

Implementation of selected control theory algorithms for embedded real-time systems

Martin Dodek*, Eva Miklovičová*, Marián Tárnik*

* Faculty of Electrical Engineering and Information Technology Slovak University of Technology in Bratislava
Email: martin.dodek@stuba.sk, eva.miklovicova@stuba.sk, marian.tarnik@stuba.sk

Abstract—In this paper a detailed insight into implementation of selected control theory algorithms is shown. These algorithms have been successively implemented using an object oriented programming language while considering a computation cost optimization and preserving the overall code readability. The resulting product, a form of a numerical computing library, can be afterwards found suitable for various control oriented engineering applications, especially those having limited hardware resources (e.g. embedded systems). Finally, the whole implemented framework was deployed in a *STM32* micro-controller based and real-time application designed for temperature control of a physical system.

Index Terms—numerical computing, real-time control, system identification, control design, implementation, embedded systems, object-oriented programming

I. INTRODUCTION

The field of control theory was formed by a significant research in the last century while resulting into huge and continuous progress in all domains of engineering and technology. Despite this fact, practical applications of many algorithms were infeasible for the insufficient computers performance. Nowadays, the embedded systems, primarily based on single chip programmable micro-controllers, are an ideal platform for a real-time operable control application deployment.

The aim of this paper is to create an open-source C++ numerical computing library focused on implementing chosen algorithms from the control system theory. Our product is meant to be available for wide research and development community, so anybody is free to share and use it in whatever project aimed on a control theory. Implemented functionality is intended to be verified being deployed in the real-system control application. The *STM32* micro-controllers family in combination with *FreeRTOS* operating system has been chosen as the target platform for this purpose.

Few open-source numerical computation libraries focused on the domain of control engineering have been already implemented as reported in the literature.

The most interesting one, matching our scope closely, is *The Control Toolbox* developed by *ETH Agile & Dexterous Robotics Lab*. This C++ library contains tools to design and evaluate controllers, model dynamic systems and solve optimal control problems yet is primarily aimed on robotics. [1]

Other numerical computing libraries are also available, such as popular *GNU Scientific Library*. The *GSL* is well maintained C language library suitable for general purpose engineering and scientific problems namely: solving systems of linear

equations, eigenvalues problems or even least squares fitting. This library also contains implementation of ODE solvers, yet it lacks any other content related to the control theory. [2]

The *SLICOT - A Subroutine Library in Systems and Control* is indeed a complex library that consists of many computational routines in various sub-domains of the control engineering. However it is written in the outdated Fortran language and the library itself is not even maintained any more. [3]

The last but not least is the proprietary software like *Matlab* and its code-generator. Using *Matlab* might be suitable for rapid algorithms development, however due to its closed source character, the final deployment in a project is kind a problematic. Auto-generated code also usually lacks of clean-written application interface, therefore it might be cumbersome to adopt and embed this code into a complex software project. However, the most significant disadvantage of the automatic code-generation is poor readability, what may result into confusion during the debugging process. Optimization of computation complexity in the case of auto-generated code is also questionable.

II. DISCRETE-TIME SYSTEMS

The cybernetics established the concept of *systems and signals* in order to describe real world processes from the engineer's point of view. This idea is actually quite suitable for a software implementation using modern programming languages, especially those object-oriented. [4]

Considering this, the abstraction of a system can be represented as an object in the source code. This *system-object* analogy may significantly aid the implementation process and improve overall code readability.

Discrete-time systems are an essentials of numeric control since computers operate at finite frequency and the source code is executed sequentially. For this type of systems, time is being assumed discrete to certain sampling period T_s .

Whereas continuous system dynamics is usually defined by differential equations, for discrete-time systems difference equations have been established. The so called "update function" implements the system specific difference equation and hence explicitly defines its dynamic properties. Concerning implementation, the generic discrete-time system class comprises the update function in a form of *virtual* member function. The final implementation of this function is hereby left to the derived class.

A. System model

The discrete transfer function model represents a linear single input - single output dynamic system. Concerning the field of digital signal processing, a discrete transfer function is considered to be a digital filter there.

Let the n_B and n_A denote the numerator and the denominator polynomials degree. Then the fully expanded form will be:

$$\frac{y(z)}{u(z)} = \frac{B(z)}{A(z)} = \frac{b_1 z^{-1} \dots + b_{n_B} z^{-n_B}}{1 + a_1 z^{-1} \dots + a_{n_A} z^{-n_A}} \quad (1)$$

Difference equation corresponding to transfer function (1) is in the form:

$$y(k) = \sum_{i=1}^{n_B} b_i u(k-i) - \sum_{i=1}^{n_A} a_i y(k-i) \quad (2)$$

The numeric realization of the equation (2) is also called the *direct form I* abbr. *DF-I* [5]. In order to apply the *DF-I* algorithm, the filter structure has to comprise separate memory vector for both input and output signal.

```
class TransferFunction: public DiscreteSystemSISO {
public:
    TransferFunction(size_t nb, size_t na);
    void update(const Vector<Signal>&, Vector<Signal>&);
private:
    Vector numerator_coeffs;
    Vector denominator_coeffs;
    VectorStates input_states;
    VectorStates output_states;
};
```

