BLIMP Platform Report

Technical Handoff and Team Management Report

Blimps Team

Arizona State University

February 10, 2026

Release Tag: `platform_report_2026_02`

GitHub Repository: https://github.com/BlimpsCompetitionAMASSLab/Blimp-Competition-AMASS-Lab-ASU

Contents

I Technical Handoff Package **4**

1 T1 — All Code in Lab GitHub Repository **4**

2 T2 — Repository Organization and Navigation **4**

3 T3 — Reproducible Setup from Clean System **6**

3.1 Hardware Requirements 6

3.2 Part 0: Flashing Ubuntu 22.04 (Raspberry Pi Imager) 6

3.3 Part 1: ROS 2 Humble Installation (Ubuntu 22.04) 9

3.4 Part 2: Xbox Controller Setup and Connection 10

3.5 Part 4: Clone Repository and Build Workspace 11

4 T4 — Release Tag **13**

5 T5 — Complete Mechanical Design Package **13**

5.1 Blimp Envelope 13

5.2 Gondola Assembly 13

5.3 3D Printed Parts 14

5.4 Mechanical Bill of Materials 14

6 T6 — Assembly Instructions with Photos **14**

6.1 Step 1: Prepare Balloon Envelope 14

6.2 Step 2: Assemble Gondola Frame 15

6.3 Step 3: Install Electronics 16

6.4 Step 4: Install Motors and Propellers 17

6.5 Step 5: Wiring and Integration 18

6.6 Step 6: Attachment to Balloon 19

6.7 Key Failure Points and Mitigation 19

7 T7 — Electronics Bill of Materials **19**

8 T8 — Wiring Diagram and Pin Map **21**

8.1 Power Architecture Overview 22

8.2 Raspberry Pi GPIO Pin Map 22

8.3 I2C Configuration 22

9 T9 — Power Architecture and Safety **23**

9.1 Safety Considerations 23

10 T10 — Interfaces and Addresses **23**
10.1 I2C Sensors . . . 23
10.2 Camera Configuration . . . 23
10.3 PWM Motor Control . . . 24
10.4 Required Kernel Overlays (/boot/config.txt) . . . 24

11 T11 — Mounting and Reliability Notes **24**
11.1 Mechanical Securing Methods . . . 24
11.2 Observed Reliability Issues and Fixes . . . 24

12 T12 — Software Stack Documentation **25**
12.1 Repository Structure and Module Purpose . . . 25
12.2 Repository Structure and Module Purpose . . . 25
12.3 Configuration Parameters . . . 27
12.4 Logging . . . 27

13 T13 — How the System Works **28**
13.1 System Architecture . . . 28
13.2 Control Loop (50 Hz) . . . 29
13.3 Vision Autonomy Pipeline (YOLOv5) . . . 32

14 T14 — How to Run Each Capability **34**
14.1 Remote Access via SSH (PuTTY on Windows) . . . 34
14.1.1 Step 1: Install PuTTY on Windows . . . 34
14.1.2 Step 2: Find the Raspberry Pi's IP Address . . . 35
14.1.3 Step 3: Configure and Launch PuTTY . . . 35
14.1.4 Step 4: Login to Raspberry Pi . . . 36
14.1.5 Step 5: Navigate and Run `updated_launch.py` . . . 36
14.1.6 Step 6: Troubleshooting SSH Connection . . . 37
14.1.7 Bonus: Hardware Configuration for SSH . . . 38
14.1.8 Bonus: Persistent SSH Sessions with `screen` or `tmux` . . . 38
14.2 1. Sensor Bring-up and Verification . . . 38
14.2.1 2. Hover Control Demo . . . 39
14.2.2 3. Camera Bring-up and Streaming Verification . . . 40
14.2.3 4. Vision Autonomy Demo . . . 40
14.2.4 5. Cage Gate Actuation Demo . . . 41

15 T15 — Validation Evidence and Acceptance Tests **42**

16 T16 — Known Issues and Troubleshooting Guide **42**
16.1 Prioritized Open Issues . . . 42
16.2 Troubleshooting by Symptom . . . 43

17 T17 — Decisions Log (Tested and Abandoned) 44

II Team Management Report 44

18 M1 — Team Roster, Roles, and Ownership 44
18.1 Team Members 44

19 M2 — Assigned Tasks and Current Status 44
19.1 Mechanical Subsystem 44
19.2 Electrical Subsystem 45
19.3 Software, Simulation, and Controls 45

20 M3 — Weekly Milestones: Plan vs. Actual 45

21 M4 — Planned Daily Working Hours and Cadence 46
21.1 Team Schedule 46
21.2 Meeting Cadence 47
21.3 Blocker Escalation 47

22 M5 — Process and Quality Control 47
22.1 Definition of Done 47
22.2 Code Review and Merge Policy 48
22.3 Hardware Change Control 48

Appendix 48

A Appendix A: Decisions Log 48

B Appendix B: Acceptance Tests Summary 49

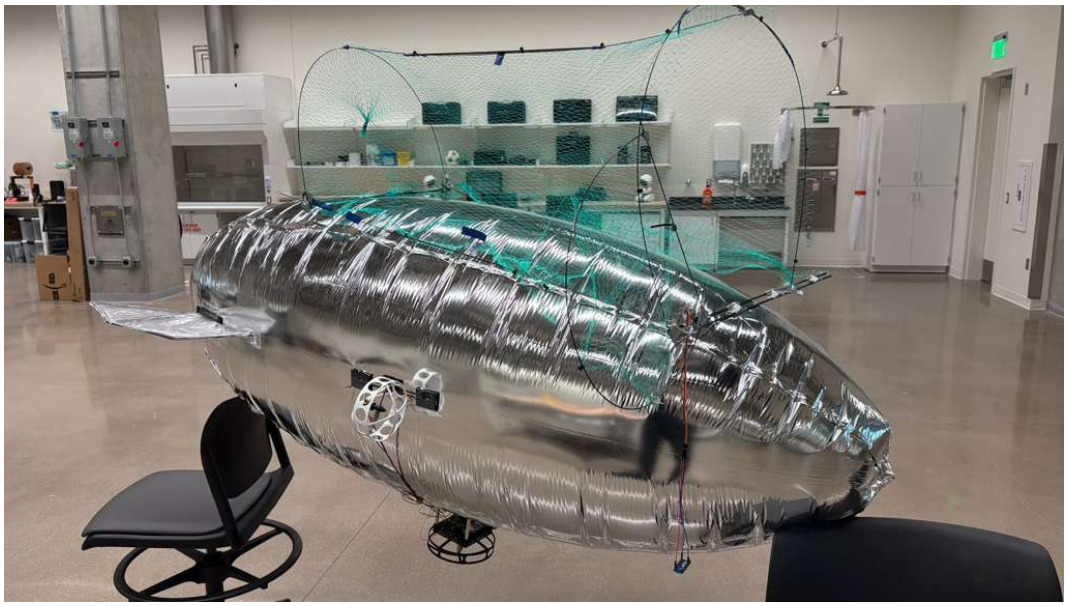


Figure 1: Fully assembled BLIMP autonomous platform.

Part I

Technical Handoff Package

1 T1 — All Code in Lab GitHub Repository

All code is hosted in the official Blimp Competition AMASS Lab GitHub repository:

- **Repository:** AMASS Lab GitHub Repo
- **Ownership:** BlimpsCompetitionAMASSLab organization
- **Access:** Public repository — cloneable by all team members

The repository contains all ROS 2 packages, launch files, configuration files, and scripts required to build and operate the BLIMP system. No code exists in personal repositories, zip packages, or local-only folders.

2 T2 — Repository Organization and Navigation

```
blimp_ws/
  README.md                    # Quick start guide and directory map
  test_environment.txt         # Environment test data

  docs/                        # Detailed build and software documentation
    BLIMP_Handoff_Report_Final.md
```

```
  GPIO_PIN_MAP.md
  hardware.md
  INSTALLATION.md
  NODES_REFERENCE.md
  SETUP_INTEGRATION_SUMMARY.md
  SYSTEM_DESIGN.md
  TROUBLESHOOTING.md
  images/

blimp_src/                  # ROS 2 source packages
  README.md
  LICENSE
  requirements.txt
  consolidated_setup.sh
  test_data
  bag_files/                # ROS 2 bag files
    Barometer_ros2bag/      # Barometer data utilities
    rosbag2_2026_02_11-12_11_47/
    rosbag2_2026_02_11-12_19_11/
  blimp_interfaces/         # Custom ROS message definitions
  camera/                   # Camera nodes and utilities
  controls/                 # Low-level control drivers
  data_scripts/             # Data analysis and visualization scripts
  launch/                   # Launch files
  logs/                     # Log files and data recordings
  manual_control/           # Manual control modes
  sensors/                  # Sensor reader nodes (Python)
  sensors_cpp/              # Sensor nodes (C++)

hardware/                   # Mechanical design and electronics
  cad/                      # CAD design files
  Datasheets/               # Component datasheets
  ESC programmer/           # ESC configuration tools
  PCB design/               # PCB layouts

Instructions/               # Setup instructions
  Controls/                 # Control setup guides

plots/                      # Generated plots and graphs
videos/                     # Project videos
```

Key Navigation Points:

- New users should start with `README.md` for quick start.
- Detailed build instructions are in `docs/`.

3 T3 — Reproducible Setup from Clean System

3.1 Hardware Requirements

- **Raspberry Pi Model:** Raspberry Pi 4 (4 GB or 8 GB RAM recommended)
- **OS:** Ubuntu 22.04 LTS (Jammy) for Raspberry Pi
- **Python Version:** Python 3.10+
- **ROS 2 Distro:** ROS 2 Humble

Setup Instructions

3.2 Part 0: Flashing Ubuntu 22.04 (Raspberry Pi Imager)

First, You must flash the Ubuntu 22.04 operating system onto your microSD card using the official Raspberry Pi Imager.

Step 1: Open Raspberry Pi Imager

Insert your microSD card into your computer and open the Raspberry Pi Imager application.

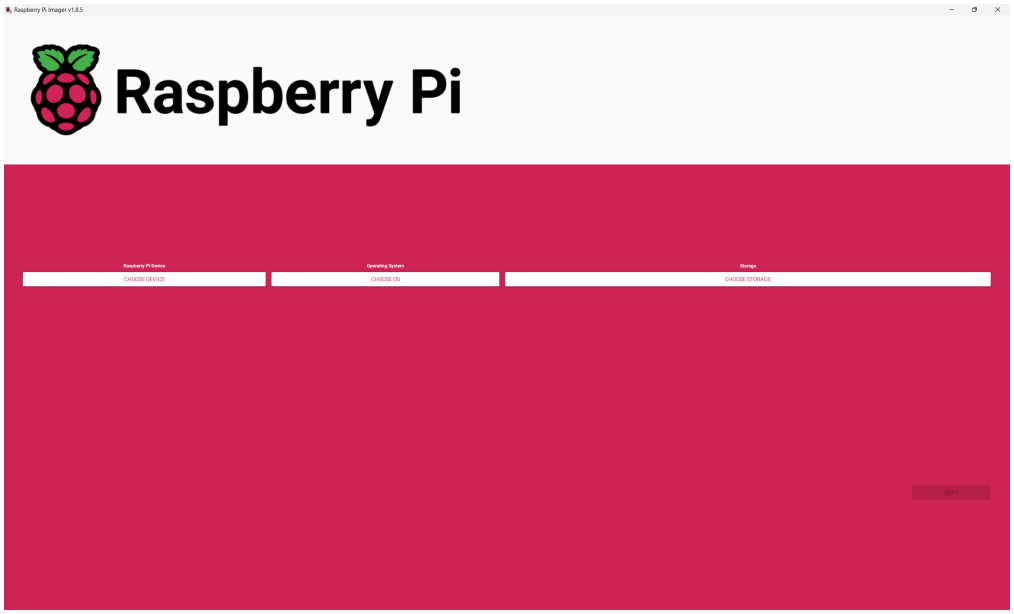


Figure 2: Raspberry Pi Imager Start Screen

Step 2: Choose Device

Click on "CHOOSE DEVICE" and select **Raspberry Pi 4** from the list.

