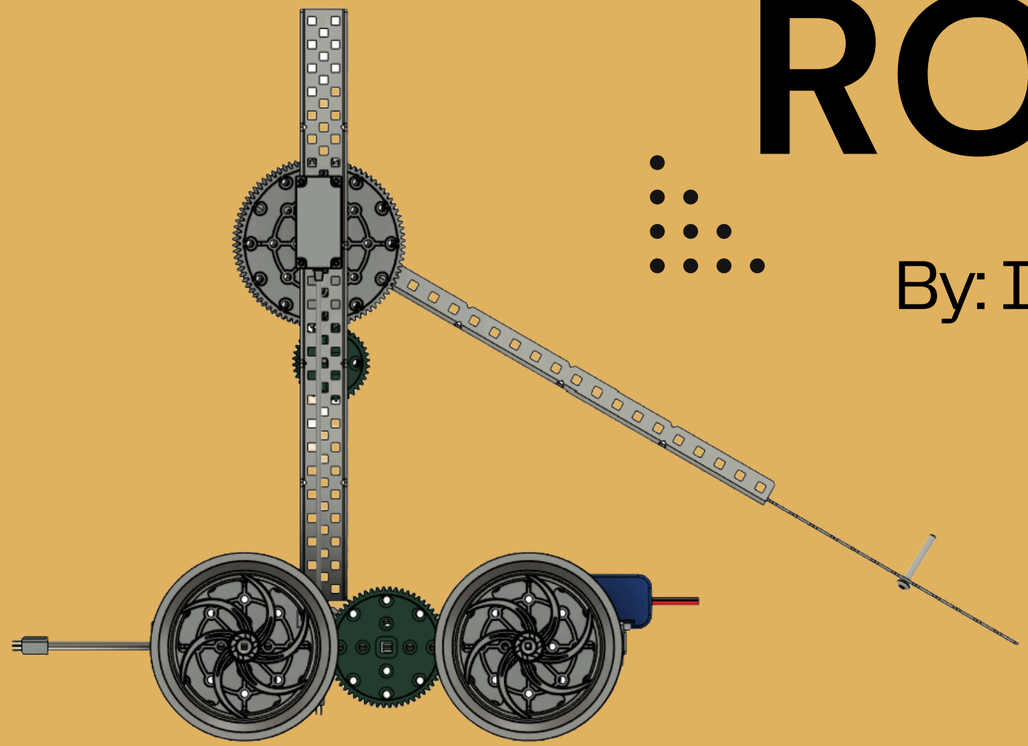
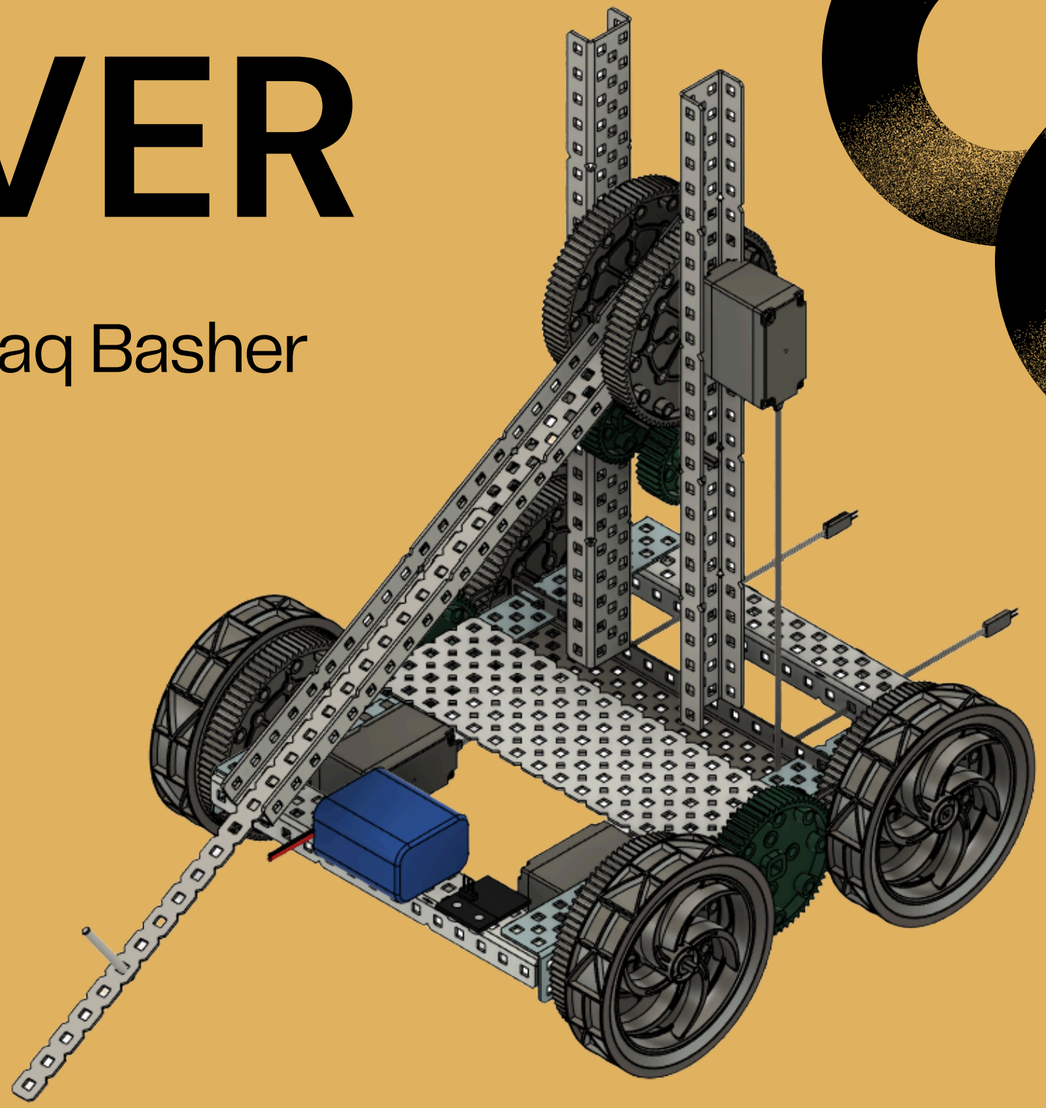
