Jurnal Inovasi Teknologi dan Rekayasa
Vol. 10, No. 1, January-June 2025, pp. 191-201 ISSN: 2581-1274

# Design and Development of a Low-Cost Programmable Air Regulator for Soft Pneumatic Actuators

Syahirul Alim Ritonga[a,1,*], Sarah Iftin Atsani [b,2], Mohammad Ardyansah [a,3], Raditya Fadhil Arva [a,4], Herianto [a,5], Adrian Axel Yohannes [a,6].

[a] *Departement of Mechanical and Industrial Engineering, Universitas Gadjah Mada Jl. Grafika 2, Yogyakarta 55281, Indonesia*
[b] *Center of Additive Manufacturing and Systems, Universitas Gadjah Mada Jl. Grafika 2, Yogyakarta 55281, Indonesia*
[1] *syahirul.alim.r@mail.ugm.ac.id*; [2]sarah.iftin.a@mail.ugm.ac.id; [3]mohammad.ardyansah@mail.ugm.ac.id; [4]raditya.fad2003@mail.ugm.ac.id ; [5]herianto@ugm.ac.id; [6]adrian.axel.yohannes@mail.ugm.ac.id.*
** corresponding author*

ARTICLE INFO

*Article history:*
Published
7 June 2025

*Keywords:*
Soft pneumatic actuator
Pressure regulator
Control
Soft Robotic

A B S T R A C T

Soft Pneumatic Actuators (SPAs) are widely utilized due to their affordability, simplicity, and ability to produce diverse movements such as bending and twisting by regulating air pressure. However, precise air pressure control remains a significant challenge in achieving desired actuator motions and maintaining performance consistency. While commercial programmable air regulators exist, their high cost often limits accessibility for budget-conscious research and educational institutions. This study addresses this challenge by designing and developing a low-cost programmable air regulator tailored for SPAs. Utilizing widely available and inexpensive components, the system aims to provide precise and reliable pressure control, democratizing SPA technology for researchers, small industries, and educational institutions. The regulator employs a stepper motor with a 1.8-degree step resolution and a 1:4 gear ratio, offering fine granularity in pressure adjustments. Carefully selected speed (1000 steps/s) and acceleration (500 steps/s) parameters ensure consistent operation under repetitive use. The program code of Arduino UNO microcontroller developed in this study can be readily adopted by researchers and practitioners. The regulator consistently achieves desired pressures through precise control of motor revolutions, proving to be a viable, cost-effective alternative to expensive commercial options.

## I. Introduction

Soft actuators have gained significant attention in the field of robotics due to their inherent flexibility, adaptability, and ability to safely interact with delicate objects [1]. These actuators are particularly valuable in applications where traditional rigid mechanisms fall short, such as medical devices, industrial automation, and human-robot interactions [2]. The soft materials used in their construction allow for a gentle and safe approach, which is ideal for handling delicate objects or interacting with human tissue, making them well-suited for healthcare and precision engineering [3].

The mechanism of soft actuators is inherently different from traditional rigid actuators. Rather than relying on motors or gears (used in conventional actuators) [4], soft actuators bend, stretch, and contract by utilizing external stimuli to create movement [5]. Various types of stimuli can be used to trigger their motion, including electric fields, magnetic forces, thermal expansion, light, pressure, and even explosions [6]. Among these actuation, pneumatic pressure has become one of the most widely used [7], with actuators driven by this mechanism commonly referred to as soft pneumatic actuators (SPA).

SPA operates by inflating and deflating the internal chambers [8], enabling them to perform a wide range of motions, from linear movements to a more complex movement such as bending and twisting [9]. This type of actuator offers numerous advantages, including ease of deployment due to its simplicity in design and integration [10]. SPA is also highly cost-effective, as it is made from inexpensive materials, use low-cost components, and affordable production costs [6], [11]. Its lightweight construction enhances the mobility and efficiency of robotic systems [12], while its inherent safety minimizes the risk of harm when interacting with fragile objects or human users [13].

Pneumatic system is affordable, readily available, and simple to maintain and operate [14]. It also offers a high-power output and naturally provide passive compliance by regulating the amount of pressurized air within their chambers [14], making it ideal for soft actuator. However, while provides significant advantages, precisely controlling the air pressure remains a key challenge in achieving desired movements and maintaining performance consistency in SPA. A programmable air regulator plays a critical role in controlling the pressure delivered to the actuators, ensuring the desired performance in terms of force and motion. While there are high-performance commercial air regulators available, their cost can be prohibitive for research environments and industries with limited budgets. Therefore, the development of a low-cost, yet efficient and programmable air regulator for SPA is crucial to advance the accessibility of this technology.

