To appear in Design, Automation and Test in Europe (DATE), 2021

Perception Computing-Aware Controller Synthesis for Autonomous Systems

Clara Hobbs*, Debayan Roy†, Parasara Sridhar Duggirala*, F. Donelson Smith*,
Soheil Samii‡, James H. Anderson*, Samarjit Chakraborty*
*University of North Carolina at Chapel Hill, USA, †Technical University of Munich, Germany, ‡General Motors, USA
Email: {cghobbs, psd, smithfd, anderson, samarjit}@cs.unc.edu, debayan.roy@tum.de, soheil.samii@gm.com

***Abstract*—Feedback control loops are ubiquitous in any autonomous system. The design flow for any controller starts by determining a control strategy, while abstracting away all implementation details. However, when designing controllers for *autonomous systems*, there is significant computation associated with the perception modules. For example, this involves vision processing using deep neural networks on multicore CPU+accelerator platforms. Such computation can be organized in many different ways, with each choice resulting in very different sensor-to-actuator delays and tradeoffs between cost, delay, and accuracy. Further, each of these choices requires the control strategy to be designed accordingly. It is not possible for a control designer to enumerate and account for all of these choices manually, or abstract them away as "implementation details" as done in traditional controller design. In this paper we outline this problem and discuss how automated controller-synthesis techniques could help in addressing it.**

I. Introduction

Feedback control loops implement many core functions in any autonomous system – be it an autonomous vehicle or a robot. The design flow for all controllers traditionally starts with determining the control strategy, which involves resolving choices regarding control laws, sampling periods, and the values of various controller parameters. This is followed by implementing the designed controller in software to run on an embedded platform. This flow results in a clean separation of concerns, where control designers need not be concerned with software and implementation architecture details, and instead communicates with embedded systems engineers via well-defined interfaces specifying sampling periods and sensor-to-actuator delays. Similarly, embedded systems engineers implementing the controllers can view them as periodic tasks with deadlines and focus only on scheduling issues.

However, when designing controllers for *autonomous* systems, this design-followed-by-implementation paradigm is starting to break down. This is because autonomous systems are invariably equipped with multiple cameras, radars, or lidars, whose data needs considerable pre-processing before they can be fed into a controller. The high computation and communication requirements associated with such sensor data processing – *e.g.*, vision processing using deep neural networks – is unlike the processing requirements of traditional simpler sensors whose readings can be directly used. More importantly, first there are several vision processing/classification algorithms to choose from [1], which represent tradeoffs between computation time, accuracy and robustness of the results they return. Second, there are many different ways in which the chosen sensor data processing algorithms can be organized in hardware and software using a combination of multicore CPUs+accelerator platforms [2]–[4]. Each of these choices results in very different sensor-to-actuator delays and once again tradeoffs between cost, delay, and accuracy.

Further, in order to get good performance, each of these choices of sensor data processing requires the control strategy to be designed accordingly. It is not possible for a control designer to enumerate and account for all of these choices manually. Hence, what is required is an automated synthesis technique that, given partial specifications of controllers and the implementation platform, can generate the optimal controller and its implementation while taking into account all the sensor data processing choices and their associated tradeoffs. This paper outlines the challenges involved in realizing this vision and presents some potential solutions.

Related work: The general problem of accounting for delays in controller design has been has been widely studied and has led to the research topic of *networked control systems* (NCS) [5], [6]. Work on NCS has primarily considered control applications whose control signals are sent over a wireless network, where occasional delays and packet losses can occur. Hence, stochastic settings [7] motivated by wireless communication have been a major focus. The core problem addressed in NCS research can be expressed as: how to incorporate the characteristics of a *given* network (*e.g.*, statistics on packet losses and delays) in a controller, to meet specified control objectives? In contrast, the problem we address is more general: we wish to design *both* the network (*viz.*, the perception-computing or the implementation architecture) and the control algorithms running with it. This results in a cyber-physical systems design or a *co-design problem* [8]–[10]. While techniques from NCS can certainly be leveraged here, the possibility to additionally design the "network" opens up new research questions [11].

The control theory research community has also considered some aspects of *network design*, such as scheduling sensor nodes to reduce packet collisions [12], [13]. More recently, the *network design* problem is also being studied in a distributed control setting [14], [15] but with a much more abstract notion of controller deployment than what we discuss in this paper, *viz.*, how to organize and schedule perception computing and what impact it has on control performance? These questions are also related to [16] on quantifying the model-implementation semantic gap in controllers, and to [17] on synthesizing scheduling and controller parameters.

