Optimal scheduling for overtaking in a road with two lanes and one opponent

Aloysio Galvão Lopes

December 8, 2022

Contents

1 Problem definition

In general lines, we are interested in the problem of finding the best scheduling strategies for a given CPS (in this case a simulated car) given that the execution time of the tasks depend on the state of the environment.

To clarify this, we will define clearly the following: the environment, our concept of ‘best’ scheduling, the tasks and the chosen scheduling strategy.

1.1 Environment

The environment is composed of two vehicles in a circular road as shown in Figure 1. The dynamics of each vehicle are simulated by external code and the cpu for the ego vehicle, which outputs the control, is also simulated. The whole simulation is composed of steps, on each step, the physics simulation performs one integration step and the cpu simulation performs one computation step.

The road is divided in two lanes (inner and outer).

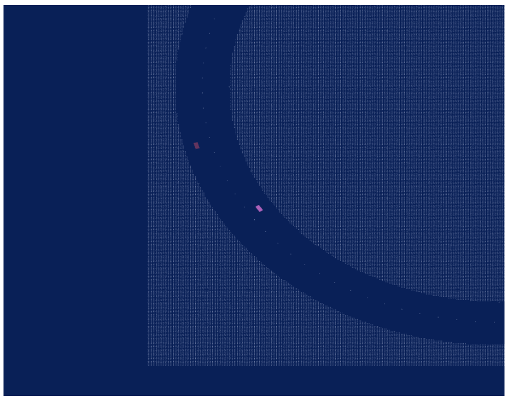

Figure 1: Circular road used in the simulation with two lanes and two vehicles.

1.1.1 Dynamics

The dynamics are simulated using *f1tenth_gym* [6], which implements the single track model from [1]. The model basically considers that the car is a bicycle, additionally, for low speeds, it considers only the kinematics.

1.1.2 CPU

The cpu simulator was developed by myself and simulates a simple monocore cpu. Periodic tasks are given to the cpu and for each step the scheduler assigns a task to the cpu. Each task is given a code and the code is executed at the end of the task if the task finished before its deadline. Figure 2 shows a diagram which explains graphically how the simulation is done.

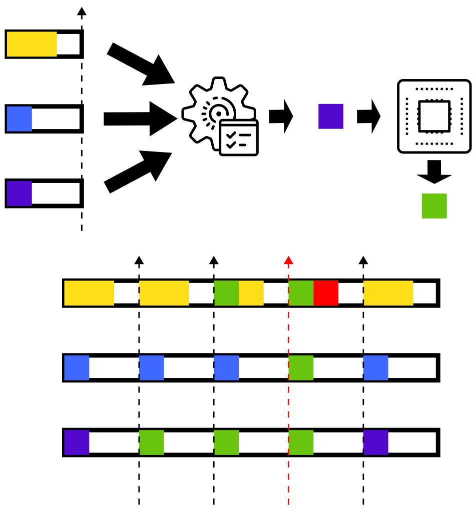

Figure 2: Three diferent periodic tasks (all with period and deadline 3) are simulated, each time step the scheduler code (the gear in the image) gives to the cpu the task it should execute. The simulated cpu adds execution units to the scheduled task. Finally, the red arrow shows the moment where the completed task's code was executed (in this case yellow lost its deadline and its code was not executed).

1.2 Tasks

Conventions:

- T_i is the period of the task τ_i.
- D_i is the deadline of the task τ_i.
- C_i is the computation time of the task τ_i.
- A_i is the arrival time of the task τ_i.
- the fundamental unit is the simulation step.
- the state S describes global variables accessible by any task.

For the moment, we use only periodic tasks with period $T = 6$ and implicit deadlines ($T = D$). Each task τ_i is described by the tuple (T_i, C_i, A_i). The tasks can communicate by updating a globally accessible state.

The following tasks are considered:

- Ego localization $\tau_{l_e} = (6, 1, 0)$: when done, updates the ego vehicle's localization in the state as well, as detects the current lane and updates the state with this information.
- Opponent localization $\tau_{l_o} = (6, 1, 0)$: when done, updates the opponent vehicle's localization (and current lane) in the state, and, if the vehicle is in a *danger situation* (to be defined further), updates the scheduler's mode.
- Control $\tau_c = (6, 2 - 3, 3)$: controls the ego vehicle with a controller to be described in detail later. If the vehicle is in a *danger situation*, the controller is changed to a *safe controller*. The *safe controller* takes 3 units of time instead of 2 in normal execution mode.
- Empty $\tau_e = (6, 2, 0)$: just wastes cpu time, used to simulate a non prioritary task such as communication for visualization purposes.

1.2.1 Controller

We use a simple PID controller to keep the ego vehicle in the middle of the inner lane. To make that possible, the PID controls the radius with regard to the center of the road using as output the steering angle of the vehicle. If the ego vehicle detects that the distance between itself and the opponent is less than a threshold and that the opponent is in the same lane, ego is in a *danger situation*.

When in a *danger situation*, the controller is changed to a *safe controller*, this controller keeps the same PID, but reduces the speed drastically. As stated before, the safe controller is supposed to take longer to execute.

1.3 The opponent

The opponent uses the same controller to follow lanes as ego, but it updates its control in every simulation step. Therefore, there is no cpu simulation for the opponent.

Also, the radius which it keeps can be changed so that it switches lanes. With that in mind, the opponent has a small probability p_ϵ of changing the controller's set point at each time step (in order to change lanes).

1.4 Definition of optimal probabilistic scheduling

Conventions:

- $DMP(\tau_i)$ is the deadline miss probability of the task τ_i.
- τ_{i} is critical if and only if $i \leq k$.

Given a set of tasks, we want the system to be able to schedule most of the time the critical tasks and, whenever possible, the tasks which are not critical.

