Depth from Stereo

Depth from Two Views: Stereo

- All points on the projective line to **P** map to **p**

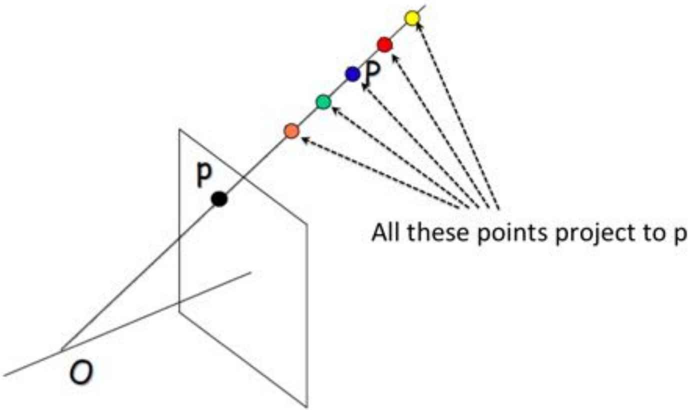
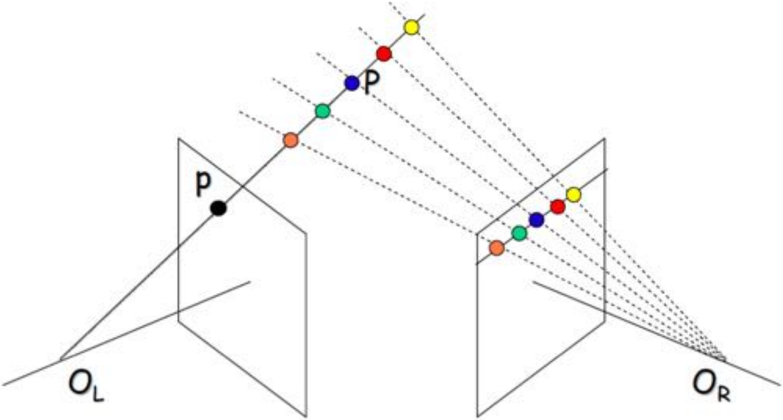


Figure: One camera

Depth from Two Views: Stereo

- All points on projective line to **P** in left camera map to a **line** in the image plane of the right camera



Depth from Two Views: Stereo

- If I search this line to find correspondences...

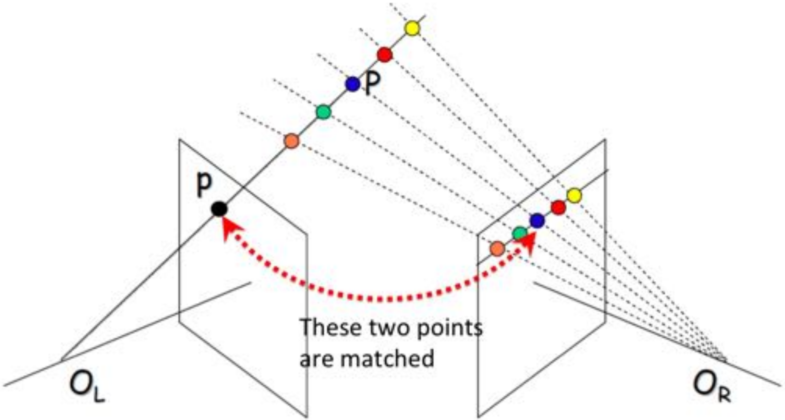


Figure: If I am able to find corresponding points in two images...

Depth from Two Views: Stereo

- I can get 3D!

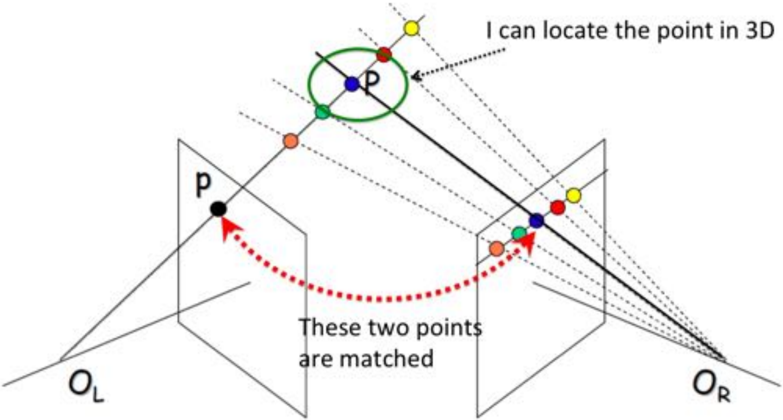
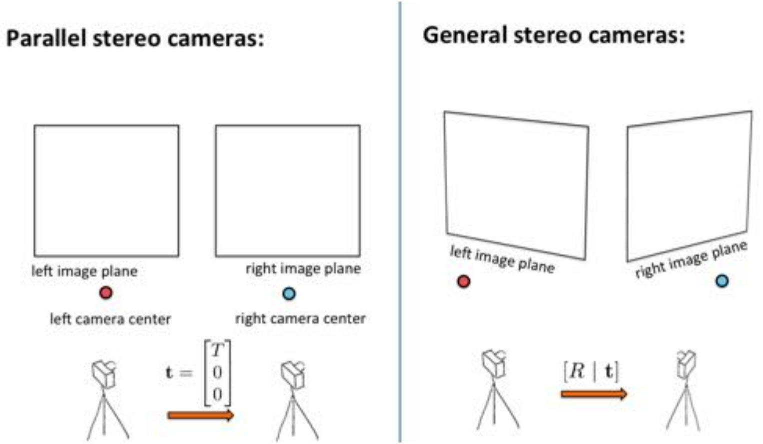


