Electronics Club

IIT KANPUR

Doodlebot

SUMMER PROJECT 2025

Project Duration : May 13th 2025 to July 2025

June 13, 2025

Contents

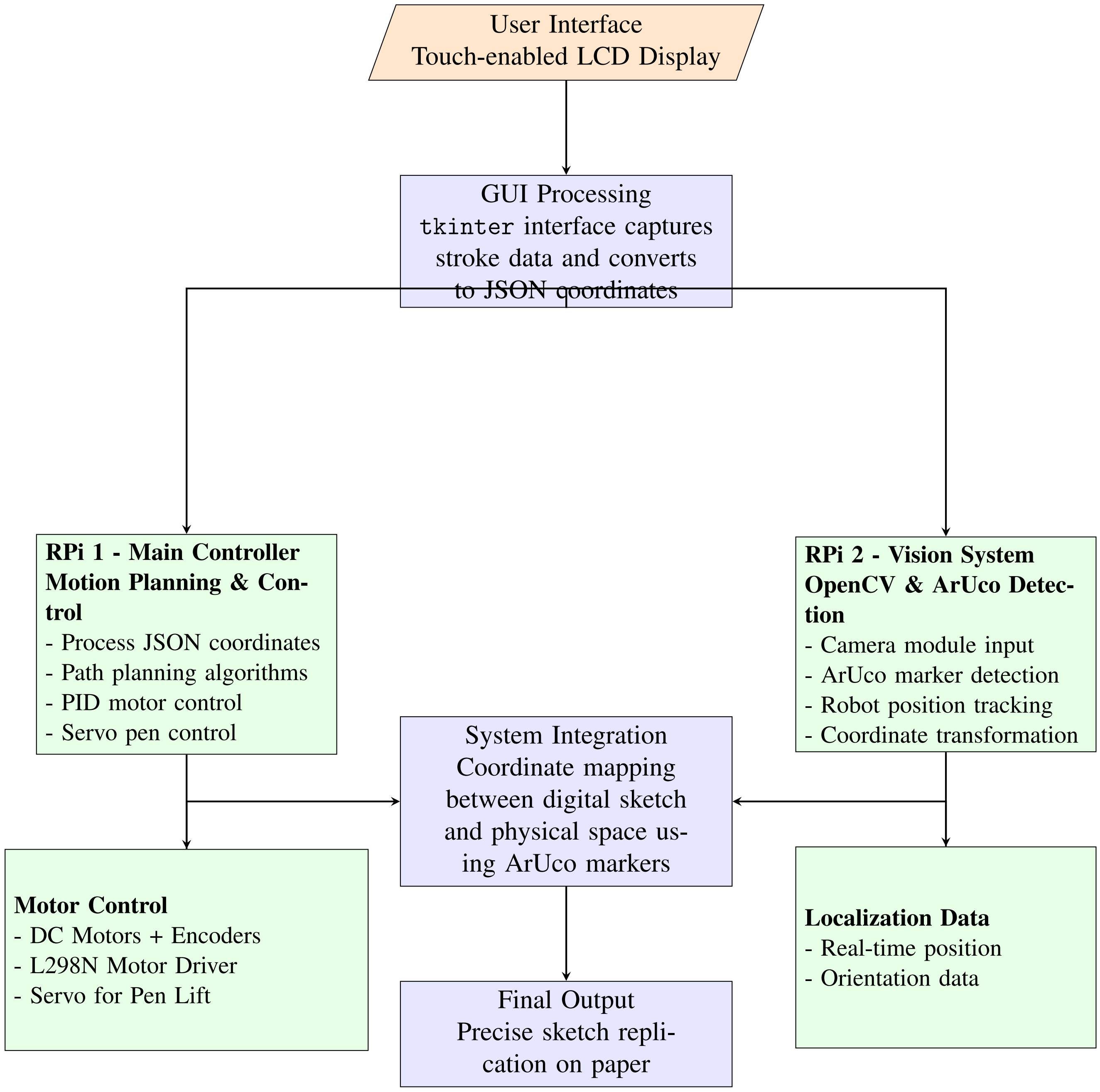
1 Project Description

- DoodleBot is an autonomous robot that draws the sketch drawn by the user on a LCD display.
- The user draws on a display, and the robot captures stroke data as an image input.
- OpenCV algorithm captures the strokes in path coordinates. These coordinates are processed for motion planning.
- A Raspberry Pi controls the 2-WD robot with
  - DC motors+encoder for precise movement.
  - A servo motor for the pen lifting mechanism (up/down movement).
- The robot physically replicates the drawing using a mounted pen on the paper, combining display input, image processing, and motor control for a real-time sketch.

