Project Name: Chitti - Remote Robot Management

Team Name: Dread Fuel

Robot Hardware Interface Guide

This guide provides a detailed overview of the interface between the robot hardware systems and the Remote Robot Management Cloud System. It covers data flows, required hardware components, communication protocols, and integration best practices.

---

1. Overview

The robot hardware system connects to the cloud platform via a Python client (`robot_firebase_client.py`). This client is responsible for:

- Collecting telemetry and health data from the robot's sensors and controllers
- Sending data to Firebase Firestore in real time
- Listening for remote commands from the cloud and executing them on the robot

---

2. Required Hardware Components

- **Onboard Computer:** Raspberry Pi, NVIDIA Jetson, or similar (runs Python 3)
- **Microcontroller (optional):** Arduino, STM32, etc. for low-level motor/sensor control
- **Sensors:**
  - IMU (gyroscope, accelerometer) for orientation and movement
  - Wheel encoders or GPS for position tracking
  - Battery voltage/current sensor
  - Temperature sensor
- **Actuators:**
  - Motor drivers for wheels/legs
  - Servo motors for arms/joints
- **Network:** Wi-Fi or Ethernet for internet connectivity

---

3. Software & Communication Stack

- **Operating System:** Linux (Raspbian, Ubuntu) or Windows
- **Python 3.x** with `firebase-admin` package
- **Robot Control Software:** Custom Python scripts or ROS nodes for hardware abstraction
- **Cloud Client:** `robot_firebase_client.py` (provided)

---

4. Data Flow & Protocols

Telemetry & Health Data (Robot → Cloud)

- The client script collects data from sensors and controllers:
  - Position (x, y, z), orientation (degrees/radians)
  - Velocity (m/s)
  - Battery level (%), cycle count, temperature, error codes
- Data is sent to Firestore under `robots/{robotId}/telemetry` and `robots/{robotId}/health`
- Data is timestamped and can be sent at intervals (e.g., every 1–5 seconds)

Command Handling (Cloud → Robot)

- The client listens for new command documents in `robots/{robotId}/commands`
- Supported commands may include:
  - Move (direction, speed)
  - Stop
  - Change operational mode
  - Manipulate joints (angles/positions)
  - System power on/off
- Upon receiving a command, the client parses and executes it using the robot's control software
- After execution, the client updates the command status in Firestore (e.g., `executed`)

---

5. Integration Steps

1. **Install Python and Dependencies**
   - Install Python 3.x and `firebase-admin` on the onboard computer
2. **Configure Service Account**
   - Download `serviceAccountKey.json` from Firebase Console
   - Place it securely on the robot (never share or commit this file)
3. **Edit `robot_firebase_client.py`**
   - Set the correct `ROBOT_ID`
   - Integrate hardware-specific code to read sensors and control actuators
   - Replace simulated data with real sensor readings and control logic
4. **Run the Client**
   - Start the script: `python robot_firebase_client.py`
   - Monitor logs for successful data uploads and command execution

---

6. Data Structures (with Explanations)

Below are real-world examples of the data structures exchanged between the robot and the cloud. Each field is explained for clarity and to guide integration with actual hardware and software systems.

Telemetry Document (robots/

```
{
  "timestamp": 1700000000000,           // Unix epoch time in milliseconds when the data was recorded
  "position": {
    "x": 1.23,                          // X coordinate in meters (local or global frame)
    "y": 3.45,                          // Y coordinate in meters
    "z": 0.00                           // Z coordinate (altitude or height, if applicable)
  },
  "orientation": {
    "yaw": 90.0,                        // Heading in degrees (0-360, or -180 to 180)
    "pitch": 0.0,                       // Pitch angle in degrees
    "roll": 0.0                         // Roll angle in degrees
  },
  "velocity": 0.5,                      // Linear velocity in meters/second
  "angular_velocity": 0.0,              // Angular velocity in degrees/second
  "joint_states": {
    "shoulder": 0.12,                   // Example: joint angle in radians or degrees
    "elbow": -0.34,
    "wrist": 0.56
  },
  "status": "RUNNING"                  // Robot operational status (RUNNING, CHARGING, OFFLINE, ERROR)
}
```

Explanation:

- `timestamp`: When the telemetry was captured (for time-series analysis)
- `position`: Robot's current location (from odometry, GPS, or SLAM)
- `orientation`: Robot's heading and tilt (from IMU or sensor fusion)
- `velocity`: Forward speed; `angular_velocity`: rotational speed
- `joint_states`: Current angles/positions of key joints (for arms, legs, etc.)
- `status`: High-level state for dashboard and alerts

Health Document (robots/

```
{
  "timestamp": 1700000000000,
  "battery": 85,                        // Battery percentage (0-100)
  "cycle_count": 120,                   // Number of charge/discharge cycles
  "temperature": 32.5,                  // System or battery temperature in Celsius
  "error_code": 0,                      // 0 = OK, nonzero = error/fault code
  "voltage": 24.1,                      // Battery/system voltage in Volts
  "current": 1.2                        // Battery/system current in Amperes
}
```

