

SESSION D: AUTOMATIC CONTROL, IT AND ARTIFICIAL INTELLIGENCE

Hybrid simulation of an inverted pendulum

Lazar Kelić¹

¹ University of Kragujevac, Faculty of Mechanical and Civil Engineering, Kraljevo, Serbia

ARTICLE INFO

* Correspondence: delboyrodney@gmail.com

DOI: <https://doi.org/10.46793/ET26.D01K>

PAGES: D1 – D9

ABSTRACT

Hybrid simulation (Hardware-in-the-Loop) of an inverted pendulum is an approach that combines numerical simulation and physical experimentation, with the aim of faithfully replicating real operating conditions in the control of dynamic systems. In this paper, an experimental setup is implemented in which the system dynamics - the cart motion and the pendulum angle - are calculated on a personal computer using the Python environment and the SciPy library, while the control algorithm is executed on the Arduino microcontroller platform. This division enables the simulation and control to run on independent processors, which brings the operating conditions closer to real systems. The mathematical model of the system is based on nonlinear differential equations that describe the cart position and the pendulum deflection angle, while the microcontroller receives the state values in real time and calculates the control signal using the LQR algorithm. In this way, interaction is achieved between the software and hardware parts - Python represents the physics model, while Arduino represents the real controller.

The advantage of this approach is the possibility of analyzing delay and time response under conditions close to real ones, which is often neglected in classical simulations. Different microcontroller platforms (ATmega328P, ATmega32u4, ESP32 with FPU) make it possible to experimentally evaluate stability, latency and control-computation speed. The influence of the sampling frequency and communication channels (serial transmission) on system performance is also analyzed. Hybrid simulation represents a bridge between theoretical modeling and practical implementation, enabling safe testing of control algorithms and comparison of hardware platforms in real time.

KEYWORDS

Inverted pendulum, Hybrid simulation, LQR control, System delay, Multithreaded modeling, Real-time systems; Hardware-in-the-Loop (HIL), Arduino Leonardo, ESP32, System dynamics, Python simulation, Controller optimization, Latency, Digital control, Experimental verification.

1. INTRODUCTION

The inverted pendulum system is a classical problem in control theory because of its unstable equilibrium and nonlinear dynamics [1]. A typical implementation includes a cart with a pendulum in the upright position, while the control task is to maintain the pendulum in that position by controlling the cart position or the actuator. Modeling of such a system often begins with linearization around the upright position, after which the state equations are analyzed in detail in the form [2]

$$\dot{x} = Ax + Bu \quad (1)$$

To stabilize this linearized system, a linear-quadratic regulator (LQR) is often used, which minimizes the cost function

$$J = \int (x^T Q x + u^T R u) dt \quad (2)$$

and provides optimal feedback boundary control[3]. Previous studies have shown that LQR provides fast transients and a minimal step in simulations [4].

However, classical simulations implemented in software environments such as MATLAB/Simulink or similar tools have limitations when it comes to real execution of the control law on embedded hardware and the influence of factors such as delay, sampling and real actuators [5]. A control law designed in simulation must also take into account further processing, data loading, communication and execution on a microcontroller, which can significantly affect real-time system performance [6].

In this context, multithreading and the software architecture of embedded platforms (including interrupts, tasks and serial communication) play an important role and may be a primary causal component of latency, even greater than the computing power itself [7].

At the same time, increasing attention is being given to hybrid simulations in which the system dynamics are calculated on one processor (for example, a computer running Python), while the control logic is implemented on an independent controller (for example, Arduino) connected to the simulation framework through real-time communication. Such an approach enables a more realistic product evaluation because the simulation and control run on different platforms, allowing the influence of real hardware latencies and processing to be assessed [8]. One recent study emphasizes that bridging the gap between theory and practice requires an environment in which the simulation “lives” together with the physical platform [9].

The literature also includes studies comparing different control techniques applied to the inverted pendulum: comparisons of PID, SMC, fuzzy control and LQR show that, although advanced algorithms are attractive, in implementations with limited microcontroller resources the path to stability also requires careful study of latency and computing architecture [10]. Examples of simulations correlated with real experiments show that — even when the control law performs well in simulation — a simple microcontroller implementation can suffer degradation due to delays and process congestion [11].

