Data-Driven Retrospective Cost Adaptive Flow Control

Jacob C. Vander Schaaf[*], Qizhi Lu[†], Krzysztof J. Fidkowski[‡], and Dennis S. Bernstein[§]
University of Michigan, Ann Arbor, Michigan, 48109

This paper focuses on active adaptive flow control. In particular, data-driven retrospective cost adaptive control (DDRCAC) is applied to several 2D internal flow-control problems. Unlike many data-driven control methods, DDRCAC requires no advance data collection and model pretraining. Instead, DDRCAC uses online closed-loop system identification, and thus the controller adapts from zero gains, that is, "cold-start" conditions. The user need only specify hyperparameters for learning and adaptation, which includes the model and controller orders as well as the hyperparameters for recursive least squares with variable-rate forgetting. These hyperparameters are selected based on nominal simulations of the fluid dynamics. Four flow-control scenarios are considered, namely, control of streamwise velocity within a laminar boundary layer in a duct; axial cylinder boundary layer in a duct; turbulence suppression in a duct with a backwards-facing step; and, finally, turbulence suppression in a duct with crossflow jets.

I. Introduction

The ability to modify the natural motion of fluids is a scientifically challenging problem with application to a wide range of engineered systems and flight vehicles [1–11]. Flow control can be divided into passive and active techniques. Passive flow control consists of static or kinematic devices that modify the flow without requiring an energy source. Examples include riblets and leading-edge rotating cylinders [4, 12]. Benefits of passive flow control include reliability and energy efficiency.

In contrast to passive flow control, active flow control is based on devices that require an energy source, such as blowing and suction [4]. Further in this direction are active flow control techniques that rely on sensors and actuators [13]. Active feedback flow control requires a feedback control algorithm to determine the desired actuator action based on the sensor data. Since active flow control requires sensors and actuators, energy sources, fluid-dynamics modeling, and real-time computation, this approach is less reliable and less energy efficient than passive flow control techniques. The advantage of this technology, however, is the ability to provide better performance over a wider range of flow regimes.

Although the physics of fluid dynamics are well understood, the models used to design control laws may possess errors that degrade closed-loop performance [14, 15]. These errors include not only those associated with imprecise physical models and parameter values, but also discretization errors due to the choice of mesh resolution and time step. Diverse controller-synthesis techniques have been applied to flow control, including optimal control [16, 17], robust control [14], and machine-learning methods [10, 18]. The present paper focuses on adaptive control techniques, where the controller learns and adapts during operation in response to actual, changing conditions with minimal prior modeling information. In particular, the present paper focuses on retrospective cost adaptive control (DDRCAC).

RCAC was originally developed as a direct adaptive control technique for stabilization, command following, and disturbance rejection, as described in [19] and applied to flow control in [20]. More recently, RCAC was extended to include online closed-loop system identification in order to determine the essential model details needed to facilitate adaptation [21]. For data-driven RCAC (DDRCAC), online closed-loop system identification is performed using recursive least squares (RLS) with variable-rate forgetting [22–25]. Since DDRCAC includes online closed-loop system identification, the controller adapts from zero gains, that is, "cold-start" conditions. The user need only specify hyperparameters for learning and adaptation, which includes the model and controller orders as well as the RLS forgetting hyperparameters.

In order to assess the performance of DDRCAC, we consider a sequence of increasingly difficult flow control problems. In particular, we first by considering control of streamwise velocity within a laminar boundary layer in a duct,

[*]Ph.D. Candidate, Department of Aerospace Engineering, `jacobcvs@umich.edu`.
[†]Master's Student, Department of Aerospace Engineering, `luqseven@umich.edu`.
[‡]Professor, Department of Aerospace Engineering, `kfid@umich.edu`.
[§]Professor, Department of Aerospace Engineering, `dsbaero@umich.edu`.

followed by a more difficult axial cylinder boundary layer in a duct. Next, we consider turbulence suppression in a duct with a backwards-facing step as well as turbulence suppression in a duct with crossflow jets.

II. Sampled-Data Adaptive-Control Architecture

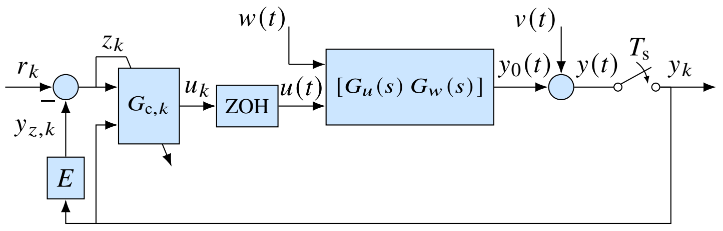

Fig. 1 Command following and disturbance rejection under sampled-data adaptive control. The objective is to follow commands r_k to the performance variable $y_{z,k} = Ey_k$. All sample-and-hold operations are synchronous.

RCAC is formulated for continuous-time linear systems under sampled-data control using a discrete-time, linear time-varying controller. As shown in [21], RCAC can be applied to nonlinear systems with unmodeled nonlinearities. Consider the adaptive control architecture shown in Figure 1, where a realization of $G(s) \triangleq [G_u(s)\ \ G_w(s)]$ is given by

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

$$y(t) = Cx(t) + D_u u(t) + v(t), \tag{2}$$

where $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the control, $w(t) \in \mathbb{R}^l$ is the disturbance, $y(t) \in \mathbb{R}^p$ is the noisy measurement of the system output, $v(t) \in \mathbb{R}^p$ is the sensor noise, and A, B, B_w, C, D_u, are real matrices. Define

$$G_u(s) \triangleq C(sI_n - A)^{-1}B + D_u, \tag{3}$$

$$G_w(s) \triangleq C(sI_n - A)^{-1}B_w + D_u, \tag{4}$$

where $G_u \in \mathbb{R}(s)^{p\times m}_{\rm prop}$ and $G_w \in \mathbb{R}(s)^{p\times l}_{\rm prop}$ are proper $p \times m$ and $p \times l$ transfer functions, respectively. The disturbance $w(t)$ is matched if there exists $\overline{U} \in \mathbb{R}^{m\times m}$ such that $B_w = B\overline{U}$; otherwise, the disturbance is unmatched. The system output $y_0(t) \in \mathbb{R}^p$ is corrupted by sensor noise $v(t)$ and sampled to produce $y_k \in \mathbb{R}^p$. The sampling operation can be realized as $y_k \triangleq y_0(kT_s) + v_k$, where $v_k \triangleq v(kT_s) \in \mathbb{R}^p$ is the sampled sensor noise and $T_s \in \mathbb{R}$ is the sample time. The performance variable is defined by $y_{z,k} \triangleq Ey_k \in \mathbb{R}^q$, where $E \in \mathbb{R}^{q\times p}$ selects components or linear combinations of components of y_k that are required to follow the command $r_k \in \mathbb{R}^q$. The command-following error is thus $z_k \triangleq r_k - y_{z,k} \in \mathbb{R}^q$. For all of the examples in this paper, $q = 1$, $E = 1$, and thus $y_{z,k} = y_k$ and $z_k = r_k - y_k$. The inputs to the adaptive feedback controller $G_{c,k}$ are the measurement y_k and the command-following error z_k. The adaptive feedback controller produces the discrete-time control $u_k \in \mathbb{R}^m$ at each step k. The continuous-time control $u(t)$ is produced by applying a zero-order-hold operator to u_k. Note that z_k serves as the adaptation variable, as denoted by the diagonal line in Figure 1 passing through $G_{c,k}$. The objective is to minimize the magnitude of the command-following error z_k in the presence of the disturbance $w(t)$ and sensor noise $v(t)$.

Figure 2 shows an equivalent representation of Figure 1, where $w(t)$ and $y_{w,k}$ are related by the operator

$$y_{w,k} \triangleq \mathcal{G}[w(t)] = C\int_{(k-1)T_s}^{kT_s} e^{A(kT_s-\tau)}B_w w(\tau)\,{\rm d}\tau. \tag{5}$$

Note that Figure 2 shows two transfer functions in feedback, namely, $G_{\rm d}(\mathbf{q})$ and $EG_{\rm d}(\mathbf{q})$, which are the transfer functions from u_k to y_k and u_k to $y_{z,k}$, respectively. Furthermore, $G_{\rm d} \in \mathbb{R}(\mathbf{q})^{p\times m}_{\rm prop}$, where $\mathbf{q}$ is the forward-shift operator, is the exact discretization of $G_u(s)$ using zero-order-hold and sampling operations. Consequently,

$$y_k = \mathcal{G}[w(t)] + G_{\rm d}(\mathbf{q})u_k + v_k, \tag{6}$$

$$z_k = r_k - Ey_k. \tag{7}$$

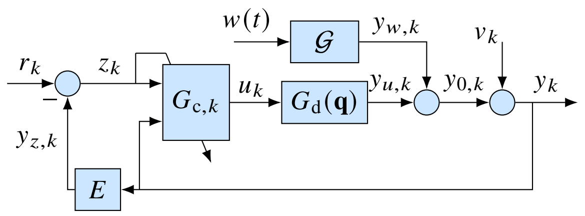

Fig. 2 Equivalent representation of Figure 1. The exact discretization $G_d(\mathbf{q})$ of $G_u(s)$ operates on u_k to generate $y_{u,k}$.

Note that the argument $\mathbf{q}$ of G_d in (6) reflects the fact that (6) is a time-domain equation whose solution depends on the initial conditions of the input-output system. Using the Z-transform variable $\mathbf{z}$ in place of the forward-shift operator $\mathbf{q}$ would account for the forced response of (6) but would implicitly assume zero initial conditions and thus would omit the free response. The distinction between $\mathbf{z}$ and $\mathbf{q}$ in accounting for initial conditions and the resulting free response is discussed in [26, 27].

III. Retrospective Cost Adaptive Control (RCAC)

This section summarizes retrospective cost adaptive control (RCAC). RCAC adapts by updating a strictly proper, discrete-time dynamic compensator of the form

$$u_k = \sum_{i=1}^{n_c} P_{i,k} u_{k-i} + \sum_{i=1}^{n_c} Q_{i,k} \tilde{y}_{k-i}, \tag{8}$$

where $k \geq 0$, $u_k \in \mathbb{R}^m$ is the requested control, n_c is the controller window length, $\tilde{y}_k \in \mathbb{R}^{l_y}$, and $Q_{1,k}, \ldots, Q_{n_c,k} \in \mathbb{R}^{m \times l_y}$ and $P_{1,k}, \ldots, P_{n_c,k} \in \mathbb{R}^{m \times m}$ are the numerator and denominator controller coefficient matrices, respectively. To avoid the need to invoke additional modeling information, "cold start" is assumed, where $Q_{1,0}, \ldots, Q_{n_c,0}$, $P_{1,0}, \ldots, P_{n_c,0}$, $u_{-n_c}, \ldots, u_{-1}$, and $\tilde{y}_{-n_c}, \ldots, \tilde{y}_{-1}$ are zero, and thus $u_0 = 0$. The controller (8) can be written as

$$u_k = \phi_{c,k} \theta_{c,k}, \tag{9}$$

where

$$\phi_{c,k} \triangleq \begin{bmatrix} u_{k-1} \\ \vdots \\ u_{k-n_c} \\ \tilde{y}_{k-1} \\ \vdots \\ \tilde{y}_{k-n_c} \end{bmatrix}^{\mathrm{T}} \otimes I_m \in \mathbb{R}^{m \times l_{\theta_c}} \tag{10}$$

is the *controller regressor*, $l_{\theta_c} \triangleq n_c m(m + l_y)$, and the *controller coefficient vector* is defined by

$$\theta_{c,k} \triangleq \operatorname{vec} \begin{bmatrix} P_{1,k} & \cdots & P_{n_c,k} & Q_{1,k} & \cdots & Q_{n_c,k} \end{bmatrix} \in \mathbb{R}^{l_{\theta_c}}. \tag{11}$$

In terms of $\mathbf{q}$, the controller (8) can be expressed as

$$u_k = G_{c,k}(\mathbf{q}) \tilde{y}_k, \tag{12}$$

where

$$N_{c,k}(\mathbf{q}) \triangleq Q_{1,k} \mathbf{q}^{n_c - 1} + \cdots + Q_{n_c,k}, \tag{13}$$

$$D_{c,k}(\mathbf{q}) \triangleq I_m \mathbf{q}^{n_c} - P_{1,k} \mathbf{q}^{n_c - 1} - \cdots - P_{n_c,k}, \tag{14}$$

$$G_{c,k}(\mathbf{q}) \triangleq D_{c,k}^{-1}(\mathbf{q}) N_{c,k}(\mathbf{q}). \tag{15}$$

The signal $\tilde{y}_k$ is constructed from z_k, y_k, and r_k. In the simplest case, $\tilde{y}_k = z_k$, whereas, when additional measurements are available, $\tilde{y}_k = [\ z_k^{\rm T}\ y_k^{\rm T}\ ]^{\rm T}$. Alternatively, command feedforward can be included by setting $\tilde{y}_k = [\ z_k^{\rm T}\ r_k^{\rm T}\ ]^{\rm T}$. More generally, the components of $\tilde{y}_k$ can be arbitrary, fixed linear combinations of the components of z_k, y_k, and r_k.