Figure: I can get a point in 3D by triangulation!

Stereo

Epipolar geometry

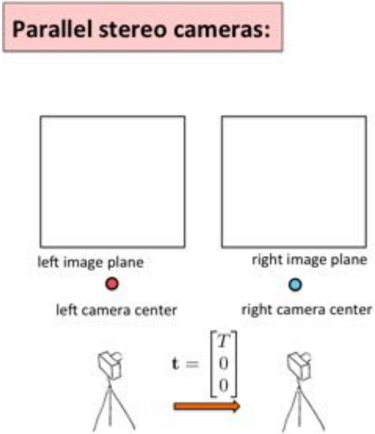
- Case with two cameras with parallel optical axes
- General case

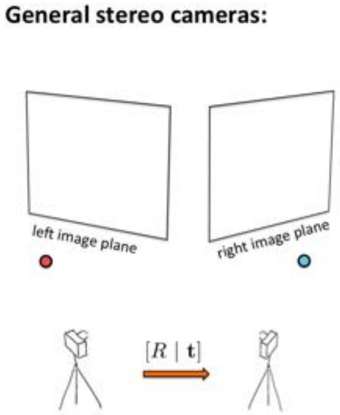


Stereo

Epipolar geometry

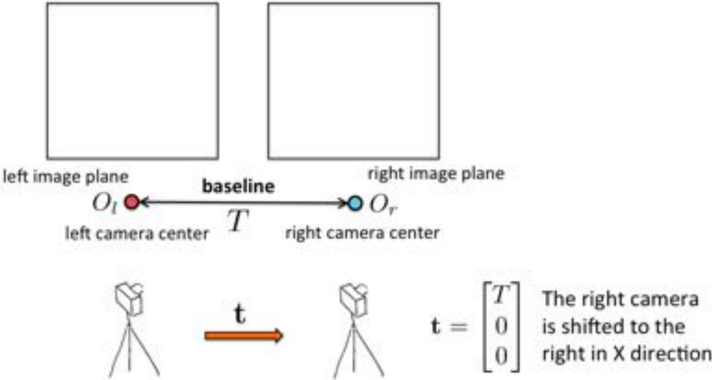
- Case with two cameras with parallel optical axes ← **First this**
- General case





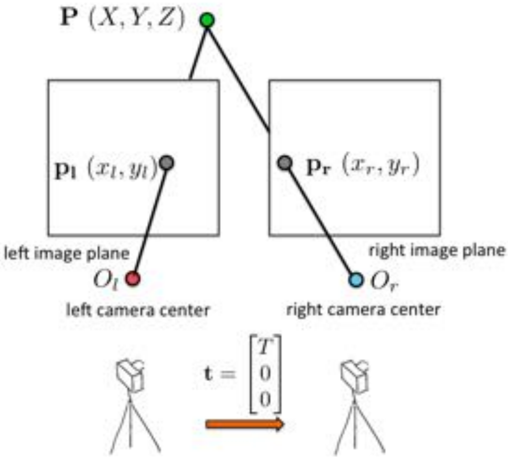
Stereo: Parallel Calibrated Cameras

- We assume that the two calibrated cameras (we know intrinsics and extrinsics) are parallel, i.e. the right camera is just some distance to the right of left camera. We assume we know this distance. We call it the **baseline**.



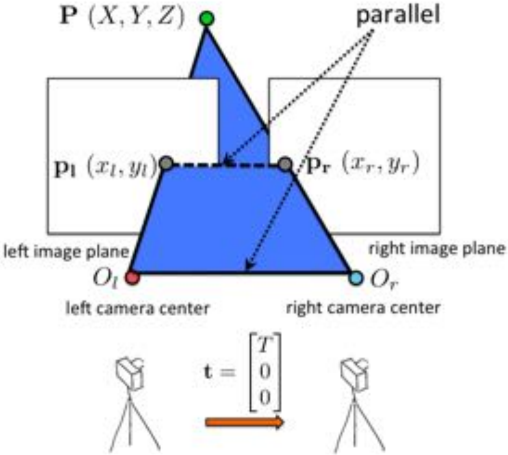
Stereo: Parallel Calibrated Cameras

- Pick a point P in the world


$\mathbf{P}\ (X, Y, Z)$
$\mathbf{p_l}\ (x_l, y_l)$
$\mathbf{p_r}\ (x_r, y_r)$
left image plane
right image plane
O_l
O_r
left camera center
right camera center
$\mathbf{t} = \begin{bmatrix} T \\ 0 \\ 0 \end{bmatrix}$

