Probabilistic Real–Time Guarantees: There is life beyond the i.i.d. assumption

Bernardo Villalba Frías, Luigi Palopoli, Luca Abeni, Daniele Fontanelli
University of Trento
Trento, Italy
{br.villalbafrias, luigi.palopoli, luca.abeni, daniele.fontanelli}@unitn.it

***Abstract*—A large class of modern real–time applications exhibits important variations in the computation time and is resilient to occasional deadline misses. In such cases, probabilistic methods, in which the probability of a deadline miss can be guaranteed and related to the scheduling design choices, can be an important tool for system design. Several techniques for probabilistic guarantees exist for the resource reservation scheduler and are based on the assumption that the process describing the application is independent and identically distributed (i.i.d.). In this paper, we consider a particular class of robotic application for which this assumption is not verified. For such applications, we have verified that the computation time is more faithfully described by a Markov model. We propose techniques based on the theory of hidden Markov models to extract the structure of the model from the observation of a number of execution traces of the application. As a second contribution, we show how to adapt probabilistic guarantees to a Markovian computation time. Our experimental results reveal a very good match between the theoretical findings and the experiments.**

I. Introduction

In the last years, probabilistic design has emerged as a viable option for a large class of real–time systems for which the strict respect of all the deadlines is not really needed. In many applications, the cost of overdesigning the system outweighs the possible advantages of respecting every single deadline. This is obviously true for multimedia applications (e.g., streaming or surveillance). As surprising as it may seem, the same applies to many robotic control applications, which are resilient to occasional and controlled timing failures [1], [2]. This opens the way for important opportunities to optimise the design of robotic systems and to reduce their costs.

Key to a provably correct probabilistic design of a real–time system is the availability of analysis algorithms that allow the designer to guarantee *probabilistic deadlines*. Contrary to a standard "deterministic deadline", a probabilistic deadline is associated with a probability that it will be respected. The flexible concept of probabilistic deadline allows the system designers to capture the multi–faceted requirements of time–sensitive applications and to reason about the impact of their scheduling choices on the performance of the system in stochastic terms.

The research leading to the results presented in the paper has been partially supported by the European Commission H2020 programme, through the ACANTO Research and Innovation Action, Grant number 643644.

In this setting, guaranteeing the analysis of probabilistic deadline amounts to finding if the deadline will be met with the desired probability for a candidate choice of scheduling parameters. In this paper, we deal with probabilistic guarantees for a class of real–time applications, which are at the heart of many modern robotic systems.

A. Related Work

Stochastic analysis of real–time systems has been proposed in different flavours. A number of results revolve around the popular notion of "probabilistic worst–case execution time (pWCET)" [3]. The idea is that for each execution, a task experiences a worst–case execution time (WCET), which changes depending on the input data set and on random effects in the system hardware (e.g., due to the presence of a cache).

The notion of the pWCET allows the designer to provide hard real–time guarantees through standard techniques (e.g., response time analysis [4]) and associate the result with a level of confidence given by the distribution of the pWCET. An example is the generalisation of the Real–time Calculus using pWCET presented in [5]. The pWCET can be derived from measurements, using statistical techniques grounded in the Extreme Value Theory (EVT) [6].

However, for the category of soft real–time applications considered in this paper, designers are interested in the QoS of the application (e.g., measured by the frequency of deadline misses) during every single run, rather than in the probability that for all possible runs a deadline will be violated.

In such situations, the notion of *probabilistic deadlines* [7] comes to the rescue. While a standard "deterministic deadline" must always be respected, a probabilistic deadline is allowed to be violated with a given probability. A probabilistic real–time guarantee is a statement that the deadline will be respected with a probability at least equal to the one specified in the probabilistic deadline.

The analysis of probabilistic guarantees is rooted into performance analysis and queueing theory [8]. Different techniques have been proposed for fixed–priority [9], [10], [11], [12], [13], [14] or Earliest Deadline First (EDF) schedulers [15]. A problem with fixed priority or EDF is that the stochastic model describing the task set is very complex and difficult to analyse because of the scheduling interference between tasks.

If reservation–based schedulers are used instead [16], each task executes as if on a dedicated computer and its behaviour can be analysed independently of the other tasks (temporal isolation property). This radical simplification makes the model easy to construct and to analyse, and brought different authors to develop efficient numerical methods [17], [18] or analytic bounds [19], [20], [21].

However, all the models proposed so far, for stochastic analysis of reservation–based systems, make the assumption that the computation time of the task is independent and identically distributed (i.i.d). While this assumption does not impair the application of the analysis in many cases of interest, this may not be true for many robotic applications. For instance, for the computer vision applications that mobile robots use to sense the environment, it can be argued that the computation workload depends on the complexity of the scene. Pictures shot in close sequence are likely to require the same computation time, which can change in magnitude when the robot moves towards an emptier area. This effect introduce a potentially strong correlation in the stochastic process describing the computation time, reducing the appeal of techniques developed for i.i.d. processes.

B. Paper Contribution

Randomised methods are a frequent encounter in robotics. In a general sense, we could say that randomised search is useful and beneficial every time a design or a decision space has to be explored without the possibility of exploiting the structure or the mathematical properties (e.g., complexity) of the model. In the wide class of robotic applications using randomised methods, we will consider a localisation method based on computer vision as a paradigm. In the experiments reported for the paper (and in many more, which are left out for the sake of brevity), the process describing the computation time is not i.i.d. and it shows an important correlation structure. Such a structure can be described by a Markovian model: the system operates in different modes and in each and every mode it generates computation times modelled as a random variable.

The first contribution of the paper is to show how to identify the different modes and the distribution of the random variable in each mode using the theory and the techniques developed for hidden Markov models (HMMs). The second contribution is the extension of the probabilistic guarantees for Resource Reservations, initially developed for the i.i.d. processes, to the case of Markovian computation time. Our results reveal a strict adherence of the theoretical results with the experiments.

Although the results of the paper are shown for a particular class of applications, our lab experience reveals that the applicability range of the technique is much larger and includes other classes of applications based on randomised methods (e.g., motion planning).

The paper is organised as follows. Section II offers a general introduction to the problem of probabilistic guarantees. Section III gives a general overview on randomised methods in robotics and offers more details on the specific application considered for the analysis presented in the paper. Section IV describes the specific type of stochastic process used to model computation time with Markovian switches. In Section V, we show the adaptation of probabilistic guarantees to the Markovian computation time. Section VI reports a number of experimental results showing the good match between theory and experimental data. Finally, in Section VII we state our conclusions and outline future work directions.

II. Probabilistic Guarantees for CPU Reservations

This section contains the basic definitions used in all the techniques that provide probabilistic guarantees for reservation–based systems and the resulting problem statement. This material is by and large translated from previous work [20] and reported here for the reader's convenience.

A. Task Model

In general, a real–time task τ_i consists of a sequence, possibly infinite, of jobs $J_{i,j}$ with $j \in \mathbb{Z}_{>0}$. Each job $J_{i,j}$ is activated at time $r_{i,j}$ and finishes at time $f_{i,j}$ after executing for an amount of time $c_{i,j}$. The computation time of each job, $c_{i,j}$, is assumed to be a stochastic process $\mathcal{C}_i$. Contrary to previous work, we do not assume here the process to be independent and identically distributed (i.i.d). Rather, we will assume it to be a Markov Modulated Process (MMP), as detailed below.

The job $J_{i,j}$ is also characterised by a deadline $d_{i,j} = r_{i,j} + D_i$ (where D_i is said relative deadline), that is respected if $f_{i,j} \leq d_{i,j}$, and is missed if $f_{i,j} > d_{i,j}$.

This paper is restricted to periodic tasks, meaning that two adjacent arrivals are T_i time units away from each other: $r_{i,j} = r_{i,j-1} + T_i$. Additionally, traditional hard deadlines $d_{i,j}$ are replaced by *probabilistic deadlines* [22]. A probabilistic deadline $(D_i,\ \beta_i)$ is respected if $\mathbf{Pr}\{f_{i,j} > r_{i,j} + D_i\} \leq \beta_i$. If $\beta_i = 0$, the deadline is considered hard.

B. The scheduling algorithm

In this paper, we will assume the use of a *reservation-based* scheduling strategy. Although any algorithm implementing CPU reservations could be used, we will adopt the Constant Bandwidth Server (CBS) [16], a scheduling policy based on the EDF strategy and available in the mainline Linux Kernel under the name of `SCHED_DEADLINE` [23].

A reservation is a pair $(Q_i^s,\ T_i^s)$[1], where Q_i^s is the amount of time that the task is allowed to use the resource within every reservation period T_i^s. The fraction of resource utilization dedicated to task τ_i is given by $B_i = Q_i^s/T_i^s$ and it is usually called *bandwidth*.