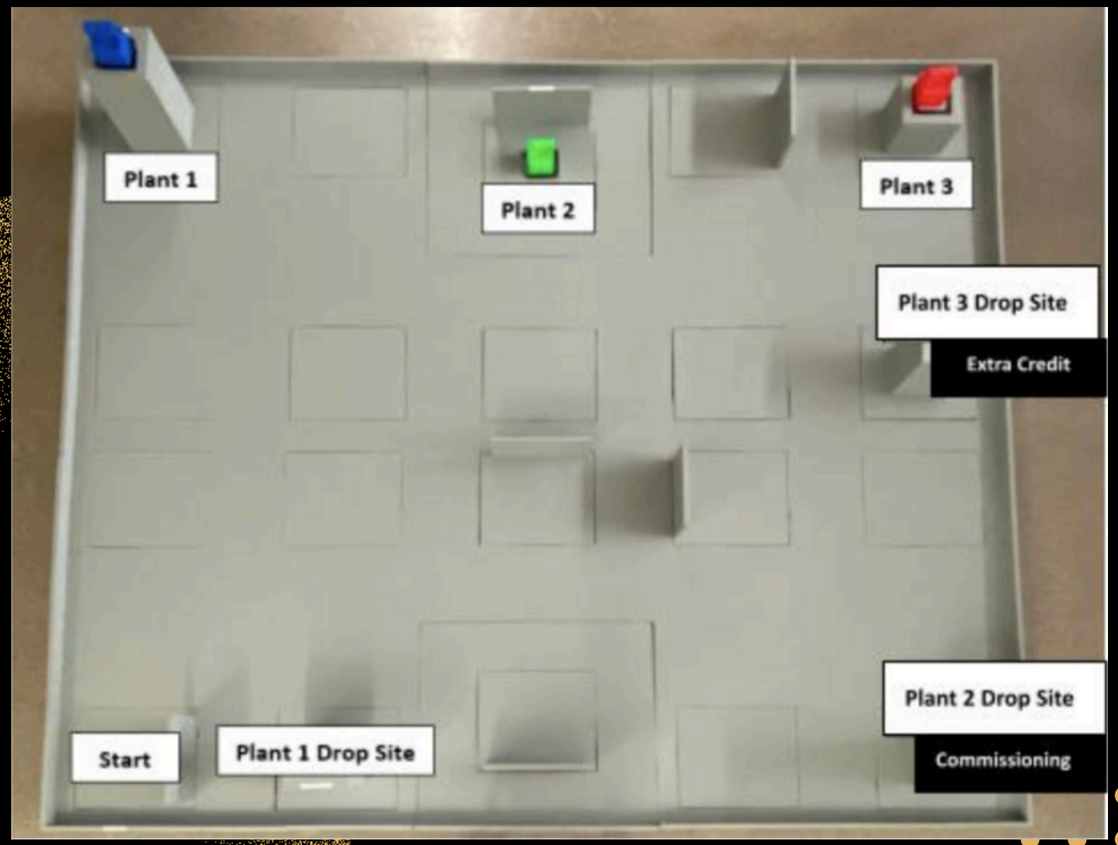
AUTONOMOUS ROVER

