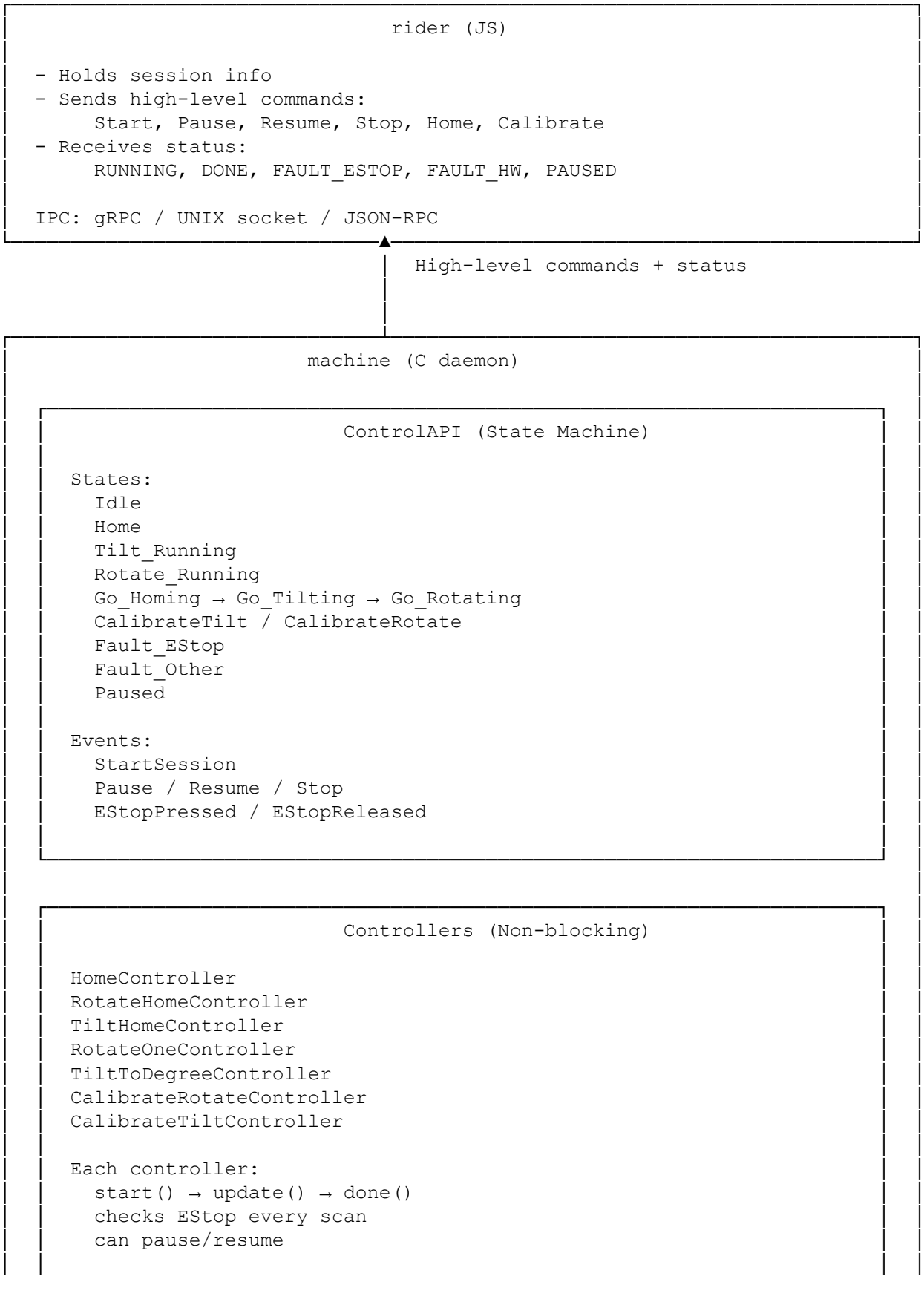
IPC ARCHITECTURE DOCUMENT

Rider ↔ Machine Control System (PLC-Style Architecture)

