Towards Event-Based MCTS for Autonomous Cars

Nicola Catenacci Volpi*, Yan Wu† and Dimitri Ognibene‡
* School of Computer Science, University of Hertfordshire, Hatfield, UK
E-mail: n.catenacci-volpi@herts.ac.uk
† Robotics Department, Institute for Infocomm Research, A*STAR, Singapore
E-mail: wuy@i2r.a-star.edu.sg
‡ ETIC, Universitat Pompeu Fabra, Barcelona, Spain
E-mail: dimitri.ognibene@upf.edu

ABSTRACT

Uncertainty in the behaviours of vehicles surrounding a self-driving car introduces substantial computational complexity in autonomous driving. In this study[1], a data-driven approach was used to extract probabilistic models of the behaviours of other cars and exploit them to support a driving system based on Monte Carlo Tree Search (MCTS). The model selection component of the architecture infers which models better explain the current behaviours of the other vehicles using maximum likelihood estimation for Bayesian model comparison. The inferred behaviours are then used for MCTS-based control to prevent rollouts on models that are not relevant in the current context. While the use of multiple models allows improved efficiency and higher flexibility, it also introduces identification-related issues, which were solved here using Bayesian machine learning. The results obtained from the performed simulations are presented comparing the proposed MCTS architecture when employing multiple models with a naive model of the other vehicles.

I. Introduction

In this study, we test a probabilistic model-based controller in a dynamic environment where stochastic evolution depends on a latent variable. This can be used to simplify multi-agent planning in contexts where the agent intentions are constant and correspond to the hidden variable values.

The task we used to test this model is autonomous driving, which requires controlling the vehicle in various environments and against different behaviours of the other agents. Multiple model-based architectures [4] offer the necessary flexibility for such variety of contexts.

To use a probabilistic approach in this context is important because a precise and deterministic model of the controlled car may not be available and, more importantly, because it allows to take into account for the uncertainty related to the actions of the surrounding vehicles. In this regard, a crucial challenge is to employ non-trivial predictive models. This requires to build or acquire these models, as well as managing them during control. In the present study, the predicted models were extracted from data using machine learning methods, or were identified directly whenever possible. The question of how to acquire models that enable safe and efficient interaction is left for future work. Usually, algorithms to learn models of behaviours of other agents are employed to directly train model-free systems that replicate such behaviours [13], or to indirectly train them to comply with other agents producing the same behaviours.

The main contribution of this study is to investigate the use of predictive models in a multiple model-based MCTS architecture, which selects those that are more suitable for prediction and control in the current context. This approach, by focusing on more relevant and likely conditions, enables more precise predictions and has computational advantages.

A mixture of linear Gaussian models is used to model the behaviours of other vehicles, each one implemented with a Kalman filters (KF) used to track online the corresponding behaviour of the other cars. The Upper Confidence bounds for Trees algorithm (UCT [11]) is then used to plan future actions using the predictions produced by the Kalman filters. A MCTS approach with limited horizon has been selected as it provides good performance even when limited knowledge of the system dynamics is available (e.g., only a black box simulator is available). Note that in each experiment the system was evaluated during a single event (i.e., only one model explains the observed dynamics).

A. Related works

MOSAIC [20], [8], [19] was one of the first architectures employing pairs of predictors-controllers for control. Each predictor-controller pair is optimised for a specific latent state of the environment. Then, the more precise a predictor is the higher influence the corresponding controller should have. However, the flexibility of this approach could be quite limited because of the use of reactive controllers. Moreover, the trivial combination of controllers outputs could result in suboptimal behaviour, especially when ambiguity persists over the predictors.

HAMMER [17], [3] is a hierarchical architecture of controllers-predictors for robot control and learning by imitation. It showed how such hierarchical structure could provide a robot with flexible learning from different sources of learning (social or autonomous exploration). It also introduces the idea

[1]This work was partially supported by EC Horizon 2020 programme under the project WiMUST (Grant agreement no: 645141, Strategic objective: H2020 - ICT-23-2014 - Robotics). The authors thank Aldo Fanelli for the precious support.

of actively controlling perception to support model predictions. Like MOSAIC, the effects of persistent ambiguity between the predictors-controller pairs have not been deeply investigated and were not considered in the employed active perception approach.

The AERIG system proposed in [15] also uses a mixture of Kalman filters to predict other agents' actions. In addition, to improve the identification of the model that better describes the environment, this method provides an information theoretic control of the robot cameras to gather relevant information (e.g., switching between observed effectors or locating new candidate action, target or obstacle).

In [2], a model-based approach was used to deal with interaction in social environments. In this work, the intentions of other agents are know or drawn from a uniform probability distribution, hence no estimation was performed. A similar method was reported in [1], where a discrete and static environment was considered (i.e., grid-world).