The CBS algorithm works by assigning dynamic *scheduling deadlines* to the tasks, and then, scheduling the tasks by applying EDF on the scheduling deadlines. If the tasks cannot

[1]The s superscript stands for "server" and it means that the two parameters are associated with the CBS, while and the i subscript refers to the task served by the CBS.

migrate between CPU cores, and for each core the parameters assigned to the different tasks are such that

$$\sum_i B_i = \sum_i \frac{Q_i^s}{T_i^s} \leq 1, \quad (1)$$

then each task is guaranteed to receive Q_i^s units of execution time within every reservation period T_i^s. This property is called *temporal isolation* and is key to the derivation of the stochastic model of the resource reservation presented in our previous work [20] and recalled in Section V.

C. Problem Statement

In view of the temporal isolation property, each task is guaranteed a minimum share of the processor Q_i^s/T_i^s independently of the behaviour of the other tasks (as long as Condition (1) is respected). We observe that the temporal isolation provided by the CBS is related to processor scheduling. In a complex multi–core environment, other types of interference between the execution of the tasks could come from conflicts on the memory bus or from cache–related delays. We will assume that such effects can be neglected and that the temporal isolation property can be applied in a broad sense.

In this setting, we will carry out a conservative analysis for each task assuming that it only receives its minimum guaranteed bandwidth with no interference from the other tasks. The result will be a lower bound for the probability of meeting the deadline. Since the evolution of each task is studied in isolation, we can safely remove the subscript i.

In this setting, our problem is formulated as follows.

Problem 1: Given a periodic real–time task with a stochastic computation time, find conditions on the reservation parameters (Q^s, T^s) such that the task respects the probabilistic deadline $(D,\ \beta)$.

III. Randomised Methods in Robotics

In the large class of robotic applications that can benefit from randomised methods, two deserve a special mention:

- computer vision algorithms used to extract information of interest from a scene captured using on–board cameras,
- path planning.

In the group of computer vision algorithms falls the popular RANSAC algorithm and its numerous derivatives [24]. This algorithm has been invented to interpret the sensed data in terms of predefined models, which typically correspond to known objects or landmarks.

Classic algorithms (e.g., based on least square analysis) consider all the data presented to the algorithm and are not able to single out and reject gross deviations from the model. Single gross deviations correspond to unexpected findings and are frequently encountered when travelling across an unknown environment. They are considered as "poisoned points" also for heuristic algorithms and can significantly impair the scene analysis.

To address the problem, the RANSAC algorithm proposes to select random points in the scene, instantiate the model on a subset of the data and measure the deviation. The procedure is repeated until a good consensus is reached between the selected subset and the model. This paradigm is easy to understand and implement, and it usually delivers excellent results in extracting the meaningful information from the scene.

In the class of the planning algorithms we find popular randomised methods such as Rapid–Exploring Random Tree (RRT) [25], or its recent development called RRT* [26]. The path planning problem is about finding a collision–free path that connects two points in the work space such that its curvature is required to be compatible with the kinematic and the dynamic constraints of the robot.

Both algorithms follow an iterative approach using a randomised search. For instance, RRT constructs two trees: one starting from the origin and one from the destination. In each iteration, a random number of points are selected and connected to the ones found at the previous step (cancelling the points that would determine a collision). The procedure stops when the two trees intersect. The algorithm selects the path along the tree with the minimum costs using a simple greedy heuristic. The advantage of using randomised algorithms is that we do not need any prior knowledge on the environment (e.g., concerning the presence of obstacles), which is "discovered" during the construction of the trees.

Vision and planning algorithms have to be executed in real–time while the robot is on the move. As an example, for a mobile robot of the size of a small vacuum cleaner, moving in a crowded space at 0.5 m/s, the planning can be safely executed three times per second (once every 400ms).

The vision algorithm is used to localise and, ultimately, to control the vehicle. Therefore, it needs to produce the data more frequently. Moving at the moderate speed of our example, 200 ms could be an acceptable sampling time. Clearly, if the speed of the robot increases, so the sampling frequency will also need to increase. Since it is a known fact that occasional timing failures can be acceptable for this type of systems [27], [1], we can easily trade such a possibility for a higher "average" sampling rate. However, an accurate assessment of the probability of these occasional failures is key to a correct design.

We now discuss in detail the application that we will use as case study for this paper and concentrate on the evolution of the computation time.

A. The lane detection algorithm

The lane detection algorithm summarised below is used in some mobile robot applications developed in our department (see Figure 1). The goal of the algorithm is to determine the position of the robot with respect to a line delimiting the lane (see Figure 2). More specifically, the expected output of the algorithm is the distance y_p of the centre of the vehicle from the line and the angle θ_p between the longitudinal axis of the car and the line.

The pair (y_p, θ_p) can be used to control the lateral position of the vehicle in the lane. Unsurprisingly, if the vehicle travels

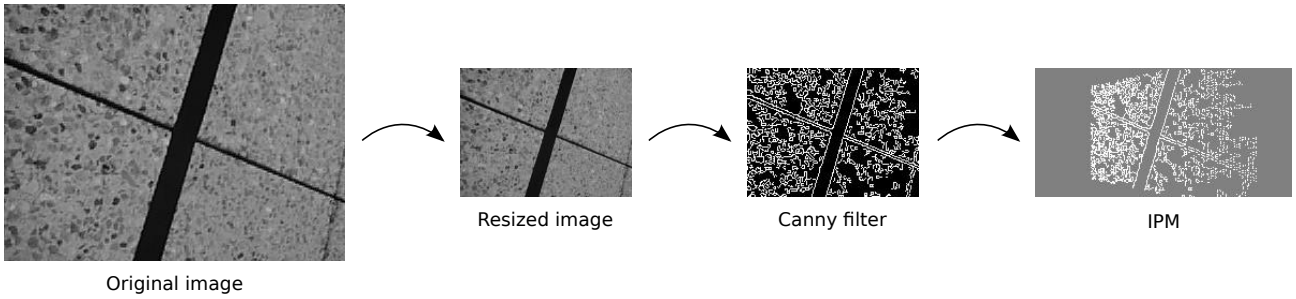

Figure 3. Sequential analysis of the image performed by the lane detection algorithm.

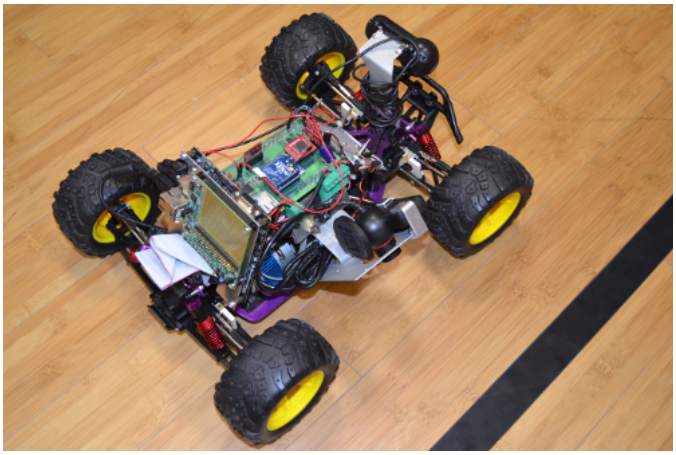

Figure 1. Robotic car used for the experiments. The lane detection algorithm is executed on the on–board platform.

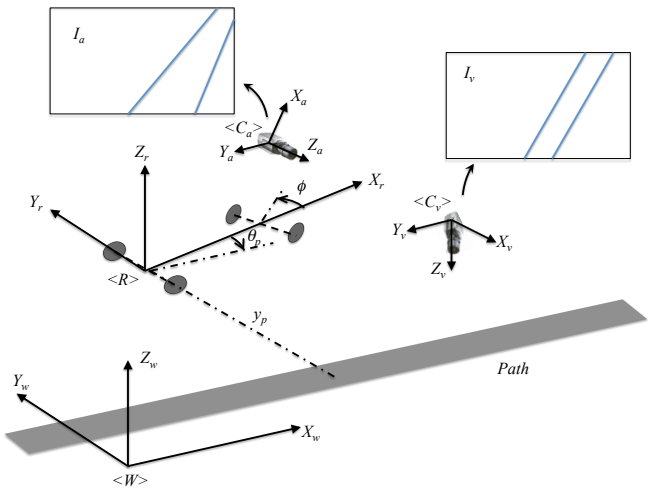

Figure 2. Vision system set–up: the real camera (looking at the road) and the virtual camera (looking from above)

at a high speed, the sampling period has to be rather small (although occasional deadline misses can be tolerated).

