VISION BASED CNC PnP MACHINE WITH MACHINE LEARNING

A PROJECT REPORT

Submitted by

DHIRAJ ZEN B K (2020105522)

RISHIKESH SELVARAJ PILLAI (2020105038)

NIRENJAN K (2020105557)

AHILESH V (2020105502)

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

COLLEGE OF ENGINEERING GUINDY,

ANNA UNIVERSITY: CHENNAI 600025

MAY 2024

ANNA UNIVERSITY, CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this report titled as, **"VISION BASED CNC PNP MACHINE WITH MACHINE LEARNING"** is the Bonafide work of **RISHIKESH SELVARAJ PILLAI (2020105038), AHILESH.V (2020105502) NIRENJAN.K(2020105557), DHIRAJ ZEN B K (2020105522)** who carried out the work under my supervision. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation based on which a degree or award was conferred on an earlier occasion on this or any other candidate.

Signature of the HOD with Date
Dr. M. MEENAKSHI
Professor and Head
Department of Electronics and Communication Engineering
Anna University
Chennai -600025.

Signature of the Supervisor with Date
Dr. N. RAMADASS
Professor
Department of Electronics and Communication Engineering
Anna University
Chennai -600025.

ACKNOWLEDGEMENT

We express our wholehearted gratitude to our dynamic and zestful Head of the Department **Dr. M. MEENAKSHI**, Professor, Department of Electronics & Communication Engineering for her continued and heartening support throughout the project.

We manifest our sincere thanks and regards to our project supervisor, **Dr. N. RAMADASS**, Professor, Department of Electronics and Communication Engineering for his ardent support, patience, valuable guidance, technical expertiseand encouragement in our project.

Special thanks to **Dr. R. SEETHARAMAN** the project coordinator and the project review committee members for their valuable suggestions.

We would like to mention a special thanks to **Mr. K.R. VIMAL**, Senior technical assistant, and **Mr. S. KUMARAN**, Professional Assistant, for their continuous encouragement during my project work.

We would like to thank **Mr. ARUNAGIRI**, Teaching Fellow, Department of Bio-Medical Engineering for his valuable suggestions and help.

Also, we would like to thank all the staff members of the Department of Electronics and Communication Engineering, who offered help during the course for this project work.

A special gratitude to our friends and family members, who supported us, encouraged us, and stood with us against all odds throughout the duration of this course.

ABSTRACT

This project presents the development of a cutting-edge CNC (Computer Numerical Control) pick-and-place machine integrated with vision-based technology and machine learning algorithms. Traditional CNC machines are limited by their fixed programmed paths, making them less adaptable to variable tasks. To overcome this limitation, our system employs a vision-based approach, enabling real-time detection and analysis of workpieces.

The core of our system lies in the implementation of machine learning algorithms for object recognition and localization. By training a convolutional neural network (CNN) on a diverse dataset, our system can accurately identify objects of interest within the workspace. This allows for dynamic adaptability, enabling the CNC machine to adjust its operations based on the type, position, and orientation of the detected objects.

Real-time object detection is achieved through the utilization of high-resolution cameras and advanced image processing techniques. This enables the system to detect and classify objects with high precision and speed. Furthermore, dynamic path planning is facilitated by the integration of machine learning algorithms, allowing the CNC machine to dynamically plan its toolpath based on the detected objects. This optimization enhances efficiency and flexibility in manufacturing processes.

In addition to its adaptive capabilities, the system incorporates error handling mechanisms and redundancy checks to ensure robustness and reliability in operation. This minimizes the risk of errors and failures, enhancing the overall performance of the system. The user interface of the system is designed to be intuitive, enabling operators to easily interact with the machine and monitor its performance. This ensures ease of use and facilitates seamless integration into existing manufacturing environments.

Through extensive testing and validation, our system demonstrates superior performance compared to conventional CNC machines, particularly in tasks requiring adaptability to variable workpieces. The integration of vision-based technology and machine learning algorithms opens up new possibilities for the automation of complex manufacturing processes, paving the way for more efficient and flexible production systems.

In conclusion, our project showcases the potential of combining vision-based systems with CNC technology, offering a promising solution for the next generation of automated manufacturing systems.

திட்டப்பணிச்சுருக்கம்

பார்வை அடிப்படையிலான தொழில்நுட்பம் மற்றும் இயந்திர கற்றல் வழிமுறைகளுடன் ஒருங்கிணைக்கப்பட்ட அதிநவீன CNC (கணினி எண் கட்டுப்பாடு) பிக்-அண்ட்-பிளேஸ் இயந்திரத்தின் வளர்ச்சியை இந்தத் திட்டம் வழங்குகிறது. பாரம்பரிய CNC இயந்திரங்கள் அவற்றின் நிலையான நிரல்படுத்தப்பட்ட பாதைகளால் வரையறுக்கப்பட்டுள்ளன, இதனால் அவை மாறி பணிகளுக்கு குறைவாக மாற்றியமைக்கப்படுகின்றன. இந்த வரம்பைக் கடக்க, எங்கள் அமைப்பு பார்வை அடிப்படையிலான அணுகுமுறையைப் பயன்படுத்துகிறது, இது நிகழ்நேர கண்டறிதல் மற்றும் பணியிடங்களின் பகுப்பாய்வு ஆகியவற்றை செயல்படுத்துகிறது.

எங்கள் அமைப்பின் மையமானது பொருள் அங்கீகாரம் மற்றும் உள்ளூர்மயமாக்கலுக்கான இயந்திர கற்றல் வழிமுறைகளை செயல்படுத்துவதில் உள்ளது. பலதரப்பட்ட தரவுத்தொகுப்பில் கன்வல்யூஷனல் நியூரல் நெட்வொர்க்கை (CNN) பயிற்றுவிப்பதன் மூலம், எங்கள் அமைப்பு பணியிடத்தில் ஆர்வமுள்ள பொருட்களை துல்லியமாக அடையாளம் காண முடியும். கண்டறியப்பட்ட பொருட்களின் வகை, நிலை மற்றும் நோக்குநிலை ஆகியவற்றின் அடிப்படையில் CNC இயந்திரத்தை அதன் செயல்பாடுகளை சரிசெய்ய, மாறும் தகவமைப்புக்கு இது அனுமதிக்கிறது.

உயர் தெளிவுத்திறன் கொண்ட கேமராக்கள் மற்றும் மேம்பட்ட பட செயலாக்க நுட்பங்களைப் பயன்படுத்துவதன் மூலம் நிகழ்நேர பொருள் கண்டறிதல் அடையப்படுகிறது. இது அதிக துல்லியம் மற்றும் வேகத்துடன் பொருட்களைக் கண்டறிந்து வகைப்படுத்த கணினியை செயல்படுத்துகிறது. மேலும், டைனமிக் பாதை திட்டமிடல் இயந்திர கற்றல் வழிமுறைகளின் ஒருங்கிணைப்பால் எளிதாக்கப்படுகிறது, இது CNC இயந்திரம் கண்டறியப்பட்ட பொருட்களின் அடிப்படையில் அதன் கருவிப்பாதையை மாறும் வகையில் திட்டமிட அனுமதிக்கிறது. இந்த தேர்வுமுறையானது உற்பத்தி செயல்முறைகளில் செயல்திறன் மற்றும் நெகிழ்வுத்தன்மையை அதிகரிக்கிறது.

அதன் தழுவல் திறன்களுக்கு கூடுதலாக, கணினியானது செயல்பாட்டில் வலிமை மற்றும் நம்பகத்தன்மையை உறுதி செய்வதற்காக பிழை கையாளும் வழிமுறைகள் மற்றும் பணிநீக்க சோதனைகளை உள்ளடக்கியது. இது பிழைகள் மற்றும் தோல்விகளின் அபாயத்தைக் குறைக்கிறது, கணினியின் ஒட்டுமொத்த செயல்திறனை அதிகரிக்கிறது. கணினியின் பயனர் இடைமுகம் உள்ளுணர்வுடன் வடிவமைக்கப்பட்டுள்ளது, ஆபரேட்டர்கள் இயந்திரத்துடன் எளிதாக தொடர்பு கொள்ளவும் அதன் செயல்திறனை கண்காணிக்கவும் உதவுகிறது. இது பயன்பாட்டின் எளிமையை உறுதி செய்கிறது மற்றும் தற்போதுள்ள உற்பத்தி சூழல்களில் தடையற்ற ஒருங்கிணைப்பை எளிதாக்குகிறது.

விரிவான சோதனை மற்றும் சரிபார்ப்பு மூலம், வழக்கமான CNC இயந்திரங்களுடன் ஒப்பிடும்போது, குறிப்பாக மாறக்கூடிய பணியிடங்களுக்கு மாற்றியமைக்கத் தேவைப்படும் பணிகளில், எங்கள் கணினி சிறந்த செயல்திறனை வெளிப்படுத்துகிறது. பார்வை அடிப்படையிலான தொழில்நுட்பம் மற்றும் இயந்திர கற்றல் வழிமுறைகளின் ஒருங்கிணைப்பு சிக்கலான உற்பத்தி செயல்முறைகளின் தன்னியக்கத்திற்கான புதிய சாத்தியங்களைத் திறக்கிறது, மேலும் திறமையான மற்றும் நெகிழ்வான உற்பத்தி அமைப்புகளுக்கு வழி வகுக்கிறது.

முடிவில், எங்கள் திட்டம் CNC தொழில்நுட்பத்துடன் பார்வை அடிப்படையிலான அமைப்புகளை இணைக்கும் திறனைக் காட்டுகிறது, இது அடுத்த தலைமுறை தானியங்கு உற்பத்தி அமைப்புகளுக்கு ஒரு நம்பிக்கைக்குரிய தீர்வை வழங்குகிறது.

ABBREVIATIONS

CNC	-	Computer Numerical Control
CAD	-	Computer Aided Design
Webcam	-	Web Camera
G-code	-	Geometric code
GPIO	-	General Purpose Input Output
QR	-	Quick Response
PnP	-	Pick and Place

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1. Overview of the Project and Objectives 1
1.2. Introduction to CNC Machines and Their Applications 1
1.3. Project Goals and Significance 2

2. UNDERSTANDING G-CODES AND MANUAL CONTROL 4

2.1. Basics of G-codes 4
2.2. Learning and writing G-codes 4
2.3. Challenges and Solutions in Manual Control 5

3. INTEGRATION OF G-CODE WITH PYTHON 7

3.1. Sending G-code Commands via Serial Port 7
3.2. Python Scripting for Automated Control 7
3.3. Enhancing Functionality with Python Integration 8

4. INTRODUCTION TO MACHINE LEARNING IN CNC MACHINES 10

4.1. Application – TicTacToe 10
4.2. Machine Learning Algorithm 11
4.3. Python code for Minimax Algorithm 12

5. INTRODUCTION TO IMAGE PROCESSING 16

5.1. Basics of image processing 16
5.2. Shape Detection 17
5.2.1. Method 1 (Contour Detection): 17
5.2.2. Method 2 (Canny edge detection): 22
5.2.3. Method 3 (Template Matching): 27
5.3. Hardware used: 39

6. INTEGRATION OF MACHINE LEARNING AND IMAGE PROCESSING WITH G-CODE 41

6.1. Using Machine Learning Algorithm with G-code 41
6.2. Integrating Computer Vision for Real-Time Perception 42
6.3. Python Scripting for Seamless Integration 44