This study focuses on the design and development of a low-cost programmable air regulator tailored for SPA. The aim is to create a system that is both affordable and capable of precise control over air pressure, enabling accurate manipulation of soft actuators for various applications. By utilizing widely available and inexpensive components, this regulator seeks to democratize the use of SPAs, making them more accessible to researchers, small industries, and educational institutions.

In this paper, the design considerations, component selection, and implementation strategies for building the regulator will be discussed, followed by an evaluation of its performance in controlling a soft pneumatic actuator. The results demonstrate that the proposed low-cost solution can provide reliable pressure control, offering a viable alternative to more expensive commercial options.

## II. Method

The tools and materials utilized in this study are detailed in Table 1, while the experimental setup and control scheme are depicted in Figure 1. The SPA requires air pressure to inflate, and its deflation is controlled by regulating the air pressure. Air is supplied from a compressor, which generates a consistent source of pressurized air. This pressure is then regulated using a mechanical pressure regulator equipped with a rotary pulley that functions as a control mechanism, similar to a knob.

Table 1. Tools and materials

| No | Items |
|---|---|
| 1 | Tekiro Air Control Unit AT-AC1704 |
| 2 | Solenoid AIRTAC 4V310-10 Voltage AC 220 / DC 24V |
| 3 | POWER SUPPLY 24V 2A |
| 4 | Adaptor 12V 8A |
| 5 | Arduino UNO CH340 |
| 6 | 2 Channel Relay Module |
| 7 | Keypad 4x4 Push Button |
| 8 | DRV8825 Stepper Motor Driver |
| 9 | NEMA 17 17HS4401 Stepper Motor |
| 10 | GT2 Timing Pulley 20 Teeth |
| 11 | Timing Belt GT2 |

The rotation of the pulley is controlled by a stepper motor, which is driven by a DRV8825 motor driver. The motor's rotations directly influence the regulator knob's position: the more the stepper motor rotates, the more the pulley turns, resulting in a higher air pressure delivered to the SPA. This precise control allows for fine-tuned adjustments to the inflation and deflation cycles of the SPA.

The solenoid valve, controlled via an Arduino microcontroller and a relay module, directs the airflow to the SPA. The Arduino is connected to a computer for programming and real-time control, enabling automated adjustments of the stepper motor and solenoid valve. This setup ensures a controlled and repeatable experiment, where the air pressure supplied to the SPA is finely regulated, providing consistent inflation and deflation. The system's modular design and programmable components make it versatile and adaptable for further experimentation.

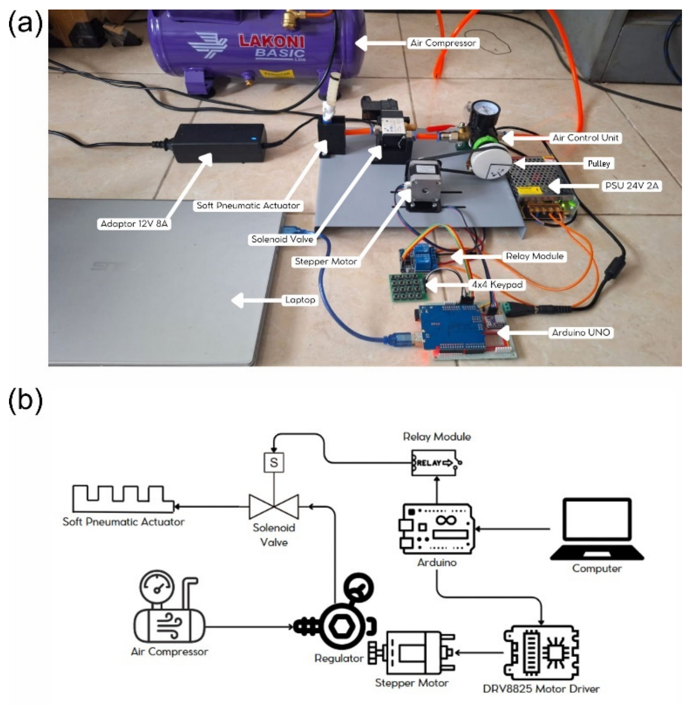


Fig 1. Method of control: (a) Experimental setup; (b) Control scheme.

This research employed an experimental methodology to investigate the effectiveness of two different SPA designs. To measure the curvature of the SPA, an image processing software was utilized. The detail method will be elaborated in this section.

