AI-POWERED TABLE TENNIS: TRAJECTORY PREDICTION AND OPTIMAL SHOT STRATEGY

Project Team

SIDDHARTH GOYAL (2022MEB1346)

SPANDAN SETH (2022MEB1348)

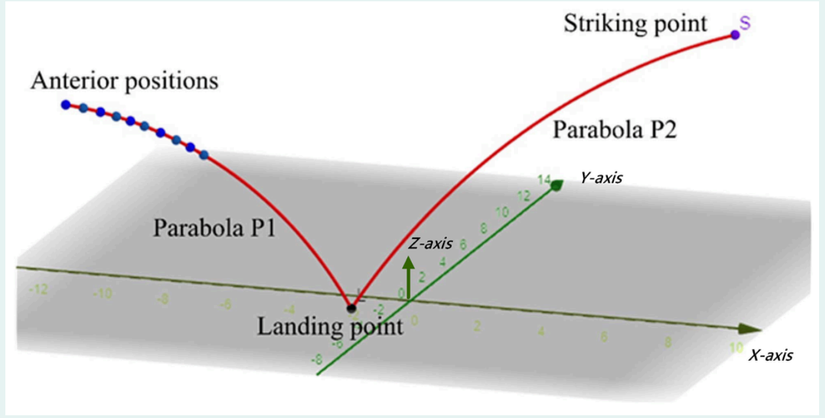
SUNNY BHARTI (2022MEB1353)

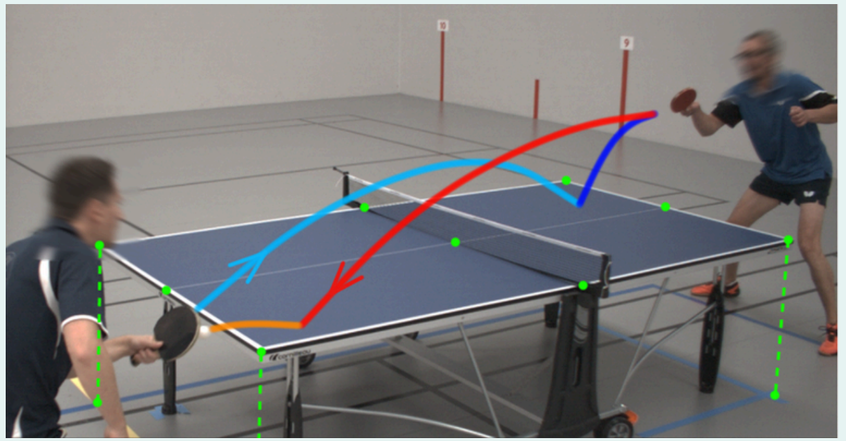
SUPERVISOR:

DR. NAVIN KUMAR (IIT ROPAR)

Introduction

- This project aims to revolutionize table tennis training by integrating artificial intelligence and computer vision into a **comprehensive coaching system.**
- **Advanced object detection (YOLOv11)** and custom-trained models identify and track the ball and paddle in real time, providing immediate feedback and detailed shot analysis.
- The system captures gameplay using synchronized **multi-angle video recordings**, enabling precise **3D reconstruction** of the ball's trajectory.
- A **user-friendly Streamlit interface** visualizes performance metrics, overlays detected trajectories, and offers interactive coaching insights, making advanced analytics accessible to players of all skill levels



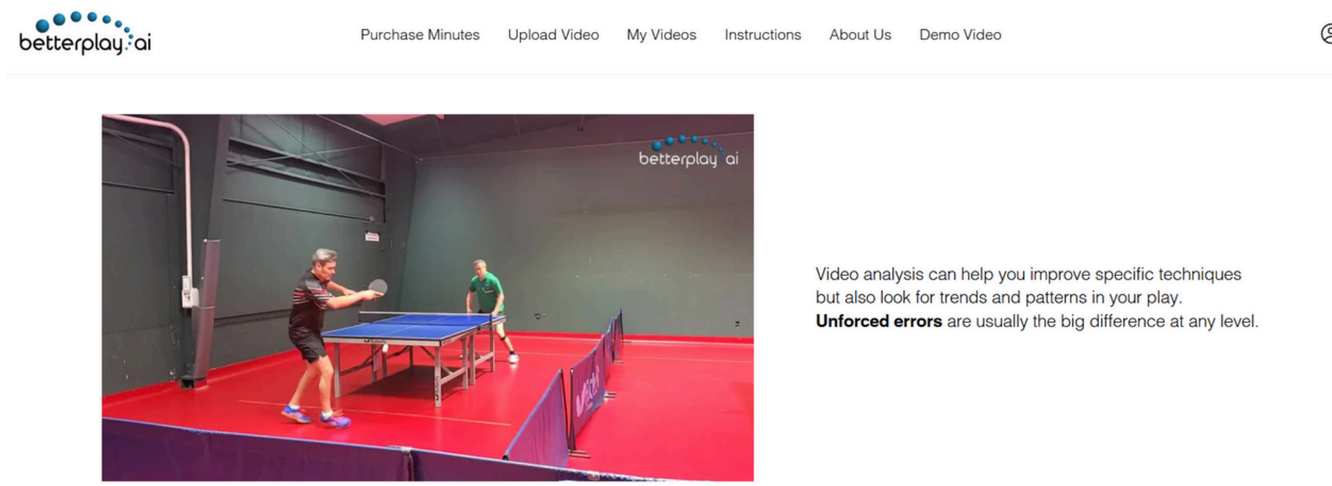


Current Solutions:

AI COACHING PLATFORMS

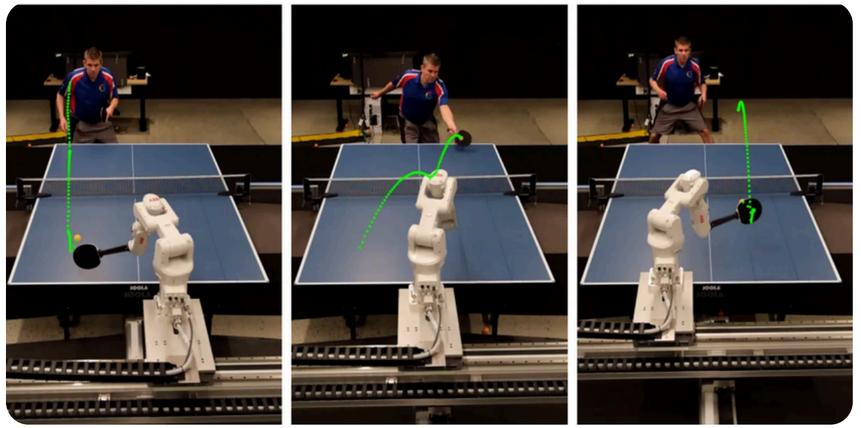
- **BetterPlay.ai:**
  - Stroke error detection (82% racket angle accuracy).
  - Comparative analytics against pro player databases.

