Design Prototype

Haoyuan Xu
Computer Engineering Department
University of Florida
Gainesville, United States
haoyuan.xu@ufl.edu

Jingxuan Xiao
Computer Engineering Department
University of Florida
Gainesville, United States
j.xiao@ufl.edu

***Abstract*—This report outlines the design prototype of the Rover Master project, detailing the efforts in software and hardware development, architectural elements, and information handling systems. It includes the research and selection of models for monocular depth estimation and object detection, the design of the main control board PCB, and the development of internal systems such as the motion planning system and IMU fusion sensor.**

***Index Terms*—Rover Master, Monocular Depth Estimation, Object Detection, PCB Design, ROS2, IMU Fusion Sensor, Motion Planning System**

I. Introduction

The Rover Master project aims to develop an autonomous robot capable of navigating and interacting with its environment using advanced sensing and control systems. This pre-alpha build report details the efforts made in both software and hardware development, architectural design, and the implementation of internal systems that contribute to the robot's functionality.

II. Effort

A. Software

1) Breakdown:

- **20% - Evidentiary Research**
  - Evaluated YOLOv11 for object detection, Marigold for depth estimation, and Nav2 for navigation.
  - Assessed alternative solutions for real-time depth sensing, such as LiDAR.
- **40% - Interface Development**
  - Implemented ROS nodes for object detection and navigation.
  - Designed modular architecture for seamless communication between components.
- **30% - System Integration and Testing**
  - Debugged ROS communication and validated YOLOv11[1] and Nav 2 [2] functionalities.
  - Integrate YOLOv11 with ROS on Rasberry Pi 5
  - Visualized outputs using RViz and tested the system with different image inputs.
- **10% - Troubleshooting and Iteration**
  - Addressed ROS2 version incompatibility issues and runtime errors.
  - Refined design to improve system performance and scalability.

2) Preparatory Research and Evidence Gathering:

a) Object Detection:: We researched the latest object detection models and selected **Ultralytics YOLO v11** [1] for its speed, accuracy, and compatibility with our project's real-time requirements.

b) Depth Estimation:: We evaluated various approaches, including depth cameras, LiDAR, and monocular depth estimation models. **Marigold** [3] emerged as the best option initially due to its high-quality output and relatively low cost.

c) Navigation Tool:: We identified **Nav2** [2] as the most suitable navigation tool for ROS-based projects. Its capabilities for efficient path planning and obstacle avoidance made it an excellent fit for our needs.

3) Experimentation:

a) Object Detection::

- Initially tested YOLO v11 [1] on a local laptop with live video streams, demonstrating exceptional efficiency and accuracy.
- Successfully integrated the model with ROS on a Raspberry Pi, where it maintained similar performance levels for real-time detection.

b) Depth Estimation::

- Tested the Marigold model [3] locally, which produced high-quality results for single images. However, processing times (˜3 minutes per image) made it unsuitable for real-time use.
- Planned to swtich to LiDAR, which offered faster feedback, aligning better with our real-time depth estimation requirements.

c) Navigation Tool::

- Ran Nav2 [2] on a local laptop, successfully simulating point-to-point navigation on a predefined map.
- Encountered challenges when testing Nav2 [2] on the Raspberry Pi due to ROS2 version incompatibility.

4) Design:

a) Modular Architecture:: Each core component (object detection, depth estimation, and navigation) is implemented as an independent ROS node, ensuring modularity for easier debugging, updates, and scalability.

b) Data Flow::

- **Camera Node**: Streams real-time frames to the object detection node.

- **Object Detection Node**: Processes frames to detect objects and sends bounding box data to the navigation module.
- **Depth Estimation**: LiDAR feeds distance data for detected objects, enabling precise path adjustments.

c) Integration with ROS::

- ROS topics and services facilitate communication between nodes.
- **TF2** ensures proper alignment of object detection outputs with navigation tasks.

d) Visualization:: **RViz** is used to display detected objects, navigation paths, and robot states for debugging and performance analysis.

e) Real-Time Considerations:: Prioritized lightweight computations to accommodate the Raspberry Pi's limited processing power. Non-essential tasks like logging and visualization were offloaded to avoid system bottlenecks.