By: Ishraq Basher





SYSTEM OBJECTIVE & CONSTRAINTS



Navigation Course

Primary Objective

Autonomously scan, acquire, and relocate plant payloads.

Envrionmental Constraint

Dynamically detect and avoid sudden obsctacles in tight corridors

Navigation Constraint

Operate 100% autonomously with zero human telemetry, remote control, or pre-programmed waypoints.

Actuation Constraint

Physical claw manipulation must not block or pause the core navigational software loop

HARDWARE ARCHITECTURE

Compute Module

Arduino Uno paired with an Adafruit Motor Shield V2 for multi-motor I2C communication.

Perception Sensors

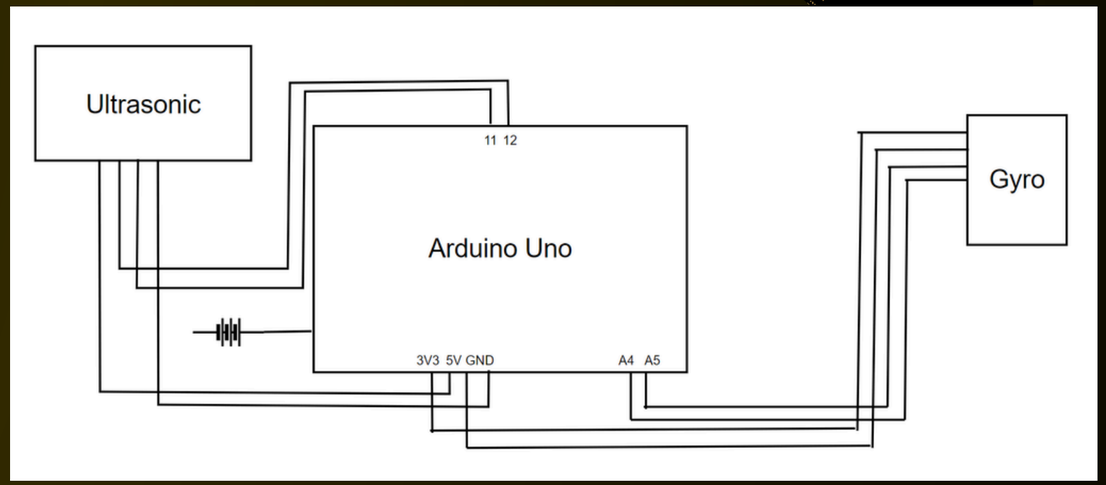
HC-SR04 Ultrasonic sensor utilized for spatial mapping and triggering distance-based state transitions.

Evasive Sensors

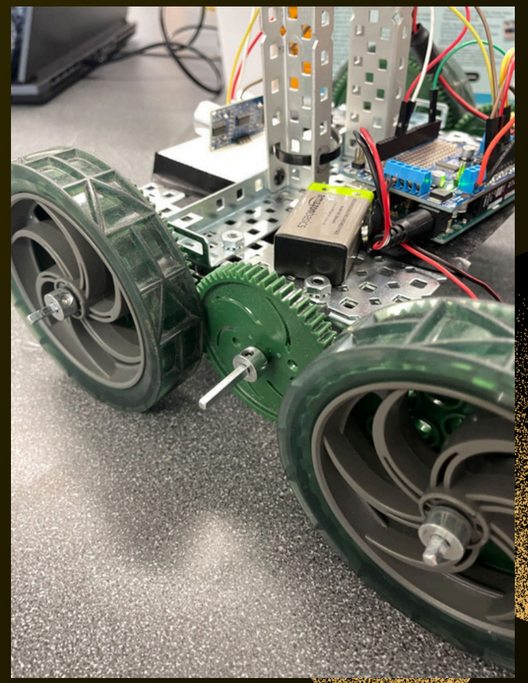
MPU-6050 Gyroscope utilized to calculate precise heading adjustments during obstacle avoidance.