2 Hardware

- Raspberry Pi 4
- 2-WD with DC motors+encoder
- Servo motor
- Motor Driver L298N
- Camera module
- touch-enabled LCD display
- ArUco markers
- Castor Wheel

3 Flow-Chart


User Interface
Touch-enabled LCD Display
GUI Processing
tkinter interface captures stroke data and converts to JSON coordinates
RPi 1 - Main Controller
Motion Planning & Control
- Process JSON coordinates
- Path planning algorithms
- PID motor control
- Servo pen control
RPi 2 - Vision System
OpenCV & ArUco Detection
- Camera module input
- ArUco marker detection
- Robot position tracking
- Coordinate transformation
System Integration
Coordinate mapping between digital sketch and physical space using ArUco markers
Motor Control
- DC Motors + Encoders
- L298N Motor Driver
- Servo for Pen Lift
Localization Data
- Real-time position
- Orientation data
Final Output
Precise sketch replication on paper

4 Week-0

- **Introduction to Micro controllers**
  - We had an introductory lecture to what a microcontroller is, which is basically a small computer on a single integrated circuit that is designed to control specific tasks within electronic systems. We were given a task of-
    - 1. Dual Potentiometer Input:
      - Potentiometer 1 (POT1): Controls three buzzers, each corresponding to a range of analog values:
      - Buzzer 1: Activated when analog input is within the first third of the range (0 to 341).
      - Buzzer 2: Activated when input is in the second third (342 to 682). and so on
      - Potentiometer 2 (POT2): Controls the speed at which a sequence of five LEDs lights up one at a time (e.g., 01000, 00100, 00010, 00001, etc. these the lights at any given time). The analog value from POT2 determines the delay between each LED transition.
- **Computer Vision Path Extraction**
  - **Image Path Extraction and SVG Conversion using Computer Vision**
  - The task was to process a scanned hand-drawn image and convert the prominent inner shape into a scalable vector graphic (SVG) format. This involved basic image preprocessing, boundary detection, and SVG path generation using Python library OpenCV.

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

img = 'image_1.jpg'
image = cv.imread(img)

plt.imshow(image)
plt.show()

gray = cv.cvtColor(image, cv.COLOR_RGB2GRAY)
blur_med = cv.medianBlur(image,7)
A = 20
for i in range(A):
```

Electronics Club
IIT Kanpur

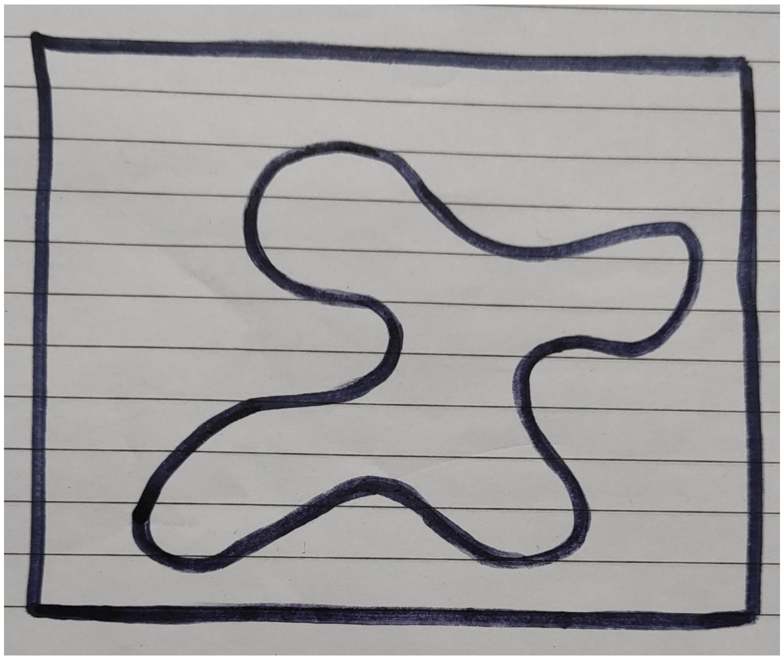


Figure 1: Input image for our OpenCV task

