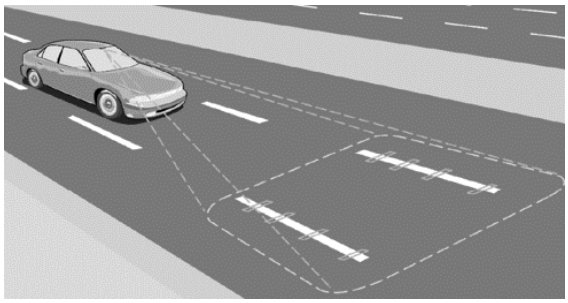
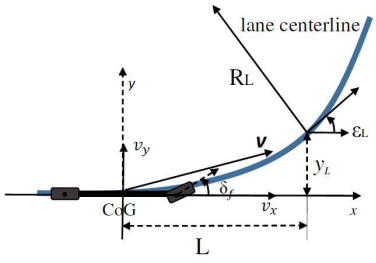
Embedded Control Systems 5LIJ0
Assignment 2:
Data-intensive Control
March, 2022

The purpose of this project is to design a controller for a dynamical system to meet a predefined set of requirements, considering its implementation constraints on a multi-processor platform.

1. Control System





v_x: vehicle longitudinal velocity
v_y : vehicle lateral velocity
R_L: curve lane radii
L : look-ahead distance at centre of gravity of vehicle (CoG)
y_L: lateral deviation at L
ε_L: angle between the tangent to lane centerline and vehicle orientation
δ_f: front tire steering angle

System Description: We consider the case study of a vision-based lateral control system model, commonly referred to as a lane-keeping assist system (LKAS). Our camera sensor captures the image stream at a fixed frame rate per second, e.g., 100 fps. The tasks in our LKAS - compute-intensive image sensing and processing, control computation, and actuation - need to be mapped to run onto a multiprocessor platform. The state-space matrices of our LKAS derived from **J. Kosecka, R. Blasi, C. Taylor, J. Malik, "Vision-based lateral control of vehicles", in Proceedings of Conference on Intelligent Transportation Systems, IEEE, 1997, pp. 900–905.** are expressed as follows:

$$\dot{x} = Ax + Bu$$

$$y = Cx + D$$

Where,

$$A = \begin{bmatrix} -10.06 & -12.99 & 0 & 0 & 0 \\ 1.096 & -11.27 & 0 & 0 & 0 \\ -1.000 & -15.00 & 0 & 15 & 0 \\ 0 & -1.000 & 0 & 0 & 15 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, B = \begin{bmatrix} 75.47 \\ 50.14 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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

Note that the above model is derived for a specific set of parameters. It should be noted that there are five system states: lateral velocity, yaw rate of the vehicle, lateral deviation from the desired centerline point at look-ahead distance *yL*, the angle between the tangent to the road and vehicle orientation, and the curvature of the road at the look-ahead distance. The control input *u(t)* is the front wheel steering angle and the output y(t) is the lateral deviation of the vehicle from the centre of the lane. Initial value of all states are equal to zero (i.e. x(0) = 0). The control system under consideration aims to design input *u(t)* such that the output y(t) follows a given constant reference. This is called a regulation problem.

NOTE: This is an individual assignment, i.e. you need to write codes and report on your own. However, you are allowed to discuss with others.

Note that this system is uncontrollable. To design a controller for an uncontrollable system, we need to first do a controllability decomposition to derive a controllable subsystem. A controller is designed for this controllable subsystem. If you would like to know more, you may refer to "R. C. Dorf and R. H. Bishop, Modern control systems. Pearson, 2011". For the scope of this assignment, a sample code that does controllability decomposition is provided.