- **PMC Coaching System:**
  - 70% overall prediction rate using GPT-4 + OpenPose integration.
  - Personalized training plans via reinforcement learning.

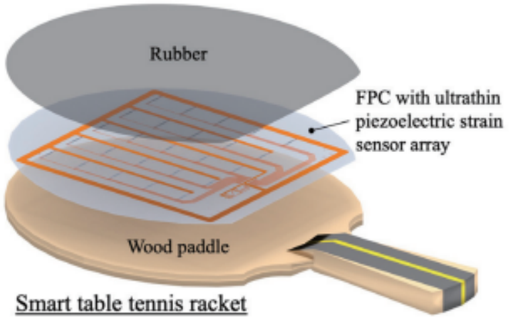


Current Solutions:

- **MIT Robotic Arm:**
  - 19m/s strike speeds with 87.5% drive shot accuracy
  - Adaptive spin handling (3000 RPM max).



- **Smart Racket v2.0:**
  - 25 piezoelectric sensors detecting spin axis (±5° accuracy),
  - Bluetooth 5.0 connectivity for instant feedback.



Smart table tennis racket

Problem Statement

PROJECT AIM:

To develop an AI-powered table tennis training system that accurately detects and predicts ball trajectories using synchronized multi-angle video analysis and advanced machine learning models.

Methodology

- **Multi-Angle Video Capture:**
  - Two 60 FPS iPhone cameras (top-view and side-view)
  - Camera synchronization for consistent 3D trajectory reconstruction

- **Object Detection:**
  - YOLOv11 model trained via Roboflow for ball and racket detection.
  - Custom dataset creation and annotation process.

- **Trajectory Analysis:**
  - 3D position estimation by combining dual camera perspectives.
  - Shot classification and recommendation system.

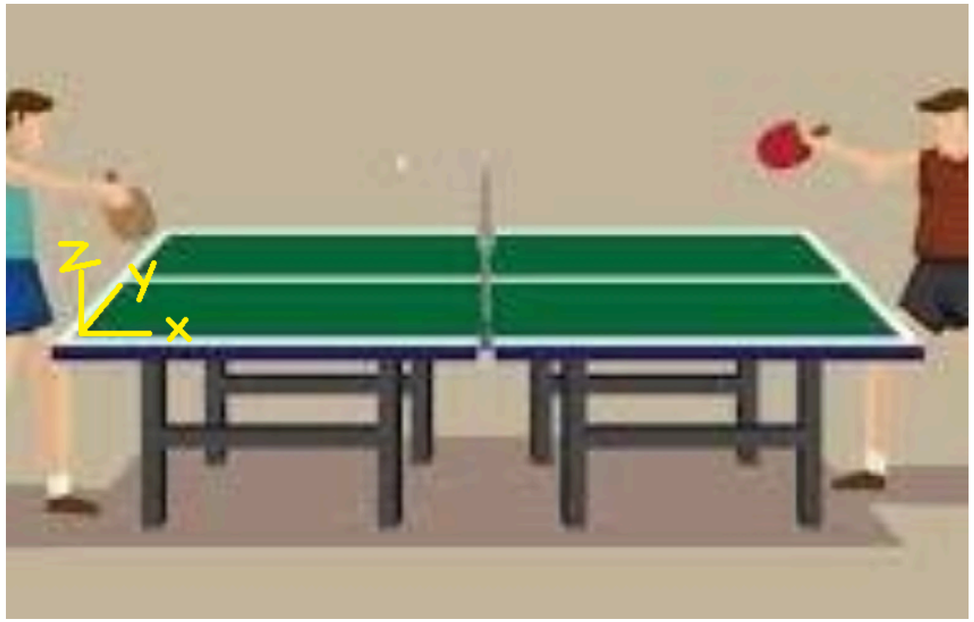
- **Real Time Interface:**
  - Usef Streamlit an open-source Python library that makes it easy to create and share custom web apps for machine learning and data science.

- **Ball Thrower Integration:**
  - Spring-loaded mechanism with servo motor control.
  - Arduino-based automation for consistent training scenarios.

Flawed Data Collection:

