Github Repository: https://github.com/gunwoo0623/3-Axis-CNC-Machinery

Comprehensive Overview of 3-Axis CNC Machinery Integration

Student Name: Geonwoo Cho, Jennifer Hengky Liandy, Hang Yuan

Published Date: 4 July 2024

Description

Explore the integrated control system of a 3-axis CNC machine utilizing Arduino and Unity. This system employs Arduino MEGA for motor control and Unity for real-time simulation through digital twin technology. Commands generated from Unity's user interface are translated into M-Code and G-Code, directing the CNC machine via Arduino. The process involves precise motor control using PWM signals and incorporates limit switch feedback for positional accuracy. The digital twin in Unity mirrors real-world operations, enhancing visualization and control of the CNC machine.

Key Features

- Detailed exploration of Arduino programming for CNC machine control.
- Unity-based simulation for real-time visualization and digital twin creation.
- Implementation of M-Code and G-Code interpretation to execute complex machining operations.
- Integration of limit switch feedback to enhance positional accuracy and safety.
- Practical examples and code snippets for setting up and operating the CNC machine through Arduino IDE.
- Comprehensive insights into firmware development for seamless communication between Arduino and Unity.

Note: This document contains excerpts from second group assignment submitted for MEC307 - Mechatronic System Development module at Xi'an Jiaotong-Liverpool University (XJTLU).

Github Repository: https://github.com/gunwoo0623/3-Axis-CNC-Machinery

1 Operation of the Control System

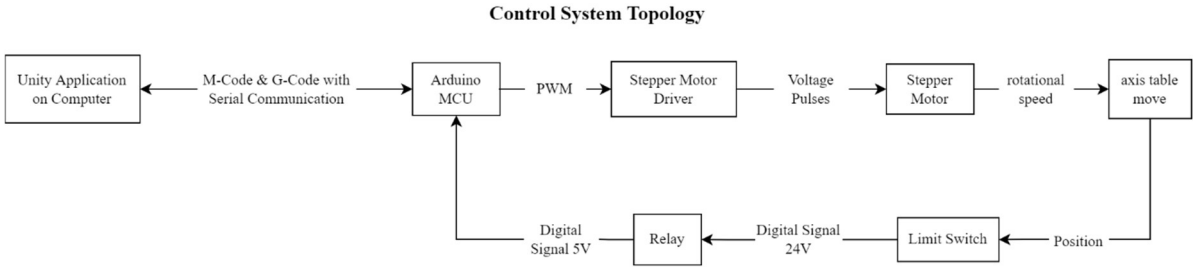

Figure 1. *The control system topology of the 3-axis CNC machine.*

The control system of the machine involves two controllers which are Arduino Environment with Arduino MEGA based and Unity with computer based. From the user input obtained in the Unity Application, the corresponding M-Code and G-Code are generated to communicate with the Arduino MCU. Upon decoding the command, the MCU transmitted appropriate PWM signals to the direction and pulse terminal of the stepper motor driver. Then, the stepper motor produces a sequence of zero-order hold signals to power the stepper motor to produce corresponding rotational speed and torque. Then, if the limit switch is triggered, the signal is sent to the relay for voltage conversion and sent to the Arduino MCU. During the process, the Arduino constantly sends the signal to Unity of the current position obtained through integration of the table velocity. Then, the positions received are visualized through the digital twin model.

2 Control Programme

Arduino Program

The machine in the lab that closely resembles the engraving machine is the 2D plotter shown in Fig. 2, sharing stroke dimensions of 250 mm × 250 mm × 100 mm. Therefore, for the purposes of operational demonstration within this report, emphasis will be placed on the 2D plotter, leveraging its functionalities that align with those of engraving machines.

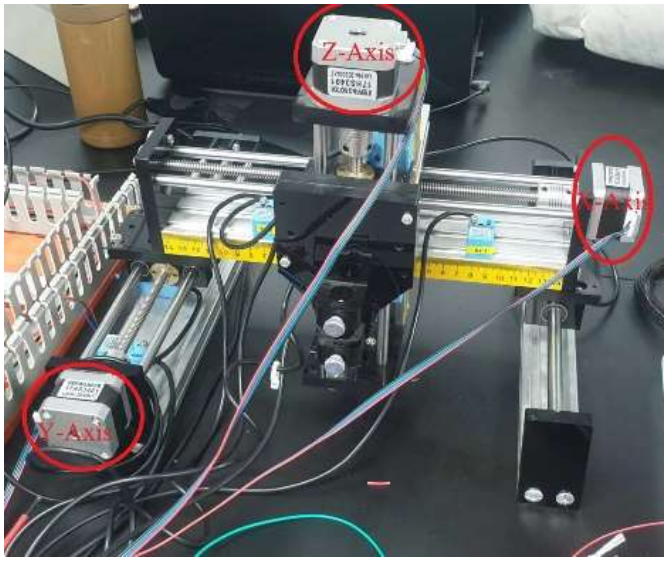

Figure 2*. Stepper motor configuration on the 2D plotter: mapping the location and axis assignment*

Github Repository: https://github.com/gunwoo0623/3-Axis-CNC-Machinery

Operating CNC Machine through Arduino IDE

Utilizing the flowchart presented in Fig. 3, the code is programmed to execute functions necessary for operating the CNC machine through the Arduino IDE. This code is applicable once the hardware and wiring of the CNC machine are finalized. Moreover, it is recommended to verify the initiation of CNC machine operation. Detailed explanations of the code will follow below.