In this paper, we discuss the impact that different options of scheduling perception computing has on the *timing* behavior experienced by the controller. We do not dwell on issues such as the uncertainty of the results returned by the perception computing algorithms, although they are also important. Clearly, the issue of timing analysis falls within the scope of our discussion, and has been studied in different concrete settings. Since autonomous vehicles are an important class of autonomous systems, timing analysis of automotive hardware/software architectures and also controllers is of relevance, and has been studied in the past [18]–[21]. There has been very little study on the impact of perception computing on control performance, as we discuss in this paper. A notable exception is [22] and our study is in the same direction.

II. Problem Formulation

In this paper, we study linear and time-invariant feedback control systems. The state-space mathematical model of the dynamic behaviour of such a system in continuous-time can be represented by the following differential equations:

$$\dot{x}(t) = Ax(t) + Bu(t), \tag{1}$$

$$y(t) = Cx(t), \tag{2}$$

where the vectors $x(t) \in \mathbb{R}^{n\times 1}$, $y(t) \in \mathbb{R}^{p\times 1}$, and $u(t) \in \mathbb{R}^{m\times 1}$ represent the system states, the system output, and the control input respectively at time instant t.[1] Here, the constant matrices $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, and $C \in \mathbb{R}^{p\times n}$ are respectively the state, the input, and the output matrices. Our goal here is to formulate the controller design problem where different control inputs u_j are associated with different delays, because of different sensors they are associated with. The different sensor processing tasks have different computation requirements and are scheduled in a certain manner.

Given a continuous-time state-space model as in Eq. (1), we can solve the first-order differential equation to determine an expression for the states at time t, given by:

$$x(t) = e^{At}x(0) + \int_0^t e^{A(t-\tau)}Bu(\tau)d\tau \tag{3}$$

Denoting t_k as the k-th sampling instant, we can also write:

$$x(t_{k+1}) = e^{A(t_{k+1}-t_k)}x(t_k) + \int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}Bu(\tau)d\tau \tag{4}$$

Let us consider $u(\tau) = \begin{bmatrix} u_1(\tau) & u_2(\tau) & \cdots & u_m(\tau)\end{bmatrix}^T$ and $B = \begin{bmatrix} B_1 & B_2 & \cdots & B_m\end{bmatrix}$, where B_j is a column vector, *i.e.*, $B_j \in \mathbb{R}^{n\times 1}$. Then, we can rewrite Eq. (4) as follows:

$$\begin{aligned} x(t_{k+1}) &= e^{A(t_{k+1}-t_k)}x(t_k) \\ &\quad + \sum_{j=1}^{m}\left[\int_{t_k}^{t_{k+1}} e^{A(t_{k+1}-\tau)}B_j u_j(\tau)d\tau\right] \end{aligned} \tag{5}$$

Now, let us assume that the delay from sensing to the actuation of u_j is d_j. Thus, we may write $u_j(t) = u_j[k-1]$ for $t_k \le t < t_k + d_j$ and $u_j(t) = u_j[k]$ for $t_k + d_j \le t < t_{k+1}$. Thus, we rewrite Eq. (5) as follows:

$$\begin{aligned} x(t_{k+1}) &= e^{A(t_{k+1}-t_k)}x(t_k) \\ &\quad + \sum_{j=1}^{m}\left[\left[\int_{t_k}^{t_k+d_j} e^{A(t_{k+1}-\tau)}B_j d\tau\right]u_j[k-1]\right. \\ &\quad \left. + \left[\int_{t_k+d}^{t_{k+1}} e^{A(t_{k+1}-\tau)}B_j d\tau\right]u_j[k]\right] \end{aligned} \tag{6}$$

Assuming a constant sampling period $h = t_{k+1} - t_k$ and substituting $\tau' = t_{k+1} - \tau$, we get:

$$\begin{aligned} x(t_{k+1}) &= e^{Ah}x(t_k) + \sum_{j=1}^{m}\left[\left[-\int_{h}^{h-d_j} e^{A\tau'}B_j d\tau'\right]u_j[k-1]\right. \\ &\quad \left. + \left[-\int_{h-d_j}^{0} e^{A\tau'}B_j d\tau'\right]u_j[k]\right] \\ &= \left[\because \text{changing integration limits}\right] \\ &\quad e^{Ah}x(t_k) + \sum_{j=1}^{m}\left[\left[\int_{h-d_j}^{h} e^{A\tau'}B_j d\tau'\right]u_j[k-1]\right. \\ &\quad \left. + \left[\int_{0}^{h-d_j} e^{A\tau'}B_j d\tau'\right]u_j[k]\right] \\ &= \Phi x(t_k) + \sum_{j=1}^{m}\left[\Gamma_{1,j}u_j[k-1] + \Gamma_{0,j}u_j[k]\right] \end{aligned} \tag{7}$$