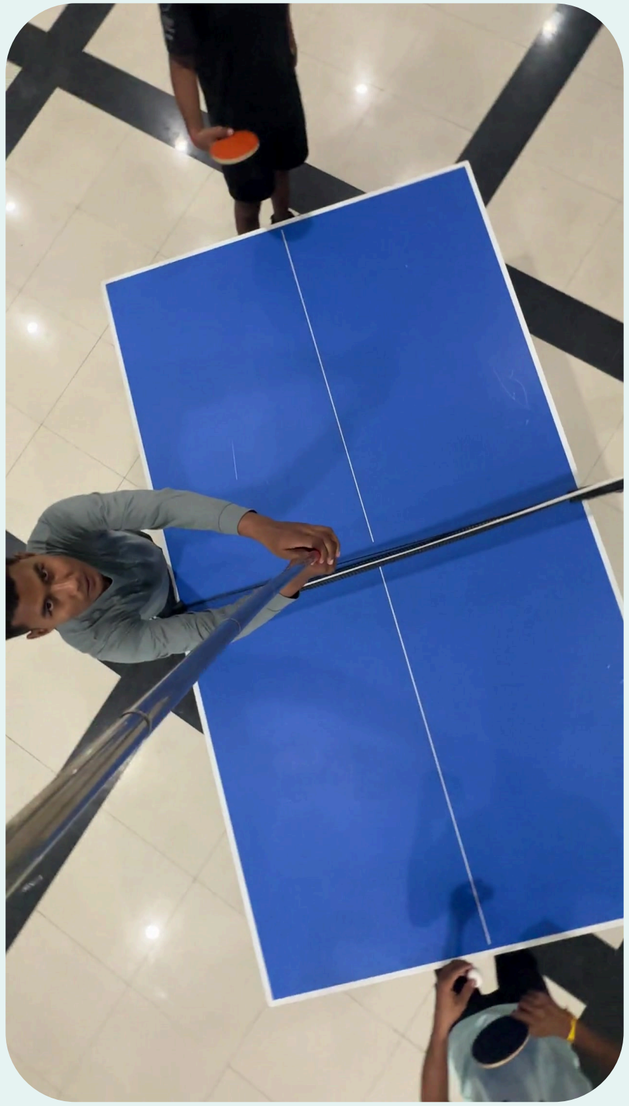
Alternate Camera Angle

TOP VIEW



SIDE VIEW

Data Collection



Coordinate Frame set up

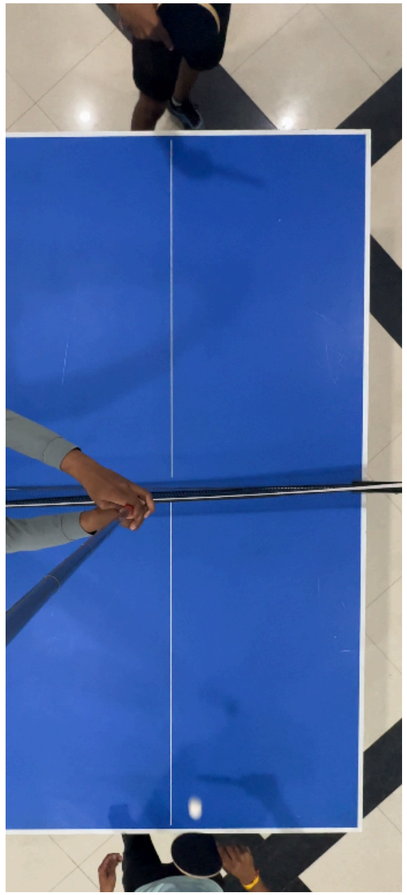
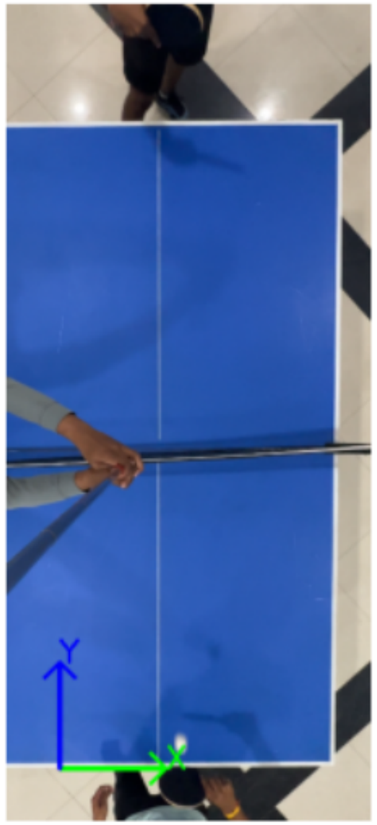


Table Coordinate Frame

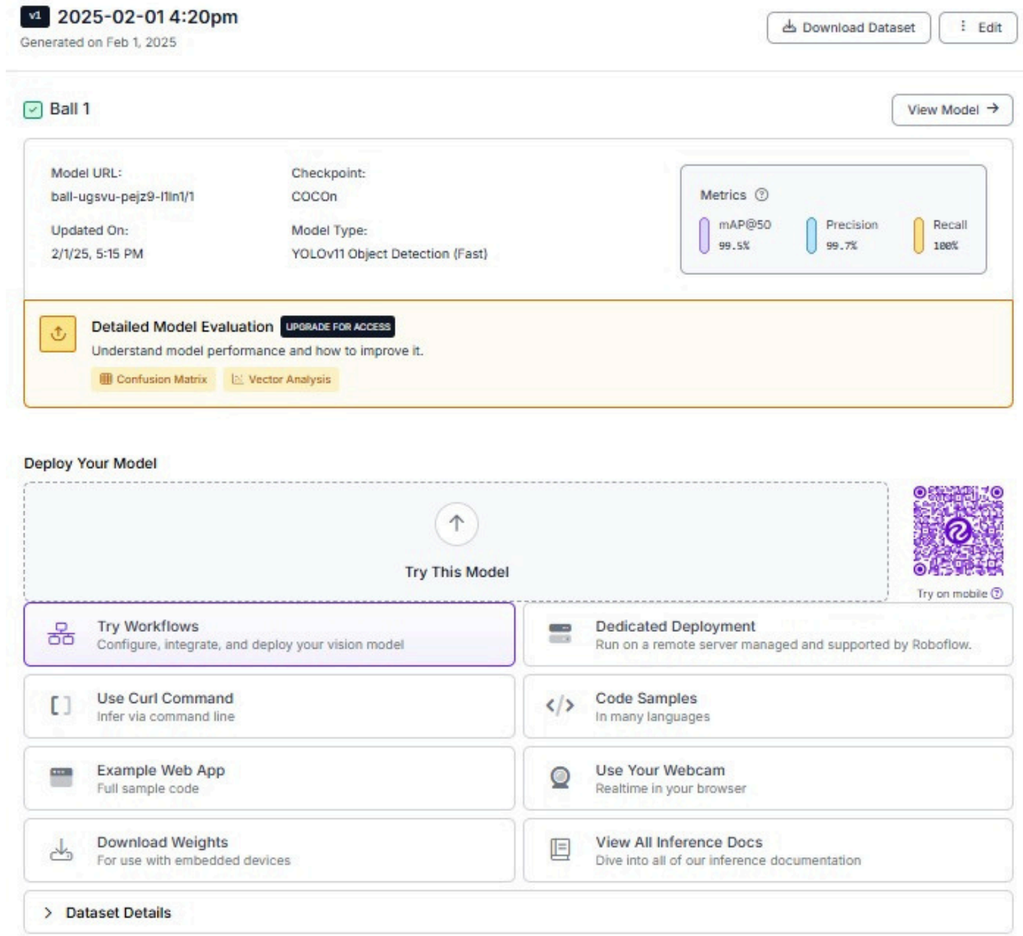


Coordinate Frame set up

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

# Define real table dimensions in meters
TABLE_LENGTH_X = 2.74  # meters
TABLE_WIDTH_Y = 1.525  # meters

# Define image dimensions (from top-view image)
IMAGE_PATH = "/content/Screenshot 2025-05-13 111344.png"
img = cv2.imread(IMAGE_PATH)
img_height, img_width = img.shape[:2]

