



Palestine Polytechnic University

College of Information Technology and Computer Engineering

Department of Computer System Engineering

Intelligent Handwriting Robot

Team:

Amir Altakroori

Mohammad Samaheen

Wisam Alhroub

Supervisor:

Dr. Alaa Halawani

May 2022

Acknowledgment

In the name of "Allah", the most beneficent and merciful who gave us strength, knowledge and helped us to get through this project.

This project gave us valuable experience in deep learning architecture and robotic systems. For that, we want to thank all the people who helped us through the project.

We would like to express our gratitude to our graduation project supervisor Dr. Alaa Halawani for his guidance, support, and encouragement throughout the project.

Furthermore, we have to show our gratitude to Eng. Wael Takrouri for answering our questions and providing us with valuable suggestions and tips that helped us in our project, especially in the hardware problems.

Moreover, we must thank our families for their generous encouragement, and continuous support throughout our life. For our friends, we are truly grateful for all of your support and help during the project and throughout the educational stage.

Abstract

There is a huge demand in the field of Artificial intelligence for making novel works that can mimic human creativity. One interesting example is handwritten calligraphy. Robotic calligraphy, as a typical application of robot movement planning, is of great significance for the education of calligraphy culture. The existing implementations of such robots often suffer from their limited ability for font generation and evaluation, leading to poor writing style diversity and writing quality. These demands are increasing rapidly, hoping to create something intelligent enough to pace with them at handwritten calligraphy without human intervention. The core idea of our project is to provide a solution that humans can utilize, to mimic the calligraphy of handwritten texts using a robotic system. The robotic system can learn from previous artworks, and produce similar strokes with a sense of learning creativity. In our work, we aim to reach a satisfying level where handwriting is done automatically, creatively, and with astonishing results. The work is utilizing reinforcement learning techniques with a long-short-term memory (LSTM) model alongside a generative adversarial model (GAN to be used as a proof of concept) to achieve an approach in which the robot will be guided. This work has shown by its outcomes that it's possible to write words smoothly on physical surfaces using the robot and not only soft-writing but as well to mimic a specified human writing style from images after pretraining the system on sequences of points of lines belonging to other styles.

Table of Content

| | |
|--|-----------|
| Abstract | I |
| Table of Content | II |
| List of Figures and Tables | VI |
| Figures | VI |
| Tables | VII |
| Chapter 1: Introduction | 1 |
| 1.1 Overview | 2 |
| 1.2 Motivation and Importance | 2 |
| 1.3 Problem Statement | 3 |
| 1.4 List of Requirements | 3 |
| 1.5 Methodologies to use | 3 |
| 1.6 System Objectives | 4 |
| 1.7 Short Description | 4 |
| 1.8 Expected Results | 5 |
| 1.9 Work Constraints | 6 |
| 1.10 Overview for The Rest of The Report | 6 |
| Chapter 2: Background | 7 |
| 2.1 Overview | 8 |
| 2.2 Theoretical background | 8 |
| 2.2.1 Machine Learning | 8 |
| 2.2.2 Reinforcement learning | 9 |
| 2.2.3 Deep Learning | 10 |
| 2.2.3.1 Convolution Neural Network | 10 |
| 2.2.3.2 Recurrent Neural Network | 11 |
| 2.2.3.3 Long Short-Term Memory | 11 |
| 2.2.3.4 Generative Adversarial Network | 12 |
| 2.2.3.5 Conditional GAN (CGAN) | 14 |
| | II |

| | |
|---|-----------|
| 2.3 Literature Review | 15 |
| 2.4 System Hardware Components | 18 |
| 2.4.1 Controller boards | 19 |
| 2.4.1.1 Arduino Uno | 19 |
| 2.4.1.2 Raspberry Pi Model B | 19 |
| 2.4.1.2.1 VNC Viewer | 21 |
| 2.4.1.2.2 SSH | 21 |
| 2.4.2 Robotic components | 21 |
| 2.4.2.1 CNC V3 Shield | 21 |
| 2.4.2.2 MG996R Servo Motor | 21 |
| 2.4.2.3 NEMA17 Stepper Motor | 22 |
| 2.4.2.4 A4988 Stepper Driver Chip | 22 |
| 2.4.2.5 12V adapter | 22 |
| 2.4.2.6 Cooling fan | 22 |
| 2.5 System Software Components | 23 |
| 2.5.1. Python | 23 |
| 2.5.2. TensorFlow | 24 |
| 2.5.3 Arduino IDE | 24 |
| Chapter 3: Design | 25 |
| 3.1 Overview | 26 |
| 3.2 Detailed Description of the Design | 26 |
| 3.3 Design Options | 28 |
| 3.3.1 Hardware Design Options | 28 |
| 3.3.2 Software Design Options | 30 |
| 3.3.2.1 The GAN Approach Overview | 31 |
| 3.3.2.1.1 Character Discriminative Module | 33 |
| 3.3.2.1.2 Character Generative Module | 33 |
| 3.3.2.2 The LSTM Approche | 35 |
| Chapter 4: Software & Implementation | 37 |

| | |
|--|-----------|
| 4.1 Model and Software Configuration | 38 |
| 4.1.1 RaspberryPi Software | 38 |
| 4.1.2 Software Building | 39 |
| 4.1.2.1 GAN Architecture | 39 |
| 4.1.2.1.1 Generative Model | 40 |
| 4.1.2.1.2 Discriminative Model | 42 |
| 4.1.2.2 LSTM Model | 42 |
| 4.2 Hardware Configuration | 43 |
| 4.2.1 Assembling the Robot | 43 |
| 4.2.2 Device Identification | 45 |
| 4.2.3 Gcode Instructions | 45 |
| Chapter 5: Validation & Results | 46 |
| 5.1 Unit Testing | 47 |
| 5.1.1 RaspberryPI 4 | 47 |
| 5.1.2 Robot Calligrapher | 47 |
| 5.2 Integration Testing | 48 |
| 5.2.1 Connecting RaspberryPI with the Robot | 48 |
| 5.2.2 Installing Necessary Software on RaspberryPI | 48 |
| 5.3 Results and Output Samples | 48 |
| 5.3.1 Output Samples - LSTM | 48 |
| 5.3.2 Output Samples - GAN | 51 |
| 5.4 Measurements and Readings | 54 |
| 5.4.1 LSTM measurements | 54 |
| 5.4.2 GAN measurements | 55 |
| Chapter 6: Conclusion & Future Work | 56 |
| 6.1 Conclusion | 57 |
| 6.2 Future Work | 57 |
| References | 58 |

List of Figures and Tables

Figures

| | |
|--|----|
| Figure 1.1: System Components with Workflow | 5 |
| Figure 2.1: Interaction between the Agent and Environment in RL | 9 |
| Figure 2.2: RNN node | 11 |
| Figure 2.3: RNN unrolled | 11 |
| Figure 2.4: The repeating module in an LSTM | 12 |
| Figure 2.5: The process that GANs move in | 13 |
| Figure 2.6: CGAN architecture | 14 |
| Figure 2.7: Substrokes assembling | 15 |
| Figure 2.8: Arduino UNO | 19 |
| Figure 2.9: Raspberry Pi 4 Model B | 20 |
| Figure 2.10: CNC V3 Shield | 23 |
| Figure 2.11: MG996r Servo motor | 23 |
| Figure 2.12: NEMA 17 Stepper motor | 23 |
| Figure 2.13: Stepper driver chip | 23 |
| Figure 2.14: Cooling fan | 23 |
| Figure 2.15: 12V adapter | 23 |
| Figure 2.16: LY plotter robot | 23 |
| Figure 3.1: Block Diagram for the different general components in the robotic system | 26 |
| Figure 3.2: Schematic Diagram of the Robotic plotter interface | 27 |
| Figure 3.3: Robot Components when assembled | 29 |
| Figure 3.4: Detailed View of the GAN's Submodels for the Robotic System | 31 |
| Figure 3.5: Handwritten synthesis network architecture | 35 |

| | |
|---|----|
| Figure 4.1: RaspbianOS GUI through VNC application | 39 |
| Figure 4.2: SSH Shell from mobile | 39 |
| Figure 4.3: localized regions for the number '0' | 40 |
| Figure 4.4: Space localizing implementation for numbers 0 to 9 | 40 |
| Figure 4.5: Implemented GAN architecture | 41 |
| Figure 4.6: Implemented Discriminator architecture | 41 |
| Figure 4.7: Training samples from the IAM online handwriting database | 42 |
| Figure 4.8: Implemented LSTM network | 43 |
| Figure 4.9: LY robot components | 44 |
| Figure 4.10: CNC shield with drivers | 44 |
| Figure 4.11: Arduino UNO attached | 44 |
| Figure 4.12: Full robot assembled | 44 |
| Figure 4.13: Installed motor drivers | 44 |
| Figure 4.14: Robot hardware identification on RaspbianOS | 45 |
| Figure 5.1: Measuring the Robot Calligrapher accuracy | 47 |
| Figure 5.2: Process of writing 'Polytechnic' using LSTM | 49 |
| Figure 5.3: Different styles of 'Polytechnic' using LSTM | 50 |
| Figure 5.4: LSTM output improvement at selected timestamps | 50 |
| Figure 5.5: GAN output improvement | 51 |
| Figure 5.6: Different variations of number '2' | 51 |
| Figure 5.7: GAN's labels output at convergence | 52 |
| Figure 5.8: Different styles of '4' using GAN | 53 |
| Figure 5.9: Process of writing '4' using GAN | 53 |
| Figure 5.10: LSTM training loss | 54 |
| Figure 5.11: Generative loss for all digits together | 55 |

Tables

| | |
|--|----|
| Table 2.1: Comparison between selected works and our work | 18 |
| Table 3.1: Comparison between Shenzhen Liyang LY machine & GKDraw X3 | 30 |
| Table 4.1: Initial training comparison between template-free and template usages | 41 |

Chapter 1

Introduction

This chapter provides an introduction to the project. It starts with the motivational statement, followed by the aims and objectives, a brief description, and the expected results for the project.

1.1 Overview

With the quick evolution of technology on the intelligent side, robotics have been widely applied to promote human culture and education, such as acting, drawing, and robotics character writing. There are new demands in this regard for producing text that can mimic human handwriting capabilities. These demands are getting big in number rapidly, hoping to create something intelligent enough to keep up with human creativity in handwritten calligraphy without outsiders' intervention. The core idea of robotics writing is the generation of sequences of robotic actions that follows specific criteria. Thus, the focus of recent research is the design of control algorithms to drive robotic end-effectors to write handwritten characters or letters.

In addition, there is a need to accomplish work (i.e. letters, mails, reports, updates) in the format of a handwritten text, as many people like the idea of something handmade. Although it takes time and effort to make some good handwritten scripts for a long time, it can be forgivable if the person was making it with passion. In the field of paleography, there are some cases of ancient manuscripts that might have some of their text almost unclear or even wiped, rewriting such old scripts again with the same font style and calligraphy can be a challenge for nonexperts.

1.2 Motivation and Importance

Robotic writing is a particularly hot topic due to the great applicability of its key technology in other applications, including robotic drawing [34], industrial welding [35][36], and medical rehabilitation [37] among others. It is important to have an intelligent calligraphy system that has the ability to generate handwritten scripts when there are a lot of fields that have been invaded by AI. For that, the robot has to get efficient enough to be as human-like as possible in its task.

The motivation increased when seeing all problems described previously, which raised the need for a system that can make handwritten text, automate the process of writing characters or words, learn existing text to mimic the way it was written, simulate human creativity, simulate human culture, and create new text styles and patterns to enrich the existing set of text calligraphy as

well. Also, since heritage is an essential base of societies, calligraphic robots could imitate old manuscripts' font styles, which could have a valuable effect on the restoration of archeological manuscripts.

1.3 Problem Statement

Some people want their paper-based messages to be old school (i.e. to be written manually), but they do not have enough time to do it. The need for a robot that can automate this task is arising, and the optimization in this field is rapidly evolving, hoping to reach a satisfying level where handwriting is done automatically, creatively, and with astonishing results.

1.4 List of Requirements

1. The system should be able to generate characters similar to human styles with no human inference.
2. The robot should be able to take a script from the user as an input, then generates handwritten strokes based on its capabilities.
3. An evaluation part should be able to distinguish between the drawn script and the one given from the system without human inference.
4. The robot should be able to draw handwritten calligraphy on different kinds of surfaces (paper, wood, plastic, etc...) with various annotating tools (pen, marker, crayon, etc...).

1.5 Methodologies to use

Our design of a robotic system that is going to learn from an external source will be using a long-short-term memory (LSTM) to generate handwritten words. In addition, we are going to design and build a generative adversarial network (or GAN in simple) to mimic a specific handwritten style. We can ensure that the robot will be smart enough to generate handwritten text from what it learns, and can judge it by either acceptance or rejection. The GAN model is going to be powered by a pre-trained generator to optimize the searching space using deep learning

networks and some reinforcement learning principles to strengthen the smartness and accuracy levels of the robot, as well as lowering error rates and preventing state lock incidents.

1.6 System Objectives

Our project is aiming towards achieving the following objectives:

- Design and build a robotic calligraphy learning system, which has two sub-systems mainly. Generator sub-system, and evaluator sub-system, that can generate handwritten text.
- Simulate existing calligraphy as well as get closer to human creativity by providing enough space for the system to generate texts without human interference.
- Generate a sequence of points to pass it to the robot system. The sequence needs to be convertible to G-Code which is a robot understandable sequence of instructions.
- Combine deep learning solutions with reinforcement learning policies to guide the robot to learn the calligraphic styles by itself.
- Observe the output of the robot and return feedback to the generation subsystem to optimize the future output.
- Install the software of the project on a microcomputer and manage the generation process via remote control.

1.7 Short Description

Figure 1.1 shows the workflow of our system to be designed. The user of the system will give a text to be written. This text is recognized by the intelligent software model hosted on a microcomputer-based subsystem like a PC, or a raspberry pi, which will give instructions for the drawing to the robot. The robot itself will draw incoming handwritten instruction data as a sequence of points to a free input space that can be a paper or a smooth touch-responsive surface, thus producing a sequence of written strokes later to be constructed as an image.

In the training period, the intelligent part will tell the robot to write something, and the generative model in that part will generate a robot understandable sequence of instructions to draw and convert the instruction to a handwritten character and give it to the robot to draw it on the input space. An image of the finished work is to be submitted to the discriminative module on the learning system module to judge if the work of the robot is mimicking the human handwritten characters or not. Improvements to the model itself will be present based on the judge's results. The discriminative part will be trained on the same real data but the generative part will train on different but similar data to the real one.

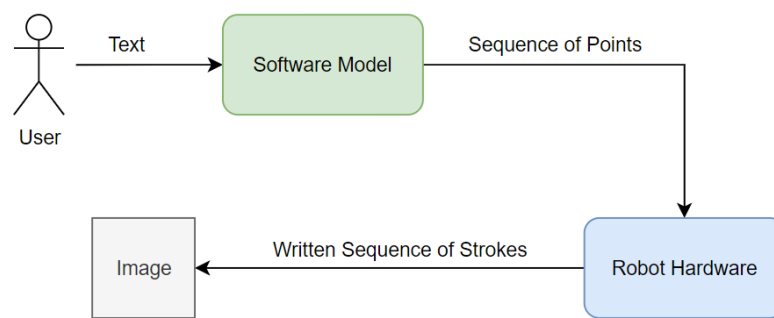


Figure 1.1: System Components with Workflow

1.8 Expected Results

1. A functional robot that can mimic the calligraphic styles of humans and simulate human creativity.
2. Decrease the gap between the digital world and the physical world by using this system to produce non-distinguishable handwritten letters to be used in multiple fields.
3. The robot should be able to draw handwritten calligraphy on different kinds of surfaces (paper, wood, plastic, etc...) with various annotating tools (pen, marker, crayon, etc...) considering the compatibility between such surfaces and tools.

1.9 Work Constraints

As we all know, nothing comes to be perfect out of the box, and the same goes for our project. When we planned the whole system of this project, we had to put some limitations. We list them as follows:

1. Taking advantage of the GAN model to produce words would be so expensive for our obtained computing resources.
2. Dealing with curvatures in GAN's strokes will be minimal.

1.10 Overview for The Rest of The Report

The rest of the report is organized as follows: Chapter 2 presents a theoretical background of the project, and a description of the hardware and software components is discussed in addition to the system specification and design constraints. Chapter 3 shows the detailed design, block diagrams, and flowcharts. Chapter 4 discusses the implementation of the used hardware and software solutions. Chapter 5 will demonstrate the results and outcomes of our project alongside some hardware and software testing. After that, there will be a conclusion and future work chapter as chapter 6. Then a list of references.

Chapter 2

Background

2.1 Overview

This chapter explores the different theoretical aspects of the project. It provides a clear and brief description of the methods/techniques employed in this project.

2.2 Theoretical background

This section provides some information about some technologies and algorithms that will be used in the project.

2.2.1 Machine Learning (ML)

ML makes systems have the ability to learn by themselves and build their model and actions from experiences without being explicitly programmed. ML gives the computer the ability to develop and change when getting new data.

ML is split into main fields based on the learning paradigm. Examples of these fields are supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. Reinforcement learning will be described in more details in the next subsection since it is one of the main parts of this project.

Supervised Learning (SL): in this field, the ML algorithm is trained on labeled data. By giving input data and its corresponding output labels to the model, through the training process, the model builds relationships and patterns between the input and the output, so the model can estimate the output for new input. For SL, the data must be labeled accurately to work well.

Unsupervised Learning (USL): it is similar to SL, but in this field, the model can work with unlabeled data. The task of this field is to find the similarity, patterns, and differences between the input data and then group them into clusters. The goal of USL is to understand the data and categorize it into similar groups (clusters) [1].

Semi-supervised Learning (SSL): it is a learning model concerned with the study of how computers and natural systems such as humans learn in the presence of both labeled and unlabeled data. The goal of semi-supervised learning is to understand how both labeled and unlabeled data could change the learning behavior, and design algorithms that take advantage of such a combination [8].