The algorithm is described in previous literature [28]. For the reader's convenience, we summarise here the main steps of the work flow:

1) The first step is a preprocessing of the image frame captured by the camera, in which the image size is reduced and the edges are detected to identify any relevant feature of the image (e.g. the line to be followed). Looking at the sample image in Figure 3 and moving from left to right, we can observe the image resizing and the canny filtering that extracts the edges, filtering out unnecessary elements.
2) Using an Inverse Perspective Mapping (IPM) technique, the scene observed by the "virtual" camera, flying above the scene, is reconstructed from the scene viewed by the actual camera. For the sample image in Figure 3, this corresponds to the fourth step.
3) By means of a RANSAC–based estimation algorithm, the position and orientation of the parallel lines in the "virtual" image are detected. The parallel lines sought in the image have to comply with an a priori model (e.g., the distance between the lines is known).
4) Finally, the position and orientation of the robot with respect to the actual line is computed. This information is then used to control the robot while moving in the environment.

The algorithm described above is suitable for real–time implementation even on a cheap hardware platform. For instance, we have compiled the algorithm for an ARM 9 platform and executed it on the WandBoard[2] mounted on the mobile robot. The robot executed a linear path in the laboratory and its task was to follow a black ribbon unfolded on the floor and used as a lane delimiter.

In Figure 4, we report an excerpt of the trace of the computation times for a sequence of 900 jobs. As we can see, the computation time fluctuates but it has recognisable trends, which are revealed by the moving average of 25 samples drawn in red and superimposed on the trace. Such trends are obviously reflected into the autocorrelation function plotted on the right part of the figure. By inspecting the autocorrelation

[2]www.wandboard.org

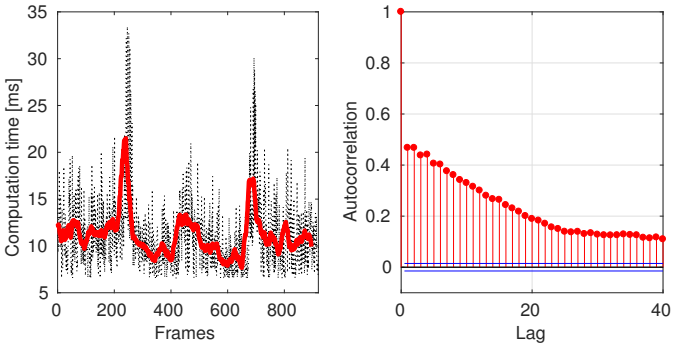

Figure 4. The left plot shows the computation times (dotted line) of the images taken by the robot while navigating the environment along with a superimposed (solid line) moving average to highlight the trend. The right plot shows the autocorrelation function of those computation times for different "lags" or time instants.

plot, we can see that a sample of the computation time is not only strongly correlated with the previous ones, but the correlation extends quite far into the past (the normalised value of the correlation remains greater than 20% even 20 samples behind).

In other words, the computation time of a job depends on the computation times of the previous jobs, hence it cannot be described using a simple probability distribution function $C(c) = \mathbf{Pr}\{c_j = c\}$, as done in previous works [22], [17], [20]. As a consequence, more advanced mathematical techniques have to be used to properly and more precisely describe the computation times. Such techniques will be introduced in the next section.

IV. The Markov Computation Time Model

The randomised nature of the algorithms described in Section III generates a random computation time even for multiple executions on the same input data. However, as long as the input data set has the same complexity, the computation time will be of comparable magnitude. When the input data changes (e.g., due to a change in the scene), the computation time is likely to grow or decrease (although there will still be significant random fluctuations).

This switching behaviour introduces correlation in the stochastic process that models the computation time, hence, as previously noticed, it is not possible to describe the computation times of a task using a simple probability distribution.

A possible way to work around this issue is to find some kind of structure in the correlation between computation times, and to describe such a structure with some stochastic model.

The switching behaviour highlighted above suggests that it could be possible to describe the computation times within the limits of a tractable model by describing the different operating conditions of the system with a finite number of states (N). In each of these different modes, the computation time is described by a random variable; if the computation times for each mode are independent and identically distributed (i.i.d.), then it is possible to describe such random variables by associating a probability distribution to each state.

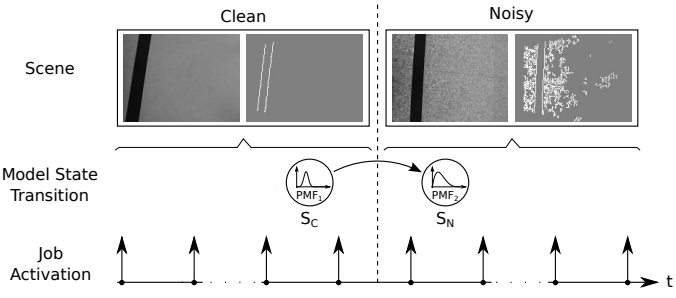

Figure 5. Example of scene variation with the corresponding mode change.

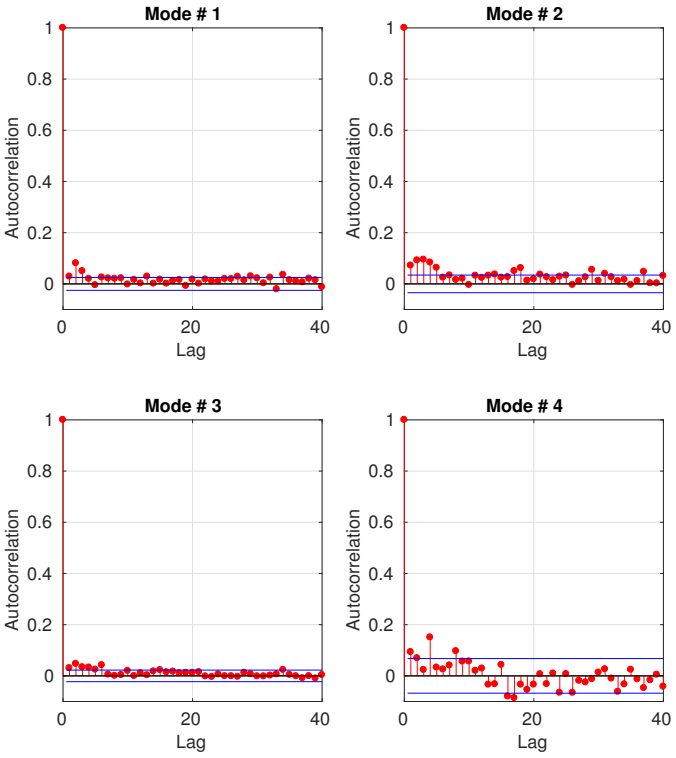

Figure 6. Autocorrelation function obtained from the 4 different modes in which the computation times presented in Figure 4 were classified.

This idea is well expressed by the lane detection algorithm reported in Figure 5. On the left hand side, we observe an execution of the algorithm on a "clean" frame. On the right hand side, we observe the execution of the same algorithm on a "noisy" frame, in which the computation time required to extract the lines is presumably much higher.

If the transitions between the different system states can be defined as a Markov process, the model just described corresponds to a particular type of non i.i.d. process that occurs very frequently in the applications, under the name of *Markov Modulated Process* (MMP) [29]. To be precise, since the state of the Markov chain is not directly accessible / observable (only the computation times are observable), the model can also be defined as a hidden Markov model (HMM). HMMs are very popular in several disciplines, such as economics and biology [30].

In this paper, we will show that a HMM can be used to model the evolution of the computation times for a large class of robotic applications, and that some probabilistic analysis developed for i.i.d computation times can be adapted to work with this non i.i.d model.

Given our specific application, we will name the model a *Markov Computation Time Model (MCTM)*. A precise definition is offered next.

Definition 1: A Markov Computation Time Model (MCTM) is defined as the triple $\{\mathcal{M}, \mathcal{P}, \mathcal{C}\}$, where

- $\mathcal{M} = \{m_1, \ldots, m_N\}$ is the set of modes;
- $\mathcal{P} = (p_{a,b})$, with a and $b \in \mathcal{M}$, is the mode transition matrix. The element $p_{a,b}$ defines the probability that, at the next step, the mode m_j will be b given that, at the previous step, m_{j-1} was equal to a:
$$p_{a,b} = \mathbf{Pr}\{m_j = b \,|m_{j-1} = a\};$$
- $\mathcal{C} = \left\{C_{m_j} : m_j \in \mathcal{M}\right\}$ is a set of distributions characterising the computation time in each mode. Each distribution is described by the Probability Mass Function (PMF) since the computation times can only take values that are multiples of the processor clock. Therefore, with this notation, $C_{m_j}(c)$ represents the probability that the computation time is equal to c when the system executes in mode m_j.

Hence, every mode has an associated probability distribution for the computation times, and the mode transitions determine changes in the distribution of the computation time. We will make the following (reasonable) hypotheses:

Assumption 1: For every job J_j, the computation time c_j is a random variable described by the distribution C_{m_j}, which only depends on the current mode m_j of the MCTM. The transitions between modes happen according to the probabilities $p_{a,b}$. Furthermore, the "mode change" event is independent both from the current computation backlog and from the computation time required by the previous execution.

