Combining Truncated Binary Search Tree and Direct Search for Flexible Piecewise Function Evaluation for Explicit MPC in Embedded Microcontrollers

Farhad Bayat * **Tor A. Johansen** ** **Ali A. Jalali** *

* *Iran University of Science and Technology, Tehran, Iran, (e-mail: fbayat@iust.ac.ir, ajalali@iust.ac.ir).*
** *Norwegian University of Science and Technology, Trondheim, Norway, (e-mail: Tor.Arne.Johansen@itk.ntnu.no)*

Abstract: Algorithms for efficient evaluation of general piecewise functions defined over convex polyhedral partitions are considered. As an important application, the explicit model predictive control (eMPC) problem requires a piecewise affine (PWA) control law to be evaluated online. The widely used Binary Search Tree (BST) method is modified to be able to deal with a wider class of problems for which the BST method becomes prohibitive in terms of preprocessing time or memory requirements. First a Truncated Binary Search Tree (TBST) is proposed which enables the designer to trade-off between preprocessing, storage and online computation time. Second, an alternative TBST algorithm is proposed utilizing artificial orthogonal hyper-planes (OHPs) and shown to simplify the preprocessing to a large extent while the pre-defined worst case online computation time is guaranteed.

Keywords: Piecewise Function Evaluation, Explicit MPC, Embedded Microcontrollers.

1. INTRODUCTION

Piecewise functions have received significant attentions over the last few years, and in most of the applications efficient evaluation of the piecewise function is a crucial problem especially when there is a large number of function pieces. Our motivation is mainly originated, but not limited, in explicit MPC application. Recently in Bemporad et al. (2002b) and Bemporad et al. (2002a) it was recognized that the solution to the linear MPC problem can be cast as a multi-parametric linear/quadratic program and solved explicitly, resulting in a PWA function of the current state x(t). These approaches lead to a PWA control law defined over several convex polyhedral regions. So the evaluation of such a control law requires to determine the polyhedral region which contains the current state $x(t)$, the so-called point location problem. By exploiting the piecewise affinity of the associated value function for MPC with linear cost function, in Baotić et al. (2008) it was shown that the point location problem can be solved with no need to store the polyhedral partitions provided that a convex PWA function defined over the polyhedral partition is available. In the case of linear cost-function in Jones et al. (2006) the point location problem was posed as a weighted Voronoi diagram, which can be solved using an approximate nearest neighbor algorithm. In Christophersen et al. (2007a) combining the concept of interval trees and bounding boxes associated with the polyhedral regions, a particular search tree was proposed to solve the point location problem. Although it can be applied to larger partitions, yet the online computational performance can be poor and there is no flexibility to trade-off between offline and online complexities in order to guarantee specific requirements. In Wen et al. (2009) a lattice PWA expression of the explicit MPC solution obtained based on the multi-parametric programming which can save some online calculation and memory requirements. More recently, in Bayat et al. (2011b), exploiting the concept of hashing theory a two-stage algorithm was proposed which combines the direct search method with an efficient set intersection algorithm together with some extremely efficiently solvable one-dimensional sub-problems (i.e. in the time of order $\mathcal{O}(1)$) to solve the whole problem. The hash-based method enables the designer to trade-off between online and offline complexities while it does not make any assumptions beyond the PWA structure. Therefore it can be applied for a general class of partitions including discontinuous and overlapped ones. Among the existing methods which can be applied to more general class of partitions including discontinuous and overlapping ones, the Binary Search Tree (BST), Tøndel et al. (2003), is acknowledged for its efficient online computational performance. Unfortunately, the pre-processing time in this method becomes prohibitive when the complexity of the partition increases. This motivates us to modify the BST method to reduce the pre-processing time and storage requirements and thereby enabling not only extended applicability of this method but also rapid prototyping of embedded control systems where a fast design iterations is of vital importance to be able to re-design controllers quickly taking advantage of extended processing power typical for rapid prototyping environments. To this aim two modifications are proposed which can be used together or independently. At first, constraining the maximum available online computation time,

a truncated binary search tree is proposed. This truncation relies on direct search among the pre-computed function pieces remaining at the leaf nodes on the TBST to locate the optimal region. The BST data structure represents a recursive splitting of the domain of the piecewise function by including a new hyper plane defining the polyhedral regions at each node in the search tree. As a second alternative, we propose to select these hyper planes among a carefully selected set of axis-orthogonal hyper planes when constructing the BST which leads to faster evaluation of each node when traversing the tree. Furthermore, some approximate alternatives are suggested to replace the required direct search in the proposed algorithms. In the case of using the approximation approaches it is shown that a considerable reduction in the storage requirements and online computation time can be achieved in the price of moderate pre-processing load and approximation error. The key features of the proposed methods is that the exact solution can be computed with pre-defined worst case online computation time guarantees, while pre-processing time and memory use is implicitly minimized subject to this constraint. The computations are readily implementable using fixed-point arithmetic on a low cost microcontroller or DSP since there is no recursive accumulation of round-off errors, and the online algorithm is simple with a small footprint suitable for formal verification of correctness of implementation. This is an advantage compared to other methods that rely on more complex online iterative floating-point computations in numerical optimization solvers, Zeilinger et al. (2008) and Ferreau et al. (2008).

