Available online at www.sciencedirect.com

ScienceDirect

IFAC PapersOnLine 56-2 (2023) 9600–9605

AeroShield: An Open-Source Propeller-Driven Pendulum Device for Control Engineering Education [★]

Anna Vargová [*,**] Ján Boldocký [*,**] Martin Gulan [*,**] Peter Tibenský [*] Erik Mikuláš [**] Gergely Takács [**]

[*] *Faculty of Mechanical Engineering, Slovak University of Technology in Bratislava, Námestie slobody 17, 812 31 Bratislava, Slovakia (e-mail: martin.gulan@stuba.sk)*
[**] *AutomationShield.com Open-Source Initiative (e-mail: automationshield@automationshield.com)*

Abstract: This paper introduces an open-source reference design of a miniature propeller-driven pendulum device for control engineering education. The presented design of the aeropendulum is built as a swappable and compact extension "shield" for Arduino embedded microcontroller development boards, utilizing off-the-shelf electronic components and 3D-printed mechanical parts. This fact, along with the full hardware documentation provided online, makes the low-cost replication of such hardware simple and available for public. The open-source software includes an application programming interface for Arduino IDE, MATLAB, and Simulink. In addition, typical classroom examples to demonstrate the use and the limitations of the discussed device are presented as well.

Keywords: aeropendulum, control education, educational aids, embedded systems, Arduino, MATLAB, Simulink, microprocessor control, microcontroller

1. INTRODUCTION

The increasing demand for practical experience of future engineers combined with the changing landscape of both industry and academia has brought upon the necessity to adjust the educational approach with a strong focus on practical application. These adjustments manifest mainly in a different approach regarding the content of classroom lessons and assignments. In addition to teaching the theoretical foundations of mathematics, electrical engineering, and cybernetics, the new curricula should support analytical thinking, creativity, and problem-solving. Usage of the hardware plays an essential role within this approach. However, limited access to laboratories, especially in recent years marked by the COVID-19 pandemic, also showed the necessity of using available alternatives to traditional, well-equipped school laboratories. As possible alternatives to simulation environments, online or remote laboratories, and, last but not least, take-home experiments are being practiced and sought after more and more often.

With this paper we aim to contribute to this effort with a miniaturized take-home experimental device that contains a pendulum with a motor-driven propeller attached to its free end—also referred to as an "aeropendulum". This system then enables one to control the angular displacement of the pendulum in closed loop by varying motor power.

To the best of authors' knowledge, only several published studies deal with the construction, modelling, control design, or even experimenting with such a device. Among the most cited papers are the works of Job and Jose (2015) and Enikov and Campa (2012). While the former focused mainly on modelling and simulation of an aeropendulum, authors of the latter presented an original physical model controlled using a custom PCB and programmed in the Simulink environment. Other authors such as Farmanbordar et al. (2011); Breganon et al. (2021); Silva et al. (2020); Marashian (2021) again focused mainly on modelling and control of the aeropendulum. Since the propeller-driven pendulum is a nonlinear system, the work of Habib et al. (2017) described nonlinear model-based parameter estimation and stability analysis of the aeropendulum. Alternatively, fuzzy logic based control approaches were proposed by Farooq et al. (2015) and Silva et al. (2021). In addition, an interesting example of a home-made aeropendulum was recently proposed by Ovalle M. and Cómbita A. (2021), however as it tends to be the case—as a one-off prototype without a detailed documentation and a software interface.

In this paper we propose a novel low-cost reference design for an open-source aeropendulum device built for Arduino prototyping boards. The hardware—that we shall refer to as the "AeroShield" in the upcoming discussion—is part of our wider initiative to create affordable control engineering and mechatronics trainers, called "AutomationShield"; see Takács et al. (2023); AutomationShield (2023).

[★] The authors gratefully acknowledge the contribution of the Slovak Research and Development Agency (APVV) under the grant APVV-18-0023, the Cultural and Educational Grant Agency (KEGA) of the Ministry of Education of Slovak Republic under the grant 012STU-4/2021, and the European Union's Horizon Europe under the grant no. 101079342 (Fostering opportunities towards Slovak excellence in advanced control for smart industries).