For the *DF-I* there are twice as many delays as necessary. As a result, the *DF-I* structure is not canonical with respect to delay. The update function is implemented in terms of the equation (2).

```
void TransferFunction::update(
    const Vector<Signal>& input, Vector<Signal>& output)
{
    real_t y = 0;
    const real_t u = input.at(0);
    input_states.at(0) = u;
    for (uint i = 0; i < numerator_coeffs.get_length(); i++)
        { y += numerator_coeffs[i] * input_states[i]; }
    ++input_states;
    size_t order = get_order();
    for (uint i = 0; i < order; i++)
        { y += -denominator_coeffs[i] * output_states[i]; }
    output_states.at(0) = y;
    ++output_states;
    output.at(0) = y;
}
```

B. Circular buffer abstraction

Transfer function like systems generally represent a canonical form of a state space representation. It means, that the equivalent state vector contains the past states of the output signal y . Therefore for each discrete step, the states should get one sample time older, e.g. value $y(k-1)$ becomes $y(k-2)$ and so on. This would normally require a computationally expensive shifting and overwriting of all last vector elements. In time critical applications, especially while using limited resources, this is unacceptable. In order to deal with this problem, the circular buffer abstraction has been proposed to be used.

The circular buffer can be seen as a generic vector of the length N extended by an indexing variable i of integer type. This variable always points to the position of the latest sample. Only the indexing variable is incremented during the states update process rather than shifting all elements of the vector. After reaching the end of the vector, the indexing variable overflows, so the buffer works in circular mode. Equivalent diagram is illustrated in the figure 1.

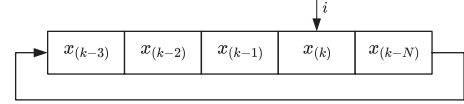


Fig. 1: Circular buffer diagram

Circular buffer was implemented as the C++ class:

```
class VectorStates: public Vector {
public:
    void operator++() { i++; i %= length; }
    real_t& at(uint k) { return Vector::at((k+i+1)%length); }
    real_t& at() { return Vector::at(i); }
private:
    uint i = 0;
};
```

The update function was implemented as the pre-increment operator *operator++()*. The indexed sample access interface implements the virtual function *at* of the base class *Vector*.

C. Controller structure

We consider the general controller structure with three degrees of freedom expressed in the linear polynomial form:

$$R(z)u(k) = T(z)w(k) - S(z)y(k) \quad (4)$$

Where $R(z)$, $S(z)$ and $T(z)$ are polynomials defined as:

$$\begin{aligned} R(z) &= 1 + r_1 z^{-1} \dots + r_{n_R} z^{-n_R} \\ S(z) &= s_0 + s_1 z^{-1} \dots + s_{n_S} z^{-n_S} \\ T(z) &= t_0 + t_1 z^{-1} \dots + t_{n_T} z^{-n_T} \end{aligned} \quad (5)$$

Compared to the traditional PID controller, separate setpoint w and feedback y signals are provided, instead of single error signal e . This allows to tune dynamic behaviour of the closed loop using an appropriate parameter synthesis method (e.g. pole placement control, model predictive control, linear quadratic control).

The controller structure (4) consists of three main sections:

- R - controller output IIR component
- S - feedback FIR component
- T - feed-forward FIR component

The software implementation is similar to a discrete transfer function yet one additional FIR component has to be used:

```
class RST_Controller: public DiscreteSystemMIMO {
public:
    RST_Controller(size_t nR, size_t nS, size_t nT);
    Vector R_coeffs;
    Vector S_coeffs;
    Vector T_coeffs;
    void update(const Vector<Signal>&, Vector<Signal>&);
private:
    VectorStates u_states;
    VectorStates y_states;
    VectorStates w_states;
};
```

$$\begin{pmatrix} a_0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \\ a_1 & a_0 & \dots & 0 & b_1 & 0 & \dots & 0 \\ a_2 & a_1 & \dots & 0 & b_2 & b_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n_A} & a_{n_A-1} & \dots & a_0 & b_{n_B} & b_{n_B-1} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_{n_A} & 0 & 0 & \dots & b_{n_B} \end{pmatrix} \begin{pmatrix} r_0 \\ \vdots \\ r_{n_R} \\ s_0 \\ \vdots \\ s_{n_S} \end{pmatrix} = \begin{pmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \\ \vdots \\ p_{n_P} \end{pmatrix} \quad (3)$$

The implemented class contains vectors of R , S and T polynomials coefficients and also the memory vectors (as defined in section II-B) for the corresponding signals u , y and w .

