防空レーダーシステム仕様概要 (更新版 v3)

1. システム目標

搭載された車両や構造物に対して、周囲の空中目標（航空機など）を探知・追跡し、そのグローバル座標 (Physics Sensor 座標系) を提供することを目的とする。拡張カルマンフィルター (EKF) を用いて、ノイズを含むレーダー観測値から目標の正確な位置と速度を推定し、敵対的な目標を識別して出力する。

2. ハードウェア構成

- **レーダー:** 4基のレーダーコンポーネントを使用。
  - 各レーダーは、方位角 (水平) FOV を狭く、仰角 (垂直) FOV を広く設定。
  - 4基のレーダーを垂直に積み重ね、**前向き (ID 0, 頂点)**、**右向き (ID 1)**、**後ろ向き (ID 2)**、**左向き (ID 3, 最下部)** に90度ずつ回転させて設置。
  - 各レーダーの最大探知距離 (EffectiveRange) は、レーダー自体の内部探知間隔 (DetectionInterval) が **2 tick** となるように調整されている。（Stormworks 仕様書参照）
  - *(オプション)* 高仰角レーダー (`PackandFilterRefactorHighElevation.lua` を使用する場合) は、物理的に上向きに設置され、座標変換ロジックが異なる。
- **物理センサー (Physics Sensor):** 1基。
  - 垂直に積み重ねられたレーダー群の一番下に取り付けられ、レーダー群全体と共にピボットで回転する。
  - センサーの向きは **Radar 0 と同じ方向（正面）**を向くように設置する。
  - 車両のグローバル座標 (X:東, Y:上, Z:北) および姿勢 (オイラー角 Pitch, Yaw, Roll) を取得する。
  - このセンサーの位置が、システム全体の**座標計算の基準点**となる。
- **マイクロコントローラー:** 以下のLuaスクリプトを実行するマイクロコントローラー群で構成される。
  - `PackandFilterRefactor.lua` x 4基 (各レーダーに対応、`RADAR_ID` を 0, 1, 2, 3 に設定)
    - *(または `PackandFilterRefactorHighElevation.lua` x N基)*
  - `RadarListRefactor.lua` x 2基 (RadarList1 は ID 0, 2 を担当、RadarList2 は ID 1, 3 を担当)
    - *(注: `RadarListRefactorv2.lua` は入力目標数が8に拡張されている)*
  - `KalmanFilterRefactor.lua` x 1基 (システム全体の処理を担当)

3. マイクロコントローラーアーキテクチャとデータフロー

(注: Physics Sensor は KF だけでなく、座標変換等で必要になる場合があるため、関連する全マイコンに接続される構成が一般的です。図では KF への直接接続と、RadarList 経由のパススルーを示しています。)

4. 各処理ステップ詳細

4.1. レーダー出力 (変更なし)

- 各レーダーは、探知した目標（最大8目標）の情報をローカル極座標（距離、方位角、仰角）と探知経過時間tickでコンポジット信号として出力する。
- 内部探知間隔は設計上 **2 tick** に調整されている。観測値にはノイズが含まれる。
- 探知からの経過時間 (数値ch4) は、レーダー内部の「真の値」が更新された瞬間に 0 となる。