2. PIECEWISE FUNCTION EVALUATION

Here we consider an arbitrary piecewise function $\mathrm{f}_{pw}(x)$, defined over convex polyhedral regions $\mathcal{X}_i$ collected in a partition $X = \cup_{i=1}^{N_r} \mathcal{X}_i$ and $\mathrm{f}_{pw}(x) = f_i(x) \, \forall \, x \in \mathcal{X}_i$. Then for a given feasible point $x \in X$, the piecewise function evaluation is to find the index $i(x) \in \{1, ..., N_r\}$ for which $x \in \mathcal{X}_{i(x)}$ and $\mathrm{f}_{pw}(x) = f_{i(x)}(x)$.

2.1 Explicit MPC Application

Consider the class of piecewise linear systems described by the following equations:

$$x(t+1) = A_j x(t) + B_j u(t) + d_j, \quad if \quad \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \in \mathcal{D}_j \quad (1)$$

where $\mathcal{D}_j$ is a non-empty polyhedral set in $R^{n_x+n_u}$ and $int(\mathcal{D}_j) \cap int(\mathcal{D}_k) = \emptyset$, $j \neq k$. Then the constrained finite time optimal control (CFTOC) problem for (1) is:

$$\begin{aligned} J^*(x_t) = \min_U \; & \|Q_f x_{t+N}\|_p + \sum_{k=0}^{N-1} \|Q x_{t+k}\|_p + \|R u_{t+k}\|_p \\ s.t. \;\; & x_t = x(t) \\ & C_x^k x_{t+k} + C_u^k u_{t+k} \leqslant C_0^k, \;\; k = 1, ..., N_c \quad (2) \\ x_{t+k+1} = A_j & x_{t+k} + B_j u_{t+k} + d_j, \; if \begin{bmatrix} x_{t+k} \\ u_{t+k} \end{bmatrix} \in \mathcal{D}_j \end{aligned}$$

where $U = \left[u_t^T, u_{t+1}^T, ..., u_{t+N-1}^T\right]^T$ is the optimization variable, $\|\cdot\|_p$ denotes the norm when $p \in \{1, \infty\}$ and for $p = 2$, $\|Qx\|_p = x^T Q x$. As shown in Bemporad et al. (2002b) and Borrelli (2003), the solution to the CFTOC problem (2) at each time t is a time-invariant PWA function of the current state $x(t)$ as:

$$\begin{aligned} u^*(t) &= K_i x(t) + k_i, \quad if \; x(t) \in \mathcal{X}_i \\ \mathcal{X}_i &= \{x \in R^{n_x} | H_i x \leq h_i\}, \forall i = 1, ..., N_r \end{aligned} \quad (3)$$

Therefore, for the state $x(t)$, the online evaluation of explicit MPC requires to find the optimal polyhedral region $\mathcal{X}_i$ containing the state $x(t)$, i.e. $x(t) \in \mathcal{X}_i$.

2.2 Binary Search Tree (BST)

In order to support efficient online evaluation of PWA functions, in Tøndel et al. (2003) a binary search tree is constructed on the basis of the HPs defining the convex polyhedral regions $\mathcal{X}_i$. The idea is to use these HPs as separating variables in the nodes of the tree to split the partitioned space in a way that the number of candidate functions decreases as much as possible from the current to the next level of the tree. In online operation, and starting from the root node, one needs to identify on which side of the corresponding HP the current state resides and move through the tree until the optimal region is found. This approach lead to an efficient solution in terms of online evaluation complexity which is (in the best case) logarithmic in the number of regions (N_r). When the number of regions increases the cost of construction of the BST increases quickly and becomes prohibitive for large N_r. This is due to the fact that for constructing such a tree, for each node the BST algorithm needs to determine on which side of every HP each polyhedral region is located. This requires $2N_rN_H$ linear programs (LPs) to be solved where N_H denotes the number of HPs and is typically much larger than the number of regions. The storage requirement of the BST is in the order of $\mathcal{O}(n_x N_n)$, where N_n denotes the number of nodes of the tree. We emphasis that, less optimal trees can be generated by solving a reduced number of LPs or selecting HPs based on other criteria. The latter will be presented in section 2.5, while section 2.4 introduces the idea of combining a truncated BST with direct search.