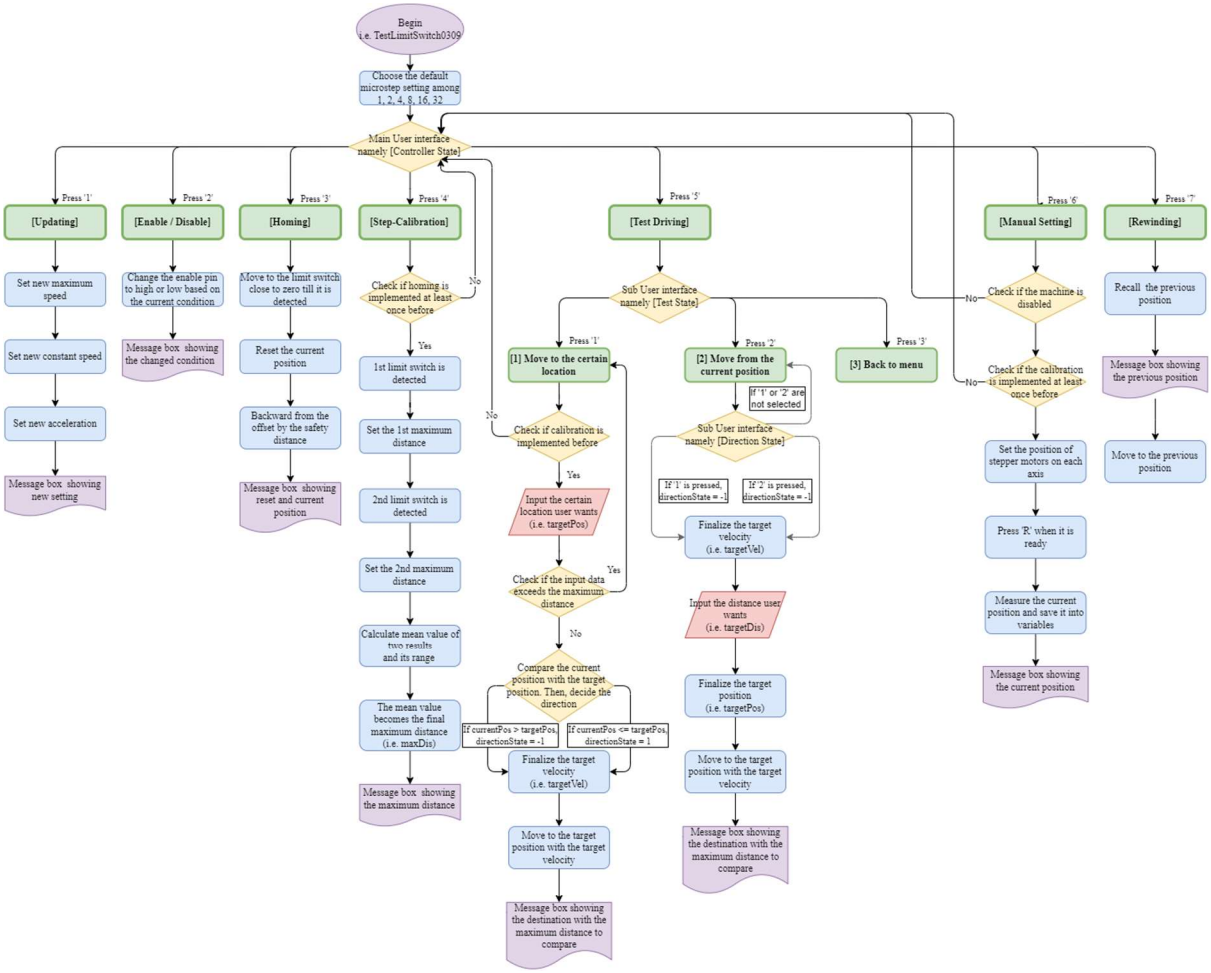

Figure 3. *Flowchart illustrating the operation of the miniature CNC machinery only via Arduino IDE.*

The interface, depicted on the left side of Fig. 4, will be presented initially for configuring micro-steps, aligning with the recommended default speed and acceleration values illustrated in Fig. 5. Then, the main menu displays seven available options as shown on the right side of Fig. 4.

Figure 4. *First interface to set micro-step setting (left) and main menu with 7 selections (right).*

```
if (microstepState == 1 || microstepState == 2 || microstepState == 4 || microstepState == 8 || microstepState == 16 || microstepState == 32) {
  switch (microstepState) {
    case 1:
      Serial.println(F("※ Default constant speed: 600 steps/s, default acceleration: 600 steps/s^2."));
      Serial.println(F("   One revolution for 1cm linear motion needs 1000 steps/s"));
      stepperMaxSpeed = 1000;
      stepperSpeed = 600;
      stepperAcceleration = 600;
      break;

    case 2:
      Serial.println(F("※ Default constant speed: 1200 steps/s, default acceleration: 1200 steps/s^2."));
      Serial.println(F("   One revolution for 1cm linear motion needs 2000 steps/s"));
      stepperMaxSpeed = 2000;
      stepperSpeed = 1200;
      stepperAcceleration = 1200;
      break;

    case 4:
      Serial.println(F("※ Default constant speed: 2400 steps/s, default acceleration: 2400 steps/s^2."));
      Serial.println(F("   One revolution for 1cm linear motion needs 4000 steps/s"));
      stepperMaxSpeed = 4000;
      stepperSpeed = 2400;
      stepperAcceleration = 2400;
      break;

    case 8:
      Serial.println(F("※ Default constant speed: 4800 steps/s, default acceleration: 4800 steps/s^2."));
      Serial.println(F("   One revolution for 1cm linear motion needs 8000 steps/s"));
      stepperMaxSpeed = 8000;
      stepperSpeed = 4800;
      stepperAcceleration = 4800;
      break;
```

Figure 5. *Recommended default speed and acceleration.*

The first function, 'Updating,' allows users to adjust the constant speed, maximum speed, and acceleration if a slower pace is desired. This is necessary as the default speed is currently set to the fastest after calculating the response time from the limit switch.

```
Selected [1]

[Updating]
Default maximum speed:2000.00
Present maximum speed:2000.00
New maximum speed value will be set as: 1000.00

Default constant speed:1200.00
Present constant speed:1200.00
New maximum speed value will be set as: 600.00

Default acceleration:1200.00
Present acceleration:1200.00
New accerelation value will be set as: 600.00
```

Figure 6. *'Updating' function for speed and acceleration adjustment.*

The second function, 'Enable / Disable,' serves the purpose of toggling the enable and disable status. Initially set to disabled for safety reasons, users must enable it before operating the machine, except during 'Manual Setting' operations.

```
※ Present Status: Disabled  ※ Present Status: Enabled
```

Figure 7. *'Enable / Disable' function for toggling machine status.*

The third function, 'Homing,' represents the initial step that must be executed before any further implementation. Its purpose is to reset the origin, directing all motors to (0, 0, 0) simultaneously. Following this, the motors slightly retreat from the origin to prevent the limit switch from being continuously engaged once it reaches the origin.