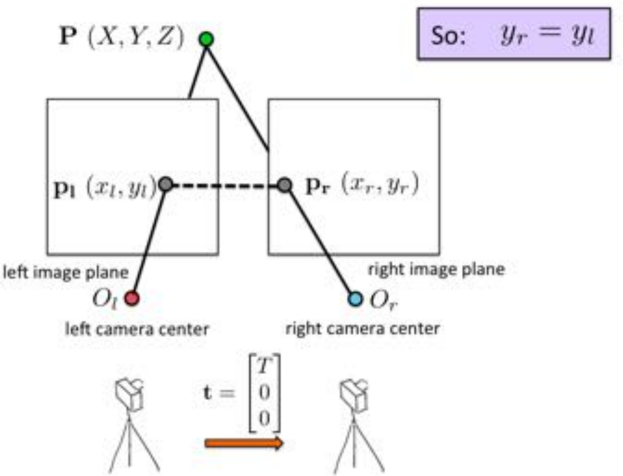
Stereo: Parallel Calibrated Cameras

- Points $\mathbf{O_l}$, $\mathbf{O_r}$ and $\mathbf{P}$ (and $\mathbf{p_l}$ and $\mathbf{p_r}$) lie on a plane. Since two image planes lie on the same plane (distance f from each camera), the lines $\mathbf{O_l O_r}$ and $\mathbf{p_l p_r}$ are parallel.



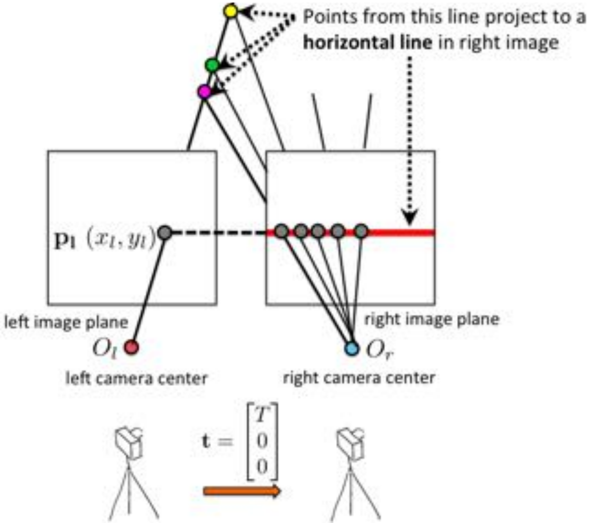
Stereo: Parallel Calibrated Cameras

- Since lines $\mathbf{O_l O_r}$ and $\mathbf{p_l p_r}$ are parallel, and $\mathbf{O_l}$ and $\mathbf{O_r}$ have the same y, then also $\mathbf{p_l}$ and $\mathbf{p_r}$ have the same y: $y_r = y_l$!



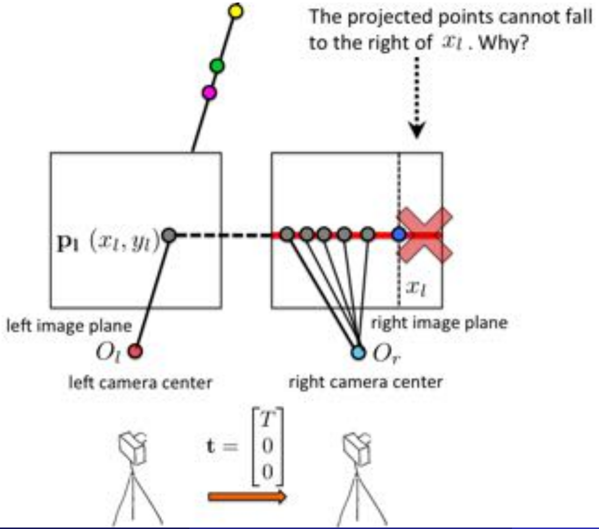
Stereo: Parallel Calibrated Cameras

- So all points on the projective line $\mathbf{O_l p_l}$ project to a horizontal line with $y = y_l$ on the right image. This is nice, let's remember this.