## III. Result and Discussion

### *A. Code overview*

The flowchart presented in Figure 2 outlines the programming workflow utilized to control the rotation of the stepper motor via an Arduino. The control algorithm for this study was written using open-source Arduino Software This section provides a detailed overview of the code's functionality, emphasizing its core operations and their role in achieving precise control of the stepper motor. For comprehensive examination, the complete code is available and can be accessed through the link provided in the Supplementary Materials section.

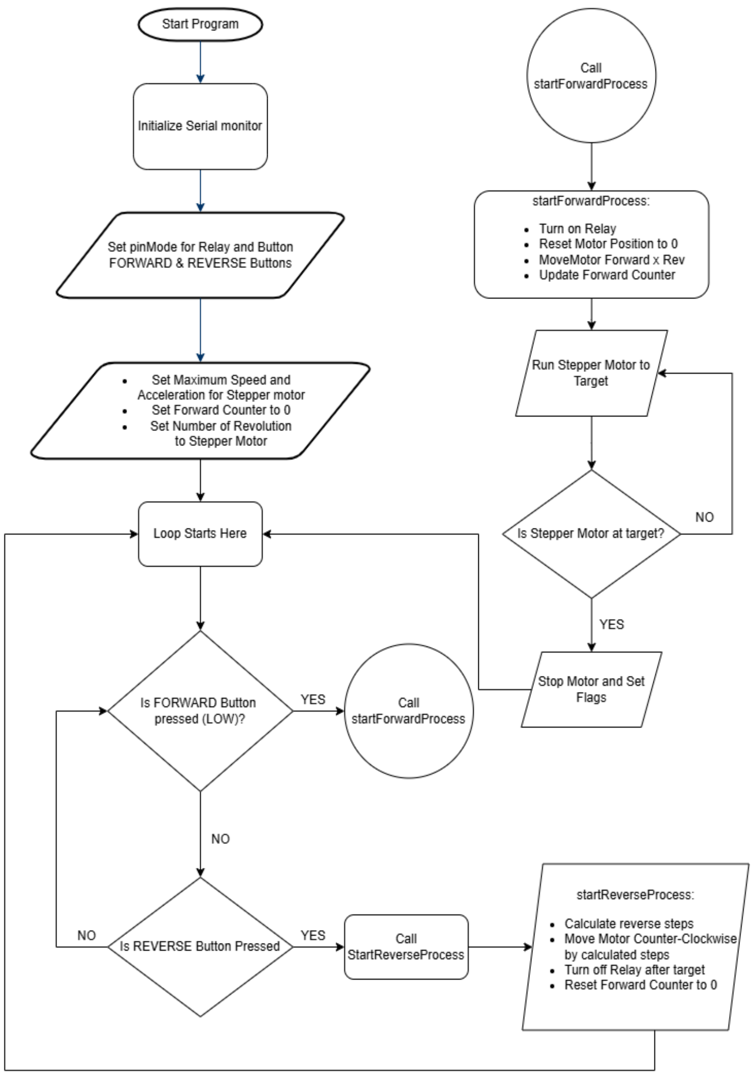
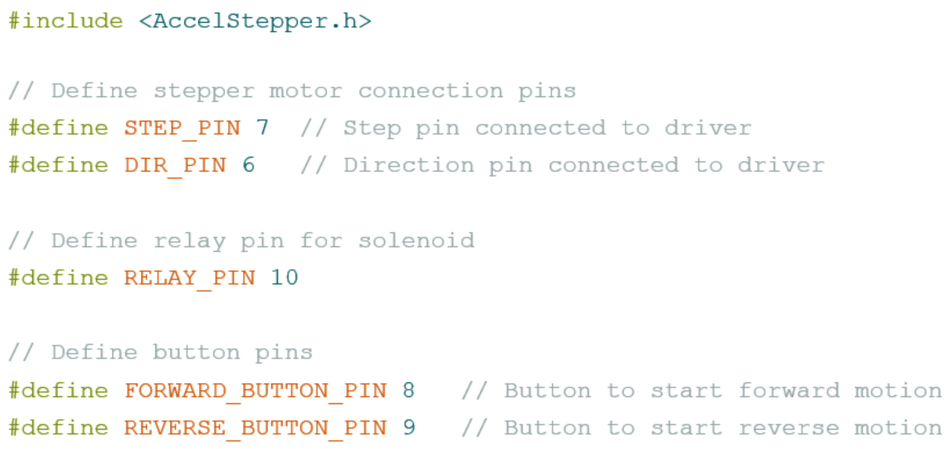