4.2. `PackandFilterRefactor.lua`

- **役割:** 各レーダーからの入力を受け、フィルタリングとデータ圧縮を行う。
- **フィルター:**
  - レーダーの探知更新タイミング (入力ch4 が 0 になった時) を基準とする。
  - プロパティ `n` で指定された期間 (通常は探知間隔に合わせて `n=2` に設定) における、各検出目標の距離・方位角・仰角の **最大値** と **最小値** を記録する (`targetMaxData`, `targetMinData`)。
  - 探知更新タイミング (次の tick で ch4 が 0 になる時) に、記録した最大値と最小値から **範囲中間値 (max + min) / 2** を計算する。
- **データ圧縮 (`packTargetData` 関数):**
  - 計算された中間値（距離、方位角、仰角）を2つの **7桁** の数値 `pack1`, `pack2` に圧縮する。
  - **エンコード方式:**
    - **距離:** 整数に四捨五入後 (`math.floor(distance + 0.5)`)、ゼロ埋め4桁の文字列 (`string.format("%04d", intDistance)`) に変換。前半2桁を `pack1` の末尾2桁に、後半2桁を `pack2` の末尾2桁に格納。(最大9999m)
    - **方位角/仰角:**
      - **符号:** `getSignCode` 関数で `'1'` (負) または `'2'` (正/ゼロ) の文字列を取得。
      - **絶対値:** `math.abs(angle) + 0.00005` で微小値を加算後、`string.format("%f", ...)` で文字列化し、小数点以下4桁を抽出 (`string.sub(..., dotPos + 1, dotPos + 4)`)。4桁に満たない場合はゼロ埋め。
      - **pack1:** `aziSignCode(1桁) + aziFractionStr(4桁) + distPart1(2桁)` を数値化。
      - **pack2:** `eleSignCode(1桁) + eleFractionStr(4桁) + distPart2(2桁)` を数値化。
  - **レーダーIDエンコード:**
    - スクリプト冒頭で設定された `RADAR_ID` (0-3) に基づき、`pack1`, `pack2` の **符号** を決定して出力する。
      - ID 0 (Front): (-, -)
      - ID 1 (Right): (-, +)
      - ID 2 (Back): (+, -)
      - ID 3 (Left): (+, +)
- **出力:** 圧縮された `pack1`, `pack2` のペアを最大 **4目標分** (**8チャンネル**) 出力する。プロパティ `Min Dist` 以下の距離の目標は出力しない。
- **高仰角版 (`PackandFilterRefactorHighElevation.lua`):**
  - 物理的に上向き設置されたレーダーからの入力を想定。
  - 入力されたローカル極座標を、**水平設置レーダー基準のローカル極座標に変換**してからフィルタリングと圧縮を行う。
  - 変換には `FRONT_OFFSET` と `HEIGHT_OFFSET` 定数を使用する。

4.3. `RadarListRefactor.lua`

- **役割:** 2つの `PackandFilterRefactor` からのデータ (それぞれ最大4目標) を集約し、`KalmanFilterRefactor` へ渡すデータを整理する。RadarList1 (ID 0, 2担当) と RadarList2 (ID 1, 3担当) の2基が独立して動作する。
- **データ集約:** 2系統 (ch 1-8 と ch 9-16、またはスクリプトにより異なる) からの入力データを結合する (`allTargets` リスト)。
- **オーバーフロー処理:**
  - 入力された合計目標数が自身の出力上限 (**6目標**) を超えた場合、そのTickでは6目標分のみを `targetsToOutput` リストに入れる。

    - 出力しきれなかったデータ (`allTargets` の残り) は内部の `overflowBuffer` に一時保存する。
- **遅延出力とフラグ:**
    - `overflowBuffer` にデータが保存された場合、**次のTick** で `isOutputtingOverflow` フラグが `true` になり、バッファ内のデータが `targetsToOutput` として出力される。
    - 遅延データを出力するTickでは、出力チャンネル **31** (RadarList1) または **32** (RadarList2) に **1** (遅延フラグ) をセットする。
    - 通常データを出力するTickでは、対応する遅延フラグチャンネルは **0** となる。
- **物理センサーデータパススルー:** 入力された物理センサーデータ (ch 25-30) をそのまま出力チャンネル 25-30 に書き込む。
- **出力:** 整理された `pack1`, `pack2` のペア (最大6目標分、チャンネル 1-12)、物理センサーデータ (チャンネル 25-30)、遅延フラグ (チャンネル31または32) を出力する。
- **`RadarListRefactorv2.lua`:** `MAX_INPUT_TARGETS_PER_SOURCE` が 8 に設定されており、より多くの入力を処理できる可能性がある。