Points from this line project to a
horizontal line in right image
$\mathbf{p_l}$ (x_l, y_l)
left image plane
O_l
left camera center
right image plane
O_r
right camera center
$\mathbf{t} = \begin{bmatrix} T \\ 0 \\ 0 \end{bmatrix}$

Stereo: Parallel Calibrated Cameras

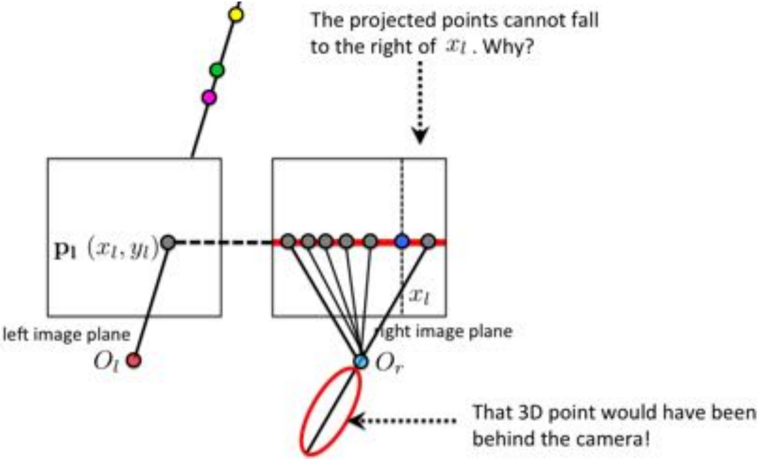
- Another observation: No point from $\mathbf{O_l p_l}$ can project to the right of x_l in the right image. **Why**?



The projected points cannot fall to the right of x_l. Why?

Stereo: Parallel Calibrated Cameras

- Because that would mean our image can see behind the camera...

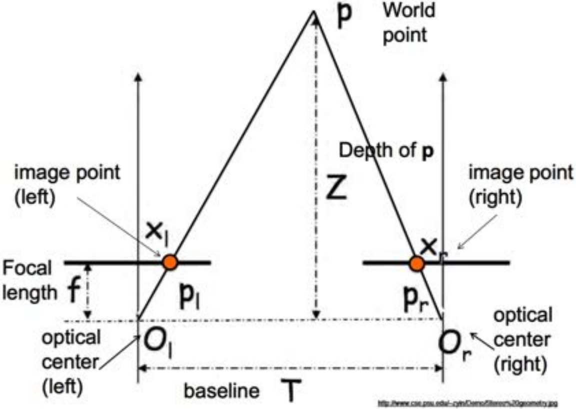


The projected points cannot fall to the right of x_l. Why?

That 3D point would have been behind the camera!

Stereo: Parallel Calibrated Cameras

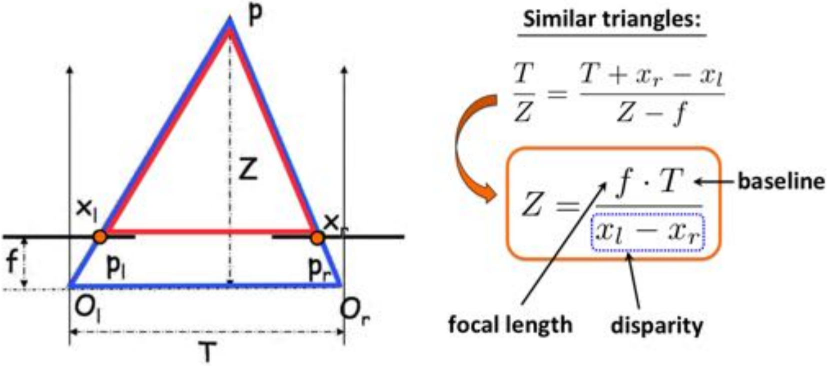
- Since our points $\mathbf{p_l}$ and $\mathbf{p_r}$ lie on a horizontal line, we can forget about y_l for a moment (it doesn't seem important). Let's look at the camera situation from the birdseye perspective instead. Let's see if we can find a connection between x_l, x_r and Z (because Z is what we want).



[Adopted from: J. Hays]

Stereo: Parallel Calibrated Cameras

- We can then use similar triangles to compute the depth of the point *P*



Similar triangles:

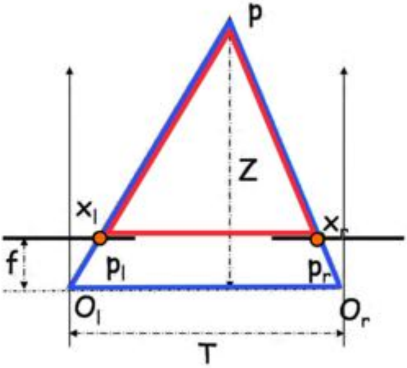
$$\frac{T}{Z} = \frac{T + x_r - x_l}{Z - f}$$

$$\boxed{Z = \frac{f \cdot T}{x_l - x_r}}$$

[Adopted from: J. Hays]

Stereo: Parallel Calibrated Cameras

- We can then use similar triangles to compute the depth of the point *P*



Similar triangles:

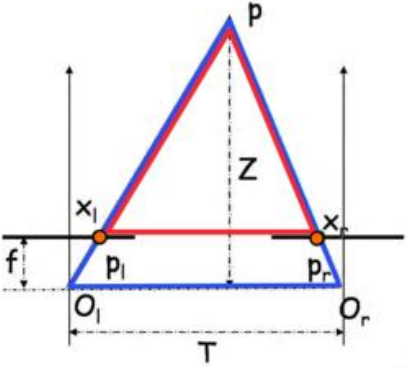
$$\frac{T}{Z} = \frac{T + x_l - x_r}{Z - f}$$

$$\boxed{Z = \frac{f \cdot T}{\boxed{x_r - x_l}}}$$

So if I know x_l and x_r, then I can compute Z!