7. HARDWARE DESIGN PROCESS 45

7.1. Designing Game Pieces and Game Base 45
7.2. Creating Dynamic Tool Head with Electromagnet 46
7.3. Casing for CNC Machine using Foam Sheets 47
7.4. Testing and Iteration 48

8. EXPLORING VARIOUS APPLICATIONS 49

8.1. Automation of Blood Sample Analysis 49
8.2. Enhancing Efficiency and Accuracy 50
8.3. Empowering Healthcare Professionals 51
8.4. Exploring Endless Possibilities 51
8.5. Conclusion 51

9. REFERENCES 52

LIST OF FIGURES

FIGURE 1: TYPES OF CNC MACHINES 2
FIGURE 2: EXAMPLE OF A BASIC G-CODE 6
FIGURE 3: SENDING G-CODE VIA SERIAL PORT USING PYSERIAL 7
FIGURE 4: RASPBERRY PI 4 MODULE 9
FIGURE 5: TICTACTOE GAME BOARD 11
FIGURE 6: GAME TREE 12
FIGURE 7: PYTHON CODE FOR MINIMAX ALGORITHM 12
FIGURE 8: GRAYSCALE IMAGE 18
FIGURE 9: BINARY IMAGE 19
FIGURE 10: OUTPUT IMAGE FOR CONTOUR DETECTION 20
FIGURE 11: ERRONEOUS IMAGE DETECTION 22
FIGURE 12: GRAYSCALE IMAGE FOR CANNY EDGE DETECTION 23
FIGURE 13: EDGE DETECTION USING CANNY METHOD 23
FIGURE 14: OUTPUT USING CANNY METHOD 24
FIGURE 15: ERRONEOUS SHAPE DETECTION USING CANNY METHOD 26
FIGURE 16: INPUT BOARD IMAGE FOR TEMPLATE MATCHING METHOD 28
FIGURE 17: TRIANGULAR COIN REFERENCE IMAGE 28
FIGURE 18: CIRCULAR COIN REFERENCE IMAGE 29
FIGURE 19: RESULT OF TEMPLATE MATCHING FOR CIRCULAR COINS 29
FIGURE 20: RESULT OF TEMPLATE MATCHING FOR TRIANGULAR COINS 30
FIGURE 21: OUTPUT IMAGE USING TEMPLATE MATCHING 30
FIGURE 22: CENTRE LOCATIONS OF CIRCULAR AND TRIANGULAR COINS 34
FIGURE 23: DETECTED LOCATIONS OF TRIANGULAR AND CIRCULAR COINS 38
FIGURE 24: LOGITECH WEBCAM 39
FIGURE 25: INITIAL PHASE OF THE APPLICATION USING PENCIL AS TOOL HEAD 41
FIGURE 26: HOLDER FOR PEN/PENCIL 42
FIGURE 27: SECOND PHASE OF THE APPLICATION USING PICK N PLACE MECHANISM 43
FIGURE 28: DYNAMIC ELECTROMAGNET TOOL HEAD FOR PICK N PLACE 43
FIGURE 29: DESIGN OF GAME PIECE EMBEDDED WITH COINS TO MAKE IT MAGNETIC 45
FIGURE 30: DESIGN OF GAME BASE 46
FIGURE 31: HOLDER DESIGN FOR THE ELECTROMAGNET TOOL HEAD SHOWN IN FIGURE 11. 47
FIGURE 32: OVERALL LOOK OF THE MACHINE WITH CASING 48
FIGURE 33: SETUP FOR AUTOMATING BLOOD SAMPLE ANALYSIS 49
FIGURE 34: RBC IMAGE CAPTURED BY THE MICROSCOPE 50
FIGURE 35: USAGE OF QR CODE TO RETRIEVE PATIENT DATA 50

LIST OF TABLES

TABLE 1: G-CODES FOR POSITIONING MODES. 5
TABLE 2: G-CODE FOR MOTION CONTROL 5

1. INTRODUCTION

1.1. Overview of the Project and Objectives

In today's rapidly evolving technological landscape, the integration of automation and intelligent systems has become increasingly prevalent across various industries. Our project endeavours to contribute to this trend by exploring the capabilities of CNC (Computer Numerical Control) machines in conjunction with machine learning and computer vision technologies.

The primary objective of our project is to develop a vision-based CNC pick-and-place machine with machine learning capabilities. This involves enhancing the traditional functionality of CNC machines by integrating them with advanced algorithms for autonomous operation and object detection. By leveraging machine learning and computer vision, we aim to create a versatile system capable of adapting to dynamic tasks and environments.

This project's genesis lies in the recognition of the limitations inherent in conventional CNC systems. While these machines excel at executing pre-defined toolpaths with high precision, they often struggle with tasks that require adaptability to varying workpiece geometries or environmental conditions. By integrating machine learning and computer vision, we envision a CNC machine that can intelligently perceive its surroundings, make informed decisions, and dynamically adjust its operations in real-time.

1.2. Introduction to CNC Machines and Their Applications

CNC machines have revolutionized manufacturing processes by enabling precise and automated control of tool movements. These machines find applications across a wide range of industries, including aerospace, automotive, electronics, and woodworking, among others. Their ability to accurately reproduce complex designs and geometries with minimal human intervention has made them indispensable in modern manufacturing.

However, traditional CNC machines are often limited by their fixed programmed paths, which can hinder their adaptability to variable tasks and workpieces. Our project seeks to overcome these limitations by introducing advanced capabilities such as machine learning and computer vision into the CNC workflow. By doing so, we aim to enhance the flexibility, efficiency, and autonomy of CNC-based systems.

Figure 1: Types of CNC machines

1.3. Project Goals and Significance

The goals of our project are multi-faceted. Firstly, we aim to deepen our understanding of CNC machines and their underlying principles, including G-code programming and control. Through hands-on experimentation and exploration, we seek to gain practical insights into the capabilities and limitations of CNC technology.

Additionally, our project aims to push the boundaries of CNC functionality by integrating machine learning and computer vision algorithms. By enabling the CNC machine to autonomously detect and interact with objects in its environment, we aim to expand its utility beyond traditional machining tasks. This has significant implications for industries such as robotics, automation, and advanced manufacturing.

Furthermore, our project holds broader significance in the context of technological innovation and interdisciplinary collaboration. By bridging the gap between mechanical engineering, computer science, and artificial intelligence, we hope to demonstrate the potential of synergistic approaches in solving complex real-world problems.

2. UNDERSTANDING G-CODES AND MANUAL CONTROL

2.1. Basics of G-codes

G-codes, or Geometric Codes, are a standardized set of commands used to control CNC machines. They dictate the precise movements, speeds, and tool actions necessary to execute a given machining operation. Understanding G-codes is fundamental to programming and operating CNC equipment effectively.

Understanding the syntax and semantics of G-codes is fundamental to programming and operating CNC equipment effectively. The syntax of G-codes typically consists of an alphabetic character (usually "G" for motion commands) followed by a numerical value that specifies a particular action or mode. For example, G0 and G1 are commonly used for rapid positioning and linear interpolation, respectively. Additionally, there are auxiliary codes such as M-codes (e.g., M03 for spindle start) and T-codes (e.g., T01 for tool selection) that control machine functions and tool changes, respectively.

2.2. Learning and writing G-codes

Our journey with G-codes began with a comprehensive study of their syntax, semantics, and application. Through textbooks, online resources, and hands-on experimentation, we familiarized ourselves with the various G-code commands and their corresponding functions. This included commands for linear and circular interpolation, tool changes, dwell times, and more.

As our proficiency grew, we began writing G-code programs to control the CNC machine manually. This involved translating design specifications into a sequence of G-code commands that the machine could interpret and execute. We experimented with different machining strategies, toolpaths, and cutting parameters to achieve desired outcomes. One of the fundamental concepts in G-code programming is the distinction between absolute and incremental coordinate systems:

Table 1: G-codes for positioning modes.

G-code	Description
G90	Set to Absolute Positioning Mode
G91	Set to Incremental Positioning Mode

Understanding and correctly applying these modes are essential for accurate positioning and machining. Additionally, we learned and utilized various motion commands for controlling tool movements:

Table 2: G-code for motion control

G-code	Description
G0	Rapid positioning
G1	Linear interpolation (straight-line movement)
G2	Clockwise circular interpolation
G3	Counterclockwise circular interpolation
G4	Dwell (pause)

2.3. Challenges and Solutions in Manual Control

While working with G-codes manually, we encountered several challenges that tested our problem-solving skills and ingenuity. One common challenge was optimizing toolpaths to minimize machining time and maximize efficiency. This required careful consideration of factors such as tool engagement and judicious use of rapid positioning (G0) and feed motion (G1) commands.

Another challenge was ensuring the accuracy and repeatability of machining operations. Small errors in G-code programming or machine setup could lead to deviations from the intended design, resulting in scrapped parts or rework. We addressed this challenge through rigorous testing, verification, and iteration to fine-tune our G-code programs and machine configurations.

Additionally, we faced challenges related to G-code interpretation and compatibility across different CNC platforms. Each machine may have its unique quirks and idiosyncrasies, requiring adjustments to G-code programs to ensure proper execution. Through collaboration and knowledge sharing, we were able to overcome these challenges and achieve our desired outcomes.

```
1   G21                         ; Set units to mm
2   G90                         ; Absolute positioning
3
4   G28                         ; Home
5   G0 Z10.0                    ; Pen up
6
7   G1 X0 Y0 F3000              ; Move to upper left of bed
8   G0 Z5.0                     ; Pen down
```

Figure 2: Example of a basic G-code

3. INTEGRATION OF G-CODE WITH PYTHON

3.1. Sending G-code Commands via Serial Port

One of the pivotal advancements in our project was the integration of Python scripting for sending G-code commands directly to the CNC machine via the serial port. This allowed for seamless communication between the computer and the CNC machine, enabling automated control and execution of machining operations.

Python provides robust libraries and modules for serial communication, such as PySerial, which facilitated the establishment of a reliable connection between the computer and the CNC machine. By leveraging these libraries, we were able to develop Python scripts that could transmit G-code commands to the CNC machine in real-time, enabling dynamic control and coordination of machining tasks.

```
import serial
ser = serial.Serial('/dev/ttyUSB0', 115200)

init_gcode ='''
G21
G90
G28\r\n
'''

ser.write(init_gcode.encode())
```

Figure 3: Sending G-code via serial port using Pyserial

3.2. Python Scripting for Automated Control

With the capability to send G-code commands via the serial port, we embarked on the development of Python scripts for automated control of the CNC machine. These scripts served as the bridge between high-level commands

generated by the user or external systems and low-level G-code instructions interpreted by the CNC machine.

Through Python scripting, we were able to orchestrate complex machining sequences, coordinate multi-axis movements, and implement conditional logic for decision-making. This level of automation significantly enhanced the efficiency and productivity of the CNC machine, enabling it to execute predefined tasks with minimal human intervention.

3.3. Enhancing Functionality with Python Integration

The integration of Python with G-code control opened up a plethora of possibilities for enhancing the functionality and versatility of the CNC machine. Beyond basic motion control, Python scripting enabled the implementation of advanced features such as:

- Dynamic toolpath generation based on input parameters or sensor data.
- Real-time monitoring and adjustment of machining parameters for optimal performance.
- Integration with external sensors or feedback mechanisms for closed-loop control.
- Seamless integration with higher-level software systems for process automation and optimisation.

By harnessing the power of Python scripting, we were able to extend the capabilities of the CNC machine beyond traditional machining tasks, unlocking new avenues for innovation and experimentation in automated manufacturing processes.