2.2.2 Reinforcement learning

RL is an area of machine learning technique that mimics the learning process of human and animal beings. It is based on the notion of reward and punishment. It encourages good behavior with rewards and depresses bad behavior with punishment. RL enables an agent to learn by trial and error. This is done using feedback from the environment to find if a current state-action pair is rewarded or punished as Figure 2.1 shows. In operation, an RL model would choose actions that maximize reward. Consequently, a machine can learn simply by trying various actions and observing which one rewards best. Just like animals and humans [20]. The difference between RL and SL is that it does not need labeled input/output pairs to be presented and also no need for sub-optimal actions to be explicitly corrected, but it focuses on finding a balance between exploration of uncharted territory and exploitation of current knowledge. In this project, RL will be embedded in the robot learning process.

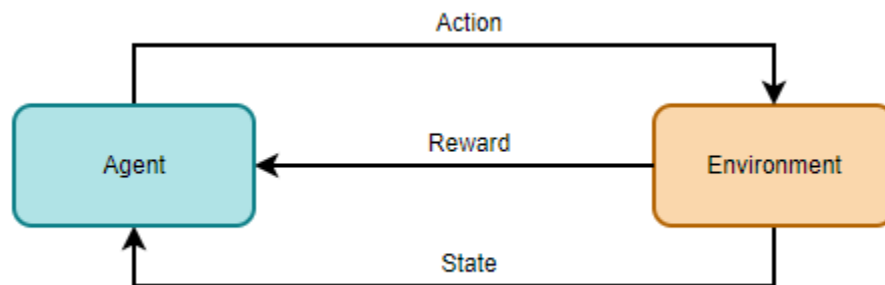


Figure 2.1: Interaction between the Agent and Environment in RL

Policy gradient is a type of reinforcement learning technique that relies upon optimizing parametrized policies concerning the expected return (long-term cumulative reward) by gradient descent. A policy represents a set of instructions to always take the agent on the best possible path concerning gaining points. It can assure that the agent will not fall into unwanted states. Policies have two types, deterministic policies and stochastic policies. Deterministic policies are used in deterministic environments. These are environments where the actions taken determine the outcome. There is no uncertainty. On the other hand, a stochastic policy outputs a probability distribution over actions. It means that instead of being sure of taking action A, there is a probability we will take a different one. Stochastic policy is used when the environment is uncertain.

2.2.3 Deep Learning

Deep learning is a branch of machine learning that extracts features without the need for human engineering. So Deep Learning achieves great power and flexibility in learning [12].

In terms of architecture, most deep learning methods use neural network architectures. A neural network consists of an input layer, an output layer, and hidden layers. Each layer consists of one or more connected nodes, each node is called a **neuron** or a perceptron. Each neuron has a weight and an activation function. The term “deep” usually refers to having a large number of hidden layers in the neural network. Deep learning models are trained by using large sets of labeled data and neural network architectures that learn features directly from the data without the need for manual feature extraction [13][14]. Deep learning contains many different architectures such as convolution neural networks and recurrent neural networks. Also, it has different frameworks, one of which is generative adversarial networks.

2.2.3.1 Convolution Neural Network

CNN is a deep learning architecture. It takes its name from the mathematical linear operation between matrices called convolution. CNN has multiple layers, including the convolutional layer, non-linearity layer, pooling layer, and fully-connected layer. The convolutional and fully-connected layers have parameters but pooling and non-linearity layers do not have parameters. CNN has an excellent performance in machine learning problems. Especially the applications that deal with image data, such as image classification, computer vision, and natural language processing (NLP), and the results achieved were very competitive [15].

2.2.3.2 Recurrent Neural Network

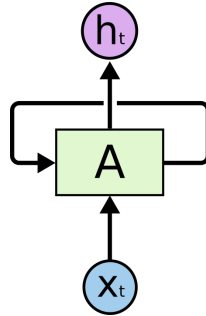


Figure 2.2: RNN node [40]

A recurrent neural network is a type of artificial neural network commonly used in speech and video recognition and natural language processing. Recurrent neural networks recognize data's sequential characteristics and use patterns to predict the next likely scenario. They are networks with loops in them, allowing information to persist. Figure 2.2 shows a single node of RNN.

This loop-like structure can be unfolded as well to form the regular shape RNN as in Figure 2.3

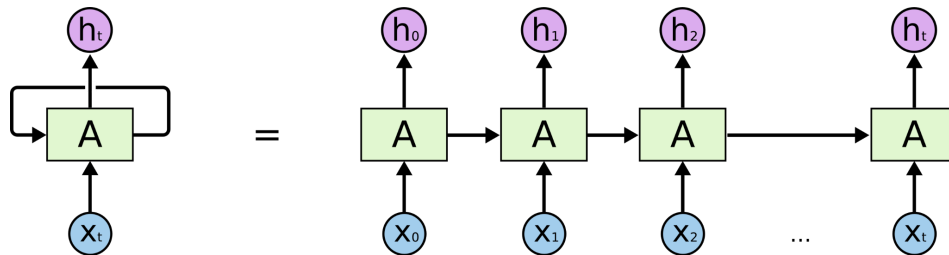


Figure 2.3: RNN unrolled [40]

The problem with RNN is Long-Term Dependencies where it is hard for the network to consider far-away data in its prediction, and limiting its prediction to short-term data span. That is where Long Short-Term Memory comes into use.

2.2.3.3 Long Short-Term Memory

Long Short-Term Memory Networks (LSTM) is a special kind of RNN, capable of learning long-term dependencies. They are explicitly designed to avoid long-term dependency problems. Remembering information for long periods is particularly their default behavior. LSTMs also have this chain-like structure as you can see in Figure 2.4, but the repeating module has a different structure. Instead of having a single neural network layer, there are four of them

interacting in a very special way. The four networks are bounded by yellow rectangles as shown in Figure 2.4.

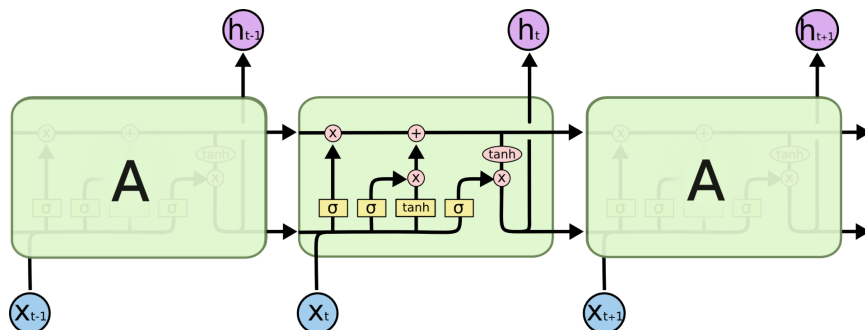


Figure 2.4: The repeating module in an LSTM [40]

We will use LSTMs as a basis for the sequence-points word generation deep neural network model since it has a better ability to remember inputs over a long period of time.

2.2.3.4 Generative Adversarial Network

A generative adversarial network (GAN) is a class of deep learning frameworks designed by Ian Goodfellow and his colleagues in June 2014. Two neural networks contest with each other in a game (in the form of a zero-sum game, where one agent's gain is another agent's loss) [5].

Given a training set, this technique learns to generate new data with the same statistics as the training set. For example, a GAN trained on photographs can generate new photographs that look at least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a form of a generative model for unsupervised learning, GANs have also proved useful for semi-supervised learning, fully supervised learning, and reinforcement learning [6].

There are two main networks that construct any GAN model; 1) the generator network that generates new data instances, and 2) the discriminator network which decides whether each instance of data that it reviews belongs to the actual training dataset or not [6]. The core idea of a GAN is based on the "indirect" training through the discriminator, which itself is also being updated dynamically. This means that the generator is not trained to minimize the distance to a specific image, but rather to fool the discriminator. This enables the model to learn in an unsupervised manner.

Figure 2.5 shows the process of GAN. The generator takes in random numbers and returns the data. Then, the generated data is fed into the discriminator alongside a stream of data taken from the actual, ground-truth dataset. Lastly, the discriminator takes in both real and fake data and returns probabilities, a number between 0 and 1, with 1 representing a prediction of authenticity and 0 representing fake.

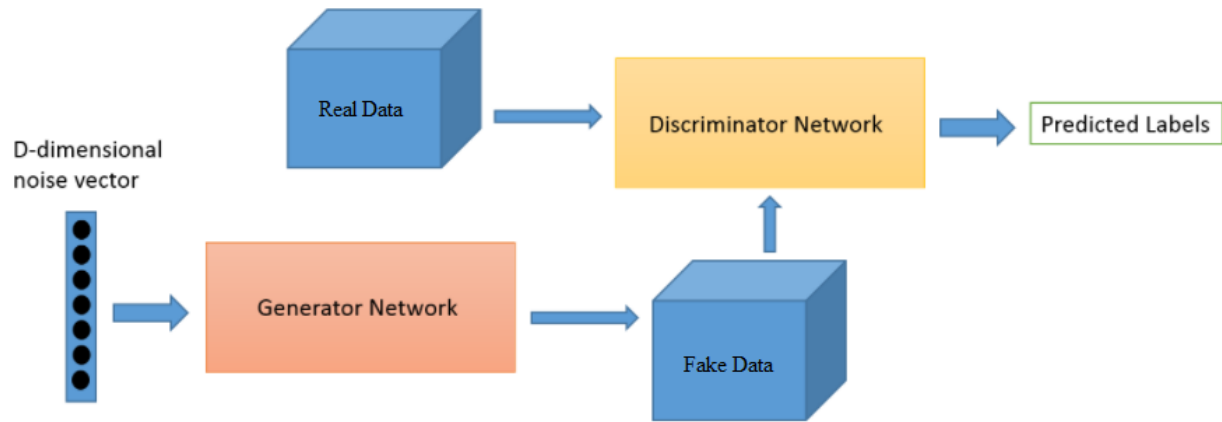


Figure 2.5: The process that GANs move in [6]

Mathematically, we can say that the goal of the discriminative network can be expressed by the equation 2.1 [7]

$$\max_{\phi} E_{x \sim P_{\theta}} [\log (1 - D_{\phi}(x))] + E_{x \sim P_r} [\log (D_{\phi}(x))] \quad \dots(2.1)$$

where \mathbf{D}_{ϕ} is an appropriate real-valued function parameterized by ϕ , and \mathbf{P}_{θ} denotes a data distribution that is mutually exclusive with \mathbf{P}_r (i.e., $\mathbf{P}_{\theta} \cap \mathbf{P}_r = \emptyset$) [7]. On the contrary, the function of the generative network is to convert the input \mathbf{z} that obeys the distribution \mathbf{P}_z into fake data \mathcal{X} that obeys the distribution \mathbf{P}_f . The goal of the generative network is to drive the distribution \mathbf{P}_f infinitely approaching the distribution \mathbf{P}_r . The Jensen-Shannon divergence can be used to measure the distance between two distributions; thus, the goal of the generative network can be essentially transformed to minimize the Jensen-Shannon divergence between \mathbf{P}_f and \mathbf{P}_r . If $\mathbf{P}_f = \mathbf{P}_r$, the Jensen-Shannon divergence is minimized, indicating the generative network perfectly replicates the real data distribution \mathbf{P}_r .

The goal of the generative model is expressed as:

$$\min_{\phi} E_{z \sim P_z} [\log (1 - D(G_{\phi}(z)))] \quad \dots(2.2)$$

where $G_\phi(\mathbf{z})$ denotes the fake data generated by the generation model according to the noise \mathbf{z} .

Here, the output of G is randomly generated with no control for what the desired output should be. To support this type of control, some tweaking can be made to the original GAN. The next subsection will talk about the conditional GAN which adds this ability.

2.2.3.5 Conditional GAN (CGAN)

CGAN is introduced by conditioning both discriminator and generator by feeding class labels. As seen in Figure 2.6, CGAN feeds the extra information \mathbf{y} to both the discriminator and generator. It should be noted that \mathbf{y} is normally encoded inside the generator and discriminator before being concatenated with the encoded \mathbf{z} and encoded \mathbf{x} .

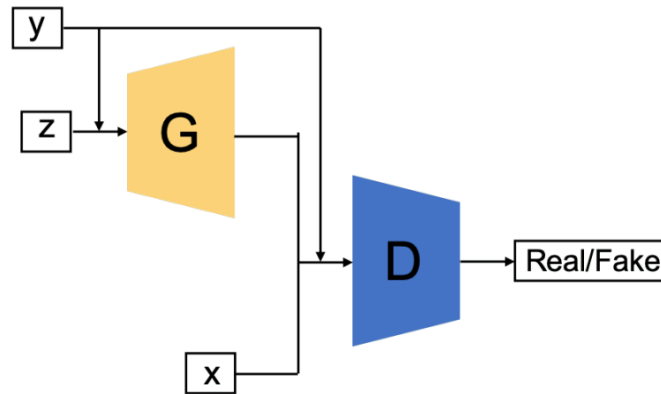


Figure 2.6: CGAN architecture

The loss function of CGAN is seen in equation 2.3, in which \mathbf{x} and \mathbf{y} are conditioned by \mathbf{z} . Benefiting from the extra encoded \mathbf{y} information, CGAN is not only able to handle unimodal image datasets but also multimodal datasets such as Flickr that contains labeled image data with their associated user-generated metadata (UGM) i.e., in particular user-tags, which brings GANs over to the area of multimodal data generation.

$$\min_G \max_D \mathbb{E}_{\mathbf{x} \sim p_r} \log[D(\mathbf{x}|\mathbf{y})] + \mathbb{E}_{\mathbf{z} \sim p_z} \log[1 - D(G(\mathbf{z}|\mathbf{y}))] \quad \dots(2.3)$$

2.3 Literature Review

This section is a brief discussion of previous work similar to what this project is doing.

With the rapid development of technologies and the rise of AI-based inventions, more researchers sought towards making machines that have the calligraphic capabilities that humans have.

A group of Chinese researchers from Xiamen University in China [7] have achieved a novel work in 2019 aimed to produce human-styled Chinese characters consisting of multiple parts in its special GAN model (i.e. Info-GAN). Based on some trajectory points to make a character, the generative part of the system uses stochastic policy gradients to sample the trajectories. These points are forwarded to the robot system which is a mechanical arm having multiple joints to control its freedom degrees, the robot draws the points in sequence considering other information like tilt and curvature. The discriminative part, which is a simple feed-forward neural network, takes a capture of the drawing and performs a binary classification on it. The system generates sub-drawings for a character, then these strokes are assembled to be one meaningful word as a character as in figure 2.7. Info-GAN was used for specifying the output stroke, and the system is updating its policy each time using policy gradients.

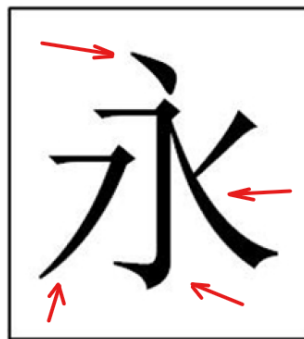


Figure 2.7: Substrokes assembling [7]

Another similar work that inherits some of the previously mentioned work and references was done by a group of researchers in the Computer Vision and Recognition Center [10] at Cornell University in 2020. This work is only software-based using a Convolutional Neural Network (or CNN) as a generator for creating understandable English word images. The discriminator is also a CNN model for binary classification. The system also used policy gradients as the work before.

What makes this work so ahead of the competition is the attention to other aspects in the handwritten data itself such as style encoding, textual content encoding on a character-wise field, and global-string-wise. Making this work capable of generating indistinguishable handwritten calligraphy images, and the ability of dynamic font styling for the same text.

The work in [7] was enhanced in [16] by introducing a Long Short-Term Memory network (or LSTM) to be used in the generative part of the system, which made a noticeable improvement over what was previously included, at the cost of being more complicated. It samples the trajectory points and then passes them to the arm-based robot to draw them by sequence, when each two-point line is drawn, the system captures it and passes it to the LSTM again to update its policy and begin sampling the next trajectory point which the robot will draw. When all the trajectories are drawn, the strokes are passed to the discriminative part which is a CNN. The system as its predecessor is also drawing sub-strokes and assembling all of them to produce one meaningful word as a character. It also uses policy gradients to update its policy while performing, and the model itself is also Info-GAN to specify the output strokes.

Another work that depended on their usage of long-short-term memory networks similarly to [16] is [40] by Alex Graves and others in 2014. They worked on handwriting synthesis, i.e. to generate handwritten versions from the digital text given by humans as input. The work showed how LSTM recurrent neural networks can be used to generate complex sequences with long-range structures, all by predicting one data point at a time. As a result, the outcome of their system is sequences of trajectory points to be converted into a digital image of the handwritten word or phrase, meaning that no robots were used in their work. Also here, no network to judge the output against the generating one.

Our work will be based on the regular GAN architecture and it is borrowing some parts used in some of the above works like the policy gradients, the trajectory points sampling approach, and the CNN discriminative model, all of being discussed in detail in chapter 3: Design. Our project will work for numbers and will integrate with a robot in the shape of a plotter unlike the arm robot used in the first and last mentioned works.

We are going to add a contribution to the Ruiqi Wu et al. [7] work. Since the space for possible outputs for the generator is extremely high, we proposed a solution that will decrease and wrap the space of possible output to a much more reasonable number. As an example, if we had a

window of 28*28 and we needed to generate the number '2' using five strokes, we need to draw five points each having the following probabilities respectfully:

$$\binom{784}{6} = 784 * 783 * 782 * 781 * 780 * 779 \approx 2 * 10^{17} \quad \dots(2.3)$$

To minimize the walking through this huge number of possibilities, we tried to localize each point in its respectful subwindow given the label of the drawing based on the frequent occurrence of each point in a given region, the outcome of this method is a much smaller space of possibilities to draw a number. Continuing on our example above, if we bordered each point with a 5*5 window, the probabilities will not exceed $2*10^8$ possible outputs.

