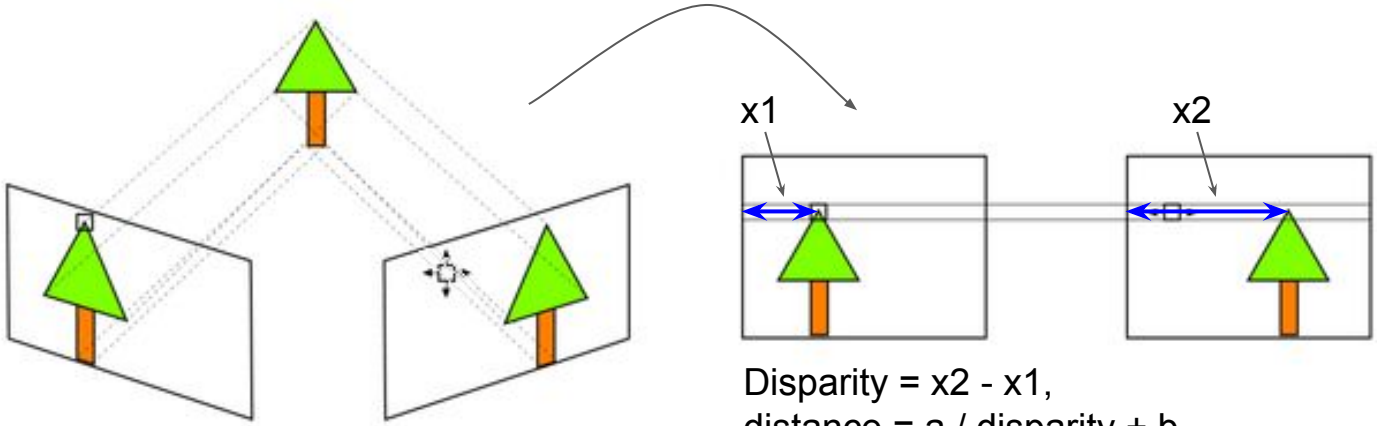
Stereo Template Matching

Mianwei

Stereo Vision

Stereo Rectification: reproject images such that the left and right image projections of any 3d point always lie on a horizontal line.



Disparity = x2 - x1,
distance = a / disparity + b
larger disparity means closer object

Computer 3d coordinate from image 2d coordinate + disparity.

$$[x, y, z, 1] = Homo(Q * [img\_x, img\_y, disparity, 1])$$

Homo: Homogenous operator. Homo([a,b,c,d]) = [a/d, b/d, c/d, 1]
Q: Given by stereo calibration.

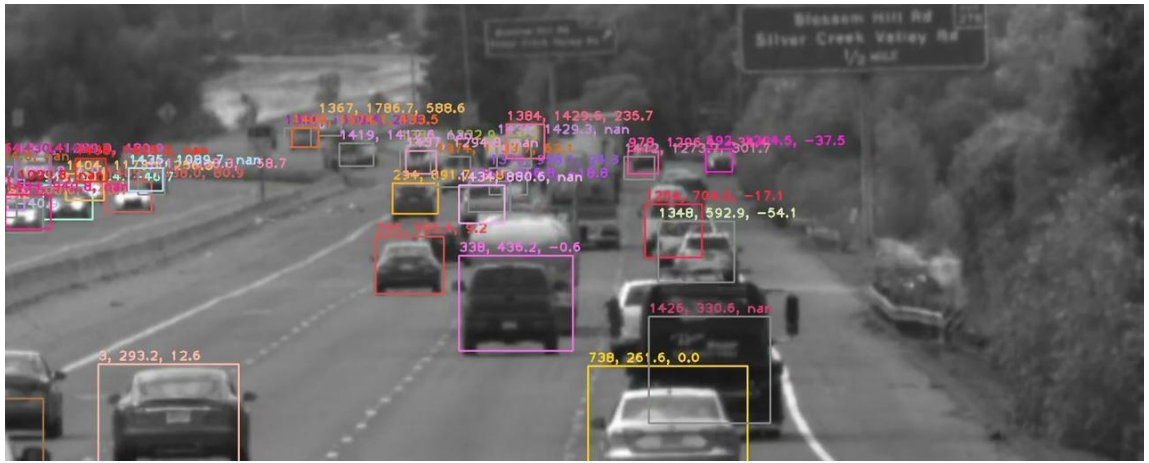
Why Stereo Vision?