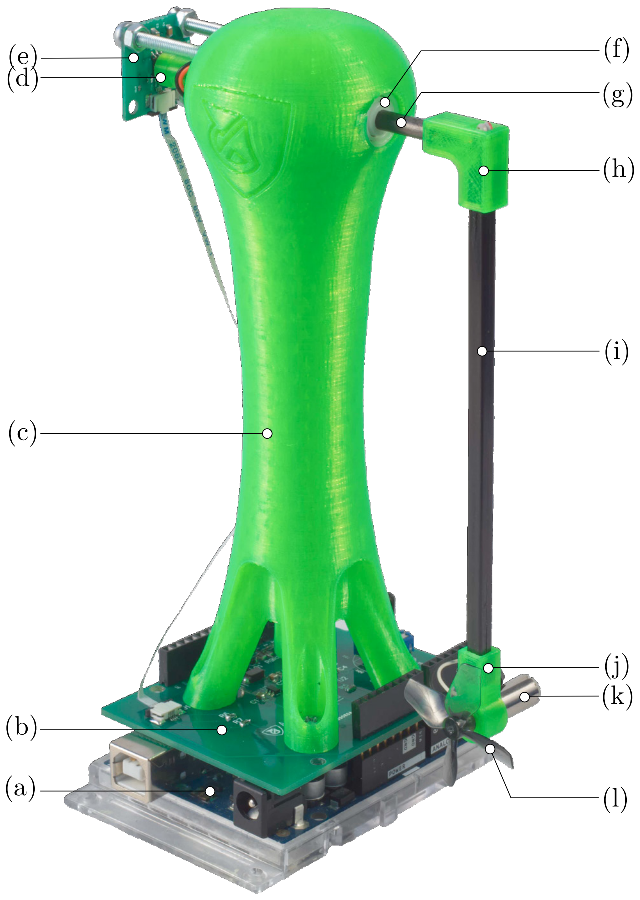


Fig. 1. Photograph of the AeroShield prototype.

2. HARDWARE

This section presents the reference design of the AeroShield device. The main aim is to provide as much information as possible, so it can be easily and robustly replicated or even improved upon by others. Both mechanical and electronic components are marked alphabetically in the text and the reader may follow along the commentary by inspecting Figs. 1–3 and Tab. 1 containing the same type of markings.

The AeroShield shown in Fig. 1 is based on an Arduino development board (a) with a standard R3 pinout, such as Arduino Uno or Arduino Mega2560 with 5 V logic levels, or any other boards with a compatible pin layout at 5 V or 3.3 V logic levels. The device itself is then built on top of a custom PCB (b, Fig. 3) with the rest of the electronic and mechanical components attached to it. The pendulum consists of a vertical (i) and a horizontal (g) carbon rod, connected with a 3D-printed elbow (h). The small coreless DC motor (k), commonly used for miniature drones, with a propeller (l), is mounted to the end of a pendulum through a 3D-printed motor connector (j). The rotating part of the pendulum is seated between two plastic bearings (f) with a magnet attached to it in a 3D-printed holder (d). As the sensor, a Hall-effect rotational encoder is located behind the ball bearing in the axis of its rotational movement on a custom breakout PCB (e, Fig. 3c–d). The whole assembly is connected to the 3D-printed main body (c) mounted on the PCB. The 3D printed parts were created using a Prusa i3 MK3, and are available for download as print-ready .stl files. The sensor and the motor are wired to the main PCB.

Following the electronic schematics in Fig. 2, the voltage for the motor M1 (k) is supplied from an external source through the barrel jack connector on the Arduino board, while the power supplied to motor is controlled by the low-side MOSFET transistor Q1 which receives a PWM signal from Arduino pin D5 through the 1000 Ω current limiting resistor. Floating states are handled by a 10 kΩ pull-down resistor R9, while a diode D1 ensures back electromotive-force (EMF) protection. Operating voltage of the motor is 3.7 V, hence a buck-down converter U1 steps down the voltage using resistors R1, R2 and R3, capacitors C1 and C2, and coil L1 layout. The microchip U2 is responsible for measurement of the current drawn by the motor, by comparing the voltage drop across the shunt resistor R4. There is also a 10 kΩ pull-down resistor R5 on the output of the current monitor, and the VIN+ and V+ pins feature filter capacitors C3 and C4. The magnetic encoder is integrated into the breakout PCB with two capacitors, and connected to the I2C bus connectors of the Arduino (SDA, SCL). Finally, there is a potentiometer POT1 with a small plastic shaft that may be used to manually set the desired reference angle signal.