Classical modeling of pendulum-dynamics simulation includes calculating position, angle, velocity and acceleration parameters, often assuming continuous time and idealized samples. However, when the system is transferred to a real platform, the time step ΔT , sampling and controller response affect performance. Studies that examined the influence of ΔT found that larger time steps can cause oscillations or even loss of stability [12]. It is particularly emphasized that microcontroller implementations must take into account deterministic execution flow and minimize variance in time response [13].

In this paper, we present an architecture that combines classical simulation of system dynamics (on a computer) and implementation of the controller on a microcontroller as an independent processor. This makes it possible to measure and optimize the simulation time step, communication latency and control cycle. In particular, we investigate how the form of the multithreaded model, interrupts and controller architecture affect the final performance of the LQR controller in a hybrid setup. The aim is to determine which device and which procedure account for latency, stability and control accuracy, and to use experimental simulation to verify whether the results correspond to theoretical expectations.

2. ANALYSIS OF LQR CONTROL COMPUTATION AND THE INFLUENCE OF DENORMALIZED NUMBERS AND NONLINEARITY

2.1. LQR on the Arduino Platform

The linear-quadratic regulator (LQR) is an optimal control method that minimizes a specified cost function based on the system state. However, in real microcontroller implementations, especially on platforms with limited computational resources, the speed of calculating the control signal has a significant influence on system stability and performance. This section provides a detailed analysis of the execution cycle for calculating LQR control, as well as the influence of denormalized numbers and system nonlinearity on execution time.

2.2. Methodology for Estimating the Number of Processor Cycles

The estimates of the number of processor cycles used in this paper are based on a combination of theoretical analysis of microcontroller architecture and empirical values obtained from the literature and documentation for the platforms used. It should be emphasized that these values do not represent absolutely exact numbers, but rather order-of-magnitude approximations that enable a qualitative analysis of the influence of computational complexity on real-time system performance.

On microcontrollers without hardware support for floating-point operations (such as ATmega328P and ATmega32u4), all arithmetic operations on the float type are implemented in software through compiler libraries. These operations include several steps: normalization, exponent alignment, mantissa manipulation and rounding, which results in a significantly larger number of instructions compared with integer arithmetic. For this reason, the number of cycles for operations such as multiplication and addition is estimated in the range from several tens to several hundreds of cycles, depending on compiler optimization and memory-access method.

In more modern microcontrollers such as the ESP32, which has a hardware floating-point unit (FPU), arithmetic operations are executed much faster, although still not in a single cycle. The actual execution time depends on the pipeline architecture, cache-memory access and possible delays caused by the operating system (FreeRTOS), which introduces variability into the execution time.

Trigonometric functions represent a special case, because in most implementations they are realized through numerical approximations or lookup tables. Therefore, their execution time is not constant, but depends on the library and the range of input values; in this paper, order-of-magnitude values (1000–2000 cycles) are used to illustrate the increase in computational complexity.

In addition to the arithmetic operations themselves, a real system is significantly affected by additional factors such as memory access, serial communication, interrupt handling and multithreading mechanisms. These factors can dominate the total execution time, which explains the experimentally obtained results in which nominally faster microcontrollers do not achieve the expected performance.

Based on the above, the cycle-count values used should be regarded as engineering approximations that enable comparison of different architectures and understanding of the influence of computational complexity on control dynamics, rather than as precise measured quantities.

2.3. Cycle-Count Estimate for Calculating the Control Signal

On microcontrollers without hardware support for floating point (such as Arduino Leonardo and Arduino Mega), all operations on real numbers are performed in software. This implies a larger number of processor cycles for each individual operation. Approximate values are:

Table 1: Estimated number of cycles

Operation	Estimated number of cycles
Multiplication (32-bit float)	~150 cycles
Addition (32-bit float)	~100 cycles
Memory read	~50 cycles
Negation (sign change)	~10 cycles

For calculating the control signal

$$u = Kx \quad (3)$$

where K is a vector of dimension 1×4 , and x is a state vector of dimension 4×1 , the following is required:

1. 4 multiplications: cycles $4 \times 150 = 600$
2. 3 additions: cycles $3 \times 100 = 300$
3. 4 memory reads: cycles $4 \times 50 = 200$
4. Negation of the result: 10 cycles

Total:

$$600 + 300 + 200 + 10 = 910 \text{ cycles} \approx 900 \text{ cycles (with optimization).}$$

2.4. Theoretical Assumption of Computation Time for Different Platforms

The computation speed directly depends on the processor clock frequency. The approximate execution time is:

Table 2: approximated computation time

Platform	Clock (MHz)	Computation time (μ s)
Arduino Leonardo	16	$900 / 16 = 56.25$
Arduino Mega	16	$900 / 16 = 56.25$
ESP32 (80 MHz)	80	$900 / 80 = 11.25$
ESP32 (240 MHz)	240	$900 / 240 = 3.75$

The ESP32 has a hardware FPU (Floating Point Unit), which enables floating-point operations to be performed in one cycle. In this case, the total number of cycles for LQR control is approximately 20, which gives the computation time:

$$20/240 = 0.083 \mu\text{s}.$$

2.5. Influence of Number Magnitude and Number of Decimal Places

The computation speed does not depend on the absolute magnitude of the number (for example, 0.1° or 45°), because all floating-point operations are performed in the same number of cycles. However, when denormalized numbers appear (very small values close to zero), the processor must perform additional internal normalization, which increases the computation time.

- Multiplication: 300–500 cycles (instead of 150)
- Addition: 200–400 cycles (instead of 100)
- Reading: 100 cycles (instead of 50)

Table 3: Influence of number magnitude and number of decimal places

Condition	Total cycles	Time on Arduino Mega and Leonardo (16 MHz)
Angle 0.1° (denormalized)	2210	138.125 μs
Angle 45° (normalized)	900	56.25 μs

Therefore, denormalized numbers slow down computation by approximately 2.5 times.

2.6. Influence of System Nonlinearity

The nonlinear model of the inverted pendulum requires the use of trigonometric functions and more complex expressions. An example of a nonlinear control equation is:

$$u = \frac{(M+m)g \sin \theta - m l \dot{\theta}^2 \sin \theta \cos \theta + u_{LQR}}{M+m(1-\cos^2 \theta)} \quad (4)$$

Such expressions require more computational resources. Approximate estimate:

Table 4: Estimated number of cycles for calculating complex operations

Operation	Estimated cycles
$\sin(\theta)$, $\cos(\theta)$	1000–2000 cycles
Other multiplications and additions	+500 cycles
Total	~3000–5000 cycles

Converted execution time:

Table 5: Number of cycles converted into execution time on different platforms

Platform	Clock (MHz)	Time (μs)
Arduino Leonardo	16	250
Arduino Mega	16	250
ESP32 (without FPU)	80	50
ESP32 (with FPU)	240	16.7

2.7. Practical Implications and Optimization

Problems:

- Denormalized numbers increase the number of cycles and reduce precision.
- Nonlinear models drastically increase computational complexity.

Solutions:

1. Data normalization — all inputs should be scaled into a range of normalized values.
2. Fixed point (Q format) — accelerates computation up to 50 times.
3. Sine/cosine lookup tables — eliminate the need to call the $\sin()$ and $\cos()$ functions.
4. Hardware FPU — using an ESP32 or ARM Cortex-M4/7 processor provides the best computational efficiency.
5. Implementation in a timer interrupt — ensures a fixed control period without delay.

2.8. Conclusion

The analysis performed shows that:

- Denormalized numbers can slow down LQR-control computation by up to 2.5 times.
- The nonlinear system model requires up to 5 times more cycles than the linearized model.
- The best performance is achieved on the ESP32 platform with a hardware FPU unit, where the computation time is on the order of 0.083 μs .
- The use of fixed point and value normalization is the most efficient way to optimize control on microcontrollers without an FPU.

These results are of key importance in the design of a real inverted-pendulum control system, where control stability directly depends on the speed of calculating the control signal.