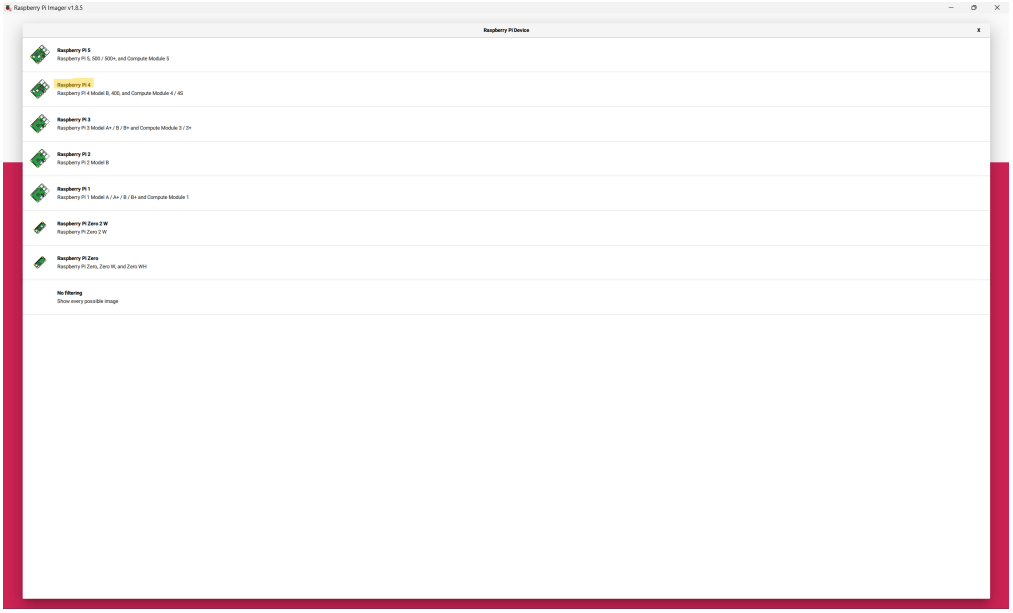


Figure 3: Select Raspberry Pi 4

Step 3: Select Operating System Category

Click on "CHOOSE OS". Scroll down the list and select **Other general-purpose OS**.

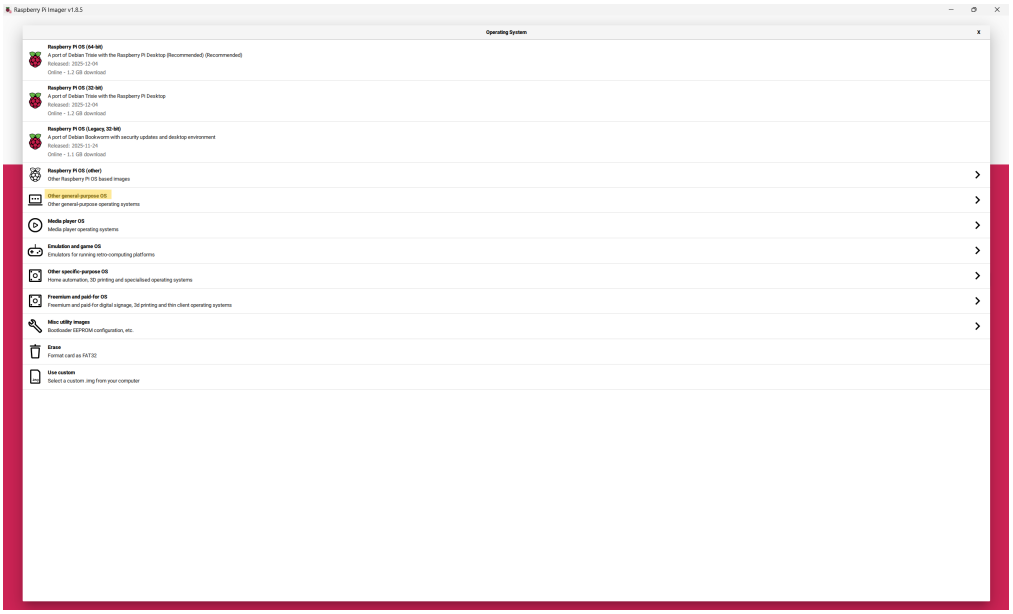


Figure 4: Operating System Menu

Step 4: Navigate to Ubuntu

Select **Ubuntu** from the list of general-purpose operating systems.

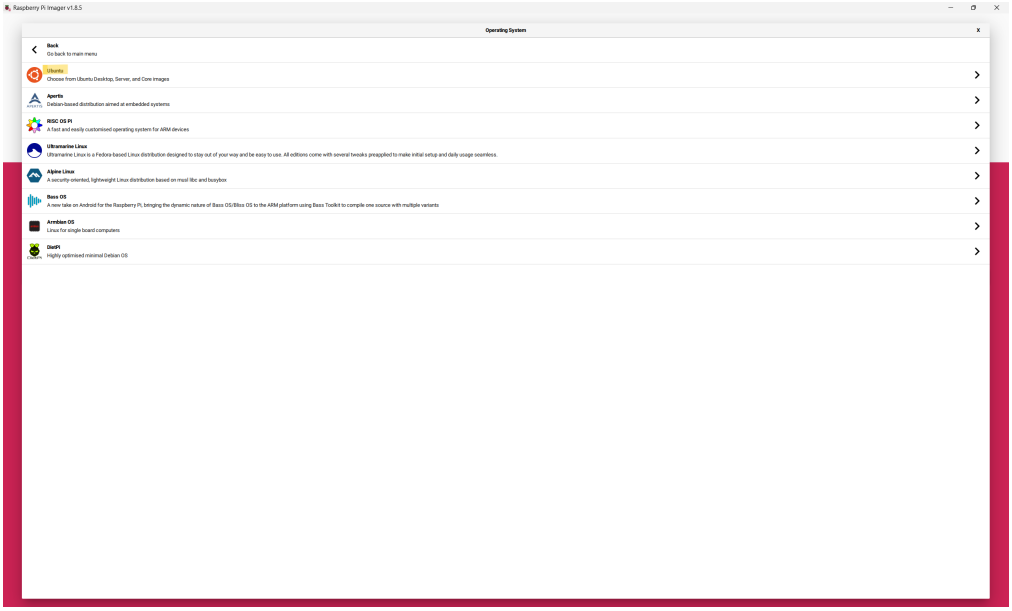


Figure 5: Select General Purpose OS

Step 5: Select Ubuntu 22.04 LTS (64-bit)

IMPORTANT: You must select **Ubuntu Desktop 22.04.5 LTS (64-bit)**. Do NOT select newer versions (like 23.10 or 24.04) as ROS 2 Humble is specifically targeted for Ubuntu 22.04.

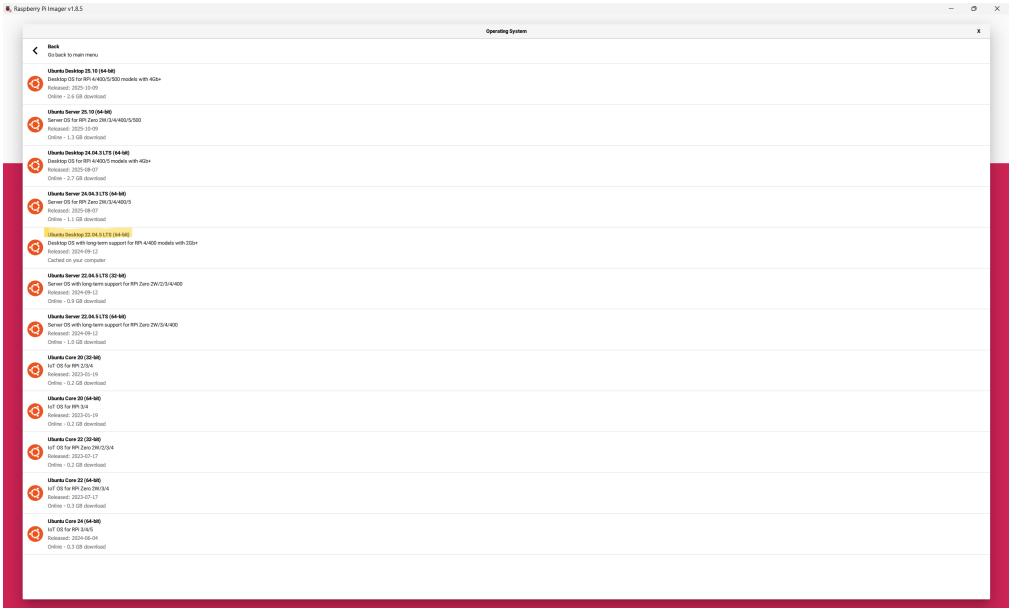


Figure 6: Select Ubuntu 22.04 LTS (64-bit)

Step 6: Write to Storage

Click "CHOOSE STORAGE" to select your SD card. Verify your selections match the image below, then, without changing settings, click **NEXT** to begin writing.

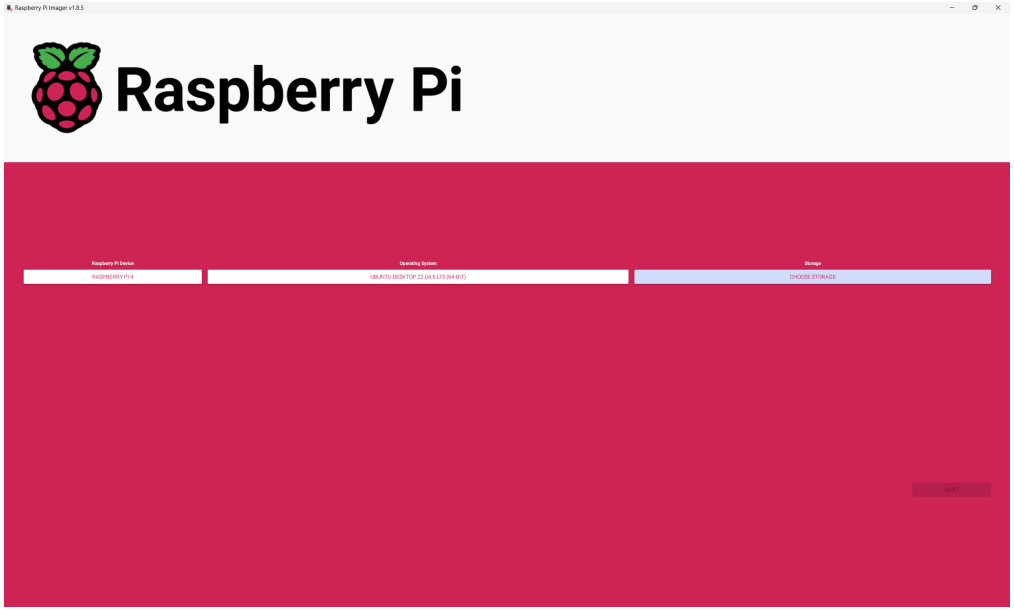


Figure 7: Final Verification Before Writing

3.3 Part 1: ROS 2 Humble Installation (Ubuntu 22.04)

After setting up the sd card, put it inside the raspberry pi and boot the system following the prompts for setting up the Ubuntu and connect to a wifi network. Once done, start the ROS2 installation.

For the most up-to-date installation details and troubleshooting, please refer to the official ROS 2 documentation: ROS 2 Humble Debian Install Guide.

Step 1: Set Locale & Add Repositories

```
locale  # check for UTF-8

sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale

sudo apt install software-properties-common
sudo add-apt-repository universe
```

Paste all the commands in the followiing block together in the terminal

```
sudo apt update && sudo apt install curl -y
export ROS_APT_SOURCE_VERSION=$(curl -s https://api.github.com/repos/ros-
    infrastructure/ros-apt-source/releases/latest | grep -F "tag_name" | awk
     -F\" '{print $4}')
curl -L -o /tmp/ros2-apt-source.deb "https://github.com/ros-infrastructure/
    ros-apt-source/releases/download/${ROS_APT_SOURCE_VERSION}/ros2-apt-
    source_${ROS_APT_SOURCE_VERSION}.$(. /etc/os-release && echo ${
    UBUNTU_CODENAME:-${VERSION_CODENAME}})_all.deb"
sudo dpkg -i /tmp/ros2-apt-source.deb
```

Step 2: Install ROS 2 Packages

```
sudo apt update
sudo apt upgrade
# Desktop Install (Recommended for dev):
sudo apt install ros-humble-desktop
# OR Base Install (Recommended for Pi/Headless):
# sudo apt install ros-humble-ros-base
sudo apt install ros-dev-tools
source /opt/ros/humble/setup.bash
sudo apt install python3-colcon-common-extensions
```

Step 3: Environment Setup Add the sourcing script to your shell startup so ROS commands work in every new terminal:

```
echo "source /opt/ros/humble/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

3.4 Part 2: Xbox Controller Setup and Connection

The BLIMP platform uses an Xbox controller for manual control. Follow these steps to connect the controller to the Raspberry Pi via Bluetooth.

Step 1: Install Bluetooth Packages

```
sudo apt update
sudo apt upgrade
sudo apt install xboxdrv
echo 'options bluetooth disable_ertm=Y' | sudo tee -a /etc/modprobe.d/
    bluetooth.conf
sudo reboot
```

Step 2: Enable Bluetooth Service

```
sudo systemctl start bluetooth
sudo systemctl enable bluetooth
```

Step 3: Pair the Xbox Controller

1. Put the Xbox controller into pairing mode by pressing and holding the pairing button (small button on top of the controller near the USB port) until the Xbox logo starts flashing rapidly.
2. On the Raspberry Pi, start the Bluetooth pairing process:

```
sudo bluetoothctl
```

Within the `bluetoothctl` prompt, enter the following commands:

```
agent on
default-agent
scan on
```

Wait for your controller to appear in the list (it will show as "Xbox Wireless Controller" or similar). Note the MAC address (format: XX:XX:XX:XX:XX:XX).

