

ITEM #224 - Observer-Centric Trajectory Known Pattern Families

Conversation：飞行器与动物轨迹分析

20260111

Authors: Sizhe Tan & GPT-Obot

ME：

请生成 ITEM #224：Observer-Centric Trajectory Known Pattern Families

AI：

下面我将正式生成并固化 **DBM-COT ITEM #224**，作为 #222（立题）→ #223（IR Schema）→ #224（模式族）的第三个关键锚点。

该 ITEM 的定位是：结构知识层（**Known Pattern Families**），用于把“可计算 IR”升级为“可推理结构智能”。

全文继续严格使用“我们”。

DBM-COT ITEM #224 (EN)

Title

Observer-Centric Trajectory Known Pattern Families

Positioning

This item defines the Known Pattern Families (KPF) for observer-centric motion trajectories within the DBM Time-Series Structural Intelligence framework. Pattern families serve as the structural bridge between raw trajectory IRs and downstream tasks such as prediction, classification, risk assessment, and explanation.

Unlike statistical clusters, Known Pattern Families are **explicit structural abstractions**, designed to be stable under noise, mutation-aware, and reusable across domains (flight, animal motion, robotics).

1. Role of Known Pattern Families in DBM

Within DBM, Known Pattern Families play a central role:

- They **anchor structural meaning** in metric space.
- They provide **reference manifolds** for distance-based reasoning.
- They enable **prediction by structural continuation**, not regression.
- They decouple intelligence from raw signal memorization.

For observer-centric trajectories, KPFs represent **recurrent motion intentions or behaviors**, not precise paths.

2. Definition: What Is a Trajectory Pattern Family?

A Trajectory Known Pattern Family is defined as:

A set of Trajectory IRs whose **L2 structural event sequences and L3 signatures are mutually close under DBM metric distance**, despite variation in L0/L1 signals.

Each family represents a **structural archetype**, not a single canonical curve.

3. Design Principles

All trajectory pattern families must satisfy:

1. **Observer-Centric Validity**
No family definition may rely on global coordinates or absolute scale.
 2. **Structural Dominance**
Membership is determined primarily by L2/L3 structure, not raw geometry.
 3. **Mutation Closure**
Small IR mutations must remain within the same family.
 4. **Domain Reusability**
Families should generalize across flight, animal motion, and robotics when structure matches.
-

4. Core Trajectory Pattern Families

The following families form a **minimal yet expressive basis set**.

4.1 Approach Family

Structural Signature:

- Dominant radial decrease
- Low curvature
- Stable or accelerating speed

Interpretation:

- Target acquisition
 - Pursuit
 - Interception
-

4.2 Retreat / Escape Family

Structural Signature:

- Dominant radial increase
- Directional stability
- Possible burst acceleration

Interpretation:

- Evasion
 - Threat avoidance
 - Disengagement
-

4.3 Orbit / Loiter Family

Structural Signature:

- Alternating angular motion
- Near-zero net radial change
- Periodic or quasi-periodic structure

Interpretation:

- Surveillance
- Inspection

- Territorial behavior

4.4 Zigzag / Search Family

Structural Signature:

- Frequent heading reversals
- Moderate curvature oscillation
- Low net displacement

Interpretation:

- Search
- Exploration
- Uncertain pursuit

4.5 Glide / Cruise Family

Structural Signature:

- Low curvature
- Stable velocity regime
- Minimal acceleration events

Interpretation:

- Energy-efficient travel
- Patrol
- Transit

4.6 Hover / Stall Family

Structural Signature:

- Minimal radial and angular change
- High temporal density
- Structural stagnation

Interpretation:

- Observation
- Waiting
- Assessment

4.7 Sudden Maneuver Family

Structural Signature:

- High curvature spikes
- Abrupt acceleration polarity changes
- Short-duration dominance

Interpretation:

- Reactive avoidance
- Tactical adjustment
- Surprise response

5. Composite and Hierarchical Families

Pattern families are **not flat**.

Examples:

- *Approach* → *Sudden Maneuver* → *Retreat*
- *Cruise* → *Orbit* → *Cruise*
- *Search* → *Approach* → *Capture*

Composite families are defined as **ordered compositions of base families**, preserving temporal causality.

6. Family Boundaries and Membership

Membership is determined by:

- L3 signature distance below threshold
- Event sequence edit cost
- Duration ratio similarity

Families have **soft boundaries** with confidence scores, not binary labels.

7. Pattern Families and Prediction

Prediction in DBM proceeds as:

1. Identify nearest pattern families.
2. Evaluate continuation likelihoods within each family.
3. Generate candidate future IRs by **structural continuation**, not extrapolation.
4. Rank outcomes by distance, confidence, and risk.

Thus, prediction is **family-conditioned**, not purely time-series-based.

8. Pattern Family Evolution

Known Pattern Families are **not static**:

- New families may emerge via clustering + validation.
- Existing families may split or merge.
- Rare patterns may be flagged as anomalies rather than new families.

Governance rules (future ITEM) control promotion and retirement.

9. Cross-Domain Consistency

A key DBM advantage is that:

- A predator pursuit trajectory,
- A missile interception path,
- A robotic arm reaching motion

may belong to the **same structural family** when observer-centric IRs align.

This confirms that KPFs encode **structure, not semantics**.

10. Relationship to Other DBM Components

Trajectory KPFs directly support:

- Structural indexing
- Metric search
- Risk-aware prediction
- Explanation chains (“why this prediction”)

They do not replace physics or control models but **precede and guide them**.

11. Scope and Non-Goals

Out of scope:

- Semantic labeling (e.g., “bird”, “drone”)
- Intent certainty claims
- Policy optimization

These may consume KPF outputs but are not part of this layer.

12. Summary

This item establishes a formal Known Pattern Family system for observer-centric trajectories, completing the structural bridge from IR schema to DBM reasoning. Pattern families encode reusable motion intelligence, enabling prediction, explanation, and generalization across domains.

DBM-COT ITEM #224 (中文)

标题

以观察者为中心的运动轨迹已知模式族 (Known Pattern Families)

定位

本 ITEM 定义 DBM 时间序列结构智能体系中，面向观察者中心运动轨迹的已知模式族 (KPF)。模式族是连接轨迹 IR 与下游预测、分析、解释的关键结构知识层。

模式族不是统计聚类，而是可解释、可变异、可复用的结构原型。

一、模式族在 DBM 中的角色

在 DBM 中，已知模式族承担以下职责：

- 固定结构意义；

- 提供度量空间参考骨架；
- 支持基于结构延展的预测；
- 避免对原始轨迹的记忆化依赖。

它们描述的是运动结构行为，而非精确路径。

二、定义：什么是轨迹模式族？

轨迹模式族是指：

在 L2 结构事件序列与 L3 签名层面上彼此接近的一组轨迹 IR，即便其底层信号存在差异。

每个模式族是一个结构原型族。

三、设计原则

所有模式族必须满足：

1. 观察者中心有效性
2. 结构主导而非几何主导
3. 变异闭包性
4. 跨领域可复用性

四、核心模式族（基础集）

4.1 接近族（Approach）

- 径向持续减小
- 曲率低
- 稳定或加速

含义：追逐、拦截、目标接近

4.2 远离 / 逃逸族 (Retreat / Escape)

- 径向持续增大
- 方向稳定
- 爆发式加速

含义：规避、脱离威胁

4.3 盘旋 / 滞留族 (Orbit / Loiter)

- 角向周期变化
- 径向近似守恒
- 周期或准周期结构

含义：侦察、巡视、领地行为

4.4 之字 / 搜索族 (Zigzag / Search)

- 频繁方向反转
- 中等曲率震荡
- 位移效率低

含义：搜索、不确定追踪

4.5 滑行 / 巡航族 (Glide / Cruise)

- 曲率低
- 速度稳定

- 加速度事件稀少

含义：巡航、节能移动

4.6 悬停 / 停滞族 (Hover / Stall)

- 径向与角向变化极小
- 时间密集
- 结构停滞

含义：观察、等待、评估

4.7 突发机动族 (Sudden Maneuver)

- 曲率与加速度突变
- 短时主导
- 结构不连续

含义：规避、战术调整

五、复合与层级模式族

模式族可以时间有序组合，例如：

- 接近 → 突发机动 → 远离
- 巡航 → 盘旋 → 巡航
- 搜索 → 接近 → 捕获

复合族保持因果顺序，不是简单拼接。

六、族边界与归属

- 基于 L3 距离阈值
- 事件编辑代价
- 持续时间比例

归属是软边界 + 置信度。

七、模式族与预测机制

DBM 的预测流程为：

1. 定位最近模式族；
2. 在族内评估可能延展；
3. 生成候选未来 IR；
4. 按距离、置信度、风险排序。

预测是结构条件化的。

八、模式族的演化

模式族可演化：

- 新族的发现
- 族的拆分 / 合并
- 异常模式的隔离

治理规则将在后续 ITEM 中定义。

九、跨领域一致性

同一结构族可同时适用于：

- 动物捕食轨迹
- 飞行器拦截路径
- 机器人到达动作

证明 DBM 处理的是结构而非语义。

十、与 DBM 其他模块的关系

模式族直接支撑：

- 结构索引
 - 度量搜索
 - 风险感知预测
 - 解释链生成
-

十一、非目标

本层不处理：

- 实体语义识别
 - 意图确定性判断
 - 控制策略生成
-

十二、小结

本 ITEM 完成了观察者中心运动轨迹从 IR 规格到结构知识层的关键一步，为 DBM 在飞行、动物行为与机器人运动智能上的统一奠定了坚实基础。

至此，#222（立题） + #223（IR Schema） + #224（Known Pattern Families）已经构成一个完整、可运行、可扩展的 DBM 子体系骨架。