where,

$$\Phi = e^{Ah}, \quad \Gamma_{1,j} = \int_{h-d_j}^{h} e^{A\tau'}B_j d\tau',$$

$$\Gamma_{0,j} = \int_{0}^{h-d_j} e^{A\tau'}B_j d\tau'.$$

Or, we can write:

$$x(t_{k+1}) = \Phi x(t_k) + \Gamma_1 u[k-1] + \Gamma_0 u[k] \tag{8}$$

where,

$$\Gamma_0 = \begin{bmatrix}\Gamma_{0,1} & \Gamma_{0,2} & \cdots & \Gamma_{0,m}\end{bmatrix}, \; \Gamma_1 = \begin{bmatrix}\Gamma_{1,1} & \Gamma_{1,2} & \cdots & \Gamma_{1,m}\end{bmatrix} \tag{9}$$

We denote $x[k] = x(t_k)$ and write Eq. (8) as follows:

$$x[k+1] = \Phi x[k] + \Gamma_1 u[k-1] + \Gamma_0 u[k]. \tag{10}$$

Now, we consider an augmented state vector $z[k] = \begin{bmatrix} x[k] \\ u[k-1]\end{bmatrix}$, and rewrite Eq. (10) as follows:

$$z[k+1] = \begin{bmatrix}\Phi & \Gamma_1 \\ \mathbf{0} & \mathbf{0}\end{bmatrix} z[k] + \begin{bmatrix}\Gamma_0 \\ \mathbf{I}\end{bmatrix} u[k] = \Phi_a z[k] + \Gamma_a u[k], \tag{11}$$

where we denote

$$\Phi_a = \begin{bmatrix}\Phi & \Gamma_1 \\ \mathbf{0} & \mathbf{0}\end{bmatrix}; \quad \Gamma_a = \begin{bmatrix}\Gamma_0 \\ \mathbf{I}\end{bmatrix}. \tag{12}$$

We define the stabilizing feedback control law as follows:

$$u[k] = -Kz[k], \tag{13}$$

where the feedback gain K can be calculated using standard techniques like pole-placement and linear quadratic regulator (LQR) design based on Φ_a and Γ_a.

[1]$\mathbb{R}$ represents the set of real numbers.

TABLE I
PERFORMANCE VS DELAYS IN THE CONTROL INPUTS

		Second input delay (d_2) in ms										
		0	**25**	**50**	**75**	**100**	**125**	**150**	**175**	**200**	**225**	**250**
First input delay (d_1) in ms	**0**	18.75	18.78	18.81	18.84	18.87	18.9	18.93	18.96	18.98	19.01	19.04
	25	19.58	19.61	19.65	19.68	19.71	19.74	19.77	19.80	19.83	19.86	19.89
	50	20.45	20.48	20.52	20.55	20.59	20.62	20.65	20.68	20.72	20.75	20.78
	75	21.35	21.39	21.43	21.46	21.5	21.53	21.57	21.60	21.64	21.67	21.71
	100	22.29	22.33	22.37	22.41	22.45	22.49	22.53	22.57	22.60	22.64	22.67
	125	23.27	23.32	23.36	23.40	23.45	23.49	23.53	23.57	23.61	23.64	23.68
	150	24.3	24.34	24.39	24.44	24.48	24.52	24.57	24.61	24.65	24.69	24.74
	175	25.36	25.41	25.46	25.51	25.56	25.61	25.65	25.7	25.74	25.79	25.83
	200	26.47	26.53	26.58	26.63	26.68	26.73	26.78	26.83	26.88	26.93	26.97
	225	27.63	27.69	27.74	27.8	27.85	27.90	27.96	28.01	28.06	28.11	28.16
	250	28.83	28.89	28.95	29.01	29.07	29.12	29.18	29.24	29.29	29.35	29.4

III. A MOTIVATIONAL EXAMPLE

To illustrate the problem, we consider a multiple-input and multiple-output (MIMO) system for which the continuous-time plant model is given by:

$$A = \begin{bmatrix} -0.0558 & -0.9968 & 0.0802 & 0.0415 \\ 0.5980 & -0.1150 & -0.0318 & 0 \\ -3.0500 & 0.3880 & -0.4650 & 0 \\ 0 & 0.0805 & 1.0000 & 0 \end{bmatrix};$$

$$B = \begin{bmatrix} 0.0073 & -0.4750 & 0.1530 & 0 \\ 0 & 0.0077 & 0.1430 & 0 \end{bmatrix}^T; \quad C = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

This system has four states, two inputs and two outputs.