In almost every practical control application, the controller output signal u is constrained to the certain interval. For example, typical digital to analog converters or PWM modulators operate within limited range of the output signal.

Because of this, the update function of the controller includes saturation operation in order to constrain the manipulated variable by the numeric way. If the autoregressive R component of the controller is internally aware of this, then the “windup” like effect won’t be present.

The update function of polynomial controller is implemented according to the equation (4) transformed into the following difference equation form:

$$u(k) = \sum_{i=0}^{n_T} t_i w(k-i) - \sum_{i=0}^{n_S} s_i y(k-i) - \sum_{i=1}^{n_R} r_i u(k-i) \quad (6)$$

```
void RST_Controller::update(
const Vector<Signal>& input, Vector<Signal>& output)
{
w_states.at(0) = input.at(0);
real_t u = 0;
for (uint i = 0; i < T_coeffs.get_length(); i++) {
u += T_coeffs[i] * w_states[i];
}
++w_states;
y_states.at(0) = input.at(1);
for (uint i = 0; i < S_coeffs.get_length(); i++) {
u += -S_coeffs[i] * y_states[i];
}
++y_states;
for (uint i = 0; i < R_coeffs.get_length() - 1; i++) {
u += -R_coeffs[i] * u_states[i];
}
u = saturate(u);
u_states.at(0) = u;
++u_states;
output.at(0) = u;
}
```

This can be also seen as the two input modification of a $DF-I$ type discrete filter as defined in the section II-A.

III. CONTROL DESIGN

The general aim of the controller synthesis procedure is to achieve the desired dynamic behaviour of the closed control loop while implicitly providing its stability. The pole-placement method has been chosen for this purpose and hence is implemented further in this paper.

This method proposes to force the dynamics of the controller-plant closed loop via placing roots of its characteristic polynomial. The pole-placement method also implicitly

provides the closed loop stability, but only if the desired polynomial P stability condition is met.

We assume the controlled system model in the form (1). The closed loop transfer function with the controller (4) has the following form:

$$\frac{y(z)}{w(z)} = \frac{B(z)T(z)}{A(z)R(z) + B(z)S(z)} \quad (7)$$

Let the desired closed loop characteristic polynomial to be $P(z)$. Applying the pole-placement method we demand the following equality:

$$A(z)R(z) + B(z)S(z) = P(z) \quad (8)$$

According to the convolution theorem for polynomial multiplication, the following dimensions conditions must be met:

$$\begin{aligned} n_P &= n_B + n_A - 1 = n_R + n_S + 1 \\ n_R &= n_B - 1 \quad n_S = n_A - 1 \end{aligned} \quad (9)$$

The diophantine equation (8) can be written as an equivalent system of linear equations (3) using convolution matrices for $B(z)$ and $A(z)$ polynomials.

Then the closed loop transfer function (7) can be rewritten:

$$\frac{y(z)}{w(z)} = \frac{B(z)T(z)}{P(z)} \quad (10)$$

The T polynomial is usually designed as a zero degree (t_0 only) to assure the unit static gain of closed loop transfer function (10):

$$T(z) = \sum \frac{p_i}{b_i} \quad (11)$$

The above approach was implemented into our library using the matrix LU decomposition subroutine.

```
void RST_poleplace(
const Vector& A, const Vector& B, const Vector& P,
Vector& R, Vector& S, Vector& T)
{
uint R_length = B.length-1;
uint S_length = A.length-1;
Matrix M(P.length, P.length);
Matrix M_submat_A(M, P.length, R_length, 0, 0);
Matrix M_submat_B(M, P.length, S_length, 0, R_length);
convMat(A, M_submat_A, R_length);
convMat(B, M_submat_B, S_length);
Vector X=M.solve(P);
R = Vector(X, R_length, 0);
S = Vector(X, S_length, R_length);
T[0] = P.sum() / B.sum();
}
```

IV. SYSTEM IDENTIFICATION

For a control synthesis method to be correctly applied, the parameters of the controlled system have to be known or at least to be estimated. Therefore the recursive least-squares method (abbr. RLS) [6] is implemented into our library.