2. Embedded Implementation

A control application is composed of three application tasks – *S* (*sensing*), *C* (*computation*), and A (*actuation*). The implementation of a control application is done by the execution of these three application tasks. The sensing task consists of an image signal (pre-)processing (ISP), an image processing algorithm that does lane detection and lateral deviation computation. Fig.1 explains the sensing task processing steps of the LKAS. Fig.2 shows the Synchronous Data Flow (SDF) graph of the overall LKAS used for the assignment. As shown in the Fig.1, the first three steps i.e., the image captured from VREP, RGB to Gray image conversion, and birds eye view transformation, are considered as isp_1 actor in the SDF graph. In these three steps, image captured from VREP is a colored RGB image which is converted to a gray image that is used for the subsequent processing steps. The birds eye view transformation technique is used to generate a top view perspective of an image. Next is to perform Sobel filtering to detect the edges in the image and perform image thresholding. That is, this step performs image segmentation on birds eye view image and produces the binary image based on a certain threshold. They are considered in the isp_2 actor. Next is to extract white lane markers from the binary images, and this step is known as white masking and is considered as the isp_3 actor. White masking allows focusing only on the portions of the image that interests us. After completing the image signal pre-processing steps, the image processing algorithm detects the regions-of-interest, which is denoted as the *RoID* actor; the *RoIP* actor models the processing of each detected region, and the outcomes are merged by the *RoIM* actor (Region of interest and polyfit lane boundaries).

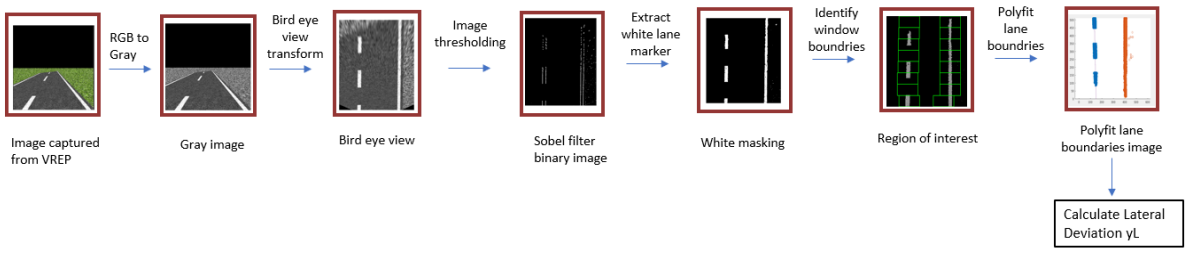
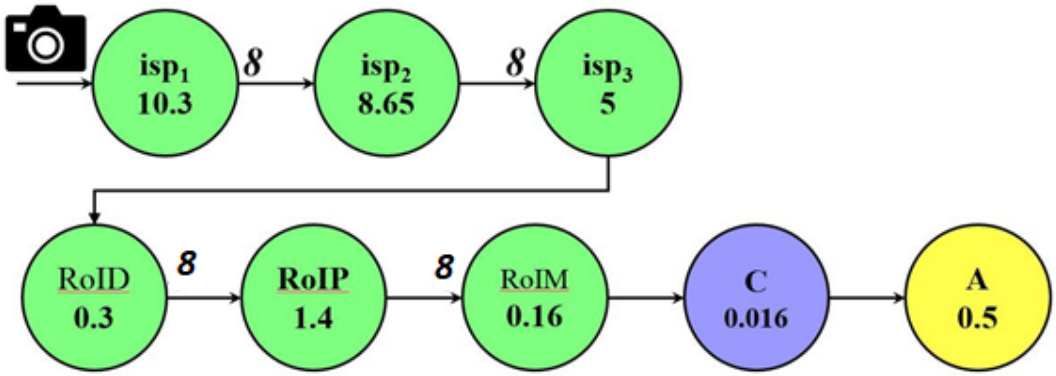


Figure 1. Sensing task processing steps of LKAS.

The SDF graph in Fig.2 shows that the isp_2 and RoIP actors can be executed in parallel in 8 processing cores. More cores allows the designer to parallelize or pipeline the execution of sensing tasks. Such implementation choice impacts the following two controller design parameters to be considered in the controller design.