Remark 1: The process governing the mode change is typically rooted in the physics of the system (e.g,, a robot travelling in a space and encountering an obstacle). Therefore, the assumed independence between the length of the computation time and the mode change is reasonable.

What is more, the mode change is generally asynchronous with respect to the start of the job, and it could, in reality, take place in the inter–sample period between the arrival of two jobs. However, for the purposes of the system, the mode change event has an effect on the computation time only in the next period (when a new sample is collected). Therefore, the synchronous assumption made above is also perfectly reasonable.

A. Estimating the parameters of the lane detection

In order to analyse the (probabilistic) schedulability of the system, it is important to accurately describe the computation times of a task using the MCTM; in other words, the values of $\mathcal{M}$, $\mathcal{P}$ and $\mathcal{C}$ must be identified. In literature, this is know as the *HMM Learning* problem: given a sequence of measured computation times, find the parameters of the HMM that describe such a sequence in the best way. More formally, find $\mathcal{P}$ and $\mathcal{C}$ that maximize the *Likelihood* (probability that a given sequence of values is generated by the identified HMM).

Fortunately, it is possible to re–use a huge corpus of powerful techniques that have been previously developed for working with HMMs: in particular, the HMM Learning problem can be solved by using the well–known Baum–Welch algorithm [31].

Given a sequence of observed computation times $c_0, \ldots, c_X$, the set of hidden states $\mathcal{M} = \{m_1, \ldots, m_N\}$, and the probabilities π_i to be in an initial state m_i, the Baum–Welch algorithm iteratively estimates the transition matrix $\mathcal{P}$ and the output matrix $\mathcal{C}$ starting from two initial guesses $\mathcal{P}^{(0)}$ and $\mathcal{C}^{(0)}$. The two initial guesses can be randomly generated, and the iteration will generally converge to the maximum likelihood matrices.

At each step of the iteration, the two estimations $\mathcal{P}^{(l)}$ and $\mathcal{C}^{(l)}$ are used to compute the probability $\xi_j(i,k)$ for the HMM to be in state m_i when job j arrives and in state m_k when job $j+1$ arrives, given the sequence of observed computation times $c_0, \ldots, c_X$. The probabilities $\xi_j(i,k)$ can then be used to compute the next estimations $\mathcal{P}^{(l+1)}$ and $\mathcal{C}^{(l+1)}$ and the iteration stops when the variations in the estimations are small enough, or when a maximum number of steps has been reached.

Once the state transition matrix and the output probability distributions have been estimated, they can be used as an input for the analysis technique described in Section V. However, it is first important to check if the identified MCTM correctly describes the computation times of the application. To do this, each observed computation time can be associated to a hidden state, so that the sequence $c_0, \ldots, c_X$ is split in N sub–sequences (one sequence per MCTM state), the independence of each sub–sequence can be tested, and the autocorrelations can be computed. This can be done by generating the most likely sequence of internal states for the identified MCTM given the computation times sequence $c_0, \ldots, c_X$. Such a problem is known as *HMM Decoding* problem in HMM literature, and can be easily solved by using the well–known Viterbi algorithm [32].

As an example, the Baum–Welch algorithm has been fed with the sequence of computation times measured for the previously described robotic application (see Figure 4). This allowed us to identify a 4–states MCTM (trying to identify an MCTM with more than 4 states resulted in duplicated states).

The Viterbi algorithm has then been used to identify the most likely sequence of internal (hidden) states corresponding to the sequence of observed computation times, so that it has been possible to associate each computation time to one of the 4 hidden states of the identified MCTM. This allowed us to partition the sequence of computation times in 4 sub–sequences, to perform a numerical test for independence [33] on such sub–sequences, and to compute the autocorrelation for each one of them.

The results are presented in Table I and in Figure 6, and

Table I
RESULTS OF THE NUMERICAL INDEPENDENCE TESTS FOR THE 4 SUB–SEQUENCES ESTIMATED BY THE BAUM–WELCH ALGORITHM. THE P–VALUE GREATER THAN 0.01 ALLOWS US TO ACCEPT THE INDEPENDENCE ASSUMPTION.

State	z–statistic	p–value
1	-0.6935	0.2440
2	-2.2925	0.0109
3	-1.0248	0.1527
4	-0.9297	0.1763

show that the computation times in each state are independent; hence, they can be correctly described by the identified output probability distributions, and the MCTM properly models the sequence of measured computation times.

V. MODEL ANALYSIS

In this section, we show the analysis of the timing evolution of a periodic task, with period T and computation time described by an MCTM, when it is scheduled through a CPU reservation with parameters (Q^s, T^s). For simplicity, we will assume that the server period T^s is chosen as an integer sub–multiple of the activation period T: $T = nT^s$, with $n \in \mathbb{N}$. Other choices are possible but make little practical sense.

First, we will show how the dynamic evolution of a CBS scheduler serving a task with computation time described by an MCTM can be described by a Markov chain. Then, we will discuss how it is possible to, efficiently, compute the steady state distribution of the probability of the states of the Markov chain. Finally, we will discuss how to compute the distribution of the response time of the task using the steady state distribution of the states of the Markov chain.

a) Dynamic Model: Let d_j^s denote the latest scheduling deadline used for job J_j, and introduce the symbol $\delta_j = d_j^s - r_j$. The latest scheduling deadline d_j^s is an upper bound for the finishing time of the job. If Equation (1) is respected, then $f_j \leq d_j^s$, i.e., a job always finishes before its latest deadline. Hence, δ_j is an upper bound for the job response time.

The quantity δ_j takes on values in a discrete set: the integer multiples of T^s and the probability $\beta = \mathbf{Pr}\{f_j \leq d_j\}$ of meeting a deadline $d_j = r_j + D$ is lower bounded by $\mathbf{Pr}\{\delta_j \leq D\}$: $\beta \geq \mathbf{Pr}\{\delta_j \leq D\}$.

The evolution of δ_j is described as follows [7]:

$$\begin{aligned} v_1 &= c_1 \\ v_j &= \max\{0, v_{j-1} - nQ^s\} + c_j \\ \delta_j &= \left\lceil \frac{v_j}{Q^s} \right\rceil T^s \end{aligned} \tag{2}$$

In the following we will use the function $[a]^+ = \max\{0, a\}$. It is useful to observe that the workload v_j is not directly measurable, as it can only be evaluated at the *end* of the job through the measurement of δ_j. Introduce the symbol $c_{m_j}^{\min}$ and $c_{m_j}^{\max}$, to denote the minimum and maximum computation time for mode m_j: $c_{m_j}^{\min} = \min\{c \text{ s.t. } C_{m_j}(c) \neq 0\}$ and $c_{m_j}^{\max} = \max\{c \text{ s.t. } C_{m_j}(c) \neq 0\}$, with $m_j \in \mathcal{M}$.

Let $c^{\min}$ and $c^{\max}$ be the minimum and the maximum computation time for all the different modes: $c^{\min} = \min\left\{c_{m_j}^{\min}\right\}$ and $c^{\max} = \max\left\{c_{m_j}^{\max}\right\}$. Since the computation time range in the bounded set $\{c^{\min}, \ldots, c^{\max}\}$ regardless of the mode of the MCTM, the value of v_j will be lower bounded by $c^{\min}$.

For i.i.d. computation times, the state of the system is entirely captured by v_j, and the system in Equation (2) is a Markov chain, which can be studied using the standard techniques for Probabilistic Guarantees [20].

For the case of MCTM computation times, we have to extend the state to consider the mode. Therefore, for the j^{th} Job, the state of the system is captured by the pair (m_j, v_j). Let us define the events $\mathcal{M}_g(j) = \{m_j = g\}$, meaning that the system is in mode g at job j, and $\mathcal{V}_h(j) = \{v_j = c^{\min} + h\}$, meaning that the workload is equal to $c^{\min} + h$ at job j, with $g \in \mathcal{M}$ and $h \in [0, \ldots, \infty[$. Hence, the state of an MCTM can be formally defined as follows.

Definition 2: The state of an MCTM, $\mathcal{S}_{g,h}(j)$, is defined as the intersection of the previously defined events: $\mathcal{S}_{g,h}(j) = \mathcal{M}_g(j) \wedge \mathcal{V}_h(j)$. In plain words, the event $\mathcal{S}_{g,h}(j)$ can be expressed as "the state (m_j, v_j) has the value $(g, c^{\min} + h)$".

