Modeling, Identification and Control, Vol. 40, No. 4, 2019, pp. 199–211, ISSN 1890–1328

Model-Free PI/PID Controller Tuning of Higher Order Nonlinear Dynamic Systems

Christer Dalen [1] David Di Ruscio [2]

[1] *Skien, Norway. E-mail:* christerdalen@hotmail.com

[2] *University of South-Eastern Norway, P.O. Box 203, N-3901 Porsgrunn, Norway. E-mail:* David.Di.Ruscio@usn.no

Abstract

This paper concerns model-free PI/PID controller tuning of possible (nonlinear) higher order systems. The method can be considered as a three-step procedure. The first step is to persistently excite the system in open loop and identify the dynamic model using a subspace identification method. The second step is to approximate the model to an integrator plus time delay or double integrator plus time delay dynamic model. The third step is to compute the PI/PID controller parameters. The proposed method/theory is verified on some motivated nonlinear higher order dynamic models.

Keywords: model-free; process control; PI/PID control; subspace system identification; nonlinear

1. Introduction

This paper proposes a method which may be viewed as an extension of the previous methods of the work in Dalen and Di Ruscio (2018a,b,c).

We will focus here on Proportional Integral (PI) or PI Derivative (PID) controller tuning of stable Single Input Single Output (SISO) parts of possible (nonlinear) higher order systems. Note that, an integrator plus time delay or double integrator plus time delay transfer function model, viz.

$$H_p(s) = K\frac{e^{-\tau s}}{s^m}, \tag{1}$$

where, $m = 1$, or, $m = 2$, may serve as a sufficient model approximation for PI or PID controller tuning, i.o[1]. In Eq. (1), K, is known as the gain velocity when, $m = 1$, and the gain acceleration when, $m = 2$, $\tau \geq 0$, is the time delay, and, s, is the Laplace operator.

The PI/PID controller parameters in the proposed method evolve only from a collection of open loop[2] SISO data, hence some circles (including this paper and previous work in Dalen et al. (2015); Dalen and Di Ruscio (2016)) may describe this method as model-free or data-driven (e.g. Dong et al. (2015); Wang et al. (2018); Hou and Wang (2013); Fliess and Join (2009, 2013)).

Note that, computing PI/PID controllers directly from open loop SISO data is not new, see e.g. the step response or Process Reaction Curve (PRC) methods in Ziegler and Nichols (1942); Dalen and Di Ruscio (2018a). However, using only a single step is only persistently exciting of the order equal to one (Söderström and Stoica (1989)), and to deal with noisy systems, a system identification step should be used. There exist limited papers on such methods in the literature. To the authors' knowledge, the closest to the proposed method is the pidTuner application (MATLAB (2016)), which combines the system identification Prediction Error Methods (PEMs) (Ljung (1999)) with a frequency-domain based pidtune algorithm patented by MathWorks. Note that, the pidTuner application

[1] i.o is a Latin abbreviation for the phrase in illo ordine, meaning in that order.

[2] Note that, it would be possible to generate PI/PID controller parameters using closed loop SISO data, however this is a topic in future work.

doi:10.4173/mic.2019.4.2

is a graphical user interface tool for interactive system analysis and control design and not a MATLAB m-file function.

The contributions in this paper may be itemised as follows:

- The PI/PID controller tuning algorithms proposed in the works of Dalen and Di Ruscio (2018a,b,c) are further developed, i.e. a system identification step using the Deterministic and Stochastic and Realization (DSR) algorithm Di Ruscio (1996) is added. A complete algorithm is presented, i.e. from SISO, generally noisy process data to PI/PID controller parameters. We assume persistently excitation input data.
- The proposed PI/PID controller tuning method is compared to the MATLAB pidTuner method applied to two motion problems and one offshore problem, i.e. three stable SISO parts of higher order motivated nonlinear models.
- The proposed PI/PID controller tuning method is applied to a nonlinear Deep Submergence Rescue Vehicle (DSRV) model in the Marine Systems Simulator toolbox (Fossen and Perez (2004)). The proposed PI/PID controller tuning method is applied to a nonlinear rocket model found in the work of Mracek and Cloutier (1997).
- The proposed PI/PID controller tuning method is applied to the well-pipeline-riser example integrated in the K-Spice/LedaFlow simulator (K-Spice; LedaFlow).
- A software solution implemented in MATLAB for the proposed PI/PID controller tuning method is given.

All numerical calculations and plotting facilities are provided by using the MATLAB software (MATLAB (2016)). The rest of this paper is organised as in the following. In Section 2 the preliminary definitions are given. In Section 3 the PI/PID controller tuning algorithm is proposed. In Section 4 the motivated nonlinear process models are used to compare the proposed method with the MATLAB pidTuner. In Section 5 the concluding and discussion remarks are given. The software of the proposed PI/PID controller tuning method is given in App. A.

2. Preliminary Definitions

2.1. System Definition

Assume that the underlying system can be described by a linear, discrete-time invariant, strictly proper, combined deterministic and stochastic State Space Model (SSM), viz.,

$$\left.\begin{aligned} x_{k+1} &= A\,x_k + B\,u_k + v_k \\ y_k &= D\,x_k + w_k, \end{aligned}\right\} \text{Initial state } x_0, \quad (2)\text{–}(3)$$

where the integer, $k \geq 0$, is discrete-time, $x_k \in \mathbb{R}^n$, is the state vector and, $n \geq 1$, the system order, $u_k \in \mathbb{R}$, is the manipulative control input, $y_k \in \mathbb{R}$, is the measured output, $v_k \in \mathbb{R}^n$, is process disturbance and, $w_k \in \mathbb{R}$, is measurement noise. A, is the state transition matrix, B, is the external input matrix and, D, is the output matrix.

Note that, the deterministic part of the SSM in Eqs. (2) and (3) is expressed as,

$$\left.\begin{aligned} x^d_{k+1} &= A\,x^d_k + B\,u_k \\ y^d_k &= D\,x^d_k. \end{aligned}\right\} \text{Initial state } x^d_0, \quad (4)\text{–}(5)$$

Furthermore, assume the following:

- The pair (A, D) is observable and (A, B) is controllable.
- The system is stable, i.e., the eigenvalues of A are inside the unit circle except for the possibility of one being on the unit circle (single integrator). To not be mistaken with the approximation model in Eq. (1), where a double integrator is used when $m = 2$.
- Consider a continuous time nonlinear SSM describing the dynamical system, $\dot{x} = f(x, u)+v, y = g(x)+w$, where, $x \in \mathbb{R}^n$, is the state vector, $u \in \mathbb{R}$, is the control signal, $y \in \mathbb{R}$, is the output vector, $x(t_0 = 0)$, is the initial state, vector functions, $f(x, u) \in \mathbb{R}^n$, and, $g(x) \in \mathbb{R}$, are assumed Lipschitz continuous.
- Note that, the data may have been generated from real systems.