Next, define the filtered signals

$$u_{{\rm f},k} \triangleq G_{\rm f}(\mathbf{q})u_k, \tag{16}$$

$$\phi_{{\rm f},k} \triangleq G_{\rm f}(\mathbf{q})\phi_{{\rm c},k}, \tag{17}$$

where, for startup, $u_{{\rm f},k}$ and $\phi_{{\rm f},k}$ are initialized at zero and thus are computed as the forced responses of (53) and (54), respectively. The $q \times m$ filter $G_{\rm f}(\mathbf{q})$ has the form

$$G_{\rm f}(\mathbf{q}) \triangleq D_{\rm f}(\mathbf{q})^{-1}N_{\rm f}(\mathbf{q}), \tag{18}$$

where

$$N_{\rm f}(\mathbf{q}) \triangleq N_{{\rm f},0}\mathbf{q}^{n_{\rm f}} + N_{{\rm f},1}\mathbf{q}^{n_{\rm f}-1} + \cdots + N_{{\rm f},n_{\rm f}}, \tag{19}$$

$$D_{\rm f}(\mathbf{q}) \triangleq I_q\mathbf{q}^{n_{\rm f}} + D_{{\rm f},1}\mathbf{q}^{n_{\rm f}-1} + \cdots + D_{{\rm f},n_{\rm f}}, \tag{20}$$

$n_{\rm f}$ is the filter window length, and $N_{{\rm f},0}, \ldots, N_{{\rm f},n_{\rm f}} \in \mathbb{R}^{q\times m}$ and $D_{{\rm f},1}, \ldots, D_{{\rm f},n_{\rm f}} \in \mathbb{R}^{q\times q}$ are the numerator and denominator coefficients of $G_{\rm f}(\mathbf{q})$, respectively. The construction of $G_{\rm f}(\mathbf{q})$ is described below.

Equivalently, (16) and (17) can be written as

$$u_{{\rm f},k} = -DU_{{\rm f},k} + NU_k, \tag{21}$$

$$\phi_{{\rm f},k} = -D\Phi_{{\rm f},k} + N\Phi_{{\rm c},k}, \tag{22}$$

where

$$U_{{\rm f},k} \triangleq \begin{bmatrix} u_{{\rm f},k-1} \\ \vdots \\ u_{{\rm f},k-n_{\rm f}} \end{bmatrix} \in \mathbb{R}^{n_{\rm f}q}, \quad U_k \triangleq \begin{bmatrix} u_k \\ \vdots \\ u_{k-n_{\rm f}} \end{bmatrix} \in \mathbb{R}^{(n_{\rm f}+1)m}, \tag{23}$$

$$\Phi_{{\rm f},k} \triangleq \begin{bmatrix} \phi_{{\rm f},k-1} \\ \vdots \\ \phi_{{\rm f},k-n_{\rm f}} \end{bmatrix} \in \mathbb{R}^{n_{\rm f}q\times l_{\theta_{\rm c}}}, \quad \Phi_{{\rm c},k} \triangleq \begin{bmatrix} \phi_{{\rm c},k} \\ \vdots \\ \phi_{{\rm c},k-n_{\rm f}} \end{bmatrix} \in \mathbb{R}^{(n_{\rm f}+1)m\times l_{\theta_{\rm c}}}, \tag{24}$$

$$N \triangleq \begin{bmatrix} N_{{\rm f},0} & \cdots & N_{{\rm f},n_{\rm f}} \end{bmatrix} \in \mathbb{R}^{q\times m(n_{\rm f}+1)}, \quad D \triangleq \begin{bmatrix} D_{{\rm f},1} & \cdots & D_{{\rm f},n_{\rm f}} \end{bmatrix} \in \mathbb{R}^{q\times qn_{\rm f}}. \tag{25}$$

Next, to update the controller coefficient vector (11), we define the *retrospective performance variable*

$$\hat{z}_k(\theta_{\rm c}) \triangleq z_k - (u_{{\rm f},k} - \phi_{{\rm f},k}\theta_{\rm c}), \tag{26}$$

where z_k is given by (7) and $\theta_{\rm c}$ is a generic variable for optimization. Note that $u_{{\rm f},k}$ depends on u_k and thus on the current controller coefficient vector $\theta_{{\rm c},k}$. The retrospective performance variable $\hat{z}_k(\theta_{\rm c})$ is used to determine the updated controller coefficient vector $\theta_{{\rm c},k+1}$ by minimizing a function of $\hat{z}_k(\theta_{\rm c})$. The optimized value of $\hat{z}_k$ is thus given by

$$\hat{z}_k(\theta_{{\rm c},k+1}) = z_k - (u_{{\rm f},k} - \phi_{{\rm f},k}\theta_{{\rm c},k+1}), \tag{27}$$

which shows that the updated controller coefficient vector $\theta_{{\rm c},k+1}$ is applied retrospectively with the filtered controller regressor $\phi_{{\rm f},k}$. Note that $G_{\rm f}(\mathbf{q})$ is used to obtain $\phi_{{\rm f},k}$ from ϕ_k by means of (17) but ignores past changes in the controller coefficient vector, as can be seen by the product $\phi_{{\rm f},k}\theta_{{\rm c},k+1}$ in (27). Consequently, the filtering used to construct (27) ignores changes in the controller coefficient vector over the window $[k - n_{\rm f}, k]$. The effect of the actual time-dependence of $\theta_{{\rm c},k}$ is analyzed in [21]. Using (21) and (22), $\hat{z}_k(\theta_{\rm c})$ can be expressed as

$$\hat{z}_k(\theta_{\rm c}) = z_k + D(U_{{\rm f},k} - \Phi_{{\rm f},k}\theta_{\rm c}) - N(U_k - \Phi_{{\rm c},k}\theta_{\rm c}). \tag{28}$$

In the case where $G_{\rm f}(\mathbf{q})$ is a finite-impulse-response (FIR) transfer function, and thus $D = 0$, it follows from (28) that

$$\hat{z}_k(\theta_{\rm c}) = z_k - NU_k + N\Phi_{{\rm c},k}\theta_{\rm c}. \tag{29}$$

In order to account for the control effort, define

$$z_{\mathrm{c},k}(\theta_\mathrm{c}) \triangleq \begin{bmatrix} E_z \hat{z}_k(\theta_\mathrm{c}) \\ E_u \phi_{\mathrm{c},k}\theta_\mathrm{c} \end{bmatrix} \in \mathbb{R}^{q+r_1}, \tag{30}$$

where the performance weighting $E_z \in \mathbb{R}^{q\times q}$ is nonsingular, and $E_u \in \mathbb{R}^{r_1\times m}$ is the control weighting. If $E_u = 0$, then all expressions involving E_u in (30), as well as in all subsequent expressions, are omitted, and $r_1 = 0$. Using (26), it follows that (30) can be expressed as

$$z_{\mathrm{c},k}(\theta_\mathrm{c}) = y_{\mathrm{c},k} - \phi_{\mathrm{fc},k}\theta_\mathrm{c}, \tag{31}$$

where

$$y_{\mathrm{c},k} \triangleq \begin{bmatrix} E_z z_k - E_z u_{\mathrm{f},k} \\ 0_{r\times 1} \end{bmatrix} \in \mathbb{R}^{q+r_1}, \quad \phi_{\mathrm{fc},k} \triangleq \begin{bmatrix} -E_z \phi_{\mathrm{f},k} \\ -E_u \phi_{\mathrm{c},k} \end{bmatrix} \in \mathbb{R}^{(q+r_1)\times l_{\theta_\mathrm{c}}}. \tag{32}$$

Using (30), define the retrospective cost

$$J_k(\theta_\mathrm{c}) \triangleq \sum_{i=0}^{k} z_{\mathrm{c},i}(\theta_\mathrm{c})^\mathrm{T} z_{\mathrm{c},i}(\theta_\mathrm{c}) + (\theta_\mathrm{c} - \theta_{\mathrm{c},0})^\mathrm{T} P_{\mathrm{c},0}^{-1}(\theta_\mathrm{c} - \theta_{\mathrm{c},0}), \tag{33}$$

and note that

$$z_{\mathrm{c},k}(\theta_\mathrm{c})^\mathrm{T} z_{\mathrm{c},k}(\theta_\mathrm{c}) = \hat{z}_k(\theta_\mathrm{c})^\mathrm{T} R_z \hat{z}_k(\theta_\mathrm{c}) + \theta_\mathrm{c}^\mathrm{T} \phi_{\mathrm{c},k}^\mathrm{T} R_u \phi_{\mathrm{c},k}\theta_\mathrm{c}, \tag{34}$$

where $R_z \triangleq E_z^\mathrm{T} E_z \in \mathbb{R}^{q\times q}$ is positive definite and $R_u \triangleq E_u^\mathrm{T} E_u \in \mathbb{R}^{m\times m}$ is positive semidefinite. For all $k \geq 0$, the minimizer $\theta_{\mathrm{c},k+1}$ of (33) is given by the recursive least squares (RLS) solution [28]

$$P_{\mathrm{c},k+1} = P_{\mathrm{c},k} - P_{\mathrm{c},k}\phi_{\mathrm{fc},k}^\mathrm{T}(I_{q+r_1} + \phi_{\mathrm{fc},k}P_{\mathrm{c},k}\phi_{\mathrm{fc},k}^\mathrm{T})^{-1}\phi_{\mathrm{fc},k}P_{\mathrm{c},k}, \tag{35}$$

$$\theta_{\mathrm{c},k+1} = \theta_{\mathrm{c},k} + P_{\mathrm{c},k+1}\phi_{\mathrm{fc},k}^\mathrm{T}(y_{\mathrm{c},k} - \phi_{\mathrm{fc},k}\theta_{\mathrm{c},k}). \tag{36}$$

Using the updated controller coefficient vector given by (36), the requested control at step $k+1$ is given by

$$u_{k+1} = \phi_{\mathrm{c},k+1}\theta_{\mathrm{c},k+1}. \tag{37}$$

Although $\theta_{\mathrm{c},0}$ can be chosen arbitrarily, $\theta_{\mathrm{c},0} = 0$ is chosen in all examples in order to reflect the absence of additional modeling information. Finally, $P_{\mathrm{c},0} = p_{\mathrm{c},0} I_{l_{\theta_\mathrm{c}}}$, where $p_{\mathrm{c},0} \in (0,\infty)$ is a tuning parameter.

IV. RLS-Based Identification (RLSID)

This section summarizes RLSID, which uses RLS for online, closed-loop identification. The goal is to estimate key features of the open-loop transfer function $-EG_\mathrm{d}(\mathbf{q})$ from u_k to z_k needed to construct $G_\mathrm{f}(\mathbf{q})$, which, as shown in Section III, serves as the target model for $\widetilde{G}_{z\tilde{u},k}(\mathbf{q})$. The transfer function $EG_\mathrm{d}(\mathbf{q})$ from u_k to $y_{z,k}$ is given by

$$EG_\mathrm{d}(\mathbf{q}) = (I_q\mathbf{q}^n + F_1\mathbf{q}^{n-1} + \cdots + F_n)^{-1}(G_0\mathbf{q}^n + G_1\mathbf{q}^{n-1} + \cdots + G_n), \tag{38}$$

where $G_0, \ldots, G_n \in \mathbb{R}^{q\times m}$ and $F_1, \ldots, F_n \in \mathbb{R}^{q\times q}$ are the numerator and denominator coefficients of the transfer function, respectively.