We consider a sampling period $h = 0.25\,\text{s}$ and we vary the delays d_1 and d_2 for the two control inputs respectively from 0 to h in a step of $0.1h$. For a given set of delays and the sampling period, we compute an infinite horizon LQR feedback control gain. We assume an initial condition as $z[0]^T = \begin{bmatrix} 0 & 0.6109 & 0 & 0.6109 & 0 & 0 \end{bmatrix}$. The cost function is given by:

$$J = \sum_{k=0}^{\infty} \Big(z[k]^T \cdot Q \cdot z[k] + u[k]^T \cdot R \cdot u[k] \Big), \tag{14}$$

where

$$Q = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}; \qquad R = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

This essentially means that the cost is given by:

$$J = \sum_{k=0}^{\infty} \Big(x_1[k]^2 + x_2[k]^2 + x_3[k]^2 + x_4[k]^2 + 0.5u_1[k]^2 + 0.5u_2[k]^2 \Big) \tag{15}$$

Here, we give equal importance to all the system states. Note that $Q(5,5) = Q(6,6) = 0$ because $z[5] = u_1[k-1]$ and $z[6] = u_2[k-1]$ and the cost for the control inputs are already considered in R. Furthermore, note that we have lower weights for the control inputs because, in this example, we want to stabilize the system within a reasonable time ($\leq 10\,\text{s}$) at the expense of higher control cost (*i.e.*, $u_1[k]^2 + u_2[k]^2$). The choices of Q and R, however, depend on the design objectives.

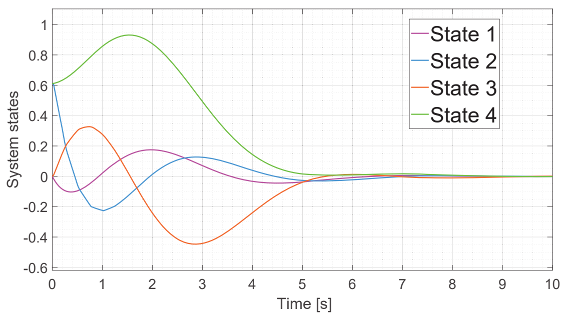
For all combinations of delay values d_1 and d_2 associated with the two control inputs, we tabulate the LQR cost in Table I. Let us assume that the computation of the first control input u_1 takes $25\,\text{ms}$ while the computation of the second input u_2 takes $175\,\text{ms}$. If both inputs are computed by the same processor, we can implement the controller in three ways: (i) u_1 is computed first and applied with a delay of $25\,\text{ms}$ and then u_2 is calculated and applied at $200\,\text{ms}$, (ii) u_2 is computed first and applied with a delay of $175\,\text{ms}$ and then u_1 is calculated and applied at $200\,\text{ms}$, and (iii) u_1 and u_2 are applied at the same time at $200\,\text{ms}$ after they are both computed. Evolution of the system states and the control inputs for the three cases are shown in Figures 1, 2, and 3 respectively. From Table 1, we can see that the LQR costs for the three cases (highlighted in red) are 19.83, 26.83, and 26.88 respectively. Thus, the performances obtained in case (ii) and case (iii) are $35.30\,\%$ and $35.65\,\%$ worse compared to case (i). This shows how the scheduling of perception computing tasks, and hence the delays associated with the different control inputs, can lead to significantly different performance.

IV. MOVING TO LARGER PROBLEMS

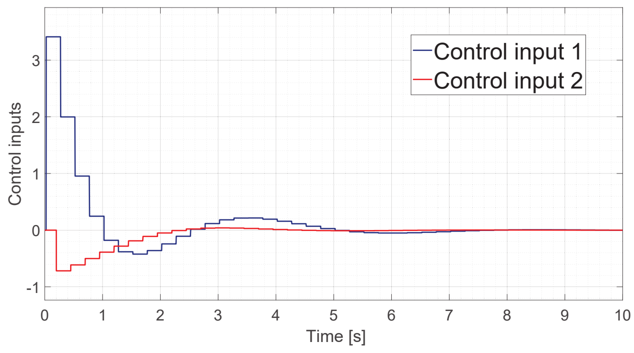
For real-world control systems with larger numbers of inputs and sensors, the naïve brute-force approach seen in Sec. III quickly becomes unwieldy. Consider a system with n control inputs, each requiring a known computation time. Assuming the control inputs are all computed on a single processor, and each control input is applied as soon as it is computed, it would be necessary to compute the LQR cost of $n!$ different controllers to find the schedule that gives the best performance. In this section, we examine heuristics to address this intractability, followed by a case study to evaluate them.