Stereo: Parallel Calibrated Cameras

- We can then use similar triangles to compute the depth of the point P



Similar triangles:

$$\frac{T}{Z} = \frac{T + x_l - x_r}{Z - f}$$

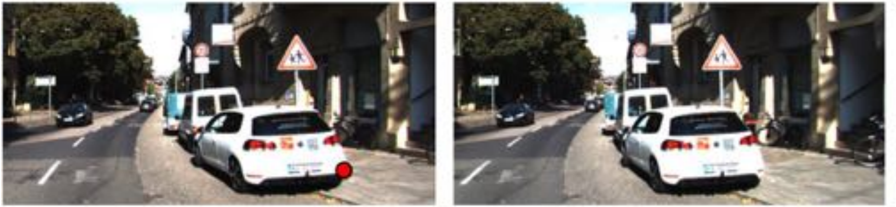
$$\boxed{Z = \frac{f \cdot T}{x_r - x_l}}$$

$$x = \frac{f \cdot X}{Z} + p_x$$

And if I know Z, I can compute X and Y, which gives me the point in 3D

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$?

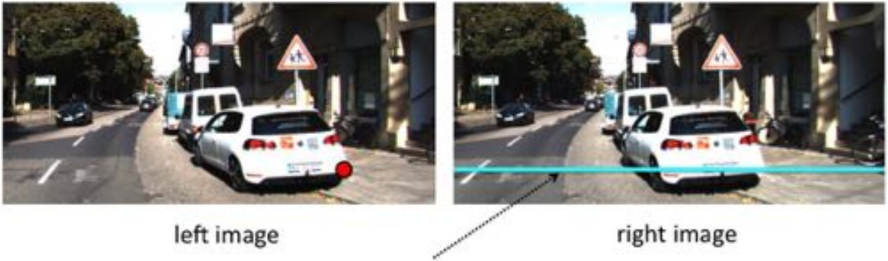


left image

right image

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching on line $y_r = y_l$.



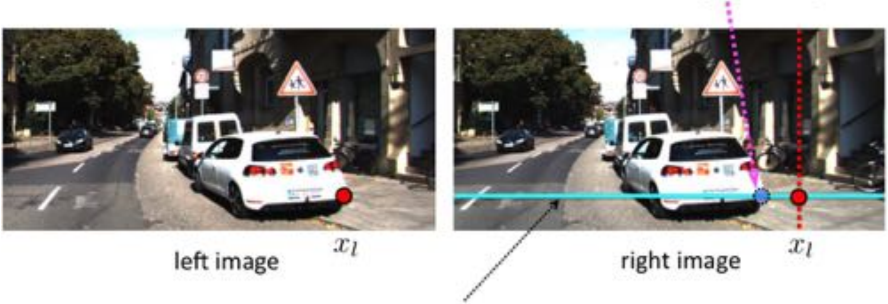
the match will be on this line (same y)

(CAREFUL: this is only true for parallel cameras. Generally, line not horizontal)

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching on line $y_r = y_l$.

We are looking for this point



left image

right image

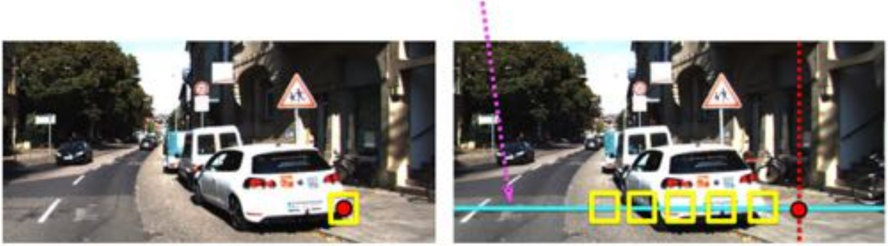
the match will be **on the left of** x_l

how do I find it?

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around (x_r, y_r)) should look similar to the patch around (x_l, y_l).

