Kevin Wang

kevinw041231@gmail.com
github.com/kevinjunkaiwang
linkedin.com/in/kevin-junkai-wang/

ROS 2 Robot Arm Controller

- **Tools Used:** ROS 2 Humble, Python, OpTas, CasADi, KUKA iiwa, Franka Emika Panda
- Robot Arm Controller is a ROS 2-based platform for controlling robotic arms, offering fundamental motion capabilities including:
  1. Set individual or multiple joint positions
  2. Execute straight-line Cartesian and twist-based (screw theory) motions
  3. Return robot to a predefined home joint configuration
  4. Ensure motion feasibility using task space reachability validation

---

- The Robot Arm Controller enables high-level control of a robot arm via mission files, which calls modular interfaces that perform lower-level actions, showing in Figure 1

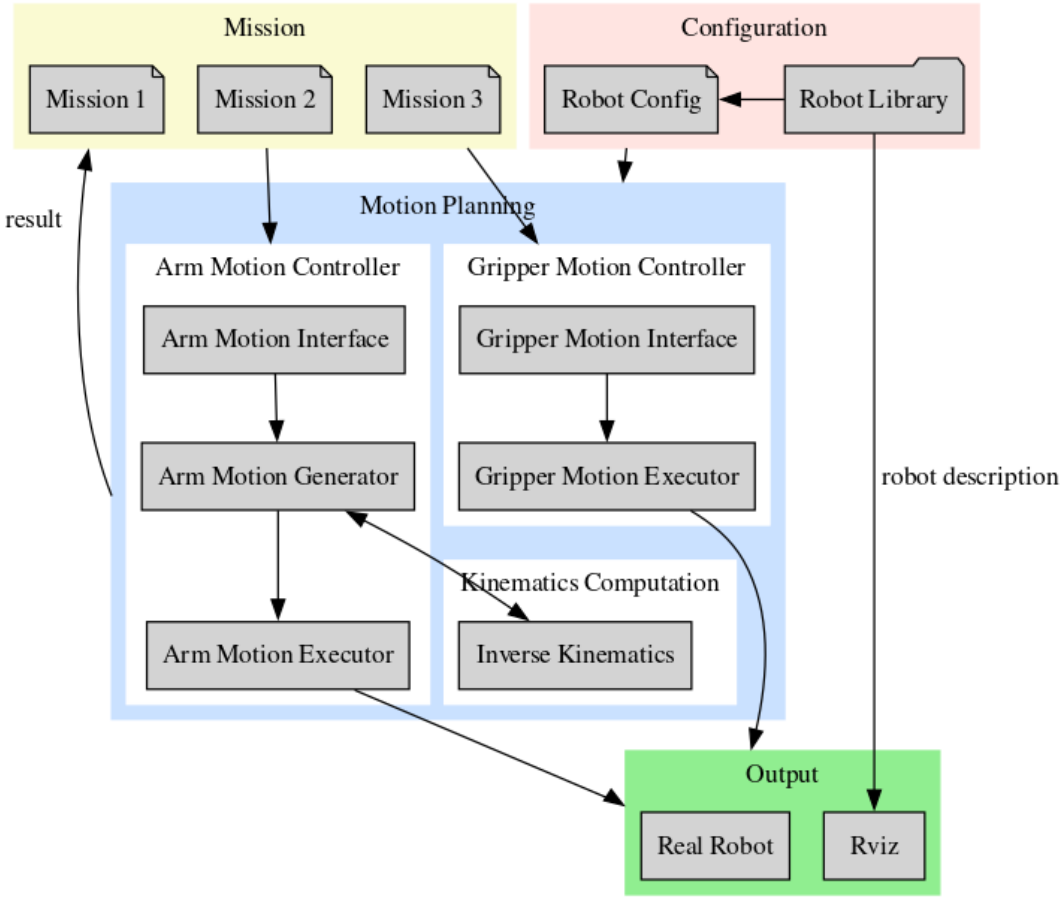

Figure 1: ROS 2 Controller System View

- **Mission**
  - Mission files are used to plan a series of movements for robot arm to execute
  - A high level control which only calls the functions from interfaces and let the rest of code perform their work
  - Avoid calling primary motion generation function or solver directly, which can be confusing and easy to make mistake
- **Motion Interface**
  - User-friendly Python functions for:
    - Trajectory planning with Cartesian or joint inputs
    - Gripper control (width, speed, force)
    - Twist motions using revolute and prismatic screw parameters
  - Execution Monitoring ensures motions are completed before proceeding

- **Motion Generator**
  - Generates smooth motions using:
    - Cartesian interpolation
    - Twist-based motion (rotation/translation along screw axis)
    - Joint-space interpolation
  - Validates motion with precomputed or on-the-fly workspace limits
  - Integrates internal inverse kinematics solver (OpTaS)
  - Publishes RViz2 visualizations for planned paths
- **Motion Executor**
  - Executes motion via `FollowJointTrajectory` ROS 2 action client
  - Sends gripper actions via `franka_ros2|` action interface
  - Monitors and signals motion completion to upstream nodes (Mission)
- **Configuration:**
  - Robot-specific configuration files define motion constraints (velocity, acceleration, jerk)
  - YAML-based setup enables quick customization without modifying URDF

- An optimization problem was formulated to generate smooth trajectory with OpTas and CasADi
- The objective includes minimizing joint motion effort, tracking a desired end-effector pose, and staying close to a nominal configuration, accounting for kinematic and dynamic limits
- **Decision Varibale:** Let $\mathbf{q}_t \in R^n$ denote the joint position and rotation vector at time step t, where $t = 1, 2, \ldots, T$

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_1^\top & \mathbf{q}_2^\top & \cdots & \mathbf{q}_T^\top \end{bmatrix}^\top \in R^{nT}$$