```
trust XX:XX:XX:XX:XX:XX
connect XX:XX:XX:XX:XX:XX
scan off
exit
```

Replace `XX:XX:XX:XX:XX:XX` with your controller's actual MAC address.

Step 4: Verify Connection

```
sudo apt install joystick
sudo jstest /dev/input/js0
```

You should see device files like `js0` or `event*` indicating the controller is connected.

Important Note:

- The controller MAC address is **hardcoded** in the control software.
- You **must** update the MAC address in `updated_launch.py` to match your specific controller's MAC address obtained during pairing.
- Failure to update this will prevent the controller from working with the BLIMP platform.

For detailed instructions and troubleshooting, refer to: Xbox Controllers on Raspberry Pi Guide.

3.5 Part 4: Clone Repository and Build Workspace

After setting up ROS 2 and the Xbox controller, you need to clone the BLIMP project repository and build the workspace.

Step 1: Install Git (if not already installed)

```
sudo apt update
sudo apt install -y git
```

Step 2: Create Workspace Directory and Clone Repository

```
# Create the workspace directory
mkdir -p ~/blimp_ws

# Navigate to the workspace
cd ~/blimp_ws

# Clone the repository
git clone https://github.com/BlimpsCompetitionAMASSLab/Blimp-Competition-
    AMASS-Lab-ASU.git blimp_src
```

Step 3: Install Dependencies

```
# Navigate to the workspace root
cd ~/blimp_ws

# Install ROS dependencies using rosdep
sudo apt install -y python3-rosdep
sudo rosdep init
rosdep update
rosdep install --from-paths blimp_src --ignore-src -r -y
```

Step 4: Build the Workspace

```
# Build the workspace using colcon
cd ~/blimp_ws
colcon build --packages-select blimp_interfaces
source ~/blimp_ws/install/setup.bash
colcon build

# Source the workspace
source ~/blimp_ws/install/setup.bash
```

Step 5: Add Workspace to Shell Startup To automatically source the workspace in every new terminal:

```
echo "source ~/blimp_ws/install/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Verify the Build:

```
# Check if packages are available
ros2 pkg list | grep blimp
```

If the build was successful, you should see the BLIMP-related packages listed.

4 T4 — Release Tag

Release Tag: `blimp_report_02_2026`

```
cd ~/blimp_ws/blimp_src
git tag blimp_report_02_2026
git push origin blimp_report_02_2026
```

5 T5 — Complete Mechanical Design Package

Primary CAD Software: Autodesk Fusion 360 (Native: `.f3d`) and Solidworks.

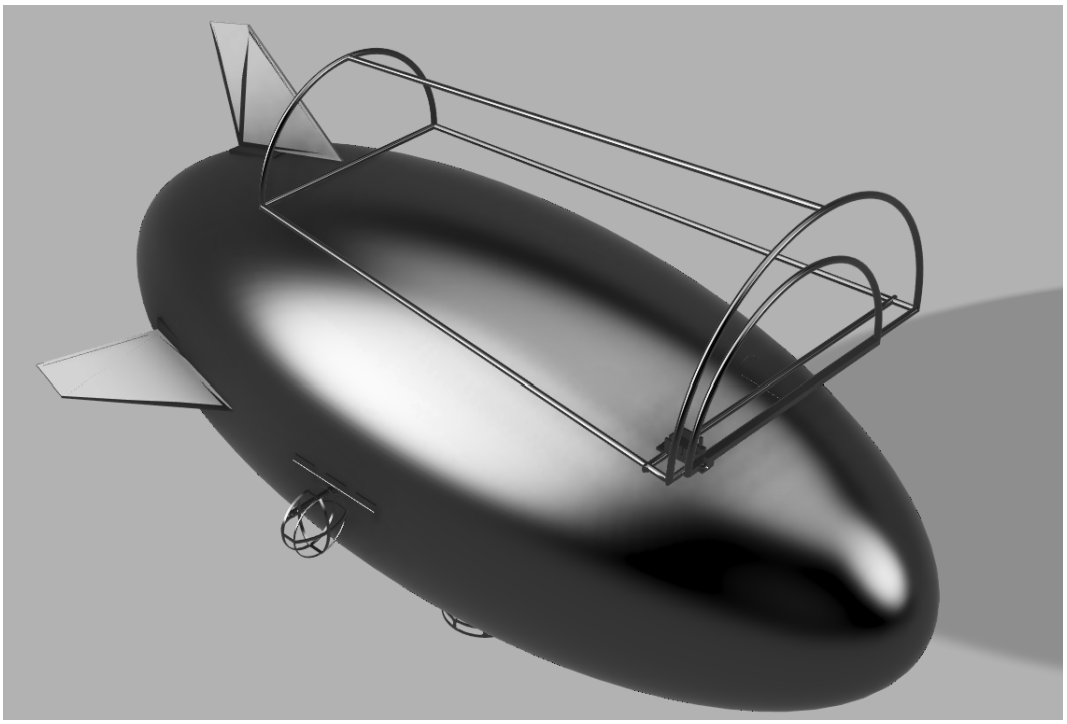


Figure 8: Isometric view of Fusion 360 CAD model of the BLIMP.

5.1 Blimp Envelope

- **Files:** `.f3d`, `.step`
- **Material:** TritaX Silver
- **Quantity:** 1

5.2 Gondola Assembly

- **Primary CAD:** `.f3d` (Fusion 360 Assembly)
- Components:

- Gondola Bottom Connectors (PLA, 3D printed)
- Balsa Wood Frame (8 structural members)
- Foam Board Electronics Platform
- 6in Prop Guard

5.3 3D Printed Parts

Global Print Settings:

- Layer Height: 0.08 mm
- Infill: 100%

Part	File Formats	Material	Supports
6in Prop Guard	.f3d, .step, .stl	Bambu PLA	Tree
4in Guard	.f3d, .step, .stl	ABS / PLA Aero	Tree
Gondola Bottom Connector	.f3d, .step, .stl	Bambu PLA	Regular
Foam to Balsa 4in	.f3d, .step	Bambu PLA	As Needed

5.4 Mechanical Bill of Materials

Part Name	File Formats	Material	Qty
Blimp Envelope	.f3d, .step	TritaX Silver	1
6in Prop Guard	.f3d, .step, .stl	Bambu PLA	1
4in Guard	.f3d, .step, .stl	ABS / PLA Aero	2
Gondola Bottom Connector	.f3d, .step, .stl	Bambu PLA	8
Foam_Fin	.f3d, .dxf, .step	Foamboard	3
Gondola Frame	.f3d, .step	Balsa Wood	8
Foam Attachment 4in Prop	.f3d, .step	Foam	6
Foam to Balsa 4in	.f3d, .step	Bambu PLA	2
Balsa Wood to 4in	.f3d, .step	Balsa Wood	2
Net + Capture Mechanism	.f3d, .step	Carbon Fiber + Net	—

6 T6 — Assembly Instructions with Photos

6.1 Step 1: Prepare Balloon Envelope

1. Inspect latex balloon for defects.

2. Connect helium inlet valve.
3. Fill to be approximately neutral buoyancy.
4. Attach net to balloon with clear tape at 4 equidistant points.
5. Verify envelope integrity, checking all points for leaking.

Figure 9: Balloon envelope prepared with helium and net attachment points.

6.2 Step 2: Assemble Gondola Frame

1. Press M5x6mm heat inserts into Gondola Bottom Connectors at bottom hole.
2. Secure balsa bottom plate to Connectors using M5x8mm screws.
3. Insert 4 vertical (sides) and 4 horizontal (top) balsa wood sticks, forming hollow cube.
4. Reinforce joints with Cyanoacrylate (Superglue).
5. Attach 6in propeller and guard in the middle of the balsa plate using two M5x8mm screws. Propeller is attached with two M2x12mm screws, and held by M2 nuts on other side of balsa wood plate.

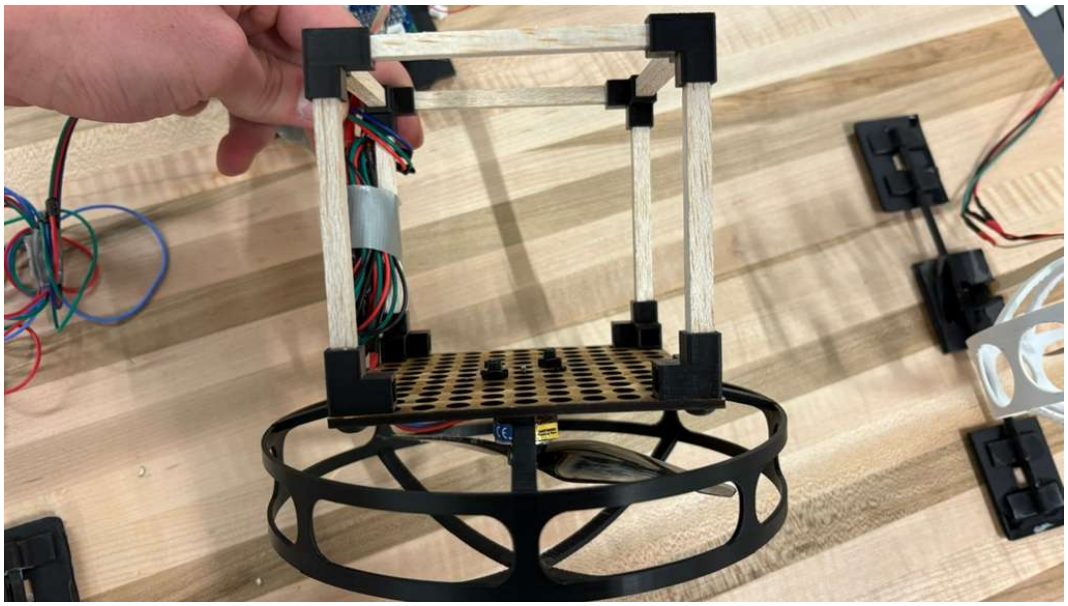


Figure 10: Gondola Frame

6.3 Step 3: Install Electronics

1. Mark Raspberry Pi 4B (8GB) attachment holes on elevated foam board and puncture.
2. Take four balsa wood pieces of 10.5mm, and puncture through lengthwise.
3. Mount Raspberry Pi 4B (8GB) on elevated foam board using M2x16mm screws with balsa pieces as spacers. Use M2 nut to secure from under foamboard.
4. Attach Custom PCB on Raspberry Pi 4B (8GB) pins for power distribution.
5. On inner protrusion of bottom gondola connectors, place double-sided tape and lay flat elevated foamboard with Raspberry Pi 4B (8GB) and PCB on tape.
6. Attach BLHeli_32 ESCs to vertical balsa pillars using Duct Tape.
7. Attach BMP390 (Baro) and BNO085 (IMU) to foamboard platform.

