Embedded Control Systems 5LIJ0
Assignment 3: Controller Design over Distributed Architectures
March 2022

The purpose of this project is to design a controller for a dynamical system to meet a pre-defined set of requirements, considering its implementation constraints on a distributed embedded platform.

1. Lane Keeping Assist System (LKAS)

A platform architecture needs to be designed for LKAS. LKAS helps the car to stay in a lane, warns the driver when there is a dangerous situation, and activates emergency braking in case of an imminent collision.

The LKAS detects the lane and environment objects using the front camera. The gyroscope and speed sensor are used to compute the steering angle for lane keeping. Autonomous steering may be enabled to keep in between lanes. The driver is given real-time information on the dashboard.

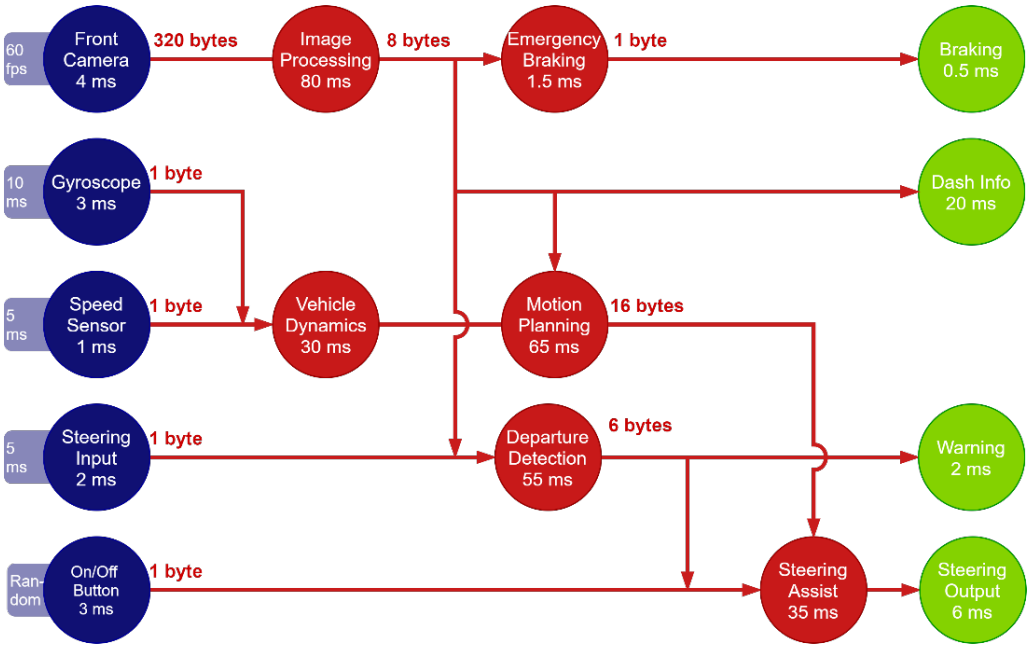


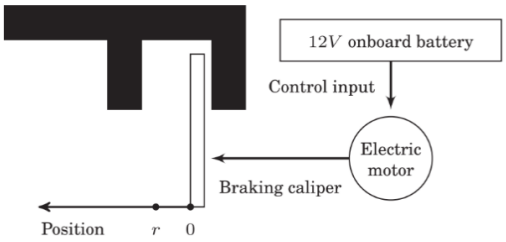
Figure 1. Task graph of LKAS

Figure 1 gives a simplified task graph description of the system. All tasks are annotated with their worst-case execution times (WCETs). Sensor tasks (in blue) are also annotated with their input rate. The directed edges of the graph model the dependencies between the tasks, and they are annotated with the message/data size. The details of these tasks are given in the tables below. The best-case execution times (BCETs) and WCETs of the tasks are computed for the running on the electronic control units (ECUs) mentioned in Section 4.