```
    blur_med = cv.medianBlur(blur_med,7)
plt.imshow(blur_med)
plt.show()

gray_1 = cv.cvtColor(blur_med, cv.COLOR_RGB2GRAY)

ret, th1 = cv.threshold(gray_1, 100, 150, cv.THRESH_BINARY) # change threshold
    values as per the marked path

plt.imshow(th1, cmap='gray')
plt.show()

plt.ion()

B = 5 #this loop is to smoothout the curve if there are any rugged lines left

for i in range(B):
    th1 = cv.blur(th1,(3,3),cv.BORDER_DEFAULT)
    ret, th1 = cv.threshold(th1, 100+(2*i), 150-(2*i), cv.THRESH_BINARY)

```

```python
    plt.clf()
    plt.imshow(th1, cmap='gray')
    plt.title(f'Iteration {i+1}')
    plt.pause(.5)

plt.ioff()
plt.show()

N = 135

cropped = th1[N:-N, N:-N]

plt.figure(figsize=(10,5))
plt.subplot(1,2,1), plt.imshow(cv.cvtColor(th1, cv.COLOR_BGR2RGB)),
plt.title("Original")

plt.subplot(1,2,2), plt.imshow(cv.cvtColor(cropped, cv.COLOR_BGR2RGB)),
plt.title("Edges Removed")
plt.show()

contours, hierarchy = cv.findContours(cropped, cv.RETR_CCOMP, cv.CHAIN_APPROX_NONE)

Path = np.zeros_like(cropped)

for i, h in enumerate(hierarchy[0]):
    if h[3] == -1:
        cv.drawContours(Path, contours, i, 255, thickness=1)

plt.imshow(Path, cmap='gray')
plt.axis('off')
plt.show()

selected_contour = None

for i, h in enumerate(hierarchy[0]):
    if h[3] == -1:
        selected_contour = contours[i]
        break

```

```python
points = [(pt[0][0], pt[0][1]) for pt in selected_contour]

import csv

with open('outer_path.csv', 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['x', 'y'])
    writer.writerows(points)

