```
>>  %Forward Collision Warning Using Sensor Fusion
>> % Set up the display
[videoReader, videoDisplayHandle, bepPlotters, sensor] = helperCreateFCWDemoDisplay↙
('01_city_c2s_fcw_10s.mp4', 'SensorConfigurationData.mat');

% Read the recorded detections file
[visionObjects, radarObjects, inertialMeasurementUnit, laneReports, ...
    timeStep, numSteps] = readSensorRecordingsFile('01_city_c2s_fcw_10s_sensor.mat');

'readSensorRecordingsFile' is not found in the current folder or on the MATLAB path, but↙
exists in:
    C:\Users\chinm\Desktop\Sensor Fusion and Tracking\Forward Collision Warning Using↙
Sensor Fusion

Change the MATLAB current folder or add its folder to the MATLAB path.

>> addpath 'C:\Users\chinm\Desktop\Sensor Fusion and Tracking\Forward Collision Warning↙
Using Sensor Fusion'
>> laneWidth = 3.6; % meters
egoLane = struct('left', [0 0 laneWidth/2], 'right', [0 0 -laneWidth/2]);
>> time = 0;           % Time since the beginning of the recording
currentStep = 0;    % Current timestep
snapTime = 9.3;     % The time to capture a snapshot of the display
>> % Initialize the tracker
[tracker, positionSelector, velocitySelector] = setupTracker();

while currentStep < numSteps && ishghandle(videoDisplayHandle)
    % Update scenario counters
    currentStep = currentStep + 1;
    time = time + timeStep;

    % Process the sensor detections as objectDetection inputs to the tracker
    [detections, laneBoundaries, egoLane] = processDetections(...
        visionObjects(currentStep), radarObjects(currentStep), ...
        inertialMeasurementUnit(currentStep), laneReports(currentStep), ...
        egoLane, time);

    % Using the list of objectDetections, return the tracks, updated to time
    confirmedTracks = updateTracks(tracker, detections, time);

    % Find the most important object and calculate the forward collision
    % warning
    mostImportantObject = findMostImportantObject(confirmedTracks, egoLane,↙
positionSelector, velocitySelector);

    % Update video and birds-eye plot displays
    frame = readFrame(videoReader);     % Read video frame
    helperUpdateFCWDemoDisplay(frame, videoDisplayHandle, bepPlotters, ...
        laneBoundaries, sensor, confirmedTracks, mostImportantObject, positionSelector,↙
...
        velocitySelector, visionObjects(currentStep), radarObjects(currentStep));
```

```
    % Capture a snapshot
    if time >= snapTime && time < snapTime + timeStep
        snapnow;
    end
end
Unrecognized function or variable 'setupTracker'.

>> % Initialize the tracker
[tracker, positionSelector, velocitySelector] = setupTracker();

while currentStep < numSteps && ishghandle(videoDisplayHandle)
    % Update scenario counters
    currentStep = currentStep + 1;
    time = time + timeStep;

    % Process the sensor detections as objectDetection inputs to the tracker
    [detections, laneBoundaries, egoLane] = processDetections(...
        visionObjects(currentStep), radarObjects(currentStep), ...
        inertialMeasurementUnit(currentStep), laneReports(currentStep), ...
        egoLane, time);

    % Using the list of objectDetections, return the tracks, updated to time
    confirmedTracks = updateTracks(tracker, detections, time);

    % Find the most important object and calculate the forward collision
    % warning
    mostImportantObject = findMostImportantObject(confirmedTracks, egoLane,↙
positionSelector, velocitySelector);

    % Update video and birds-eye plot displays
    frame = readFrame(videoReader);     % Read video frame
    helperUpdateFCWDemoDisplay(frame, videoDisplayHandle, bepPlotters, ...
        laneBoundaries, sensor, confirmedTracks, mostImportantObject, positionSelector,↙
...
        velocitySelector, visionObjects(currentStep), radarObjects(currentStep));

    % Capture a snapshot
    if time >= snapTime && time < snapTime + timeStep
        snapnow;
    end
end
Unrecognized function or variable 'numSteps'.

>> % Set up the display
[videoReader, videoDisplayHandle, bepPlotters, sensor] = helperCreateFCWDemoDisplay↙
('01_city_c2s_fcw_10s.mp4', 'SensorConfigurationData.mat');

% Read the recorded detections file
[visionObjects, radarObjects, inertialMeasurementUnit, laneReports, ...
    timeStep, numSteps] = readSensorRecordingsFile('01_city_c2s_fcw_10s_sensor.mat');
```

```
% An initial ego lane is calculated. If the recorded lane information is
% invalid, define the lane boundaries as straight lines half a lane
% distance on each side of the car
laneWidth = 3.6; % meters
egoLane = struct('left', [0 0 laneWidth/2], 'right', [0 0 -laneWidth/2]);

% Prepare some time variables
time = 0;           % Time since the beginning of the recording
currentStep = 0;    % Current timestep
snapTime = 9.3;     % The time to capture a snapshot of the display

% Initialize the tracker
[tracker, positionSelector, velocitySelector] = setupTracker();

while currentStep < numSteps && ishghandle(videoDisplayHandle)
    % Update scenario counters
    currentStep = currentStep + 1;
    time = time + timeStep;

    % Process the sensor detections as objectDetection inputs to the tracker
    [detections, laneBoundaries, egoLane] = processDetections(...
        visionObjects(currentStep), radarObjects(currentStep), ...
        inertialMeasurementUnit(currentStep), laneReports(currentStep), ...
        egoLane, time);

    % Using the list of objectDetections, return the tracks, updated to time
    confirmedTracks = updateTracks(tracker, detections, time);

    % Find the most important object and calculate the forward collision
    % warning
    mostImportantObject = findMostImportantObject(confirmedTracks, egoLane,↙
positionSelector, velocitySelector);

    % Update video and birds-eye plot displays
    frame = readFrame(videoReader);     % Read video frame
    helperUpdateFCWDemoDisplay(frame, videoDisplayHandle, bepPlotters, ...
        laneBoundaries, sensor, confirmedTracks, mostImportantObject, positionSelector,↙
...
        velocitySelector, visionObjects(currentStep), radarObjects(currentStep));

    % Capture a snapshot
    if time >= snapTime && time < snapTime + timeStep
        snapnow;
    end
end
>>
```