Consider the sampled-data identification architecture shown in Figure 3, which is based on Figure 2. Since E is known, $y_{z,k} = Ey_k$ can be computed internally by RLSID. Furthermore, at each step k, the requested control input u_k and the measurement y_k are assumed to be available. In order to identify $EG_\mathrm{d}(\mathbf{q})$, a model of the form

$$y_{z,k} = -\sum_{i=1}^{\eta} F_{i,k} y_{z,k-i} + \sum_{i=0}^{\eta} G_{i,k} u_{k-i} \tag{39}$$

is fit to data, where η is the RLSID window length, and $G_{0,k}, \ldots, G_{\eta,k} \in \mathbb{R}^{q\times m}$ and $F_{1,k}, \ldots, F_{\eta,k} \in \mathbb{R}^{q\times q}$ are numerator and denominator coefficient matrices that are to be estimated.

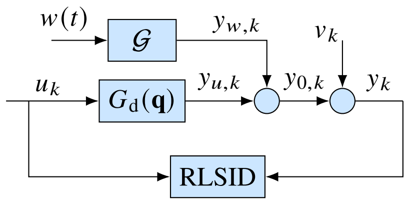

Fig. 3 Online identification using RLSID.

Next, note that (39) can be written as

$$y_{z,k} = \phi_{\mathrm{m},k}\theta_{\mathrm{m},k}, \tag{40}$$

where

$$\phi_{\mathrm{m},k} \triangleq \begin{bmatrix} -y_{z,k-1} \\ \vdots \\ -y_{z,k-\eta} \\ u_k \\ \vdots \\ u_{k-\eta} \end{bmatrix}^{\mathrm{T}} \otimes I_q \in \mathbb{R}^{q\times l_{\theta_\mathrm{m}}}, \tag{41}$$

is the *model regressor*, $l_{\theta_\mathrm{m}} = \eta q^2 + (\eta+1)qm$, and

$$\theta_{\mathrm{m},k} \triangleq \mathrm{vec}\left[\, F_{1,k} \ \cdots \ F_{\eta,k} \ G_{0,k} \ \cdots \ G_{\eta,k} \right] \in \mathbb{R}^{l_{\theta_\mathrm{m}}} \tag{42}$$

is the *model coefficient vector*. The *model-output error* is defined by

$$z_{\mathrm{m},k}(\theta_\mathrm{m}) \triangleq y_{z,k} - \phi_{\mathrm{m},k}\theta_\mathrm{m}, \tag{43}$$

where θ_m is an argument for optimization of the form

$$\theta_\mathrm{m} \triangleq \mathrm{vec}\left[\, F_1 \ \cdots \ F_\eta \ \ G_0 \ \cdots \ G_\eta \right] \in \mathbb{R}^{l_{\theta_\mathrm{m}}}. \tag{44}$$

V. Data-Driven Retrospective Cost Adaptive Control (DDRCAC)

This section summarizes DDRCAC [21], which combines RCAC with RLSID. The online identification uses RLS to fit an infinite-impulse-response (IIR) model based on data $y_{z,k}$ and u_k collected during closed-loop operation. At each step, the identified IIR model is used to construct a time-dependent target model $G_{\mathrm{f},k}(\mathbf{q})$. In particular, $G_{\mathrm{f},k}(\mathbf{q})$ is constructed as an FIR filter whose numerator is chosen to be the numerator of the latest identified IIR model. This approach avoids the need to compute NMP zeros during online operation and can be used in the MIMO case, where the numerator of the RLSID model is a $q\times m$ polynomial matrix. This target model is then used by RCAC to update the coefficients of an IIR controller. For DDRCAC, both RLS implementations use variable-rate forgetting (VRF), as given in [29].

Proposition 1 *For all $k \ge 0$, let $\bar{y}_k \in \mathbb{R}^{l_{\bar y}}$, $\phi_k \in \mathbb{R}^{l_{\bar y}\times l_{\bar\theta}}$, $\lambda_k \in (0,1]$, and define $\rho_k \triangleq \prod_{j=0}^k \lambda_j$. Let $\bar\theta_0 \in \mathbb{R}^{l_{\bar\theta}}$, and let $\bar P_0 \in \mathbb{R}^{l_{\bar\theta}\times l_{\bar\theta}}$ be positive definite. Furthermore, for all $k\ge 0$, denote the minimizer of*

$$J_k(\bar\theta) \triangleq \sum_{i=0}^{k} \frac{\rho_k}{\rho_i}(\bar y_i - \phi_i\bar\theta)^{\mathrm{T}}(\bar y_i - \phi_i\bar\theta) + \rho_k(\bar\theta-\bar\theta_0)^{\mathrm{T}}\bar P_0^{-1}(\bar\theta-\bar\theta_0). \tag{45}$$

where $\bar\theta \in \mathbb{R}^{l_{\bar\theta}}$, by $\bar\theta_{k+1} \triangleq \underset{\bar\theta\,\in\,\mathbb{R}^{l_{\bar\theta}}}{\mathrm{argmin}}\ J_k(\bar\theta)$. Then, for all $k\ge 0$, $\bar\theta_{k+1}$ is given by

$$\bar P_{k+1} = \tfrac{1}{\lambda_k}\bar P_k - \tfrac{1}{\lambda_k}\bar P_k\phi_k^{\mathrm{T}}(\lambda_k I_{l_{\bar y}} + \phi_k\bar P_k\phi_k^{\mathrm{T}})^{-1}\phi_k\bar P_k, \tag{46}$$

$$\bar\theta_{k+1} = \bar\theta_k + \bar P_{k+1}\phi_k^{\mathrm{T}}(\bar y_k - \phi_k\bar\theta_k). \tag{47}$$

For RLSID and RCAC, a technique for specifying λ_k is given later in this section.

A. RLSID

In order to identify $EG_{\mathrm{d}}(\mathbf{q})$, an IIR model of the form (39) is fit to data. Since E is known, $y_{z,k} = Ey_k$ can be computed internally by RLSID. Using Proposition 1, for all $k \geq 0$ the model coefficient vector $\theta_{\mathrm{m},k}$ is updated recursively using

$$P_{\mathrm{m},k+1} = \tfrac{1}{\lambda_{\mathrm{m},k}} P_{\mathrm{m},k} - \tfrac{1}{\lambda_{\mathrm{m},k}} P_{\mathrm{m},k}\phi_{\mathrm{m},k}^{\mathrm{T}}(\lambda_{\mathrm{m},k} I_q + \phi_{\mathrm{m},k} P_{\mathrm{m},k}\phi_{\mathrm{m},k}^{\mathrm{T}})^{-1}\phi_{\mathrm{m},k}P_{\mathrm{m},k}, \tag{48}$$

$$\theta_{\mathrm{m},k+1} = \theta_{\mathrm{m},k} + P_{\mathrm{m},k+1}\phi_{\mathrm{m},k}^{\mathrm{T}}(y_{z,k} - \phi_{\mathrm{m},k}\theta_{\mathrm{m},k}), \tag{49}$$

where $\phi_{\mathrm{m},k}$ and $\theta_{\mathrm{m},k}$ are given by (41) and (42), respectively, and $P_{\mathrm{m},0} \in \mathbb{R}^{l_{\theta_{\mathrm{m}}} \times l_{\theta_{\mathrm{m}}}}$ is positive definite. The RLSID model at step k is given by

$$EG_{\mathrm{d},k}(\mathbf{q}) = (I_q \mathbf{q}^{\eta} + F_{1,k}\mathbf{q}^{\eta-1} + \cdots + F_{\eta,k})^{-1}(G_{0,k}\mathbf{q}^{\eta} + \cdots + G_{\eta,k}). \tag{50}$$

B. RCAC

Define the strictly proper dynamic compensator

$$u_k \overset{\triangle}{=} \mathrm{sat}_{\bar{u}}(\phi_{\mathrm{c},k}\theta_{\mathrm{c},k}), \tag{51}$$

where $\phi_{\mathrm{c},k}$ and $\theta_{\mathrm{c},k}$ are given by (10) and (11), respectively. The definition (51) represents an IIR controller whose output is saturated component-wise by the scalar saturation function $\mathrm{sat}_{\bar{u}}$ defined by

$$\mathrm{sat}_{\bar{u}_i}(x_i) \overset{\triangle}{=} \begin{cases} x_i, & |x_i| < \bar{u}_i, \\ \mathrm{sign}(x_i)\bar{u}_i, & |x_i| \geq \bar{u}_i. \end{cases} \tag{52}$$

Next, define the filtered signals

$$u_{\mathrm{f},k} \overset{\triangle}{=} G_{\mathrm{f},k}(\mathbf{q})u_k, \tag{53}$$

$$\phi_{\mathrm{f},k} \overset{\triangle}{=} G_{\mathrm{f},k}(\mathbf{q})\phi_{\mathrm{c},k}, \tag{54}$$

where, for startup, $u_{\mathrm{f},k}$ and $\phi_{\mathrm{f},k}$ are initialized at zero and thus are computed as the forced responses of (53) and (54), respectively, and where $G_{\mathrm{f},k}(\mathbf{q})$ is the time-dependent target model constructed using the updated numerator coefficients $G_{0,k+1}, \ldots, G_{\eta,k+1}$ of the model (39). In particular,

$$G_{\mathrm{f},k}(\mathbf{q}) \overset{\triangle}{=} -\sum_{i=0}^{\eta} G_{i,k+1}\frac{1}{\mathbf{q}^i}, \tag{55}$$

In the case where $q = m = 1$, it follows from $G_{0,k} = \cdots = G_{\xi-1,k} = 0$ and $G_{\xi,k} = G_{\xi}$ that (55) and $-EG_{\mathrm{d}}(\mathbf{q})$ have the same leading numerator coefficient and relative degree. Note that, at each step k, the numerator of (55) is chosen to be the numerator of (50). If there exists $k \geq 0$ such that $G_{0,k} = \cdots = G_{\eta,k} = 0_{q\times m}$, then $G_{\mathrm{f},k}(\mathbf{q})$ is chosen to be

$$G_{\mathrm{f},k}(\mathbf{q}) \overset{\triangle}{=} -\mathbf{1}_{q\times m}. \tag{56}$$

The retrospective performance variable is defined to be

$$\hat{z}_k(\theta_{\mathrm{c}}) \overset{\triangle}{=} z_k - u_{\mathrm{f},k} + \phi_{\mathrm{f},k}\theta_{\mathrm{c}}. \tag{57}$$

Using (55) and (56), (57) can be expressed as

$$\hat{z}_k(\theta_{\mathrm{c}}) \overset{\triangle}{=} z_k - N_k U_k + N_k \Phi_{\mathrm{c},k}\theta_{\mathrm{c}}. \tag{58}$$

where

$$N_k \overset{\triangle}{=} \begin{cases} \begin{bmatrix} -\mathbf{1}_{q\times m} \; 0 \; \cdots \; 0 \end{bmatrix}, & G_{0,k+1} = \cdots = G_{\eta,k} = 0, \\ \begin{bmatrix} -G_{0,k+1} \; \cdots \; -G_{\eta,k+1} \end{bmatrix}, & \text{otherwise}, \end{cases} \tag{59}$$

$N_k \in \mathbb{R}^{q\times(\eta+1)m}$, U_k and $\Phi_{\mathrm{c},k}$ are given by (23) and (24) with $n_\mathrm{f} = \eta$, respectively, and $G_{0,k+1}, \ldots, G_{\eta,k+1} \in \mathbb{R}^{q\times m}$ are the numerator coefficients of the RLSID model. Note that, by performing the RLSID update at step k before the RCAC update, it follows thus the estimated numerator coefficients $G_{0,k+1}, \ldots, G_{\eta,k+1}$ are available for constructing N_k at step k.

