TRIBHUVAN UNIVERSITY

INSTITUTE OF ENGINEERING

THAPATHALI CAMPUS

A Project Report

On

AN IMPLEMENTATION OF A UNIVERSAL TURING MACHINE

Submitted By:

Agrima Regmi	(THA078BEI003)
Jenish Pant	(THA078BEI018)
Pratistha Saptoka	(THA078BEI028)
Sanskriti Khatiwada	(THA078BEI037)

Submitted To:

Department of Electronics and Computer Engineering

Thapathali Campus

Kathmandu, Nepal

December, 2024

ACKNOWLEDGEMENT

We are deeply grateful to the Department of Electronics and Computer Engineering, Thapathali Campus, for granting us the opportunity to undertake the project, ”An Implementation of A Universal Turing Machine.” Our supervisor, Er. Binod Sapkota provided invaluable guidance and support for the project. We acknowledge the contributions of various open-source communities and contributors, whose resources and innovations have greatly inspired and supported us during different stages of ”An Implementation of A Universal Turing Machine.” Their collective efforts have been instrumental in fostering a creative and informed environment for the incubation of our ideas. We also thank our partners for their collaboration, resource-sharing, and expertise, which have been pivotal to our progress. Additionally, we appreciate the constructive criticism and feedback from peers and faculty, which have helped refine our approach and enhance the project’s quality.

Finally, we express our heartfelt gratitude to the department and all faculty members who provided us with the necessary resources and guidance. Their feedback and suggestions have been invaluable in shaping our project journey. We are highly indebted to everyone who is guiding us in this project with their kind support and encouragement.

Agrima Regmi	(THA078BEI003)
Jenish Pant	(THA078BEI018)
Pratistha Saptoka	(THA078BEI028)
Sanskriti Khatiwada	(THA078BEI037)

ABSTRACT

The Universal Turing Machine(UTM) is a cornerstone of modern computational theory, providing a theoretical model for the limits of computation. It serves as a foundation for understanding algorithmic processes, computational complexity, and the nature of problem-solving through machines. State-based logic, a critical component of Turing machines, forms the basis for much of contemporary computing systems, as it defines how a machine progresses through a series of states based on input conditions. These concepts are not only fundamental to computer science but also to the development of real-world computing systems that execute tasks based on logical progression.

This project aims to bring these abstract computational theories into the physical realm by designing and implementing a UTM as a hardware model. The machine is programmable through punched cards, allowing users to define state transitions and execute algorithms on a memory tape, effectively simulating the operation of a Turing machine. It integrates mechanical actuation, electronic sensors, and computational controls to perform essential computational tasks such as reading, writing, and erasing data on the tape. These capabilities mirror the core functions of a theoretical computing device and demonstrate the principles of discrete logic and state machines in a tangible form.

Beyond serving as an educational tool, this project offers valuable research opportunities for exploring the limits and potential applications of computational models. By providing a hands-on approach, the project allows users to better understand how algorithms operate in practice, giving insights into the broader field of computational theory, including automata theory, computational complexity, and the foundations of computer science. The machine's modular and cost-effective design ensures accessibility, making it suitable for both academic environments and low-resource settings. UTM serves as a powerful resource for teaching and research, offering an engaging platform for studying the fundamental aspects of computation, algorithm execution, and logical progression in machines.

Keywords: Universal Turing Machine(UTM), State-based logic, Discrete Logic, State Machines, Punched Cards, Computational Theory, Automata

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Acronym	Full Form
AMS	Advanced Monolithic System
BCD	Binary Coded Decimal
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
DC	Direct Current
EBCDIC	Extended Binary Coded Decimal Interchange Code
EDA	Electronic Design Automation
FA	Finite Automaton
FSM	Finite State Machine
IBM	International Business Machines
IDE	Integrated Development Environment
IR	Infrared Radiation
JFLAP	Java Formal Languages and Automata Package
LAT	Lateral Histogram Method
LCD	Liquid Crystal Display
LiPo	Lithium-ion Polymer
OS	Operating System
PCB	Printed Circuit Board
PDF	Portable Document Format
TM	Turing Machine
UTM	Universal Turing Machine

1. INTRODUCTION

1.1 Background

The Universal Turing Machine (UTM), first introduced by Alan Turing in his groundbreaking 1936 paper *On Computable Numbers, with an Application to the Entscheidungsproblem [1]*, is a foundational concept in computer science. It serves as a theoretical framework for understanding the boundaries of computation and the principles governing algorithmic processes. The UTM formalizes the notion of computation by using an abstract machine that consists of an infinite tape, segmented into discrete cells capable of holding symbols, and a movable head that reads, writes, and transitions between states based on a defined set of rules.

These rules dictate the machine's behavior by associating the current symbol being read and the machine's state with specific actions, such as writing a symbol, moving the head, or changing states. Despite its theoretical nature, the UTM remains a critical tool for studying fundamental concepts in computer science, including computability (identifying problems solvable by computational means) and decidability (determining which problems can be conclusively answered by an algorithm). Formally, a Turing machine is defined as a 7-tuple [2]:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, \phi, F)$$

Where:

- Q is the finite set of states of the finite control,
- Σ is the finite set of input symbols,
- Γ is the finite set of tape symbols, where $\Sigma \subset \Gamma$ and Γ,
- δ is the transition function, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$, where for each state and tape symbol, the machine transitions to a new state, writes a symbol on the tape, and moves the tape head left (L) or right (R),
- $q_0 \in Q$ is the start state, a member of Q, in which the finite control is found initially,
- ϕ is the blank symbol. It is in Γ but not in Σ; i.e., it is not an input symbol. It appears initially in all but the finite number of initial cells that hold input symbols,
- F is the set of final or accepting states, where $F \subset Q$.

Although no physical machine today operates precisely as Turing's model describes, UTM has profoundly influenced the theoretical underpinnings of modern computing. It demonstrates which problems can be resolved algorithmically and highlights the inherent limitations of computation, such as problems that are unsolvable regardless of computational power. The simplicity and generality of the UTM make it an enduring paradigm for analyzing how computers process information and for exploring the theoretical limits of computational systems. This abstract model continues to shape contemporary approaches to problem-solving, computation theory, and the design of modern computing systems.

1.2 Motivation

The Universal Turing Machine (UTM) serves as a cornerstone of computational theory, yet its abstract nature often presents challenges for students trying to grasp its operation and significance. Traditional teaching methods may struggle to effectively convey the intricate mechanics of state transitions, tape operations, and algorithmic execution, leaving gaps in understanding fundamental concepts such as decidability, computability, and algorithm design.

This project aims to address these challenges by creating an interactive and visual implementation of the UTM. By designing a hardware model that vividly demonstrates the movement of the tape, the head's actions, and state transitions based on programmed state tables, we seek to offer an engaging and accessible learning experience. This simulation provides a dynamic representation of how a Turing Machine processes input, enabling students to observe the principles of computational theory in action.