- **Objective Function:** The objective function minimizes a weighted sum of the following:
  - **Smoothness cost:** Penalizes large joint motions between timesteps.
  - **Rotation matching:** Optionally encourages tracking of a desired end-effector orientation.
  - **Null-space cost:** Keeps the joint configuration close to a nominal pose.

$$\min_{\mathbf{q}} \sum_{t=1}^{T} \left[10 \left\| \mathbf{q}_t - \mathbf{q}_{t-1} \right\|^2 + w_r \left\| R_t(\mathbf{q}_t) - R_t^{\text{des}} \right\|^2 + \left\| \mathbf{q}_t - \mathbf{q}_{\text{nominal}} \right\|^2 \right]$$

- **Constraints:** The optimization is subject to the following constraints for all $t \in [1, T]$:
  1. **Initial condition ($t = 0$):**
  $$\mathbf{q}_0 = \mathbf{q}_{\text{prev}}$$
  2. **End-effector pose matching (equality constraint):** Ensure the end effector positions ($\mathbf{p}_t^{\text{eef}}$) in task space always match with the way points ($\mathbf{p}_t^{\text{des}}$)
  $$\mathbf{p}_t^{\text{eef}}(\mathbf{q}_t) = \mathbf{p}_t^{\text{des}}$$
  3. **Joint limits:**
  $$\mathbf{q}_{\min} \leq \mathbf{q}_t \leq \mathbf{q}_{\max}$$
  4. **Joint velocity limits:**
  $$-\dot{\mathbf{q}}_{\max} \leq \dot{\mathbf{q}}_t \leq \dot{\mathbf{q}}_{\max}$$
  5. **Joint acceleration limits:**
  $$-\ddot{\mathbf{q}}_{\max} \leq \ddot{\mathbf{q}}_t \leq \ddot{\mathbf{q}}_{\max}$$
  6. **Joint jerk limits:**
  $$-\dddot{\mathbf{q}}_{\max} \leq \dddot{\mathbf{q}}_t \leq \dddot{\mathbf{q}}_{\max}$$
  7. **Position change limits (step-size constraint):** Provide some flexibilities when the constraints are strict, especially helpful during the twist trajectory generation
  $$-\Delta q_{\max} \leq \mathbf{q}_t - \mathbf{q}_{t-1} \leq \Delta q_{\max}$$

Automatic Waste Sorting System

- **Tools Used:** Python (TensorFlow, Flask), C++, HTTP, ESP 32
- Automatic Waste Recognition System aims to address the issue of the increase of recycle contamination, which focuses on developing a system capable of detecting and classifying waste items to direct users towards the correct bin for proper disposal
- The system will identify waste types using images captured by an ESP32 camera module, send the image via HTTP POST method to the Flask web server, as shown in the front-end section in Figure 2
- The captured image is processed on a Flask server using **MobileNetV2**, which then controls a motor to sort waste into the appropriate bin (mechanical sorting in progress). See the backend design in Figure 2