with open('outer_path.svg', 'w') as f:
    f.write('<svg xmlns="http://www.w3.org/2000/svg" version="1.1">\n')
    f.write('<polyline points="')
    f.write(' '.join([f'{x},{y}' for x, y in points]))
    f.write('" style="fill:none;stroke:black;stroke-width:1"/>\n')
    f.write('</svg>')
```

1) Image Input:

- The hand-drawn image was first read using OpenCV's `cv2.imread()` function.

2) Median Blur:

- To reduce noise while preserving edges, a median blur was applied instead of a normal blur. This was executed multiple times in a loop to improve edge retention and noise suppression.

3) Thresholding:

- Binary thresholding was used to extract the prominent boundaries. This made the inner shape appear distinct from the background.

4) Smoothing the Contour:

- Repeated application of thresholding and blurring helped to smooth the extracted boundary curves. This step ensured the extracted path would be clean and continuous.

5) Boundary Cropping:

- The outer rectangle was manually cropped using parameter `N`, which controlled how many pixels from the edges were removed to avoid detecting the outer frame.

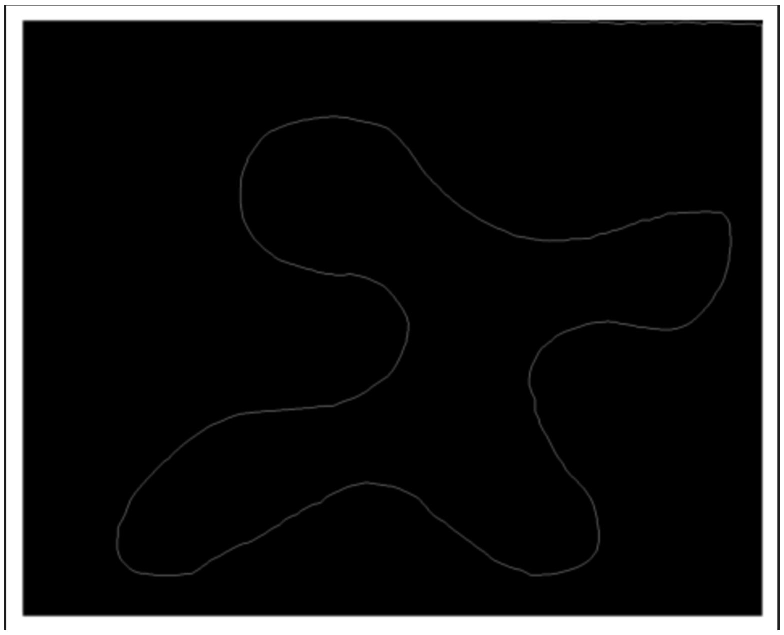


Figure 2: Output of OpenCV task

6) Contour Detection:

- Contours were extracted using OpenCV's `findContours` function. This yielded a list of all curves present in the image.

7) Extracting the Outermost Path:

- Contours with no parent in the hierarchy were identified as outermost. This ensured that the correct path was selected by using the hierarchy information

8) Coordinate Extraction:

- The coordinates of the selected contour were extracted into a list of (x, y) points which were used to generate both CSV and SVG files.

9) SVG Path Generation:

- The final step was to convert the list of coordinates into an SVG path. This was achieved by formatting the points into SVG `path` syntax and saving them using Python's file handling tools

10) Output:

- The output of the process is a smooth SVG path accurately representing the inner curve of the image. The final file `outer_path.svg` can be used for various vector-based applications like laser cutting or digital rendering.

- **PCB designing**
  - We learned how to design a PCB on KiCAD.

5 Week-1

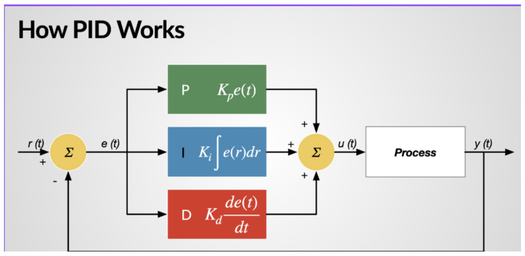


Figure 3: PID controller

- PID controller on SimuLink
  - PID (Proportional-Integral-Derivative) control is a feedback control loop mechanism used in various industrial and automated systems to regulate process variables like temperature, flow, pressure, and speed. It uses a combination of proportional, integral, and derivative actions to maintain the system output close to a desired setpoint

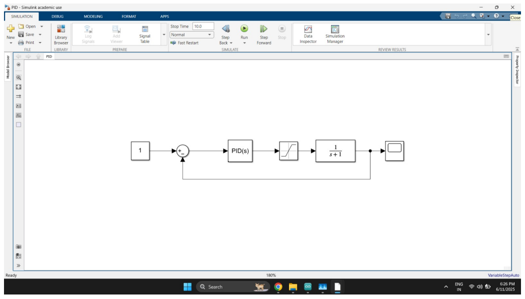


Figure 4: PID controller in Simulink

6 Week-2

- **Designing the Chassis using Fusion360**
  - Fusion 360 being a cloud-based 3D CAD tool by AutoDesk. On Fusion 360, we designed the chassis for 3-D printing.
- **ArUco marker**

- ArUco markers are black-and-white square fiducial markers used for camera pose estimation and object tracking in computer vision applications. Each marker encodes a unique ID within a grid-like pattern, which allows it to be easily detected and distinguished from others using image processing.