A. *Heuristics for Perception Computing-Aware Controllers*

We will now propose several heuristics designed to efficiently determine an order in which to apply the control inputs of an MIMO system that yields a low LQR cost. The inputs available for use in these heuristics include the plant model A, B, and C, the cost matrices Q and R, the sampling period h, and the

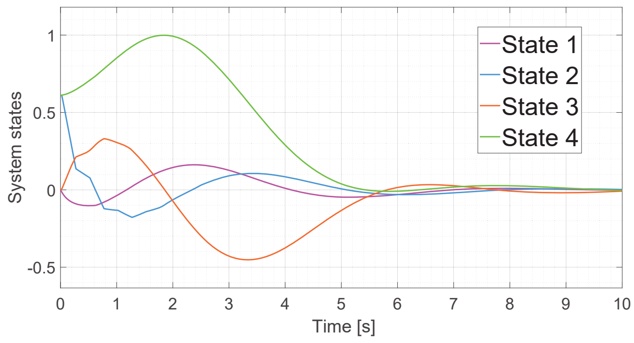


(a) System response.

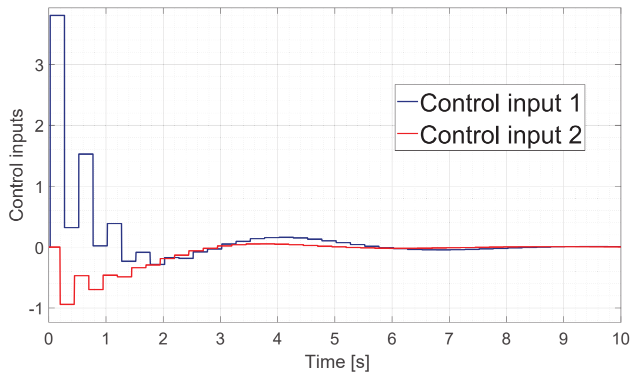


(b) Control inputs

Fig. 1. Input 1 applied at 25 ms and input 2 applied at 200 ms

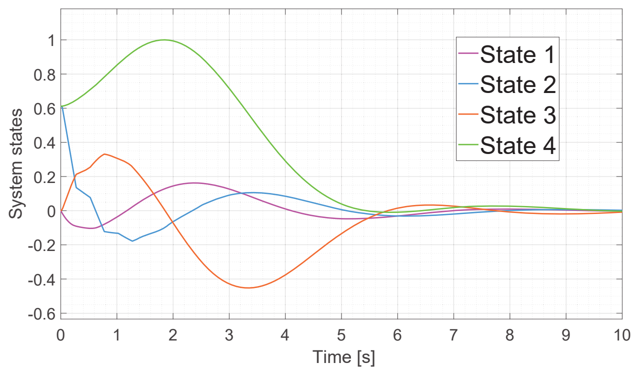


(a) System response.

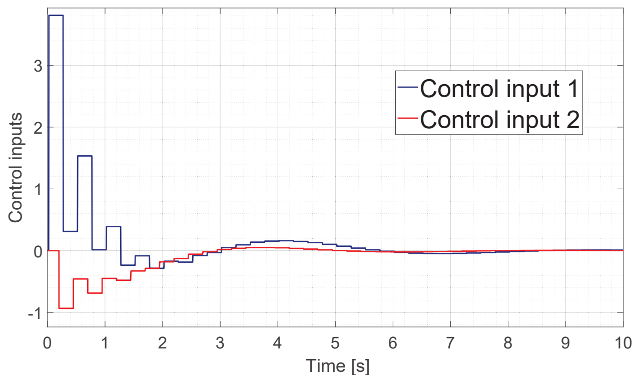


(b) Control inputs

Fig. 2. Input 1 applied at 200 ms and input 2 applied at 175 ms



(a) System response.



(b) Control inputs

Fig. 3. Both control inputs applied together at 200 ms

computation time required for each control input. Our first and simplest heuristic under consideration is shown in Alg. 1, which we discuss next.

Attempting to determine a good ordering of the control inputs, Alg. 1 makes use of the open-loop gain of the discrete-time system. The gain can be computed as $G = C_a(I - \Phi_a)^{-1}\Gamma_a$ using the augmented state-space model given in Eq. (12). The gain from each of the system's control inputs u_j to system output y_i is given by $G_{i,j}$. Intuitively, a control input whose gain values have a larger magnitude has a greater effect on the state of the system, so it would likely be preferable to apply the inputs in order of descending absolute gains. Unfortunately, this sorting is not possible to do directly, since the gains of each control input are in general not totally ordered (*e.g.*, for output y_1, $G_{1,1} > G_{1,2}$, but for output y_2, $G_{2,1} < G_{2,2}$). Therefore, for a system with m inputs and p outputs, we compute a combined gain G_j for each input u_j as

$$G_j = \sum_{i=1}^{p} G_{i,j}^2. \tag{16}$$

These combined gain values can be sorted directly, giving an order with which to apply the control inputs.