Next, define the *controller cost variable*

$$z_{\mathrm{c},k}(\theta_\mathrm{c}) \triangleq \begin{bmatrix} E_z \hat{z}_k(\theta_\mathrm{c}) \\ E_u \phi_{\mathrm{c},k}\theta_\mathrm{c} \\ E_{\Delta u}(\phi_{\mathrm{c},k}\theta_\mathrm{c} - u_k) \end{bmatrix} \in \mathbb{R}^{q+r_1+r_2}, \tag{60}$$

where the performance weighting $E_z \in \mathbb{R}^{q\times q}$ is nonsingular and $E_u \in \mathbb{R}^{r_1\times m}$ and $E_{\Delta u} \in \mathbb{R}^{r_2\times m}$ are the control weighting and control-move weighting, respectively. If $E_u = 0$ and $E_{\Delta u} = 0$, then $r_1 = 0$ and $r_2 = 0$, respectively, and all expressions involving E_u and $E_{\Delta u}$ are omitted from (60), as well as from all subsequent expressions. Note that

$$z_{\mathrm{c},k}(\theta_\mathrm{c})^\mathrm{T} z_{\mathrm{c},k}(\theta_\mathrm{c}) = \hat{z}_k(\theta_\mathrm{c})^\mathrm{T} R_z \hat{z}_k(\theta_\mathrm{c}) + \theta_\mathrm{c}^\mathrm{T}\phi_{\mathrm{c},k}^\mathrm{T} R_u \phi_{\mathrm{c},k}\theta_\mathrm{c} + (\phi_{\mathrm{c},k}\theta_\mathrm{c} - u_k)^\mathrm{T}\phi_{\mathrm{c},k}^\mathrm{T} R_{\Delta u}\phi_{\mathrm{c},k}(\phi_{\mathrm{c},k}\theta_\mathrm{c} - u_k), \tag{61}$$

where $R_z \triangleq E_z^\mathrm{T} E_z \in \mathbb{R}^{q\times q}$ is positive definite, and $R_u \triangleq E_u^\mathrm{T} E_u \in \mathbb{R}^{m\times m}$, $R_{\Delta u} \triangleq E_{\Delta u}^\mathrm{T} E_{\Delta u} \in \mathbb{R}^{m\times m}$ are positive semidefinite. Using Proposition 1, for all $k \geq 0$ the controller coefficient vector $\theta_{\mathrm{c},k}$ is updated recursively using

$$P_{\mathrm{c},k+1} = \tfrac{1}{\lambda_{\mathrm{c},k}} P_{\mathrm{c},k} - \tfrac{1}{\lambda_{\mathrm{c},k}} P_{\mathrm{c},k}\phi_{\mathrm{fc},k}^\mathrm{T}(\lambda_{\mathrm{c},k} I_{q+r_1+r_2} + \phi_{\mathrm{fc},k} P_{\mathrm{c},k}\phi_{\mathrm{fc},k}^\mathrm{T})^{-1}\phi_{\mathrm{fc},k} P_{\mathrm{c},k}, \tag{62}$$

$$\theta_{\mathrm{c},k+1} = \theta_{\mathrm{c},k} + P_{\mathrm{c},k+1}\phi_{\mathrm{fc},k}^\mathrm{T}(y_{\mathrm{c},k} - \phi_{\mathrm{fc},k}\theta_{\mathrm{c},k}), \tag{63}$$

where

$$y_{\mathrm{c},k} \triangleq \begin{bmatrix} E_z z_k - E_z N_k U_k \\ 0 \\ -E_{\Delta u} u_k \end{bmatrix} \in \mathbb{R}^{q+r_1+r_2}, \quad \phi_{\mathrm{fc},k} \triangleq \begin{bmatrix} -E_z N_k \Phi_{\mathrm{c},k} \\ -E_u \phi_{\mathrm{c},k} \\ -E_{\Delta u}\phi_{\mathrm{c},k} \end{bmatrix} \in \mathbb{R}^{(q+r_1+r_2)\times l_{\theta_\mathrm{c}}}. \tag{64}$$

and $P_{\mathrm{c},0} \in \mathbb{R}^{l_{\theta_\mathrm{c}}\times l_{\theta_\mathrm{c}}}$ is positive definite.

For all of the examples in this paper, $\theta_{\mathrm{m},k}$ and $\theta_{\mathrm{c},k}$ are initialized at 0, and thus (56) is invoked at startup. This assumption reflects the absence of additional prior modeling information; in practice, however, $\theta_{\mathrm{m},k}$ and $\theta_{\mathrm{c},k}$ can be initialized based on any available modeling information. To initialize RLSID and RCAC, $P_{\mathrm{c},0} = p_{\mathrm{c},0} I_{l_{\theta_\mathrm{c}}}$ and $P_{\mathrm{m},0} = p_{\mathrm{c},0} I_{l_{\theta_\mathrm{m}}}$ are chosen, where, for convenience, $p_{\mathrm{c},0} > 0$ is a common tuning parameter.

C. Data-Dependent Variable Rate Forgetting

For data-dependent variable-rate forgetting, set

$$\lambda_{\mathrm{m},k} = \frac{1}{1 + \varepsilon e(z_{\mathrm{m},k-\tau_\mathrm{d}}, \ldots, z_{\mathrm{m},k})\mathbf{1}[e(z_{\mathrm{m},k-\tau_\mathrm{d}}, \ldots, z_{\mathrm{m},k})]}, \tag{65}$$

$$\lambda_{\mathrm{c},k} = \frac{1}{1 + \varepsilon e(z_{k-\tau_\mathrm{d}}, \ldots, z_k)\mathbf{1}[e(z_{k-\tau_\mathrm{d}}, \ldots, z_k)]}, \tag{66}$$

where

$$e(x_{k-\tau_\mathrm{d}}, \ldots, x_k) \triangleq \frac{\sqrt{\frac{1}{\tau_\mathrm{n}}\sum_{i=k-\tau_\mathrm{n}}^{k} x_i^\mathrm{T} x_i}}{\sqrt{\frac{1}{\tau_\mathrm{d}}\sum_{i=k-\tau_\mathrm{d}}^{k} x_i^\mathrm{T} x_i}} - 1.2, \tag{67}$$

"$\mathbf{1}$" is the step function that is 0 for negative arguments and 1 for nonnegative arguments, and $e(0, \ldots, 0) \triangleq 0$. In (65)–(67), $\varepsilon \geq 0$, $0 < \tau_\mathrm{n} < \tau_\mathrm{d}$ are numerator and denominator window lengths, respectively. If the sequence $x_{k-\tau_\mathrm{d}}, \ldots, x_k$ is zero-mean noise, then the numerator and denominator of (67) approximate the average standard deviation of the noise over the intervals $[k - \tau_\mathrm{n}, k]$ and $[k - \tau_\mathrm{d}, k]$, respectively. In particular, by choosing $\tau_\mathrm{d} >> \tau_\mathrm{n}$, it follows that the denominator of (67) approximates the long-term-average standard deviation of x_k, whereas the numerator of (67) approximates the short-term-average standard deviation of x_k. Consequently, the case $e(x_{k-\tau_\mathrm{d}}, \ldots, x_k) > 0$ implies that the short-term-average standard deviation of x_k is greater than the long-term-average standard deviation of x_k plus a threshold of 0.2. The function $e(x_{k-\tau_{\rm d}},\ldots,x_k)$ used in VRF suspends forgetting when the short-term-average standard deviation of x_k drops below 1.2 times the long-term-average standard deviation of x_k. This technique thus prevents forgetting in RLSID and RCAC due to zero-mean sensor noise with constant standard deviation rather than due to the magnitude of the noise-free identification error and command-following error.

A list of parameters to be selected for RCAC is presented in Table 1.

Table 1 **Tuning parameters that need to be selected for DDRCAC. Typical ranges are given.**

Parameter	Description	Selection
η	Model window length	Integer ≥ 1 (1–10)
$n_{\rm c}$	Controller window length	Integer ≥ 1 (2–40)
E_u	Control weighting	scaled $m \times m$ identity
$E_{\Delta u}$	Control move weighting	scaled $m \times m$ identity
$\bar{u}$	Control saturation-limit vector	95% actuator saturation limit
$p_{\rm c,0}$	Initial RLS covariance scaling for RLSID and RCAC	$p_{\rm c,0} > 0$
ε	Forgetting parameter	$0 \leq \varepsilon < 1$ (0.001 – 0.2)
$\tau_{\rm n}$, $\tau_{\rm d}$	Forgetting window lengths	Integers $\tau_{\rm d} > \tau_{\rm n}$ ($\tau_{\rm n} \in$ [1–400], $\tau_{\rm d} \sim 3\tau_{\rm n}$)

VI. Fluent and DDRCAC Interface

A Fluent User Defined Function (UDF) is used to couple Ansys Fluent with the MATLAB-based DDRCAC controller. This integration employs a TCP WinSocket script for bidirectional communication between the Fluent UDF (acting as the client) and the DDRCAC MATLAB script (serving as the server) at each iteration.

Before the Fluent simulation is run, a server script is compiled and run to initiate the DDRCAC server. The DEFINE_ON_DEMAND macro in this script reads the sensor location coordinates. Since Fluent stores flow data at mesh cell centroids, which might not correspond to sensor locations, this function identifies the nearest mesh cell to the selected sensor locations. The UDF client connects to the server prior to the fluid computation.

At the conclusion of each fluid simulation time step, the DEFINE_AT_END macro is called. This macro computes the flow properties at the sensor location via linear interpolation of the neighboring mesh cell values and gradients (when available). This data are then transferred to the DDRCAC server via TCP communication. The server processes and loads this controller data with the MATLAB Engine API. It then calls the DDRCAC MATLAB script, and the server then sends the resulting control command back to the DDRCAC/Fluent interface. A DEFINE_PROFILE macro, called at the start of each time step, sets a uniform velocity profile across the control inlet based on the requested control from DDRCAC.

VII. Application of DDRCAC to Boundary Layer Control

In this section, the flow-control objective is to have the fluid follow a constant streamwise velocity setpoint at a sensor location. To achieve this objective, DDRCAC is used to specify the free-stream inlet velocity u_k. As in all of the examples in this paper, no model of the fluid dynamics is assumed to be available to DDRCAC; rather, RLSID learns a linear model, which is used by RCAC.

Example 1. *Streamwise velocity setpoint in a 2D duct boundary layer.* We consider a 2D duct with width 0.5 m and length 1 m. A setpoint is specified for the streamwise velocity at a specified location that is halfway along the length of the duct and within the expected boundary layer. This geometry is defined with a symmetry boundary condition along the midheight plane of the duct. A velocity inlet is set along the left edge, and a pressure outlet is placed along the right edge. The bottom edge is a no-slip wall. A single idealized streamwise velocity sensor measures the streamwise flow velocity y_k at the specified location. This data is used to compute the command-following error z_k.

The control input requested by RCAC is the magnitude of the uniform velocity profile at the duct inlet. The fluid computation is incompressible, unsteady, viscous, and laminar. The viscosity is 1.7894e-5 kg/(ms). The velocity of the flow field in the duct is initialized to be 0 m/s at all locations. The flow state is then advanced forward in time with a time step of 0.01 s using first order implicit time stepping. RCAC uses a sampled-data architecture discussed in Section II that samples the velocity sensor with a matching time step of 0.01 s.

Figure 4 shows the sensor measurements and requested control input as well as the coefficients estimated by RLSID and RCAC. Figure 5 shows the boundary layer profile at the location of the sensor. Note that the sensor is within the boundary layer.