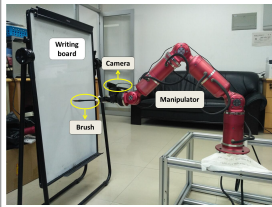
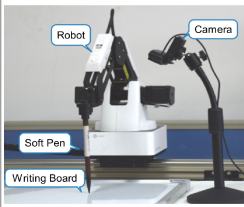
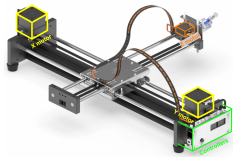
Note that the general and efficient way to find the most frequent regions can be achieved by analyzing a hand-written-word dataset that is represented using a sequence of points. However, due to hardware limitations and the lack of popper resources, we will build these localizations using hand-designed regions.

In addition to reducing the generator output space, we controlled the output of the generator by feeding the generator with the label of the desired digit output.

Moreover, we are going to use and enhance Alex Graves's works [40] by involving the calligrapher robot in the loop of generation as a contribution. Instead of generating digital images as an output, we are going to draw the expected output on surfaces using our robot within a complete and usable system.

Table 2.1 summarizes the differences between our works and previous works. Also, it briefly shows our contributions and enhancement.

Table 2.1: Comparison between selected works and our work

| Work | Ruiqi Wu et al. [7] | Alex Graves et al. [40] | Fei Chao et al. [16] | Our Works |
|--|--|---|---|--|
| <p>Intelligent Model Type</p> <p><i>Generator as "G"</i> <i>Discriminator as "D"</i></p> | <p>G: Stochastic Policy Gradients</p> <p>D: Simple feed-forward neural network</p> | <p>Only G: Modified LSTM that accepts sequences of points</p> | <p>G: LSTM</p> <p>D: CNN</p> | <p><u>Part1:</u></p> <p>G: Stochastic Policy Gradients</p> <p>D: CNN</p> <hr/> <p><u>Part2:</u></p> <p>G: Modified LSTM that accepts sequences of points</p> |
| <p>External Hardware</p> |  | <p>Did not use external hardware, only software</p> |  |  |
| <p>Output</p> | <p>Six Chinese character strokes</p> | <p>Real handwritten word images</p> | <p>Six Chinese character strokes</p> | <p>Numerical digits strokes & English handwritten words strokes</p> |

2.4 System Hardware Components

This section presents the hardware components in this project, provides a brief description, displays the important specifications, and explains the usage of each component. It also demonstrates the alternative hardware kits for robotic plotters with their components and the alternative controller boards. Other alternatives for each component might be available and might not be, so we are going to mention them if they exist.

2.4.1 Controller boards

This subsection lists the controller boards that are needed in this project. The controller boards are needed to run and execute deep learning models. Also, they are used for controlling the robotic plotter components.

2.4.1.1 Arduino Uno

Arduino Uno (figure 2.8) is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator (CSTCE16M0V53-R0), and a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller. It can be turned on by connecting it to a PC through a USB cable for power and data transfer, or it can be powered alone with an AC-to-DC adapter.

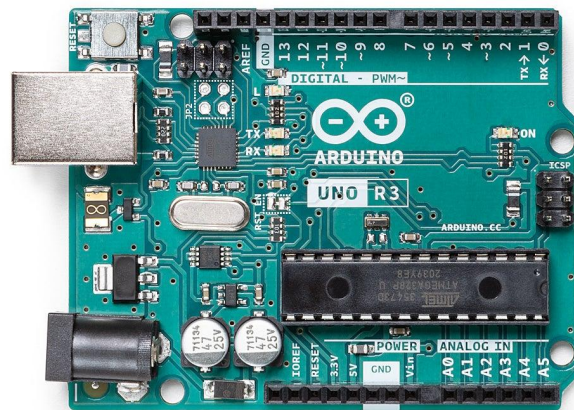


Figure 2.8: Arduino UNO [22]

The UNO version is the one that will come with the plotter hardware, and we think this choice is the best fit since another board which is the CNC shield, is designed specifically to be mounted on top of the UNO, possessing all pinholes the UNO board have.

2.4.1.2 Raspberry Pi Model B

A Raspberry Pi 4 (figure 2.9) is a general-purpose computer, that provides a set of general-purpose input/output pins that allows you to control electronic components. The main operating system that installs on Raspberry Pi is Linux-based systems i.e. Debian. It provides

ground-breaking increases in processor speed, multimedia performance, memory, and connectivity compared to the prior-generation Raspberry Pi 3 Model B+ while retaining backward compatibility and similar power consumption.

This product's key features include a high-performance 64-bit quad-core processor, dual-display support at resolutions up to 4K via a pair of micro-HDMI ports, hardware video decode at up to 4Kp60, up to 8GB of RAM, dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet, USB 3.0, and PoE capability (via a separate PoE HAT add-on).

The dual-band wireless LAN and Bluetooth have modular compliance certification, allowing the board to be designed into end products with significantly reduced compliance testing, improving both cost and time to market.

Raspberry Pi has a slightly powerful processor compared to Arduino that allows us to run multiple programs and control many electronic components simultaneously [19].

We chose this model over the previous one (Pi 3) since it has a better computational power which allows for better performance and quick response time. Note that this microcomputer is not included in the robot hardware bundle and it is something to be purchased separately.

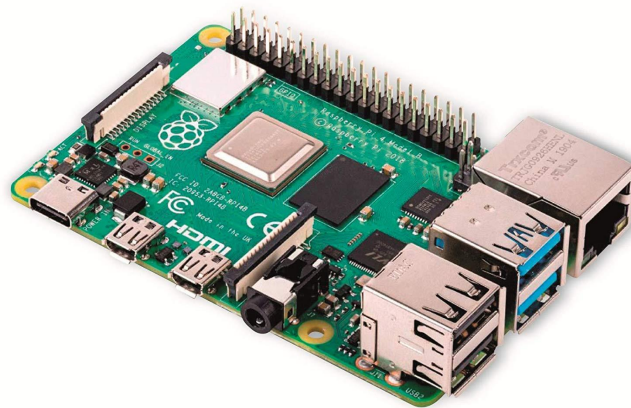


Figure 2.9: Raspberry Pi 4 Model B [19]

Both Arduino and Raspberry solutions will be used in this project. The Arduino microcontroller will be responsible for managing the robot hardware, and the raspberry pi will host the learning model and do its required computations.

To control the raspberry pi using a remote device, there are two technologies mainly used. The Virtual Network Computing (VNC) and Secure Shell Protocol (SSH) are described briefly in the next two sub-sections.

2.4.1.2.1 VNC Viewer

Virtual Network Computing is a graphical desktop sharing system that uses the Remote Frame Buffer protocol to remotely control another computer. It transmits the keyboard and mouse input from one computer to another, relaying the graphical-screen updates, over a network [41].

2.4.1.2.2 SSH

The Secure Shell Protocol is a cryptographic network protocol for operating network services securely over an unsecured network. Its most notable applications are remote login and command-line execution. SSH applications are based on a client-server architecture, connecting an SSH client instance with an SSH server [42].

2.4.2 Robotic components

This robotic plotter will allow the user to reflect what the system is generating, and pass it to be drawn to a plain paper sheet or a touch-interactive surface like tablets. This subsection contains a CNC shield, servo motors, stepper motors, stepper drive chip, 12V adapter, and fan.

2.4.2.1 CNC V3 Shield

The CNC shield (figure 2.10) can plug on top of an Arduino requiring no external connections and wiring. There are 4 slots on the board for plugging in a stepper motor drive module which can drive 1 stepper motor each. Controlling each step stepper motor requires only two I/O pins on the Arduino [25].

2.4.2.2 MG996R Servo Motor

A servo motor (figure 2.11) is a linear actuator or rotary actuator that allows for precise control of linear or angular position, acceleration, and velocity. It consists of a motor coupled to a sensor for position feedback. It also requires a relatively sophisticated controller, often a dedicated module designed specifically for use with servomotors [27].

Our project will use one servo motor for robot movements on the Z-axis, and since these movements require speed and accuracy, the servo motor will be a better fit over other motors

2.4.2.3 NEMA17 Stepper Motor

Stepper motors, shown in figure 2.12, are DC motors that move in discrete steps. They have multiple coils that are organized in groups called "phases". By energizing each phase in sequence, the motor will rotate, one step at a time [28].

We will use two stepper motors for controlling robot movements in both the X and Y axes.

2.4.2.4 A4988 Stepper Driver Chip

The A4988 driver is a commonly used bipolar stepper motor driver that could provide up to 2A current per coil from 8V to 35V. These drivers are used to drive NEMA 17 motors on the Mill One [30].

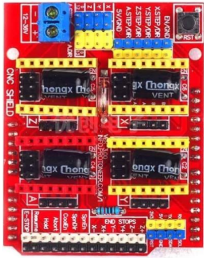
2.4.2.5 12V adapter

Power supply AC Adapter. Input 100-240V / - 50-60Hz/ 200mA, and output 12V-2,5A

2.4.2.6 Cooling fan

Cooling fan DC 5V, 0.13A for Arduino.

It is **important** to note that all components mentioned in this section are all **part of the robotic plotter hardware bundle** which we decided to buy, and since they are not specific in naming or manufacturing to a certain company as each of them has its standard codename, blueprint, and manual sheet, we had to mention them here.



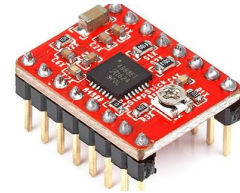
*Figure 2.10:
CNC V3 Shield [24]*



*Figure 2.11:
MG996r
Servo motor [26]*



*Figure 2.12:
NEMA 17 Stepper
motor [29]*



*Figure 2.13:
Stepper
driver chip [30]*



*Figure 2.14:
Cooling fan [31]*



*Figure 2.15:
12V adapter [32]*

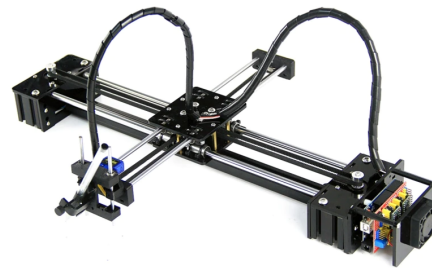


Figure 2.16: LY plotter robot [21]

2.5 System Software Components

In this section, we will display the main software components for the project and some detailed descriptions of them.

2.5.1 Python

Python is a very popular high-level interpreter-based programming language. It was first released in 1991 but continued to evolve to this day. Python programs are very simple and easy to read. Few lines in python often correspond to tens of lines in other programming languages like C. This means that writing programs in Python is time-saving practice. In Addition, programs written in Python are shorter in length and more readable. Furthermore, Python scripts can run on many platforms including Windows, Linux, and Unix.

Due to its popularity, Python has a very large community and a lot of resources. Python users can easily add code packages to their programs simply by importing them at the top of the script. Programmers can also download non-standard packages with PIP, which is a package manager for python [33].

2.5.2 TensorFlow

TensorFlow is an open-source platform that was developed by Google's Brain team, that contains a comprehensive, flexible ecosystem of tools, libraries, and community resources. This tool is widely used within the community, where huge contributions to make such models and algorithms more and more accurate and efficient will help us out during our work on this side of the project [9].

2.5.3 Arduino IDE

Arduino IDE is an environment that could be used for writing the code and uploading it into Arduino Uno. By default, any Arduino system can take a C++ code with specific defined functions and structure to achieve any kind of task, like continuous repetitive work inside a loop function and pre-work definitions in a setup function.

Chapter 3

Design

3.1 Overview

This chapter discusses the overall design of the handwritten calligraphy robotic system and the way its components are integrated, showing some diagrams for the design, in addition to some details about the software our system is using.

3.2 Detailed Description of the Design

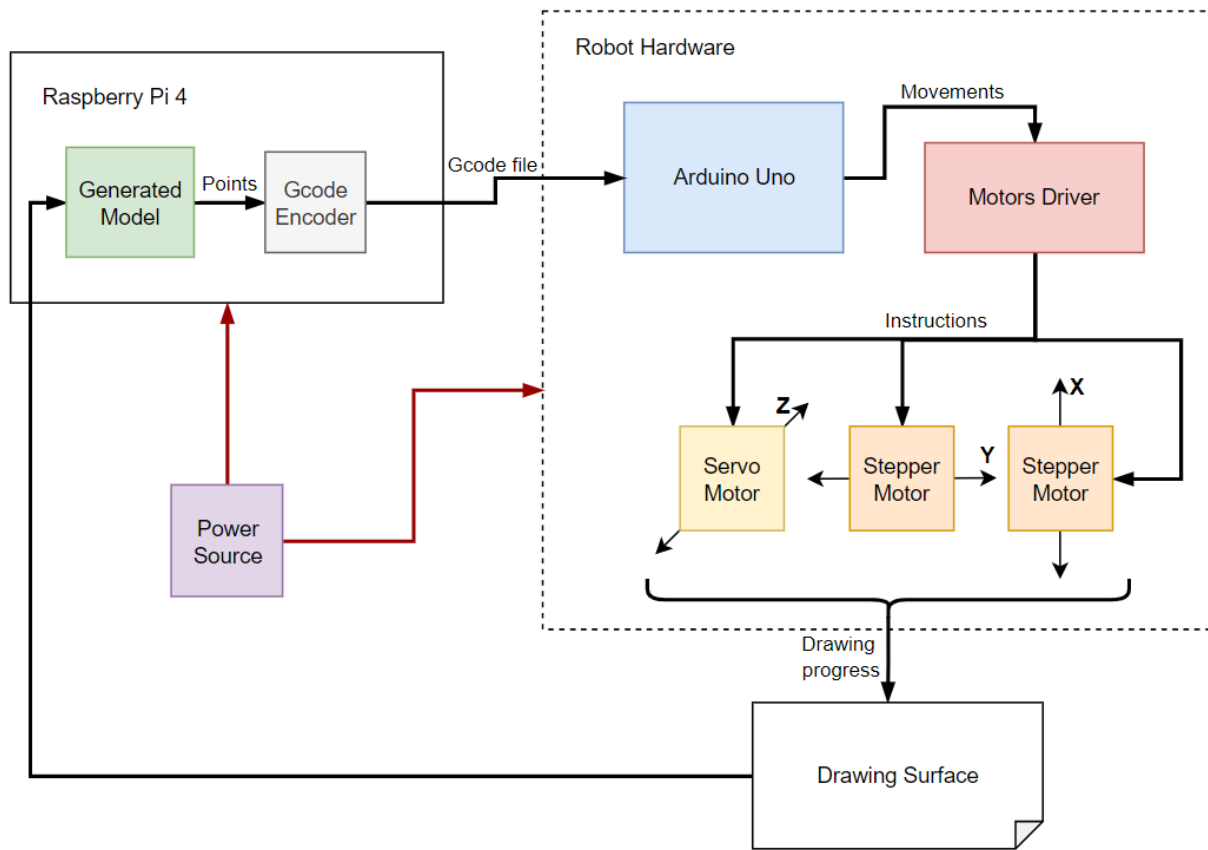


Figure 3.1: Block Diagram for the different general components in the robotic system

Figure 3.1 shows all general hardware components that were specified in chapter 2 of the system, which is briefly connected here to demonstrate the relationship between every component and its neighboring components. The model, as well as the G-Code encoder, are going to be hosted on the raspberry pi 4 microcomputer. This microcomputer will be connected to the Arduino Uno microcontroller, which is communicating with a motors driver interface (i.e the stepper motor drivers and the CNC shield driver for the servo motor) connected to the different motors that

construct the basic calligraphy robotic system movements in the three-axis as shown in the above figure. All electronic components are supplied with power from an external power supply component (i.e the brick that delivers power to the robot components, as well as a 5V power source through a normal adapter to each of the microcomputer and microcontroller parts). The capturing module is also supplied and it is directly connected with the raspberry pi to send captured strokes to the model to complete the robot’s objective loop.

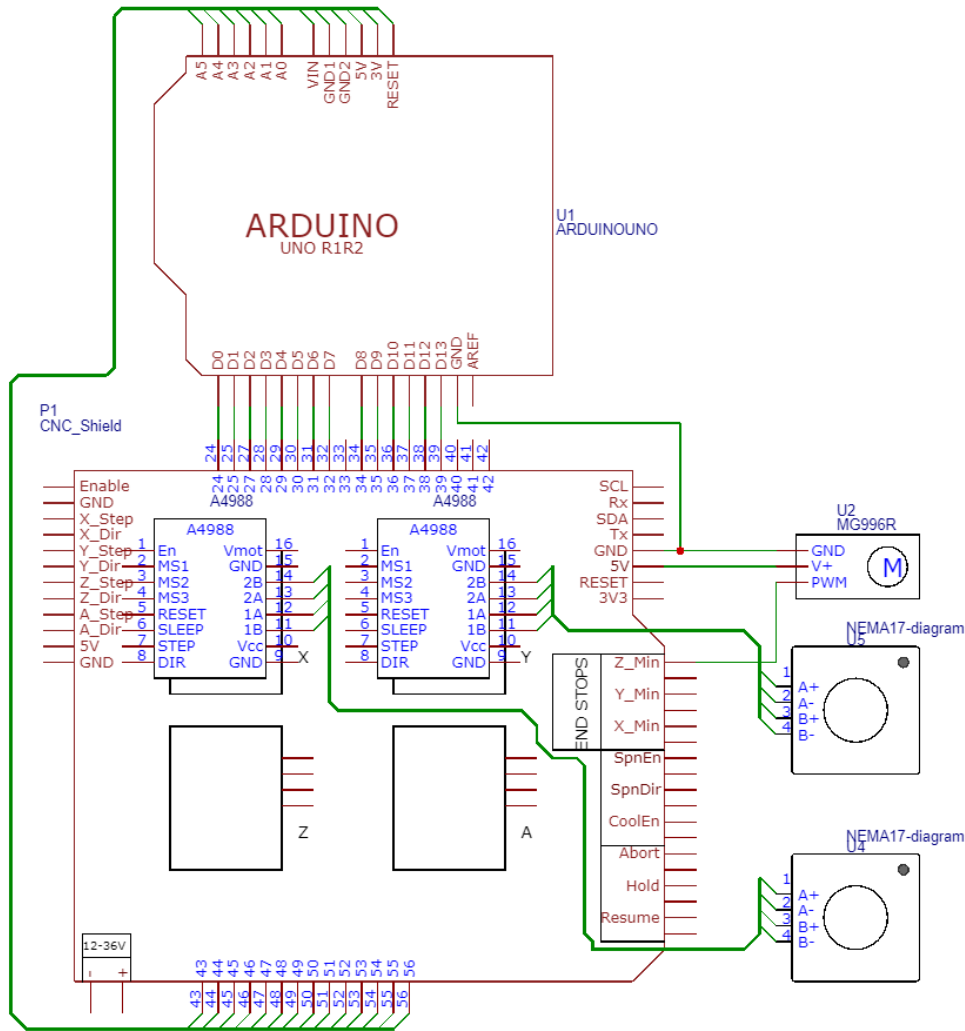


Figure 3.2: Schematic Diagram of the Robotic plotter interface

In figure 3.2, the motors used in the robotic plotter system are connected to their motor driver chips respectively, these chips are mounted on the CNC shield driver chip where each slot represents the interface for which movement the shield will instruct the stepper motors with. Currently, the first stepper motor labeled U4 is connected to its driver mounted on the X-axis slot

on the shield chip, the same goes for the other motor (U5). As for the Z-axis, since the required movement on the vertical axis needs to be quick, accurate, and not wobbly, the robot needed the servo motor because it is faster, more accurate, and stops with no jiggling. For that, the servo motor is directly connected to the shield chip.