3. EXPERIMENT

The objective of the experiment is to determine unambiguously whether hardware can be used for inverted-pendulum control and which hardware can be used. For the purposes of the experiment, a simulation of the inverted-pendulum dynamic system was used according to the formula

$$(M + m)\ddot{x} + m\ddot{\theta}\cos\theta - m\ddot{\theta}2\sin\theta + b\dot{x} = F \quad (5)$$

and

$$l\ddot{\theta} + g\sin\theta = \ddot{x}\cos\theta \quad (6)$$

The simulation was written in the Python programming language. Another Python script was written to communicate between the above-mentioned Arduino platforms and the Python script on the computer. The Python script takes the system state vector and sends it via a serial connection to the Arduino platform. For each Arduino platform, a script was written that receives data from the serial port, in this case the system state vector; the LQR controller then calculates the next action and returns the control value to the Python simulation through the serial connection. The Python simulation takes that value, calculates the next state of the system and sends a new state vector to the Arduino platform.

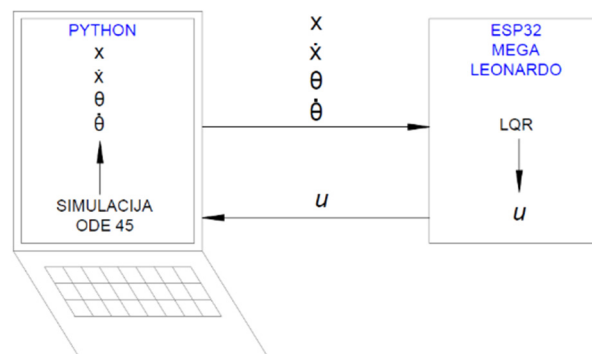


Figure 1: Communication between the Python script on the computer and the Arduino platform

3.1. Determination of LQR Gain

The parameters of the LQR regulator were determined using the standard optimal-control method for the linearized model of the inverted pendulum. Starting from the nonlinear equations of motion of the system, linearization was performed around the upright equilibrium position ($\theta \approx 0$), which produced a state-space model — equation (1), where $x = [x, \theta, \dot{x}, \dot{\theta}]$.

$\dot{\theta}]^T$ is the state vector and u is the control signal. The A and B matrices were formed on the basis of the known system parameters (cart mass, pendulum mass, pendulum length, gravity and friction coefficients), defined in the experimental part of the paper.

The LQR gain was obtained by minimizing the quadratic cost function — equation (2), where Q and R are weighting matrices that define the trade-off between state-tracking accuracy and control energy. By selecting appropriate Q and R matrices and solving the algebraic Riccati equation in the Python environment (SciPy library), the optimal gain was calculated, equation (3), where the gain vector K was used in the microcontroller implementation. The resulting gain vector has the following form: $K = [-10.0000 \ -136.6005 \ -20.5942 \ -27.7585]$, which gives the control law: $u = -(-10x - 136.6005\dot{x} - 20.5942\ddot{x} - 27.7585\ddot{\theta})$. The simulation is executed for up to 10 seconds. The obtained diagrams of pendulum stabilization and controller response speed are given below.

3.2. Experiment Settings

For the purposes of the experiment, the following simulation values were used:

- pendulum length l - 0.5 m,
- cart mass 200 g,
- pendulum mass 180 g
- gravitational acceleration g is 9.81 m/s²,
- friction in the pendulum shaft 0.1,
- cart sliding friction 0.1.

The script on the real hardware is executed as fast as possible, and the number of control-computation cycles in one second is limited exclusively by the hardware power of the controller itself. ΔT in the pendulum-dynamics simulation was set to 5 ms the first time and to 1 ms the second time. This experimental setup gives us the possibility to compare the classical simulation approach, where the state of the controlled system is presented with respect to the current iteration, with a more complex approach in which several independent processes are used and the state of the system is presented with respect to time regardless of the number of iterations (described in the system-delay section). In this case, it also enables testing of the simulation on real hardware while checking the capability of the hardware itself to control the given process.