5) Implementation:

a) Object Detection::

- Integrated YOLO v11 [1] with ROS by creating a custom node.
- The node subscribes to the camera input topic, processes frames, and outputs bounding boxes for detected objects.

b) Depth Estimation::

- Tested the Marigold model [3] on images of varying quality but concluded it was unsuitable for real-time use.
- Decided to transit to LiDAR which will be implemented as a separate ROS node for real-time depth sensing.

c) Navigation Tool::

- Installed ROS2 and Nav2 [2] on a local laptop to test navigation in a simulated map environment.
- Encountered challenges when transitioning this setup to the Raspberry Pi.

6) Failed Attempts and Difficulties:

a) ROS2 Version Incompatibility:: The ROS2 version used on the local laptop (**Humble Hawksbill**) differed from the version on the Raspberry Pi (**Rolling Ridley**), which caused compatibility issues. Rolling's development-focused nature resulted in frequent build failures and runtime errors.

b) Initial Depth Estimation Model Limitations:: The Marigold model's [3] slow processing times made it unsuitable for real-time use. Switching to LiDAR resolved the issue but introduced new setup and calibration challenges.

c) Lack of Familiarity with ROS:: As a newcomer to ROS, understanding its ecosystem and troubleshooting issues required significant effort. Extensive research and experimentation were necessary to overcome these challenges.

d) Integration Challenges:: Debugging communication between ROS nodes, especially across devices (laptop and Raspberry Pi), was time-intensive. Careful testing of individual modules prior to integration resolved many issues.

B. Hardware

1) Breakdown:

- **30% - Evidentiary Research**
  - Most of the time are spent in researching on battery balancer.
  - Tried to reverse engineer existing battery balancing board.
  - Researched on the schematic design of Power Delivery circuit.
- **10% - Interface Development**
  - Consider what communication protocol and interface between main control board and on board computer.
  - The interface for power delivery.
- **20% - System Integration and Testing**
  - The oscillator for MOSFET driver is tested on breadboard (Figure 1).
  - Battery balancing circuit is tested on LTspice.
- **40% - Troubleshooting and Iteration**
  - Oscillator is iterated to ensure a good square pulse is generated.
  - Many troubleshooting are done to try to simulate battery balancer.

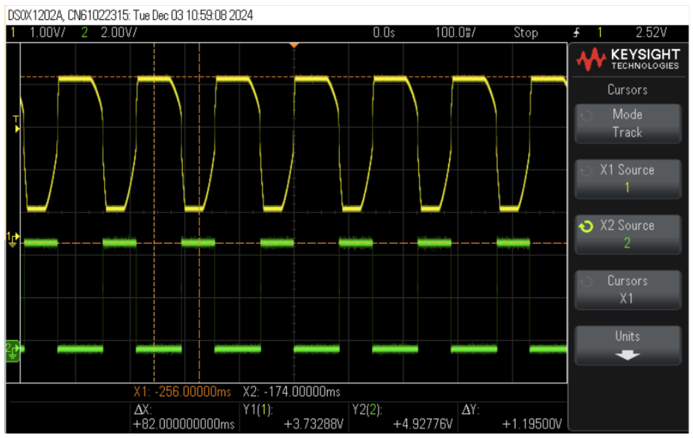
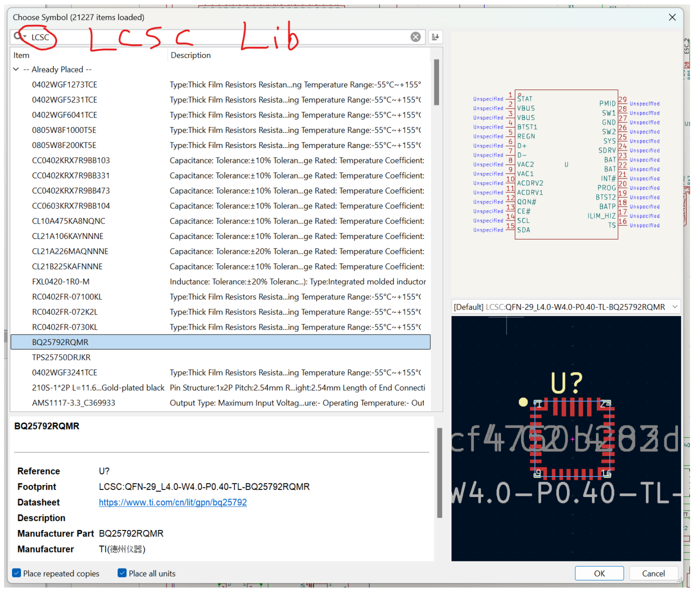


