Design And Verification of Autonomous System Controllers Under Timing Uncertainties

Bineet Ghosh
The University of North Carolina at Chapel Hill, Chapel Hill, USA.
bineet@cs.unc.edu

Abstract—Design and verification of autonomous system controllers is becoming increasingly challenging due to the heterogeneous nature of the implementation platforms. Further, autonomous systems are required to run several processes together on such platforms. Owing to all such difficulties, it is often hard or impossible to schedule all the processes in a timely manner—as a result, several processes might miss its deadlines—even though the controller has been designed with the requirement that all the deadlines are always met. However, it is often the case that the safety properties of the controllers are not violated even if the deadlines are not always met. This observation lets us develop a flexible design paradigm that is not overly conservative (i.e., the deadline is always met). In this paper, I propose a framework to identify deadline miss patterns that do not violate safety. Using such artifacts, one can propose correct-by-construction synthesis algorithm that can generate such schedules.

Index Terms—control, reachability, real-time systems, safety, weakly-hard systems, statistical hypothesis testing

I. INTRODUCTION

The core functionality of most emerging autonomous systems—like autonomous vehicles or robots—are implemented as a collection of feedback control loops. The design of such controllers is generally a two-step process, where each step is isolated from the other—often happening in parallel. The first step involves designing the controller by setting the appropriate parameters, sampling period and control laws. The second step involves implementing this controller on embedded controllers. This design paradigm results in clear separation of concerns between the controller designers and the embedded systems engineers, where the control designer designs the controller with some assumptions (such as fixed actuator to controller delay etc.), and the embedded systems engineers ensure that the assumptions hold true. However, with the increasing complexity of the autonomous systems, several controllers are required to be implemented on shared heterogeneous resources (such as multicore processors, GPUs, FPGAs etc.). As a result, the timing behavior of the controllers can be highly variable due to complex interference patterns. In particular, the requirement of meeting all deadlines might not be possible on such platforms.

In this work, I show that the requirement of having to meet all the deadlines is not necessary when considering "system level properties" like control safety [1], [2]. Towards this, a number of recent papers have addressed the problem of stability analysis of feedback control loops in the presence of deadline misses (due to timing uncertainties) [3]. In this paper, I address a different class of safety properties, viz., whether the system trajectory deviates too much from the nominal trajectory, with the latter computed for the ideal timing behavior. Consider the following example of an F1Tenth model car, as given in Fig. 1 [1]. The car’s steering angle and velocity are computed by a feedback controller, designed for the car to follow a predetermined path. Running the controller as designed, with no timing uncertainty, results in the nominal trajectory shown in black. Around this trajectory, there is a safety envelope shown in light blue, representing a safe space for the car to occupy without hitting any obstacles. Due to timing uncertainties in the implementation platform, the software task that computes the control inputs can miss the deadline imposed by the scheduler, resulting in deviation from the nominal trajectory. 100 such trajectories where the control task missed its deadline are shown in the figure—the green trajectories are safe, remaining within the safety envelope for the entire time horizon. However, the trajectory shown in red deviates too far from the nominal trajectory potentially resulting in a collision with an obstacle. This illustrates the importance of bounding deviations of control systems if timing variations in the control software may occur.

Contributions. Computing such quantitative safety properties, unlike stability, requires computing reachable sets [2], which are computationally expensive and computes conservative bounds. To alleviate these issues, I will propose a statistical approach, using statistical hypothesis testing, that can compute tighter bounds on the maximum deviation with probabilistic guarantees; since reachability based analysis can be computationally expensive and the results might also be conservative (i.e., the computed upper bounds are not tight) [1]. A short video presentation of the proposed statistical methods is available in a public drive

Fig. 1: Deviation in the path of an F1Tenth car due to timing uncertainty [1]
can schedule multiple processes, under timing uncertainties, that are guaranteed to be safe [4].

II. MAIN APPROACH

In this section, I will propose a statistical hypothesis testing based framework that can compute tighter upper-bounds on deviation, in reasonable time, with very a high probability. The approach is given in Fig. 2. The proposed approach has three major modules, namely Hypothesizer, Verifier and Refiner. Broadly speaking, the Hypothesizer module simply guesses an upper-bound on the maximum deviation. The Verifier module verifies if the guess is correct using statistical hypothesis testing. If the guess is correct, it is returned as an accepted answer. If the guess is incorrect, it generates a counterexample and sends it to the Refiner module. The Refiner module then updates the guessed deviation bound using the obtained counter example and returns it to the Verifier for re-verification. This process continues until a valid upper-bound is obtained. The Hypothesizer module generates a guess on the upper-bound by computing the maximum deviation from small number or random trajectories—where a random trajectory is a trajectory due to pattern of deadline misses. The Verifier module verifies the bound using statistical hypothesis testing. It formulates two hypotheses: the null hypothesis encoding that the bound is not acceptable, while the alternate hypothesis encodes that the bound is acceptable. Then gathering sufficient number of random trajectories (depending on the strength of guarantee required) it determines whether $H_1$ is accepted or not. If $H_1$ is accepted, the bound is returned as a valid answer. If not, it generates a counterexample, which is then used by Refiner to update the guess and sends it back for re-verification (the details are provided in [1]). Leveraging the artifact of computing upper bounds under timing uncertainties, one can schedule multiple processes on shared heterogeneous resources. As discussed, if the focus is on system level properties, it is not required to meet all the deadlines at all time-steps. Using this observation, [4] proposes a technique to schedule several processes on a shared resource.

III. EXPERIMENTAL RESULTS

I have implemented my approach as Python-based prototype tool, named StatDev. The tool and the models are available through a public GitHub repository sites.google.com/view/statdev. The approach was tested on four benchmarks, namely RC Network, Electric Steering, Unstable second-order system, and F1Tenth. The statistical approach computed significantly tighter upper bounds for all the benchmarks (compared to reachable set based approach), except RC Network, for which it computed comparable bounds. For all the benchmarks, the statistical approach took significantly less time. The bound was computed in less than 8s on a standard laptop, for all the benchmarks, while the reachable set based approaches timed-out (with a bound of 1hr) for Unstable second-order system and F1Tenth. The details of the result is given in [1, Table 1]. Consider the results obtained from Electric Steering in Fig. 3. The black line denote the nominal trajectory. The safety envelope, around the nominal trajectory, is given in cyan. That is, a trajectory is considered safe, iff, it’s with the cyan region. Few random trajectories with at most 3 consecutive deadline misses is given in black. With at most 3 consecutive deadline misses, no unsafe behaviors were observed. But, if the number of consecutive deadline misses increased by just one, that is, with at most 4 consecutive deadline misses, unsafe behaviors were witnessed. The red trajectory in plot shows one such unsafe behavior with at most 4 consecutive deadline misses. In particular, observe the red circle, showing the violating safety envelope. The trajectory, shown in red, would have been safe, if the behavior was within the red circle, with a red dot as its center. While the actual behavior went out of the red circle to the point marked with a bold cross.

REFERENCES