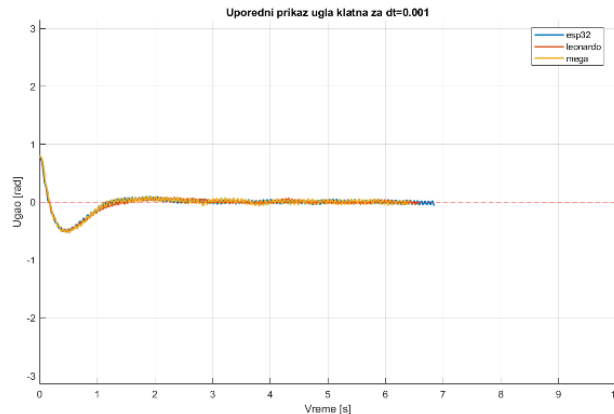


Figure 2: Comparative presentation of pendulum stabilization for $\Delta T = 1$ ms

Figure 2 shows that all three tested controllers perform the task in a very similar way. There is slight oscillation, but the results are essentially similar, with a negligibly better result for the Leonardo, which is certainly unexpected.

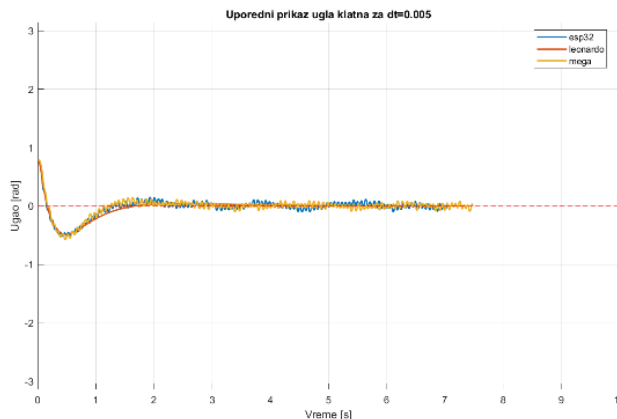


Figure 3: Comparative presentation of pendulum stabilization for $\Delta T = 5$ ms

Figure 3 shows a larger difference in favor of the Leonardo, with a much more accurate result, while the other two controllers show increased instability when ΔT in the system-dynamics simulation is increased to 5 ms.

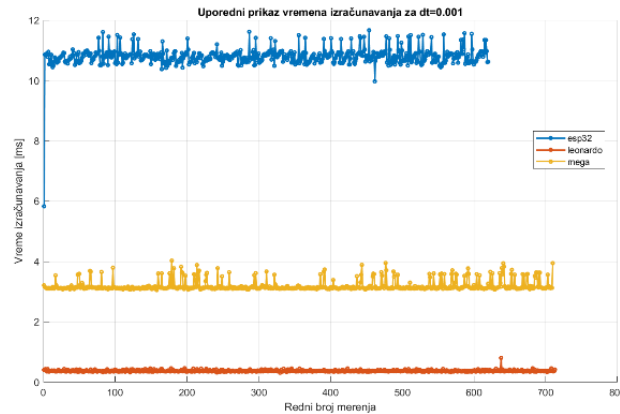


Figure 4: Comparative presentation of the computation speed of each iteration for $\Delta T = 1$ ms

Figure 4 shows the clear dominance of the Leonardo, while the ESP32 is considerably delayed in calculating the control action, which is certainly an unexpected result.



Figure 5: Comparative presentation of the computation speed of each iteration for $\Delta T = 5$ ms

Figure 5 shows a result that differs even more from expectations. The time step of the system-dynamics simulation was increased from 1 ms to 5 ms, and the result is that the Mega is the fastest, the Leonardo is slower than the Mega, while the ESP32 still far behind the weaker and slower controllers.