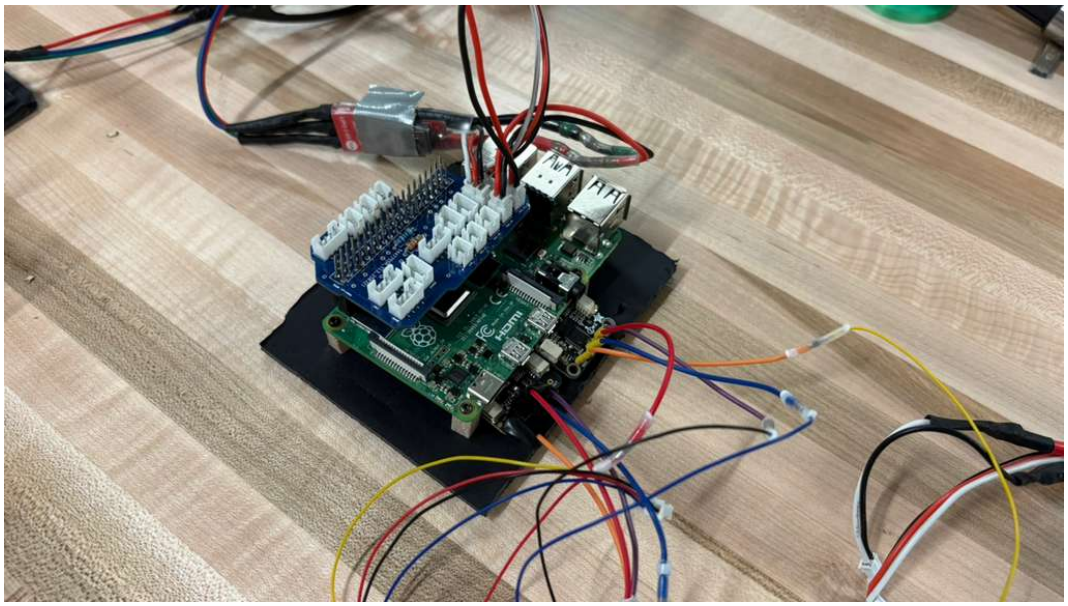


Figure 11: Electronics tray with Raspberry Pi, ESC, and sensors mounted.

6.4 Step 4: Install Motors and Propellers

1. Connect three foam board pieces with rectangular hole in the middle using 4mm diameter, 205mm carbon fiber rod, keeping hole clear.
2. In the middle foamboard piece, insert 95mm balsa wood piece and secure with Super Glue.
3. Mount 4-inch Propellers for side motor mounts by screwing wood screws directly into wood.
4. Ensure correct rotational direction (If directions are different just switch 2 of the wires connection of the motor for changing direction from clockwise to counter clockwise.
5. Verify motor arms do not interfere with frame or cables.

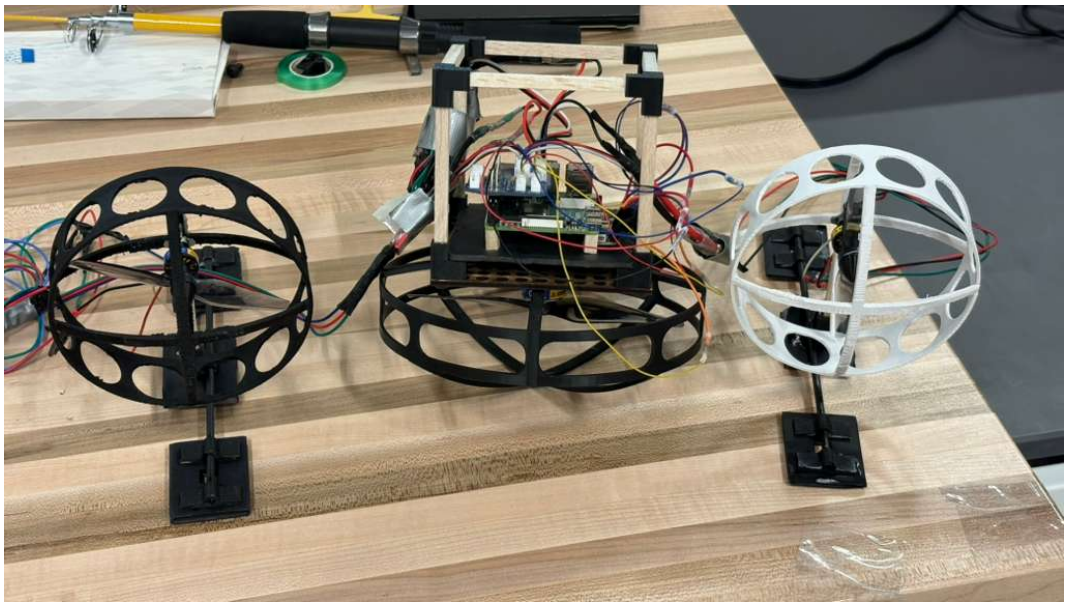


Figure 12: Gondola and motor mounts.

6.5 Step 5: Wiring and Integration

1. Route motor wires outisde propeller guard, away from propeller.
2. Connect motor wires to ESC.
3. Attach camera forward-facing.
4. Route all sensor wires to relevant ports on Custom PCB on Raspberry Pi.

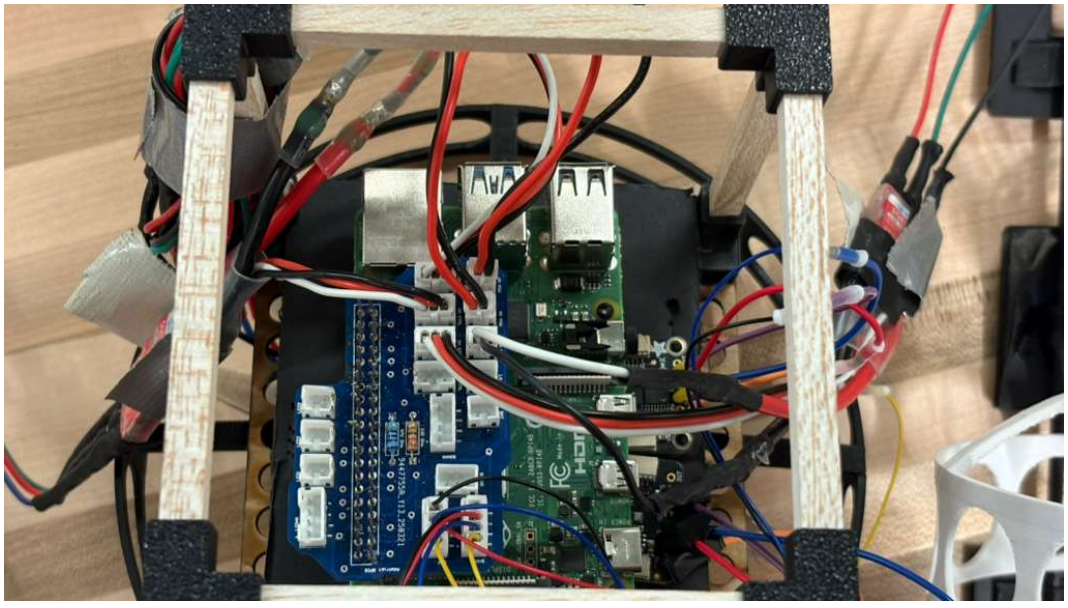


Figure 13: Wiring and sensor integration on the gondola.

6.6 Step 6: Attachment to Balloon

1. Use tape to attach gondola to blimp bottom Center of Gravity
2. Ensure equal weight distribution at all four attachment points..
3. Test that system is securely attached.

Figure 14: Final assembly: gondola suspended from balloon.

6.7 Key Failure Points and Mitigation

- **Wire strain:** Secure all wires with cable clips; avoid sharp bends; leave slight slack in the wires.
- **Propeller imbalance:** Verify weight and alignment; replace if damaged.
- **Battery sag:** Support battery with secondary straps.

7 T7 — Electronics Bill of Materials

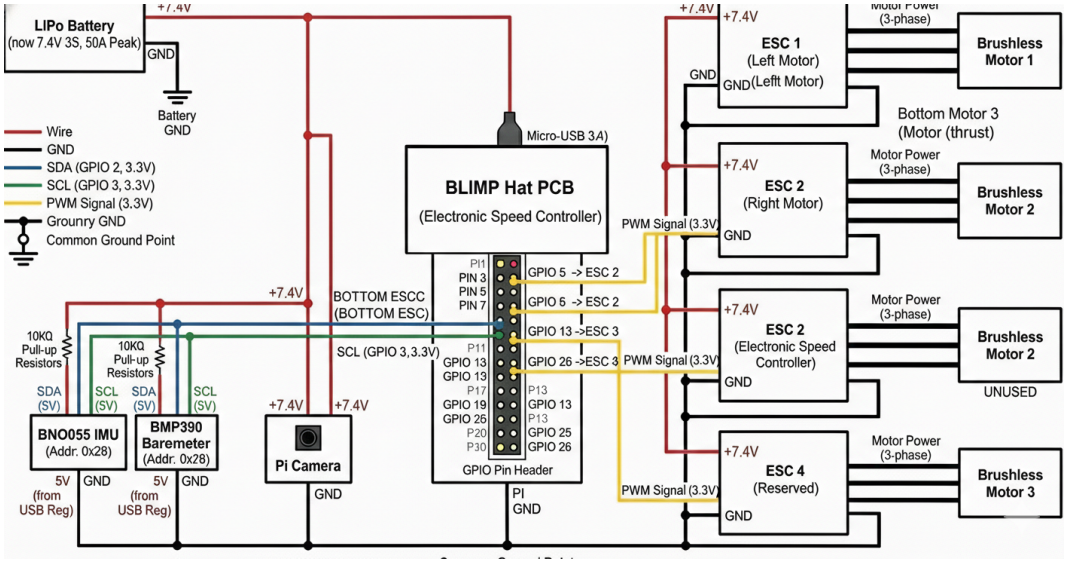
#	Component	Vendor	Part No.	Qty	Cost	Notes
1	Raspberry Pi 4B (8 GB)	Adafruit	B0B7RN5PPN	1	$75	Primary compute
2	IMU	Adafruit	BNO085	1	$25	9-DOF sensor fusion
3	Barometer	Adafruit	BMP390	1	$11	Altitude estimation
4	Pi Camera v2 (8 MP)	Adafruit	SC0049	1	$30	Object detection

#	Component	Vendor	Part No.	Qty	Cost	Notes
5	SG90 Servo	Amazon	B01CT…	1	$6 ea	Cage Motor
6	50A ESC	Amazon	N/A	3	$35	PWM motor control
7	900mAh Battery	Amazon	2879	1	$40	7.4 V nominal
8	GPIO Header Kit	Adafruit	2822	1	$5	Connection headers
9	USB Micro B Cable	Standard	N/A	2	$3	Power delivery
10	F–F Jumper Wires	Various	N/A	1pk	$5	Prototyping
11	M–F Jumper Wires	Various	N/A	1pk	$5	I2C connections
12	Micro SD 64 GB	SanDisk	SDSQ…	1	$12	OS storage
13	Park 180 Motor	Amazon	EFLM1120	2	$42	Side thrust motors
14	Park 250 Motor	Amazon	EFLM1130	1	$ 45	Bottom motor

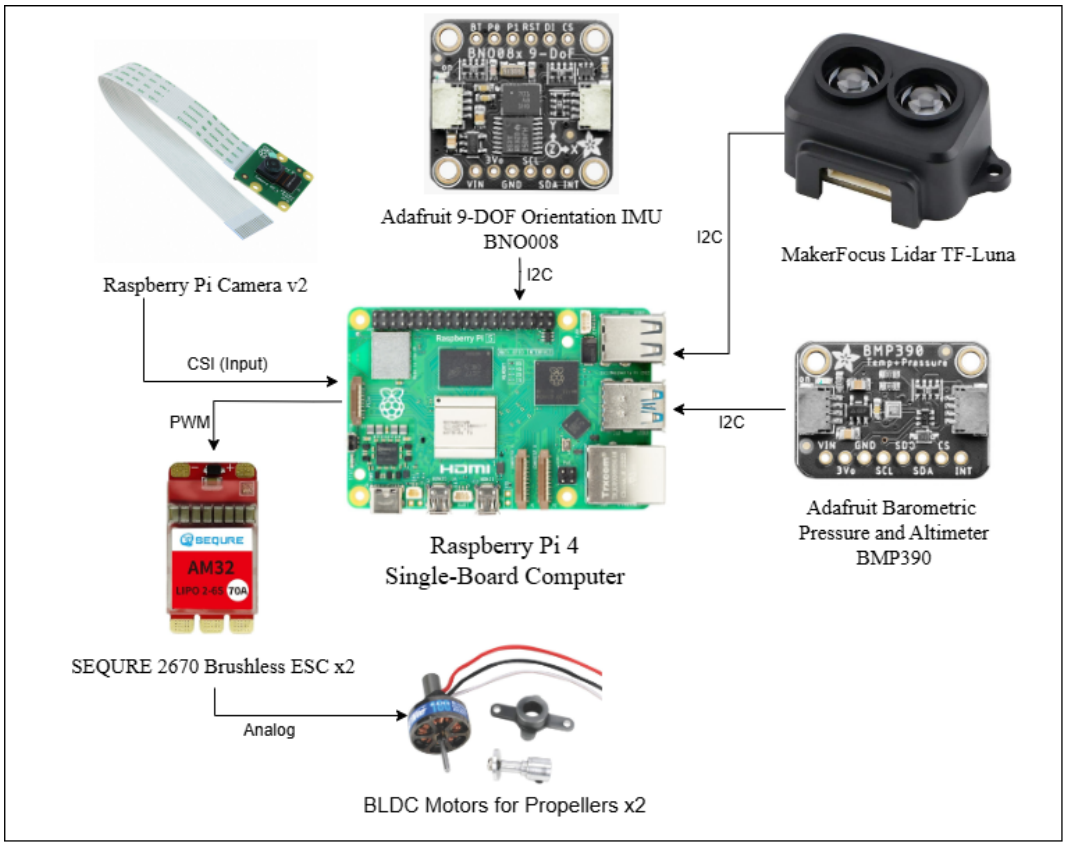
Acceptable Substitutions:

- Servo motors: Any standard servo (6–9 g, 4.8–6 V).
- Barometer: DPS310 or HP303B (must support I2C).
- IMU: MPU9250 or LSM9DS1 (requires driver modification).
- LiPo Battery: Any Standard LiPo battery within the range.