For a generic couple of events A and B, by the definition of conditional probability, we have that $\mathbf{Pr}\{A \wedge B\} = \mathbf{Pr}\{B\}\,\mathbf{Pr}\{A\,|B\}$. The same property can be generalised if all probabilities are conditioned to a third event C: $\mathbf{Pr}\{A \wedge B\,|C\} = \mathbf{Pr}\{B\,|C\}\,\mathbf{Pr}\{A\,|B \wedge C\}$. Indeed, by the definition of conditional probability, for the right hand side, we can write:

$$\begin{aligned} &\mathbf{Pr}\{B\,|C\}\,\mathbf{Pr}\{A\,|B \wedge C\} \\ &= \frac{\mathbf{Pr}\{B \wedge C\}}{\mathbf{Pr}\{C\}} \frac{\mathbf{Pr}\{A \wedge B \wedge C\}}{\mathbf{Pr}\{B \wedge C\}} \\ &= \frac{\mathbf{Pr}\{A \wedge B \wedge C\}}{\mathbf{Pr}\{C\}} \\ &= \mathbf{Pr}\{A \wedge B\,|C\}, \end{aligned} \tag{3}$$

where the last step descends from the very definition of conditional probability.

Based on Equation (3) and on Definition 2, we can write:

$$\begin{aligned} &\mathbf{Pr}\{\mathcal{S}_{g',h'}(j)\,|\mathcal{S}_{g,h}(j-1)\} \\ &= \mathbf{Pr}\{\mathcal{M}_{g'}(j) \wedge \mathcal{V}_{h'}(j)\,|\mathcal{S}_{g,h}(j-1)\} \\ &= \mathbf{Pr}\{\mathcal{M}_{g'}(j)\,|\mathcal{S}_{g,h}(j-1)\} \cdot \\ &\quad \mathbf{Pr}\{\mathcal{V}_{h'}(j)\,|\mathcal{S}_{g,h}(j-1) \wedge \mathcal{M}_{g'}(j)\}. \end{aligned} \tag{4}$$

As a direct application of Definition 2, the first term of Equation (4) can be written as:

$$\begin{aligned} &\mathbf{Pr}\{\mathcal{M}_{g'}(j)\,|\mathcal{S}_{g,h}(j-1)\} \\ &= \mathbf{Pr}\{\mathcal{M}_{g'}(j)\,|\mathcal{M}_g(j-1) \wedge \mathcal{V}_h(j-1)\}. \end{aligned} \tag{5}$$

In view of Assumption 1, the mode change is independent of the computation time of the previous job. Hence, it is possible to write Equation (5) as:

$$\begin{aligned}&\mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\}\\&=\mathbf{Pr}\left\{\mathcal{M}_{g'}(j)\,|\mathcal{M}_{g}(j-1)\right\}\\&=\mathbf{Pr}\left\{m_j=g'\,|m_{j-1}=g\right\}\\&=p_{g,\,g'}.\end{aligned}\tag{6}$$

The second term of Equation (4) can be written as:

$$\begin{aligned}&\mathbf{Pr}\left\{\mathcal{V}_{h'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\wedge\mathcal{M}_{g'}(j)\right\}\\&=\mathbf{Pr}\left\{\mathcal{V}_{h'}(j)\,|\mathcal{M}_{g}(j-1)\wedge\mathcal{V}_{h}(j-1)\wedge\mathcal{M}_{g'}(j)\right\}\\&=\mathbf{Pr}\left\{v_j=c^{\min}+h'|\right.\\&\quad\left.m_j=g'\wedge m_{j-1}=g\wedge v_{j-1}=c^{\min}+h\right\}.\end{aligned}\tag{7}$$

In view of Assumption 1, the process modelling the sequence c_j of the computation time depends neither on v_j nor on m_{j-1}, but solely on m_j. Thereby, taking into account Equation (2), it is possible to write Equation (7) as:

$$\begin{aligned}&\mathbf{Pr}\left\{v_j=c^{\min}+h'\,\middle|m_j=g'\wedge m_{j-1}=g\wedge v_{j-1}=c^{\min}+h\right\}\\&=\mathbf{Pr}\left\{\left[c^{\min}+h-nQ^s\right]^+ + c_j=c^{\min}+h'\,|m_j=g'\right\}\\&=\mathbf{Pr}\left\{c_j=c^{\min}+h'-\left[c^{\min}+h-nQ^s\right]^+\,|m_j=g'\right\}\\&=\begin{cases}C_{g'}(c^{\min}+h') & \text{if } h\le nQ^s-c^{\min}\\C_{g'}(h'-h+nQ^s) & \text{otherwise .}\end{cases}\end{aligned}\tag{8}$$

As a result, by combining Equation (6) and Equation (8), we have:

$$\begin{aligned}&\mathbf{Pr}\left\{\mathcal{S}_{g',\,h'}(j)\,|\mathcal{S}_{g,\,h}(j-1)\right\}\\&=\begin{cases}p_{g,\,g'}C_{g'}(c^{\min}+h') & \text{if } h\le nQ^s-c^{\min}\\p_{g,\,g'}C_{g'}(h'-h+nQ^s) & \text{otherwise .}\end{cases}\end{aligned}\tag{9}$$

We can now introduce a vector containing all the probabilities of the different states. Let $\pi_{g,\,h}(j)$ be defined as $\mathbf{Pr}\left\{\mathcal{S}_{g,\,h}(j)\right\}=\mathbf{Pr}\left\{\mathcal{M}_g(j)\wedge\mathcal{V}_h(j)\right\}$. In plain words, $\pi_{g,\,h}$ represents the probability that at the j^{th} job the mode m_j be g and the workload v_j be $c^{\min}+h$. Introduce the vector $\Pi_h(j)=[\pi_{1,h}(j)\ \pi_{2,h}(j)\ \dots\ \pi_{N,h}(j)]$, which essentially represents the probability for the workload v_j to be $c^{\min}+h$ for all the possible modes of the MCTM. We can stack the vectors $\Pi_h(j)$ into a single probability vector $\Pi(j)$ composed by an infinite number of elements:

$$\Pi(j)=[\Pi_0(j)\ \ \Pi_1(j)\ \ \Pi_2(j)\ \ \dots].\tag{10}$$

The evolution of the vector $\Pi(j)$, presented in Equation (10), can be described using the standard notation of the Markov chains [34]: $\Pi(j)=\Pi(j-1)P$.

The matrix P is called transition matrix and describes the evolution of the probability of finding the system in each state starting from an initial distribution. In our case, the vector $\Pi(j)$, presented in Equation (10), is made of an infinite number of elements, and so will be the matrix P.

By applying Equation (9), we can see that the matrix has a recursive structure. To this end, introduce the notation $\alpha_{g,h}=C_g(c^{\min}+h)$, $R=c^{\max}-c^{\min}$, and $S=nQ^s-c^{\min}$. The matrix has the following block structure:

$$P=\begin{bmatrix}P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots\\P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots\\\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \dots\\P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \mathbf{0} & \mathbf{0} & \dots\\\mathbf{0} & P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \dots & \\\mathbf{0} & \mathbf{0} & P_0 & P_1 & P_2 & \dots & P_R & \mathbf{0} & \dots\\\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots\end{bmatrix},\tag{11}$$

where

$$P_e=\begin{bmatrix}p_{1,1}\cdot\alpha_{1,e} & p_{1,2}\cdot\alpha_{2,e} & \dots & p_{1,N}\cdot\alpha_{N,e}\\p_{2,1}\cdot\alpha_{1,e} & p_{2,2}\cdot\alpha_{2,e} & \dots & p_{2,N}\cdot\alpha_{N,e}\\\dots & \dots & \dots & \dots\\p_{N,1}\cdot\alpha_{1,e} & p_{N,2}\cdot\alpha_{2,e} & \dots & p_{N,N}\cdot\alpha_{N,e}\end{bmatrix},$$

and $\mathbf{0}$ denotes a block of zero of size $N\times N$.

b) Computation of the steady state probability: A few observations are in order:

1) Each row of the block matrix has at most $R+1$ non–zero blocks. Indeed, each new block is associated with a different value of the computation time and such values run from $c^{\min}$ to $c^{\min}+R=c^{\max}$;
2) As a consequence of Equation (9), starting from a value of the workload h smaller than or equal to S, the probability does not change with h but only depend on the workload h' after the transition; therefore, the first $S+1$ blocks of rows are repeated;
3) As a consequence of the same equation, starting from a value of the workload h greater than S, the probability is a function of the difference $h'-h$; therefore, from the block of rows $S+2$ onward, each block of row is obtained shifting the previous one to the right of one block and padding with a block of zeros $\mathbf{0}$.