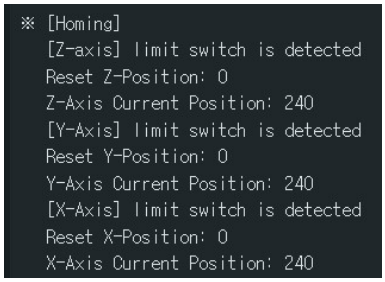

Figure 8. *'Homing' function for origin reset and position adjustment.*

The fourth function, 'Step Calibration,' serves as the second step to assess the range where motors can produce linear motion between two limit switches on each lead screw. This calibration is conducted sequentially for precise measurements and can also help minimize vibrations caused by other motors.

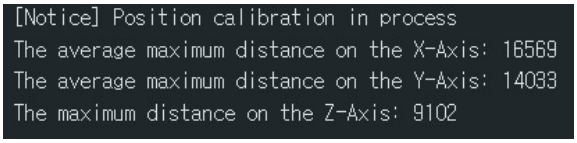

Figure 9. *'Step Calibration' function for linear motion range assessment.*

The fifth function, 'Test Driving,' involves two operations (e.g., [1] Moving to a specific location and [2] Moving from the current position). In these operations, the X and Y axis motors move first, followed by the Z axis motor, differing from 'Homing' and 'Calibration.' The choice '[1] Move to a specific location' sets the absolute position, initiating movement from the origin. This prompts the motors to navigate to a predetermined position within their specified ranges. (e.g., X: 16569, Y: 14033, Z: 9102). Conversely, the second option, '[2] Move from the current position', defines the relative position for precision adjustments. If initial positions match those in the first option (e.g., 8250, 7000, 4550), choosing the second option with 2000 steps in the counterclockwise direction yields final positions of (6250, 5000, 6550). The variation in the Z-axis motor's location, even when moving in the same direction as the X and Y axes motors, is solely attributed to the CNC machine's hardware configuration.

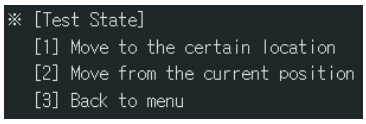

Figure 10. *User interface for absolution or relative opsitiong modes.*

Upon selecting either of the two options, five choices will appear on the screen (e.g., [1] X-Axis, [2] Y-Axis, [3] Z-Axis, [4] Run, and [5] Back to menu), as depicted in Fig. 11. Users simply input the corresponding numbers to specify the axis they wish to move, eliminating the need to configure all the X, Y, and Z axes simultaneously.

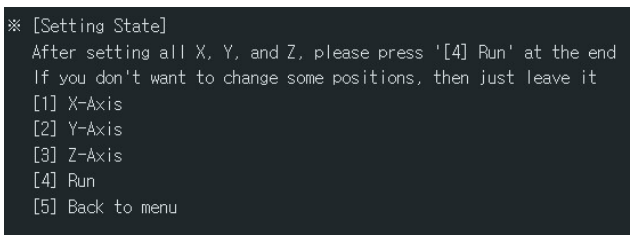

Figure 11. *User interface for axis selection options.*

The sixth function, 'Manual Setting,' is designed to manually recalibrate the positions of the three motors through physical adjustment, rather than numerical input. Initially, this necessitates disabling all motors to enable manual movement. Once the motors are manually positioned as desired, the second step involves pressing 'R' as shown in Fig. 12. Then, the CNC machine automatically returns to the origin and measures its new positions.

Figure 12. *'Manual Setting' operation with position re-calibration.*

Fig. 13 displays the manually set temporary positions as 11635 for the x-axis, 9215 for the y-axis, and 7266 for the z-axis, respectively. The inversion of the display order follows the programmed sequence of showcasing positions based on their arrival times.

Figure 13. *Display of manually set temporary positions.*

The final function, 'Rewinding,' is implemented to safeguard against accidental loss of previous positions. Given the presence of the 'Enable / Disable' function in the code, there's a risk of inadvertently losing positions by interacting with the lead screw or other components. Consequently, this function is designed to consistently retain previous positions whenever executed. For instance, the location just prior to this explanation was set through 'manual setting' at (11635, 9215, 7366). Even if intentionally altered to random positions manually, this function serves to restore the backup data.

Figure 14. *Display of position retention mechanism.*

Firmware Development for Unity Communication

The key distinction from operating the CNC machine through Arduino IDE lies in the requirement to interpret G-code and M-code, assign them to appropriate functions, particularly based on the positioning mode. Subsequently, the current positions post-implementation are transmitted to Unity, and in return, a new line of G or M-code is obtained. This process will be repeated till the end of line is executed.