Fig. 1. signal from oscillator(Top) and Schmitt Trigger regulated square pulse(Bottom)

2) Web Scraper for Building KiCAD Library: We utilized a web scraper (https://github.com/zhangyx1998/easyeda-scraper) to build a KiCAD library by scraping electronic components information from LCSC.com. The program extracts critical data from the API call and formats it into a component library compatible with KiCAD. This tool ensures that every part in our design corresponds to an existing electronic component available for purchase online. The library will grow as the user scrap more components with the program (Figure 2).

a) Difficulties:

- The APIs to LCSC.com are not directly exposed to users. We had to use the developer tool of Chrome browser to capture potential requests.
- The component library structure is not clearly documented and we had to look at the exisitng libraries to determine what information are needed.



Choose Symbol (21227 items loaded)

LCSC LCSC Lib

Item	Description
-- Already Placed --	
0402WGF1273TCE	Type:Thick Film Resistors Resistan...ng Temperature Range:-55°C~+155°
0402WGF5231TCE	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
0402WGF6041TCE	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
0805W8F1000T5E	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
0805W8F200KT5E	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
CC0402KRX7R9BB103	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
CC0402KRX7R9BB331	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
CC0402KRX7R9BB473	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
CC0603KRX7R9BB104	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
CL10A475KA8NQNC	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
CL21A106KAYNNNE	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
CL21A226MAQNNNE	Capacitance: Tolerance:±20% Toleran...ge Rated: Temperature Coefficient:
CL21B225KAFNNNE	Capacitance: Tolerance:±10% Toleran...ge Rated: Temperature Coefficient:
FXL0420-1R0-M	Inductance: Tolerance:±20% Toleranc...): Type:Integrated molded inductor
RC0402FR-07100KL	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
RC0402FR-072K2L	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
RC0402FR-0730KL	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
BQ25792RQMR	
TPS25750DRJKR	
0402WGF3241TCE	Type:Thick Film Resistors Resista...ing Temperature Range:-55°C~+155°(
210S-1*2P L=11.6...Gold-plated black	Pin Structure:1x2P Pitch:2.54mm R...ight:2.54mm Length of End Connecti
AMS1117-3.3_C369933	Output Type: Maximum Input Voltag...ure:- Operating Temperature:- Out

BQ25792RQMR

Reference	U?
Footprint	LCSC:QFN-29_L4.0-W4.0-P0.40-TL-BQ25792RQMR
Datasheet	https://www.ti.com/cn/lit/gpn/bq25792
Description	
Manufacturer Part	BQ25792RQMR
Manufacturer	TI(德州仪器)

[Default] LCSC:QFN-29_L4.0-W4.0-P0.40-TL-BQ25792QMR

Fig. 2. imported components with scrapper shown inside KiCad

- There is no one API that can retrieve all the information we need. More time was spent to dig APIs that contain component description we need.

3) Main Control Board PCB Design: The main control board PCB design is available at https://github.com/Rover-Master/RoverMaster-PCB.