We call this line a **scanline**



left image

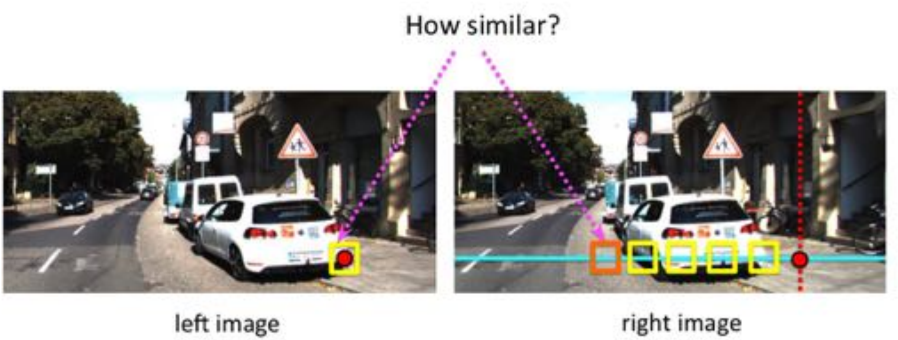
right image

we **scan** the line and **compare** patches to the one in the left image

We are looking for a patch on scanline most similar to patch on the left

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around (x_r, y_r)) should look similar to the patch around (x_l, y_l).

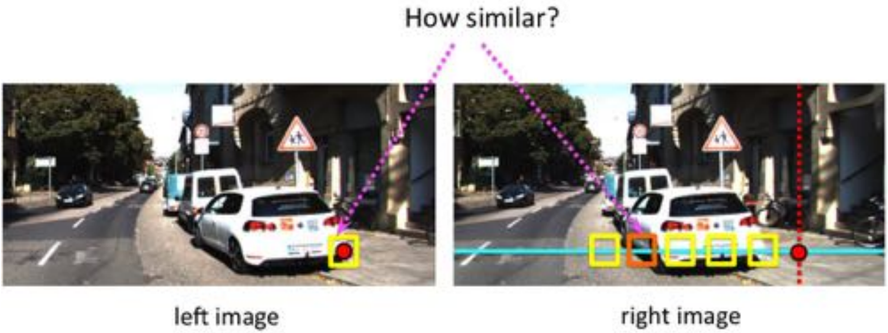


we **scan** the line and **compare** patches to the one in the left image

We are looking for a patch on scanline most similar to patch on the left

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around $(x_r, y_r))$ should look similar to the patch around (x_l, y_l).



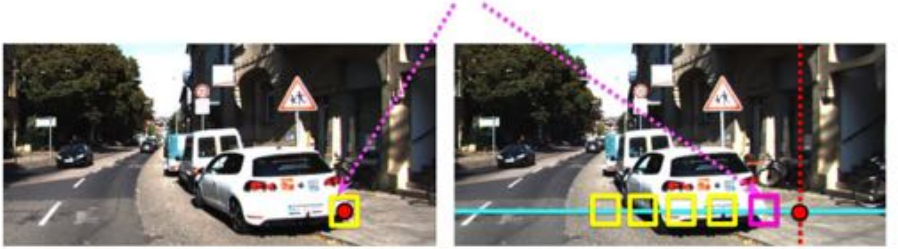
we **scan** the line and **compare** patches to the one in the left image

We are looking for a patch on scanline most similar to patch on the left

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around (x_r, y_r)) should look similar to the patch around (x_l, y_l).

Most similar. A match!



left image

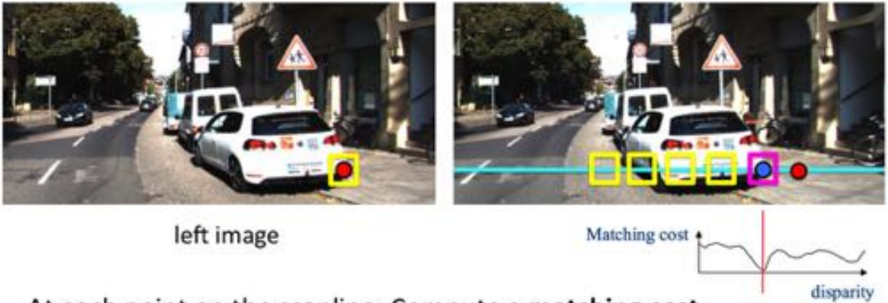
right image

we **scan** the line and **compare** patches to the one in the left image

We are looking for a patch on scanline most similar to patch on the left

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around (x_r, y_r)) should look similar to the patch around (x_l, y_l).



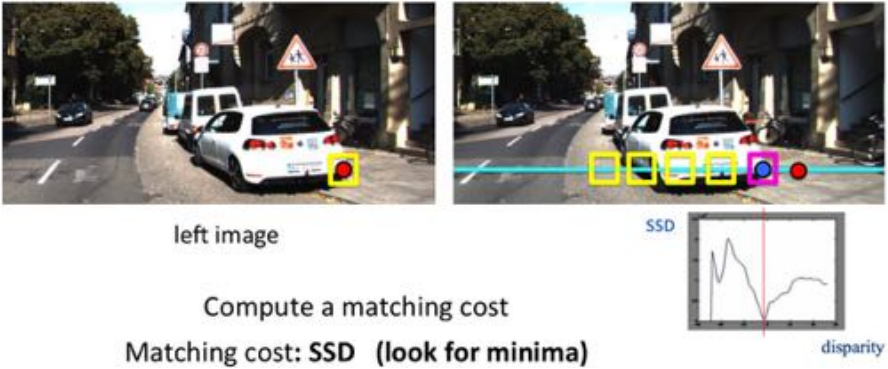
At each point on the scanline: Compute a **matching cost**

