machines

MDPI

Article

Development of a Robotic Manipulator for Piano Performance via Numbered Musical Notation Recognition

Pu-Sheng Tsai [1], Ter-Feng Wu [2,*] and Chen-Ting Liao [1]

[1] Department of Electrical Engineering, Ming Chuan University, Taoyuan 333321, Taiwan; pusheng@mail.mcu.edu.tw (P.-S.T.); sjjsjj22528794@gmail.com (C.-T.L.)
[2] Department of Electrical Engineering, National Ilan University, Yilan 260007, Taiwan
* Correspondence: tfwu@niu.edu.tw

Abstract

This paper presents a piano-playing robotic system that integrates numbered musical notation recognition with automated manipulator control. The system captures the notation using a camera, applies four-point detection for perspective correction, and performs measure segmentation through an orthogonal projection method. A pixel-scanning technique is then used to locate the positions of numerical notes, pitch dots, and rhythmic markers. Digit recognition is achieved using a CNN model trained on both the MNIST handwritten digit dataset and a custom computer-font digit dataset (CFDD), enabling robust identification of numerical symbols under varying font styles. The hardware platform consists of a 3D-printed robotic hand mounted on a linear rail and driven by an ESP32-based embedded controller with custom driver circuits. According to the recognized musical notes, the manipulator executes lateral positioning and vertical key-press motions to reproduce piano melodies. Experimental results demonstrate reliable notation recognition and accurate performance execution, confirming the feasibility of combining computer vision and robotic manipulation for low-cost, automated musical performance.

Keywords: numbered musical notation recognition; convolutional neural network (CNN); orthogonal projection method; four-point detection; pixel-scanning method; MNIST handwritten digit dataset

check for updates

Academic Editors: Quang Minh Ta and Tan Chen

Received: 27 October 2025
Revised: 25 November 2025
Accepted: 1 December 2025
Published: 5 December 2025

Citation: Tsai, P.-S.; Wu, T.-F.; Liao, C.-T. Development of a Robotic Manipulator for Piano Performance via Numbered Musical Notation Recognition. *Machines* **2025**, *13*, 1121. https://doi.org/10.3390/machines13121121

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Piano performance requires extensive training; therefore, the concept of automated playing has attracted attention since the late nineteenth century. Roehl [1] noted that, during this period in Europe, mechanical devices equipped with wooden fingers were developed to mimic the hand movements of musicians and realize automatic piano performance. In 1984, Waseda University introduced WABOT-2, a humanoid keyboard-playing robot with vision, speech, performance, and control functions. The system featured 21 degrees of freedom in its arms and a three-layer control architecture, enabling it to recognize musical scores, play the keyboard and pedals, and establish a foundation for the development of music robots (see Figure 1) [2,3]. Building upon WABOT-2, Sumitomo Electric developed WASUBOT, which was exhibited at the Japan Pavilion during the 1985 World Exposition to demonstrate its musical performance capabilities.

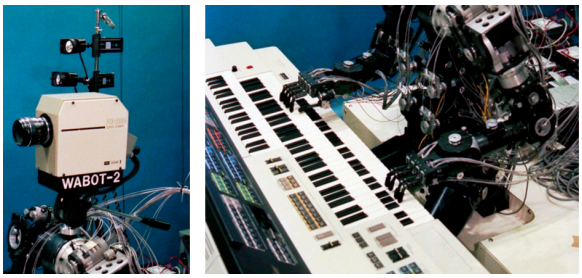


Figure 1. The WABOT-2 piano-playing robot.

Zhang et al. [4] designed a piano-playing robot that integrates dexterous hand mechanisms with linear motion control, achieving precise performance through CAD simulations and a gear–rail structure. Jaju et al. [5] developed a robotic system in which finger actuation is realized by pressing keys downward and returning them via elastic cords, with an embedded microcontroller controlling musical execution. Fahn et al. [6] further advanced the field by introducing a two-handed piano-playing robot capable of both score recognition and autonomous performance. The system comprises two main modules, vision and performance, where fingers are actuated by electromagnets for accurate key pressing, while the arms employ linear motors to slide across the keyboard. This design enables simultaneous playing of melody and accompaniment with chordal capability. Teotronico, developed in 2015 by the Italian company TeoTronica, is a piano-playing robot equipped with 88 electromagnetically actuated fingers covering the full keyboard range. It is capable of score reading and sustaining pedal operation. Designed with a humanoid appearance featuring movable facial components, the robot provides a highly realistic performance effect (see Figure 2). Palmieri et al. [7] reported that Teotronico includes a "mirror pianist" mode, enabling synchronous playing, solo performance, and remote concerts, along with speech synchronization and human–robot interaction capabilities. The expressive facial movements and sensing devices enhance the realism of its performances, which have been showcased on international stages multiple times.



Figure 2. The Teotronico piano-playing robot.

Vear [8] examined performance-oriented robotic projects' chronological progress and introduced the human-centered artificial intelligence (HC-AI) concept, which emphasizes real-time collaboration and interaction between AI and humans. Using Teotronico as an example, the study explored the potential of robotic participation in improvisational music creation. Building upon this context, the present work develops a piano-playing robotic system equipped with numbered musical notation recognition, linear rail actuation, and finger-striking mechanisms. Unlike most systems based on staff notation, the proposed system is specifically designed for numbered musical notation, which is widely used in

East Asian music education and performance, converting score images into finger actuation commands through vision-based recognition. To achieve this goal, vertical and horizontal projection, digit-scanning, and a convolutional neural network (CNN) are employed for note segmentation and recognition. The following section reviews related techniques and applications.

Music Notation Recognition (MNR) combines computer vision and artificial intelligence to automatically detect and identify numeric symbols in numbered musical notation, with broad applications in instruments such as piano and guitar. The core workflow typically includes image acquisition, orthogonal projection, symbol segmentation, and symbol recognition. First, score images are captured using a camera or scanner, and four-point detection is introduced in this study to correct distortions caused by shooting angles. The images are then processed through grayscale conversion, binarization, and denoising to eliminate background interference and ensure clarity. For symbol detection, early approaches such as edge and contour analysis showed limited accuracy when dealing with tilted or overlapping symbols. More recently, deep learning models such as YOLO and U-Net have been adopted to enhance recognition precision. Finally, segmented symbols can be classified using Optical Character Recognition (OCR) or hybrid architectures such as CRNN, which integrates convolutional neural networks (CNNs) with recurrent neural networks (RNNs), allowing robust handling of variations in fonts and handwritten styles.

Handwritten digit string segmentation remains challenging due to factors such as character skew, overlap, connectivity, and the unknown length of digit sequences. Gattal and Chibani [9,10] proposed several segmentation methods to address connected and variable-length digit strings, including oriented sliding windows, vertical projection histograms, contour analysis, and the Radon transform. These methods were integrated with a support vector machine (SVM) classifier for recognition and validation, while a global decision mechanism was employed to determine whether each segmented digit should be accepted, thereby improving both segmentation and recognition performance. In addition, multiple submodules were incorporated into the system to enhance robustness, and evaluations were conducted using the NIST SD19 database. Chen and Guo [11] addressed the problem of connected digits using spectral clustering (SC) combined with an SVM-based affinity matrix prediction, while incorporating graph partitioning and a secondary optimization strategy to improve overall accuracy. Ribas et al. [12] evaluated various segmentation algorithms on both synthetic and real databases, highlighting their complementary strengths across different connection patterns. They observed that some algorithms could successfully process samples where others failed, thereby recommending fusion strategies to enhance overall segmentation performance. Hochuli et al. [13] proposed replacing the traditional segmentation pipeline with multiple convolutional neural network (CNN) classifiers, which directly predicted string length and performed classification. This approach effectively handled overlapping digits and demonstrated superior performance compared with conventional methods. The reviewed studies highlight the strengths and limitations of segmentation strategies, from conventional projection and clustering methods to CNN-based end-to-end solutions. As segmentation lays the foundation for accurate interpretation of musical notation, the next section addresses the equally crucial problem of digit recognition.

As traditional machine learning approaches increasingly encounter limitations in handling high-dimensional and nonlinear problems, researchers began constructing multilayer neural networks to mimic the hierarchical structure of neurons in the human brain, leading to the emergence of deep learning techniques. This section briefly reviews the historical development and theoretical foundations of deep learning. Minsky and Papert (1969) [14] demonstrated that a single-layer perceptron was incapable of solving nonlin-

ear problems such as XOR, which caused a significant stagnation in artificial intelligence research. In the 1980s, Rumelhart et al. (1986) [15] introduced the multilayer perceptron (MLP) with hidden layers and the backpropagation (BP) algorithm, effectively overcoming the limitations of single-layer models. Hecht-Nielsen (1989) [16] further demonstrated that a three-layer neural network could approximate any $\mathcal{L}^2$ function, thereby establishing a theoretical foundation for neural network research. LeCun (1989) et al. [17] proposed a network architecture incorporating the concepts of local connectivity and weight sharing, and later developed the classical convolutional neural network (CNN) model LeNet-5 in 1998 [18], which was applied to postal code and check recognition. This network integrated convolutional layers, pooling layers, and fully connected layers, forming the foundation for subsequent deep CNN architectures such as AlexNet, VGG, and ResNet. Hinton and Salakhutdinov (2006) [19] introduced the restricted Boltzmann machine (RBM), which enabled layer-wise unsupervised pretraining of deep neural networks to effectively alleviate the vanishing gradient problem. They further proposed the deep belief network (DBN), marking the beginning of a new era in deep learning.

Convolutional neural networks (CNNs) constitute the core architecture of deep learning, with numerous subsequent models evolving from this foundation. AlexNet, proposed by Krizhevsky et al., introduced multiple convolutional layers, max pooling, and dropout techniques, achieving a decisive victory in the 2012 ImageNet competition and marking the beginning of the deep learning era [20]. VGGNet, developed at the University of Oxford, employed stacks of 3×3 convolutional kernels to construct networks with 16–19 layers, demonstrating outstanding performance in the 2014 ImageNet classification and localization tasks and highlighting the advantages of deeper architectures [21]. GoogLeNet (also known as Inception Network), developed by Google, innovatively introduced the Inception module, which applied multiple convolutional kernels and pooling operations within a single layer to achieve multi-scale feature extraction while reducing parameter count, showing strong performance in the 2014 ImageNet challenge [22]. ResNet (Residual Network), proposed by He et al. [23], introduced residual learning with skip connections to address the vanishing gradient problem in very deep networks. This approach enabled the successful training of networks up to 152 layers and achieved state-of-the-art results, winning first place in multiple tasks at ILSVRC and COCO in 2015, thereby becoming a milestone in deep learning [23].

In the evolution of CNN architectures, LeNet established the foundational framework, AlexNet drove practical adoption, VGGNet and GoogLeNet expanded the diversity of architectural design, and ResNet overcame the limitations of network depth. Building upon these foundations, Redmon et al. introduced YOLO (You Only Look Once) in 2015, a real-time object detection algorithm based on CNNs. Unlike traditional two-stage detection pipelines, YOLO reformulated object detection as a regression problem by directly predicting bounding box locations and class probabilities from the entire image, thereby eliminating the need for a region proposal stage. This innovation significantly improved detection speed and has since become a key technique in real-time computer vision applications. These breakthroughs paved the way for modern applications of CNNs in digit recognition tasks, thereby providing the theoretical and technical foundation for the approach adopted in this study.

2. Hardware Architecture

The proposed system integrates image processing, deep learning, and robotic manipulator control to achieve automated recognition of numbered musical notation and piano performance. The overall hardware architecture is illustrated in Figure 3 and primarily consists of a camera, a Raspberry Pi 4, an ESP32 microcontroller, a custom driver circuit

board, and a mechanical hand. The camera captures images of the musical score, which are transmitted to the Raspberry Pi for preprocessing and note recognition. The recognition results are then sent to the ESP32 to control the manipulator for key actuation. This architecture features a modular design with a high degree of integration, enabling real-time conversion of visual inputs into concrete keyboard outputs, thereby demonstrating the synergy between vision-based recognition and embedded control.