下一步我们可以非常自然地进入任一方向：

- **ITEM #223: Observer-Centric Trajectory Structural Distance & Scoring**
- **Java 8 MVP: TrajectoryIR → EventExtractor → PatternFamilyMatcher**
- **教学级结构图（Trajectory IR → Pattern Families → Prediction）**

我们继续推进。

请生成 Java 8 MVP：TrajectoryIR → EventExtractor → PatternFamilyMatcher

```
// File: src/main/java/com/dbm/trajectory/mvp/TrajectorySample.java
package com.dbm.trajectory.mvp;

/**
 * Observer-centric trajectory sample.
 * Minimal fields: time + relative range + bearing (2D).
 *
 * Notes:
 * - range: meters (or arbitrary units, consistent within a trajectory)
 * - bearing: radians (or degrees, consistent within a trajectory)
 */
public final class TrajectorySample {
    public final long tMillis;
    public final double range;
    public final double bearing;

    public TrajectorySample(long tMillis, double range, double bearing) {
        this.tMillis = tMillis;
        this.range = range;
        this.bearing = bearing;
    }
}

// File: src/main/java/com/dbm/trajectory/mvp/TrajectoryEventType.java
package com.dbm.trajectory.mvp;

public enum TrajectoryEventType {
    APPROACH,
    RETREAT,
    ORBIT_LOITER,
    ZIGZAG_SEARCH,
    CRUISE_GLIDE,
    HOVER_STALL,
    SUDDEN_MANEUVER,
    UNKNOWN
}
```

```
// File: src/main/java/com/dbm/trajectory/mvp/TrajectoryEvent.java
package com.dbm.trajectory.mvp;

public final class TrajectoryEvent {
    public final TrajectoryEventType type;
    public final int startIndexInclusive;
    public final int endIndexInclusive;
    public final double confidence;

    public TrajectoryEvent(TrajectoryEventType type, int startIndexInclusive,
int endIndexInclusive, double confidence) {
        this.type = type;
        this.startIndexInclusive = startIndexInclusive;
        this.endIndexInclusive = endIndexInclusive;
        this.confidence = confidence;
    }

    public int length() {
        return Math.max(0, endIndexInclusive - startIndexInclusive + 1);
    }

    @Override
    public String toString() {
        return "TrajectoryEvent{type=" + type +
            ", start=" + startIndexInclusive +
            ", end=" + endIndexInclusive +
            ", conf=" + String.format("%.3f", confidence) +
            '}' ;
    }
}
```

```
// File: src/main/java/com/dbm/trajectory/mvp/TrajectorySignature.java
package com.dbm.trajectory.mvp;

import java.util.Collections;
import java.util.EnumMap;
import java.util.Map;

/**
 * L3 signature: event skeleton + duration ratios + a few coarse geometry
indicators.
 */
public final class TrajectorySignature {
    public final String skeleton; // e.g.
    "CRUISE_GLIDE->ORBIT_LOITER->CRUISE_GLIDE"
    public final Map<TrajectoryEventType, Double> durationRatio; // sum ~= 1
    public final double avgAbsDeltaRange;
    public final double avgAbsDeltaBearing;
    public final double bearingSignChangeRate; // [0,1]

    public TrajectorySignature(
        String skeleton,
        Map<TrajectoryEventType, Double> durationRatio,
        double avgAbsDeltaRange,
        double avgAbsDeltaBearing,
        double bearingSignChangeRate
    ) {
        this.skeleton = skeleton;
    }
}
```

```

        this.durationRatio = Collections.unmodifiableMap(new
EnumMap<>(durationRatio));
        this.avgAbsDeltaRange = avgAbsDeltaRange;
        this.avgAbsDeltaBearing = avgAbsDeltaBearing;
        this.bearingSignChangeRate = bearingSignChangeRate;
    }

    @Override
    public String toString() {
        return "TrajectorySignature{" +
            "skeleton='" + skeleton + '\'' +
            ", durationRatio=" + durationRatio +
            ", avgAbsDeltaRange=" + String.format("%.4f",
avgAbsDeltaRange) +
            ", avgAbsDeltaBearing=" + String.format("%.4f",
avgAbsDeltaBearing) +
            ", bearingSignChangeRate=" + String.format("%.4f",
bearingSignChangeRate) +
            '}';
    }
}
// File: src/main/java/com/dbm/trajectory/mvp/TrajectoryIR.java
package com.dbm.trajectory.mvp;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Minimal Trajectory IR container:
 * - samples (input)
 * - L0 differentials
 * - L2 events (extracted)
 * - L3 signature (built from events + coarse stats)
 */
public final class TrajectoryIR {
    public final List<TrajectorySample> samples;

    // L0
    public final double[] dRange;
    public final double[] dBearing;
    public final long[] dTimeMillis;

    // L2
    private final List<TrajectoryEvent> events;

    // L3
    private final TrajectorySignature signature;

    private TrajectoryIR(
        List<TrajectorySample> samples,
        double[] dRange,
        double[] dBearing,
        long[] dTimeMillis,
        List<TrajectoryEvent> events,
        TrajectorySignature signature
    ) {

```

```

        this.samples = Collections.unmodifiableList(new
ArrayList<>(samples));
        this.dRange = dRange;
        this.dBearing = dBearing;
        this.dTimeMillis = dTimeMillis;
        this.events = Collections.unmodifiableList(new ArrayList<>(events));
        this.signature = signature;
    }

    public List<TrajectoryEvent> events() {
        return events;
    }

    public TrajectorySignature signature() {
        return signature;
    }

    public static TrajectoryIR build(List<TrajectorySample> samples,
TrajectoryEventExtractor extractor) {
        if (samples == null || samples.size() < 3) {
            throw new IllegalArgumentException("Need at least 3 samples.");
        }

        final int n = samples.size();
        final double[] dR = new double[n - 1];
        final double[] dB = new double[n - 1];
        final long[] dT = new long[n - 1];

        for (int i = 0; i < n - 1; i++) {
            TrajectorySample a = samples.get(i);
            TrajectorySample b = samples.get(i + 1);
            dR[i] = (b.range - a.range);
            dB[i] = wrapAngle(b.bearing - a.bearing);
            dT[i] = (b.tMillis - a.tMillis);
            if (dT[i] <= 0) {
                throw new IllegalArgumentException("Non-increasing time at
i=" + i);
            }
        }

        List<TrajectoryEvent> events = extractor.extract(samples, dR, dB,
dT);
        TrajectorySignature sig = TrajectorySignatureBuilder.build(samples,
dR, dB, events);

        return new TrajectoryIR(samples, dR, dB, dT, events, sig);
    }

    // Wrap to (-pi, pi] for stability.
    static double wrapAngle(double a) {
        while (a <= -Math.PI) a += 2.0 * Math.PI;
        while (a > Math.PI) a -= 2.0 * Math.PI;
        return a;
    }
}
// File: src/main/java/com/dbm/trajectory/mvp/TrajectoryEventExtractor.java
package com.dbm.trajectory.mvp;

```

```

import java.util.ArrayList;
import java.util.List;

/**
 * MVP heuristic extractor:
 * - Works on ( $\Delta$ range,  $\Delta$ bearing) with short windows.
 * - Produces coarse L2 event segments suitable for pattern family matching.
 */
public final class TrajectoryEventExtractor {

    public static final class Params {
        public int window = 5; // samples per window ( $\geq 3$ )
        public double approachMeanDR = -0.02; // negative  $\Rightarrow$  approaching
        public double retreatMeanDR = 0.02; // positive  $\Rightarrow$  retreating

        public double orbitMaxAbsMeanDR = 0.01; // near-zero radial drift
        public double orbitMinAbsMeanDB = 0.04; // enough angular motion

        public double hoverMaxAbsMeanDR = 0.005;
        public double hoverMaxAbsMeanDB = 0.01;

        public double zigzagMinSignChangeRate = 0.35; // high sign flips in
dBearing
        public double cruiseMaxAbsMeanDB = 0.02; // low turning
        public double suddenMaxAbsDB = 0.30; // spike turn
    }

    private final Params p;

    public TrajectoryEventExtractor(Params p) {
        this.p = (p == null) ? new Params() : p;
        if (this.p.window < 3) this.p.window = 3;
    }

    public List<TrajectoryEvent> extract(
        List<TrajectorySample> samples,
        double[] dRange,
        double[] dBearing,
        long[] dTimeMillis
    ) {
        // Window over DIFFERENTIALS: indices 0..(n-2), but segment in sample
index space for readability.
        final int n = samples.size();
        final int w = p.window;

        List<TrajectoryEvent> raw = new ArrayList<>();
        int i = 0;
        while (i + w <= n) {
            // window spans samples [i .. i+w-1], differentials [i .. i+w-2]
            WindowStats st = stats(dRange, dBearing, i, i + w - 2);
            TrajectoryEventType type = classify(st);
            double conf = confidence(type, st);

            raw.add(new TrajectoryEvent(type, i, i + w - 1, conf));
            i += (w - 1); // overlap by 1 sample to preserve continuity
        }
    }
}