- **Sampling period**: time between the start times of two consecutive sensing tasks.
- **Sensor-to-actuator delay**: time between the start of a sensing (S) task and the finish of the corresponding actuation (A) task.



3. Embedded Platform

The platform to be considered for implementation is a multi-processor platform with 8 cores. The platform runs under a time-triggered static-order scheduling scheme. The camera image frames arrive at a frame rate of **100 fps**. The timing details for the tasks are shown in Fig. 2.

4. Problem Definition

The problem is to design a discrete-time controller $u[k]$ i.e., to control the front wheel steering angle such that all the following constraints are satisfied

- $y[k] - r \to 0$ as $k \to \infty$, where $y[k]$ is the lateral deviation of the vehicle and r is the reference
- The time needed for $y[k]$ to be within 2% of r is called settling time which should be as short as possible.

The **design objective** is to achieve a settling time **as short as possible i.e., the vehicle must come back to the middle of the lane as soon as possible**. Note that the reference is zero for LKAS (as per system dynamics). You need to vary the third state (x3) to inject disturbances, e.g., set x3(0) = -0.15, for an initial disturbance.

5. Analysis

i. Design controllers for **sequential** execution of control tasks with and without parallelization of sensing tasks for the following two cases (Case I and Case II). Sensing tasks can be parallelized if we know the internal structure (e.g. SDF graph) of the algorithm. An illustrative example of the two cases is given in Fig. 3 when sampling period is equal to delay. Please consider a frame rate of **100 fps** and **consider the SDF graph in Fig. 2**. The sensor-to-actuator delay should be less than sampling period.

(6 points)

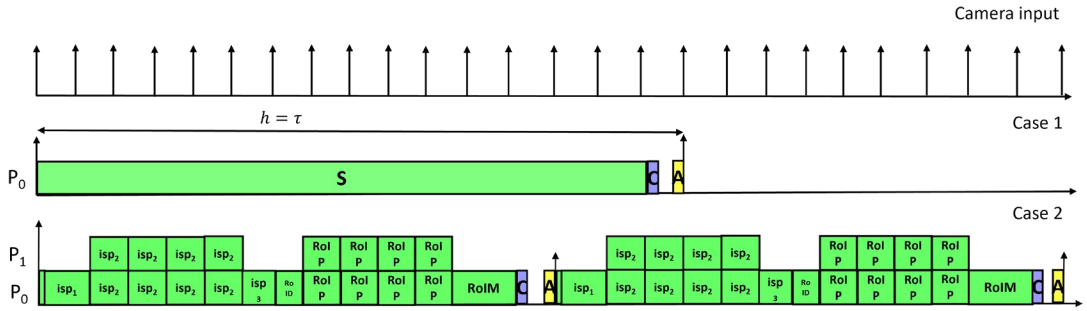
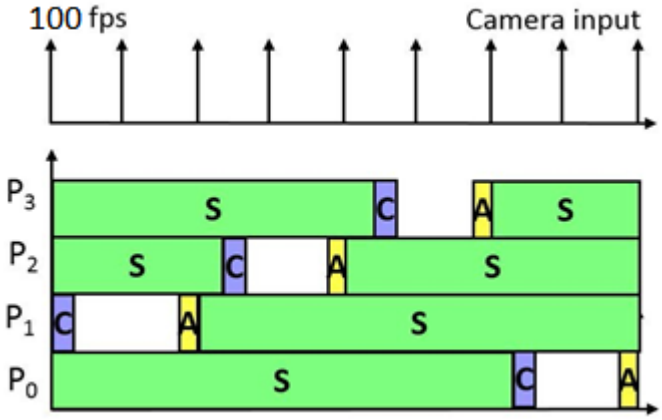