Github Repository: https://github.com/gunwoo0623/3-Axis-CNC-Machinery

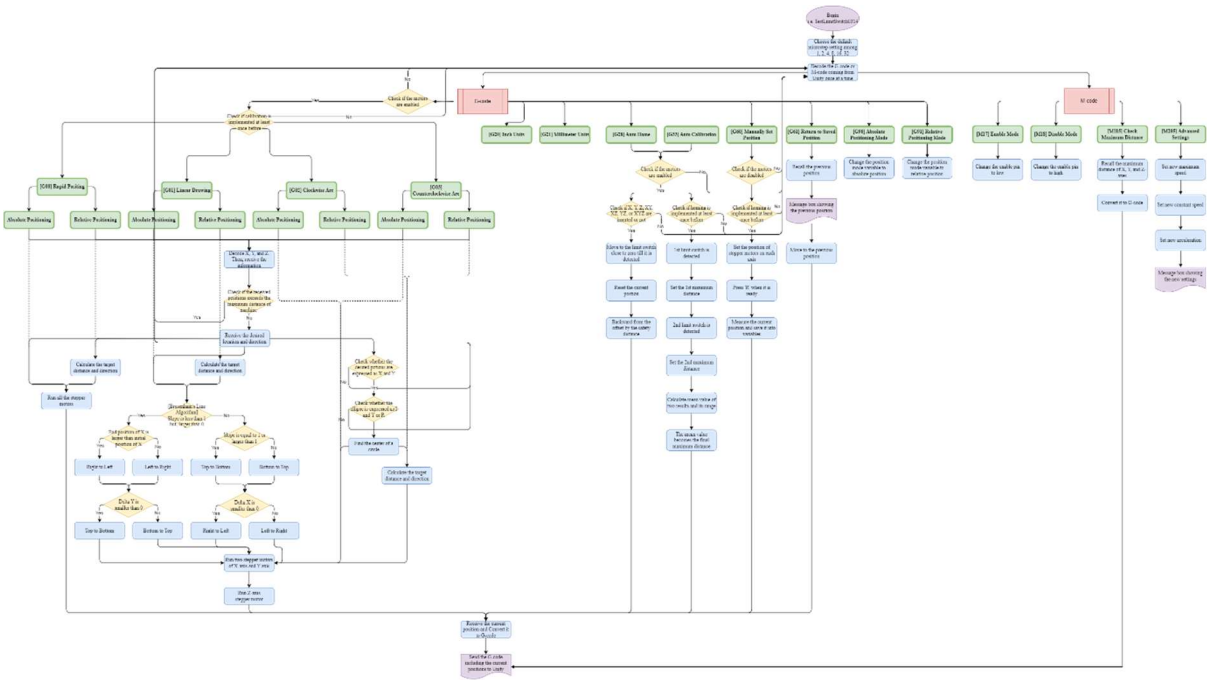

Figure 15. *Flowchart illustrating the firmware communicating with Unity for simulation.*

Below is a list of facilitated G and M-codes essential for the entire system: G00, G01, G02, G03, G28, G33, G90, G91, M17, and M18.

```
// [Codes]
// 0. G-code
#define STATE1_RAPID_POSITIONING 0
#define STATE1_LINEAR_DRAWING 1
#define STATE1_CLOCKWISE_ARC 2
#define STATE1_COUNTER_CLOCKWISE_ARC 3
#define STATE1_INCH_UNITS 20
#define STATE1_MILLIMETER_UNITS 21
#define STATE1_AUTO_HOME 28
#define STATE1_AUTO_CALIBRATION 33
#define STATE1_MANUALLY_SET_POSITON 60
#define STATE1_RETURN_TO_SAVED_POSITION 61
#define STATE1_ABSOLUTE_POSITINING 90
#define STATE1_RELATIVE_POSITINING 91

#define STATE2_ABSOLUTE_POSITINING 0  // Absolute positioning is the default
#define STATE2_RELATIVE_POSITINING 1

// 1. M-code
#define STATE1_ENABLE_STEPPERS 17
#define STATE1_DISABLE_STEPPERS 18
#define STATE1_MANUALLY_SET_POSITON 114
#define STATE1_CHECK_MAXIMUM_DISTANCE 115
#define STATE1_SET_ADVANCED_SETTINGS 205
```

Figure 16. *Key G and M-codes for CNC system operation.*

The ‘Decoding’ phase involves assigning the received G or M-codes to connect with other user-defined functions. Crucially, this step is essential for examining conditions before proceeding with the implementation.

Github Repository: https://github.com/gunwoo0623/3-Axis-CNC-Machinery

```
void Decoding() {  // Understand the command and seperate the word 'G', 'M', 'F', and 'S'
  // 0. G-code
  if ((Finding('G', noLetr) == noLetr) && (Finding('M', noLetr) == noLetr) && (stoAxn == 0 || stoAxn == 1 || stoAxn == 2 || stoAxn == 3)) {
    command = stoAxn;
    if (digitalRead(enablePin) == 0) {
      if (maxPosX == 0 || maxPosY == 0 || maxPosZ == 0) {
        Serial.println(F("[Warning] Calibration"));
      } else {
        switch (command) {
          case STATE1_RAPID_POSITIONING:
            switch (posMode) {
              case STATE2_ABSOLUTE_POSITINING:
                AbsLinPos();
                Positioning();
                break;
              case STATE2_RELATIVE_POSITINING:
                RelLinPos();
                Positioning();
                break;
            }
            break;
          case STATE1_LINEAR_DRAWING:
            switch (posMode) {
              case STATE2_ABSOLUTE_POSITINING:
                AbsLinPos();
                plotLine(initPosX, initPosY, endPosX, endPosY);
                break;
              case STATE2_RELATIVE_POSITINING:
                RelLinPos();
                plotLine(initPosX, initPosY, endPosX, endPosY);
```

Figure 17. *Decoding process for G and M-codes in CNC System.*

Prior to entering the 'decoding' phase, it is crucial to categorize the English alphabet, such as 'G' or 'M', and store the numerical values following the alphabet. This ‘Finding’ function essentially involves dismantling the structure of the received sentences.

```
double Finding(char letter, double medium) {
  char *pointer = array;
  retMed = 1;                                                                    // Reset the variable 'retMed' to be 0
  while ((long)pointer > 1 && (*pointer) && (long)pointer < ((long)array + curBuf)) {  // Refers to (https://github.com/MarginallyClever/GcodeCNCDemo)
    if (*pointer == letter) {                                                    // Check whether Arduino found one of characters such as X, Y, or Z. If yes, cha
      retVal = atof(pointer + 1);

      if (letter == 'G' && (retVal == 0 || retVal == 1 || retVal == 2 || retVal == 3)) {  // Return zero succesfully to 'Decoding();' function when G00 or G0 came
        stoAxn = retVal;
      } else {
        if ((letter == 'X' || letter == 'Y' || letter == 'Z') && retVal == 0) {
          return 'e';
        } else {
          return retVal;  // If the letter we are looking for is found, then return the double number only
        }
      }
    }
    pointer = (strchr(pointer, ' ') + 1);  // By using the white space as the seperator, it can know and read the position of the letter till the end of the array
  }
  retMed = medium + 1;
  return medium;  // Nothing found on the sentences
}
```