1. Effective range:
    a. Lidar: ~120 meters
    b. Radar: ~ 200 meters.
    c. Stereo: > 400 meters if needed.

Long Range Stereo Tracking

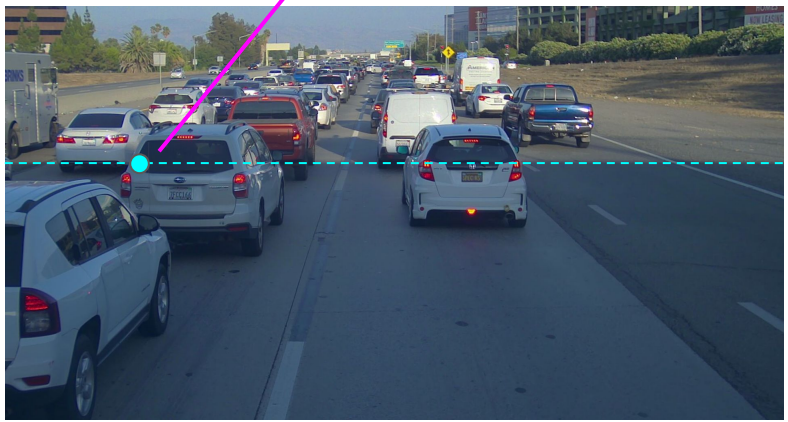
Original

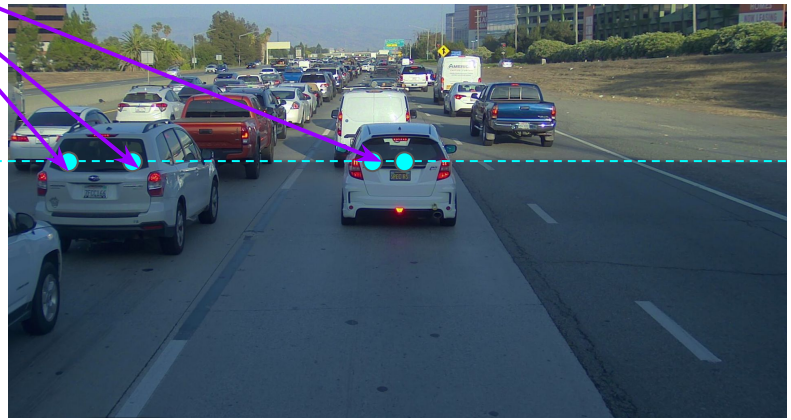
Zoom in



Challenge: Stereo Matching.

Left point, correspond to?

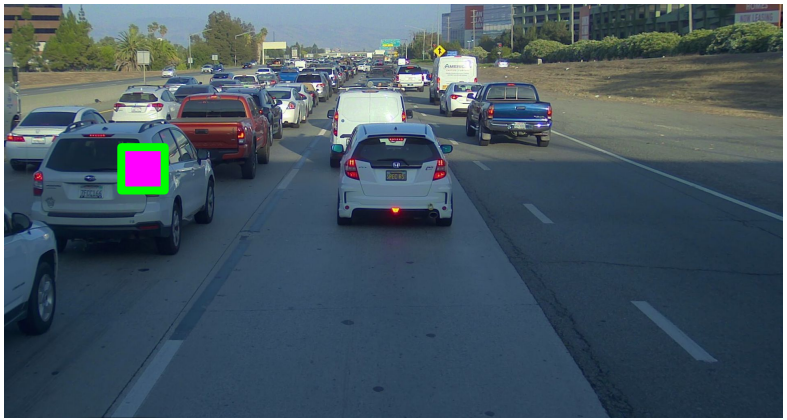




A lot of pixels have the same color on the same search line

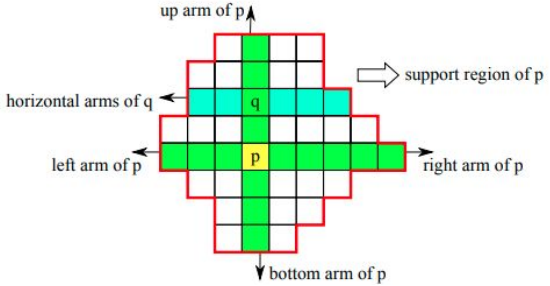
Key idea for stereo matching: support region.

- Single pixel matching is ambiguous, using multiple surrounding pixels on the same surface (i.e., support region) to match together will reduce ambiguity
- The **larger and more accurate** support region (i.e., containing pixels with the same disparity) will give **better disparity result**
- **Questions**: given a pixel, how to get its support region?

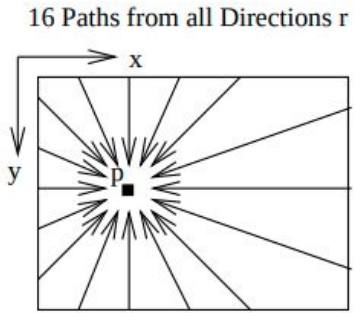


Different methods to define support region.

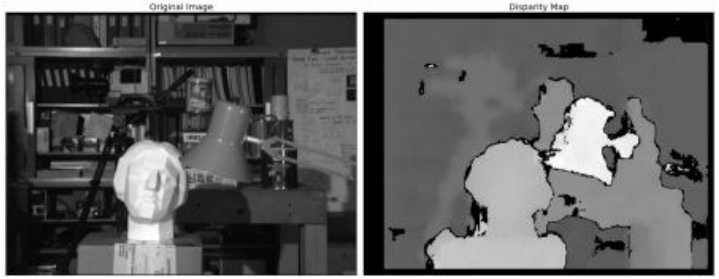
1. StereoBM (currently in our system): assume pixels in a m*m block all share the same disparity.
2. Cross-based local stereo matching: assume neighb pixels with similar color has similar disparities, grow the regions in a cross pattern.



