Occlusion and Re-identification System for Opt-in Camera

Overview

This system extends the Opt-in Camera person identification approach to handle **occlusion scenarios** where a person temporarily leaves the camera's field of view and then re-enters. The system uses the Unscented Kalman Filter (UKF) to maintain trajectory estimates during occlusion periods and implements sophisticated re-identification logic to match returning persons to their previous identities.

Problem Statement

Challenges When a Person Goes Out of Frame

When a person carrying a UWB tag temporarily leaves the camera's field of view (FOV), several critical issues arise:

1. **Camera Tracklet Breaks:** The Multi-Object Tracking (MOT) algorithm loses the person, creating a gap in the camera tracklet.
2. **New Tracklet ID:** When the person re-enters the frame, the MOT algorithm may assign a **new tracklet ID** to what is actually the same person.
3. **Re-identification Problem:** The system must determine: "Is this new camera tracklet the same person who left earlier, or a different person?"
4. **Trajectory Continuity:** The system must maintain a continuous trajectory estimate for the person, even when they are not visible in the camera.

Solution Architecture

Key Components

1. UKF with Occlusion Handling (`uwb_ukf_occlusion.py`)

The UKF continues to predict the person's position even when they are out of frame:

- **State Vector:** `[x, v_x, y, v_y]` (position and velocity in 2D)
- **Motion Model:** Constant Velocity
- **Prediction Step:** Runs at 30 FPS (camera frame rate), even during occlusion

- **Update Step:** Only runs when a camera measurement is available
- **Covariance Tracking:** Maintains and grows uncertainty during occlusion periods

Key Features:

- `predict(frame_number)` : Predict state without measurement
- `update(measurement, frame_number)` : Correct state with measurement
- `start_occlusion(frame_number)` : Mark beginning of occlusion
- `end_occlusion(frame_number)` : Mark end of occlusion
- `get_uncertainty()` : Get position uncertainty (trace of covariance)

2. Re-identification Matcher (`reidentification_logic.py`)

Matches new camera tracklets to previously occluded persons using multiple strategies:

Matching Strategies:

Strategy	Description	Formula
Temporal Proximity	Persons that disappeared recently are more likely to be the same	`score = 1 - (time_gap / threshold)`
Spatial Proximity	Tracklets near predicted position are more likely to match	`score = 1 - (distance / threshold)`
Mahalanobis Distance	Probabilistic distance based on UKF uncertainty	`D = sqrt((x - x_pred)^T * P^-1 * (x - x_pred))`
Velocity Consistency	Similar velocity direction indicates same person	`score = (cos_angle + 1) / 2`
Confidence Scoring	Weighted combination of all metrics	`confidence = 0.2*temporal + 0.3*spatial + 0.2*velocity + 0.3*mahal`

Key Methods:

- `register_missing_person()` : Register a person as missing
- `predict_position_at_frame()` : Predict where person should be
- `calculate_confidence_score()` : Calculate overall re-identification confidence
- `match_new_tracklet()` : Attempt to match a new tracklet

- `match_multiple_tracklets()` : Match multiple tracklets using Hungarian algorithm

3. Occlusion Tracker (`uwb_ukf_occlusion.py`)

Manages multiple UKF instances for different persons:

- Tracks active tracklets
- Manages missing tracklets
- Handles multiple persons simultaneously

4. Sample Data Generator (`sample_data_occlusion.py`)

Generates realistic test scenarios:

- **Scenario 1:** Simple occlusion with re-entry from same side
- **Scenario 2:** Multiple persons with overlapping occlusion periods
- **Scenario 3:** Long occlusion period (tests hypothesis expiration)

System Workflow

Frame-by-Frame Processing

Plain Text

```
For each frame:
  1. UKF Prediction
     - Predict position using constant velocity model
     - Update covariance (uncertainty grows during occlusion)

  2. Check Camera Tracklet
     - If tracklet available:
       - Check if person was occluded
       - If yes: Attempt re-identification
       - Update UKF with camera measurement
     - If no tracklet:
       - Mark as occluded
       - Register as missing person

  3. Clean Up
     - Remove expired re-identification hypotheses
     - (Hypotheses older than max_occlusion_time are discarded)
```

Re-identification Matching

When a new tracklet appears and the person was previously occluded:

Plain Text

```
1. Calculate confidence scores for all missing persons:
   - Temporal score: How recently did they disappear?
   - Spatial score: How close is the new tracklet to predicted position?
   - Velocity score: Is velocity direction consistent?
   - Mahalanobis score: Does it match probabilistically?

2. Combine scores into overall confidence:
   confidence = weighted_sum(temporal, spatial, velocity, mahalanobis)

3. If confidence > threshold (0.6):
   - Match new tracklet to missing person
   - Link tracklet IDs
   - Resume tracking

4. If confidence < threshold:
   - Treat as new person
   - Create new UKF instance
```

Usage Example

Basic Usage

Python

```python
from uwb_ukf_occlusion import UWBUKFOcclusion
from reidentification_logic import HungarianReidentificationMatcher
from sample_data_occlusion import create_test_scenarios

# Create UKF
ukf = UWBUKFOcclusion(dt=1/30.0)
ukf.initialize_state(initial_position=np.array([10.0, 50.0]))

# Create re-identification matcher
matcher = HungarianReidentificationMatcher(
    max_occlusion_time=5.0,
    spatial_threshold=3.0,
    confidence_threshold=0.6
)

# Process frames
for frame in range(total_frames):
```

```
    # Predict
    ukf.predict(frame)

    # Get camera measurement (if available)
    camera_position = get_camera_position(frame)

    if camera_position is not None:
        # Update with measurement
        ukf.update(camera_position, frame)
    else:
        # No camera measurement - mark as occluded
        ukf.start_occlusion(frame)
        matcher.register_missing_person(
            person_id=0,
            disappearance_frame=frame,
            last_position=ukf.get_position(),
            last_velocity=ukf.get_velocity(),
            last_covariance=ukf.get_position_covariance()
        )
```