We assume the controlled system in the form of an ARX model.

$$A(z^{-1})y_{(k)} = B(z^{-1})u_{(k)} + \epsilon_{(k)} \quad (12)$$

Where ϵ stands for the exogenous random disturbance signal. The model output $y_{(k)}$ can be expressed in vector form:

$$y_{(k)} = h_{(k)}^T \theta + \epsilon_{(k)} \quad (13)$$

$$\theta^T = [b_1, b_2 \dots b_{n_B}, a_1, a_2 \dots a_{n_A}] \quad (14)$$

$$h_{(k)}^T = [u_{(k-1)}, \dots, u_{(k-n_B)}, -y_{(k-1)}, \dots, -y_{(k-n_A)}] \quad (15)$$

Regression vector $h_{(k)}$ has to be updated (iterated) for each sample time. The slightly modified circular buffer, proposed in the section II-B, can be conveniently exploited for this purpose. The implementation of the regression vector leads to the following class:

```
class VectorARXRegressor: public Vector {
public:
    void update(real_t, real_t);
    real_t& at(uint);
private:
    VectorStates u_states;
    VectorStates y_states;
};
```

The update function implementation iterates circular buffer for the input and the output signal separately.

```
void VectorARXRegressor::update(real_t u, real_t y) {
    ++u_states; u_states.at() = u;
    ++y_states; y_states.at() = -y;
}
```

The indexed element access function implements the virtual function *at* of the base class *Vector*.

```
real_t& VectorARXRegressor::at(uint n)
if (n < u_states.get_length())
    return u_states[n];
else if (n < y_states.get_length() + u_states.get_length())
    return y_states[n - u_states.get_length()];
}
```

Finally the RLS algorithm equations are given as follows:

$$Y_{(k+1)} = \frac{P_{(k)} h_{(k+1)}}{\lambda + h_{(k+1)}^T P_N h_{(k+1)}} \quad (16)$$

$$P_{(k+1)} = \frac{1}{\lambda} \left(I - Y_{(k+1)} h_{(k+1)}^T \right) P_{(k)} \quad (17)$$

$$\hat{y}_{(k+1)} = h_{(k+1)}^T \hat{\theta}_{(k)} \quad (18)$$

$$e_{(k+1)} = y_{(k+1)} - \hat{y}_{(k+1)} \quad (19)$$

$$\hat{\theta}_{(k+1)} = \hat{\theta}_{(k)} + e_{(k+1)} Y_{(k+1)} \quad (20)$$

Where λ denotes forgetting factor and P is the covariance matrix.

For the RLS method to evaluate, only basic matrix operations are required to be implemented. In this paper we implemented the above algorithm in the C++ language exclusively using the original matrix and vector interface.

```
class RLS {
public:
    RLS(size_t n_params, real_t P0);
    real_t estimate(const Vector& hk_1, real_t yk_1);
private:
    Matrix Pk_0;
    Matrix Pk_1;
    Matrix YhT;
    Vector theta;
    Vector Y;
    real_t lambda;
};
```

The above RLS class stores some of the matrices and the vectors objects as member ones, since their dynamic allocation in every iteration would be ineffective. However this allocation is performed only during the class construction because all the objects dimensions are fixed for the certain number of estimated parameters.

```
real_t RLS::estimate(const Vector& hk_1, real_t yk_1)
{
    Pk_0 = Pk_1;
    error = yk_1 - hk_1.dot_product(theta);
    Pk_0.multiply(hk_1, Y);
    Y /= (hk_1.dot_product(Y) + lambda);
    YhT.multiplyTvector(Y, hk_1);
    YhT.diagonal() -= 1.0;
    YhT /= -lambda;
    Pk_1.multiply(YhT, Pk_0);
    Y *= error;
    theta += Y;
    return error;
}
```

In order to optimize algorithm complexity and memory requirements, some of equations (16) - (20) were slightly reshaped in the way of minimizing the overall number of performed operations as possible.

V. REAL-TIME APPLICATION

All the above mentioned algorithms have been applied in the real-system control application. Such a practical demonstration, among other things, demands real-time operation support, so using dedicated operating system is necessary. Namely the *FreeRTOS* minimalistic real time kernel operating system suitable for embedded systems was chosen for this purpose. The *FreeRTOS* contains tasks scheduler for preemptive multi-tasking and includes basic interface for periodic control loops execution as well.

The controlled system is a simple thermal system consisting of a resistor being heated by the input electric power. Temperature $T(t)$ of the system, representing the controlled signal, has been sensed using NTC thermistor and then sampled by on-chip analog-digital converter.

However, the system input voltage signal, representing the manipulated variable, is not continuous in time since the pulse width modulation has been used. This modification leads to proportional relation of manipulated variable (aka. PWM duty $d(t)$) to the input electric power.

Even if this signal is dis-continuous in reality, yet for a small enough PWM period, relatively to the time constant of the system, it can be further assumed continuous. The duty signal $d(t)$ [unitless] lies within the interval $(0,1)$ therefore the controller output has to be saturated.

Since the system operates in real conditions, a non-zero initial value of temperature is always present. Also the steady state of the system is affected by ambient temperature $T_{amb}(t)$. For the identification and control purposes the feedback signal has to be modified. Signal $T_{\Delta}(t)$ has to be relative to the ambient temperature.

$$T_{\Delta}(t) = T(t) - T_{amb}(t) \quad (21)$$

The ambient temperature T_{amb} can be measured by an additional sensor. In order to avoid the necessity of another sensor, the T_{amb} can be determined as system steady state (for $d = 0$).

