This patch combines the detection logic, the virtual fallback, and the OSD alert system for iNav build. It bridges
`pitotmeter.c` with the navigation and OSD subsystems.
Part 1: Sensor Logic & Fallback (`src/main/sensors/pitotmeter.c`)
Add these headers and the detection/latch logic.
c

```
#include "fc/runtime_config.h"
#include "navigation/navigation.h"
#include "io/gps.h"

// Global flag for the OSD system
bool pitotHardwareFailed = false;

static uint16_t pitotFailureCounter = 0;
#define PITOT_FAILURE_THRESHOLD 100 // 1 second at 100Hz

static bool isPitotReportingSane(void)
{
    // Need GPS and a valid wind estimate to perform sanity check
    if (!STATE(GPS_FIX) || !isWindEstimatorValid()) {
        return true;
    }

    float estimatedAirspeedMS = getEstimatedAirspeed();
    float currentPitotMS = pitot.airspeed / 100.0f;

    // Thresholds: Only check if estimated speed > 7m/s
    // Fail if hardware Pitot reports < 40% of estimated speed
    const float minCheckSpeed = 7.0f;
    const float sanityRatio = 0.4f;

    if (estimatedAirspeedMS > minCheckSpeed) {
        if (currentPitotMS < (estimatedAirspeedMS * sanityRatio)) {
            return false;
        }
    }
    return true;
}

// Update the validation function
bool pitotValidForAirspeed(void)
{
    // Check for failure persistence while armed
    if (ARMING_FLAG(ARMED)) {
        if (!isPitotReportingSane() && gpsSol.groundSpeed > 200) {
            pitotFailureCounter++;
        } else if (pitotFailureCounter > 0) {
            pitotFailureCounter--;
        }

        if (pitotFailureCounter >= PITOT_FAILURE_THRESHOLD) {
            pitotHardwareFailed = true;
        }
    } else {
        // Reset flag on disarm for next flight/ground test
        pitotHardwareFailed = false;
        pitotFailureCounter = 0;
    }

    if (pitotHardwareFailed) {
        return STATE(GPS_FIX); // Act as Virtual Pitot
    }

    bool ret = pitotIsHealthy() && pitotIsCalibrationComplete();
    if (detectedSensors[SENSOR_INDEX_PITOT] == PITOT_VIRTUAL) {
        return ret && STATE(GPS_FIX);
    }
    return ret;
}
```

Use code with caution.
Part 2: OSD Warning System (`src/main/io/osd.h` & `osd.c`)
Add the visual alert strings and update the message handler.
In `src/main/io/osd.h`:
c

```
// Add to the OSD_MSG definitions section
#define OSD_MSG_PITOT_FAIL "PITOT FAIL - VIRTUAL"
```

Use code with caution.
In `src/main/io/osd.c`:
c

```
// Locate the system message processing function (osdGetSystemMessage)
void osdGetSystemMessage(char *buff)
{
    extern bool pitotHardwareFailed;

    // Priority check: PITOT FAIL
    if (pitotHardwareFailed) {
        strcpy(buff, OSD_MSG_PITOT_FAIL);
        return;
    }

    // ... [existing message logic: Low Battery, Failsafe, etc.] ...
}
```

Use code with caution.
Part 3: Required CLI Configuration
Ensure your aircraft is configured to handle the virtual fallback accurately. You can find more documentation on CLI settings in the iNav repository.
bash

```
# 1. Calibrate to your aircraft's cruise speed (50km/h example)
set nav_fw_cruise_speed = 1389

# 2. Set min ground speed to prevent stalling in high headwind
set nav_fw_min_gs = 1000

# 3. Enable stall prevention logic
set fw_stall_index_enable = ON

# 4. Save and Reboot
save
```

Use code with caution.
Implementation Summary

1. **Transparency:** The code monitors the hardware pitot in the background using `isPitotReportingSane()`.
2. **Safety Latch:** Once a failure is detected (likely due to a blocked tube), it latches

`pitotHardwareFailed` to `true`.

3. **Seamless Transition:** `pitotValidForAirspeed` then instructs the flight controller that the hardware is gone, forcing it to use the Wind Estimator and GPS data.
4. **Pilot Alert:** The OSD displays a flashing warning so the pilot understands that airspeed is now an estimate.