Galceran et al. [6], focus on the urban driving context. Similar to our approach, they use a Partially Observable Markov Decision Process (POMPD) formulation where recent observations are used to estimate the distribution of current behaviours of the other cars, encoded as closed-loop policies that react to the actions of the other vehicles. This approach employs low-level precomputed closed-loop policies to describe also the controlled car. The closed-loop policy to execute is selected by sampling forward simulations of all the nearby cars using the estimated distribution over their current behaviours. While this leads to computational advantages, it limits flexibility and may require a large number of low-level policies to account for the different structures of the road, other car states and the current goal.

II. Methods

The event-based MCTS algorithm for Markov Decision Processes (MDP) is based on the combination of a *Mixture of Kalman Filters* and a continuous state MDP. These two components are coupled in order to make MCTS plan for the events currently detected by the KFs. In the following sections, these components will be described. The overall structure of the system with the interaction between the different components is depicted in Figure 1.

A. Mixture of Kalman Filters

A mixture of KFs is composed of a set of Kalman filters $KF^1, KF^2, \ldots, KF^N$ such that $KF^v = (A^v, Q^v, H, R)$ for all $v = 1, \ldots, N$. Each KF is specialized in recognition of one *event* v characterised by a linear dynamic system (LDS). Each LDS is described by the following equations. Its state evolves according to $x^v_{t+1} = A^v x^v_t + w^v_{t+1}$ with $w^v_{t+1} \sim \mathcal{N}(0, Q^v)$, where A^v is the transition matrix and Q^v is the covariance matrix of event v. The observations o^v_{t+1} produced by each LDS are: $o^v_{t+1} = H x^v_{t+1} + \eta_{t+1}$ with $\eta_{t+1} \sim \mathcal{N}(0, R)$. We assume that all the KFs of the mixture have the same observation matrix H and observation noise covariance matrix R. In other words, events represent possible Gaussian linear transitions of the non-stationary elements of the state vector x_t (e.g. an object moving straight subjected to Gaussian noise).

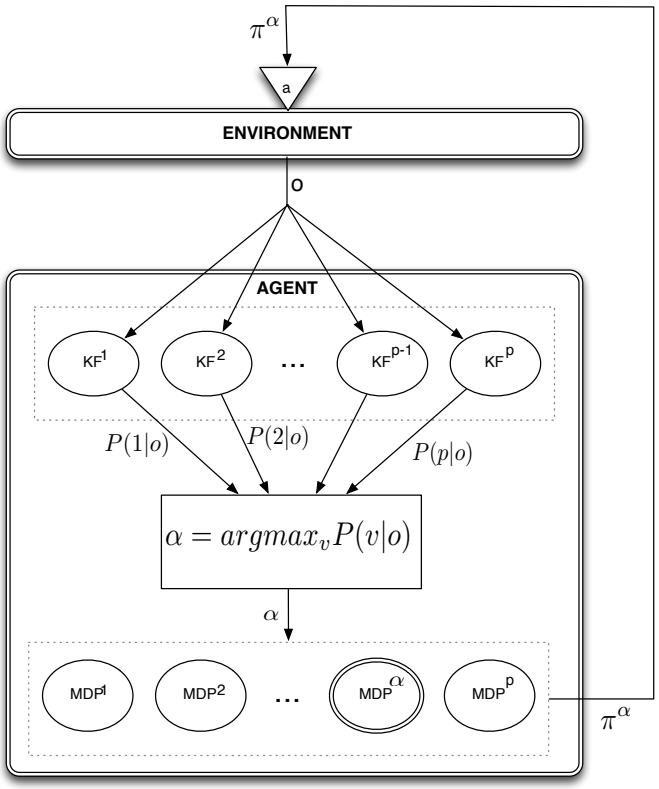

Fig. 1. Architecture overview.

Assuming that the event is constant [2]for the whole observation time T, if we have $o_{1:T}$, the posterior probability of each event can be recursively obtained using Bayes' Formula:

$$p(v|o_{1:t}) = \frac{p(o_t|v)p(v|o_{1:t-1})}{p(o_t|o_{1:t-1})}. \quad (1)$$

This provides a measure of current relevance of the event v at a certain time t.