Additionally, utilizing Python for control and automation also makes it easy to implement the project using Raspberry Pi. The Raspberry Pi provides a cost-effective and compact platform for running Python scripts, making it ideal for embedded applications. By integrating Python with Raspberry Pi, we were able

to create a standalone system capable of autonomous CNC operation, further enhancing the project's accessibility and scalability.

Figure 4: Raspberry Pi 4 Module

4. INTRODUCTION TO MACHINE LEARNING IN CNC MACHINES

In our project, the Vision-based CNC Pick-and-Place (PnP) machine integrates machine learning algorithms to drive intelligent decision-making, advancing beyond traditional applications to enhance manufacturing processes. By harnessing the power of machine learning, our system transcends mere gaming applications and demonstrates its potential as a versatile tool for automation and optimization in various industries.

At the heart of our application lies the utilization of machine learning algorithms, particularly the Minimax algorithm, to implement not just Tic-Tac-Toe, but a broader range of applications, including a game we've named Pyconnect. Tic-Tac-Toe, a renowned paper-and-pencil game, serves as a captivating example to showcase the capabilities of our Vision-based CNC PnP machine.

4.1. Application – TicTacToe

In our rendition of Tic-Tac-Toe, players engage in strategic maneuvers on a 3x3 grid, aiming to align three marks either horizontally, vertically, or diagonally. While seemingly simple, Tic-Tac-Toe embodies the essence of strategic decision-making, teaching valuable lessons in sportsmanship and inviting exploration into the realm of artificial intelligence.

The game unfolds as players strategically place their marks on the grid, with each move branching out into a myriad of possible outcomes. Here, the Minimax algorithm steps in, analyzing the game state and traversing through the intricate branches of the game tree to determine the optimal move. By evaluating all possible outcomes, the machine learns to make informed decisions, adapting its strategy to outmaneuver its opponent.

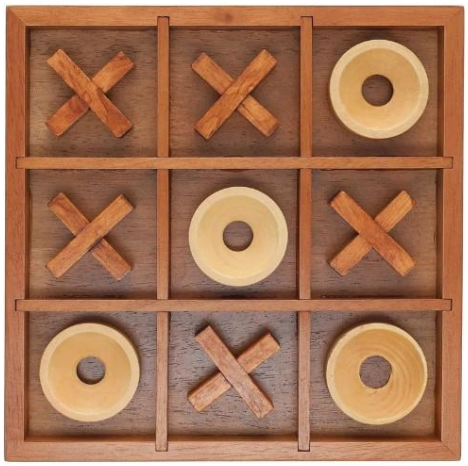


Figure 5: TicTacToe Game Board

4.2. Machine Learning Algorithm

An algorithm for Tic Tac Toe typically involves determining the best move for a given player based on the current state of the game.

- **Representation of the Game State:** The first step is to represent the current state of the game. This can be done using a 3x3 grid or an array where each cell contains either 'X', 'O', or is empty.
- **Evaluation Function:** The algorithm needs a way to evaluate the current state of the game and determine how good it is for the player. This function assigns a score to each possible game state. A state where the player has won would have a high score, a state where the opponent has won would have a low score, and a state where the game is tied might have a neutral score.
- **Minimax Algorithm:** The minimax algorithm is a recursive algorithm used to choose the best move for the current player, if the opponent also plays optimally. It works by exploring all possible future moves up to a certain depth and selecting the move that leads to the best outcome for the current player.

- **Choosing the Best Move:** Once the minimax algorithm has been applied to all possible moves, the algorithm selects the move with the highest score if it is the algorithm's turn.

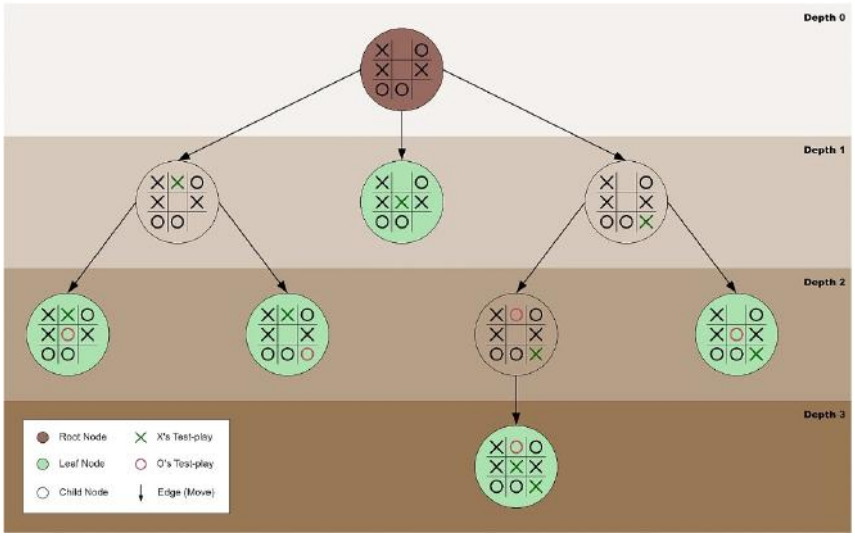


Figure 6: Game tree

4.3. Python code for Minimax Algorithm

```
def minimax(state, depth, player):
    if player == COMP:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]
    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]
    for cell in empty_cells(state):
        x, y = cell[0], cell[1]
        state[x][y] = player
        score = minimax(state, depth - 1, -player)
        state[x][y] = 0
        score[0], score[1] = x, y
        if player == COMP:
            if score[2] > best[2]:
                best = score
        else:
            if score[2] < best[2]:
                best = score
    return best
```

Figure 7: Python code for Minimax Algorithm

The above image is a snippet from our main code which depicts the minimax function

- The Minimax function as you can see takes three arguments
  - **"state"** of the game i.e. the current positions on the 3x3 matrix board.
  - **"depth"** of the search tree, indicating how many steps ahead should the algorithm think for the computer to win or draw the game.
  - **"player" –** The current player (COMP) or the opponent.

```
    if player == COMP:
        best = [-1, -1, -infinity]
    else:
        best = [-1, -1, +infinity]
```

- Here, the variable best is initialized. It is a list containing three elements:
  - The first two elements [-1, -1] represent the coordinates of the best move found so far. They are initialized to -1 because the actual coordinates are yet to be determined.

  - The third element is the score of the best move. If the player is the computer (COMP), the initial score is set to negative infinity, indicating the worst possible score for the computer's perspective. If the player is the opponent, the initial score is set to positive infinity, indicating the worst possible score for the opponent's perspective.

```
    if depth == 0 or game_over(state):
        score = evaluate(state)
        return [-1, -1, score]
```

- The second if statement checks if either the depth limit has been reached (depth equals 0) or if the game is over (based on some "game over"

function). If either condition is true, the function evaluates the current state using some evaluation function (evaluate) and returns a list containing the coordinates [-1, -1] and the evaluated score.

```
for cell in empty_cells(state):
    x, y = cell[0], cell[1]
    state[x][y] = player
```

- This loop iterates over each empty cell on the game board. For each empty cell, it assigns its coordinates to x and y, and then it updates the state by placing the current player's symbol (either COMP or the opponent) in that cell.

```
score = minimax(state, depth - 1, -player)
```

- Within the loop, this line recursively calls the minimax function with the updated state, decremented depth (indicating one step deeper into the search tree), and the opponent's turn (indicated by -player). This step simulates the opponent's move and evaluates its consequences.

```
state[x][y] = 0
```

- After evaluating the potential move, the function resets the cell back to empty to backtrack and try another possible move.

```
score[0], score[1] = x, y
```

- The algorithm assigns the coordinates x and y of the current cell to the first and second elements of the score list, which represent the best move found so far.

```
if player == COMP:
    if score[2] > best[2]:
        best = score
else:
    if score[2] < best[2]:
        best = score
```

- This part updates the best move found so far. If it is the computer's turn, it compares the score of the current move (‘score [2]’) with the best score found so far (‘best [2]’). If the current move's score is greater than the best score, it updates best to the current move. If it is the opponent's turn, it does the opposite: if the current move's score is less than the best score, it updates best to the current move.

```
return best
```

- Finally, the function returns the best move found after considering all possible moves from the current state.

5. INTRODUCTION TO IMAGE PROCESSING

OpenCV served as the primary tool for gathering real-world data, such as identifying object types and their placements, which was then relayed to the microprocessor for computing the system's next move. We utilized image processing techniques to identify shapes and colours of parts within the workspace, facilitating efficient sorting and movement through a pick-and-place mechanism. By calibrating the captured images, OpenCV allowed us to map positions in the workspace to their corresponding image pixels, thus providing precise information on the location of objects within the workspace.

5.1. Basics of image processing

To initiate the image processing aspect of the project, we began by acquainting ourselves with fundamental commands essential for tasks such as reading and writing images, formatting in RGB and grayscale, as well as performing cropping and resizing operations using sample images. To utilize the image processing commands, we leveraged the CV2 library in Python, while opting for Visual Studio as our compiler platform.

For importing CV2 we used the command,

```
import cv2
```

For reading and writing an image we used the command,

cv2.imread(<*image_path*>, <*mode_of_reading*>)

```
cv2.imread("/home/pcblab/Downloads/alpha_image.png",cv2.IMREAD_GRAYSCALE)
```

cv2.imwrite(<*image_name*>, <*image_to_write*>)

```
cv2.imwrite('alpha_image.png',crop)
```

For displaying an image we used the command,

Cv2.imshow(<*output_name*>, <*image_to_display*>)

```
cv2.imshow('circle',o)
```

For reading an image in RGB/ Grayscale we used the command,

```
cv2.IMREAD_COLOR  #for reading in colour

cv2.IMREAD_GRAYSCALE  #for reading in grayscale
```

For resizing an image we used the command,

cv2.resize(<*source_image*>, <*size_in_pixels*>)

```
cv2.resize(image, (960,560))
```

For cropping an image we used the command,

<*source_image*> [<*row_pixels*>, <*column_pixels*>]

```
image[60:480, 220:680]
```

For capturing an image using an external webcam,

cv2.VideoCapture( )

```
cv2.VideoCapture(0)
```

5.2. Shape Detection

5.2.1.Method 1 (Contour Detection):

In our image processing methodology, we employed contour detection as a fundamental technique for shape identification. This method involved analysing the contours of objects within an image to discern their shapes. Specifically, we determined shapes by counting the number of edges present in a particular contour. By leveraging contour detection, we were able to accurately identify and

categorize shapes within our images, laying the groundwork for subsequent processing and analysis tasks.