```python
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import json

class DrawingApp:
    def init(self, root):
        self.root = root
        self.root.title("Drawing Interface")
        self.root.geometry("650x500")

        # Canvas dimensions
        self.canvas_width = 600
        self.canvas_height = 400

        # Drawing variables
        self.old_x = None
        self.old_y = None
        self.brush_size = 3
        self.brush_color = "black"

        # Store coordinates of drawn lines
        self.line_coordinates = []

        self.setup_ui()

    def setup_ui(self):
        # Main frame
        main_frame = ttk.Frame(self.root, padding="10")
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # Control frame
        control_frame = ttk.Frame(main_frame)
        control_frame.grid(row=0, column=0, columnspan=2,
                           sticky=(tk.W, tk.E), pady=(0, 10))

```

Electronics Club
IIT Kanpur

```python
        # Action buttons — all three buttons are always visible
        ttk.Button(control_frame, text="Clear",
                   command=self.clear_canvas).pack(side=tk.LEFT, padx=(0, 10))
        ttk.Button(control_frame, text="Done",
                   command=self.done_drawing).pack(side=tk.LEFT, padx=(0, 10))
        ttk.Button(control_frame, text="Quit",
                   command=self.quit_app).pack(side=tk.LEFT, padx=(0, 10))

        # Canvas
        self.canvas = tk.Canvas(main_frame, bg="white",
                               width=self.canvas_width,
                               height=self.canvas_height,
                               relief=tk.SUNKEN, borderwidth=2)
        self.canvas.grid(row=1, column=0, columnspan=2, pady=(0, 10))

        # Bind mouse events
        self.canvas.bind("<B1-Motion>", self.paint)
        self.canvas.bind("<ButtonPress-1>", self.start_paint)
        self.canvas.bind("<ButtonRelease-1>", self.stop_paint)

        # Status label
        self.status_label = ttk.Label(main_frame, text="Draw on the canvas...")
        self.status_label.grid(row=2, column=0, columnspan=2, sticky=tk.W)

    def start_paint(self, event):
        self.old_x = event.x
        self.old_y = event.y

    def paint(self, event):
        if self.old_x and self.old_y:
            # Draw on canvas
            self.canvas.create_line(self.old_x, self.old_y, event.x, event.y,
                                  width=self.brush_size, fill=self.brush_color,
                                  capstyle=tk.ROUND, smooth=tk.TRUE)

            # Store line coordinates
            self.line_coordinates.append([self.old_x, self.old_y,
                                         event.x, event.y])

        self.old_x = event.x
        self.old_y = event.y

    def stop_paint(self, event):
        self.old_x = None
        self.old_y = None

    def clear_canvas(self):
        self.canvas.delete("all")
        self.line_coordinates = []
        self.status_label.config(text="Canvas cleared — ready for new drawing")

    def done_drawing(self):
        # Save coordinates to JSON file without removing the Done button
        if self.line_coordinates:
            # Save coordinates to JSON file
            self.save_coordinates_json()
        else:
            messagebox.showwarning("No Drawing",
                                 "Please draw something before clicking Done!")

    def save_coordinates_json(self):
        try:
            # Ask user where to save the JSON file
            file_path = filedialog.asksaveasfilename(
                defaultextension=".json",
                filetypes=[("JSON files", ".json"), ("All files", ".*")],
                title="Save coordinates as JSON"
            )

            if file_path:
                # Create JSON data structure
                coordinates_data = {
                    "drawing_info": {
                        "canvas_width": self.canvas_width,
                        "canvas_height": self.canvas_height,
                        "total_line_segments": len(self.line_coordinates)
                    },
                    "line_coordinates": []
                }

```

Electronics Club
IIT Kanpur

```
                # Add all line coordinates
                for i, coords in enumerate(self.line_coordinates):
                    line_data = {
                        "line_id": i + 1,
                        "start_point": {"x": coords[0], "y": coords[1]},
                        "end_point": {"x": coords[2], "y": coords[3]}
                    }
                    coordinates_data["line_coordinates"].append(line_data)