A. Identification

For the numerical control implementation, the parameters of ARX model have been estimated using the RLS method. The recursive least squares method can fully supply the functionality of general least squares, while the forgetting factor has to be set as $\lambda = 1$ (no forgetting). Even this recursive version is generally an on-line method, we applied it as off-line for memory requirements reduction.

A step-like multilevel signal has been applied as the exciting signal for the identification procedure. The sampling period has been set to $T_s = 3$ s. ARX model orders were determined by a qualified guess as $n_B = 2$ and $n_A = 2$ respectively.

The identification is then performed within the loop:

```
ADC_input input(&hadc1, ADC1, ADC_channels, 12, 3.3);
PWM_output output(&htim3, TIM3, PWM_channels, 5000);
NTC_thermistor thermistor(...);

const real_t Ts = 3.0;
real_t T_amb_est = thermistor.U_to_T(input.read());
size_t nb = 2; size_t na = 2;

VectorARXRegressor h(nb, na);
RecursiveLeastSquares RLS(nb + na, 1e6);
Vector B(nb + 1); Vector A(na + 1);
VectorDiscreteParameters theta(B, A);
real_t ident_data[] = {0.2, 0.5, 0.0, 0.75, 1.0, 0.0};
SignalSampled ident_signal(6, 30, ident_data);

while (1) {
    try {
        input.sample();
        real_t T = thermistor.U_to_T(input.read());
        real_t u = ident_signal.at(i * Ts);
        RLS.estimate(h, T - T_amb_est);
        h.update(u, T - T_amb_est);
        output.setDuty(u, 0);
        vTaskDelayUntil(&xLastWakeTime, Ts);
        i++;
    }
    catch (...)
    { break; }
}
```

The ambient temperature has been estimated as $T_{amb} = 25.3$ °C and has been considered to be constant during identification and control experiments.

The resulting transfer function is as follows:

$$F(z) = \frac{T_{\Delta}(z)}{d(z)} = \frac{1.047z^{-1} + 0.049z^{-2}}{1 - 1.451z^{-1} + 0.460z^{-2}} \quad (22)$$

Figure 2a shows the measured system response T together with the input signal u . The estimated output of the model \hat{T} using single step ahead prediction $\hat{T} = h^T\theta + T_{amb}$ and also the simulated model output T_{sim} using the discrete

transfer function evaluation are plotted in the figure in order to validate the model. Both estimated \hat{T} and simulated T_{sim} output signals of the model fit the measured temperature data sufficiently, so the estimated model can be considered valid.

B. Control

Let the continuous-time desired characteristic polynomial $P(s)$ represent an aperiodic transient response with the multiple time constant T_c . For the final deployment the time constant was chosen as $T_c = 2.75$ s.

$$P(s) = \prod_{i=1}^{n_P} (T_c s + 1) \quad (23)$$

This desired polynomial $P(s)$ needs to be transformed to its discrete equivalent first. The *poles-zeros match* method can provide exact (not just approximative) transformation of poles and zeros between continuous and discrete time domains while preserving full dynamic and stability properties.

Consequently, the discrete desired polynomial $P(z)$ can be written as product:

$$P(z) = \prod_{i=1}^{n_P} \left(z - e^{-\frac{T_s}{T_c}} \right) \quad (24)$$

Following the control design procedure described in section III the resulting linear polynomial controller (4) has been obtained.

```
real_t T_c = 2.75;
Polynom P=create_aperiodic_polynom(nb + na - 1, T_c, Ts);
Discrete_RST_Controller controller(nb - 1, na - 1, 0);
controller.set_saturation(1.0, 0.0);
RST_poleplace(
    A, B, P,
    controller.R, controller.S_coeffs, controller.T_coeffs);

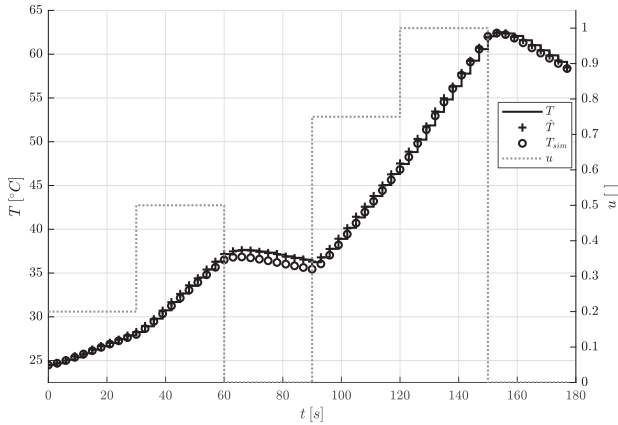
while (1) {
    input.sample();
    real_t T = thermistor.U_to_T(input.read());
    controller.input(0) = T_ref - T_amb_est;
    controller.input(1) = T - T_amb_est;
    controller.update();
    real_t u = controller.output(0);
    output.setDuty(u);
    vTaskDelayUntil(&xLastWakeTime, Ts);
}
```

Finally, the control can be applied in a periodic loop using sampled temperature signal.