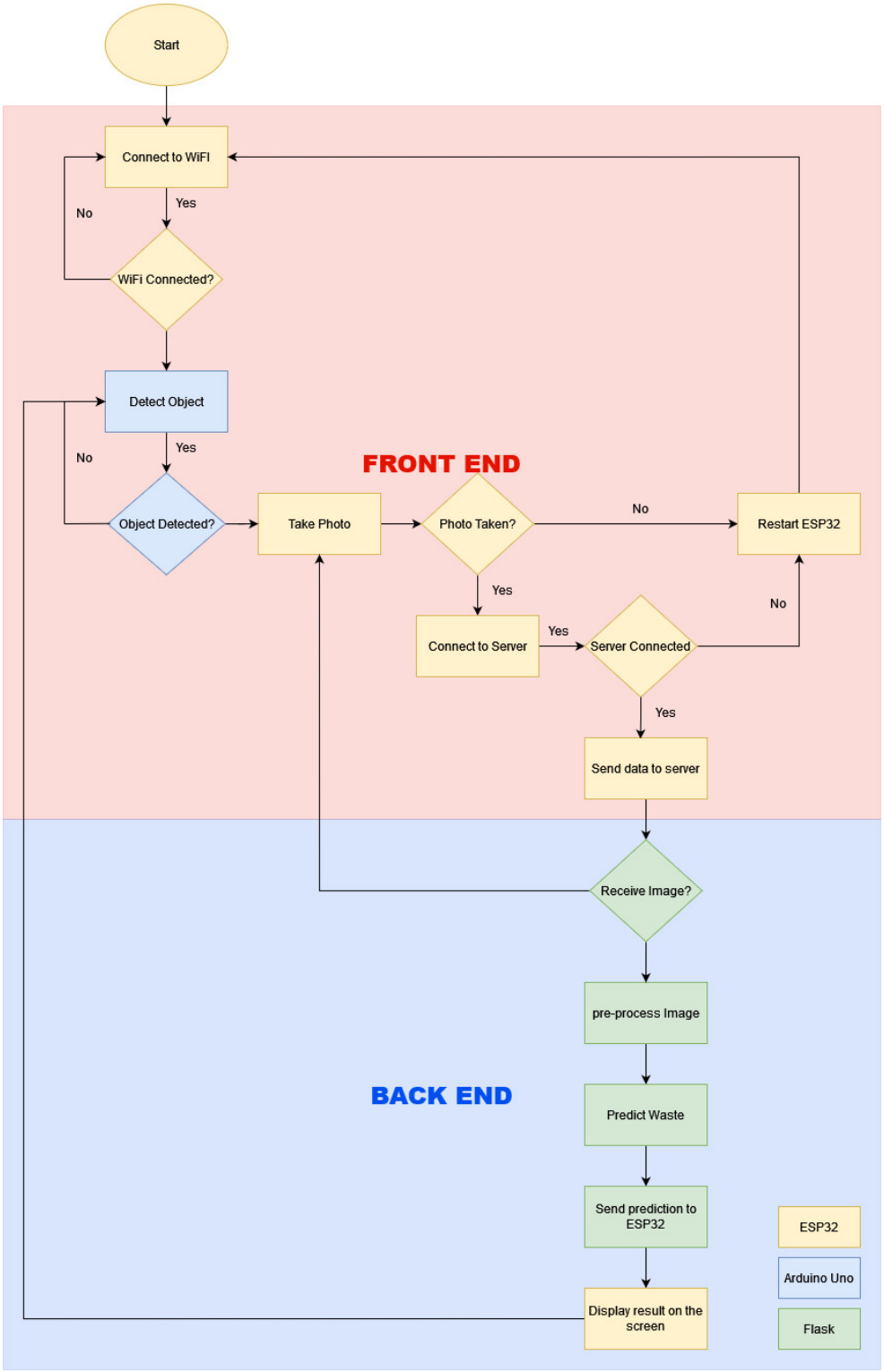

Figure 2: Flow Chart of Data Capture and Processing on Automatic Waste Sorting System

- Selected the **RealWaste** dataset (4,752 single-object images with plain backgrounds) to train the model due to its suitability for lightweight image classification. See example in Figure 3.

Figure 3: Screen Capture of Cardboard Images in RealWaste

- The accuracy and loss from the validation data set after 26 epochs with 32 batches stays at 86% and 48%, as shown in Figure 4

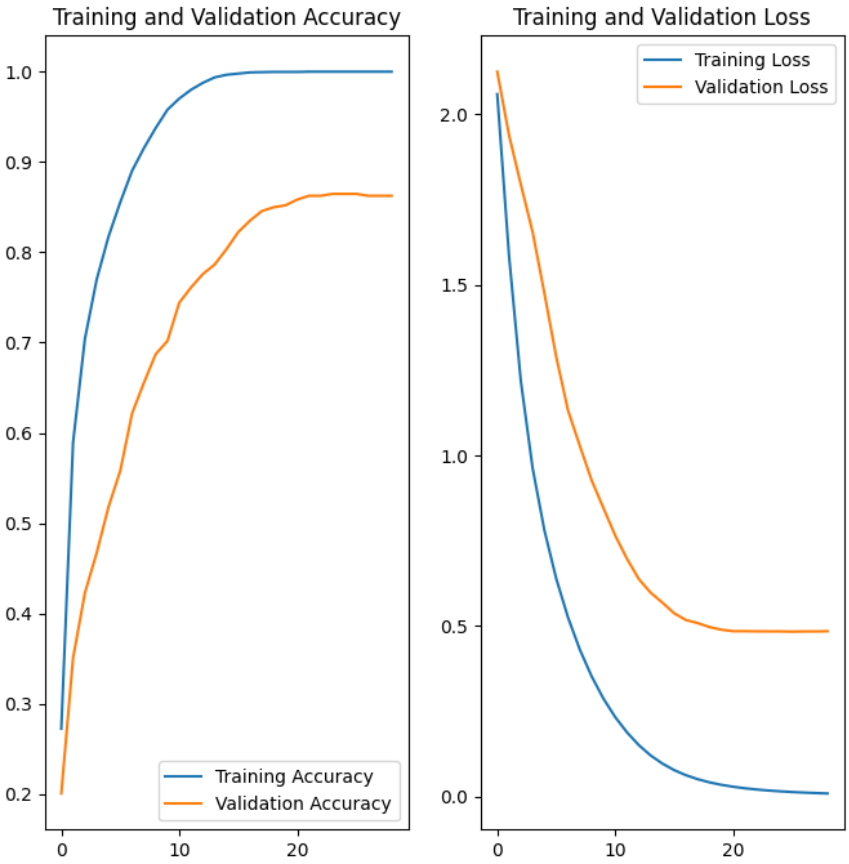

Figure 4: Accuracy and Loss on Training and Validation Data Set

Portable Multi-Function Device

- **Tools Used:** ESP32, C, EasyEDA, SolidWorks
- A portable ESP32 device that can display the current time & weather, act as a Pomodoro timer with alarm and connect to the internet
- Starting from an idea to the requirements analysis, component selection, schematic design, PCB layout, code development, debugging and installation, in which is currently in the code development phase
- **Electrical:**
  - Implemented backflow prevention design using a MOS and a diode in the USB To UART section, which prevents the reverse voltage to damage the components
- **Mechanical:**
  - Utilized SolidWorks to design a protective case, with the specific purpose of attaching the screen and protecting the PCB by adding support to the empty areas of the board, effectively preventing damage from the external environment
  - Selected C2 continuity for all fillets and left spaces for magnets to provide practical and aesthetic attributes
- **Software:** in progress