Start Program
Initialize Serial monitor
Set pinMode for Relay and Button FORWARD & REVERSE Buttons
Set Maximum Speed and Acceleration for Stepper motor
Set Forward Counter to 0
Set Number of Revolution to Stepper Motor
Loop Starts Here
Is FORWARD Button pressed (LOW)?
YES
Call startForwardProcess
NO
Is REVERSE Button Pressed
NO
YES
Call StartReverseProcess
startReverseProcess:
Calculate reverse steps
Move Motor Counter-Clockwise by calculated steps
Turn off Relay after target
Reset Forward Counter to 0
Call startForwardProcess
startForwardProcess:
Turn on Relay
Reset Motor Position to 0
MoveMotor Forward x Rev
Update Forward Counter
Run Stepper Motor to Target
Is Stepper Motor at target?
NO
YES
Stop Motor and Set Flags

Fig 2. Flowchart of Arduino programming

The code begins by including the necessary libraries, such as AccelStepper (Figure 3), which is widely used for precise control of stepper motors, including speed and acceleration adjustments. Following this, the code defines the connection pins for the stepper motor, solenoid, and button (Figure 3), with each pin assigned a specific function, as outlined below:

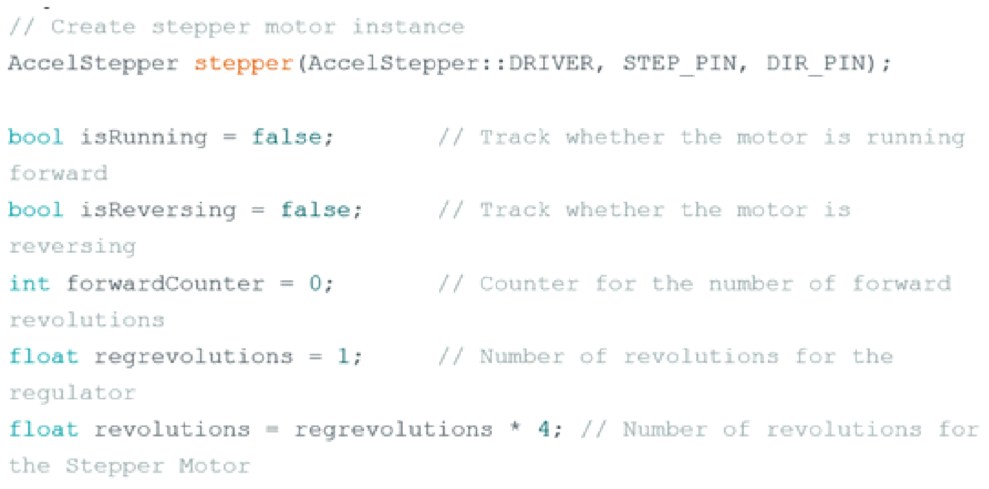
1. STEP_PIN: Provides the step signal to the motor driver.
2. DIR_PIN: Controls the direction of motor rotation, either clockwise or counterclockwise.
3. RELAY_PIN: Activates or deactivates the relay that controls the solenoid valve.
4. FORWARD_BUTTON_PIN: Serves as the input for the button that initiates forward movement.
5. REVERSE_BUTTON_PIN: Serves as the input for the button that initiates reverse movement.



```
#include <AccelStepper.h>

// Define stepper motor connection pins
#define STEP_PIN 7  // Step pin connected to driver
#define DIR_PIN 6   // Direction pin connected to driver

// Define relay pin for solenoid
#define RELAY_PIN 10

// Define button pins
#define FORWARD_BUTTON_PIN 8   // Button to start forward motion
#define REVERSE_BUTTON_PIN 9   // Button to start reverse motion
```

Fig 3. Defining library and connection pins.

The code in Figure 4 initializes an instance of the AccelStepper library to control a stepper motor using a driver, with the motor's step and direction pins specified as STEP_PIN and DIR_PIN. Two Boolean variables, isRunning and isReversing, are used to track whether the motor is moving forward or reversing, respectively. An integer variable, forwardCounter, is employed to count the number of forward revolutions completed by the motor. The variable regrevolutions defines the number of revolutions required for the regulator, and this value is multiplied by a factor of 4 to calculate revolutions, representing the total number of revolutions the stepper motor must complete.



```
// Create stepper motor instance
AccelStepper stepper(AccelStepper::DRIVER, STEP_PIN, DIR_PIN);

bool isRunning = false;        // Track whether the motor is running
forward
bool isReversing = false;      // Track whether the motor is
reversing
int forwardCounter = 0;        // Counter for the number of forward
revolutions
float regrevolutions = 1;      // Number of revolutions for the
regulator
float revolutions = regrevolutions * 4; // Number of revolutions for
the Stepper Motor
```

