Vision Tracking and visual speed estimation

Hamed Kiani Galoogahi, Ph.D.

July 2022

Content

- Vision tracking: motivation & background
- Visual speed estimation
- Vision tracking design/implementation in perception
- Vision tracking use cases and applications
- Vision tracking improvements
- Q&A

Vision tracking: motivation & background

Tracking a (generic) object in 2D images, mainly vision ID and 2D bounding box (x, y, h, w)

- Advantages
  - Use of both visual (deep features) and spatial features (2D IoU)
  - Low computation
  - Make use of accurate image detections
- Challenges
  - Sensitive to photometric and geometric variation
  - Limited tracking states (mainly 2D bbox)
- Vision tracking for AV trucks/vehicles
  - Long range perception of cameras
  - Accurate object detection in images

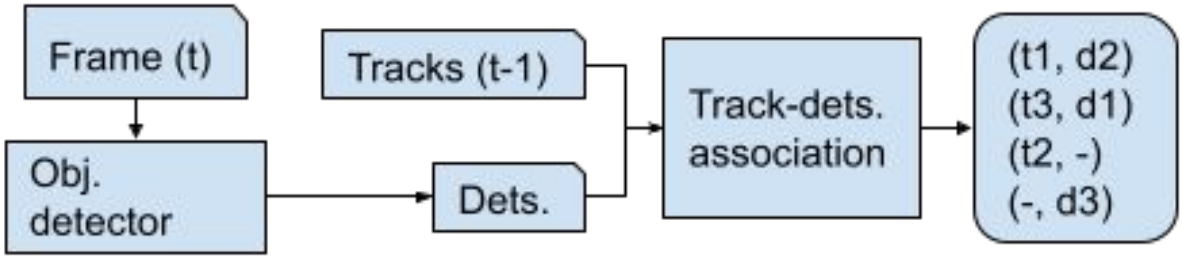
Vision tracking: motivation & background (cnt.)

Vision tracking approaches:

- Single object tracking (SOT)
  - **Tracking by detection**
  - Tracking without detection (e.g. correlation filters)
  - Deep learning based trackers
- Multi object tracking (MOT)
  - Traditional- learning based- techniques
  - Deep learning based techniques

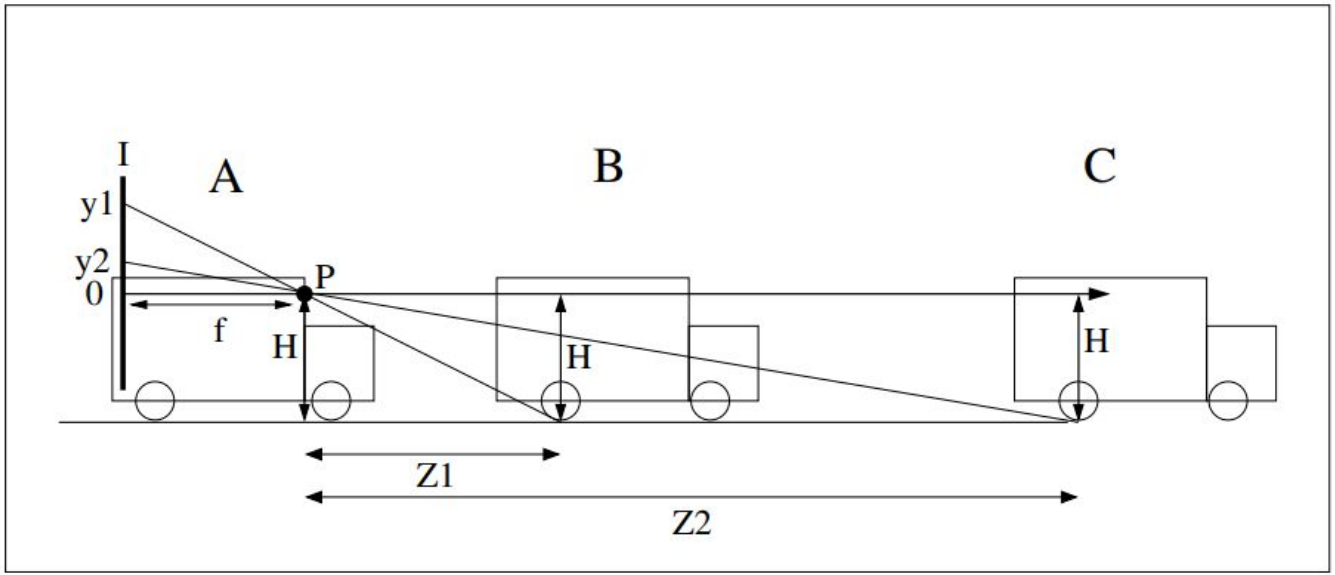
Vision tracking: motivation & background (cnt.)

Tracking by Detection



- Update t1 with detection d2
- Update t3 with d1
- Create new track for d3
- t2 is not tracked!