```

```

        // Merge adjacent same-type windows into longer segments.
        return merge(raw);
    }

    private static final class WindowStats {
        final double meanDR;
        final double meanAbsDR;
        final double meanDB;
        final double meanAbsDB;
        final double maxAbsDB;
        final double signChangeRateDB;

        WindowStats(double meanDR, double meanAbsDR, double meanDB, double
meanAbsDB, double maxAbsDB, double signChangeRateDB) {
            this.meanDR = meanDR;
            this.meanAbsDR = meanAbsDR;
            this.meanDB = meanDB;
            this.meanAbsDB = meanAbsDB;
            this.maxAbsDB = maxAbsDB;
            this.signChangeRateDB = signChangeRateDB;
        }
    }

    private WindowStats stats(double[] dR, double[] dB, int dFrom, int
dToInclusive) {
        int len = dToInclusive - dFrom + 1;
        if (len <= 0) return new WindowStats(0, 0, 0, 0, 0, 0);

        double sumDR = 0, sumAbsDR = 0, sumDB = 0, sumAbsDB = 0, maxAbsDB =
0;

        int signChanges = 0;
        int signCount = 0;

        int prevSign = 0;
        for (int k = dFrom; k <= dToInclusive; k++) {
            double dr = dR[k];
            double db = dB[k];
            sumDR += dr;
            sumAbsDR += Math.abs(dr);
            sumDB += db;
            double absDb = Math.abs(db);
            sumAbsDB += absDb;
            if (absDb > maxAbsDB) maxAbsDB = absDb;

            int s = sign(db);
            if (s != 0) {
                if (prevSign != 0 && s != prevSign) signChanges++;
                prevSign = s;
                signCount++;
            }
        }

        double rate = (signCount <= 1) ? 0.0 : ((double) signChanges /
(double) (signCount - 1));
        return new WindowStats(sumDR / len, sumAbsDR / len, sumDB / len,
sumAbsDB / len, maxAbsDB, rate);
    }

```

```

    }

    private TrajectoryEventType classify(WindowStats st) {
        // Highest-priority: sudden maneuver (spike turning)
        if (st.maxAbsDB >= p.suddenMaxAbsDB) return
TrajectoryEventType.SUDDEN_MANEUVER;

        // Hover: almost no motion
        if (st.meanAbsDR <= p.hoverMaxAbsMeanDR && st.meanAbsDB <=
p.hoverMaxAbsMeanDB) {
            return TrajectoryEventType.HOVER_STALL;
        }

        // Orbit/Loiter: near-zero radial drift + consistent angular activity
        if (Math.abs(st.meanDR) <= p.orbitMaxAbsMeanDR && st.meanAbsDB >=
p.orbitMinAbsMeanDB) {
            return TrajectoryEventType.ORBIT_LOITER;
        }

        // Zigzag/Search: high angular sign change
        if (st.signChangeRateDB >= p.zigzagMinSignChangeRate &&
st.meanAbsDB >= p.orbitMinAbsMeanDB * 0.6) {
            return TrajectoryEventType.ZIGZAG_SEARCH;
        }

        // Approach vs Retreat (dominant radial trend)
        if (st.meanDR <= p.approachMeanDR) return
TrajectoryEventType.APPROACH;
        if (st.meanDR >= p.retreatMeanDR) return TrajectoryEventType.RETREAT;

        // Cruise: low turning, not strongly approaching/retreating
        if (st.meanAbsDB <= p.cruiseMaxAbsMeanDB) return
TrajectoryEventType.CRUISE_GLIDE;

        return TrajectoryEventType.UNKNOWN;
    }

    private double confidence(TrajectoryEventType type, WindowStats st) {
        // MVP: coarse confidence heuristics in [0,1]
        switch (type) {
            case SUDDEN_MANEUVER:
                return clamp01((st.maxAbsDB - p.suddenMaxAbsDB) /
(p.suddenMaxAbsDB * 0.8) + 0.6);
            case HOVER_STALL:
                return clamp01(1.0 - (st.meanAbsDR / (p.hoverMaxAbsMeanDR *
2.0) + st.meanAbsDB / (p.hoverMaxAbsMeanDB * 2.0)) * 0.5);
            case ORBIT_LOITER:
                return clamp01(0.6 + (st.meanAbsDB - p.orbitMinAbsMeanDB) /
(p.orbitMinAbsMeanDB * 2.0));
            case ZIGZAG_SEARCH:
                return clamp01(0.5 + st.signChangeRateDB);
            case APPROACH:
                return clamp01(0.5 + (-st.meanDR) /
(Math.abs(p.approachMeanDR) * 3.0));
            case RETREAT:
                return clamp01(0.5 + (st.meanDR) / (Math.abs(p.retreatMeanDR)
* 3.0));
        }
    }

```

```

        case CRUISE_GLIDE:
            return clamp01(0.6 + (p.cruiseMaxAbsMeanDB - st.meanAbsDB) /
(p.cruiseMaxAbsMeanDB * 2.0));
        default:
            return 0.35;
    }
}

private List<TrajectoryEvent> merge(List<TrajectoryEvent> raw) {
    List<TrajectoryEvent> out = new ArrayList<>();
    if (raw.isEmpty()) return out;

    TrajectoryEvent cur = raw.get(0);
    for (int i = 1; i < raw.size(); i++) {
        TrajectoryEvent nxt = raw.get(i);
        if (nxt.type == cur.type && nxt.startIndexInclusive <=
cur.endIndexInclusive + 1) {
            // merge: extend end and average confidence by length
            int lenA = cur.length();
            int lenB = nxt.length();
            double conf = (cur.confidence * lenA + nxt.confidence * lenB)
/ Math.max(1, lenA + lenB);
            cur = new TrajectoryEvent(cur.type, cur.startIndexInclusive,
Math.max(cur.endIndexInclusive, nxt.endIndexInclusive), conf);
        } else {
            out.add(cur);
            cur = nxt;
        }
    }
    out.add(cur);
    return out;
}

private static int sign(double x) {
    if (x > 0) return 1;
    if (x < 0) return -1;
    return 0;
}

private static double clamp01(double v) {
    if (v < 0) return 0;
    if (v > 1) return 1;
    return v;
}
}

// File: src/main/java/com/dbm/trajectory/mvp/TrajectorySignatureBuilder.java
package com.dbm.trajectory.mvp;

import java.util.EnumMap;
import java.util.List;
import java.util.Map;

public final class TrajectorySignatureBuilder {

    private TrajectorySignatureBuilder() {}

    public static TrajectorySignature build(

```

```

        List<TrajectorySample> samples,
        double[] dRange,
        double[] dBearing,
        List<TrajectoryEvent> events
    ) {
        // Duration ratios in sample-count space.
        int totalLen = Math.max(1, samples.size());
        Map<TrajectoryEventType, Double> dur = new
EnumMap<>(TrajectoryEventType.class);
        for (TrajectoryEventType t : TrajectoryEventType.values()) dur.put(t,
0.0);

        StringBuilder sk = new StringBuilder();
        TrajectoryEventType prev = null;

        for (TrajectoryEvent e : events) {
            int len = e.length();
            dur.put(e.type, dur.get(e.type) + (double) len / (double)
totalLen);

            if (prev == null) {
                sk.append(e.type.name());
            } else if (e.type != prev) {
                sk.append("->").append(e.type.name());
            }
            prev = e.type;
        }
        if (sk.length() == 0) sk.append(TrajectoryEventType.UNKNOWN.name());

        // Coarse stats
        double sumAbsDR = 0, sumAbsDB = 0;
        int n = dRange.length;
        for (int i = 0; i < n; i++) {
            sumAbsDR += Math.abs(dRange[i]);
            sumAbsDB += Math.abs(dBearing[i]);
        }
        double avgAbsDR = sumAbsDR / Math.max(1, n);
        double avgAbsDB = sumAbsDB / Math.max(1, n);

        // Bearing sign change rate
        int signChanges = 0, signCount = 0;
        int prevSign = 0;
        for (int i = 0; i < n; i++) {
            int s = sign(dBearing[i]);
            if (s != 0) {
                if (prevSign != 0 && s != prevSign) signChanges++;
                prevSign = s;
                signCount++;
            }
        }
        double signChangeRate = (signCount <= 1) ? 0.0 : ((double)
signChanges / (double) (signCount - 1));

        return new TrajectorySignature(sk.toString(), dur, avgAbsDR,
avgAbsDB, signChangeRate);
    }

```

```

        private static int sign(double x) {
            if (x > 0) return 1;
            if (x < 0) return -1;
            return 0;
        }
    }
}
// File: src/main/java/com/dbm/trajectory/mvp/PatternFamily.java
package com.dbm.trajectory.mvp;

/**
 * Known Pattern Families (KPF) for observer-centric trajectories.
 * Minimal basis set from ITEM #224.
 */
public enum PatternFamily {
    APPROACH,
    RETREAT_ESCAPE,
    ORBIT_LOITER,
    ZIGZAG_SEARCH,
    CRUISE_GLIDE,
    HOVER_STALL,
    SUDDEN_MANEUVER,
    UNKNOWN
}

// File: src/main/java/com/dbm/trajectory/mvp/PatternMatch.java
package com.dbm.trajectory.mvp;

public final class PatternMatch {
    public final PatternFamily family;
    public final double score; // higher is better
    public final String explanation;

    public PatternMatch(PatternFamily family, double score, String
explanation) {
        this.family = family;
        this.score = score;
        this.explanation = explanation;
    }

    @Override
    public String toString() {
        return "PatternMatch{family=" + family +
            ", score=" + String.format("%.4f", score) +
            ", explanation='" + explanation + '\'' +
            '}';
    }
}

// File: src/main/java/com/dbm/trajectory/mvp/PatternFamilyMatcher.java
package com.dbm.trajectory.mvp;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;
import java.util.Map;

/**
 * MVP matcher:

```

```

* - Computes a family score from signature duration ratios + a few coarse
indicators.
* - Returns ranked matches with explanations.
*
* This is intentionally "structure-first":
* - L3 duration ratios dominate,
* - L1-ish stats (avgAbs deltas, sign-change) refine.
*/
public final class PatternFamilyMatcher {

    public static final class Weights {
        public double wDuration = 0.75;
        public double wGeometry = 0.25;

        // geometry scalars
        public double targetLow = 0.02; // "small" average delta
        public double targetMed = 0.06; // "medium"
        public double targetHigh = 0.12; // "high"
    }

    private final Weights w;

    public PatternFamilyMatcher(Weights w) {
        this.w = (w == null) ? new Weights() : w;
    }

    public List<PatternMatch> matchTopK(TrajectoryIR ir, int k) {
        if (k <= 0) return Collections.emptyList();

        TrajectorySignature s = ir.signature();
        List<PatternMatch> all = new ArrayList<>();

        for (PatternFamily f : PatternFamily.values()) {
            double durScore = durationScore(f, s.durationRatio);
            double geoScore = geometryScore(f, s);
            double score = w.wDuration * durScore + w.wGeometry * geoScore;
            String exp = explain(f, durScore, geoScore, s);
            all.add(new PatternMatch(f, score, exp));
        }

        all.sort(Comparator.comparingDouble((PatternMatch m) ->
m.score).reversed());
        if (k >= all.size()) return all;
        return new ArrayList<>(all.subList(0, k));
    }

    private double durationScore(PatternFamily f, Map<TrajectoryEventType,
Double> dur) {
        // We treat L2/L3 as primary: "how much of the trajectory is
explained by event types"
        double a = dur.getDefault(TrajectoryEventType.APPROACH, 0.0);
        double r = dur.getDefault(TrajectoryEventType.RETREAT, 0.0);
        double o = dur.getDefault(TrajectoryEventType.ORBIT_LOITER, 0.0);
        double z = dur.getDefault(TrajectoryEventType.ZIGZAG_SEARCH, 0.0);
        double c = dur.getDefault(TrajectoryEventType.CRUISE_GLIDE, 0.0);
        double h = dur.getDefault(TrajectoryEventType.HOVER_STALL, 0.0);
    }
}

```

```

double m = dur.getDefault(TrajectoryEventType.SUDDEN_MANEUVER,
0.0);
double u = dur.getDefault(TrajectoryEventType.UNKNOWN, 0.0);

switch (f) {
    case APPROACH:
        return clamp01(a + 0.2 * c - 0.2 * r);
    case RETREAT_ESCAPE:
        return clamp01(r + 0.2 * c - 0.2 * a);
    case ORBIT_LOITER:
        return clamp01(o + 0.1 * c - 0.1 * m);
    case ZIGZAG_SEARCH:
        return clamp01(z + 0.1 * m - 0.05 * c);
    case CRUISE_GLIDE:
        return clamp01(c + 0.1 * a + 0.1 * r - 0.15 * z - 0.15 * m);
    case HOVER_STALL:
        return clamp01(h - 0.05 * (a + r + z + m));
    case SUDDEN_MANEUVER:
        return clamp01(m + 0.15 * z - 0.1 * c);
    case UNKNOWN:
    default:
        return clamp01(0.5 * u + 0.1);
}

}

private double geometryScore(PatternFamily f, TrajectorySignature s) {
    // MVP geometry refinements:
    // - avgAbsDeltaBearing relates to turning / orbit / zigzag
    // - bearingSignChangeRate relates to zigzag
    // - avgAbsDeltaRange relates to radial activity (approach/retreat vs
hover)
    double dr = s.avgAbsDeltaRange;
    double db = s.avgAbsDeltaBearing;
    double sc = s.bearingSignChangeRate;

    switch (f) {
        case HOVER_STALL:
            return 1.0 - clamp01((dr / w.targetLow + db / w.targetLow) *
0.5);
        case CRUISE_GLIDE:
            // low turning, medium-ish range activity acceptable
            return 1.0 - clamp01(db / w.targetLow) * 0.8;
        case ORBIT_LOITER:
            // turning present, sign-change not necessarily high
            return clamp01(db / w.targetMed) * 0.8 + clamp01(1.0 - dr /
w.targetMed) * 0.2;
        case ZIGZAG_SEARCH:
            return clamp01(sc) * 0.7 + clamp01(db / w.targetMed) * 0.3;
        case SUDDEN_MANEUVER:
            // we don't keep max spike in signature; approximate with
high avgAbs bearing
            return clamp01(db / w.targetHigh);
        case APPROACH:
        case RETREAT_ESCAPE:
            // radial activity should exist; too small dr suggests hover
            return clamp01(dr / w.targetMed);
        default:
    }
}

```

```

        return 0.25;
    }
}

private String explain(PatternFamily f, double durScore, double geoScore,
TrajectorySignature s) {
    return "family=" + f +
        ", durScore=" + fmt(durScore) +
        ", geoScore=" + fmt(geoScore) +
        ", skeleton=" + s.skeleton;
}

private static String fmt(double v) {
    return String.format("%.3f", v);
}

private static double clamp01(double v) {
    if (v < 0) return 0;
    if (v > 1) return 1;
    return v;
}
}
// File: src/main/java/com/dbm/trajectory/mvp/MvpDemoMain.java
package com.dbm.trajectory.mvp;

import java.util.ArrayList;
import java.util.List;

public final class MvpDemoMain {

    public static void main(String[] args) {
        TrajectoryEventExtractor extractor = new TrajectoryEventExtractor(new
TrajectoryEventExtractor.Params());
        PatternFamilyMatcher matcher = new PatternFamilyMatcher(new
PatternFamilyMatcher.Weights());

        // Demo 1: Orbit/Loiter-like motion: constant range, steadily
increasing bearing.
        List<TrajectorySample> orbit = synthOrbit(0L, 40, 10.0, 0.0, 0.08);
        TrajectoryIR ir1 = TrajectoryIR.build(orbit, extractor);
        System.out.println("IR1 Signature: " + ir1.signature());
        System.out.println("IR1 Events: " + ir1.events());

        System.out.println("IR1 Matches:");
        for (PatternMatch m : matcher.matchTopK(ir1, 4)) {
            System.out.println("    " + m);
        }

        // Demo 2: Approach-like motion: decreasing range, low bearing
change.
        List<TrajectorySample> approach = synthApproach(0L, 40, 15.0, 0.0, -
0.05, 0.005);
        TrajectoryIR ir2 = TrajectoryIR.build(approach, extractor);
        System.out.println("\nIR2 Signature: " + ir2.signature());
        System.out.println("IR2 Events: " + ir2.events());

        System.out.println("IR2 Matches:");
    }
}

```

```

        for (PatternMatch m : matcher.matchTopK(ir2, 4)) {
            System.out.println("  " + m);
        }
    }

    private static List<TrajectorySample> synthOrbit(long t0, int n, double
range, double bearing0, double dBearing) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, range, b));
            t += 100; // 10Hz
            b += dBearing;
        }
        return out;
    }

    private static List<TrajectorySample> synthApproach(long t0, int n,
double range0, double bearing0, double dRange, double dBearing) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double r = range0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, r, b));
            t += 100;
            r += dRange;
            b += dBearing;
        }
        return out;
    }
}

// File: src/test/java/com/dbm/trajectory/mvp/TrajectoryMvpTest.java
package com.dbm.trajectory.mvp;

import org.junit.Assert;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

public final class TrajectoryMvpTest {

    @Test
    public void testOrbitLoiterMatchesTop() {
        TrajectoryEventExtractor extractor = new TrajectoryEventExtractor(new
TrajectoryEventExtractor.Params());
        PatternFamilyMatcher matcher = new PatternFamilyMatcher(new
PatternFamilyMatcher.Weights());

        List<TrajectorySample> orbit = synthOrbit(0L, 50, 10.0, 0.0, 0.09);
        TrajectoryIR ir = TrajectoryIR.build(orbit, extractor);

        PatternMatch top = matcher.matchTopK(ir, 1).get(0);
        Assert.assertEquals(PatternFamily.ORBIT_LOITER, top.family);
        Assert.assertTrue("Expected decent score", top.score > 0.55);
    }
}

```

```

    }

    @Test
    public void testApproachMatchesTop() {
        TrajectoryEventExtractor.Params p = new
TrajectoryEventExtractor.Params();
        p.approachMeanDR = -0.015;

        TrajectoryEventExtractor extractor = new TrajectoryEventExtractor(p);
        PatternFamilyMatcher matcher = new PatternFamilyMatcher(new
PatternFamilyMatcher.Weights());

        List<TrajectorySample> approach = synthApproach(0L, 60, 18.0, 0.0, -
0.05, 0.004);
        TrajectoryIR ir = TrajectoryIR.build(approach, extractor);

        PatternMatch top = matcher.matchTopK(ir, 1).get(0);
        Assert.assertEquals(PatternFamily.APPROACH, top.family);
        Assert.assertTrue(top.score > 0.55);
    }

    @Test
    public void testZigzagMatchesTop() {
        TrajectoryEventExtractor extractor = new TrajectoryEventExtractor(new
TrajectoryEventExtractor.Params());
        PatternFamilyMatcher matcher = new PatternFamilyMatcher(new
PatternFamilyMatcher.Weights());

        List<TrajectorySample> zigzag = synthZigzag(0L, 60, 12.0, 0.0, 0.08);
        TrajectoryIR ir = TrajectoryIR.build(zigzag, extractor);

        PatternMatch top = matcher.matchTopK(ir, 2).get(0);
        // Depending on params, zigzag may compete with sudden maneuver;
        assert in top-2.
        List<PatternMatch> top2 = matcher.matchTopK(ir, 2);
        Assert.assertTrue(
            top2.get(0).family == PatternFamily.ZIGZAG_SEARCH ||
            top2.get(1).family == PatternFamily.ZIGZAG_SEARCH
        );
    }

    @Test(expected = IllegalArgumentException.class)
    public void testRejectNonIncreasingTime() {
        TrajectoryEventExtractor extractor = new TrajectoryEventExtractor(new
TrajectoryEventExtractor.Params());

        List<TrajectorySample> bad = new ArrayList<>();
        bad.add(new TrajectorySample(100, 10, 0));
        bad.add(new TrajectorySample(100, 10, 0.1)); // non-increasing
        bad.add(new TrajectorySample(200, 10, 0.2));
        TrajectoryIR.build(bad, extractor);
    }

    private static List<TrajectorySample> synthOrbit(long t0, int n, double
range, double bearing0, double dBearing) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;

