajectories. This allows to entirely skip the learning process on the new tracks. This is especially useful in situations when the track on which the vehicle has to run changes frequently or if the track is long. Learning a very long track from scratch would take tremendous time while the migration process can be performed very quickly. Note also that the learned SS can be migrated to non-recursive paths as well. This allows to use the power of LMPC for non-iterative tasks.

6. ACKNOWLEDGEMENT

This work was partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and the ÚNKP- 19-4 New National Excellence Program of the Ministry for Innovation and Technology. It was also supported by the research program titled "Exploring the Mathematical Foundations of Artificial Intelligence (2018-1.2.1-NKP-00008)".

REFERENCES

Bristow, D.A., Tharayil, M., and Alleyne, A.G. (2006). A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3), 96–114.

Brunner, M., Rosolia, U., Gonzales, J., and Borrelli, F. (2017). Repetitive learning model predictive control: An autonomous racing example. In *Proceedings of the Conference on Decision and Control (CDC)*, 2545–2550.

Gao, Y., Gray, A., Frasch, J.V., Lin, T., Tseng, E., Hedrick, J.K., and Borrelli, F. (2012). Spatial predictive control for agile semi-autonomous ground vehicles. In *Proceedings of the International Symposium on Advanced Vehicle Control*.

Ju, T., Schaefer, S., Warren, J.D., and Desbrun, M. (2005). A geometric construction of coordinates for convex polyhedra using polar duals. In *Symposium on Geometry Processing*, 181–186.

Kabzan, J., Hewing, L., Liniger, A., and Zeilinger, M.N. (2019). Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4), 3363–3370.

Kapania, N.R. and Gerdes, J.C. (2015). Path tracking of highly dynamic autonomous vehicle trajectories via iterative learning control. In *American Control Conference (ACC)*, 2753–2758.

Kuwata, Y., Teo, J., Fiore, G., Karaman, S., Frazzoli, E., and How, J.P. (2019). Real-Time Motion Planning With Applications to Autonomous Urban Driving. *IEEE Transactions on Control Systems Technology*, 17(5), 1105–1115.

Lam, D., Manzie, C., and Good, M. (2010). Model predictive contouring control. In *Proceedings of the Conference on Decision and Control (CDC)*, 6137–6142.

Liniger, A., Domahidi, A., and Morari, M. (2015a). Optimization based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 628–647.

Liniger, A., Domahidi, A., and Morari, M. (2015b). Optimization-based autonomous racing of 1: 43 scale rc cars. *Optimal Control Applications and Methods*, 36(5), 628–647.

Piazzi, A., Bianco, C.G.L., Bertozzi, M., Fascioli, A., and Broggi, A. (2002). Quintic G^2-splines for the iterative steering of vision-based autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 3(1), 27–36.

Rosolia, U. and Borrelli, F. (2017). Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7), 1883–1896.

Rosolia, U., Carvalho, A., and Borrelli, F. (2017). Autonomous racing using learning model predictive control. In *Proceedings of the American Control Conference (ACC)*, 5115–5120.

Warren, J., Schaefer, S., Hirani, A.N., and Desbrun, M. (2007). Barycentric coordinates for convex sets. *Advances in computational mathematics*, 27(3), 319–338.