Visual speed estimation

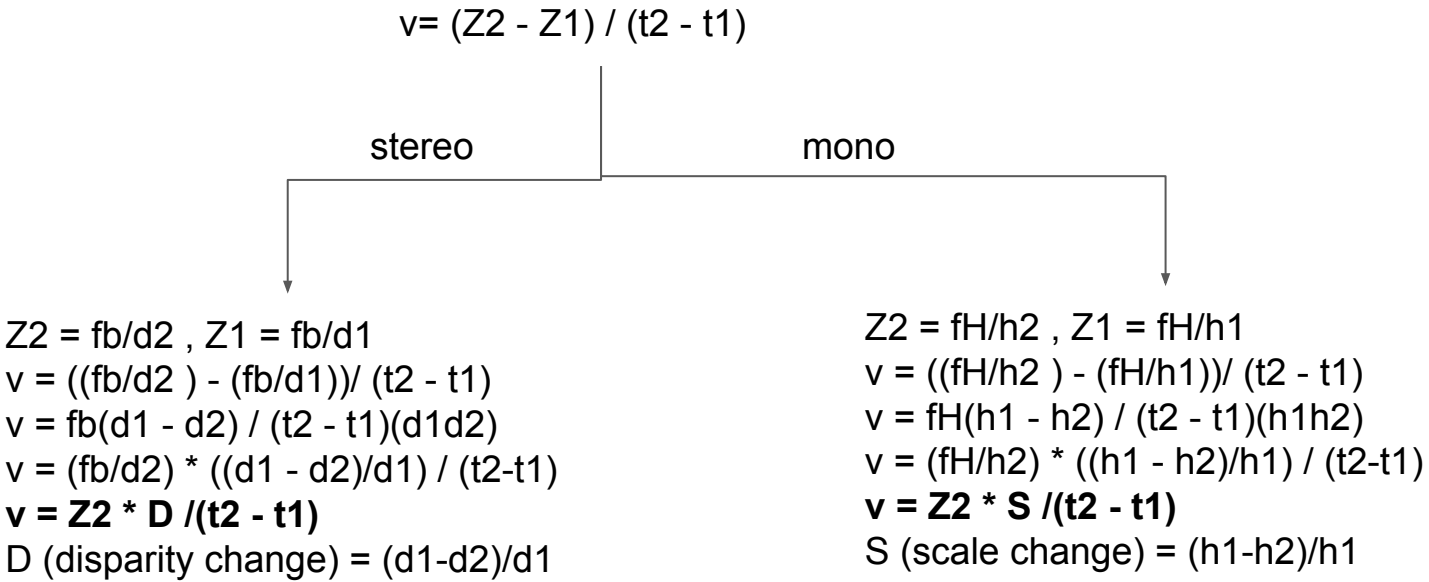


Z1/f = H/y1
Z2/f = H/y2

V = (Z2-Z1) / (t2-t1)

[Vision-based ACC with a Single Camera: Bounds on Range and Range Rate Accuracy]

Visual speed estimation (cnt.)



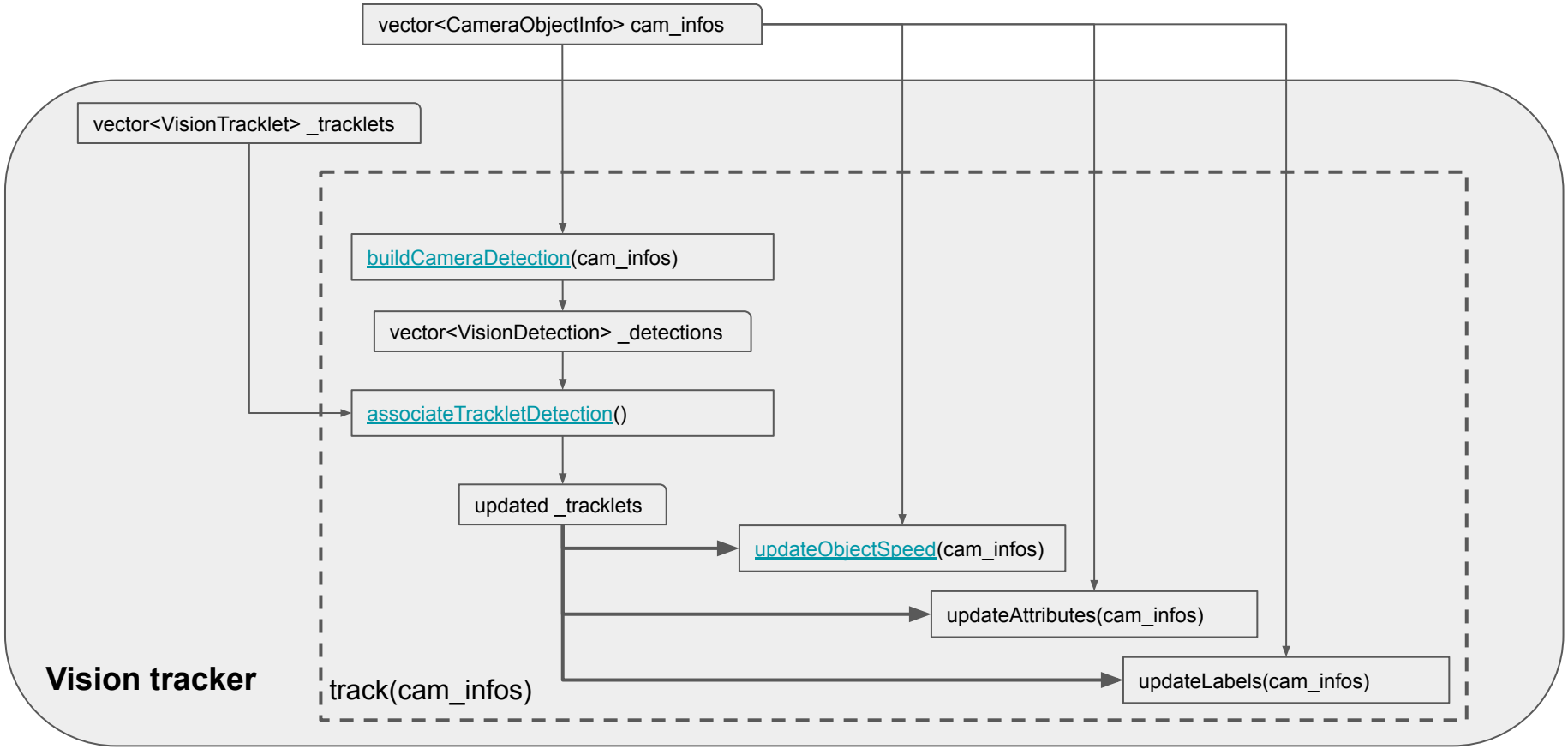
Vision tracking design/implementation

Design and implementation document

drive/perception/obstacle_detection/include/stereo/vision_tracker.h

drive/perception/obstacle_detection/src/stereo/vision_tracker.cpp

Vision tracking design/implementation (cnt.)