The CNC shield with all of its attachments is mounted on top of the Arduino board, connecting its rear-faced pins directly to the Arduino pin holes accordingly. It is noted that the design of the CNC V3 shield chip is configured to be mounted on top of the Arduino UNO board with correct pin positioning.

3.3 Design Options

In this section, we will be going over each component in our system and showcase its available options and what we ended up choosing from these options.

3.3.1 Hardware Design Options

This robotic plotter will allow the user to reflect what the system is generating, and pass it to be drawn to a plain paper sheet or a touch-interactive surface like tablets. The movement of this robot is so wide open upon the rectangular area it is covering, having three degrees of freedom (or DOF). The first two DOFs are for movements in the X and Y axes respectively, the third one is for the subtle elevation the robot can do for the pen, this is for sequence cutting and multi-strokes drawing. There is another movement for rotating the pen along its vertical axis, but it is fixed at setup time. All the previous DOFs are changing with the input it takes at a time. The robot can accept a format of the drawing called GCode, which is a representation of the sequence this robot has to follow to draw the strokes. Normally, the drawing itself is converted to a GCode format and then passed to the robot to move the different parts and draw strokes of the drawing while doing so.

This plotter is the robot that will be involved in the learning process, taking input as a GCode from the GAN model, and outputting it on a tablet as a drawing using the attached compatible pen on its arm. The robot plotter is consisting of multiple moving parts, the part of moving the pen in a fixed width to height ratio is the two stepper motors that have four metal rods and moving specific strips in a loop, every two rods go parallel to each other to construct a linear

route which allows the other two rods that are at a right angle to the first two to move in. This setup is covering an entire square of the reachable area under it. An ordinary computer is responsible for holding the model and feeding the robot with compatible data, as well as retrieving information from it. Figure 3.3 shows exactly the robot's components when assembled.

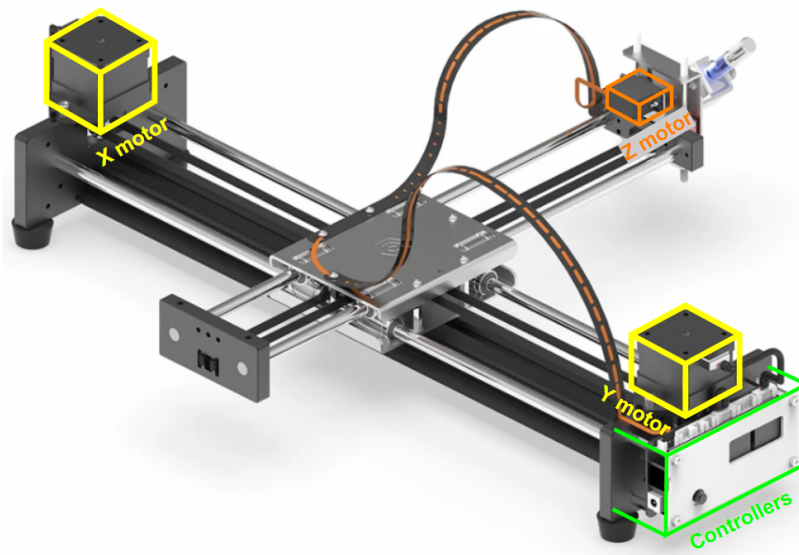
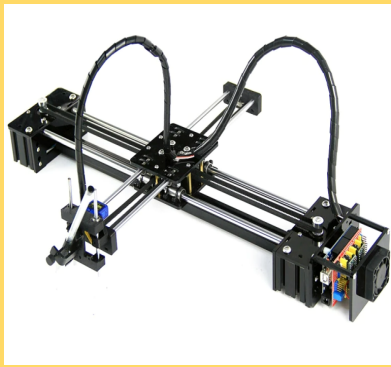
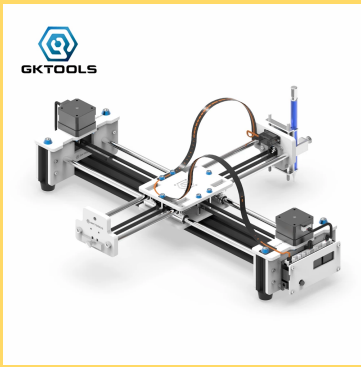


Figure 3.3: Robot Components when assembled [39]

The majority of plotters on the internet have typically the same form factor, as well as the component themselves. So there are no issues with picking a random one and starting work with it, but due to manufacturing material differences, size differences, and type differences (i.e. engraving or printing), we ended up comparing between a selective hand of choices that varies on some other specifications, but we think any of them is going to fulfill the requirements. It was a challenge to choose the right one. The following table showcases a comparison between the options we sampled and what plotter hardware we chose at the end.

Table 3.1: Comparison between Shenzhen Liyang LY machine [21] & GKDraw X3 [39]

| |  Shenzhen Liyang LY machine (selected) |  GKDraw X3 |
|-------------------------------|--|---|
| Manufacturer | Shenzhen Liyang CO. | GKTools |
| Working Area | 490mm × 390mm | 230mm × 315mm |
| Coordinates | X, Y, and short Z axes | X, Y, and short Z axes |
| Power | DC 12V, 5A | DC 12V, 5A |
| Cross-Platform Support | Windows, Mac, & Linux | Windows, Mac, & Linux |
| Official Firmware | Available, with open source | Available, with open source |
| Price | \$75 | \$177 |

3.3.2 Software Design Options

Current trends in the AI field are using the LSTM networks entirely to build handwritten intelligence systems like [23]. To mimic a specific style using LSTM, we need the exact sequence of points for any word written having the needed style. That sequence of print represents the training dataset. However, there are many cases in which the only available dataset for a particular style contains images only. Hence, our move is to use the GAN approach to mimic the mentioned case.

This section will discuss the options available for us when we decide to design the software part of the system.

3.3.2.1 The GAN Approach Overview

As shown in figure 3.4, on the software side, there are multiple components to look for which combined are called the GAN model, these components are the generator, the discriminator, and the distribution network. The points generator network is an MLP CNN network, which can generate sequences of trajectory points. These points can be fed into the next component which is a GCode encoder, it takes the points by order and encodes them to produce a set of instructions to be given to the plotter hardware in the format of a GCode file. The robot takes the GCode file and starts drawing trajectory points inside it. A capturing module submits images taken from the handwritten text the plotter draws to the discriminator network, which is also an MLP CNN network that does a zero-one comparison between what the model learned and what the plotter has submitted. It all starts with random weights and not accurate probabilities, so a points distribution network is making sure of updating these values whenever the discriminator is doing a comparison.

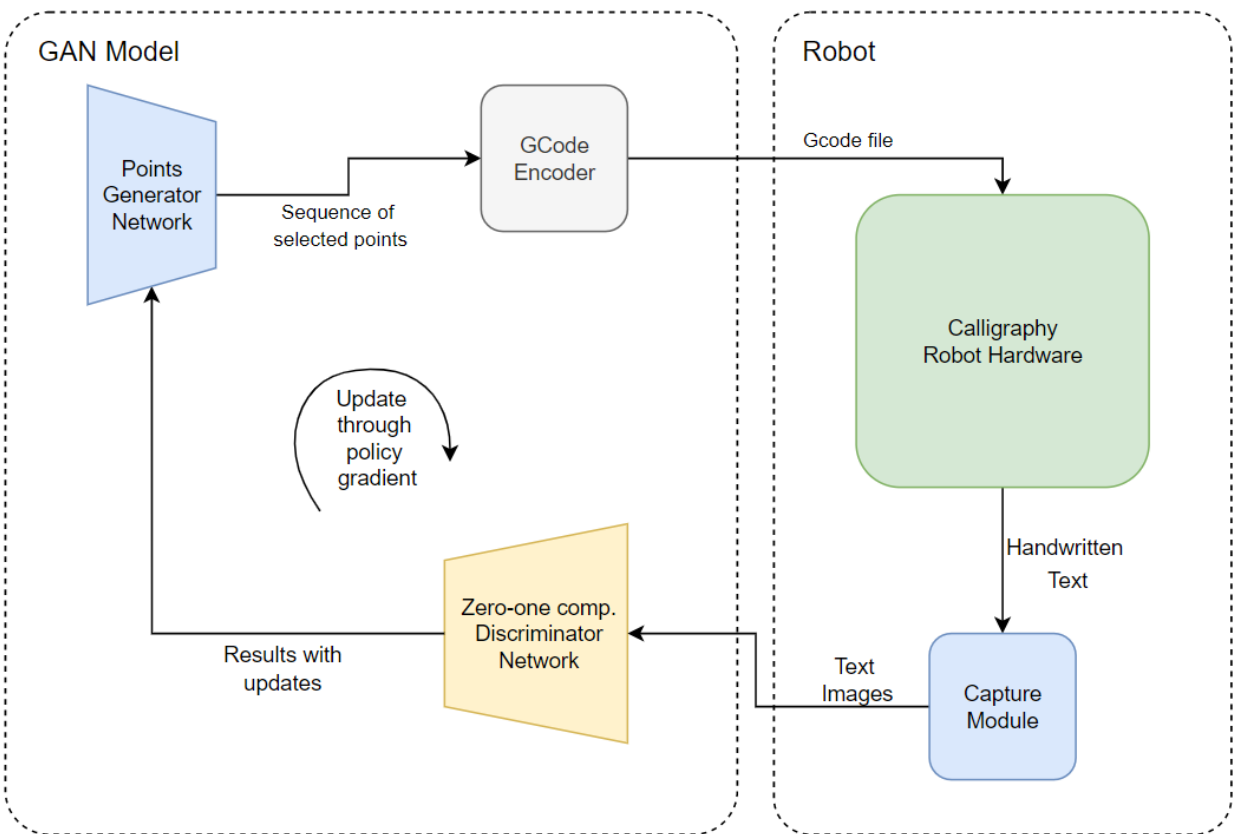


Figure 3.4: Detailed View of the GAN's Submodels for the Robotic System

The proposed approach consists of three parts: 1) character generative module, 2) character discriminative module, and 3) training module. The entire method will be built upon the GAN. The task of the discriminative module is to distinguish samples from the character generative module or the real training data; meanwhile, the character generative module's task is to maximally confuse the discriminative module. Moving forward, we will refer to the generator as \mathbf{G} , the discriminator as \mathbf{D} , the input data as \mathbf{X} , and the random number set as \mathbf{Z} .

Thus, the training objectives of the approach are summarized to 1) train the discriminative module to maximize the probability of the real character data and minimize the probability of the character image written by the robot; and 2) train the generative module to minimize the probability that the discriminative module recognizes the robotic written images. In the character generative module, a generative network, \mathbf{G} , uses random numbers as input to produce probability distributions of character trajectory points. Then, the calligraphy robot applies a sampling method to obtain the character's position information from the probability distributions. From this, the robot uses the obtained position to write the character on a touch screen, then the tablet captures the written character. The discriminative network, \mathbf{D} , receives character images from the tablet, either captured from the written character by the robot or sampled from calligraphy textbooks, and produces a discriminative result for each image. The original training approach proposed in info GAN used the gradient descent method.

However, the gradient descent method must face a non-differentiable problem during the training phase in the proposed framework, due to the involvement of the robotic manipulator in the proposed approach. Such a problem cannot be solved by the backpropagation algorithm, and thus the policy gradient typically used in the reinforcement learning algorithm is applied here to train the framework, motivated by the work of Yu et al. [11].

For each output of the \mathbf{G} network, the discriminative result generated by the \mathbf{D} network indirectly reflects the performance of the \mathbf{G} network. In this case, outputs of the \mathbf{D} network are used as rewards for the robotic actions, which are generated by the \mathbf{G} network. Therefore, the training objective of the \mathbf{G} network is changed to obtain maximum rewards in the proposed system. The \mathbf{G} network must learn to increase the occurring probability of outputs such that the writing performances can be improved. Based on this consideration, if a character writing image of the robot has better performance determined by \mathbf{D} , the \mathbf{G} network must increase the probability of

the robotic trajectory points. The detailed implementations of these three modules are described below.

3.3.2.1.1 Character Discriminative Module

The character discriminative module will be established by a convolutional neural network (CNN). CNN is chosen over other machine learning approaches because it takes just the image's raw pixel data as input and "learns" how to extract these features instead of preprocessing the data to derive features like textures and shapes. The network will consist of an input layer, hidden layers, and an output layer. The details of the input layer and hidden layer will be set in the next semester, and the dimension of the output is 1 since it is a binary classifier. The network's input data \mathbf{X} will consist of two types of images: 1) real character images, \mathbf{X}_{real} , and 2) "fake" character images, \mathbf{X}_{fake} , written by the calligraphy robot. The fake images will be written and captured on a tablet. The output values predict the probability that \mathbf{x} came from the \mathbf{X}_{real} data distribution against that from the robot-generated images.

3.3.2.1.2 Character Generative Module

The character generative module has a Gaussian noise input, which produces random numbers to seed the generative module. In this project, a feed-forward neural network is adopted to implement the character generative module. The G network also has an input of \mathbf{G} as an array of generated random numbers, \mathbf{Z} , the hidden layers, and the output of \mathbf{G} contains a probability distribution for the trajectory point position distribution. When all the probability distributions are generated, the random sampling method is used to generate some trajectory points from the probability distributions. After sampling, the position information of the points is used to generate G-Code instructions which will be sent to the robotic system.

Pseudo-code for the process

```
real_images := real character images  
Z := random number  
Tn := number of needed trajectories to draw characters  
  
while (GAN not converged)  
  
    fake_images := []
```

for G-iteration

Z := generate new random Z
INPUT Z into G
trajectorie_list := []
trajectories := OUTPUT of G

for (t := 1 to Tn)
sampled_trajectorie := SAMPLE(trajectories)
trajectorie_list.add(sampled_trajectories)
end

g_code = G_CODE_CONVERTOR(trajectorie_list)
fake_image := Draw g_code by robotic system
UPDATE policy gradient parameters in G
fake_images.add(fake_image)

end

combine fake_images with real_images
TRAIN D with combined images

end

GAN's method as in suggested papers that have done similar work, The traditional GAN uses the back-propagation algorithm to update model parameters, and the generator updates its parameters by back-propagating the discriminator's predicted error. However, in the proposed model, the output of the generator cannot be directly used as the input of the discriminator, as the model itself is non-differentiable. To address this issue, the policy-gradient method employed in the work is introduced in our work to train the GAN model. Briefly, the policy-gradient method is a basic training mechanism in the reinforcement learning algorithm that can implement the error backpropagation of the discriminator on the generator with the participation of the robot [7].

We preferred to choose the policy-gradient method over the traditional policy of the normal GAN model because it allows the robot to discover other possibilities of achieving better-handwritten work (i.e. stochastic approach), then just going through one possible route of

reaching the goal (i.e. succeeding in making the discriminator confused), which is known as the deterministic approach.

3.3.2.2 The LSTM Approche

Recurrent neural networks (RNNs) are a rich class of dynamic models that have been used to generate sequences in domains as diverse as music, text, and motion capture data. RNNs can be trained for sequence generation by processing real data sequences one step at a time and predicting what comes next. As we previously discussed LSTM as a unique type of RNN in chapter 2, this network is explicitly designed to avoid long-term dependency problems. So we needed it because remembering information for long periods is particularly its default behavior.

Assuming the predictions are probabilistic, novel sequences can be generated from a trained network by iteratively sampling from the network’s output distribution, then feeding in the sample as input at the next step. In other words, by making the network treat its inventions as if they were real, much like a person dreaming. Although the network itself is deterministic, the stochasticity injected by picking samples induces a distribution over sequences. This distribution is conditional, since the internal state of the network, and hence its predictive distribution, depends on the previous inputs.