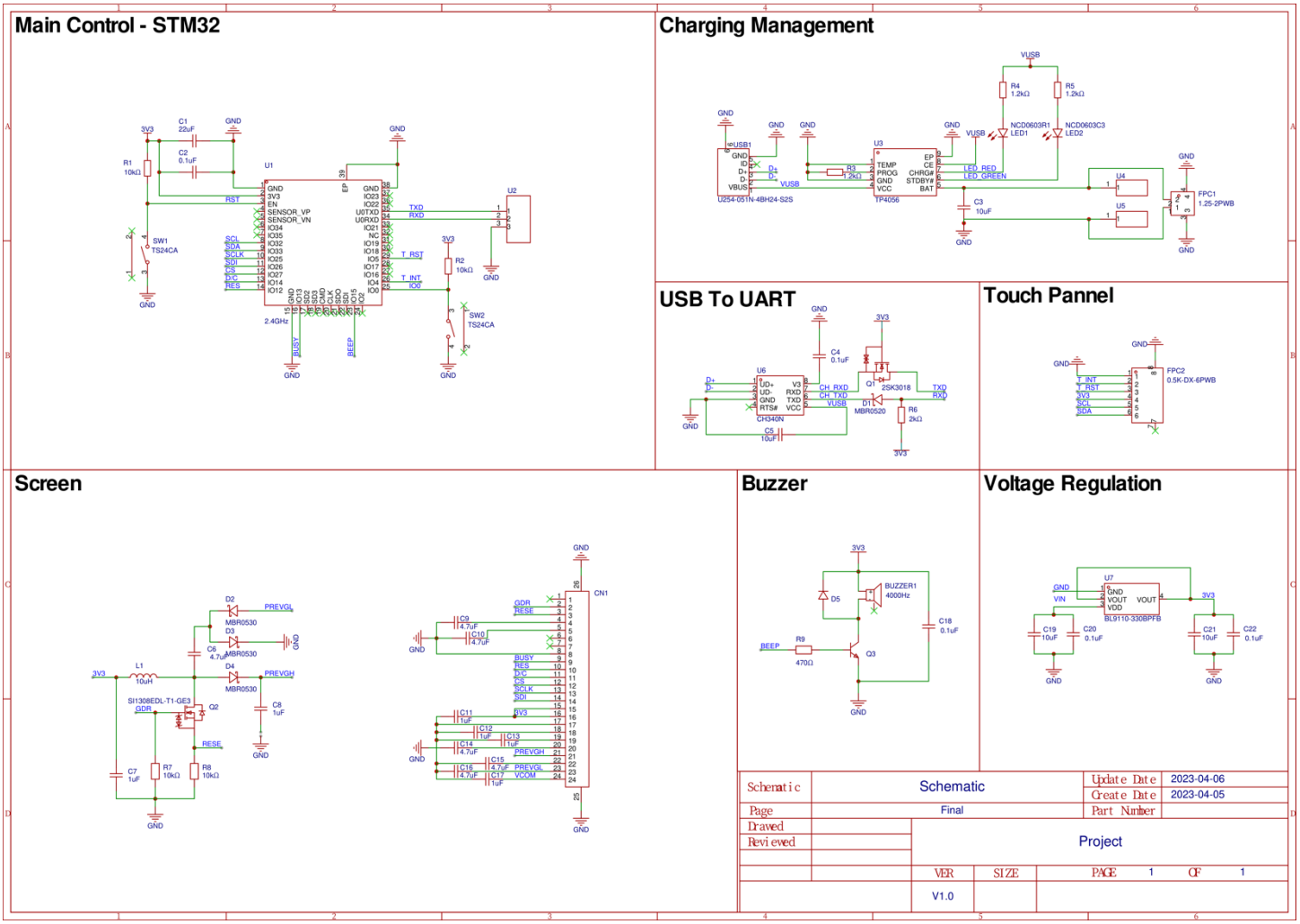

Figure 5: Portable Multi-Function Device Schematics

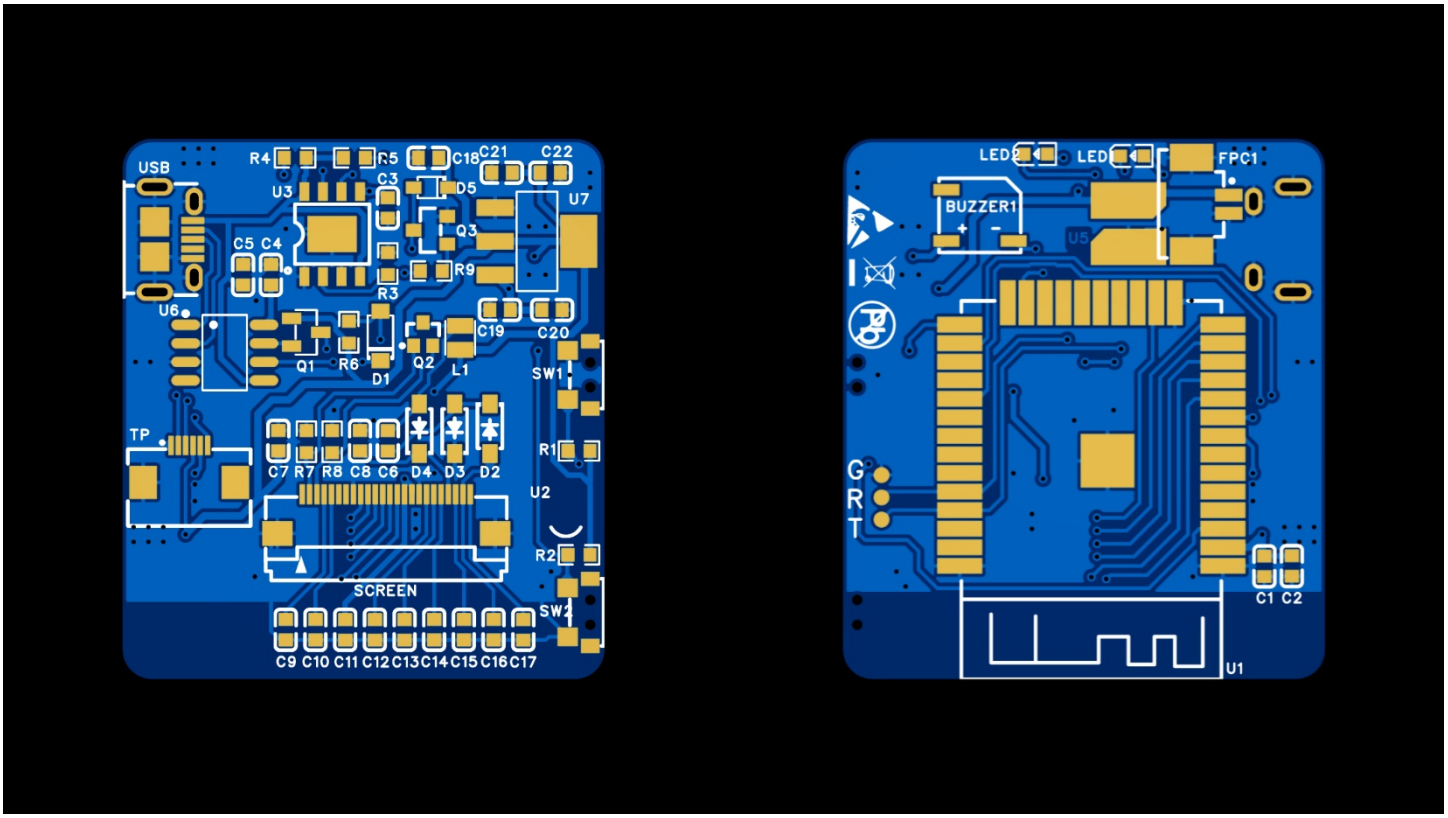

Figure 6: Portable Multi-Function Device Schematics PCB

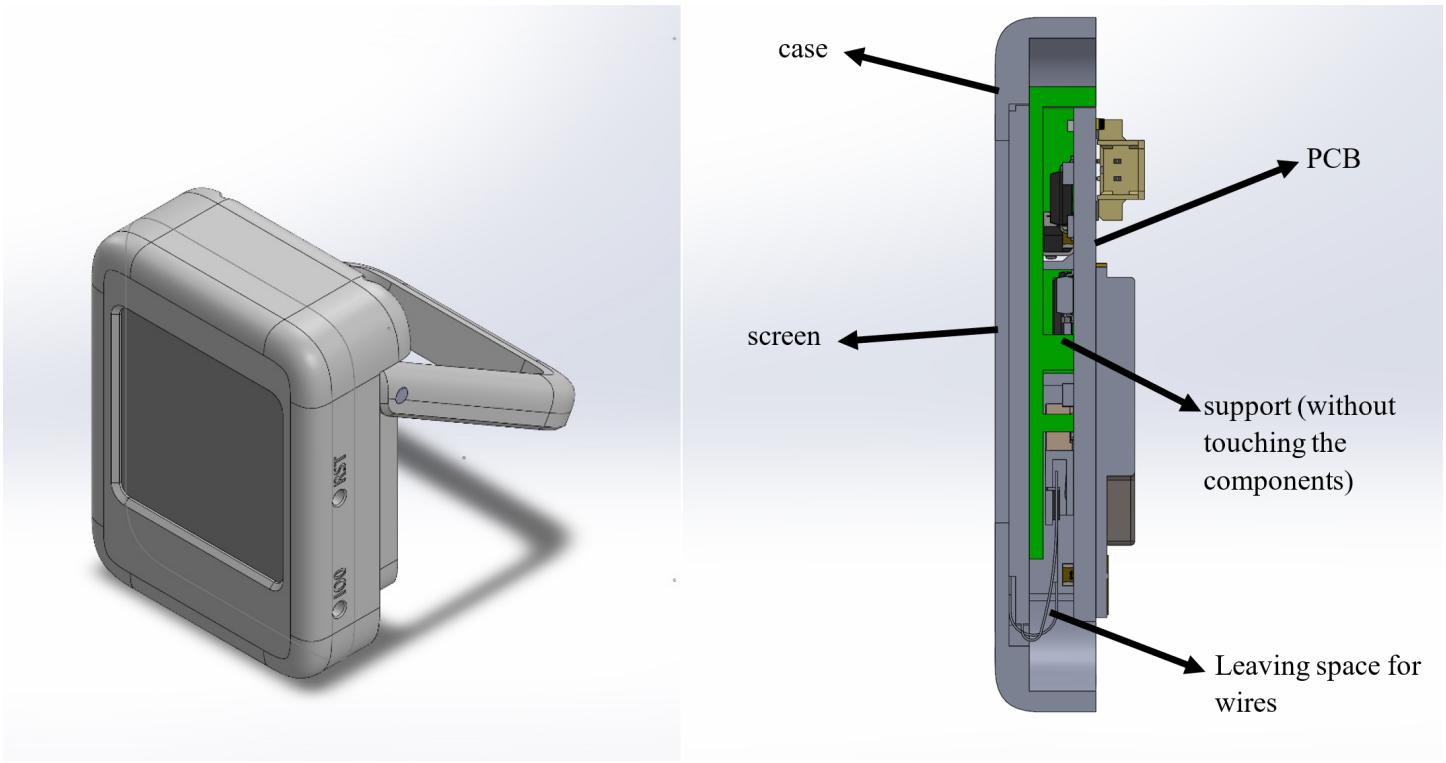

Figure 7: Portable Multi-Function Device Schematics Case Design & Structure

BreathEase Control Hub

- **Tools Used:** Arduino, C++, Python, OpenCV
- A hackathon project developed independently during Hack Western 10, aiming to support elders, disabilities, and patients with ALS (Amyotrophic Lateral Sclerosis)
- Utilized a humidity and temperature sensor to detect the humidity level released from the user's breath every second, reporting the status to the console and the built-in screen immediately
- Developed a Python program with OpenCV to track the movement of the left eyeball and measure the distance between the upper and bottom eyelids of the right eye, enabling the user to control the cursor on the laptop with only their heads and eyes
- Transferred the ASCIS data from Python program to Arduino through the serial port and converted them to characters for indicating if the user is in the camera, eventually displaying the result to the screen