2.3 Truncated Binary Search Tree (TBST)

Consider the set of polyhedral regions $\{\mathcal{X}_1, ..., \mathcal{X}_{N_r}\}$ and the corresponding set of distinct functions $\{f_1, ..., f_K\}$ where $K \leq N_r$, since some regions may have the same function. Let $\{a_j^T x - b_j = 0, j = 1, ..., N_H\}$ denotes the set of all separating HPs defining the polyhedral partition. Let $d_j(x) = a_j^T x - b_j$ and define $\mathcal{J}$ as the index representation of a polyhedron (Tøndel et al. (2003)) which consists of the corresponding HPs determined by d_j and their sign, e.g. $\mathcal{J} = \{2^+, 5^-\}$ means that $d_2(x) \geq 0$ and $d_5(x) \leq 0$. Such a set defines a polyhedron in the state space, i.e. $\mathcal{P}(\mathcal{J}) = \{x | d_2(x) \geq 0, \; d_5(x) \leq 0\}$. Further define the index sets $\mathcal{I}(\mathcal{J}) = \{i | \mathcal{X}_i \cap \mathcal{P}(\mathcal{J}) \text{ is full-dimensional}\}$ and $\mathcal{F}(\mathcal{I}) = \{k | \; f_k \text{ corresponds to } \mathcal{X}_i, \; i \in \mathcal{I}\}$ where $\mathcal{F}(\mathcal{I})$ denotes the index set of all functions corresponding to the index set $\mathcal{I}(\mathcal{J})$. We use the notation $'\pm'$ for statements which should be repeated for both $'+'$ and $'-'$.

Lemma 1. (Tøndel et al. (2003)) If $i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+)$ and $i \notin \mathcal{I}(\mathcal{J} \cup j^+)$, then $\mathcal{X}_i$ is split into two full-dimensional

polyhedra by hyper plane j, i.e. $i \in \mathcal{I}(\mathcal{J}) \cap \mathcal{I}(j^+) \cap \mathcal{I}(j^-)$. The same result holds when j^+ is interchanged by j^-.

Lemma 1 provides a computationally efficient approximation of $\mathcal{I}_k^{\pm} = \mathcal{I}(\mathcal{J}_k \cup j^{\pm})$ as $\mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^{\pm})$ when one builds the k-th node of the tree. Based on these definitions and results, the modified algorithm to build the truncated binary search tree is presented in Alg.1.

Let $\mathcal{M}_{clk}$ be the maximum admissible number of clock cycles allocated for online piecewise function evaluation in the processor. We will use $\mathcal{U}$ denoting the list of all indices of those search tree nodes which are currently unexplored. When exploring the k-th node of the tree $\mathcal{N}_k$, let $(\mathcal{I}_k, \mathcal{J}_k)$ be the corresponding index set of regions and hyper planes as already defined and $\mathcal{TBST}_k$ denoting the data structure corresponding to the k-th node of search tree. We introduce $\mathcal{C}_k = \mathcal{C}\{\mathcal{I}_k\}$ denoting the number of clock cycles which is required to find the optimal region through the candidates in the index set $\mathcal{I}_k$ by a combination of TBST traversal and direct search. Direct search means that the inequalities characterizing the polyhedral regions are evaluated explicitly one by one until the region containing the current state is identified. Then, one of the following events may occur during construction of the tree:

1. $|\mathcal{F}(\mathcal{I}_k)| = 1$, it means the exact solution is obtained and this node is flagged as a leaf node ($\mathcal{L}$).
2. $|\mathcal{F}(\mathcal{I}_k)| > 1$, $\mathcal{C}\{\mathcal{I}_k\} \leq \mathcal{M}_{clk}$ this means the online computational requirement is satisfied and it is possible to directly search through the members of $\mathcal{I}_k$ to find the optimal solution. This node is flagged as a truncated leaf ($\mathcal{TL}$), which is not pursued further in the BST construction.
3. $|\mathcal{F}(\mathcal{I}_k)| > 1$, $\mathcal{C}\{\mathcal{I}_k\} > \mathcal{M}_{clk}$ in this case the child node will be added to the tree and pursued further in the same way as in the BST method.

Algorithm 1: Building Truncated Binary Search Tree

1. Compute $\mathcal{I}(j^{\pm})$ for every $j \in J = \{1, ..., N_H\}$.
2. Initialize as $\mathcal{N}_1 \leftarrow \{(1, ..., N_r), \emptyset\}$ and $\mathcal{U} \leftarrow \{\mathcal{N}_1\}$.
3. **WHILE** $\mathcal{U} \neq \emptyset$ **DO**
   i. Select any $\mathcal{N}_k \in \mathcal{U}$ and set $\mathcal{U} \leftarrow \mathcal{U} \backslash \mathcal{N}_k$.
   ii. For all $j \in J$, compute $\mathcal{I}^{\pm} = \mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j^{\pm})$ and sort the candidate optimal HPs by $\max_{j \in J} (|\mathcal{F}(\mathcal{I}^+)|, |\mathcal{F}(\mathcal{I}^-)|)$, and collect the index set of all optimal HPs as J_k^*.
   iii. Compute the exact solutions $\mathcal{I}_k^{\pm} = \mathcal{I}(\mathcal{J}_k \cup j_k^{\pm})$ for all $j_k \in J_k^*$. This is done by solving two LPs $\min_{x \in \mathcal{X}_i} \pm d_{j_k}(x)$ for each $i \in \mathcal{I}(\mathcal{J}_k) \cap \mathcal{I}(j_k^+) \cap \mathcal{I}(j_k^-)$ considering Lemma 1. Choose any optimal solution $j_k = \arg\min_{j_k} \max(|\mathcal{F}(\mathcal{I}_k^+)|, |\mathcal{F}(\mathcal{I}_k^-)|)$, $\mathcal{I}_k^{\pm} = \mathcal{I}(\mathcal{J}_k \cup j_k^{\pm})$.
   iv. **IF** $|\mathcal{F}(\mathcal{I}_k^{\pm})| = 1$ **THEN**
   Flag the resulting new node $\mathcal{N}^{\pm}$ as a leaf node ($\mathcal{L}$), and put $\mathcal{N}^{\pm} \leftarrow \{\mathcal{F}(\mathcal{I}_k^{\pm}), 0\}$.
   v. **ELSE IF** $\mathcal{C}\{\mathcal{I}_k^{\pm}\} \leq \mathcal{M}_{clk}$ **THEN**
   Flag node $\mathcal{N}^{\pm}$ as a truncated leaf node ($\mathcal{TL}$), and put $\mathcal{N}^{\pm} \leftarrow \{\mathcal{F}(\mathcal{I}_k^{\pm}), -1\}$.
   vi. **ELSE**
   Put $\mathcal{N}^{\pm} \leftarrow \{\mathcal{I}_k^{\pm}, \mathcal{J}_k \cup j_k^{\pm}\}$ as left/right child of the current node and add $\mathcal{N}^{\pm}$ to the list $\mathcal{U}$ as new unexplored node(s).
   vii. **END IF**
4. **END EHILE**