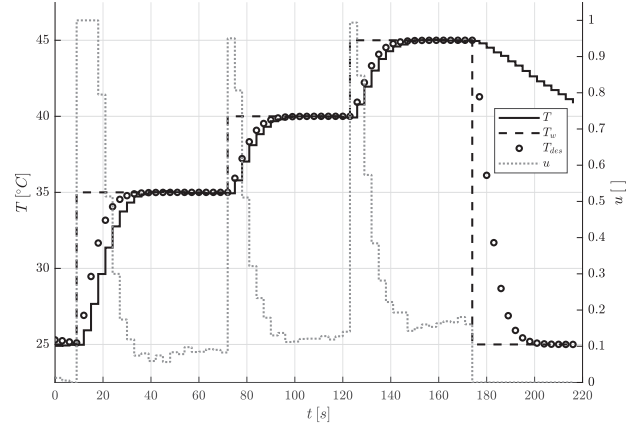
For the hardware realisation of control system see figures 3 and 4. In details, the *STM32F446RE* (180 MHz, 512 KB Flash, 128 KB SRAM) micro-controller embedded in the *Nucleo* evaluation board was used.

C. Results

Figure 2b shows the closed loop system response T together with the set-point signal T_w and corresponding controller output signal u . The desired closed loop response T_{des} is also depicted in order to validate the control performances. It can be seen that the controlled temperature signal T matches closely the desired dynamics T_{des} . Slight differences occur in case of manipulated variable saturation, yet in such a case no windup effect has raised up.



(a) Thermal system identification



(b) Thermal system control

Fig. 2: Thermal system measured identification and control responses

VI. CONCLUSION

An optimized software implementation of the specific algorithms from the control theory domain has been shown in this paper. The library was implemented using the C++ programming language while taking into account possible limited hardware resources.

The linear polynomial controller structure was proposed as an versatile discrete control algorithm, moreover it's suitable for effective software implementation. The pole-placement method has shown to be an appropriate controller synthesis method, hence was implemented using the convolution matrix subroutine and the LU decomposition subroutine. For the system identification purposes, the recursive least squares method algorithm was effectively implemented and further applied for the ARX model parameters estimate.

The implemented framework, comprising all the mentioned algorithms, was successfully verified in the practical real-time temperature control application. The *STM32* family micro-controller running the *FreeRTOS* operating system was exploited for this purpose. Created software library has been published as open-source project and can be found at: <https://github.com/dodekm/System-control-library>.

ACKNOWLEDGEMENT

This publication was created thanks to support under the Operational Program Integrated Infrastructure for the project: *International Center of Excellence for Research on Intelligent and Secure Information and Communication Technologies and Systems - II. stage*, ITMS code: *313021W404*, co-financed by the European Regional Development Fund

REFERENCES

- [1] M. Gifftaler, M. Neunert, M. Stäuble, and J. Buchli, "The control toolbox — an open-source c++ library for robotics, optimal and model predictive control," *2018 IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAN)*, pp. 123–129, 2018.
- [2] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi, "Gnu scientific library reference manual," *Bristol*, 01 2009.

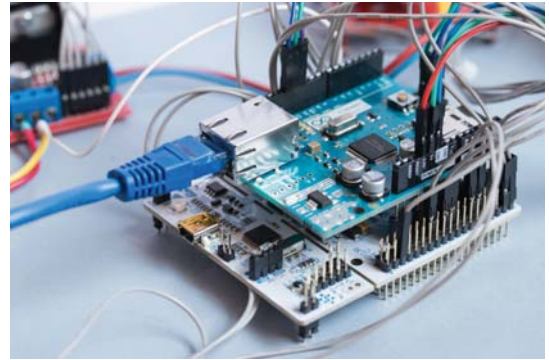


Fig. 3: Micro-controller board

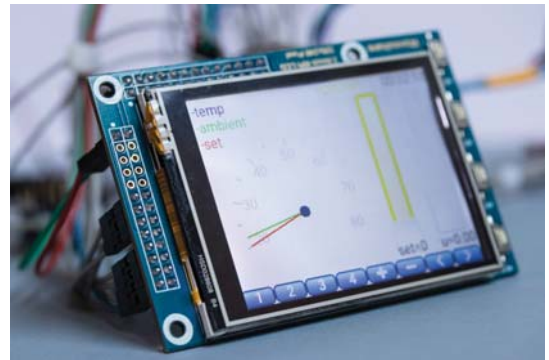


Fig. 4: Display touch-screen

- [3] P. Benner, V. Mehrmann, V. Sima, S. Huffel, and Varga, "Slicot: A subroutine library in systems and control theory," *European Community BRITE-EURAM III Thematic Networks Programme NICONET - Working Group on Software*. ESAT Katholieke Universiteit Leuven, 07 1998.
- [4] R. T. Berman, "Using c++ to write automation controller software," *JALA: Journal of the Association for Laboratory Automation*, vol. 12, no. 1, pp. 12–16, 2007.
- [5] A. Oppenheim, R. Schaffer, and R. Schaffer, *Discrete-time Signal Processing*, ser. Prentice-Hall signal processing series. Pearson, 2010.
- [6] L. Ljung, *System Identification: Theory for the User*, ser. Prentice Hall information and system sciences series. Prentice Hall PTR, 1999.