As part of our methodology, we initiated by reading an image and converting it from the default BGR (Blue-Green-Red) format to grayscale. This conversion process was essential for simplifying subsequent image processing tasks, as grayscale images contain intensity values representing pixel brightness rather than color information.

cv2.cvtColor(<*source_image*>, cv2.COLOR_BGR2GRAY)

```
grey_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
```

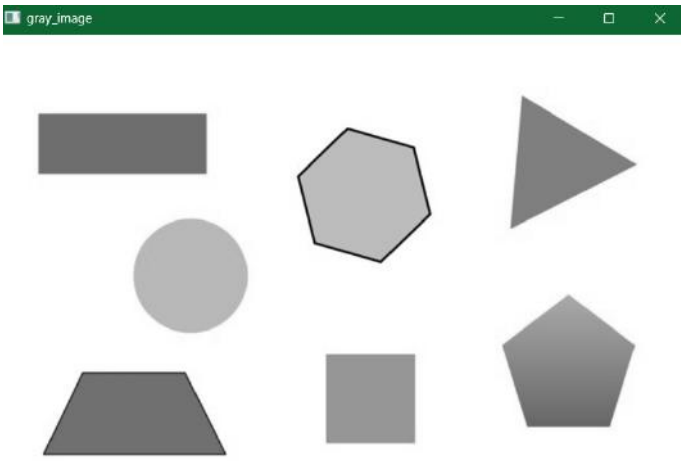


Figure 8: Grayscale image

Following the conversion of the image to grayscale, we proceeded to transform it into a binary image. This involved setting a threshold value, below which all pixel intensities would be assigned a value of black (0), and above which they would be assigned a value of white (255).

cv2.threshold(<*grayscale_image*>, <*threshold_value*>, <*max_grayscale_value*>, cv2.THRESH_BINARY)

```
    _,                          thresh_image                          =
cv2.threshold(grey_image,220,225,cv2.THRESH_BINARY)
```

This command facilitated the conversion process, enabling us to create a binary representation of the image based on the specified threshold value.

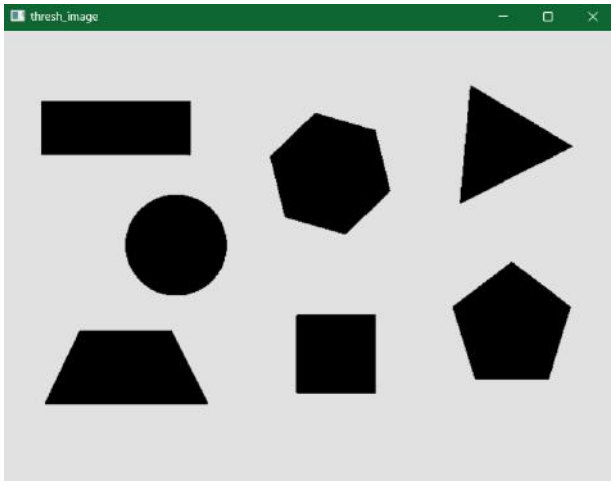


Figure 9: Binary image

Once the image has been converted into a binary representation, we proceeded to identify the contours present within it. This step is crucial for shape detection and analysis.

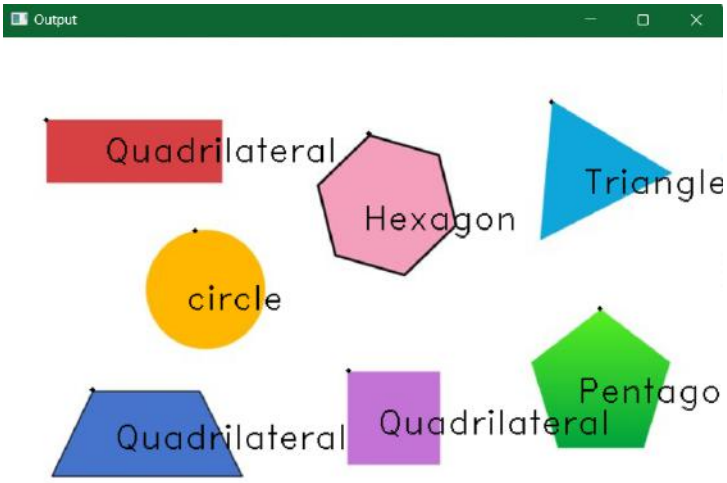
cv2.findContours(<*binary_image*>, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)

```
    contours,  hierarchy  =  cv2.findContours(thresh_image,  cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

This command allowed us to locate and extract the contours present in the binary image, providing us with the necessary information for further processing and analysis. Once contours are detected within the binary image, we proceeded to analyse them to determine the number of edges of each shape. This information

allowed us to distinguish between different shapes based on their unique edge characteristics.

Additionally, we computed the midpoint pixel value for each detected shape from its contour. This enabled us to precisely locate the centre of each shape. Subsequently, we utilized this centre point to display the name of the detected shape, ensuring that the identification process began from the centre and radiated outward.This combined approach facilitated accurate shape recognition



and labelling within our image processing pipeline.

Figure 10: Output image for contour detection

Python Program for the above algorithm:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image = cv2.imread("C:\\Users\\ahile\\OneDrive\\Documents\\Image processing samples\\X_fam.jpg")
grey_image = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
_, thresh_image = cv2.threshold(grey_image,220,225,cv2.THRESH_BINARY)
contours, hierarchy = cv2.findContours(thresh_image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

```
for i, contour in enumerate(contours):
    if i == 0:
        continue

    epsilon = 0.01*cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)

    cv2.drawContours(image, contour, 0, (0,0,0), 4)
    x,y,w,h = cv2.boundingRect(approx)
    x_mid = int(x + w/3)
    y_mid = int(y + h/1.5)

    coords = (x_mid,y_mid)
    color = (0,0,0)
    font = cv2.FONT_HERSHEY_DUPLEX

    if len(approx) == 3:
        cv2.putText(image,"Triangle",coords, font, 1, color, 1)
    elif len(approx) == 4:
        cv2.putText(image,"Quadrilateral",coords, font, 1, color, 1)
    elif len(approx) == 5:
        cv2.putText(image,"Pentagon",coords, font, 1, color, 1)
    elif len(approx) == 6:
        cv2.putText(image,"Hexagon",coords, font, 1, color, 1)
    else:
        cv2.putText(image,"circle",coords, font, 1, color, 1)


cv2.imshow('Output', image)
cv2.imshow('gray_image', grey_image)
cv2.imshow('thresh_image',thresh_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

Challenges encountered:

In our application, which involved a tic-tac-toe game where players drew either 'o' or 'x' during their turns, we encountered challenges with the contour detection method. Specifically, accurately detecting 'x' shapes proved problematic due to irregularities in human-drawn 'x's. These irregularities led to the detection of multiple contours instead of a single coherent 'x' shape, making it difficult for the algorithm to discern the player's intended move.

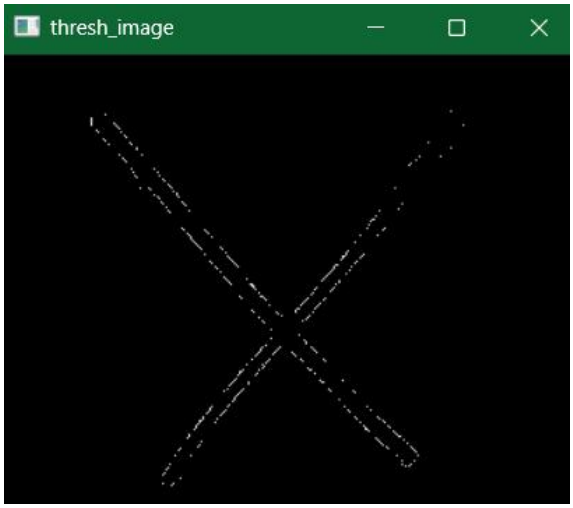


Figure 11: Erroneous image detection

5.2.2.Method 2 (Canny edge detection):

In this alternative method, we rely on the Canny edge detection algorithm to identify edges within the image. Unlike contour detection, which directly identifies contours, this approach first detects edges and then determines shapes based on the edge information.

Similar to the previous method, we start by reading the image and converting it from BGR (Blue-Green-Red) format to grayscale. This grayscale conversion simplifies the subsequent edge detection process by focusing solely on intensity variations in the image.

cv2.cvtColor(<*source_image*>, cv2.COLOR_BGR2GRAY)

```
grey_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

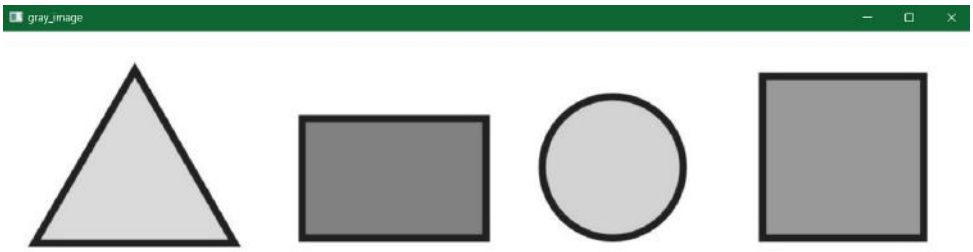


Figure 12: Grayscale image for canny edge detection

By employing the Canny edge detection algorithm on the grayscale image, we obtain a binary image highlighting the edges present in the scene. We then proceed to analyse these edges to determine the shapes based on the number of edges detected.

cv2.canny(<*grayscale_image*>, <*lower_threshold*>, <*upper_threshold*>)

```
edges = cv2.Canny(grey_image, 30, 200)
```

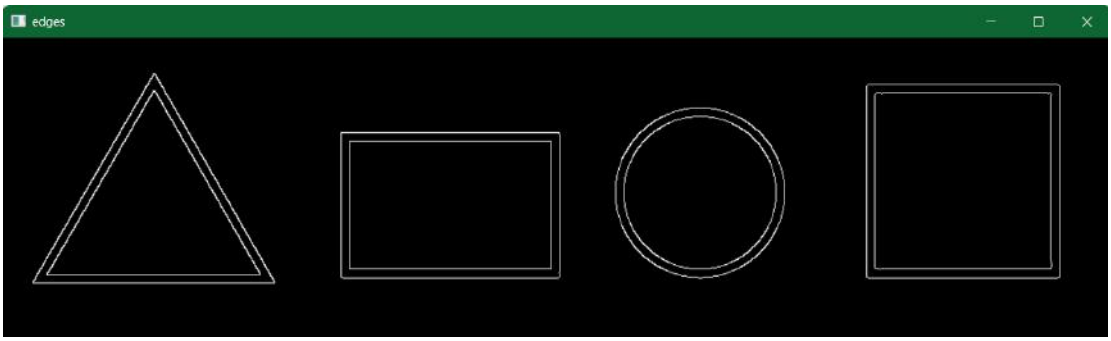


Figure 13: Edge detection using canny method

Following the application of the Canny edge detection algorithm, we proceed to identify contours within the processed image. This step is crucial for shape identification and analysis.

cv2.findContours(*<edges_found>*, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

```
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Similar to the previous method, we utilize the 'findCountours' function to locate and extract contours from the binary image generated by the Canny edge detection algorithm. These contours represent the outlines of shapes present in the image.

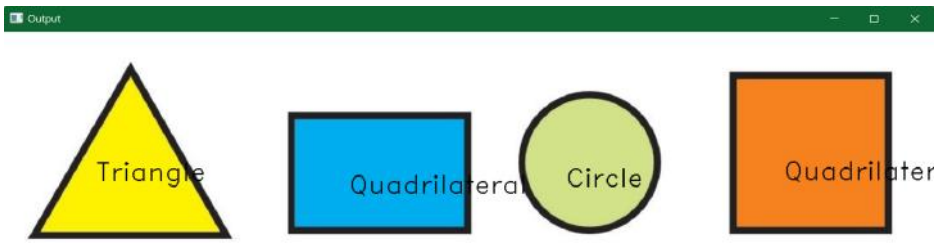


Figure 14: Output using canny method

By combining edge detection with contour analysis, we aim to overcome the limitations faced in the previous method, particularly in scenarios where irregularly drawn shapes pose challenges for contour detection.

