06 - RouteTrack Pi — Route Processing & Summary Generation

Date: December 25, 2025
Category: Raspberry Pi / GPS / Data Processing

Backlink: RouteTrack Pi — GPS Logging & Data Ingestion

Project Goal

This phase transforms RouteTrack from a **raw GPS logger** into a **route intelligence system**.

Instead of calculating metrics at ingestion time, RouteTrack uses a **post-processing model**:

- Raw GPS points are logged continuously
- Route intelligence is calculated later
- Derived data can be safely regenerated if logic changes

This mirrors how professional telemetry and fleet-tracking systems are built.

High-Level Architecture

Layer	Responsibility
GPS Logger	Writes raw telemetry (`gps_points`)
Route Processor	Computes stops, mileage, summaries
SQLite	Stores raw + derived data

Layer	Responsibility
Dashboard (next phase)	Reads only processed tables

Only **one component writes raw data**.
All intelligence is derived afterward.

Unified Database Schema

All RouteTrack data structures are defined in a **single schema file**:

```
/opt/routetrack/config/schema.sql
```

This file defines:

- Raw GPS telemetry
- Derived stop events
- Aggregated daily summaries

SQLite Schema (with WAL enabled)

```
-- ============================================================
-- RouteTrack SQLite Schema
-- ============================================================

PRAGMA journal_mode=WAL;

-- ============================================================
-- TABLE: gps_points (RAW TELEMETRY)
-- ============================================================
CREATE TABLE IF NOT EXISTS gps_points (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  ts TEXT NOT NULL,
  lat REAL,
  lon REAL,
  speed REAL,
  track REAL,
  alt REAL,
```

```
  mode INTEGER,
  epx REAL,
  epy REAL,
  eps REAL
);

CREATE INDEX IF NOT EXISTS idx_gps_points_ts
  ON gps_points(ts);

-- ============================================================
-- TABLE: stop_events (DERIVED)
-- ============================================================
CREATE TABLE IF NOT EXISTS stop_events (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  start_ts TEXT NOT NULL,
  end_ts TEXT NOT NULL,
  duration_seconds INTEGER NOT NULL,
  lat REAL,
  lon REAL
);

CREATE INDEX IF NOT EXISTS idx_stop_events_start_ts
  ON stop_events(start_ts);

-- ============================================================
-- TABLE: daily_summary (AGGREGATED)
-- ============================================================
CREATE TABLE IF NOT EXISTS daily_summary (
  date TEXT PRIMARY KEY,
  start_ts TEXT,
  end_ts TEXT,
  total_distance_miles REAL,
  moving_time_seconds INTEGER,
  stopped_time_seconds INTEGER,
  stop_count INTEGER
);
```

Schema Design Breakdown

`gps_points` — Source of Truth

- Stores raw TPV messages from `gpsd`
- Append-only
- Never modified or recalculated
- All other tables derive from this data

`stop_events` — Route Intelligence

- Represents **continuous stationary periods**
- Derived using:
  - Speed threshold
  - Minimum dwell time
- Used for:
  - Time-on-site tracking
  - Map stop markers
  - Shift analysis

`daily_summary` — Fast Reporting

- One row per calendar day
- Stores:
  - Start / end timestamps
  - Total mileage
  - Moving vs stopped time
  - Stop count
- Prevents rescanning raw GPS points for dashboards

Applying the Schema (Important!)

SQLite requires an **exclusive lock** for schema changes.

Safe Workflow

```
sudo systemctl stop routetrack-logger.service
sqlite3 /opt/routetrack/data/routetrack.sqlite < /opt/routetrack/config/schema.sql
sqlite3 /opt/routetrack/data/routetrack.sqlite ".tables"
sudo systemctl start routetrack-logger.service
```

Expected tables:

```
gps_points
stop_events
daily_summary
```

Route Processing Script

The route processor converts raw GPS points into usable metrics.

Responsibilities

- Filter invalid fixes (`mode != 3`)
- Ignore GPS drift
- Calculate distance (Haversine)
- Track moving vs stopped time
- Detect stop events
- Populate `stop_events` and `daily_summary`

Route Processor Script (Final Version Used)