In view of these considerations, the block matrix shown in Equation (11) has the periodic structure characteristic of a Quasi–Birth–Death Process (QBDP). A QBDP is generally described by the following transition matrix:

$$\begin{bmatrix}C & A_2 & \mathbf{0} & \mathbf{0} & \dots\\A_0 & A_1 & A_2 & \mathbf{0} & \dots\\\mathbf{0} & A_0 & A_1 & A_2 & \ddots\\\mathbf{0} & \mathbf{0} & A_0 & A_1 & \ddots\\\vdots & \vdots & \ddots & \ddots & \ddots\end{bmatrix}.$$

The four matrices C, A_0, A_1, A_2 describing the QBDP can be expressed in terms of the blocks P_e in Equation (11). In order to see this, introduce the symbol $P(p_z:p_Z\,;q_z:q_Z)$ to denote the submatrix obtained from P extracting the block of rows from p_z to p_Z and the block of columns from q_z to

q_Z. For example, $P(S+1:S+2\,;1:2)$ will denote the submatrix $\left[\begin{smallmatrix} P_0 & P_1 \\ \mathbf{0} & P_0 \end{smallmatrix}\right]$. By setting $F = \max\{S,\, R\}$, we have:

$$\begin{aligned} C &= P(1:F\,;1:F),\\ A_2 &= P(1:F\,;F+1:2F),\\ A_0 &= P(F+1:2F\,;1:F),\\ A_1 &= P(F+1:2F\,;F+1:2F). \end{aligned} \tag{12}$$

Through easy, but tedious, computations it is possible to show that by setting the matrices as in Equation (12), the transition matrix indeed reduces to the QBDP structure mentioned above. After casting our system into the QBDP framework, we can capitalise on the rich body of results in the field. In particular, following the same line of arguments as in [20], it is possible to show that:

1) The system admits a steady state
$$\begin{aligned} \overline{\Pi} &= [\overline{\Pi}_0,\, \overline{\Pi}_1,\, \ldots]\\ &= \lim_{j\to\infty} \Pi(j)\\ &= \lim_{j\to\infty} [\Pi_0(j),\, \Pi_1(j), \ldots], \end{aligned} \tag{13}$$
where $\lim_{j\to\infty} \Pi_h(j) = \lim_{j\to\infty} [\pi_{1,h}(j)\pi_{2,h}(j)\ldots\pi_{N,h}(j)]$.
2) The steady state distribution $\overline{\Pi}$ is unique and independent from the initial distribution $\Pi(0)$,
3) The computation of the steady state probability can be done using the very efficient numeric solutions available in the literature. In particular, this paper uses the Cyclic Reduction algorithm presented in [35] for the numeric solution, although the Logarithmic Reduction algorithm [36] is also suitable for this purpose.

c) Computation of the distribution of the response time: From the steady state probability $\overline{\Pi}$, it is possible to recover the steady state distribution of the variable δ_j, which as we said is an upper bound of the task's response time. The steady state CDF can be reconstructed using the following formula:

$$\begin{aligned} &\lim_{j\to\infty} \mathbf{Pr}\{\delta_j = \delta T^s\}\\ &= \lim_{j\to\infty} \mathbf{Pr}\left\{\left\lceil \frac{v_j}{Q^s} \right\rceil = \delta\right\}\\ &= \lim_{j\to\infty} \mathbf{Pr}\left\{\delta - 1 < \frac{v_j}{Q^s} \leq \delta\right\}\\ &= \lim_{j\to\infty} \mathbf{Pr}\{(\delta-1)Q_s < v_j \leq \delta Q^s\}\\ &= \sum_{h=(\delta-1)Q^s+1}^{\delta Q^s} \lim_{j\to\infty} \mathbf{Pr}\{v_j = h\}, \end{aligned} \tag{14}$$

where:

$$\lim_{j\to\infty} \mathbf{Pr}\{v_j = h\} = \begin{cases} 0 & \text{if } h < c^{\min}\\ \lim_{j\to\infty} \sum_{g=1}^{N} \mathbf{Pr}\{\mathcal{V}_{h-c^{\min}}(j) \wedge \mathcal{M}_g(j)\} & \text{otherwise .} \end{cases}$$

We can finally observe that $\lim_{j\to\infty} \sum_{g=1}^{N} \mathbf{Pr}\{\mathcal{V}_{h-c^{\min}}(j) \wedge \mathcal{M}_g(j)\} = \sum_{g=1}^{N} \lim_{j\to\infty} \pi_{g,h}(j)$. As discussed in Equation (13), each element $\sum_{g=1}^{N} \lim_{j\to\infty} \pi_{g,h}(j)$ of the sum can be extracted from the steady state distribution $\overline{\Pi}$.

By using the steady state distribution of the process δ_j, presented in Equation (14), it is possible to guarantee probabilistic deadlines. Indeed, a probabilistic deadline $(D,\, \beta)$ is guaranteed if $\lim_{j\to\infty} \mathbf{Pr}\{\delta_j \leq D\} = \sum_{h=1}^{D/T^s} \lim_{\delta\to\infty} \mathbf{Pr}\{\delta_j = \delta T^s\} \geq \beta$, where we made the natural assumption that D is chosen as a multiple of T^s.

VI. Experiments

In order to show the practical applicability of the presented approach, we have evaluated it on the real robotic application presented in Section III-A, where a robotic vision programme is used to identify the boundaries of the lane, estimating the position and orientation of a mobile robot while moving in a predefined track.

A. Experimental setup

The experimental setup consisted of a black ribbon placed on the floor creating a track of 37 meters long. The linear velocity of the robot was constant and set to 0.6 m/s. The robot executed 20 laps in the track, allowing us to capture, with a frame rate of 30 fps, a video stream containing the ribbon. This data set roughly consisted of 18400 frames.

Moreover, in order to test the vision algorithm under different conditions, there were created 2 distinct tracks with different floor characteristics, as presented in Figure 5.

The first track was considered a "clean" one given that the lane detection algorithm was able to remove all the undesired elements from the image. On the other hand, the second track was considered a "noisy" track because most of the undesired elements remain in the image after the preprocessing step. As expected, the computation times of the different tracks differ considerably.

As a result, the experiments were performed on 2 sets (1 "clean" and 1 "noisy") of 18400 frames each. These sets were used as input for several off–line runs of the task executing the vision algorithm.

All these runs of the lane detection algorithm were carried out using a WandBoard running Ubuntu. The version of the Kernel used (4.8.1) implements the CBS algorithm (under the name of SCHED_DEADLINE [23] policy) alongside the standard POSIX real–time fixed priority policies (SCHED_FIFO and SCHED_RR).

B. Experimental results

In order to collect statistics of the computation time associated with each data set, the task running the vision algorithm, scheduled with the maximum real–time priority (99 for SCHED_FIFO), was run 100 times for each data set. In this way, each run will consist of 18400 jobs, with its corresponding computation time, each one representing the processing of one frame from the data set.

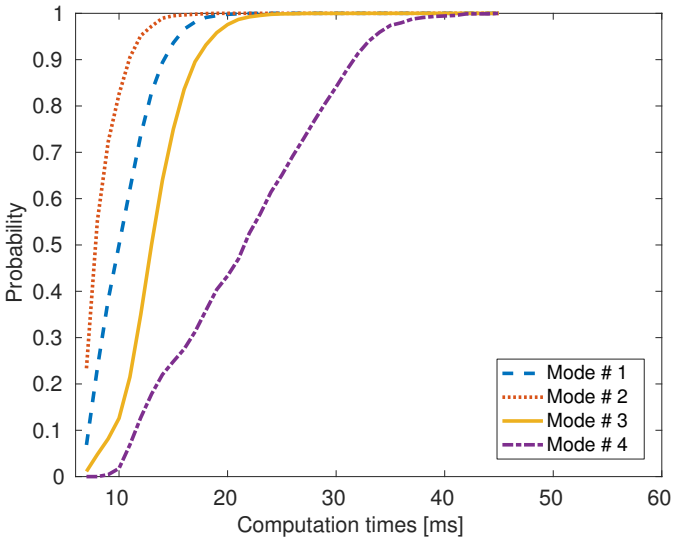

Figure 7. Cumulative distribution for the 4 different modes of the "clean" track identified by the Baum–Welch algorithm.

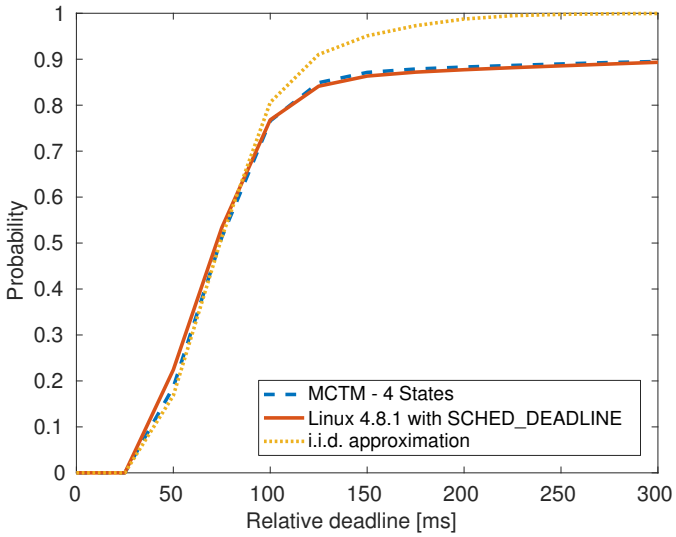

Figure 8. Probability of respecting the deadline, for the different methods, with a bandwidth fixed to 16% in the "clean" track.