Note that in eq. (1), the denominator is independent of the event and that $p(o_t|v)$ corresponds to the probability of the observation for the KF specialized in the estimation of the state of the event v

$$P(o_t|v) \sim \mathcal{N}(o_t; \hat{o}^v_t, \hat{H}^v_t) \quad (2)$$

where $\hat{o}^v_t = H^v \hat{x^v_t}$ is the predicted mean of the observation distribution of the event v and $\hat{H}^v_t = H\hat{\Sigma}^v_t H^T + R$ is the expected covariance of the observation for event v, with $\hat{\Sigma}^v_t$ the covariance of the state. Obtaining these quantities is not an additional computational burden being computed as part of the standard Kalman filter update step. The event-based MCTS algorithm finds the index of the most probable event at time t,

[2]Here we focus only on the detection of the first event. When the event can switch during the observation, the treatment becomes more complex. See [16], [14], [5]

$\alpha_t = argmax_v P(v|o_{1:t})$, doing Bayesian recursive estimation and Bayesian model comparison through maximum likelihood estimation [15]. As we will see, this information can be used to focus the online planning machinery on the most probable event α_t that is currently occurring in the environment.

B. Online Markov Decision Process Generation

To reduce the computational complexity of a real-time agent embedded in a continuous dynamical system, it is often necessary to decompose the decision-making process among several micro-behaviors [18]. Usually, only a small subset of possible future states and transitions are relevant in each context. Hence, in this work we generate online a continuous MDP using both the LDS and the estimated state corresponding to the KF α with the highest posterior probability. The MDP does not need to be represented explicitly. Only the states visited during the online decision process are generated and stored.

The state of the generated Continuous MDP corresponds to the tuple $\mathbf{S} = \{S_0, S_i, i = 1 : M\}$, where M is the number of cars, S_0 is the state of the car controlled by the agent and S_i is the state of the i-th car.

The transition probability distributions $\mathcal{T}_a^{S',S} = P_a(S^{t+1} = S'|S^t = S)$ can be factored in $\mathcal{T}_a^{S',S} = P_a(S_0^{t+1}|S_0^t)\prod_i^N P^\alpha(S_i^{t+1}|S_i^t)$ with $P_a(S_0^{t+1}|S_0^t)$ describing the controlled car dynamics and $P^\alpha(S_i^{t+1}|S_i^t)$ corresponding to the transition matrix of the car i for the winning KF α.

The reward function of the MDP is tuned to give punishment when the controlled car collides with another one and incentives when the vehicle is proceeding forward along the track.

The combination of a mixture of KFs and MCTS yields an online planner that computes only the control of the event α, disregarding events that are not relevant in the current context.

III. Experimental set-up and Results

A. Task and Model Validation

The proposed model was tested on a simulated task that concerns overtaking between two cars in a driving scenario. For the sake of simplicity, in the following experiments we assume that our self-driving car is interacting with only another car. To conduct the experiments, the Open Racing Car Simulator (TORCS) [21] [3] was used. TORCS is an open-source car simulator written in C++ widely used by the research community. It has a sophisticated physics engine that allows realistic simulation of the cars' dynamics with three-dimensional body physics, aerodynamics effects taken into account and noisy actuators deployed on the simulated cars.

In the following experiments, we show how the car controlled by the event-based MCTS can overtake another car through smooth trajectories and avoid going out of the track. To underline the ability of the proposed method to perform well in various contexts, two different behaviours of the other car are considered in the following sections: in *Scenario 1* the other car is moving at constant speed towards a certain heading direction; in *Scenario 2* the other car is decelerating. The number of successful overtakes over several trials is used as a measure of performance. In addition, how this quantity varies according to the number of rollouts used within the MCTS algorithm is considered in order to evaluate the computational complexity of the proposed method.

Fig. 2. A snapshot of the TORCS simulator.

To validate our model, a comparison with a simple overtaking car model is investigated. This model, instead of considering a mixture of Kalman filters, represents the dynamics of the other car with a simple bi-dimensional transition function. This function updates the car position $P_t = (p_t^x, p_t^y)$ using a kinematic equation with no acceleration, and assumes that the other car keeps the last observed speed $V_t = (v_t^x, v_t^y)$ plus a Gaussian noise term $\eta \sim \mathcal{N}(0, \Sigma)$:

$$\begin{aligned} P_{t+1} &= P_t + \Delta V_t \\ V_{t+1} &= V_t + \eta \end{aligned} \tag{3}$$

Where Δ is the time step of the difference equation. In order to perform a successful overtake with this model, the variance of this Gaussian random variable has to be large enough to capture all the possible dynamics of the other car (e.g., unexpected change of acceleration). In the performed simulations the covariance matrix of this distribution was set as $\Sigma = diag(0.1, 1.1)$.

Although the other car moves according to only one of the aforementioned two behaviours during one experiment, it is important to underline that the controlled car will have no direct knowledge about which of these behaviours will be chosen. Hence both the event-based MCTS controller and the simple overtaking car model will have to perform well in both scenarios.The event-based MCTS controller will do so by picking up the proper Kalman filter from the mixture and then plan according to the corresponding other car's transition model, whereas the simple overtaking car model has to be

[3] http://torcs.sourceforge.net/index.php

defined with a large enough speed variance to take into account both dynamics.