Proceedings of the
**2021 23rd International Conference
on Process Control (PC)**

Virtual Event
Štrbské Pleso, Slovakia
June 1 – 4, 2021

Editors

R. Paulen and M. Fikar

IEEE Catalog Number: CFP21PCB-ART

ISBN: 978-1-6654-0330-6

Technically Sponsored by



IEEE Czechoslovakia Section

Organisers

Institute of Information Engineering, Automation, and Mathematics

Faculty of Chemical and Food Technology
Slovak University of Technology in Bratislava

Copyright and Reprint Permission: Abstracting is permitted with credit to the source. Libraries are permitted to photocopy beyond the limit of U.S. copyright law for private use of patrons those articles in this volume that carry a code at the bottom of the first page, provided the per-copy fee indicated in the code is paid through Copyright Clearance Center, 222 Rosewood Drive, Danvers, MA 01923. For reprint or republication permission, email to IEEE Copyrights Manager –at pubs-permissions@ieee.org. All rights reserved. Copyright ©2019 by IEEE.

System Maintenance



The site is currently undergoing maintenance at this time.
There may be intermittent impact on performance. We apologize for any inconvenience.

SUBSCRIBE

Cart



IEEE Xplore[®]

Institutional Sign In

All



Search within Publication

ADVANCED SEARCH



[Browse Conferences](#) > [International Conference on Pr...](#) > [2021 23rd International Confer...](#)



International Conference on Process Control (PC)



Proceedings

All Proceedings

Popular

2021 23rd International Conference on Process Control (PC)

1-4 June 2021

Showing 1-25 of 59

Filter

Sort

Email



[Title page]



[2021 23rd International Conference on Process Control \(PC\)](#)

Year: 2021

IEEE websites place cookies on your device to give you the best user experience. By using our websites, you agree to the placement of these cookies. To learn more, read our

[Privacy Policy.](#)

Accept & Close



Committees

2021 23rd International Conference on Process Control (PC)

Year: 2021



Foreword

2021 23rd International Conference on Process Control (PC)

Year: 2021



Lyapunov Based Adaptive Control for Varying Length Pendulum with Unknown Viscous Friction

Milan Anderle; Sergej Čelikovský; Tomáš Vyhlídal

2021 23rd International Conference on Process Control (PC)

Year: 2021



Swing compensation of a payload suspended to a planar copter

Matěj Kuře; Jaroslav Bušek; Tomáš Vyhlídal

2021 23rd International Conference on Process Control (PC)

Year: 2021



Stabilization of large-scale systems and consensus of multi-agent systems with time delay - common features and differences

Branislav Rehák; Volodymyr Lynnyk

2021 23rd International Conference on Process Control (PC)

Year: 2021



Implementation of selected control theory algorithms for embedded real-time systems

Martin Dodek; Eva Miklovičová; Marián Tárník

2021 23rd International Conference on Process Control (PC)

Year: 2021



Decentralized control for deployment of multi-agent dynamical systems

Katarzyna Topolewicz; Ewa Girejko; Sorin Olaru

2021 23rd International Conference on Process Control (PC)

Year: 2021



Control of Discrete-Time State-Multiplicative Systems with Polytopic Region of Uncertainty 

Dušan Krokavec; Anna Filasová

2021 23rd International Conference on Process Control (PC)

Year: 2021

Generalized synchronization of chaotic systems in a master-slave configuration 

Volodymyr Lynnyk; Sergej Čelikovský

2021 23rd International Conference on Process Control (PC)


Year: 2021

Usage of Homomorphic Encryption Algorithms in Process Control 

Matúš Furka; Karol Kiš; Martin Klaučo; Michal Kvasnica

2021 23rd International Conference on Process Control (PC)


Year: 2021

On the Quadratic Programming Solution for Model Predictive Control with Move Blocking 

Pavel Otta; Ondřej Šantin; Vladimír Havlena

2021 23rd International Conference on Process Control (PC)

Year: 2021

State space sets with common optimal feedback laws for nonlinear MPC 

Ruth Mitze; Raphael Dyrcka; Kai König; Martin Mönnigmann

2021 23rd International Conference on Process Control (PC)


Year: 2021

Hidden Markov Model-based Warm-start of Active Set Method in Model Predictive Control 