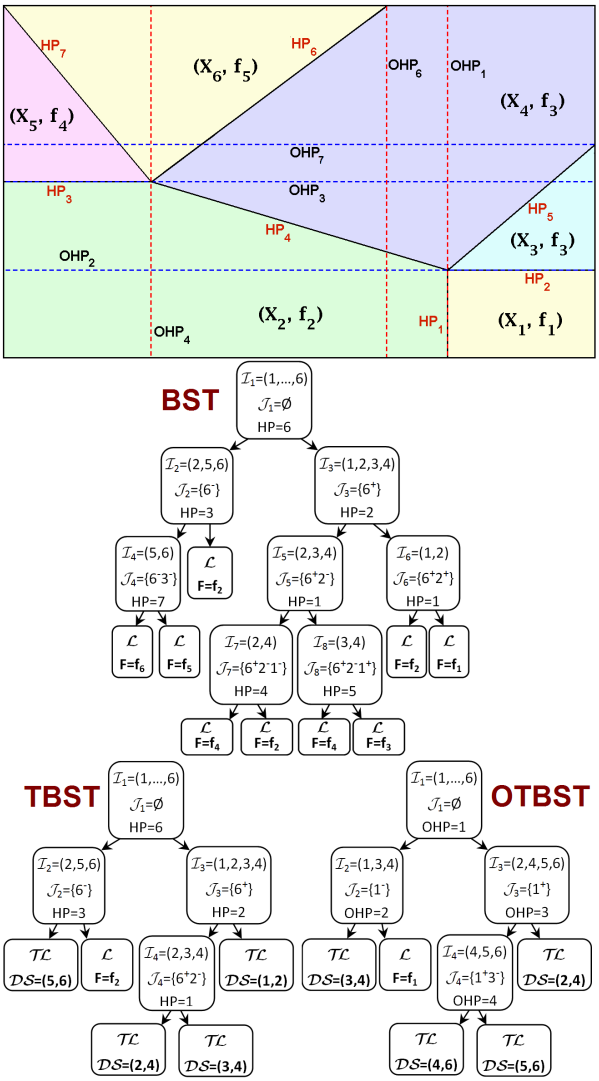

Fig. 1. Illustration of the BST, TBST and OTBST.

Note that 0 and -1 in steps 3.iv and 3.v denote the leaf and truncated leaf, respectively. The proposed TBST method in Alg.1 has been illustrated and compared to the BST approach in Fig.1.

Remark 1. The resulting data structure from Alg.1, i.e. $\mathcal{N}$, should be post-processed to build a compact search tree, i.e. $\mathcal{TBST}$, in which the first $n_x + 1$ numbers stores the information of optimal HP, i.e. (a_k, b_k), the next two numbers point to the left and right children of the current node respectively and the last two numbers point to the indices of left or right optimal candidate regions, respectively, if either left or right child is truncated leaf.

Remark 2. The maximum required number of clock cycles corresponding to the direct search is typically calculated as $\mathcal{C}_k = H_k^{DS}(n_x C_{Mul} + 2C_{Comp} + (n_x - 1)C_{Sum} + (n_x + 2)C_{Mem} + 2C_{Branch})$, where H_k^{DS} denotes the number of all HPs corresponding to the candidate regions in node $\mathcal{TBST}_k$ excluding the region with maximum number of HPs. This is due to the fact that after ordering the

$n_{\mathcal{TBST}_k}$ candidate regions in node $\mathcal{TBST}_k$ in terms of number of HPs, one needs to check only first $n_{\mathcal{TBST}_k} - 1$ candidates to find the optimal one. The numbers $(C_{Mul}, C_{Sum}, C_{Comp}, C_{Mem}, C_{Branch})$ represent the number of clock cycles required for performing multiplication, summation, comparison, memory access and branching operations, respectively, on the target processor.