Beyond enhancing comprehension, this project fosters a deeper appreciation of the historical and theoretical significance of Turing's work. By simulating a traditional Turing Machine in a tangible and interactive format, students will not only better understand its foundational role in computer science but also draw connections to modern computing systems. This exploration bridges the gap between theoretical concepts and real-world applications, motivating students to engage with the principles that underpin contemporary algorithmic and computational frameworks.

1.3 Problem Definition

The theoretical nature of the Turing Machine, while fundamental to computational theory, poses significant challenges for students attempting to comprehend its operation and relevance. The abstract representation of the machine, consisting of an infinite tape, a head, and a set of state-based rules, often makes it difficult for learners to visualize how the machine functions step-by-step. This lack of intuitive understanding can hinder students' ability to grasp essential concepts such as state transitions, data manipulation, and algorithmic execution.

A major issue is the absence of practical, interactive learning tools that can bridge the gap between theory and application. Without visual aids or simulations, students struggle to connect the machine's theoretical framework to real-world computational systems, further exacerbating the challenge of understanding its importance. Additionally, the limited exposure to Turing Machines in everyday applications contributes to a disconnect between theoretical knowledge and practical insight, resulting in an incomplete appreciation of their role in shaping modern computing. This disconnect underscores the need for an engaging and interactive approach to teaching the Turing Machine, one that not only demonstrates its mechanics but also highlights its relevance to contemporary computational paradigms.

1.4 Objectives

- To design and implement a physical model of Turing's theoretical computation framework, effectively bridging the gap between abstract theory and tangible understanding.
- To develop an educational platform that demonstrates key computational concepts such as state transitions, memory operations, and algorithm execution, fostering a deeper comprehension of foundational computational principles.

1.5 Scope and Application

1.5.1 Project Capabilities

- Conversion of punch card data to binary using an ESP32-CAM module, enabling the loading of the state table and memory tape into the system.
- Implementation of precise control mechanisms for reading, writing, and erasing data on a moving tape through the use of stepper and servo motors.
- Execution of state logic using an Arduino microcontroller, with user interaction facilitated via an LCD display and control buttons.
- Validation of state transitions and identification of operational errors, ensuring the machine operates within defined parameters.

1.5.2 Project Applications

- Bridging the gap between abstract computational theories and practical implementations through a tangible, physical realization of a Universal Turing Machine.
- Demonstrating core principles of computation, such as state transitions, memory operations, and algorithmic execution, in an accessible and interactive manner.
- Serving as a hands-on educational tool to inspire and enhance the understanding of computational models and their relevance to modern computing systems.

1.5.3 Project Limitations

- Limited to basic state transitions and computations due to material and construction constraints, which may restrict the complexity of the algorithms demonstrated.
- Programming and interaction with the machine can be cumbersome and less intuitive, deviating slightly from the simplicity envisioned in Alan Turing's original model.
- The system relies on predefined state tables, limiting its adaptability for more complex or dynamic computational scenarios.

2. LITERATURE REVIEW

2.1 Theory

The concept of the Turing Machine was introduced by Alan Turing in his seminal paper, *On Computable Numbers, with an Application to the Entscheidungsproblem.[1]* This work laid the foundation for computation theory, or computability, which examines the capabilities and limitations of digital computers.

A Turing Machine is a mathematical model of computation that manipulates symbols on a tape according to a predefined set of rules. Initially referred to as an ”a-machine” (automatic machine) by Turing, the term ”Turing Machine” was popularized by Alonzo Church in a review of Turing’s work [3]. The Turing Machine model is fundamental to computer science, as it defines the theoretical framework for understanding computability and algorithmic processes.

The operation of a Turing Machine is determined by a finite state machine (FSM) that processes input in binary form from the tape and produces an output based on the machine’s halting state. However, each Turing Machine is tailored to a specific computation, necessitating the construction of a new machine for every unique task. To overcome this limitation, Turing proposed the Universal Turing Machine (UTM), which can simulate any other Turing Machine by accepting the description of the machine alongside the input on the tape [1]. This innovation is regarded as a precursor to modern programmable computers.

2.2 Similar Projects

2.2.1 Early Models of Turing Machines

In the 1950s, efforts to translate Turing’s abstract concepts into practical implementations gained traction. Claude Shannon made significant contributions through his work on finite-state machines (FSM), presented in the book *Automata Studies* (1956), co-edited by Shannon and John McCarthy. While Shannon’s focus was primarily on theoretical frameworks, his research laid the foundation for designing hardware-based computational models, including Turing Machines [4]. These insights were pivotal in bridging the gap between abstract automata and their physical implementations.

2.2.2 Turing Machine Simulators

Numerous software tools have been developed to simulate Turing Machines, facilitating the study of theoretical computer science. A notable example is JFLAP (Java Formal Languages and Automata Package), created by Sharon H. Rodger in the 1990s. JFLAP enables users to experiment with topics such as finite automata, pushdown automata, multi-tape Turing Machines, and various types of grammars [5]. This tool has been widely used in academia for teaching and research, providing valuable insights into the operation and universality of Turing Machines.

To gain a deeper understanding of the practical challenges associated with this project, we developed our own emulator. This approach allowed us to explore the feasibility of implementing a Universal Turing Machine in hardware.

2.2.3 Physical Models of Turing Machines

Physical models of Turing Machines have been constructed by researchers and enthusiasts to demonstrate Turing's concepts in tangible ways. These models typically involve mechanical components to replicate the tape movement, state transitions, and symbol manipulation processes. For instance, Mike Davey's model is a noteworthy example that brings Turing's abstract ideas into a physical form for educational purposes [6]. However, these implementations are often constrained by material and design limitations, restricting their ability to handle complex computations.

2.2.4 Advances in Turing Machine Simulations

With advancements in computational complexity theory and algorithm development, Turing Machine simulators have evolved to support diverse applications in fields such as quantum computing, artificial intelligence, and cryptography.

The Quantum Turing Machine (QTM), introduced by David Deutsch and later expanded upon by Bernstein and Vazirani, represents a significant breakthrough. QTMs extend the traditional UTM model by incorporating quantum mechanical principles, such as superposition and entanglement, while ensuring unitary evolution [7, 8]. These machines exemplify the fusion of classical computation theory with quantum mechanics, paving the way for revolutionary developments in computational research.