Consider the PID controller on ideal or parallel form,

$$H_c(s) = K_p(1 + \frac{1}{T_i s} + T_d s), \quad (6)$$

where, K_p is the proportional gain, T_i is the integral time constant and T_d is the derivative time constant.

3. From Data to PI/PID Controller Parameters

Given known output and excited input time series,

$$\left.\begin{matrix} t_k \\ y_k \\ u_k \end{matrix}\right\} \forall\ k = 0, \ldots, N-1, \quad (7)$$

which are organised as output and input vectors,

$$Y = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{N-1} \end{bmatrix} \in \mathbb{R}^N, \quad U = \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \in \mathbb{R}^N. \tag{8}$$

The time duration, t_{N-1}, and amplitude of the excitation signal, u_k, should be as small as possible, i.e not to disrupt the operation of the process too much. Note that, the ratio of the variance of the signal to noise should be as small as possible, however, a value, $< 10\%$, seems advisable. In this paper, we consider a Pseudo Random Binary Signal (PRBS). The reason for using a PRBS excitation signal is that we want to be able to identify a model with a sufficiently high system order, n. Consider the excited input vector as,

$$U = \textbf{prbs1}(N, T_{min}, T_{max}), \tag{9}$$

where, $u_k \in \{-1, 1\} \ \forall \ k = 0, \ldots, N-1$. The signal, u_k, is constant on random intervals, T, specified by the band, $T_{min} \leq T \leq T_{max}$. A MATLAB m-file function **prbs1.m** is available on request. Note that, normally, we have user-chosen scaling and offset parameters in the excitation signal in Eq. (9), however these are removed for simplicity.

We will identify the model matrix triples (A, B, D) of the deterministic SSM in Eqs. (4) and (5) by using the DSR algorithm (Di Ruscio (1996)). A MATLAB p-file function **dsr.p** is available on request.

The PI/PID controller parameters are based on the work in Dalen and Di Ruscio (2018a,b,c), and found as,

$$[K_p, T_i] = \textbf{delta\_prc\_pi\_tun}(T^1, Y^1, \Delta t, \delta, \zeta), \tag{10}$$

$$[K_p, T_i, T_d] = \textbf{delta\_prc\_pid\_tun1}(T^1, Y^1, \Delta t, \delta, \zeta), \tag{11}$$

where, T^1 and Y^1, are the PRC or open loop unit step response data obtained from the (identified) deterministic SSM (Eqs. (4) and (5)) on continuous time form (i.e. converted using zero-order hold with sampling interval, $\Delta t = t_k - t_{k-1}$). In Eqs. (10) and (11), $\zeta > 0$, is the "response speed" parameter and, $\delta > 0$, is the relative time delay error. The MATLAB m-file functions **delta_prc_pi_tun.m** and **delta_prc_pid_tun1.m** are enclosed in Apps. B and C.

The following algorithm is proposed.

Algorithm 1 *Given output and excited input data, Y and U. The following pseudocode calculates the PI ($m = 1$) or PID ($m = 2$) controller parameters. The future horizon, L, the relative time delay error, δ, "speed of response" ζ, and, J (default: $J = L$) are the tuning parameters.*

function : $[K_p, T_i, T_d] =$ **pidtun** ...
$(Y, U, \Delta t, m, L, \delta, J, \zeta, n)$
$Y := Y -$ **mean**(Y); $U := U -$ **mean**(U)
switch nargin
case *3;* $m = 1$; $L = 5$; $\delta = 3.6$; $\zeta = 1$
$[A, B, D] =$ **dsr**$(Y, U, L, 0)$
case *4;* $L = 5$; $\delta = 3.6$; $\zeta = 1$
$[A, B, D] =$ **dsr**$(Y, U, L, 0)$
case *5;* $\delta = 3.6$; $\zeta = 1$
$[A, B, D] =$ **dsr**$(Y, U, L, 0)$
case *6;* $\zeta = 1$
$[A, B, D] =$ **dsr**$(Y, U, L, 0)$
case *7;* $\zeta = 1$
$[A, B, D] =$ **dsr**$(Y, U, L, 0, J)$
case *8;*
$[A, B, D] =$ **dsr**$(Y, U, L, 0, J)$
case *9;*
$[A, B, D] =$ **dsr**$(Y, U, L, 0, J, 1, n)$
end
$dsys =$ **ss**$(A, B, D, 0, \Delta t)$; $csys =$ **d2c**$(dsys)$
$[Y^1, T^1] =$ **step**$(csys)$
if $m == 1$
$[K_p, T_i] =$ **delta_prc_pi_tun**$(T^1, Y^1, \Delta t, \delta, \zeta)$
$T_d = 0$
elseif $m == 2$
$[K_p, T_i, T_d] = \ldots$
delta_prc_pid_tun1$(T^1, Y^1, \Delta t, \delta, \zeta)$
end

A MATLAB m-file function **pidtun.m** for Algorithm 1 is enclosed in App. A. A block diagram illustrating a control system with the proposed PI/PID controller tuning method in Algorithm 1 is shown in Figure 1.

3.1. Parameter Recommendations

Suggestions on choosing parameters for Algorithm 1 are given in the following.

- Generally, we want to choose the future horizon, L, as low as possible. Ideally, a good choice could be, $L = n + 1$, if the model order, n, is a priori known and one extra state to identify a possibly unknown offset. We suggest using, $L = 5$, as the default setting.

- The relative time delay error, δ, is usually chosen in the interval, $1.1 \leq \delta \leq 3.6$ (as suggested in Di Ruscio (2010); Di Ruscio and Dalen (2017)). In this paper, we will use, $\delta = 3.6$, as default (robust) setting.

- Choose the past horizon integer, $J > 1$, and normally "large" for noisy systems. Or, use the default setting, $J = L$.

