Sampling Algorithms Combination with Machine Learning for Efficient Safe Trajectory Planning

Amit Chaulwar, Hussein Al-Hashimi, Michael Botsch, and Wolfgang Utschick

Abstract—**The planning of safe trajectories in critical traffic scenarios using model-based algorithms is a very computationally intensive task. Recently proposed algorithms, namely Hybrid Augmented CL-RRT, Hybrid Augmented CL-RRT+ and GATE-ARRT+, reduce the computation time for safe trajectory planning drastically using a combination of a deep learning algorithm 3D-ConvNet with a vehicle dynamic model. An efficient embedded implementation of these algorithms is required as the vehicle on-board micro-controller resources are limited. This work proposes methodologies for replacing the computationally intensive modules of these trajectory planning algorithms using different efficient machine learning and analytical methods. The required computational resources are measured by downloading and running the algorithms on various hardware platforms. The results show significant reduction in computational resources and the potential of proposed algorithms to run in real time. Also, alternative architectures for 3D-ConvNet are presented for further reduction of required computational resources.**

Index Terms—**Safe trajectory planning, hybrid machine learning, collision avoidance and mitigation.**

I. Introduction

Active safety systems for collision avoidance have a huge potential for increasing the road-traffic safety and are necessary components of vehicles for autonomous driving. Since the driverless car competitions DARPA Grand Challenge [1] in 2005 and DARPA Urban Challenge [2] in 2007, automotive industry is broadly engaged in the development of advanced driver assistance systems having varying degrees of autonomous driving. Autonomous driving comprises of many tasks such as perception of the environment, localization and mapping, motion planning, vehicle control, etc. One of the most difficult problems for realizing fully autonomous driving is the safe trajectory planning in complex, critical traffic-scenarios with multiple static and dynamic objects with the simultaneous intervention in both the lateral and longitudinal dynamics of the vehicle. Such trajectory planning algorithms need to be efficient in terms of required computational resources, both time and memory, as they are limited by the on-board vehicle micro-controllers.

Manuscript received on September 2, 2019; revised August 13, 2020. This work was funded by the German Federal Ministry of Education and Research (BMBF) under grant number 03FH034PX4.

Amit Chaulwar, Hussein Al-Hashimi, and Michael Botsch are with the Faculty of Electrical Enginering, Ingolstadt University of Applied Sciences, Ingolstadt, Germany (e-mail: firstname.lastname@thi.de).

Wolfgang Utschick is with the Department of Electrical Engineering, Technical University of Munich, Munich, Germany (e-mail: utschick@tum.de).

Trajectory planning is a already very widely researched area in the robotics community. Many deterministic [3]-[5] and probabilistic trajectory planning approaches [6], [7] have already been proposed and a survey can be found in [8]. A probabilistic sampling algorithm called *Rapidly-exploring Random Tree* (RRT) algorithm [7] is very popular because of its fast runtimes and ability for planning the trajectory with nonholonomic constraints of the vehicle without discretizing the state-space. Several variants of RRT algorithms have been proposed but only few variants of the RRT algorithm [9], [10] claim to find a safe trajectory with simultaneous intervention in the lateral and the longitudinal dynamics of the vehicle in real-time. CL-RRT [9] was the motion planning algorithm for Team MIT at the 2007 DARPA Urban challenge and due to high computational resources requirements it was executed on a high-end 'Car-PC'. Also the algorithm in [10] needs to precompute and store many safe states in order to assure real-time capability.

The computation time for an RRT algorithm increases with the complexity of a scenario, i. e., with the increase in the number of objects and constraints arising due the road infrastructure. The RRT algorithm is probabilistically complete. It means that it will find a solution, if it exists, given a sufficient computation time. The key for increasing the convergence speed is to find an appropriate heuristic based on the application. There is a long history of using heuristics to refine the RRT search [11]-[17]. All of these algorithms use some heuristic for the biased-sampling to increase the convergence speed of finding an optimal solution, but they all need to find at least an initial approximate solution before they can use the biased-sampling.

Machine learning algorithms especially Deep Neural Networks (DNN) can be used to find the solutions for complex problems with short inference time. Since they are purely data-based methods they are seen as black-box methods. Therefore, they are not used in safety critical applications like vehicle trajectory planning. A learned Gaussian Mixture Models distribution is used for the biased-sampling in learned free spaces in [18] to decrease the number of collision checks drastically for the trajectory planning with the RRT algorithm. Biased sampling in free spaces might increase the probability of finding collision-free states, but it is not necessarily a good sampling strategy for the long term trajectory planning which requires a long chain of collision-free states. In an another approach [19], a conditional variational autoencoder is used to generate biased samples in space from a learned sampling distribution on the specific planning problem to increase the convergence rate of the RRT algorithm. However, the approach for simultaneous biased-sampling in the lateral and longitudinal dynamics of the vehicle, is missing.

The use of hybrid machine learning algorithms, a combination of machine learning algorithms and model-based search algorithms, opens a new way of using machine learning algorithms in safety critical applications. AlphaGo [20] and ExIT [21] are two examples of guided tree search algorithms with neural networks for the board games Go and Hex, respectively. But, these algorithms are limited to discrete state-spaces and action-spaces. The Hybrid Augmented CL-RRT (HARRT) [22] and the Hybrid Augmented CL-RRT+ (HARRT+) [23] and GATE-ARRT+ [24] are examples of hybrid machine learning algorithms for long term safe trajectory planning in complex, critical traffic scenarios which use 3D convolutional neural networks (3D-ConvNets) [25], a type of DNN, to assist RRT variants the Augmented CL-RRT (ARRT) [26] and the Augmented CL-RRT+ (ARRT+) [23] to reduce the computation time. These algorithms are described briefly in Section III.

The high accuracy of DNNs comes at the cost of high computational complexity and therefore computational engines such as GPU, ASIC, FPGA are used for applications with DNN implementation. Therefore, even if the hybrid machine learning algorithms proposed in [22], [23] and [24] reduce the time required by the model-based search algorithm, the total time and memory requirements are still high for the implementation on an automotive micro-controller. In this work the required computational resources for the implementation of the HARRT algorithm are evaluated on various hardware platforms. Additionally, the modules of the HARRT and HARRT+ algorithm requiring high computation time and memory are identified and alternative machine learning and model-based algorithms for an efficient embedded-implementation are proposed. The results obtained with these changes that lead to embedded-implementation of the HARRT algorithm are summarized in Section VII. The same methods can be used with the HARRT+ and GATE-ARRT+ algorithm to reduce the required computational resources.

3D-ConvNet is chosen as machine learning algorithm for learning spatiotemporal features. This algorithm is very computationally expensive because of 3D convolution and 3D pooling operations. Therefore, alternative architectures suitable for learning spatiotemporal features to replace 3D-ConvNet are presented which also required less computational resources.

This paper is organised as follows: Section II describes the functional and embedded implementation requirements for vehicle trajectory planning in critical traffic-scenarios, having the goal of collision avoidance/mitigation. Section III illustrates the model-based trajectory planning algorithms ARRT and ARRT+ and their corresponding hybrid machine learning trajectory planning algorithms HARRT, HARRT+ and GATE-ARRT+. Section IV and V introduce different machine learning and analytical methods for the optimization of the computation time and memory, respectively. These approaches are exemplarily explained with the HARRT algorithm but they are applicable to HARRT+ and GATE-ARRT+ algorithms as well. Section VI presents different alternative architectures for learning spatiotemporal features. Finally, Section VII presents the results of the measurements of computational resources required in different hardware platforms along with the comparison of alternative architectures.

Throughout this work, matrices are denoted by upper case bold letters, and vectors are denoted by lower case bold letters.

II. Vehicle Trajectory Planning Requirements

In this Section, the requirements for the safe trajectory planning in critical dynamic traffic-scenarios are formulated. These requirements are divided into functional and embedded implementation requirements.

Functional requirements describe what a system should do with regard to the desired functionality. The functional requirements for trajectory planning in vehicle safety applications are as follows:

• **Criticality estimation of the traffic-scenario:** In a first step, a traffic situation must be categorized as “critical" for the EGO-vehicle. The EGO-vehicle is the vehicle in which the trajectory planning algorithm is running.