8 T8 — Wiring Diagram and Pin Map



(a) Circuit Connection



(b) System Components

Figure 15: Complete system overview showing wiring and components.

8.1 Power Architecture Overview

```
LiPo Battery (7.4V, 900 mAh)
    |
    +-> Custom PCB
    |
    +-> [50A ESC] --> [Servo Motors x4]
            |
            +-> [Raspberry Pi 4B]
                +-> GPIO Header
                +-> I2C Bus (SDA/SCL)
                +-> Camera CSI Ribbon
                +-> USB

I2C Sensors (3.3V logic, level-shifted)
   +-> IMU (BNO085)     @ 0x28
   +-> Barometer (BMP390) @ 0x76
```

8.2 Raspberry Pi GPIO Pin Map

GPIO Pin	Function	Device	I2C Addr	Notes
GPIO 2 (SDA1)	I2C Data	IMU, Baro	0x28, 0x76	3.3 V
GPIO 3 (SCL1)	I2C Clock	IMU, Baro	—	3.3 V
GPIO 5	ESC 1 PWM	Motor 1	—	1000–2000 μs, 50 Hz
GPIO 6	ESC 2 PWM	Motor 2	—	1000–2000 μs, 50 Hz
GPIO 13	ESC 3 PWM	Motor 3	—	1000–2000 μs, 50 Hz
GPIO 26	ESC 4 PWM	Motor 4	—	1000–2000 μs, 50 Hz
GPIO 4	Camera Enable	Camera	—	CSI ribbon
3.3 V	Logic Power	Level Shifter	—	Max 500 mA
5 V (USB)	Sensor Power	IMU, Baro	—	USB regulator
GND	Ground	All devices	—	Common ground

8.3 I2C Configuration

```
Master: Raspberry Pi (GPIO 2/3 = SDA1/SCL1)
Speed: 400 kHz
Devices:
  - BNO085 IMU @ 0x28 (COM3 low) or 0x29 (COM3 high)
  - BMP390 Barometer @ 0x76 (SDO low) or 0x77 (SDO high)
```

9 T9 — Power Architecture and Safety

Battery Endurance: 5000 mAh ÷ 3 A average ≈ 90 min (ground), ∼15–20 min (flight).

9.1 Safety Considerations

1. **Manual Kill Switch:** ESC arm/disarm before every flight.
2. **Thermal Management:** ESC adequate airflow (up to 50 W dissipation).
3. **Ground Testing:** Secure blimp with tether before motor tests.
4. **Failsafe Modes:**
    - Communication lost → zero thrust.
    - Pi crash → motors retain last command (mitigate with watchdog).
    - Emergency → Quick battery disconnect.

10 T10 — Interfaces and Addresses

10.1 I2C Sensors

```
Bus: /dev/i2c-1 (SDA1/SCL1)
Baud Rate: 400 kHz

Devices:
  BNO085 IMU
    Address: 0x28 (COM3->GND) or 0x29 (COM3->VDD)
    Rates: 1-100 Hz (firmware selectable)
    Output: Euler, Quaternion, Linear Acceleration
    Format: Calibrated 16-bit integers

  BMP390 Barometer
    Address: 0x76 (SDO->GND) or 0x77 (SDO->VDD)
    Rate: 1-200 Hz
    Measurement: Pressure + Temperature
    Enable: 10ms power-up delay
```

10.2 Camera Configuration

```
Camera: Raspberry Pi Camera Module v2
Interface: CSI Ribbon
Resolution: 1640x1232 @ 30 fps (recommended)
Field of View: ~120 degrees diagonal
```

```
Enable in /boot/config.txt:
  start_x=1
  gpu_mem=256
```

10.3 PWM Motor Control

```
Frequency: 50 Hz (20ms period)
Duty Cycle: 1000-2000 us
Idle: 1500 us (center)
Min Thrust: 1100 us
Max Thrust: 1900 us
Deadzone: +/- 50 us
```

10.4 Required Kernel Overlays (/boot/config.txt)

```
dtparam=i2c_arm=on
dtparam=spi=on
start_x=1
gpu_mem=256
```

11 T11 — Mounting and Reliability Notes

11.1 Mechanical Securing Methods

1. **Raspberry Pi:** Foamboard base, 4 M2x16mm screws, balsa wood spacers, M2 nuts
2. **IMU (BNO055):** Foamboard attachment.
3. **Barometer (BMP390):** Foamboard attachment.
4. **Camera:** 3D-printed bracket, taped to blimp nose, ribbon along blimp body to gondola.
5. **ESC & Battery:** Duct tape to vertical ESC, Lithium battery attached at gondola top.

11.2 Observed Reliability Issues and Fixes

Issue	Root Cause	Fix	Status
Motor Jetter at hover	PWM noise from ESC	Made the slot a bit bigger and smoother	Resolved
I2C errors with IMU	Loose ribbons, weak pull-ups	Soldered connections	Resolved

Issue	Root Cause	Fix	Status
Camera ribbon disconnect	Insufficient strain relief	larger bends	Resolved
Battery voltage sag	Thin cables, long ESC run	12 AWG cables, ESC closer	Partial
Helium leakage	Small continuous leaks	Fill up every 7 Days	Mitigation

12 T12 — Software Stack Documentation

12.1 Repository Structure and Module Purpose

12.2 Repository Structure and Module Purpose

```
blimp_src/
  requirements.txt          # Python dependencies
  consolidated_setup.sh     # Setup script
  LICENSE
  README.md
  test_data                 # Test data files

  blimp_interfaces/         # Custom ROS message definitions
    msg/
      BaroData.msg          # Barometer altitude and temperature
      Bool.msg              # Boolean message
      CameraCoord.msg       # Camera coordinates
      CartCoord.msg         # Cartesian coordinates
      EscInput.msg          # ESC input signals
      ImuData.msg           # Raw IMU outputs
      LidarData.msg         # LIDAR measurements
      UtcTime.msg           # UTC timestamp
    srv/
      Detection.srv         # Detection service
      GetCamera.srv         # Camera service

  sensors/ (Python)         # Sensor reader nodes
    sensors/
      barometer.py          # BMP390 -> /barometer_data
      BNO055.py             # BNO055 IMU sensor
      BNO085.py             # BNO085 IMU sensor
      Lidar.py              # LIDAR sensor
      Sonar.py              # Sonar sensor
      servo.py              # Servo control
```

```
    LED.py                  # LED control
    record_data.py          # Data recording utilities

sensors_cpp/ (C++)          # Processing nodes
  src/
    Balloon_pi.cpp          # Balloon detection with PI controller
    Extremum_Seeking.cpp    # Extremum seeking control
    force_to_ESC_input.cpp # Force -> PWM conversion
    inv_kine.cpp            # Inverse kinematics
    old_cam.cpp             # Legacy camera processing

controls/ (Python)          # Control drivers
  controls/
    esc_driver.py           # ESC PWM via pigpio
    mode_switcher.py        # Auto/Manual selector
    baro_control.py         # Barometer-based control
    Bomber_Cntrl.py         # Bomber control logic
    bomber_mux.py           # Bomber multiplexer
    net_servo.py            # Network servo control
    random_walk.py          # Random walk algorithm
    rudolph_mode_switcher.py # Rudolph mode switcher

camera/                     # Camera integration
  camera/
    camera_test.py          # Camera test utilities

manual_control/             # Manual operation
  manual_control/
    joy_to_esc_input.py     # Joystick -> motor mapping
    manual_bridge.py        # Manual control bridge

launch/                     # Launch files
  updated_launch.py         # Main launch file
  autonomous_launch.py      # Autonomous mode launch
  cpp_auto_launch.py        # C++ autonomous launch
  drone_manual_launch.py    # Manual mode launch
  bomber_launch.py          # Bomber launch
  Rudolph_Launch.py         # Rudolph configuration launch
  arming.py                 # ESC arming sequence
  program_esc.py            # ESC programming
  BNO085_calibrate.py       # IMU calibration
  Extremum_test.py          # Extremum seeking test
  LED_test.py               # LED test
  Servo_Test.py             # Servo test
  test_launch.py            # General test launch
```

```
  data_scripts/              # Data analysis tools
    extract_and_plot.py      # Extract and visualize data
    extract_to_excel.py      # Export data to Excel
    export_to_excel.py       # Excel export utilities
    imu_motor_dashboard.py   # Real-time IMU/motor dashboard
    thrust_analysis.py       # Thrust analysis tools

  bag_files/                 # ROS 2 bag files
    Barometer_ros2bag/       # Barometer recordings
    rosbag2_2026_02_11-12_11_47/
    rosbag2_2026_02_11-12_19_11/

  logs/                      # Log files and recordings
```

12.3 Configuration Parameters

Motor Calibration (`configs/motor_calibration.yaml`):

```
motors:
  motor_1:
    gpio_pin: 27
    min_pwm_us: 1100
    max_pwm_us: 1900
    center_pwm_us: 1500
    reversed: false
    thrust_coefficient: 0.015  # N per PWM unit
```

PID Gains (`configs/pid_gains.yaml`):

```
pi_controller:
  yaw:
    kp: 0.5
    ki: 0.1
  altitude:
    kp: 0.3
    ki: 0.05
  camera_frame:
    center_x: 320
    center_y: 240
    error_threshold: 50
```

12.4 Logging

- ROS 2 logs: `~/.ros/log`

- Custom sensor logs: `~/blimp_ws/blimp_src/logs/imu_YYYYMMDD_HHMMSS.csv`
- ROS 2 bag recording: `ros2 bag record -a`

13 T13 — How the System Works

13.1 System Architecture

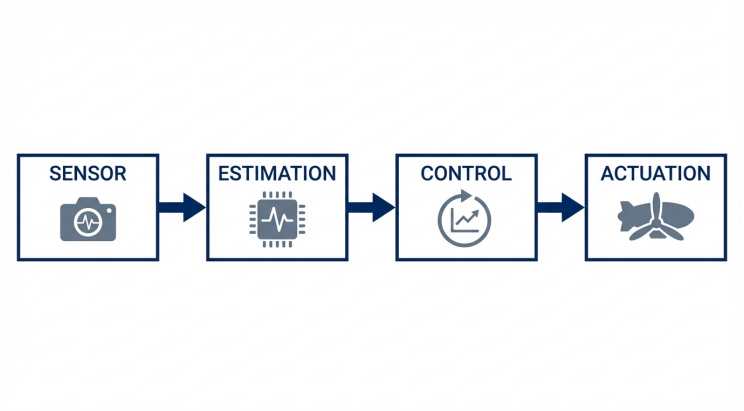


Figure 16: BLIMP system architecture overview.