While Alg. 1 does succeed in creating an order with which to apply the control inputs, its choice is oblivious to the LQR cost matrices Q and R. To address this limitation, we propose a second heuristic, given in Alg. 2. This heuristic modifies Alg. 1

Algorithm 1: A simple heuristic for ordering control inputs by open-loop gain

```
Data: Matrices $A$, $B$, $C$, sampling period $h$
Result: Order to apply control inputs
Compute augmented discrete-time plant model $\Phi_a$, $\Gamma_a$, $C_a$
  assuming period $h$ and zero delay;
$G = C_a(I - \Phi_a)^{-1}\Gamma_a$;
for $j = 1$ to $m$ do
    $G_j = \sum_{i=1}^{p} G_{i,j}^2$;
end
schedule = ⟨inputs by descending $G_j$ values⟩;
return schedule;
```

Algorithm 2: A heuristic for ordering control inputs by closed-loop gain

```
Data: Matrices $A$, $B$, $C$, $Q$, $R$, sampling period $h$
Result: Order to apply control inputs
Compute augmented discrete-time plant model $\Phi_a$, $\Gamma_a$, $C_a$
  assuming period $h$ and zero delay;
Compute feedback gain matrix $K$ with LQR;
$G = C_a(I - (\Phi_a - \Gamma_a K))^{-1}\Gamma_a$;
for $j = 1$ to $m$ do
    $G_j = \sum_{i=1}^{p} G_{i,j}^2$;
end
schedule = ⟨inputs by descending $G_j$ values⟩;
return schedule;
```

by computing feedback control gains using LQR. Since initially no ordering of the control inputs is known, it is assumed that the control inputs are applied with zero delay when computing this controller. Once the feedback control gain matrix K is known, we compute the gain of the resulting closed-loop system, and order the inputs as before. Because the controller in this closed-loop system is designed using LQR, the cost matrices are thus incorporated in the schedule.

Alg. 2 addresses the limitation of ignoring the LQR cost function present in Alg. 1 by computing a feedback gain matrix using LQR. It is however still limited, in that it orders the control inputs without considering the computation time required for each input. To help address this issue, we further propose Alg. 3, an iterative version of Alg. 2. The first iteration creates a controller assuming zero delay and scheduling the control inputs based on closed-loop gain, as before. Each subsequent iteration creates an augmented plant model using the delays calculated in the prior iteration. A controller is then computed using LQR, and a new schedule is created using the gain of the new closed-loop system. This iteration generally continues until two consecutive iterations result in the same schedule. However, we leave it as future work to determine if this will always occur, so in order to guarantee termination the loop also ends after $n!$ iterations.

B. Case Study

To evaluate these heuristics, we consider the following MIMO system, with five states, four inputs, and four outputs:

$$A = \begin{bmatrix} -0.0558 & -0.9968 & 0.0802 & 0.0415 & 1.302 \\ 0.5980 & -0.1150 & -0.0318 & 0 & 0.153 \\ -3.0500 & 0.3880 & -0.4650 & 0 & -0.649 \\ 0 & 0.0805 & 1.0000 & 0 & 0 \\ 1.0325 & 0.1032 & 0.326 & -0.0681 & 0.126 \end{bmatrix};$$

Algorithm 3: Iterative heuristic for ordering control inputs by closed-loop gain

```
Data: Matrices $A$, $B$, $C$, $Q$, $R$, input computation time
      $c_1, \dots, c_m$, sampling period $h$
Result: Order to apply control inputs, schedule
schedule = ⟨⟩;
$d_1, \dots, d_m = 0, \dots, 0$;
$t = 0$;
repeat
    last_schedule = schedule;
    Compute augmented discrete-time plant model $\Phi_a$, $\Gamma_a$,
      $C_a$ assuming period $h$ and delays $d_1, \dots, d_m$;
    Compute feedback gain matrix $K$ with LQR;
    $G = C_a(I - (\Phi_a - \Gamma_a K))^{-1}\Gamma_a$;
    for $j = 1$ to $m$ do
        $G_j = \sum_{i=1}^{p} G_{i,j}^2$;
    end
    schedule = ⟨inputs by descending $G_j$ values⟩;
    for $j = 1$ to $m$ do
        $d_j = \sum_{i=1}^{j} c_{\text{schedule}[i]}$;
    end
    $t = t + 1$;
until last_schedule == schedule $\vee$ $t == n!$;
return schedule;
```