3. SGBM (semi-global matching algorithm): optimize in 16 directions get smooth disparities.



StrereoBM (in our current system)



How we use stereo in our system for now?

1. Compute whole image disparity
2. For each bounding box, get the closest point as object distance.

Drawbacks of Stereobm

1. Slow, even it is already the simplest stereo algorithm
   a. On Xavier: >60ms for 1920X1020 image
   b. We only use half resolution now, which will lower accuracy of far-away object disparity.
2. The assumption of equal disparity in a block often fails (border of vehicles, small vehicles…)

Template Matching Idea.

1. If we only care about object disparity (which is the only usage in our system for now) , object detection bounding box gives perfect support region.

Pixels in each bounding box share similar disparity

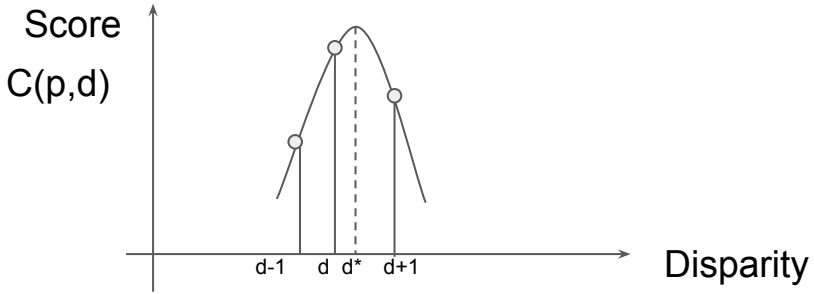
2. Computing disparity for objects only will greatly improve speed.

Basic case: Far Away Objects

- **Assume pixels in the same bounding box share the same disparity.**
- For each box, use pixels in left bounding box to search the right image within max_disparity range (e.g., 0 to 400 pixels)

Subpixel: More accurate distance estimation for far-away objects

1. For far away objects (>200m), 1 pixel of disparity difference represents >20m distance estimation.
2. Subpixel: go beyond the distance resolution caused by pixel discreteness.
   a. Fit a quadratic polynomial line to get the sub-pixel disparity.



$$d^* = d - \frac{C_2(\mathbf{p}, d_+) - C_2(\mathbf{p}, d_-)}{2(C_2(\mathbf{p}, d_+) + C_2(\mathbf{p}, d_-) - 2C_2(\mathbf{p}, d))}$$

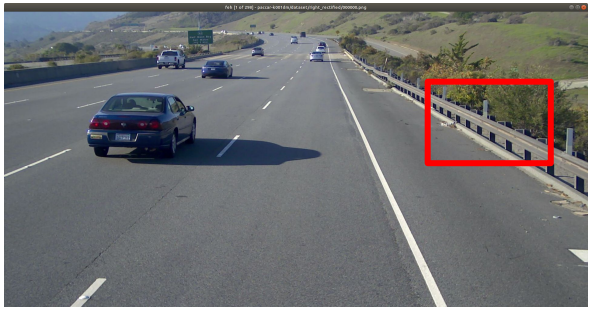
Check stereo result quality: Right to left validation