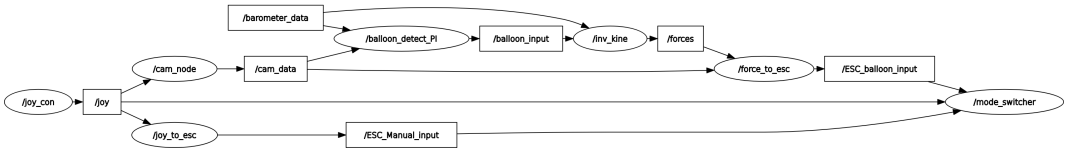


Figure 17: ROS 2 computation graph (`rqt_graph`).

```
SENSOR LAYER
  IMU: BNO055 @ 200Hz    -> /imu_data
  Barometer: BMP390 @ 50Hz -> /barometer_data
  Camera: 30fps RGB 640x480 -> /camera/detections
        |
ESTIMATION LAYER
  Kalman Filter (Sensor Fusion)
  State: [x,y,z, phi,theta,psi, vx,vy,vz, wx,wy,wz]
  Publishes: /state_estimate
        |
CONTROL LAYER
  PI Controller (vision-based yaw + altitude)
  Inverse Kinematics: F -> u = T_pseudoinverse * F
  Mode Switch: MANUAL <-> AUTONOMOUS
  Output: /esc_input (1000-2000 us PWM)
        |
```

```
ACTUATION LAYER
  ESC Driver (pigpio) -> 4x PWM @ 50Hz
```

13.2 Control Loop (50 Hz)

The BLIMP control system operates at 50 Hz, synchronizing sensor inputs from multiple sources operating at different rates. The system fuses high-frequency IMU data (200 Hz) with lower-frequency altitude measurements (50 Hz) and vision detections ($\sim$30 fps) to generate coordinated motor commands.

Sensor Fusion and Data Acquisition

The control loop begins by asynchronously acquiring data from three independent sensor streams. The BNO085 inertial measurement unit (IMU) provides 9-DOF orientation at 200 Hz, continuously updated via ROS 2 callbacks. The BMP390 barometric pressure sensor samples altitude at 50 Hz, providing vertical position feedback. The Raspberry Pi Camera coupled with YOLOv5 deep learning model detects the balloon target at approximately 30 fps, publishing centroid coordinates in the camera frame. These heterogeneous sensor rates are temporally synchronized through the Kalman filter, which fuses measurements into a consistent state estimate running at the control loop frequency.

State Estimation via Kalman Filtering

Prior to control actuation, the system estimates the complete 12-dimensional state vector: $\mathbf{x} = [x, y, z, \phi, \theta, \psi, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z]^T$, comprising 6-DOF pose (position and orientation) and 6-DOF velocities (linear and angular). The Kalman filter recursively predicts state evolution using blimp dynamics and corrects predictions using sensor measurements. IMU quaternion outputs are converted to Euler angles and fused with barometric altitude to provide altitude-corrected orientation estimates. This filtering reduces sensor noise and provides state derivatives (velocities) not directly measured, essential for smooth control law computation.

Vision-Based PI Control

Upon state estimation, the control layer evaluates balloon detection status. If the YOLOv5 detector reports a valid balloon with confidence > 0.5, the system computes two independent PI control laws:

Yaw Control: The horizontal image coordinate error $e_u = u_{\text{centroid}} - u_{\text{center}}$ quantifies lateral deviation from frame center. A proportional-integral controller regulates this error:

$$\psi_{\text{cmd}} = K_{p,\psi} \cdot e_u + K_{i,\psi} \cdot \int e_u \, dt$$

This command rotates the blimp (yaw moment) to center the balloon horizontally in the camera view.

Altitude Control: The vertical error $e_z = z_{\text{target}} - z_{\text{measured}}$ from barometric altitude drives altitude regulation:

$$f_{z,\text{cmd}} = K_{p,z} \cdot e_z + K_{i,z} \cdot \int e_z \, dt$$

This command adjusts vertical thrust to maintain desired altitude. Distinct proportional gains ($K_{p,\text{up}}, K_{p,\text{down}}$) can be applied for asymmetric up/down dynamics.

If the balloon is undetected (confidence < 0.5 or centroid missing), the system defaults to a search mode: yaw command becomes zero (hover to reduce drift), and altitude control operates in barometric-hold mode to maintain current altitude until re-detection.

Inverse Kinematics and Motor Allocation

The desired control outputs (yaw moment, vertical force) are converted to individual motor pulse-width modulation (PWM) commands via inverse kinematics. The system solves:

$$\mathbf{u} = \mathbf{T}^{\dagger} \left(\mathbf{F}_{\text{desired}} - \mathbf{F}_{\text{buoyancy}} \right)$$

where $\mathbf{T}^{\dagger}$ is the pseudo-inverse of the motor thrust allocation matrix, incorporating motor mounting positions and thrust directions. The resulting 4-dimensional PWM vector $\mathbf{u} = [u_1, u_2, u_3, u_4]^T$ is saturated to ESC operational bounds.

Mode Switching and Actuation

The mode switcher node arbitrates between manual and autonomous control paths. In **manual mode** (Xbox controller), joystick input is directly mapped to ESC commands via linear scaling. In **autonomous mode** (right bumper pressed), the inverse kinematics PWM output is used. The final arbitrated **u** vector is published to the ESC driver.

Finally, the ESC driver translates the PWM vector into GPIO hardware signals. The ‘pigpio‘ daemon generates 50 Hz square waves on GPIO pins 5, 6, 13, and 26, with pulse widths corresponding to desired motor speeds. This 20 ms control cycle ensures stable closed-loop operation and responsive disturbance rejection.

Listing 1: Control Loop Pseudocode (50 Hz Cycle)

```
while system_running:
    # 1. Asynchronous Sensor Data Acquisition
    # IMU callbacks @ 200Hz, Barometer @ 50Hz, Camera @ ~30Hz
    imu_euler = /imu_data  # [roll, pitch, yaw] in degrees
    altitude = /barometer_data  # relative altitude in meters
    balloon_centroid = /camera/detections  # [u, v] in pixels

    # 2. State Estimation: Fuse heterogeneous sensor rates
    state_estimate = KalmanFilter.update(
        imu_measurement=imu_euler,
        altitude_measurement=altitude,
```

```
        camera_measurement=balloon_centroid
    )
    # Output: 12-DOF pose + velocity estimate

    # 3. Vision-Based Feedback Control
    if balloon_detected and confidence > 0.5:
        # Yaw error in image frame
        error_u = balloon_centroid[0] - FRAME_CENTER_X  # pixels
        yaw_cmd = K_p_yaw * error_u + K_i_yaw * integral_error_yaw

        # Altitude error in world frame
        alt_error = TARGET_ALTITUDE - altitude  # meters
        heave_cmd = K_p_alt * alt_error + K_i_alt * integral_error_alt
    else:
        # Balloon lost or low confidence: hold current altitude, cease yaw
        yaw_cmd = 0.0
        heave_cmd = 0.0
        # Optionally execute search pattern

    # 4. Inverse Kinematics: Convert control commands to motor forces
    desired_force = [surge_cmd, sway_cmd, heave_cmd, yaw_cmd, roll_moment,
    pitch_moment]
    esc_pwm = inv_kinematics.solve(desired_force)
    # Output: [u_1, u_2, u_3, u_4] in microseconds [1000, 2000]

    # 5. Mode Arbitration
    if mode == MANUAL:
        # Override with joystick input
        esc_pwm = joystick_to_esc(joy_input)
    elif mode == AUTONOMOUS:
        # Use computed inverse kinematics output
        pass

    # 6. ESC Actuation via GPIO PWM @ 50Hz
    for motor_id in range(1, 5):
        gpio_pin = GPIO_PINS[motor_id]
        pwm_microseconds = esc_pwm[motor_id]
        pigpio.hardware_PWM(gpio_pin, 50, pwm_microseconds)

    # Sleep for 20ms to maintain 50Hz loop rate
    sleep(1.0 / 50.0)
```

—

13.3 Vision Autonomy Pipeline (YOLOv5)

Autonomous balloon tracking leverages a trained YOLOv5 small (YOLOv5s) deep neural network optimized for edge deployment on Raspberry Pi 4. The vision pipeline runs on the onboard single-board computer, requiring no external computation, enabling persistent autonomous operation within radio range.

Real-Time Object Detection

The object detection pipeline processes camera frames captured at 30 fps from the Raspberry Pi Camera Module v2 (IMX219 sensor, CSI interface). The YOLOv5s model, quantized to 8-bit integer precision for CPU inference efficiency, processes 640×480 RGB frames. The model predicts bounding boxes $[x_1, y_1, x_2, y_2]$ (top-left and bottom-right corners) and per-class confidence scores for all detected objects. A confidence threshold of 0.5 filters low-confidence predictions, mitigating false positives. For each detection passing the threshold with class ID matching "balloon", the centroid $(x_c, y_c) = \left(\frac{x_1+x_2}{2}, \frac{y_1+y_2}{2}\right)$ is computed and published to the '/camera/detections' ROS 2 topic at approximately 10 Hz (inference latency ∼80–150 ms on Pi 4 CPU).

Detection Filtering and Validation

To ensure robust autonomous operation, detected bounding boxes are validated against physical and geometric constraints:

- **Confidence Threshold:** Only detections with confidence > 0.5 are retained. This threshold balances sensitivity (catching distant balloons) with specificity (avoiding false alarms from visual ambiguities).
- **Spatial Constraints:** Bounding box width and height are validated. Detections with area $< A_{\text{min}}$ are rejected as false positives or noise. Conversely, boxes exceeding A_{max} may indicate the blimp has collided or extremely close approach; such cases trigger a safety "retreat" command.

Centroid-Based Navigation

Upon valid detection, the centroid (u_c, v_c) is transformed into control commands via quadrant-based logic. The image frame is partitioned into 5 regions: 4 quadrants (Q1–Q4) and a center region. The frame center is defined as $(u_{\text{center}}, v_{\text{center}}) = (320, 240)$ for a 640×480 frame. Offsets from center trigger corresponding control actions:

Region	Centroid Location	Yaw Command	Vertical Command
Q1 (Northeast)	$u > 320, v < 240$	Right (positive yaw)	Climb (positive f_z)
Q2 (Northwest)	$u < 320, v < 240$	Left (negative yaw)	Climb
Q3 (Southeast)	$u > 320, v > 240$	Right	Descend (negative f_z)
Q4 (Southwest)	$u < 320, v > 240$	Left	Descend
Center	$\lvert u - 320\rvert < \Delta, \lvert v - 240\rvert < \Delta$	No yaw	Hold altitude

Table 6: Quadrant-Based Navigation Logic for Vision-Guided Autonomous Flight

This quadrant approach provides intuitive, robust navigation: if the balloon appears in the top-left of the camera, the blimp turns left and climbs. The PI controller continuously refines yaw and altitude to center the balloon, maximizing approach stability.

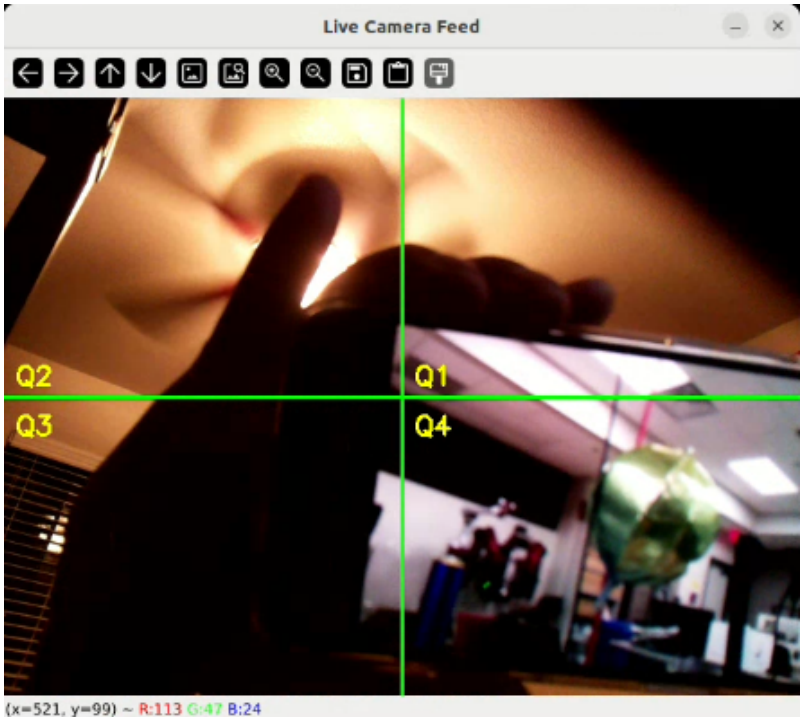


Figure 18: YOLOv5 balloon detection output overlaid on camera frame, showing bounding box $[x_1, y_1, x_2, y_2]$, centroid marker, and confidence score. Green box indicates high-confidence detection; centroid tracking enables real-time position feedback for autonomous control.

Autonomy Pipeline Summary

The complete vision-driven autonomy loop consists of six sequential stages:

1. **Frame Capture:** Raspberry Pi Camera Module v2 captures RGB frame at 30 fps (33 ms period), resolution 640×480 pixels.