3. METHODOLOGY

3.1 Proposed System Block Diagram

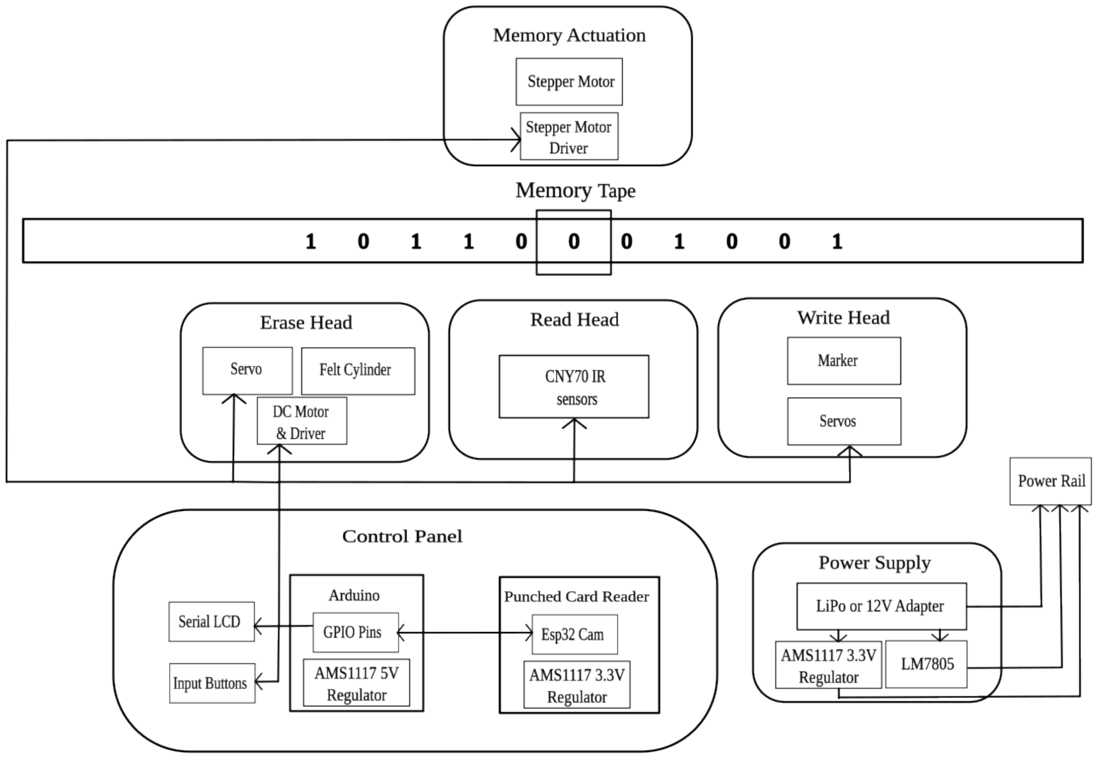


Figure 3.1: Proposed system block diagram

The proposed system architecture in Figure 3.1 outlines the design and structure of the UTM, detailing how each component of the machine works together to simulate a Turing machine in the physical environment. The given architecture consists of several key components that interact to achieve the desired functionality of data manipulation on the memory tape. It is designed to be modular to facilitate easy understanding and construction. It integrates mechanical, electronic, and computational components that work in tandem to simulate a Universal Turing Machine's operations. In the following subsections, we present the key elements of the architecture.

3.1.1 Memory Tape and Actuation System

The memory tape serves as the primary data storage medium for the UTM. The tape is designed to be a 35mm film leader or another suitable plastic material that can be easily written to and erased. It is divided into discrete cells, each representing a binary symbol (0 or 1), with an additional blank symbol, if required, for computational purposes. Although the tape is not infinite in length, it is sufficiently long to accommodate any practical computation task. A stepper motor, controlled by an Arduino through its dedicated driver, moves the tape left or right with high precision. This ensures accurate positioning of the tape relative to the read, write, and erase heads.

The movement of the tape is regulated by a pulley system, where one full rotation of the stepper motor corresponds to the movement of one cell in either direction. This design ensures that each tape cell aligns with the heads during read, write, and erase operations, allowing for precise control over data manipulation.

3.1.2 Read Head

The read head consists of three or more CNY70 IR sensors arranged in a specific pattern to detect the reflectivity of the tape's cells. These sensors translate the reflected light from the tape into binary data, corresponding to the symbols on the tape. As the tape moves, the sensors read the state of each cell on the basis of its reflectivity, allowing the system to determine the symbol it contains.

The reflectivity data is translated into binary values as follows:

- **0**: Black, White, Black
- **1**: White, Black, White
- **B**: All Black (blank)

These sensor readings are then sent to the control unit for processing, where they are interpreted and used for further computational operations.

3.1.3 Write Head

The write head is equipped with a marker controlled by two servo motors, which guide the marker along a rail. The servo motors position the marker to either write a new symbol or leave the existing symbol intact, based on the instructions provided by the control unit. This mechanism ensures that the machine can perform write operations on the tape, modifying its content as specified by the transition rules.

3.1.4 Erase Head

The erase head consists of a felt cylinder driven by a DC motor. This cylinder is used to erase the current symbol on the tape, preparing the cell for overwriting. Upon receiving an erase command from the control unit, the felt cylinder engages with the tape, clearing the symbol in the current cell.

3.1.5 Control Panel

An Arduino serves as the central control unit of the system, managing the operation of all components based on the transition rules and the initial memory configuration, which is derived from the punched card input. The control panel features input buttons that allow the user to start, stop, step through the program, load the state table, or reset the machine. An LCD display provides real-time feedback to the user, showing the current state of the Turing machine, including the content of the tape, the control state, and the operational status of the machine.

3.1.6 Punched Card Reader

The punched card reader is an essential component for reading binary data encoded in the punch cards. A microcontroller with imaging capabilities, specifically the ESP32-CAM, is employed to capture and decode the data. The ESP32-CAM takes an image of the punch card, detecting the hole patterns that represent binary data. It then processes the image, deciphers the binary patterns, and sends the decoded transition rules to the Arduino control unit. This system allows for the efficient input of computational instructions into the UTM, enabling automated program execution based on the provided punch card data.