Left

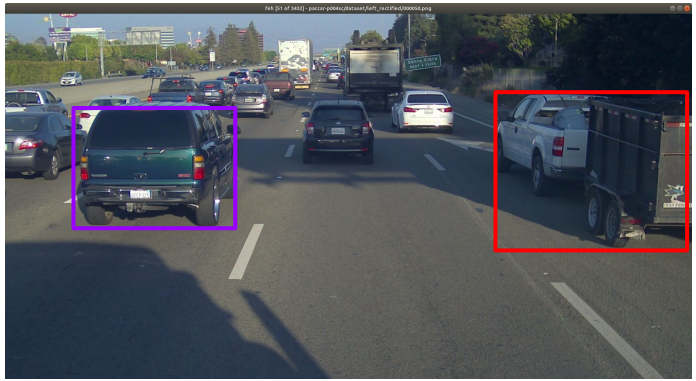
Not matching,
invalidate the result

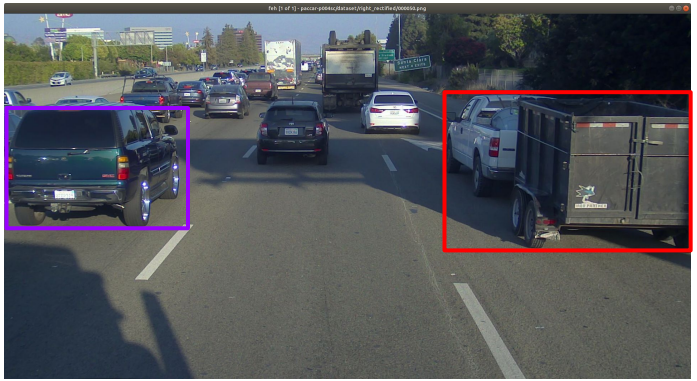
Left

Right



Challenge 1: Close Objects

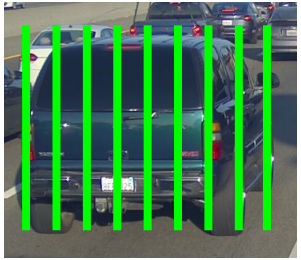




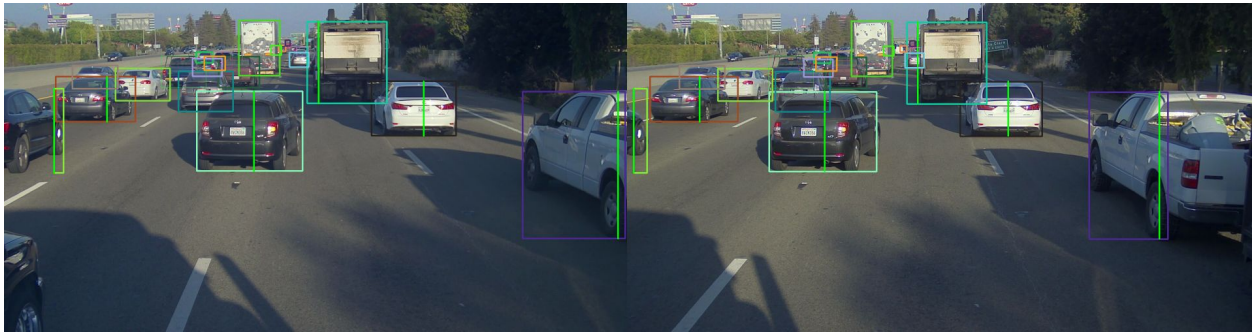
Pixels in a close object bounding box DOES NOT share the same disparity

My “Brilliant” ideas for Close Objects

1. Stripe Matching (assumption: each vertical stripe should share the same disparity)



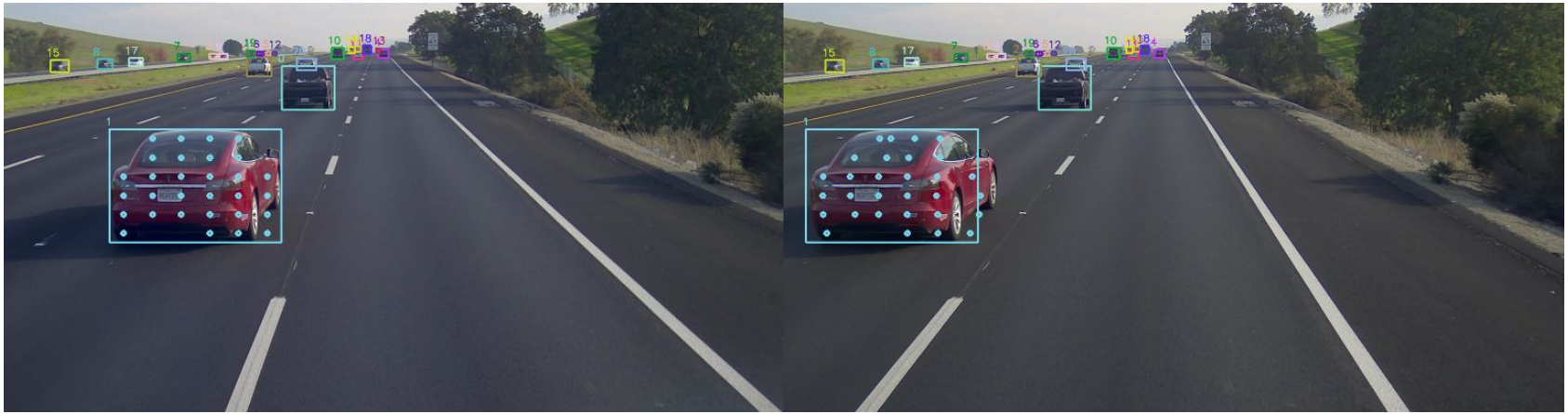
2. Cuboid Matching (assumption: vehicles could be viewed as a cuboid)



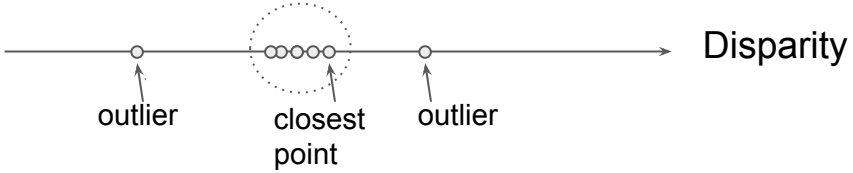
And then I see this case …

Current Solution for Close Objects.

- For close objects, similar to stereobm, compute disparities for small blocks on the object, and analyze the disparity distribution to get the closest point.