• **Computation of collision-free trajectories:** In the second step, collision-free trajectories for the EGO vehicle must be computed. Collision-free trajectories are the path followed by a vehicle as a function of time in the time interval (e. g. $[t_0, t_0 + \tau]$) without a collision. EGO vehicle can follow such trajectory with a simultaneous intervention in both the lateral and longitudinal dynamics. This is because the trajectories resulting alone from full braking, as the ones that can be realized by an autonomous emergency braking system, are not enough to avoid collisions in many traffic scenarios.

• **Prediction of the severity of the injury:** For unavoidable collisions or when a collision-free trajectory is not found by search algorithms, a trajectory with low severity of collision must be computed and followed for the collision mitigation.

• **Selection of best trajectory**: The best trajectory should be selected from all the computed trajectories based on the criteria that reflects the aspects safety (e. g. collision-free), severity of injury (e. g. low collision speed), comfort (low accelerations during the maneuver), etc. The overall safety algorithm should also include a strategy when no trajectory that is collision-free or with a nonsevere collision is found.

These functional requirements are fullfilled with the analytical search algorithms ARRT and ARRT+.

The embedded implementation requirements describe the properties the algorithms should have to implement them in vehicles for safety applications. They are summarized below:

• **Efficiency:** The trajectory planning algorithms should be efficient so that they should run in real time on an automotive micro-controller.

• **Interpretability:** The requirement of interpretability arises, especially when machine learning methods are used in safety-critical applications.

• **No dynamic memory allocation:** For embedded applications especially safety critical applications like trajectory planning in complex traffic scenarios, a dynamic memory allocation should not be allowed [27], because many coding errors stem from mishandling of memory allocation like attempting to allocate more memory than physically available, forgetting to free memory or continuing to use memory after it was free, etc.

III. Hybrid Machine Learning Algorithms for Vehicle Trajectory Planning

In this section, the analytical trajectory planning algorithms the Augmented CL-RRT (ARRT) and Augmented CL-RRT+ (ARRT+) and their combinations with machine learning algorithms for faster convergence are described.

A. *Criteria for Traffic-Scenario Criticality*

A traffic scenario is considered to be critical for the EGO-vehicle at time instance t_0 if a collision will occur in the next τ' seconds given the assumption that the EGO-vehicle and other dynamic objects continue to follow the lane and pedestrians travel linearly with their velocity at time t_0 in the prediction interval $[t_0, t_0 + \tau']$. The trajectory planning algorithm only runs when a critical scenario is detected. The value of τ' is taken as 2 seconds in this work.

B. *Constructing a Tree $\mathcal{T}$*

The ARRT and ARRT+ algorithm find safe trajectories in critical traffic-scenarios with a simultaneous intervention in the lateral and longitudinal dynamics of the vehicle. They consider vehicle nonlinear dynamics for trajectory planning in the form

$$\dot{\boldsymbol{s}}(t) = \boldsymbol{f}(\boldsymbol{s}(t), \boldsymbol{u}(t)), \tag{1}$$

where $\boldsymbol{u}(t) \in \mathbb{R}^m$ is the control input and $\boldsymbol{s}(t)$ is the area occupied by the EGO vehicle at time t which is the subspace of $\mathbb{R}^2$. In an iterative process, these algorithms construct a tree $\mathcal{T}$ with multiple safe states $\boldsymbol{s}(t)$. Throughout this paper, the term *safe* used in context of states and trajectories means either collision-free or with the predicted nonsevere collision. In every iteration, a random point s_{rand} is sampled with some bias towards a goal region S_{goal}. The state $\boldsymbol{s}_{nearest}(t)$ which are previously stored in the tree $\mathcal{T}$ nearest to s_{rand} is found. The tree is extended by an incremental motion towards s_{rand} from $\boldsymbol{s}_{nearest}(t)$. The incremental extension is performed for the time interval Δt using differential constraints $\boldsymbol{f}$ as in Eq. (1) to get the new state $\boldsymbol{s}_{new}(t+\Delta t)$. The inputs $\boldsymbol{u}(t)$ here define inputs for both the lateral and longitudinal dynamics intervention. The two-track model is used as a constraint $\boldsymbol{f}$ while extending the tree, therefore the control input is defined as

$$\boldsymbol{u}(t) = [\delta_{fl}, \delta_{fr}, \delta_{rl}, \delta_{rr}, s_{l,fl}, s_{l,fr}, s_{l,rl}, s_{l,rr}], \tag{2}$$

where $\delta_{fl}, \delta_{fr}, \delta_{rl}, \delta_{rr}$ are the four angles of the wheels with respect to the longitudinal axis of the vehicle, and $s_{l,fl}, s_{l,fr}, s_{l,rl}, s_{l,rr}$ are longitudinal slip values of the vehicle tires. The letters in subscript stand for "front-left", "front-right", "rear-left", and "rear-right".

The controller calculates the steering wheel angle values based on the sampled point s_{rand}. The difference is in the calculation of longitudinal slip values. In case of the ARRT algorithm, different N predefined longitudinal acceleration profiles $\boldsymbol{a}_x^{\,n}$, where $n = 1, \ldots, N$, are used sequentially while for the ARRT+ algorithm the longitudinal acceleration $a_{x,rand}$ is randomly sampled with the consideration of the actuator and realistic-profile constraints. Here, the actuator constraints means only sampling acceleration values within the acceleration and jerk limits of actuators. The realistic-profile constraints are to get a smooth acceleration profile and avoid acceleration and deceleration alternatively in small time-intervals.

C. *Criteria for Addition of $\boldsymbol{s}_{new}(t+\Delta t)$ to $\mathcal{T}$*

The state $\boldsymbol{s}_{new}(t+\Delta t)$ found with the input $\boldsymbol{u}(t)$ is added to the tree, if two indicator functions I_{obj} and I_{road} have value of 1 at each time step within time interval $[t, t+\Delta t]$ during extension of the tree from $\boldsymbol{s}_{nearest}(t)$ to $\boldsymbol{s}_{new}(t+\Delta t)$. The indicator function I_{obj} at time t is defined as

$$I_{obj}(t) = \begin{cases} 1, & \text{if } \boldsymbol{s}(t) \bigcap S_{obj}(t) = \varnothing \vee v_{r,\boldsymbol{s}(t)} < v_c \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $S_{obj}(t) \subset \mathbb{R}^2$ is the area occupied by the N objectssuch that $\bigcup_1^N \boldsymbol{s}_{obj,n}$. The predicted relative collision velocity $v_{r,\boldsymbol{s}(t)}$ at state $\boldsymbol{s}(t)$ is between the EGO vehicle and the collision object while v_c is the critical impact speed defined according to the proposed generalized relationship between the impact speed, the impact angle, the type of collision object and the probability of severity of injury as in [28]. Finding states with a predicted nonsevere collision makes it possible to choose a trajectory with a low crash severity in case a collision-free trajectory is not found. These states with predicted non severe collision are not extended further as they are not collision-free states. Similarly, the indicator function I_{road} at time t is

$$I_{road}(t) = \begin{cases} 1, & \text{if } \mathbf{s}(t) \bigcap S_{nr}(t) = \varnothing \vee v_{r,\mathbf{s}(t)} < v_c \\ 0, & \text{otherwise,} \end{cases} \tag{4}$$

where, $S_{nr} \subset \mathbb{R}^2$ is the area not belonging to the road. The yaw angle Ψ, the velocity $\boldsymbol{v}$, the steering angle δ, time

$(t+\Delta t)$, state index, parent index for tracing back the trajectory, information related to the predicted severity of injury for each state is added to the tree $\mathcal{T}$. This is possible due to the use of the two-track vehicle dynamic model while planning trajectories.