                # Save to JSON file
                with open(file_path, 'w') as json_file:
                    json.dump(coordinates_data, json_file, indent=2)

                messagebox.showinfo("Success",
                    f"Coordinates saved to JSON!\nFile: {file_path}\n"
                    f"Total segments: {len(self.line_coordinates)}")
                self.status_label.config(text="Drawing saved! You can "
                    "continue drawing or save again.")

        except Exception as e:
            messagebox.showerror("Error",
                                 f"Failed to save coordinates: {str(e)}")

    def quit_app(self):
        self.root.destroy()

    def get_coordinates(self):
        """Return the stored line coordinates"""
        return self.line_coordinates

def main():
    root = tk.Tk()
    app = DrawingApp(root)
    root.mainloop()

if name == "main":
    main()
```

1) Marker Detection System Initialization: The system initializes with a predefined ArUco dictionary (DICT_4X4_1000) containing 1000 unique markers, each 4x4 in size. The marker size is set to 18.7 cm for distance calculation purposes.

2) Camera Frame Processing: Each frame from the camera is first converted to grayscale using cv2.cvtColor() since ArUco detection works more efficiently on grayscale images rather than color images.

3) Marker Detection Algorithm: The detectMarkers() function uses OpenCV's ArUco detector to scan the grayscale frame and identify square patterns that match the predefined dictionary markers.

4) Marker Information Extraction: For each detected marker, the system extracts the marker ID, calculates the centroid using image moments, and stores the four corner coordinates for further processing.

5) Distance Calculation: The system calculates the real-world distance to each marker by comparing the pixel size of the detected marker with the known physical size (18.7 cm) using the formula: distance = (marker_size_cm × frame_width) / marker_size_pixels.

6) Visual Marker Annotation: Detected markers are visually annotated on the frame with their ID numbers, bounding boxes, and calculated distances displayed as text overlays near each marker.

7) Centroid Marking: The centroid of each detected marker is marked with a magenta circle to clearly indicate the center point used for position tracking and distance measurements.

8) Inter-marker Distance Calculation: When multiple markers are detected, the system draws lines between all pairs of markers and calculates the pixel distance between their centroids for spatial relationship analysis.

9) Real-time Display: All processing results are displayed in real-time on the video feed, showing detected markers, distances, connections, and annotations for immediate visual feedback.

10) System Integration: The marker detection system serves as the foundation for robot localization, providing precise position and orientation data that can be used for navigation and coordinate mapping in the DoodleBot project.

7 Week-3

- **Interfacing camera module with RPi**
  - The Raspberry Pi Camera Module is a small, lightweight, and high-resolution camera used for image and video capture. It connects directly to the Raspberry Pi board via the CSI (Camera Serial Interface) port. We got familiar by using camera module through RPi.

- **Controlling DC motor with RPi**
  - PID (Proportional-Integral-Derivative) control is a widely used feedback mechanism in control systems. It continuously calculates an error value as the difference between a desired setpoint and a measured process variable and applies a correction based on proportional, integral, and derivative terms.

8 Week-4

- **Created GUI to accept user input doodles**

```python
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import json

class DrawingApp:
    def _init_(self, root):
        self.root = root
        self.root.title("Drawing Interface")
        self.root.geometry("650x500")

        # Canvas dimensions
        self.canvas_width = 600
        self.canvas_height = 400

        # Drawing variables
        self.old_x = None
        self.old_y = None
        self.brush_size = 3
        self.brush_color = "black"

        # Store coordinates of drawn lines
        self.line_coordinates = []

        self.setup_ui()

    def setup_ui(self):
        # Main frame
        main_frame = ttk.Frame(self.root, padding="10")
```

Electronics Club
IIT Kanpur

```
        main_frame.grid(row=0, column=0, sticky=(tk.W, tk.E, tk.N, tk.S))

        # Control frame
        control_frame = ttk.Frame(main_frame)
        control_frame.grid(row=0, column=0, columnspan=2,
                          sticky=(tk.W, tk.E), pady=(0, 10))