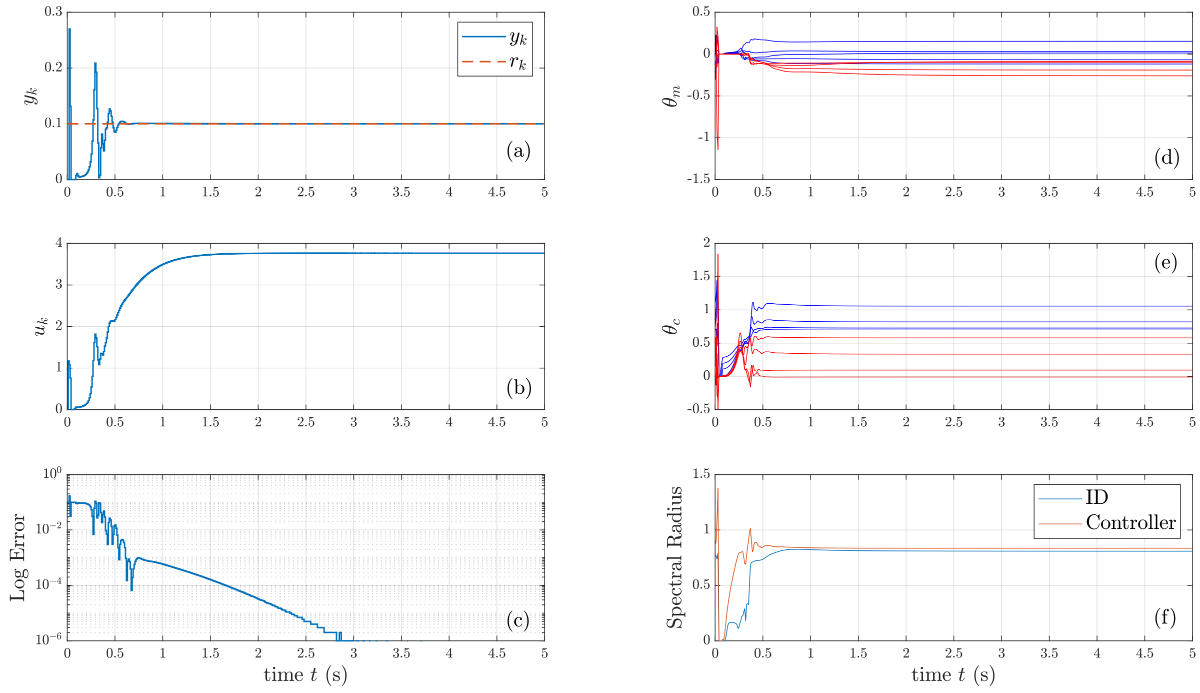

Fig. 4 Example 1: After an initial learning period, (a) shows that the streamwise velocity converges to the commanded value of 0.1 m/s in about 1 s. (b) shows that the resulting converged inlet velocity is 3.7646 m/s, in agreement with the classical Blasius equation. (c) shows the log of the error between the commanded value and the streamwise velocity. (d) shows the time history of the estimated coefficients θ_{m} of the identified model, where numerator coefficients are plotted in blue and denominator coefficients are plotted in red. (e) shows the time history of the updated controller coefficients θ_{c}. (f) shows the spectral radius of the identified model and controller. Note that both RLS updates converge in about 0.75 s.

As an independent check of the simulation and asymptotic performance, we use the classical Blasius laminar boundary layer analysis to determine the required inlet velocity that produces the setpoint velocity at the sensor location. In particular, the Blasius solution is governed by [30]

$$f''' + \frac{1}{2} f f'' = 0, \tag{68}$$

with the boundary conditions

$$f(0) = 0, \quad f'(0) = 0, \quad f'(\infty) = 1. \tag{69}$$

For these boundary conditions, we solve (68) iteratively until $f'(\infty) = 1$, which yields $f''(0) = 0.3320$. The sensor location in the boundary layer is specified by the nondimensional parameter η given by

$$\eta = y\sqrt{\frac{u_\infty}{\nu x}}, \tag{70}$$

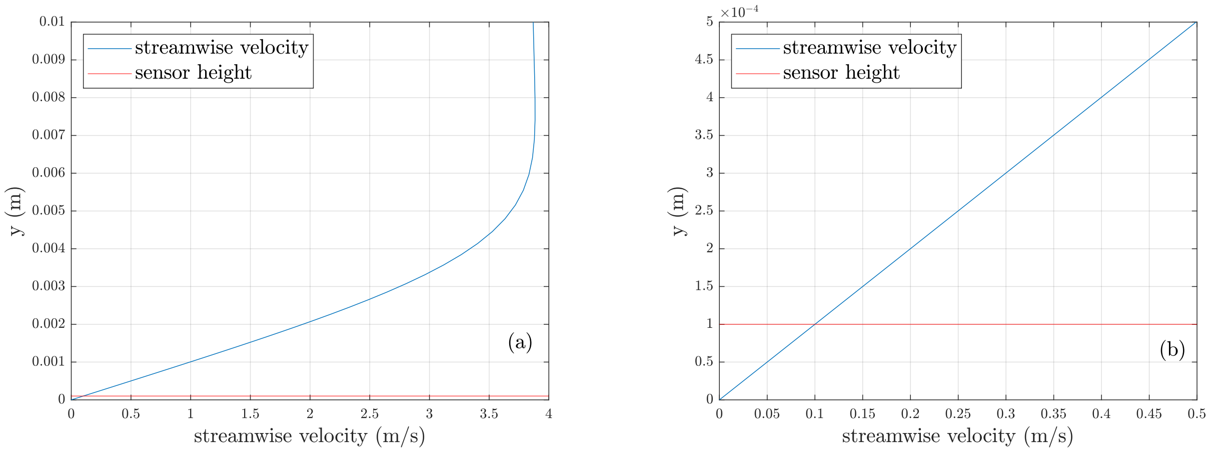

Fig. 5 Example 1: Boundary layer profile at the sensor location at step $k = 500$. (a) shows the boundary layer profile, which indicates that the sensor height of 1e-4 m places it well within the boundary layer. (b) is zoomed in to show that the streamwise velocity at the sensor height is 0.1 m/s, matching the setpoint.

where y is the distance from the wall, u_∞ is the free-stream velocity, ν is the kinematic viscosity, and x is the distance along the plate from the leading edge to the sensor. Solving the Blasius equation yields the velocity profile in the boundary layer. We then find the velocity at the sensor location, which is given by

$$u = f'(\eta)u_\infty. \tag{71}$$

To determine the inlet velocity that results in the setpoint velocity at the sensor location, we substitute the setpoint r_k for the velocity at sensor location u and substitute the inlet velocity u_k for the freestream velocity u_∞. We then initialize inlet velocity u_k and then use an iterative method to compute the desired velocity r_k. Figure 6 shows the relationship between the setpoint velocity at the sensor location and the required inlet velocity obtained from the Blasius solution.

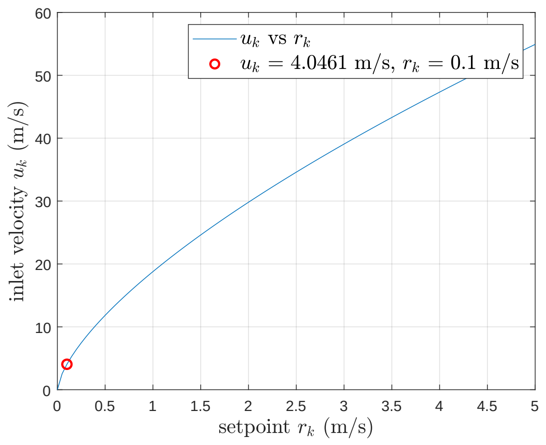

Fig. 6 Example 1: Required inlet velocity versus the setpoint at the sensor location computed by the Blasius flat plate laminar boundary layer theory. The red dot indicates the setpoint for Example 1. For the setpoint 0.1 m/s at the sensor location, Blasius theory yields the required inlet velocity 4.0461 m/s. RCAC converges to 3.7646 m/s.

The above procedure results in a required inlet velocity of 4.0461 m/s. This is close to the RCAC converged value of 3.7646 m/s. Note that RCAC achieves this value with no knowledge of the fluid dynamics and no model of the flow being controlled. We note that an exact match is not expected since the simulation does not adhere exactly to flat plate laminar boundary layer theory; in particular, this case is not a pure flat plate but rather a duct. Additionally, Re_x at the sensor location is 1.29e5 using the converged inlet velocity. The Blasius solution is more accurate with higher Reynolds numbers flows.

Example 2. *Stepwise-varying streamwise velocity setpoint in a 2D duct boundary layer.* We extend Example 1 and use DDRCAC to follow a stepwise variable setpoint. This example uses an identical setup to Example 1. Figure 7 shows the sensor measurements, setpoint, requested control input, and coefficients estimated by RLSID and RCAC. As in Example 1, the controller specifies the inlet velocity to achieve the commanded velocity at the sensor location.

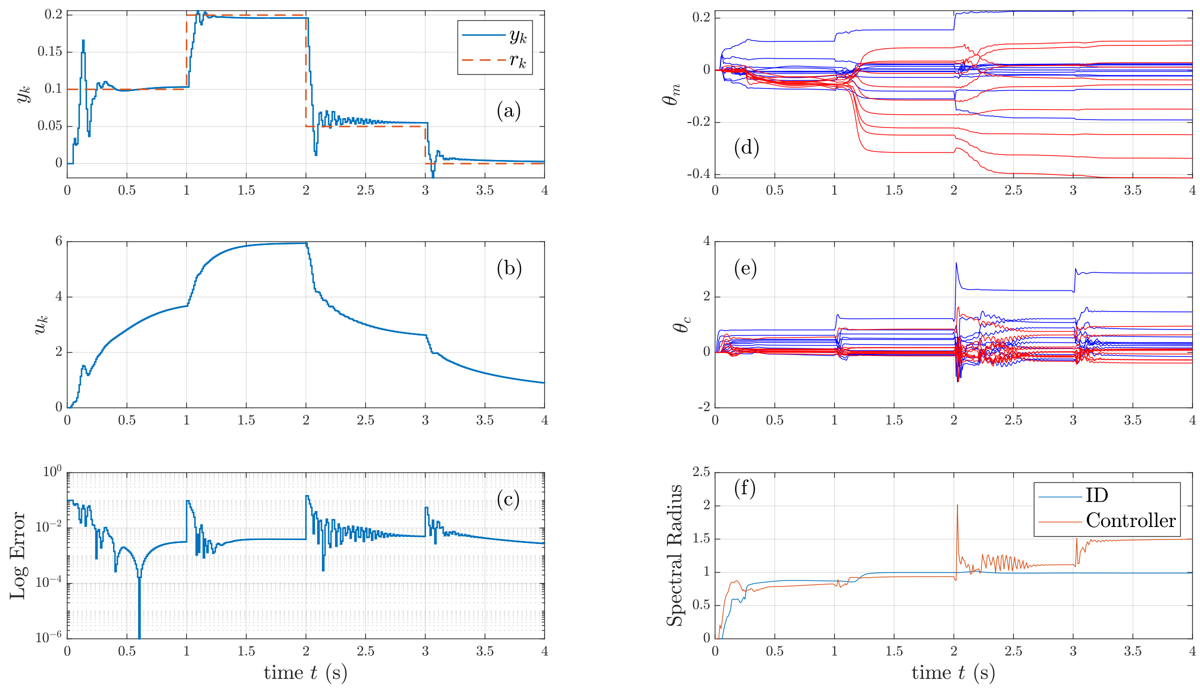

Fig. 7 Example 2: After an initial learning period and transient response occurring each time the setpoint changes, (a) shows that the streamwise velocity converges to the commanded value after about 0.5 s. (b) shows the corresponding inlet velocities. (c) shows the log of the error between the setpoint and the streamwise velocity. (d) shows the time history of the estimated coefficients $\theta_{\rm m}$ of the identified model, where numerator coefficients are plotted in blue and denominator coefficients are plotted in red. (e) shows the time history of the updated controller coefficients $\theta_{\rm c}$. (f) shows the spectral radius of the identified model and the controller. Note that, after each setpoint change, both RLS updates converge in about 0.5 s.

Example 3. *Streamwise velocity setpoint in a cylinder boundary layer in a 2D duct.* In this example, we command the streamwise velocity at a location within the boundary layer on the front of an axial cylinder within a duct. We use the same duct geometry from Example 1, with the addition of an axial cylinder with a radius of 0.1 m. This geometry is shown in Figure 8.

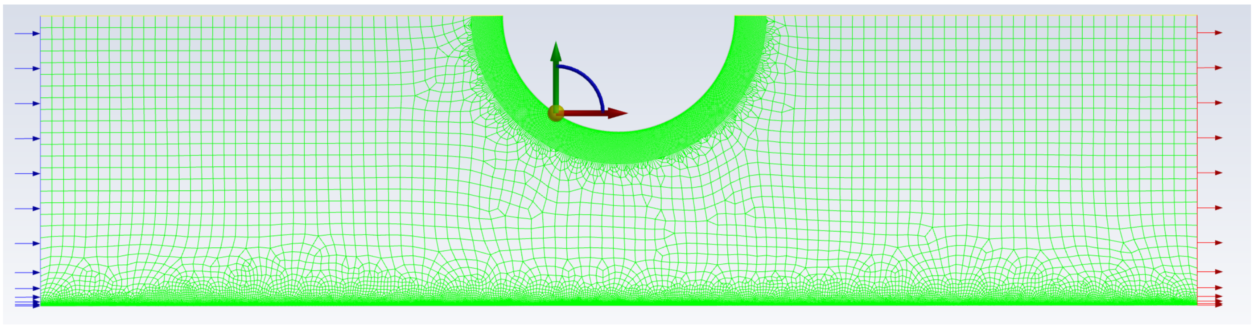

Fig. 8 Example 3: Axial cylinder in duct geometry and mesh with sensor location indicated. A symmetry condition is applied at the top edge of the geometry, resulting in an axial cylinder centered within the duct.