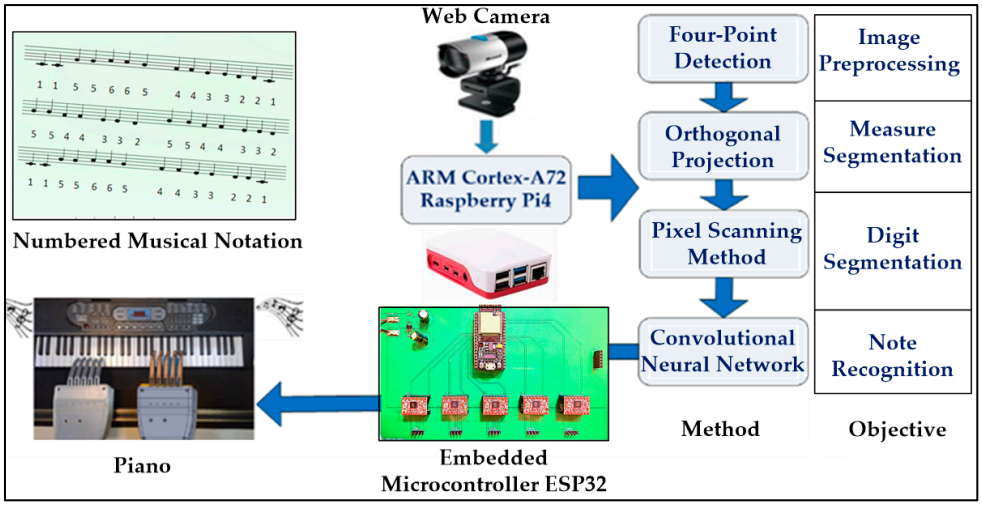


Figure 3. System architecture.

2.1. Robot Hand

The primary actuation system of the piano-playing robot consists of a linear rail mechanism and a dual-hand structure with ten fingers, among which the design of the palm and finger mechanisms poses the greatest challenge. Initially, commercial bionic robotic hands were evaluated, as they provide high realism in appearance and motion, with basic grasping functionality (see Figure 4). However, experimental results revealed that the finger configuration did not align precisely with the piano keys, and the servo motors lacked sufficient torque, resulting in unstable key pressing and reduced fluency and accuracy in performance. To address these limitations, this study adopted a custom-developed design tailored to the specific requirements of piano keys, incorporating dedicated actuation and control mechanisms to ensure stable and feasible performance.

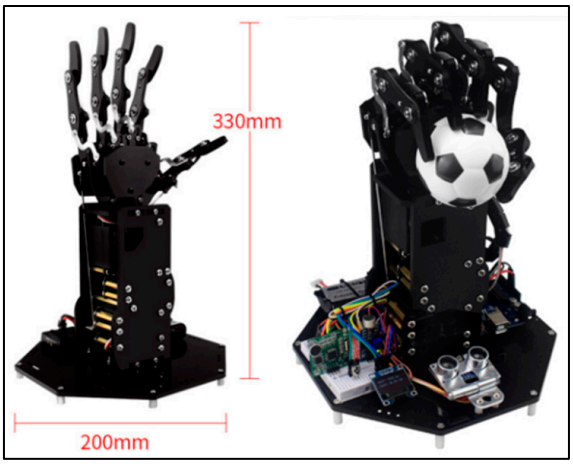


Figure 4. Bionic robotic hand.

To address the misalignment between finger spacing and piano keys, Wang [24] designed a custom 3D-printed robotic hand with servo motors mounted on the back of the

hand to provide sufficient torque for key pressing. Five high-torque servo motors were installed in recessed slots on the dorsal side of the hand (Figure 5(3)), each connected to a corresponding finger via steel wires. Finger depression was achieved through pulley traction (Figure 5(1)), while release allowed the fingers to rebound (Figure 5(2)). Each finger incorporated a single movable joint, using a 2 mm copper wire as a flexible pivot to enable basic vertical motion. The fingers were spaced 2 cm apart and independently controlled, enhancing multi-finger coordination. In addition, helical springs were attached to both the back of the hand and the fingers (Figure 5(4)) to facilitate rapid finger lifting and resetting, forming a simple yet stable key-striking mechanism.

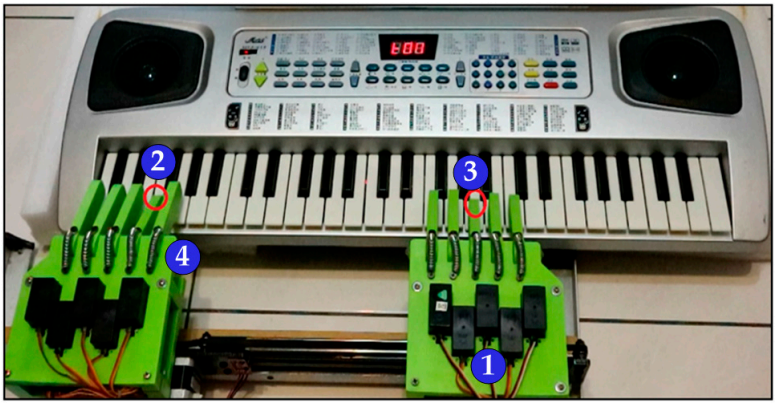
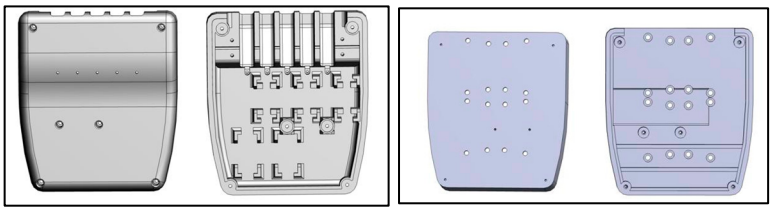


Figure 5. System schematic proposed by Wang [24].

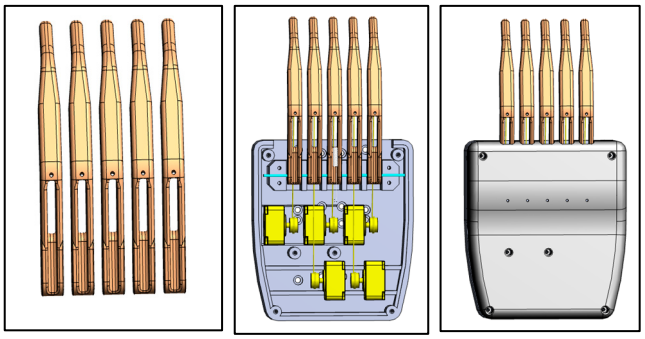
Although the robotic hand shown in Figure 5 was capable of successfully performing musical pieces, its performance remained far from comparable to that of professional pianists. The primary limitation stemmed from the fact that both key pressing and finger lifting were driven by servo motors, resulting in motion delays. When encountering rapid notes with intervals shorter than 250 ms, missed notes frequently occurred. Furthermore, lateral movement of the hand required approximately 113 ms per key, causing rhythmic delays that reduced fluency. In addition, the high noise generated by servo motor operation adversely affected the sound quality of the performance. These limitations indicate that the current robotic hand still requires further optimization. Therefore, this study redesigns and fabricates a robotic hand mechanism with high speed, low latency, and low noise, in order to enhance both performance efficiency and musical quality.

In this study, the palm model was fabricated using 3D printing. Figure 6a,b illustrate the top views of the upper and lower palm covers, respectively. Figure 7a illustrates a top view of the five fingers, whereas Figure 7b,c depict the perspective and overall views of the integrated design combining the stepper motors with the fingers, respectively.



(**a**) (**b**)

Figure 6. Structures of the palm covers: (**a**) top view of the upper cover; (**b**) top view of the lower cover.



(**a**) (**b**) (**c**)

Figure 7. Assembly process of the robotic hand: (**a**) finger modules; (**b**) perspective view of finger installation; (**c**) integration of fingers with upper and lower covers.

The palm structure employs stepper motors to pull fishing lines for finger actuation, with springs enabling the fingers to return to their initial positions. In the preliminary experiments, the edges of the PCB board were used as anchor and guiding points for the strings to ensure stable motion along the intended path. To validate the effectiveness of finger actuation, both enameled wire and fishing line were tested as connections to the motors for traction-based key pressing. The enameled wire exhibited insufficient toughness and frequently broke under repeated pulling. In contrast, the fishing line demonstrated high tensile strength and durability, significantly improving actuation stability. As a result, fishing line was adopted as the standard transmission material in the current design. From a mechanical perspective, the required torque and output power increase with longer fishing lines or larger angles between the line and the motor's exit direction. Consequently, parameters such as motor placement, initial finger angle, and the lengths of the springs and lines are critical design factors. Since motor torque is constrained by size and installation space, simply increasing torque is not feasible; therefore, the reduction of finger actuation force must be prioritized. To address this, multiple experiments were conducted to iteratively adjust motor height, string attachment points, and the lengths of the springs and lines. Through this process, a balanced set of parameters was achieved, while simultaneously optimizing finger arrangement and string routing.

In the initial experiments, it was observed that the finger reset time was excessively long. Since the stepper motor required the same number of steps for both pressing and returning, a 500 ms pressing motion also required 500 ms to reset, thereby affecting hand movement and overall playing rhythm. To address this issue, the enable signal of the stepper motor driver was set to high after each key press, placing the motor shaft in an unlocked state and allowing the spring to provide the restoring force. This modification significantly reduced the reset time. However, because the spring rebound lacked precise step control, excessive return could occur, resulting in an elevated initial angle that increased the torque required for the next key press and reduced the success rate of actuation. To resolve this, a limit screw was installed at the upper-right corner of the motor module, serving as a mechanical stop. When the finger returned, the PCB board edge contacted the screw, ensuring consistent motion and preventing deviations in control parameters.

2.2. Linear Rail Mechanism

The robotic hand is mounted on the slider of a linear rail (circled in Figure 8), and its movement is driven by a belt actuated by a stepper motor, enabling a travel range of 750 mm. The specifications of the rail are provided in Table 1. As shown in Figure 8, the hand is fixed to the slider, while the linear rail assembly includes a belt, a stepper motor, and limit switches. A limit switch is installed at the starting position; when triggered by the

slider, it stops the motor to prevent structural damage. At the end of the rail, the terminal position is recorded by the firmware to avoid overshooting. Each time the robot is powered on or upon completion of a musical piece, the firmware commands the slider to return to the starting point until the limit switch produces a feedback signal. This procedure ensures rapid and accurate localization of the hand before initiating the next piece, thereby improving control efficiency.

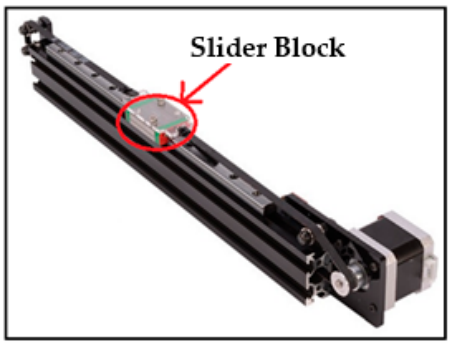


Figure 8. Linear rail mechanism.

Table 1. Linear rail specifications.

Item	Specification
Aluminum Profile	2040
Stepper Motor	NEMA23
Linear Guide	MGN-12H
Guide Length	300 mm
Total Length	415 mm
Timing Belt	GT2 6 mm
Synchronous Pulley	20 teeth
Other Components	Coupler

The system employs a GT2 timing belt with a 2 mm tooth pitch (Figure 9) and a 20-tooth pulley (Figure 10), which is mounted on the shaft of a stepper motor. When the motor rotates, the pulley drives the GT2 belt, causing the slider to move linearly along the rail. Since the belt has a 2 mm pitch, one full rotation of the 20-tooth pulley results in a slider displacement of 40 mm. The system utilizes a NEMA23 stepper motor with a two-phase, four-pole design $n = 2$ and a rotor tooth count of $Z_r = 50$, combined with a driver module configured for 1/4 microstepping $N = 4$. According to Equation (1), each microstep corresponds to a linear displacement of 0.05 mm along the rail, enabling high-precision position control.

$$d_N = \frac{40\ \text{mm}}{(2 \times n \times Z_r) \times N} \tag{1}$$

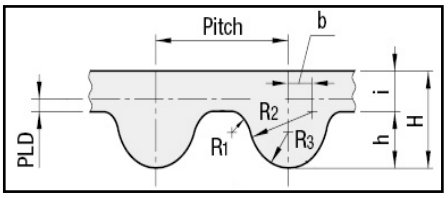


Figure 9. GT2 belt–pulley mechanism of the stepper motor.



Figure 10. 20-tooth timing pulley.