$$B = \begin{bmatrix} 0.0073 & 0 & 0.5 & 0.23 \\ -0.4750 & 0.0077 & 0.105 & 0.86 \\ 0.1530 & 0.1430 & 0 & -0.12 \\ 0 & 0 & 0.073 & 0.02 \\ 0.2020 & 0.0192 & 0 & 0.4 \end{bmatrix}; \; C = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The four control inputs u_1, u_2, u_3, and u_4 are assumed to require computation times of $0.025\,\text{s}$, $0.025\,\text{s}$, $0.050\,\text{s}$, and $0.125\,\text{s}$, respectively. Similar to the example in Sec. III, we consider a sampling period $h = 0.25\,\text{s}$, and an initial condition $z[0]^T = \begin{bmatrix} 0 & 0.6109 & 0 & 0.6109 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$. The cost function is of the form in Eq. (14), where

$$Q = \text{diag}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \end{pmatrix};$$
$$R = \text{diag}\begin{pmatrix} 0.5 & 0.5 & 0.5 & 0.5 \end{pmatrix},$$

giving equal weight to all system states, and lower weight to the control inputs to stabilize the system in a shorter time.

Using this problem setup, we ran Algs. 1, 2, and 3 to create schedules of the control inputs, and computed the corresponding LQR cost. Additionally, we ran a brute-force search to find the LQR cost resulting from all 24 orderings of the control inputs. The results are presented in Table II; the schedules produced by our heuristics are labeled, along with the best and worst schedules overall.

For this particular set of parameters, determining a schedule using the closed-loop gain with zero delay as in Alg. 2 gave an LQR cost $2.46\,\%$ higher than using the open-loop gain in Alg. 1. However, by iterating over the schedules using Alg. 3, we were able to find a schedule with $9.42\,\%$ better performance than the one found by Alg. 1. Unfortunately, this is still $14.12\,\%$ higher than the LQR cost resulting from the optimal schedule, as determined using a brute-force search.

A visualization of the design space of the different schedules is shown in Fig. 4. Each node in the directed graph is a schedule of the control inputs. Each node has one outgoing edge

TABLE II
SCHEDULES AND CORRESPONDING LQR COSTS

	Schedule	LQR Cost		Schedule	LQR Cost
	1 2 3 4	10.123		3 1 2 4	10.230
	1 2 4 3	11.039		3 1 4 2	10.247
Best	1 3 2 4	9.946		3 2 1 4	10.457
	1 3 4 2	9.965		3 2 4 1	11.542
	1 4 2 3	11.071	Alg. 3	3 4 1 2	11.351
	1 4 3 2	10.903		3 4 2 1	11.554
	2 1 3 4	10.383		4 1 2 3	12.794
	2 1 4 3	11.380	Alg. 1	4 1 3 2	12.532
	2 3 1 4	10.675		4 2 1 3	13.122
	2 3 4 1	11.829	Worst	4 2 3 1	13.438
	2 4 1 3	13.089	Alg. 2	4 3 1 2	12.840
	2 4 3 1	13.408		4 3 2 1	13.118

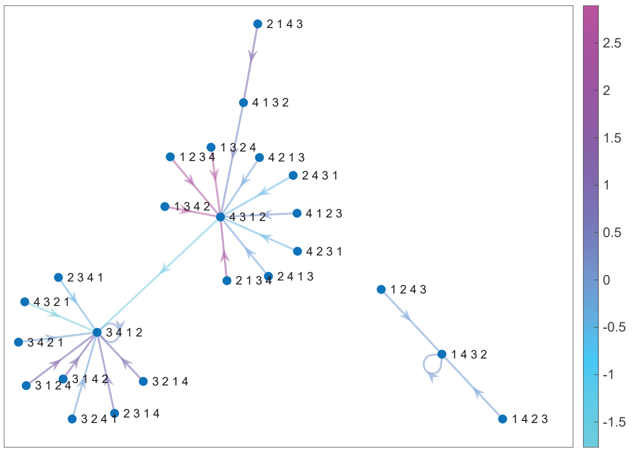


Fig. 4. All orders of the four control inputs, with edges showing the next schedule indicated by closed-loop gain for each.

indicating the next schedule that our iterative heuristic would try, based on the closed-loop gain of the system. Thus, self-loops indicate schedules where the iteration in Alg. 3 would terminate. The edges are colored to indicate the change in LQR cost between the two controllers. Several observations can be made from this graph. First, ordering the control inputs by closed-loop gain does not necessarily lead to a single "best" schedule: depending on the initial schedule chosen, iteration could lead to the schedule 3 4 1 2 or 1 4 3 2. It can also be seen that the next iteration step does not always give a lower LQR cost. A greedy iteration that stops if the next schedule produces a higher LQR cost may not give better results than Alg. 3 though, since later transitions may give further performance improvements. In the future, we would like to design heuristics that are able to overcome these limitations of gain-based input scheduling to produce better control performance.