- An STM32F05 microcontroller is used for communication with ESCs, voltage, and current sensing. Initially, an ESP32C3 was considered due to its Wi-Fi and Bluetooth capabilities; however, since wireless communication is not required at this stage, the STM32 was chosen to minimize size.
- A BNO055 sensor is used for gyroscope and acceleration data.
- The PD charging unit is under research. Two chips are selected, TPS25750 and BQ25792. TPS25730 was considered. The 50 version allows more configuration using GPIOs, while the 30 version's configuration is more restricted. Multiple resistor ladder is required for configuring 30. BQ25792 is a mature solution for charging 1-4s batteries while providing power to system load. The difficulty lies on how to design the peripheral circuit based on our needs, and how to design the circuit for configuration.
- The Battery Management System(BMS) is also under research. Passive balancing is not considered due to efficiency. Capacitor balancing is chosen over inductor balancing because it is easier to set up and control. For the capacitors within BMS, aluminum electrolytic capacitors are chosen over ceramic capacitors. Although electrolytic capacitors are bigger in size, they provide better stability under constant DC bias, and they have bigger capacitance. The difficulty lies on how to design the mosfet bridges and how to simplify the full bridges to half bridges.

a) Difficulties:

- We were trying to avoid buying small modules like ideal diodes, current/voltage sensing ICs. We had to research how to design the circuits ourselves.
- The power design is iterated multiple times because we are always thinking the best way to do it based on our hypothetical use case.
- In our use case, battery charging is involved besides getting power for the robot, so we need to integrate two charging ICs to handle this.
- The simulation requires the use of battery, but there are always compromises in the solutions I find. A real battery model is hard to find and also slow to simulate. We ended up using a big capacitor to simulate the battery but the behaviors is sometimes unexpected.

b) Failed Attemps:

- We started by simulating the simplified 4 cell balancing circuit with our designed MOSFET bridge circuit. The voltage measured in the simulation is not ideal (Figure 3).
- We decided to further simplify the model to just 2 cells, but the behavior of the capacitor and battery is still confusing.
- Finally, we decided to further simplify the model to 2 batteries balanced by 1 capacitor. And we tried to eliminate the potential issue with MOSFETs' conduction direction by adding an extra pair of MOSFETs (Figure 4).

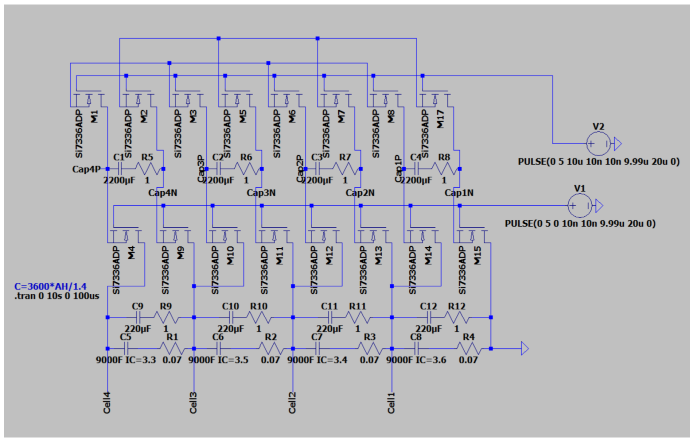


Fig. 3. 4s battery balancing design

III. Evidence of Soundness

A. Navigation with Nav2 [2]

a) Theoretical Background: Nav2 [2], built on ROS2, uses algorithms like A* and behavior trees for navigation, enabling robust path planning and obstacle avoidance. Its modularity and prior successful deployments on similar platforms confirm its feasibility on resource-constrained systems like the Raspberry Pi.

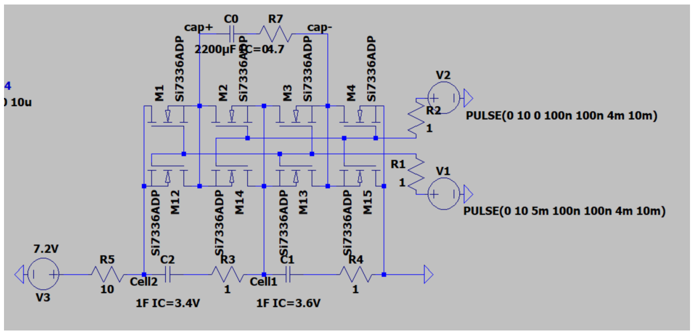


Fig. 4. Further simplified 2s balancing design

b) Prior Art: TurtleBot 4 has demonstrated successful integration of navigation tools using 2D LiDAR, IMU, Nav 2 and ROS 2. These tools allow efficient real-time navigation and localization, providing a proven foundation for implementing similar systems in Omnibot.