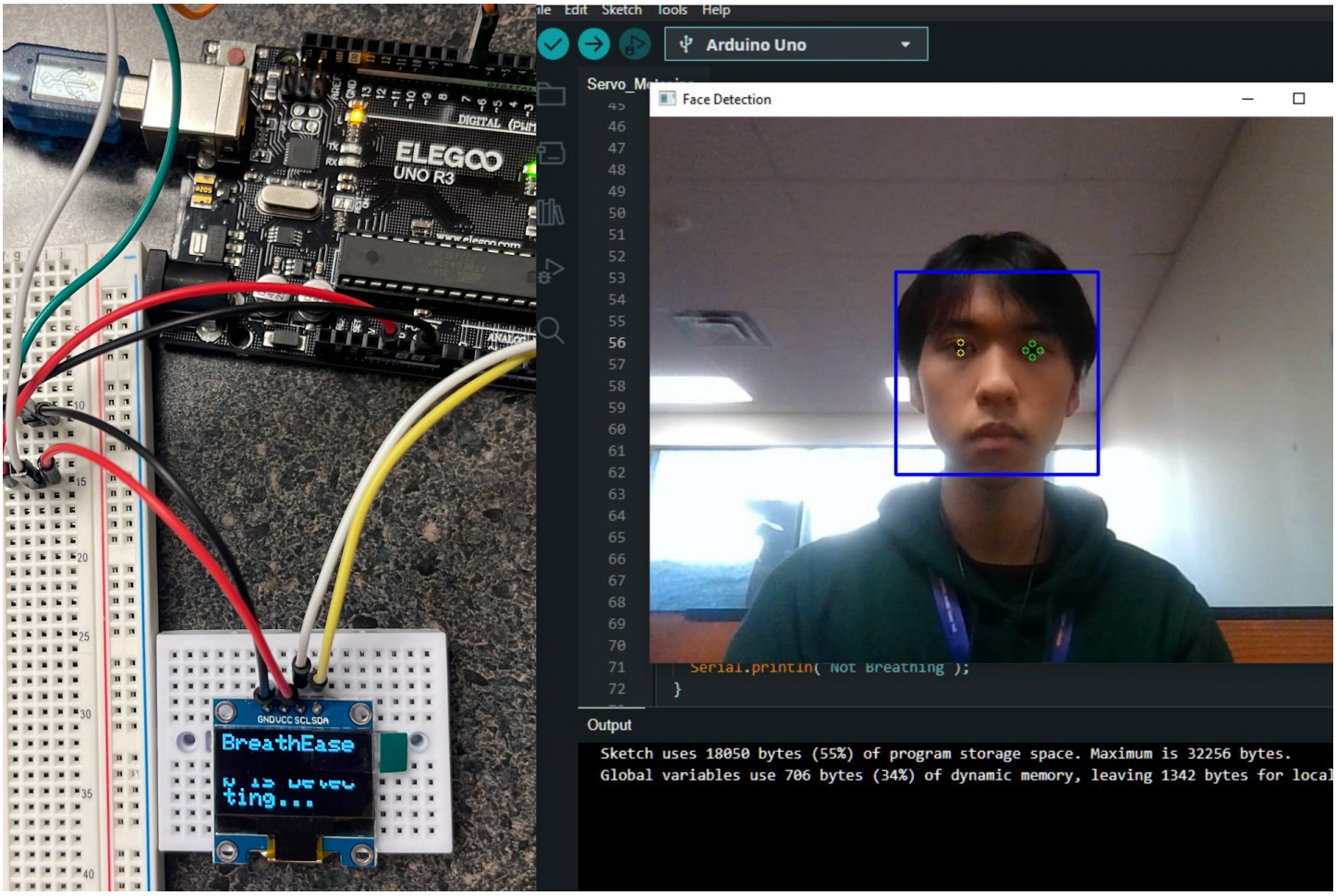

Figure 8: BreathEase Control Hub Operating Demonstration

Arduino IoT Panda Messenger

- **Tools Used:** Arduino, C++
- Utilized Arduino IoT 33 with the Arduino IoT Cloud to send and display messages on the attached OLED screen to my girlfriend during the long-distance relationship
- Equipped with a photoresistor for her to indicate whether she has seen the message by taping it, which will sync the status to the IoT cloud for me to access and confirm