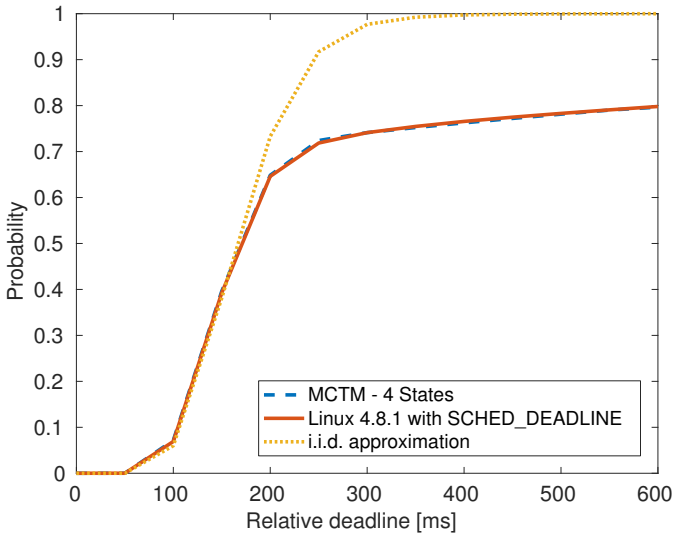

Figure 9. Probability of respecting the deadline, for the different methods, with a bandwidth fixed to 70% in the "noisy" track.

This group of 100 runs was divided into:

- Training set, which was analysed using the HMM identification techniques presented in Section IV-A to identify if a MCTM exists that fits the data, and
- Testing set, which was used to validate the results of the HMM training phase.

As mentioned in Section IV-A, the Baum–Welch algorithm was fed with different sequences of measured computation times taken from the training set. This process has been repeated multiple times, for different numbers of modes (ranging from 1 to 6) and using different numbers of measures for training the HMM. The algorithm identified 4 different modes for each track, which present independence in the computation times associated to them. Figure 7 presents the cumulative distribution function for each mode of the "clean" track.

In a second group of 50 runs, we have replicated a real–life condition. The vision algorithm was, in this case, executed in a periodic task, receiving as input the whole date set (18400 frames per track), and processing one frame every $T = 100$ ms. The task was scheduled using SCHED_DEADLINE, with server period $T^s = 25$ ms and budget $Q^s = 4$ ms for the "clean" track.

The measured probability of respecting a deadline, expressed as the ratio between the number of jobs that finished before the deadline over the total number of jobs, was averaged through the 50 runs and compared with the one estimated by the MCTM model and the simplified i.i.d model presented in [20]. Figure 8 presents such a comparison, and shows two important things: first of all, the results obtained with MCTM are pretty similar to the ones obtained by executing a real application with SCHED_DEADLINE. Then, the i.i.d. approximation results in an *overestimation* in the probability of respecting a deadline (around 5% for the task period and reaches 10% for 200 ms). Such an optimistic analysis can be dangerous when designing a real–time system. As shown, MCTM allows to avoid this error.

In the case of the "noisy" track, the scheduling parameters were $T = 200$ ms, with server period $T^s = 50$ ms and budget $Q^s = 35$ ms. Also in this case, as presented in Figure 9, we observe a good match between the proposed technique (labelled MCTM) and the experimental result. In this case, the error introduced by making the i.i.d. approximation is evidently much larger (around 10% for the task period, but it reaches 25% for a deadline equal to 400 ms).

Additionally, Figure 10 shows the accuracy of the MCTM approach when the relative deadline is fixed to the task period and different options of bandwidths are explored for the case of the "clean" track. Once again, the proposed approach shows a very good performance when compared with the real–life

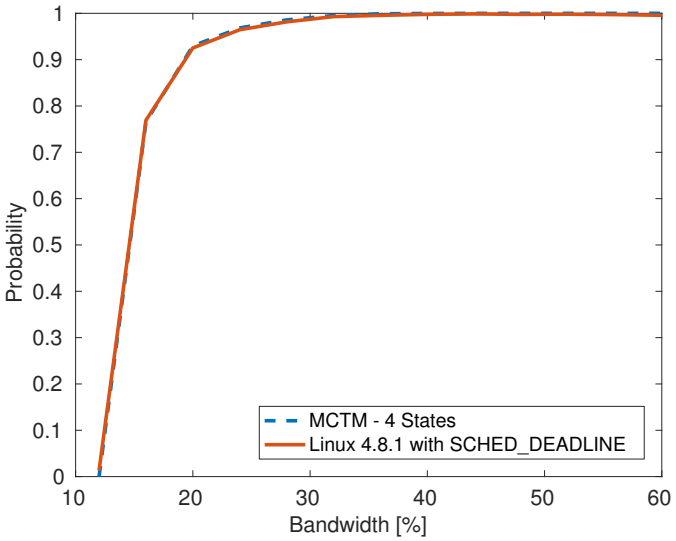

Figure 10. Distribution of the mean probability of respecting a deadline equal to the task period (100 ms) for different values of the bandwidth assigned to the task in the case of the "clean" track.

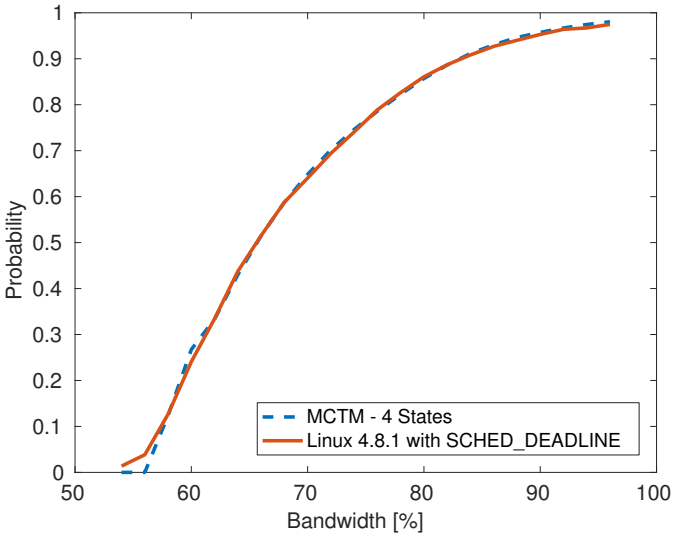

Figure 11. Distribution of the mean probability of respecting a deadline equal to the task period (200 ms) for different values of the bandwidth assigned to the task in the case of the "noisy" track.

application. Similar results were obtained for the "noisy" track, as shown in Figure 11.

The close match between the analysis and the experimental data is apparently the result of a good correspondence between the process expressing the computation time and the MCTM. For other applications, the MCTM fitting could not be so good or require a number of modes that is beyond the reach of our analysis tool. This evaluation is reserved for future investigations.

VII. Conclusions

In this paper, we have studied the best way to provide temporal execution guarantees to a large class of robotic

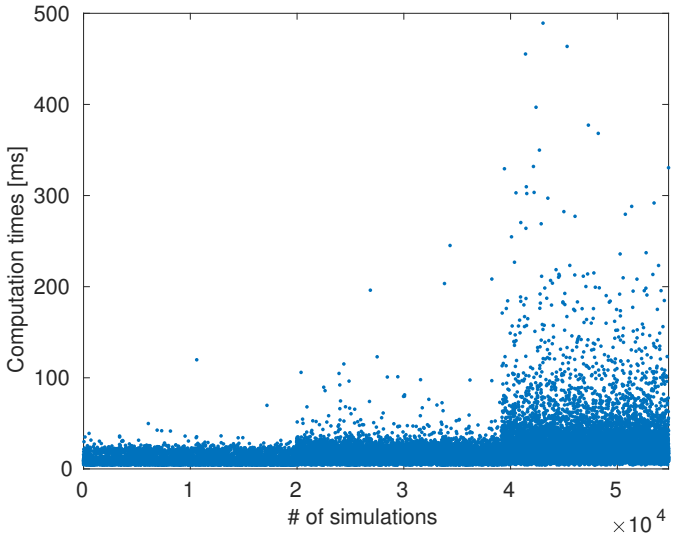

Figure 12. Computation times obtained from a RRT* motion planner.

applications using randomised methods. The wide variation in the computation time and the known resilience to occasional deadline misses discourage the use of standard hard real–time techniques. On the other hand, probabilistic guarantees have been developed for i.i.d. processes, while robotic applications exhibit a strong correlation structure depending on the conditions in which the robot operates. In such cases, a Markovian model that we have defined MCTM is a better fit for the process of the computation time.