Drivetrain & Actuation

4-wheel differential drive combined with a custom-geared VEX robotic arm for payload acquisition

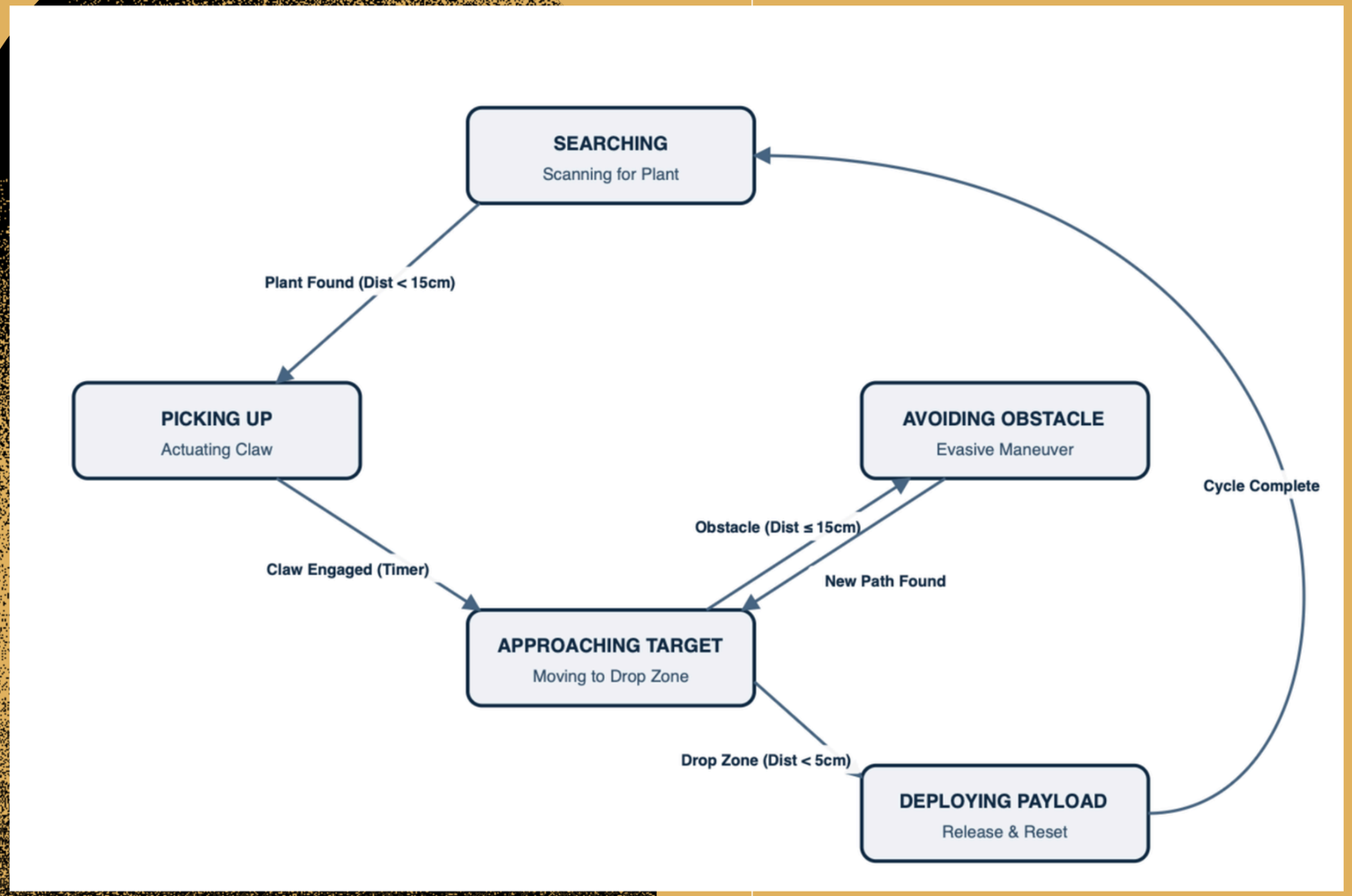


Schematic Diagram



Gear Train of the Wheels

SOFTWARE ARCHITECTURE



Finite State Machine

- Separated physical manipulation (PICKING UP, DEPLOYING) from navigation (APPROACHING, AVOIDING)
- Designed an active feedback loop for evasive maneuvers.
- Ensures the system never hangs or gets stuck in a “blind” execution phase.