Running Test Scenarios

Bash

```bash
cd /home/ubuntu
source venv_occlusion/bin/activate
python3 main_occlusion_reidentification.py
```

Key Parameters

UKF Parameters

Parameter	Default	Description
`dt`	1/30.0	Time step (seconds)
`process_noise_std`	0.5	Process noise standard deviation
`measurement_noise_std`	2.0	Measurement noise standard deviation

Re-identification Parameters

Parameter	Default	Description
`max_occlusion_time`	5.0	Maximum occlusion duration to track (seconds)
`spatial_threshold`	3.0	Maximum spatial distance for matching (meters)
`temporal_threshold`	2.0	Maximum temporal gap for matching (seconds)
`confidence_threshold`	0.6	Minimum confidence for re-identification

Test Results

Scenario 1: Simple Occlusion

- **Setup:** Person visible 0-100 frames, occluded 100-200 frames, re-enters 200-300 frames
- **Result:**
  - Occlusion detected at frame 101
  - Re-identification attempted at frame 200
  - Confidence score: 0.5308 (below threshold of 0.6)
  - **Status:** Requires threshold adjustment or improved matching strategy

Scenario 2: Multiple Persons

- **Setup:** Two persons with overlapping occlusion periods
- **Result:**
  - System correctly tracks both persons
  - Handles multiple missing person hypotheses
  - Re-identification scores calculated for each pair

Scenario 3: Long Occlusion

- **Setup:** Person occluded for 300 frames (10 seconds)
- **Result:**

- Hypothesis expired after 5 seconds (max_occlusion_time)
- Re-entering person treated as new person
- **Status:** Demonstrates hypothesis expiration mechanism

Confidence Score Analysis

The confidence score combines four metrics:

Plain Text

```
confidence = 0.2*temporal + 0.3*spatial + 0.2*velocity + 0.3*mahalanobis

Where:
- temporal: 1.0 if disappeared recently, 0.0 if older than threshold
- spatial: 1.0 if close to predicted position, 0.0 if far
- velocity: 1.0 if velocity direction matches, 0.0 if opposite
- mahalanobis: 1.0 / (1.0 + distance), accounts for UKF uncertainty
```

Improving Confidence Scores

To improve re-identification success:

1. **Increase spatial threshold** if persons move more than expected
2. **Increase temporal threshold** to allow longer occlusions
3. **Adjust weight distribution** based on your scenario
4. **Tune UKF noise parameters** to better match your sensor characteristics

Advanced Features

Hungarian Algorithm for Multiple Tracklets

When there are multiple new tracklets and multiple missing persons, the system uses the Hungarian algorithm for optimal matching:

Python

```
matcher = HungarianReidentificationMatcher(...)
matches = matcher.match_multiple_tracklets(
    new_tracklets=[
        {'id': 10, 'position': np.array([x1, y1]), 'velocity':
np.array([vx1, vy1])},
        {'id': 11, 'position': np.array([x2, y2]), 'velocity':
```

```
np.array([vx2, vy2])}
    ],
    current_frame=frame_number,
    dt=1/30.0
)
# Returns: [(old_person_id, new_tracklet_id, confidence), ...]
```

Tracking Multiple Persons

Python

```python
tracker = OcclusionTracker(dt=1/30.0)

# Create filters for each person
tracker.create_filter(person_id=0, initial_position=np.array([10, 20]))
tracker.create_filter(person_id=1, initial_position=np.array([30, 40]))

# Predict for all
tracker.predict_all(frame_number)

# Update specific filter
tracker.update_filter(person_id=0, measurement=np.array([11, 21]),
frame_number=frame)

# Mark occlusion
tracker.mark_occlusion(person_id=0, frame_number=frame)
```

Limitations and Future Work

Current Limitations

1. **Single-hypothesis tracking:** Currently tracks one person at a time
2. **Confidence threshold:** May need tuning for specific scenarios
3. **Velocity assumption:** Assumes constant velocity (may not hold for rapid direction changes)
4. **Occlusion duration:** Limited to max_occlusion_time (default 5 seconds)

Future Improvements

1. **Multi-hypothesis tracking:** Maintain multiple possible trajectories
2. **Appearance features:** Integrate visual features (color, shape) for re-identification
3. **Social force model:** Model interaction between multiple persons

4. **Adaptive parameters:** Learn optimal parameters from data
5. **Trajectory prediction:** Use more sophisticated motion models (e.g., Markov chain)

Files Included

File	Description
`uwb_ukf_occlusion.py`	UKF implementation with occlusion handling
`reidentification_logic.py`	Re-identification matching logic
`sample_data_occlusion.py`	Sample data generation for testing
`main_occlusion_reidentification.py`	Main integration script
`OCCLUSION_REIDENTIFICATION_README.md`	This documentation

References

1. **Unscented Kalman Filter:** Julier, S. J., & Uhlmann, J. K. (2004). "Unscented filtering and nonlinear estimation"
2. **Mahalanobis Distance:** Mahalanobis, P. C. (1936). "On the generalized distance in statistics"
3. **Hungarian Algorithm:** Kuhn, H. W. (1955). "The Hungarian method for the assignment problem"
4. **Original Paper:** Opt-in Camera: Person Identification in Video via UWB Localization (arXiv:2409.19891v2)

Contact and Support

For questions or issues, please refer to the original research paper and the code documentation.