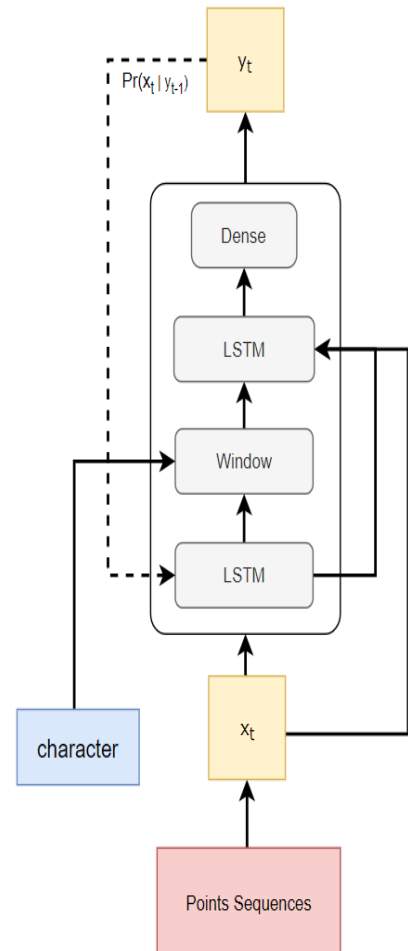


Figure 3.5: Handwritten synthesis network architecture

To summarize in a couple of words, the LSTM network is thinking of possibilities, and not simply having a direct way to do them.

Handwriting synthesis can be described as the generation of handwriting strokes for a given text. Figure 3.5 illustrates the network architectures used for handwriting synthesis. The hidden layers

are stacked on top of each other, and each one is feeding up to the layer above. There are skipped connections from the inputs to all hidden layers and from all hidden layers to the outputs.

This architecture contains a window layer, which is convolved with the text string and then fed in as an extra input to the synthesis network. The parameters of the window are output by the network.

The input of the model is represented by a one-hot-encoded vector as a character sequence. And the output of the model can be described by \mathbf{x} , \mathbf{y} , and \mathbf{z} . Where \mathbf{x} and \mathbf{y} represent the coordinates of the points, and \mathbf{z} indicates whether the character ended or not.

The loss function can be calculated as in equation 3.1:

$$\mathcal{L}(\mathbf{x}) = -\log \Pr(\mathbf{x}|\mathbf{c}) \quad \dots(3.1)$$

Where \Pr is the predictive distribution as in equation 3.1:

$$\Pr(\mathbf{x}|\mathbf{c}) = \prod_{t=1}^T \Pr(x_{t+1}|y_t) \quad \dots(3.2)$$

Where:

- \mathbf{x}_{t+1} is the possible next input.
- $\Pr(\mathbf{x}_{t+1} | \mathbf{c})$ is the predictive distribution of the next possible input given the character \mathbf{c} .
- \mathbf{y}_t is the current possible output.

Chapter 4

Software & Implementation

In this chapter, we demonstrate the various steps we came across to implement both the hardware side and the software side of our project, we briefly mention some of the key takeaways we saw during the process on each side.

4.1 Model and Software Configuration

4.1.1 RaspberryPi Software

We started with installing the RaspbianOS image from the official RaspberryPi imager application, then installed that image into an SD card and mounted it to our Raspberry Pi.

However, the raspberry pi 4 we have supports the RaspbianOS 64bit, we installed and used the 32bit distribution due to some compatibility issues with the robot's official driver.

We accessed the raspberry pi using SSH protocol after enabling it using an external monitor, then we activated the VNC server application inside it to have a remote graphical connection to it from a computer. We booted into the system and then installed the development tools we needed such as IDE, JDK, and the interpreter of python 3.7.

We set up the two sub-projects as described in the next subsections, then we disconnected the VNC connection and we connected the raspberry pi to an android phone. Using the RaspController mobile application we controlled the raspberry pi and ran our learned models by launching terminal scripts via SSH protocol.

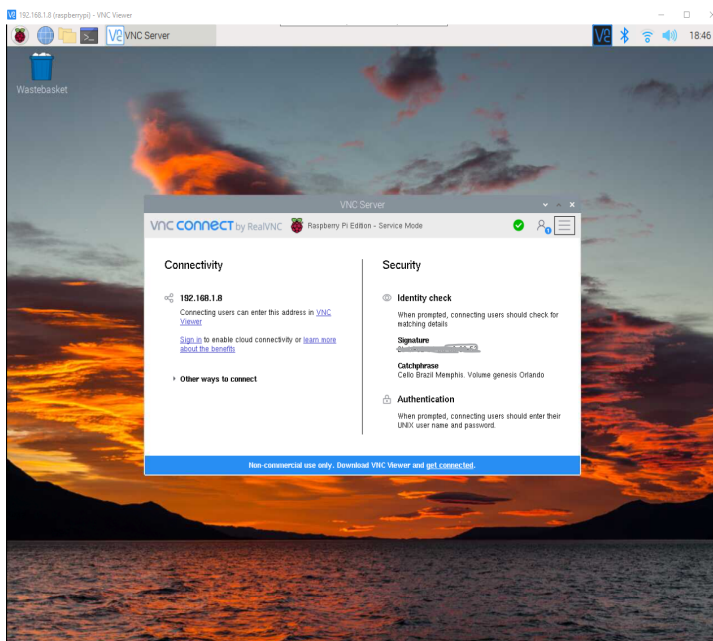


Figure 4.1: RaspbianOS GUI through VNC application

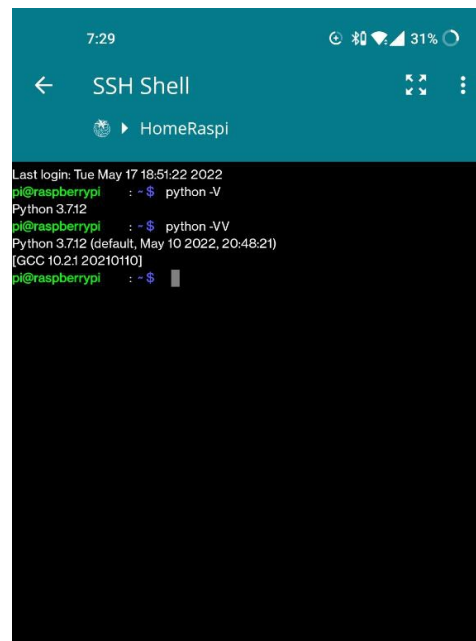


Figure 4.2: SSH Shell from mobile

4.1.2 Software Building

This section demonstrates the implementation of the GAN and LSTM models in two sub-sections. Each sub-section contains the used technologies and the model architecture.

4.1.2.1 GAN Architecture

The GAN architecture includes two models: the generative and discriminative models. The task of the generative module is to generate stroke trajectory points based on certain given input information that includes a random noise \mathbf{Z} , and the label of the desired digit. On the other hand, the task of the discriminative module is to distinguish between the generated digits and the actual training digits, as well as to give feedback for the generative model to improve the generated digits.

4.1.2.1.1 Generative Model

This model will have the following network structure: first of all, the input layer starts with 128 neurons for noise, with additional 10 ones representing the labels for numbers in a one-hot-encoder manner. Secondly, the intermediate hidden layer consists of $(5 * 138) = 690$ neurons, the reason we multiply the sum of the input layer by 5 is due to the need for separation between each of the five trajectories we want to generate so that we make sure each trajectory's network does not affect others. Finally, the output layer contains five parallel layers of $28 * 28 = 784$ neurons, each one is mapped to the corresponding point in the space of the possible points.

Because the output space is too large to the extent we could not handle it using whatever resources in our hand, we built hand-design templates to localize the searching space for each trajectory of each label based on our knowledge of writing digits. For example, the search space for the number zero '0' in the hand-design template is 864,000 possibilities. On the other hand, the same number when not using the dedicated template can reach up to $2 * 10^{17}$ possibilities as discussed in chapter 2.

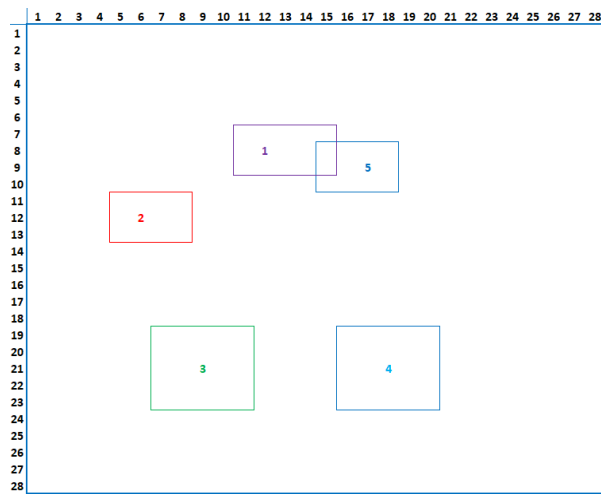


Figure 4.3: localized regions for the number '0'

```
space = [
# Each array in the space represents the square diagonal heads of template where trajectory points may take place
#
# x1 x2 y1 y2  x1 x2 y1 y2  x1 x2 y1 y2  x1 x2 y1 y2  x1 x2 y1 y2
[[ 7,  9, 11, 15], [11, 13,  5,  8], [19, 22,  7, 11], [19, 22, 16, 20], [ 8, 10, 15, 18]], # tj0
[[ 5,  7, 12, 16], [ 9, 11, 13, 15], [13, 15, 14, 15], [17, 19, 13, 15], [21, 23, 12, 16]], # tj1
[[11, 13,  5,  8], [ 5,  7, 12, 16], [11, 13, 18, 22], [20, 24,  5,  8], [20, 24, 18, 22]], # tj2
[[ 5,  7,  5, 11], [ 7, 11, 18, 22], [13, 15, 10, 15], [17, 19, 18, 22], [20, 24,  5,  8]], # tj3
[[ 5,  7,  8, 10], [15, 18,  8, 10], [15, 18, 18, 22], [ 5,  7, 18, 22], [24, 26, 18, 22]], # tj4
[[ 5,  7, 18, 22], [ 5,  7,  8, 10], [13, 15,  8, 10], [18, 20, 18, 22], [24, 26,  8, 10]], # tj5
[[ 8, 10, 13, 18], [20, 22,  5,  7], [24, 26, 10, 12], [20, 22, 14, 16], [18, 20,  7, 10]], # tj6
[[10, 12,  6,  8], [ 6,  8,  9, 11], [ 6,  8, 18, 20], [10, 12, 20, 22], [22, 26, 14, 18]], # tj7
[[ 6,  8,  8, 10], [ 6,  8, 18, 22], [22, 24,  8, 10], [22, 24, 18, 22], [ 6,  8,  6,  8]], # tj8
[[13, 15, 16, 18], [13, 15, 10, 12], [ 7,  9, 12, 14], [ 7,  9, 16, 18], [22, 24, 16, 18]], # tj9
]
```

Figure 4.4: Space localizing implementation for numbers 0 to 9

Once we figured out the huge possibilities space number, we started feeding the TensorFlow-implemented network hosted on a computer instance having 8 vCPUs, 8 GB of

RAM, and an Nvidia Quadro M4000 with the training data. The following table 4.1 presents the results of the initial testing.

Table 4.1: Initial training comparison between template-free and template usages

| | Without using templates | Using templates |
|-------------------------------------|--|---------------------------------------|
| Training Periods (Active hours) | Around 126 hours of training (divided by one week) | Around 20 hours (divided by two days) |
| Number of epochs before convergence | More than 35,000 epochs | Around 1900 epochs |
| Corollaries | Can not write all digits | Writes all digits |

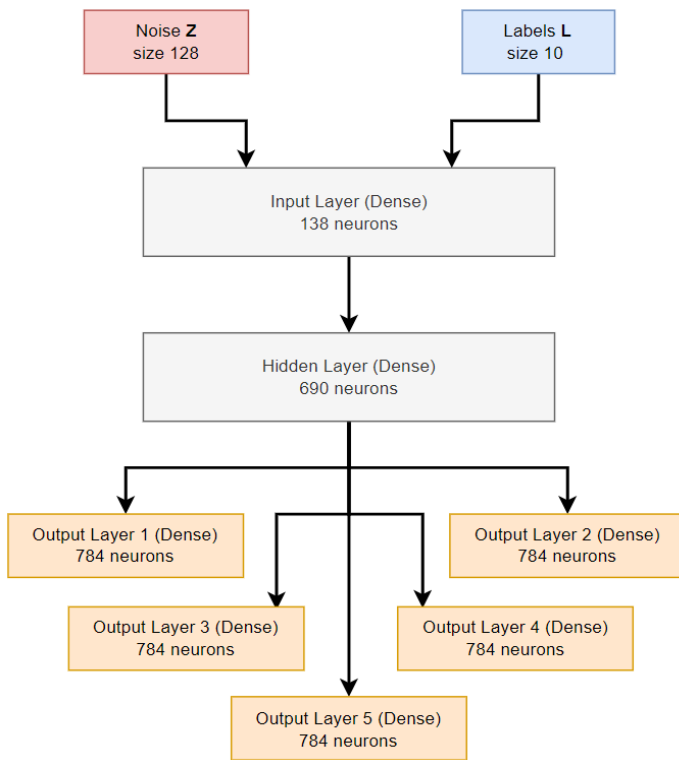


Figure 4.5: Implemented GAN architecture

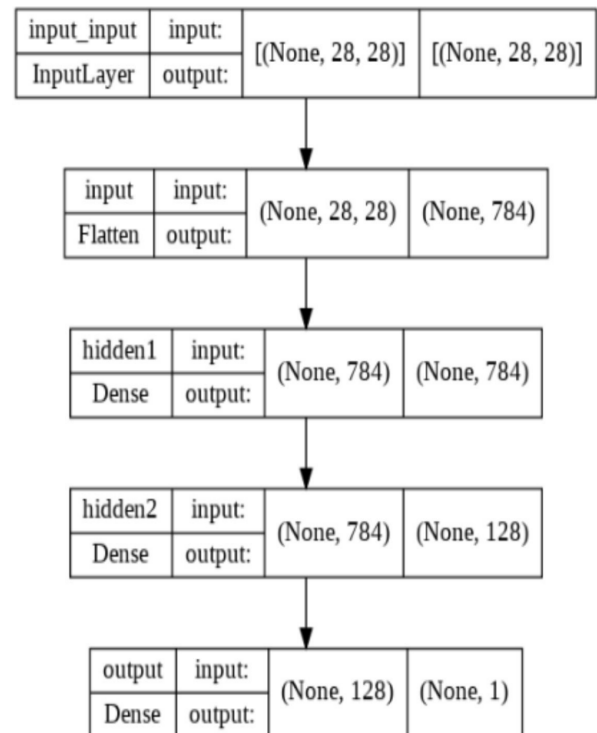


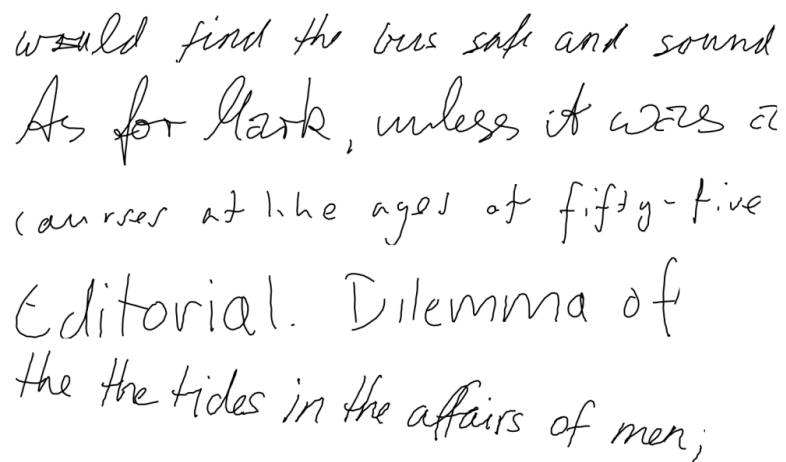
Figure 4.6: Implemented Discriminator architecture

4.1.2.1.2 Discriminative Model

Figure 4.6 demonstrates the architecture of the discriminative model we used in our proposed system. It is considered a binary classifier to distinguish fake images from real ones. The model receives a $28 * 28$ sized image as an input and outputs either '0' for a fake judge or '1' for a real one. There is one flatten layer that converts the input 2D image to a 1D flattened version. After that, two hidden layers are used to make feature extraction from the image, then the extracted features are used for the discrimination purpose. The last layer which is the output layer has one neuron with RELU activation function which produces the final judge value.

4.1.2.2 LSTM Model

Since LSTM needs sequences of points in our project, we used the IAM online handwriting database (IAM-OnDB). This database consists of handwritten lines collected from 221 different writers using a 'smart whiteboard'. The positions of the pens were tracked using an infra-red device in the corner of the board. Samples from the training data are shown in Figure 4.7. The original input data consists of the x and y pen coordinates, and the points in the sequence when the pen is lifted off the whiteboard. Beyond that, no preprocessing was used and the network was trained to predict the XY coordinates and the end-of-stroke markers one point at a time.



would find the bus safe and sound
As for Mark, unless it were a
course at the age of fifty-five
Editorial. Dilemma of
the the tides in the affairs of men;

Figure 4.7: Training samples from the IAM online handwriting database

The database as a set of scripts contains 80 distinct characters (capital letters, lower case letters, digits, and punctuation). However, we used only a subset of 57 in size having all the digits and most of the punctuation characters replaced with a generic ‘nonletter’ label.

This model was implemented using Tensorflow as well. At the core of a model, there are three hidden layers each having 400 neurons. Hidden layers are a long-short-term memory layer, a window layer, and another LSTM layer. Next, there is a dense layer with 3 neurons to generate coordinates at each step. First, two coordinates are x and y on a plane, third coordinate informs whether the character ends or not.