Figure 18. *Alphabet classification and numeric extraction for decoding.*

This constitutes the central aspect of implementing the Bresenham’s Line Algorithm for G01, representing linear drawing. Initially, all variables pertaining to the initial positions of X and Y, as well as the end positions of X and Y, are processed through the 'plotline' function. Subsequently, utilizing conditional statements ('if'), the algorithm identifies the octant and proceeds to operate the X and Y stepper motors first, followed by the sequential operation of the Z-axis stepper motor.

```
void plotLine(int iPosX, int iPosY, int ePosX, int ePosY) {
  if (abs(ePosY - iPosY) < abs(ePosX - iPosX)) {  // slope(= dX/dY) is less than 1 but, larger than 0
    if (-iPosX > -ePosX) {
      plotLineLeft(-ePosX, ePosY, -iPosX, iPosY);  // Right to Left
    } else {
      plotLineRight(-iPosX, iPosY, -ePosX, ePosY);  // Left to Right
    }
  } else {  // slope is equal to 1 or larger than 1
    if (iPosY > ePosY) {
      plotLineBottom(-ePosX, ePosY, -iPosX, iPosY);  // Top to Bottom
    } else {
      plotLineTop(-iPosX, iPosY, -ePosX, ePosY);  // Bottom to Top
    }
  }
}
void plotLineLeft(int i_X, int i_Y, int e_X, int e_Y) {···
}
void plotLineRight(int i_X, int i_Y, int e_X, int e_Y) {···
}
void plotLineBottom(int i_X, int i_Y, int e_X, int e_Y) {···
}
void plotLineTop(int i_X, int i_Y, int e_X, int e_Y) {···
}
void OperatingXY(int targetPosX, int targetPosY) {  // Run X and Y first and then, Z···
}
void OperatingZ(int targetPosZ) {···
}
```

Figure 19. *Bresenham's Line Algorithm Implementation for G01.*

The 'Timepos' function employs a timer mechanism for executing a designated task at consistent intervals. This piece of code establishes a timer to activate the 'Generating' function at specified intervals, offering a systematic approach for periodically carrying out a specific task within an Arduino program.

```
void TimerPos() {
  unsigned long currMS = millis();
  if ((currMS - prevMS) > timeInr) {  // Every (= timeInr) seconds, this will be executed
    prevMS = currMS;
    Generating();
  }
}
```

Figure 20. *Timer mechanism in Timepos function to use 'Generating' function.*

The 'Generating' function, referenced within the 'Timepos' function, serves the purpose of converting current positions into G-code, preparing them for subsequent commands to be transmitted by Unity.

```
void Generating() {
  currPosX = 0, currPosY = 0, currPosZ = 0;

  strcpy(strDis, M114);

  currPosX = -1 * stepper1.currentPosition();
  currPosY = stepper2.currentPosition();
  currPosZ = stepper3.currentPosition();
  currDisX = (currPosX / stepsForMM);
  currDisY = (currPosY / stepsForMM);
  currDisZ = (currPosZ / stepsForMM);

  // If there is no new incoming number, then don't make the information about X, Y, and Z
  ReceivePosX(currDisX);
  ReceivePosY(currDisY);
  ReceivePosZ(currDisZ);

  Serial.println(strDis);
}
```

Figure 21. *'Generating' function to translate the current positions to G-code.*

Github Repository: https://github.com/gunwoo0623/3-Axis-CNC-Machinery

Unity Program

Unity Application is made to interpret the machine language G-Code and M-Code into a human language which is accessible through the GUI.

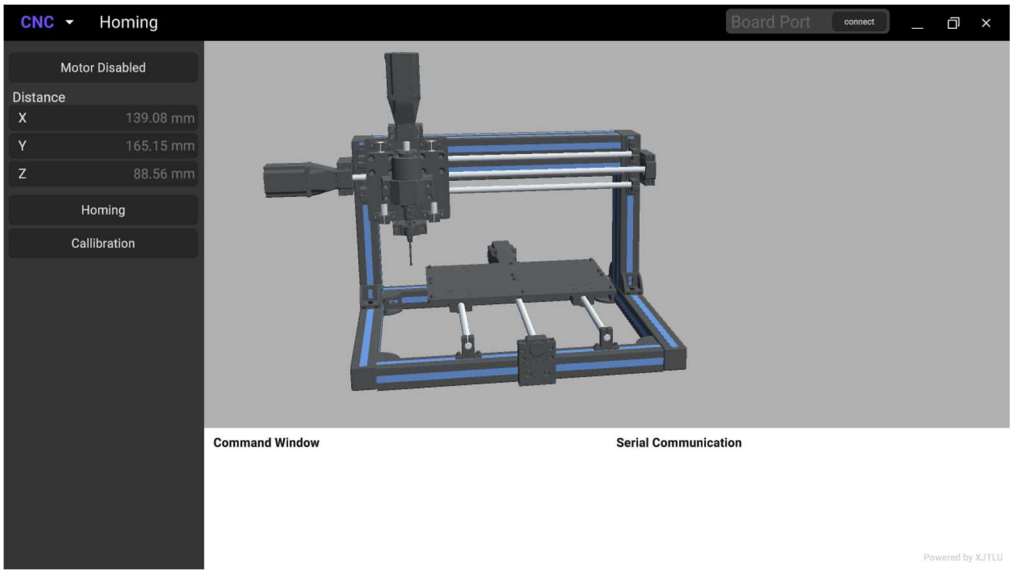

Figure 22. *The home screen of the Unity application was developed for 3-axis CNC machines and digital twin.*

The application sends signals of lines of stepper motor positions in terms of M-Code and G-Code to achieve the shape defined by the user. Then, position based on movement analysis and limit switch sensors data is received back to Unity to create digital twin systems.