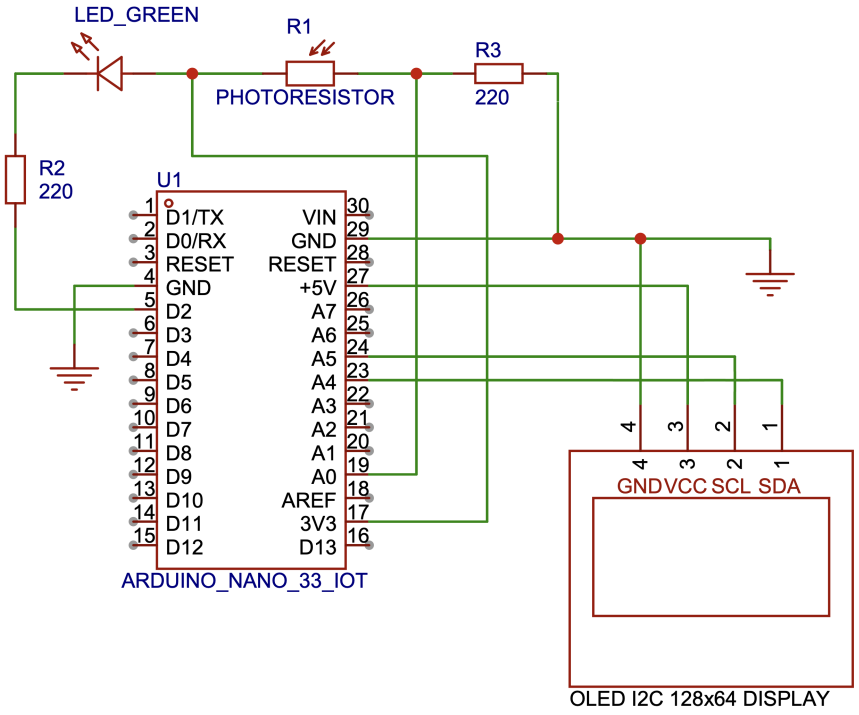

Figure 9: Arduino IoT Panda Messenger Schematics

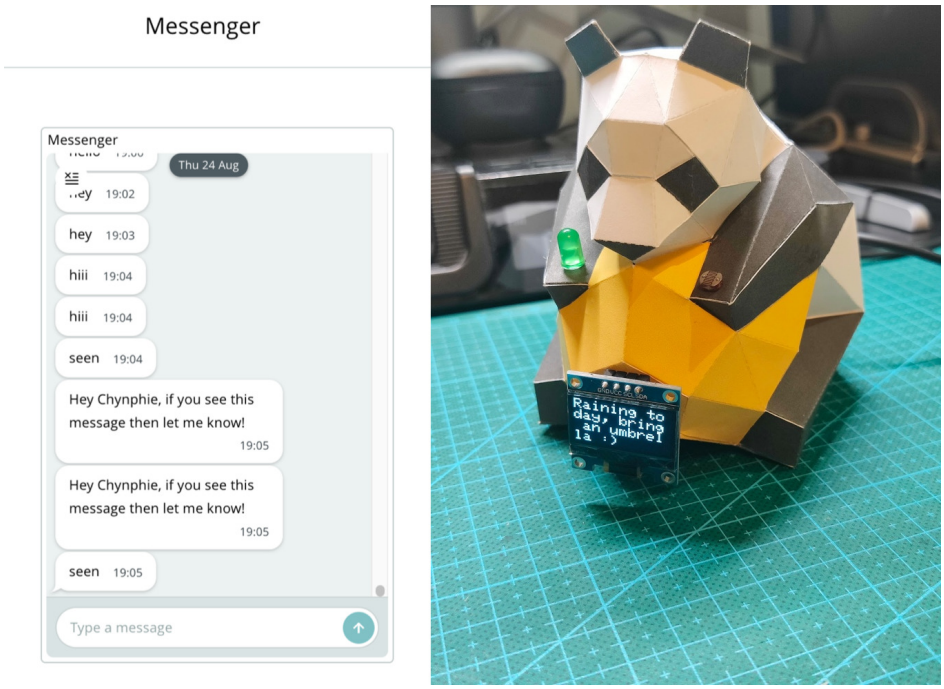

Figure 10: Arduino IoT Panda Messenger Operating Demonstratio

Quadro Pressure Gauge

- **Tools Used:** Altium Designer
- Designed to measure the gas pressure from the shooting BB, which simulates the saltation process on Mars
- Capability to use two different gauge sensor models
- Implemented a Zener diode for each sensor, protecting sensors from frequently dismounting
- Connected a 10 Ohms resistor on every data line to limit the current and perform impedance matching
- Large vias are used to solder copper tube transferring gases from compressor with enough spacing around for mechanical sealing