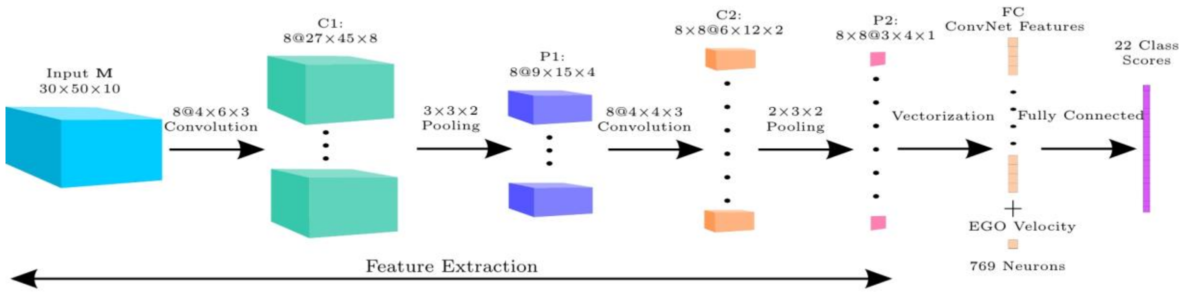


Fig. 1. 3D ConvNet architecture.

D. Selection of Best Trajectory

The trajectory planning with the ARRT and ARRT+ algorithm is divided into two subintervals of a long time interval $[t_0, t_0+\tau]$. In the first subinterval $[t_0, t_0+\tau_1]$, multiple collision-free trajectories are found by constructing a tree as explained in Section III.B. While choosing a best trajectory it is important to choose a trajectory which leads to a final vehicle state from where the vehicle can easily be controlled to drive towards the goal. Therefore, a maximum steering angle input required for the vehicle to follow the road safely with constant velocity towards the region S_{goal} in the second subinterval $[t_0+\tau_1, t_0+\tau]$ from the end position of the trajectories found in first subinterval is predicted. This is named as *steering effort* and is taken as one of the parameter while choosing the final trajectory.

The selection of best trajectory is done considering the parameters in preference order safety, steering effort, and acceleration values. If no collision-free trajectory is found then the trajectories with a collision are compared solely based on the predicted severity of injury and the one with lowest severity of injury is selected as the best trajectory for that scenario. If collision-free trajectories are found then only those are considered for further comparison. Among these trajectories, the trajectories with high steering effort above the defined threshold are eliminated or if there is none below the defined threshold then the one with lowest steering effort is selected as a label. Finally, if still multiple trajectories are remaining then the one with the lowest maximum absolute acceleration value along the trajectory is selected as the best trajectory.

If no trajectory without a collision or predicted low severity of collision is found, then the vehicle breaks sharply as fail-safe case.

E. HARRT, HARRT+ and GATE-ARRT+ Algorithms

A traffic-scenario is converted into a sequence of predicted occupancy grids $\mathbf{M}=\{\mathcal{G}_{t_0},\ldots,\mathcal{G}_{t_0+\tau_1}\}$ for the prediction interval $[t_0, t_0+\tau_1]$ with each occupancy grid $\mathcal{G}_t$ representing the occupancies of road objects at time t. The size of each occupancy grid is 30×50 and a stack of 10 such occupancy grids at equal time intervals from t_0 to $t_0+\tau_1$ are formed. Thus the size of input $\mathbf{M}$ is $30\times50\times10$. Every predicted occupancy grid is constructed with respect to the orientation of the EGO vehicle at time t_0. The cells in the predicted occupancy grid which lie in $S_{nr}(t)$ or $S_{obj}(t)$ are assigned a value 1. Rest of the cells are assigned a value 0 indicating they are free. A scenario described by $\{\mathbf{M},\boldsymbol{\eta}\}$, where $\boldsymbol{\eta}$ are EGO vehicle physical parameters like velocity, yaw-rate, etc. The critical traffic-scenario described by a sequence of predicted occupancy grids is used as an input for the machine learning algorithm as its size remains the same irrespective of the road infrastructure, type and the number of road-traffic participants in the vehicle environment. Due to the 3D structure of the input, the 3D-ConvNet is used as a machine learning algorithm. The architecture of the 3D-ConvNet used for the HARRT algorithms is as shown in the Fig. 1.

The labels for training the 3D-ConvNet is generated from simulations. In a simulation environment developed in Matlab, many critical traffic-scenarios are simulated and best trajectories are found by the ARRT and ARRT+ algorithm. The label for best trajectory found by the ARRT algorithm is the corresponding predefined acceleration profile used. The HARRT algorithm uses only few predicted acceleration profiles, instead of all, for finding the safe trajectory which increases the convergence speed of the ARRT algorithm. In case of the ARRT+ algorithm, the best trajectory does not have any fixed acceleration profile because of the random sampling of the longitudinal acceleration. Therefore, the clusters of the steering wheel angle profile and longitudinal acceleration profile are formed and the combination of these clusters in which the best trajectory for a scenario lies is considered as a label for that scenario. The HARRT+ algorithm uses the predicted clusters for the biased-sampling in both the lateral and longitudinal dynamics of the algorithm to increase the convergence speed of the algorithm.

GATE-ARRT+ algorithm distiguishes itself from the HARRT and HARRT+ in the usage of the 3D-ConvNet for regression instead of classification. Initially, a variational autoencoder is trained on trajectories. The encoder part of this variational autoencoder converts the safe trajectories found in every scenario into latent space values which then are used as labels for training the 3D-ConvNet. For testing, the predicted values by 3D-ConvNet are converted into trajectories by the decoder part of trained variational autoencoder. This trajectory is used as reference trajectory for biasing the sampling with the ARRT+ algorithm, therefore the the name of the algorithm is GATE-ARRT+.

IV. Machine Learning Algorithms for Reducing Computation Time

In the HARRT algorithm, 3D-ConvNet is used only for assisting the ARRT algorithm, but the actual safe trajectory planning is done by model-based methods which are interpretable and so suitable for safety-critical applications. As explained in Section III, different computationally intensive analytical algorithms such as collision detection and calculation of the severity of injury, I_{road} and I_{obj}, along with a vehicle dynamic model are used in each prediction time-step to generate a tree $\mathcal{T}$ with many branches, i. e., many safe trajectories as shown in the Fig. 2. As only one safe trajectory (e. g. red) is selected for the vehicle to follow, the computation time required for finding other branches is wasted. Also, the number of times the indicator functions are calculated increases with the number of objects in the surrounding of the EGO vehicle and the number of time steps required to find the final trajectory. This number is in thousands for finding a collision-free trajectory of few seconds. Therefore, machine learning algorithms $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1) \mapsto \hat{\boldsymbol{y}}_1$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2) \mapsto \hat{\boldsymbol{y}}_2$ are proposed for the indicator functions I_{obj} and I_{road}, respectively. Although this results into minor reduction in the computation time, because the algorithms are called thousands of times for generating the tree, it results into significant reduction in the required computation time. The vectors $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ are the parameters to learn by minimizing the cross-entropy loss function between the true and estimated posterior probabilities of the output labels. The vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are the input feature vectors for two machine learning algorithms while the vectors $\hat{\boldsymbol{y}}_1$ and $\hat{\boldsymbol{y}}_1$ are the outputs of the machine learning algorithms which defines the output of indicator functions I_{obj} and I_{road}. Once a safe trajectory is selected, it is validated with analytical algorithms for I_{obj} and I_{road}. This way, it is assured that the final found trajectory is fully validated with analytical methods without wasing computation time on branches of the tree $\mathcal{T}$ which will not be part of the final selected trajectory.

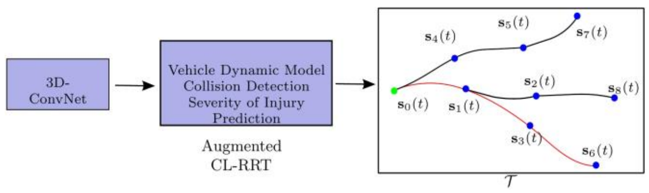


Fig. 2. Tree with multiple collision-free trajectories.