The electronic schematics and the PCB were designed in the free version of DipTrace. All files are available online, along with the manufacturing-ready Gerber format PCB. From Fig. 3 it is clear that the placement of electrical parts and the use of larger footprints on the PCB enables even less experienced makers to replicate the device.

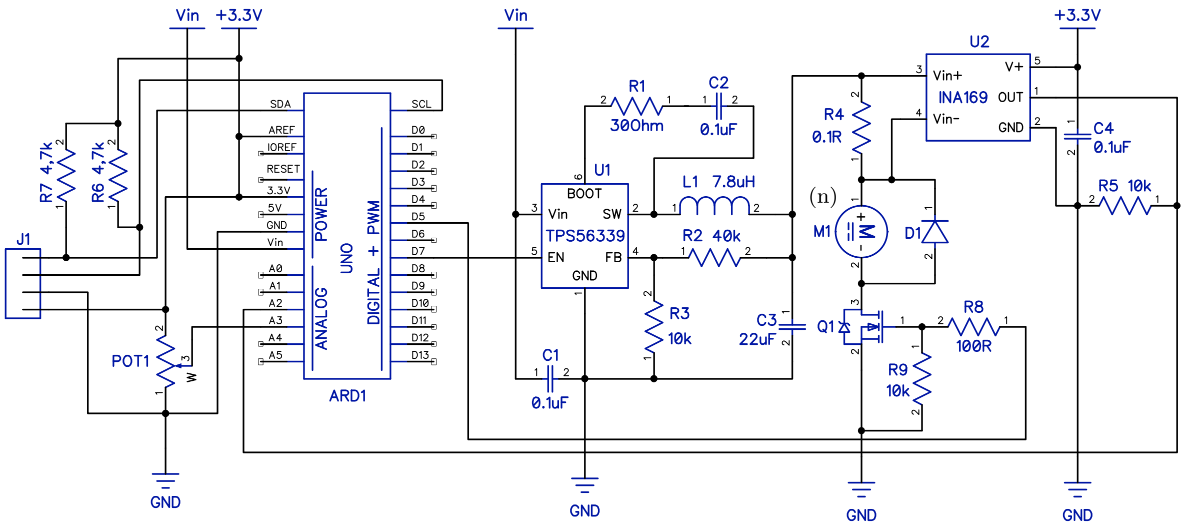
The total cost of all the components (see Tab. 1) and thus of the entire AeroShield comes under €23, excluding the Arduino board (€20 for original, €5 for third party), an external power adaptor (∼€1–5), soldering material, labor and postage, which makes it affordable to laboratories with limited funding and even individual students.

3. SOFTWARE

As introduced, AeroShield is the latest addition to the wide range of devices contained in the AutomationShield open-source hardware and software initiative that comes with basic software support. As the aim of the initiative is to create small and affordable educational aids for control and mechatronics education, the software support focuses on making the usage of each device as easy as possible, so that the students can focus on achieving the comprehension of more complex concepts, instead of learning operating the device. The goal of the “AutomationShield” application programming interfaces (APIs) is to simplify working with various hardware trainers across different platforms while retaining as much consistency as possible. Currently APIs are available for Arduino IDE, MATLAB and Simulink.

3.1 *Arduino IDE API*

The AutomationShield library for Arduino IDE can be easily installed using the graphical user interface (GUI). This library includes header files for all available devices from the AutomationShield family. The `AeroShield.h` header file contains the forward declaration of the `AeroClass` for hardware interface methods. An `AeroShield` object is initialized from within this header. After including the header, the system can be simply started by calling the


(a) Electrical schematics of the main PCB.