4.4. `KalmanFilterRefactor.lua`

- **役割:** システムの中核。目標情報の展開、座標変換、EKFによる追跡、データアソシエーション、目標管理、敵対判定、最終的な座標出力を行う。
- **入力:**
    - 数値 1-12: RadarList1 からの出力 (`pack1`, `pack2` x 6目標)
    - 数値 13-24: RadarList2 からの出力 (`pack1`, `pack2` x 6目標)
    - 数値 25-30: 物理センサーデータ (X, Y, Z, Pitch, Yaw, Roll) - RadarList経由または直接接続
    - 数値 31: RadarList1 からの遅延フラグ (`isDelayed1`)
    - 数値 32: RadarList2 からの遅延フラグ (`isDelayed2`)
- **データ展開 (`unpackTargetData` 関数):**
    - 入力された `pack1`, `pack2` をデコードし、距離(m)、ローカル方位角(rad)、ローカル仰角(rad)、レーダーID(0-3) を復元する。
    - `PackandFilterRefactor.lua` の `packTargetData` 関数に対応した7桁数値の構造を解析する。
- **座標変換 (`localToGlobalCoords` 関数):**
    - レーダー基準のローカル直交座標を計算 (`locX`, `locY`, `locZ`)。
    - `radarId` に基づき **ヨー回転** を行い、車両前方基準のローカル座標に変換 (`vehLocVec_rotated`)。
        - ID 0: 回転なし
        - ID 1: +90度 (PI/2) 回転
        - ID 2: +180度 (PI) 回転
        - ID 3: -90度 (-PI/2) 回転
    - レーダーのY軸オフセット `2.5 / (rId + 1)` を追加。
    - 車両の姿勢 (`pitch`, `yaw`, `roll`) に基づき **クォータニオン回転** (`eulerZYX_to_quaternion`, `rotateVectorByQuaternion`) を適用し、グローバルな**相対ベクトル**を得る (`globalRelativeVector`)。
    - 物理センサーのグローバル座標 (`physicsSensorData.x`, `.y`, `.z`) を加算し、最終的な目標のグローバル座標 (Physics Sensor 座標系) を得る (`gX`, `gY`, `gZ`)。
- **EKF (`extendedKalmanFilterUpdate` 関数):**
    - **状態ベクトル X:** `[x, vx, y, vy, z, vz]^T` (グローバル位置・速度、計6状態)
    - **予測ステップ:**
        - 状態遷移行列 F を作成 (等速直線運動モデル)。
        - 状態 X を予測: `X_predicted = mul(F, stateVector)`。

        - **プロセスノイズ共分散行列 Q** を計算:
            - 基本となる `Q_base` を `dt` (時間差) から計算。
            - 適応的係数 `adaptiveFactor` を、前回の誤差 `lastEpsilon` とパラメータ (`PROCESS_NOISE_BASE`, `PROCESS_NOISE_ADAPTIVE_SCALE`, `PROCESS_NOISE_EPSILON_THRESHOLD`, `PROCESS_NOISE_EPSILON_SLOPE`) を用いて計算。
            - `Q_adapted = scalar(adaptiveFactor, Q_base)`。
        - 誤差共分散 P を予測: `P_predicted = sum(scalar(uncertaintyIncreaseFactor, mul(F, covariance, T(F))), Q_adapted)`。
        - `uncertaintyIncreaseFactor` は `PREDICTION_UNCERTAINTY_FACTOR_BASE ^ (2 * (dt * 60))` で計算。
    - **更新ステップ:**
        - 観測値 Z (`distance`, `elevation`, `azimuth`) を用意。
        - 観測ヤコビ行列 H と観測予測値 h を `getObservationJacobianAndPrediction` 関数で計算。
        - **観測ノイズ共分散行列 R** を計算: テンプレート `OBSERVATION_NOISE_MATRIX_TEMPLATE` を基に、距離の分散 `R[1][1]` を `distance^2` でスケール。
        - イノベーション (観測残差) Y を計算: `Y = Z - h`。角度差は `CalculateAngleDifference` (atan2相当) を使用して正規化。
        - カルマンゲイン K を計算: `K = mul(P_predicted, T(H), inv(sum(mul(H, P_predicted, T(H)), R)))`。
        - 状態 X を更新: `X_updated = sum(X_predicted, mul(K, Y))`。
        - 共分散 P を更新: `P_updated = sum(mul(I_minus_KH, P_predicted, T(I_minus_KH)), mul(K, R, T(K)))` (Joseph form)。
        - 誤差指標 `epsilon` を計算: `epsilon = mul(T(Y), S_inv, Y)[1][1]`。
    - `dt` **(時間差) 計算:**
        - `kalmanfilter.lua` 内部の `currentTick` カウンターを使用。
        - 各観測データについて、入力された遅延フラグ (`isDelayed1` or `isDelayed2`) を確認。
        - 観測データの真の発生Tickを推定: `observationTick = isDelayed and (currentTick - 1) or currentTick`。
        - EKFで使用する時間差 (秒) を計算: `dt_ticks = observation.obsTick - currentTarget.lastTick` (前回更新Tickからの差)、`dt_sec = dt_ticks / 60.0`。
- **データアソシエーション:**
    - 既存の各追跡目標 (`targetList`) に対して、全ての新規観測データ (`currentObservations`) でEKF更新を **試算**。
    - 更新時の誤差指標 `epsilon` が最小で、かつプロパティ `D_ASOC` 以下の観測データを、その目標に対応するものとして割り当てる (**最近傍法**)。
- **目標管理:**
    - **新規登録:** どの既存目標にも割り当てられなかった観測データを新規目標として登録。
        - 内部ID (`internalId`) を付与。出力ID (`outputId`) は `nil` で初期化。
        - 初期位置は観測時のグローバル座標、初期速度はゼロ。
        - 初期誤差共分散 `P_init` は、位置の分散を観測ノイズから、速度の分散を `INITIAL_VELOCITY_VARIANCE` から設定。
    - **削除:**
        - 一定時間 (プロパティ `T_OUT`) 更新されなかった目標。