V. CONCLUDING REMARKS

The goal of this paper was to initiate a study on how different choices of perception computing and their implementation might impact control performance in autonomous systems. As outlined in [1]–[4] there are many different choices of perception (*e.g.*, vision) processing algorithms and their implementations in modern autonomous systems. How to best choose and implement them – given their large design space – is still a relatively open problem that could be studied in conjunction with designing the associated controllers, leading to new avenues in cyber-physical systems design.

ACKNOWLEDGEMENTS

This work was supported by the NSF award #2038960. D. Roy was supported by the Chair for CPS in Production Engineering at TUM and the AvH Foundation.

REFERENCES

[1] J. Janai *et al.*, "Computer vision for autonomous vehicles: Problems, datasets and state of the art," *Found. Trends Comput. Graph. Vis.*, vol. 12, no. 1-3, pp. 1–308, 2020.
[2] M. Yang *et al.*, "Re-thinking CNN frameworks for time-sensitive autonomous-driving applications: Addressing an industrial challenge," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2019.
[3] M. Balszun, M. Geier, and S. Chakraborty, "Predictable vision for autonomous systems," in *23rd IEEE International Symposium on Real-Time Distributed Computing (ISORC)*, 2020.
[4] M. Geier *et al.*, "Debugging FPGA-accelerated real-time systems," in *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, 2020.
[5] J. P. Hespanha, P. Naghshtabrizi, and Y. Xu, "A survey of recent results in networked control systems," *Proceedings of the IEEE*, vol. 95, no. 1, pp. 138–162, 2007.
[6] M. B. G. Cloosterman *et al.*, "Controller synthesis for networked control systems," *Automatica*, vol. 46, no. 10, pp. 1584–1594, 2010.
[7] J. P. Hespanha, "Modeling and analysis of networked control systems using stochastic hybrid systems," *Annual Reviews in Control*, vol. 38, no. 2, pp. 155–170, 2014.
[8] S. Chakraborty *et al.*, "Automotive cyber-physical systems: A tutorial introduction," *IEEE Design & Test*, vol. 33, no. 4, pp. 92–108, 2016.
[9] D. Goswami *et al.*, "Challenges in automotive cyber-physical systems design," in *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*, 2012.
[10] M. Broy *et al.*, "Cross-layer analysis, testing and verification of automotive control software," in *11th International Conference on Embedded Software (EMSOFT)*, 2011.
[11] R. Schneider *et al.*, "Multi-layered scheduling of mixed-criticality cyber-physical systems," *J. Syst. Archit.*, vol. 59, no. 10-D, pp. 1215–1230, 2013.
[12] E. Garone, B. Sinopoli, and A. Casavola, "LQG control over lossy tcp-like networks with probabilistic packet acknowledgements," in *47th IEEE Conference on Decision and Control (CDC)*, 2008.
[13] L. Shi *et al.*, "Sensor scheduling over a packet-delaying network," *Automatica*, vol. 47, no. 5, pp. 1089–1092, 2011.
[14] S. Tseng and J. Anderson, "Deployment architectures for cyber-physical control systems," in *American Control Conference (ACC)*, 2020.
[15] J. Anderson *et al.*, "System level synthesis," *Annu. Rev. Control.*, vol. 47, pp. 364–393, 2019.
[16] T. Nghiem, G. J. Pappas, R. Alur, and A. Girard, "Time-triggered implementations of dynamic controllers," *ACM TECS*, vol. 11, no. S2, pp. 58:1–58:24, 2012.
[17] M. A. Khatib, A. Girard, and T. Dang, "Scheduling of embedded controllers under timing contracts," in *HSCC*, 2017.
[18] M. Lukasiewycz *et al.*, "System architecture and software design for electric vehicles," in *Design Automation Conference (DAC)*, 2013.
[19] A. Masrur *et al.*, "VM-based real-time services for automotive control applications," in *16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, 2010.
[20] D. Goswami, R. Schneider, and S. Chakraborty, "Relaxing signal delay constraints in distributed embedded controllers," *IEEE Trans. Contr. Sys. Techn.*, vol. 22, no. 6, pp. 2337–2345, 2014.
[21] L. Zhang *et al.*, "Timing challenges in automotive software architectures," in *36th International Conference on Software Engineering (ICSE)*, 2014.
[22] E. P. van Horssen, D. Antunes, and M. Heemels, "Switched LQG control for linear systems with multiple sensing methods," *Autom.*, vol. 103, pp. 217–229, 2019.