```

```

        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, range, b));
            t += 100;
            b += dBearing;
        }
        return out;
    }

    private static List<TrajectorySample> synthApproach(long t0, int n,
double range0, double bearing0, double dRange, double dBearing) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double r = range0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, r, b));
            t += 100;
            r += dRange;
            b += dBearing;
        }
        return out;
    }

    private static List<TrajectorySample> synthZigzag(long t0, int n, double
range, double bearing0, double amplitude) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, range, b));
            t += 100;
            // alternating sign to induce sign changes
            double step = (i % 2 == 0) ? amplitude : -amplitude;
            b += step;
        }
        return out;
    }
}
// File: pom.xml (JUnit4 only; Java 8)
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.dbm</groupId>
    <artifactId>trajectory-ir-mvp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
    </properties>

    <dependencies>
        <dependency>

```

```

        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
        <version>4.13.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
</project>

```

说明与“下一步最有价值的对齐点”

1. 接口对齐点（最小改动，高收益）

- 把 `TrajectoryIR.build(...)` 的 `TrajectoryEventExtractor` 改成你们 runtime 的 `IPhase1Provider` / `IEventOperator` 风格适配器（thin adapter）。
- 把 `PatternFamilyMatcher.matchTopK` 的输出改成你们习惯的 `SearchResult` / `EvidenceChain` 结构。

2. Step-5 / Step-8 / Step-9 的关键不变量（我们可以下一步补齐）

- Step-5：对 `events + signature` 做“边界抖动 / duration scaling / 插入删除惩罚”的 `mutation generator`。
- Step-8：多 family 的 aggregation（dominance + oscillation penalty）。
- Step-9：family-conditioned continuation（从当前 skeleton 推未来 skeleton）。

你如果希望我直接把这一套“更 DBM runtime 风味”地并入你们既有包名与接口风格（例如对齐你们的 `ir.runtime / IrPipelineV2` 语义），我也可以基于上述 MVP 继续给出 drop-in 版本（仍保持 Java 8 + JUnit4）。

请直接把这一套“更 DBM runtime 风味”地并入你们既有包名与接口风格（例如对齐你们的 `ir.runtime / IrPipelineV2` 语义），我也可以基于上述 MVP 继续给出 drop-in 版本（仍保持 Java 8 + JUnit4）。

```

// File: src/main/java/com/dbm/ir/runtime/v2/CheckResult.java
package com.dbm.ir.runtime.v2;

```

```

/**
 * Minimal runtime-style check result.
 * Keep it tiny and drop-in friendly (Java 8).
 */
public final class CheckResult {
    public final boolean ok;
    public final String message;
    public final Throwable error;
}

```

```

private CheckResult(boolean ok, String message, Throwable error) {
    this.ok = ok;
    this.message = message;
    this.error = error;
}

public static CheckResult ok() {
    return new CheckResult(true, "OK", null);
}

public static CheckResult fail(String message) {
    return new CheckResult(false, message, null);
}

public static CheckResult fail(String message, Throwable error) {
    return new CheckResult(false, message, error);
}

@Override
public String toString() {
    return "CheckResult{ok=" + ok + ", message='" + message + "', error="
+ (error == null ? "null" : error.getClass().getName()) + "}";
}
}
// File: src/main/java/com/dbm/ir/runtime/v2/IIRNode.java
package com.dbm.ir.runtime.v2;

/**
 * Runtime-style node marker.
 * In a larger DBM runtime this would unify IR nodes/events/operators.
 */
public interface IIRNode {
    String kind();
}
// File: src/main/java/com/dbm/ir/runtime/v2/IEvidence.java
package com.dbm.ir.runtime.v2;

/**
 * Minimal evidence object for explanation chains.
 */
public interface IEvidence extends IIRNode {
    String summary();
}
// File: src/main/java/com/dbm/ir/runtime/v2/EvidenceItem.java
package com.dbm.ir.runtime.v2;

public final class EvidenceItem implements IEvidence {
    private final String kind;
    private final String summary;

    public EvidenceItem(String kind, String summary) {
        this.kind = kind;
        this.summary = summary;
    }

    @Override

```

```

    public String kind() {
        return kind;
    }

    @Override
    public String summary() {
        return summary;
    }

    @Override
    public String toString() {
        return "EvidenceItem{kind='" + kind + "', summary='" + summary +
        "'}";
    }
}
// File: src/main/java/com/dbm/ir/runtime/v2/IEvidenceChain.java
package com.dbm.ir.runtime.v2;

import java.util.List;

public interface IEvidenceChain extends IIRNode {
    List<IEvidence> items();
}
// File: src/main/java/com/dbm/ir/runtime/v2/EvidenceChain.java
package com.dbm.ir.runtime.v2;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

public final class EvidenceChain implements IEvidenceChain {
    private final List<IEvidence> items;

    public EvidenceChain(List<IEvidence> items) {
        this.items = Collections.unmodifiableList(new ArrayList<>(items));
    }

    public static EvidenceChain of(IEvidence... items) {
        List<IEvidence> out = new ArrayList<>();
        if (items != null) {
            for (IEvidence e : items) {
                if (e != null) out.add(e);
            }
        }
        return new EvidenceChain(out);
    }

    @Override
    public String kind() {
        return "evidence_chain";
    }

    @Override
    public List<IEvidence> items() {
        return items;
    }
}

```

```

        @Override
        public String toString() {
            return "EvidenceChain{items=" + items + "}";
        }
    }
}
// File: src/main/java/com/dbm/ir/runtime/v2/IrResult.java
package com.dbm.ir.runtime.v2;

/**
 * Runtime-style result container:
 * - ok/fail
 * - value
 * - evidence
 */
public final class IrResult<T> {
    public final CheckResult check;
    public final T value;
    public final IEvidenceChain evidence;

    private IrResult(CheckResult check, T value, IEvidenceChain evidence) {
        this.check = check;
        this.value = value;
        this.evidence = evidence;
    }

    public static <T> IrResult<T> ok(T value, IEvidenceChain evidence) {
        return new IrResult<>(CheckResult.ok(), value, evidence);
    }

    public static <T> IrResult<T> fail(String message, IEvidenceChain
evidence) {
        return new IrResult<>(CheckResult.fail(message), null, evidence);
    }

    public static <T> IrResult<T> fail(String message, Throwable error,
IEvidenceChain evidence) {
        return new IrResult<>(CheckResult.fail(message, error), null,
evidence);
    }

    @Override
    public String toString() {
        return "IrResult{check=" + check + ", value=" + value + ", evidence="
+ evidence + "}";
    }
}
// File: src/main/java/com/dbm/ir/runtime/v2/IPhase1Provider.java
package com.dbm.ir.runtime.v2;

/**
 * Phase-1: raw input -> IR
 */
public interface IPhase1Provider<I, IR> {
    IrResult<IR> buildIr(I input);
}
// File: src/main/java/com/dbm/ir/runtime/v2/IMetricSearcher.java
package com.dbm.ir.runtime.v2;

```

```

/**
 * Phase-2: IR -> ranked matches / decisions
 */
public interface IMetricSearcher<IR, OUT> {
    IrResult<OUT> search(IR ir);
}
// File: src/main/java/com/dbm/ir/runtime/v2/IrPipelineV2.java
package com.dbm.ir.runtime.v2;

/**
 * Minimal "IrPipelineV2" semantics:
 * - Phase1Provider builds IR
 * - MetricSearcher consumes IR and yields output + evidence
 */
public final class IrPipelineV2<I, IR, OUT> {
    private final IPhase1Provider<I, IR> phase1;
    private final IMetricSearcher<IR, OUT> searcher;

    public IrPipelineV2(IPhase1Provider<I, IR> phase1, IMetricSearcher<IR,
OUT> searcher) {
        if (phase1 == null) throw new IllegalArgumentException("phase1 is
null");
        if (searcher == null) throw new IllegalArgumentException("searcher is
null");
        this.phase1 = phase1;
        this.searcher = searcher;
    }

    public IrResult<OUT> run(I input) {
        IrResult<IR> ir = phase1.buildIr(input);
        if (!ir.check.ok) return IrResult.fail("Phase1 failed: " +
ir.check.message, ir.evidence);
        return searcher.search(ir.value);
    }
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/TrajectorySample.java
package com.dbm.ir.runtime.trajectory;

/**
 * Observer-centric sample: time + range + bearing (2D).
 */
public final class TrajectorySample {
    public final long tMillis;
    public final double range;
    public final double bearing;

    public TrajectorySample(long tMillis, double range, double bearing) {
        this.tMillis = tMillis;
        this.range = range;
        this.bearing = bearing;
    }
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryInput.java
package com.dbm.ir.runtime.trajectory;

import java.util.ArrayList;

```

```

import java.util.Collections;
import java.util.List;

/**
 * Pipeline input wrapper (runtime flavor).
 */
public final class TrajectoryInput {
    public final String trackId;
    public final List<TrajectorySample> samples;

    public TrajectoryInput(String trackId, List<TrajectorySample> samples) {
        this.trackId = trackId == null ? "" : trackId;
        this.samples = Collections.unmodifiableList(new
ArrayList<>(samples));
    }
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryEventType.java
package com.dbm.ir.runtime.trajectory;

public enum TrajectoryEventType {
    APPROACH,
    RETREAT,
    ORBIT_LOITER,
    ZIGZAG_SEARCH,
    CRUISE_GLIDE,
    HOVER_STALL,
    SUDDEN_MANEUVER,
    UNKNOWN
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryEvent.java
package com.dbm.ir.runtime.trajectory;

public final class TrajectoryEvent {
    public final TrajectoryEventType type;
    public final int startIndexInclusive;
    public final int endIndexInclusive;
    public final double confidence;

    public TrajectoryEvent(TrajectoryEventType type, int startIndexInclusive,
int endIndexInclusive, double confidence) {
        this.type = type;
        this.startIndexInclusive = startIndexInclusive;
        this.endIndexInclusive = endIndexInclusive;
        this.confidence = confidence;
    }

    public int length() {
        return Math.max(0, endIndexInclusive - startIndexInclusive + 1);
    }

    @Override
    public String toString() {
        return "TrajectoryEvent{type=" + type +
            ", start=" + startIndexInclusive +
            ", end=" + endIndexInclusive +
            ", conf=" + String.format("%.3f", confidence) +
            '}';
    }
}