2. **YOLOv5 Inference:** Deep learning model processes frame, predicting all objects in scene. Inference time: 80–150 ms on Raspberry Pi 4 CPU.

3. **Confidence Filtering:** Detections with confidence score < 0.5 or class $\neq$ "balloon" are discarded. Remaining detections are candidates for control input.

4. **Centroid Computation:** For each valid detection, compute bounding box centroid:

$$\mathbf{c} = \begin{bmatrix} u_c \\ v_c \end{bmatrix} = \begin{bmatrix} (x_1 + x_2)/2 \\ (y_1 + y_2)/2 \end{bmatrix}$$

   Centroid is in pixel coordinates, image origin at top-left.

5. **Quadrant Evaluation:** Compare centroid to frame center (320, 240). Determine which of 5 regions (Q1, Q2, Q3, Q4, or Center) contains the centroid. Assign preliminary yaw and altitude commands based on Table 6.

Limitations and Failure Modes

The vision autonomy pipeline exhibits the following limitations:

- **Lighting Dependency**
- **Training Dataset Specificity**
- **Computational Latency**
- **Occlusion**

14 T14 — How to Run Each Capability

14.1 Remote Access via SSH (PuTTY on Windows)

Overview: This subsection covers connecting to the Raspberry Pi from a Windows machine using PuTTY, then running the `updated_launch.py` script to start the BLIMP system remotely.

14.1.1 Step 1: Install PuTTY on Windows

1. Download PuTTY from: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
2. Download the Windows Installer (e.g., `putty-64bit-0.78-installer.msi`)
3. Run the installer and follow the default installation prompts.
4. Verify installation: Open Command Prompt or PowerShell and type `putty` to launch.

5. Alternative: Use `putty.exe` directly from the installation directory (default: `C:\Program Files (x86)\PuTTY\`).

Note: Windows 10/11 also includes OpenSSH client in PowerShell. Advanced users can use `ssh` command directly in PowerShell instead of PuTTY.

14.1.2 Step 2: Find the Raspberry Pi's IP Address

Before connecting via SSH, you must identify the Raspberry Pi's IP address on the network.

Option A: From the Lab Network (Recommended)

Connect the Raspberry Pi to your lab WiFi or Ethernet. Then:

```
# From any Linux/Mac terminal or SSH session:
arp-scan --localnet | grep -i raspberry
# OR
nmap -sn 10.0.0.0/24  # Scan for devices; look for "4c:xx:xx:xx:xx:xx" (Pi
    MAC)
```

Option B: Via Serial Console

Connect a USB-to-serial adapter to the Raspberry Pi's UART pins (GPIO 14/15) and open with `screen` or `minicom` at 115200 baud. Login and run:

```
hostname -I
```

Option C: Check Your Router

Log into your WiFi router's admin panel (`192.168.1.1` or `10.0.0.1`). Look for connected devices and find the Raspberry Pi by MAC address or hostname (`raspberrypi`).

Example IP: Assume the Raspberry Pi is at `10.0.0.50`.

14.1.3 Step 3: Configure and Launch PuTTY

1. **Open PuTTY.**
2. **Host Name:** Enter `170.20.10.2` (or your Pi's actual IP).
3. **Port:** Leave as `22` (SSH default).
4. **Connection Type:** Select `SSH`.
5. **(Optional) Save Session:** Under `Session`, enter a name like `BLIMP-RPi` and click `Save`. This allows you to reconnect quickly later without re-entering the IP.
6. **Click** `Open` to connect.

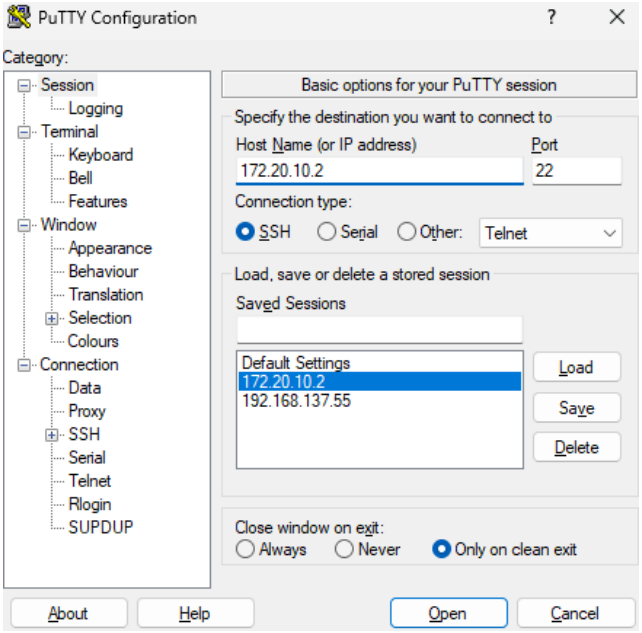


Figure 19: PuTTY configuration for Raspberry Pi SSH connection.

14.1.4 Step 4: Login to Raspberry Pi

PuTTY will open a terminal window. You should see:

```
login as:
```

Enter your Raspberry Pi credentials:

```
login as: pi
pi@172.20.10.2's password: raspberry
```

After successful login, you will see the prompt:

```
pi@raspberrypi:~ $
```

Security Note: If this is a persistent lab system, **change the default password** immediately:

```
passwd
# Enter new password (twice)
```

Note: ASU internet or any public internet wont allow you to connect to another machine. So, for ssh please make sure your own native machine and the pi is in same network and neither of them are in a public network.

14.1.5 Step 5: Navigate and Run `updated_launch.py`

Once logged in, run the BLIMP launch script:

```
# First, source the ROS 2 workspace
source ~/blimp_ws/install/setup.bash

# Navigate to the launch directory
cd ~/blimp_ws/src/launch

# Verify the file exists
ls -la updated_launch.py

# Run the launch script (Python)
ros2 launch updated_launch.py
```

Expected Output:

```
[INFO] [launch]: Starting BLIMP Launch Sequence...
[INFO] [sensors_node]: IMU node started on /imu_data
[INFO] [sensors_node]: Barometer node started on /barometer_data
[INFO] [camera_node]: Camera publisher started
[INFO] [esc_driver]: ESC driver initialized
...
```

Verify Nodes Running: In a **second PuTTY window**, connect again and run:

```
source ~/blimp_ws/install/setup.bash
ros2 node list
```

You should see active nodes like `/imu_reader`, `/barometer_reader`, `/camera_publisher`, etc.

14.1.6 Step 6: Troubleshooting SSH Connection

Error	Likely Cause	Solution
"Connection refused"	SSH not running on Pi	Enable SSH: `sudo raspi-config` → Interface Options → SSH
"No route to host"	Wrong IP or unreachable network	Verify IP with `hostname -I` on Pi or router admin panel
"Permission denied (publickey, password)"	Wrong credentials or SSH config	Use default `pi:raspberry`; check `/etc/ssh/sshd_config`

Error	Likely Cause	Solution
"PuTTY hangs after login"	SSH timeout or excessive load	Restart Pi; SSH again; or attach serial console
`updated_launch.py` not found	File not in correct directory	Verify directory with `pwd` and `ls`; clone repo if needed
`ImportError: No module named rclpy`	ROS 2 not sourced	Run `source /opt/ros/humble/setup.bash` before launch

14.1.7 Bonus: Hardware Configuration for SSH

Ensure SSH is permanently enabled on the Raspberry Pi by editing the system configuration:

```
sudo raspi-config
# Navigate to: Interface Options -> SSH
# Enable SSH (if not already enabled)
# Select <Finish> and confirm reboot

# Verify SSH is running after reboot:
sudo systemctl status ssh
# Output should show: active (running)
```

14.1.8 Bonus: Persistent SSH Sessions with `screen` or `tmux`

To keep the `updated_launch.py` running even after PuTTY closes:

```
# On the Raspberry Pi via SSH:
screen -S blimp_main
source ~/blimp_ws/install/setup.bash
cd ~/blimp_ws/src/launch
python3 arming.py
ros2 launch updated_launch.py

# To detach (keep running): Ctrl+A, then D
# To reattach: screen -r blimp_main
# To list sessions: screen -ls
```

14.2 1. Sensor Bring-up and Verification

Component Locations:

- IMU: `blimp_src/sensors/sensors/BNO085.py`

- Barometer: `blimp_src/sensors/sensors/barometer.py`
- Camera: `blimp_src/sensors_cpp/src/old_cam.cpp`

Test IMU (BNO085):

```
source ~/blimp_ws/install/setup.bash
ros2 run sensors read_imu          # Terminal 1
ros2 topic echo /imu_data          # Terminal 2
ros2 topic hz /imu_data            # Check 200 Hz
```

Pass Criteria: Z-accel ≈ 9.81 m/s^2 at rest ± 0.2, gyro < 0.05 rad/s stationary, 200 Hz ± 10 Hz update rate.

Test Barometer (BMP390):

```
ros2 run sensors read_barometer    # Terminal 1
ros2 topic echo /barometer_data    # Terminal 2
ros2 topic hz /barometer_data      # Check 50 Hz
```

Pass Criteria: Height $= 0.0 \pm 0.3$ m on ground, 50 Hz rate, noise < 0.5 m std-dev over 10 s.

Test Camera:

```
ros2 run sensors_cpp old_cam          # Terminal 1
ros2 topic hz /camera/image_raw       # Terminal 2 -> 30 Hz
ros2 run sensors_cpp detect_node      # Terminal 3
ros2 topic echo /cam_data             # Terminal 4
```

Pass Criteria: 30 fps camera, detection latency < 100 ms, centroid error $< 5\%$ frame width (< 32 px), confidence > 0.75 with balloon visible.

14.2.1 2. Hover Control Demo

Pre-flight:

1. Battery ≥ 7.4 V (verify battery checker)
2. Propellers intact, ESCs powered
3. Pi SSH responsive
4. Tether attached to support pole

Arm ESCs:

```
source ~/blimp_ws/install/setup.bash
python3 ~/blimp_ws/src/launch/arming.py   # Wait for beeps
```

Launch and Hover:

```
ros2 launch updated_launch.py      # Terminal 1
ros2 node list                     # Terminal 2 -> Verify all nodes
ros2 topic echo /barometer_data    # Terminal 3 -> Monitor altitude
ros2 bag record /imu_data /barometer_data /esc_input  # Terminal 4 (
    optional)
```

Pass Criteria: Sustained hover > 60 s, altitude variation < 50 cm (std-dev), control responsive (joystick response < 500 ms), motors smooth (no grinding).

—

14.2.2 3. Camera Bring-up and Streaming Verification

Hardware Check:

```
libcamera-hello -t 5s    # Visual check; if blurry, focus lens
```

Launch and Monitor:

```
source ~/blimp_ws/install/setup.bash
ros2 run sensors_cpp old_cam       # Terminal 1
ros2 topic hz /camera/image_raw    # Terminal 2 -> 30 Hz
ros2 run sensors_cpp detect_node   # Terminal 3
ros2 topic echo /cam_data          # Terminal 4 (with balloon visible)
ros2 topic hz /cam_data            # Terminal 5 -> 5-10 Hz
```

Pass Criteria: Camera 30 fps, detection 5–10 Hz, latency < 150 ms, centroid error < 32 px, confidence jitter < 0.10, no spurious detections ($< 5\%$ false positive rate).

—

14.2.3 4. Vision Autonomy Demo

Setup: Balloon ~ 2 m away, good lighting, blimp tethered, 5 m clear airspace.

Launch:

```
python3 ~/blimp_ws/src/launch/arming.py       # Arm ESCs
ros2 launch updated_launch.py                 # Terminal 1
# Use joystick to climb to ~1m, hold 10 sec (MANUAL mode)
# Press A -> switch to AUTONOMOUS mode
```

Monitor Autonomy:

```
ros2 topic echo /cam_data                  # Terminal 2
ros2 topic echo /balloon_detect_PI         # Terminal 3
ros2 bag record /imu_data /barometer_data /cam_data /esc_input \
  -o autonomy_test_$(date +%s)             # Terminal 4
```

Expected Sequence:

1. 0–3 s: Balloon detected, centroid in ‘/camdata‘
2. 3–10 s: Yaw error decreases monotonically (center balloon horizontally)
3. 5–15 s: Altitude adjusts to target
4. 10–15 s: Lock achieved (centroid < 50 px error, altitude < 0.3 m error)
5. 15–30 s: Stable lock hold ≥ 5 s

Pass Criteria: Detection within 3 s, yaw error monotonic decrease < 10 s to lock, altitude stability < 0.3 m, time-to-lock < 15 s, hold ≥ 5 s, no divergent oscillations.