# Define pixel-to-meter scale
scale_x = TABLE_LENGTH_X / img_width
scale_y = TABLE_WIDTH_Y / img_height

def pixel_to_table_coords(px, py, ball_height_m=0.0):
    """
    Convert pixel coordinates to real-world table coordinates (X, Y, Z)
    px, py: Pixel coordinates from image (top-down)
    ball_height_m: Height of the ball above the table (Z-axis)
    Returns: (X, Y, Z) in meters
    """
    x_m = px * scale_x
    y_m = py * scale_y
    z_m = ball_height_m
    return (x_m, y_m, z_m)

# Example usage
ball_pixel = (500, 300)  # example pixel coordinates (X, Y)
ball_real_world = pixel_to_table_coords(*ball_pixel, ball_height_m=0.1)

print(f"Ball real-world coordinates: X={ball_real_world[0]:.3f}m, Y={ball_real_world[1]:.3f}m, Z={ball_real_world[2]:.3f}m")
```

```python
def draw_axes(img, origin=(50, img_height - 50), length=100):
    x_axis = (origin[0] + length, origin[1])
    y_axis = (origin[0], origin[1] - length)

    cv2.arrowedLine(img, origin, x_axis, (0, 255, 0), 3, tipLength=0.2)  # X-axis: Green
    cv2.arrowedLine(img, origin, y_axis, (255, 0, 0), 3, tipLength=0.2)  # Y-axis: Blue
    cv2.putText(img, 'X', x_axis, cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0), 2)
    cv2.putText(img, 'Y', y_axis, cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)
    return img

img_with_axes = draw_axes(img.copy())
plt.imshow(cv2.cvtColor(img_with_axes, cv2.COLOR_BGR2RGB))
plt.title("Table Coordinate Frame")
plt.axis('off')
plt.show()
```

Object Detection Using Yolo v11



Codes

BALL DETECTION

```python
import cv2
import numpy as np
from collections import deque

def track_ball(frame, buffer_size=64):
    """
    Track a table tennis ball in the given frame

    Args:
        frame: Input video frame
        buffer_size: Size of points buffer for trajectory tracking

    Returns:
        processed_frame: Frame with tracking visualization
        ball_data: Dictionary with ball metrics if detected, None otherwise
    """
    # Initialize ball data
    ball_data = None

    # Convert to HSV for better color segmentation
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define color range for orange/white table tennis ball
    # Adjust these values based on your specific lighting conditions
    lower_bound = np.array([0, 0, 200])  # For white ball
    upper_bound = np.array([180, 70, 255])

    # Create mask
    mask = cv2.inRange(hsv, lower_bound, upper_bound)

    # Apply erosion and dilation to remove noise
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.erode(mask, kernel, iterations=1)
    mask = cv2.dilate(mask, kernel, iterations=2)
```

```python
    # Find contours
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Process frame to add visualization
    processed_frame = frame.copy()

    # Initialize point buffer for trajectory tracking if not already created
    if not hasattr(track_ball, "pts"):
        track_ball.pts = deque(maxlen=buffer_size)

    # If ball contour found
    if contours:
        # Find the largest contour (assumed to be the ball)
        c = max(contours, key=cv2.contourArea)

        # Check if the contour is large enough to be a ball
        if cv2.contourArea(c) > 50:
            # Find the minimum enclosing circle
            ((x, y), radius) = cv2.minEnclosingCircle(c)

            # Calculate ball center using moments
            M = cv2.moments(c)
            center = (int(M["m10"] / M["m00"]), int(M["m01"] / M["m00"]))

            # Only proceed if the radius is larger than a minimum size
            if radius > 5:
                # Draw circle and centroid
                cv2.circle(processed_frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
                cv2.circle(processed_frame, center, 5, (0, 0, 255), -1)

                # Add the centroid to the points queue
                track_ball.pts.appendleft(center)

                # Calculate ball metrics (simplified)
                ball_data = {
                    'position': (x, y),
                    'radius': radius,
                    'speed': calculate_speed(track_ball.pts),
                    'spin_type': estimate_spin_type(track_ball.pts)
                    }
```

```python
    # Draw trajectory
    for i in range(1, len(track_ball.pts)):
        if track_ball.pts[i - 1] is None or track_ball.pts[i] is None:
            continue

        # Thicker lines for more recent motion
        thickness = int(np.sqrt(buffer_size / float(i + 1)) * 2.5)
        cv2.line(processed_frame, track_ball.pts[i - 1], track_ball.pts[i], (0, 0, 255), thickness)

    return processed_frame, ball_data
```

Codes

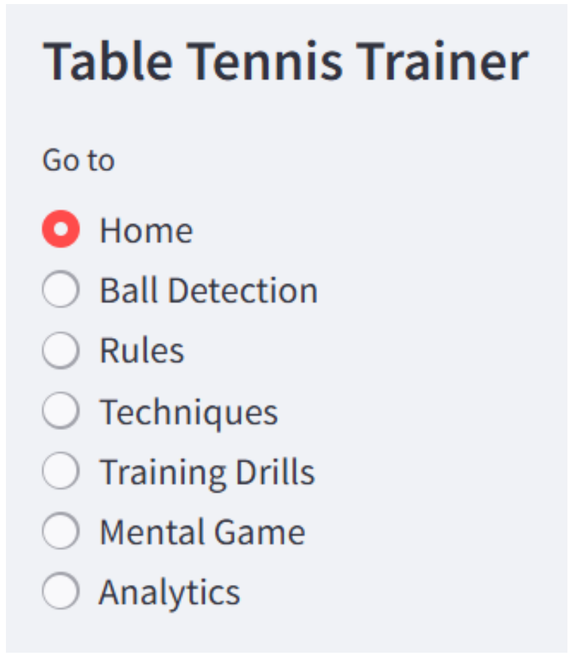
CALCULATING SPEED