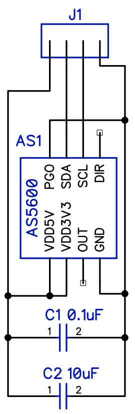
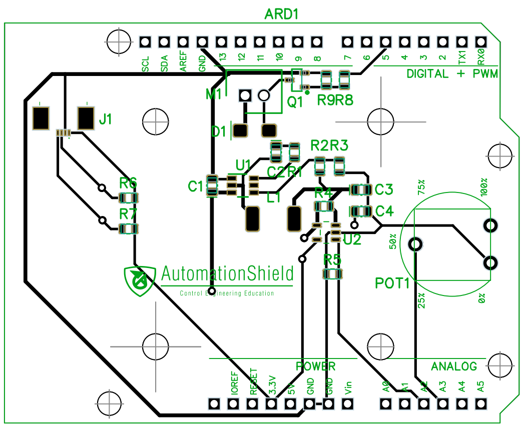
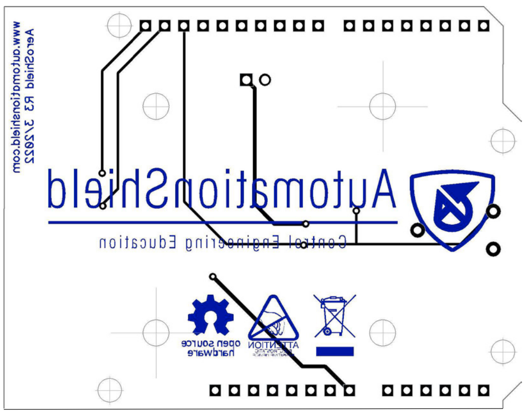
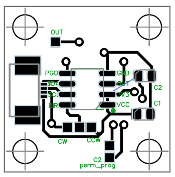

(b) Electrical schematics of the sensor breakout.

Fig. 2. Electrical schematics of the AeroShield with the sensor breakout.


(a) Top layer.


(b) Bottom layer.


(c) Top layer.

(d) Bottom layer.

Fig. 3. Main (a,b) and sensor breakout (c,d) printed circuit boards in true size showing traces and pads (black), drilled holes (gray), and the silkscreen layers (green, blue).

```
AeroShield.begin();
```

method launching the I2C interface and starting the magnetic encoder itself. This method also provides information about the sensor's functionality by detecting the magnet and printing to the serial port whether the magnet can be sensed or not. After initializing the shield, the calibration procedure is performed by calling the

```
AeroShield.calibrate();
```

method, which reads the pendulum's angle and stores its value that will denote a zero for the relative angle measurement. Note that to ensure an undisturbed calibration process, it is necessary that the pendulum arm is steady.

The angular displacement of the pendulum is read by the

```
y=AeroShield.sensorRead();
```

method returning the angle reading in percentage, where 0% corresponds to the value stored within the calibration method and 100% is determined by the input argument (in degrees). If the input argument is not specified, the default value of 90° is used. Methods for angle measurement in degrees and radians are also available within the API.

The current drawn by the motor is read by the

```
y=AeroShield.readCurrent();
```

method, which provides an averaged current reading in the form of a floating point number.

The speed of the motor can be controlled by supplying the input `u` in the range of 0–100% to the

```
AeroShield.actuatorWrite(u);
```

method. The input `u` represents the duty cycle of a PWM actuating signal which is mapped to an 8-bit integer and sent to the D5 PWM capable pin of the Arduino. To avoid any ambiguities and possible overflow, the method checks that the user input does not exceed the allowed range.

Finally, a reference angle set by the user is acquired from the potentiometer by calling

```
r=AeroShield.referenceRead();
```

Table 1. Component list of the AeroShield, without an Arduino board[†].