Fig 4. Global variable

Subsequently, the setup code shown in Figure 5 is implemented to ensure that the motor and relay are properly initialized and prepared for precise operation. The code begins by initializing the serial monitor, configuring the relay pin, and setting up the button pins with appropriate input modes. This is followed by defining the maximum speed and acceleration parameters for the stepper motor, ensuring smooth and controlled movement during operation.

```
void setup() {
  // Initialize serial monitor
  Serial.begin(9600);

  // Initialize relay pin
  pinMode(RELAY_PIN, OUTPUT);
  digitalWrite(RELAY_PIN, HIGH); // Ensure relay is off initially

  // Initialize button pins
  pinMode(FORWARD_BUTTON_PIN, INPUT_PULLUP); // Use internal pull-up
resistor
  pinMode(REVERSE_BUTTON_PIN, INPUT_PULLUP); // Use internal pull-up
resistor

  // Set maximum speed and acceleration for stepper motor
  stepper.setMaxSpeed(1000);  // Maximum speed (steps per second)
  stepper.setAcceleration(500); // Acceleration (steps per second^2)
}
```

Fig 5. Setup code

The Arduino code in Figure 6 defines the main operational loop for controlling the stepper motor and relay system. Its primary function is to handle button inputs for initiating forward or reverse motion, manage the motor's operation based on these inputs, and ensure proper stopping conditions. The code checks if the forward button is pressed while the motor is idle (!isRunning and !isReversing), triggering the startForwardProcess() function to initiate forward movement. Similarly, it checks for the reverse button press to start the reverse process using startReverseProcess(). Once a motion is initiated, the stepper.run() function is called to execute the motor's movement as long as the target distance has not been reached. When the target distance is reached, the corresponding isRunning or isReversing flag is reset to false to stop the motor. Additionally, in the reverse process, the relay is turned off by setting RELAY_PIN to HIGH upon completion. This code ensures precise control and seamless operation of the motor and relay system in response to user inputs.

```
void loop() {
  // Check for forward button press
  if (!isRunning && !isReversing && digitalRead(FORWARD_BUTTON_PIN) ==
LOW) {
    startForwardProcess();
  }

  // Check for reverse button press
  if (!isRunning && !isReversing && digitalRead(REVERSE_BUTTON_PIN) ==
LOW) {
    startReverseProcess();
  }

  // Run motor if it has a target
  if ((isRunning || isReversing) && stepper.distanceToGo() != 0) {
    stepper.run();
  } else if (isRunning && stepper.distanceToGo() == 0) {
    isRunning = false; // Stop the motor after forward motion
  } else if (isReversing && stepper.distanceToGo() == 0) {
    isReversing = false; // Stop the motor after reverse motion
    digitalWrite(RELAY_PIN, HIGH); // Turn off relay
  }
}
```

Fig 6. Loop function

In Figure 7, startForwardProcess() initiates the forward rotation process of the stepper motor while activating the solenoid via the relay. This function is essential for initiating and managing the forward operation of the motor, ensuring precise and controlled rotation while maintaining real-time feedback through the serial monitor. The primary functions of the code in Figure 7 are as follows:

1. Activate the Solenoid: The relay is turned on by setting RELAY_PIN to LOW, which engages the solenoid.

2. Set Motor Parameters: The stepper motor's current position is reset to 0 using setCurrentPosition(0). This establishes a reference point for the motor's movements.

3. Calculate Steps: The total number of steps required for the desired number of revolutions (revolutions) is calculated using the motor's steps-per-revolution specification (stepsPerRevolution).

4. Command the Motor: The motor is instructed to move a specified number of steps (equivalent to the desired revolutions) in the forward (clockwise) direction using moveTo(stepsToMove). The isRunning flag is set to true to indicate that the motor is in motion.

