PX4 Sensor Data Collection and Processing Guide

Overview

This document outlines the complete process for collecting raw sensor data, Kalman filter readings, and quaternions from a PX4-based system running on a Raspberry Pi, and then processing this data into a usable format.

Step 1: Setting Up Log Directory

The first step was to create the necessary log directory structure on the Raspberry Pi:

```bash
# Create the required directory structure
sudo mkdir -p /fs/microsd/log

# Set the appropriate permissions
sudo chmod -R 777 /fs/microsd
```

Step 2: Configuring and Starting the Logger

Next, we configured the PX4 logger to capture the necessary sensor data:

```bash
# Set logging parameters
param set SDLOG_MODE 2       # For logging from boot to shutdown
param set SDLOG_PROFILE 1    # Full logging profile

# Start the logger with specific topics
logger start -a -t sensor_combined -t vehicle_attitude -t vehicle_attitude_groundtruth

# Begin logging
logger on
```

The console displayed confirmation messages:

```
INFO [logger] Start file log (type: full)
INFO [logger] [logger] /fs/microsd/log/2025-04-13/10_10_05.ulg
INFO [logger] Opened full log file: /fs/microsd/log/2025-04-13/10_10_05.ulg
```

Step 3: Transferring Log Files

After collecting data, we transferred the log file from the Raspberry Pi to a local computer:

```bash
# On the local computer
mkdir -p ~/Desktop/px4_logs

# SCP command to copy files
scp -v pi@navio.local:/fs/microsd/log/2025-04-13/10_10_05.ulg ~/Desktop/px4_logs/
```

Step 4: Converting ULog to CSV

We converted the ULog file to CSV format for easier processing:

```bash
# Using pyulog
pip install pyulog
ulog2csv ~/Desktop/px4_logs/10_10_05.ulg -o ~/Desktop/px4_logs/csv_output/

# Alternatively, using PX4 tools
# python3 ~/path-to-px4/Tools/ulog2csv.py ~/Desktop/px4_logs/10_10_05.ulg -o ~/Desktop
```

This created multiple CSV files, one for each message topic:

- 10_10_05_sensor_accel_0.csv
- 10_10_05_sensor_gyro_0.csv
- 10_10_05_vehicle_attitude_0.csv
- 10_10_05_vehicle_magnetometer_0.csv
- etc.

Step 5: Creating a Python Script to Merge CSV Files

We created a Python script to merge the relevant CSV files into a single file matching the desired format:

python

```python
import pandas as pd
import os
import numpy as np
from datetime import datetime

# Path to the directory containing the CSV files
csv_dir = "~/Desktop/px4_logs/csv_output/"
csv_dir = os.path.expanduser(csv_dir)  # Expand the ~ to the full home directory path

# Path for the output CSV
output_file = os.path.join(os.path.dirname(csv_dir), "merged_data.csv")

# List of expected files (these should match the topics you logged)
expected_files = [
    "10_10_05_sensor_accel_0.csv",
    "10_10_05_sensor_gyro_0.csv",
    "10_10_05_vehicle_magnetometer_0.csv",  # or 10_10_05_sensor_mag_0.csv if it exist
    "10_10_05_vehicle_attitude_0.csv",
    "10_10_05_estimator_innovations_0.csv"  # Renamed from ekf2_innovations in newer P
]

# Check which files actually exist
available_files = []
missing_files = []
for file in expected_files:
    file_path = os.path.join(csv_dir, file)
    if os.path.exists(file_path):
        available_files.append(file)
    else:
        missing_files.append(file)

if missing_files:
    print(f"Warning: The following expected files are missing: {', '.join(missing_file

if not available_files:
    print("Error: No expected CSV files found in the directory.")
    print("Available dataframes:", list(dataframes.keys()))
    exit(1)

# Function to load and prepare a dataframe
def load_dataframe(filename):
    file_path = os.path.join(csv_dir, filename)
    print(f"Loading {filename}...")
    df = pd.read_csv(file_path)

    # Convert timestamp to seconds (PX4 logs use microseconds)
    if 'timestamp' in df.columns:
```

```
        df['timestamp'] = df['timestamp'] / 1e6

    return df

# Load all available dataframes
dataframes = {}
for file in available_files:
    base_name = file.split('.')[0]  # Remove .csv
    dataframes[base_name] = load_dataframe(file)

# Create a merged dataframe starting with timestamps
# We'll use the accelerometer's timestamps as the base
if '10_10_05_sensor_accel_0' in dataframes:
    merged_df = pd.DataFrame()
    merged_df['Timestamp'] = dataframes['10_10_05_sensor_accel_0']['timestamp']

    # Add accelerometer data
    if '10_10_05_sensor_accel_0' in dataframes:
        accel_df = dataframes['10_10_05_sensor_accel_0']
        merged_df['Accelerometer_X'] = accel_df['x']
        merged_df['Accelerometer_Y'] = accel_df['y']
        merged_df['Accelerometer_Z'] = accel_df['z']

    # Add gyroscope data
    if '10_10_05_sensor_gyro_0' in dataframes:
        # Resample to match timestamps
        gyro_df = dataframes['10_10_05_sensor_gyro_0']
        # Create temporary dataframes for interpolation
        temp_gyro = pd.DataFrame()
        temp_gyro['timestamp'] = gyro_df['timestamp']
        temp_gyro['x'] = gyro_df['x']
        temp_gyro['y'] = gyro_df['y']
        temp_gyro['z'] = gyro_df['z']
        # Set timestamp as index for interpolation
        temp_gyro.set_index('timestamp', inplace=True)
        # Reindex to match merged_df timestamps with interpolation
        temp_gyro = temp_gyro.reindex(index=merged_df['Timestamp'], method='nearest')
        # Add to merged dataframe
        merged_df['Gyroscope_X'] = temp_gyro['x'].values
        merged_df['Gyroscope_Y'] = temp_gyro['y'].values
        merged_df['Gyroscope_Z'] = temp_gyro['z'].values

    # Find the magnetometer file from the files we've already loaded
    mag_file = None
    for key in dataframes.keys():
        if "magnetometer" in key or "mag" in key:
            mag_file = key
            break

    # If found, add magnetometer data
```

```
if mag_file:
    mag_df = dataframes[mag_file]
    # Create temporary dataframes for interpolation
    temp_mag = pd.DataFrame()
    temp_mag['timestamp'] = mag_df['timestamp']
    # Handle different column naming conventions
    if 'magnetometer_ga[0]' in mag_df.columns:
        temp_mag['x'] = mag_df['magnetometer_ga[0]']
        temp_mag['y'] = mag_df['magnetometer_ga[1]']
        temp_mag['z'] = mag_df['magnetometer_ga[2]']
    elif 'x' in mag_df.columns:
        temp_mag['x'] = mag_df['x']
        temp_mag['y'] = mag_df['y']
        temp_mag['z'] = mag_df['z']
    # Set timestamp as index for interpolation
    temp_mag.set_index('timestamp', inplace=True)
    # Reindex to match merged_df timestamps with interpolation
    temp_mag = temp_mag.reindex(index=merged_df['Timestamp'], method='nearest')
    # Add to merged dataframe
    merged_df['Magnetometer_X'] = temp_mag['x'].values
    merged_df['Magnetometer_Y'] = temp_mag['y'].values
    merged_df['Magnetometer_Z'] = temp_mag['z'].values

# Add attitude data
if '10_10_05_vehicle_attitude_0' in dataframes:
    att_df = dataframes['10_10_05_vehicle_attitude_0']
    # Create temporary dataframes for interpolation
    temp_att = pd.DataFrame()
    temp_att['timestamp'] = att_df['timestamp']
    temp_att['q[0]'] = att_df['q[0]']
    temp_att['q[1]'] = att_df['q[1]']
    temp_att['q[2]'] = att_df['q[2]']
    temp_att['q[3]'] = att_df['q[3]']
    # Set timestamp as index for interpolation
    temp_att.set_index('timestamp', inplace=True)
    # Reindex to match merged_df timestamps with interpolation
    temp_att = temp_att.reindex(index=merged_df['Timestamp'], method='nearest')

    # Add quaternion data
    merged_df['Quat_w'] = temp_att['q[0]'].values
    merged_df['Quat_x'] = temp_att['q[1]'].values
    merged_df['Quat_y'] = temp_att['q[2]'].values
    merged_df['Quat_z'] = temp_att['q[3]'].values

    # Calculate roll, pitch, yaw from quaternions
    # This function converts quaternions to Euler angles
    def quaternion_to_euler(w, x, y, z):
        # Roll (x-axis rotation)
        sinr_cosp = 2 * (w * x + y * z)
```

```
        cosr_cosp = 1 - 2 * (x * x + y * y)
        roll = np.arctan2(sinr_cosp, cosr_cosp)

        # Pitch (y-axis rotation)
        sinp = 2 * (w * y - z * x)
        pitch = np.arcsin(np.clip(sinp, -1.0, 1.0))

        # Yaw (z-axis rotation)
        siny_cosp = 2 * (w * z + x * y)
        cosy_cosp = 1 - 2 * (y * y + z * z)
        yaw = np.arctan2(siny_cosp, cosy_cosp)

        return roll, pitch, yaw

    # Apply the conversion
    roll_list = []
    pitch_list = []
    yaw_list = []

    for i in range(len(merged_df)):
        roll, pitch, yaw = quaternion_to_euler(
            merged_df['Quat_w'].iloc[i],
            merged_df['Quat_x'].iloc[i],
            merged_df['Quat_y'].iloc[i],
            merged_df['Quat_z'].iloc[i]
        )
        roll_list.append(roll)
        pitch_list.append(pitch)
        yaw_list.append(yaw)

    merged_df['Roll'] = roll_list
    merged_df['Pitch'] = pitch_list
    merged_df['Yaw'] = yaw_list

    # Convert to degrees if needed
    # merged_df['Roll'] = np.degrees(merged_df['Roll'])
    # merged_df['Pitch'] = np.degrees(merged_df['Pitch'])
    # merged_df['Yaw'] = np.degrees(merged_df['Yaw'])

# Make sure we have all the columns from data_new.csv
required_columns = [
    'Timestamp',
    'Accelerometer_X', 'Accelerometer_Y', 'Accelerometer_Z',
    'Gyroscope_X', 'Gyroscope_Y', 'Gyroscope_Z',
    'Magnetometer_X', 'Magnetometer_Y', 'Magnetometer_Z',
    'Roll', 'Pitch', 'Yaw',
    'Quat_w', 'Quat_x', 'Quat_y', 'Quat_z'
]

# Fill in any missing columns with NaN
```

```
    # Fill in any missing columns with NaN
    for col in required_columns:
        if col not in merged_df.columns:
            print(f"Warning: Column {col} is missing and will be filled with NaN value
            merged_df[col] = np.nan

    # Reorder columns to match data_new.csv
    merged_df = merged_df[required_columns]

    # Save to CSV
    print(f"Saving merged data to {output_file}")
    merged_df.to_csv(output_file, index=False)
    print(f"Successfully saved merged data with {len(merged_df)} rows")

else:
    print("Error: No accelerometer data found. Cannot create base timestamps.")
    print("Available dataframes:", list(dataframes.keys()))
    exit(1)
```