B. Experimental Set-Up

1) Kalman Filters and LDS learning: In the present work events are defined as given subsets of potential dynamics of the other car. Considering the two different scenarios described in the previous section, two different ways of obtaining the corresponding Kalman filters' noisy transition matrices A^v and Q^v were considered. Then, all these families of Kalman filters will be added to the mixture of the event-based MCTS controller.

In Scenario 1, a simple behaviour of the other car is considered, hence a proper set of matrices could be identified directly. In this case, the other car moving at constant speed towards a certain heading direction is modelled with several Kalman filters, each one used to detect different combination of time step Δ^v (used to represent the current speed with the proper time resolution) and heading direction T^v. The state vector of the Kalman filter v is $X^v = [P^v, V^v, T^v]$, where $P^v = (P^v_x, P^v_y)$ represents the observed car's position, $V^v = (V^v_x, V^v_y)$ represents the latent variable of car's velocity and $T^v = (T^v_x, T^v_y)$ is the observed heading direction. To this aim, the matrix A^v is defined in order to represent the following transition function:

$$\begin{aligned} P^v_{t+1} &= P^v_t + \Delta^v V^v_t \\ V^v_{t+1} &= \Delta^v (T^v_t - P^v_t) \\ T^v_{t+1} &= T^v_t + \Delta^v V^v_t \end{aligned} \tag{4}$$

At the beginning of the simulation, the variables T^v are initialized with the coordinates of points that lie towards the corresponding heading direction and the distance between these coordinates and the car position is kept constant during an event. In the experiments investigated in this work, 50 possible time steps Δ^v and 32 possible heading directions T^v were considered for a total of 1600 events representing the other car moving at a constant speed. The covariance matrices used for all these events were chosen to be $Q^v = diag(0.01, 0.01, 0.008, 0.008, 0.5, 0.5)$.

In addition, a Kalman filter used to recognize a stationary car was added to the mixture. Hence, the transition matrix used to model this car was defined as follows $A^v = diag(P^v_x, P^v_y, 0, 0)$ (note that no heading direction is considered here). The covariance matrix adopted in this Kalman filter is $Q^v = diag(0.001, 0.001, 0.001, 0.001)$.

Regarding Scenario 2, more complex dynamics of the other car is considered. In this case, it would be more complicated to identify directly what stochastic linear dynamical system should be used in the Kalman filters to detect those dynamics. Hence, in this case, a data-driven approach based on Bayesian machine learning was adopted. The utilized methodology uses the MATLAB toolbox of Kevin Murphy [4] to find the maximum likelihood parameters for the stochastic linear dynamical systems of the Kalman filters of the mixture. Instead of representing the state variable as a deterministic value with added normally distributed noise, the state and noise variables are combined into a single Gaussian random variable. Such linear Gaussian models can be used both for supervised and unsupervised modelling of time series. In this work, since input control variables are not considered, the problem can be seen as an unsupervised one. The goal is to model the unconditional density of the observations. These estimates were produced using the Expectation Maximization algorithm [7] starting from a given data set of trajectories that were produced using the TORCS simulator. It is important to note that Kalman filters are the continuous state counterparts of Hidden Markov models, which can be used to infer the posterior probabilities of the state given the observation sequence (i.e., smoothing). These posterior probabilities are the same as used within the E-step of the adopted Expectation Maximization algorithm.

Kalman filters can deal only with linear dynamical systems, but the trajectory of a car is usually non-linear. In this regard, to simplify the the treatment of the events used in this work, we decided to represent the positions of the cars using a coordinate reference system that is moving along the center of the road at constant velocity. In doing so the non-linearity given by the curvature of the road was easily avoided. This is why in the following section we will consider "longitudinal" and "lateral" positions and velocities instead of the absolute ones.

2) Planning: The algorithm used to control the self-driving car within the generated MDP is a continuous version of the UCT algorithm. This algorithm was already applied to real autonomous driving in [9]. UCT was chosen due to its efficiency, its online and anytime nature. The UCT exploration rate parameters used in the performed simulations was $C_p = 0.05\frac{1}{\sqrt{2}}$, whereas the length of a rollout was set to 65.

The definition of the tuple (S, A, P, R) representing the Markov Decision Process used to control the self-driving car will be provided as follows. Note that the transition probability distribution regarding the dynamics of the other car will be generated online starting from the most probable stochastic linear dynamical system α contained in the mixture of Kalman filters.