        - または、自機から離反している (接近速度 `closingSpeed` < プロパティ `TGT_LVING`) と判定された目標をリストから削除。
        - 削除時に割り当てられていた `outputId` を解放 (`releaseOutputId`)。
- **敵対判定 (`checkHostileCondition`):**
    - 目標ごとに `identification_count` (EKF更新成功回数) と `recent_closing_speeds` (直近の接近速度リスト) を記録。
    - `identification_count` が `HOSTILE_IDENTIFICATION_THRESHOLD` 以上、**かつ**、直近 `HOSTILE_RECENT_UPDATES_THRESHOLD` 回の `closingSpeed` がすべて `HOSTILE_CLOSING_SPEED_THRESHOLD` を超えている場合に、`target.is_hostile` フラグを `true` に設定。
- **出力ID管理 (`assignOutputId`, `releaseOutputId`):**
    - `target.is_hostile` が `true` になった目標に対して、空いている出力ID (1～`MAX_TRACKED_TARGETS`) を割り当てる (`assignOutputId`)。
    - `target.is_hostile` が `false` になった、または目標が削除される場合に、割り当てられていた出力IDを解放する (`releaseOutputId`)。
- **出力:**
    - **Output ID が割り当てられている** 追跡中の各目標について、現在の `currentTick` における **予測位置** (X, Y, Z) を計算 (`target.X[1][1] + target.X[2][1] * dt_pred_sec`, etc.)。
    - 対応する **Output ID に基づく固定チャンネル** に座標を出力。
        - 目標 Output ID `i` の座標はチャンネル `(i-1)*3 + 1` (X), `(i-1)*3 + 2` (Y), `(i-1)*3 + 3` (Z) に出力。
    - 出力は最大 `MAX_TRACKED_TARGETS` (デフォルト10) 目標まで。
    - オンオフ出力チャンネル 1～`MAX_TRACKED_TARGETS` に、対応する Output ID の目標が **そのTickで更新されたかどうか** を示すフラグ (`target.isUpdated`) を出力。

5. 座標系 (変更なし)

- **レーダーローカル:** 各レーダーの正面方向を基準とする極座標および直交座標。
- **車両ローカル:** 車両の前方を基準とする直交座標 (+X:右, +Y:上, +Z:前)。`localToGlobalCoords` 内で一時的に使用。
- **グローバル:** Stormworks の **Physics Sensor 座標系** (**左手系**: +X:東, +Y:上, +Z:北) を基準とする。最終出力はこの座標系。

6. 主要パラメータ・設定項目

- **`PackandFilterRefactor.lua`:**
    - `RADAR_ID`: 0, 1, 2, 3 のいずれかを手動設定。
    - プロパティ `n`: フィルター期間 (tick)。
    - プロパティ `Min Dist`: 最小探知距離 (m)。
- **`PackandFilterRefactorHighElevation.lua`:**
    - 上記に加え、`FRONT_OFFSET`, `HEIGHT_OFFSET` 定数。
- **`KalmanFilterRefactor.lua`:**
    - プロパティ `D_ASOC`: データアソシエーション閾値 (epsilon)。
    - プロパティ `T_OUT`: 目標タイムアウトtick数。
    - プロパティ `TGT_LVING`: 目標離反判定の接近速度閾値 (m/s)。
    - プロパティ `P_BASE`: プロセスノイズの基本係数。
    - プロパティ `P_ADPT`: プロセスノイズの適応的スケーリング係数。