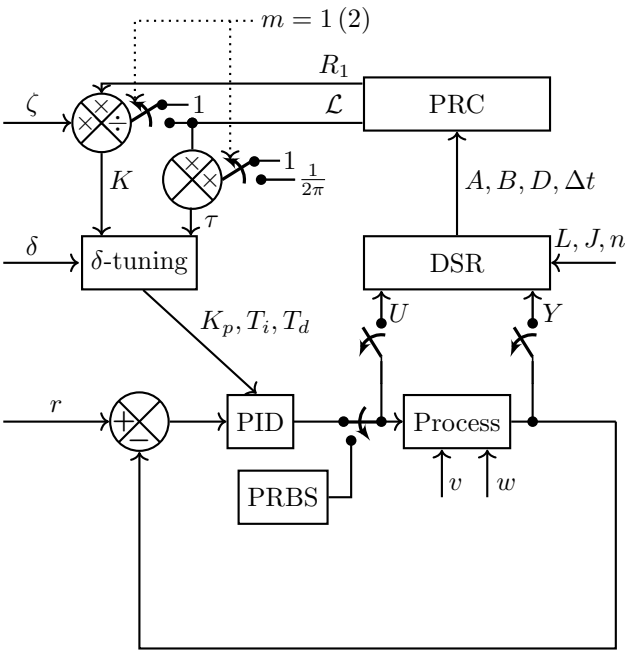

Figure 1: The figure is illustrating the proposed Algortihm 1 with a standard feedback system. The data, U and Y, are logged as long as the PRBS switch is turned on. The figure shows the integrator plus time delay ($m = 1$) or double integrator plus time delay model ($m = 2$) (Eq. (1)) approximation method based on the unit reaction rate, R_1, and lag, $\mathcal{L}$, computed from a open loop unit step response or Process Reaction Rate (PRC) (see the works Dalen and Di Ruscio (2018a,b,c) for details).

- The "speed of response", $\zeta > 0$, is normally chosen as, $\zeta = 1$ or $\zeta = 6$, (as suggested in Dalen and Di Ruscio (2018a,b)).

4. Numerical Examples

The closest to the proposed method is arguably the pidTuner application (MATLAB (2016)), which combines the system identification PEMs (Ljung (1999)) with the patented frequency-domain based pidtune algorithm. The pidTuner application has in general two tuning parameters; the prescribed gain crossover frequency, $\omega_c > 0$, and phase margin, $PM > 0$.

In the numerical examples we use the following algorithm.

Algorithm 2 *Given output and excited input data, Y and U. The following pseudocode calculates the PI ($m = 1$) or PID ($m = 2$) controller parameters. The phase margin, PM, (default: $PM = 60$) and gain crossover frequency, ω_c, are the tuning parameters.*

function : $[K_p, T_i, T_d] =$ **pidTuner1** ...
  $(Y, U, \Delta t, m, PM, \omega_c, n)$
  $dat =$ **iddata**$(Y, U, \Delta t)$
  switch nargin
    case *3;* $dsys =$ **pem**(dat); $m = 1$; $PM = 60$
  $\omega_c = []$
    case *4;* $dsys =$ **pem**(dat); $PM = 60$
  $\omega_c = []$
    case *5;* $dsys =$ **pem**(dat); $\omega_c = []$
    case *6;* $dsys =$ **pem**(dat)
    case *7;* $dsys =$ **pem**(dat, n)
  $csys =$ **d2c**$(dsys)$
  $opt = pidtuneOptions('PhaseMargin', PM)$
  if $m == 1$
    $c =$ **pidtune**$(csys,$ **pidstd**$(1,1), \omega_c, opt)$
    $Td = 0$
  elseif $m == 2$
    $c =$ **pidtune**$(csys,$ **pidstd**$(1,1,1), \omega_c, opt)$
    $T_d = c.Td$
  end
  $K_p = c.Kp$; $T_i = c.Ti$

A MATLAB m-file function **pidTuner1.m** for Algorithm 2 is enclosed in App. D.

We adopt the performance indices from Åström and Hägglund (1995); Seborg et al. (1989); Skogestad (2003). We define these in the following. In order to measure performance in a feedback system, the Integrated Absolute Error (IAE) is defined as,

$$IAE = \int_0^\infty |\, e(t) \,|\, dt, \tag{12}$$

where r is the reference and, $e = r - y$, is the error.

Furthermore, the following is defined:

- IAE_v, evaluates the performance in the case of input disturbance step, v, with constant reference, r.
- IAE_r, evaluates the performance in the case of a reference step, r, with constant disturbance, v.

To evaluate the amount of input usage we include the following Total input Value (TV) measure,

$$TV = \int_0^\infty |\Delta u_k| \, dt, \tag{13}$$

where, $\Delta u_k = u_k - u_{k-1}$, is the control rate of change.

- TV_v, evaluates the input usage in the case of input disturbance step, v, with constant reference, r.
- TV_r, evaluates the input usage in the case of a reference step, r, with constant disturbance, v.

Example 4.1 (Deep Submergence Rescue Vehicle) *Consider the closed loop system illustrated in Figure 2 where the system is a DSRV, i.e. the continuous time fifth order nonlinear SSM,* $\dot{x} =$ **DSRV**(x,u), *with measurement Eq.,* $y = x_5$, *where* **DSRV.m** *is a m-file function found in the Marine Systems Simulator toolbox (Fossen and Perez (2004)). We will consider the following SISO control case:*

$$y_k \in \mathbb{R} := \Big\{ \textit{Pitch angle, [deg]}$$
$$u_k \in \mathbb{R} := \Big\{ \textit{Stern angle, [deg]}$$

The PID controller ($m = 2$) parameters, given in Table 1 (rows 2:4, columns 2:3), are found using **pidtun** *(Algorithm 1) and* **pidTuner1** *(Algorithm 2), where both algorithms are used with default parameters. Note that, the output and excited input data, Y and U, are shown in Figure 3 (Time $400:900$).*

It is seen in Table 1 (rows 5:8, columns 2:3) that the proposed **pidtun** *method has an edge over MATLAB* **pidTuner1** *in terms of the input disturbance response, viz.* **pidtun** *is seen to be,* $\frac{IAE_v^{\textbf{pidTuner1}}}{IAE_v^{\textbf{pidtun}}} = 2.5$, *times better than* **pidTuner1** *in performance and,* $\frac{TV_v^{\textbf{pidTuner1}}}{TV_v^{\textbf{pidtun}}} = 1.2$, *times better in terms of input usage. In terms of the reference response, the* **pidTuner1** *is seen,* $\frac{IAE_r^{\textbf{pidtun}}}{IAE_r^{\textbf{pidTuner1}}} = 1.2$, *times better than* **pidtun** *in performance, however* **pidtun** *is seen,* $\frac{TV_r^{\textbf{pidTuner1}}}{TV_r^{\textbf{pidtun}}} = 1.8$, *times better than* **pidTuner1** *in terms of input usage.*

Example 4.2 (Rocket) *Consider a rocket, i.e. the continuous time sixth order nonlinear SSM,* $\dot{x} =$ **rocket**(x,u), *with measurement*

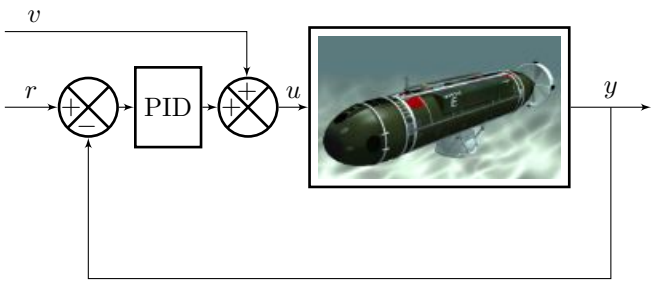

Figure 2: Example 4.1. The figure shows the closed loop system where the process is a nonlinear deep submergence rescue vehicle. Output, $y :=$ Pitch angle [deg]. Input, $u :=$ Stern angle [deg].