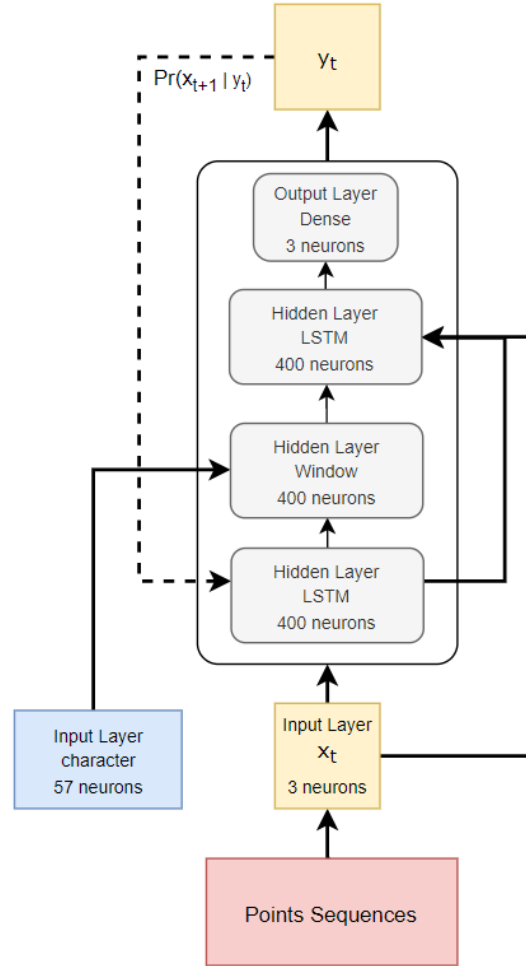


Figure 4.8: Implemented LSTM network

4.2 Hardware Configuration

4.2.1 Assembling the Robot

The robot we purchased was sent to us as a package of small individual pieces and subcomponents that needs to be put together so that they construct larger components that can be connected for the full plotter to be finished. In figure 4.8 we reveal the whole content of the package, then we captured some photos of the assembly process for some components in the following figures 4.9 to 4.13.



Figure 4.9: LY robot components

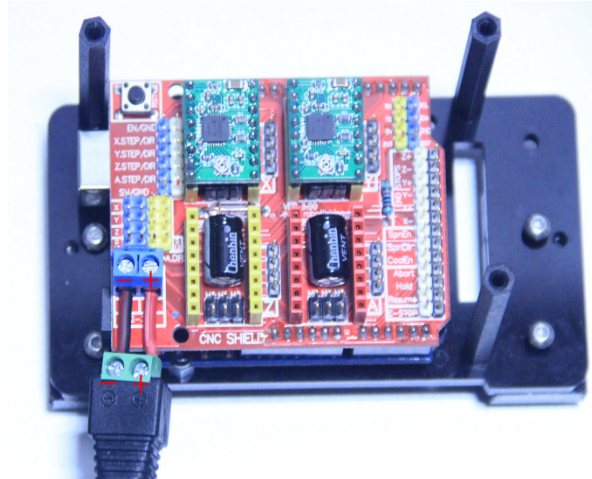


Figure 4.10: CNC shield with drivers

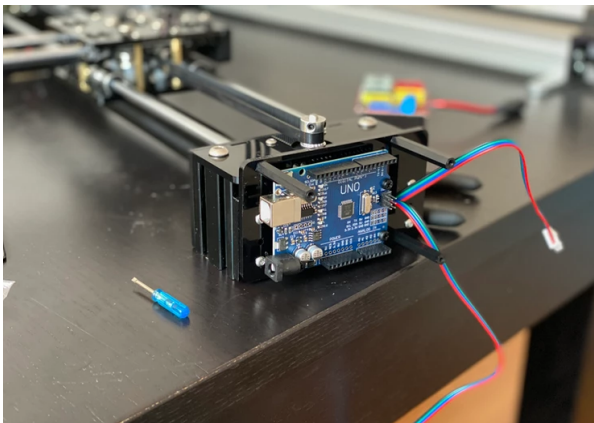


Figure 4.11: Arduino UNO attached

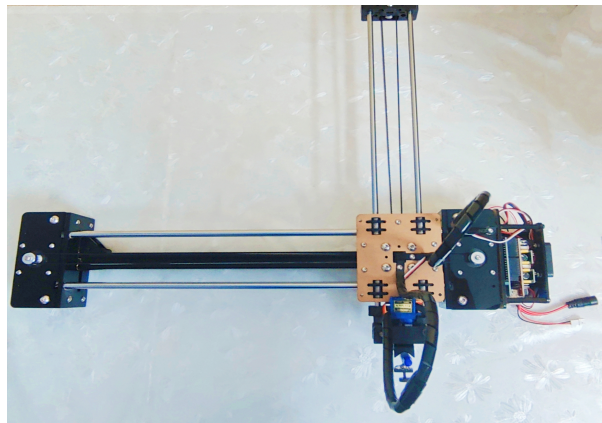


Figure 4.12: Full robot assembled

When installing the drivers for the stepper motors, we noticed that the CNC shield board has 4 slots; each can accept a driver for a specific axis. The servo motor for Z movements is only going up and down in two values, meaning that it does not need a driver as the only required cables are data (0 or 1) and power. The slot for the Z driver is used for attaching another stepper motor that can accept various height values (like ones seen in 3D printers).

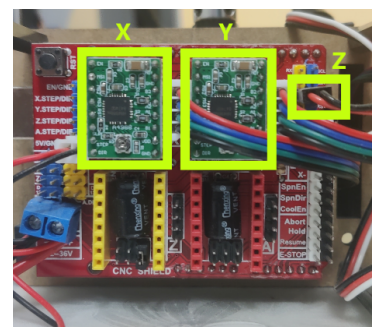


Figure 4.13: Installed motor drivers

4.2.2 Device Identification

```
pi@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 1a86:7523 QinHeng Electronics CH340 serial converter
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $ █
```

Figure 4.14: Robot hardware identification on RaspbianOS

The manufacturing company attached a zip folder containing the standard CH34x compatible machine driver as well as documentation for robot building, and since the driver is cross-platform, we initially installed it in Windows on a PC to see whether the robot is identified. Unfortunately, the version of that driver was behind a newer one that already existed on the internet, and there were some glitches when recognizing the steppers. Luckily, the newest version of the driver solved them and the robot was identified ideally. Figure 4.14 shows that the Raspberry pi read the robot port.

4.2.3 Gcode Instructions

In short, Gcode is a programming language for Computer Numerical Control robots like our robot, and the word Gcode stands for “Geometric Code”. We used this language to tell the robot what to draw and how to draw it accordingly. Gcode commands will instruct the machine on where to move, how fast to move, and what path to follow [43].

The following steps summarize the work done to convert the sequence of points from their digital representation to a creative manuscript written accurately by our robot calligrapher on physical paper:

1. Creating a communication channel between the hardware components of the robot and the generating model without a third-party program by controlling the robot hardware components.
2. Sending an acknowledgment signal to the robot as an indication to begin receiving the geometric code instructions.
3. Choosing the speed of the robot’s motors carefully to achieve the most accurate results as close as possible to the digital images. The accuracy test is recorded in chapter 5.
4. Designing and developing a proper technique that minimizes the discontinuity of strokes by treating the same points of continuous character as one connected component and traversing them sequentially.

Chapter 5

Validation & Results

In this section, we evince the different tests applied to several project building blocks, as well as some of the selected outputs and results for this project at various timestamps with detailed charts and metrics for demonstrating the level of performance this project has achieved during the research and development period.

5.1 Unit Testing

5.1.1 RaspberryPI 4

We connected the raspberry pi microcomputer to an external power supply using its USB-C power delivery port and ensured that the device was working and booting from the SDCard containing the RaspbianOS image (**Passed**).

5.1.2 Robot Calligrapher

The robot has two out cables; one for power delivery up to 12V to supply all the different components, and the other is a USB data cable coming from the Arduino microcontroller to fetch compatible instructions from the host computer. We powered the robot first (**Passed**), then sent some Gcode instructions to the robot and saw the motors moving according to the input accurately (**Passed**). Figure 5.1 shows the accuracy of the robot testing results on drawing $[(0, 0), (50, 0), (50, 50)]$ points in millimeter unit.

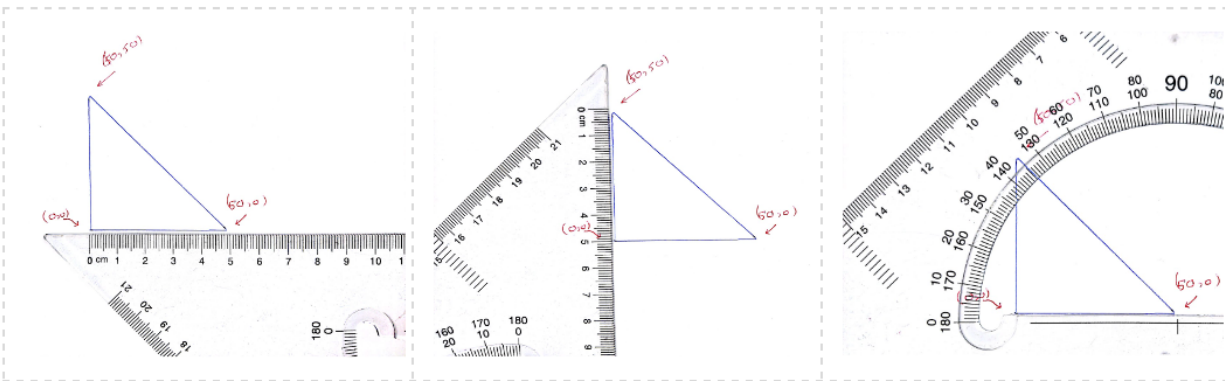


Figure 5.1: Measuring the Robot Calligrapher accuracy

5.2 Integration Testing

5.2.1 Connecting RaspberryPI with the Robot

The robot is connected using a USB interface, so the host OS treats it as an external device and serves it as a platform-related mount point. On Windows the mount point was **COM1**, but on RaspbianOS it was **/dev/ttyUSB0**, both of which the respectful host OS was recognizing up to the robot hardware with the help of the driver (**Passed**).

5.2.2 Installing Necessary Software on RaspberryPI

Although the installation of dependencies and third-party software on the raspberry pi was producing problems more than progress at the beginning, we managed to run our project's software using the method mentioned in chapter 4 successfully after installing Tensorflow and the robot's driver, as well as configuring the SSH protocol to control the raspberry via a remote device. All were done when switching to the 32bit version of the host OS (**Passed**).

5.3 Results and Output Samples

5.3.1 Output Samples - LSTM

After the training period has ended for each of the models we prepared, we can showcase some samples for the outputs constructed from both sides using the robot on plain paper. The following figure 5.2 demonstrates the steps for the drawing process of the handwritten-like word 'Polytechnic' generated from the LSTM model. Note that this word is new to the model as a test input to try out.



Figure 5.2: Process of writing 'Polytechnic' using LSTM

Additionally, the LSTM model is capable of drawing the same word but with different styles as seen in figure 5.3, simply by tuning a parameter for style type for the model.

A list of videos for the calligrapher while writing other handwritten text will be available in a drive link listed in [47].



Figure 5.3: Different styles of 'Polytechnic' using LSTM

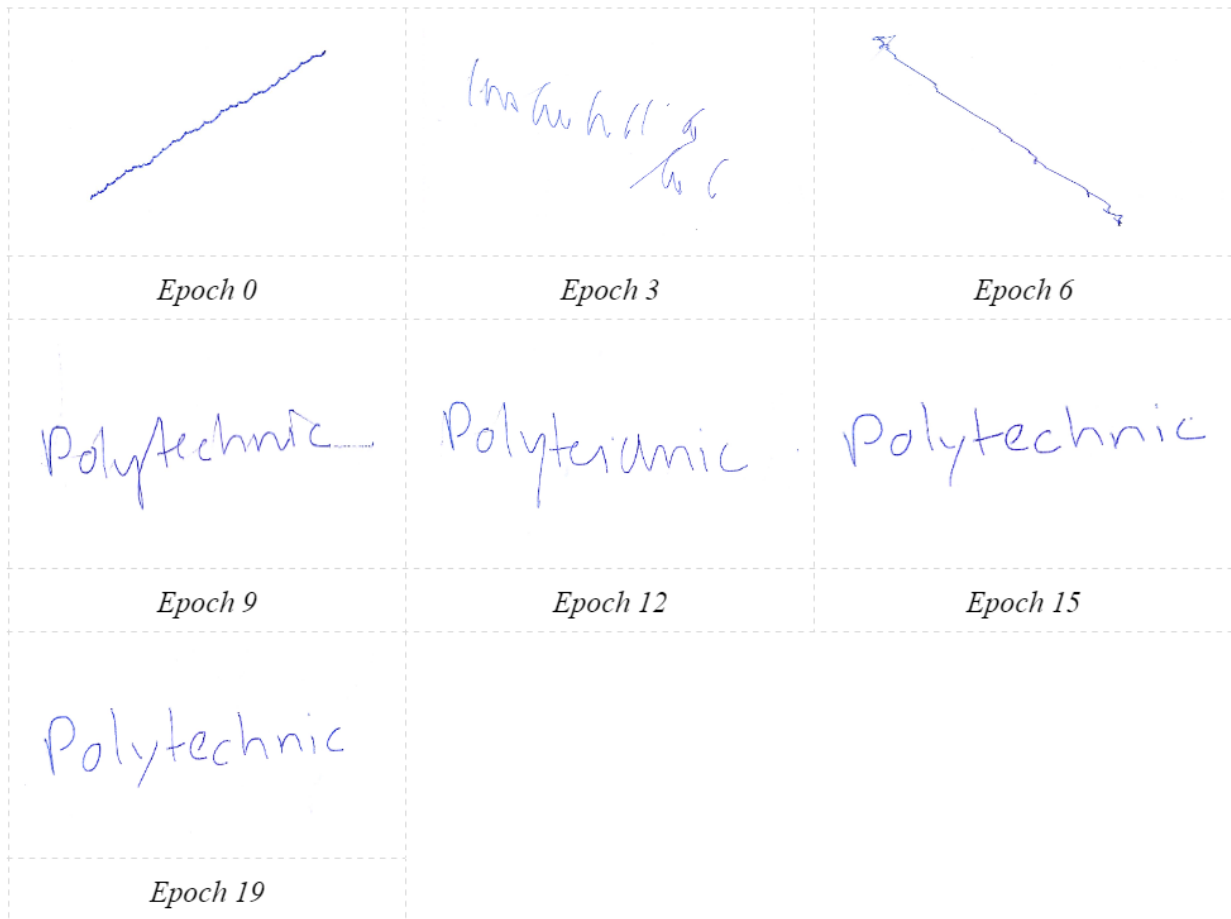


Figure 5.4: LSTM output improvement at selected timestamps

For a better and distinguishable view, we took samples from the robot while training at every three epochs. As shown in figure 5.4, the intelligence model was unaware of anything and missed up either by drawing jittery lines or unknown characters. But from epoch 8 to epoch 9, we started to see the model leaping forward in embracing some of the styles, then moving on the model has learned multiple styles and details of these styles as shown in the previous pictures of figure 5.9.

5.3.2 Output Samples - GAN

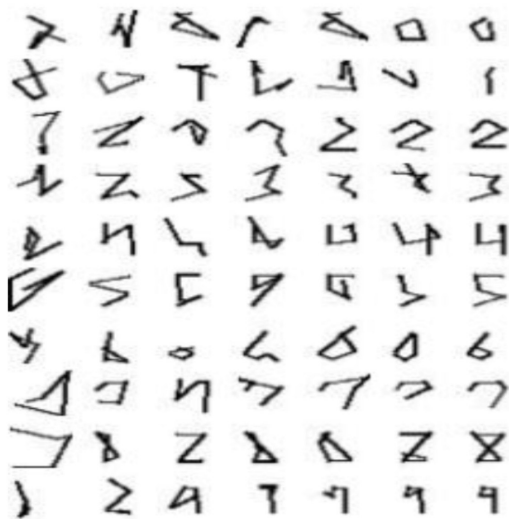


Figure 5.5: GAN output improvement

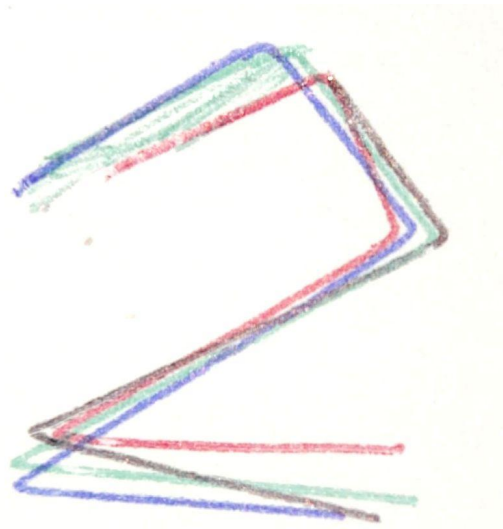


Figure 5.6: Different variations of number '2'

The first figure 5.5 demonstrates the improvement of the GAN model output with samples taken at different training periods. At convergence, the generator network is capable of producing numeric drawings that are readable and clear using only five trajectory points. Right figure 5.6 shows the various possibilities of these trajectories to construct the number as each trajectory is generated within a known region to speed up training time and reduce network overhead.

In the next figure 5.7, we gave the model one of each label given to the model at the convergence state, then collected the drawings done by the robot.