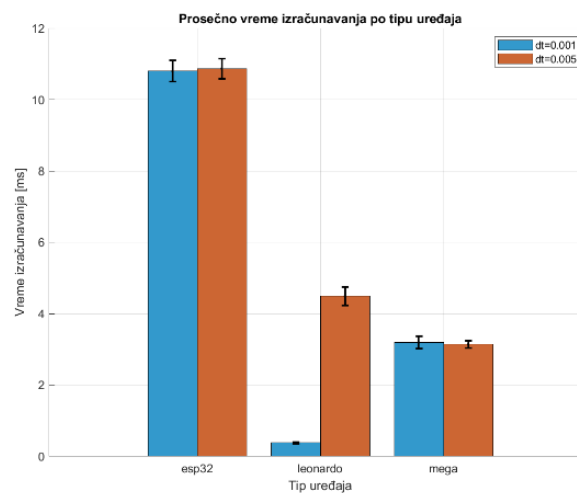


Figure 6: Average control-computation time for $\Delta T = 1$ ms and $\Delta T = 5$ ms

Figure 6 shows a simplified comparative presentation of the data shown in Figures 4 and 5. It can be seen that the ESP32 is by far the slowest, which is very unexpected despite all the hardware advantages it possesses. The computation speed on the Leonardo differs significantly and depends on the time step ΔT of the system-dynamics simulation, while the Mega operates in a balanced manner in both cases of the dynamic-system simulation time step ΔT .

4. INTERPRETATION OF RESULTS AND CONCLUSION

The analysis of the simulation results shown in Figures 2–6 reveals several very interesting and largely unexpected phenomena in the behavior of different controllers (Arduino Leonardo, Arduino Mega and ESP32) when calculating real-time control for the inverted-pendulum model.

Figure 2 shows that all three controllers achieve very similar results in system stabilization, with only slight oscillations and minimal deviations in the transient process. Although the ESP32 was expected to show a convincing advantage because of its significantly higher processor speed and support for hardware floating point, the best results in the initial tests were achieved by the Arduino Leonardo. The difference is small but consistent, indicating that in this operating mode the hardware potential of the ESP32 controller is probably not fully utilized, while the Leonardo, although weaker in hardware, is more stable in communication and data processing at a lower frequency level.

In Figure 3, the situation becomes more pronounced. When the time step of the system-dynamics simulation is increased to 5 ms, the Leonardo maintains good stability and control precision, while the Mega and especially the ESP32 show a noticeable increase in instability. This indicates a possible synchronization problem between the simulation and control processes. In the ESP32, due to parallel processing and the multithreaded structure, a delay may occur in serial communication or in the execution of the task that processes the pendulum data. In the Leonardo, by contrast, the entire process takes place linearly, without an internal scheduler, resulting in lower latencies and more consistent behavior over time.

Figure 4 further confirms the dominance of the Leonardo. While the ESP32 shows a pronounced response delay, the Leonardo tracks the desired signal almost perfectly. This result is completely contrary to expectations and indicates that hardware speed is not the only relevant factor. The probable cause is the different implementation of libraries and functions for floating-point arithmetic. While the ESP32 uses a 32-bit FPU that operates faster but requires multiple type conversions and data buffering within the FreeRTOS scheduler, the Leonardo executes all operations more simply and without an interrupt context, which results in lower effective delay.

Figure 5 shows another surprise: when ΔT is increased from 1 ms to 5 ms, the Mega proves to be the fastest, followed by the Leonardo, while the ESP32 remains the slowest. Here, the different architecture of memory access and timing interpretation within the simulation becomes important. The Mega has a stable timer and a deterministic execution flow, meaning that each computation is repeated in almost the same time interval. By contrast, the ESP32, which uses multiple threads and has a more complex internal organization, shows greater variability in computation time because the scheduler distributes processes depending on system priorities.

Figure 6 provides an overview of the entire situation and clearly shows that the ESP32 is the slowest, which is completely opposite to theoretical assumptions. Based on hardware specifications, the ESP32 would be expected to be several times faster than classical Arduino controllers; however, in practice it is evident that software architecture, internal task priorities and the method of communication with the external system play a key role in real time. The Leonardo, as a single-threaded microcontroller without an operating system, has an advantage in execution predictability and time consistency, which makes it more efficient in control tasks requiring minimal delay.

In general, the obtained results show that in real-time control systems, hardware power alone is not decisive; the way in which computation is integrated into the overall control process is also crucial. The ESP32, although technically superior, suffers losses due to software complexity, scheduler delays and multithreaded architecture. Arduino Leonardo and Mega, with a simple and deterministic execution flow, show better alignment with the system dynamics and lower latency.