Sensor Task	Message Size	BCET	WCET	Explanation
Gyroscope	1 byte	2 ms	3 ms	The yaw rate is updated every 10 ms
Front Camera	320 bytes	3 ms	4 ms	The front camera has a frame rate of 60 fps
Speed Sensor	1 byte	1 ms	1 ms	The vehicle speed is updated every 5 ms
Steering Input	1 byte	1 ms	2 ms	Current steering angle is updated every 5 ms
On/Off Button	1 byte	2 ms	3 ms	The On/Off button can be randomly pressed and will cause the steering assist control task to execute
Control Task	**Message Size**	**BCET**	**WCET**	**Explanation**
Image Processing	8 bytes	70 ms	80 ms	Process the image given by the front camera to detect the lane and objects
Vehicle Dynamics	8 bytes	25 ms	30 ms	Compute the current trajectory of the car with the use of the yaw rate and speed
Motion Planning	16 bytes	60 ms	65 ms	Planning of desired trajectory based on vehicle dynamics and detected position in the lane
Departure Detection	6 bytes	50 ms	55 ms	Using current steering angle and the detected lane, detect if the car is moving away from the lane
Steering Assist	2 bytes	30 ms	35 ms	Compute the required steering angle to follow the desired trajectory, considering the steering angle input and any detected lane departure
Emergency Braking	1 byte	1 ms	1.5 ms	Send a braking message when an object is detected within certain distance
Actuating Tasks	**Message Size**	**BCET**	**WCET**	**Explanation**
Steering Output	-	4 ms	6 ms	Steers the car with the computed steering angle
Warning	-	1 ms	2 ms	Give a warning to the driver when the driver is departing a lane
Dash Info	-	15 ms	20 ms	Display Information of the detected lane
Braking	-	0.5 ms	0.5 ms	Braking action of the car

2. Control System: Emergency Braking

In addition to designing the platform architecture, the emergency braking controller also needs to be designed. The event chain for the emergency braking control system consists of the tasks (front camera, image processing, emergency braking, and brake).



System Description: In the above figure, a simplified model of the Electro-Mechanical Braking (EMB) system is shown. When the EMB is active, the braking caliper should reach a reference position r, which is at the braking disc, within the desired settling time. This is the position mode. The requirement for the settling time ensures the reactiveness of the system. After that, a certain force is applied in the force mode. The electric motor mobilizing the braking caliper is powered by the onboard battery, which has a voltage of 12V (see "Chang, Wanli, et al. "OS-Aware Automotive Controller Design Using Non-Uniform Sampling." ACM Transactions on Cyber-Physical Systems 2.4 (2018): 26." for more details).

In this project, we consider the position mode of the EMB system, which is of interest in several scenarios, such as braking, disc wiping, and pre-crash preparations. The state-space matrices of the dynamic system under consideration is given by:

$$A = \begin{bmatrix} -520 & -220 & 0 & 0 & 0 \\ 220 & -500 & -999994 & 0 & 2\times 10^8 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 66667 & -0.1667 & -1.3333\times 10^7 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 1000 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C = [0 \quad 0 \quad 0 \quad 0 \quad 1], \quad D = 0 .$$

It should be noted that there are five system states (state vector $x(t)$): motor current, motor angular velocity, motor angular position, caliper velocity, and caliper position. The control input $u(t)$ is the applied voltage on the motor. Initial value of all states is equal to zero, i.e., $x(0) = 0$. The control system under consideration aims to design input $u(t)$ such that the caliper position, i.e., output $y(t)$, follows a given constant reference $r = 0.002$m as the purpose of control. This is called a *regulation* problem.

3. Embedded Implementation of Emergency Braking

In our embedded implementation, the control application is composed of three application tasks (Figure 2), i.e., τ_s (*sensing*), τ_c (*computation*), and τ_a (*actuation*). The implementation of a control application is done by the execution of these three application tasks. The τ_s task is an image processing algorithm: front camera sensor task and the image processing task (Figure 1). The algorithm detects the regions-of-interest (RoID), processes each region (RoIP), and merges the data for each region (RoIM). RoID is done by the front camera sensor task. We assume that any image pre-processing is done by a dedicated

processor in the camera and the time for this is included in the RoID WCET. The image processing task in Figure 1 abstracts the RoIP and RoIM tasks. An example is given below(Figure 2, Image Processing Task): i) the regions-of-interest detection (RoID) for the image yields the blue boxes; ii) the blue boxes are then processed (RoIP); and iii) the processed data for the regions are merged (RoIM) to obtain the object tracking and the depth clustering cue.

