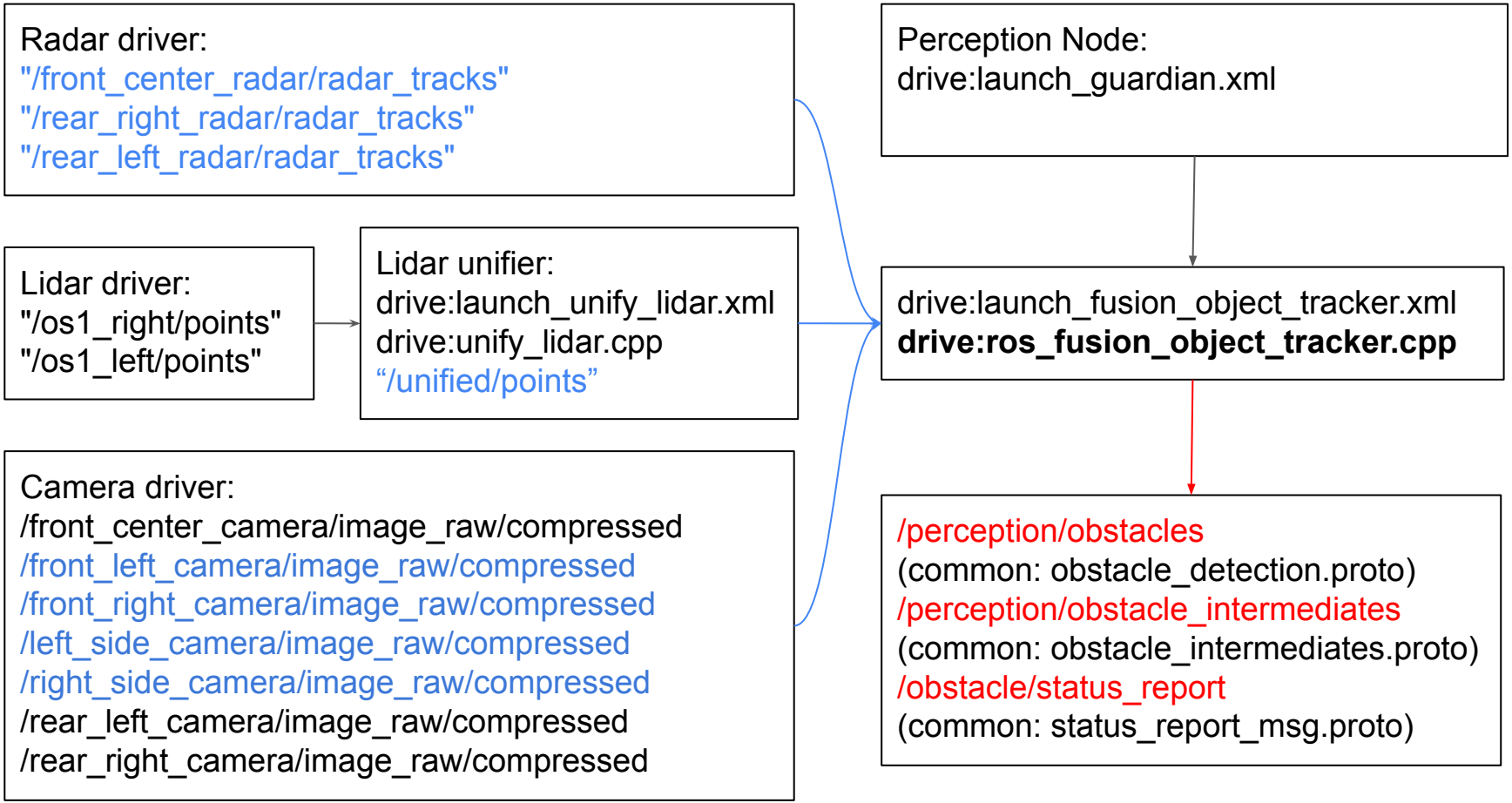
An Overview of Object Tracking and Sensor Fusion