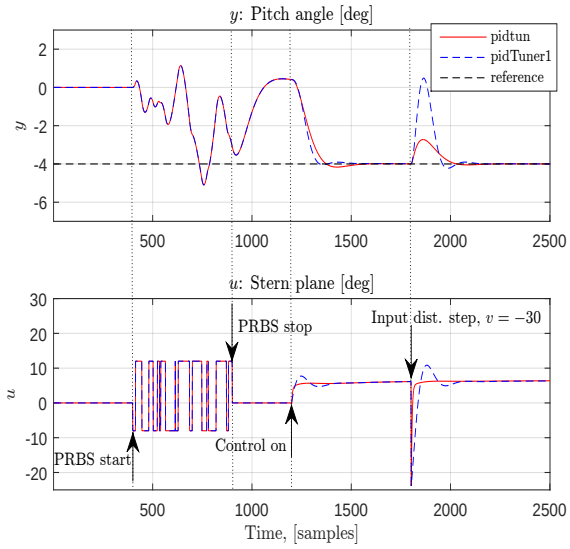

Figure 3: Example 4.1. The figure shows the open loop output and excited input data used for computing the PID controller ($m = 2$) parameters (see Table 1) in **pidtun** (Algorithm 1) and **pidTuner1** (Algorithm 2). It shows the reference, $r = -4$, and input disturbance step, $v = -30$, time domain responses of the closed loop system in Figure 2, comparing **pidtun** vs. **pidTuner1**. The corresponding performance and input usage indices are found in Table 1.

Table 1: Example 4.1. The table shows the PID controller ($m = 2$) parameters, K_p, T_i, and, T_d. It shows performance and input usage indices, IAE_r, IAE_v, TV_r, and, TV_v, corresponding to the reference and input disturbance step time domain responses shown in Figure 3.

	pidtun	**pidTuner1**
K_p	-15.0911	-4.6345
T_i	0.7094	0.5648
T_d	0.3346	0.1372
IAE_r	0.0650	0.0560
IAE_v	0.0267	0.0679
TV_r	0.1147	0.2109
TV_v	1.0524	1.2607

Eq., $y = x_2$, where **rocket.m** *is a MATLAB m-file function found in App. E. The m-file implementation is based on the nonlinear SSM in Eqs. (40) and (44) in Mracek and Cloutier (1997). The closed loop system with the rocket is illustrated in Figure 4.*

We will consider the following SISO control case:

$$y_k \in \mathbb{R} := \{ \textit{Angle of attack, [deg]}$$
$$u_k \in \mathbb{R} := \{ \textit{Rudder angle, [deg]}$$

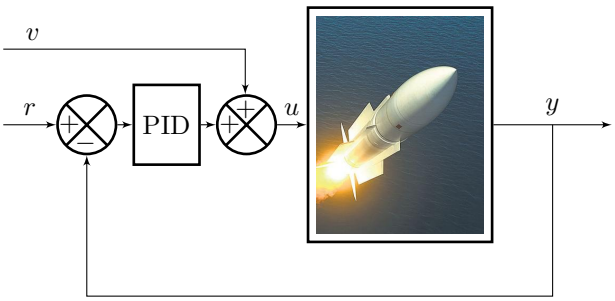

Figure 4: Example 4.2. The figure shows the closed loop system where the process is a nonlinear rocket. Output, $y :=$ Pitch angle [deg]. Input, $u :=$ Stern angle [deg].

The PID controller ($m = 2$) parameters, given in Table 2 (rows 2:4, columns 2:3), are found using **pidtun** *(Algorithm 1) and* **pidTuner1** *(Algorithm 2), where both algorithms are used with default parameters. Note that, the output and excited input data, Y and U, are shown in Figure 5 (Time $200:600$).*

It is seen in Table 2 (rows 5:8, columns 2:3) that the proposed **pidtun** *method has an edge over MATLAB* **pidTuner1** *in terms of the input disturbance response, viz.* **pidtun** *is seen to be, $\frac{IAE_v^{\text{pidTuner1}}}{IAE_v^{\text{pidtun}}} = 2.1$, times better than* **pidTuner1** *in terms of performance and, $\frac{TV_v^{\text{pidTuner1}}}{TV_v^{\text{pidtun}}} = 1.1$, times better in terms of input usage. In terms of the reference response, the methods are seen similar, i.e.,* **pidtun** *is seen, $\frac{IAE_r^{\text{pidTuner1}}}{IAE_r^{\text{pidtun}}} = 1.1$, times better in terms of performance, however* **pidTuner1** *is seen, $\frac{TV_r^{\text{pidtun}}}{TV_r^{\text{pidTuner1}}} = 1.0$, equal to* **pidtun** *in terms of input usage.*

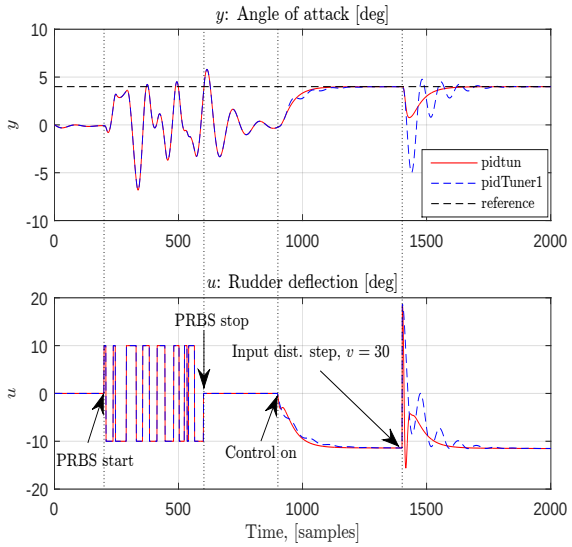

Figure 5: Example 4.2. The figure shows the open loop output and excited input data used for computing the PID controller ($m = 2$) parameters (see Table 2) in **pidtun** (Algorithm 1) and **pidTuner1** (Algorithm 2). It shows the reference, $r = 4$, and input disturbance step, $v = 30$, time domain responses of the closed loop system illustrated in Figure 4, comparing **pidtun** vs. **pidTuner1**. The corresponding performance and input usage indices are found in Table 2.