A. *Feature and Labels for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$*

The input features for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ are $\boldsymbol{x}_1 = \{x_{obj}, y_{obj}, v_r, t_{obj}\}$, where x_{obj}, y_{obj} are x- and y-coordinates of the four corners predictions of the collision object in the body coordinate frame of the EGO vehicle, v_r is the relative velocity between the EGO vehicle and the collision object, and t_{obj} is the type of the collision object, i. e., if the collision object is a pedestrian, bicyclist or a vehicle. Similarly, the input features for $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$ consist of only a fixed number of x- and y-coordinates of the nearest road outer line points x_{road}, y_{road} from the vehicle's center of gravity in the body coordinate frame of the EGO and the EGO velocity v, i. e., $\boldsymbol{x}_2 = \{x_{road}, y_{road}, v\}$. Going outside of the road is considered as colliding with a stationary object, therefore, the EGO velocity is considered as the relative collision velocity for calculating the severity of injury. Fig. 3 shows an example of extracted features (x_{obj}, y_{obj}) and (x_{road}, y_{road}). The outputs $\hat{\boldsymbol{y}}_1$ and $\hat{\boldsymbol{y}}_2$ are vectors of three labels, *no collision*, *predicted nonsevere collision*, and *predicted severe collision*. The states, for which *predicted severe collision* label for either of the indicator function I_{obj} and I_{road} is predicted, it is discarded. Otherwise, the states are added to tree $\mathcal{T}$.

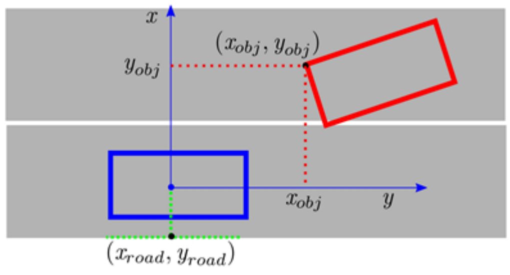
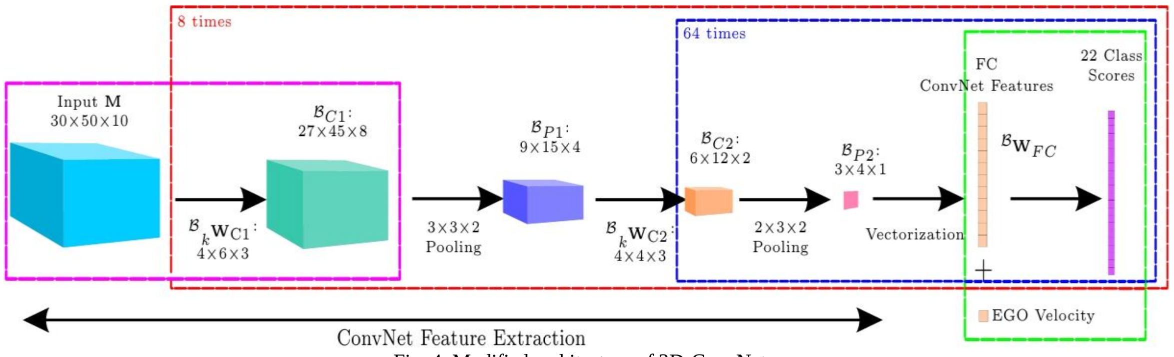


Fig. 3. Features for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$.

B. *Data Generation*

The training samples for $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ are generated by randomly sampling different (x_{obj}, y_{obj}) in the body coordinate frame of the EGO vehicle and the relative velocity v_r. The output labels are found by using the indicator function I_{obj}, which can be easily evaluated in the simulation environement. The coordinates (x_{obj}, y_{obj}) are randomly sampled such that their center lies only within 10 meters range from the center of EGO body coordinate frame as objects outside of this range eliminates the need for the collision detection. Similarly, training samples for $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$ are generated by using different road infrastructures, i. e., different road structures like curved road, straight road, intersections, etc., EGO velocities and positions. The labels for each training sample are found by using the indicator function I_{road}, which can also be easily evaluated in the simulation environment.

The machine learning algorithm $f_{\gamma_1}(x_1)$ performs the collision detection with each object in sequence while $f_{\gamma_2}(x_2)$ is trained with all types of road infrastructures. Therefore, they continue to adhere to the HARRT and HARRT+ algorithms principle of using one algorithm for all types of traffic-scenarios irrespective of the number and type of traffic participants.


8 times
64 times
Input M 30×50×10
$^{\mathcal{B}}{}_{k}\mathbf{W}_{C1}$: 4×6×3
$^{\mathcal{B}}C1$: 27×45×8
3×3×2 Pooling
$^{\mathcal{B}}P1$: 9×15×4
$^{\mathcal{B}}{}_{k}\mathbf{W}_{C2}$: 4×4×3
$^{\mathcal{B}}C2$: 6×12×2
2×3×2 Pooling
$^{\mathcal{B}}P2$: 3×4×1
Vectorization
FC ConvNet Features
$^{\mathcal{B}}\mathbf{W}_{FC}$
22 Class Scores
EGO Velocity
ConvNet Feature Extraction
Fig. 4. Modified architecture of 3D-ConvNets.

V. Analytical Algorithms for Reducing the Memory (SRAM)

As mentioned in Section II, a dynamic memory allocation should be avoided for safety critical applications like safe trajectory planning. The memory required in static RAM (SRAM) for 3D-ConvNets having an architecture as shown in the Fig. 1 will be very high if memory is preallocated for the input $\mathbf{M}$, all weights $\mathbf{W}$ for the convolution operations and fully-connected layers, feature maps $\mathbf{V}$ in all the convolution and pooling layers. The DNN processing in the cloud is not desirable due to latency, security, and communication bandwidth concerns. Therefore, different analytical algorithms are proposed to reduce the memory requirements for different parts of 3D-ConvNets as well as the ARRT algorithm. Sections V.A, V.B, V.C present the methodologies for required memory reduction for 3D-ConvNets and Section V.D is the methodology for reducing the required memory for generated trees $\mathcal{T}$ using the ARRT and ARRT+ algorithm.

A. Iterative ConvNet Feature Generation

A 3D-ConvNet architecture comprises of multiple 3D convolutions and 3D pooling layers generating multiple output feature maps as shown in the Fig. 1. In a 3D convolution layer, the value of output unit ${}_k\mathbf{V}^{xyt}$ at position (x, y) and time t in k^{th} out of K output feature maps is obtained by

$$ {}_k\mathbf{V}^{xyt} = \left(\sigma\left({}_k b + \sum_{i=0}^{P-1}\sum_{j=0}^{Q-1}\sum_{n=0}^{R-1} {}_k\mathbf{W}_{ijn}\mathbf{U}_{ijn}^{xyt}\right)\right), \quad (5)$$

where, $P, Q,$ and R are the dimensions of all K kernels ${}_k\mathbf{W}$ and $\mathbf{U}_{ijn}^{xyt}$ is the input at position $(x+i\Delta x, y+j\Delta y, t+n\Delta t)$. ReLU activation function is denoted by σ. It shows that for each input for the 3D convolution, the number of output feature maps generated is equal to the number of kernels K used. Similarly, the value of unit ${}_l\mathbf{V}^{xyt}$ in l^{th} out of L output feature maps of 3D pooling layer at position (x, y) and time t is obtained by

$$ {}_l\mathbf{V}^{xyt} = \left(\sum_{i=0}^{P'-1}\sum_{j=0}^{Q'-1}\sum_{n=0}^{R'-1} {}_l\mathbf{U}_{ijn}^{x'y't'}\right) / \left(P'.Q'.R'\right), \quad (6)$$

where ${}_l\mathbf{U}_{ijn}^{x'y't'}$ is l th out of total L inputs for the pooling layer at position $(P'.x+i\Delta x', Q'.y+j\Delta y', R'.t+n\Delta t')$ and $P', Q',$ and R' are the dimensions of slice of the input over which pooling is performed. It shows that the number of output feature maps in the pooling layer is equal to the number of input feature maps.

As can be seen in Fig. 2, for a single input $\mathbf{M}$, 8 feature maps are generated in layer C1 using 8 kernels ${}_k\mathbf{W}_{C1}$, $k = 1, 2, \dots, 8,$ followed by a pooling layer P1 which has also 8 output feature maps. For each output feature map from P1 layer further 8 output feature maps are generated with 8 kernels ${}_k\mathbf{W}_{C2}, k = 1, 2, \dots, 8$, generating in total 64 output feature maps for C2 ass well as for P2 layer. Preallocation of memory for all of these feature maps and kernels will require a huge amount of memory which is not available in automotive micro-controllers.