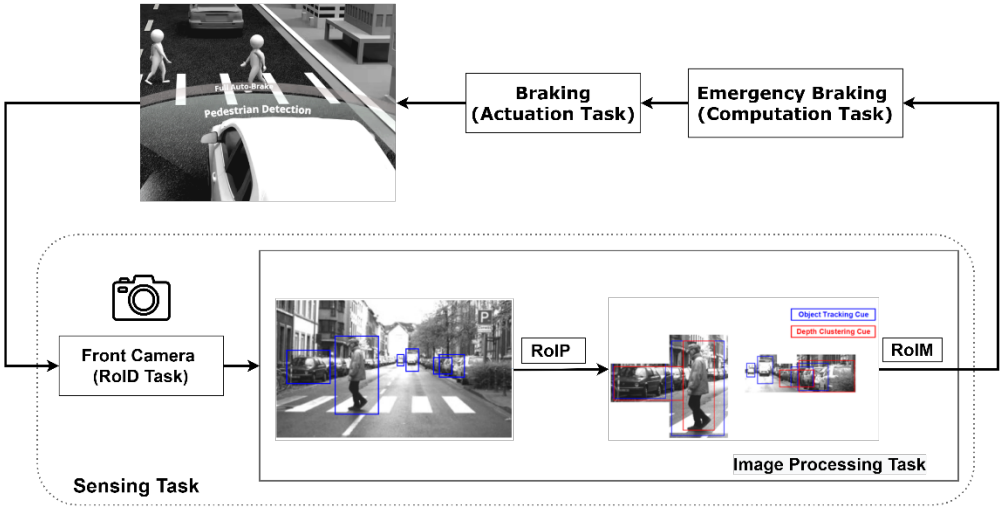


Figure 2.: Emergency braking application

A task graph description of the τ_s (*sensing*) task is given in Figure 3. This task graph implies that the RoIP task can be executed in parallel depending on the availability of processing resources/cores.

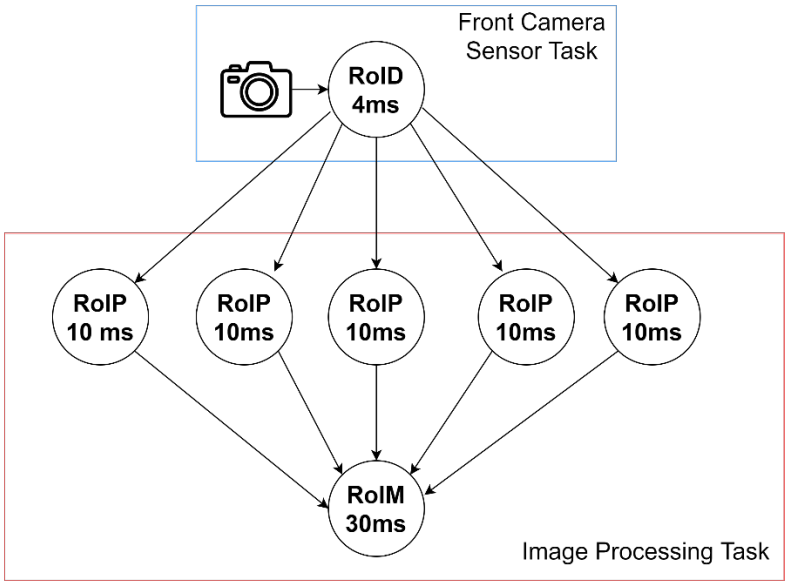


Figure 3. The task graph for the sensing task

Having more processing cores mean that you could choose to parallelize or pipeline the execution of sensing tasks and have a sequential or pipelined controller implementation.

We suppose that the τ_c (*computation*) and τ_a (*actuation*) tasks are the 'emergency braking' and 'braking' tasks respectively in Figure 1.

The two important parameters defined for the controller design are:

- Sampling period h: time between start time of two consecutive τ_s tasks
- Sensor to actuation delay τ: time between start of a τ_s task and the finish time of the corresponding τ_a task

4. Embedded platform

You have to design a platform architecture for implementing the LKAS system for the following cases:

- **Case I (10 points)**: The components you could use are electronic control units (ECUs) (generic OSEK, fixed priority pre-emptive; generic.ism with TDMA scheduling) and CAN busses (45 kbits/sec and/or 65 kbits/sec). The cost of a single OSEK ECU is 50 euros, generic TDMA-based ECU is 30 euros and for CAN bus is 200 euros for 45 kbits/second bus and 350 euros for 65 kbits/second bus. You could use either classic or FD frame types (max payload length = 64 bytes) for the CAN bus. The total cost of your platform should not exceed 850 euros. It is **required** that all sensor tasks (except on/off button) and the braking task use generic TDMA based ECU(s).
- **Case II (10 points)**: The components you could use are ECUs (generic OSEK, fixed priority pre-emptive; generic with TDMA schedule) and FlexRay bus (5 Mbits/second). The FlexRay parameters need to be defined or can be chosen by you. You can only use one FlexRay bus (with cost 450 euros). Use the same (or less) number of ECUs than that you used in Case I.