Python Program for the above algorithm:

```
import cv2
import numpy as np
```

```
image = cv2.imread("C:\\Users\\ahile\\OneDrive\\Documents\\Image processing samples\\sample_shapes_12.jpg")
grey_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Step 1: Apply Canny edge detection
edges = cv2.Canny(grey_image, 30, 200)

# Step 2: Process edges directly without contour detection
contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

for contour in contours:
    epsilon = 0.01 * cv2.arcLength(contour, True)
    approx = cv2.approxPolyDP(contour, epsilon, True)

    x, y, w, h = cv2.boundingRect(approx)
    x_mid = int(x + w / 3)
    y_mid = int(y + h / 1.5)

    coords = (x_mid, y_mid)
    color = (0, 0, 0)
    font = cv2.FONT_HERSHEY_DUPLEX

    if len(approx) == 3:
        cv2.putText(image, "Triangle", coords, font, 1, color, 1)
    elif len(approx) == 4:
        cv2.putText(image, "Quadrilateral", coords, font, 1, color, 1)
    elif len(approx) == 5:
        cv2.putText(image, "Pentagon", coords, font, 1, color, 1)
    elif len(approx) == 6:
        cv2.putText(image, "Hexagon", coords, font, 1, color, 1)
    else:
        cv2.putText(image, "Circle", coords, font, 1, color, 1)

# Displaying the results
cv2.imshow('Output', image)
cv2.imshow('gray_image', grey_image)
cv2.imshow('edges', edges)
```

```
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Challenges encountered:

Despite achieving improved results with the Canny edge detection algorithm compared to contour detection, we continued to encounter errors, particularly in cases of misidentification of shapes and not identifying shapes. These inaccuracies posed challenges, especially considering our goal of implementing a pick-and-place mechanism.

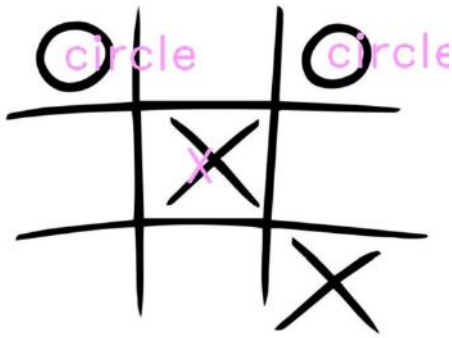


Figure 15: Erroneous shape detection using canny method

Recognizing the limitations of the edge detection methods, we explored an alternate approach known as template matching. By transitioning to the template matching method, we aimed to enhance the accuracy and reliability of shape detection, thus laying a stronger foundation for the subsequent stages of our project, including the implementation of the pick-and-place mechanism.

5.2.3.Method 3 (Template Matching):

Template matching is a powerful technique used to locate areas within an image that closely resemble a predefined template or patch. A template, in this context, refers to a small image containing distinctive features that we aim to identify within a larger source image.

The process of template matching involves comparing the template image against regions of the source image to find occurrences that closely match the template. The goal is to locate regions in the source image where the template appears, even if it's at a different scale or orientation. By analysing similarities between the template and different regions of the source image, template matching helps identify where the template appears, aiding in tasks such as object detection, recognition, and localization.

Shape Detection using Template Matching:

In our application for shape detection within our game board, we employed template matching by utilizing template images of the coins used in our game. These template images included representations of both circles and triangles, corresponding to the shapes present on the game board.

For each shape, multiple template images were provided, capturing variations in position and orientation. This comprehensive approach allowed us to account for different configurations of the shapes within the game board. By feeding these template images into the template matching algorithm, we aimed to accurately detect the presence and location of circles and triangles on the game board. The image captured using our external webcam, depicted below, serves as the source image for template matching.

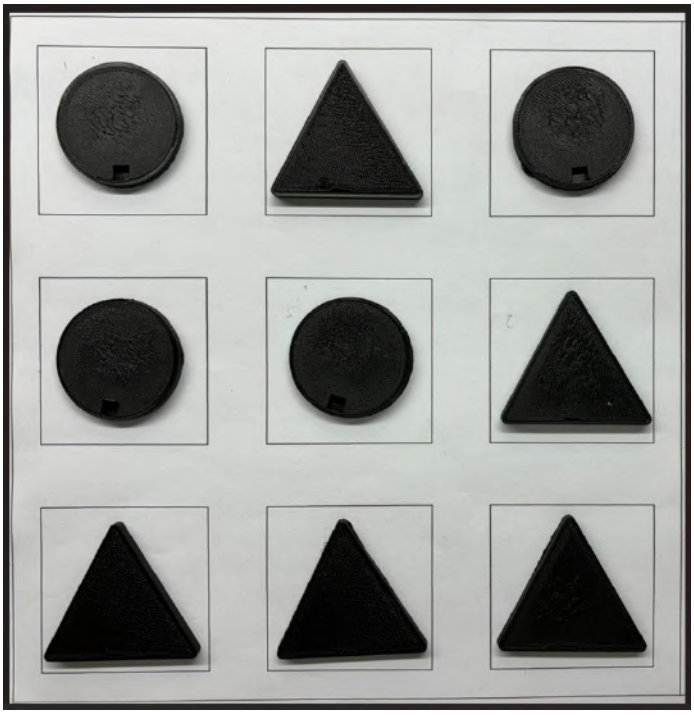


Figure 16: Input board image for template matching method

The template images provided as references, shown below, include representations of shapes such as circles and triangles in the form of coins.

Figure 17: Triangular coin reference image

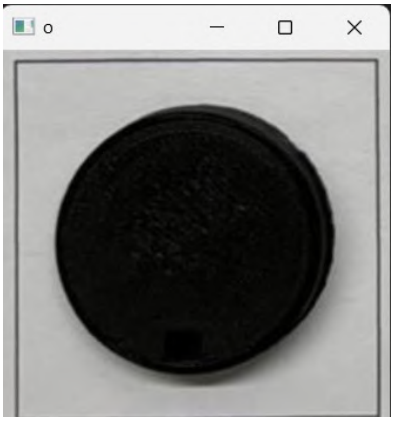


Figure 18: Circular coin reference image

The match template function operates by comparing the reference image with the source image across all points of the source image. Upon comparison, the function returns an output indicating the location where the template has the highest probability of presence. This output is represented as a black and white image, highlighting the regions with the maximum likelihood of containing the template and providing its position.

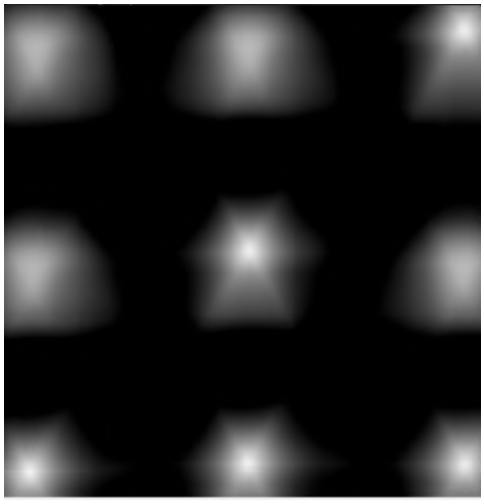


Figure 19: Result of template matching for circular coins

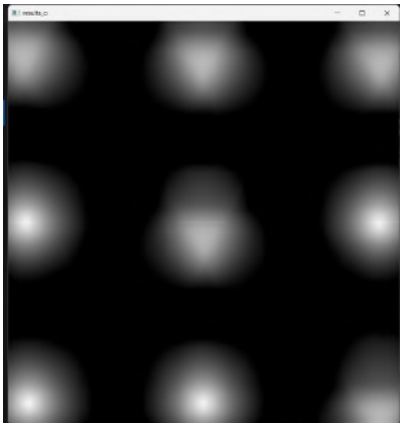


Figure 20: Result of template matching for triangular coins

Utilizing this probability distribution, we can discern accurately detect the shapes of the given objects within a source image. By analysing the regions with higher probability scores, we can reliably identify the locations of the shapes, facilitating efficient shape recognition.

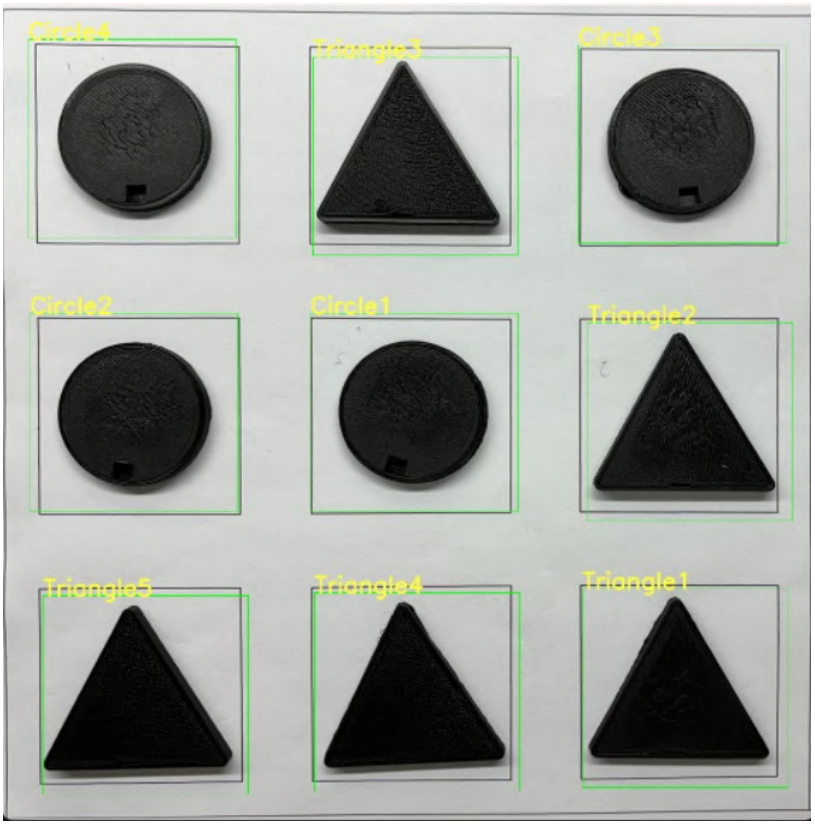


Figure 21: Output image using template matching

Python Program for template matching method:

```
import cv2
import numpy as np

xo_table    =    cv2.imread("C:\\Users\\91883\\OneDrive\\Desktop\\python
progs\\noqhie1.jpg",cv2.IMREAD_UNCHANGED)
ims = cv2.resize(xo_table, (960, 540))
x        =        cv2.imread("C:\\Users\\91883\\OneDrive\\Desktop\\python
progs\\black_triangle_5.jpg",cv2.IMREAD_UNCHANGED)

o        =        cv2.imread("C:\\Users\\91883\\OneDrive\\Desktop\\python
progs\\black_circle_5.jpg",cv2.IMREAD_UNCHANGED)

cv2.imshow('XO_table',ims)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imshow('x',x)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imshow('o',o)
cv2.waitKey()
cv2.destroyAllWindows()

font = cv2.FONT_HERSHEY_SIMPLEX
fontScale = 1

color = (60, 255, 255)
thickness = 2

result_x = cv2.matchTemplate(xo_table, x, cv2.TM_CCOEFF_NORMED)
result_o = cv2.matchTemplate(xo_table, o, cv2.TM_CCOEFF_NORMED)

cv2.imshow('results_x',result_x)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imshow('results_o',result_o)
cv2.waitKey()
```