Vision tracking design/implementation (cnt.)

buildCameraDetection(cam_infos) → vector<VisionDetection> _detections

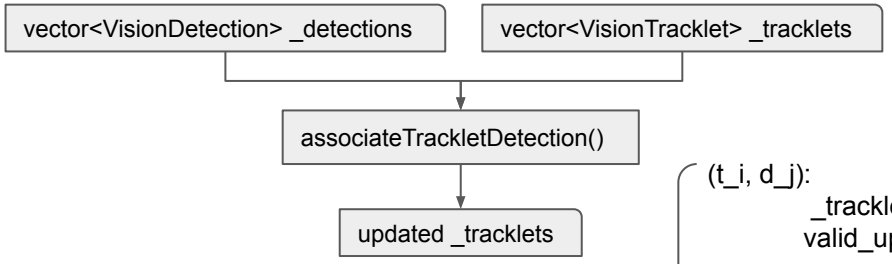
struct VisionDetection

```
+  cv::Rect2d box
+  float cx
+  float cy
+  float z
+  DepthMethod depth_method_used
+  int detection_id
+  double time_elapsed
+  bool position_valid
+  array<bool,kNumTrackedAttributes> has_attribute
```

back

Vision tracking design/implementation (cnt.)

back



	t1	t2	t3
d1	0.9	0.2	0.
d2	0.2	0.3	0.7
d3	0.1	0.2	0.1
d4	0.	0.	0.1

Assignment solver (e.g. Hungarian Algo.)

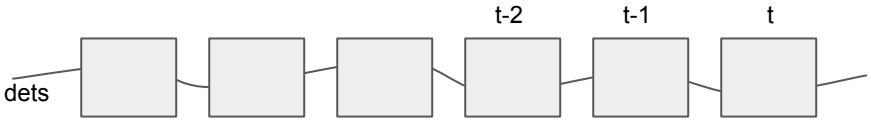
Sim. score: 2D IoU + conv feat.

```
(t_i, d_j):
        _tracklets[j].update(_detections[i],)
        valid_update = true
        age = age + 1

(t_i, ~):
        _tracklets[i].keep(),
        valid_update = false
        age = age - 1

(~, d_j):
        _cur_id++
        _new_tracklets.emplace_back(_detections[i],)

// Update tracklets, remove tracklets whose age <=0
    for (size_t i = 0; i < _tracklets.size(); ++i) {
        if (_tracklets[i].age > 0) {
            _new_tracklets.push_back(_tracklets[i])
        }
    }
    std::swap(_new_tracklets, _tracklets)
```

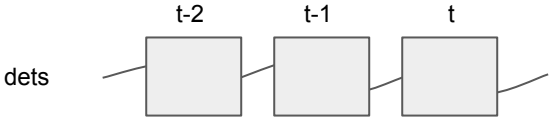
```
void keep() {
    // keep the latest tracklet's detection which was associated to a camera detection
    dets.push_back(dets.back());
    // set detection_id = -1, i.e. this tracklet is not updated,
    // so it is not used to update it's associated camera object's visual speed
    dets.back().detection_id = -1;
    // reduce the tracklet's age if it's not updated,
    // a tracklet whose age <=0 will be removed as dead tracklet
    age -= 1;
    valid_update = false;
}

void update(const VisionDetection& det,
            const VisionTrackerParam& param,
            const Eigen::VectorXf& conv_feat) {
    // if a tracklet is associated to a camera detection, update it!
    // an updated tracklet is used for visual speed estimation
    dets.push_back(det);
    conv_feature = conv_feat;
    // increaament the age of tracklet (number of frames tracklet is updated)
    age += 1;
    // truncate tracklet age
    // it helps to identify/remove tracklets which are updated for many frames but are not
    // updated for min_alive_frames times
    age = std::min(age, min_alive_frames);
    valid_update = true;
}

boost::circular_buffer<VisionDetection> dets;
Eigen::VectorXf conv_feature;
unsigned int id = 0;
// age indicates how many times the tracklet is updated
// age <= 0 results in removing tracklet as a dead/stale tracklet
int age = 1;
int min_alive_frames = 0;
size_t buffer_size = 1;
bool valid_update = false;
VisionSpeedResult vision_speed_result;
```

visionTracklet

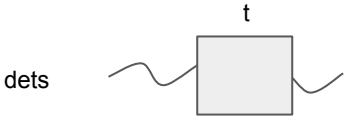
Vision tracking design/implementation (cnt.)



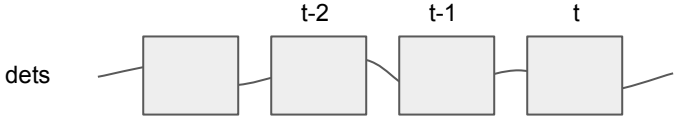
visionID = 1, age = 6, valid_update = T, vision_speed_result, conv_feat



visionID = 2, age = 3, valid_update = T, vision_speed_result, conv_feat



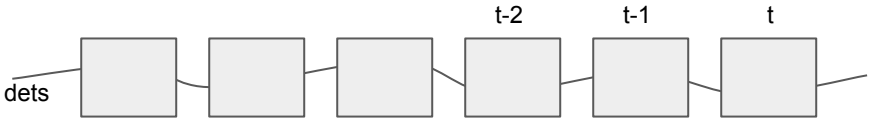
visionID = 11, age = 1, valid_update = T, vision_speed_result, conv_feat



visionID = 4, age = 2, valid_update = F, vision_speed_result, conv_feat

Vision tracking design/implementation (cnt.)