```

```

    }
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/TrajectorySignature.java
package com.dbm.ir.runtime.trajectory;

import java.util.Collections;
import java.util.EnumMap;
import java.util.Map;

/**
 * L3 signature: event skeleton + duration ratios + coarse motion indicators.
 */
public final class TrajectorySignature {
    public final String skeleton;
    public final Map<TrajectoryEventType, Double> durationRatio; // sum ~= 1
    public final double avgAbsDeltaRange;
    public final double avgAbsDeltaBearing;
    public final double bearingSignChangeRate;

    public TrajectorySignature(
        String skeleton,
        Map<TrajectoryEventType, Double> durationRatio,
        double avgAbsDeltaRange,
        double avgAbsDeltaBearing,
        double bearingSignChangeRate
    ) {
        this.skeleton = skeleton;
        this.durationRatio = Collections.unmodifiableMap(new
EnumMap<>(durationRatio));
        this.avgAbsDeltaRange = avgAbsDeltaRange;
        this.avgAbsDeltaBearing = avgAbsDeltaBearing;
        this.bearingSignChangeRate = bearingSignChangeRate;
    }

    @Override
    public String toString() {
        return "TrajectorySignature{" +
            "skeleton='" + skeleton + '\'' +
            ", durationRatio=" + durationRatio +
            ", avgAbsDeltaRange=" + String.format("%.4f",
avgAbsDeltaRange) +
            ", avgAbsDeltaBearing=" + String.format("%.4f",
avgAbsDeltaBearing) +
            ", bearingSignChangeRate=" + String.format("%.4f",
bearingSignChangeRate) +
            '}';
    }
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryIr.java
package com.dbm.ir.runtime.trajectory;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

/**
 * Runtime-friendly Trajectory IR:

```

```

* - L0 differentials
* - L2 events
* - L3 signature
*/
public final class TrajectoryIr {
    public final String trackId;
    public final List<TrajectorySample> samples;

    public final double[] dRange;
    public final double[] dBearing;
    public final long[] dTimeMillis;

    private final List<TrajectoryEvent> events;
    private final TrajectorySignature signature;

    public TrajectoryIr(
        String trackId,
        List<TrajectorySample> samples,
        double[] dRange,
        double[] dBearing,
        long[] dTimeMillis,
        List<TrajectoryEvent> events,
        TrajectorySignature signature
    ) {
        this.trackId = trackId == null ? "" : trackId;
        this.samples = Collections.unmodifiableList(new
ArrayList<>(samples));
        this.dRange = dRange;
        this.dBearing = dBearing;
        this.dTimeMillis = dTimeMillis;
        this.events = Collections.unmodifiableList(new ArrayList<>(events));
        this.signature = signature;
    }

    public List<TrajectoryEvent> events() {
        return events;
    }

    public TrajectorySignature signature() {
        return signature;
    }

    static double wrapAngle(double a) {
        while (a <= -Math.PI) a += 2.0 * Math.PI;
        while (a > Math.PI) a -= 2.0 * Math.PI;
        return a;
    }
}
// File:
src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryEventExtractor.java
package com.dbm.ir.runtime.trajectory;

import java.util.ArrayList;
import java.util.List;

/**
 * MVP extractor (runtime flavor).

```

```

* Intentionally heuristic; later we can swap to operator-graph style.
*/
public final class TrajectoryEventExtractor {

    public static final class Params {
        public int window = 5;

        public double approachMeanDR = -0.02;
        public double retreatMeanDR = 0.02;

        public double orbitMaxAbsMeanDR = 0.01;
        public double orbitMinAbsMeanDB = 0.04;

        public double hoverMaxAbsMeanDR = 0.005;
        public double hoverMaxAbsMeanDB = 0.01;

        public double zigzagMinSignChangeRate = 0.35;
        public double cruiseMaxAbsMeanDB = 0.02;

        public double suddenMaxAbsDB = 0.30;
    }

    private final Params p;

    public TrajectoryEventExtractor(Params p) {
        this.p = (p == null) ? new Params() : p;
        if (this.p.window < 3) this.p.window = 3;
    }

    public List<TrajectoryEvent> extract(List<TrajectorySample> samples,
double[] dRange, double[] dBearing) {
        final int n = samples.size();
        final int w = p.window;

        List<TrajectoryEvent> raw = new ArrayList<>();
        int i = 0;
        while (i + w <= n) {
            WindowStats st = stats(dRange, dBearing, i, i + w - 2);
            TrajectoryEventType type = classify(st);
            double conf = confidence(type, st);
            raw.add(new TrajectoryEvent(type, i, i + w - 1, conf));
            i += (w - 1);
        }

        return merge(raw);
    }

    private static final class WindowStats {
        final double meanDR;
        final double meanAbsDR;
        final double meanAbsDB;
        final double maxAbsDB;
        final double signChangeRateDB;

        WindowStats(double meanDR, double meanAbsDR, double meanAbsDB, double
maxAbsDB, double signChangeRateDB) {
            this.meanDR = meanDR;

```

```

        this.meanAbsDR = meanAbsDR;
        this.meanAbsDB = meanAbsDB;
        this.maxAbsDB = maxAbsDB;
        this.signChangeRateDB = signChangeRateDB;
    }
}

private WindowStats stats(double[] dR, double[] dB, int dFrom, int
dToInclusive) {
    int len = dToInclusive - dFrom + 1;
    if (len <= 0) return new WindowStats(0, 0, 0, 0, 0);

    double sumDR = 0, sumAbsDR = 0, sumAbsDB = 0, maxAbsDB = 0;
    int signChanges = 0;
    int signCount = 0;
    int prevSign = 0;

    for (int k = dFrom; k <= dToInclusive; k++) {
        double dr = dR[k];
        double db = dB[k];
        sumDR += dr;
        sumAbsDR += Math.abs(dr);

        double absDb = Math.abs(db);
        sumAbsDB += absDb;
        if (absDb > maxAbsDB) maxAbsDB = absDb;

        int s = sign(db);
        if (s != 0) {
            if (prevSign != 0 && s != prevSign) signChanges++;
            prevSign = s;
            signCount++;
        }
    }

    double rate = (signCount <= 1) ? 0.0 : ((double) signChanges /
(double) (signCount - 1));
    return new WindowStats(sumDR / len, sumAbsDR / len, sumAbsDB / len,
maxAbsDB, rate);
}

private TrajectoryEventType classify(WindowStats st) {
    if (st.maxAbsDB >= p.suddenMaxAbsDB) return
TrajectoryEventType.SUDDEN_MANEUVER;

    if (st.meanAbsDR <= p.hoverMaxAbsMeanDR && st.meanAbsDB <=
p.hoverMaxAbsMeanDB) {
        return TrajectoryEventType.HOVER_STALL;
    }

    if (Math.abs(st.meanDR) <= p.orbitMaxAbsMeanDR && st.meanAbsDB >=
p.orbitMinAbsMeanDB) {
        return TrajectoryEventType.ORBIT_LOITER;
    }

    if (st.signChangeRateDB >= p.zigzagMinSignChangeRate &&
st.meanAbsDB >= p.orbitMinAbsMeanDB * 0.6) {

```

```

        return TrajectoryEventType.ZIGZAG_SEARCH;
    }

    if (st.meanDR <= p.approachMeanDR) return
TrajectoryEventType.APPROACH;
    if (st.meanDR >= p.retreatMeanDR) return TrajectoryEventType.RETREAT;

    if (st.meanAbsDB <= p.cruiseMaxAbsMeanDB) return
TrajectoryEventType.CRUISE_GLIDE;

    return TrajectoryEventType.UNKNOWN;
}

private double confidence(TrajectoryEventType type, WindowStats st) {
    switch (type) {
        case SUDDEN_MANEUVER:
            return clamp01((st.maxAbsDB - p.suddenMaxAbsDB) /
(p.suddenMaxAbsDB * 0.8) + 0.6);
        case HOVER_STALL:
            return clamp01(1.0 - (st.meanAbsDR / (p.hoverMaxAbsMeanDR *
2.0) + st.meanAbsDB / (p.hoverMaxAbsMeanDB * 2.0)) * 0.5);
        case ORBIT_LOITER:
            return clamp01(0.6 + (st.meanAbsDB - p.orbitMinAbsMeanDB) /
(p.orbitMinAbsMeanDB * 2.0));
        case ZIGZAG_SEARCH:
            return clamp01(0.5 + st.signChangeRateDB);
        case APPROACH:
            return clamp01(0.5 + (-st.meanDR) /
(Math.abs(p.approachMeanDR) * 3.0));
        case RETREAT:
            return clamp01(0.5 + (st.meanDR) / (Math.abs(p.retreatMeanDR)
* 3.0));
        case CRUISE_GLIDE:
            return clamp01(0.6 + (p.cruiseMaxAbsMeanDB - st.meanAbsDB) /
(p.cruiseMaxAbsMeanDB * 2.0));
        default:
            return 0.35;
    }
}

private List<TrajectoryEvent> merge(List<TrajectoryEvent> raw) {
    List<TrajectoryEvent> out = new ArrayList<>();
    if (raw.isEmpty()) return out;

    TrajectoryEvent cur = raw.get(0);
    for (int i = 1; i < raw.size(); i++) {
        TrajectoryEvent nxt = raw.get(i);
        if (nxt.type == cur.type && nxt.startIndexInclusive <=
cur.endIndexInclusive + 1) {
            int lenA = cur.length();
            int lenB = nxt.length();
            double conf = (cur.confidence * lenA + nxt.confidence * lenB)
/ Math.max(1, lenA + lenB);
            cur = new TrajectoryEvent(cur.type, cur.startIndexInclusive,
Math.max(cur.endIndexInclusive, nxt.endIndexInclusive), conf);
        } else {
            out.add(cur);