TABLE I: Non-Iterative and Iterative 3D-ConvNet Feature Generation

Layer	Non-Iterative method		Iterative method	
	# Parameters	Memory (Kilobytes)	# Parameters	Memory (Kilobytes)
C1	77760	303.75	9720	37.97
P1	4320	16.88	540	2.11
C2	9216	36	144	0.563
P2	768	3	12	0.047
Total	92064	359.63	10416	40.69

The convNet features for FC layer in 3D-ConvNet are obtained just by the vectorization of the output feature maps of the layer P2. Therefore, the calculation of all feature maps in P2 layer at once for the inference is not necessary. Instead they can be calculated iteratively and appended to get ConvNet features for the FC layer. Fig. 4 shows this operational change for calculating ConvNet features in sequential manner with two nested iterations in the red box with kernels ${}_k\mathbf{W}_{C1}$ and blue box for kernels ${}_k\mathbf{W}_{C2}$. The buffers $\mathcal{B}_{C1}$, $\mathcal{B}_{P1}$, $\mathcal{B}_{C2}$, and $\mathcal{B}_{P2}$ are created for layers C1, P1, C2, and P2, respectively, equal to the dimension for a feature map in respective layers. Kernels are used in sequence to calculate one feature map at the end of the P2 layer, so buffers $\mathcal{B}_{{}_k\mathbf{W}_{C1}}$ and $\mathcal{B}_{{}_k\mathbf{W}_{C2}}$ are also defined for kernels ${}_k\mathbf{W}_{C1}$ and ${}_k\mathbf{W}_{C2}$, respectively. These iteratively generated output feature maps can be vectorized into a vector of size 12 and appended continuously to get the ConvNet features for the FC layer. If parallel processing is used to speed the features computation, the buffers equal to the number of parallel processors can be created which will help increase the computation speed without assigning memory to all feature maps. In this work no parallel processing is used.

Table I shows the difference between the iterative and non-iterative ConvNet features generation in terms of the required number of parameters to be stored, thereby the size of memory required (considering float datatype with memory required for 1 parameter is 4 bytes) in convolution and pooling layers of the 3D-ConvNet.

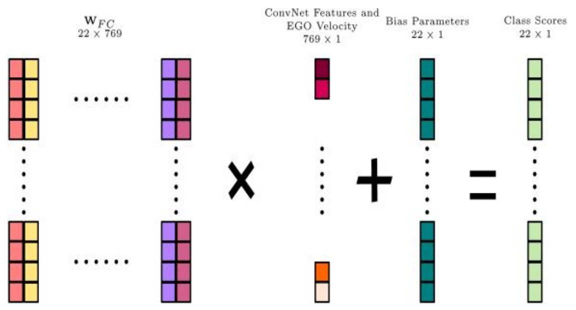

$\mathbf{W}_{FC}$ 22 × 769
ConvNet Features and EGO Velocity 769 × 1
Bias Parameters 22 × 1
Class Scores 22 × 1

Fig. 5. Normal FC layer operation.

B. Iterative Convolution of the Input $\mathbf{M}$

As explained in Section III, the input $\mathbf{M}$ is the sequence of occupancy grids having values either 0 or 1 for each cell. Therefore, its data type can be defined as ‘unsigned int’. But the kernels used for convolution operation have data type ‘float’ because of which the input must also be ‘float’. Even if the input is defined as ‘unsigned int’, the compiler changes the datatype to float before performing convolution operation. As the input contains 15000 (dimention of) parameters in total, this results into huge memory consumption. Each unit of the output feature map in the C1 layer of 3D-ConvNet is calculated iteratively with a kernel convolving over a part of the input equal to the dimension of a kernel . This offers a possibility to define the input as ‘unsigned int’ and only copy part of the input into the buffer of dimension equal to kernel with data type ‘float’. Therefore, instead of using 15000 ‘float’ parameters, 15000 ‘unsigned int’ parameters and a buffer of dimension equal to kernel with data type ‘float’ are used. This iterative convolution of the input , shown in the part of the 3D-ConvNet with magenta box in Fig. 4, results into a saving of memory equal to 43.66 kilobytes.

C. Fully-Connected Layer in Compressed Sparse Column Format

Most number of learned parameters of 3D-ConvNet are in the fully-connected layer FC. The 768 ConvNet features are generated in sequence because of the use of iterative ConvNet features generation explained in Section V.A. At every iteration only 12 ConvNet features are generated which can be appended continuously to previously found ConvNet features. The weights $\mathbf{W}_{FC}$ having dimension of 22×769 (16918 parameters) are multiplied with a vector of dimension 769×1, consisting of the generated ConvNet features and EGO velocity, to get a vector of dimension 22×1 followed by addition of 22 bias parameters as shown in the Fig. 5. This is further used as an input for a softmax function to generate 22 class scores. Thus, the total number of parameter in FC layer are 16940. Although these parameters are stored in FLASH memory of the micro-controller, they are fetched to SRAM during the calculation. Fetching all 16940 parameters at once consumes a lot of memory in SRAM. Therefore, the multiplication of $\mathbf{W}_{FC}$ and ConvNet features is done in a Compressed Sparse Column (CSC) format [29] as shown in Fig. 6.

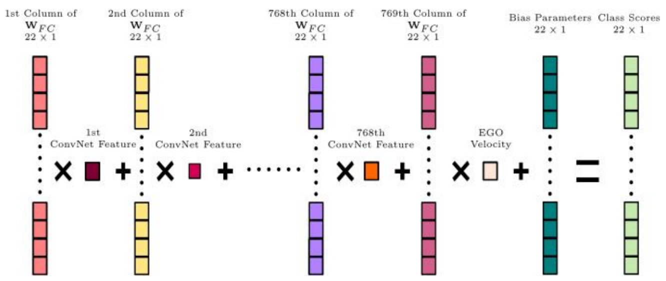

1st Column of $\mathbf{W}_{FC}$ 22 × 1
2nd Column of $\mathbf{W}_{FC}$ 22 × 1
768th Column of $\mathbf{W}_{FC}$ 22 × 1
769th Column of $\mathbf{W}_{FC}$ 22 × 1
Bias Parameters 22 × 1
Class Scores 22 × 1
1st ConvNet Feature
2nd ConvNet Feature
768th ConvNet Feature
EGO Velocity

Fig. 6. Compressed sparse column format.

The output of the product is calculated by taking iteratively the parameters in a single column of $\mathbf{W}_{FC}$ at a time in the buffer $\mathcal{B}_{\mathbf{W}_{FC}}$ of dimension 22×1 and multiplying it with corresponding ConvNet feature which is added to the output continuously to get the class scores. This part of the 3D-ConvNet architecture is shown as a green box in the Fig. 4. This eliminates the requirement for fetching all 16940 parameters at once but just require memory equivalent to 22 parameters for the buffer $\mathcal{B}_{\mathbf{W}_{FC}}$ and 22 bias parameters saving memory for 16896 parameters. Also, as a single ConvNet feature is used in each iteration, finding all 768 ConvNet features at once is not required. Using iterative ConvNet feature generation 12 features of FC layer are generated iteratively as explaind in Section V.A. Therefore, the buffer $\mathcal{B}_{\mathbf{W}_{FC}}$ of dimention 12×1 is used to store these features. This also reduces the required number of parameters by 756. Thus, the usage of the CSC format multiplication in FC layer saves the memory equivalent to 17652 parameters having data type ‘float’, i. e., 68.95 kilobytes.

D. Storage of Only One State with Low Severity of Injury

In the HARRT algorithm, the found states that are not collision-free, are still added to the tree, if the predicted severity of injury at these states is low. It provides an option of following a trajectory with predicted low severity of collision, when a collision-free trajectory is not found at the cost of increasing the number of states to be stored in the tree, thereby increasing the required SRAM memory. This is especially because there is a lot of information saved along with these states as described in Section III. The number of states to be stored has to be predefined in order to preallocate the memory for these states. Storing each state with the predicted low severity of collision will fill this memory quickly and the algorithm will not be able to find and store more collision-free states, even if the computation time is available. To avoid this, only one state with the predicted low severity of collision is stored in the tree $\mathcal{T}$. If a new state with even lower predicted severity of injury is found, it just replaces the previously stored state.

