2025

Project Proposal

MECH 421

KYLE AH VON

Concept

The project is a **vision-guided robotic arm that plays Tic-Tac-Toe against a human opponent**. A camera captures the board state, a server runs the game logic and user interface, and the MSP430 microcontroller drives the servos via a PCA9685 board to physically place tokens on the board. The mechanical arm is pre-assembled with metal gear servos, reducing fabrication effort and allowing focus on **integration, calibration, and control**. The novelty lies in combining **computer vision, embedded control, and interactive gameplay** into a single system. The final demo will showcase the robot playing full games against a human, with smooth motion, accurate placement, and a polished interface.

1. Answers to Concept Proposal Questions

1. **Value to end-user**: The system provides an engaging demonstration of robotics and AI in a familiar game format. It has educational value (showing vision, control, and communication integration) and entertainment appeal.
2. **Closest alternative**: A digital Tic-Tac-Toe app. Unlike a screen-based game, this project offers tangible interaction with a physical robot arm, making it more engaging and memorable.
3. **Metric of success**: Success will be measured by (a) accurate token placement in ≥95% of moves, (b) latency ≤2 seconds from user input to robot action, and (c) completion of full games without illegal moves or system crashes.
4. **Polished aspect**: Motion mechanism — smooth, repeatable trajectories, no jitter, reliable pick-and-place. A polished motion system demonstrates mastery of servo control and kinematics.
5. **Not polished**: Mechanical housing and aesthetics. The arm will remain functional but not cosmetically refined.
6. **Most Critical Module (MCM)**: Vision + board recognition. This is the riskiest due to segmentation accuracy, lighting variability, and coordinate mapping.
7. **Data infrastructure**: Camera → server segmentation → UI → ESP32 → MSP430 → servos. Mock data (synthetic board states, coordinate packets) will be used to test communication and control before full vision integration.

2. Overview of Functional Requirements

1. Components

1. MSP430FR5739 development board
2. ESP32-S3 development board + Camera Module
3. Mechanical Arm – 6 degrees of freedom
4. 6 metal gear servo motors
5. PCA9685 board (Multi servo driver)
6. Wall power supply (high amps 12A)
7. Fuses (to protect servo motors in case I stall them…)

2. Most Critical Module (MCM)

The **vision pipeline** is the MCM. Challenges include:

- Detecting tokens reliably under varying lighting conditions.
- Mapping camera pixels to board coordinates (homography calibration).
- Handling misreads and ensuring synchronization between vision and game logic.

3. Functional Component #1: Vision & Sensor Interface

- **Approach and Design**: Use ESP32-CAM or PC webcam to capture frames. Apply OpenCV color thresholding or ArUco markers to detect tokens. Homography transform maps pixels to board squares.
- **Inputs/Outputs**: Input = RGB frames. Output = 3×3 board state array, token coordinates.
- **Parameters**: Color thresholds, grid calibration matrix, frame resolution.
- **Development Plan**:
    1. Capture frames.
    2. Implement color thresholding.
    3. Calibrate grid mapping.
    4. Output board state to UI.

- **Test Plan**:
- Test detection accuracy with mock tokens.
- Vary lighting conditions.
- Validate coordinate mapping by comparing detected vs actual positions.

4. Functional Component #2: Mechanical Arm Calibration & Integration

- **Approach and Design**: Pre-assembled arm with metal gear servos. Focus on calibration and workspace mapping.
- **Inputs/Outputs**: Input = servo angle commands. Output = token placed at target square.
- **Parameters**: Servo speed, gripper force, joint angle limits.
- **Development Plan**:
    1. Calibrate servo ranges.
    2. Map workspace to board coordinates.
    3. Tune gripper force.
- **Test Plan**:
- Place tokens in all 9 squares.
- Measure accuracy and repeatability.
- Test gripper reliability with different token sizes.
-

5. Functional Component #3: MSP430 Servo Control

- **Approach and Design**: MSP430 communicates with PCA9685 via I2C. Implements inverse kinematics and trajectory planning.
- **Inputs/Outputs**: Input = target coordinates from ESP32. Output = PWM signals to servos.
- **Parameters**: Joint angle limits, trajectory interpolation step size.
- **Development Plan**:

1. Implement PCA9685 driver.
2. Develop inverse kinematics solver.
3. Integrate trajectory planning.

- **Test Plan**:
- Verify servo angles.
- Test smoothness of trajectories.
- Ensure safety limits are respected.

6. Functional Component #4: Communication Protocol

- **Approach and Design**: Define UART/SPI packet format for commands. ESP32 relays server instructions to MSP430.
- **Inputs/Outputs**: Input = board state, user moves. Output = compact command packets.
- **Parameters**: Baud rate, packet format, error detection.
- **Development Plan**:
  1. Define packet structure.
  2. Implement parsing on MSP430.
  3. Test with mock commands.
- **Test Plan**:
- Send mock commands.
- Verify correct parsing and execution.
- Stress test with rapid moves.

7. Functional Component #5: System Feedback & Telemetry

- **Approach and Design**: Provide real-time feedback on system state (move executed, error detected, servo status). ESP32 collects status and displays logs in UI.

- **Inputs/Outputs**: Input = servo position confirmations, vision confidence scores. Output = status messages in UI, error alerts.
- **Parameters**: Logging frequency, error thresholds, feedback channels.
- **Development Plan**:
    1. Define telemetry packet format.
    2. Implement MSP430 → ESP32 status reporting.
    3. Integrate logs into server UI.
    4. Add error handling routines.
- **Test Plan**:
- Inject mock errors and verify detection.
- Test logging under normal and stress conditions.
- Confirm UI displays accurate system state.

3. Data Infrastructure

Data Flow Stages

1. **Capture Layer (ESP32-CAM / PC webcam)**
    - Captures live video frames of the Tic-Tac-Toe board.
    - Streams frames to the server for processing.
2. **Processing Layer (Server / PC)**
    - Runs computer vision algorithms (OpenCV color thresholding or marker detection).
    - Converts raw frames into a structured **board state array (3×3)**.
    - Runs Tic-Tac-Toe game logic (AI opponent using minimax).
3. **User Interface Layer (Server-hosted UI)**
    - Displays live video feed with overlays showing detected tokens.
    - Allows human player to click on a square to place their move.
    - Shows game status (win, loss, draw).

4. **Communication Layer (Server ↔ ESP32 ↔ MSP430)**
    - Server sends compact command packets (source square, destination square, token type).
    - ESP32 relays packets to MSP430 via UART/SPI.
    - Includes error detection and acknowledgments.
5. **Control Layer (MSP430 + PCA9685)**
    - MSP430 parses commands and computes inverse kinematics.
    - Sends PWM signals via PCA9685 to servos.
    - Executes smooth pick-and-place trajectories.
6. **Feedback Layer (Telemetry)**

- MSP430 reports servo status and move completion back to ESP32.
- Vision pipeline verifies token placement.
- Logs are displayed in UI for debugging and reliability.

4. System-Level Testing

- **Metrics of success**:
    - ≥95% accuracy in token placement.
    - ≤2 seconds latency from user move to robot response.
    - Robust error handling: system detects and reports ≥90% of misreads or servo errors.
- **Tests**:
- Play multiple full games against human opponents.
- Stress test communication with mock rapid moves.
- Evaluate polished aspect (motion mechanism): smooth, repeatable trajectories, no jitter.
- Verify telemetry logs match actual system state during full games.