Matching cost: **SSD** or **normalized correlation**

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around $(x_r, y_r))$ should look similar to the patch around (x_l, y_l).

$$SSD(\text{patch}_l, \text{patch}_r) = \sum_x \sum_y (I_{\text{patch}_l}(x, y) - I_{\text{patch}_r}(x, y))^2$$



left image

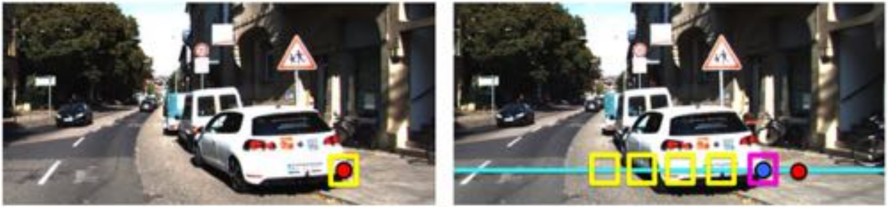
Compute a matching cost

Matching cost: **SSD** **(look for minima)**

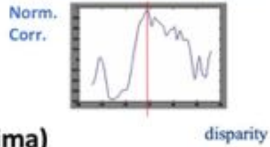
Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around $(x_r, y_r))$ should look similar to the patch around (x_l, y_l).

$$NC(\text{patch}_l, \text{patch}_r) = \frac{\sum_x \sum_y (I_{\text{patch}_l}(x, y) \cdot I_{\text{patch}_r}(x, y))}{||I_{\text{patch}_l}|| \cdot ||I_{\text{patch}_r}||}$$



left image



Compute a matching cost

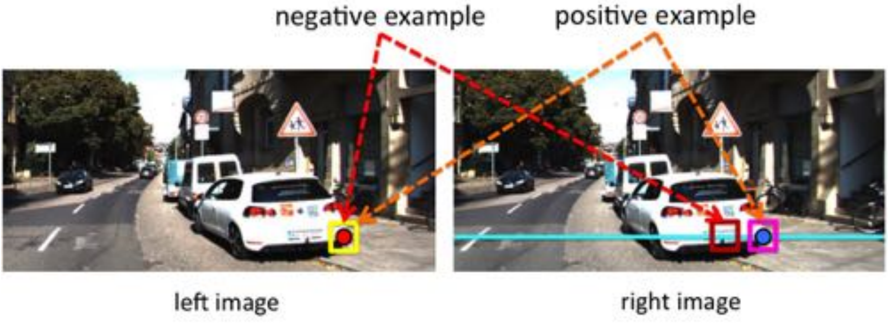
Matching cost: **Normalized Corr. (look for maxima)**

Stereo: Parallel Calibrated Cameras

- Version'2015: Can I do this task even better?

Stereo: Parallel Calibrated Cameras

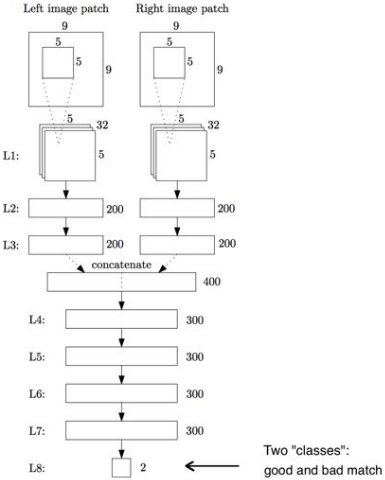
- Version'2015: Train a classifier! How can I get ground-truth?



Training examples: get positive and negative matches

Stereo: Parallel Calibrated Cameras

- Version'2015: Train a Neural Network classifier!

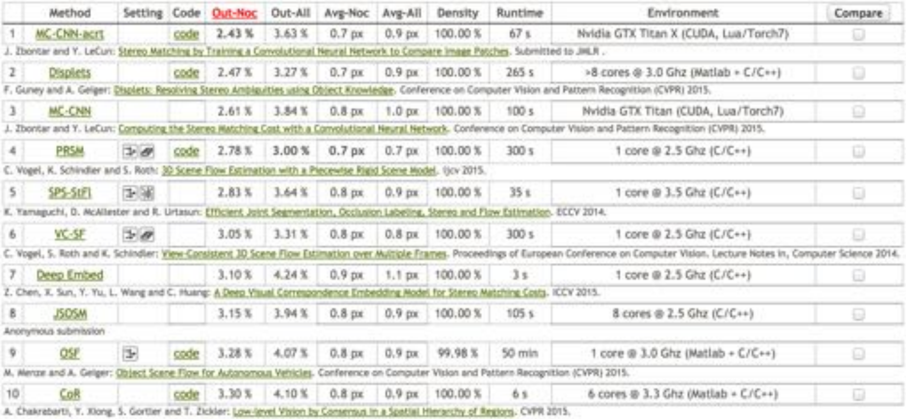


Two "classes":
good and bad match

[J. Zbontar and Y. LeCun: Computing the Stereo Matching Cost with a Convolutional Neural Network. CVPR'15]