When the belt advances by 10 teeth (20 mm), it corresponds to the width of one piano key, while the motor shaft simultaneously rotates 10 teeth, equivalent to 180°. Therefore, moving the robotic hand across the distance of one piano key requires 400 microsteps of the motor, as expressed in Equation (2).

$$\text{Motor microsteps} = \frac{\text{Shaft rotation angle}}{\text{Miceostep angle}} = \frac{180^{o}}{0.45^{o}} = 400 \quad (2)$$

The required number of motor microsteps for moving the robotic hand across one piano key is obtained by dividing the shaft rotation angle (180°) by the microstep angle. Based on the system configuration, this results in 400 microsteps per key displacement. The overall system architecture is illustrated in Figure 11.

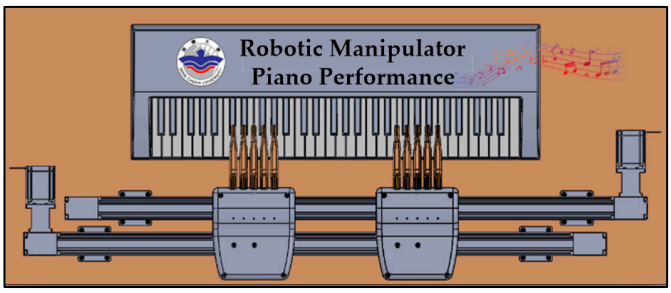


Figure 11. Overall system architecture.

2.3. Driver Circuit

The system incorporates two types of stepper motor drivers to control different subsystems: the A4988 is used for finger actuation, while the DM556 is dedicated to linear rail positioning and movement. Both drivers are controlled via pulse signals, with the motion precision and speed adjusted through the microcontroller. The following subsections describe the circuit design and parameter configurations of each driver module.

(1) A4988 Stepper Motor Driver Module

The system utilizes the A4988 driver for controlling bipolar stepper motors, featuring built-in current regulation and microstepping capability that enables precise control through simple digital signals. The A4988 requires only two control pins: STEP, which receives rising-edge pulses to drive motor movement, and DIR, which specifies the rotation direction. An onboard variable resistor allows the maximum current to be set, providing overcurrent protection. The A4988 supports multiple microstepping modes; however, for simplicity, the system operates in the default full-step mode. The ENABLE pin is fixed in the active state, ensuring that the driver is always ready to receive control signals after system startup, thereby improving responsiveness and stability. The A4988 circuit is illustrated in Figure 12.

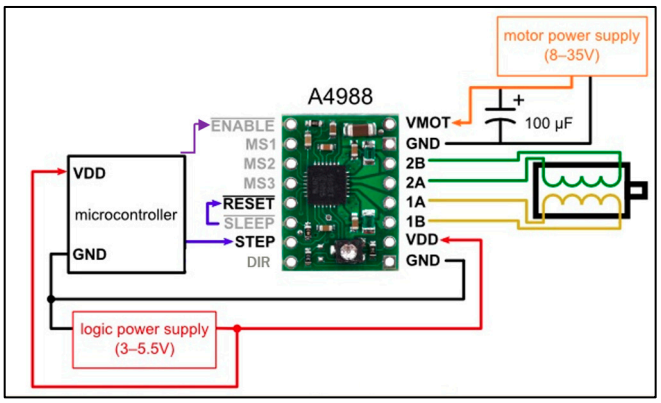


Figure 12. A4988 driver circuit diagram.

To improve the thermal performance and stability of the A4988 driver module, this study adopted the factory-supplied aluminum heat sink, which was directly attached to the top of the chip. Based on experimental measurements, the actual output current of the A4988 was observed to be approximately 70% of the configured current limit. For the stepper motor used in this system, with a rated current of 1.7 A, the maximum current limit of the A4988 should therefore be set to 2.43 A. The A4988 module regulates the internal current reference voltage V_{ref} through an adjustable variable resistor (VR) to determine the maximum output current I_{max}. The corresponding setting equation is given as follows:

$$I_{max} = \frac{V_{ref}}{8 \times R_s} \tag{3}$$

where R_s represents the current sense resistor, which is labeled as 0.1 Ω on the module. For the system requirement of $I_{max} = 2.43$ A, the corresponding reference voltage can be calculated as:

$$V_{ref} = I_{max} \times 0.8 \approx 1.94 \text{ V} \tag{4}$$

The module allows users to adjust the reference voltage V_{ref} by rotating the metal trimmer potentiometer located at the bottom of the board (Figure 12). During adjustment, the black probe of a multimeter is connected to ground, while the red probe contacts the center metal point of the potentiometer to measure the voltage. Once V_{ref} is set, the module automatically limits the maximum current per phase to ensure stable operation and prevent overheating. When the reference voltage of the A4988 driver module is set to 1.94 V, the maximum output current reaches 2.43 A. Considering that the stepper motor used in this system has a rated current of 1.7 A and is driven at 12 V DC, the rated power of the motor can be further calculated as $P = 20.4$ W.

(2) DM556 Stepper Motor Driver Module

For linear rail actuation and positioning, the system adopts the DM556 digital two-phase stepper motor driver as the control interface. The DM556 supports a DC input range of 20–50 V and delivers a maximum output current of 5.6 A, making it suitable for medium- to large-sized NEMA 23–34 two-phase stepper motors. With high torque output and stable long-duration operation, it is particularly well suited for driving linear rails under heavy load conditions. The pin configuration is shown in Figure 13. The DM556 operates on a digital pulse control architecture with three control terminals: PUL (pulse input), DIR (direction control), and ENA (enable/disable function). Built-in DIP switches allow configuration of microstepping resolution (up to 51,200 steps/rev) and maximum drive current, thereby enhancing positioning accuracy and overall system efficiency. This makes the DM556 highly suitable for the high-precision and smooth movement requirements of the rail module in this system.

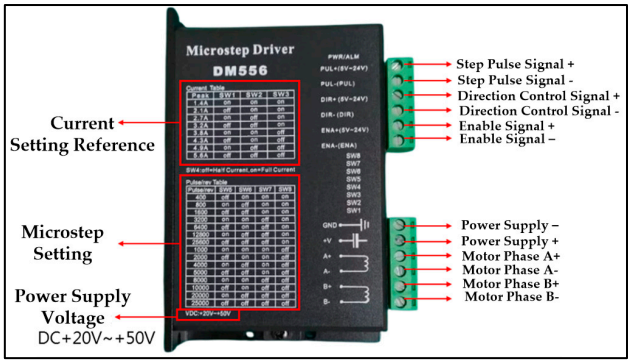


Figure 13. DM556 driver pin functions.

2.4. Embedded Microcontroller

The ESP32 is built on the Tensilica Xtensa 32-bit LX6 microprocessor, delivering up to 600 MIPS of computational performance, enabling standalone execution of embedded applications. Taking the Anxin ESP32-C3F as an example, it integrates 400 KB of SRAM, 384 KB of ROM, 8 KB of RTC SRAM, and 4 MB of flash memory. The chip supports multiple low-power modes, with deep sleep current consumption below 5 μA, making it highly suitable for battery-powered devices. Wireless communication includes IEEE 802.11 b/g/n Wi-Fi and Bluetooth 5.0, supporting STA, AP, STA + AP, and promiscuous modes, providing high network flexibility. The interface design integrates multiple communication protocols such as UART, PWM, SPI, I^2C, and ADC, with support for up to 15 GPIO pins, offering strong expandability.

In this system, the A4988 driver is used for finger actuation and the DM556 driver for rail movement, both operating with standard pulse control. The STEP pin receives rising-edge pulses to advance the motor, while the DIR pin determines the rotation direction. For the unidirectional finger actuation mechanism, resetting is accomplished by springs; therefore, the hardware design omits the DIR pin, retaining only STEP and ENABLE. The ESP32 controller allocates 10 GPIO pins for each hand circuit to drive five A4988 modules, corresponding to five fingers for key actuation. After a keystroke is completed, the ENABLE signal is pulled high to release the motor shaft, allowing spring force to reset the finger rapidly and thereby improving responsiveness under high-tempo performance conditions. The DM556 driver module controls the lateral movement and positioning of the linear rail, enabling precise finger placement on the corresponding piano keys in synchronization with rhythm signals and performance strategies. The ESP32 controller uses two GPIO pins to control the PUL+ and DIR+ inputs of the DM556, generating stable pulse sequences through step counting to achieve rapid and accurate rail actuation. Both the A4988 and DM556 feature four motor output terminals (A+, A−, B+, B−), corresponding to the two windings of the stepper motor. The circuit schematic and PCB layout were designed in Altium Designer, integrating the ESP32 microcontroller, five A4988 finger driver modules, and one DM556 rail control module. The system operates with a DC 24 V input, which powers the circuit board, A4988, and DM556, while a low-dropout regulator (LDO) converts 24 V to 5 V to supply the ESP32 and the VDD pins of the A4988. The modular design simplifies both hardware and firmware development, enhances stability and maintainability, and the completed PCB is shown in Figure 14.

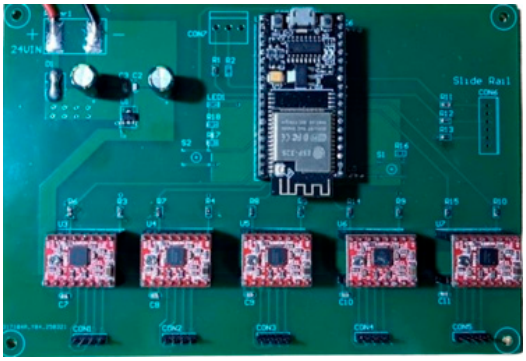


Figure 14. Fabricated PCB of the embedded microcontroller.

2.5. Embedded Microprocessor

The system incorporates a Raspberry Pi 4, powered by the Broadcom BCM2711 SoC, which features a quad-core 64-bit Cortex-A72 processor with a clock speed of up to 1.5 GHz and an integrated VideoCore VI GPU providing 3D graphics and hardware acceleration. The board is equipped with 8 GB of LPDDR4 RAM and supports dual-band 2.4 GHz/5 GHz Wi-Fi, Bluetooth 5.0, and BLE. Video output includes two micro HDMI ports (supporting HDMI 2.0 dual 4K displays), along with CSI camera and DSI display interfaces, and a 3.5 mm audio output. Connectivity is further enhanced with two USB 3.0 ports, two USB 2.0 ports, and a 40-pin GPIO header. The board supports multiple operating systems, including Raspberry Pi OS, Ubuntu, and Windows 10 IoT Core. With compact dimensions of only 85 × 56 × 17 mm (Figure 15), the Raspberry Pi 4 is well suited for embedded and IoT development applications.

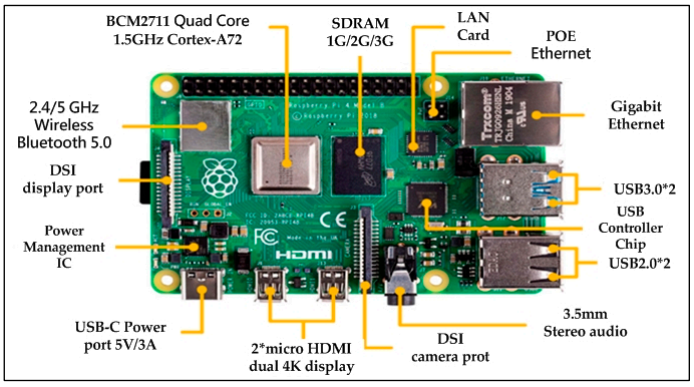


Figure 15. Raspberry Pi 4 peripheral interfaces.

In the following section, the note recognition process is introduced, covering image preprocessing, segmentation, and digit recognition.

3. Note Recognition System

The proposed system is operated through a graphical user interface (GUI) that provides nine functional buttons in sequence: score capture, score preprocessing (including four-point detection, grayscale conversion, and binarization), score segmentation (including measure and note segmentation), digit recognition, MIDI conversion, start performance, end performance, serial transmission, and exit program. This section represents the core contribution of this study, integrating computer vision and deep learning techniques to achieve robust recognition of numbered musical notation. The functionality of each button is detailed in the following subsections.

3.1. Score Preprocessing (Four-Point Detection)

Before numbered musical notation can be recognized, the captured images must undergo preprocessing. Since variations in camera angle may cause skew or distortion, which would reduce the accuracy of subsequent measure segmentation and projection-based localization, image rectification is required. A commonly used approach is four-point detection with perspective correction, which detects the four corner points of the score image, constructs a perspective transformation matrix, and rectifies the image to a frontal view for reliable analysis. This subsection describes the four-point detection process, including rectangle identification, corner point ordering, transformation matrix construction, and implementation steps.