c) Empirical Evidence:

- **Goal:** Evaluate the functionality of Nav2 [2]for autonomous navigation.
- **Methodology:**
  - Ran Nav2 [2] on a local computer in a simulated environment.
  - Measured route planning time, execution speed, and obstacle avoidance.
  - Attempted integration on Raspberry Pi for real-world deployment.
- **Results:**
  - Nav2 [2] successfully executed navigation tasks in simulation with efficient route planning and collision-free movements.
  - Integration on Raspberry Pi faced challenges due to ROS2 version incompatibilities.
- **Interpretation:** The simulation validated Nav2's [2] capabilities. Ongoing work focuses on overcoming hardware and software constraints for full integration.

B. Depth Estimation

a) Theoretical Background: LiDAR provides precise environmental mapping using laser pulses and point cloud generation. Integration with ROS via nodes like `rplidar_ros` is a proven approach for real-time navigation. Initial experiments with the Marigold model [3] highlighted its potential for depth estimation in static frames.

b) Prior Art: TurtleBot 4 uses 2D LiDAR for mapping and navigation. Its integration with ROS 2 demonstrates LiDAR's effectiveness in dynamic environments, reinforcing its viability for Omnibot's depth estimation needs.

c) Empirical Evidence:

- **Goal:** Assess the feasibility of depth estimation using the Marigold model and evaluate LiDAR as an alternative.
- **Methodology:**
  - Tested the Marigold model [3]with single-frame inputs.

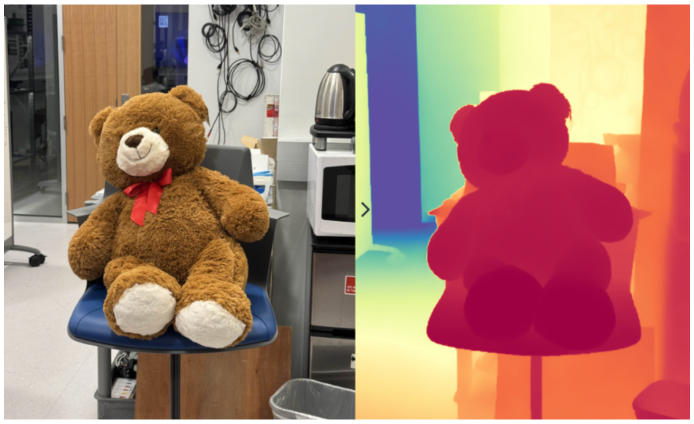


Fig. 5. Marigold Depth Estimation

  - Measured generation time and analyzed depth map quality.
  - Planned out LiDAR integration to compare speed and reliability.
- **Results:**
  - The Marigold model [3] produced high-quality depth maps but required ˜3 minutes per frame, making it unsuitable for real-time use.
  - Expect LiDAR to demonstrate faster feedback with sufficient accuracy for navigation.
- **Interpretation:** Plan to transition to LiDAR for real-time depth estimation.

C. Object Detection

a) Theoretical Background: Object detection relies on convolutional neural networks (CNNs) like YOLO v11 [1]for real-time classification. These networks have been validated extensively for dynamic environments requiring high-speed and accurate detection.

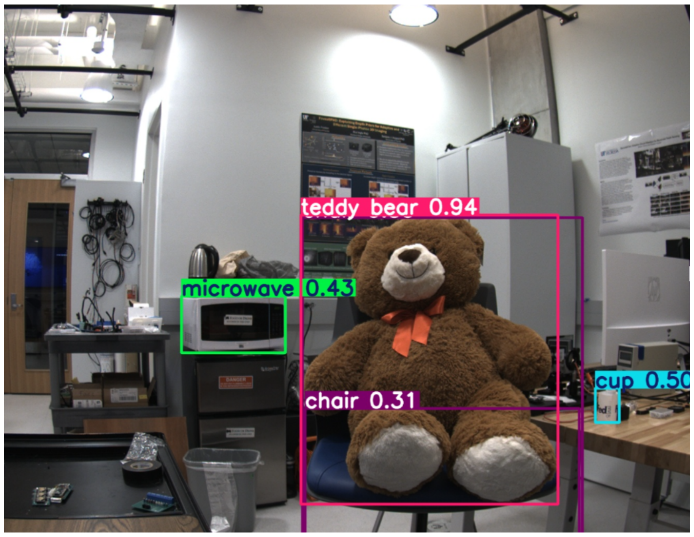


Fig. 6. Object Deteciton by YOLO v11

b) Prior Art: While TurtleBot 4 does not natively include object detection, a lot of existing tutorials proves that Yolo 11 is compatible with Ros 2.