        # Action buttons — all three buttons are always visible
        ttk.Button(control_frame, text="Clear",
                  command=self.clear_canvas).pack(side=tk.LEFT, padx=(0, 10))
        ttk.Button(control_frame, text="Done",
                  command=self.done_drawing).pack(side=tk.LEFT, padx=(0, 10))
        ttk.Button(control_frame, text="Quit",
                  command=self.quit_app).pack(side=tk.LEFT, padx=(0, 10))

        # Canvas
        self.canvas = tk.Canvas(main_frame, bg="white",
                               width=self.canvas_width,
                               height=self.canvas_height,
                               relief=tk.SUNKEN, borderwidth=2)
        self.canvas.grid(row=1, column=0, columnspan=2, pady=(0, 10))

        # Bind mouse events
        self.canvas.bind("<B1-Motion>", self.paint)
        self.canvas.bind("<ButtonPress-1>", self.start_paint)
        self.canvas.bind("<ButtonRelease-1>", self.stop_paint)

        # Status label
        self.status_label = ttk.Label(main_frame, text="Draw on the canvas...")
        self.status_label.grid(row=2, column=0, columnspan=2, sticky=tk.W)

    def start_paint(self, event):
        self.old_x = event.x
        self.old_y = event.y

    def paint(self, event):
        if self.old_x and self.old_y:
            # Draw on canvas
            self.canvas.create_line(self.old_x, self.old_y, event.x, event.y,
                                   width=self.brush_size, fill=self.brush_color,
```

Electronics Club
IIT Kanpur

```
                                        capstyle=tk.ROUND, smooth=tk.TRUE)

                # Store line coordinates
                line_coords = [self.old_x, self.old_y, event.x, event.y]
                self.line_coordinates.append(line_coords)

            self.old_x = event.x
            self.old_y = event.y

        def stop_paint(self, event):
            self.old_x = None
            self.old_y = None

        def clear_canvas(self):
            self.canvas.delete("all")
            self.line_coordinates = []
            self.status_label.config(text="Canvas cleared — ready for new drawing")

        def done_drawing(self):
            # Save coordinates to JSON file without removing the Done button
            if self.line_coordinates:
                # Save coordinates to JSON file
                self.save_coordinates_json()
            else:
                messagebox.showwarning("No Drawing",
                                      "Please draw something before clicking Done!")

        def save_coordinates_json(self):
            try:
                # Ask user where to save the JSON file
                file_path = filedialog.asksaveasfilename(
                    defaultextension=".json",
                    filetypes=[("JSON files", ".json"), ("All files", ".*")],
                    title="Save coordinates as JSON"
                )

                if file_path:
                    # Create JSON data structure
                    coordinates_data = {
                        "drawing_info": {
```

```
                        "canvas_width": self.canvas_width,
                        "canvas_height": self.canvas_height,
                        "total_line_segments": len(self.line_coordinates)
                    },
                    "line_coordinates": []
                }

                # Add all line coordinates
                for i, coords in enumerate(self.line_coordinates):
                    line_data = {
                        "line_id": i + 1,
                        "start_point": {"x": coords[0], "y": coords[1]},
                        "end_point": {"x": coords[2], "y": coords[3]}
                    }
                    coordinates_data["line_coordinates"].append(line_data)

                # Save to JSON file
                with open(file_path, 'w') as json_file:
                    json.dump(coordinates_data, json_file, indent=2)

                success_msg = (f"Coordinates saved to JSON!\n"
                             f"File: {file_path}\n"
                             f"Total segments: {len(self.line_coordinates)}")
                messagebox.showinfo("Success", success_msg)

                status_text = ("Drawing saved! You can continue drawing "
                             "or save again.")
                self.status_label.config(text=status_text)

        except Exception as e:
            messagebox.showerror("Error",
                               f"Failed to save coordinates: {str(e)}")

    def quit_app(self):
        self.root.destroy()

    def get_coordinates(self):
        """Return the stored line coordinates"""
        return self.line_coordinates

```

Electronics Club
IIT Kanpur