Symbol	**Part**	**Description**	**Qty.**	**UP**	**Price (€)**
(b),(e)	PCB (shield + breakout)	FR4, 2 layer, 1.6 mm thick	2	0.35	0.7
(k),M1	Motor	High-speed DC coreless motor 7 mm×16 mm, 3.7 V, 5000 min^{-1}, 1 mm shaft	1	2.1	2.1
(l)	Propeller	4-blade propeller with ∅30 mm, bundled with the motor	1	-	-
AS1	Magnetic encoder	AS5600, 12-bit on-axis magnetic rotary position sensor	1	3.03	3.03
(d)	Disc magnet	∅6 mm, 2.5 mm thick	1	0.381	0.381
(f)	Plastic ball bearings	IGUS BB-694-B180-30-ES	2	2.75	5.5
(g)	Carbon rod	70 mm long, circular cross-section ∅4 mm×∅3 mm, 1 m	0.07	2.12	0.15
(i)	Carbon rod	100 mm, square cross-section ,4 mm×3 mm, 1 m	0.1	5.81	0.58
(c),(d),(h),(j)	3D printed parts	3D printed, 49 g of PETG filament, print time of 5:37 hour	4	-	1.73
U1	Buck converter	TPS56339, 3 A output synchronous buck converter	1	1.27	1.27
U2	Current sensor	INA169, high-side, unipolar, current shunt monitor	1	2.66	2.66
POT1	Potentiometer	10 kΩ ca14	1	0.25	0.25
	Turning knob	5 mm×18.7 mm, e.g. ACP 14187-NE	1	0.08	0.08
Q1	MOSFET	PMW45EN2, 30 V, N-channel trench MOSFET, SOT23	1	0.35	0.35
R1	Resistor	30 Ω, 5%, SMD, 0805	1	0.015	0.015
R2	Resistor	40 kΩ, 1% SMD, 0805	1	0.068	0.068
R3,R5,R9	Resistor	10 kΩ, 1% SMD, 0805	3	0.02	0.06
R4	Resistor	100 mΩ, 1%, shunt resistor, SMD, 0805	1	0.326	0.326
R6,R7	Resistor	4.7 kΩ, 5%, SMD, 0805	2	0.01	0.02
R8	Resistor	100 Ω, 5%, SMD, 0805	1	0.02	0.02
C1,C2,C4,C1B	Capacitor	0.1 µF, SMD, 0805	4	0.021	0.084
C3	Capacitor	22 µF,50V, SMD, 0805	1	0.15	0.15
C2B	Capacitor	10 µF, SMD, 0805	1	0.021	0.021
D1	Diode	1N4001 DO-214, 50 V, 1 A	1	0.038	0.038
J1	Connector (sensor)	FFC, 0.5 mm, 4-pin	2	0.18	0.36
	FFC cable (sensor)	4-pin, min 15 cm	1	0.52	0.52
L1	Inductor	10 µH, SPM6530T-100M-HZ	1	0.63	0.63
	Connector (motor)	2×1 pin, 0.1" pitch	1	0.17	0.17
	Jumper cable (motor)	min. 35 cm	1	0.3	0.3
	Header (motor)	2×1pin, 0.1" pitch	1	0.48	0.48
	Screws	DIN 912, 4 pcs. M3×40, 4 pcs. M4×15	8	0.09	0.72
	Nuts	DIN 934, M4	4	0.012	0.048
[†] For low quantity orders, excluding labor and postage.				**Total:**	**€22.81**[†]

which returns the position of the potentiometer wiper as a floating point scaled to 0–100%.

The rest of the AutomationShield library for the Arduino IDE also provides numerous universal functions, such as a comprehensive interruptbased sampling framework, a PID routine, signal processing, Kalman filtering, etc.

3.2 MATLAB API

To prevent confusion in different programming interfaces, the naming convention of methods is also kept consistent for MATLAB. The library is initialized once by running an installation script. Then, assuming that the MATLAB Hardware Support Package for Arduino is installed, the user first creates an instance from the `AeroShield` class by calling the `AeroShield=AeroShield` statement.

The AeroShield device is then initialized by calling

```
AeroShield.begin('COM3', 'UNO');
```

which will simply load a server code to the microcontroller, unless it is not already present. This means that the closed-loop control by the API in MATLAB is not strictly real-time, since commands are communicated through the serial link between the board and the host computer and may be affected by transfer speed or operating system behavior. However, being able to use the high-level MATLAB script allows the user to run live experiments under this already familiar software platform and, most importantly, to create and test more advanced (e.g. nonlinear, predictive, robust, etc.) control frameworks with minimal effort.

Operation of the MATLAB API is otherwise completely identical to the Arduino version, mentioned before. Therefore, methods such as `calibrate()`, `actuatorWrite()` and `referenceRead()` are all implemented for the AeroShield in the MATLAB library as well.

3.3 Simulink API

An even more intuitive tool for creating control loops and performing real-time experiments is the Simulink API for the AeroShield. It relies on the Simulink Support Package for Arduino Hardware which supplies algorithmic units in blocks that access the core hardware functionality.