This case is a more complex version of Example 1 since the required free-stream velocity is more difficult to compute from theory. The flow field is initialized to have 0 m/s velocity at all locations within the duct. The fluid computation is incompressible, unsteady, viscous, and turbulent using the $k - \omega$ turbulence model. The viscosity is 1.7894e-5 kg/(ms). At the inlet, the turbulent intensity is 5% and the turbulent viscosity ratio is 10. The velocity of the flow field in the duct is initialized to be 0 m/s at all locations. The flow state is then advanced forward in time with a time step of 0.01 s using first order implicit time stepping. RCAC uses a sampled-data architecture discussed in Section II with a matching time step of 0.01 s.

A single idealized streamwise velocity sensor measures the streamwise flow velocity y_k within the boundary layer at the sensor location indicated in Figure 8. For this example, we consider the streamwise direction to be with respect to the inlet and not tangent to the cylinder surface. This data is used to compute the performance variable z_k for RCAC. Figure 9 shows that DDRCAC determines the inlet velocity needed to achieve the commanded velocity at the sensor location.

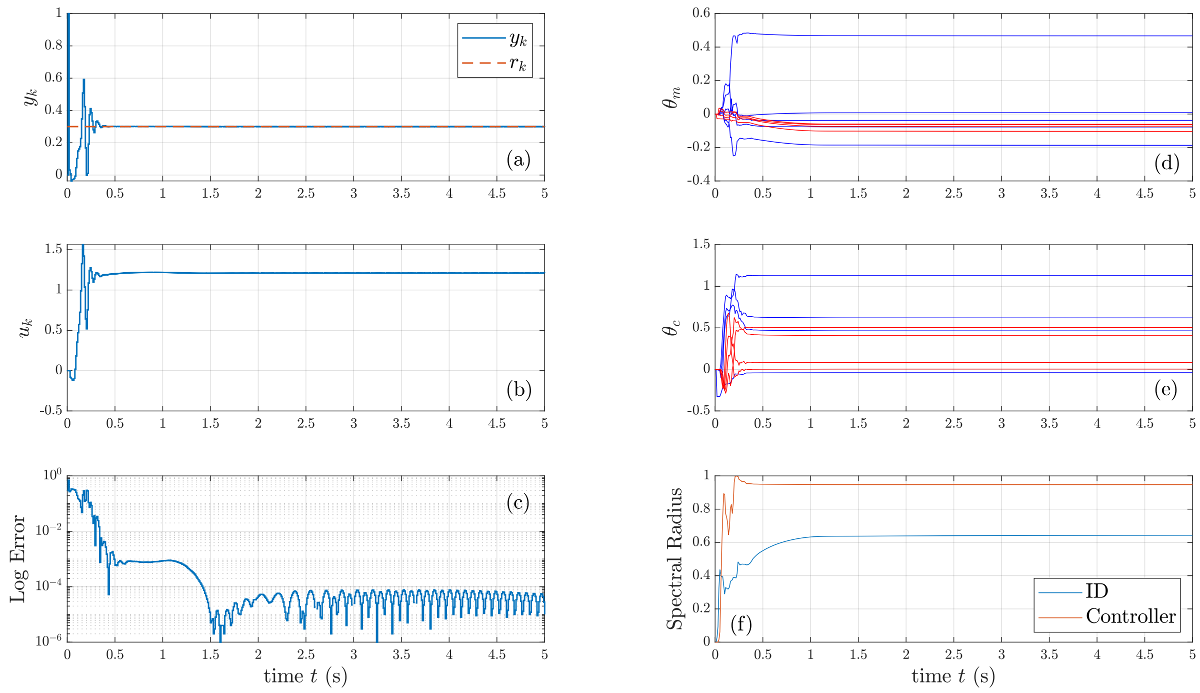

Fig. 9 Example 3: After an initial learning period, (a) shows that the streamwise velocity converges to the commanded value of 0.3 m/s in about 0.5 s. (b) shows that the resulting converged inlet velocity is 1.2099 m/s. (c) shows the log of the error between the commanded value and the streamwise velocity. (d) shows the time history of the estimated coefficients $\theta_{\rm m}$ of the identified model, where numerator coefficients are plotted in blue and denominator coefficients are plotted in red. (e) shows the time history of the updated controller coefficients $\theta_{\rm c}$. (f) shows the spectral radius of the identified model and controller. Note that both RLS updates converge in about 1 s.

VIII. Application of RCAC to Turbulence Control in Mixing Layers

In this section, the flow-control objective is to minimize turbulence at the sensor location. In contrast to Section VII, these examples have a constant, uncontrolled inlet velocity. To achieve the control objective, RCAC is used to specify a control inlet velocity u_k used in the presence of the uncontrolled inlet. The turbulence at the sensor location is assessed by an idealized measurement y_k of turbulent kinetic energy (TKE). TKE is the mean kinetic energy per unit mass due to turbulent eddies in the flow and can be computed using measurements of flow velocity fluctuations in time. In these Reynolds-averaged simulations using the $k-\omega$ model, the turbulent kinetic energy is one of the state variables. Once again, no model of the fluid dynamics is assumed to be available; RLSID learns a linear model, which is used by RCAC.

Example 4. *Turbulent kinetic energy suppression in a mixing layer due to a backwards-facing step in 2D duct.* For flow over a backwards-facing step, the objective is to use DDRCAC to suppress turbulence in the resulting mixing layer. We retain the same duct geometry from Example 1, but we now add a backwards-facing step of length 0.2 m and height 0.075 m at the inlet side of the duct. A constant, uniform inlet velocity is specified with velocity 5 m/s along the left edge of the duct above the step. The horizontal surface of the step and the lower surface of the duct are no-slip walls, the upper surface of the duct is a symmetry plane, and the right edge is a pressure outlet. The vertical edge of the step is the control inlet with velocity requested by RCAC. The viscosity is 1.7894e-5 kg/(ms). At all inlets, the turbulent intensity is 5% and the turbulent viscosity ratio is 10.

The flow field is initialized to have 0 m/s velocity at all locations within the duct. The fluid computation is incompressible, unsteady, viscous, and turbulent using the $k-\omega$ turbulence model. A single idealized TKE sensor provides the TKE y_k. This data is used to compute the performance variable z_k for RCAC. The flow state is advanced forward in time with a time step of 0.01 s using first order implicit time stepping. RCAC uses a sampled-data architecture discussed in Section II with a matching time step of 0.01 s.

A mixing layer forms due to the difference in velocity between the flow above and below the step. Figure 10 shows the TKE contour for the uncontrolled flow with the sensor location indicated. Meanwhile, Figure 11 shows flow pathlines for the uncontrolled flow.

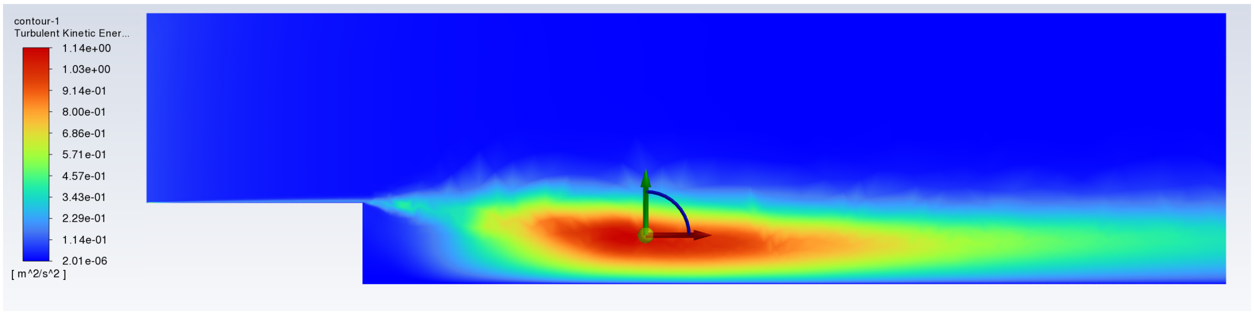

Fig. 10 Example 4: Turbulent kinetic energy contour and sensor location for the uncontrolled flow. A region of high TKE forms due to shear between the layers of fast-moving air above the step and slow-moving air below the step. The TKE sensor is located in this region.

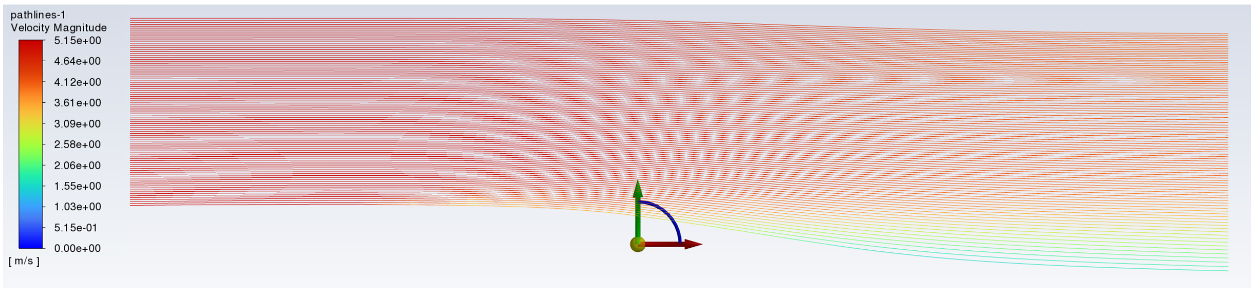

Fig. 11 Example 4: Pathlines in the uncontrolled flow. A region of circulation appears near where the TKE is high.

For the controlled flow, we specify a TKE setpoint of 0 J/kg. The controller is enabled at 1 s. Figure 12 shows the TKE values at the sensor location, the controlled inlet velocity requested by RCAC, and the RLS coefficients from RLSID and RCAC. Figure 13 shows the TKE contour for the controlled flow. Figure 14 shows pathlines in the controlled flow.

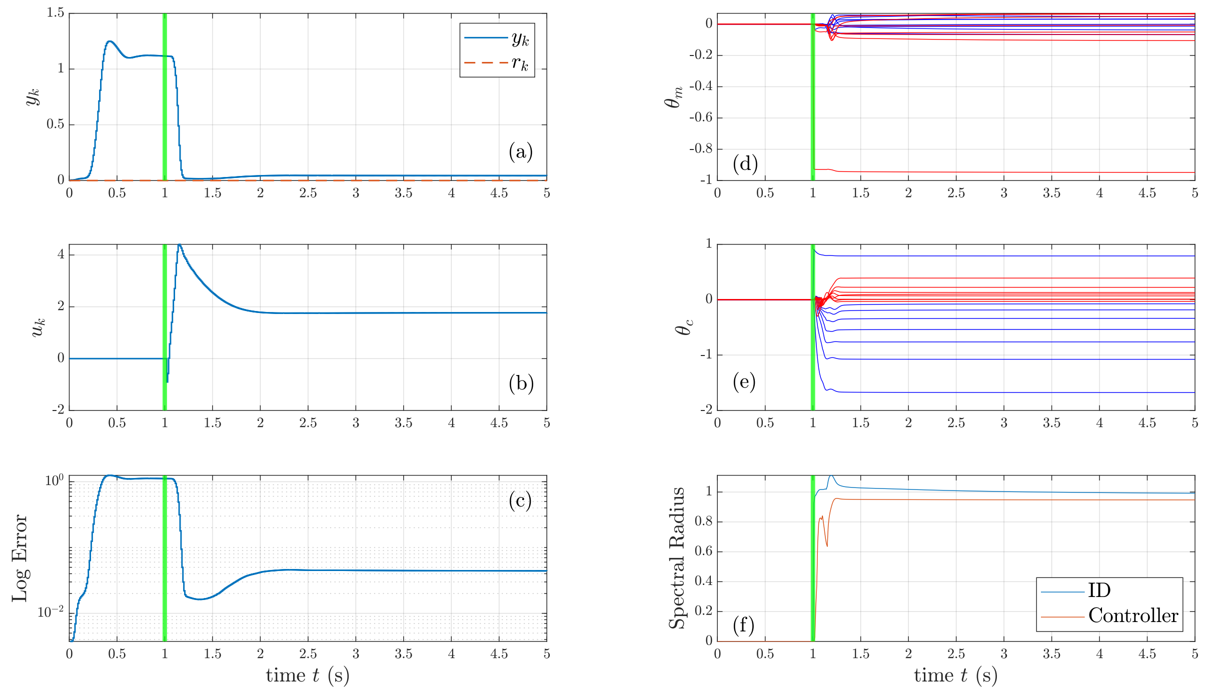

Fig. 12 Example 4: The controller is enabled at 1 s (indicated by the green line). Once enabled, the controller begins learning. After this learning period, (a) shows that the TKE at the sensor location is reduced from the uncontrolled value of 1.11 J/kg to 0.0441 J/kg. (b) shows that the resulting converged inlet velocity is 1.7688 m/s. (c) shows the log of the error between the setpoint and the TKE at the sensor location. (d) shows the time history of the estimated coefficients $\theta_{\rm m}$ of the identified model, where numerator coefficients are plotted in blue and denominator coefficients are plotted in red. (e) shows the time history of the updated controller coefficients $\theta_{\rm c}$. (f) shows the spectral radius of the identified model and controller. Note that both RLS updates converge in about 1 s.