Zack


Radar driver:
"/front_center_radar/radar_tracks"
"/rear_right_radar/radar_tracks"
"/rear_left_radar/radar_tracks"
Lidar driver:
"/os1_right/points"
"/os1_left/points"
Lidar unifier:
drive:launch_unify_lidar.xml
drive:unify_lidar.cpp
“/unified/points”
Camera driver:
/front_center_camera/image_raw/compressed
/front_left_camera/image_raw/compressed
/front_right_camera/image_raw/compressed
/left_side_camera/image_raw/compressed
/right_side_camera/image_raw/compressed
/rear_left_camera/image_raw/compressed
/rear_right_camera/image_raw/compressed
Perception Node:
drive:launch_guardian.xml
drive:launch_fusion_object_tracker.xml
drive:ros_fusion_object_tracker.cpp
/perception/obstacles
(common: obstacle_detection.proto)
/perception/obstacle_intermediates
(common: obstacle_intermediates.proto)
/obstacle/status_report
(common: status_report_msg.proto)

ros_fusion_object_tracker.cpp

- Initialize PerceptionPublisher, StatusReporter, PubSubSpinner, **ROSAsyncFrameProvider, FusionTrackerRuntimeInterface**
- Start fusion tracker thread and start PubSubSpinner

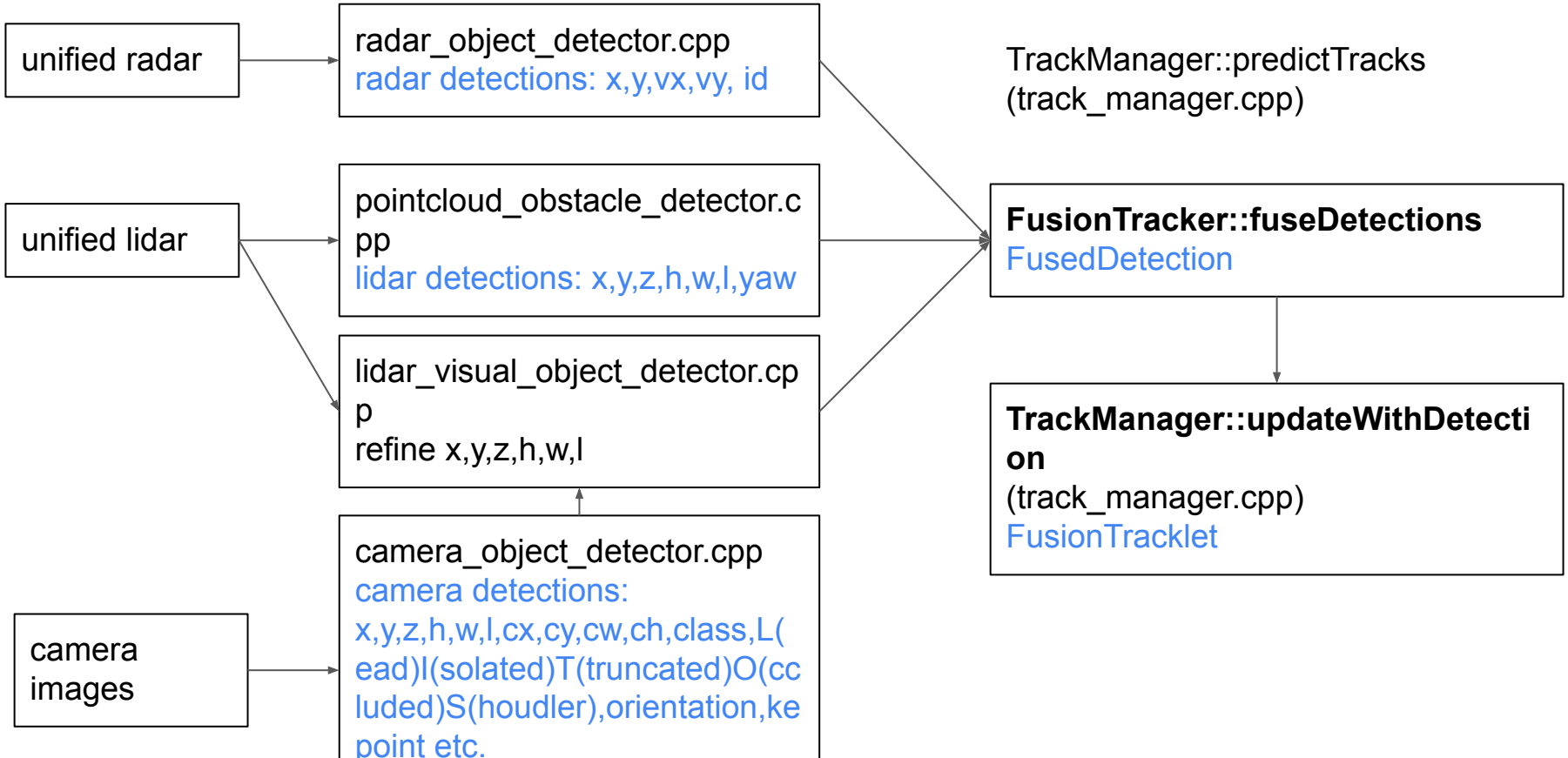
ROSAsyncFrameProvider [ros_async_frame_provider.cpp]