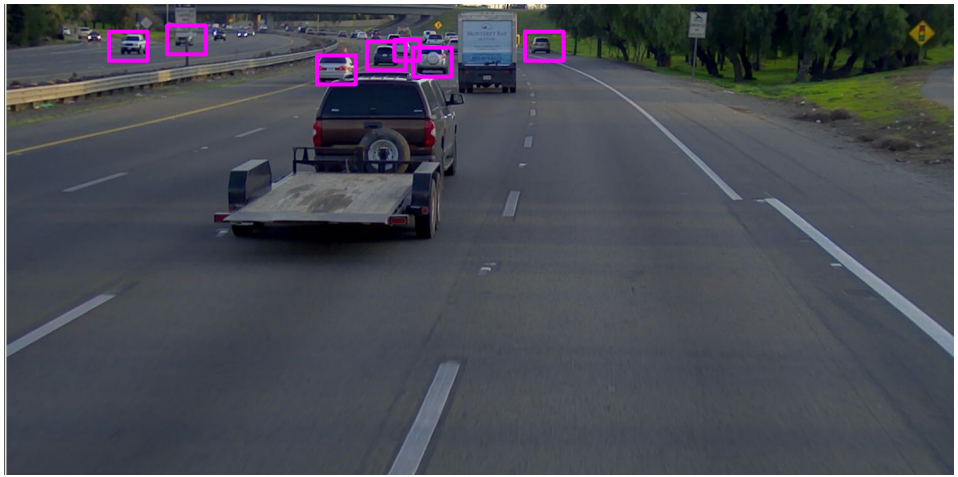


Disparity distribution in the same bounding box

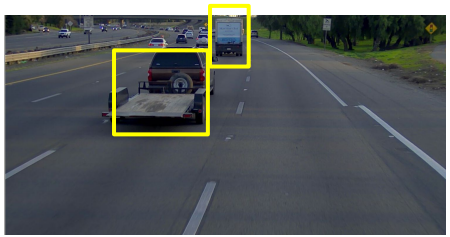


For efficiency: two scales template matching.

Full Resolution for Far Objects



Half Resolution for Close Objects



Challenge 2: Discrepancy between left and right images.

- Classical stereo matching way
  - Convert rgb stereo images to gray scale
  - SAD(sum of absolute difference): diff = abs(pixel_left - pixel_right), smaller difference represents higher matching score
- The discrepancy between stereo images might fail SAD
  - Camera difference.
  - Lightning differences.
  - Random noises in dark environment
  - Water drops / dirts on one camera

Census Transform: robust matching function.

- Census Transform [wiki]: a non-parametric transform that depends only on relative ordering of intensities, and not on the actual values of intensity, making it invariant with respect to monotonic variations of illumination, and it behaves well in presence of multimodal distributions of intensity

$$\xi(p, p') = \begin{cases} 0 & \text{if } p > p' \\ 1 & \text{if } p \leq p' \end{cases}.$$

124	74	32
124	64	18
157	116	84

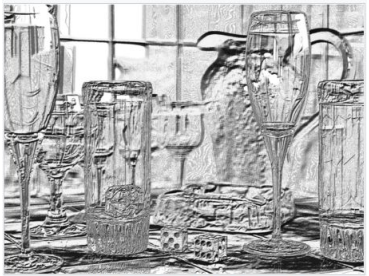
$\longrightarrow$

1	1	0
1	x	0
1	1	1

$\longrightarrow 11010111$

Example of census transform

A synthetic scene



Grayscale conversion followed by census transform

Stereo Matching based on Census Transform

Left Census Query Box

1011101101	0110001011
1010101001	0100001001

Right Census Image

1110101010	1010101101	1110101011	…
0010101011	0010001001	010100101	…
…	…	…	…

Diff_census = num_of_diff_bits(left_census, right_census)
For example, diff_census(1100, 0000) = 2

GPU coding tip 1: convert a gray image to a census image

```
__global__ void convert(const uint8_t* gray_images, uint64_t* census_images)
```

Implementation 1

```
in each thread
for x, y in gray_images
    census = 0
    for i in [-3, 3]
      for j in [-4, 4]
        d = gray_images[x, y] <
              gray_images[x + i, y + j]
        add d to census
    census_images[x, y] = census
```

Implementation 2

```
Cache a region in gray_image to
shared_memory shm_image
for x, y in shm_image
    census = 0
    for i in [-3, 3]
      for j in [-4, 4]
        d = shm_image[x, y] <
              shm_image[x + i, y + j]
        add d to census
    census_images[x, y] = census
```

2 is much more efficient than implementation 1 (like 0.8ms vs 4ms on my desktop)

GPU coding tip 2: Efficient match on the census image.