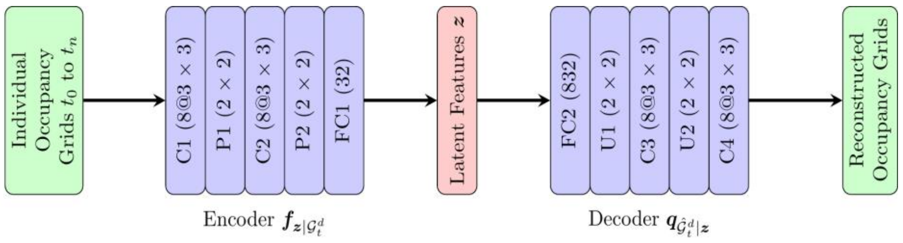

Individual Occupancy Grids t_0 to t_n
C1 (8@3 × 3)
P1 (2 × 2)
C2 (8@3 × 3)
P2 (2 × 2)
FC1 (32)
Latent Features $\boldsymbol{z}$
FC2 (832)
U1 (2 × 2)
C3 (8@3 × 3)
U2 (2 × 2)
C4 (8@3 × 3)
Reconstructed Occupancy Grids
Encoder $\boldsymbol{f}_{\boldsymbol{z}|\mathcal{G}_t^d}$
Decoder $\boldsymbol{q}_{\hat{\mathcal{G}}_t^d|\boldsymbol{z}}$

Fig. 7. Autoenocoder for occupancy grids.

VI. Alternative Architectures

In HARRT+ algorithm, 3D-ConvNet is chosen for extracting features from the input $\mathbf{M}$ as it is a suitable model for extracting the required spatio-temporal features. However, it requires a high memory consumption. Section V has proposed few methodologies for reducing the required computational memory. An alternative approach to reduce the required computational memory is to choose a different computationally efficient network architecture which also learns the spatio-temporal features from the input $\mathbf{M}$.

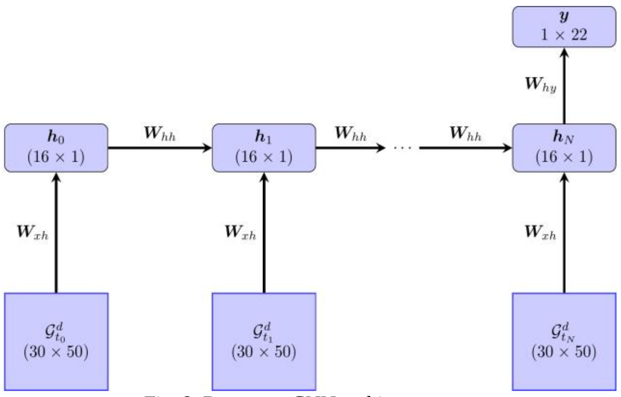


Fig. 8. Recurrent-CNN architecture.

In 3D ConvNet, the 3D feature maps are generated by using 3D filters having smaller depth dimension than its input depth. These feature maps are not appended like in 2D ConvNet. In fact, the second convolutional layer performs convolution on each generated 3D feature map individually. This is shown in Fig. 1. This might affect the performance of the network as the spatial feature correlation in different feature maps at particular time step is not learned in intermediate layers. Only in the fully connected layer, all the extracted features are combined. However, the fully connected layer surrenders all the spatio-temporal correlation information in the input. Therefore, an analysis of other possible architectures is needed. In addition, 3D-ConvNet involves high number of computational operations compared to 2D-ConvNet. This section proposes alternative architectures to solve the problem of high computational resources requirements while not surrendering the spatio-temporal feature correlation.

Recurrent convolutional neural network (Recurrent-CNN) [30], as shown in Fig. 8, is a natural selection for learning spatio-temporal features because of its suitable architecture. The weights $\boldsymbol{W}_{xh}$ represent weights of 2D-ConvNet which will learn the relevant spatial features from input occupancy grids while the connection of latent spaces using the weights $\boldsymbol{W}_{hh}$ will learn the temporal correlation of these spatial features. RNNs share the weights at each step and the extraction of features from inputs at different time-steps can be parallelized which makes them highly computationally efficient. Another advantage for RNNs is that they don’t need fixed size of the input. As the number of time-steps, here 10, are small, the LSTM or GRU network, which eliminate the problem of vanishing and exploding gradients, are not suitable. Also, because the input $\mathbf{M}$ have the fixed size, a complicated architecture such as RNN is not necessary. Another way is to extract initially only spatial features from individual occupancy grids of $\mathbf{M}$ using simple 2D-ConvNet followed by a fully-connected layer. The weights of 2D-ConvNet can be shared similar to the RNN architecture. This is achieved by initially training an autoencoder with indiviual occupancy grids of all data D as input as shown in Fig. 7. The encoder $\boldsymbol{f}_{\mathbf{z}|\mathcal{G}_t^d}$, where $t \in \{t_0, \dots, t_n\}$ and $d \in \{1, \dots, D\}$, is used as a common 2D-Convnet. The problem with this methodology is that although spatial feature correlation is learned by the 2D-ConvNet, the temporal features correlation is surrendered in fully-connected layer. This problem can be solved by using a Temporal Convolutional Networks (TCN) [31] architectures.

TCN is a family of networks for convolutional sequence prediction. A temporal convolution using 1D filters is applied in [31] to capture how the input signals evolve over the course of an action for video-based action recognition. Dilated causal convolutions are used in [32] for raw audio generation to learn long range temporal dependencies. A convolutional encoder network with soft attention followed by LSTM decoder is used in [33] for language translation. A gated convolutional network is used for language modeling in [34]. All these approaches are for long time series. As the time-sequence in given problem is fixed and short, the 1D-TCN followed by fully-connected layer is used as shown in Fig. 9.

VII. Results and Discussion

A. Accuracy for $f_{\Upsilon_1}(\boldsymbol{x}_1)$ and $f_{\Upsilon_2}(\boldsymbol{x}_2)$

Using the data generation procedure explained in Section IV, 2 million and 1 million training samples are generated for the machine learning functions $f_{\Upsilon_1}(\boldsymbol{x}_1)$ and $f_{\Upsilon_2}(\boldsymbol{x}_2)$, respectively. For both algorithms, neural networks with one hidden layer are used for training. 70% of the total data was applied for training and 15% of the data for both validation and testing. The results of training with different numbers of neurons in the hidden layer for both algorithms is shown in Table II.

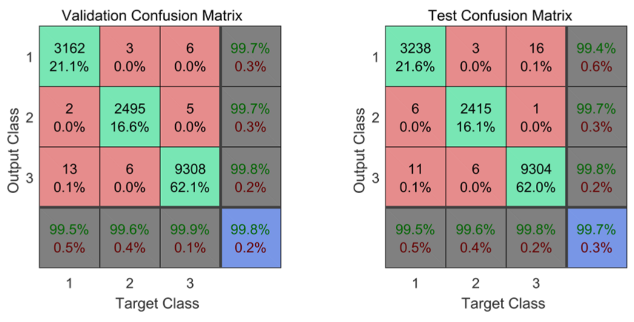


Fig. 9. Confusion matrices $f_{\Upsilon_1}(x_1)$.

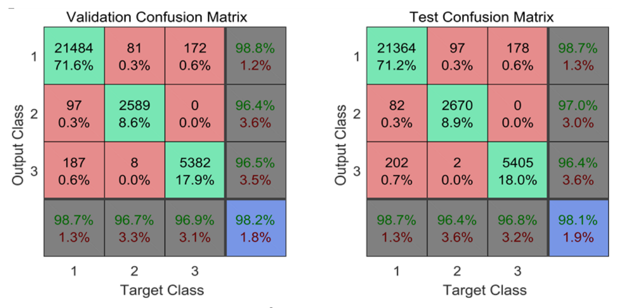


Fig. 10. Confusion matrices $f_{\Upsilon_2}(x_2)$.

A neural network with 20 neurons in the hidden layer is selected for implementation as it gives best results with the validation and training data of both machine learning algorithms as shown in confusion matrices in Fig. 10 and Fig. 11. The class labels 1, 2, 3 defined in Fig. 10 and Fig. 11 are for *no collision*, *nonsevere collision*, and *severe collision*, respectively.