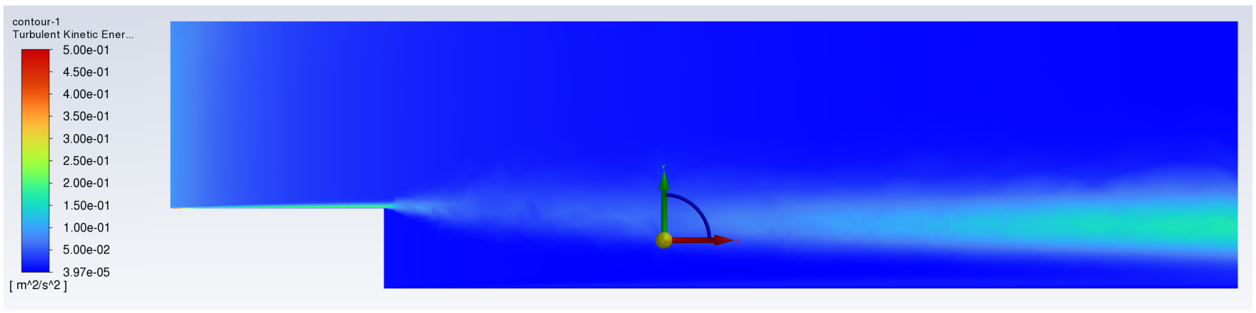

Fig. 13 Example 4: Turbulent kinetic energy contour and sensor location for the controlled flow. RCAC reduces the TKE at the sensor location by requesting the flow velocity through the vertical step edge. RCAC does not decrease TKE away from the sensor; TKE is higher in other locations, particularly at the step face and downstream of the sensor.

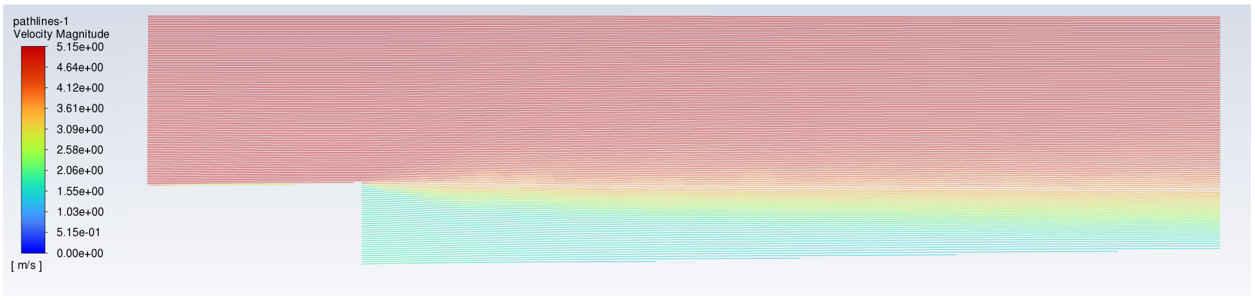

Fig. 14 Example 4: Pathlines in the controlled flow. The circulation region near the sensor in the uncontrolled flow is removed in the controlled flow.

Example 5. *Turbulent kinetic energy suppression in a 2D duct with crossflow jets.* We apply DDRCAC to a duct with width 0.25 m and length 1 m. The top and bottom duct surfaces are no-slip walls. The left surface is an uncontrolled velocity inlet with constant velocity 10 m/s. The right surface is a pressure outlet. On the bottom surface, 0.333 m downstream of the inlet is an uncontrolled jet with constant velocity 50 m/s. On the top surface, 0.500 m downstream of the inlet is a velocity inlet control jet controlled by DDRCAC. The viscosity is 1.7894e-5 kg/(ms). At all inlets, the turbulent intensity is 5% and the turbulent viscosity ratio is 10.

Figure 15 shows the duct geometry and the location of the uncontrolled constant jet and the control jet. Using a single idealized TKE sensor, the control objective is to minimize TKE at the sensor location. The flow field is initialized to have 0 m/s velocity at all locations within the duct. The fluid computation is incompressible, unsteady, viscous, and turbulent using the $k-\omega$ turbulence model. A single idealized sensor provides the TKE y_k. This data are used to compute the performance variable z_k for RCAC. The flow state is advanced forward in time with a time step of 0.01 s using first order implicit time stepping. RCAC uses a sampled-data architecture discussed in Section II with a matching time step of 0.01 s.

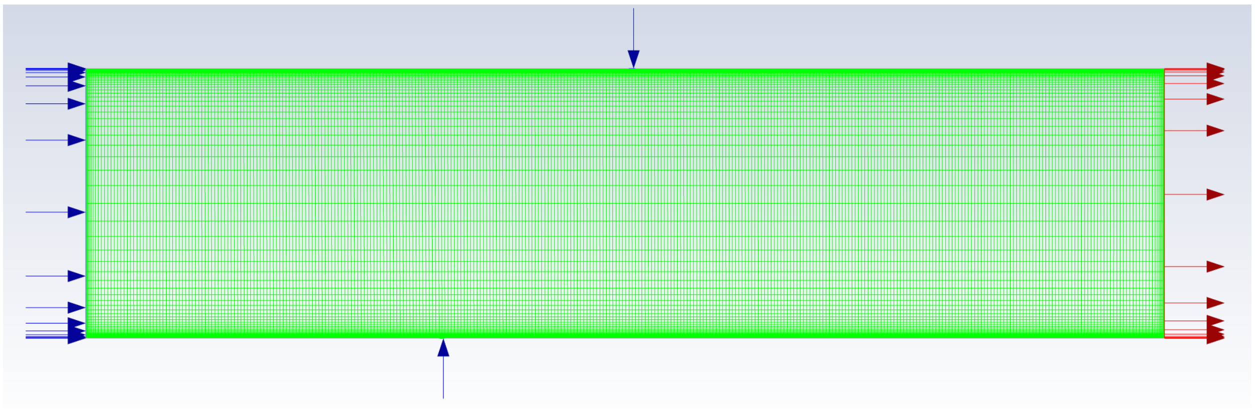

Fig. 15 Example 5: Duct geometry and mesh. The constant jet is located on the bottom surface of the duct, and the control jet is located on the top surface of the duct.

The constant velocity jet introduces turbulence into the duct. Figure 16 shows the TKE contour for the uncontrolled flow with the sensor location indicated. Meanwhile, Figure 17 shows flow pathlines for the uncontrolled flow.

For the controlled flow, we specify a setpoint of 0 J/kg TKE at the sensor location. The controller is enabled at 0.5 s, allowing for the flow to develop. Figure 18 shows the TKE measurement, controlled inlet velocity requested by RCAC, and RLS coefficients from RLSID and RCAC. Figure 19 shows the contour of TKE for the converged controlled flow.

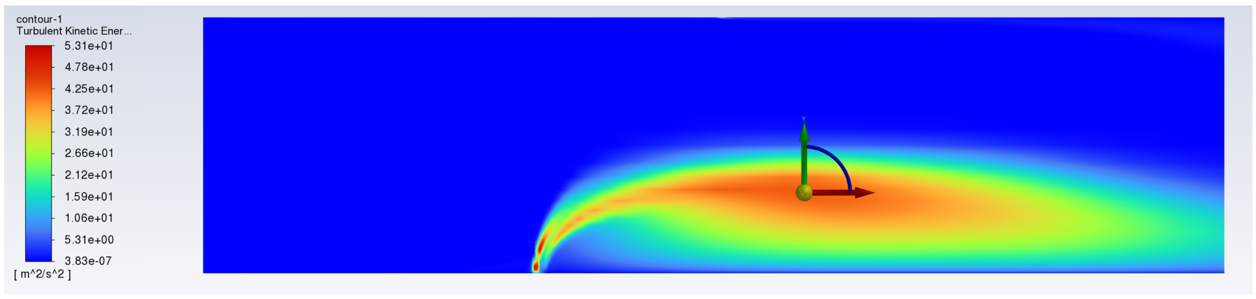

Fig. 16 Example 5: Turbulent kinetic energy contour and sensor location for the uncontrolled flow. A region of high TKE forms due the constant, uncontrolled crossflow jet. At the location of the sensor, the TKE is 42.1 J/kg.

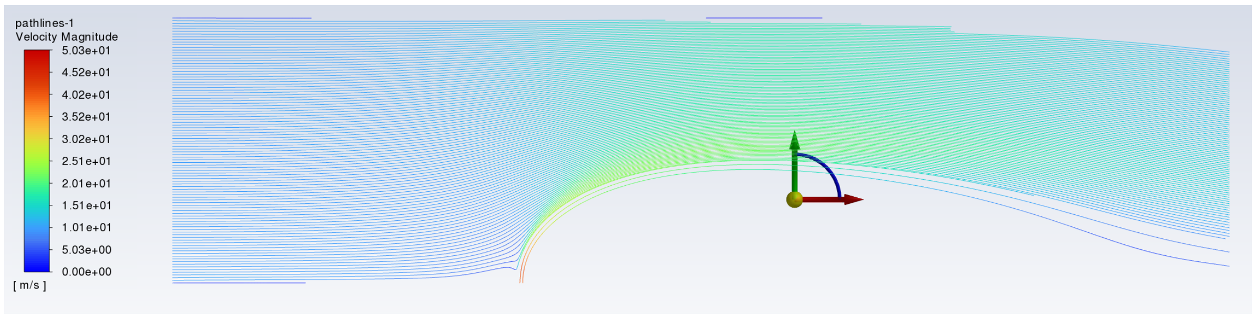

Fig. 17 Example 5: Pathlines in the uncontrolled flow. A region of circulation occurs near where the TKE is high.

