RTSS 2021 Industry Challenge

Perceptln
https://www.perceptin.io

2021

1 Background

The commercialization of autonomous machines, including mobile robots, drones, and autonomous vehicles *etc.*, is a thriving sector, and likely to be the next major computing demand driver, after PC, cloud computing, and mobile computing. However, autonomous machines are complex and safety critical systems with strict real-time and resource constraints.

Different from other computing workloads, autonomous machines have a very deep processing pipeline with strong dependencies between different stages and various local and end-to-end time constraints. Figure 1 shows an example of the processing graph of an autonomous driving system. Starting from the left side, the system consumes raw sensing data from mmWave radars, LiDARs, cameras, and GNSS/IMUs, and each sensor produces raw data at a different frequency. The processing components are invoked with different frequencies, performing computation using the latest input data and produce outputs to downstream components periodically.

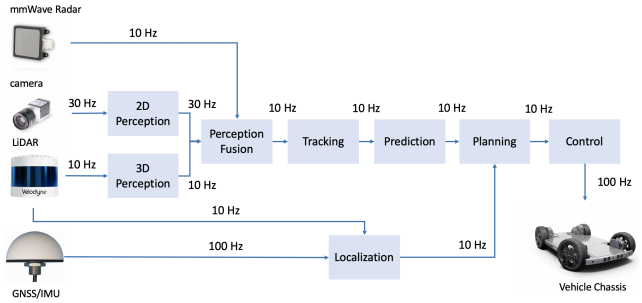

Figure 1: Processing Graph of An Autonomous Driving System

The cameras capture images at 30 FPS and feed the raw data to the *2D Perception module*, the LiDARs capture point clouds at 10 FPS and feed the raw data to the *3D Perception module* as well as the *Localization module*, the GNSS/IMUs generate positional updates at 100 Hz and feed the raw data to

the *Localization module*, the mmWave radars detect obstacles at 10 FPS and feed the raw data to the *Perception Fusion module*. The result of 2D and 3D Perception are fed into the *Perception Fusion module* to create a comprehensive perception list of all detected objects. The perception list is then fed into the *Tracking module* to create a tracking list of all detected objects. The tracking list then is fed into the *Prediction module* to create a prediction list of all objects. After that, both the prediction results and the localization results are fed into the *Planning module* to generate a navigation plan. The navigation plan then is fed into the *Control module* to generate control commands, which are finally sent to the autonomous vehicle for execution at 100 Hz. The computation components are deployed on different types of processing platforms. For example, the sensor data processing may be deployed on DSPs, the perception and localization may be deployed on GPUs as they typically require vector processing, and planning and control tasks can be deployed on CPUs as they mainly involve scalar processing. In general, the processing graph could be more complicated than the example shown in Figure 1. There could be more processing steps in the system. The activation frequency of each component could also be different from the example.

The system must comply with timing constraints in several aspects to guarantee that the final control command outputs can be executed correctly and timely. First, if there is some object appears close to the vehicle, it must be guaranteed that its related information can be perceived, processed and finally used to generate control commands with in a certain time limit. Second, the control command should be performed based on status information (e.g., the GNSS/IMU data) sufficiently fresh. Third, when some component receive data originated from different sensors, the time difference among the timestamps of the corresponding raw data must be no larger than a pre-defined threshold so that information from different sensors can be synchronized and fused.

2 Problem Model

In the following, we introduce a problem model based on the architecture and timing constraints of real-time computing systems for autonomous machines. Note that this abstract problem model could be significantly more complex than the processing systems that we already deployed in reality. However, studying the problem with a more general setting is meaningful to explore larger design space and deal with more complex systems that we may develop in the future.

The system consists of a number of tasks executing on a hardware platform consists of several processing units (e.g., CPU, GPU or DSP). Each task is statically mapped to a processing unit and the mapping is fixed in prior. The worst-case execution time (WCET) of each task on the corresponding processing unit is known in advance.

Each task is activated for execution according to a given frequency. The frequencies of different tasks are not necessarily harmonic. Each task reads input data tokens from one or multiple input ports, and produce output data

tokens to one output port. Tasks are connected with their input/output ports, with buffers of size 1 in between. The old data token in the buffer is over-written when a new data token is produced. Tasks reads and writes data from/to the buffers in a non-blocking manner. In each activation, a task reads the current data token in the buffer of each input port when it starts execution, and writes the output data token to buffer at its port at the end of its execution. We assume that the data communication delay between different tasks is zero or can be bounded by a constant (even if tasks are executed on different processing units).

Some task simply generates data tokens based on a given frequency (but do not read any input data token). We call such tasks the sensing tasks. Some sensor data are status sensing data and some sensor data are event sensing data, so the sensing tasks are classified into two types: status sensing tasks and event sensing tasks. Status sensing data are used for reporting the status. Event sensing data are used for detecting some event. When an event happens, all the output data token produced by the corresponding sensing task after that will capture this event.

When a task reads several input data tokens (from different input ports) and produces a output data token, we say the input data token is the "cause" of the output data token. The original sensor data that indirectly being the cause of a data token is the "source" of the data token. Each sensor data token is associated with a timestamp.

The system should satisfy the following timing constraints

- For each task, the difference of the timestamps among all its source data tokens must be upper-bounded by a pre-defined value.
- For any event, it must be guaranteed that the first final data output caused by the event sensor data capturing this event is produced within a pre-defined time delay after the event occurs.
- Let a be a sensor data token and b be the final data output indirectly caused by a. If b is produced at t, then a must be produced no earlier than a pre-defined value before t.

The problem to solve is to develop scheduling strategies and analysis techniques to guarantee the systems to meet all the above timing constraints. The scheduling strategy on all processing units must all be non-preemptive.

Note that we invite not only general solutions for the above described problem model, but also solutions for more restrictive versions based on the above described problem model. In other words, you may consider to add more constraints to the problem model if they are helpful to improve the real-time performance and/or simply the design and analysis of the problem (e.g., the relative relation of frequency of different tasks, the maximal number of tasks mapped to each processing unit, various structural constraints of the processing graph and so on). We are also open to possible collaboration with the solution provider to implement and evaluate their solutions with our realistic systems.