Left Census Query Box

1011101101	0110001011
1010101001	0100001001

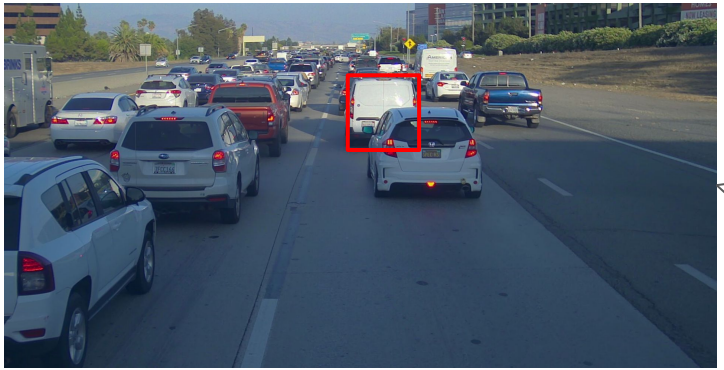
Right Census Image

1110101010	1010101101	1110101011	…
0010101011	0010001001	010100101	…
…	…	…	…

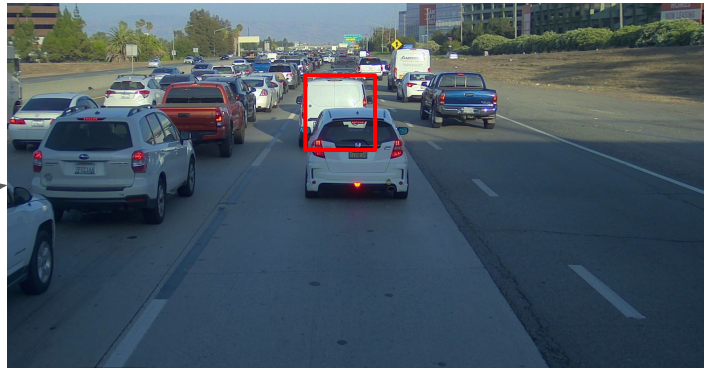
Cache census image in texture memory [doc] for efficient 2d query.

Challenge 3: Object occlusion.

Left

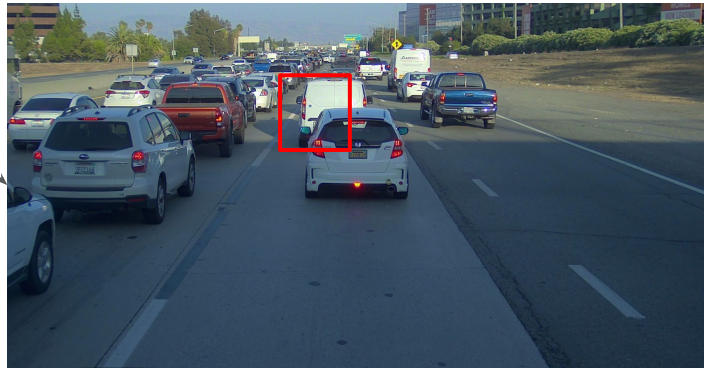


Right



Expected

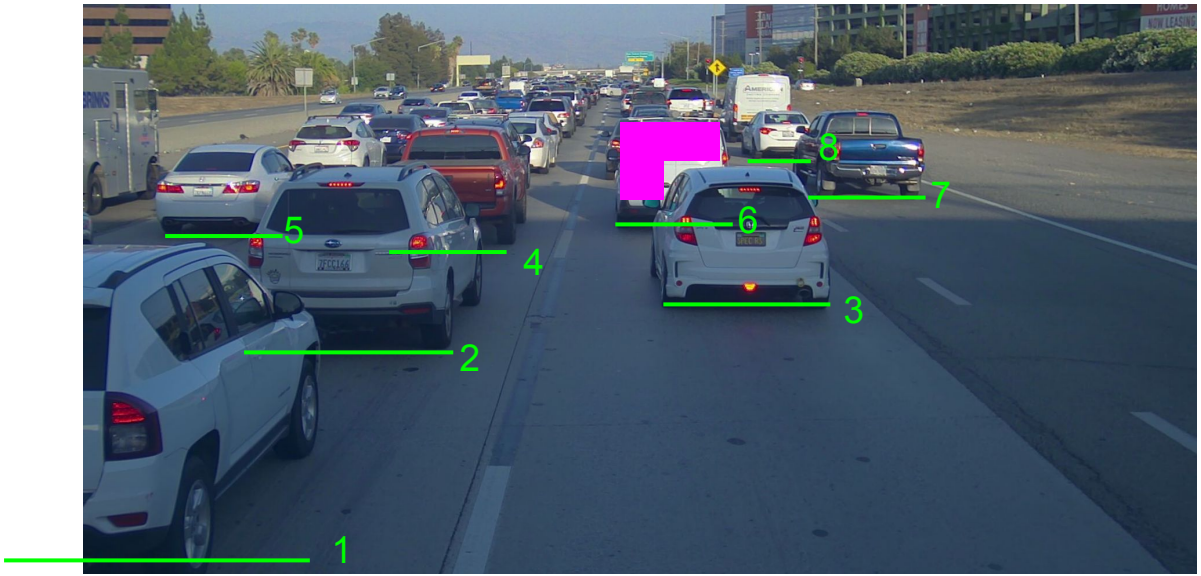
Mislead by closer vehicle



- If occlusion relations is known, we could exclude points that do not belong to target objects.
- **Chicken-egg problem: need distance to know occlusion relations?**

Borrow the idea in our current system

Use the bottom line of the bounding box to decide the occlusion relation

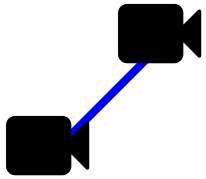


Challenge 4: Stereo Calibration Error

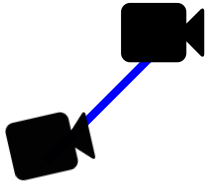
Error Type 1: Pitch Offset

Side view

Expected



Actual



Cause vertical offset between left and right and failed to match

Solution: add vertical search range

- Search +- 5 pixels in vertical direction along with disparity search
- very common to have 1~3 pixels offsets in our ci bags.