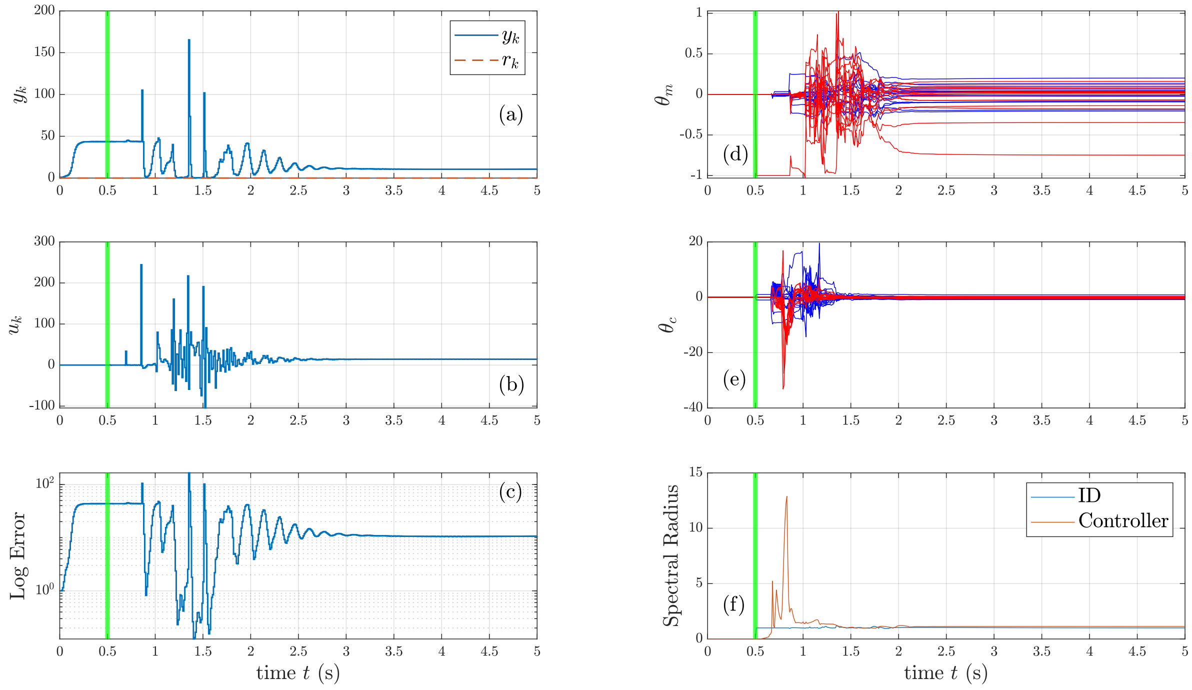

Fig. 18 Example 5: The controller is enabled at 0.5 s (indicated by the green line). Once enabled, the controller begins learning. After this learning period, (a) shows that the TKE at the sensor location does not reach the setpoint of 0 J/kg, but is reduced to 10.6770 J/kg from the uncontrolled flow value of 42.1 J/kg. (b) shows that the resulting converged control inlet velocity is 14.2643 m/s. (c) shows the log of the error between the setpoint and the TKE at the sensor location. (d) shows the time history of the estimated coefficients θ_{m} of the identified model, where numerator coefficients are plotted in blue and denominator coefficients are plotted in red. (e) shows the time history of the updated controller coefficients θ_{c}. (f) shows the spectral radius of the identified model and controller. Note that both RLS updates converge in about 2 s.

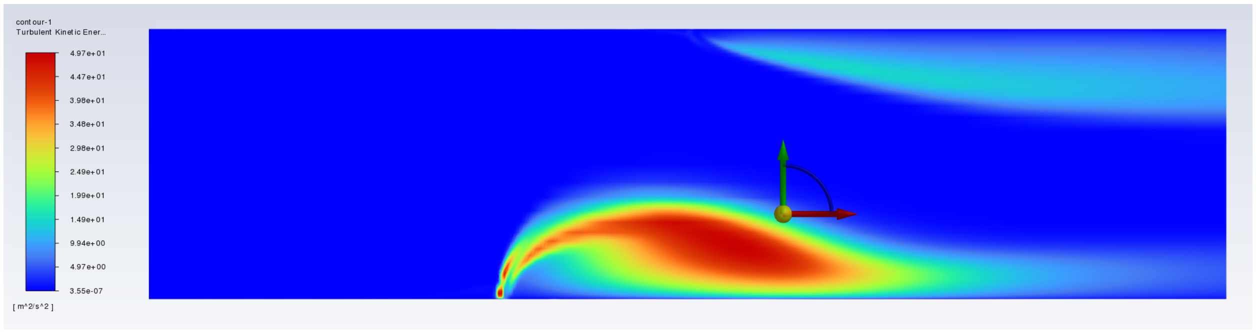

Fig. 19 Example 5: Turbulent kinetic energy contour and sensor location for the controlled flow. RCAC reduces the TKE at the sensor location by requesting the flow velocity through the control jet.

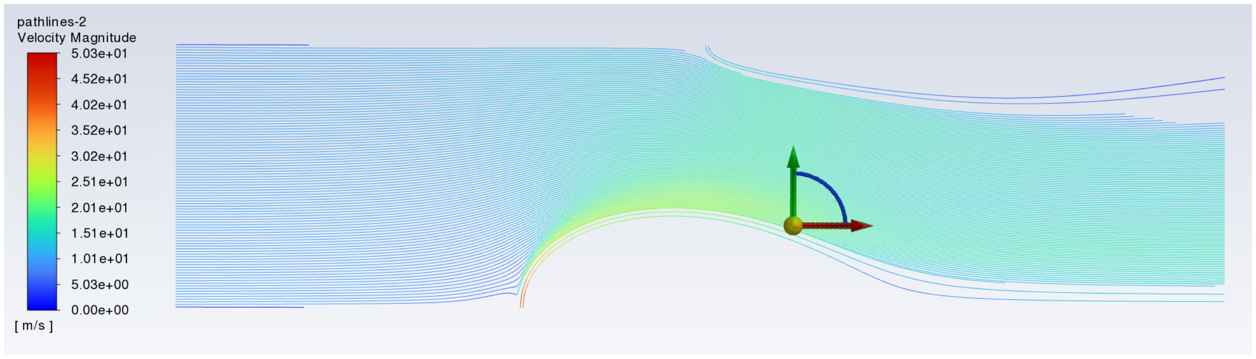

Fig. 20 Example 5: Pathlines in the controlled flow with sensor location indicated. In the controlled flow, the circulation region is moved away from the sensor location.

IX. Conclusions

This paper presented preliminary results on active flow control using data-driven retrospective cost adaptive control (DDRCAC). DDRCAC extends RCAC to include online closed-loop system identification, which facilitates the application of RCAC to flow control [20]. For all of the examples considered in this paper, DDRCAC was implemented as a "cold-start" technique, where no prior modeling information was assumed to be available. Since DDRCAC requires the user to specify various hyperparameters, future research will focus on the robustness of DDRCAC to the choice of these parameters as well as additional flow scenarios and performance objectives, including 3D flows with multiple sensors and actuators.

X. Acknowledgments

This research was supported in part by ONR under grant N00014-22-1-2457.

References

[1] Bewley, T. R., "Flow control: new challenges for a new renaissance," *Progress in Aerospace sciences*, Vol. 37, No. 1, 2001, pp. 21–58.

[2] Aamo, O. M., and Krstic, M., *Flow Control by Beedback: Stabilization and Mixing*, Springer, 2003.

[3] Gunzburger, M. D., *Perspectives in Flow Control and Optimization*, SIAM, 2002.

[4] Gad-el Hak, M., *Flow Control: Passive, Active, and Reactive Flow Management*, Cambridge University Press, 2000.

[5] Joslin, R. D., and Miller, D. N. (eds.), *Fundamentals and Applications of Modern Flow Control*, AIAA, 2009.

[6] Barbu, V., *Stabilization of Navier–Stokes Flows*, Springer, 2011.

[7] Cattafesta III, L. N., and Sheplak, M., "Actuators for active flow control," *Ann. Rev. Fluid Mech.*, Vol. 43, 2011, pp. 247–272.

[8] Luhar, M., Sharma, A. S., and McKeon, B. J., "Opposition control within the resolvent analysis framework," *J. Fluid Mech.*, Vol. 749, 2014, p. 597–626.

[9] Brunton, S. L., and Noack, B. R., "Closed-loop turbulence control: Progress and challenges," *Appl. Mech. Rev.*, Vol. 67, No. 5, 2015.

[10] Duriez, T., Brunton, S. L., and Noack, B. R., *Machine Learning Control—Taming Nonlinear Dynamics and Turbulence*, Springer, 2017.

[11] Wang, J., and Feng, L., *Flow Control Techniques and Applications*, Cambridge University Press, 2019.

[12] Modi, V. J., Munshi, S. R., Bandyopadhyay, G., and Yokomizo, T., "High-Performance Airfoil with Moving Surface Boundary-Layer Control," *J. Aircraft*, Vol. 35, No. 4, 2021, p. 544–553.

[13] King, R. (ed.), *Active Flow Control*, Springer, 2007.

[14] Bewley, T. R., and Liu, S., "Optimal and robust control and estimation of linear paths to transition," *J. Fluid Mech.*, Vol. 365, 2001, pp. 305–349.

[15] Mushtaq, T., Seiler, P., and Hemati, M. S., "Feedback control of transitional flows: A framework for controller verification using quadratic constraints," *AIAA Aviation Forum*, virtual, 2021. AIAA-2021-2825.

[16] Joshi, S. S., Speyer, J. L., and Kim, J., "A Systems Theory Approach to the Feedback Stabilization of Infinitesimal and Finite-Amplitude Disturbances in Plane Poiseuille Flow," *J. Fluid Mech.*, Vol. 332, 1997, pp. 157–184.

[17] Joshi, S. S., Speyer, J. L., and Kim, J., "Finite Dimensional Optimal Control of Poiseuille Flow," *J. Guid. Contr. Dyn.*, Vol. 22, 1999, pp. 340–348.

[18] Gautier, N., Aider, J.-L., Duriez, T., Noack, B., Segond, M., and Abel, M., "Closed-loop separation control using machine learning," *J. Fluid Mech.*, Vol. 770, No. 5, 2015, pp. 442–457.

[19] Rahman, Y., Xie, A., and Bernstein, D. S., "Retrospective Cost Adaptive Control: Pole Placement, Frequency Response, and Connections with LQG Control," *IEEE Contr. Sys. Mag.*, Vol. 37, 2017, pp. 28–69.

[20] Rizzo, M., Santillo, M. A., Padthe, A., Hoagg, J. B., Akhtar, S., Bernstein, D. S., and Powell, K. G., "CFD-Based Identification for Adaptive Flow Control Using ARMARKOV Disturbance Rejection," *Proc. Amer. Contr. Conf.*, Minneapolis, MN, 2006, pp. 3783–3788.

[21] Islam, S. A. U., Nguyen, T. W., Kolmanovsky, I. V., and Bernstein, D. S., "Data-Driven Retrospective Cost Adaptive Control for Flight Control Application," *Journal of Guidance, Control, and Dynamics*, Vol. 44, 2021, pp. 1732–1758.

[22] Goel, A., Bruce, A. L., and Bernstein, D. S., "Recursive Least Squares With Variable-Direction Forgetting: Compensating for the Loss of Persistency [Lecture Notes]," *IEEE Control Systems Magazine*, Vol. 40, No. 4, 2020, pp. 80–102.

[23] Bruce, A. L., Goel, A., and Bernstein, D. S., "Convergence and consistency of recursive least squares with variable-rate forgetting," *Automatica*, Vol. 119, 2020, p. 109052.

[24] Mohseni, N., and Bernstein, D. S., "Recursive least squares with variable-rate forgetting based on the F-test," *Proc. Amer. Contr. Conf.*, 2022, pp. 3937–3942.

[25] Lai, B., and Bernstein, D. S., "Exponential Resetting and Cyclic Resetting Recursive Least Squares," 2015, pp. 985–990.

[26] Aljanaideh, K. F., and Bernstein, D. S., "Initial Conditions in Time- and Frequency-Domain System Identification: Implications of the Shift Operator Versus the Z and Discrete Fourier Transforms," *IEEE Control Systems Magazine*, Vol. 38, No. 2, 2018, pp. 80–93. https://doi.org/https://doi.org/10.1109/MCS.2017.2786419.

[27] Middleton, R., and Goodwin, G., *Digital Control and Estimation: A Unified Approach*, Prentice Hall, 1990. https://doi.org/https://doi.org/10.1002/rnc.4590040308.

[28] Islam, S. A. U., and Bernstein, D. S., "Recursive Least Squares for Real-Time Implementation," *IEEE Control Systems Magazine*, Vol. 39, No. 3, 2019, pp. 82–85. https://doi.org/https://doi.org/10.1109/MCS.2019.2900788.

[29] Bruce, A. L., Goel, A., and Bernstein, D. S., "Convergence and Consistency of Recursive Least Squares with Variable-Rate Forgetting," *Automatica*, Vol. 119, 2020, p. 109052. https://doi.org/https://doi.org/10.1016/j.automatica.2020.109052.

[30] White, F. M., *Viscous Fluid Flow (2nd ed.)*, McGraw Hill, 1991.