File:

```
/opt/routetrack/bin/routetrack-process.py
```

```
#!/usr/bin/env python3
"""
RouteTrack Route Processor (Daily)
"""

import math
import sqlite3
import sys
from datetime import datetime, date, timezone

DB_PATH = "/opt/routetrack/data/routetrack.sqlite"

# 5 mph ≈ 2.235 m/s
MOVEMENT_THRESHOLD_MPS = 2.235
STOP_DWELL_SECONDS = 120
EARTH_RADIUS_KM = 6371.0

def haversine_meters(lat1, lon1, lat2, lon2):
    phi1, phi2 = math.radians(lat1), math.radians(lat2)
    dphi = math.radians(lat2 - lat1)
    dlambda = math.radians(lon2 - lon1)

    a = (
        math.sin(dphi / 2) ** 2 +
        math.cos(phi1) * math.cos(phi2) *
        math.sin(dlambda / 2) ** 2
    )
    return 2 * EARTH_RADIUS_KM * 1000 * math.atan2(
        math.sqrt(a), math.sqrt(1 - a)
    )

def parse_ts(ts):
    return datetime.fromisoformat(ts.replace("Z", "+00:00"))

def main():
    day = sys.argv[1] if len(sys.argv) == 2 else date.today().isoformat()

    start = f"{day}T00:00:00Z"
    end = f"{day}T23:59:59Z"
```

```
conn = sqlite3.connect(DB_PATH)
cur = conn.cursor()

cur.execute("""
    SELECT ts, lat, lon, speed, mode
    FROM gps_points
    WHERE ts >= ? AND ts <= ?
    ORDER BY ts
""", (start, end))

rows = cur.fetchall()
if not rows:
    print("No GPS data for this date.")
    return

cur.execute("DELETE FROM stop_events WHERE start_ts >= ? AND start_ts <= ?", (start, end))
cur.execute("DELETE FROM daily_summary WHERE date = ?", (day,))

total_dist = 0
moving = 0
stopped = 0
stops = []

last = None
stop_start = None

for ts, lat, lon, speed, mode in rows:
    if mode != 3 or lat is None or lon is None:
        continue

    now = parse_ts(ts)

    if last:
        prev_t, prev_lat, prev_lon = last
        dt = (now - prev_t).total_seconds()

        if speed and speed >= MOVEMENT_THRESHOLD_MPS:
            total_dist += haversine_meters(prev_lat, prev_lon, lat, lon)
            moving += int(dt)
```

```
            if stop_start:
                dur = int((now - stop_start[0]).total_seconds())
                if dur >= STOP_DWELL_SECONDS:
                    stops.append((stop_start[0], now, dur, stop_start[1], stop_start[2]))
                    stopped += dur
                stop_start = None
        else:
            if not stop_start:
                stop_start = (now, lat, lon)

    last = (now, lat, lon)

for s in stops:
    cur.execute("""
        INSERT INTO stop_events
        (start_ts, end_ts, duration_seconds, lat, lon)
        VALUES (?, ?, ?, ?, ?)
    """, (s[0].isoformat(), s[1].isoformat(), s[2], s[3], s[4]))

miles = total_dist * 0.000621371

cur.execute("""
    INSERT INTO daily_summary
    (date, start_ts, end_ts, total_distance_miles,
     moving_time_seconds, stopped_time_seconds, stop_count)
    VALUES (?, ?, ?, ?, ?, ?, ?)
""", (day, rows[0][0], rows[-1][0], round(miles, 2), moving, stopped, len(stops)))

conn.commit()
conn.close()

print(f"Processed {day}: miles={round(miles,2)} stops={len(stops)}")

if __name__ == "__main__":
    main()
```

Make executable:

```
sudo chmod +x /opt/routetrack/bin/routetrack-process.py
```

Running the Processor (Safe Method)

Because SQLite needs exclusive access for deletes/inserts:

```
sudo systemctl stop routetrack-logger.service
/opt/routetrack/venv/bin/python /opt/routetrack/bin/routetrack-process.py
sudo systemctl start routetrack-logger.service
```

Verification Queries

Daily Summary

```
sqlite3 /opt/routetrack/data/routetrack.sqlite \
"SELECT * FROM daily_summary ORDER BY date DESC LIMIT 1;"
```

Stop Events

```
sqlite3 /opt/routetrack/data/routetrack.sqlite \
"SELECT start_ts,end_ts,duration_seconds FROM stop_events ORDER BY id DESC LIMIT 5;"
```

Real-World Validation (Stationary Test)

With the GPS unit **not moving at all**:

Metric	Result
Distance	0.0 miles
Moving time	0 seconds

Metric	Result
Stops	1
Stopped time	Entire duration

This confirmed:

- GPS drift was eliminated
- Movement detection is stable
- Stop logic behaves correctly

Why This Matters

This phase turns RouteTrack into a **true telemetry system**:

- Accurate mileage
- Reliable stop detection
- Regenerable summaries
- Dashboard-ready data model

The UI is now just a **viewer**, not a calculator.

Next Steps

The next phase will focus on:

1. Automating route processing (systemd timer)
2. Local Flask API for data access
3. Leaflet map dashboard for:
   - Routes
   - Stops
   - Daily summaries

Revision #1
Created 25 December 2025 14:15:10 by Nate Nash
Updated 25 December 2025 18:15:28 by Nate Nash