```python
def calculate_speed(pts):
    """
    Calculate estimated ball speed based on points history
    This is a simplified calculation and would need calibration
    """
    if len(pts) < 2:
        return 0

    # Calculate distance between the most recent points
    # This needs calibration with actual distances for accuracy
    dx = pts[0][0] - pts[1][0]
    dy = pts[0][1] - pts[1][1]
    distance = np.sqrt(dx*2 + dy*2)

    # Assuming 30 fps, convert to m/s (very rough approximation)
    # Would need proper calibration based on camera and table setup
    return distance * 0.01  # Arbitrary scaling factor
```

ESTIMATING SPIN TYPE

```python
def estimate_spin_type(pts):
    """
    Estimate type of spin based on ball trajectory
    This is a simplified heuristic and would need refinement
    """
    if len(pts) < 10:
        return "Unknown"

    # Analyze trajectory curvature
    # This is a simplification and would need more sophisticated algorithms
    x_values = [pt[0] for pt in pts if pt is not None]
    y_values = [pt[1] for pt in pts if pt is not None]

    if len(x_values) < 3:
        return "Unknown"

    # Very basic heuristic
    x_diff = max(x_values) - min(x_values)
    y_diff = max(y_values) - min(y_values)

    if x_diff > y_diff * 2:
        return "Sidespin"
    elif y_diff > x_diff * 2:
        return "Topspin/Backspin"
    else:
        return "Mixed Spin"
```

Table Tennis Skill Development Application



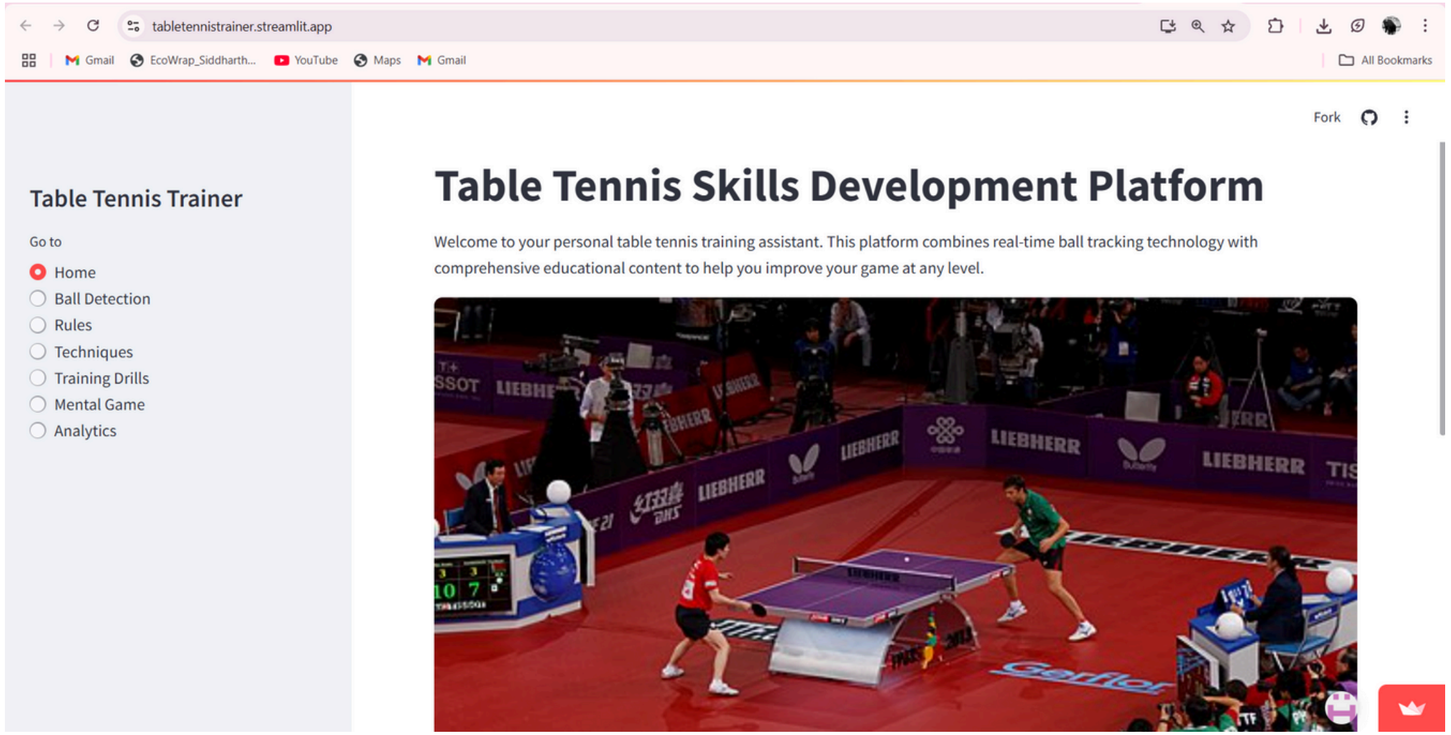


Table Tennis Skill Development Application

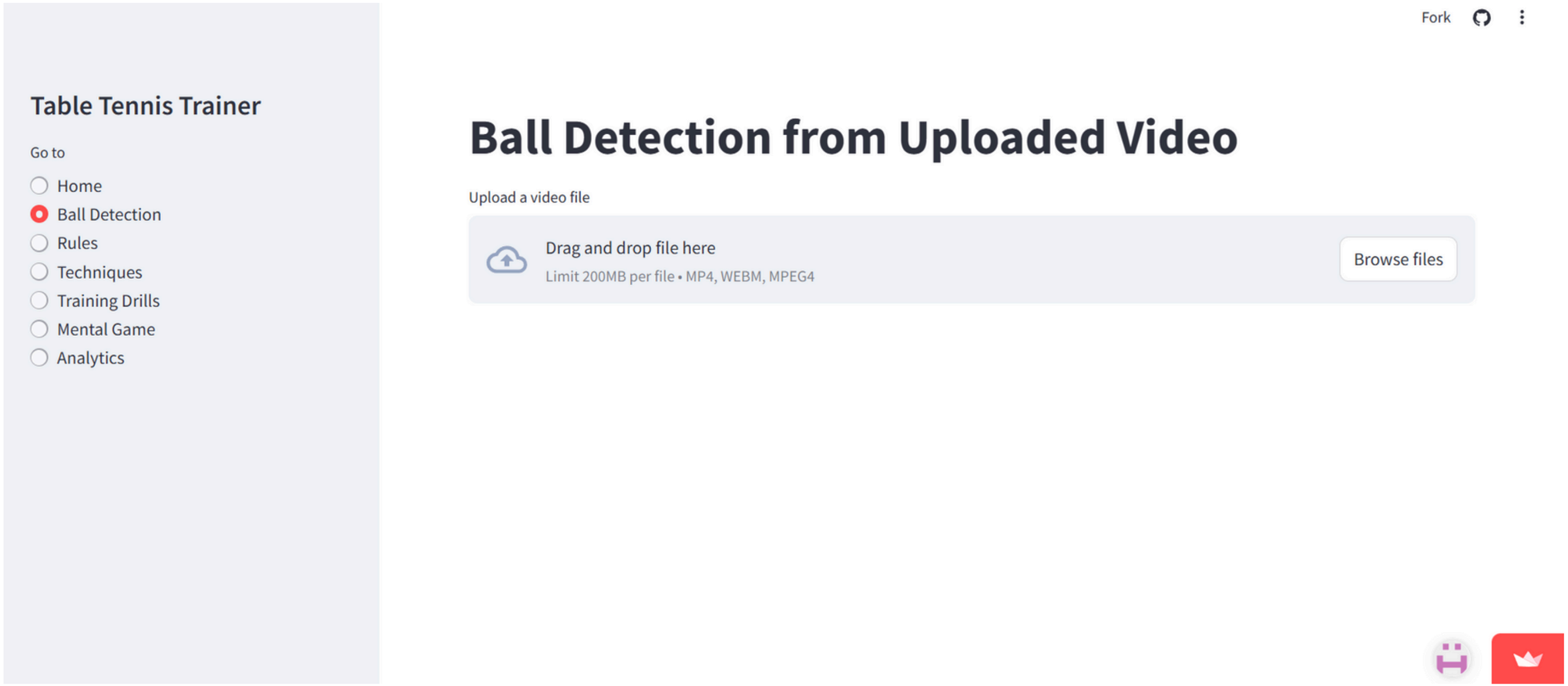
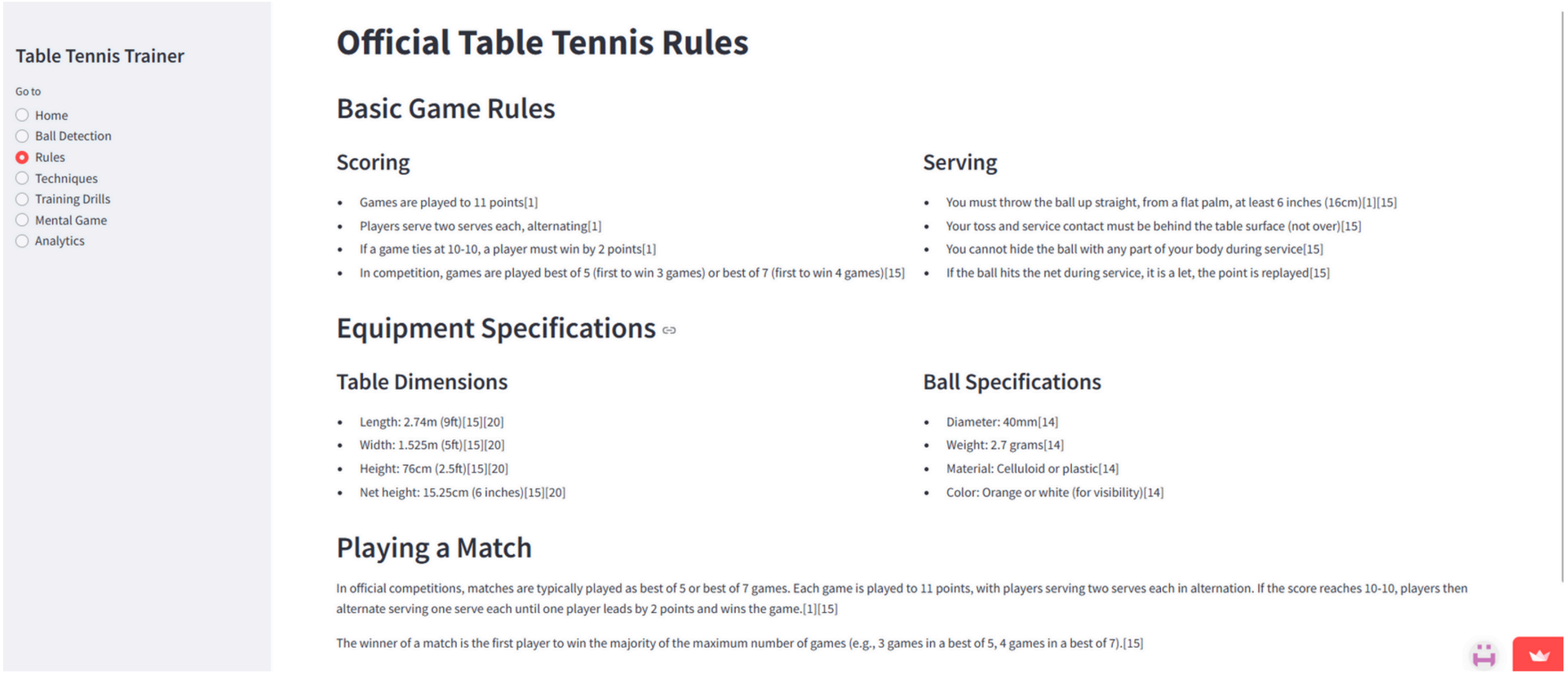


Table Tennis Skill Development Application



Codes

STREAMLIT APP DEPLOYED CODE

Streamlit App Deployed Code