    - プロパティ `P_NOISE_EPS_THRS`: プロセスノイズ適応調整のepsilon閾値。
    - プロパティ `P_NOISE_EPS_SLOPE`: プロセスノイズ適応調整のepsilon傾き。
    - プロパティ `PRED_UNCERTAINTY_FACT`: 予測不確かさ増加係数の底。
    - プロパティ `IDENTI_THRS`: 敵対判定に必要な同定成功回数。
    - プロパティ `TGT_CLOSING_SPD`: 敵対判定の接近速度閾値 (m/s)。
    - プロパティ `TGT_RECENT_UPDATE`: 敵対判定に必要な閾値超えの連続更新回数。
    - (コード内定数) `MAX_TRACKED_TARGETS`: 最大追跡・出力目標数。
    - (コード内定数) `INITIAL_VELOCITY_VARIANCE`: 新規目標の初期速度分散。
    - (コード内定数) `OBSERVATION_NOISE_MATRIX_TEMPLATE`: 観測ノイズの基本設定。

7. 関連システム (参考)

以下のスクリプトは、本レーダーシステムからの出力を利用する火器管制システム (FCS) の一部と考えられます。これらの仕様は本ドキュメントの範囲外ですが、連携のために存在を記載します。

- `PackRefactor.lua` **(MissileFCS):**
    - `KalmanFilterRefactor.lua` から出力される敵対目標座標 (Output ID ベース) を入力。
    - 共有バスを通じて他のFCSと連携し、担当目標を選択・ロックオン。
    - 発射トリガーとミサイル信号強度を監視し、発射信号とデータリンク座標を出力。
- `VLSFCS.lua`**:**
    - 垂直発射システム (VLS) 用のFCS。
    - `PackRefactor.lua` と同様にKF出力を受け取る。
    - ハッチ制御、発射タイミング調整、競合チェックなどのロジックを含む。

8. 現状と課題

- 各スクリプトはリファクタリングされ、可読性と機能性が向上した。
- データ圧縮/展開、遅延Tick管理、座標変換ロジックが新しい実装に合わせて更新された。
- 敵対判定ロジックが追加され、条件を満たす目標に対して Output ID が割り当てられ、固定チャンネルで出力されるようになった。
- 課題:
    - EKFパラメータや目標管理パラメータは、実際の運用環境に合わせて **さらなる調整・最適化が必要** となる可能性がある。
    - データアソシエーションは最近傍法であり、目標が交差する場合などに誤割り当てが発生する可能性がある。
    - 高仰角レーダーやVLSFCSなど、特定の構成要素に関する詳細な仕様統合が必要な場合がある。

```
graph LR
    subgraph Radar_Units
        R0[Radar 0 Front]
        R1[Radar 1 Right]
        R2[Radar 2 Back]
        R3[Radar 3 Left]
    end
    subgraph Preprocessing
        PF0(PackandFilter ID 0)
        PF1(PackandFilter ID 1)
        PF2(PackandFilter ID 2)
```

```
        PF3(PackandFilter ID 3)
    end
    subgraph Aggregation
        RL1(RadarList 1)
        RL2(RadarList 2)
    end
    KF(KalmanFilter)
    PS[Physics Sensor]
    Out((Output))

    R0 --> PF0
    R1 --> PF1
    R2 --> PF2
    R3 --> PF3

    PF0 --> RL1
    PF2 --> RL1
    PF1 --> RL2
    PF3 --> RL2

    PS -- "ch 1-6 (Pose/Pos)" --> RL1
    PS -- "ch 1-6 (Pose/Pos)" --> RL2
    PS -- "ch 1-6 (Pose/Pos)" --> KF

    RL1 -- "ch 1-12 (Data), ch 25-30 (Pose/Pos Passthrough), ch 31 (DelayFlag1)" -
-> KF
    RL2 -- "ch 13-24 (Data), ch 25-30 (Pose/Pos Passthrough), ch 32 (DelayFlag2)"
--> KF

    KF -- "ch 1-30 (Target Coords), ch 1-10 (Update Flags)" --> Out
```