Firmware Implementation

```
void loop() {
  int frontDist = readPing();

  // Linear, hardcoded sequence triggered by proximity
  if (frontDist > 20 && frontDist < 30) {

    // Stop drivetrain
    motor1->run(RELEASE);
    motor2->run(RELEASE);
    motor4->run(RELEASE);

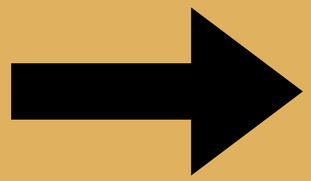
    // Actuate claw
    motor3->setSpeed(250);
    motor3->run(BACKWARD);

    // CRITICAL FLAW: Microcontroller is blocked/blind for
1.5s
    delay(1500);
    motor3->run(RELEASE);

    // Hardcoded navigation
    motor1->run(BACKWARD);
    motor2->run(BACKWARD);
    motor4->run(BACKWARD);

    // CRITICAL FLAW: Cannot detect sudden obstacles for 2s
    delay(2000);

    // ... [further blocking steps omitted] ...
  }
}
```



```
switch (currentState) {

    case SEARCHING:
      scanEnvironment();

      // plant detected close by, lock on and grab it
      if (dist > 0 && dist <= 15.0) {
        stopMotors();
        currentState = PICKING_UP;
        stateStartTime = millis(); // start claw timer
        Serial.println("switching to PICKING_UP");
      }
      break;

    case PICKING_UP:
      closeClaw();

      // let the claw finish closing before we try to drive
      if (millis() - stateStartTime >= CLAW_ENGAGE_TIME) {
        stopClaw();
        currentState = APPROACHING_TARGET;
        Serial.println("switching to APPROACHING_TARGET");
      }
      break;

    case APPROACHING_TARGET:
      driveForward();

      // we hit the drop zone
      if (dist > 0 && dist <= 5.0) {
        stopMotors();
        currentState = DEPLOYING_PAYLOAD;
        stateStartTime = millis();
        Serial.println("switching to DEPLOYING_PAYLOAD");
      }
      // something jumped in the way, time to dodge
      else if (dist > 5.0 && dist <= 15.0) {
        stopMotors();
        currentState = AVOIDING_OBSTACLE;
        targetHeading = currentHeading + 90.0; // calculate
a 90 deg evasive turn
        Serial.println("switching to AVOIDING_OBSTACLE");
      }
      break;

    case AVOIDING_OBSTACLE:
      evasiveManeuver();

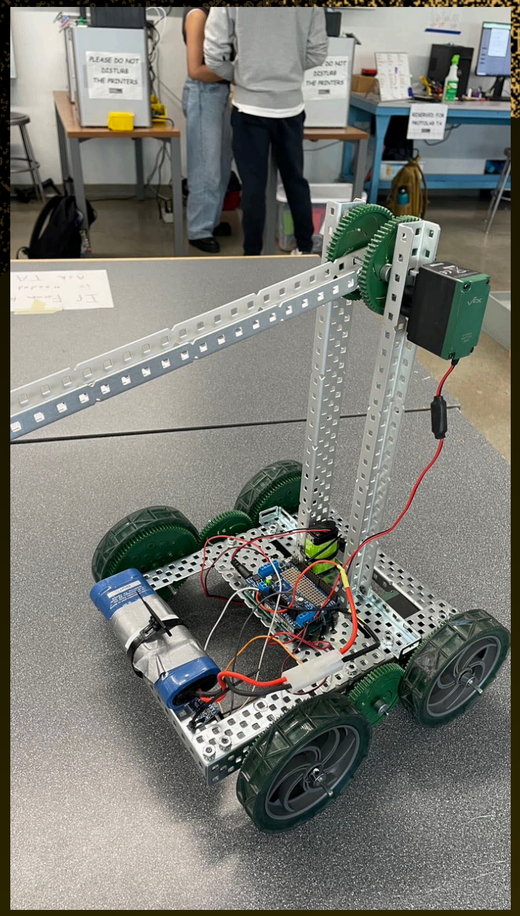
      // check if we finished our evasive turn yet (5 deg
tolerance)
      if (abs(currentHeading - targetHeading) < 5.0) {
        stopMotors();
        currentState = APPROACHING_TARGET;
        Serial.println("path clear. back to
APPROACHING_TARGET");
      }
      break;

    case DEPLOYING_PAYLOAD:
      openClaw();

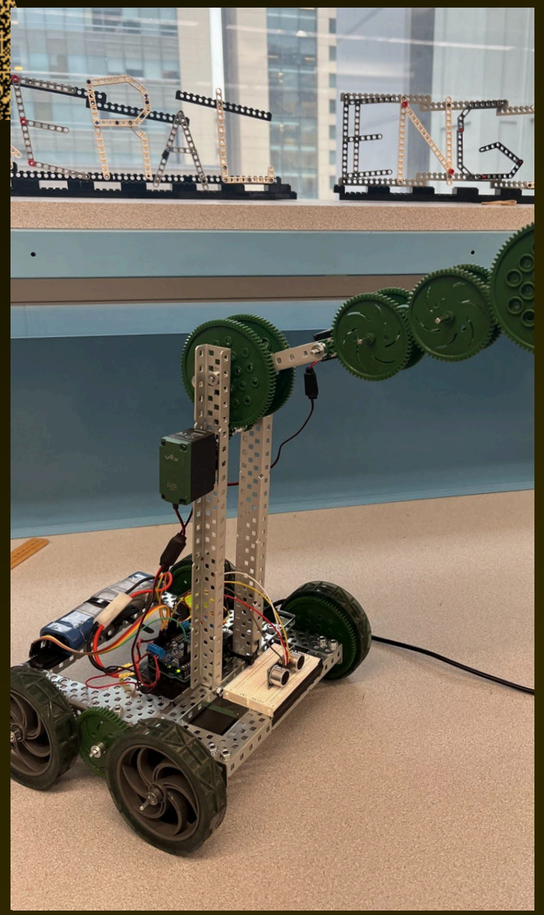
      // wait for claw to fully open
      if (millis() - stateStartTime >= CLAW_RELEASE_TIME) {
        stopClaw();
        // job's done, start the loop over
        currentState = SEARCHING;
        Serial.println("cycle complete. back to SEARCHING");
      }
      break;
  }
```