```
cv2.destroyAllWindows()
wx = x.shape[1]
hx = x.shape[0]
wo = o.shape[1]
ho = o.shape[0]
threshold = 0.75
i = 1
max_val = 1
prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = None, None,
None, None
while max_val > threshold:
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result_x)

      if prev_min_val == min_val and prev_max_val == max_val and
prev_min_loc == min_loc and prev_max_loc == max_loc:
        break
    else:
       prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = min_val,
max_val, min_loc, max_loc

    if max_val > threshold:

        start_row = max(0, max_loc[1] - hx // 2)
        start_col = max(0, max_loc[0] - wx // 2)
        end_row = min(result_x.shape[0], max_loc[1] + hx // 2 + 1)
        end_col = min(result_x.shape[1], max_loc[0] + wx // 2 + 1)

        result_x[start_row: end_row, start_col: end_col] = 0
            xo_table = cv2.rectangle(xo_table,(max_loc[0],max_loc[1]),
(max_loc[0]+wx+1, max_loc[1]+hx+1), (0,255,0) )
         xo_table = cv2.putText(xo_table, 'Triangle' + str(i), max_loc,
font, fontScale, color, thickness, cv2.LINE_AA)


        i += 1
i = 1
max_val = 1
```

```
prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = None, None, None, None
while max_val > threshold:
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result_o)


      if prev_min_val == min_val and prev_max_val == max_val and prev_min_loc == min_loc and prev_max_loc == max_loc:
        break
    else:
       prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = min_val, max_val, min_loc, max_loc

    if max_val > threshold:
        start_row = max(0, max_loc[1] - ho // 2)
        start_col = max(0, max_loc[0] - wo // 2)
        end_row = min(result_o.shape[0], max_loc[1] + ho // 2 + 1)
        end_col = min(result_o.shape[1], max_loc[0] + wo // 2 + 1)

        result_o[start_row: end_row, start_col: end_col] = 0
            xo_table = cv2.rectangle(xo_table,(max_loc[0],max_loc[1]), (max_loc[0]+wo+1, max_loc[1]+ho+1), (0,255,0) )
       xo_table = cv2.putText(xo_table, 'Circle' + str(i), max_loc, font, fontScale, color, thickness, cv2.LINE_AA)

        i += 1

xo_table=cv2.resize(xo_table,(540,540))
cv2.imshow('XO_table',xo_table)
cv2.waitKey()
cv2.destroyAllWindows()
```

Position Detection using Template Matching:

Through template matching, we extract the position of the object, enabling us to determine its placement on the game board. This positional information is then utilized to relay data to the system, informing it of the current state of the game board and aiding in the planning of the next move. The location data is represented in pixel coordinates, facilitating precise coordination and subsequent actions.

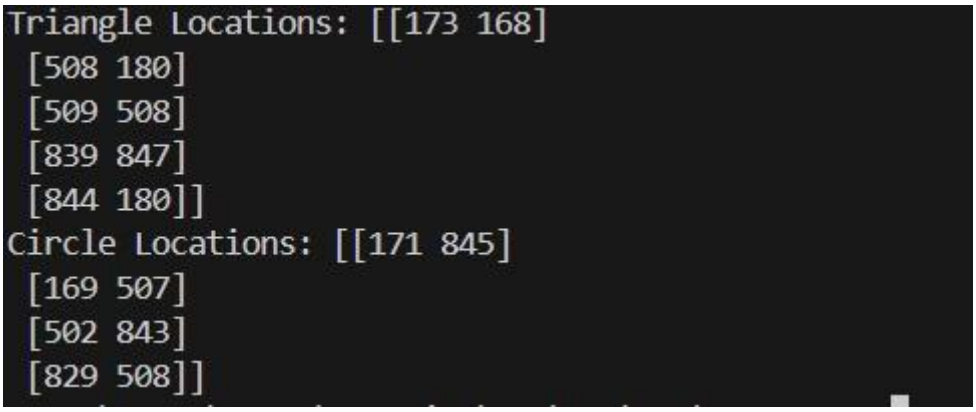


Figure 22: Centre locations of circular and triangular coins

Python Program for Position Detection using Template Matching:

```
import cv2
import numpy as np
triangle_locations = []
circle_locations = []
xo_table    =    cv2.imread("C:\\Users\\91883\\OneDrive\\Desktop\\python
progs\\noqhie1.jpg",cv2.IMREAD_UNCHANGED)
ims = cv2.resize(xo_table, (960, 540))
x       =       cv2.imread("C:\\Users\\91883\\OneDrive\\Desktop\\python
progs\\black_triangle_5.jpg",cv2.IMREAD_UNCHANGED)
o       =       cv2.imread("C:\\Users\\91883\\OneDrive\\Desktop\\python
progs\\black_circle_5.jpg",cv2.IMREAD_UNCHANGED)

cv2.imshow('XO_table',ims)
cv2.waitKey()
cv2.destroyAllWindows()
```

```
cv2.imshow('x',x)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imshow('o',o)
cv2.waitKey()
cv2.destroyAllWindows()

font = cv2.FONT_HERSHEY_SIMPLEX
fontScale = 1

color = (60, 255, 255)
thickness = 2

result_x = cv2.matchTemplate(xo_table, x, cv2.TM_CCOEFF_NORMED)
result_o = cv2.matchTemplate(xo_table, o, cv2.TM_CCOEFF_NORMED)



cv2.imshow('results_x',result_x)
cv2.waitKey()
cv2.destroyAllWindows()
cv2.imshow('results_o',result_o)
cv2.waitKey()
cv2.destroyAllWindows()

wx = x.shape[1]
hx = x.shape[0]
wo = o.shape[1]
ho = o.shape[0]

threshold = 0.75

i = 1
max_val = 1
prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = None, None,
None, None
while max_val > threshold:
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result_x)
```

```
      if  prev_min_val  ==  min_val  and  prev_max_val  ==  max_val  and
prev_min_loc == min_loc and prev_max_loc == max_loc:
        break
    else:
       prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = min_val,
max_val, min_loc, max_loc

    if max_val > threshold:
        start_row = max(0, max_loc[1] - hx // 2)
        start_col = max(0, max_loc[0] - wx // 2)
        end_row = min(result_x.shape[0], max_loc[1] + hx // 2 + 1)
        end_col = min(result_x.shape[1], max_loc[0] + wx // 2 + 1)

        result_x[start_row: end_row, start_col: end_col] = 0
            xo_table  =  cv2.rectangle(xo_table,(max_loc[0],max_loc[1]),
(max_loc[0]+wx+1, max_loc[1]+hx+1), (0,255,0) )
         xo_table = cv2.putText(xo_table, 'Triangle' + str(i), max_loc,
font, fontScale, color, thickness, cv2.LINE_AA)

        triangle_locations.append((max_loc[0] + wo // 2, max_loc[1] + ho
// 2))
        i += 1


i = 1
max_val = 1
prev_min_val,  prev_max_val,  prev_min_loc,  prev_max_loc  =  None,  None,
None, None
while max_val > threshold:
    min_val, max_val, min_loc, max_loc = cv2.minMaxLoc(result_o)

      if  prev_min_val  ==  min_val  and  prev_max_val  ==  max_val  and
prev_min_loc == min_loc and prev_max_loc == max_loc:
        break
    else:
       prev_min_val, prev_max_val, prev_min_loc, prev_max_loc = min_val,
max_val, min_loc, max_loc
```

```
    if max_val > threshold:
        start_row = max(0, max_loc[1] - ho // 2)
        start_col = max(0, max_loc[0] - wo // 2)
        end_row = min(result_o.shape[0], max_loc[1] + ho // 2 + 1)
        end_col = min(result_o.shape[1], max_loc[0] + wo // 2 + 1)

        result_o[start_row: end_row, start_col: end_col] = 0
            xo_table  =  cv2.rectangle(xo_table,(max_loc[0],max_loc[1]),
(max_loc[0]+wo+1, max_loc[1]+ho+1), (0,255,0) )
       xo_table = cv2.putText(xo_table, 'Circle' + str(i), max_loc, font,
fontScale, color, thickness, cv2.LINE_AA)
         circle_locations.append((max_loc[0] + wo // 2, max_loc[1] + ho
// 2))
        i += 1

xo_table=cv2.resize(xo_table,(540,540))
cv2.imshow('XO_table',xo_table)
cv2.waitKey()
cv2.destroyAllWindows()
triangle_locations = np.array(triangle_locations)
circle_locations = np.array(circle_locations)

print("Triangle Locations:", triangle_locations)
print("Circle Locations:", circle_locations)
```

Returning position of coin with respect to board position:

By correlating the returned results with the actual image and the previously obtained centre values, we can calibrate the pixel values. This calibration process enables us to assign numerical values ranging from 1 to 9 to each respective position on the game board. Analysing the pixel range of each board box and utilizing the centre pixel values allows us to accurately assign a unique number to each box. These assigned numbers serve as valuable outputs, facilitating further processing and decision-making within the system.

```
PS C:\Users\91883\OneDrive\Desktop\python progs> &
Triangle Locations: [9 6 2 8 7]
Circle Locations: [5 4 3 1]
PS C:\Users\91883\OneDrive\Desktop\python progs>
```

Figure 23: Detected locations of triangular and circular coins

Recognising coins in multiple orientations:

Given the possibility of multiple orientations for the triangle coin, a single template might not suffice for accurate detection. To address this, we have incorporated multiple reference images, ensured adaptability, and enhanced detection accuracy. These reference images are iteratively processed within a loop, systematically checking for the highest probability position in the source image. This iterative approach enables comprehensive coverage and robust detection, regardless of the orientation of the triangle coin.

```
triangles = [cv2.imread("/home/pcblab/Downloads/alpha_triangle.png", cv2.IMREAD_GRAYSCALE),
             cv2.imread("/home/pcblab/Downloads/alpha_triangle1.png", cv2.IMREAD_GRAYSCALE),
             cv2.imread("/home/pcblab/Downloads/alpha_triangle2.png", cv2.IMREAD_GRAYSCALE),
             cv2.imread("/home/pcblab/Downloads/alpha_triangle3.png", cv2.IMREAD_GRAYSCALE)]
```

Returning only the latest move done by the user:

One of the crucial functionalities of the image processing function is to determine the position of the board where the player has made the latest move. This information is vital for guiding the machine learning algorithm in planning its subsequent move. To isolate the latest move as the output from the image processing function, we utilize history variables that store the sequence of recent

moves. Additionally, a temporary variable retains the move history up to the previous iteration.

```
triangle_locations = []
circle_locations = []
hist_T = []
hist_C = []
```

By comparing these arrays, we can identify the number corresponding to the latest move, which is then provided as the output.