c) Empirical Evidence:

- **Goal:** Validate the accuracy and real-time performance of YOLO v11 with Ros 2 on Raspberry Pi 5.
- **Methodology:**
  - Integrated YOLO v11 [1] into a ROS environment as a custom node.
  - Tested live video feeds from the camera to detect and classify objects.
  - Monitored detection speed (frames per second) and accuracy.
- **Results:**
  - YOLO v11 [1] achieved high accuracy with real-time processing.
  - ROS integration enabled smooth communication with other nodes.
- **Interpretation:** The experiment confirmed YOLO v11's [1] suitability for real-time object detection.

D. Battery Balancing System (BBS)

a) Theoretical Background: The most common battery balancing strategies are resistance, capacitor and inductor balancing. Among that, resistance battery balancing is passively releasing excess energy as heat, while capacitor and inductor balancing are active strategies that are more efficient.

b) Prior Art: My current design is inspired by a commercial active balancer product from Heltec BMS. The strategy is that every cell first charge a capacitor, and next the capacitors are connected in parallel so they can equalize among themselves. Finally, the capacitors are connected back to the battery to either charge the battery if V_{cap} is higher than V_{bat} and vise versa. However, what I plan to do next is called single switch capacitor balancer [4].

c) Empirical Evidence: Although theoretically the design looks sound, there are many problems in real life implementation. The first and most important question is that how to drive the MOSFET half bridge correctly. Unlike commonly half bridge applicatino where the lower MOS will connect to ground and charge the bootstrap capacitor for upper MOS. Here the MOSFETs are connect to battery and capacitors instead of ground. I will need more investigation on this part.

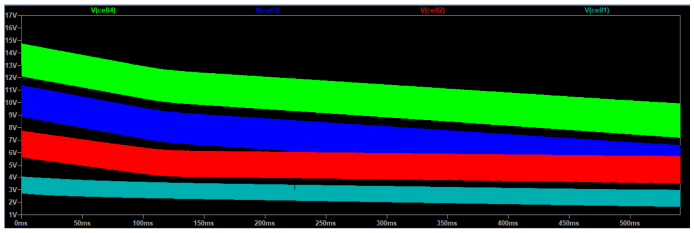


Fig. 7. Simulation on every cell voltage

E. Power Delivery and Battery Charging System

1) Theoretical Background: USB Power Delivery (USB-PD) represents a significant advancement in charging technology, enabling higher power delivery through standard USB connections. The system works by allowing devices to negotiate power requirements dynamically, supporting voltages up to 20V and currents up to 5A for a maximum power of 100W.

The charging system in our design combines two key components:

- BQ25702: An advanced battery charging controller that manages the charging process
- TPS25750: A USB-PD controller that handles power negotiation and delivery

The system operates on several key principles:

a) Power Negotiation:

- The TPS25750 communicates with the power source using USB-PD protocol
- Negotiates optimal voltage and current based on system requirements
- Supports multiple power profiles up to 20V/5A

b) Battery Charging Management:

- The BQ25702 implements a three-stage charging process:
  - Pre-conditioning for depleted batteries
  - Constant current charging for rapid charging
  - Constant voltage charging for completion
- Includes protection features for temperature, current, and voltage
- Provides real-time monitoring and adjustment of charging parameters

2) Prior Art: Traditional robot power systems often relied on separate charging and power distribution systems, which led to increased complexity and board space requirements. Several key developments have influenced our design:

a) Early USB Power Systems (2000s):

- Limited to 5V/500mA
- Required separate high-power charging solutions for robots
- Complex power management circuits

b) Quick Charge Solutions (2010s):