Step 6: Running the Merge Script

We executed the Python script to merge the CSV files:

```
bash

# Navigate to the directory containing the script
cd ~/Desktop/px4_logs/csv_output/

# Run the script
python3 merge_px4_logs.py
```

Step 7: Verifying the Merged Data

After running the script, we verified the structure of the merged data:

- The merged_data.csv file was created in ~/Desktop/px4_logs/
- The file contains 24 rows of data
- The file includes all 17 required columns:
  - Timestamp
  - Accelerometer_X, Y, Z
  - Gyroscope_X, Y, Z
  - Magnetometer_X, Y, Z
  - Roll, Pitch, Yaw
  - Quat_w, x, y, z

Summary of PX4 Logger Commands

Here's a summary of the key logger commands used:

1. **Configure logging parameters**:

   ```
   param set SDLOG_MODE 2
   param set SDLOG_PROFILE 1
   ```

2. **Start the logger with specific topics**:

   ```
   logger start -a -t sensor_combined -t vehicle_attitude -t
   vehicle_attitude_groundtruth -t sensor_mag -t sensor_accel -t sensor_gyro -t
   ekf2_innovations
   ```

3. **Begin active logging**:

   ```
   logger on
   ```

4. **Stop logging**:

   ```
   logger off
   ```

5. **Completely stop the logger**:

   ```
   logger stop
   ```

Troubleshooting Tips

1. **If the logger is already running**: Stop it first with `logger stop` before starting it with new parameters.
2. **If CSV files have different names**: Check the actual filenames in your output directory and update the script accordingly.
3. **If magnetometer data is missing**: Check which file contains magnetometer data using file listings or by examining header rows.
4. **If data is not aligned properly**: The script uses interpolation to align data from different topics to the accelerometer timestamps.

This completes the process of collecting sensor data from PX4 and processing it into a format matching the required data_new.csv structure.