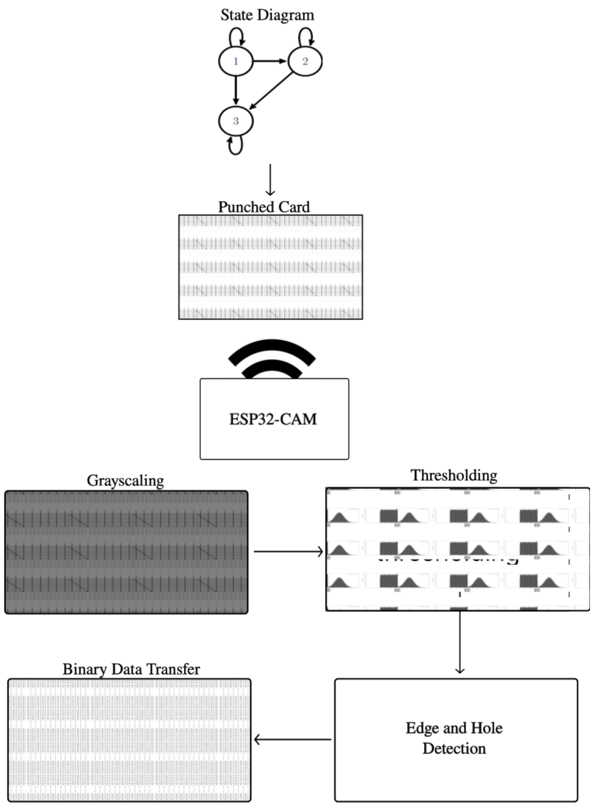


Figure 3.2: Working of the punched card Reader

3.2 Working Principle

3.2.1 Image Processing Pipeline

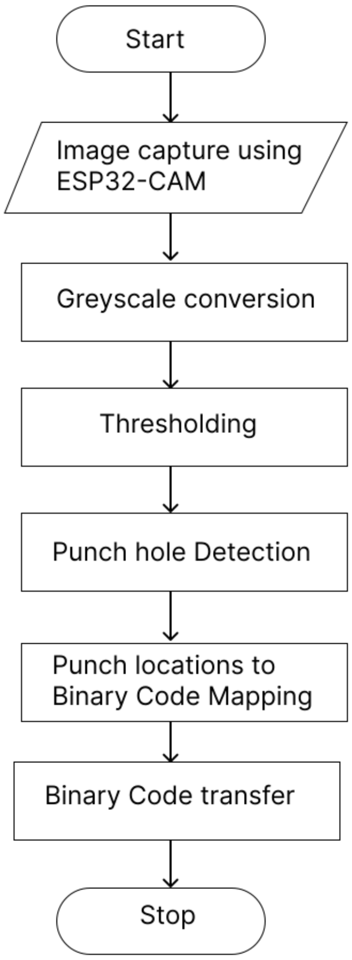


Figure 3.3: Proposed image processing flowchart

The flowchart in Figure 3.3 illustrates the image processing steps executed by the ESP32-CAM microcontroller. The input data, provided through the punch card, is captured by the ESP32-CAM using its onboard camera. The image is processed through a series of steps outlined below. Preprocessing ensures that the captured image is suitable for decoding. This step involves several stages, including greyscaling, binarization, and edge detection.

Greyscaling

Greyscaling is the process of converting a color image, typically composed of three channels—Red (R), Green (G), and Blue (B)—into a single-channel grayscale image. Each pixel in a color image has three values, for example: $[255, 0, 0]$ for red, and

$[0, 255, 0]$ for green. A grayscale image contains only one intensity value ranging from black (0) to white (255). We use the weighted method (also known as the luminosity method) to convert an RGB image to grayscale. This method accounts for the human eye's sensitivity to different colors by applying a weight to each color channel:

$$\text{Grayscale} = 0.299R + 0.587G + 0.114B$$

Binarization

Binarization converts a grayscale image into a black-and-white (binary) image. This is a critical step for simplifying the image and enhancing the detection of objects or patterns. Two main methods are used for binarization: local thresholding and global thresholding.

Local thresholding calculates a threshold for each pixel based on its local neighborhood statistics. This method divides the image into smaller regions, where thresholds are calculated individually for each region based on pixel intensity variations. Local thresholding is particularly useful when dealing with images that exhibit varying lighting conditions across the image.

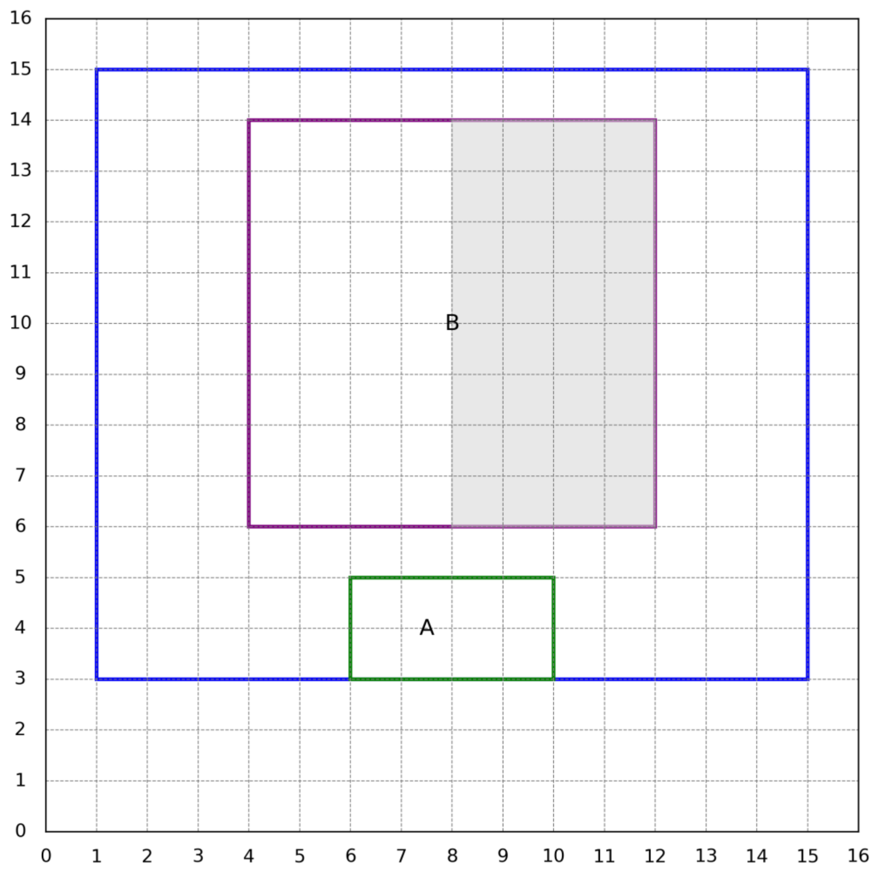


Figure 3.4: Binarization through local thresholding

In Figure 3.4, the 16x16 grid is divided into regions, and thresholds are calculated for each region to segment elements such as grid lines and shaded areas.

Global thresholding uses the image histogram to determine a threshold. The histogram plots the pixel intensities on the x-axis and the number of pixels at each intensity on the y-axis. The threshold is typically determined as the intensity that divides the foreground from the background, and any pixel with intensity above the threshold is set to white, while pixels below it are set to black.

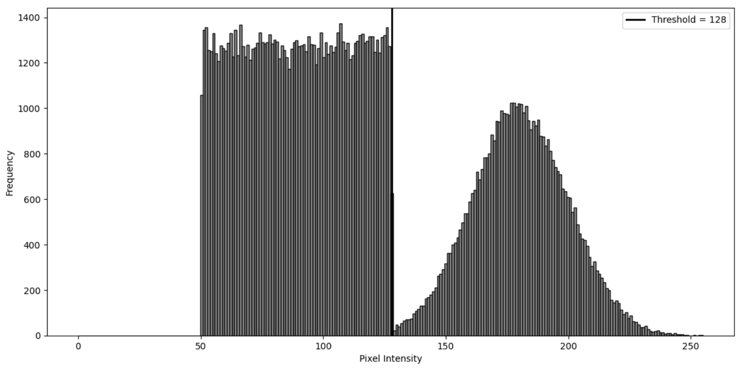


Figure 3.5: Binarization through global thresholding

In Figure 3.5, synthetic data is used to demonstrate the binarization process. The threshold divides the pixel intensities into foreground and background regions, simplifying the image for subsequent processing.