Explanation:

- `battery`: Remaining battery percentage
- `cycle_count`: Useful for maintenance and battery health tracking
- `temperature`: Monitors for overheating or environmental issues
- `error_code`: For reporting hardware or software faults
- `voltage`/`current`: For advanced diagnostics and power monitoring

Command Document (robots/

```
{
  "timestamp": 1700000000000,
  "type": "move",                      // Command type: move, stop, set_joint, power, etc.
  "parameters": {
    "direction": "forward",            // For move: forward, backward, left, right
    "speed": 1.0                        // Speed in m/s
  },
  "status": "pending",                 // Command status: pending, executed, failed
  "issued_by": "user@example.com"      // (Optional) User or system that issued the command
}
```

Explanation:

- `type`: What action the robot should perform
- `parameters`: Arguments for the command (direction, speed, joint angles, etc.)
- `status`: Tracks command lifecycle for audit and troubleshooting
- `issued_by`: (Optional) For traceability and security

Note:

- These structures can be extended to include additional fields as needed (e.g., GPS accuracy, sensor health, custom commands).
- All numeric values should use SI units (meters, seconds, degrees, etc.) for consistency.
- Timestamps should be in UTC (Unix epoch milliseconds) for synchronization.

---

7. Network Interfaces

The robot hardware system communicates with the cloud platform over standard network interfaces. The following describes the typical setup and requirements:

Supported Network Interfaces

- **Wi-Fi (802.11x):** Most common for mobile robots; requires access to a secure wireless network.
- **Ethernet (RJ45):** Recommended for stationary robots or where high reliability/low latency is needed.
- **Cellular (4G/5G, optional):** For field robots operating outside Wi-Fi range; requires a compatible modem and data plan.

Network Configuration

- The robot must have outbound internet access to connect to Firebase services (HTTPS, port 443).
- DHCP is recommended for IP assignment, but static IPs can be used for advanced setups.
- Firewalls must allow outbound HTTPS traffic to `*.firebaseio.com` and `*.googleapis.com`.
- For security, restrict inbound connections to the robot unless remote SSH or diagnostics are required.

Security Considerations

- Use WPA2/WPA3 for Wi-Fi networks.
- Change default passwords on all networked devices.
- Regularly update firmware and OS to patch vulnerabilities.

---

8. Protocol Exchange Formats

The communication between the robot and the cloud is based on HTTPS REST API calls via the Firebase Admin SDK. The following describes the protocol and data exchange formats:

Transport Protocol

- **HTTPS (TLS 1.2+):** All data is encrypted in transit using industry-standard TLS.
- **RESTful API:** The robot client uses the Firebase Admin SDK, which internally makes REST API calls to Firestore.

Data Serialization

- **JSON:** All telemetry, health, and command data is serialized as JSON documents.
- **UTF-8 Encoding:** All text data is encoded in UTF-8.

Message Exchange Patterns

- **Telemetry/Health Upload:**
  - Robot sends a POST (add) request to Firestore with a new telemetry or health document.
  - Example: `robots/{robotId}/telemetry/{docId}`
- **Command Reception:**
  - Robot subscribes to Firestore for new command documents in `robots/{robotId}/commands`.
  - On receiving a new command, the robot parses the JSON, executes the action, and updates the command status.
- **Status Updates:**
  - Robot updates the command document with execution results (e.g., `status: executed`, `timestamp: <completed>`).

Example Exchange

1. **Robot Uploads Telemetry:**
   - Sends a JSON document as described in the Data Structures section.
2. **Cloud Issues Command:**
   - Dashboard writes a new command JSON document to Firestore.
3. **Robot Receives and Executes Command:**
   - Robot client reads the command, performs the action, and updates the status field in the same document.

Error Handling

- All API calls should be wrapped in try/except (Python) or equivalent error handling.
- On network failure, buffer data locally and retry until successful.
- Log all failed exchanges for diagnostics.

---

9. Best Practices & Troubleshooting

- **Secure the service account key** and never expose it publicly
- **Handle network interruptions** by buffering data and retrying uploads
- **Log all received commands and actions** for audit and debugging
- **Validate all incoming commands** before execution
- **Monitor system health** and send alerts for critical issues (e.g., low battery)
- **Test integration** with simulated commands before deploying on real hardware

---

10. Common Issues

- **No data in dashboard:** Check network connection and service account permissions
- **Commands not executed:** Ensure the client is running and listening for commands
- **Sensor errors:** Validate hardware connections and sensor drivers
- **Authentication errors:** Verify the service account key and Firebase project configuration

---

For further integration support, contact the Engineering department or your robotics lead.