- The *state space* is $S = [x, y, v_x, v_y, d_x, d_y, d_{v_x}, d_{v_y}]$, where $x, y \in \mathbb{R}$ are the longitudinal and lateral offsets from the road center of our self-driving vehicle; $v_x, v_y \in \mathbb{R}$ are respectively the lateral and longitudinal velocity of our self-driving car. $d_x, d_y \in \mathbb{R}$ are respectively the lateral and longitudinal distances of the other car from our self-driving vehicle; $d_{v_x}, d_{v_y} \in \mathbb{R}$ are respectively the lateral and longitudinal velocities of the other car.
- The *action space* is $A = [steer, brake\_accel, stay]$, where $steer \in \mathbb{Z}$ represents the steering control of our self-driving system and $brake\_accel \in \mathbb{Z}$ represents its braking control when it assumes negative values and its acceleration control when it assumes positive values. Finally, $stay$ represents an idle action, indicating null steer and brake controls.
- The *transitional probability distribution* is

[4] https://www.cs.ubc.ca/ murphyk/Software/

$P^{\alpha}_{\{steer,brake\_accel,stay\}}(s,s')$. On one side, we used motor bobbling within the TORCS simulator to learn the transition probabilities $P_a(S_0^{t+1}|S_0^t)$ about the control of our self-driving vehicle; on the other side, we extracted the dynamics $P^{\alpha}(S_i^{t+1}|S_i^t)$ of the other vehicle's trajectory using the LDS of the most probable event α.

- The *reward function* is $R_{\{steer,brake\_accel,stay\}}(s,s') = C_1 + C_2 + C_3 + C_4$, where C_1 is a positive reward representing the will of our self-driving system to advance towards the road; C_2 is another negative reward used to incentivise the car to maintain a straight trajectory; C_3 is a reward used to maintain the autonomous car in the center of the road as much as possible; C_4 is a negative reward representing the necessity to avoid collisions with the other car. Note that the magnitudes of C_3 and C_4 have to be larger than the magnitudes of C_1 and C_2 because it is more important to avoid dangers than to continue driving.
  Let us see the definitions of these parameters in more details:

$$C_1 = c_1 \mathbf{v} \cdot \mathbf{v}_{\rightarrow} \in \mathbb{R}^+ \tag{5}$$

  is the scalar product between the velocity of our self-driving vehicle $\mathbf{v}$ and the unit vector tangent to the road direction $\mathbf{v}_{\rightarrow}$.

$$C_2 = -c_2(|\mathbf{v} \cdot \mathbf{v}_{\uparrow}| + |\mathbf{v} \cdot \mathbf{v}_{\downarrow}|) \in \mathbb{R}^- \tag{6}$$

  where the two scalar products here are computed between the velocity of our self-driving vehicle $\mathbf{v}$ and the unit vectors perpendicular to tangent to the road direction $\mathbf{v}_{\uparrow}$ and $\mathbf{v}_{\downarrow}$.

$$C_3 = c3\mathcal{N}(road\_center, \delta)(y) \in \mathbb{R} \tag{7}$$

  where $\mathcal{N}(road\_center, \delta)$ is a normal distribution with its mean equals to the longitudinal coordinate of the road center, variance δ and argument equals to the longitudinal position y of our car.

$$C_4 = \begin{cases} c_4 \in \mathbb{R}^- & \text{if } d_x < \epsilon \wedge d_y < \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

  is used to penalize the self-driving car when its position is at a distance less than ϵ with the other car. The value chosen for this distance in the reported experiments is $\epsilon = 6\ m$.

Since only a generative model of the MDP is needed, to plan online using UCT we can simplify the representation of the MDP as follows. To this aim, we assume that the control of our car does not depend on its position. In other words, given the same actions, the self-driving car behaves in the same way in all the positions of the road. This makes possible to have a compact generative model that does not consider the positions as input. In addition, the generative model can output only the displacements instead of the absolute positions, and only the displaced velocity components instead of the velocity's absolute values . Although only these displacements are stored in the generative model, when one step of the forward model is executed, the actual position and velocity of the car (necessary to compute the reward function) can be obtained simply by adding the displacements to the previous original positions and velocities.

Moreover, within the MDP we assume that the control of our car does not depend on its heading direction of motion. In other words, given the actions, the car behaves in the same way in all the directions (e.g., north, south, west, east). This makes possible to represent the coordinates stored in the generative model using a local system of reference centred in the car position instead of an absolute one. The advantage of using this representation is to discard the two components of the velocity vector as input of the generative model and to substitute them with the magnitude of the velocity only, hence leading to a more compact representation. When one step of the forward model is executed, the original absolute positions and velocities of the car can be reconstructed using trigonometry and rotation matrices.