```

```

        cur = nxt;
    }
}
out.add(cur);
return out;
}

private static int sign(double x) {
    if (x > 0) return 1;
    if (x < 0) return -1;
    return 0;
}

private static double clamp01(double v) {
    if (v < 0) return 0;
    if (v > 1) return 1;
    return v;
}
}
// File:
src/main/java/com/dbm/ir/runtime/trajectory/TrajectorySignatureBuilder.java
package com.dbm.ir.runtime.trajectory;

import java.util.EnumMap;
import java.util.List;
import java.util.Map;

public final class TrajectorySignatureBuilder {

    private TrajectorySignatureBuilder() {}

    public static TrajectorySignature build(List<TrajectorySample> samples,
double[] dRange, double[] dBearing, List<TrajectoryEvent> events) {
        int totalLen = Math.max(1, samples.size());
        Map<TrajectoryEventType, Double> dur = new
EnumMap<>(TrajectoryEventType.class);
        for (TrajectoryEventType t : TrajectoryEventType.values()) dur.put(t,
0.0);

        StringBuilder sk = new StringBuilder();
        TrajectoryEventType prev = null;

        for (TrajectoryEvent e : events) {
            int len = e.length();
            dur.put(e.type, dur.get(e.type) + (double) len / (double)
totalLen);

            if (prev == null) sk.append(e.type.name());
            else if (e.type != prev) sk.append("->").append(e.type.name());
            prev = e.type;
        }
        if (sk.length() == 0) sk.append(TrajectoryEventType.UNKNOWN.name());

        double sumAbsDR = 0, sumAbsDB = 0;
        int n = dRange.length;
        for (int i = 0; i < n; i++) {
            sumAbsDR += Math.abs(dRange[i]);

```

```

        sumAbsDB += Math.abs(dBearing[i]);
    }
    double avgAbsDR = sumAbsDR / Math.max(1, n);
    double avgAbsDB = sumAbsDB / Math.max(1, n);

    int signChanges = 0, signCount = 0;
    int prevSign = 0;
    for (int i = 0; i < n; i++) {
        int s = sign(dBearing[i]);
        if (s != 0) {
            if (prevSign != 0 && s != prevSign) signChanges++;
            prevSign = s;
            signCount++;
        }
    }
    double signChangeRate = (signCount <= 1) ? 0.0 : ((double)
signChanges / (double) (signCount - 1));

    return new TrajectorySignature(sk.toString(), dur, avgAbsDR,
avgAbsDB, signChangeRate);
}

private static int sign(double x) {
    if (x > 0) return 1;
    if (x < 0) return -1;
    return 0;
}
}
// File:
src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryPhase1Provider.java
package com.dbm.ir.runtime.trajectory;

import com.dbm.ir.runtime.v2.*;

import java.util.ArrayList;
import java.util.List;

/**
 * Phase1: TrajectoryInput -> TrajectoryIr
 * Aligns to IrPipelineV2 semantics.
 */
public final class TrajectoryPhase1Provider implements
IPhase1Provider<TrajectoryInput, TrajectoryIr> {

    private final TrajectoryEventExtractor extractor;

    public TrajectoryPhase1Provider(TrajectoryEventExtractor extractor) {
        this.extractor = extractor == null ? new TrajectoryEventExtractor(new
TrajectoryEventExtractor.Params()) : extractor;
    }

    @Override
    public IrResult<TrajectoryIr> buildIr(TrajectoryInput input) {
        try {
            if (input == null) return IrResult.fail("input is null",
EvidenceChain.of(new EvidenceItem("phase1", "input=null")));
            if (input.samples == null || input.samples.size() < 3) {

```

```

        return IrResult.fail("need >= 3 samples",
EvidenceChain.of(new EvidenceItem("phase1", "insufficient samples")));
    }

    final List<TrajectorySample> s = input.samples;
    final int n = s.size();

    double[] dR = new double[n - 1];
    double[] dB = new double[n - 1];
    long[] dT = new long[n - 1];

    for (int i = 0; i < n - 1; i++) {
        TrajectorySample a = s.get(i);
        TrajectorySample b = s.get(i + 1);

        long dt = b.tMillis - a.tMillis;
        if (dt <= 0) {
            return IrResult.fail("non-increasing time at i=" + i,
EvidenceChain.of(new EvidenceItem("phase1",
"dt<=0 at i=" + i))));
        }
        dT[i] = dt;

        dR[i] = (b.range - a.range);
        dB[i] = TrajectoryIr.wrapAngle(b.bearing - a.bearing);
    }

    List<TrajectoryEvent> events = extractor.extract(s, dR, dB);
    TrajectorySignature sig = TrajectorySignatureBuilder.build(s, dR,
dB, events);

    TrajectoryIr ir = new TrajectoryIr(input.trackId, s, dR, dB, dT,
events, sig);

    EvidenceChain ev = EvidenceChain.of(
        new EvidenceItem("phase1", "built TrajectoryIr: n=" + n),
        new EvidenceItem("signature", "skeleton=" + sig.skeleton)
    );
    return IrResult.ok(ir, ev);

    } catch (Exception e) {
        return IrResult.fail("exception in phase1", e,
EvidenceChain.of(new EvidenceItem("phase1", "exception=" +
e.getClass().getName())));
    }
}

// File: src/main/java/com/dbm/ir/runtime/trajectory/PatternFamily.java
package com.dbm.ir.runtime.trajectory;

/**
 * Known Pattern Families from ITEM #224.
 */
public enum PatternFamily {
    APPROACH,
    RETREAT_ESCAPE,
    ORBIT_LOITER,

```

```

        ZIGZAG_SEARCH,
        CRUISE_GLIDE,
        HOVER_STALL,
        SUDDEN_MANEUVER,
        UNKNOWN
    }
}
// File: src/main/java/com/dbm/ir/runtime/trajectory/PatternMatch.java
package com.dbm.ir.runtime.trajectory;

import com.dbm.ir.runtime.v2.IEvidenceChain;

public final class PatternMatch {
    public final PatternFamily family;
    public final double score; // higher is better
    public final IEvidenceChain evidence;

    public PatternMatch(PatternFamily family, double score, IEvidenceChain
evidence) {
        this.family = family;
        this.score = score;
        this.evidence = evidence;
    }

    @Override
    public String toString() {
        return "PatternMatch{family=" + family +
            ", score=" + String.format("%.4f", score) +
            ", evidence=" + evidence +
            '}';
    }
}
// File:
src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryPatternFamilyMatcher.ja
va
package com.dbm.ir.runtime.trajectory;

import com.dbm.ir.runtime.v2.*;

import java.util.*;

/**
 * Phase2 searcher: TrajectoryIr -> ranked PatternMatch list.
 * DBM runtime flavor:
 * - decomposable score: duration-dominant + geometry-refine
 * - evidence chain included per match
 */
public final class TrajectoryPatternFamilyMatcher implements
IMetricSearcher<TrajectoryIr, List<PatternMatch>> {

    public static final class Weights {
        public double wDuration = 0.75;
        public double wGeometry = 0.25;

        public double targetLow = 0.02;
        public double targetMed = 0.06;
        public double targetHigh = 0.12;
    }
}

```

```

        public int topK = 4;
    }

    private final Weights w;

    public TrajectoryPatternFamilyMatcher(Weights w) {
        this.w = (w == null) ? new Weights() : w;
    }

    @Override
    public IrResult<List<PatternMatch>> search(TrajectoryIr ir) {
        if (ir == null) return IrResult.fail("ir is null",
EvidenceChain.of(new EvidenceItem("phase2", "ir=null")));

        TrajectorySignature s = ir.signature();
        List<PatternMatch> all = new ArrayList<>();

        for (PatternFamily f : PatternFamily.values()) {
            double durScore = durationScore(f, s.durationRatio);
            double geoScore = geometryScore(f, s);
            double score = w.wDuration * durScore + w.wGeometry * geoScore;

            IEvidenceChain ev = EvidenceChain.of(
                new EvidenceItem("match", "family=" + f),
                new EvidenceItem("score", "dur=" + fmt(durScore) + ",
geo=" + fmt(geoScore) + ", total=" + fmt(score)),
                new EvidenceItem("skeleton", s.skeleton),
                new EvidenceItem("geom", "avg|dR|" +
fmt(s.avgAbsDeltaRange) +
                    ", avg|dB|" + fmt(s.avgAbsDeltaBearing) +
                    ", signFlip=" + fmt(s.bearingSignChangeRate))
            );
            all.add(new PatternMatch(f, score, ev));
        }

        all.sort(Comparator.comparingDouble((PatternMatch m) ->
m.score).reversed());

        int k = Math.max(1, w.topK);
        List<PatternMatch> top = (k >= all.size()) ? all : new
ArrayList<>(all.subList(0, k));

        IEvidenceChain root = EvidenceChain.of(
            new EvidenceItem("phase2", "matched families for trackId=" +
ir.trackId),
            new EvidenceItem("signature", "skeleton=" + s.skeleton)
        );

        // We attach root evidence at result-level; each match already
        carries its own chain.
        return IrResult.ok(top, root);
    }

    private double durationScore(PatternFamily f, Map<TrajectoryEventType,
Double> dur) {
        double a = dur.getDefault(TrajectoryEventType.APPROACH, 0.0);
        double r = dur.getDefault(TrajectoryEventType.RETREAT, 0.0);