```python
import streamlit as st
import cv2
import numpy as np
from ball_tracker import track_ball
from pages.rules import show_rules_page
from pages.techniques import show_techniques_page
from pages.drills import show_drills_page
from pages.mental_game import show_mental_game_page

# Set page config
st.set_page_config(
    page_title="Table Tennis Trainer",
    page_icon="🏓",
    layout="wide",
    initial_sidebar_state="expanded"
)

# Sidebar navigation
st.sidebar.title("Navigation")
page = st.sidebar.radio(
    "Go to",
    ["Home", "Ball Tracking", "Rules", "Techniques", "Training Drills", "Mental Game", "Analytics"]
)
```

```python
# Home page
if page == "Home":
    st.title("Table Tennis Skills Development Platform")
    st.write("""
    Welcome to your personal table tennis training assistant. This platform combines
    real-time ball tracking technology with comprehensive educational content to help
    you improve your game at any level.
    """)

    st.image("assets/table_tennis_hero.jpg", use_column_width=True)

    st.subheader("Features")
    col1, col2, col3 = st.columns(3)

    with col1:
        st.markdown("### 🎯 Ball Tracking")
        st.write("Analyze your shots with real-time ball tracking")

    with col2:
        st.markdown("### 📚 Learning Resources")
        st.write("Comprehensive guides on techniques, rules, and strategy")

    with col3:
        st.markdown("### 🏆 Progress Tracking")
        st.write("Monitor your improvement over time")
```

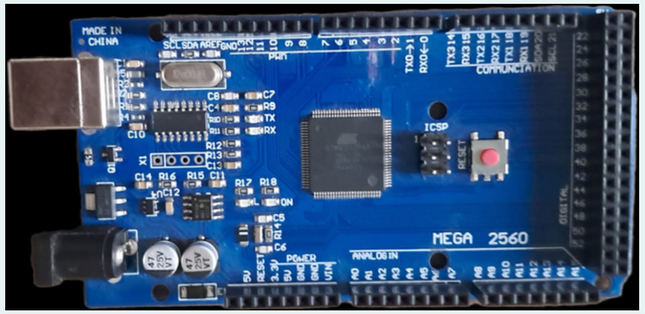
Codes

STREAMLIT APP DEPLOYED CODE

```python
# Ball Tracking page
elif page == "Ball Tracking":
    st.title("Ball Tracking Analysis")

    run = st.button("Start Camera")
    stop = st.button("Stop")

    frame_placeholder = st.empty()

    if run and not stop:
        cap = cv2.VideoCapture(0)

        while cap.isOpened() and not stop:
            success, frame = cap.read()
            if not success:
                st.write("Video capture has ended")
                break

            # Process frame with ball tracking
            processed_frame, ball_data = track_ball(frame)

            # Convert from BGR to RGB for Streamlit
            processed_frame_rgb = cv2.cvtColor(processed_frame, cv2.COLOR_BGR2RGB)

            # Display frame
            frame_placeholder.image(processed_frame_rgb, channels="RGB")

            # Display metrics if ball detected
            if ball_data:
                st.metric("Ball Speed", f"{ball_data['speed']:.2f} m/s")
                st.metric("Spin Type", ball_data['spin_type'])

            if cv2.waitKey(1) & 0xFF == ord('q') or stop:
                break

        cap.release()
        cv2.destroyAllWindows()
```

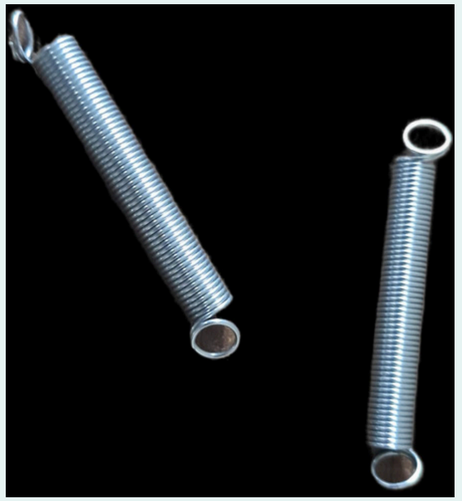
```python
# Rules page
elif page == "Rules":
    show_rules_page()