```python
def main():
    root = tk.Tk()
    app = DrawingApp(root)
    root.mainloop()

if _name_ == "_main_":
    main()
```

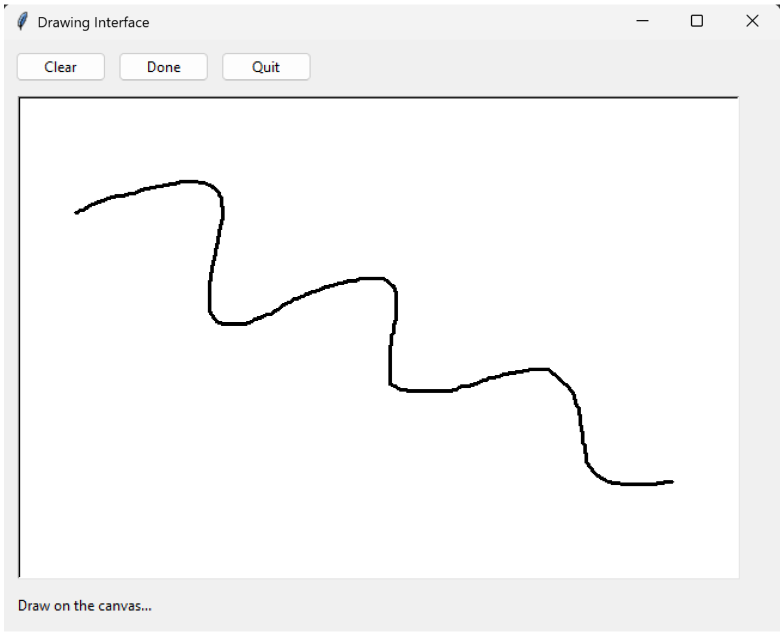


Figure 5: DoodleBot GUI

- **Logic-**
    - Initialize GUI: It creates a window with a white canvas (600x400 pixels) and three buttons (Clear, Done, Quit)
    - Mouse Click Detection: When user clicks on canvas, it records the starting position (x, y) coordinates
    - Drawing Lines: As user drags mouse, it draws continuous lines by connecting previous position to current position
    - Store Coordinates: Each line segment gets stored as [start_x, start_y, end_x, end_y] in a list called line_coordinates

  - Real-time Display: Lines appear on screen immediately as user draws, with black color
  - Clear Button: "Clear" button erases everything from canvas and empties the coordinate storage
  - Done Button: "Done" button allows us to save the coordinates when user finishes drawing
  - JSON Conversion: Drawing gets converted into structured JSON format with all line coordinates and canvas information
  - File Save:I can choose where to save the JSON file containing all the drawing coordinate data

- **Configured a touch controller that interfaces with RPi**
  - The GUI designed to accept the user-drawn sketch served as the primary user interface for the DoodleBot, allowing users to draw sketches that will be replicated by the robot.
  - The touch-enabled LCD display interfaces directly with the RPi to capture user input and convert it into coordinate data for robot motion planning. We were able to run GUI on RPi and hence capture stroke data.

- **Assembly for 3-D printed parts**
  - We got all the 3D printed components of the chassis of the DoodleBot and initiated assembling of the various components.

9 Outcome

- Capturing user-drawn sketches via a touchscreen interface.
- Converting those sketches into coordinate-based paths.
- Mapping the paths to motion commands using image processing and ArUco marker tracking
- Controlling a wheeled robot equipped with DC motor+wheel encoder with a pen-lift mechanism to accurately draw the sketch on paper
- Executing smooth movements with PID-based motor control+encoder with RPi for precision.

10 Upcoming weeks project plan

- Assemble and build a functional prototype of the bot
- Develop a coordinate system using the RPi camera and ARUco markers
- Integrate interfacing and communication mechanisms with the RPi to complete the final working version of the DoodleBot
- PID tuning to achieve accurate results

11 Mentors

- Ashish Upadhyay
- Bhuvan Kumar
- Dhakshith
- Mrigeesh Ashwin

12 Mentees

- Akash Kumar
- Akriti
- Anantham R
- Ayush Muttepawar
- Jugal Pahuja
- Manish Kumar Meena
- Pratham Gupta
- Sarthak
- Shubham
- Siva