Edge Detection and Region of Interest(ROI) Analysis

Edge detection represents a critical phase in our implementation, utilizing the Sobel operator for its efficient computation and reliable results. Depending on the results during the experimentation phase, we may switch to other kernels or methods like Prewitt or CNN. The Sobel operator applies two 3x3 kernels to calculate horizontal and vertical gradients, followed by magnitude computation through the square root of their squared sums. This process effectively identifies punch card boundaries and hole edges, crucial for subsequent analysis stages.

$$\text{Horizontal Gradient (Gx)} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$\text{Vertical Gradient (Gy)} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The magnitude of the gradient is computed as:

$$\text{Magnitude} = \sqrt{Gx^2 + Gy^2}$$

The ROI extraction follows edge detection, focusing computational resources on the relevant portions of the image. This stage implements precise boundary detection algorithms to isolate the punch card area, eliminating background noise and irrelevant features. The system then establishes a coordinate system aligned with the punch card's physical layout, creating a virtual grid that maps to potential hole locations while compensating for minor alignment deviations.

Grid Mapping and Lateral Histogram Method(LAT)

Grid mapping divides the image into a fixed grid of pre-defined rows and columns, each corresponding to a specific data point or character on the punch card. The coordinates of each grid cell are computed based on the known dimensions of the punch card, ensuring alignment with the actual punch locations.

For hole detection, we employ the LAT method which analyzes pixel intensity distributions along rows and columns. Horizontal and vertical histograms are created, showing intensity variations that indicate the presence of vertical and horizontal punch lines, respectively. Peaks in these histograms correspond to potential holes. By cross-referencing these peaks, the exact locations of the holes are determined.

Table 3.1: Efficiency of Lateral Histogram Method (LAT)

LAT	Efficiency Formula
LAT	$[2N^2 + (4 + 4r)N + 4p + (20 + 24r + 8r^2)p^2]t_2 + 2Nt_2$
LAT'	$[2N^2 + (4 + 4r)N + 4p + 18p^2 + (20 + 24 + 8r^2)q^2]t_1 + 2Nt_2$

Where N is the number of pixels, r is the number of edge points, p is the number of detected holes, and q is the number of true holes.

Hole Classification

Once the holes are detected, the intensity values within each cell of the grid are analyzed. Dark areas (holes) typically have lower intensity values, while light areas (non-holes) have higher intensity values. The relative intensity of each grid cell is compared to its neighbors to classify each cell as either a hole or non-hole.

$$\text{Binary Matrix: } \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

In the binary matrix above, "1" represents the presence of a hole, and "0" represents no hole. This binary data is then sent to an Arduino Mega for further processing.

3.3 Turing Machine Simulation - tlang

To supplement the hardware implementation, a simulation of the Turing Machine, along with a custom language to manipulate the simulated machine, was developed to observe the machine's operations digitally [9]. This simulation serves multiple purposes, including validating transition rules, testing various algorithms, and providing a reference for the hardware's expected behavior. tlang was implemented using C, chosen for its simplicity and efficiency, which also makes it suitable for potential porting to the hardware during the development phase.

3.3.1 Design and Features

The simulator is designed to replicate the core components and operations of a Turing Machine, featuring:

- A custom language for defining the state transition table, which specifies the current state, tape symbol modification, head movement direction, and the next state for each input state-symbol pair.
- A memory tape represented as an extendable list, initialized with defined input data.
- A head pointer that moves left or right across the tape, controlled by the state transitions defined in the language.

The language comprises a minimal set of keywords: {F, N, R, L, S, H}, into which any arbitrary state table can be reduced. Detailed documentation of tlang's syntax and functionality is available on its GitHub page. The simulation executes step-by-step, enabling users to observe the machine's operation as it transitions through states, modifies the tape, and moves the head.

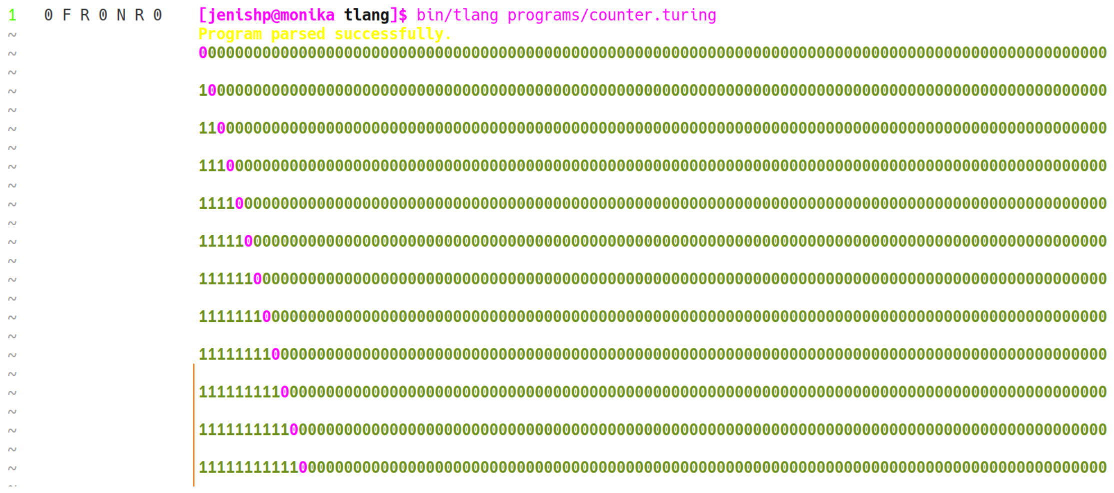
3.3.2 Application and Insights

Before the hardware implementation, the transition rules were extensively tested using the simulator. This helped identify logical errors and edge cases that could arise during execution. By simulating various scenarios, the simulator ensured the correctness of the rules and validated the feasibility of algorithms intended for the hardware. These tests provided valuable insights into the inner workings of a Turing Machine and informed design decisions for the hardware implementation.

3.3.3 Visualization

The simulator features a graphical interface that displays:

- The current tape content, with the active cell highlighted.
- The current state of the Turing Machine.
- The head's position on the tape.



```
0 F R 0 N R 0
```

```
[jenishp@monika tlang]$ bin/tlang programs/counter.turing
Program parsed successfully.
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111111110000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111111111000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

1111111111100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

Figure 3.6: Visualization of the tlang simulator in operation

3.4 System Loop Flowchart

The state-driven process of the system begins with loading the state table and memory tape. These inputs are provided via punched cards and processed by the ESP32-CAM, which converts the punch locations into binary code. The resulting binary data is then transferred to the Arduino Mega, where both the state table and the data are loaded simultaneously with the memory tape.

The system enters a continuous loop where the current state is validated against the state table. If the state is valid, the Arduino Mega coordinates the necessary actions, including:

- Moving the memory tape using the stepper motor.
- Controlling the read, write, and erase heads via servo motors.
- Processing sensor inputs.

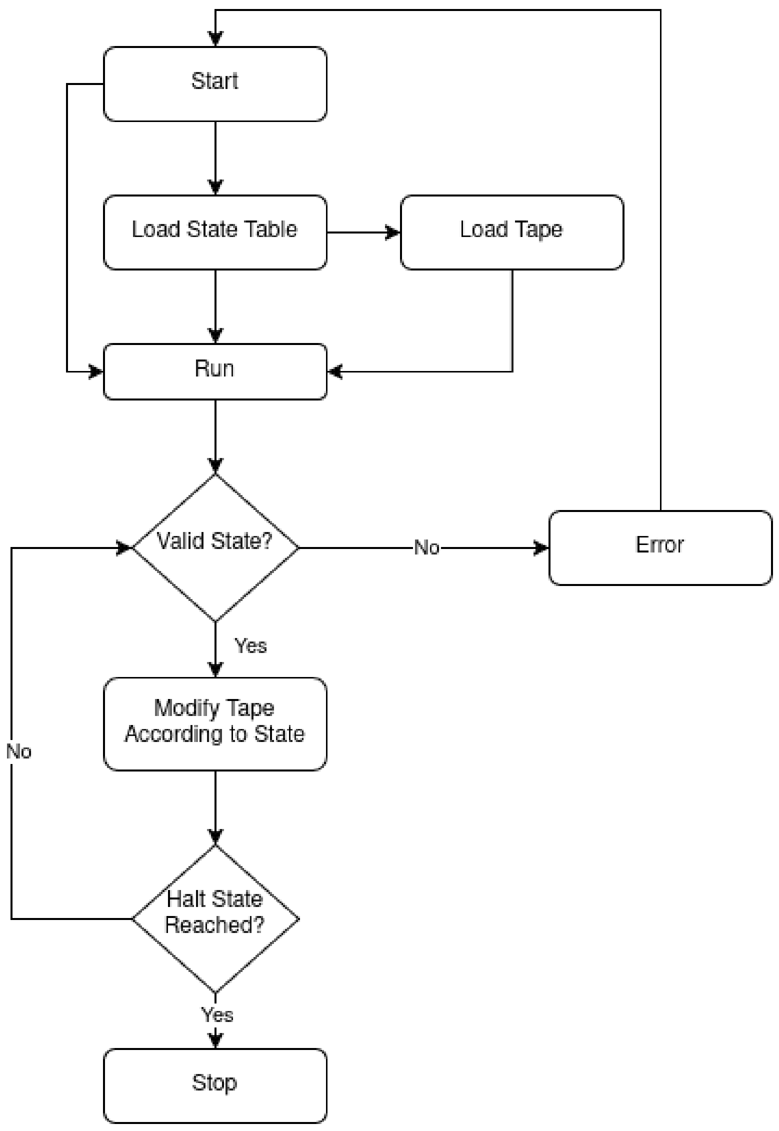


Figure 3.7: System Flowchart

The memory tape is modified according to the current state, and the system transitions to the next state as defined in the state table. If the halt state has not been reached, the system continues to process the next bit and repeats the loop.

If an invalid state is encountered at any point, the process halts and displays an error message. The loop terminates only when the halt state is successfully reached, signaling the completion of the operation.

3.5 Instrumentation Specification

3.5.1 Hardware Requirements

The following hardware components are required for the project:

- **Arduino Mega** (1 unit): A microcontroller board with 256KB Flash and 8KB

SRAM, used for controlling project components.

- **ESP32 Cam** (1 unit): A 2MP camera with Wi-Fi, Bluetooth, 3.3V, and MicroSD functionality (27.5x40mm), used for image capture and processing.
- **Nema 17 Stepper Motor** (1 unit): A motor with 2A current and 1.8° step angle, used for precise, repeatable rotational movement.
- **9g Servos** (3 units): Servos with 4.8-6V, 2.5 kg.cm torque, 0.1s/60° speed, and 9g weight, used for precise angular movement and control.
- **Matrix Boards** (1 unit): A fiberglass board with a 2.54 mm grid and copper pads, used for prototyping and circuit connections.
- **White 35mm Film Leader or Lamination Sheet** (1 unit): A flexible and durable material with 35mm width that can be easily written to and erased using a marker, used as the memory.
- **Soldering Iron and Wire** (1 unit): A 60W soldering iron and soldering wire, used for assembling electronic components onto the matrix board.
- **Display and Buttons** (1 LCD and 3 push buttons): Used for user interface and input control.
- **3D Printed Structural Components** (multiple units): Custom-designed 3D printed parts for the physical framework.
- **7805 5V Regulator** (2 units): Provides a 5V output (1A max) to regulate voltage for project components.
- **AMS 1117 3.3V Regulator** (1 unit): Provides a 3.3V output (1A max) to regulate voltage for project components.
- **12V Adapter** (1 unit): A 12V DC output adapter to provide power supply to the project components.
- **CNY-70 IR Sensor** (1 unit): An infrared emitter and detector (5mm gap), used to output different values based on the reflectance of the surface in front of it.
- **Capacitors and Resistors**: $2 \times 10\ \mu$F capacitors and $3 \times (220\ \Omega, 1\ \text{k}\Omega)$ resistors: Capacitors used for the 7805 voltage regulator and resistors with CNY70 sensors as current limiter and voltage divider.

3.5.2 Software Requirements

Arduino IDE

An integrated development environment (IDE) for writing and uploading code to Arduino boards. It is available on Windows, macOS, and Linux platforms, and supports programming in C/C++ along with a wide range of libraries.

Onshape

A cloud-based 3D CAD software for designing and modeling electronic and mechanical parts. Onshape is platform-independent (works through browsers on Windows, macOS, and Linux). It supports parametric modeling, assemblies, and real-time collaboration. Files can be exported in common formats like STL, STEP, and IGES.

EasyEDA

An online electronic design automation (EDA) tool for circuit design, simulation, and PCB layout. EasyEDA is web-based and provides schematic capture, PCB design, and simulation tools. It supports exporting files in Gerber, PDF, and more. EasyEDA integrates with JLCPCB for PCB fabrication.

4. EXPECTED RESULT

The proposed Universal Turing Machine (UTM) is expected to closely resemble Mike Davey's Turing Machine in form and functionality but will feature several key improvements and innovations aimed at enhancing its usability, versatility, and educational value [6]. Mike Davey's ”A Turing Machine” is widely regarded as a foundational demonstration of Turing's theoretical principles in physical form. Its design effectively captures the essential components of a Turing Machine, including the tape, read/write heads, and control mechanisms. However, the proposed UTM aims to build upon this foundation by addressing certain limitations and introducing features, such as:

- **Punched Card Programming:** Unlike Davey's machine, the proposed UTM will utilize punched cards for programming. This allows for a more user-friendly and modular way of defining state transition tables and algorithms, simplifying the process of reprogramming the machine for different tasks.