All necessary functions of the AeroShield are built into separate blocks available for use through the custom Simulink library (see Fig. 4) and may be combined with all the other available blocks to create feedback control applications. It retains the naming convention and features for individual input and output blocks (`Reference Read`, `Actuator Write`, `Angle Read`) and a comprehensive block representing the entire device is also available (`AeroShield`).

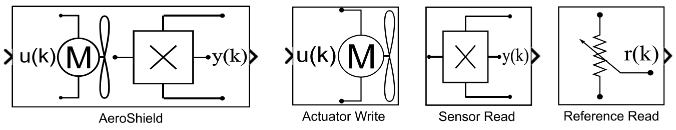


Fig. 4. Algorithmic blocks for the AeroShield in Simulink.

Note that descriptions of the above APIs are, among other documentation, also available on the GitHub wiki page of the AeroShield accessible via AutomationShield (2023).

4. EXAMPLES

This section demonstrates typical fundamental classroom examples for basic or advanced control engineering education that are included with the API. Note that the goal of these examples is to show the capabilities of each API and to help the students understand the basic concepts of system identification and feedback control. Therefore the examples are kept as simple as possible, although neither the software nor the hardware limits the implementation of more complex algorithms.

4.1 System Identification

AeroShield represents a single-input single-output (SISO) system, where the manipulated input is the voltage of the motor, $u(t)$, which translates into the lift force generated by the propeller, $T(t)$, while the controlled variable is the angular deflection of the pendulum, θ, which is measured by the rotary sensor. The mechanical part of the system's dynamics can be described by the following ODE:

$$ml^2\ddot{\theta}(t) = -mgl\sin\theta(t) - b\dot{\theta}(t) + lT(t), \tag{1}$$

where m and l denote the pendulum's mass and length, respectively, g denotes the gravitational acceleration, and b is the coefficient of viscous damping in the rotary bearing joint. The ODE (1) can be easily transformed into a state-space representation:

$$\dot{x}_1(t) = x_2(t), \tag{2a}$$

$$\dot{x}_2(t) = -\frac{g}{l}\sin x_1(t) - \frac{b}{ml^2}x_2(t) + \frac{K}{ml}u(t), \tag{2b}$$

with $x_1(t) = \theta(t)$ and $x_2(t) = \dot{\theta}(t)$. Note that we assumed a linear approximation of the correlation between the motor voltage $u(t)$ and the lift force $T(t)$ via a thrust coefficient K, c.f. Enikov and Campa (2012).

The nonlinear state-space model (2) can be expressed as a MATLAB ODE function, which can be then used within a grey-box identification procedure. One may also choose to linearize it at a desired angle to derive a linear state-space model which is omitted here for brevity.

The worked example `AeroShield_Ident_Greybox.m` takes the experimental data and searches for the unknown parameters of the first-principle model (2). First, an initial guess of the system model is created based on assumed parameter values and model structure. The input-output data for system identification were obtained by applying an amplitude-modulated pseudo-random binary sequence (APRBS) excitation signal. The unknown parameters are then found by a grey-box estimation procedure utilizing an appropriate search method implemented in MATLAB's System Identification Toolbox. The resulting models provide a very good match to the measured data (from 63% to 80% fit) as indicated in Fig. 5. The estimated parameters and their units are listed in Tab. 2.

Note that in terms of modelling and system identification, it is also possible to take into account the current draw of the motor. Although the hardware enables it, we did not exploit the current measurement in the model itself in order to keep it SISO and the identification procedure as simple as possible. Needless to say, we are also open to any contribution in terms of a more accurate model and more complex system identification examples.

Table 2. Parameter values for the AeroShield's first-principle grey-box model (fixed – known, free – estimated).

Parameter	Value
Mass of the motor (m)	5 g (fixed)
Gravitational acceleration (g)	$9.81\,\mathrm{m\,s^{-2}}$ (fixed)
Damping coefficient (b)	$4.26 \times 10^{-4}\,\mathrm{N\,m\,s^{-1}}$ (free)
Length of the pendulum (l)	153.5 mm (free)
Gain of the fan (K)	$0.0293\,\mathrm{N\,V^{-1}}$ (free)