```
if (mid_pt[0] > 310 and mid_pt[0] < 410) and (mid_pt[1] > 290 and mid_pt[1] < 395):
    if 1 not in triangle_locations:
        if 1 not in hist_T:
            triangle_locations.append(1)
            hist_T.append(1)
```

5.3. Hardware used:

External Webcam:

We employed an external Logitech camera with a resolution of 1080 pixels, connected to a Raspberry Pi, to capture images of the current state of the game board.

Figure 24: Logitech Webcam

Python program used to capture images using a webcam:

```
def board_capture():
    cam_port = 0
    cam = cv2.VideoCapture(cam_port)
    result, image = cam.read()
    image = cv2.resize(image, (960,560))
    crop = image[60:480, 220:680]
    cv2.imwrite('alpha_image.png',crop)
    return crop
```

6. INTEGRATION OF MACHINE LEARNING AND IMAGE PROCESSING WITH G-CODE

6.1. Using Machine Learning Algorithm with G-code

After completing the machine learning algorithm, we proceeded to integrate it with G-code to make the machine operate according to the application. Initially, we used a pencil as the tool head. We designed a dynamic holder for it and used it to draw movements on paper. This worked perfectly. When run, the CNC machine initializes and then draws the game grid on the paper placed on the base. The player makes their move using the Numpad, and then the machine draws the player's move on the paper using the appropriate G-code for the player's symbol. For the machine's move, it uses the ML algorithm to find the next best move and then autonomously draws its symbol.

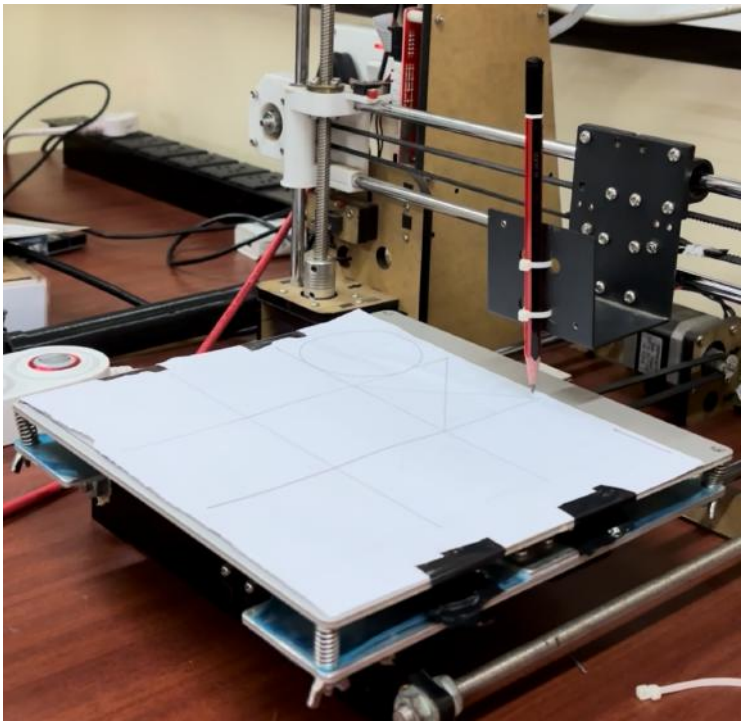


Figure 25: Initial phase of the application using Pencil as tool head

Figure 26: Holder for Pen/Pencil

6.2. Integrating Computer Vision for Real-Time Perception

In phase two we designed a dynamic tool head in order to provide pick and place mechanism to the machine. We used an electromagnet and designed a holder for it. Instead of drawing on a paper, we designed two different game pieces of different shapes which can be picked up by the dynamic tool head. Now instead of drawing X and O on a paper, the machine will pick and place the pieces according to the moves.

The integration of computer vision techniques introduced a new dimension to the project's autonomy. Initially, the CNC machine interacted with the game board via manual input or pre-programmed sequences. With the integration of computer vision, the machine gained the ability to perceive and interpret the game board in real time.

A webcam was employed to capture images of the game board, which were processed using computer vision algorithms explained in previous sections. These algorithms analyzed the images to identify the positions of human moves. The detected moves were then passed to the Minimax algorithm for decision-

making. And then once the human move which is made manually is detected, the machine makes its move by picking its shape and placing it the next best move.

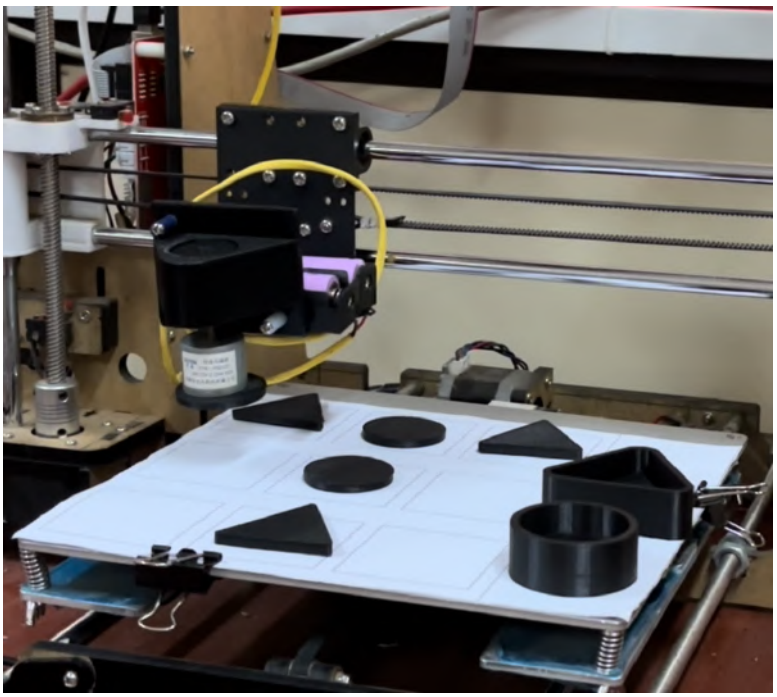


Figure 27: Second phase of the application using Pick n Place mechanism

Figure 28: Dynamic Electromagnet tool head for Pick n Place

6.3. Python Scripting for Seamless Integration

Python served as the glue that integrated machine learning, computer vision, and G-code control. Python scripts were developed to orchestrate the interaction between these components seamlessly. Upon receiving input from the computer vision system, the Python script processed the data and invoked the Minimax algorithm to determine the CNC machine's response.

The output of the Minimax algorithm was translated into G-code commands, specifying the precise movements required to place the CNC machine's game piece on the physical board. These G-code commands were then transmitted to the CNC machine, enabling it to execute its move autonomously.

This integration of advanced technologies demonstrated the potential of synergistic approaches in solving complex real-world problems, showcasing the convergence of artificial intelligence, computer vision, and CNC machining in the realm of autonomous systems.

The Git Repository for the Main code, Image processing code and the G-code is - https://github.com/Dhirajzen/FYP

7. HARDWARE DESIGN PROCESS

7.1. Designing Game Pieces and Game Base

The design of the game pieces and game base was a crucial aspect of our project, as they formed the physical interface for playing tic-tac-toe with the CNC machine. We opted to design game pieces in two different shapes - circles and triangles - to add variety and visual interest to the gameplay. These pieces were designed using computer-aided design (CAD) software, ensuring precise dimensions and compatibility with the dynamic tool head.

Figure 29: Design of Game piece embedded with coins to make it magnetic

The game base, on the other hand, served as the foundation for the tic-tac-toe board and provided a stable surface for the game pieces to be placed upon. It was designed to accommodate the dimensions of the game board and provide sufficient space for the CNC machine to maneuver and interact with the game pieces.

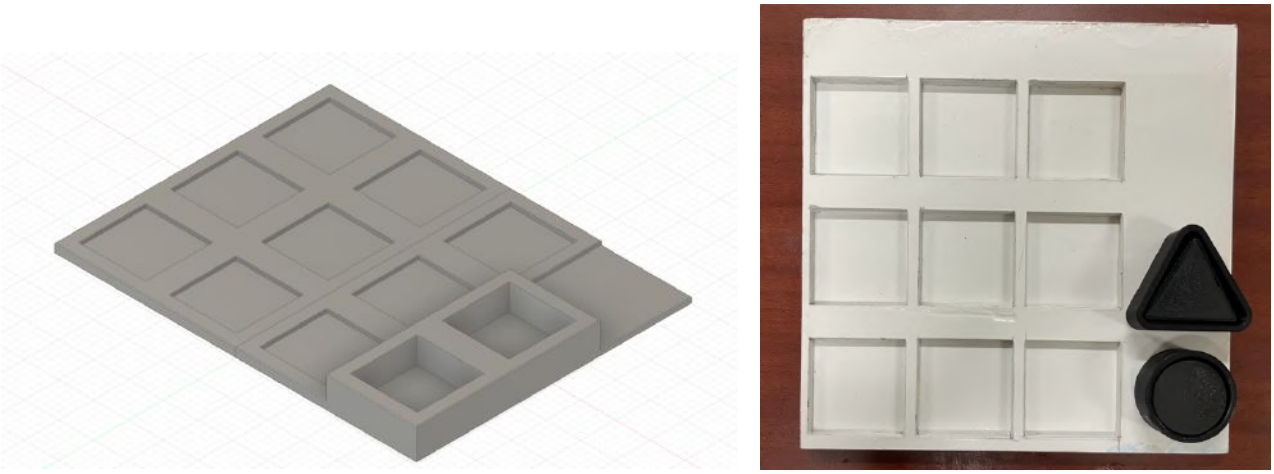


Figure 30: Design of game Base

7.2. Creating Dynamic Tool Head with Electromagnet

The dynamic tool head was a key component of our CNC pick-and-place machine, enabling it to pick up and place game pieces on the board. We opted to use an electromagnet as the gripping mechanism due to its simplicity and reliability. The electromagnet was controlled using GPIO pins on the Raspberry Pi, allowing for precise activation and deactivation as needed.

The design of the dynamic tool head involved careful consideration of factors such as weight, size, and compatibility with the CNC machine. We utilized CAD software to design a custom holder for the electromagnet, ensuring a secure fit and optimal positioning for picking up game pieces. Additionally, we incorporated features such as adjustable height and gripping force to accommodate different game piece sizes and weights.

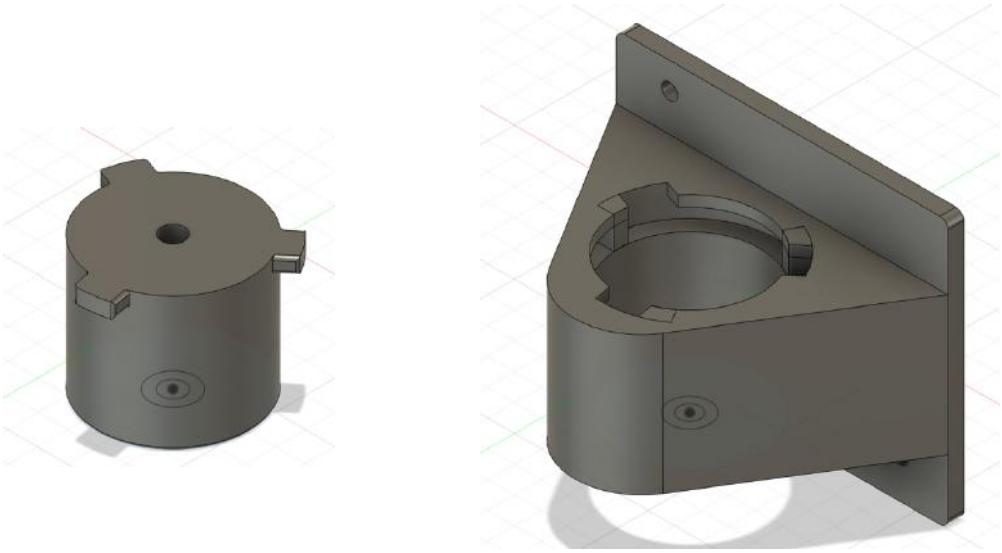


Figure 31: Holder design for the Electromagnet tool head shown in Figure 11.

7.3. Casing for CNC Machine using Foam Sheets