TABLE II: Training Results (% Accuracy) for $f_{\Upsilon_1}(\boldsymbol{x}_1)$ and $f_{\Upsilon_2}(\boldsymbol{x}_2)$

	$f_{\Upsilon_1}(x_1)$		$f_{\Upsilon_2}(x_2)$	
# Neurons	Validation Data	Training Data	Validation Data	Training Data
10	93.3	93.2	97.3	98.7
20	98.2	98.1	99.8	99.7
30	97.7	97.6	99.3	97.3

A Matlab neural network toolbox [35] is used to to train the neural networks. The cross-entropy loss function is used as loss function for training the neural networks and all the layers use ReLU activation function.

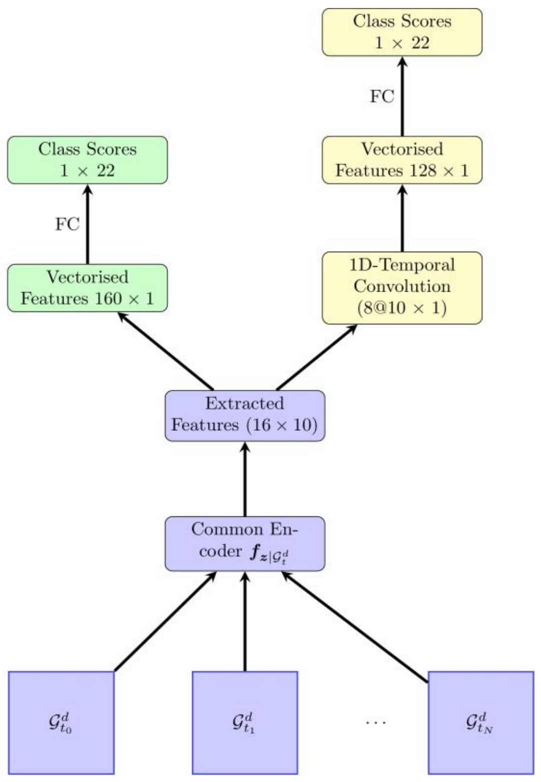

Class Scores 1 × 22
FC
Vectorised Features 160 × 1
Class Scores 1 × 22
FC
Vectorised Features 128 × 1
1D-Temporal Convolution (8@10 × 1)
Extracted Features (16 × 10)
Common En-coder $f_{z|\mathcal{G}_t^d}$
$\mathcal{G}_{t_0}^d$
$\mathcal{G}_{t_1}^d$
· · ·
$\mathcal{G}_{t_N}^d$

Fig. 11. FC and 1D-TCN network architecture.

B. Implementation and Measurement Results for Various Hardware Platforms

For the implementation of the algorithms a TMS570LS series Texas Instrument micro-controller TMS570LS20216, the dSPACE MicroAutobox II and a Raspberry Pi 3 are used. TMS570LS20216 is an automotive micro-controller certified for the use in IEC 61508 SIL3 safety systems. It has a ARM Cortex-R4F floating point CPU which runs with speed of 160 MHz and have 2 MB and 160 KB of FLASH and SRAM size, respectively. The small amount of SRAM available in the TMS570LS20216 micro-controller indicates the significance of the amount of memory reduction obtained by methodologies that has been described in Section V. The dSPACE MicroAutobox II is a real-time system for performing fast function prototyping. It runs with the speed of 900 MHz and 16 MB RAM while the Raspberry Pi 3 has 1.2 GHz frequency and 1 GB RAM. The results for the mex-implementations in Matlab are also presented. The computer used for the mex-implementations measurements has an Intel Core-i7 processor with 2.8 GHz and 8GB RAM.

The Simulink coder is used to generate the optimized C-code of the Matlab implementation of the ARRT, HARRT and optimized HARRT algorithms with the machine learning and analytical methods proposed in Sections IV and V. This generated code is downloaded to TMS570LS20216, dSPACE MicroAutobox II, Raspberry Pi 3 and the required amount of memory (SRAM) and time for execution is measured. The results are summarized in Table III. The first two rows show the required computational resources for the ARRT and HARRT algorithm without the methodologies proposed in this paper. Subsequent rows shows the gradual reduction in computational resource with each methodology. The results show that the computation time for the mex-implementation and dSPACE MicroAutobox is very low, but the computation time for the automotive micro-controller TMS570LS20216 and raspberry pi is still quite high. A solution to this result is to use parallel processing which can lead to significant reduction in computation time. The proposed algorithm allows the parallelization of many modules, e. g., predictions of road objects, 3D-ConvNet as well as RRT algorithms. However, parallelization is not the main scope of this paper and will be considered in further work. It can be seen from Table III that the proposed techniques lead to a significant reduction of computational resources and show that there is potential to use the proposed algorithms for trajectory planning in real time.

TABLE III: COMPUTATION TIME MEASUREMENT IN VARIOUS HARDWARE PLATFORMS

Method	SRAM (KB)	Mex implementation (ms)	TMS570LS2021 6 (ms)	MicroAutobox II (ms)	Raspberry Pi (ms)
Augmented CL-RRT	374.01	28.7410	-	340	800
Hybrid Augmented CL-RRT	812.30	12.5790	-	98	290
+ $\boldsymbol{f}_{\gamma_1}(\boldsymbol{x}_1)$ and $\boldsymbol{f}_{\gamma_2}(\boldsymbol{x}_2)$	590.89	10.1994	-	47	220
+ Iterative ConvNet feature generation	271.50	8.9062	-	36	200
+ Iterative Convolution of	227.84	8.2747	-	19	140
+FC layer in CSC format	158.89	8.2747	423	19	140

C. Alternative Architecture Comparison

42191 scenarios for curved roads from [22] were used to train the models with the architectures defined in Section VI. Table IV provides the comparison between all the presented networks in this section based on accuracy in terms of the top-2 error and the number of parameters in the network. A top-2 error tests if the reference class was within 2 hypothesis with highest probability. There is no significant difference in the number of parameters in all networks, however 1D-TCN network provides best accuracy on the test data.

TABLE IV: COMPARISON OF DIFFERENT ARCHITECTURES

	3D-CNN	FC	1D-TCN	Recurrent-CNN
top-2 (%) error	9.32	11.36	**7.75**	9.69
# parameters	17916	13488+ 3542	13488+ 2926	**13488+ 646**

VIII. CONCLUSION

This work describes combination of model-based search algorithms for safe trajectory planning in critical traffic-scenarios, with simultaneous intervention in the vehicle lateral and longitudinal dynamics, along with their combination with deep learning algorithm. The motivation for this hybrid approach is a much faster convergence of the model-based algorithms. In order to make these algorithms suitable for embedded implementation further machine learning and analytical approaches have been proposed. The results of the optimization and embedded implementation, i. e., the memory and time required in different hardware platforms is presented. The results show multiple times reduction in both memory and time requirements. Also, different alternative architectures for learning spatiotemporal features are presented and comapred in this work.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

AUTHOR CONTRIBUTIONS

Amit Chaulwar developed methodologies for efficient implementation of hybrid machine learning algorithms. He also tested alternative architecture for learning spatio-temporal features. Hussein Al-Hashimi optimized and implemented the algorithms in various hardware platforms and measured the required computational resources. Michael Botsch and Wolfgang Utschick have developed the hybrid machine learning algorithms for the presented application together with Amit Chaulwar. All authors had approved the final version.

REFERENCES

[1] M. Buehler, K. Lagnemma, and S. Singh, "The 2005 DARPA grand challenge," *Springer Tracts in Advanced Robotics*, 2005.

[2] M. Buehler, K. Lagnemma, and S. Singh, "The DARPA urban challenge," *Springer Tracts in Advanced Robotics*, 2009.

[3] S. Waydo and R. Murray, "Vehicle motion planning using stream functions," in *Proc. International Conference on Robotics and Automation*, 2003.

[4] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transaction on System Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, 1968.

[5] A. Stenz, "Optimal and efficient path planing for partially-known environments," in *Proc. International Conference on Robotics and Automation*, 1994.

[6] L. Kavraki, P. Svetska, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," in *Proc. International Transactions on Robotics and Automation*, 1996.

[7] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proc. International Conference on Robotics and Automation*, 2000.

[8] M. Otte, "A survey of machine learning approaches to robotic path planning," *Springer Tracts in Advanced Robotics*, 2009.