(1) Identification of Approximate Rectangular Regions:

To extract target areas with well-defined geometric shapes (e.g., documents or signs) from the input image, contour detection is first applied to identify the boundaries of all closed regions. In this study, the findContours() function from the OpenCV library was used to extract boundary information of all closed shapes in the image. However, findContours() does not restrict the contour shapes and may return triangles, circles, or irregular polygons; therefore, an additional filtering process is required to identify candidate regions approximating rectangles. Subsequently, the approxPolyDP() function is applied to each contour to perform polygonal approximation, simplifying the contour structure and obtaining the number of vertices. If the approximated polygon has four edges, it is considered a potential rectangular region. Moreover, to exclude concave or irregular quadrilaterals, the isContourConvex() function is used to verify whether the contour is convex. Finally, an area threshold is set to filter out noisy or non-representative shapes by discarding regions with excessively small areas. The identified rectangular region serves as the basis for subsequent corner point detection.

(2) Corner Point Ordering Method:

Before performing perspective transformation, the four detected corner points must be consistently ordered to ensure correct coordinate mapping. A common ordering scheme arranges the vertices as top-left, top-right, bottom-right, and bottom-left, establishing the correspondence for the perspective transformation. In this study, the Sum-Difference Method was adopted, which calculates $x + y$ and $y - x$ for each point to determine their positions: the point with the smallest $x + y$ is assigned as the top-left, while the largest corresponds to the bottom-right; the point with the smallest $y - x$ is designated as the top-right, and the largest as the bottom-left. This method effectively preserves the geometric relationships among the vertices, enabling accurate perspective correction.

(3) Perspective Transformation:

Perspective transformation is a commonly used technique in projective geometry that maps four points in an image onto four designated points on another plane. This process corrects geometric distortions caused by camera angles, converting skewed or deformed regions into a frontal rectangle. The transformation is represented by a 3×3 homogeneous coordinate matrix M, which maps any point $P = (x, y)$ in the original image to a point $P' = (x', y')$ on the target plane. Since planar coordinates are processed, they must first be converted from the Euclidean coordinate system to the homogeneous coordinate system used in projective geometry, as shown in Equation (5).

$$(x, y) \equiv \lambda(x, y, 1) = (\lambda x, \lambda y, \lambda) \tag{5}$$

In projective geometry, any homogeneous coordinate $(\lambda x, \lambda y, \lambda)$, with $\lambda \neq 0$ represents the same projected point as $(x, y, 1)$. In other words, all proportional three-dimensional

vectors correspond to the same two-dimensional Euclidean coordinate. The use of homogeneous coordinates enables a unified representation of geometric transformations in images, including translation, rotation, scaling, and perspective projection, through matrix multiplication. The introduction of the scale factor λ preserves the invariance of proportions in projective geometry and allows these transformations to be handled via linear algebra. After completing a perspective transformation, the resulting three-dimensional homogeneous coordinates must be normalized by dividing by the third component λ to restore the two-dimensional coordinates (x', y'). To derive the perspective transformation matrix $M \in R^{3\times3}$, four known points in the source image, $P_i = (x_i, y_i),\ i = 1-4$, must be mapped to their corresponding points on the target plane, $P_i' = \left(x_i',\ y_i'\right)$. The perspective transformation can thus be expressed as:

$$\lambda \begin{bmatrix} x_i' \\ y_i' \\ 1 \end{bmatrix} = M \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \tag{6}$$

Expanding the right-hand side of (6) with $M = \left[m_{ij}\right]_{3\times3}$ and dividing both sides by the homogeneous scale factor λ rewrites the perspective transform as two relations between the output (x', y') and the input (x, y):

$$x_i' = \frac{m_{11}x_i + m_{12}y_i + m_{13}}{m_{31}x_i + m_{32}y_i + m_{33}} \tag{7}$$

$$y_i' = \frac{m_{21}x_i + m_{22}y_i + m_{23}}{m_{31}x_i + m_{32}y_i + m_{33}} \tag{8}$$

By applying cross-multiplication to Equations (7) and (8), the linear equation system expressed in Equation (9) can be obtained:

$$\begin{cases} (m_{11}x_i + m_{12}y_i + m_{13}) - (m_{31}x_i + m_{32}y_i + m_{33})x_i' = 0 \\ (m_{21}x_i + m_{22}y_i + m_{23}) - (m_{31}x_i + m_{32}y_i + m_{33})y_i' = 0 \end{cases} \tag{9}$$

For each pair of corresponding points $(x_i,\ y_i) \leftrightarrow \left(x_i',\ y_i'\right)$, two linear equations can be established. When four pairs of correspondences are provided, a total of eight independent linear equations are generated, which can be assembled into the following matrix form:

$$\boldsymbol{Ah} = 0 \tag{10}$$

Here, $\boldsymbol{A} \in R^{8\times9}$ denotes the coefficient matrix, and $\boldsymbol{h} \in R^9$ represents the vectorized form of the transformation matrix M (the 3×3 perspective transformation matrix) after flattening. Its structure is given as follows:

$$\boldsymbol{h} = [m_{11}\ m_{12}\ m_{13}\ m_{21}\ m_{22}\ m_{23}\ m_{31}\ m_{32}\ m_{33}]^T \tag{11}$$

For each pair of corresponding points, the two associated rows in matrix $\boldsymbol{A}$ are expressed as follows:

$$\boldsymbol{A_i} = \begin{bmatrix} x_i\ y_i\ 1\ 0\ 0\ 0\ -x_ix_i'\ -y_ix_i'\ -x_i' \\ 0\ 0\ 0\ x_i\ y_i\ 1\ -x_iy_i'\ -y_iy_i'\ -y_i' \end{bmatrix} \tag{12}$$

To solve the homogeneous linear system $\boldsymbol{Ah} = 0$, this study employs the Singular Value Decomposition (SVD) method, which decomposes the coefficient matrix $\boldsymbol{A} \in R^{\mathrm{m}\times\mathrm{n}}$ as follows:

$$\boldsymbol{A} = \boldsymbol{U\Sigma V}^T \tag{13}$$

Here, $\boldsymbol{U} \in R^{m \times m}$ is the left singular orthogonal matrix, $\boldsymbol{V} \in R^{n \times n}$ is the right singular orthogonal matrix, and $\boldsymbol{\Sigma} \in R^{m \times n}$ is the diagonal matrix containing the singular values. Since this problem is formulated as the least-squares solution that minimizes $\|\boldsymbol{Ah}\|$, the solution vector $\boldsymbol{h}$ is chosen as the right singular vector in $\boldsymbol{V}$ corresponding to the smallest singular value (i.e., the last column). This selection also satisfies the normalization condition $\|\boldsymbol{h}\| = \boldsymbol{I}$, and effectively corresponds to the vectorized form of the perspective transformation matrix. In this study, the perspective transformation matrix between the original and target point coordinates was obtained using the cv2.getPerspectiveTransform function provided by OpenCV. The image was then rectified through the cv2.warpPerspective function, which applies the transformation to correct distorted regions. After completing the four-point detection of the score image, grayscale conversion and binarization were performed to enhance recognition accuracy. Since the system analyzes the distribution of black pixels, accurate conversion to binary images highlights symbol contours and suppresses background noise, thereby establishing a solid foundation for symbol localization and classification. The preprocessing workflow for numbered musical notation is illustrated in Figure 16.

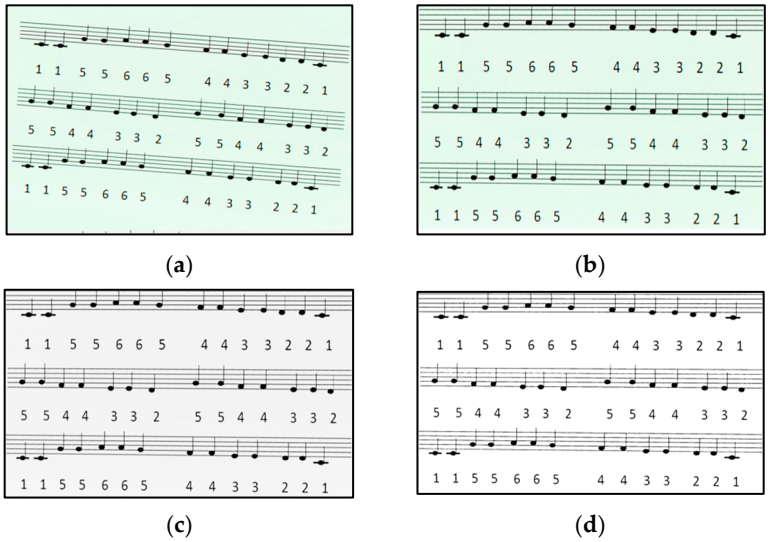


Figure 16. Score preprocessing: (**a**) original score, (**b**) four-point detection, (**c**) grayscale conversion, (**d**) binarization.

This perspective rectification ensures that subsequent projection-based segmentation and symbol recognition can be performed with improved accuracy.

3.2. Orthogonal Projection

To enable the robotic performance system to play automatically according to the input score, this study adopts the orthogonal projection method to segment the structure of the numbered musical notation image. The method analyzes the distribution of black and white pixels along the horizontal (X-axis) and vertical (Y-axis) directions to identify boundaries between measures and score lines. Horizontal projection along the X-axis is used to detect blank spaces between measures, while vertical projection along the Y-axis assists in separating notation rows. Segmenting the score measure by measure enhances the accuracy of note recognition and avoids misclassification across measure boundaries. To further improve segmentation performance, preprocessing steps such as binarization and spurious line removal are incorporated to eliminate decorative lines and noise symbols.

(1) Vertical Projection along the Y-axis

In this work, rows denote horizontal pixel lines and columns denote vertical pixel lines. For the vertical projection onto the Y-axis, the image is scanned row by row to compute the number of black pixels in each row, yielding a projection profile as a function of y. Pronounced minima in this profile typically correspond to vertical blank spaces between measure blocks in the score. In our system, the rows with the fewest black pixels are selected as cut points, and approximately four measures are extracted per segment (see Figure 17). This procedure suppresses boundary noise and facilitates batch recognition. For a binarized image with $p(x, y) = 1$ for black (symbols/lines) and $p(x, y) = 0$ for white (background), width w and height h, the row-wise black-pixel count is

$$v(y) = \sum_{x=1}^{w} p(x, y),\ y = 1 \ldots h \tag{14}$$

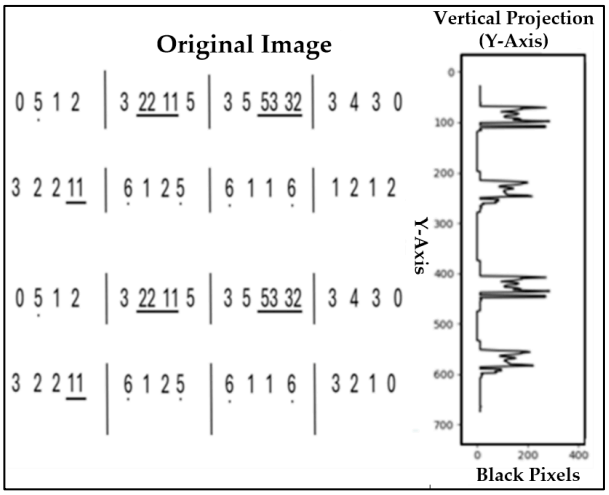


Figure 17. Vertical projection along the Y-axis.