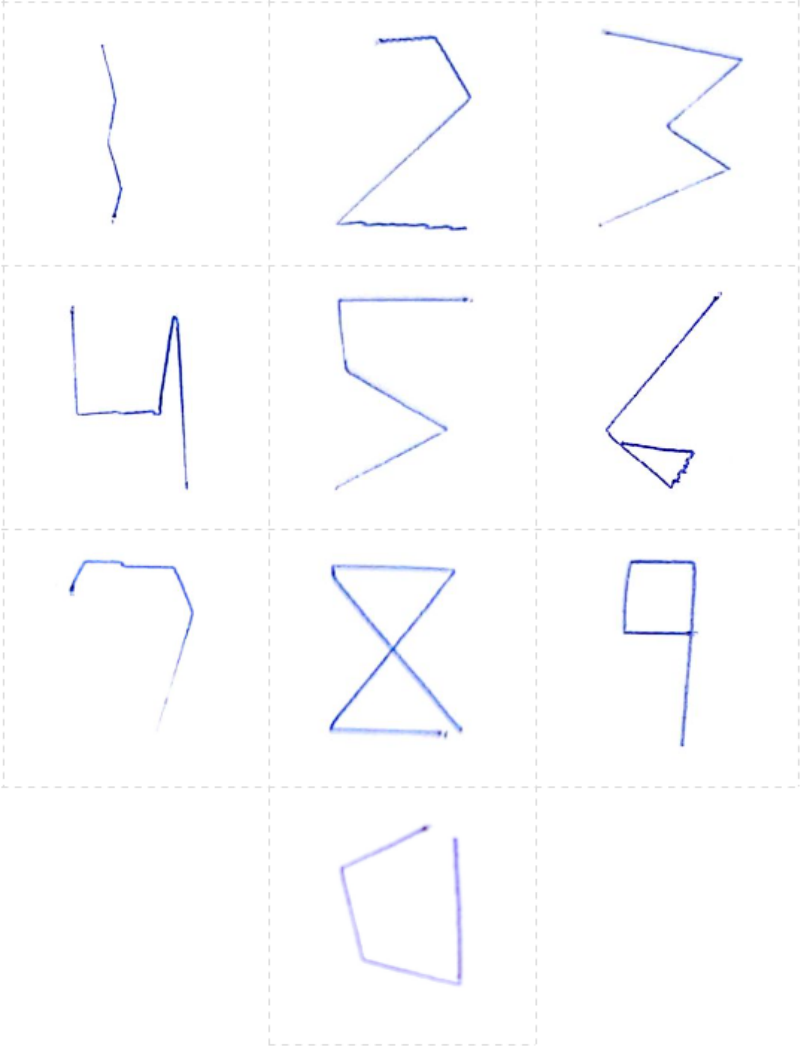


Figure 5.7: GAN's labels output at the convergence

As shown in figure 5.8, the GAN model has the ability to generate different styles for the same label. There is no static style, and the style varies randomly.

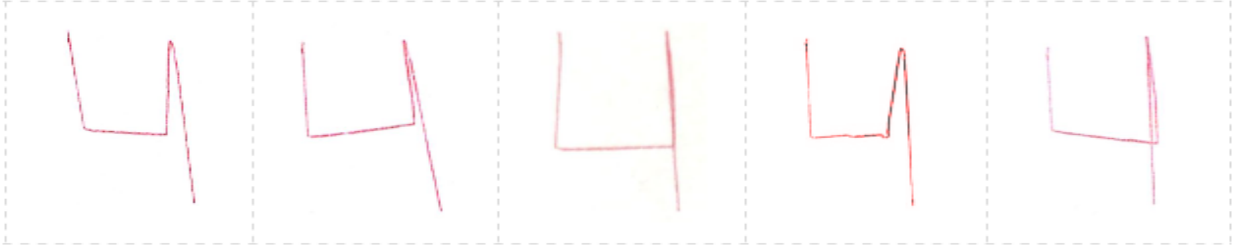


Figure 5.8: Different styles of '4' using GAN

Figure 5.9 shows the writing steps for sequence points of digit '4'. It is clear that the model has the ability to mimic the human handwriting attitude.



Figure 5.9: Process of writing '4' using GAN

5.4 Measurements and Readings

5.4.1 LSTM measurements

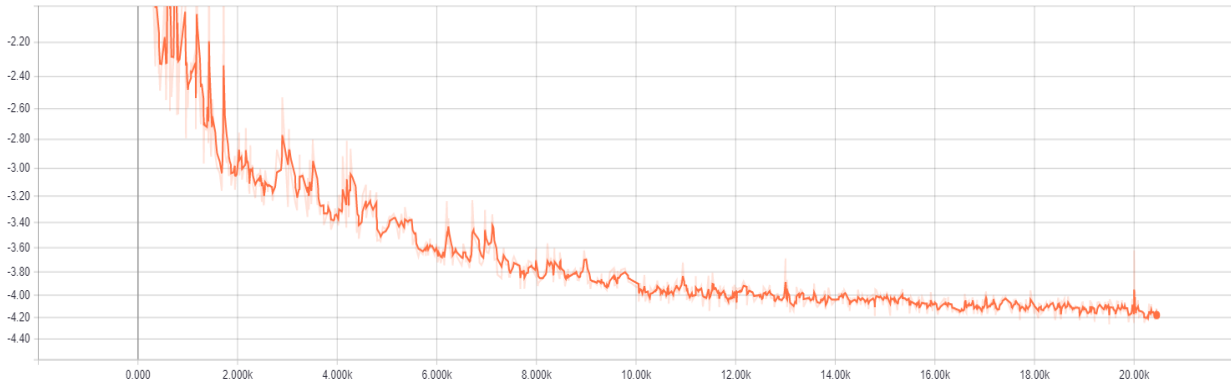


Figure 5.10: LSTM training loss

Figure 5.10 shows the LSTM generator loss graph plotted using TensorBoard. The graph is based on 20 epochs of training, each epoch has 1000 steps. While major learning techniques can have training loss for hyperparameters tuning and validation loss, using the whole dataset for training loss values can go with the LSTM network more efficiently since it is important to take all its given data as training data. The reason for this is that this model is expected to generate indistinguishable handwritten text, meaning that we feed all possible data we have with no splitting if we want to mimic certain styles, thus the measurement for training loss is the only reading we could generate to tweak the hyperparameters. The graph also indicates an increase in values rather than the opposite, and that is because the loss function illustrated by equation 3.1 represents the predictive probability distribution. We are more interested in revealing the gradient ascent representation since it gives us an indication of the matching level between the real data and the generated data, as well as a clear reading of the model performance which is getting better. Tensorflow has only the implementation for gradient descent so we multiplied the result from that function by -1.

The actual improvement is touched on in the previous figure 5.4 where the ascent value is increasing when the model is getting used to what he learned.

5.4.2 GAN measurements

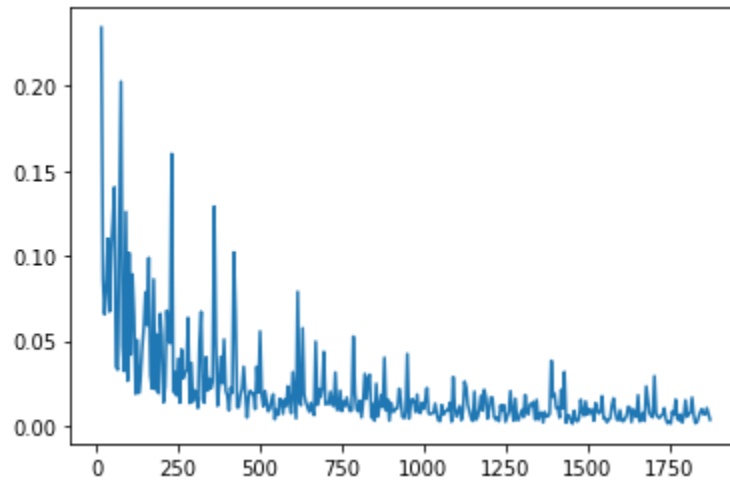


Figure 5.11: Generative loss for all digits together

Figure 5.11 represents the GAN generator loss graph plotted using the Matplotlib library. The graph is based on 1900 epochs of training. Also, there is only training loss here for the same mentioned reason in the 5.4.1 subsection. The graph indicates a decrease in values rather than the LSTM, and that is because the loss function here is represented by the MSE which the lowest loss value indicates that a similarity between generated data and real data is better.

Chapter 6

Conclusion & Future Work

6.1 Conclusion

This project has featured a learning robotic system that can mimic the calligraphy of human beings, making it a robot that wants to compete with human creativity. Our work showed that LSTM is useful in mimicking font styles for words represented by point sequences. Also, we proved that it is possible to solve the enormous search space problem in GAN by pretraining the generator. Moreover, the pretraining will give the GAN the ability to generate points sequentially. We proposed that it is more efficient to take the whole expected actions from the software, then reflect the decisions on the hardware rather than reflecting each decision on the hardware directly. That will achieve continuity in the manuscripts, accelerate the generation process, as well as mimic the thinking of humans.

6.2 Future Work

This work has allowed us to achieve a handful of requirements, and additional features, as well as explore the field of artificial intelligence and deep learning interestingly. While our project has accomplished some of the key points like having a trained system that can mimic the calligraphy of human handwriting and using a state of the art methods for that, there will always be room for improvement. Here we show some of the future enhancements that could potentially have an impact on the project positively:

1. Improve the search space for the generative adversarial network by applying and automating data analysis on the dataset in the pretraining step for better output diversity while maintaining consistency rather than using hand-made templates and localized regions.
2. Support more languages other than English in generating long sentences. Arabic for example.
3. Support a wider set of characters for the GAN model and increase the number of generated trajectory points for smoother synthesis.
4. Upgrade the robot to allow it to work on different rough surfaces like stone, and extend the range of tools to draw with like using a laser or engraver.

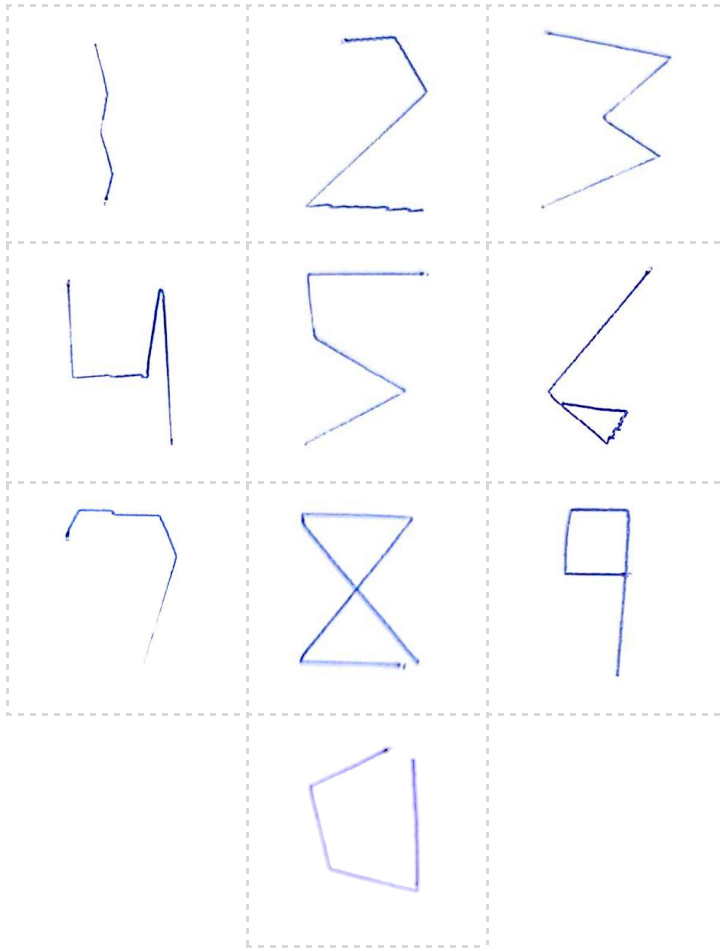
References

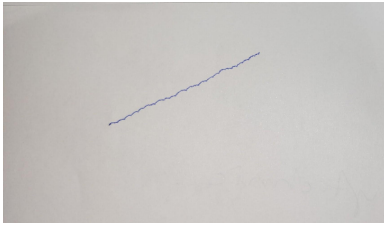
- [1] Machine learning - Wikipedia, the free encyclopedia, Retrieved in November 2021, from: [\[en.wikipedia.org/wiki/Machine_learning\]](https://en.wikipedia.org/wiki/Machine_learning).
- [2] Reinforcement learning - Wikipedia, the free encyclopedia, Retrieved in November 2021, from: [\[en.wikipedia.org/wiki/Reinforcement_learning\]](https://en.wikipedia.org/wiki/Reinforcement_learning).
- [3] Michael L. Littman, and Andrew W. Moore, Reinforcement Learning: A Survey, Journal of Artificial Intelligence Research 4, pp. 237-285, 1996, Retrieved in November 2021.
- [4] Train a software agent to behave rationally with reinforcement learning, IBM, Retrieved in November 2021, from [\[developer.ibm.com\]](https://developer.ibm.com).
- [5] Generative Adversarial Networks (GANs). (n.d.). Wikipedia, the free encyclopedia. Retrieved in November 2021, from [\[https://en.wikipedia.org/wiki/Generative_adversarial_network\]](https://en.wikipedia.org/wiki/Generative_adversarial_network).
- [6] Nicholson, C. (n.d.). “A Beginner's Guide to Generative Adversarial Networks (GANs)”. A Beginner’s Guide to Important Topics in AI, Machine Learning, and Deep Learning, Retrieved in November 2021, from [\[wiki.pathmind.com/generative-adversarial-network\]](https://wiki.pathmind.com/generative-adversarial-network).
- [7] Ruiqi Wu, Changle Zhou, Fei Chao, Longzhi Yang, Chih-Min Lin, Changjing Shang, ”GANCCRobot: Generative Adversarial Nets based Chinese Calligraphy Robot”, Information Sciences (2019), Retrieved in November 2021, from [\[doi.org\]](https://doi.org).
- [8] XiaojinZhu and Andrew B.Goldberg. “Synthesis Lectures on Artificial Intelligence and Machine Learning”, 2009, Vol. 3, No. 1, Pages 1-130, Retrieved in December 2021.
- [9] “TensorFlow”, Retrieved in November 2021, from [\[tensorflow.org\]](https://tensorflow.org).
- [10] Lei Kang, Pau Riba, Yaxing Wang, Marçal Rusiñol, Alicia Fornés, Mauricio Villegas, “GANwriting: Content-Conditioned Generation of Styled Handwritten Word Images”, Computer Vision and Pattern Recognition Subject at Cornell University (2020), Retrieved in November 2021, from [\[arxiv.org\]](https://arxiv.org).
- [11] L. Yu, W. Zhang, J. Wang, and Y. Yu, “Seqgan: Sequence generative adversarial nets with policy gradient.” in AAAI, 2017, pp. 2852–2858. Retrieved in December 2021.

- [12] “Deep Learning”, Investopedia, 2020, Retrieved in December 2021, from: [\[investopedia.com/terms/d/deep-learning.asp\]](https://www.investopedia.com/terms/d/deep-learning.asp).
- [13] “What Is Deep Learning?”, mathworks.com/, 2020, Retrieved in December 2021, from: [\[mathworks.com/discovery/deep-learning\]](https://www.mathworks.com/discovery/deep-learning).
- [14] “Why Deep Learning over Traditional Machine Learning?”, TowardsDataScience, 2018, Retrieved in December 2021, from [\[towardsdatascience.com\]](https://towardsdatascience.com).
- [15] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," 2017 International Conference on Engineering and Technology (ICET), 2017, pp. 1-6. Retrieved in December 2021.
- [16] Chao, Fei, Gan Lin, Ling Zheng, Xiang Chang, Chih-Min Lin, Longzhi Yang, and Changjing Shang. "An LSTM Based Generative Adversarial Architecture for Robotic Calligraphy Learning System", Sustainability (2020), Retrieved in December 2021, from: [\[doi.org\]](https://doi.org).
- [17] Sak, H. & Senior, Andrew & Beaufays, F.. (2014). Long short-term memory recurrent neural network architectures for large-scale acoustic modeling. Proceedings of the Annual Conference of the International Speech Communication Association. 338-342. Retrieved in December 2021.
- [18] GKTools, “GKDraw Software Installation and integration with GK X3 Pro”, Retrieved in December 2021, from [\[mygktools.com/doku.php/gkdraw_software\]](https://mygktools.com/doku.php/gkdraw_software).
- [19] Raspberry Pi 4 Model B 4Gb, Retrieved in December 2021, from [\[robot-advance.com\]](https://robot-advance.com).
- [20] Kaelbling, Leslie P., Littman, Michael L., Moore, Andrew W., (1996). Reinforcement Learning: A Survey. Retrieved in December 2021.
- [21] LY pen drawing robot machine lettering XY-plotter robot machine, Retrieved in December 2021, from [\[AliExpress.com\]](https://AliExpress.com).
- [22] Arduino Uno Rev3 – Arduino Official Store, Retrieved in December 2021, from [\[Arduino Uno Rev3 — Arduino Official Store\]](https://www.arduino.cc/en/Store).

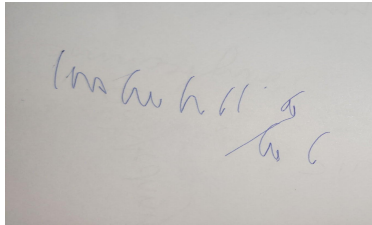
- [23] Graves, A. (2013). Generating sequences with recurrent neural networks. arXiv:1308.0850 [cs.NE]. Retrieved in April 2022, from [arxiv.org]
- [24] CNC Shield V3 - Euro-Makers, Retrieved in December 2021, from [euro-makers.com].
- [25] CNC Shield V3 – Engraving Machine 3D Printer Expansion Board, Retrieved in December 2021, from [ElectronicsComp.com].
- [26] MG996r Servo Motor, Retrieved in December 2021, from:
[electronicoscaldas.com/datasheet/MG996R].
- [27] About Servo Motors, Retrieved in December 2021, from:
[jsumo.com/mg996r-servo-motor-digital].
- [28] Stepper Motors, Retrieved in December 2021, from:
[learn.adafruit.com/all-about-stepper-motors].
- [29] Stepper Motors in Detail, Retrieved in December 2021, from:
[ato.com/4-wire-nema-17-bipolar-stepper-motor].
- [30] A4988 Stepper Motor Driver, Retrieved in December 2021, from [twinschip.com].
- [31] Cooling Fans for Raspberry Pi and Arduino chips, Retrieved in December 2021, from:
[electan.com].
- [32] Power supply AC Adapter for Speedport, Retrieved in December 2021, from:
[remotes4you.eu].
- [33] Downey A., Elkner J. & Meyers C, (2002). How to Think Like a Computer Scientist, Learning with Python, “The Python Programming Language”, Wellesley: Green Tea Press. Retrieved in December 2021.
- [34] D. Berio, S. Calinon, F. F. Leymarie, Dynamic graffiti stylization with stochastic optimal control, in: Proceedings of the 4th International Conference on Movement Computing, ACM, 2017. Retrieved in December 2021.
- [35] V. Mohan, P. Morasso, J. Zenzeri, G. Metta, V. S. Chakravarthy, G. Sandini, Teaching a humanoid robot to draw “shapes”, Autonomous Robots 31 (1) (2011) 21–53. Retrieved in December 2021.