Example 4.3 (Offshore: Topside choking) *Consider in the following an anti-slug control case on a well-pipeline-riser example integrated in the K-Spice/LedaFlow simulator (K-Spice; LedaFlow). Initially meant for the operators to predict incoming slugging regime, the bottom-hole riser pressure is suitable for measurement output. The (general) goal is to maximise or minimise the outlet flow or bottom-hole pressure, i.o. The work in Schmidt et al. (1979) documents the first successful application of an automatic control system on a pipeline-riser process with a topside choke*

Table 2: Example 4.2. The table shows the PID controller ($m = 2$) parameters, K_p, T_i, and, T_d. It shows the performance, and input usage indices, IAE_r, IAE_v, TV_r, and, TV_v, corresponding to the reference and input disturbance step time domain responses shown in Figure 5.

	pidtun	**pidTuner1**
K_p	-5.1808	-1.8596
T_i	0.4220	0.2923
T_d	0.1991	0.0679
IAE_r	0.0453	0.0513
IAE_v	0.0428	0.0905
TV_r	0.2040	0.1965
TV_v	1.4470	1.5740

as the manipulative control input. Note that, if the PI controller ($m = 1$) tuner method works on this simulator, it is likely to work on the real process [3].

Hence, we consider the following SISO control case:

$$y_k \in \mathbb{R} := \{ \textit{Bottom hole riser pressure, [bara]}$$
$$u_k \in \mathbb{R} := \{ \textit{Topside choke valve, [\%]}$$

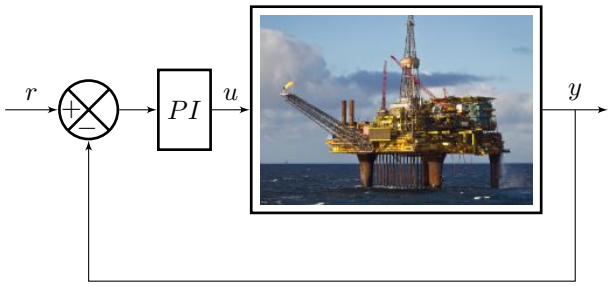

Figure 6: Example 4.3. The figure illustrates the closed loop system with an anti-slugging case example implemented in the K-Spice/Ledaflow simulator (K-Spice; LedaFlow). Output, $y :=$ Bottom hole riser pressure, [bara]. Input, $u :=$ Topside choke valve, [%].

600*, input and output samples were logged from an open loop PRBS input experiment in the K-Spice/LedaFlow simulator where the first* 400 *samples was used for identification and the last* 200 *samples for validation (see Figure 7).*

Consider the mean squared error, $V = \frac{1}{N}\sum_{k=1}^{N}(y_k - y_k^d)^2$*, where* y_k^d *is the output of the (identified) deterministic SSM in Eqs. (4) and (5) and actual output,* y_k*. The model identified from using the DSR algorithm (i.e. the SSM as in Eqs. (4) and (5) with matrices as in Eq. (14)) showed a slight edge over PEM, i.e. the DSR model was seen in Table 3 (rows 2:3, columns 2:3) to be,* $\frac{V_{id}^{\mathbf{pem}}}{V_{id}^{\mathbf{dsr}}} = 1.12$*, and,* $\frac{V_{val}^{\mathbf{pem}}}{V_{val}^{\mathbf{dsr}}} = 1.03$*, times better then PEM on the identification and validation set, i.o. See Figures 8 and 9 for the corresponding simulated outputs,* y_k^d*, from the the (identified) deterministic SSMs (Eqs. (4) and (5)), vs. the identification and validation set, i.o. Note that, the results shown in Table 3 (similar results are also seen in Dalen and Di Ruscio (2016)) may be explained by a trade-off between bias and variance (Di Ruscio (2009)).*

The PI controller ($m = 1$) parameters, given in Table 4 (rows 2:3, columns 2:3), are computed using **pidtun** *(Algorithm 1) and* **pidTuner1** *(Algorithm 2), where both methods are used with default parameters, except that the "speed of response",* $\zeta = 2$*, and, gain crossover frequency,* $\omega_c = 0.01$*, are both used to prescribe the sensitivity function peak,* $M_s = \max_{0\leq\omega<\infty} |\frac{1}{1+H_p(j\omega)H_c(j\omega)}| = 1.3$,[4] *where,* $H_p(s)$*, is the transfer function corresponding to the (identified) deterministic SSM in Eqs. (4) and (5).*

The two PI controllers ($m = 1$) performed approximately equal when applied to the well-pipeline-riser example in the K-Spice/LedaFlow simulator. See Figure 10 for the time series of the reference ramping experiment and Table 4 (rows 2:3, columns 4:5) for the corresponding performance and input usage indices.

$$\overbrace{A = \begin{bmatrix} 0.9563 & 1.1959 \\ -0.0030 & 0.9898 \end{bmatrix}}^{\textbf{DSR}}, \quad B = \begin{bmatrix} 0.2688 \\ 0.0064 \end{bmatrix}, \quad D = \begin{bmatrix} -0.6016 & 0.6941 \end{bmatrix}. \tag{14}$$

Table 3: Example 4.3. The table shows the mean square error, $V = \frac{1}{N}\sum_{k=1}^{N}(y_k - y_k^d)^2$, where y_k^d is the output of the (identified) deterministic SSM in Eqs. (4) and (5) and actual output, y_k, corresponding to the identification and validation data shown in Figures 8 and 9, i.o.

V_{alg}	Identification set	Validation set
V_{DSR}	0.2378	0.1455
V_{PEM}	0.2672	0.1498

[3] Personal communication with Kongsberg Digital, The closest you can get to the real plant behavior.

[4] For robust controllers we consider the interval, $1.3 \leq M_s \leq 2.0$ (p. 125 in Åström and Hägglund (1995)).

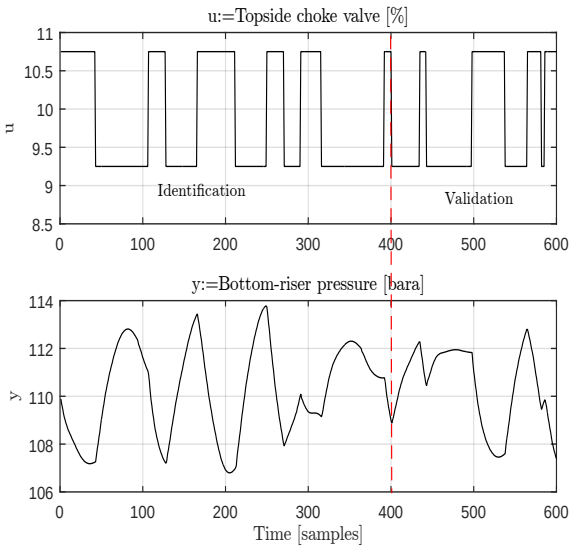

Figure 7: Example 4.3. The figure shows the open loop PRBS input experiment of the well-pipeline-riser example in the K-Spice/LedaFlow simulator.