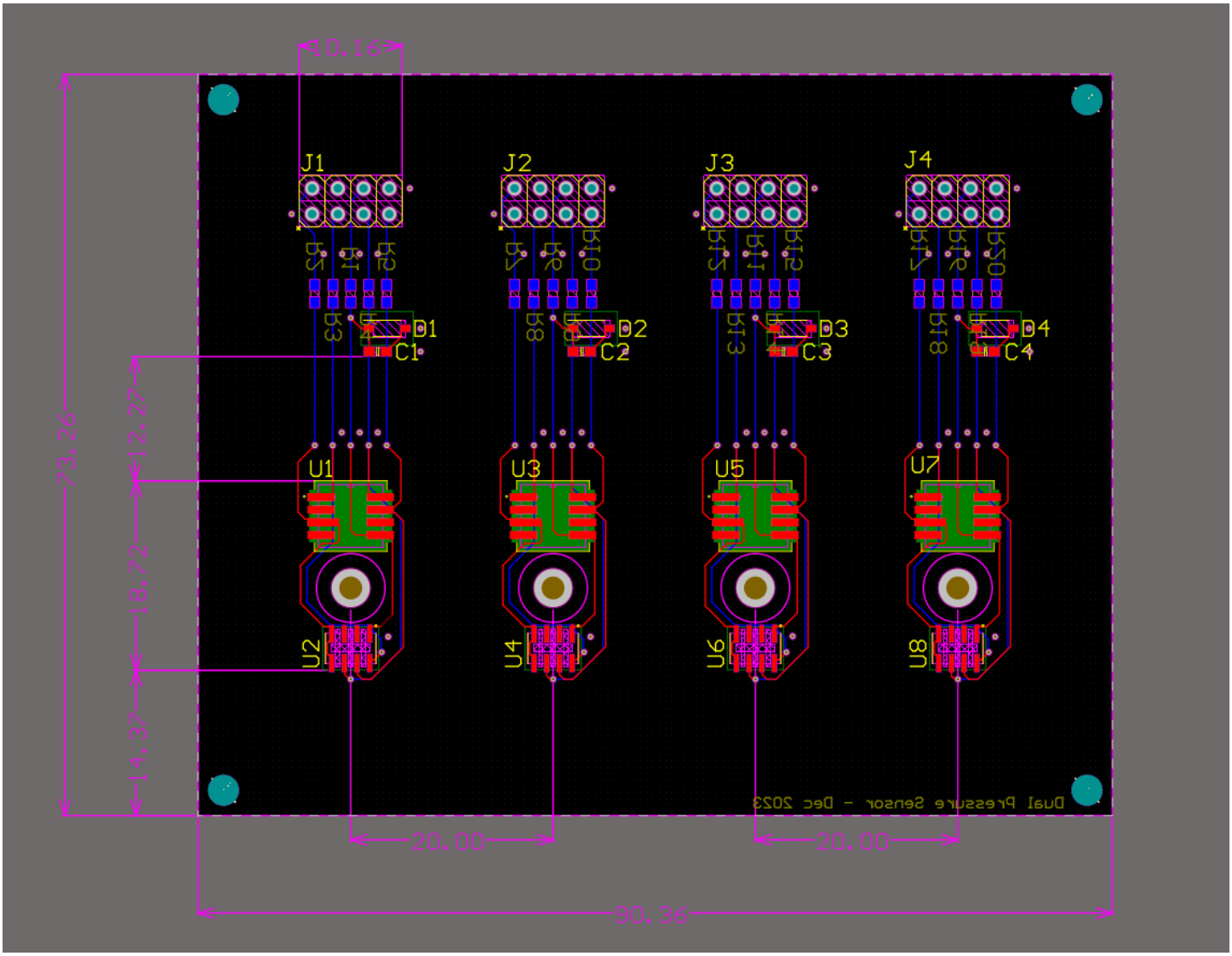

Figure 11: Quadro Pressure Gauge PCB Layout in Altium Designer

Canadian Rescue Assistance Prototype

- **Tool Used:** LEGO EV3, RobotC
- The robot simulated the automated rescue process under extreme environment using LEGO EV3 and RobotC as the mechanical and software design respectively
- Systematically designed the frame with careful consideration of the placement of motors and sensors (including IR, ultrasonic, gyro, and color sensors) to optimize the robot's functionality and ensure that all components function properly
- Programmed the robot to travel in a boustrophedon pattern to find the target patient (cup) and return to its start point, while avoiding obstacles along the way

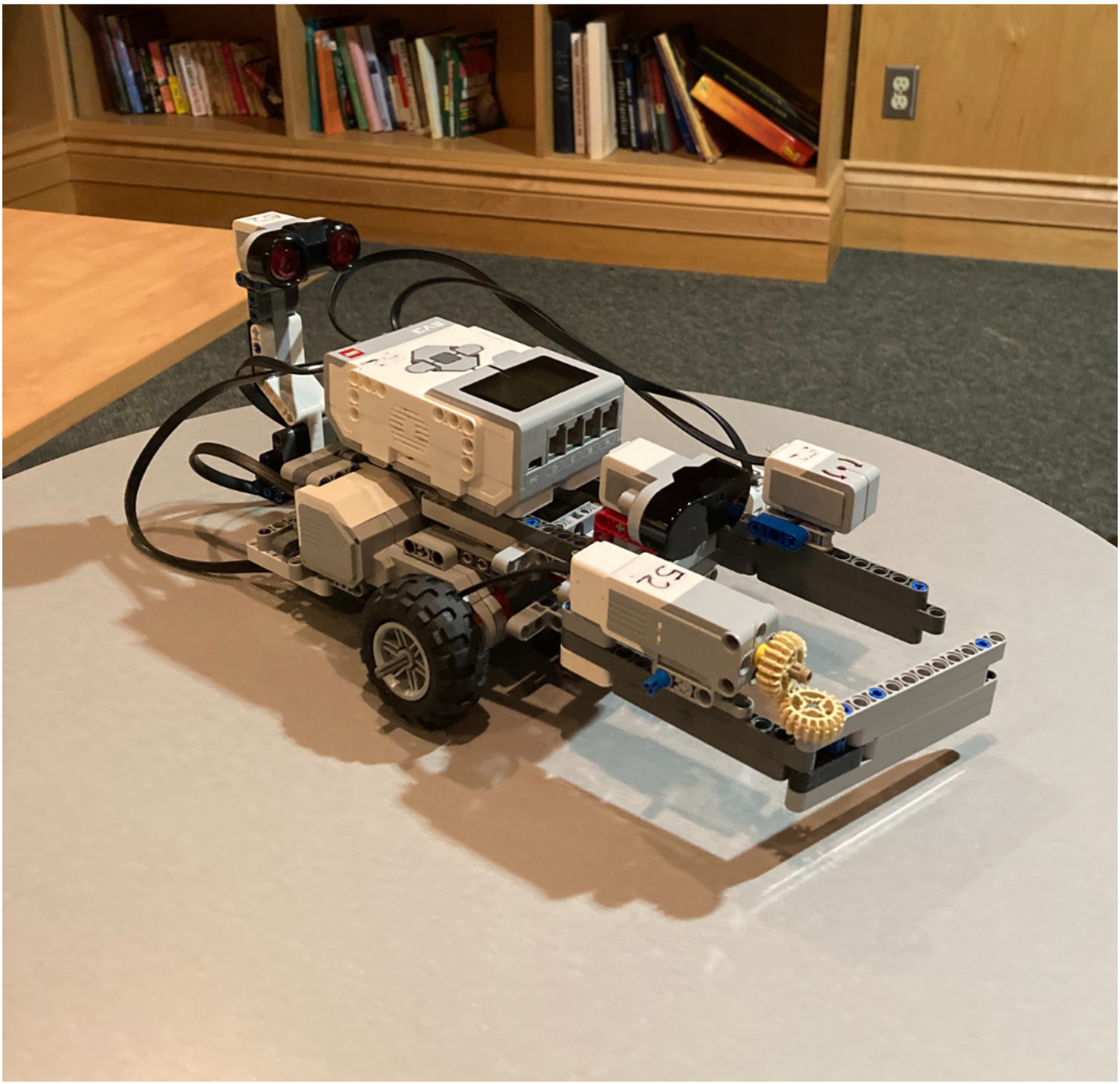

Figure 12: Enter Caption