- [36] D. Berio, S. Calinon, F. F. Leymarie, Learning dynamic graffiti strokes with a compliant robot, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2016, pp. 3981–3986. Retrieved in December 2021.
- [37] R. J. Hendrick, C. R. Mitchell, S. D. Herrell, I. Robert J. Webster, Hand-held transendoscopic robotic manipulators: A transurethral laser prostate surgery case study, The International Journal of Robotics Research 34 (13) (2015) 1559–1572. Retrieved in December 2021.
- [39] GKTOOLS Official Store, Retrieved in December 2021, from [gktools.aliexpress.com].
- [40] Graves, Alex. “Understanding LSTM Networks -- colah's blog.” Colah's blog, August 2015, Retrieved in December 2021, from [[colah.github](https://colah.github.io/)].
- [41] Virtual Network Computing - Wikipedia, Retrieved in May 2022, from [[Wikipedia](https://en.wikipedia.org/wiki/Virtual_Network_Computing)].
- [42] Secure Shell - Wikipedia, the free encyclopedia, Retrieved in April 2022, from [[Secure Shell - Wikipedia](https://en.wikipedia.org/wiki/Secure_Shell)].
- [43] List of Most Important G-code Commands, Retrieved in April 2022, from: [t.ly/0A0X]
- [44] M. Liwicki and H. Bunke. IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard. In Proc. 8th Int. Conf. on Document Analysis and Recognition, volume 2, pages 956-961, 2005.
- [45] Zhengwei Wang, Qi She, and Tomás E. Ward. 2021. Generative Adversarial Networks in Computer Vision: A Survey and Taxonomy. ACM Comput. Surv. 54, 2, Article 37 (March 2022), 38 pages. Retrieved in February 2022, from [<https://doi.org/10.1145/3439723>]
- [46] M. Liwicki and H. Bunke. IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard. In Proc. 8th Int. Conf. on Document Analysis and Recognition, volume 2, pages 956-961, 2005. Retrieved in May 2022.
- [47] Calligrapher Robot Videos, Accessed from [t.ly/--sH].

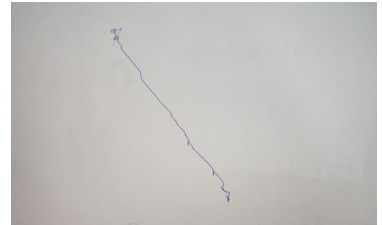




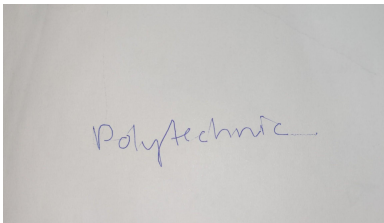
Epoch 0



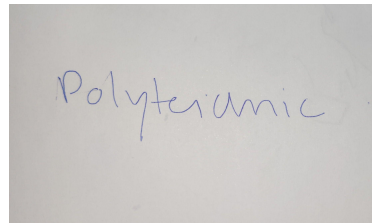
Epoch 3



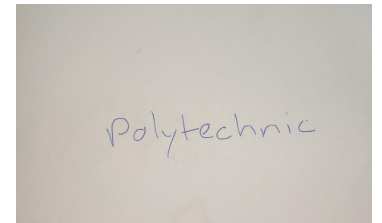
Epoch 6



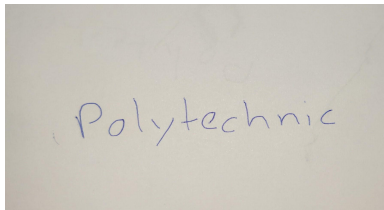
Epoch 9



Epoch 12



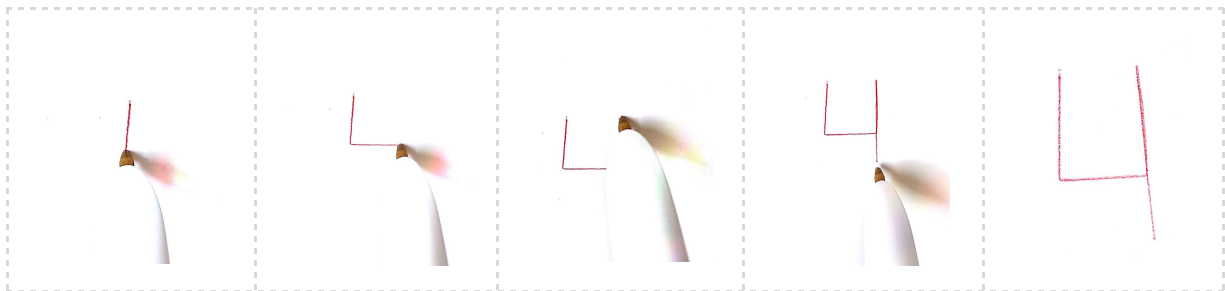
Epoch 15

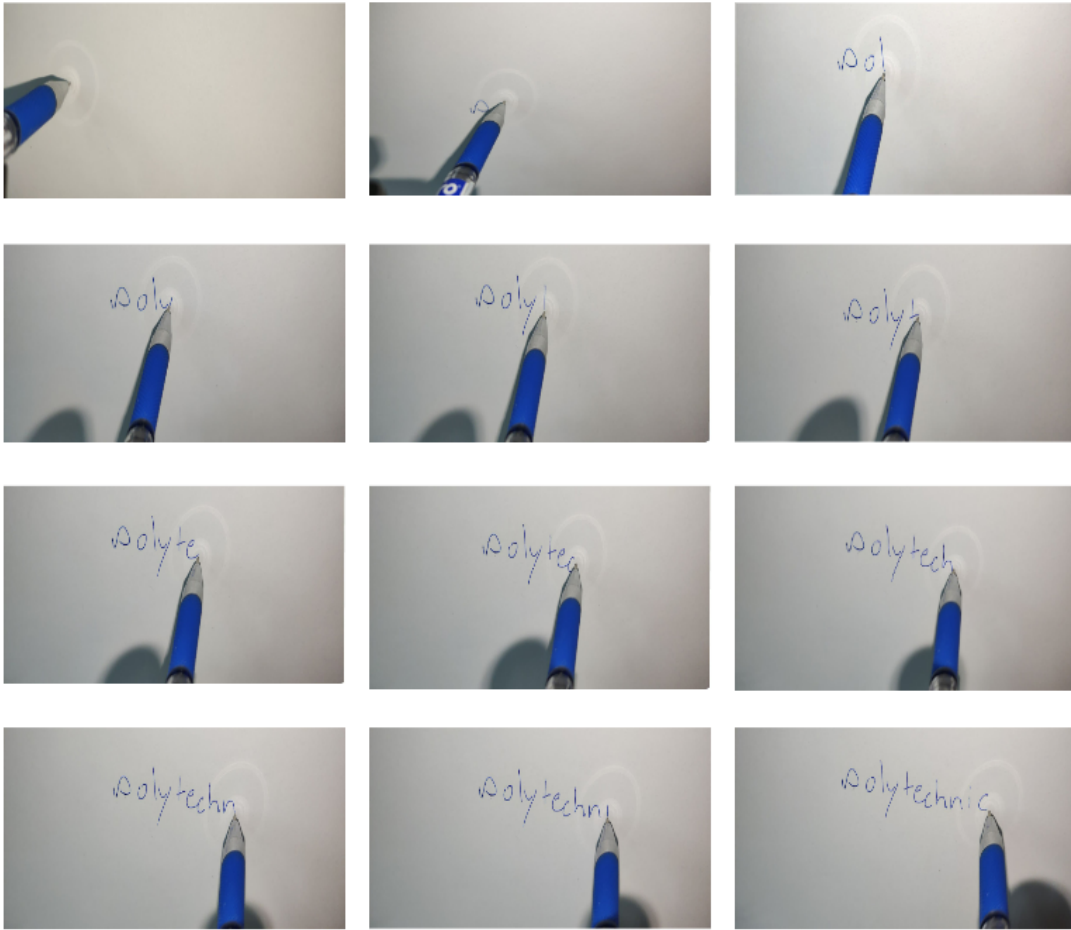


Epoch 19

| | | |
|-------------|-------------|-------------|
| Polytechnic | Polytechnic | Polytechnic |
| Polytechnic | Polytechnic | Polytechnic |

| | | | | |
|---|---|---|---|---|
| 4 | 4 | 4 | 4 | 4 |
|---|---|---|---|---|





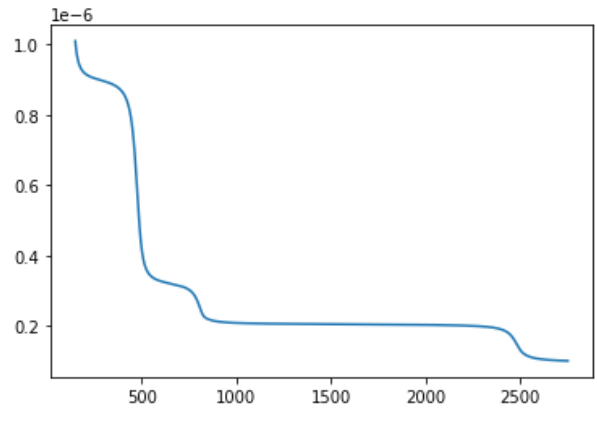
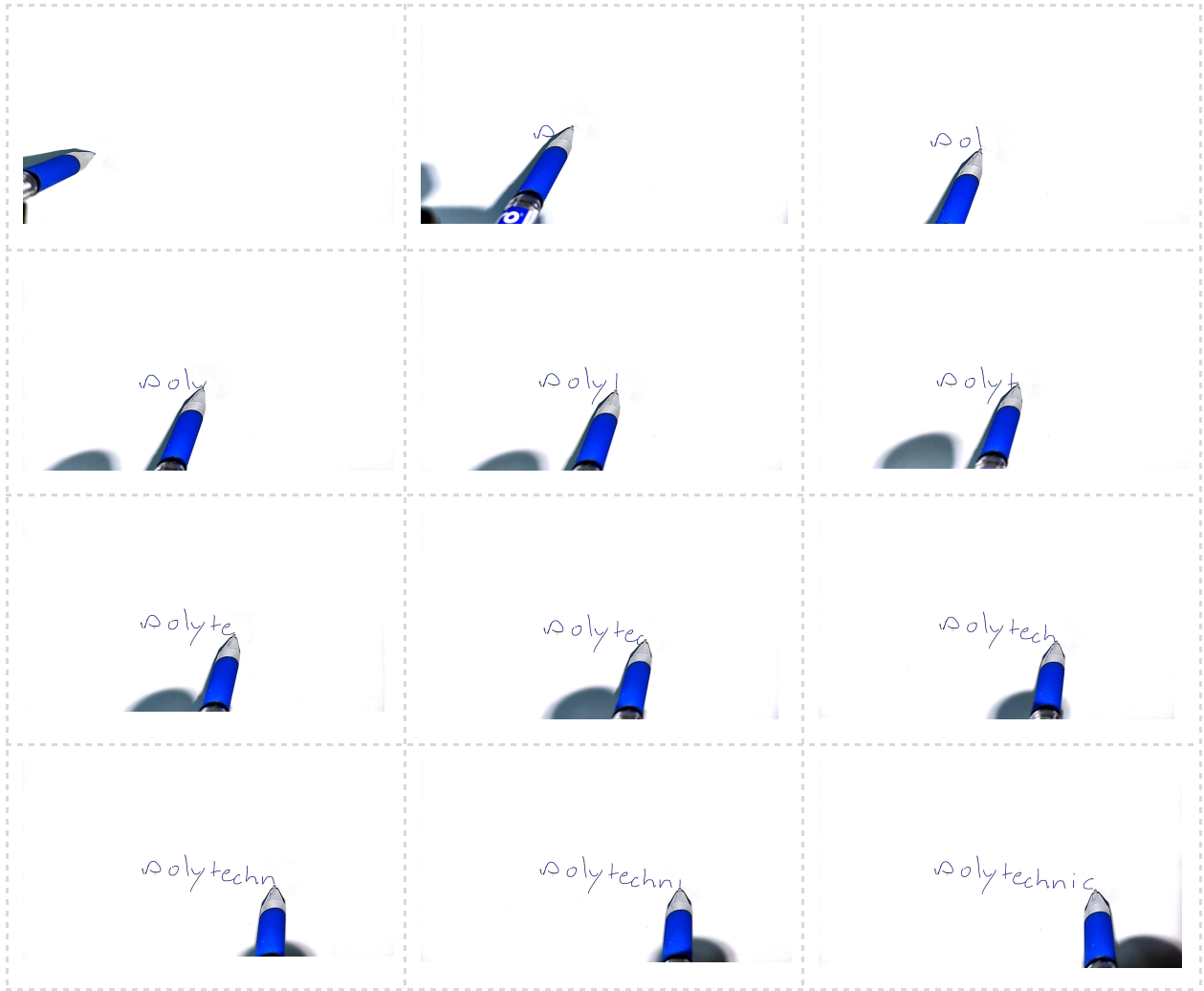
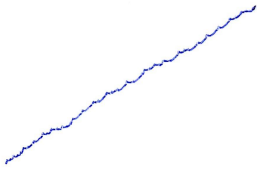
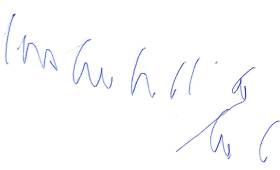
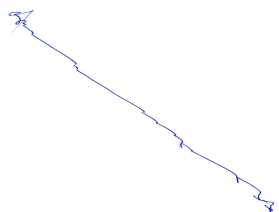


Figure 5.11: Pre-training generative loss for number '7'

| | | |
|---|---|---|
|  |  |  |
| <i>Epoch 0</i> | <i>Epoch 3</i> | <i>Epoch 6</i> |
| <i>Polytechnic</i> | <i>Polytechnic</i> | <i>Polytechnic</i> |
| <i>Epoch 9</i> | <i>Epoch 12</i> | <i>Epoch 15</i> |
| <i>Polytechnic</i> | | |
| <i>Epoch 19</i> | | |

```

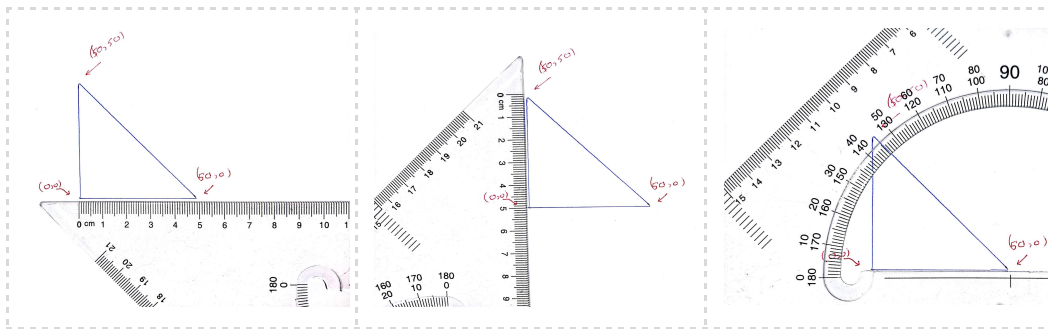
pi@raspberrypi:~ $ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 003: ID 1a86:7523 QinHeng Electronics CH340 serial converter
Bus 001 Device 002: ID 2109:3431 VIA Labs, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:~ $ █

```


In this project, we used a handful of instructions to translate the trajectories of the model output for the robot to draw. All with their respective short descriptions in the following table 4.1.

| Instructions | Functionalities |
|--------------|---|
| G90 | Absolute mode coordinations. |
| G91 | Relative mode coordinations. |
| G21 | Define G-code units in millimeters. |
| F [Num] | Define the speed of a robot per minute. |
| M03 | Spindle ON. |
| M05 | Spindle stop. |
| G4 S [Num] | Pause robot [Num] of ms. |
| G1 X_ Y_ Z_ | Direct the robot's motor to change its state. |

Table 4.1: Gcode Instructions



Abstract

There is a huge demand in the field of Artificial intelligence for making novel works that can mimic human creativity. One interesting example is handwritten calligraphy. Robotic calligraphy, as a typical application of robot movement planning, is of great significance for the education of calligraphy culture. The existing implementations of such robots often suffer from their limited ability for font generation and evaluation, leading to poor writing style diversity and writing quality. Our work aims to provide a solution that humans can utilize, to mimic the calligraphy of handwritten texts using the provided robotic system, reaching a satisfying level where handwriting is done automatically with astonishing results. The work is utilizing long-short-term memory (LSTM) technology alongside a generative adversarial model (GAN to be used as a proof of concept). This work has shown in its outcomes the possibility to write words using the robot, as well to mimic a specified human writing style from images after pretraining the system on sequences of points of lines belonging to other styles. We proposed that it is more efficient to take the whole expected actions from the software, then reflect the decisions on the hardware rather than reflecting each decision on the hardware directly

[- دراسة إمكانية تقبل طلاب جامعة بوليتكنك فلسطين استخدام تطبيقات الذكاء الاصطناعي في حياتهم العملية - Google Docs](#)

Before pre-train: took around 126 hours before convergence for some digits.

After pre-train: took around 20 hours for all digits.

Once we figured out the huge possibilities space number, we started feeding the TensorFlow-implemented network hosted on a computer instance having 8 vCPUs, 8 GB of RAM, and an Nvidia Quadro M4000 with the training data. The following table 4.1 presents the results of the initial testing.

- Previous works wrote the digits in 126 hours
- Our work took benefits from the human learning way by used pre training so it wrote the digits in 20 hours

- We proposed a proof of concept