With that in mind, a possible definition of optimal scheduling is one which minimizes the deadline miss probability of the non-critical tasks and, at the same time, ensures that the critical tasks are always scheduled respecting a given deadline miss probability. More formally, given a set of tasks $\mathcal{T}$, a scheduling strategy $\mathcal{S}$ is optimal for a monocore system if it is able to provide an order for the tasks to be executed such that (1) is minimal among any possible scheduler, respecting the constraints (2). In (2), p_i are the maximal deadline miss probabilities for each critical task.

$$\max_{\tau_i \in \mathcal{T},\ i>k} DMP(\tau_i) \tag{1}$$

$$\forall \tau_i \in \mathcal{T},\ i \leq k \implies DMP(\tau_i) \leq p_i \tag{2}$$

1.5 Scheduling strategy

For the moment, one basic scheduling strategy has been implemented. The objective is, in the end, to find an optimal strategy following the definition in Section 1.4. The current scheduler, however, does not respect the given definition.

In the current setting, our high priority tasks are τ_c, τ_{l_e} and τ_{l_o}, and the low priority task is τ_e. We wanted explicitly to find a situation where we needed to choose between running only the prioritary tasks (if τ_c exceeds) or running all tasks. Those two scenarios needed to be mutually exclusive, therefore if we are in a *danger situation* and we chose to run τ_e we would have to miss the deadline for τ_c. This situation is illustrated in Figure 3. We highlight that we chose to restrain ourselves to **non-preemptive** scheduling because the same problem would not arise in a preemptive scheduling as we show in Figure 4.

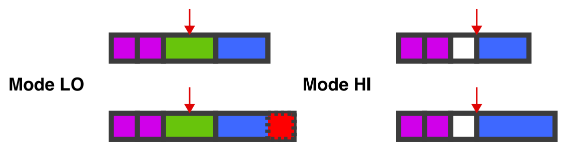

Figure 3: Four possible situations and two possible schedules (in modes HI and LO) defined by task priorities. The magenta tasks correspond to τ_{l_e} and τ_{l_o}, the green task corresponds to τ_e and the blue task corresponds to τ_c. The red arrow corresponds to the arrival of τ_c (all other arrivals are in the beggining of the period). The top line corresponds to normal operation and the bottom to a *danger situation*.

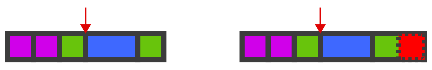

Figure 4: Same notation as in Figure 3 but with preemptive scheduling. Now, the prioritary tasks are always executed and the non-prioritary tasks are executed only if possible.

1.5.1 Ideal scheduler

Given this scenario, we can define our ideal scheduler. From now on, *danger situation* will be considered a random variable $\Gamma \in \{\top, \bot\}$, with $\Gamma \sim p_\Gamma$. We can express p_Γ as a function of the current state of the system S and the probability of lane change of the opponent p_ϵ.

The principle that we wish to study to build the ideal scheduler is the fact that we could express $DMP(\tau_i)$ as function of $p_{\Gamma|S}$. The fact that the distribution of Γ is conditioned on the current state of the system S is important because we could derive a better scheduler using the observations of the opponent's position and the ego current position.

1.5.2 Current scheduler

For the moment, in the last meeting, you have seen a scheduler with fixed priorities and two modes. The mode change is triggered by *danger situation*. The modes and priorities (from left to right) are:

- LO: τ_{l_e}, τ_{l_o}, τ_e, τ_c
- HI: τ_{l_e}, τ_{l_o}, τ_c

This allows us to have low priority tasks running only when the system is not in a *danger situation*.

1.6 TODOs

- Understand what is happening while we change modes.
- Create a scheduler that tries to achieve the optimality concept defined in Section 1.4 through mode changes.

2 Bibliography

The notion of optimality defined here was heavily inspired by the definitions present in [5]. The work in PROARTIS [3] [4] and[2] can be useful to define the relation between S and Γ in more complex scenarios via static analysis or measurements.

References

[1] Matthias Althoff, Markus Koschi, and Stefanie Manzinger. CommonRoad: Composable benchmarks for motion planning on roads. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 719–726.

[2] Guillem Bernat, Antoine Colin, and Stefan Petters. pWCET: A Tool for Probabilistic Worst-Case Execution Time Analysis of Real-Time Systems.

[3] Francisco J. Cazorla, Eduardo Quiñones, Tullio Vardanega, Liliana Cucu, Benoit Triquet, Guillem Bernat, Emery Berger, Jaume Abella, Franck Wartel, Michael Houston, Luca Santinelli, Leonidas Kosmidis, Code Lo, and Dorin Maxim. PROARTIS: Probabilistically Analyzable Real-Time Systems. 12:94:1–94:26.

[4] Liliana Cucu-Grosjean, Luca Santinelli, Michael Houston, Code Lo, Tullio Vardanega, Leonidas Kosmidis, Jaume Abella, Enrico Mezzetti, Eduardo Quiñones, and Francisco J. Cazorla. Measurement-Based Probabilistic Timing Analysis for Multi-path Programs. In *2012 24th Euromicro Conference on Real-Time Systems*, pages 91–101.

[5] Dorin Maxim, Olivier Buffet, Luca Santinelli, Liliana Cucu-Grosjean, Robert I Davis, and Nancy Grand-Est. Optimal Priority Assignment Algorithms for Probabilistic Real-Time Systems. page 10.

[6] Matthew O'Kelly, Hongrui Zheng, Dhruv Karthik, and Rahul Mangharam. F1TENTH: An open-source evaluation environment for continuous control and reinforcement learning. In *NeurIPS 2019 Competition and Demonstration Track*, pages 77–89. PMLR.