Vision speed estimation

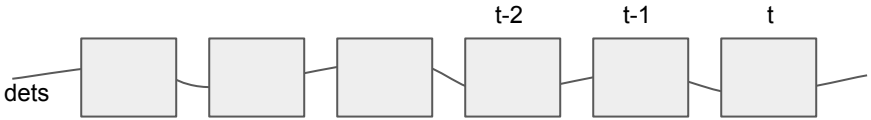


Each det has a timestamp and a distance (Z), so easy to compute v = delta_z/delta_t

```
enum class VisionSpeedStatus {
    UNTRACKED,
    UNINITIALIZED,
    JUMPED,
    INVALID,
    UPDATED,
};

You, 3 months ago | 1 author (You)
struct VisionSpeedResult {
    double speed = NAN;
    double speed_error = NAN;
    unsigned int vision_id = 0;
    VisionSpeedStatus speed_status = VisionSpeedStatus::UNINITIALIZED;
    void reset_vision_id() { vision_id = 0; }
    bool vision_id_is_valid() const { return vision_id > 0; }
};
```

Vision tracking design/implementation (cnt.)

Vision speed estimation



Each det has a timestamp and a distance (Z), so easy to compute v = delta_z/delta_t

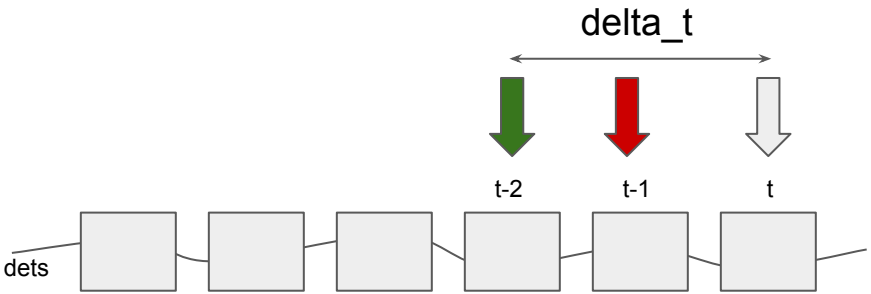
- Let’s be conservative on speed estimation
  - Position sanity check
  - Speed sanity check
  - Vision speed computation, v = delta_z/delta_t
  - Tracklet/detection speed update
  - tracklet/detection vision speed status transitions

```
enum class VisionSpeedStatus {
    UNTRACKED,
    UNINITIALIZED,
    JUMPED,
    INVALID,
    UPDATED,
};

You, 3 months ago | 1 author (You)
struct VisionSpeedResult {
    double speed = NAN;
    double speed_error = NAN;
    unsigned int vision_id = 0;
    VisionSpeedStatus speed_status = VisionSpeedStatus::UNINITIALIZED;
    void reset_vision_id() { vision_id = 0; }
    bool vision_id_is_valid() const { return vision_id > 0; }
};
```

Vision tracking design/implementation (cnt.)

Vision speed estimation (cnt.)

- Position sanity check
  - Find a *det* with position_valid in *dets* that validates position check: abs(z_t - z_i) < threshold?
    - Found? Go to speed sanity check
    - Not found? Change speed status to JUMPED if it was not UNINITIALIZED & set det at *t* : position_valid = false



threshold: delta_t * ego_speed + z_th_cnst

z_th_cnst computation?

Vision tracking design/implementation (cnt.)

Vision speed estimation (cnt.)

In code

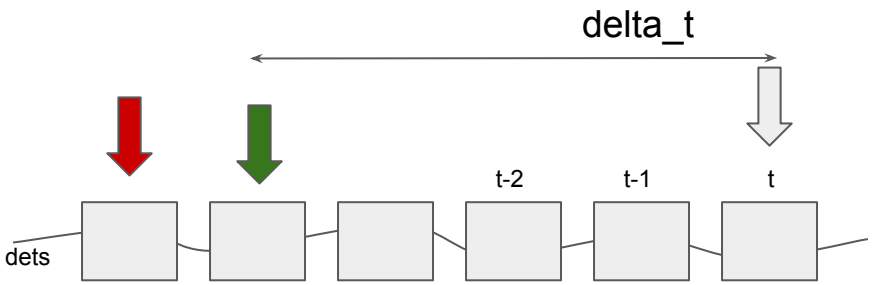
(z, h1)
(z + delta_z, h2)

z+delta_z/h2 = z/h1
delta_z = (z/h1)* h2 - z = z (h2/h1) - 1 = z (h2 - h1)/h1
= z (acceptable_pxl_error)/h1

For stereo:
 delta_z = z (acceptable_pxl_error)/d1
delta_z -> threshold

Vision tracking design/implementation (cnt.)