To provide protection and aesthetic appeal to the CNC machine, we designed a casing using foam sheets. Foam sheets offered a lightweight and easy-to-work-with material for creating custom enclosures. The design process involved measuring the dimensions of the CNC machine and designing individual panels for the casing using CAD software.

Once the panel designs were finalized, the foam sheets were cut to size using a precision cutting tool such as a laser cutter or CNC router. The individual panels were then assembled and secured using adhesive or fasteners, creating a seamless and visually appealing casing for the CNC machine.

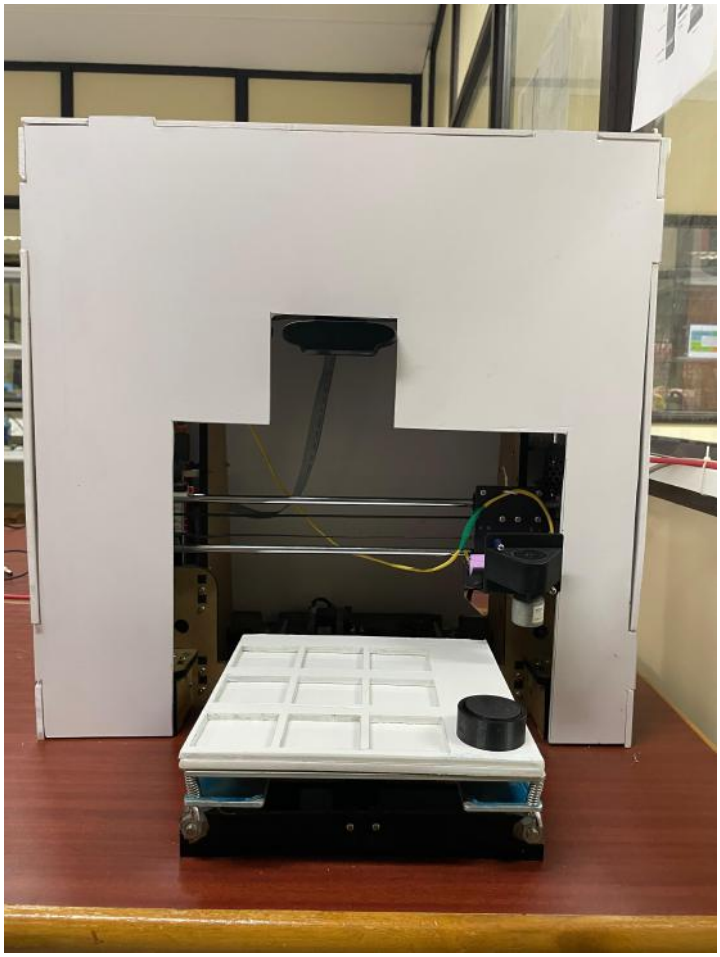


Figure 32: Overall look of the machine with casing

7.4. Testing and Iteration

Throughout the design process, extensive testing and iteration were conducted to ensure the functionality, reliability, and durability of the hardware components. Prototypes were fabricated and subjected to rigorous testing under simulated operating conditions to identify any design flaws or performance issues. Feedback from testing was used to refine and improve the designs, resulting in final iterations that met the project's requirements and specifications.

8. EXPLORING VARIOUS APPLICATIONS

While Tic-Tac-Toe serves as a captivating demonstration of our machine's capabilities, its application extends far beyond the realm of gaming. By integrating machine learning into CNC machining, our Vision-based PnP machine emerges as a powerful tool for automation, precision engineering, and intelligent manufacturing. With the ability to analyze complex scenarios and make strategic decisions autonomously, our machine paves the way for innovation across a multitude of industries.

8.1. Automation of Blood Sample Analysis

One such compelling application we explored is the automation of blood sample analysis using a microscope integrated with the CNC machine. In this innovative approach, we automated the movement of the microscope to navigate through different blood samples, capturing images for subsequent analysis. By seamlessly integrating image processing algorithms, we enabled automated analysis of blood samples, revolutionizing the diagnostic process.

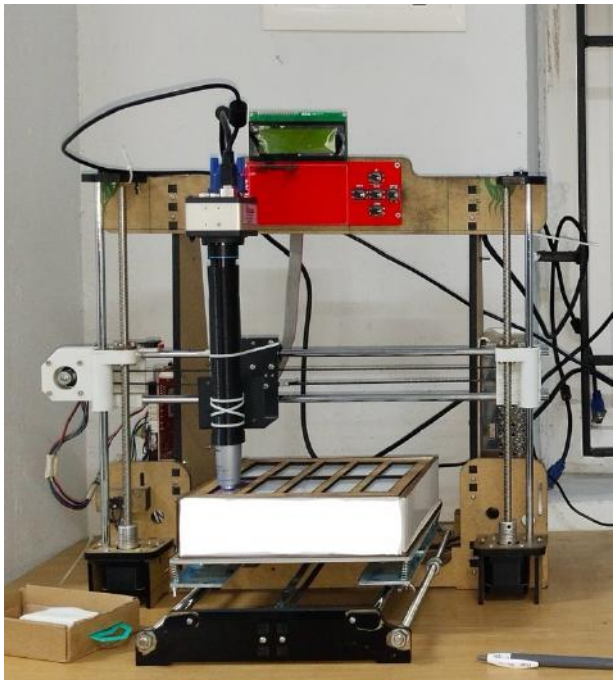


Figure 33: Setup for automating blood sample analysis

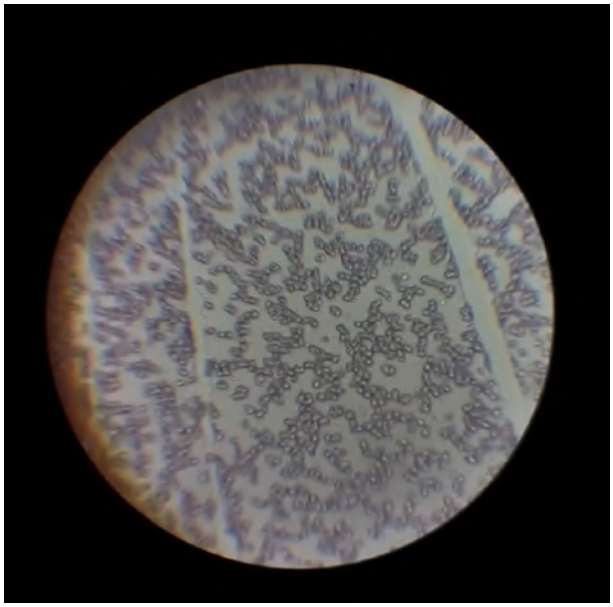


Figure 34: RBC image captured by the microscope

8.2. Enhancing Efficiency and Accuracy

By automating the movement of the microscope, our system eliminates manual intervention, reducing the risk of errors and enhancing the efficiency of the analysis process. Moreover, the integration of QR codes on blood sample slides enables automatic retrieval of patient data, streamlining the workflow and ensuring accuracy in record-keeping.

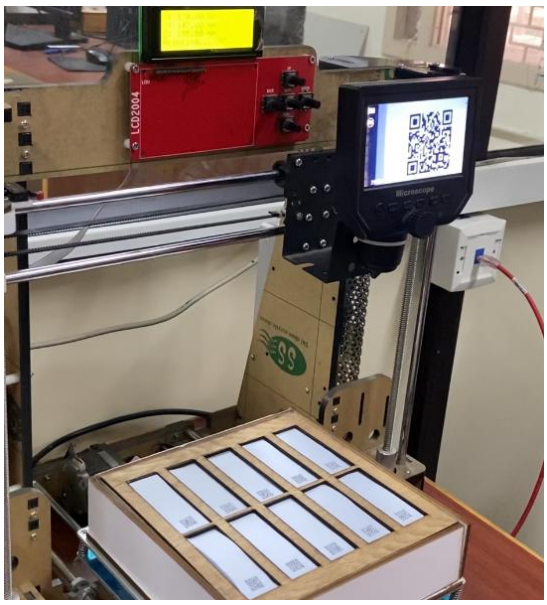


Figure 35: Usage of QR code to retrieve patient data

8.3. Empowering Healthcare Professionals

This application not only enhances the speed and accuracy of blood sample analysis but also empowers healthcare professionals by providing them with timely and reliable diagnostic insights. By leveraging the capabilities of our Vision-based CNC PnP machine, healthcare facilities can optimize their diagnostic workflows, improving patient outcomes and fostering innovation in the field of medical diagnostics.

8.4. Exploring Endless Possibilities

Beyond blood sample analysis, our system opens doors to a wide range of applications in diverse industries. From automated quality control in manufacturing to precision agriculture and beyond, the potential applications of our Vision-based CNC PnP machine are limitless. By providing a versatile platform for automation and innovation, our system catalyzes progress and drives positive change across various domains.

8.5. Conclusion

In conclusion, our Vision-based CNC PnP machine transcends traditional boundaries, offering a platform for innovation and exploration in diverse applications. From gaming and precision engineering to healthcare and beyond, the adaptability and versatility of our system empower us to tackle real-world challenges and drive meaningful progress. As we continue to explore new avenues and push the boundaries of what's possible, we pave the way for a future defined by innovation, collaboration, and endless possibilities.

9. REFERENCES

- W. Cai, T. Xiong and Z. Yin, "Vision-based kinematic calibration of a 4-DOF pick-and-place robot," 2012 IEEE International Conference on Mechatronics and Automation, Chengdu, China, 2012, pp. 87-91, doi: 10.1109/ICMA.2012.6282812
- K. Kim, J. Kim, S. Kang, J. Kim and J. Lee, "Vision-based bin picking system for industrial robotics applications," 2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), Daejeon, Korea (South), 2012, pp. 515-516, doi: 10.1109/URAI.2012.6463057
- L. F. A. Canales, D. M. Hernandez and F. Núñez, "A Model-Based Low-Cost Autonomous Pick-and-Place Cartesian Robot," 2023 IEEE Central America and Panama Student Conference (CONESCAPAN), Guatemala, Guatemala, 2023, pp. 128-133, doi: 10.1109/CONESCAPAN60431.2023.10328435
- M. U. Anjum, U. S. Khan, W. S. Qureshi, A. Hamza and W. A. Khan, "Vision-Based Hybrid Detection For Pick And Place Application In Robotic Manipulators," 2023 International Conference on Robotics and Automation in Industry (ICRAI), Peshawar, Pakistan, 2023, pp. 1-5, doi: 10.1109/ICRAI57502.2023.10089602.
- G. -Y. Luo, M. -Y. Cheng and C. -L. Chiang, "Vision-based 3-D object pick-and-place tasks of industrial manipulator," 2017 International Automatic Control Conference (CACS), Pingtung, Taiwan, 2017, pp. 1-7, doi: 10.1109/CACS.2017.8284250
- I. Rokhim, N. J. Ramadhan and T. Rusdiana, "Image Processing based UR5E Manipulator Robot Control in Pick and Place Application for Random Position and Orientation of Object," 2021 3rd International Symposium on Material and Electrical Engineering Conference (ISMEE), Bandung, Indonesia, 2021, pp. 124-130, doi: 10.1109/ISMEE54273.2021.9774170
- F. Nagata, K. Miki, A. Otsuka, K. Yoshida, K. Watanabe and M. K. Habib, "Pick and Place Robot Using Visual Feedback Control and Transfer Learning-Based CNN," 2020 IEEE International Conference on Mechatronics and Automation (ICMA), Beijing, China, 2020, pp. 850-855, doi: 10.1109/ICMA49215.2020.9233829.
- R. Szabo and R. -S. Ricman, "Building a Tic-tac-toe Playing Robotic Arm," 2022 30th Telecommunications Forum (TELFOR), Belgrade, Serbia, 2022, pp. 1-4, doi: 10.1109/TELFOR56187.2022.9983737