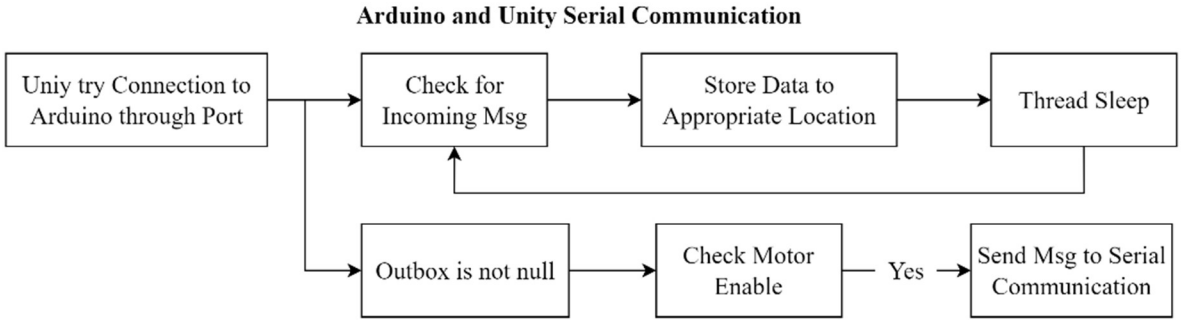

Figure 23. *The flowchart of Arduino and Unity serial communication script.*

The communication between the Arduino board and the Unity Application is established through Serial Communication. The program utilizes the function thread to receive incoming messages and it will sleep for 100 microseconds before checking for a new incoming message. The Outbox instead is always on a loop for sending outbox messages only when the stepper motor is enabled.

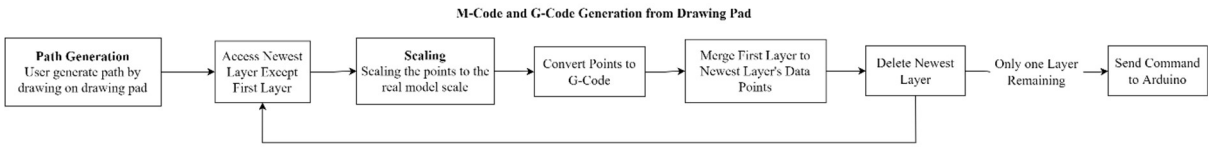

Figure 24. *The machine language command generation from drawing pad script flowchart.*

Then the first method of G-Code position generation is using the drawing pad. When the user creates an illustration on the drawing pad, the lines generated have position points that can be converted and generated to real-scale positioning. The process involves creating layers of drawing and processing the data points in every layer and converting all the points into G-Code. Then, the lines of G-Code are sent to Arduino after every execution.

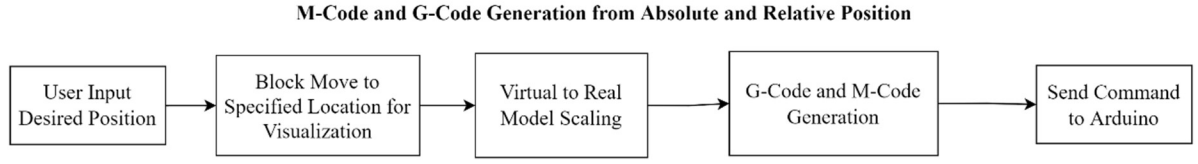

Figure 25. *The machine language command generation from absolute and relative positioning script flowchart.*

For absolute and relative positioning, the user has two options to input the data which are through typing and a joystick. Both inputs updated the database and positioned the box to the corresponding position for visualization. Then, the position is sent to Arduino in the form of G-Code

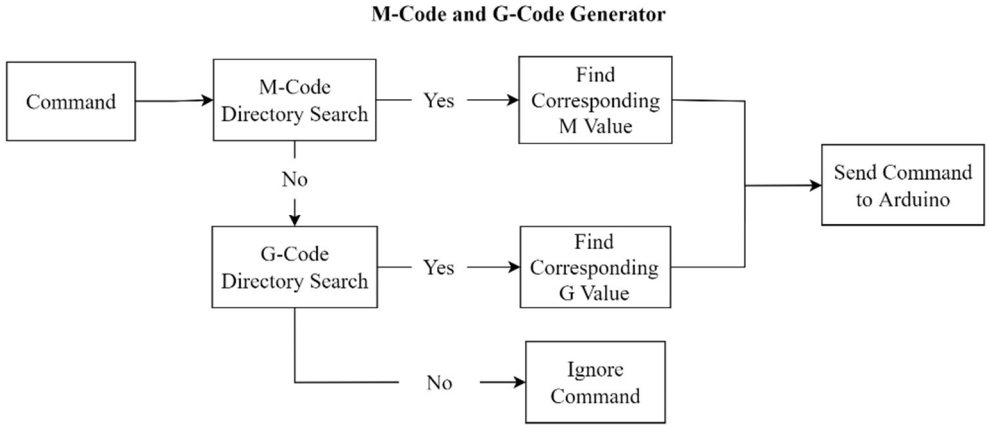

Figure 26. *The M-Code and G-Code generator script flowchart.*

The machine language used for the communication code encryption are M-Code and G-Code which are commonly used in the CNC machinery. Using the directory data structure, the appropriate code and commands can be encoded and decoded. For outbox messages, the message is sent to Arduino. For incoming messages, the process of flowchart in Fig. 26 is reversed with the data stored and processed according to the command at the end.

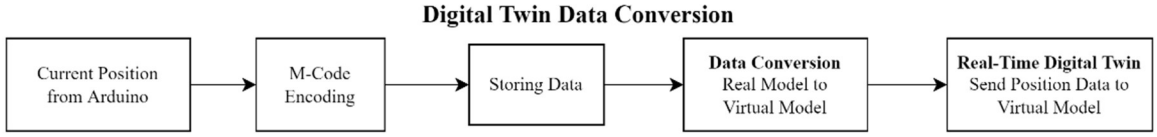

Figure 27. *The current position data conversion to real-time digital twin.*

The digital twin data conversion involves five steps. Firstly, the current position is obtained through serial communication in the form of M-Code. After the code is encoded, the data is stored. Then the data is processed to fit the scale of the virtual model. Finally, the digital twin is established by the constant rigid body positioning update.