```

```

double o = dur.getOrDefault(TrajectoryEventType.ORBIT_LOITER, 0.0);
double z = dur.getOrDefault(TrajectoryEventType.ZIGZAG_SEARCH, 0.0);
double c = dur.getOrDefault(TrajectoryEventType.CRUISE_GLIDE, 0.0);
double h = dur.getOrDefault(TrajectoryEventType.HOVER_STALL, 0.0);
double m = dur.getOrDefault(TrajectoryEventType.SUDDEN_MANEUVER,
0.0);
double u = dur.getOrDefault(TrajectoryEventType.UNKNOWN, 0.0);

switch (f) {
    case APPROACH:
        return clamp01(a + 0.2 * c - 0.2 * r);
    case RETREAT_ESCAPE:
        return clamp01(r + 0.2 * c - 0.2 * a);
    case ORBIT_LOITER:
        return clamp01(o + 0.1 * c - 0.1 * m);
    case ZIGZAG_SEARCH:
        return clamp01(z + 0.1 * m - 0.05 * c);
    case CRUISE_GLIDE:
        return clamp01(c + 0.1 * a + 0.1 * r - 0.15 * z - 0.15 * m);
    case HOVER_STALL:
        return clamp01(h - 0.05 * (a + r + z + m));
    case SUDDEN_MANEUVER:
        return clamp01(m + 0.15 * z - 0.1 * c);
    case UNKNOWN:
    default:
        return clamp01(0.5 * u + 0.1);
}

}

private double geometryScore(PatternFamily f, TrajectorySignature s) {
    double dr = s.avgAbsDeltaRange;
    double db = s.avgAbsDeltaBearing;
    double sc = s.bearingSignChangeRate;

    switch (f) {
        case HOVER_STALL:
            return 1.0 - clamp01((dr / w.targetLow + db / w.targetLow) *
0.5);
        case CRUISE_GLIDE:
            return 1.0 - clamp01(db / w.targetLow) * 0.8;
        case ORBIT_LOITER:
            return clamp01(db / w.targetMed) * 0.8 + clamp01(1.0 - dr /
w.targetMed) * 0.2;
        case ZIGZAG_SEARCH:
            return clamp01(sc) * 0.7 + clamp01(db / w.targetMed) * 0.3;
        case SUDDEN_MANEUVER:
            return clamp01(db / w.targetHigh);
        case APPROACH:
        case RETREAT_ESCAPE:
            return clamp01(dr / w.targetMed);
        default:
            return 0.25;
    }
}

}

private static String fmt(double v) {
    return String.format("%.3f", v);
}

```

```

    }

    private static double clamp01(double v) {
        if (v < 0) return 0;
        if (v > 1) return 1;
        return v;
    }
}

// File:
src/main/java/com/dbm/ir/runtime/trajectory/TrajectoryIrPipelineV2Factory.java
a
package com.dbm.ir.runtime.trajectory;

import com.dbm.ir.runtime.v2.IrPipelineV2;

/**
 * Convenience factory aligned to "IrPipelineV2 semantics".
 */
public final class TrajectoryIrPipelineV2Factory {

    private TrajectoryIrPipelineV2Factory() {}

    public static IrPipelineV2<TrajectoryInput, TrajectoryIr,
java.util.List<PatternMatch>> createDefault() {
        TrajectoryEventExtractor extractor = new TrajectoryEventExtractor(new
TrajectoryEventExtractor.Params());
        TrajectoryPhase1Provider phasel = new
TrajectoryPhase1Provider(extractor);

        TrajectoryPatternFamilyMatcher.Weights w = new
TrajectoryPatternFamilyMatcher.Weights();
        TrajectoryPatternFamilyMatcher matcher = new
TrajectoryPatternFamilyMatcher(w);

        return new IrPipelineV2<>(phasel, matcher);
    }
}

// File:
src/test/java/com/dbm/ir/runtime/trajectory/TrajectoryIrPipelineV2Test.java
package com.dbm.ir.runtime.trajectory;

import com.dbm.ir.runtime.v2.IrPipelineV2;
import com.dbm.ir.runtime.v2.IrResult;
import org.junit.Assert;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;

public final class TrajectoryIrPipelineV2Test {

    @Test
    public void testOrbitLoiterTop1() {
        IrPipelineV2<TrajectoryInput, TrajectoryIr, List<PatternMatch>> p =
TrajectoryIrPipelineV2Factory.createDefault();
        TrajectoryInput in = new TrajectoryInput("orbit-1", synthOrbit(0L,
60, 10.0, 0.0, 0.09));

```

```

        IrResult<List<PatternMatch>> out = p.run(in);

        Assert.assertTrue(out.check.ok);
        Assert.assertNotNull(out.value);
        Assert.assertFalse(out.value.isEmpty());

        PatternMatch top = out.value.get(0);
        Assert.assertEquals(PatternFamily.ORBIT_LOITER, top.family);
        Assert.assertTrue("score should be decent", top.score > 0.55);
    }

    @Test
    public void testApproachTop1() {
        IrPipelineV2<TrajectoryInput, TrajectoryIr, List<PatternMatch>> p =
        TrajectoryIrPipelineV2Factory.createDefault();
        TrajectoryInput in = new TrajectoryInput("approach-1",
        synthApproach(0L, 60, 18.0, 0.0, -0.05, 0.004));
        IrResult<List<PatternMatch>> out = p.run(in);

        Assert.assertTrue(out.check.ok);
        PatternMatch top = out.value.get(0);
        Assert.assertEquals(PatternFamily.APPROACH, top.family);
        Assert.assertTrue(top.score > 0.55);
    }

    @Test
    public void testZigzagInTop2() {
        IrPipelineV2<TrajectoryInput, TrajectoryIr, List<PatternMatch>> p =
        TrajectoryIrPipelineV2Factory.createDefault();
        TrajectoryInput in = new TrajectoryInput("zigzag-1", synthZigzag(0L,
        70, 12.0, 0.0, 0.08));
        IrResult<List<PatternMatch>> out = p.run(in);

        Assert.assertTrue(out.check.ok);

        List<PatternMatch> top = out.value;
        Assert.assertTrue(top.size() >= 2);

        boolean ok =
            top.get(0).family == PatternFamily.ZIGZAG_SEARCH ||
            top.get(1).family == PatternFamily.ZIGZAG_SEARCH;
        Assert.assertTrue("zigzag should appear in top-2", ok);
    }

    @Test
    public void testRejectNonIncreasingTime() {
        IrPipelineV2<TrajectoryInput, TrajectoryIr, List<PatternMatch>> p =
        TrajectoryIrPipelineV2Factory.createDefault();

        List<TrajectorySample> bad = new ArrayList<>();
        bad.add(new TrajectorySample(100, 10, 0));
        bad.add(new TrajectorySample(100, 10, 0.1)); // non-increasing
        bad.add(new TrajectorySample(200, 10, 0.2));

        IrResult<List<PatternMatch>> out = p.run(new TrajectoryInput("bad-1",
        bad));
        Assert.assertFalse(out.check.ok);
    }

```

```

        Assert.assertTrue(out.check.message.contains("Phase1 failed"));
    }

    private static List<TrajectorySample> synthOrbit(long t0, int n, double
range, double bearing0, double dBearing) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, range, b));
            t += 100;
            b += dBearing;
        }
        return out;
    }

    private static List<TrajectorySample> synthApproach(long t0, int n,
double range0, double bearing0, double dRange, double dBearing) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double r = range0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, r, b));
            t += 100;
            r += dRange;
            b += dBearing;
        }
        return out;
    }

    private static List<TrajectorySample> synthZigzag(long t0, int n, double
range, double bearing0, double amplitude) {
        List<TrajectorySample> out = new ArrayList<>();
        long t = t0;
        double b = bearing0;
        for (int i = 0; i < n; i++) {
            out.add(new TrajectorySample(t, range, b));
            t += 100;
            double step = (i % 2 == 0) ? amplitude : -amplitude;
            b += step;
        }
        return out;
    }
}

<!-- File: pom.xml (Java 8 + JUnit4) -->
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>com.dbm</groupId>
    <artifactId>dbm-ir-trajectory-runtime-v2-mvp</artifactId>
    <version>1.0-SNAPSHOT</version>

    <properties>

```

```

    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>
</project>

```

这版“更 DBM runtime 风味”的关键点（直接可 drop-in）

1. IrPipelineV2 语义对齐

- IPhase1Provider<TrajectoryInput, TrajectoryIr>：负责 IR 构建（含 L0/L2/L3），并输出 evidence。
- IMetricSearcher<TrajectoryIr, List<PatternMatch>>：负责匹配并输出 rank + evidence。
- IrPipelineV2.run(input)：串起来即跑。

2. EvidenceChain 是一等公民

- Phase1 给出 “built IR + skeleton”
 - Phase2 给出 “family + (dur/geo/total) + skeleton + geom stats”
- 这为你们后续 Step-8/9 的解释链扩展打好了结构位。

3. 后续你们 repo 真正对齐时，改动点极少

- 若你们已有 IrPipelineV2 / IPhase1Provider / IMetricSearcher / Evidence：只需把本文件夹下 com.dbm.ir.runtime.v2.* 删除，改为 import 你们已有接口，并做一层 thin adapter 即可。
- 轨迹域代码（com.dbm.ir.runtime.trajectory.*）可原封不动迁移。

如果你把你们 repo 里 **IrPipelineV2 / EvidenceChain / SearchResult** 的真实类名（或任意一段现有接口签名）贴一小段出来，我们就可以把这套代码从“语义对齐”升级到“签名完全对齐”，实现真正的零摩擦 drop-in。