The ECUs are **synchronized** over a global clock. The ECUs have a fixed priority pre-emptive scheduler for generic OSEK and TDMA scheduler for generic.ism ECU. For Case I, the CAN busses are asynchronous with respect to the global clock for ECUs. For Case II, assume that FlexRay bus can be synchronized with the global clock for ECUs, if needed.

- **Trade-off Analysis (5 points)**: A comparison between the Case I and Case II design in terms of the sampling period h, sensor to actuation delay τ and the overall cost of the design based on the component selection should be reported.

5. Problem Definition

The problem is to design a discrete-time controller $u[k]$ such that all the following constraints are satisfied:

- $y[k] - r \rightarrow 0$ as $k \rightarrow \infty$.
- The settling time should be **as short as possible**.

This should be done by choosing the right electrical/electronic (E/E) architecture on which the task graph shown in Figure 1 is implemented. Use Inchron Tool Suite to model the architecture and obtain optimal sensor-to-actuator delay and sampling period for the controller meeting the following requirements using **chronSIM requirements evaluation**. The functional, safety, and manufacturing requirements (FR, SR, and MR) to be satisfied for the distributed LKAS implementation are given in the table below:

ID	Name	Text
FR.1	Working system	Every task in the LKAS system needs to take place eventually, maximum end-to-end response times of any possible trace (start of sensing to end of actuating) is 300ms
FR.2	Detecting lanes	The worst-case response time from receiving a camera frame to the end of the Image Processing task is 150ms

FR.3	Dash info	The end-to-end latency from a camera frame to the dash info is maximum 150 ms
FR.4	Dash info frequency	The dashboard should be updated at least every 150 ms
FR.5	Turning on	The car should steer within 80 ms from the moment the On/Off Button is pressed
SR.1	Departure warning	The end-to-end latency from a Steering angle input to the lane departure warning is maximum 80 ms
SR.2	Safe Steering	The worst-case response time of the Steering Output task is 10 ms
SR.3	Warning frequency	The warning should be updated at least every 170 ms
SR.4	Steering frequency	The steering should be updated at least every 170 ms
MR.1	Max cost	The cost of the Lane Keeping Assist System cannot exceed €850

For all the Cases, i.e., Case I and Case II, the **design objective** is to derive a settling time **as short as possible** for the emergency braking controller and **satisfy all the requirements** of the distributed system implementation (in the tables above).
For satisfying the requirements, you could use chronSIM requirements evaluation. Further, the chronVAL validation of your inchron models should not give **"infinite"** response times for the individual tasks.

6. Questions to be answered in this project and deliverables

1. Report **Group_no_project3.pdf** answering the following questions with justifications and explanations.
   - **Cases I and II**: design, mapping, results, and analysis (Putting some comments on the obtained results).
   - Comparison of **trade-offs** between **Case I and Case II** designs.
2. Inchon models (for each case) for the E/E architecture, i.e. **group_no_Case_i.ipr** (two *.ipr files, one for each case). Make sure that the models have all the requirements (at least one event chain per requirement) included in it, so that by running chronSIM you can immediately verify if the requirements are met or not.
3. Corresponding MATLAB scripts (for each case) for the controllers to obtain the control/braking performance, i.e. **group_no_Case_i.m** (two *.m files, one for each case).
4. Upload the above documents as a zip file, **Group_no_project3.zip.**

7. Provided materials

Inchron installation instructions and tutorial would be given to you:

1. **"Inchron Installation Instructions.pdf"**: It has the installation instructions for using Inchron Tool Suite.
2. **"Inchron Tutorial.pdf"**: It has the instructions to get started with Inchron. Use the basic knowledge from the tutorial to model the setup asked in this assignment.
3. **"Inchron tutorial videos"**.