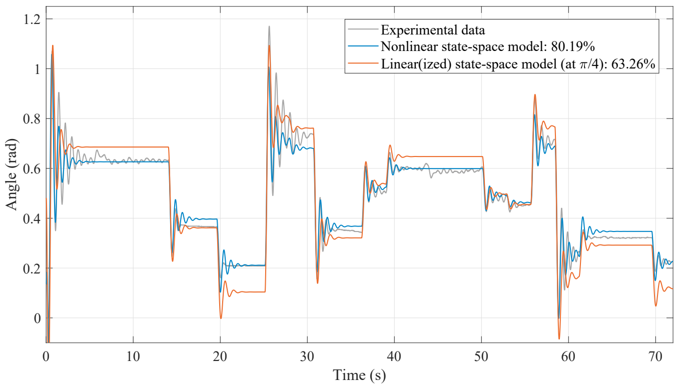


Fig. 5. Simulated response comparison of grey-box models – nonlinear state-space model (blue) and linear state-space model (orange) – with experimental data (grey).

4.2 Feedback Control

Experiments with feedback control can be designed and executed easily in the Arduino IDE, MATLAB, or Simulink environment with the help of the provided APIs. With the AeroShield, one must however pay attention to the tuning and closed-loop stability of designed controllers, as spinning of the pendulum is undesirable and may result in irreversible cable damage. A tenable angular displacement of the pendulum ranges from 0° to ∼120°.

The following sample examples were implemented in both Arduino IDE and Simulink API. The implementation in C/C++ uses the interrupt-driven sampling subsystem of the AutomationShield library. The same experiment can be conveniently launched from MATLAB and Simulink as well. The MATLAB API shall be used mainly for didactic and simulation purposes, but we do not recommend using it for research, since the computing of the control action is performed by the PC, hence the real-time execution of the instructions cannot be guaranteed. On the other hand, Simulink offers straightforward building of control loops using preprogrammed Simulink blocks. Using the Arduino IDE or Simulink API, the progress of the experiments can be followed in real-time through the Serial Plotter or a Scope block, respectively, or can be logged in MATLAB.

Let us start with PID control, since it most certainly forms an important part of any fundamental control course. The discrete PID controller assumed in this demonstration was hand-tuned, sampled at 3 ms and features hard input saturation and basic integral-windup handling by clamping. Figure 6 shows the result of its real-time implementation in Arduino IDE with the help of API, plotted in MATLAB. The aeropendulum tracks the desired angle reference, set manually using the potentiometer, swiftly and accurately.

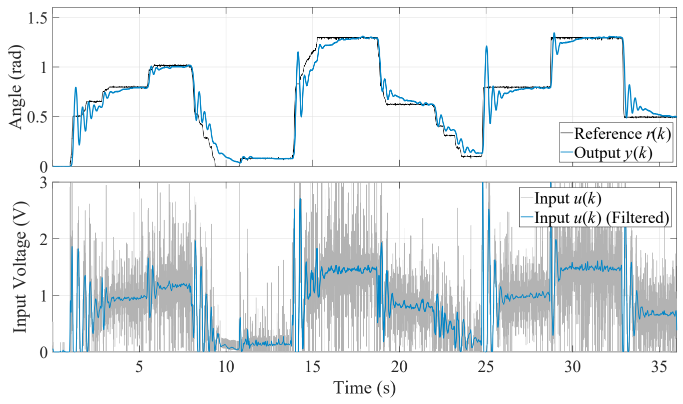


Fig. 6. Reference tracking by PID, designed and executed in Arduino IDE, with the reference manually.

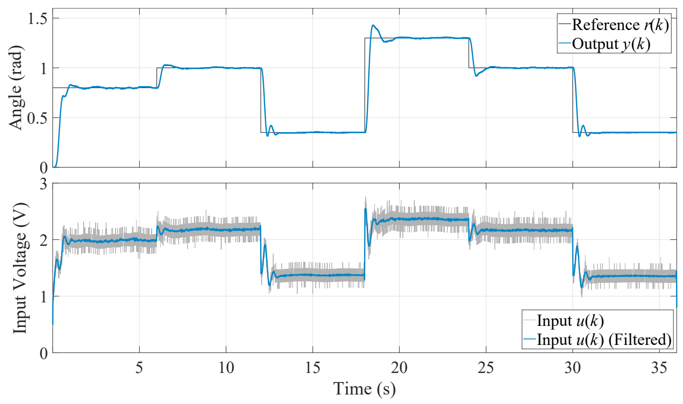


Fig. 7. Reference tracking by LQR, designed and executed in Simulink.