Challenge 4: Stereo Calibration Error

Error Type 2: Yaw Offset

Birdeye view

Expected

Actual

Cause over/under estimation of stereo distance

One idea I tried: Track static points for calibration

- Find out some feature points not covered by vehicles bounding boxes, assume that are static
- Use the same code to track feature points across frames and compute their disparities.
- If calibration error exists, those static points will move. Optimize the offset to make them not moving
- It works, but need more tuning. Cons
  - Not every scene has sufficient feature points.
  - Tracking might introduce new noises.

Current implementation: Use Radar to Calibrate.

- Project Stereo 3d point to radar frame.
- Search an offset such that radar points and stereo points match.

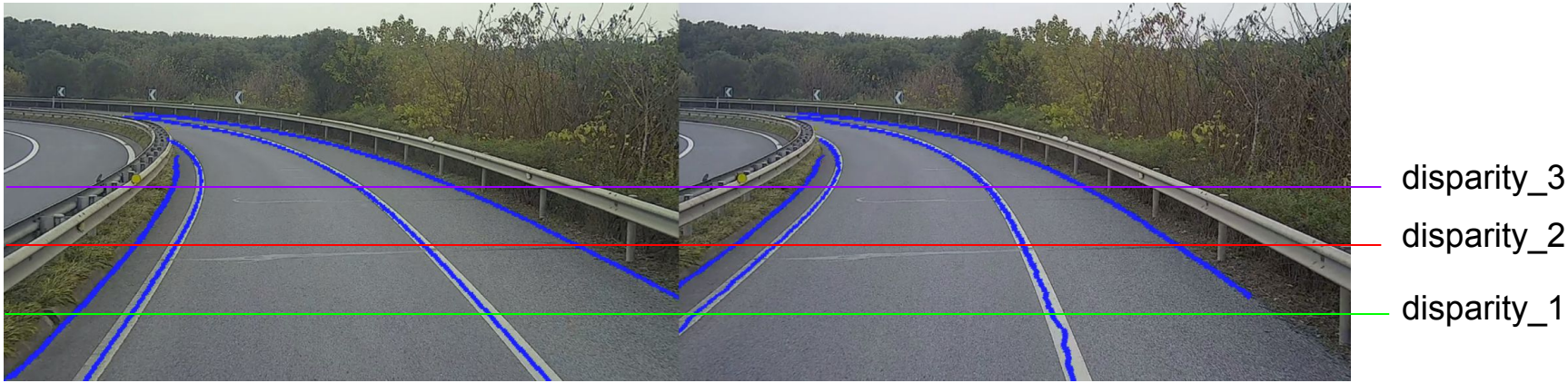
CI result

- http://jenkins/job/replay_report/1284486/paccar-k001dm_5fbatch3_20mot_20report/
- http://jenkins/job/replay_report/1284494/nike_5fbatch1_20mot_20report/

[WIP] Drivable Area Template Matching.

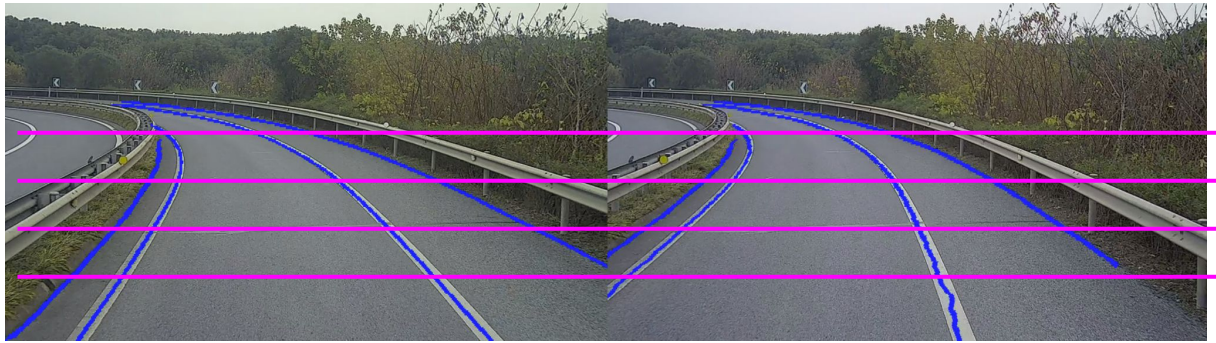
Can we estimate the 3d ground based on drivable area detection?