# Techniques page
elif page == "Techniques":
    show_techniques_page()

# Training Drills page
elif page == "Training Drills":
    show_drills_page()

# Mental Game page
elif page == "Mental Game":
    show_mental_game_page()

# Analytics page
elif page == "Analytics":
    st.title("Performance Analytics")
    st.write("Analytics content will go here.")
```

Ball Thrower

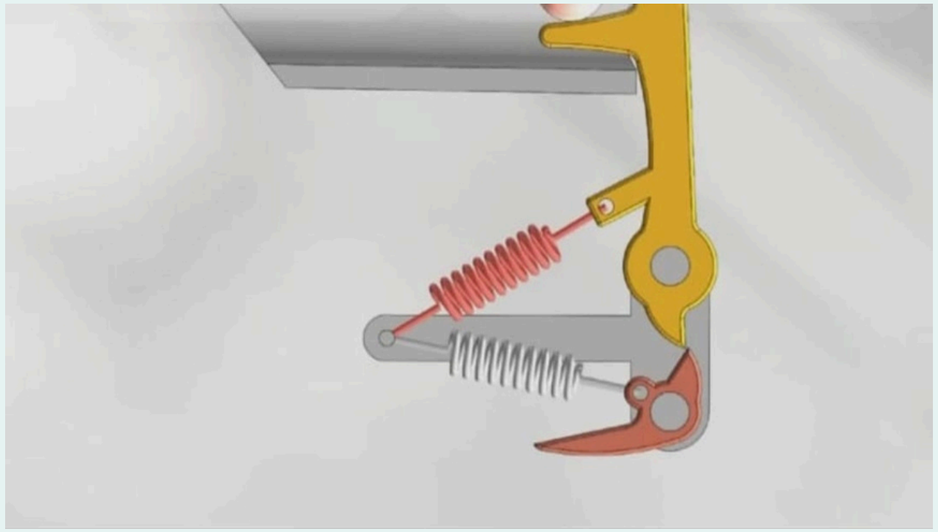


ARDUINO MEGA 2560

SERVO MG996R



DUAL-SPRING SYSTEM

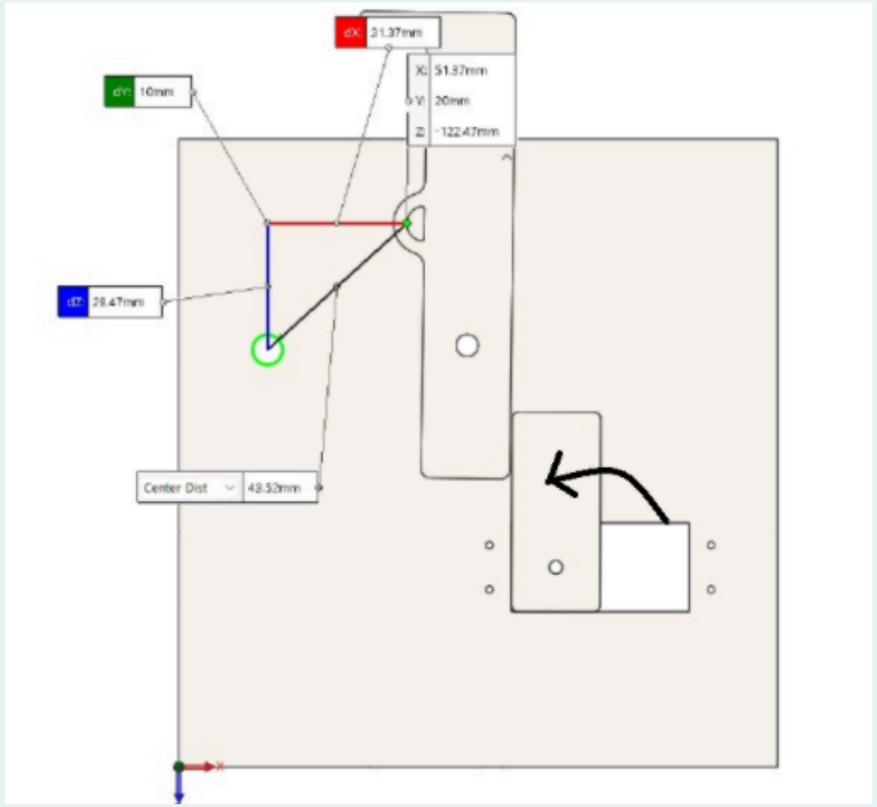


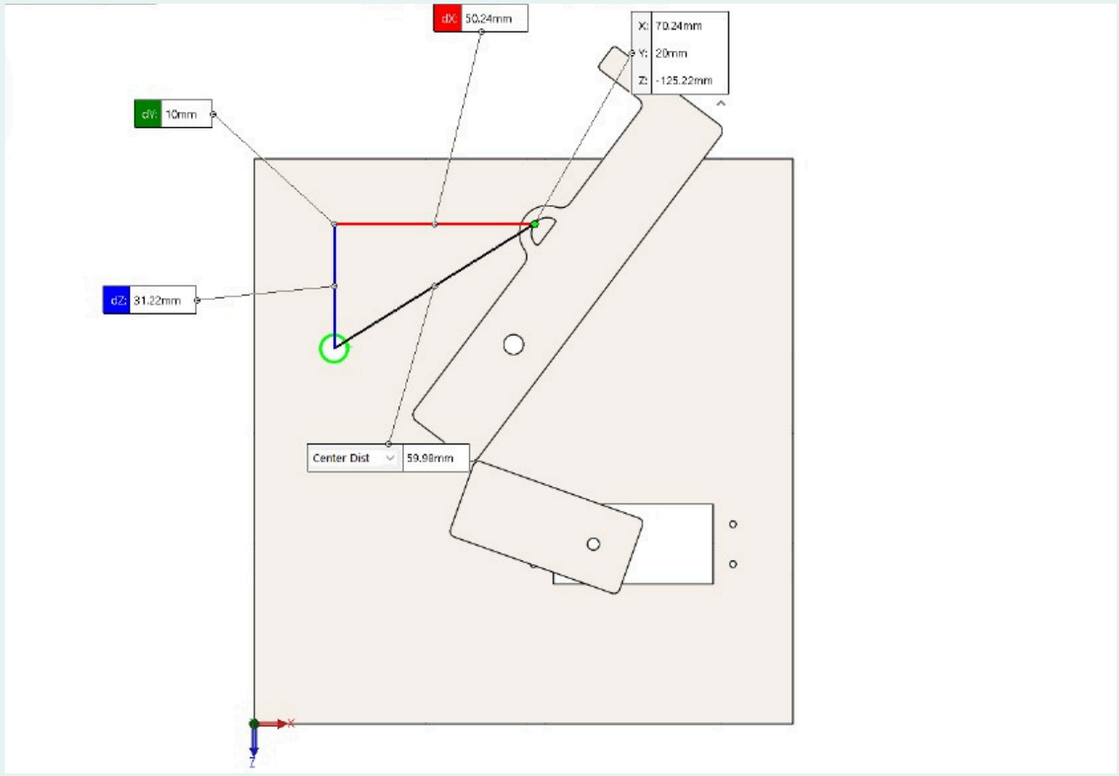
Theoretical calculations were done to estimate ball velocity and throw distance:

- Spring Constant: 100 N/m
- Extension: 16.5 mm
- Resulting launch velocity: ~3.18 m/s
- Distance traveled: ~0.64 m (valid for mid-table practice shots)

Ball Thrower

MECHANISM





Key Challenges

KEY CHALLENGES:

- Only 60 fps camera available, leading to less frames and low frame quality.
- Fast moving ball having difficulty in tracking.
- Due to low frames, ball couldnt be tracked as a white dot, but appeared as a dash line.
- lightening in video recording caused hindrance in ball and trajectory tracking.
- Limited accessibility of high computer power
- Ensuring exact synchronization between the two cameras to generate accurate 3D trajectories required careful calibration.
- Arduino Failure.

Thank you very much!