This research emphasizes that in real-time control systems with high requirements for timing accuracy, simpler microcontrollers can often achieve better results than nominally faster devices. Future work should include analysis of the influence of arithmetic libraries, optimization of communication with the host system, and testing at larger ΔT values and with more complex control models.

REFERENCES

- [1] J. de la Cruz-Alejo, H. Beatriz-Cuellar, I. C. A. Guillermo, and A. M. Ortega, “A decision-making approach on control techniques for an inverted pendulum based on, neuro-fuzzy, indirect adaptive and PID controllers”, *Discover Applied Sciences*, Vol. 6, p. 251, <https://doi.org/10.1007/s42452-024-05921-2>, (2024)
- [2] U. Yıldırım, “Adaptive Control of an Inverted Pendulum by a Reinforcement Learning-based LQR Method”, *Sakarya University Journal of Science*, <https://doi.org/10.48550/arXiv.2310.04436>, (2023)
- [3] E. Shala, X. Bajrami, R. Likaj, and A. Pajaziti, “Real Time Swinging Up and Stabilizing a Double Inverted Pendulum Using PID-LQR”, *Strojnícky časopis - Journal of Mechanical Engineering*, Vol. 73(1), pp. 159–168, <https://doi.org/10.2478/scjme-2023-0013>, (2023)
- [4] M.-S. Phan, T.-C. Do, V.-Q. Truong, and T.-V.-A. Nguyen, “Comparative Analysis of SMC-LMI and LQR Controllers for Double Inverted Pendulum”, *Journal of Measurement, Control and Automation*, Vol. 4(3), pp. 1–7, <https://mca-journal.org/index.php/mca/article/view/184>, (2023)

- [5] Z. Ben Hazem, “Study of Q-learning and deep Q-network learning control for a rotary inverted pendulum system”, *Discover Applied Sciences*, Vol. 6, p. 49, <https://doi.org/10.1007/s42452-024-05690-y>, (2024)
- [6] L. Chen and Y. Zhang, “Linear-quadratic optimal control problems of state delay systems under full and partial information”, *Systems & Control Letters*, Vol. 176, p. 105528, <https://doi.org/10.1016/j.sysconle.2023.105528>, (2023)
- [7] M. Khosravi, H. Azarinfar, and K. Sabzevari, “Design of infinite horizon LQR controller for discrete delay systems in satellite orbit control: a predictive controller and reduction method approach”, *Heliyon*, Vol. 10(2), p. e24265, <https://doi.org/10.1016/j.heliyon.2024.e24265>, (2024)
- [8] M. Costanzo, R. Mazza, and C. Natale, “Digital Control of an Inverted Pendulum Using a Velocity-Controlled Robot”, *Machines*, Vol. 13(6), p. 528, <https://doi.org/10.3390/machines13060528>, (2025)
- [9] Y. Martínez-Armero, S. López-Blandón, and E. Giraldo, “Low-cost Arduino-based Hardware-In-the-Loop Platform for Simulation and Control of Dynamic Systems”, *IAENG International Journal of Computer Science*, Vol. 50(4), pp. 1312–1318, https://www.iaeng.org/IJCS/issues_v50/issue_4/IJCS_50_4_18.pdf, (2023)
- [10] <https://www.mathworks.com/discovery/hardware-in-the-loop-hil.html>
- [11] A. Zhang, X. He, A. Cammi, and X. Wang, “Modelica and Arduino-based hardware-in-the-loop simulation of a nuclear-powered engineering ship”, *Nuclear Engineering and Design*, Vol. 429, p. 113650, <https://doi.org/10.1016/j.nucengdes.2024.113650>, (2024)
- [12] Y.-H. Chang, F.-C. Wu, and H.-W. Lin, “Design and Implementation of ESP32-Based Edge Computing for Object Detection”, *Sensors*, Vol. 25(6), p. 1656, <https://doi.org/10.3390/s25061656>, (2025)
- [13] A. Elnwegy, F. Youssef and T. A. Kern, “Study to compare different transmission procedures in the context of a robotic telemanipulation system”, *Measurement: Sensors*, Vol. 38, p. 101847, <https://doi.org/10.1016/j.measen.2025.101847>, (2025)