3 Simulation

The operation of the 3-axis CNC machine involves three main steps: initialization, user input acquired, command execution, and digital twin real-time simulation. Initially, the Arduino Board should be encoded with the Arduino Program. Then, the Unity Application is opened. After establishing the connection to the Arduino board through the serial connection and defining the microstep, the machine is ready to be operated, which concludes the initialization of the machine. The Unity application has three different methods to obtain commands and send them to the Arduino board, which are position specification, shape illustration for path definition, and G-Code and M-Code reader. For position specification, the machine is commanded to go to an absolute or relative position defined by the user. Shape illustration for path definition is a feature to draw on a drawing pad, which converts the points drawn to lines of M-Code and G-Code commands to send to Arduino. Lastly, the G-Code and M-Code reader is a function that interprets a premade M-Code and G-Code to be sent promptly to the Arduino board. Thus, the user input is obtained. The Arduino board then decodes the obtained G-Code and M-Code and processes the appropriate code accordingly. For any carving instructions, the path is broken down into shorter distances to ensure the x-y motor arrives at the destination simultaneously. However, if not, the command is sent to the x, y, and z stepper motor drivers separately. The appropriate command is then executed and sent to the motor driver to run the stepper motor. Then, according to the position data obtained from position derivation or the limit switches, the data is sent back to Unity to simulate digital twin motion. Through data manipulation, the 3D model of the system is simulated and shown as a real-time digital twin of the system inside the Unity Application. In the case of a hardware error or emergency, the circuit breaker should be pulled off to turn off the system.

Appendix A. Engineering Drawing

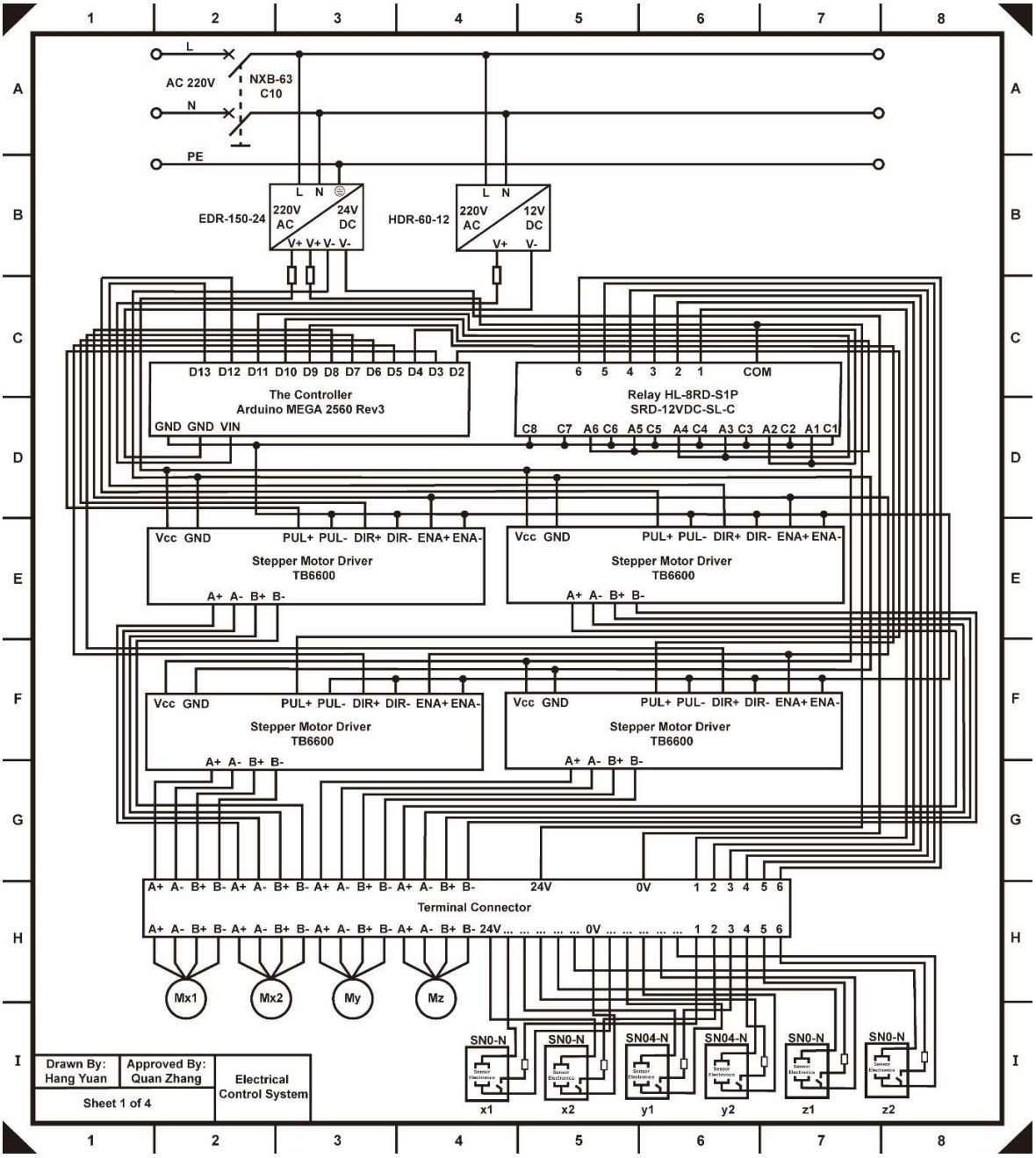

Figure A1. Electrical control system.

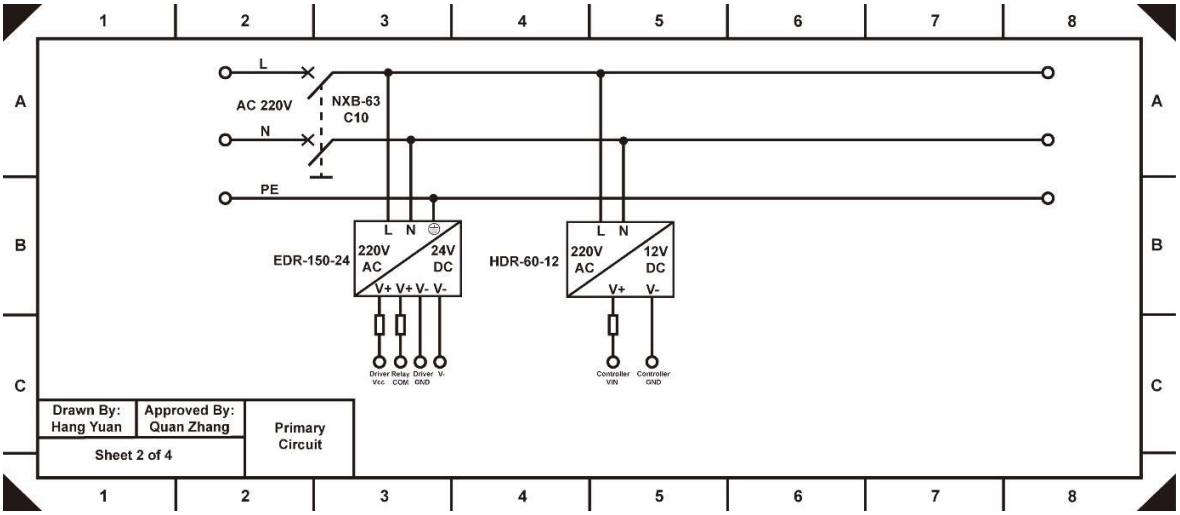

Figure A2. Primary circuit.