Roman Kohút; Lenka Galčířová; Kristína Fedorová; Tereza Ábelová; Monika Bakořová; Michal Kvasnica

2021 23rd International Conference on Process Control (PC)

Year: 2021

-
- Worst-case optimal scheduling and real-time control of a microgrid offering active power reserve** 

Marko Kovačević; Branimir Brkić; Mario Vašak
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Dynamic optimization of low-pressure carburizing furnaces** 

Fatima Matamoros Marin; Abderrazak M. Latifi; Pierre-Alexandre Glaude; Roda Bounaceur; Hubert Monnier
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Distributed Optimal Heating Control of Building Zones** 


Filip Vrbanc; Vinko Lešić
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Tumor Growth Control with Positive Input LPV Controller** 


György Eigner; Dániel András Drexler; Alajos Mészáros; Levente Kovács
2021 23rd International Conference on Process Control (PC)
Year: 2021


- Reduced model of boundary value dynamics of 1D diffusion - application to heat conduction** 


Pavel Skopec; Tomas Vyhlidal; Jan Knobloch
2021 23rd International Conference on Process Control (PC)
Year: 2021


- Control Principles of Autonomous Mobile Robots Used in Cyber-Physical Factories** 

Květoslav Belda; Jan Jirsa
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Determination of the optimal packing configuration of a catalytic fixed-bed reactor using geometry and multi-objective optimization methods** 
Alexis Courtais; François Lesage; Yannick Privat; Abderrazak M. Latifi
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Hydroelectric Power-Plant Simulator Implemented in Python** 
Michal Kuchař; Adam Pechl; Milan Kučera; Jaromír Fišer; Pavel Kulík; Tomáš Vyhlídal
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Envelope Time as Measure of Control Quality** 
Darina Bártová; Jaromír Kuka; Jan Mareš; Jan Vrba
2021 23rd International Conference on Process Control (PC)
Year: 2021

- Engineering Psychology as a Humanities for Engineers** 
Jakub Jura; Matous Cejnek; Pavel Trnka; Martin Cahyna
2021 23rd International Conference on Process Control (PC)
Year: 2021

[Load More](#)

IEEE Personal Account

[CHANGE USERNAME/PASSWORD](#)

Purchase Details

[PAYMENT OPTIONS](#)

[VIEW PURCHASED DOCUMENTS](#)

Profile Information

COMMUNICATIONS PREFERENCES

PROFESSION AND EDUCATION

TECHNICAL INTERESTS

Need Help?


US & CANADA: +1 800 678 4333

WORLDWIDE: +1 732 981 0060

CONTACT & SUPPORT

Follow



[About IEEE Xplore](#) | [Contact Us](#) | [Help](#) | [Accessibility](#) | [Terms of Use](#) | [Nondiscrimination Policy](#) | [IEEE Ethics Reporting](#)  | [Sitemap](#) | [Privacy & Opting Out of Cookies](#)

A not-for-profit organization, IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity.

© Copyright 2021 IEEE - All rights reserved. Use of this web site signifies your agreement to the terms and conditions.

International Program Committee

Chairman: Paulen, R. (SK)

Honorary chairman: Fikar, M. (SK)

Area Chairs:

- Hovd, M. (NO): Linear and Nonlinear Control
- Houska, B. (CN): Optimization and Computing in Control
- Lucia, S. (DE): Machine Learning and Control
- Haniš, T. (CZ): Sustainable Industrial Production and Applications

Members:

- Camacho, E. (ES)
- Chachuat, B. (UK)
- Chida, Y. (JP)
- Czczot, J. (PL)
- Dochain, D. (BE)
- Doležel, P. (CZ)
- Engell, S. (DE)
- Faulwasser, T. (DE)
- Haniš, T. (CZ)
- Henrion, D. (FR)
- Houska, B. (CN)
- Hovd, M. (NO)
- Hromčík, M. (CZ)
- Hurák, Z. (CZ)
- Kvasnica, M. (SK)
- Kovács, L. (HU)
- Kozáková, A. (SK)
- Latifi, R. (FR)
- Lucia, S. (DE)
- Mareš, J. (CZ)
- Moennigmann, M. (DE)
- Olaru, S. (FR)
- Pekař, L. (CZ)
- Petráš, I. (SK)
- Rohal'-Ilkiv, B. (SK)
- Rosinová, D. (SK)
- Rossiter, A. (UK)
- Rudas, I.J. (HU)
- Šebek, M. (CZ)
- Schlegel, M. (CZ)
- Stojanovski, G. (MK)
- Streif, S. (DE)
- Tanaskovic, M. (RS)

National Organising Committee

Chairman: Klaučo, M. (SK)

Co-chairman: Oravec, J. (SK)

Members:

- Bakaráč, P. (SK)
- Čirka, Ľ. (SK)
- Horváthová, M. (SK)