(2) Horizontal Projection along the X-axis

The four-measure score images obtained after Y-axis segmentation are further processed by horizontal projection along the X-axis. In this step, the image is scanned column by column to calculate the distribution of black pixels in each column, producing a horizontal projection profile. Experimental results show that regions with a high density of black pixels typically correspond to vertical bar lines in the numbered musical notation. Accordingly, the system uses these peaks as segmentation references and divides the image into equally spaced sections according to the predefined number of measures. For instance, a four-measure image is evenly divided into four segments of equal width, as illustrated in Figure 18. This procedure facilitates beat localization and improves the accuracy of note recognition. The total number of black pixels in each column is calculated as expressed in Equation (15).

$$h(x) = \sum_{y=1}^{h} p(x, y),\ x = 1 \ldots w \tag{15}$$

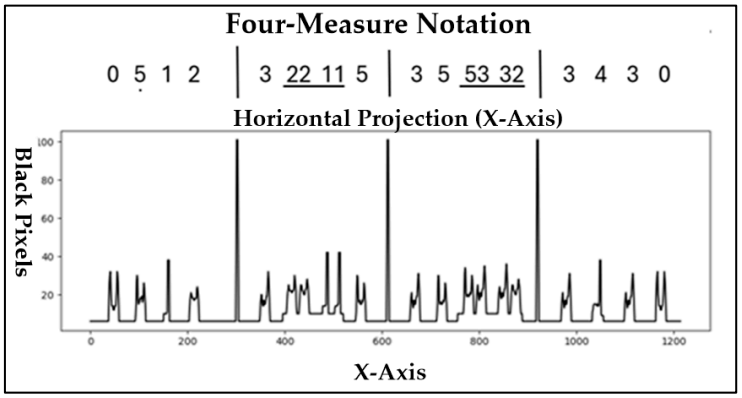


Figure 18. Horizontal projection along the X-axis.

3.3. Note Segmentation

After score preprocessing and structural partitioning through orthogonal projection, the next step is to segment individual notes from the extracted measures. Accurate note segmentation is critical, as it directly affects the performance of subsequent digit recognition. In this study, note segmentation is implemented using the pixel-scanning method, which analyzes the pixel distribution within each measure to isolate note symbols while effectively reducing interference from noise and auxiliary markings. After completing measure segmentation, the system specifically detects and analyzes three core features of the numbered musical notation: digit notes, pitch dots, and beat lines, as illustrated in Figure 19.

Syllable	Numbered Note	Note Name	Lower Octave	Middle Octave	Higher Octave
Do	1	C	C3	C4	C5
Re	2	D	D3	D4	D5
Mi	3	E	E3	E4	E5
Fa	4	F	F3	F4	F5
So	5	G	G3	G4	G5
La	6	A	A3	A4	A5
Si	7	B	B3	B4	B5

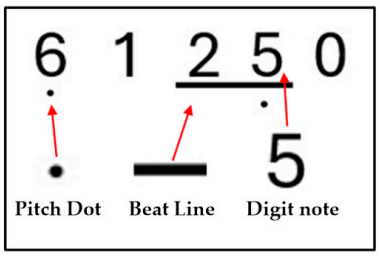


Figure 19. Mapping of digit notes, pitch dots, and beat lines.

(1) Note Detection

The system employs the pixel-scanning method to locate and extract digit notes in the numbered musical notation. The image is first scanned row by row, and when black pixels are detected, an extended scan is performed along both the X-axis and Y-axis to determine the width and height of the corresponding region. To exclude non-digit objects such as pitch dots and beat lines, an area threshold is applied, and only regions with black pixel counts exceeding this threshold are regarded as valid notes. Once validated, the system records the bounding box coordinates (x_1, y_1) and (x_2, y_2), which are subsequently used for pitch-dot and beat-line detection, as illustrated in Figure 19. To ensure accurate extraction of each digit block, the system adopts a scanning strategy from left to right and top to bottom, executing modularized steps sequentially. The overall workflow is depicted in the diagram shown in Figure 20.

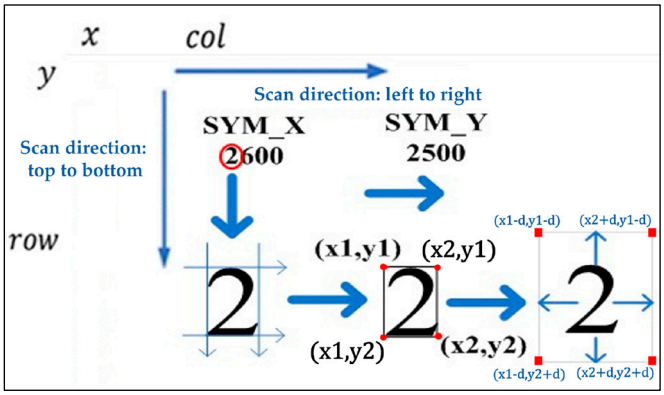


Figure 20. Illustration of the pixel-scanning method.

Pixel-Scanning Algorithm Workflow

Module A: Top Boundary Detection

Starting from the top of the image, the system scans row by row from left to right, analyzing the pixel distribution of each horizontal row. When the total number of black pixels in a given row exceeds a predefined threshold, that row is identified as the top boundary of a digit block.

Module B: Bottom Boundary Detection

Starting from the top boundary of the digit block, the system scans row by row from left to right, analyzing the horizontal pixel distribution. When the number of black pixels in a row falls below the threshold, it indicates that the scan has exited the digit region, and that row is identified as the bottom boundary.

Module C: Left Boundary Detection

Beginning from the left edge of the image, the system scans column by column within the range of the top and bottom boundaries, analyzing the vertical pixel distribution. When a column is found in which the density of black pixels exceeds the threshold, that column is designated as the left boundary of the digit block.

Module D: Right Boundary Detection

Starting from the left boundary of the digit block, the system continues scanning column by column to the right, monitoring the vertical pixel density. When the number of black pixels in a column drops below the threshold, it indicates that the scan has exited the digit block, and that column is identified as the right boundary.

Module E: Digit Block Extraction and Expansion

Based on the detected top, bottom, left, and right boundaries, the region of each digit block is defined. If the block is cropped exactly along the four boundary coordinates, the resulting image may be too tightly fitted, potentially affecting recognition accuracy. To address this, all four boundaries are expanded outward by a fixed number of pixels before extraction. The extracted image block is then stored in the corresponding data structure for subsequent neural network recognition.

Module F: Digit Removal

For each extracted block, the rectangular region from the top-left corner $(x1, y1)$ to the bottom-right corner $(x2, y2)$ is masked with a white rectangle, effectively removing the processed digit from the score image.

Module G: Iterative Detection

Modules A through F are repeated iteratively to detect and extract the next digit block, until the entire score image has been fully scanned and processed.

This modular pixel-scanning strategy ensures that each digit block is extracted with high reliability, providing clean input for subsequent pitch-dot and beat-line detection.

(2) Pitch Dot Detection

In numbered musical notation, high and low pitches are indicated by small dots placed above or below the digit symbols. Based on the previously detected digit block coordinates, the system expands the search region both upward and downward to locate potential dots. Contour detection, combined with geometric filtering, is then applied to identify and classify these dots, thereby determining the pitch attributes of the corresponding notes.

(a) Region Expansion and Preprocessing: For each detected digit block, two extended regions are defined: one above and one below the block. The upper extended region is obtained by extending upward from the top boundary y_1 by a fixed distance (e.g., 1.5 times the digit height), while the lower extended region is obtained by extending downward from the bottom boundary y_2 by the same distance. These two regions are extracted as new image blocks and then converted into grayscale and binarized forms in preparation for contour analysis.

(b) Contour Detection and Geometric Filtering: Within the extended regions, the cv2.findContours() function in OpenCV is applied to extract all closed contours. These contours are first filtered by area to exclude those that are too small or excessively large. Subsequently, additional geometric conditions, such as aspect ratio of the bounding box and circular fitting error, are applied to further refine the candidates. Only contours with diameters ranging from 3 to 8 pixels and shapes closely approximating a circle are retained.

(c) Pitch Classification: Each filtered circular contour is associated with its corresponding digit block. If the center of the circle lies within the upper extended region, it is labeled as a high-pitch dot; if it lies within the lower extended region, it is labeled as a low-pitch dot. Contours outside these regions are disregarded. To avoid duplicate detections or misclassifications, each digit is paired with only the nearest circle.

(d) Storage and Labeling: The category, position, and height information of each detected dot are stored in the data structure associated with the corresponding digit, such as note [k]. pitch = “high”, “low”, or “none”, for use in subsequent recognition or performance logic.

This dot detection strategy ensures reliable pitch annotation in numbered musical notation, thereby enhancing the accuracy of subsequent recognition and robotic performance.

(3) Beat Lines Detection

Beat lines are common rhythmic structures in numbered musical notation, typically located below the digit notes and appearing as horizontal lines with lengths close to the width of the corresponding digit. These lines indicate rhythm types (e.g., quarter notes, eighth notes). In this system, contour analysis is applied to detect horizontal lines beneath each digit block based on the following conditions: (i) the detection range is restricted to a designated region below the digit block; (ii) if the detected line length is approximately equal to the width of the digit block (around $x_2 - x_1$), it is regarded as a valid beat line; and (iii) once validated, the corresponding digit note is annotated with rhythm information. This module processes the lower region of each digit block sequentially, performing contour extraction and geometric filtering to identify and label the associated beat lines.

(a) Region Definition: For each extracted digit block with coordinates (x_1, y_1) to (x_2, y_2), the system defines a rectangular search region beneath the block. The vertical extent of this region is set to approximately 0.5–1.0 times the digit height (i.e., extending downward from y_2, ensuring that all potential beat line contours are included.

(b) Contour Detection: The defined region is first converted into grayscale and then binarized. The cv2.findContours() function in OpenCV is applied to extract both closed and open contour segments. For each detected contour, the corresponding bounding box is computed and represented as (x, y, w, h).

(c) Horizontal Line Filtering Criteria: Only contours with a very high aspect ratio ($\omega \gg h$, e.g., $\omega/h > 10$) are retained and considered horizontal line segments. In addition, the contour length ω must fall within the range of the corresponding digit width, satisfying $0.8(x_2 - x_1) < \omega < 1.2(x_2 - x_1)$ Furthermore, the horizontal position of the line must slightly overlap with the digit block to confirm its association as a constituent element of the note.

(d) Association and Label Storage: If a valid horizontal contour is detected beneath a digit block, the system annotates the block as containing beat line information, for example by setting note [k]. beat_line = True.

For further classification into quarter notes or eighth notes, the system can analyze the thickness, number, or relative position of the detected lines. Corresponding data structures such as measure_line or beat_annotation are then created to store the category and coordinate information of the beat line. The detected beat lines establish a direct mapping between digit notes and their rhythmic attributes, ensuring accurate rhythm annotation for subsequent recognition and robotic performance.

Although the current prototype focuses on basic numbered musical notation (digits, octave dots, and beat-duration lines), the recognition pipeline is modular and can be extended to additional symbols. Accidentals (♯/♭), slurs, double-octave markers, and vertically aligned multi-note chords can be incorporated by augmenting the CNN classifier with new symbol categories or by adding lightweight rule-based shape and position analysis modules.

3.4. Digit Recognition

After completing the segmentation of digit notes, the system proceeds to the digit recognition stage. To identify the most suitable classification model, this study first compared two common architectures: the Multi-Layer Perceptron (MLP) and the Convolutional Neural Network (CNN). Both models were trained and evaluated for performance. The experimental results demonstrated that CNNs significantly outperformed MLPs in terms of recognition accuracy and stability. Unlike MLPs, CNNs effectively preserve the spatial structure of images, leveraging multilayer abstraction to capture local variations and shape features, while parameter sharing reduces model complexity and mitigates overfitting risks. These properties confer superior generalization ability and overall recognition performance. Consequently, CNN was adopted as the core architecture for digit recognition in this study.

In the practical implementation, the CNN architecture adopted in this study consists of two convolutional layers (Conv2D) paired with corresponding pooling layers (MaxPooling2D) to progressively extract local and higher-level image features. The first convolutional layer employs 16 filters for basic feature extraction, followed by a pooling layer for dimensionality reduction. The second convolutional layer applies 36 filters for more advanced feature extraction, also accompanied by pooling. To prevent overfitting, a dropout layer is introduced at this stage to randomly deactivate a portion of neurons. The extracted feature maps are then flattened into a vector through a flatten layer and passed to a hidden layer containing 128 neurons, which performs feature integration and prepares for

classification. Finally, the output layer uses a softmax activation function to classify digits into ten categories. The overall model comprises 242,062 trainable parameters, achieving stable and highly accurate recognition under limited computational resources. The architecture of the CNN model is illustrated in Figure 21.