- **Improved Hardware Design:** By integrating modern components such as stepper motors, servo motors, and IR sensors, the hardware will provide higher precision, reliability, and flexibility. The control system, based on Arduino and ESP32-CAM, ensures seamless coordination of all mechanical and computational operations.
- **Educational Utility:** The proposed design emphasizes accessibility for educational purposes. Features such as the paper-programmable interface, LCD display, intuitive control panel, and a digital simulation (tlang) complement the physical machine to aid understanding of Turing's concepts.

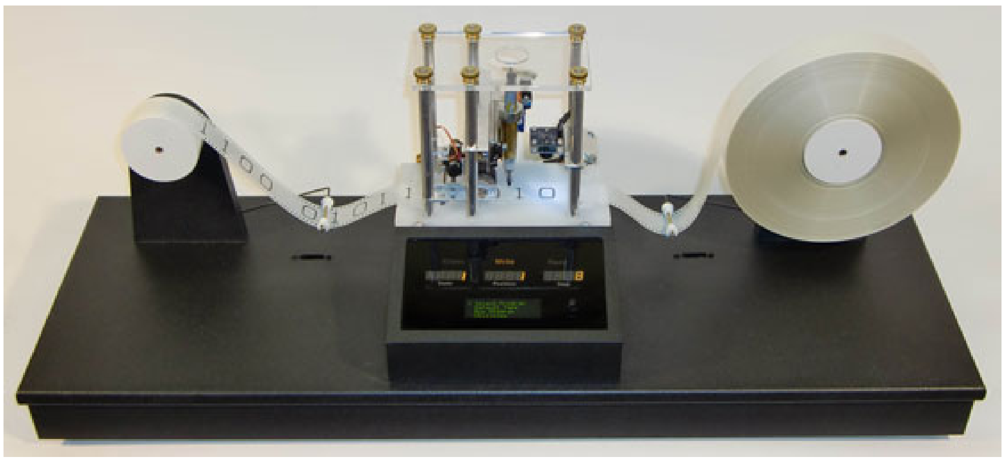


Figure 4.1: Mike Davey's ”A Turing Machine”

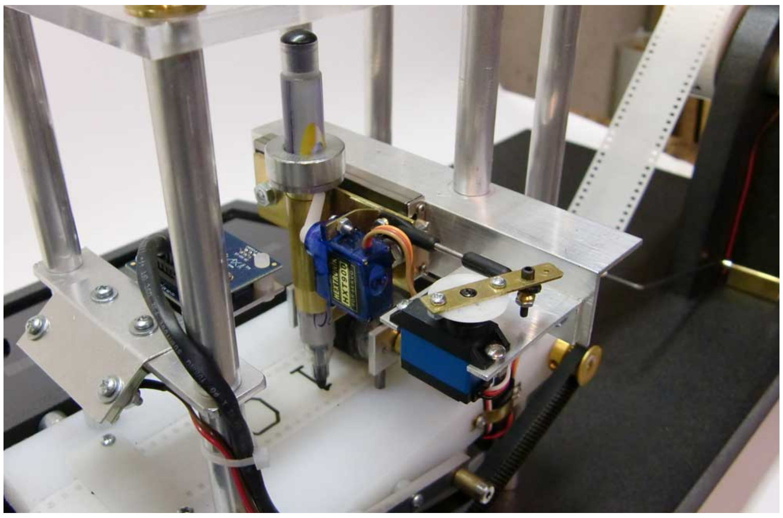


Figure 4.2: Mike Davey's Read/Write Head

Figure 4.1 illustrates the physical design of Mike Davey's Turing Machine, which serves as an inspiration for this project. While the proposed UTM shares a similar conceptual framework, the above-mentioned enhancements are expected to elevate its practical utility and user experience. The completed UTM will not only demonstrate Turing's theoretical concepts in hardware but also provide a platform for practical experimentation with algorithms, computational logic, and the principles of discrete mathematics. It is expected to serve as a valuable educational tool, as well as a proof-of-concept for bridging theoretical computation with real-world applications.

Figure 4.2 illustrates Mike Davey's Turing Machines's read/write head, which forms a critical part of the machine's operation. The proposed UTM's read/write head is expected to work in a similar manner with two servos moving the writing mechanism in two axes. However, the proposed design introduces several enhancements aimed at improving precision, reliability, and versatility. The proposed UTM will employ an infrared (IR) sensor for symbol detection on the tape, ensuring accurate readings under various lighting conditions. Writing and erasing operations will be facilitated by a servo motor-driven mechanism, allowing precise symbol placement and modification. Additionally, the hardware will include a more robust tracking system for the tape position, reducing mechanical drift and improving overall operational accuracy.

In conclusion, while inspired by Mike Davey's pioneering work, the proposed UTM seeks to extend the capabilities of its predecessor. By integrating advanced hardware components and user-friendly features such as punched card programming and a complementary digital simulation, the UTM aims to provide a comprehensive educational and research platform that bridges the gap between theoretical and applied computational science.

5. FEASIBILITY ANALYSIS

The development of a Universal Turing Machine (UTM) as a physical hardware model is highly feasible, given the accessibility of modern tools, components, and resources. This section evaluates the project's feasibility across key dimensions: technical, operational, temporal, economic, and complexity considerations.

5.1 Technical Feasibility

The proposed UTM relies on widely available and cost-effective components. A microcontroller, such as the Arduino Mega, will manage the machine's logic and operations. Stepper motors and servo motors will power the tape movement system, while an ESP32-CAM module and CNY70 IR sensors will handle input processing and tape position detection. The system will use punched cards for input, enabling modular and reprogrammable state transition definitions.

All required components, including power supplies, voltage regulators, resistors, and capacitors, are inexpensive and supported by extensive documentation and tutorials. Software tools such as the Arduino IDE simplify microcontroller programming, while the simulator developed alongside, tlang[9] can simulate and test the UTM's logic before hardware integration.

Reliable power sources, such as 12V adapters or 3S LiPo batteries, will ensure consistent operation and voltage regulators (e.g., AMS1117 3.3V, LM7805) will maintain safe power levels for all components, ensuring robust and uninterrupted performance.

5.2 Operational Feasibility

The project is operationally viable due to its use of well-documented, modular components. Systems such as tape movement, input processing, and state transitions can be developed and tested independently before integration. The hardware's modularity reduces complexity and allows for incremental assembly.

The UTM is designed with educational usability in mind. The punched card input system is intuitive and user-friendly, while the Arduino IDE provides a straightforward programming environment. Comprehensive online resources and community support further ensure ease of operation and troubleshooting during development and deployment.

5.3 Time Feasibility

A detailed project timeline has been prepared to guide the development phase. The project is achievable within a structured eleven-week timeline after the initial stages of proposals and presentations:

- **Weeks 1–2:** Beginning of Research, procurement of components, and hardware design.
- **Weeks 3–5:** Assembly and start of real world testing of individual subsystems, including tape movement and input processing while simultaneous documentation.
- **Weeks 6–7:** Integration of hardware and software, programming machine logic, and debugging.
- **Weeks 8–11:** Final testing, demonstrations, and report preparation.

This phased approach ensures manageable workloads, regular progress, and timely project completion. Regular milestones and progress reviews will help maintain the schedule and ensure timely completion.

5.4 Economic Feasibility

The project is economically feasible due to the affordability of its components. The primary hardware, including the ESP32-CAM, Arduino Mega, stepper motors, and servo motors, is budget-friendly and widely accessible. Supporting components, such as sensors, resistors, and capacitors, incur minimal costs.