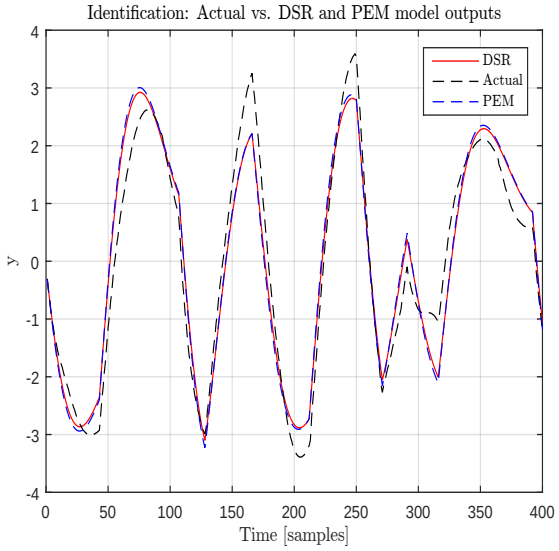

Figure 8: Example 4.3. The figure shows the comparison of the simulated output, y_k^d, from the DSR and PEM model, with the actual output, y_k, from the identification set. The corresponding mean square errors are given in Table 3.

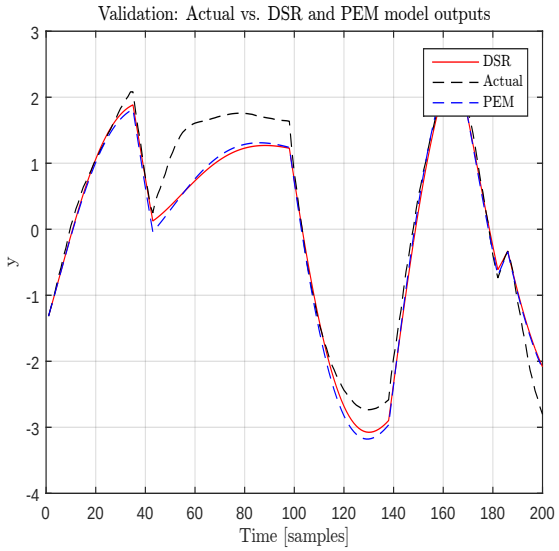

Figure 9: Example 4.3. The figure shows the comparison of the simulated output, y_k^d, from the DSR and PEM model, with the actual output, y_k, from the validation set. The corresponding mean square errors are given in Table 3.

Table 4: Example 4.3. The table shows the performance index, $IAE_r = \int_0^\infty |\, e(t)\,|\, dt$, and the input usage index, $TV_r = \int_0^\infty |\, \Delta u_k|dt$, for prescribed sensitivity function peak, $M_s = \max_{0\leq\omega<\infty} |\frac{1}{1+H_p(j\omega)H_c(j\omega)}| = 1.3$. The corresponding reference stepping time domain responses are found in Figure 10.

Algorithm	K_p	T_i	IAE_r	TV_r	M_s
pidtun	1.55	183.5	57207	6794	1.3
pidTuner1	1.89	163.0	57089	6800	1.3

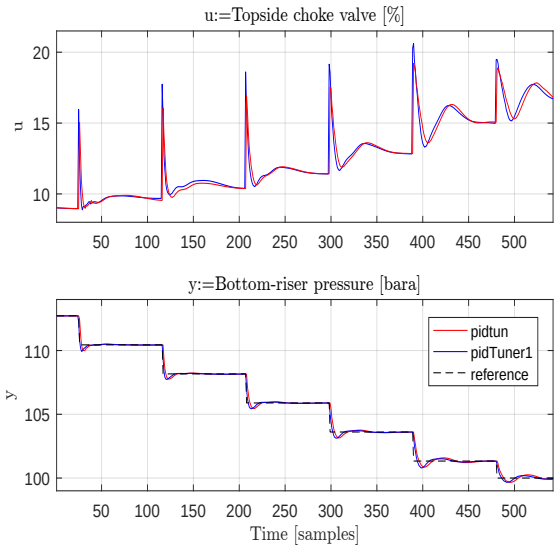

Figure 10: Example 4.3. The figure shows the reference ramping experiment of the closed loop system illustrated in Figure 6, i.e. comparing the PI controllers ($m = 1$) from the proposed **pidtun** (Algorithm 1) vs. **pidTuner1** (Algorithm 2). The corresponding performance and input usage indices are found in Table 4.

5. Concluding and Discussion Remarks

The discussion and concluding remarks are itemised as follows:

- A model-free PI/PID controller tuning method **pidtun** (Algorithm 1) algorithm was presented in Section 3, i.e. a method which computes PI/PID controller parameters only based on open loop output and excited input data.

- The proposed **pidtun** (Algorithm 1), and the MATLAB **pidTuner1** (Algorithm 2) method are successfully applied and compared on two motion problems and one offshore problem in Section 4, i.e. three stable SISO parts of higher order motivated nonlinear SSMs.

- Consider the motion problems in the numerical examples in Section 4 in the following. The PID controllers ($m = 2$) computed from the proposed **pidtun** is seen to have an edge over MATLAB **pidTuner1** in terms of the input disturbance response, viz. **pidtun** is seen to be atleast, $\frac{IAE_v^{\mathbf{pidTuner1}}}{IAE_v^{\mathbf{pidtun}}} = 2.1$, times better than **pidTuner1** in performance and, $\frac{TV_v^{\mathbf{pidtun}}}{TV_v^{\mathbf{pidTuner1}}} = 1.1$, times better in terms of input usage. However, in terms of the reference response, both algorithms are seen similar in performance, however **pidtun** is seen, $\frac{TV_r^{\mathbf{pidTuner1}}}{TV_r^{\mathbf{pidtun}}} = 1.8$, times better than **pidTuner1** in terms of input usage for Example 4.1.

- Consider the results in the offshore problem in Example 4.3 in the following. The PI controllers ($m = 1$) from **pidtun** and **pidTuner1** were seen to perform approximately equal. This is because both PI controllers ($m = 1$) were designed on prescribing the maximum sensitivity peak, $M_s = 1.3$.