Nice property for 3d grounds.



1. For ground points on the same horizontal line, their disparities are the same (assume calibration error is not that crazy)
2. The line disparity is monotonic from image bottom to the top (i.e., disparity_1 > disparity_2 > disparity_3)

Nice property for 3d grounds.



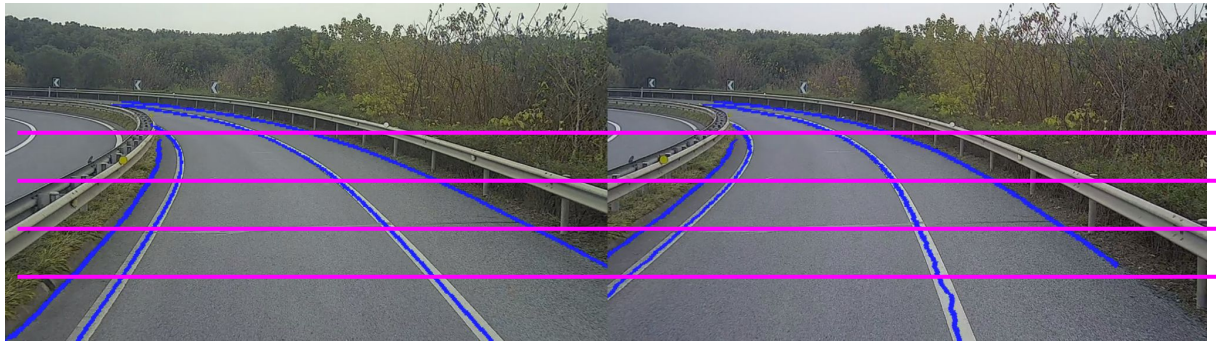
Disparity

10	44	2	2	31	76
9	10	34	21	6	6
6	4	43	10	29	30
10	20	21	50	20	19

Disparity Matching Score
X: disparity, y: img_y

Interview question: how to find a continuous path with maximal score, extending from upper left to bottom right? :)

Dynamic programming to estimate 3d ground.



Disparity

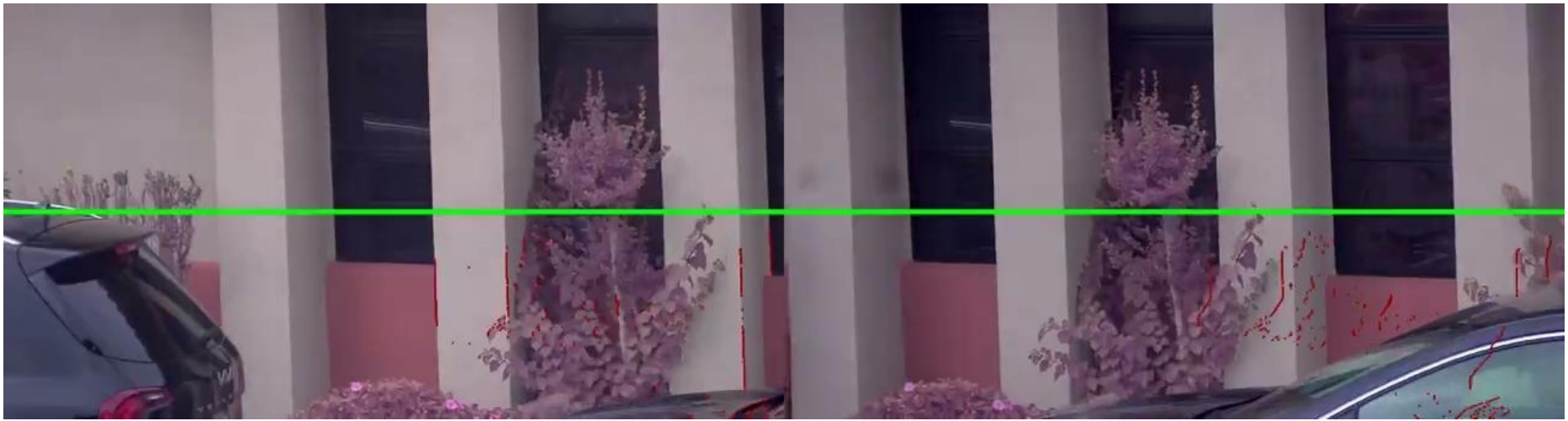
10	44	2	2	31	76
9	10	34	21	6	6
6	4	43	10	29	30
10	20	21	50	20	19

Disparity Matching Score
X: disparity, y: img_y

Interview question: how to find a continuous path with maximal score, extending from upper left to bottom right? :)

Answer: Dynamic Programming. Check doc for details

3d ground point demo.



1. Detect Feature points on the drivable area using Sobel Filter (lanemark / dirt / shadows …)
2. Match feature points to the right using dynamic programming.
3. NOTE: we don’t need lane detection here, lane detection is projected to the right image just for visual verification.

Q & A