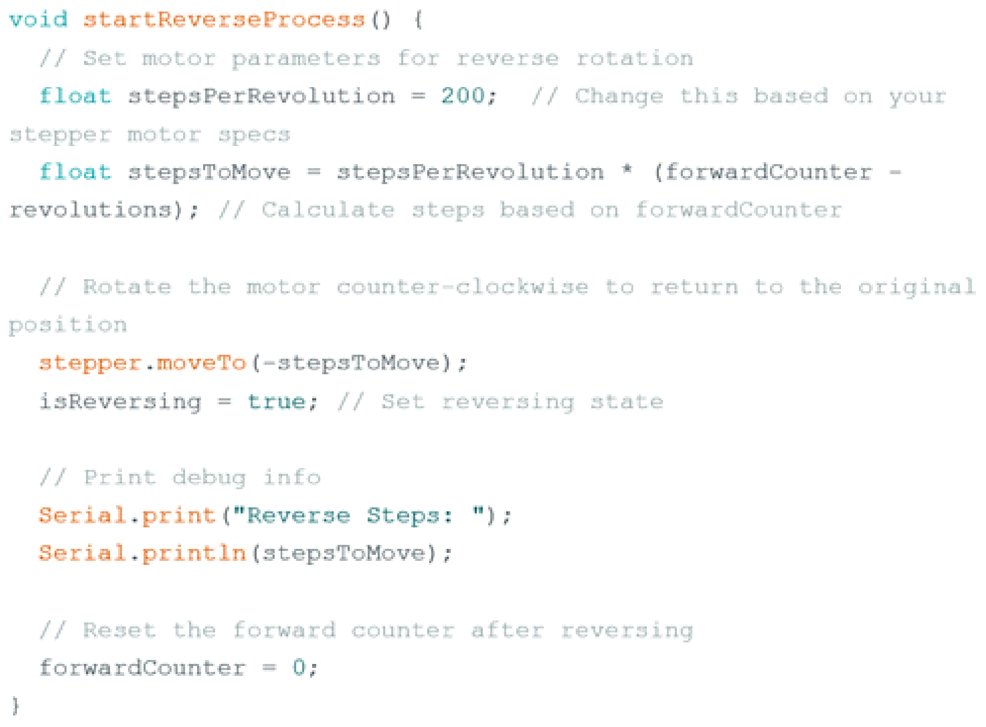
5. Update the Counter: The forwardCounter is incremented by the number of revolutions, keeping track of the total forward movements completed.

6. Debugging Output: The updated forwardCounter value is printed to the serial monitor for debugging and tracking purposes.

```
void startForwardProcess() {
  // Activate solenoid
  digitalWrite(RELAY_PIN, LOW); // Turn on relay (activate solenoid)

  // Set motor parameters for forward rotation
  stepper.setCurrentPosition(0); // Reset position to 0
  float stepsPerRevolution = 200;  // Change this based on your
stepper motor specs

  float stepsToMove = stepsPerRevolution * revolutions;

  // Rotate the motor 4 revolutions clockwise
  stepper.moveTo(stepsToMove);
  isRunning = true; // Set running state

  // Increment the forward counter by the number of revolutions
  forwardCounter += revolutions;

  // Print debug info
  Serial.print("Forward Counter: ");
  Serial.println(forwardCounter);
}
```

Fig 7. Forward rotation function

The startReverseProcess() function in Figure 8 initiates the reverse rotation of the stepper motor to return it to its original position. To achieve this, the function first specifies the motor's steps per revolution (stepsPerRevolution) and calculates the total steps required for the reverse movement. This calculation is based on the accumulated forwardCounter, which tracks the total forward revolutions, minus the predefined number of revolutions (revolutions). The calculated steps are then set as the target position using the moveTo() method, with a negative value to indicate counter-clockwise rotation. The function sets the isReversing flag to true, signalling that the motor is in the reverse motion state. For real-time monitoring and debugging, the calculated number of reverse steps is printed to the serial monitor. Additionally, the forwardCounter is reset to zero after initiating the reverse process, ensuring accurate tracking for subsequent forward operations.



```
void startReverseProcess() {
  // Set motor parameters for reverse rotation
  float stepsPerRevolution = 200;  // Change this based on your
stepper motor specs
  float stepsToMove = stepsPerRevolution * (forwardCounter -
revolutions); // Calculate steps based on forwardCounter

  // Rotate the motor counter-clockwise to return to the original
position
  stepper.moveTo(-stepsToMove);
  isReversing = true; // Set reversing state

  // Print debug info
  Serial.print("Reverse Steps: ");
  Serial.println(stepsToMove);

  // Reset the forward counter after reversing
  forwardCounter = 0;
}
```

Fig 8. Reverse rotation function

## *B. Affordability and Accessibility of Resource*

The tools and materials utilized in this study were intentionally selected to ensure economic feasibility while maintaining the functionality and reliability required for the experimental setup. Table 2 provides a detailed list of the items used, including their quantities and costs. The total expenditure for the materials amounted to Rp 661,500. This modest total reflects the affordability of the components, which were procured from widely accessible sources, including local and online marketplaces. Besides, the total price represents a highly cost-effective approach compared to commercial pressure regulators, which can cost significantly more, typically ranging from Rp 2,000,000 to Rp 5,000,000, depending on the specifications and features.