- It was demonstrated in the numerical examples in Section 4 that the default settings for **pidtun**, proposed in Subsection 3.1, were found satisfying for all the 3 examples, with exception of Example 4.3 where the "speed of response", $\zeta = 2$ was prescribing the maximum sensitivity peak, $M_s = 1.3$.

- Note that, given PID control ($m = 2$), it is in practice suggested to add a filter on the derivative term with a first order system with time constant, $\frac{T_d}{N_f}$, where typical values for, N_f, are 8 to 20 (see p. 77 in Åström and Hägglund (1995)).

- Note that, the pidtune algorithm is patented by MathWorks.

A. pidtun method m-file implementation

A MATLAB m-file function for the pseudocode in Algorithm 1 is given.

```
    function...
[Kp,Ti,Td] =pidtun(Y,U,dt,m,L,delta,J,zeta,n)
% PURPOSE. Tuning an ideal PI/PID controller
% hc(s)=Kp(1+1/(Ti*s)+Td*s)
% based on input step response data.
%[Kp,Ti,Td] =pidtun(Y,U,dt,m,L,delta,J,zeta,n)
% T, Y  - Step response data
%         T time vector
%         Y output vector
%
% delta - Tuning parameter,
%         relative time delay error
%         delta=2.3 (default)
%
% Kp    - Proportional gain
% Ti    - Integral time constant
% Td    - Derivative time constant

Y=Y-mean(Y);U=U-mean(U);
switch nargin
    case 3;m=1;L=5;delta=3.6;zeta=1;
        [A,B,D]=dsr(Y,U,L,0);
    case 4;L=5;delta=3.6;zeta=1;
        [A,B,D]=dsr(Y,U,L,0);
    case 5;delta=3.6;zeta=1;
        [A,B,D]=dsr(Y,U,L,0);
    case 6;zeta=1;
        [A,B,D]=dsr(Y,U,L,0);
    case 7;zeta=1;
        [A,B,D]=dsr(Y,U,L,0,J);
    case 8;
        [A,B,D]=dsr(Y,U,L,0,J);
    case 9
        [A,B,D]=dsr(Y,U,L,0,J,1,n);
end

dsys=ss(A,B,D,0,dt); csys=d2c(dsys);
[Y1,T1]=step(csys);
if m==1
[Kp,Ti]=delta_prc_pi_tun(T1,Y1,dt,delta,zeta);
Td=0;
elseif m==2
[Kp,Ti,Td]...
=delta_prc_pid_tun1(T1,Y1,dt,delta,zeta);
end
```

The DSR algorithm (Di Ruscio (1996)) is implemented in MATLAB, and available as a p-file function **dsr.p** in the D-SR Toolbox. The toolbox is available on request.

The MATLAB m-file functions **delta_prc_pi_tun.m** and **delta_prc_pid_tun1.m** are enclosed in Apps. B and C.

B. δ-PRC method MATLAB m-file

```
function [Kp,Ti]=delta_prc_pi_tun...
(T,Y,dt,delta,z,cb,du)
% PURPOSE. Tuning an ideal PI controller
% hc(s)=Kp(1+1/(Ti*s))
% based on input step response data.
% [Kp,Ti]=delta_prc_pi_tun...
% (T,Y,delta,z,cb,du)
%
% T, Y  - Step response data
%         T time vector
%         Y output vector
%
% delta - Tuning parameter,
%         relative time delay error
%         delta=2.12 (default)
%
% du    - Input step change
%         du=1 (default)
%
% Kp    - Proportional gain
% Ti    - Integral time constant

if nargin == 4; z=1; cb=2.5; du=1; end
if nargin == 5; cb=2.5; du=1; end
if nargin == 6; du=1; end

h=T(2)-T(1); A=diff(Y)/h;

if Y(end)<0
 [R,i]=min(A);
else
 [R,i]=max(A);
end

R1=R/du;
t1=T(i);

y1=Y(i); L=t1-y1/R1;

k=z*R1; tau=L+dt/2;

[alfa,beta]=pi_tun_maxdelay(cb,delta);
Kp=alfa/(k*tau);
Ti=beta*tau;
% end delta_prc_pi_tun.m
```

Note that, the above MATLAB m-file function was

not given in the paper Dalen and Di Ruscio (2018c).

C. Modified δ-PRC method MATLAB m-file

```
function [Kp,Ti,Td]=delta_prc_pid_tun1....
(T,Y,dt,delta,zeta)
% PURPOSE. Tuning an ideal PID controller
% hc(s)=Kp(1+1/(Ti*s)+Td*s)
% based on input step response data.
% [Kp,Ti,Td]=delta_prc_pid_tun1...
% (T,Y,delta,zeta)
%
% T, Y  - Step response data
%         T time vector
%         Y output vector
%
% delta - Tuning parameter,
%         relative time delay error
%         delta=2.12 (default)
%
% zeta - "speed of response"
%
% Kp   - Proportional constant
% Ti   - Integral time constant
% Td   - Derivative time constant

if nargin == 4; zeta=1; end
if nargin == 5; delta=2.12; zeta=1;  end

h=T(2)-T(1); calA=diff(Y)/h;

if Y(end)<0
    [R1,i]=min(calA);
else
    [R1,i]=max(calA);
end

t1=T(i);

y1=Y(i); L=t1-y1/R1;

tau=L/(2*pi)+dt/2;
K=zeta*R1/L;

cb=2.12; gamm=2.12;
[Kp,Td]=pd_tun_maxdelay(K,tau,delta,cb,1);
Ti=gamm*Td;

% end delta_prc_pid_tun1.m
```

Note that, this MATLAB m-file function is a slightly modified version of the m-file in App. A in Dalen and Di Ruscio (2018b).

D. pidTuner1 method m-file implementation

A m-file function for the pseudocode in Algorithm 2 is given.

```
    function...
[Kp,Ti,Td]=pidTuner1(Y,U,dt,m,pm,wc,n)
dat=iddata(Y,U,dt);
switch nargin
case 3;dsys=pem(dat);m=1;pm=60;wc=[];
    case 4;dsys=pem(dat);pm=60;wc=[];
    case 5;dsys=pem(dat);wc=[];
    case 6;dsys=pem(dat);
    case 7;dsys=pem(dat,n);
end
opt = pidtuneOptions('PhaseMargin',pm);
csys=d2c(dsys);
if m==1
c=pidtune(csys,pidstd(1,1),wc,opt);Td=0;
elseif m==2
c=pidtune(csys,pidstd(1,1,1),wc,opt);Td=c.Td;
end
Kp=c.Kp;Ti=c.Ti;
```