Finally, the generative model concerning the control of our self-driving vehicle is defined as $G : |\mathbf{v_{loc}}| \times steer \times brake\_accel \mapsto \delta x_{loc} \times \delta y_{loc} \times \delta v_{x_{loc}} \times \delta v_{y_{loc}} \times [0,1]$. That is, for a given velocity magnitude and input controls the generative model's output is composed by the displacements of position and velocity (in the local system or reference) with the associated probability.

C. Results

In this section, the results obtained by the event-based MTCS controller and the simple overtaking car model in the two aforementioned scenarios are reported.

In the Scenario 1, we tested the performance of the two models in overtaking the other car moving at constant speed towards a given heading direction. In the experiment reported here, the other car was approaching the controlled car at a constant high speed against the flow. Both the simple overtaking car model and the event-based MCTS controller were able to successfully overtake the other car but, as it is possible to see from Figure 3, each one with a different computational cost.

In Figure 3, the ratios of successful collision avoidances during a set of 20 overtaking manoeuvres are reported for the event-based MCTS controller (blue line) and the simple overtaking car model (red line). In addition, such ratios are compared with the number of rollouts that UCT used to achieve such performance. As it is possible to see in this scenario, only 300 rollouts sufficed for the event-based MCTS controller in order to avoid the other car for most of the investigated trials, whereas a successful overtake was performed 100% of the times when 600 rollouts are utilized by the algorithm. Note that in this case the other car's trajectories were modelled using the Kalman filter defined in (4). In the

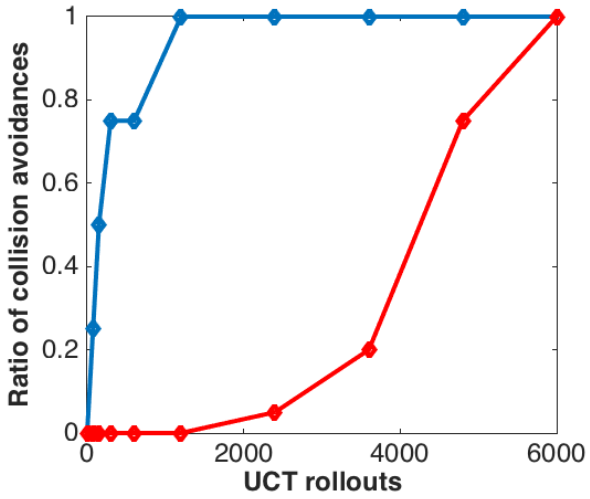

Fig. 3. Success ratio of collision avoidance of the event-based MCTS controller (blue line) and the simple overtaking car model (red line) in the constant speed scenario.

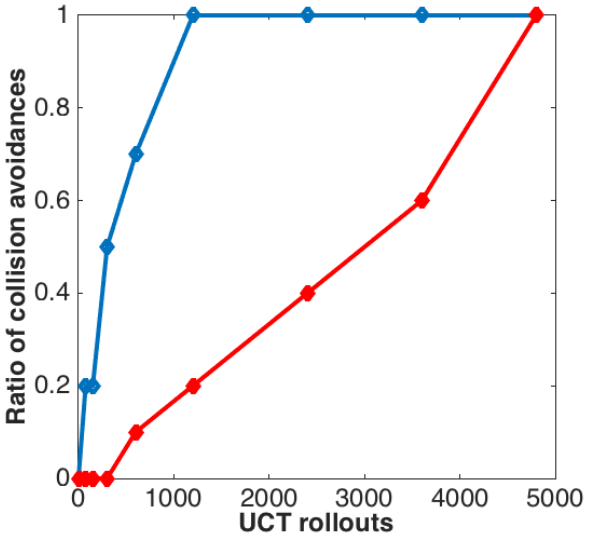

Fig. 4. Success ratio of collision avoidance of the event-based MCTS controller (blue line) and the simple overtaking car model (red line) in the decelerating scenario.

case of the simple overtaking car model for most of the trials, 4800 rollouts were necessary to make UCT find the proper sequence of controls that led to collision avoidance, whereas with 6000 rollouts the algorithm always produced successful overtaking manoeuvres.

In the following scenario, the other car followed trajectories with a varying acceleration. In the experiment reported here, the other car decelerated with a non-uniform acceleration in front of the controlled car. In Figure 4, the results of this experiment are reported. Also in this case, both the event-based MCTS controller and the simple overtaking model are able to perform successful overtaking manoeuvres. The