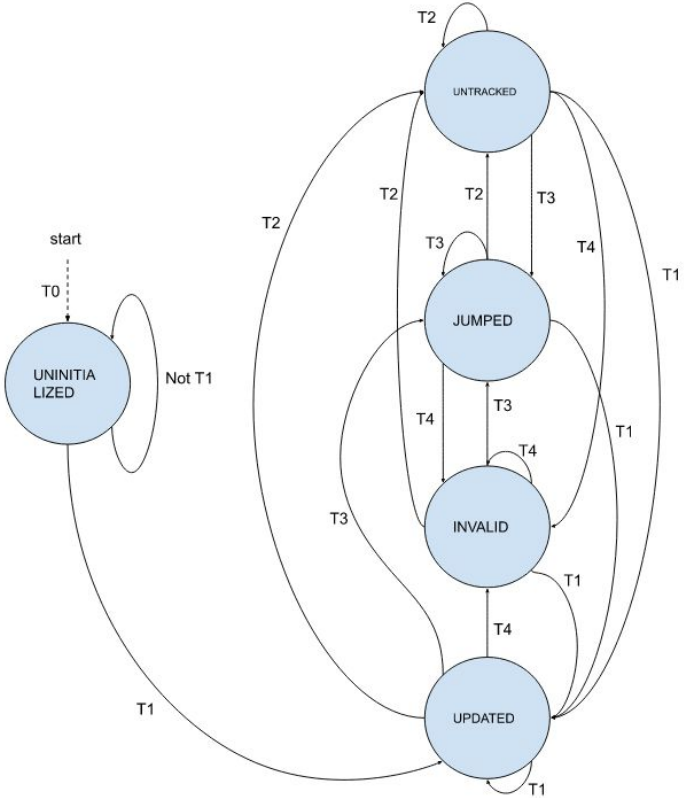
Vision speed estimation (cnt.)



- Speed sanity check
  - Find the maximum valid window for speed estimation
    - Find z_i that abs(z_t - z_i) < threshold
      - found? Got to speed check
      - Not found? Change speed status to INVALID if it was not UNINITIALIZED
  - Speed check based on Range Rate Error:
    - Pass speed check? Compute and Update tracklet vision speed && Change speed status to UPDATED (**v = Z2 * D /(t2 - t1) or v = Z2 * S /(t2 - t1))**
    - Not pass speed check? Change speed status to INVALID if it was not UNINITIALIZED

Vision tracking design/implementation (cnt.)

Vision speed status transitions



C0: tracklet.dets.size() > 1

C1: tracklet.valid_update

C2: speedIsValid(tracklet)

C3: positionIsValid(tracklet)

C4: windowIsValid(tracklet)

T0: create a new tracklet

T1: C0 & C1 & C2 & C3 & C4

T2: not C1

T3: not C3

T4: not C4 || not C2

```
enum class VisionSpeedStatus {
    UNTRACKED,
    UNINITIALIZED,
    JUMPED,
    INVALID,
    UPDATED,
};
```

Vision tracking use cases & applications

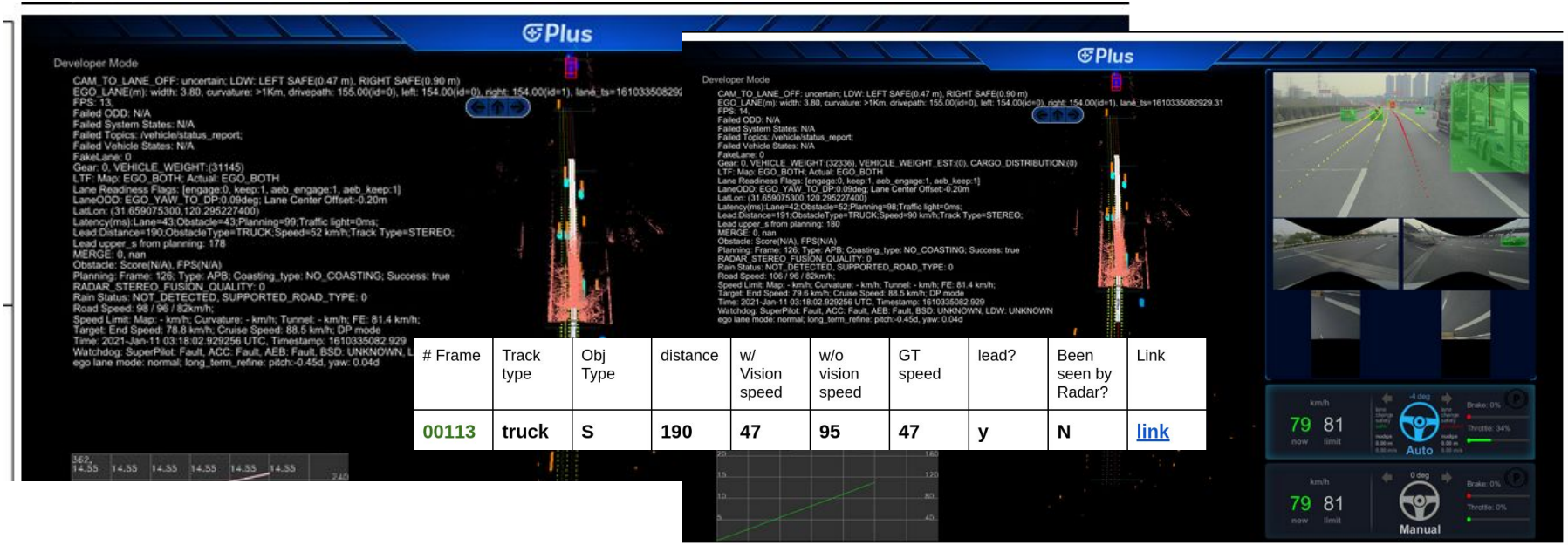
- Vision track ID
- Vision speed
- Vision tracklets

Vision tracking use cases & applications

- Vision speed when radar detection is not available for the last K timestamps (PR), report and analysis document
  - Enabled just for front cameras
  - Use vision speed if the vision speed status is UPDATED

Drawbacks:

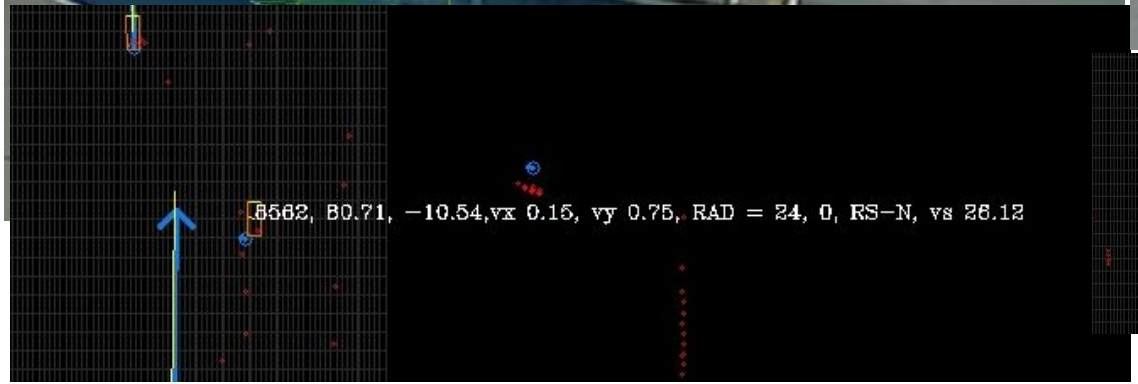
- Barely get vision speed UPDATED for far obstacles
- Pot. solutio: vision KF

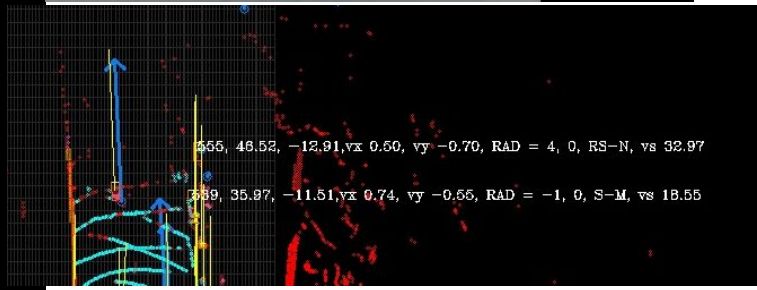


# Frame	Track type	Obj Type	distance	w/ Vision speed	w/o vision speed	GT speed	lead?	Been seen by Radar?	Link
00113	truck	S	190	47	95	47	y	N	link

Vision tracking use cases & applications

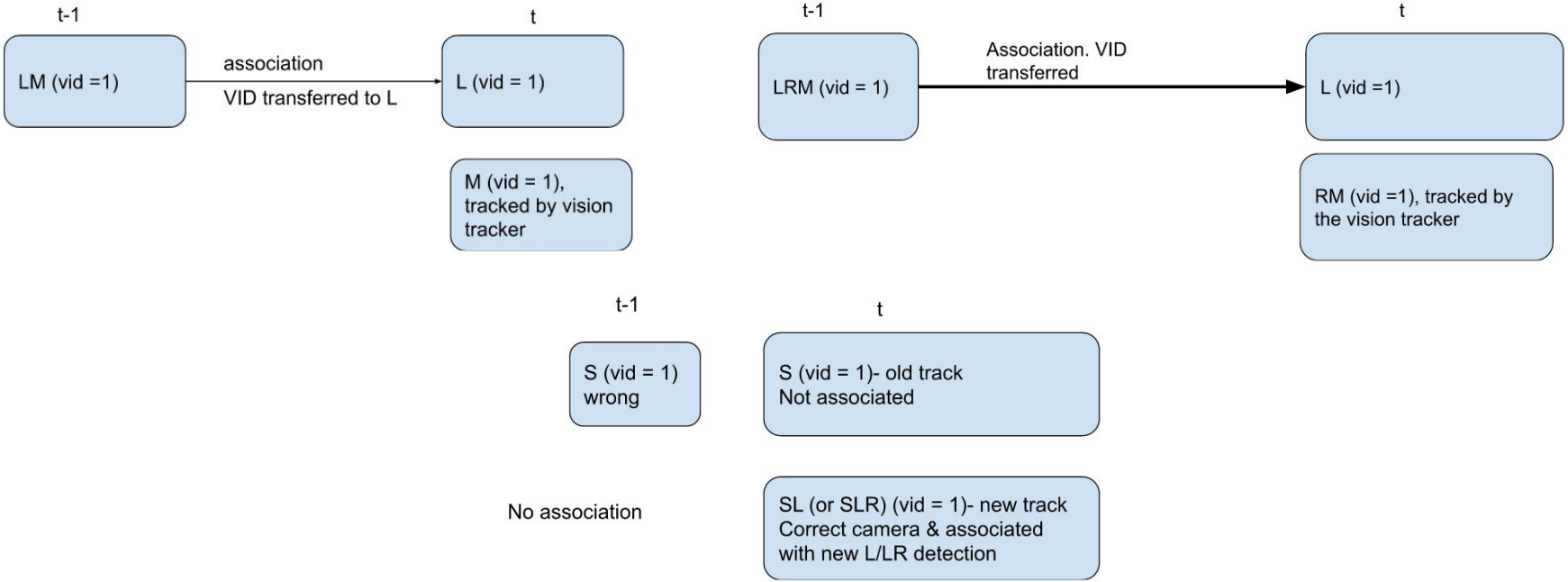
- Use vision speed to avoid associating static radar detection with dynamic fused detection with UPDATED vision speed, PR,