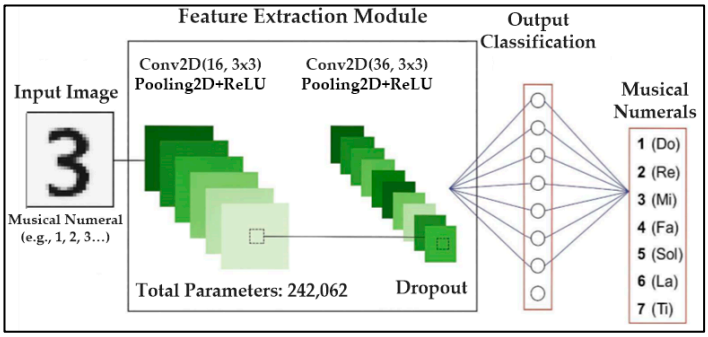


Figure 21. Architecture of the CNN model.

In the initial training phase, this study employed both the MNIST handwritten digit dataset (60,000 images) and a self-constructed dataset of 4580 computer font digit images, described as follows:

(1) MNIST handwritten digit dataset: MNIST (Modified National Institute of Standards and Technology) is a widely used benchmark dataset for handwritten digit recognition, containing grayscale images of Arabic digits from 0 to 9 (see Figure 22). Each image has a size of 28 × 28 pixels and is preprocessed for alignment and centering. The dataset consists of 60,000 training images and 10,000 testing images, featuring a wide variety of handwriting styles. It is commonly used to evaluate the performance and generalization ability of classification algorithms and serves as a standard example in deep learning research.

(2) CFDD custom computer font digit dataset: The CFDD (Custom Computer Font Digit Dataset) was constructed in this study using 458 different computer font styles, including serif, sans serif, monospace, geometric, and decorative fonts. Each font set generates digit images from 0 to 9, resulting in a total of 4580 images (see Figure 23). This dataset provides high font diversity, enhancing the model's ability to recognize digits across varying font styles.

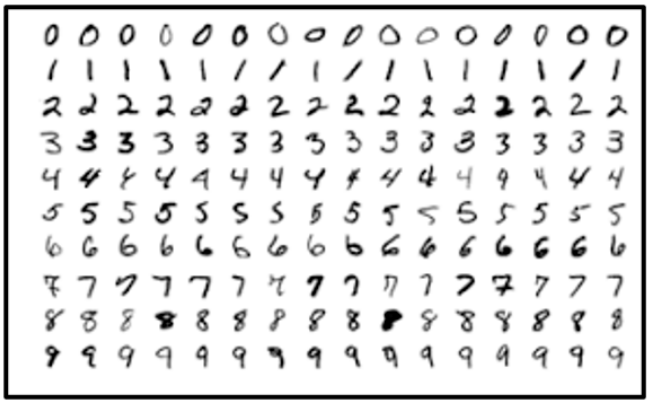


Figure 22. MNIST digit dataset.

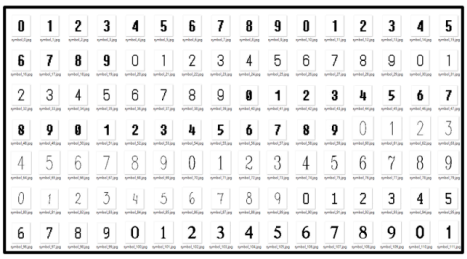


Figure 23. CFDD custom digit dataset.

The training data for the proposed model were derived from two sources: 60,000 handwritten digit images from MNIST and 4580 computer-font digit images from the custom CFDD. To investigate the effect of different training combinations on digit recognition performance, four configurations were designed for comparison:

(a) Combination 1: MNIST only (all 60,000 samples)
(b) Combination 2: CFDD only (all 4580 samples)
(c) Combination 3: MNIST (first 30,000 samples) + CFDD (4580 samples)
(d) Combination 4: MNIST (first 20,000 samples) + CFDD (4580 samples)

The MNIST subsets were extracted sequentially without randomization. Although the digit distribution is slightly imbalanced, the samples remain representative and suitable for effective training. After preprocessing, the datasets were input into the CNN model for training, with 20% reserved as a validation set to monitor convergence and prevent overfitting. Upon completion of training, 49 manually annotated digit images extracted from actual numbered musical notation were used as the test set for model evaluation. The results indicated that Combination 4 (20,000 MNIST samples + 4580 CFDD samples) achieved the best performance (see Figure 24), with detailed results summarized in Tables 2 and 3. Note that Figure 24 illustrates digit extraction from an actual numbered musical notation example and is not derived from the CFDD. This outcome suggests that, since musical notation images are often captured from printed fonts subject to lighting and geometric distortions resembling handwriting, incorporating handwritten samples improves accuracy and generalization under non-ideal conditions.

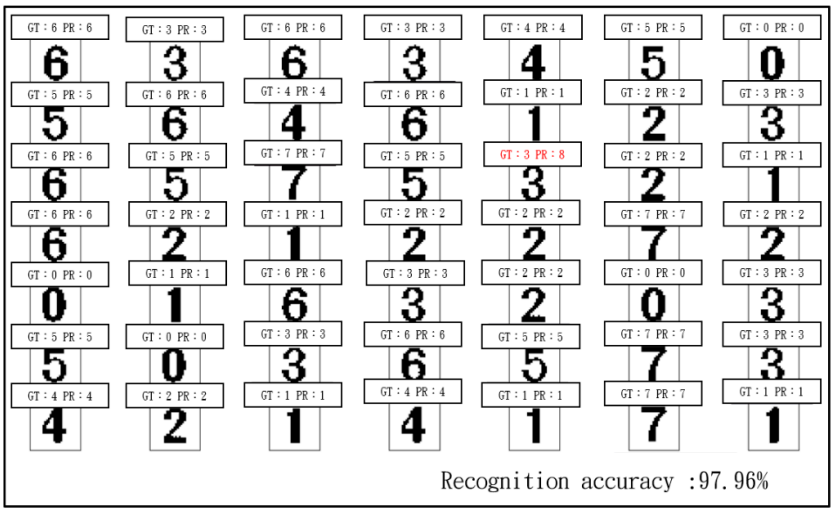


Figure 24. Recognition results.

Table 2. Recognition accuracy of MLP based on four training set combinations.

Training Set Combinations	(1)	(2)	(3)	(4)
Recognition Accuracy	79.6%	81.6%	87.8%	91.8%

Table 3. Recognition accuracy of CNN based on four training set combinations.

Training Set Combinations	(1)	(2)	(3)	(4)
Recognition Accuracy	91.8%	93.9%	95.9%	98.0%

3.5. Note Naming Convention

After digit recognition is completed, the system stores each note image file using the format music_sequence_pitch_duration.jpg. Here, sequence indicates the order in which the note appears in the musical piece, ensuring that playback follows the correct sequence. For example, the note $\underline{\dot{6}}$ represents the first note, corresponding to the high-pitch La (A5) with a duration of 0.25 beats. Its filename is therefore saved as music_1_A5_0.25.jpg. This filename is transmitted via serial communication to the ESP32 embedded microcontroller, which parses the sequence, pitch, and duration information from the string and subsequently triggers the appropriate finger and linear rail control signals. As a result, the robotic hand presses the correct piano key and executes the specified rhythmic duration.

The proposed recognition framework provides a complete pipeline from score image acquisition to performance-ready symbolic representation, forming the foundation for the robotic piano performance system described in the following sections.

4. Experimental Results

The experimental results are presented in two stages. The first stage focuses on the software results, including the display of the graphical user interface (GUI) and the process of numbered musical notation recognition. The second stage presents the hardware results, covering the robotic hand, linear rail, and driver circuit board.

4.1. GUI Functionality

The graphical user interface (GUI) of the system integrates the functions of score recognition and performance, allowing users to operate step by step through the workflow (Figure 25). The functionality of each button is described as follows:

(a) CAMA (Open Camera): Activates the camera for real-time image acquisition in preparation for capturing the numbered musical notation.
(b) CAPT (Capture Score): Captures an image of the score and saves it as a file, avoiding the need to re-capture for each recognition task.
(c) PROC (Score Preprocessing): Processes the captured image, including four-point detection and perspective transformation to correct skew. The color image is then converted into grayscale to simplify data and eliminate color interference, followed by binarization using either fixed or adaptive thresholding to enhance contrast between digits and background. Finally, orthogonal projection is applied to analyze horizontal and vertical pixel distributions, accurately segmenting measures and score lines for subsequent digit localization and analysis.
(d) SEGM (Note Segmentation): Applies pixel scanning and orthogonal projection to automatically segment note blocks, while detecting pitch dots and beat lines. The coordinates of each block are recorded for recognition and control.
(e) RECO (Digit Recognition): Classifies the segmented digit images using the CNN model, outputting the corresponding note values.

(f) MIDI (MIDI Conversion): Converts recognition results into standard MIDI encoding and generates .mid files.
(g) SEND (Serial Communication): Opens the communication channel with the ESP32 to transmit note names, rhythm, and key position information.
(h) PLAY (Start Performance): Sends note data to the ESP32, which drives the robotic hand to play the corresponding piano keys.
(i) STOP (End Performance): Stops the robotic hand operation and terminates control commands.
(j) EXIT (Exit Program): Closes the system and releases resources.

Figure 25 shows the graphical user interface (GUI) designed in this study. This subsection concludes with an overview of the functional buttons integrated into the interface.

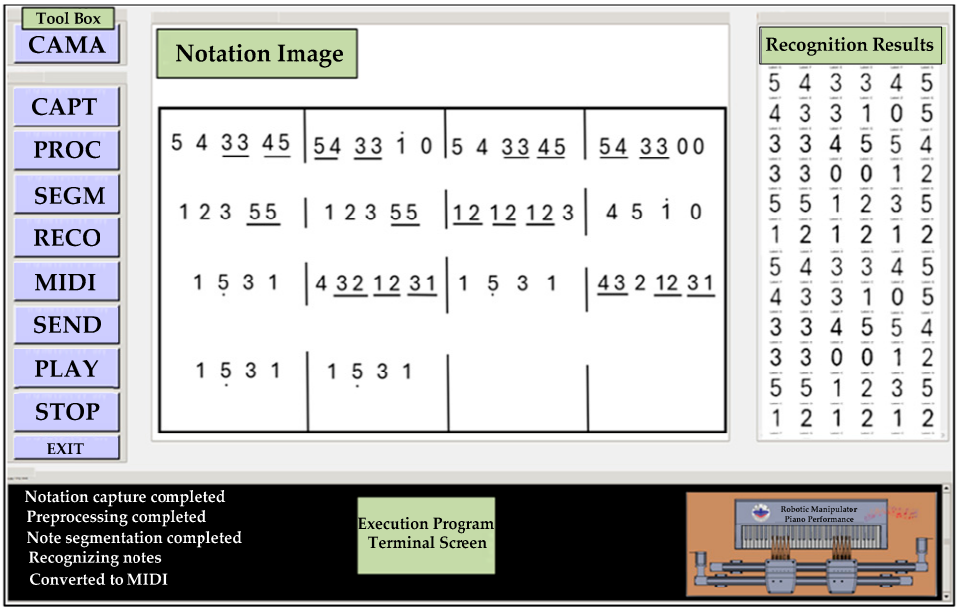


Figure 25. Graphical user interface (GUI) of the proposed system.

4.2. Score Image Recognition

Step 1: The score is placed in the designated position, after which the camera is activated by pressing the Open Camera button. The user then aligns the camera with the score image and clicks the Capture Score button. The captured result is displayed in the window, as shown in Figure 25.

Step 2: By clicking the Score Preprocessing button, the system applies four-point detection and corner ordering to determine the coordinates of the quadrilateral corners:

$$P_1 = (153,\ 4),\ P_2 = (2734,\ 4),\ P_3 = (2880,\ 1472),\ P_4 = (7,\ 1472).$$

To transform the skewed or distorted region into a front-facing rectangular structure while preserving image fidelity, the target rectangle width and height are set to 2874 and 1468, respectively. The mapped coordinates are defined as:

$$P_1' = (0,\ 0),\ P_2' = (2874,\ 0),\ P_3' = (2874,\ 1468), P_4' = (0,\ 1468).$$

The perspective transformation matrix $M \in R^{3\times3}$ is introduced to map the four known points $P_i = (x_i,\ y_i)$ in the original image to the corresponding points $P_i' = (x_i',\ y_i')$, $i = 1,\ 2,\ 3,\ 4$ on the target plane. By applying Equation (9), a system of eight linear equations with nine unknowns is obtained, forming a non-square matrix $\boldsymbol{A} \in R^{8\times9}$. Singular value decomposition (SVD) is then applied to decompose $\boldsymbol{A} = \boldsymbol{U\Sigma V^T}$, solving the homoge-

neous linear system $\boldsymbol{Ah} = 0$ to obtain the transformation matrix M. The transformed image is shown in Figure 26b.

$$M = \begin{bmatrix} 1.11 & 0.11 & -50.16 \\ 0 & 1.11 & -1.11 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

Step 3: The transformed image is then converted to grayscale and binarized, as illustrated in Figure 26c,d.