Algorithm 2: Online Piecewise Function Evaluation
Input: Any feasible query point $x = [x_1, ..., x_{n_x}]^T$.

1. Start from the root node $\mathcal{TBST}_k \leftarrow \mathcal{TBST}_1$.
2. If $\mathcal{TBST}_k$ is leaf or truncated leaf, go to step 4.
3. Evaluate $d_k(x) = a_k^T x - b_k$ corresponding to the current node $\mathcal{TBST}_k$. If $d_k(x) \leq 0$ then take the right child as the current node, else take the left child as the current node. Return to step 2.
4. If $\mathcal{TBST}_k$ is leaf node, then retrieve the index of the optimal PWA function, else apply direct search to the set of candidate regions indicated by $\mathcal{TBST}_k$ to evaluate the optimal PWA function.

2.4 Orthogonal Truncated Binary Search Tree (OTBST)

In this section we further modify the BST method to achieve simpler structure of the search tree with considerably less pre-processing time. The main idea is to let the node decision criteria (HPs in TBST method) to be free (not restricted to the existing HPs). Although this can lead to a better solution, it can be expensive in terms of pre-processing time. So among all combinations of decision criteria we are interested in orthogonal hyper-planes. We select some candidate OHPs directly based on the information of the existing polyhedral regions. Such a set of candidate OHPs can be obtained by the following simple LPs for each region $\mathcal{X}_r \in X,\ r = 1, ..., N_r$:

$$\min_{x \in \mathcal{X}_r} \pm e_i^T x, \qquad i = 1, ..., n_x \tag{4}$$

where e_i denotes the $i-th$ basic unit vector. The LPs in (4) provide the Bounding Hyper-Rectangle (BHR) representation of each polyhedral region. So $2n_xN_r$ candidate OHPs are produced as a set of candidate discriminating hyper-planes which is extensively less than its counterpart in the BST and also TBST methods, i.e. N_H. In the next step one can further refine the set of candidate decision variables by removing very close OHPs. This relies on the fact that the present method allows the tree to end up with a truncated leaf instead of leaf which can contain more than one region. In Alg.3 the conceptual steps of building the proposed OTBST are presented. The main features of the proposed OTBST method can be summarized as follows. (i) The total number of discriminating OHPs is usually much less than the HPs in the BST and TBST methods, leading to a significant reduction in the pre-processing time, (ii) In the first step of building the search tree one has to determine for each node decision criteria (OHP), which side every region $\mathcal{X}_r$ lies on. This requires $2N_HN_r$ linear programs which can be replaced by simple comparison when OHPs are used (see Alg.3, step 2), giving further reduction in the pre-processing demands, (iii) Unlike the BST method, both TBST and OTBST approachs enable the designer to trade-off between offline and online complexities, and (iv) In the OTBST the online evaluation of each node is more efficient than the BST.

Algorithm 3: Building the OTBST
Input: Maximum admissible clock cycles $\mathcal{M}_{clk}$ and the minimum distance thresholds $\delta_i,\ \ i = 1, ..., n_x$.

1. Compute the initial set of candidate OHPs using (4) and store the upper and lower bounds of each region as $\mathcal{W}_r = [L_r\ \ U_r],\ r = 1, ..., N_r$. Refine the initial set by removing the redundant OHPs which are identical or closer than δ_i to any other OHPs.
2. Let $j \in \bar{J} = \{1, ..., \bar{N}_H\}$ be the set of refined OHPs defined by a pair (i_j, h_j) where $d_j = e_{i_j}^T x - h_j$. Then compute the index set $\mathcal{I}(j^{\pm})$ for every OHP by a simple comparison based on the available information from step 1, i.e. $\mathcal{W}_r,\ r = 1, ..., N_r$, as
   - **IF** $L_r^{ij} \leq h_j$ **THEN** add r to the index set $\mathcal{I}(j^-)$
   - **IF** $U_r^{ij} \geq h_j$ **THEN** add r to the index set $\mathcal{I}(j^+)$
3. Initialize the root node by setting $\mathcal{N}_1 = \{\mathcal{I}_1, \mathcal{J}_1\} \leftarrow \{(1, ..., N_r), \emptyset\}$ and $\mathcal{U} \leftarrow \{\mathcal{N}_1\}$.
4. The rest of algorithm is similar to the steps 3-4 in Alg.1 replacing HPs with OHPs and $\mathcal{TBST}$ with $\mathcal{OTBST}$, where $\mathcal{OTBST}_k\{1, 2\}$ store the information of optimal OHP in node k, $\mathcal{OTBST}_k\{3\ or\ 4\} > 0$ points to the left/right child, $\mathcal{OTBST}_k\{3\ or\ 4\} < 0$ denotes the leaf node with negative sign of corresponding function index, and $\mathcal{OTBST}_k\{3\ or\ 4\} = 0$ indicates the truncated leaf with the indices of candidate optimal regions stored in $\mathcal{OTBST}_k\{5\ or\ 6\}$.