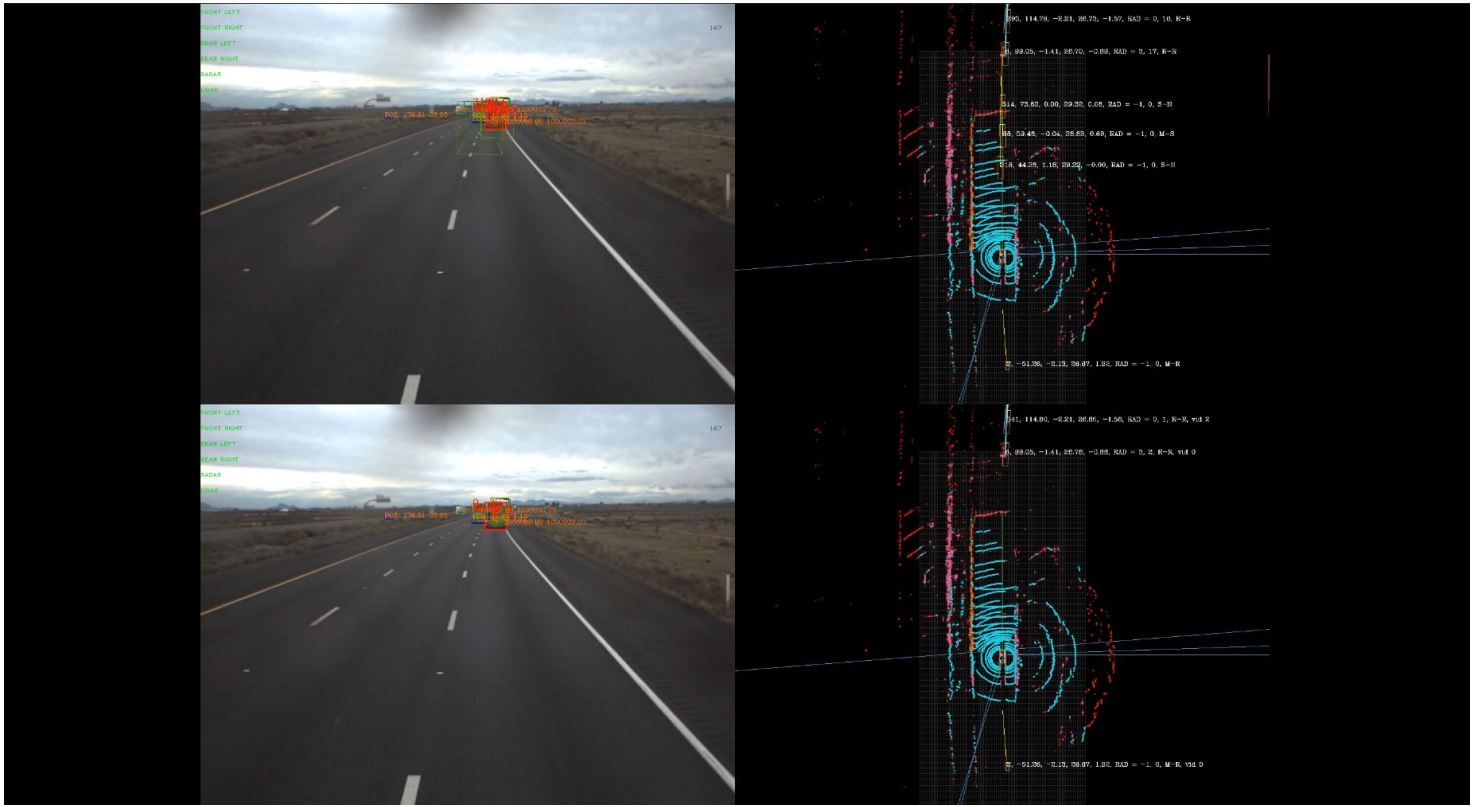
Vision tracking use cases & applications

- Use vision ID to merge different tracks of the same obstacles (caused by inaccurate camera 3D distance) PR, report and analysis document (please refer to Table 2 in this document for all cases improved by this idea)



Vision tracking use cases & applications

- Use vision ID to merge different tracks of the same obstacles (cnt.)



Vision tracking use cases & applications

- Use vision ID to merge different tracks of the same obstacles (cnt.)

Equation	Overall	close_front_ego_ego_lane x: (2, 50) , |y|: (0, 2)	close_front_neighbor_lane x: (2, 50) , |y|: (2, 6)	medium_front_ego_lane x: (50, 100) , |y|: (0, 2)	medium_front_neighbor_lan x: (50, 100) , |y|: (2, 6)
TP / (TP + FN)	0.367 -> 0.367	0.949 -> 0.949	0.996 -> 0.997	0.869 -> 0.865	0.938 -> 0.939
TP / (TP + FP)	0.556 -> 0.564	0.463 -> 0.561	0.924 -> 0.926	0.165 -> 0.316	0.866 -> 0.887

m_front_ego_lane , 100) , |y|: (0, 2)	medium_front_neighbor_lane x: (50, 100) , |y|: (2, 6)	far_front_ego_lane x: (100, 150) , |y|: (0, 2)	far_front_neighbor_lane x: (100, 150) , |y|: (2, 6)	very_far_front x: (150, 200) , |y|: (0, 6)	so_far_front x: (200, inf) , |y|: (0, 6)	side x: (-20, 2) , |y|: (0, 6)	rear x: (-inf, -20) , |y|: (0, 6)
0.941 -> 0.941	0.821 -> 0.821	0.987 -> 0.992	0.939 -> 0.939	0.884 -> 0.883	0.953 -> 0.953	0.821 -> 0.820	0.051 -> 0.051
0.769 -> 0.770	0.858 -> 0.865	0.603 -> 0.660	0.508 -> 0.520	0.548 -> 0.553	0.343 -> 0.338	0.430 -> 0.429	0.197 -> 0.198