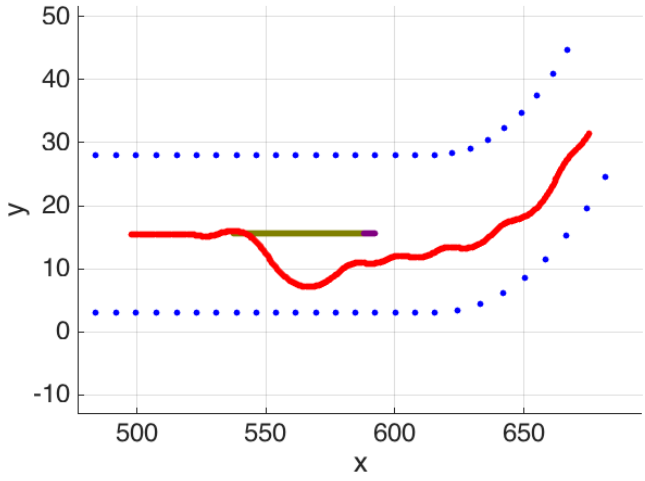

Fig. 5. Trajectory of the self-driving car controlled by the event-based MCTS (red line) and the other car's trajectory (green line) in the decelerating scenario.

first controller needed at least 600 rollouts to avoid most of the possible collisions, whereas with 1200 rollouts collision avoidance was attained for all the trials. In this setup, the used Kalman filters were obtained using the data-driven approach with maximum likelihood estimation of the linear dynamical system parameters. For the simple overtaking car model, a number of 4800 rollouts were necessary to achieve a similar performance. To show the smoothness and the alignment of the autonomous car's trajectories relative to the center of the road, we reported in Figure 5 the resulting trajectory of the controls found by the event-based controller in the decelerating scenario with 600 rollouts. Here the red line represents the self-driving car trajectory and the green line represents the trajectory of the other car, which decreases its speed until stopped. It is possible to observe that the autonomous car avoided the other car when it approached the other car and then proceeded its route.

Comparing the success ratios of the event-based MCTS controller and the simple overtaking car model reported in Figures 3 and 4, we note that the first model requires a considerably smaller number of rollouts to perform successful collision avoidances. Indeed, with this simple model there is no partition of the other car dynamics and so there is no corresponding transition matrix factorization that can be exploited. This implies that the represented trajectories of the other car are less structured when compared with the event-based MCTS model and consequently they can be predicted less precisely. Hence, although this model is rich enough to make the controlled car successfully overtake the other car, this is attained with a larger computational cost given by the smaller sparsity of the represented trajectories and a corresponding larger areas of the road to avoid, when compared with the event-based MCTS method.

A more concrete comparison between the event-based MCTS algorithm and classic MCTS in the simple overtaking approach can be made reporting the processing time of the

two methods. The time spent by these two algorithms at each simulation loop of the TORCS driving simulator can be obtained from the duration of a rollout in MCTS (that includes the update of all the counters and the computation of the UCB1 function), which is on average 0.78 milliseconds [5]. In addition, at each simulation loop the time spent by the event-based MCTS algorithm in updating the Kalman filters and the estimates of the corresponding maximum likelihood estimation is 0.2 milliseconds on average. Hence, in the constant speed scenario the event-based algorithm took 0.47 seconds during a simulation loop to be successful in all the investigated overtaking trials, where the simple overtaking car took 4.68 seconds. In the decelerating scenario for each simulation loop the event-based MCTS algorithm took 0.94 seconds to be successful in all the performed overtaking tests, where the simple overtaking car took 3.74 seconds. Let us underline that the usage of the mixture of Kalman filters of the event-based approach introduces only an overhead of 0.2 milliseconds in order to spend a significant smaller number of rollouts, and consequently less seconds, to perform successfully the task.

IV. Conclusions

The design of self-driving vehicles is a hot topic today in Robotics and Artificial Intelligence. Although many progresses have be done in the last years, this area of research still poses big challenges to the scientific community. One of the main difficulties comes from the large number of dynamic obstacles that a vehicle is facing in a typical driving environment. One example is represented by the dynamics of other cars whose trajectories are usually difficult to predict.

One possible solution, that we proposed in this paper, is to decompose the original problem in many simpler sub-tasks in order to plan only according to the sub-tasks that are relevant in the current context. Indeed, instead of computing the vehicle's control according to all the possible car's dynamics, usually only a subset of decisions are necessary in each possible context.

In this work, a mixture of Kalman filters is used to represent the dynamics of other cars in order to specialize the control of an autonomous vehicle towards the trajectories that are currently detected online. The performance of the proposed approach was tested in simulated overtaking scenarios where a MCTS controller was used to control the self-driving car. We showed that, when compared with a simple overtaking model, our event-based MCTS algorithm required less rollouts to perform successful collision avoidances of another moving car. In fact, the factored transition matrix resulting from our event-based approach implied increased computational efficiency and precision of control.