- Subscribe topics
- Check topic message integrity e.g. timestamp, latency
- Unify radar messages [unify_radar.cpp]
- Feed valid messages to **FusionTrackerRuntimeInterface::**processFusedSensorMessages (and LaneDetectorRuntimeInterface, TrafficLightDetectorRuntimeInterface) at a fixed rate in a separate thread
- Publish topics

FusionTrackerRuntimeInterface [fusion_tracker_runtime_interface.cpp]

- Initialize calibration, **FusionTracker**, ODD Checker (fusion quality, rain detection etc.)
- **FusionTracker::step** , process messages and return detected obstacles, occupancy grid
- Create obstacle segments for publishing

FusionTracker::step [fusion_tracker.cpp]



FusionTracker::fuseDetections

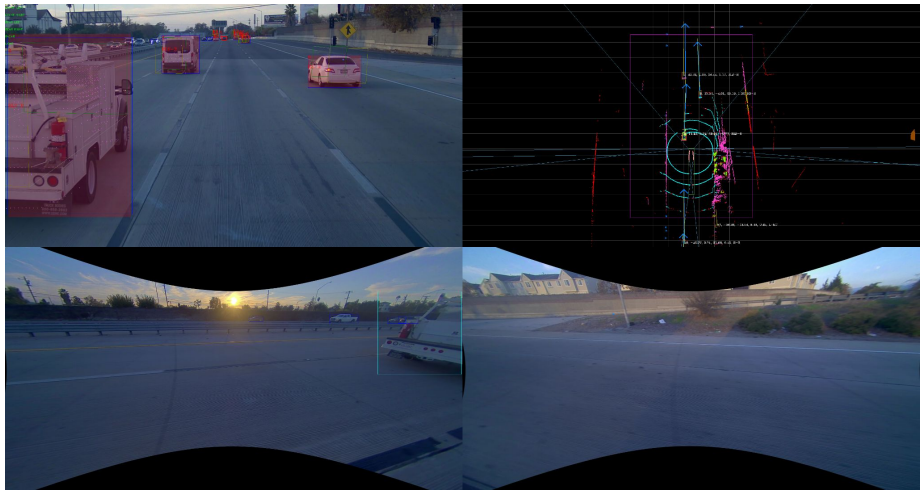
1. Form camera only FusedDetections -> (S/M)
2. associateLidarToDetections -> (S/M, LS/LM, L)
    - Metrics: dist, iou2d
3. associateRadarsToDetections/associateRadarToDetections -> (R, L, S/M, RL, RS/RM, LS/LM, RLS/RLM)
    - Metrics: dist, iou2d, speed, lateral, class, dimension, isOverpassFP, isCrossTraffic, isRadarFPByOccupancyGrid