Figure 3. Illustration example with two cores for the SDF graph in Fig.2**.**

- **Case I**: One core is allocated to the tasks *S*, *C* and *A* (see above figure: Case 1).
- **Case II**: Eight cores are allocated to the tasks *S, C* and *A*.
  - Compare and comment on the settling time achieved in Case I and Case II.
  - How will the performance be affected if we increase or decrease the number of cores available and the camera frame rate? Discuss.

ii. Design controllers for **pipelined** execution. In this case, assume that sensing tasks **cannot** be parallelized. This situation occurs when the internal structure (SDF graph) of the sensing algorithm is not available to the control/embedded systems designer, i.e. the sensing algorithm is a black box. **(6 points)**

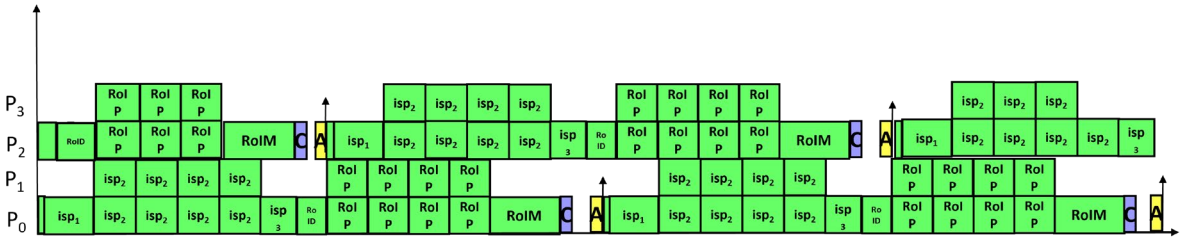


Above figure is an illustrative example with four cores for the SDF in Fig. 2 (which is not known to the designer). You can also implement the case where sensor-to-actuator delay is not an integer multiple of sampling period.

- **Case III**: Eight cores are available and the camera has 100 fps.
- **Case IV**: Evaluate the trade-offs between number of cores and control performance. Assume that you could have any number of cores between 1 to 8.

iii. Design a controller for pipelined execution. Assume that sensing tasks **can** be parallelized. This means that the designer knows the internal structure of the sensing algorithm and he could choose to have an implementation with pipelined parallelism. Pipelined parallelism refers to the combination of parallel and pipelined execution as shown in the figure below. The figure gives the example of hybrid implementation with 4 cores for SDF in Fig. 2.

(8 points)



6. Software-in-the-Loop (SiL) Validation using IMACS

Use the **IMACS** SiL framework (see installation and user guide) to validate your control design. The state-space matrices for the LKAS model is given as in **vrepLKAS.m** and an example controller is also provided in the SiL framework.

- Evaluate your own controllers designed in the five cases and compare the vehicle behaviour in terms of stability and performance?
- Compare the results obtained from the Model-in- the-Loop and comment.

(15 points)

7. Questions to be answered in this project and deliverables

1. Report <**fullName>_P2.pdf** answering the questions with justifications and explanations. Have a results table, e.g. with columns: cases, delay, period, settling time.
   - **Cases I-V**: design, results and analysis.
2. Corresponding MATLAB script(s): <**name>.m** (*.m file(s)).
3. Upload all the files you have changed in the imacs framework as zip (maintaining the folder structure).
4. SiL video: provide a link to the video in your report.
5. Upload the above documents as a zip file, **name_P2.zip.**

8. Provided materials

1. System Models - systemModel.m. You can re-use the MATLAB codes from the lectures. Alternately, you can design using an optimal control method of your choice. Note that you need to justify your choices and approach.
2. Assignment 2 SIL tutorial.
3. IMACS files - guide to installation, link to *.ova, IMACS overview video, IMACS virtual machine usage instructions video.
4. discreteTimeController.m- contains the sample code for controllability decomposition, augmentSystem.m (for system augmentation), designControlGainsLQR.m (for designing LQR controller).