The proposed approach could be considered as a greedy alternative to game-theory when strict real-time constraints are imposed by the task [12]. It could be extended to tune the system to match the correct control time-scale with the available computational power and the necessary flexibility. In future, we would like to propose the utilization of a "safe policy" to avoid unexpected costs when the uncertainty about which event is going to occur is too high. In addition, although in this paper we only used a probabilistic model based controller, we aim to combine model-based and model-free control as our next step. The integration of AERIG [15] with the event-based MCTS could reduce sensor processing costs. It appears relatively simple given the similarities between the two frameworks. However, how the MCTS control system will react to missing data [10] is an open problem that must be addressed.

[5]The reported simulations were performed on a computer equipped with a 2.5Ghz i5 processor.

References

[1] Tirthankar Bandyopadhyay, Kok Sung Won, Emilio Frazzoli, David Hsu, Wee Sun Lee, and Daniela Rus. Intention-aware motion planning. In *Algorithmic Foundations of Robotics X*, pages 475–491. Springer, 2013.

[2] Adrian Broadhurst, Simon Baker, and Takeo Kanade. Monte carlo road safety reasoning. In *Intelligent Vehicles Symposium, 2005. Proceedings. IEEE*, pages 319–324. IEEE, 2005.

[3] Y. Demiris and B. Khadhouri. Hierarchical attentive multiple models for execution and recognition of actions. *Robotics and Autonomous Systems*, 54(5):361–369, 2006.

[4] Kenji Doya, Kazuyuki Samejima, Ken ichi Katagiri, and Mitsuo Kawato. Multiple model-based reinforcement learning. *Neural Comput*, 14(6):1347–1369, Jun 2002.

[5] Emily B Fox, Erik B Sudderth, Michael I Jordan, and Alan S Willsky. Nonparametric bayesian learning of switching linear dynamical systems. In *NIPS*, pages 457–464, 2008.

[6] Enric Galceran, Alexander G Cunningham, Ryan M Eustice, and Edwin Olson. Multipolicy decision-making for autonomous driving via changepoint-based behavior prediction. In *Robotics: Science and Systems*, 2015.

[7] Zoubin Ghahramani and Geoffrey E Hinton. Parameter estimation for linear dynamical systems. Technical report, Technical Report CRG-TR-96-2, University of Totronto, Dept. of Computer Science, 1996.

[8] M. Haruno, D.M. Wolpert, and M. Kawato. Mosaic model for sensorimotor learning and control. *Neural Computation*, 13(10):2201–2220, 2001.

[9] Todd Hester. *TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains*, volume 503. Springer, 2013.

[10] Seliz G Karadogan, Letizia Marchegiani, Lars Kai Hansen, and Jan Larsen. How efficient is estimation with missing data? In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 2260–2263. IEEE, 2011.

[11] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 6, pages 282–293. Springer, 2006.

[12] Nicolas Le Guillarme, Abdel-Illah Mouaddib, Sylvain Gatepaille, and Amandine Bellenger. Adversarial intention recognition as inverse game-theoretic planning for threat assessment. In *Tools with Artificial Intelligence (ICTAI), 2016 IEEE 28th International Conference on*, pages 698–705. IEEE, 2016.

[13] Guilherme J Maeda, Gerhard Neumann, Marco Ewerton, Rudolf Lioutikov, Oliver Kroemer, and Jan Peters. Probabilistic movement primitives for coordination of multiple human–robot collaborative tasks. *Autonomous Robots*, 41(3):593–612, 2017.

[14] Kevin P Murphy. Switching kalman filters. Technical report, Citeseer, 1998.

[15] Dimitri Ognibene and Yiannis Demiris. Towards active event recognition. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2495–2501. AAAI Press, 2013.

[16] Vladimir Pavlovic, James M Rehg, and John MacCormick. Learning switching linear models of human motion. In *NIPS*, pages 981–987, 2000.

[17] Miguel Sarabia, Raquel Ros, and Yiannis Demiris. Towards an open-source social middleware for humanoid robots. In *Proceedings of the IEEE/RAS International Conference on Humanoid Robotics*, 2011.

[18] Nathan Sprague, Dana Ballard, and Al Robinson. Modeling embodied visual behaviors. *ACM Trans. Appl. Percept.*, 4(2):11, 2007.

[19] D. Wolpert and M. Kawato. Multiple paired forward and inverse models for motor control. *Neural Netw*, 11(7-8):1317–1329, Oct 1998.
[20] Daniel M. Wolpert, Kenji Doya, and Mitsuo Kawato. A unifying computational framework for motor control and social interaction. *Phil. Trans. R. Soc. Lond. B*, 358:593602, 2003.
[21] Bernhard Wymann, Eric Espié, Christophe Guionneau, Christos Dimitrakakis, Rémi Coulom, and Andrew Sumner. Torcs, the open racing car simulator. *Software available at http://torcs. sourceforge. net*, 2000.