For a given feasible query point x, the online procedure of piecewise function evaluation is explained in Alg. 4.

Algorithm 4: Online Evaluation Using OTBST
Input: Any feasible query point $x = [x_1, ...x_{n_x}]^T$.

1. Start from the root node $\mathcal{OTBST}_k \leftarrow \mathcal{OTBST}_1$.
2. If $x_{\mathcal{OTBST}_k\{1\}} \leq \mathcal{OTBST}_k\{2\}$, then $r \leftarrow \mathcal{OTBST}_k\{3\}$, else $r \leftarrow \mathcal{OTBST}_k\{4\}$. Go to step 2.
3. If $r > 0$ then $k \leftarrow r$ and go to step 2, else go to step 4 which means the associated child node is a leaf or truncated leaf node.
4. If $r < 0$ (leaf) then $-\mathcal{OTBST}_k\{3\ or\ 4\}$ denotes the index of the optimal PWA function, respectively, else apply direct search to the set of candidate regions indicated by $\mathcal{OTBST}_k\{5\ or\ 6\}$ to find and evaluate optimal PWA function.

2.5 An Approximation Approach

In this section, assuming eMPC applications with input constraints and soft-state (output) constraints we introduce a simple approach to approximate the PWA function piece in each truncated leaf. The idea is relied on the main observations of Johansen and Grancharov (2003), Kvasnica et al. (2010) and Christophersen et al. (2007b) where it is shown that with the price of sub-optimality it is possible to avoid storing the whole feasible partition. From the point of views of Johansen and Grancharov (2003) and Christophersen et al. (2007b), if the piecewise control law is continuous then relatively small regions have in practice little influence on the closed-loop stability and performance of the overall system, when perturbing or approximating the control law. Also the existence of measured noise in practice is another factor which makes the relatively small regions less important. Then, for a given truncated leaf, we propose the following algorithm to

approximate the PWA function. In any case, the approximation relies on appropriate scaling of the state space.

Algorithm 5: Approximation
Offline Procedure

1. Let $\mathcal{S} = |\mathcal{I}|$ denotes the number of candidate regions, then compute $\mathbf{R} = \{\mathcal{R}_1, ..., \mathcal{R}_\mathcal{S}\}$, $\mathcal{R}_i = \mathcal{P}(\mathcal{J}) \cap \mathcal{X}_{\mathcal{I}(i)}$, $i = 1, ..., \mathcal{S}$.
2. Compute and store the Chebyshev center and squared radius for all $\mathcal{R}_i$, i.e. $\mathbf{c} = \{c_1, ..., c_\mathcal{S}\}, \mathbf{r}^2 = \{r_1, ..., r_\mathcal{S}\}$, as presented in Bemporad et al. (2002b). The square is used to avoid online square root operation.
3. For $i \in \{1, ..., \mathcal{S}\}$, if $\mathbf{0}_{n_x \times 1} \in \mathcal{R}_i$ then set $c_i = \mathbf{0}_{n_x \times 1}$.

Online Procedure

1. Compute $\Delta x = \{dx_1, ..., dx_\mathcal{S}\}$, $dx_i = \|x - c_i\|_2^2$.
2. For $i \in \{1, ..., \mathcal{S}\}$, if $dx_i \leq r_i$ then $u(x) = f_{\mathcal{I}(i)}(x)$, else $u(x) = \left(\sum_{i=1}^{\mathcal{S}} (r_i/dx_i)^\alpha\right)^{-1} \sum_{i=1}^{\mathcal{S}} \left((r_i/dx_i)^\alpha f_{\mathcal{I}(i)}(x)\right)$, where α is a positive integer tuning parameter, e.g. $\alpha = 1$ or 2.
3. Apply an appropriate saturation to preserve input constraints, i.e. $\tilde{u}(x) = sat(u(x))$.

We remark, for online application one just needs to store Chebyshev centers and squared radiuses $(\mathbf{c}, \mathbf{r}^2)$ instead of storing all polyhedral regions. However, when the number of candidate regions in the truncated leaves increases, then the storage requirement for $(\mathbf{c}, \mathbf{r}^2)$ can be considerable and there might be no benefits for the approximation versus storing critical regions. To overcome this limitation, recently in Bayat et al. (2011a) by analytical representation of each truncated leaf an exact solution is proposed.

3. EXAMPLES

In this section the proposed general piecewise function evaluation methods (TBST and OTBST) as well as pure direct search and binary search methods (DS and BST) have been implemented and applied to some examples. The clock cycle specifications in the following simulations are chosen based on the typical embedded (micro) controllers where multiplication, summation, comparison, memory access and branching operations take $(2, 1, 1, 2, 2)$ clock cycle(s) to be carried out, respectively.