OVERALL SYSTEM ARCHITECTURE DIAGRAM



```
┌──────────────────────────────────────────────────────────────┐
│                        Device Layer                          │
│                                                              │
│  RotateAxis                                                  │
│     - rotateOne(dir)                                         │
│     - stop()                                                 │
│     - isHome()                                               │
│                                                              │
│  TiltAxis                                                    │
│     - tiltOne(dir)                                           │
│     - stop()                                                 │
│     - readPositionDegrees()                                  │
│                                                              │
│  Safety                                                      │
│     - isEStopActive()                                        │
│                                                              │
└──────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────┐
│                        IO Layer (Typed IO)                   │
│                                                              │
│  DigitalInput: rotateHome, tiltHome, estop                   │
│  AnalogInput: tiltPosition                                   │
│  RelayOutput: rotateRelay, rotateDir, tiltRelay, tiltDir     │
│                                                              │
└──────────────────────────────────────────────────────────────┘

┌──────────────────────────────────────────────────────────────┐
│                        HAL (RevPi Process Image)             │
│                                                              │
│  scanInputs(): read DI/AI                                    │
│  scanOutputs(): write RO/DO/AO                               │
│  forceAllOutputsSafe(): drop all relays                      │
│  readEStop(): DI3                                            │
│                                                              │
└──────────────────────────────────────────────────────────────┘

Main Loop (PLC-style):
  while(true):
     hal.scanInputs()
     if (estop):
         controlAPI.handleEStop()
         hal.forceAllOutputsSafe()
         continue
     controlAPI.update()
     hal.scanOutputs()
     sleep(10ms)
```

---

1. System Overview

The system consists of two cooperating containers:

1.1 Rider Container (JavaScript)

Acts as the **HMI / Orchestrator**.

- Holds session information
- Sends high-level commands to the machine
- Receives status updates
- Handles user inputs (Pause, Resume, Stop)
- Handles safety notifications (E-Stop, Faults)

1.2 Machine Container (C Daemon)

Acts as the **PLC CPU**.

- Runs a deterministic control loop
- Owns the ControlAPI state machine
- Executes Home, Tilt, Rotate, Calibrate
- Monitors hardware E-Stop
- Sends status updates to rider
- Ensures safety and sequencing

---

2. IPC Mechanism

2.1 Transport

- **gRPC** (recommended)
- Runs over **UNIX domain socket** or **localhost TCP**
- Strong typing via `.proto`
- Supports **bidirectional streaming** for status updates

2.2 Message Types

Commands (Rider → Machine)

Rider Command	Machine Function Called
`CalibrateTilt`	`CalibrateTilt()`
`CalibrateRotate`	`CalibrateRotate()`
`Home`	`Home()`
`Start { Session }`	`Session()`
`Pause`	`ControlAPI.pause()`
`Resume`	`ControlAPI.resume()`
`Stop`	`ControlAPI.stop()`

Status (Machine → Rider)

- `RUNNING`
- `PAUSED`
- `DONE`
- `FAULT_ESTOP`
- `FAULT_HW`
- `WARNING_NEEDS_CALIBRATION`
- `WARNING_NOT_HOME`

---

3. Machine ControlAPI State Machine

The machine daemon runs a PLC-style loop:

```
while(true):
    hal.scanInputs()
    if (EStop):
        handleEStop()
        hal.forceAllOutputsSafe()
        continue
    processIncomingCommands()
    controlAPI.update()
    hal.scanOutputs()
    sleep(10ms)
```

---

4. Machine Command Execution Logic

Below is the exact behavior for each command.