Commercial units often include integrated controllers and high-precision components, but their elevated prices make them less accessible, particularly for small-scale researchers or educational purposes. In contrast, the system developed in this study uses readily available, low-cost components such as the Arduino UNO CH340 and the DRV8825 Stepper Motor Driver, which together deliver comparable functionality at a fraction of the cost. This not only lowers the barrier to entry for conducting research but also makes the technology more accessible for small-scale industries or educational institutions with limited budgets.

Table 2. Cost of tools and materials.

| **Items** | **Quantity** | **Price** |
|---|---|---|
| Tekiro Air Control Unit AT-AC1704 | 1 | Rp 150,000 |
| Solenoid AIRTAC 4V310-10 Voltage AC 220 / DC 24V | 1 | Rp 125,000 |
| Power Supply 24V 2A | 1 | Rp 58,000 |
| Adaptor 12V 8A | 1 | Rp 115,000 |
| Arduino UNO CH340 | 1 | Rp 60,000 |

| Items | Quantity | Price |
|---|---|---|
| 2 Channel Relay Module | 1 | Rp 15,000 |
| Keypad 4x4 Push Button | 1 | Rp 6,000 |
| DRV8825 Stepper Motor Driver | 1 | Rp 20,000 |
| NEMA 17 17HS4401 Stepper Motor | 1 | Rp 90,000 |
| GT2 Timing Pulley 20 Teeth | 1 | Rp 7,500 |
| Timing Belt GT2 | 1 | Rp 15,000 |
| **Total cost** | | Rp 661,500 |

### *C. Performance*

The performance of the pressure regulator controller was critically dependent on the mechanical and operational parameters of the system, including the stepper motor, gear ratio, and control settings for speed and acceleration. The stepper motor utilized in the system has a resolution of 1.8 degrees per step, translating to 200 steps per revolution. This level of granularity provided a fine degree of control over the system's movement. The 1:4 gear ratio between the motor and the pulley further enhanced this precision by effectively amplifying the motor's resolution. As a result, each step of the motor corresponds to a smaller angular displacement of the pulley, enabling more refined control over the air control unit. This mechanical advantage was critical in achieving the desired accuracy in pressure regulation.

The system's speed was set at 1000 steps/s. This parameter was carefully chosen based on empirical observations to balance performance and mechanical limitations. Speeds higher than 1000 steps/s caused slippage in the belt and pulley system, which compromised the reliability and precision of the controller. Conversely, speeds lower than this threshold resulted in the pulley failing to rotate, likely due to insufficient torque generation to overcome system resistance.

Similarly, the acceleration of the stepper motor was optimized at 500 steps/s$^2$. Higher acceleration rates caused the pulley to drift, indicating a loss of synchronization between the motor and the pulley, likely due to inertial effects or mechanical backlash. By setting the acceleration at 500 steps/s$^2$, the system-maintained stability and prevented such drifting, ensuring reliable operation across different operating conditions.

The selected parameters for speed and acceleration, combined with the mechanical configuration, enabled the pressure regulator to achieve consistent and precise adjustments. Experimental results demonstrate that the motor's step control directly correlates with the resulting air pressure, as detailed in Table 3. For instance, to achieve an air pressure of 1 bar, the stepper motor must complete 22 revolutions, which translates to the pulley completing 5.5 revolutions due to the gear ratio. An increase of 0.5 bar requires the stepper motor to complete an additional 2 revolutions, corresponding to 0.5 revolution of pulley rotation. By controlling the number of motor revolutions as listed in Table 3, the desired pressure can be accurately and reliably achieved with 100% rate of success after 55 trials. This operation has proven to be consistent and reliable, even under repetitive use, ensuring dependable performance across various applications.

Table 3. Correlation between stepper motor rotations, pulley rotations, and resulting air pressure.

| Air pressure (bar) | Pulley rotation (revolution) | Stepper motor rotation (revolution) |
|---|---|---|
| 1 | 5.5 | 22 |
| 1.5 | 6 | 24 |

| Air pressure (bar) | Pulley rotation (revolution) | Stepper motor rotation (revolution) |
|---|---|---|
| 2 | 6.5 | 26 |
| 2.5 | 7 | 28 |
| 3 | 7.5 | 30 |
| 3.5 | 8 | 32 |
| 4 | 8.5 | 34 |
| 4.5 | 9 | 36 |
| 5 | 9.5 | 38 |
| 5.5 | 10 | 40 |
| 6 | 10.5 | 42 |

## IV. Conclusion

This research successfully demonstrates the feasibility of a low-cost programmable air regulator tailored for soft pneumatic actuators (SPAs), achieving precise and reliable control over air pressure through the use of widely available and inexpensive components. The design utilizes a stepper motor with a resolution of 1.8 degrees per step and a 1:4 gear ratio, providing fine granularity in pressure adjustment. The carefully selected speed (1000 steps/s) and acceleration (500 steps/s$^2$) parameters balance performance and mechanical limitations, ensuring consistent operation under repetitive use.