Example 1: Double Integrator
Consider the discrete-time version of the double integrator

$$x(t+1) = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x(t) + \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix} u(t) \qquad (5)$$

where $T_s = 0.05s$ and the system is subject to input constraints, $|u(t)| \leq 1$, and output constraint $|y(t) = x_2| \leq 1$. The CFTOC problem (2) is solved with $p = 2, Q = diag([1, 0]), R = 1, Q_f = 0_{2\times 2}$ for different horizon values $N = 2, 4, 8$ and 12. The explicit solutions characteristics and the comparison results of applying different PWA function evaluation methods are presented in Table 1. The pre-processing time in the OTBST method has been extensively reduced comparing to the BST method with moderate extra online computation time. However the online number of required clock cycles is guaranteed to be admissible while constructing the tree in the OTBST as well as the TBST methods using the parameter $\mathcal{M}_{clk}$. The off-line computations were done on a 3GHz Pentium IV computer with the aid of the MPT toolbox for Matlab Kvasnica et al. (2004). Considering the third test case $(N = 8)$, in Fig.2(a) it is shown that the proposed TBST method efficiently parameterizes alternative solutions with performance between the direct search (DS) and the binary search tree (BST) when the parameter $\mathcal{M}_{clk}$ decreases from its upper to lower value limited by the DS and BST methods. Also Fig.2(b) shows a similar trade-off property for the OTBST method. However, in this case there is no monotonic trade-off because of replacing the HPs in the BST with the artificial OHPs in the OTBST method. Consequently, even for very small $\mathcal{M}_{clk}$, most often there are some regions to be directly searched and it is not possible to achieve the BST performance. This fact can be seen from the right upper most subfigure in Fig.2 where decreasing the $\mathcal{M}_{clk}$ does not affect the online clock cycles. For the last test case $(N = 12)$ and using a low cost processor, e.g. AVR-XMEGA series, with 32 MHz clock frequency, the PWA function evaluation procedure in the BST, TBST, OTBST and Approximate-OTBST (AOTBST) methods, loads the processor with about 0.02%, 0.1%, 0.09% and 0.05%, respectively, when the sampling time $T_s = 0.05s$ is considered. In this case the direct search loads the processor with about 5%. For test case with $N = 8$, the closed-loop response for both exact solution and proposed approximation (Alg.5) are shown in Fig.3 for initial condition $x_0 = [-2 \ \ 0.9]^T$.

Table 1. The comparison results for example 1.

N	Method	N_r	$\mathcal{M}_{clk}$	N_n	Storage (Numbers)	Preprocessing Time(sec)	Online Clock-Cycles Min	Online Clock-Cycles Max
2	**BST**	83	—	188	752	27	147	187
	TBST		550	22	1096	11	107	513
	OTBST			33	1160	3	62	454
	AOTBST				862	3.5	62	242
4	**BST**	219	—	421	1792	160	147	207
	TBST		1000	33	2825	62	107	933
	OTBST			39	2890	6	73	952
	AOTBST				1779	7	73	474
8	**BST**	627	—	1177	4736	1321	187	247
	TBST		1250	84	8204	514	107	1227
	OTBST			90	8230	40	62	1165
	AOTBST				4070	42	62	619
12	**BST**	1325	—	2181	8704	5769	207	267
	TBST		1700	166	17202	2420	127	1655
	OTBST			131	17095	118	73	1531
	AOTBST				6794	121	73	793

Example 2: Ball & Plate
This example deals with a so-called Ball & Plate system of Borrelli (2003) in the form of regulating to the origin. The explicit controller has 2024 regions in 4 dimensions. The comparison results of applying different methods to this example are presented in Table 2 for $M_{clk} = 51e3$. At sampling time $T_s = 0.03s$ the PWA function evaluation task loads the processor with about 5.3% in the TBST method, while it takes about 4.9% (1%) when the OTBST (AOTBST) method is used. The direct search method takes 43% of the processor time in each sampling interval. Note that in the OTBST method the pre-processing time has been reduced more than 470 times and the storage requirements has been reduced about 30% comparing to the BST method. Furthermore, applying the approximation algorithm (Alg.5) leads to about 55% reduction in the

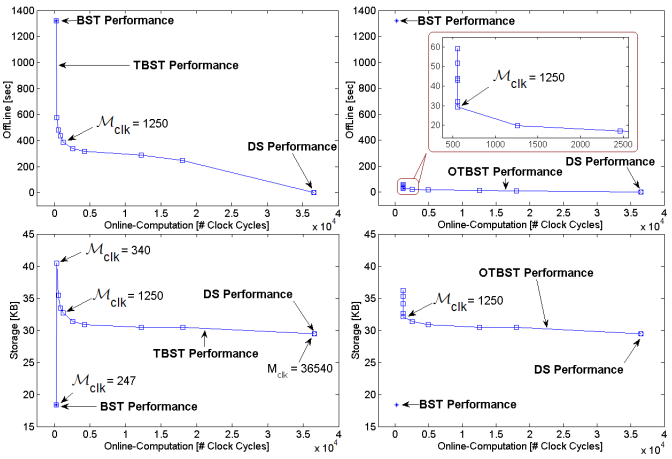

Fig. 2. Trade-off characteristics of the proposed methods, (a) TBST, and (b) OTBST (exact), in the worst cases.