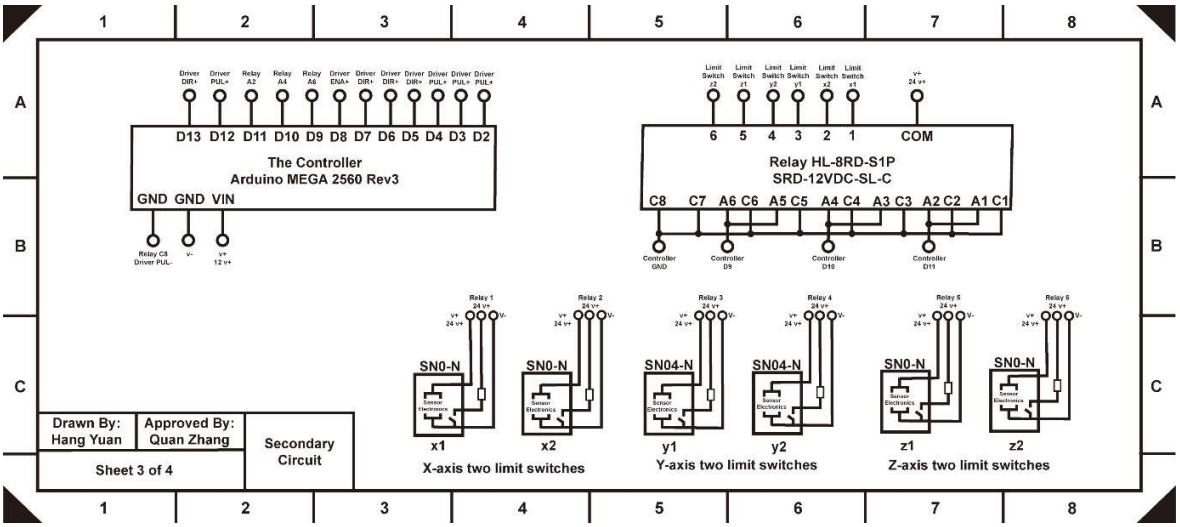

Figure A3. Secondary circuit.

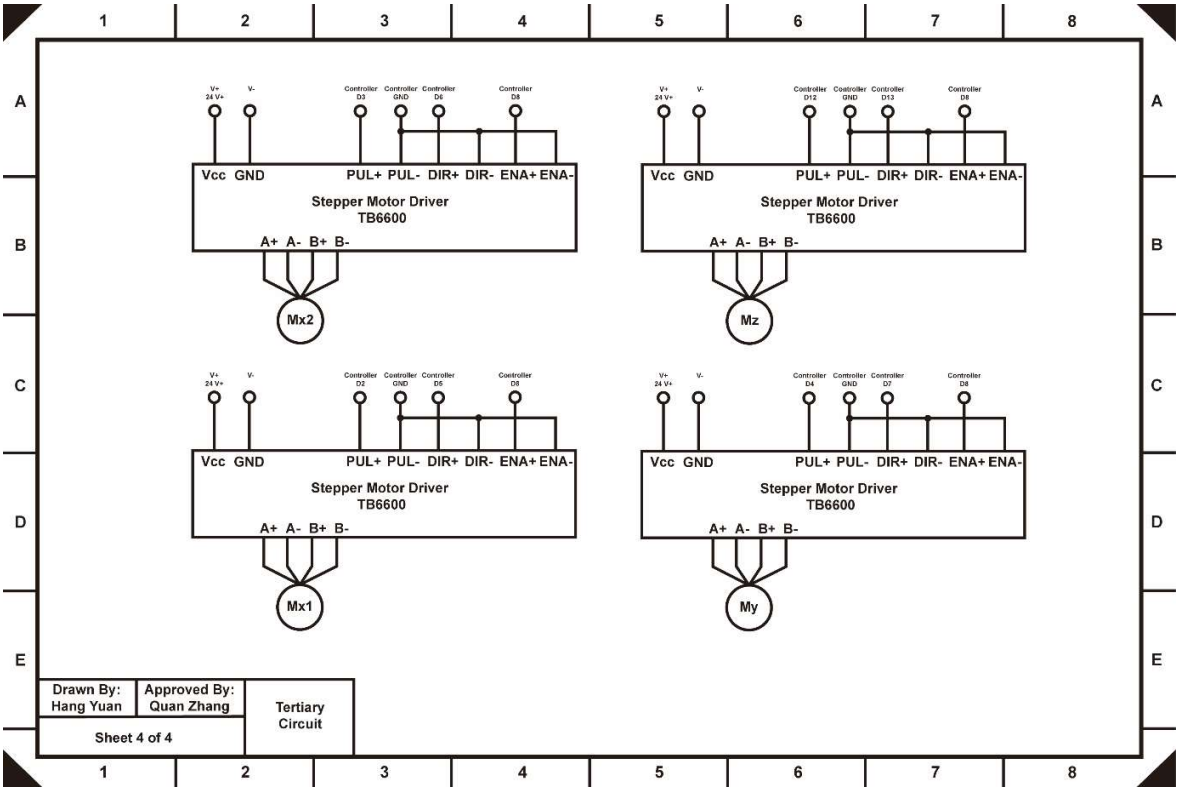

Figure A4. Tertiary circuit.