TrackManager::updateWithDetection

- associateDetectionToTracks
  - computeAssociationMetric: motion, dimension, iouBEV, iou2d, type, class, dist, lateral dist, longitudinal dist, etc.
  - FusionTracklet::correct [fusion_tracklet.cpp]: update the old tracklet states using kalman filter [default_motion_model.cpp, unicycle_motion_model.cpp]
  - filter FPs
  - coast old tracks
  - generate new tracks

Debugging with unified simulator

- ros_unified_perception_simulator.cpp
  - Running fusion tracker (and optionally lane tracker) in a single process. which is also more deterministic and gives users more visualization options
- A little more user friendly python wrapper of running ros_unified_perception_simulator.cpp: run_unified_simulator.py
- example output:



Debugging with runtime pipeline

- involves replaying a bag and running the same node as on the vehicles. e.g.

```
rm -rf /dev/shm/plusai*
CODE=~/drive
BUILD=relwithdebinfo
VEHICLE=paccar-amazonph4
BAG_NAME=20211114T173733_paccar-amazonph4-k005sm_0_5to19.db
BAG_PATH=$HOME/rosbag_files/$BAG_NAME
OUTPUT_PATH=~/replay_results/${BAG_NAME}
RATE=1.0
export VEHICLE_NAME=paccar-amazonph4-p0005sc
export BAG_FILE=$BAG_PATH
export GLOG_log_dir=$OUTPUT_PATH/log
rm -r $OUTPUT_PATH || echo "failed to remove output path"
mkdir -p $OUTPUT_PATH
mkdir -p $OUTPUT_PATH/log
cd $CODE/opt/$BUILD
rosparam set use_sim_time true
 cd $CODE/opt/$BUILD
./run/perception/ros_launch.sh -d run/perception/launch_guardian_${VEHICLE}.xml &> $GLOG_log_dir/perception.log &
./run/perception/ros_launch.sh -d run/plusview/launch_plusview_${VEHICLE}.xml \
     benchmark_mode:=true drive_root:=$CODE/opt/$BUILD instdir:=$CODE/opt/$BUILD \
     benchmark_output_path:=$OUTPUT_PATH headless_mode:=false &> $GLOG_log_dir/plusview.log &
# you can run other nodes as well e.g. planning, control, etc.
sleep 5
/opt/plusai/bin/fastbag play -e -i ${BAG_PATH} \
          -s /tf -d 2000 -q 100 -r $RATE --hz 1000 --clock --envelope-timestamp \
          -s /perception/lane_path -s /perception/obstacles -s /perception/traffic_lights \
          -s /plusai/control/cmd_report -s /threat_assessment/obstacles -s /vehicle/brake_cmd \
          -s /vehicle/misc_1_report -s /vehicle/steering_cmd -s /vehicle/throttle_cmd \
          -s /shm//perception/lane_path -s /shm//perception/obstacles -s /shm//perception/traffic_lights \
          -s /shm//plusai/control/cmd_report -s /shm//tf -s /shm//threat_assessment/obstacles \
          -s /shm//vehicle/brake_cmd -s /shm//vehicle/misc_1_report -s /shm//vehicle/steering_cmd \ ( and so on, skip topics as needed, depends on which nodes are you running)

kill -SIGINT $(pidof ros_lane_detector) || echo "failed to kill lane"
```

(Cont’d) Latency report

- Use following environment to control log levels and latency logging
  - export GLOG_v=3
  - export PLUSAI_LOGGER_LOG_VERBOSITY_LEVEL=3
  - export PLUSAI_TELEMETRY_MODES=ScopedTimerTelemetry:log_text,all:log_json
- Dump a readable aggregated latency by e.g.:
  /opt/plusai/share/drive_common/aggregate_latency.py --log-files $OUTPUT_PATH/log/ros_fusion_object_tracker/ros_fusion_object_tracker.[timestamp].log.structured &> $OUTPUT_PATH/log/ros_fusion_object_tracker/ros_fusion_object_tracker.[timestamp].log.latency