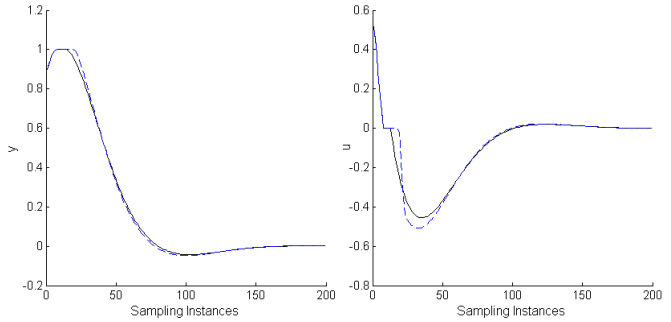

Fig. 3. The evolutions of output and input signals for example 1 with $N = 8$, exact solution (solid) and approximation (dashed).

storage requirement comparing to the BST, and the online computation time has been reduced about 80% comparing to the exact OTBST approach.

Table 2. The comparison results for example 2.

Method	N_r	$\mathcal{M}_{clk}$	N_n	Storage (Numbers)	Preprocessing Time(sec)	Online Clock-Cycles Min	Online Clock-Cycles Max
BST	2024	—	23514	141084	343297	373	613
TBST		51e3	35	88136	11591	223	50911
OTBST			69	91196	724	68	47370
AOTBST				63868	917	68	9761

4. CONCLUSIONS

The problem of evaluating general piecewise functions defined over polyhedral regions was investigated regarding its application in the embedded microcontrollers. The well known BST approach was modified in a way that by making use of available online computation time a truncated tree was generated leading to a considerable reduction in the pre-processing time and providing a trade-off between online and offline performances. This flexibility provides the user direct control of the use of embedded system processor resources such as processor time, and in some cases also online memory use, which is essential both in rapid prototyping design and production implementation. The simulation results showed that the proposed approaches open up the use of binary search trees to a wider class of problems since offline computations can be drastically reduced with the expense of some moderate online computational performance reduction.

REFERENCES

Baotić, M., Borrelli, F., Bemporad, A., and Morari, M. (2008). Efficient on-line computation of constrained optimal control. *SIAM Journal on Cont. and Opt.*, 47(5), 2470–2489.

Bayat, F., Johansen, T.A., and Jalali, A.A. (2011a). Flexible piecewise function evaluation methods with application to explicit model predictive control. In *IEEE International Conf. on Mechatronics*, Istanbul.

Bayat, F., Johansen, T.A., and Jalali, A.A. (2011b). Using hash tables to manage the time-storage complexity in a point location problem: Application to explicit model predictive control. *Automatica*, 47(3), 571–577.

Bemporad, A., Borrelli, F., and Morari, M. (2002a). Model predictive control based on linear programming - the explicit solution. *IEEE Trans. on Automatic Control*, 47(12), 1974–1985.

Bemporad, A., Morari, M., Dua, V., and Pistikopoulos, E.N. (2002b). The explicit linear quadratic regulator for constrained systems. *Automatica*, 38(1), 3–20.

Borrelli, F. (2003). Constrained optimal control of linear and hybrid systems. In *Lecture Notes in Control and Information Sciences*, volume 290.

Christophersen, F., Kvasnica, M., Jones, C., and Morari, M. (2007a). Efficient evaluation of piecewise control laws defined over a large number of polyhedra. In *ECC*, 2360–2367.

Christophersen, F., Zeilinger, M., Jones, C., and Morari, M. (2007b). Controller complexity reduction for piecewise affine systems through safe region elimination. In *46th IEEE CDC*, 4773–4778.

Ferreau, H., Bock, H., and Diehl, M. (2008). An online active set strategy to overcome the limitations of explicit mpc. *Int. J. Robust and Nonlinear Control*, 18, 816–830.

Johansen, T.A. and Grancharov, A. (2003). Approximate explicit constrained linear model predictive control via orthogonal search tree. *IEEE Trans. Automatic Control*, 48(5), 810–81.

Jones, C.N., Grieder, P., and Raković, S.V. (2006). A logarithmic-time solution to the point location problem for parametric linear programming. *Automatica*, 42(12), 2215–2218.

Kvasnica, M., Grieder, P., Baotić, M., and Morari, M. (2004). *Multi-parametric toolbox (MPT)*. Hybrid Systems: Computation and Control. Springe.

Kvasnica, M., Löfberg, J., Herceg, M., Cirka, L., and Fikar, M. (2010). Low-complexity polynomial approximation of explicit mpc via linear programming. In *ACC*, 4713–4718.

Tøndel, P., Johansen, T.A., and Bemporad, A. (2003). Evaluation of piecewise affine control via binary search tree. *Automatica*, 39(5), 945–950.

Wen, C., Ma, X., and Ydstie, B.E. (2009). Analytical expression of explicit mpc solution via lattice piecewise-affine function. *Automatica*, 45(4), 910–917.

Zeilinger, M., Jones, C., and Morari, M. (2008). Real-time suboptimal model predictive control using a combination of explicit mpc and online optimization. In *47th IEEE CDC*, 4718 – 4723.