[9] Y. Kuwata, J. Teo, G. Fiore, S. Karamann, and E. Frazzoli, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control System Technology*, 2009.

[10] M. Otte and E. Frazzoli, "RRT-X: Real time motion planning/replanning for environments with unpredictable obstacles," in *Proc. International Workshop on Algorithmic Foundations of Robotics (WAFR)*, 2014.
[11] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proc. International Conference on Intelligent Robotics and Systems*, 2003.
[12] S. Kiesel, E. Burns, and W. Ruml, "Abstraction-guided sampling for motion planning," *SoCS*, 2012.
[13] S. Karamann, M. Walter, A. Perez, and E. Frazzoli, "Anytime motion planning using the RRT," in *Proc. International Conference on Robotics and Automation*, 2011.
[14] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *Proc. International Conference on Robotics and Systems*, 2011.
[15] R. Alterovitz, S. Patil, and A. Derbakova, "Rapidly-exploring roadmaps: weighing exploration vs. refinement in optimal motion planning," in *Proc. International Conference on Robotics and Automation*, 2011.
[16] D. Kim, J. Lee, and S. Yoon, "Cloud RRT*," in *Proc. International Conference on Robotics and Automation*, 2014.
[17] J. Gammell, S. Srinivasa, and T. Barfoot, "Batch informed trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Proc. International Conference on Robotics and Automation*, 2015.
[18] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. International Conference on Robotics and Automation*, 2016.
[19] J. Huh and D. Lee, "Learning high-dimensional mixture models for fast collision detection in rapidly-exploring random trees," in *Proc. International Conference on Robotics and Automation*, 2016.
[20] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, pp. 484-489, 2018.
[21] T. Anthony, Z. Tian, and D. Barber, "Thinking fast and slow with deep learning and tree search," arxiv:1705.08439, 2017.
[22] A. Chaulwar, M. Botsch, and W. Utschick, "A hybrid machine learning approach for planning safe trajectories in complex traffic-scenarios," in *Proc. International Conference on Machine Learning and Applications*, 2016.
[23] A. Chaulwar, M. Botsch, and W. Utschick, "A machine learning based biased-sampling approach for planning safe trajectories in complex traffic-scenarios," in *Proc. Intelligent Vehicle Symposium*, 2017.
[24] A. Chaulwar, M. Botsch, and W. Utschick, "Generation of reference trajectories for safe trajectory planning," in *Proc. International Conference of Artificial Neural Networks*, 2018.
[25] S. Ji, W. Xu, M. Yang, and K. Yu, "3D convolutional neural networks for human action recognition," *TPAMI*, 2013.
[26] A. Chaulwar, M. Botsch, T. Krueger, and T. Miehling, "Planning of safe trajectories in dynamic multi-object traffic-scenarios," *Journal of Traffic and Logistics Engineering*, vol. 4, no. 2, pp. 135-140, 2016.
[27] G. Holzmann, "The power of 10: Rules for developing safety-critical code," *10 Years of Innovation in IEEE Computer*, 2016.
[28] C. Jurewicza, A. Sobhania, J. Woolleyb, and J. Dutschkeb, "Exploration of vehicle impact speed-injury severity relationship for application in safer in safer road design," *Transportation Research Procedia*, 2016.
[29] R. Dorrance, F. Ren, and D. Markovic, "A scalable sparse matrix-vector multiplication kernel for energy-efficient sparse blas on FPGAs," in *Proc. ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2014.
[30] P. Pinheiro and R. Collobert, "Recurrent convolutional neural networks for scene labeling," in *Proc. International Conference on Machine Learning*, 2014.
[31] C. Lea, M. Flynn, R. Vidal, A. Reiter, and G. Hager, "Temporal convolutional networks for action segmentation and detection," *Computer Vision and Pattern Recognition*, 2017.
[32] A. Oord *et al.* (2016). WaveNet: A generative model for raw audio. [Online]. Available: https://arxiv.org/abs/1609.03499
[33] J. Gehring, M. Auli, D. Grangier, and Y. Dauphin, "A convolutional encoder model for neural machine translation," *Association of Computational Linguistics*, 2017.
[34] Y. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *Proc. International Conference on Machine Learning*, 2017.
[35] Mathworks, *Neural Network Toolbox for Use with MATLAB: User's Guide*, Mathworks Inc., 2001.
[36] Mathworks, *Simulink Coder Toolbox for Use with MATLAB: User's Guide*, Mathworks Inc., 2001.

Copyright © 2021 by the authors. This is an open access article distributed under the Creative Commons Attribution License which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited (CC BY 4.0).

Amit Chaulwar received the B.Tech. degree in mechanical engineering from the College of Engineering, Pune (COEP), India, in 2009, and the M.Eng. degree in automotive engineering from Technische Hochschule Ingolstadt, Ingolstadt, Germany, in 2015.

Prior starting his M.Eng. degree, he was working as a system engineer in Tata Consultancy Services, Pune, India. From 2014 to 2017, he worked as a research assistant in research center CARISSMA at Technische Hochschule Ingolstadt. In 2018, he was the technical head of Vehicle Integral Safety Department at the same institute. Currently, he is working as a perception software project manager at ZF Friedrichshafen. His research interests include machine learning applications to autonomous vehicles.

Hussein Al-Hashimi received his bachelor (B.Sc.) and master (M.Sc.) degrees in laser and optoelectronics engineering from Al-Nahrain University, Baghdad, Iraq, in 2000 and 2003, respectively. From 2003-2009, he was involved in RF optimization and planning of mobile networks in Iraq with Zain company. He had awarded a doctoral scholarship from the German Academic and Exchange Service (DAAD) in 2009. In 2014, he received the doctoral degree (Dr.-Ing.) from the Institute of Microwaves and Photonics, Friedrich-Alexander University Erlangen-Nuremberg (FAU), Erlangen, Germany.

From 2016 to 2018, he had joined Technische Hochschule Ingolstadt, Research Center CARISSMA, where he worked on developing of embedded algorithm systems for safe trajectory planning in complex traffic scenarios. Currently, he is senior system engineer in Expleo Germany GmbH. His current interests include signal processing, automotive radar sensors, embedded systems, machine learning, and advanced driver-assistance systems (ADAS) applications.

Michael Botsch received the diploma and doctoral degrees in electrical engineering, both with honors, from Technische Universität München, München, Germany, in 2005 and 2009.

He worked for five years in the automotive industry as a development engineer at Audi AG in active safety systems. In October 2013, he was appointed as a professor for vehicle safety and signal processing at Technische Hochschule Ingolstadt in the Department of Electrical Engineering and Computer Science. He is the scientific director of the technology field integrated safety in the research center CARISSMA at Technische Hochschule Ingolstadt and the Scientific Director of the artificial intelligence center AININ. His research interests are in signal processing, machine learning and automotive applications. He is a member of IEEE and VDE.

Wolfgang Utschick completed several years of industrial training programs before he received the diploma degree in 1993 and the doctoral degree (both with Hons.) in 1998 in electrical engineering with a dissertation on machine learning from Technische Universität München (TUM), München, Germany. Since 2002, he has been a professor at TUM, where he is chairing the professorship of signal processing. He teaches courses on signal processing, stochastic processes, and optimization theory in the field of wireless communications, various application areas of signal processing, and power transmission systems. Since 2011, he has been a regular guest professor at Singapore's new autonomous university, Singapore Institute of Technology, and since 2017, he has been serving as the dean of the Department for Electrical and Computer Engineering, TUM. He holds several patents in the field of multiantenna signal processing and has authored and co-authored a large number of technical articles in international journals and conference proceedings, and has been awarded with a couple of best paper awards. He edited several books and is a founder and the editor of the Springer book series Foundations in Signal Processing, Communications and Networking. He has been a principal investigator in multiple research projects funded by the German Research Fund (DFG) and a coordinator of the German DFG priority program Communications Over Interference Limited Networks. He is a member of the VDE and therein a member of the Expert Group 5.1 for Information and System Theory of the German Information Technology Society. He is currently chairing the German Signal Processing Section. He also had been serving as an associate editor for the IEEE Transactions on Signal Processing and had been member of the IEEE Signal Processing Society Technical Committee on Signal Processing for Communications and Networking.