4.1 Home()

Rider Command

```
Home
```

Machine Behavior

```
Home() {
    HomeTilt();
    HomeRotate();
}
```

Sequence

1. TiltHomeController.start()
2. Wait until done
3. RotateHomeController.start()
4. Wait until done
5. Send status: `DONE`

Fault Handling

- If E-Stop → send `FAULT_ESTOP`
- If sensor timeout → send `FAULT_HW`

4.2 Calibration Commands

CalibrateTilt

```
CalibrateTilt()
```

- Move tilt axis to calibration reference
- Store calibration result in file
- Send `DONE`

CalibrateRotate

```
CalibrateRotate()
```

- Move rotate axis to calibration reference
- Store calibration result in file
- Send `DONE`

Fault Handling

If calibration fails → send `FAULT_HW`

4.3 Start Session

Rider Command

```
Start {
  Session: [
    Tilt(45),
    Rotate(CW),
    Rotate(CW),
    Rotate(CW)
  ]
}
```

Machine Behavior

```
Session() {
    CheckCalibrate();
    CheckHome();
    CheckSession();
    Tilt(degree);
    Rotate(dir) x many times;
    Send Done;
}
```

Detailed Sequence

1. CheckCalibrate()

- If either axis is not calibrated:
  - Send `WARNING_NEEDS_CALIBRATION`
  - Stop session
  - Send `DONE`

2. CheckHome()

- If not home:
  - Send `WARNING_NOT_HOME`
  - Stop session
  - Send `DONE`

3. CheckSession()

- Validate session structure
- If invalid → send `FAULT_HW` and stop

4. Execute Tilt

```
TiltToDegreeController.start(45)
```

Wait until done.

5. Execute Rotate x N

For each Rotate(CW):

```
RotateOneController.start(CW)
wait until done
```

6. Send `DONE`

---

5. Runtime Behavior During Session

The machine continuously checks:

5.1 Pause

If rider sends:

```
Pause
```

Machine:

- Enters `STATE_PAUSED`
- Controllers freeze internal progress
- Outputs are set to safe idle state
- Sends status: `PAUSED`

5.2 Resume

If rider sends:

```
Resume
```

Machine:

- Returns to previous state
- Controllers continue from paused state
- Sends status: `RUNNING`

5.3 Stop

If rider sends:

```
Stop
```

Machine:

- Stops all controllers
- Clears session
- Sends status: `DONE`

5.4 E-Stop

If hardware E-Stop is pressed:

- Immediately stop all motion
- Drop all relays
- Enter `FAULT_ESTOP`
- Send status to rider
- Wait for E-Stop release
- Rider must send `Home` or `Resume`

---

6. Rider Responsibilities

6.1 At Startup

1. Send `CheckCalibration`
    - If not calibrated → prompt user to run calibration

2

2. Send `CheckHome`
    - If not home → prompt user to run Home

6.2 Start Session

Send:

```
Start { Session: Tilt(45), Rotate(CW)x3 }
```

6.3 User Controls

- If user presses Pause → send `Pause`
- If user presses Resume → send `Resume`
- If user presses Stop → send `Stop`

6.4 Safety Handling

If machine sends:

```
FAULT_ESTOP
```

Rider must:

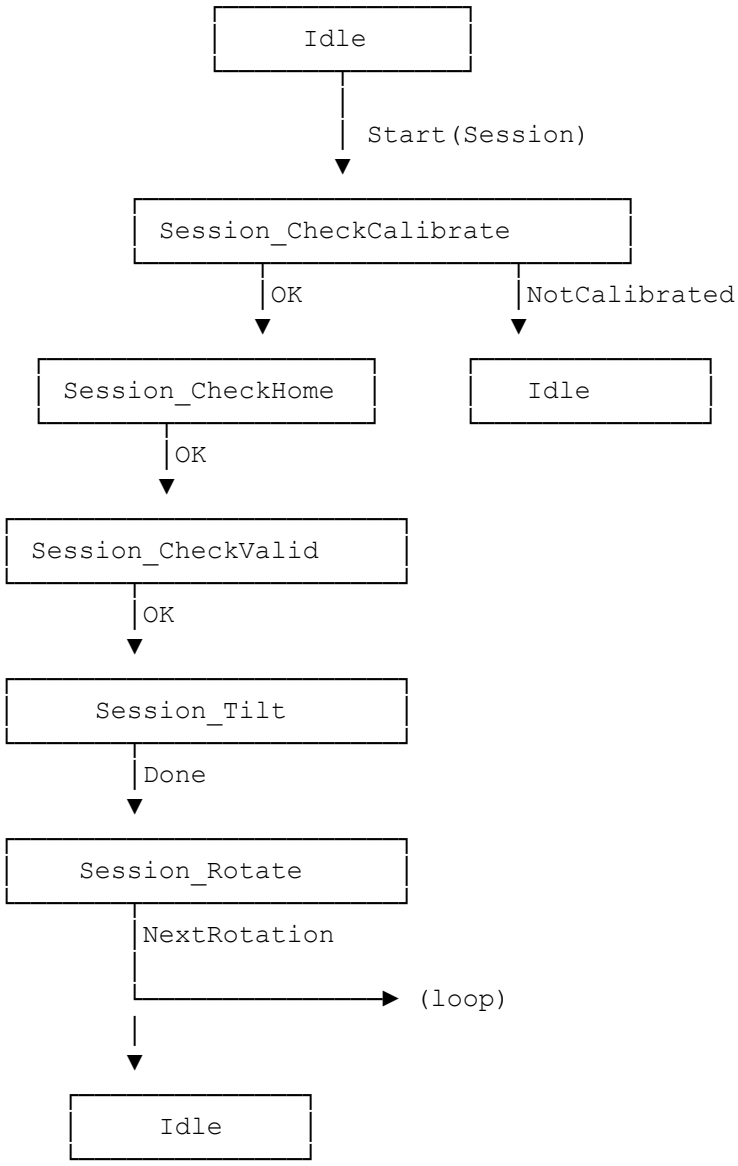
- Stop UI
- Notify user
- Wait for machine to recover

---

7. Summary of IPC Flow

```
Rider → Machine: Start Session
Machine → Rider: RUNNING

During session:
Rider → Machine: Pause
Machine → Rider: PAUSED

Rider → Machine: Resume
Machine → Rider: RUNNING

If E□Stop:
Machine → Rider: FAULT_ESTOP

When complete:
Machine → Rider: DONE
```

ControlAPI — UML State Machine Diagram



Pause / Resume / Stop Behavior

```
ANY_RUNNING_STATE ── Pause ──▶ Paused
Paused ── Resume ──▶ previous_state
Paused ── Stop ──▶ Idle
```

E-Stop Behavior

```
ANY_STATE ── EStopPressed ──▶ Fault_EStop
Fault_EStop ── EStopReleased + Resume ──▶ previous_state
Fault_EStop ── EStopReleased + Home ──▶ Home
```

Calibration & Home

```
CalibrateTilt ── Done ──► Idle
CalibrateRotate ── Done ──► Idle

Home ── TiltHome ──► RotateHome ──► Idle
```

Full Rider ↔ Machine Sequence Diagram

```
Rider → Machine: CMD_START { Tilt(45), Rotate(CW)x3 }
Machine → Rider: RESP_START_ACCEPTED

Machine → ControlAPI: StartSession()
ControlAPI: ValidateSession()
ControlAPI: CheckCalibrate()
ControlAPI: CheckHome()
ControlAPI → TiltController: start(45)

[PLC Loop]
  Machine → HAL: scanInputs()
  ControlAPI → TiltController: update()
  TiltController → HAL: actuator commands
  HAL: scanOutputs()
  TiltController → ControlAPI: done?

ControlAPI → RotateController: start(CW)

Rider → Machine: CMD_PAUSE
Machine → ControlAPI: Pause()
ControlAPI: state = Paused
ControlAPI → Controllers: pause()
Machine → Rider: RESP_PAUSED

Rider → Machine: CMD_RESUME
Machine → ControlAPI: Resume()
ControlAPI: state = previous_state
ControlAPI → Controllers: resume()
Machine → Rider: RESP_RESUMED

[PLC Loop for each rotation]
  Machine → HAL: scanInputs()
  ControlAPI → RotateController: update()
  RotateController → HAL: actuator commands
  HAL: scanOutputs()
  RotateController → ControlAPI: done?

ControlAPI: state = Idle
Machine → Rider: RESP_DONE
```