- Proprietary charging protocols
- Limited compatibility between systems
- Higher power but non-standardized approaches

c) Modern Integrated Solutions: Texas Instruments introduced the BQ25702 in 2019, offering:

- Integration of high-power charging capabilities
- Advanced battery management features
- Improved thermal performance

The TPS25750 represents the latest generation of USB-PD controllers, providing:

- Simplified USB-PD implementation
- Comprehensive protection features
- Compact solution for high-power applications

Our implementation builds upon these developments by combining these advanced ICs into a single, efficient charging solution that provides both the high power needed for robot operation and safe, efficient battery charging in a compact form factor (Figure 8).

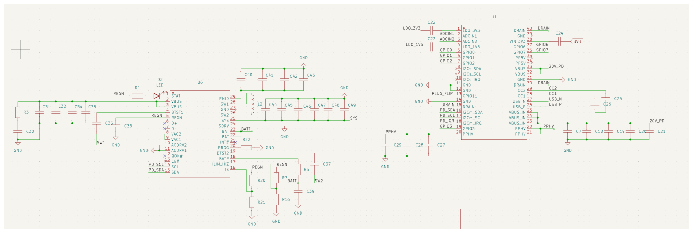


Fig. 8. Caption

IV. External Interface

A. Presentation

The robot's user interface is designed to be accessible, user-friendly, and interactive. It incorporates both physical and graphical elements to ensure intuitive control and effective monitoring of the system:

- **Camera:** Captures frames of the environment and objects, providing critical input for object detection and depth estimation. The live feed from the camera is accessible via the website for real-time observation.
- **Robot Arm:** Equipped with multiple sensors and designed for seamless operation in picking up and releasing objects. Controlled through the user interface with clear feedback on actions performed.
- **Charging:** The robot supports 20V PD charging via a Type-C cable, with the Battery Management System (BMS) monitoring and managing the charging process. Charging status and alerts are displayed on the dashboard.
- **Control:** The robot is currently operated via a controller connected to a laptop, which forwards control information to the robot. The interface highlights active control states for user clarity.
- **Website Features:** The web-based graphical interface provides the following major elements:
  - **Dashboard Overview:** Displays real-time status, including battery level, connection status, and operating mode (active, idle, or error). The currently selected states and options are highlighted for accessibility.
  - **Live Video Feed:** Streams live footage from the robot's camera, offering users a clear visual of the robot's surroundings and operations.
  - **Sensor Data Display:** Presents real-time sensor readings, such as depth estimation results and object detection outputs with their locations.
  - **Action Logs:** Tracks the robot's actions (e.g., items picked up or dropped, navigation history, errors) for transparency and review.
  - **User Controls:** Provides an interface for sending commands like moving to specific locations or picking up objects. Interactive controls highlight selected actions for ease of use.
  - **Data History:** Displays historical data on performance and logged actions, aiding in system analysis and optimization.
  - **Settings:** Allows users to configure the robot's operational parameters, with clear labeling of options and descriptions of their effects.
  - **Alerts:** Notifies users of critical issues like low battery or system errors with clear, prominent messages.

B. Perception

The interface is designed to be highly responsive, ensuring users receive immediate feedback on their interactions:

- **Visual Feedback:** Changes in system state are reflected instantly in relevant interface elements, such as highlighting active buttons, updating status indicators, or displaying new alerts.
- **Auditory Feedback:** Key actions, errors, or alerts are accompanied by distinct sounds to ensure they are noticed promptly.
- **Tactile Feedback:** For physical interactions with the robot or its controller, tactile feedback (e.g., vibration) confirms actions where applicable.

C. Usability

The system prioritizes predictable and consistent behavior across all interactions:

- **Predictable Interaction:** For identical inputs, the system consistently produces the same outputs, ensuring reliability and user confidence.
- **Error Handling:** When the interface behaves unpredictably (e.g., due to errors), clear alerts and logs explain the cause, guiding users toward resolution.
- **Aesthetic Design:** The interface is visually clear and organized, avoiding any clutter or design flaws that might hinder usability.