These preprocessing steps establish a rectified and noise-free score image, providing a reliable foundation for subsequent segmentation and recognition.

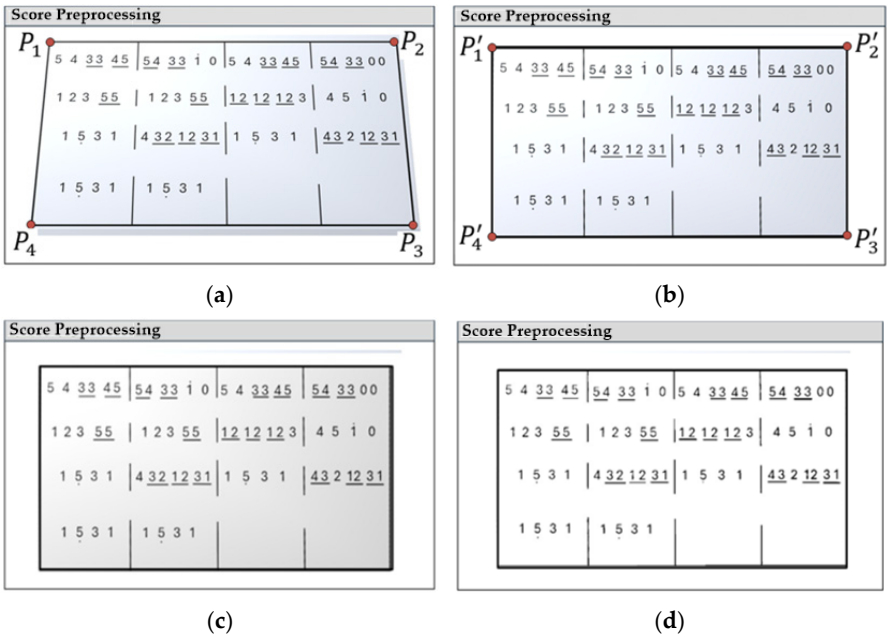


Figure 26. Experimental procedure: (**a**) original score, (**b**) four-point detection, (**c**) grayscale conversion, (**d**) binarization.

Step 4: After clicking the Note Segmentation button, the system begins processing the score image, which includes both measure segmentation and note segmentation.

For measure segmentation, the orthogonal projection method is applied to obtain the pixel distribution. Specifically, vertical projection along the Y-axis is performed, where the cut points are determined by analyzing the positions with the lowest number of black pixels in the projection profile. Approximately four measures of notation are extracted at a time, generating a segmented row image. An example of such vertical segmentation is highlighted by the green rectangular box in Figure 27. The results of vertical segmentation are listed below, showing the Y-axis coordinates of the start and end positions for each row block. The vertical segmentation results are summarized in Table 4, where y_i^s and y_i^e denote the start and end coordinates of the i-th row block along the Y-axis.

Table 4. The Y-axis coordinates of vertical segmentation.

Row Block	Start Coordinate (y_i^s)	End Coordinate (y_i^e)
1st row	$y_1^s = 0$	$y_1^e = 238$
2nd row	$y_2^s = 354$	$y_2^e = 562$
3rd row	$y_3^s = 659$	$y_3^e = 916$
4th row	$y_4^s = 1083$	$y_4^e = 1439$

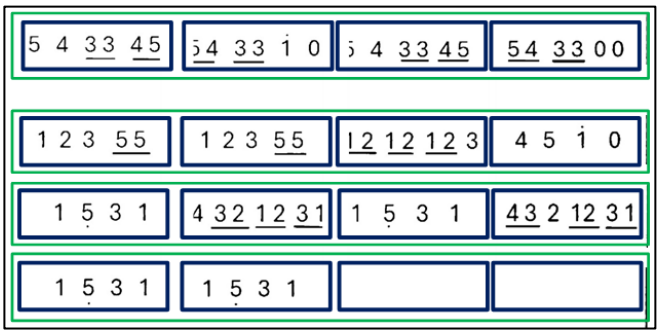


Figure 27. Measure segmentation.

Next, the system performs horizontal segmentation along the X-axis on the previously extracted four-measure image. By analyzing the horizontal projection curve of pixel density, regions with high pixel density are identified as the basis for segmentation. As illustrated by the green rectangular box in Figure 27, the image contains four measures; therefore, it is evenly divided into four equal-width segments corresponding to the respective measure regions. The resulting X-axis coordinates of each segmented measure are summarized in Table 5.

Table 5. The X-axis coordinates of horizontal segmentation.

Measure Block	Start Coordinate (x_i^s)	End Coordinate (x_i^e)
1st measure	$x_1^s = 0$	$x_1^e = 625$
2nd measure	$x_2^s = 625$	$x_2^e = 1250$
3rd measure	$x_3^s = 1250$	$x_3^e = 1875$
4th measure	$x_4^s = 1875$	$x_4^e = 2500$

After completing measure segmentation, the system proceeds to note segmentation, in which three core features of the numbered musical notation are detected and analyzed: digit notes, pitch dots, and beat lines. In the experiments, the pixel-scanning method was applied to locate and extract digit notes, while contour detection was used to identify circular objects. If a detected dot was located above a digit, it was classified as a high-pitch dot; if it was located below, it was classified as a low-pitch dot. In addition, contour analysis was employed to detect horizontal line objects beneath the digits, and those satisfying the filtering conditions for horizontal segments were identified as beat lines.

Step 5: By clicking the Digit Recognition button, the system initiates the recognition of digit notes using the pre-trained CNN model. The recognition results are shown in Figure 28.

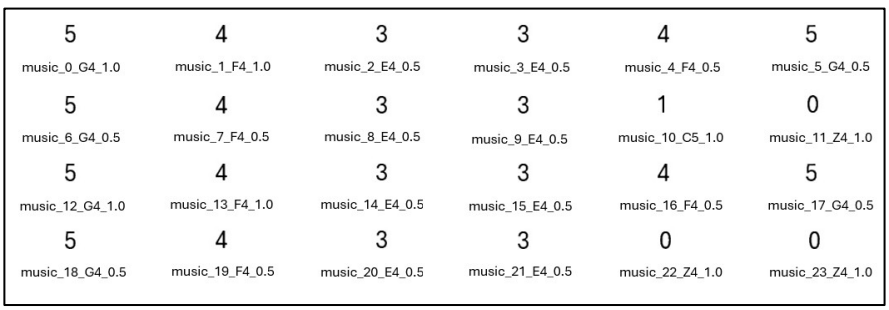


Figure 28. Recognition results of musical notes with naming rules.

Step 6: After recognition is completed, clicking the Convert to MIDI button generates a MIDI file, allowing the user to verify whether the recognition results match the expected performance through auditory playback.

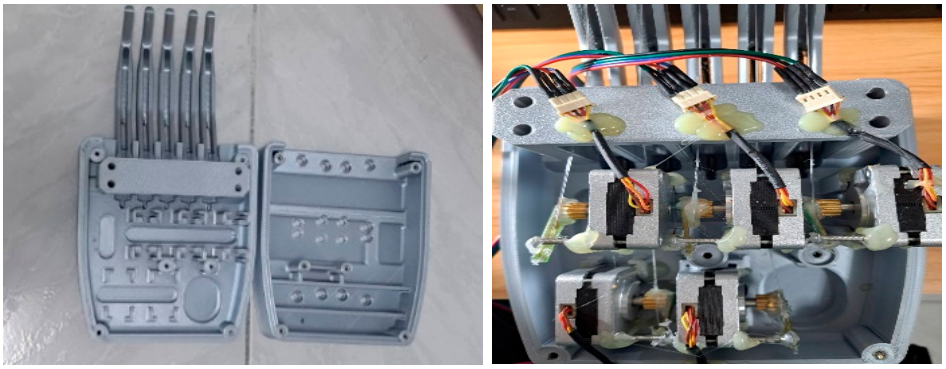
Step 7: By pressing the Serial Transmission button, the recognition results are transmitted to the ESP32 embedded microcontroller. During the execution of each functional button, the console window displays the corresponding operation currently being performed.

Several limitations and issues were identified during the experiments. First, in the score capture stage, the selected style of numbered musical notation had a critical impact on recognition accuracy. If the score contained lyric text, the system often misclassified it as part of the notes, leading to segmentation and classification errors. This limitation arises because the image processing method adopted in this study extracts note blocks based on black pixel distribution, which cannot effectively distinguish between lyrics and notes. Consequently, the system is currently applicable only to custom scores without lyrics. Second, the digits in numbered notation are relatively small in size, and variations in lighting or camera angle during image capture may introduce noise and shadows during grayscale conversion and binarization. Such artifacts severely interfere with subsequent image processing and recognition accuracy. To clarify the reliability of long-duration performances, we note that erroneous keystrokes are almost exclusively caused by misclassified digits during the recognition stage. When the linear guide moves smoothly without mechanical obstruction, the robot's playing accuracy remains consistent across the entire performance, and the number of incorrect notes closely matches the recognition errors without cumulative drift.

4.3. Hardware Integration Test

(1) Installation of the Robotic Hand Mechanism

The hardware design is divided into three main modules: (a) robotic hand structure, (b) stepper motor control (including both finger actuation and linear rail movement), and (c) implementation of the finger-striking mechanism. The hand components were fabricated using 3D printing; since the design was not monolithic, each part was printed separately and then assembled (Figure 29a). For motor control, two types of stepper motors were utilized: one for actuating finger strikes, driven by the A4988 controller, and the other for linear rail movement, driven by the DM556 driver. The most time-consuming stage during implementation involved fine-tuning motor parameters to ensure stable and reliable key-striking performance.



(**a**) (**b**)

Figure 29. (**a**) Palm covers (front and back) with finger assembly; (**b**) installation of stepper motors.

After 3D printing was completed, the robotic hand was first assembled, and the stepper motors were installed into the predesigned slots. To prevent loosening, each motor was secured at the four corners with hot-melt adhesive. Subsequently, the custom driver circuit and ESP32 control board were integrated, followed by staged testing: first verifying the functionality of the stepper motors driving the fingers, and then testing the linear rail movement. Once both subsystems demonstrated stable operation, the motor and finger modules were connected. For the transmission lines, enameled wire was initially employed

but was found to lack tensile strength and broke easily. It was eventually replaced with durable fishing line, which significantly improved reliability. To ensure smooth finger striking and rebound, repeated adjustments of the spring and fishing line lengths were required to determine the optimal combination of tension and transmission distance. Due to slight variations in motor installation positions, the transmission mechanisms also required individual fine-tuning. Figure 29b illustrates the integration of the stepper motor into the robotic hand structure.

After completing the installation of all components of the hand module, the upper and lower covers were assembled and secured. Since the hand structure was designed to mimic the appearance of a human hand, it had to be mounted on the linear rail at a slightly inclined angle. If the hand were positioned too vertically, the fingers would fail to make precise contact with the piano keys during downward strikes, thereby reducing striking accuracy and overall performance quality. In the testing of the circuit and actuation modules, the system was verified in stages according to functional modules, including single-finger motor testing, five-finger coordinated motor testing, linear rail movement control testing, and overall system integration testing. Each stage focused on validating functional correctness and parameter configurations, ensuring that all modules could operate stably in coordination to support the complete note performance process.

(2) Single-Finger Motor Test

The first experiment involved functional testing of the stepper motor driving a single finger. To ensure that the motor provided sufficient torque to actuate the finger mechanism, the required drive current was measured and calculated, followed by adjusting the reference voltage V_{ref} on the A4988 driver module via its variable resistor. If the test results indicated that the motor could not reliably drive the finger, the current setting was increased accordingly to achieve stable operation. Based on Equation (4), when the reference voltage V_{ref} of the A4988 driver was set to 1.94 V, the corresponding maximum output current was approximately 2.43 A. Considering that the stepper motor used in this system has a rated current of 1.7 A and is driven at 12 V DC, the rated power can be estimated at about 20.4 W. However, experimental results showed that although the A4988 could stably drive the stepper motor with precise motion, prolonged operation caused the aluminum heatsink to heat rapidly to a level that was uncomfortably hot to the touch, potentially compromising system stability and component longevity.