Vision tracking use cases & applications

- Use vision ID to merge different tracks of the same obstacles: **follow-up** PR, details and analysis document

m_front_ego_lane , 100) , |y|: (0, 2)	medium_front_neighbor_lane x: (50, 100) , |y|: (2, 6)	far_front_ego_lane x: (100, 150) , |y|: (0, 2)	far_front_neighbor_lane x: (100, 150) , |y|: (2, 6)	very_far_front x: (150, 200) , |y|: (0, 6)	so_far_front x: (200, inf) , |y|: (0, 6)	side x: (-20, 2) , |y|: (0, 6)	rear x: (-inf, -20) , |y|: (0, 6)
0.865 -> 0.943		0.939 -> 0.949	0.990 -> 0.977	0.882 -> 0.898	0.623 -> 0.699	0.241 -> 0.368	1.000 -> 1.000	0.794 -> 0.796
0.316 -> 0.332		0.887 -> 0.881	0.568 -> 0.561	0.695 -> 0.698	0.604 -> 0.686	0.493 -> 0.570	0.817 -> 0.822	0.458 -> 0.459

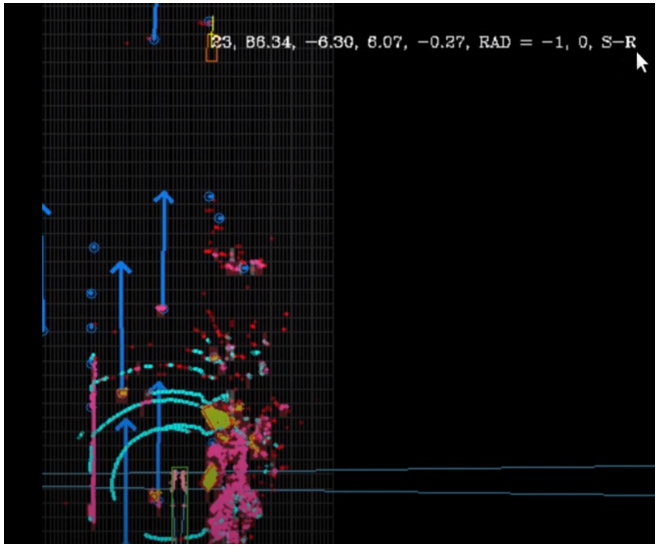
m_front_ego_lane , 100) , |y|: (0, 2)	medium_front_neighbor_lane x: (50, 100) , |y|: (2, 6)	far_front_ego_lane x: (100, 150) , |y|: (0, 2)	far_front_neighbor_lane x: (100, 150) , |y|: (2, 6)	very_far_front x: (150, 200) , |y|: (0, 6)	so_far_front x: (200, inf) , |y|: (0, 6)	side x: (-20, 2) , |y|: (0, 6)	rear x: (-inf, -20) , |y|: (0, 6)
0.827 -> 0.830		0.863 -> 0.865	0.862 -> 0.867	0.789 -> 0.795	0.729 -> 0.765	0.576 -> 0.661	0.963 -> 0.963	0.775 -> 0.776
0.971 -> 0.970		0.969 -> 0.967	0.811 -> 0.812	0.746 -> 0.751	0.217 -> 0.217	0.021 -> 0.024	0.885 -> 0.885	0.483 -> 0.483

Vision tracking use cases & applications

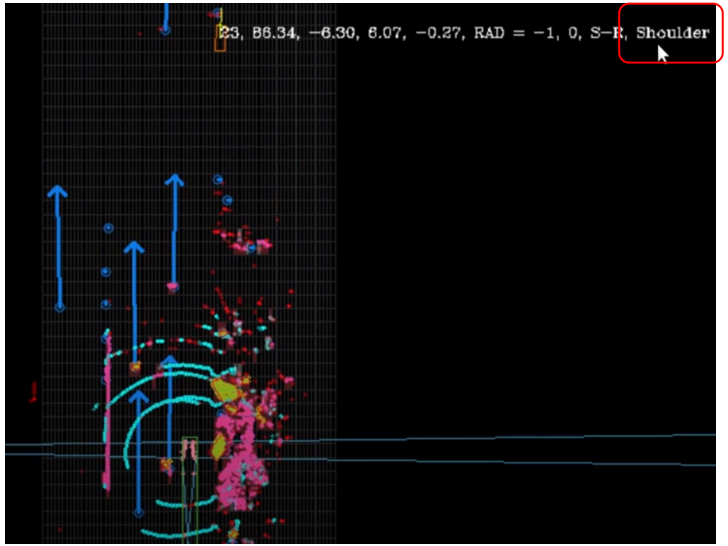
- Use vision ID to stabilize detection label (e.g. traffic sign detection), PR, a work done by Abhi
  Visually track static detection, and assign the major label over the last K timestamps to detected objects (e.g. traffic sign)

Vision tracking use cases & applications

- [PR #18963 from Yasen] Use vision tracker to track object dynamic attributes, such as leading/occluded/shoulder.
- Why? We observed inconsistency of shoulder vehicle detection, especially at longer range. In order to keep stable shoulder vehicle attribute, we take advantage of vision tracker to maintain shoulder attribute for more frames.



Before [Video]



After [Video]

Next on Vision tracker: long range tracking

- After 300m there is no radar, thus, not velocity/speed
  - Use vision speed
- Challenge: vision speed is highly dependent on 3D distance (z), object size, disparity
- Z is not stable/accurate for long range data (template matching showed very promising result on long range data)
- Better disparity, tighter bounding box

Vision tracker: drawbacks and potential solutions

- Many sanity checks (easy to fails) and heuristics
- jumpy /unstable vision speed with inaccurate distance, disparity/size
- Occlusion sensitive (feature similarity, object detector)
- Better distance (template matching), better size (more stable object detector), and more accurate disparity!

- Vision KF (developed at china side, will be ported/tested in US code soon)
- Using segmentation masks for occlusion
- Improving 3D distance estimation, currently working on stereo template matching approach

Q&A