D. Discoverability and Accessibility

All features are designed to be discoverable, with intuitive layouts and clear descriptions. Highlights and tooltips guide users through interactive elements, ensuring ease of navigation and use.

V. Internal Systems

Engineering projects rely on internal systems that interact seamlessly to support user-facing features. Although abstracted from the user's perspective, these systems form the backbone of the artifact, enabling it to fulfill its intended purpose effectively. This section details the architecture, communication mechanisms, and resilience of the internal systems in the prototype.

A. Component Architecture

The prototype integrates multiple systems, each fulfilling a critical role in the robot's operation. These components demonstrate the project's architecture and connections:

- **Object Detection Model:** Identifies and classifies objects in frames captured by the robot's camera using advanced algorithms like convolutional neural networks (CNNs). Outputs include bounding boxes and object labels, which are essential for navigation and task execution.
- **Motion Planning System:** Utilizes input from the depth estimation and object detection models to calculate optimal arm movements. This includes determining the position, orientation, and trajectory for tasks like picking up or releasing objects, factoring in size, shape, and environmental obstacles.
- **IMU Fusion Sensor:** Provides 9 DOF (degrees of freedom) data by fusing accelerometer, magnetometer, and gyroscope outputs. This ensures accurate posture and motion tracking of the robot, essential for stable and precise operations.
- **MCU on the Main Board:** The microcontroller manages the Electronic Speed Controllers (ESCs) for motor control, ensuring precise speed and direction adjustments. It also monitors voltage and current from the power distribution system, using sensors to track battery health and power consumption for efficient energy management.

Schematics of these components and their interconnections are provided to illustrate how hardware and software elements are integrated.

B. Communication Mechanisms

The prototype demonstrates robust communication within and between components to ensure seamless operation:

- **Media for Communication:** Components communicate via wired connections for reliability and minimal latency. Future iterations may explore wireless protocols for specific modules.
- **Communication Protocols:** - *I2C Bus:* Used for communication between the microcontroller and IMU Fusion Sensor, ensuring efficient data exchange. - *UART:* Facilitates communication between the camera module and processing unit for image data transfer. - *ROS Topics:* Enables message passing between object detection, depth estimation, and motion planning nodes, ensuring modularity and scalability.
- **Context for Communication:** Data is transmitted in real-time to ensure the robot's ability to respond promptly to environmental changes or user commands. Timing mechanisms synchronize operations across components.

Simulated communication protocols are employed for components under development, with placeholders to ensure smooth integration in subsequent iterations.

C. Resilience

To maintain reliability, error correction and recovery mechanisms are incorporated:

- **Error Detection:** Sensors and processing units monitor data consistency, flagging anomalies such as invalid depth readings or misclassified objects.
- **Recovery Mechanisms:** In cases of communication failures or sensor malfunctions, fallback procedures allow the robot to halt operations safely and notify the user via alerts.
- **Placeholders for Incomplete Features:** For unimplemented features, such as advanced motion correction, placeholders simulate expected behavior. These will be replaced with fully functional implementations in the final iteration.

These systems ensure the robot can handle errors gracefully while maintaining core functionality during prototype testing.

VI. Links

- **Project Repository**: https://github.com/haoyuanxu430/Rover-Master-Logs/
- **Video Overview**: https://youtu.be/6k8LeupdFZE

References

[1] R. Khanam and M. Hussain, "Yolov11: An overview of the key architectural enhancements," *arXiv*, vol. 2410.17725, no. v1, pp. 1–9, 2024, Accessed October 24, 2024. [Online]. Available: https://arxiv.org/abs/2410.17725.

[2] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[3] B. Ke, A. Obukhov, S. Huang, N. Metzger, R. C. Daudt, and K. Schindler, "Repurposing diffusion-based image generators for monocular depth estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[4] M. Daowd, M. Antoine, N. Omar, P. Van den Bossche, and J. Van Mierlo, "Single switched capacitor battery balancing system enhancements," *Energies*, vol. 6, no. 4, pp. 2149–2174, 2013, ISSN: 1996-1073. DOI: 10.3390/en6042149. [Online]. Available: https://www.mdpi.com/1996-1073/6/4/2149.