To address this overheating issue, the output current setting was reduced by fine-tuning the module's variable resistor, lowering V_{ref} from 1.94 V to 0.85 V. The corresponding current limit was calculated using Equation (3).

$$I_{max} = \frac{V_{ref}}{8 \times R_s} = \frac{0.85}{8 \times 0.1} = 1.0625 \text{ A} \tag{17}$$

Under this configuration, the driving current of the stepper motor was estimated to be approximately 0.8 A, which is about half of its original rated current of 1.7 A. Despite the significant reduction in output current, experimental results demonstrated that the motor continued to operate stably without observable missed steps. At the same time, the driver chip temperature was markedly reduced, indicating improved heat dissipation. These results confirm that by moderately lowering the current limit of the A4988, the system's thermal management performance can be effectively enhanced and component lifespan prolonged, without compromising operational reliability. Therefore, this adjustment was adopted as the default configuration for subsequent experiments.

(3) Five-Finger Coordinated Motor Test

Following the verification of single-finger operation, tests were conducted on the simultaneous actuation of all five fingers. Due to variations in motor installation positions and geometric differences in the transmission mechanisms, slight discrepancies in output torque were observed. To ensure consistent striking performance across all fingers, the drive current settings and tension of the transmission lines were individually adjusted for each finger. These calibrations compensated for mechanical deviations and ensured coordinated and stable execution of the playing actions. This ensured that the robotic hand could achieve reliable multi-finger coordination during performance.

(4) Linear Rail Motion Control Test

After mounting the assembled hand module onto the rail slider, horizontal sliding tests along the keyboard direction were performed to verify whether the overall structure could move smoothly and stably without jamming or excessive resistance. The test items included checking the continuity of bidirectional travel and evaluating positioning accuracy. In the initial tests, noticeable stuttering and vibration were observed when the hand moved across specific rail segments. Inspection suggested that excessive belt tension was the primary cause, as the motor torque was insufficient to overcome the slider resistance, thereby affecting motion smoothness. By moderately loosening the timing belt tension, this issue was successfully resolved, restoring smooth operation of the rail. The results of this stage confirmed that, after adjusting belt tension and fine-tuning the mechanical fixtures, the rail system achieved stable horizontal displacement, providing a reliable foundation for accurate hand positioning and cross-key performance.

(5) System Integration Test

After confirming that each module functioned correctly in individual tests, a full system integration test was conducted by performing an actual piece of numbered musical notation. This experiment was designed to validate mechanical coordination and overall performance. Several key indicators were closely monitored, including the accuracy of rail positioning, the presence of missed steps or synchronization delays in the stepper motors, and the stability and continuity of finger strikes. If anomalies such as slider jamming, insufficient torque, or striking misalignment were observed, adjustments were made according to the results of the unit tests, including reconfiguring drive currents, fine-tuning line tension, or correcting belt tension parameters. Since the system was manually designed and assembled, frequent adjustments for mechanical tolerances and assembly errors were necessary during the integration stage to ensure smooth coordination across modules.

The integration environment consisted of two linear rails, two robotic hands, two driver circuit boards, and two DM556 stepper motor drivers, combined with a digital piano to form a complete performance platform. The integration test demonstrated that the modules operated in coordinated fashion, accurately executing the performance commands in sequence, thereby confirming the feasibility and practicality of the proposed system for automatic numbered musical notation performance (Figure 30). With the hardware modules successfully integrated and validated, the system was confirmed to be capable of reliable automatic performance, thus establishing the foundation for subsequent evaluation and discussion.

The overall timing behavior of each keystroke is primarily determined by the downward travel of the fingertip and the spring-assisted return motion. The downward phase includes both the control-signal propagation through the ESP32–A4988 driver and the subsequent mechanical actuation of the finger, while the upward phase is accelerated by the added return spring, which shortens the recovery time compared with purely

motor-driven bidirectional actuation. Within a local playing region where no horizontal repositioning of the right-hand mechanism is required, these timing characteristics allow the prototype to sustain practical musical tempos. When a passage requires movement across distant keys, additional delay is introduced by the linear slide repositioning, which represents the primary mechanical constraint on global tempo. This limitation stems from the lightweight, low-cost linear guide used in the prototype and will be addressed in future hardware iterations.



Figure 30. Assembled robotic manipulator for piano performance.

5. Conclusions and Directions for Future Research

This study successfully developed a robotic system capable of numbered musical notation recognition and piano performance. The system integrates computer vision, deep learning, linear rail mechanisms, and circuit control to achieve automatic piano playing. Specifically, musical scores are captured by a camera, processed through image preprocessing, and recognized using a convolutional neural network (CNN). The recognized digit notes are then converted into performance commands that control the robotic hand to strike the piano keys, thereby accomplishing fundamental piano performance. In terms of image processing, the system applies four-point detection for perspective correction, combined with orthogonal projection and pixel-scanning methods to segment note regions, while simultaneously detecting pitch dots and beat lines to enhance recognition efficiency and accuracy. For digit recognition, multiple training datasets were compared, and the model trained with 20,000 MNIST samples combined with 4580 computer font images was selected. This configuration achieved a recognition accuracy of 98% on real score images, demonstrating strong generalization capability. On the hardware side, the robotic hand was fabricated using 3D-printed modular components embedded with stepper motors and spring mechanisms. Coordinated control of the five fingers and linear rail was accomplished through a dual-driver configuration of A4988 and DM556. At the control level, an ESP32 served as the core processor, integrated with a custom-designed circuit board to deliver high-precision signal outputs.

The main contribution of this work lies in its dedicated design for numbered musical notation, which is widely used in East Asian music education but seldom addressed in robotic performance research. The system demonstrates the feasibility of bridging symbolic notation recognition and physical execution, providing a low-cost, modular, and extensible platform for music robotics. Although the performance refinement and stability remain below the level of commercial products or advanced prototypes, the results

confirm fundamental capabilities in automatic playing and note recognition, highlighting the potential of developing functional robotic musicians under limited resources. Despite these achievements, the current prototype still exhibits several limitations. First, the system only supports basic numbered musical notation and does not yet handle advanced musical symbols or wide two-octave transitions. Second, the horizontal slide mechanism and the weight of the 3D-printed hand restrict the speed and smoothness of large-range movements across the keyboard. In addition, the present actuation mechanism produces a fixed striking force, preventing expressive dynamic control. These limitations stem primarily from the prototype-level mechanical design and will be addressed in future system revisions.

Future research will focus on addressing current limitations, such as misclassification of embedded lyrics and sensitivity to lighting variations, while enhancing mechanical precision, playing speed, and recognition robustness. Additional directions include AI-based path optimization, multi-finger cooperative control, sustain pedal integration, and support for polyphonic or ensemble performance. These advancements will further improve the practicality and expressiveness of the robotic pianist, expanding its applications in artistic creation, education, and entertainment.

Author Contributions: Conceptualization, P.-S.T.; Methodology, P.-S.T.; Software, C.-T.L.; Validation, P.-S.T.; Formal Analysis, P.-S.T.; Investigation, P.-S.T. and C.-T.L.; Resources, P.-S.T.; Data Curation, C.-T.L.; Writing—Original Draft Preparation, P.-S.T.; Writing—Review and Editing, T.-F.W.; Visualization, C.-T.L.; Supervision, T.-F.W.; Project Administration, P.-S.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Science and Technology Council (NSTC) (https://www.nstc.gov.tw), grant number NSTC 114-2221-E-197-017.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Acknowledgments: The authors would like to thank the National Science and Technology Council (NSTC) of Taiwan, Republic of China, for financially supporting this research (Contract No. NSTC 114-2221-E-197-017). (https://www.most.gov.tw).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Roehl, H. *Player Piano Treasury*; Vestal Press: Vestal, NY, USA, 1973.
2. Kato, I.; Ohteru, S.; Shirai, K.; Matsushima, T.; Narita, S.; Sugano, S.; Kobayashi, T.; Fujisawa, E. The robot musician WABOT-2 (Waseda Robot-2). *Robotics* **1987**, *3*, 143–155. [CrossRef]
3. Sugano, S.; Kato, I. WABOT-2: Autonomous robot with dexterous finger-arm—Finger-arm coordination control in keyboard performance. In Proceedings of the 1987 IEEE International Conference on Robotics and Automation (ICRA), Raleigh, NC, USA, 31 March–3 April 1987; pp. 90–97.
4. Zhang, D.; Lei, J.; Li, B.; Lau, D.; Cameron, C. Design and analysis of a piano playing robot. In Proceedings of the 2009 International Conference on Information and Automation (ICIA), Zhuhai, China, 22–24 June 2009; pp. 757–761.
5. Jaju, V.; Sukhpal, A.; Shinde, P.; Shroff, A.; Patankar, A.B. Piano playing robot. In Proceedings of the 2016 International Conference on Internet of Things and Applications (IOTA), Pune, India, 22–24 January 2016; pp. 211–215.
6. Fahn, C.-S.; Tsai, C.-F.; Lin, Y.-W. Development of a novel two-hand playing piano robot. In Proceedings of the 4th International Conference on Control, Dynamic Systems, and Robotics (CDSR'17), Toronto, ON, Canada, 21–23 August 2017.
7. Palmieri, S.; Di Noia, M.; Bisson, M.; Ianniello, A. Human to artificial (H2A): From duets with robot to a new model of relationship. In Proceedings of the Thirteenth International Conference on Creative Content Technologies (CONTENT 2021), Porto, Portugal, 18–22 July 2021; pp. 1–5.
8. Vear, C. Creative AI and musicking robots. *Front. Robot. AI* **2021**, *8*, 631752. [CrossRef] [PubMed]
9. Gattal, Y.; Chibani, Y. Segmentation and recognition strategy of handwritten connected digits based on the oriented sliding window. In Proceedings of the 2012 International Conference on Frontiers in Handwriting Recognition (ICFHR), Bari, Italy, 18–20 September 2012; pp. 297–301.

10. Gattal, Y.; Chibani, Y.; Hadjadji, B. Segmentation and recognition system for unknown-length handwritten digit strings. *Pattern Anal. Appl.* **2017**, *20*, 307–323. [CrossRef]
11. Chen, C.; Guo, J. A general approach for handwritten digits segmentation using spectral clustering. In Proceedings of the 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Kyoto, Japan, 9–15 November 2017; pp. 547–552.
12. Ribas, F.C.; Oliveira, L.S.; Britto, A.S., Jr.; Sabourin, R. Handwritten digit segmentation: A comparative study. *Int. J. Doc. Anal. Recogn.* **2013**, *16*, 127–137. [CrossRef]
13. Hochuli, A.G.; Britto, A.S., Jr.; de Oliveira, L.E.S.; Sabourin, R. Handwritten digit segmentation: Is it still necessary? *Pattern Recognit.* **2018**, *78*, 1–11. [CrossRef]
14. Minsky, M.; Papert, S. *Perceptrons: An Introduction to Computational Geometry*; MIT Press: Cambridge, MA, USA, 1969.
15. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536. [CrossRef]
16. Hecht-Nielsen, R. Theory of the backpropagation neural network. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Washington, DC, USA, 18–21 June 1989; pp. 593–605.
17. LeCun, Y.; Boser, B.; Denker, J.S.; Henderson, D.; Howard, R.E.; Hubbard, W.; Jackel, L.D. Handwritten digit recognition with a back-propagation network. In Proceedings of the Advances in Neural Information Processing Systems (NIPS 1989), Denver, CO, USA, 27–30 November 1989; pp. 396–404.
18. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proc. IEEE* **1998**, *86*, 2278–2324. [CrossRef]
19. Hinton, G.E.; Salakhutdinov, R.R. Reducing the dimensionality of data with neural networks. *Science* **2006**, *313*, 504–507. [CrossRef] [PubMed]
20. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet classification with deep convolutional neural networks. In Proceedings of the Advances in Neural Information Processing Systems (NIPS 2012), Lake Tahoe, NV, USA, 3–6 December 2012; pp. 1097–1105.
21. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
22. Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Rabinovich, A. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2015), Boston, MA, USA, 7–12 June 2015; pp. 1–9.
23. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2016), Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
24. Wang, C.-H. Cloud Controls the Playing Robot. Master's Thesis, China University of Science and Technology, Taipei, Taiwan, 2019.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.