E. Rocket function m-file implementation

```
function f=rocket(t,x,u)
f=zeros(6,1);

f(1)=0.4008*x(1)^2*x(2)^3*sin(x(2))...
-0.6419*x(1)^2*abs(x(2))*x(2)*sin(x(2))...
-0.2010*x(1)^2*(2-x(1)/3)*x(2)*sin(x(2))...
-0.0062*x(1)^2-0.0403*x(1)^2*sin(x(2))*x(5)...
-0.0311*sin(x(3));
f(2)=0.4008*x(1)*x(2)^3*cos(x(2))...
-0.6419*x(1)*abs(x(2))*x(2)*cos(x(2))...
-0.2010*x(1)*(2-x(1)/3)*x(2)*cos(x(2))...
-0.0403*x(1)*cos(x(2))*x(5)...
-0.0311*cos(x(3))/x(1)+x(4);
f(3)=0.4008*x(1)*x(2)^3*cos(x(2))...
-0.6419*x(1)*abs(x(2))*x(2)*cos(x(2))...
-0.2010*x(1)*(2-x(1)/3)*x(2)*cos(x(2))...
-0.0403*x(1)*cos(x(2))*x(5)...
-0.0311*cos(x(3))/x(1);
f(4)=49.82*x(1)^2*x(2)^3 ...
-78.86*x(1)^2*abs(x(2))*x(2)...
    +3.6*x(1)^2*(-7-8*x(1)/3)*x(2)...
    -14.54*x(1)^2*x(5)-2.12*x(1)^2*x(4);
f(5)=x(6);
```

```
xi=0.7; wa=50;
f(6)=-wa^2*x(5)-2*xi*wa*x(6)+wa^2*u;
end
```

The m-file implementation is based on the nonlinear SSM in Eqs. (40) and (44) in Mracek and Cloutier (1997).

References

Åström, K. and Hägglund, T. *PID Controllers: Theory, Design, and Tuning*. Instrument Society of America, 1995.

Dalen, C. and Di Ruscio, D. Model-Free Predictive Anti-Slug Control of a Well-Pipeline-Riser. *Modeling, Identification and Control*, 2016. 37(1):41–52. doi:10.4173/mic.2016.1.4.

Dalen, C. and Di Ruscio, D. A Novel Process-Reaction Curve Method for Tuning PID Controllers. *Modeling, Identification and Control*, 2018a. 39(4):273–291. doi:10.4173/mic.2018.4.4.

Dalen, C. and Di Ruscio, D. A Semi-Heuristic Process-Reaction Curve PID Controller Tuning Method. *Modeling, Identification and Control*, 2018b. 39(1):37–43. doi:10.4173/mic.2018.1.4.

Dalen, C. and Di Ruscio, D. PI Controller Tuning Based on Integrating Plus Time Delay Models: Performance Optimal tuning. *Algorithms*, 2018c. 11(4).

Dalen, C., Di Ruscio, D., and Nilsen, R. Model-free optimal anti-slug control of a well-pipeline-riser in the K-Spice/LedaFlow simulator. *Modeling, Identification and Control*, 2015. 36(3):179–188. doi:10.4173/mic.2015.3.5.

Di Ruscio, D. Combined Deterministic and Stochastic System Identification and Realization: DSR - A Subspace Approach Based on Observations. *Modeling, Identification and Control*, 1996. 17(3):193–230. doi:10.4173/mic.1996.3.3.

Di Ruscio, D. A Bootstrap Subspace Identification Method: Comparing Methods for Closed Loop Subspace Identification by Monte Carlo Simulations. *Modeling, Identification and Control*, 2009. 30(4):203–222. doi:10.4173/mic.2009.4.2.

Di Ruscio, D. On Tuning PI Controllers for Integrating Plus Time Delay Systems. *Modeling, Identification and Control*, 2010. 31(4):145–164. doi:10.4173/mic.2010.4.3.

Di Ruscio, D. and Dalen, C. Tuning PD and PID Controllers for Double Integrating Plus Time Delay Systems. *Modeling, Identification and Control*, 2017. 38(2):95–110. doi:10.4173/mic.2017.2.4.

Dong, H.-R., Jin, S., and Hou, Z.-S. Model free adaptive control for automatic car parking systems. 2015. 2015:1769–1774.

Fliess, M. and Join, C. Model-free control and intelligent PID controllers: towards a possible trivialization of nonlinear control? In *15th IFAC Symposium on System Identification (SYSID 2009)*. IFAC, Saint-Malo, France, 2009.

Fliess, M. and Join, C. Model-free control. *International Journal of Control*, 2013. 86(12):2228–2252. doi:10.1080/00207179.2013.810345.

Fossen, T. I. and Perez, T. Marine Systems Simulator (MSS). 2004. URL https://github.com/cybergalactic/MSS.

Hou, Z.-S. and Wang, Z. From model-based control to data-driven control: Survey, classification and perspective. *Information Sciences*, 2013. 235:3–35. doi:10.1016/j.ins.2012.07.014.

K-Spice. K-Spice version 2.11. 2015. URL kongsberg.com/k-spice.

LedaFlow. LedaFlow version 1.7. 2015. URL kongsberg.com/ledaflow.

Ljung, L. *System Identification (2nd ed.): Theory for the User*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

MATLAB. *Version 9.1.0.441655 (R2016b)*. The MathWorks Inc., Natick, Massachusetts, USA, 2016. Control System Toolbox, Version 9.3. Optimization Toolbox, Version 6.2.

Mracek, C. and Cloutier, J. Missile longitudinal autopilot design using the state-dependent riccati equation method. *Guidance, Navigation, and Control Conference*, 1997. pages 1–6.

Schmidt, Z., Brill, J. P., and Beggs, H. D. Choking can eliminate severe pipeline slugging. *Oil & Gas J.*, 1979. (12):230–238.

Seborg, D., Edgar, T., and Mellichamp, D. *Process Dynamics and Control*. Number v. 1 in Chemical Engineering Series. Wiley, 1989.

Skogestad, S. Simple analytic rules for model reduction and PID controller tuning. *Journal of Process Control*, 2003. 13(4):291–309. doi:10.1016/S0959-1524(02)00062-8.

Söderström, T. and Stoica, P. *System Identification*. Prentice-Hall international series in systems and control engineering. Prentice-Hall, 1989.

Wang, J., Zhang, Y., Jin, X., and Su, H. A Recursive Tuning Approach for the Model-Free PIDController Design. *IFAC-PapersOnLine*, 2018. 51(4):143–147. doi:10.1016/j.ifacol.2018.06.116. 3rd IFAC Conference on Advances in Proportional-Integral-Derivative Control PID 2018.

Ziegler, J. G. and Nichols, N. B. Optimum Settings for Automatic Controllers. *Trans. American Society of Mechanical Engineers*, 1942. 64:759–768.