—

14.2.4 5. Cage Gate Actuation Demo

Hardware Configuration:

- **Stepper Motor:** GPIO pins
- **Software:** `blimp_src/controls/controls/net_servo.py`

Manual Mode (Joystick Control):

```
source ~/blimp_ws/install/setup.bash
ros2 launch updated_launch.py          # Terminal 1
ros2 topic echo /net_servo           # Terminal 2
```

Xbox Control:

- **B button:** Toggle gate open/close (stepper motor rotates)

Manual Operation:

1. Press B $\rightarrow$ stepper rotates CCW, gate opens (audible motor clicks)
2. Press B again $\rightarrow$ stepper rotates CW, gate closes
3. Repeat as needed

Expected Output:

```
/net_servo:
  net_servo: 0.0    # 0 = open, 1 = closed
  is_manually_controlled: True
  sonar_distance: None  # Not active in manual mode
```

Pass Criteria (Manual): Stepper rotates smoothly with clicks, gate physically opens/closes, gate_position toggles $0 \leftrightarrow 1$, response < 500 ms, motor holds position without drift.

—

Autonomous Mode (Sonar-Triggered Closure):

Launch Autonomy:

```
python3 ~/blimp_ws/src/launch/arming.py
sleep 2
ros2 launch updated_launch.py          # Terminal 1
# Manually hover blimp above cage opening
# Position balloon to enter cage cavity
```

Monitor Sonar and Gate:

```
ros2 topic echo /net_servo           # Terminal 2
ros2 topic echo /cam_data              # Terminal 3
```

Autonomous Sequence:

1. Park blimp with balloon near cage opening (manual throttle control)
2. Balloon enters cage cavity $\rightarrow$ sonar detects distance $< 20\,\text{cm}$
3. RPi receives sonar trigger signal (GPIO 24 ECHO pulse)
4. Stepper motor receives PWM command on GPIO [12, 16, 20, 21]
5. Gate rotates CW, closes over 5–10 seconds
6. gate_position transitions $0 \rightarrow 1$

Pass Criteria (Autonomous): sonar detects the balloon and the sensor becomes 0 to 1. The gate closes smoothly and the ball gets trapped inside.

Manual Override: At any time, operator can press B button to manually open gate, overriding autonomous closure state.

```
# Gate open mid-closure
Press B on Xbox controller -> stepper reverses, gate opens
```

15 T15 — Validation Evidence and Acceptance Tests

Appendix B summarizes the system-level verification tests conducted, including the test procedure, explicit pass criteria, and the corresponding evidence artifacts (e.g., logs, plots, and videos with commit references). This table serves as a traceability record(stability, sensor performance and actuation).

16 T16 — Known Issues and Troubleshooting Guide

16.1 Prioritized Open Issues

Pri	Issue	Severity	Status	Workaround
1	Helium leakage (0.5%/day)	HIGH	Open	Refill every 7–10 days
2	WiFi packet loss (5–10%)	MEDIUM	Open	Setup own wifi
3	PI yaw overshoot $> 30°$	LOW	Open	Kp $0.5 \rightarrow 0.3$ + damping
4	Baro noise ± 0.5 m	LOW	Open	Kalman filter

16.2 Troubleshooting by Symptom

Blimp does not respond to joystick:

1. Check `ros2 node list | grep game_controller`.
2. Check `ls /dev/input/` — should see `jsX`.
3. Check `ros2 topic echo /joy` — should see updates.
4. Check `ros2 node info /esc_driver` — subscribes to `/esc_input`.
5. Reconnect joystick, calibrate, check mode_switch.

Motors spin but blimp does not move:

1. Verify propeller alignment and balance.
2. Test with manual topic publish.
3. Check battery voltage.
4. Inspect for wire entanglement.

Blimp oscillates wildly:

1. Reduce Kp (yaw: $0.5 \rightarrow 0.3$, altitude: $0.3 \rightarrow 0.2$).
2. Verify 50 Hz control loop rate.
3. Balance rotors.

IMU reports erratic Euler angles:

1. Check `i2cdetect -y 1` — BNO085 at 0x28.
2. Verify BNO085 mode.
3. Separate ESC wires from I2C cables.

"pigpio daemon not running" error:

```
sudo systemctl start pigpiod
sudo systemctl enable pigpiod
```

17 T17 — Decisions Log (Tested and Abandoned)

As documented in **Appendix A**, multiple candidate approaches were evaluated and ultimately abandoned due to performance or integration constraints. The decision log summarizes what was tried, how each option was tested, the observed results, and the rationale for discarding it, along with brief notes on when revisiting the decision may be appropriate.

Part II

Team Management Report

18 M1 — Team Roster, Roles, and Ownership

18.1 Team Members

Name	Role	Ownership
Prajjwal	Software Development	Software Architecture, Integration
Fangrui	Software and model development	Vision Models, AI
Akshan	Mechanical	Mechanical Systems, Structure
Kaushik	Mechanical	Component Mounting, Simulation
Aadi	Electronics	Power Systems, Soldering
Richard	Mechanical design	Mechanism Design, Prototyping
Ishan	Electronics	Actuation, Assembly Integration
Toshan	Simulation	Physics Simulation, Environment
Nick	Mechanical design	Fabrication (3D Printing/Laser Cutting)
Prajval	Electronics and coding	Embedded Systems, Camera HW

19 M2 — Assigned Tasks and Current Status

19.1 Mechanical Subsystem

Task	Owner	Output	Status	Issue
SD Card Preparation	Kaushik	Working baseline	Done	None
Net Mechanism (Design)	Richard, Adi, Ishan	Open/Close Mech	Updates ongoing	Printing Parts
Cage Prototype V2	Richard	Prototype Assembly	In Progress	Iterations

Task	Owner	Output	Status	Issue
Motor Mount Design	Nick	CAD Files	Done	#–
Fabrication (Print/Cut)	Nick	Physical Parts	In Progress	#–

19.2 Electrical Subsystem

Task	Owner	Output	Status
Soldering Work	Aadi	Power Distribution	Done
ESC Configuration	Kaushik	Bidirectional Setup	Pending
Net Actuation (Elec)	Ishan	Servo/Motor Wiring	In Progress
Blimp Assembly	Ishan	Integrated Blimp	Pending
Camera Hardware	Prajval	RPi Cam Integration	Pending
Electronics Wiring	Prajval	Functional Circuit	In Progress

19.3 Software, Simulation, and Controls

Task	Owner	Output	Status
General Software Integration	Prajjwal	Core Stack	In Progress
Model Development	Fangrui	Vision Model	In Progress
System Simulation	Toshan	Sim Environment	In Progress
Camera Drivers	Prajval	Video Stream	Pending

20 M3 — Weekly Milestones: Plan vs. Actual

Wk	Dates	Planned	Deliverable/Status	Blockers
1–5	Jan 12 – Feb 13	Setup, Basic Mech/-Elec	*Completed*	None
6	**Feb 16 – 20**	**1. Pi Cam Streaming** **2. Vision Model Prep** **3. Cage Design (V2)**	Stream running (60fps) Dataset collected CAD ready for print	WiFi Latency No Available Camera

Wk	Dates	Planned	Deliverable/Status	Blockers
7	Feb 23 – 27	**4. Detection Algo Integration** **5. Localization Setup** **6. Cage Fabrication**	Node publishing bbox Pose estimate (x, y, z) Cage V2 mounted	Lighting noise
8	Mar 02 – 06	**7. Controller Tuning** **8. System Integration**	Stable hover (PID) Full stack launch	Battery sag
9	Mar 09 – 13	*Spring Break*	—	—
10	Mar 16 – 20	Tethered Flight Tests	Flight logs	Safety tether
11	Mar 23 – 27	Autonomous Circuit	Demo video (1 lap)	Drift
12	Mar 30 – Apr 03	Refinement & Optimization	Max speed test	–
13	Apr 06 – 10	Final Demo Prep	Full rehearse	None
14	Apr 13 – 17	Final Demo	**Live Demo**	None
15	Apr 20 – 24	Documentation	Final Report	None

21 M4 — Planned Daily Working Hours and Cadence

21.1 Team Schedule

Member	Planned	Actual
Prajjwal Fangrui	M–F	avg 3 h/d
Nick	M-F	avg 2 h/d
Richard	M-W-F	avg 2 h/d
Aadi Toshan	T-Th-F	avg 2 h/d
Ishan	T-Th-F	avg 2 h/d
Akshan	F-S-Sun	Avg 2 h/d
Kaushik	M-W-F	avg 2 h/d
Prajval	T-Th-F-S-Sun	avg 2 h/d

21.2 Meeting Cadence

Meeting	Frequency	Time	Purpose
Standup	Daily M–F	10:00 (15 min)	Status, blockers
Flight Test Review	1×/wk (Fri)	14:00 (1 h)	Data analysis
Weekly Report	1×/wk (Fri)	Async	GitHub updates

21.3 Blocker Escalation

1. **Level 1 (Team):** Resolve within 24 hrs.
2. **Level 2 (PI):** Flag to Dr. Xi Yu.
3. **Level 3 (External):** Hardware fault.

22 M5 — Process and Quality Control

22.1 Definition of Done

A feature is "done" only when:

Code:

- Committed to lab GitHub repo.
- Passes `ros2 build` without errors.
- Follows ROS 2 naming conventions (snake_case).
- Includes docstrings/comments.
- Unit tests pass.

Integration:

- Node launches via `ros2 launch`.
- Expected topics appear in `ros2 topic list`.
- No missing dependencies or runtime errors.
- Tested with existing nodes.

Testing:

- Acceptance test defined with pass criteria.
- Test executed with evidence collected.

Documentation:

- Function/node purpose documented.
- Input/output topics listed.
- Config parameters documented.
- Known issues noted.

22.2 Code Review and Merge Policy

1. Create `feature/task-name` branch off `main`.
2. Commit frequently with clear messages.
3. Open Pull Request with description.
4. Primary reviewer: Prajjwal Dutta (1 approval required).
5. Review criteria: conventions, no hardcoded values, error handling, minimal deps.
6. Rebase + merge to `main` after approval.
7. Tag release commits (e.g., `blimp_report_02_2026`).

22.3 Hardware Change Control

- **Level 1 (Minor):** CAD/wire change → owner reviews, Slack notify, commit.
- **Level 2 (Moderate):** Component substitution → update BOM + wiring.

Appendix

A Appendix A: Decisions Log

Tried	How Tested	Result	Reason Discarded	Revisit Notes
YOLOv5 Full (yolov5l)	Inference test	250 ms/frame	Too slow for 30 fps	GPU or quantization
Arducam Tof	Assembly + ROS2	Only Depth Data, No RGB image; CPU 100%	Not standalone	Upgraded to Pi camera 2

Tried	How Tested	Result	Reason Discarded	Revisit Notes
Open-loop thrust	Early tests	Unpredictable	No feedback	System ID
Wired Ethernet	Prototype	Cable management	WiFi adequate	If WiFi critical

B Appendix B: Acceptance Tests Summary

Test	Procedure	Pass Criteria	Evidence
Hover Stability	Arm, install props, hover at 0.5 m, record 60 s	Std-dev < 0.1 m, drift < 0.2 m	`videos/hover_demo.mp4` (commit: `f88c267`)
IMU Performance	Record IMU data during hover test	No significant drift, noise < 0.05 m/s^2	`plots/imu_data.jpg`, `plots/imu_motor_dashboard.png` (commit: `f88c267`)
Barometer Accuracy	Record altitude during hover	Error < 0.3 m from commanded	`logs/barometer_data.xlsx`, `plots/barometer_plot.png` (commit: `f88c267`)
Thrust Command Analysis	Record motor PWM commands during position control tests	Vertical motors active (U/D), lateral motors inactive (L/R)	`logs/PI_Data_2.txt`, `logs/PI_Data_3.txt`, `plots/thrust_commands_over_time` (commit: `f88c267`)
IMU + Motor Dashboard	Comprehensive sensor and control logging during two flight tests	All sensors operational, control responses within limits	`logs/data_record_4.txt`, `logs/data_record_5.txt`, `plots/imu_motor_dashboard.png` (commit: `f88c267`)
Cage Actuation	Command open/close 20 cycles	100% success rate, $<$ 2 s actuation time	`videos/cage_actuation.mp4` (commit: `f88c267`)
Quadrant Detection	Present targets in all 4 quadrants, 10 trials each	$> 90\%$ detection accuracy	`videos/quadrant_detection.mp4` (commit: `f88c267`)
Test Environment	Document conditions for all tests	Consistent Temperature, indoor, no wind	`test_environment.txt` (commit: `f88c267`)