Software development leverages free tools such as the Arduino IDE and open-source libraries, further reducing expenses. The overall cost remains low while maintaining the project's educational and functional goals.

5.5 Efficiency

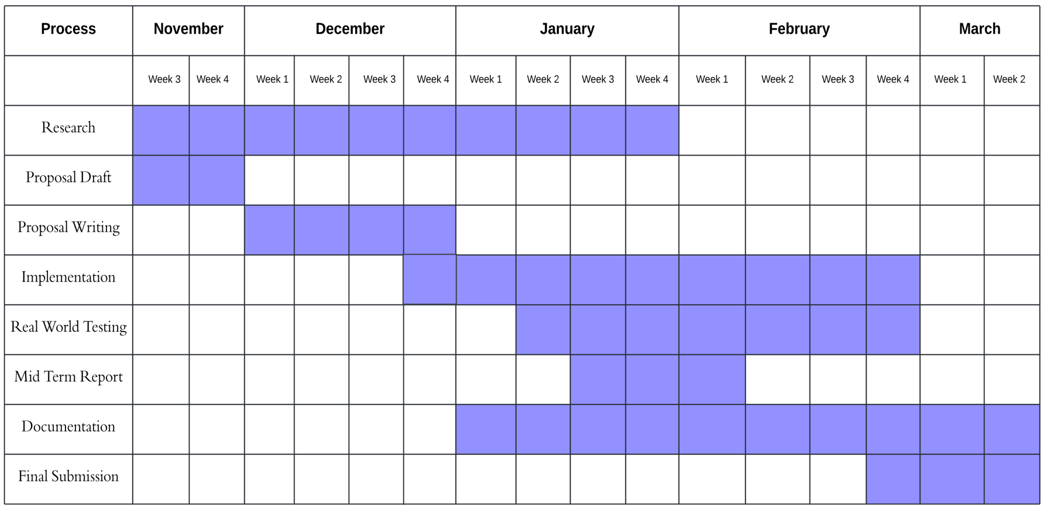
The UTM is not designed for high-speed or large-scale computation but excels at demonstrating the fundamental principles of computation. The punched card input system emphasizes the historical context of early computational methods, making it an effective educational tool. Although slower than digital systems, it efficiently illustrates the core ideas of computation for small-scale tasks and learning purposes.

5.6 Complexity

The project strikes a balance between complexity and manageability. It requires basic knowledge of electronics, programming, and mechanical systems. The assembly process is straightforward, supported by detailed documentation and community resources. The software includes binary conversion of punched card inputs, state machine logic, and motor control, which, while requiring attention to detail, is well within the capabilities of those with foundational programming skills. The integration of hardware and software provides practical experience in system design, making the project suitable for students and hobbyists seeking to deepen their understanding of computational theory and hardware development.

6. APPENDICES

Appendix A: Gantt Chart



Process	November		December				January				February				March	
	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2
Research	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦						
Proposal Draft	🟦	🟦														
Proposal Writing			🟦	🟦	🟦	🟦										
Implementation						🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦		
Real World Testing								🟦	🟦	🟦	🟦	🟦	🟦	🟦		
Mid Term Report									🟦	🟦	🟦					
Documentation							🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦	🟦
Final Submission														🟦	🟦	🟦

Figure 6.1: Project Timeline

Appendix B: Budget Estimation

Table 6.1: Project Budget

S.N.	**Components**	**Quantity**	**Unit Price (NPR)**	**Total Price (NPR)**
1	Arduino	1	1500	1500
2	ESP32-CAM	1	900	900
3	Nema 17 Stepper Motor	1	1500	1500
4	A4988 Stepper Driver	1	210	210
5	9g Servo	3	270	810
6	Matrix Board	3	100	300
7	Lamination Sheets	10	10	100
8	Display	1	250	250
9	Buttons	3	10	30
10	7805 5V Regulator	2	15	30
11	AMS1117 3.3V Regulator	1	10	10
12	Capacitors and Resistors	2 + 6	5	40
13	12V Adapter	1	300	300
14	CNY70 IR Sensor	3	69	207
Total				**6187**

Appendix C: Detailed Component Specifications

- Arduino: 16MHz clock speed, 54 digital I/O pins, 16 analog inputs, 4 UARTs, 6 PWM outputs, 1 USB connection, and power jack.
- ESP32-CAM: Wi-Fi and Bluetooth functionality, 2MP camera with OV2640, 520MHz dual-core processor, supports up to 16MB of flash memory.
- Nema 17 Stepper Motor: 1.8° step angle, 12V rated voltage, holding torque 40N·cm, stepper motor type, typically used in 3D printers.
- A4988 Stepper Driver: Full-step, half-step, and microstep operation modes, adjustable current control, thermal shutdown and overload protection.
- 9g Servo: Continuous rotation, torque 2.5kg.cm, operating voltage 4.8V to 6.0V, with a speed of 0.12s/60°.
- Matrix Board: Breadboard with a 400 tie-points capacity, ideal for prototyping.
- Lamination Sheets: A4 size, standard thickness for memory cells.
- Display: 16x2 LCD Display, used for displaying text and information, operates on 5V, with a built-in controller.
- Buttons: Pushbutton switches, rated for 50,000 cycles, typically used for user inputs.
- 7805 5V Regulator: Output voltage 5V, output current up to 1A, used to regulate input voltages to a stable 5V for powering logic circuits.
- AMS1117 3.3V Regulator: Output voltage 3.3V, output current up to 800mA, widely used in low power applications.
- Capacitors and Resistors: Capacitors for voltage regulator, resistors for current limiting and voltage division.
- 12V Adapter: 12V DC output, typically used to power high-voltage components such as motors and sensors.
- CNY70 IR Sensor: Reflective optical sensor, with phototransistor output, used for optoelectronic scanning.

BIBLIOGRAPHY

[1] A. Turing, “On computable numbers, with an application to the entscheidungsproblem,” *Proceedings of the London Mathematical Society*, vol. 2, no. 42, pp. 230–265, Dec. 1936. [Online]. Available: https://www.cs.virginia.edu/~robins/Turing_Paper_1936.pdf

[2] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 3rd ed. Boston, MA: Addison-Wesley, 2006.

[3] A. Church, “Review of turing’s paper,” *Journal of Symbolic Logic*, vol. 2, pp. 41–45, Dec. 1937. [Online]. Available: https://www.jstor.org/stable/1990093

[4] C. Shannon and J. McCarthy, *Automata Studies*. Princeton University Press, Dec. 1956. [Online]. Available: https://www.amazon.com/Automata-Studies-Addison-Wesley-Monographs-Mathematics/dp/B0007E5YCG

[5] S. H. Rodger, “Jflap: Java formal languages and automata package,” Dec. 1990s. [Online]. Available: https://www.jflap.org

[6] M. Davey, “Physical models of turing machines,” Dec. 2010. [Online]. Available: http://www.mikedavey.com

[7] D. Deutsch, *The Structure and Interpretation of Quantum Mechanics*. Cambridge, MA: Harvard University Press, Dec. 1985. [Online]. Available: https://www.hup.harvard.edu/catalog.php?isbn=9780674445829

[8] E. Bernstein and U. Vazirani, “Quantum turing machines,” *SIAM Journal on Computing*, vol. 26, no. 5, pp. 1400–1411, Dec. 1993. [Online]. Available: https://link.springer.com/chapter/10.1007/3-540-57564-6_24

[9] monoastro, “tlang: A turing machine simulator and language,” https://github.com/monoastro/tlang, 2024, accessed: 2024-12-28.