We have shown a technique for the extraction of the parameters of the MCTM from a limited number of observation and shown how to adapt the techniques developed for probabilistic guarantees of resource reservations to MCTM computation time. The adherence between theory and experiments is very good.

In our future work, we plan to extend the analysis to other type of application, including visual tracking of moving objects and RRT* motion planning. Our preliminary data collected on the latter application show that this is indeed a promising work direction.

Consider the execution trace collected with a RRT* motion planner shown in Figure 12. We show the computation time for several executions collected from the planner in three different conditions corresponding to a different density of obstacles in the environment. We can see that the computation time:

1) Fluctuates (as is typical for randomised methods),
2) Depends on the complexity of the input data set, and
3) For a given operating condition, it is uncorrelated.

Under these premises, the computation time can be described by an MCTM when the robot is on the move.

These applications have the potential to present a larger variability in the computation time, leading to a considerable increment in the number of modes. The further analysis of these aspects is also relevant and we consider them as an interesting work direction.

A different work direction is to extend the analysis presented in the paper to the case of aperiodic tasks, which are a frequently encounter in robotic applications. Probabilistic guarantees for aperiodic tasks can be given in case of i.i.d. computation and inter–arrival times, as shown in our previous work [18]. We conjecture that the analysis presented in the paper could be extended to the case of Markovian computation and/or inter–arrival times following a similar approach.

Finally, since it is possible to construct online estimators for the hidden state of an MCTM based on the Viterbi algorithm, we believe that this possibility could be used to create adaptive schedulers in which the bandwidth reserved to the task changes depending on the estimated operating conditions.

REFERENCES

[1] A. Cervin, B. Lincoln, J. Eker, K. Arzen, and G. Buttazzo, "The jitter margin and its application in the design of real-time control systems," in *Proceedings of the IEEE International Conference on Real-Time and Embedded Computing Systems and Applications*. Gothenburg, Sweden, 2004.

[2] D. Fontanelli, L. Greco, and L. Palopoli, "Soft RealTime Scheduling for Embedded Control Systems," *Automatica*, vol. 49, pp. 2330–2338, July 2013.

[3] G. Bernat, A. Colin, and S. Petters, "pwcet: A tool for probabilistic worst-case execution time analysis of real-time systems," *REPORT-UNIVERSITY OF YORK DEPARTMENT OF COMPUTER SCIENCE YCS*, 2003.

[4] M. Joseph and P. Pandya, "Finding response times in a real-time system," *The Computer Journal*, vol. 29, no. 5, p. 390, 1986.

[5] L. Santinelli and L. Cucu-Grosjean, "A probabilistic calculus for probabilistic real-time systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 14, no. 3, p. 52, 2015.

[6] L. Cucu-Grosjean, L. Santinelli, M. Houston, C. Lo, T. Vardanega, L. Kosmidis, J. Abella, E. Mezzetti, E. Quinones, and F. Cazorla, "Measurement-based probabilistic timing analysis for multi-path programs," in *Proceedings of the Euromicro Conference on Real-Time Systems*, Pisa, Italy, July 2012.

[7] L. Abeni and G. Buttazzo, "Stochastic analysis of a reservation-based system," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium.*, San Francisco, California, April 2001.

[8] A. O. Allen, *Probability, statistics, and queueing theory*. Academic Press, 2014.

[9] M. K. Gardner and J. Liu, "Analyzing stochastic fixed-priority real-time systems," in *Proceedings of the 5th International Conference on Tools and Algorithms for Construction and Analysis of Systems (TACAS99)*. Springer, March 1999.

[10] J. L. Diaz, D. F. Garcia, K. Kim, C. G. Lee, L. Lo Bello, J. M. López, S. L. Min, and O. Mirabella, "Stochastic analysis of periodic real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 2002.

[11] J. L. Diaz, J. M. López, M. Garcia, A. M. Campos, K. Kim, and L. Lo Bello, "Pessimism in the stochastic analysis of real-time systems: Concept and applications," in *Proceedings of the IEEE Real-Time Systems Symposium*. IEEE, 2004.

[12] L. Cucu and E. Tovar, "A framework for the response time analysis of fixed-priority tasks with stochastic inter-arrival times," *ACM SIGBED Review - Special issue: The work-in-progress (WIP) session of the RTSS 2005*, vol. 3, no. 1, pp. 7–12, January 2006.

[13] G. A. Kaczynski, L. Lo Bello, and T. Nolte, "Deriving exact stochastic response times of periodic tasks in hybrid priority-driven soft real-time systems," in *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, Patras, Greece, September 2007.

[14] D. Maxim and L. Cucu-Grosjean, "Response time analysis for fixed-priority tasks with multiple probabilistic parameters," in *Proceedings of the IEEE Real-Time Systems Symposium*, Vancouver, British Columbia, Canada, December 2013.

[15] K. Kim, J. L. Diaz, L. Lo Bello, J. M. López, C. G. Lee, and S. L. Min, "An exact stochastic analysis of priority-driven periodic real-time systems and its approximations," *IEEE Transactions on Computers*, vol. 54, no. 11, pp. 1460–1466, 2005.

[16] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *Proceedings of the IEEE Real-Time Systems Symposium*, Madrid, Spain, December 1998.

[17] L. Abeni, N. Manica, and L. Palopoli, "Efficient and robust probabilistic guarantees for real-time tasks," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1147–1156, May 2012.

[18] N. Manica, L. Palopoli, and L. Abeni, "Numerically efficient probabilistic guarantees for resource reservations," in *Emerging Technologies Factory Automation (ETFA), 2012 IEEE 17th Conference on*, Sept 2012, pp. 1–8.

[19] L. Palopoli, D. Fontanelli, N. Manica, and L. Abeni, "An analytical bound for probabilistic deadline," in *Proceedings of the Euromicro Conference on Real-Time Systems*. Pisa, Italy: IEEE, September 2012.

[20] L. Palopoli, D. Fontanelli, L. Abeni, and B. Villalba Frías, "An analytical solution for probabilistic guarantees of reservation based soft real-time systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 3, pp. 640–653, March 2016.

[21] A. Mills and J. Anderson, "A stochastic framework for multiprocessor soft real-time scheduling," in *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*. Stockholm, Sweden: IEEE, April 2010.

[22] L. Abeni and G. Buttazzo, "Qos guarantee using probabilistic deadlines," in *Proceedings of the Euromicro Conference on Real-Time Systems*, York, England, June 1999.

[23] J. Lelli, C. Scordino, L. Abeni, and D. Faggioli, "Deadline scheduling in the Linux kernel," *Software: Practice and Experience*, vol. 46, no. 6, pp. 821–839, 2016, spe.2335. [Online]. Available: http://dx.doi.org/10.1002/spe.2335

[24] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[25] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2. IEEE, 2000, pp. 995–1001.

[26] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 1478–1483.

[27] D. Fontanelli, L. Palopoli, and L. Abeni, "The Continuous Stream Model of Computation for Real–Time Control," in *Proceedings of the IEEE Real-Time Systems Symposium*. Vancouver, Canada: IEEE, 4-6 Dec. 2013.

[28] D. Fontanelli, F. Moro, T. Rizano, and L. Palopoli, "Vision-based robust path reconstruction for robot control," *IEEE Transactions on Instrumentation and Measurement*, vol. 63, no. 4, pp. 826–837, 2014.

[29] W. Fischer and K. Meier-Hellstern, "The markov-modulated poisson process (mmpp) cookbook," *Performance evaluation*, vol. 18, no. 2, pp. 149–171, 1993.

[30] S. R. Eddy, "Hidden markov models," *Current opinion in structural biology*, vol. 6, no. 3, pp. 361–365, 1996.

[31] L. E. Baum, "An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes," in *Proceedings of the 3rd Symposium on Inequalities*, 1972, pp. 1–8.

[32] G. D. Forney, "The viterbi algorithm," *Proceedings of the IEEE*, vol. 61, no. 3, pp. 268–278, 1973.

[33] R. Liu, A. Mills, and J. Anderson, "Independence thresholds: Balancing tractability and practicality in soft real-time stochastic analysis," in *Proceedings of the IEEE Real-Time Systems Symposium*, Rome, Italy, December 2014.

[34] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[35] D. Bini, G. Latouche, and B. Meini, *Numerical methods for structured Markov chains*. Oxford University Press, 2005.

[36] G. Latouche and Ramaswami, "A Logarithmic Reduction Algorithm for Qouasi-Birth-Death Processes," *Journal of Applied Probability*, pp. 650–674, 1993.