The Arduino UNO, an accessible and user-friendly microcontroller, forms the foundation of the control system. The program code developed in this study can be readily adopted by researchers and practitioners. By controlling the number of motor revolutions, the desired pressure can be accurately and reliably achieved. The regulator's ability to reliably achieve desired pressures highlights its potential as a cost-effective alternative to commercially available options, making it particularly appealing for researchers, small industries, and educational institutions with budget constraints.

## V. Acknowledgement

This research was funded by Hibah DTMI 2024 with number 2322901/UN1.FTK/SK/HK/2024.

## References

[1] C. Tawk and G. Alici, “A Review of 3D-Printable Soft Pneumatic Actuators and Sensors: Research Challenges and Opportunities,” Adv. Intell. Syst., vol. 3, no. 6, 2021, doi: 10.1002/aisy.202000223.

[2] M. Li, A. Pal, A. Aghakhani, A. Pena-Francesch, and M. Sitti, “Soft actuators for real-world applications,” Nat. Rev. Mater., vol. 7, no. 3, pp. 235–249, 2022, doi: 10.1038/s41578-021-00389-7.

[3] Y. Yang, Y. Wu, C. Li, X. Yang, and W. Chen, “Flexible Actuators for Soft Robotics,” Adv. Intell. Syst., vol. 2, no. 1, 2020, doi: 10.1002/aisy.201900077.

[4] P. L. García, S. Crispel, E. Saerens, T. Verstraten, and D. Lefeber, “Compact Gearboxes for Modern Robotics: A Review,” Front. Robot. AI, vol. 7, no. August, 2020, doi: 10.3389/frobt.2020.00103.

[5] M. Zou et al., “Progresses in Tensile, Torsional, and Multifunctional Soft Actuators,” Adv. Funct. Mater., vol. 31, no. 39, p. 2007437, Sep. 2021, doi: https://doi.org/10.1002/adfm.202007437.

[6] N. El-Atab et al., “Soft Actuators for Soft Robotic Applications: A Review,” Adv. Intell. Syst., vol. 2, no. 10, 2020, doi: 10.1002/aisy.202000128.

[7] H. Li, J. Yao, P. Zhou, X. Chen, Y. Xu, and Y. Zhao, "High-force soft pneumatic actuators based on novel casting method for robotic applications," Sensors Actuators, A Phys., vol. 306, p. 111957, 2020, doi: 10.1016/j.sna.2020.111957.

[8] T. Takayama and Y. Sumi, "Self-excited air flow passage changing device for periodic pressurization of soft robot," ROBOMECH J., vol. 8, no. 1, 2021, doi: 10.1186/s40648-021-00207-3.

[9] B. Gorissen, D. Reynaerts, S. Konishi, K. Yoshida, J.-W. Kim, and M. De Volder, "Elastic Inflatable Actuators for Soft Robotic Applications," Adv. Mater., vol. 29, no. 43, p. 1604977, Nov. 2017, doi: https://doi.org/10.1002/adma.201604977.

[10] G. M. Whitesides, "Soft Robotics," Angew. Chemie Int. Ed., vol. 57, no. 16, pp. 4258–4273, Apr. 2018, doi: https://doi.org/10.1002/anie.201800907.

[11] M. S. Xavier, A. J. Fleming, and Y. K. Yong, "Design and Control of Pneumatic Systems for Soft Robotics: A Simulation Approach," IEEE Robot. Autom. Lett., vol. 6, no. 3, pp. 5800–5807, 2021, doi: 10.1109/LRA.2021.3086425.

[12] W. Dou, G. Zhong, J. Cao, Z. Shi, B. Peng, and L. Jiang, "Soft Robotic Manipulators: Designs, Actuation, Stiffness Tuning, and Sensing," Adv. Mater. Technol., vol. 6, no. 9, p. 2100018, Sep. 2021, doi: https://doi.org/10.1002/admt.202100018.

[13] J. Walker et al., "Soft Robotics: A Review of Recent Developments of Pneumatic Soft Actuators," Actuators, vol. 9, no. 1. p. 3, 2020. doi: 10.3390/act9010003.

[14] T. Sénac et al., "A review of pneumatic actuators used for the design of medical simulators and medical tools," Multimodal Technol. Interact., vol. 3, no. 3, 2019, doi: 10.3390/mti3030047.