The example uses the PID framework from AutomationShield to aid the learning progress of beginners even more.

Another example demonstrates the state-feedback control, in particular a linear-quadratic regulator (LQR). To design it, we used the state-space model obtained by linearizing (2) around the angle of 45° (≈0.79 rad), which was identified and validated in Fig. 5. In order to eliminate the steady-state error when tracking the reference, we augmented the discretized state-space model with an integrator state. The controller was implemented using Simulink API, assuming calculation of angular velocity from position measurements via a finite difference method. The resulting control performance is beyond sufficient. As shown in Fig. 7, the linear controller copes well with tracking reference output signal ranging from 20° (≈0.35 rad) to 75° (≈1.31 rad), i.e. even far from the linearization point of the model itself.

5. CONCLUSION

We have presented an open-source reference design for a propeller-driven pendulum experiment based on Arduino boards as an extension shield, which can be manufactured under €23. The application programming interface for the Arduino IDE, MATLAB, and Simulink retains compatible functional and naming conventions, enabling students to conveniently switch between programming environments. The low price, simplicity, and openness of the AeroShield renders it a viable mechatronic trainer, enabling students to create and run feedback control experiments outside the laboratory, even with an initial lack of practical experience.

Although the introduced device is fully functional and can be directly used in control education or research, it is still under development. We are continuously working on further improvements both of hardware and software, such as incorporating the current measurement into examples, or searching for other possible solutions for sensors and actuators, while keeping in mind the low-cost and compact nature of the device. Currently, we are also integrating all the shields into the curricula of selected courses, and collecting and analyzing feedback of the students in order to identify areas for potential improvement.

REFERENCES

AutomationShield (2023). AutomationShield. GitHub Wiki page. www.AutomationShield.com.

Breganon, R., Alves, U., Almeida, J., Fonteque Ribeiro, F., Mendonça, M., Palácios, R., and Montezuma, M. (2021). Loop-shaping $\mathcal{H}_\infty$ control of an aeropendulum model. *International Journal of Applied Mechanics and Engineering*, 26, 1–16.

Enikov, E. and Campa, G. (2012). Mechatronic aeropendulum: Demonstration of linear and nonlinear feedback control principles with MATLAB/Simulink Real-Time Windows Target. *IEEE Transactions on Education*, 55, 538–545.

Farmanbordar, A., Zaeri, N., and Rahimi, S. (2011). Stabilizing a driven pendulum using DLQR control. In *5th Asia Modelling Symposium*, 123–126.

Farooq, U., Gu, J., El-Hawary, M., Luo, J., and Asad, M.U. (2015). Observer based fuzzy LMI regulator for stabilization and tracking control of an aeropendulum. *28th Canadian Conference on Electrical and Computer Engineering*, 1508–1513.

Habib, G., Miklós, A., Enikov, E., Stépán, G., and Rega, G. (2017). Nonlinear model-based parameter estimation and stability analysis of an aero-pendulum subject to digital delayed control. *International Journal of Dynamics and Control*, 5, 629–643.

Job, M. and Jose, P. (2015). Modeling and control of mechatronic aeropendulum. In *2015 International Conference on Innovations in Information, Embedded and Communication Systems*, 1–5.

Marashian, A. (2021). Modeling and control of mechatronic aeropendulum. Online.

Ovalle M., D.M. and Cómbita A., L.F. (2021). On the implementation of low-cost home-made aeropendulum prototypes for improving understanding of dynamical systems concepts. In *5th Colombian Conference on Automatic Control*, 297–302.

Silva, H., Ramos, I., Cardim, R., Assunção, E., and Teixeira, M. (2020). Identification and switched control of an aeropendulum system. Online.

Silva, H.R.M., Cardim, R., Teixeira, M.C.M., Assunção, E., and Ramos, I.T.M. (2021). Switched control and tracking application in aeropendulum system using fuzzy models. In *2021 International Conference on Fuzzy Systems*, 1–6.

Takács, G., Mikuláš, E., Gulan, M., Vargová, A., and Boldocký, J. (2023). Automationshield: An open-source hardware and software initiative for control engineering education. In *22nd IFAC World Congress*, To appear.