- Stripped out standard delay() functions that blind the robot's sensor polling.
- Implemented millis() for timer-based claw actuation, allowing simultaneous sensor reading
- Event-driven architecture allows for easy integration of future states.

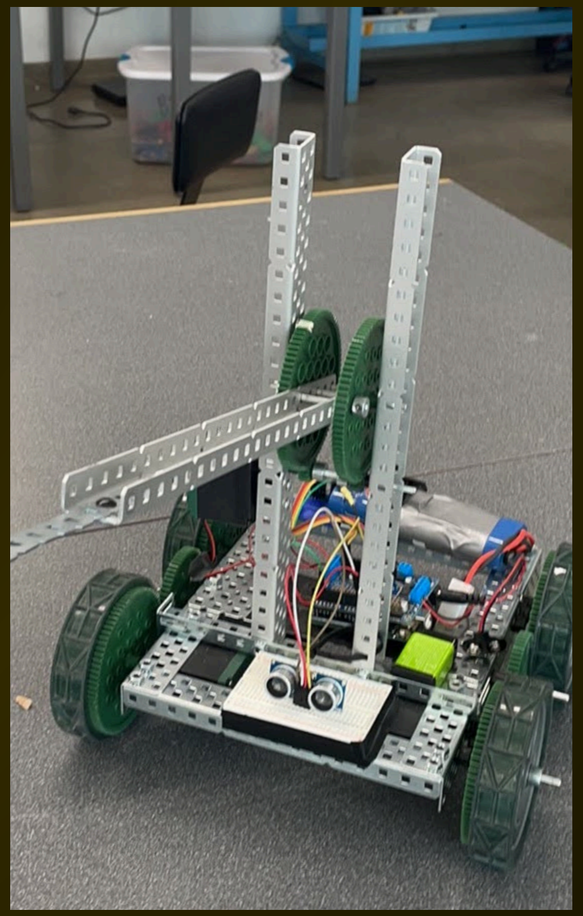
HARDWARE/SOFTWARE DEBUGGING



Version 1



Version 2



Final Version

- **Failure**: The heavy steel claw extended the center of gravity past the front axle, causing rear-wheel lift and massive traction loss.
- **Investigation**: Attempted to patch the issue via software (motor PWM adjustments), but identified the root cause as a physics limitation, not a code bug.
- **Solution**: Redesigned chassis weight distribution. Relocated the battery pack and mainboard to the rear chassis as a counterweight.
- **Result**: Restored 4-wheel differential traction, allowing the FSM navigation logic to execute maneuvers accurately.

AUTONOMOUS DELIVERY CYCLE