Stereo: Parallel Calibrated Cameras

- Version'2015: Train a Neural Network classifier!
- To get the most amazing performance

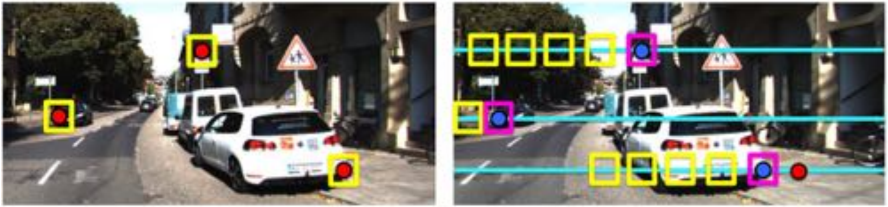


	Method	Setting	Code	Out-Noc	Out-All	Avg-Noc	Avg-All	Density	Runtime	Environment	Compare
1	MC-CNN-acrt		code	2.43 %	3.63 %	0.7 px	0.9 px	100.00 %	67 s	Nvidia GTX Titan X (CUDA, Lua/Torch7)	☐
	J. Zbontar and Y. LeCun: Stereo Matching by Training a Convolutional Neural Network to Compare Image Patches. Submitted to JMLR .										
2	Displets		code	2.47 %	3.27 %	0.7 px	0.9 px	100.00 %	265 s	>8 cores @ 3.0 Ghz (Matlab + C/C++)	☐
	F. Guney and A. Geiger: Displets: Resolving Stereo Ambiguities using Object Knowledge. Conference on Computer Vision and Pattern Recognition (CVPR) 2015.										
3	MC-CNN			2.61 %	3.84 %	0.8 px	1.0 px	100.00 %	100 s	Nvidia GTX Titan (CUDA, Lua/Torch7)	☐
	J. Zbontar and Y. LeCun: Computing the Stereo Matching Cost with a Convolutional Neural Network. Conference on Computer Vision and Pattern Recognition (CVPR) 2015.										
4	PRSM		code	2.78 %	3.00 %	0.7 px	0.7 px	100.00 %	300 s	1 core @ 2.5 Ghz (C/C++)	☐
	C. Vogel, K. Schindler and S. Roth: 3D Scene Flow Estimation with a Piecewise Rigid Scene Model. ijcv 2015.										
5	SPS-StFl			2.83 %	3.64 %	0.8 px	0.9 px	100.00 %	35 s	1 core @ 3.5 Ghz (C/C++)	☐
	K. Yamaguchi, D. McAllester and R. Urtasun: Efficient Joint Segmentation, Occlusion Labeling, Stereo and Flow Estimation. ECCV 2014.										
6	VC-SF			3.05 %	3.31 %	0.8 px	0.8 px	100.00 %	300 s	1 core @ 2.5 Ghz (C/C++)	☐
	C. Vogel, S. Roth and K. Schindler: View-Consistent 3D Scene Flow Estimation over Multiple Frames. Proceedings of European Conference on Computer Vision. Lecture Notes in, Computer Science 2014.										
7	Deep Embed			3.10 %	4.24 %	0.9 px	1.1 px	100.00 %	3 s	1 core @ 2.5 Ghz (C/C++)	☐
	Z. Chen, X. Sun, Y. Yu, L. Wang and C. Huang: A Deep Visual Correspondence Embedding Model for Stereo Matching Costs. ICCV 2015.										
8	JSOSM			3.15 %	3.94 %	0.8 px	0.9 px	100.00 %	105 s	8 cores @ 2.5 Ghz (C/C++)	☐
	Anonymous submission										
9	OSF		code	3.28 %	4.07 %	0.8 px	0.9 px	99.98 %	50 min	1 core @ 3.0 Ghz (Matlab + C/C++)	☐
	M. Menze and A. Geiger: Object Scene Flow for Autonomous Vehicles. Conference on Computer Vision and Pattern Recognition (CVPR) 2015.										
10	CoR		code	3.30 %	4.10 %	0.8 px	0.9 px	100.00 %	6 s	6 cores @ 3.3 Ghz (Matlab + C/C++)	☐
	A. Chakrabarti, Y. Xiong, S. Gortler and T. Zickler: Low-level Vision by Consensus in a Spatial Hierarchy of Regions. CVPR 2015.										

Figure: Performance on KITTI (metrics is error, so lower is better)

Stereo: Parallel Calibrated Cameras

- For each point $\mathbf{p_l} = (x_l, y_l)$, how do I get $\mathbf{p_r} = (x_r, y_r)$? By matching. Patch around (x_r, y_r)) should look similar to the patch around (x_l, y_l).

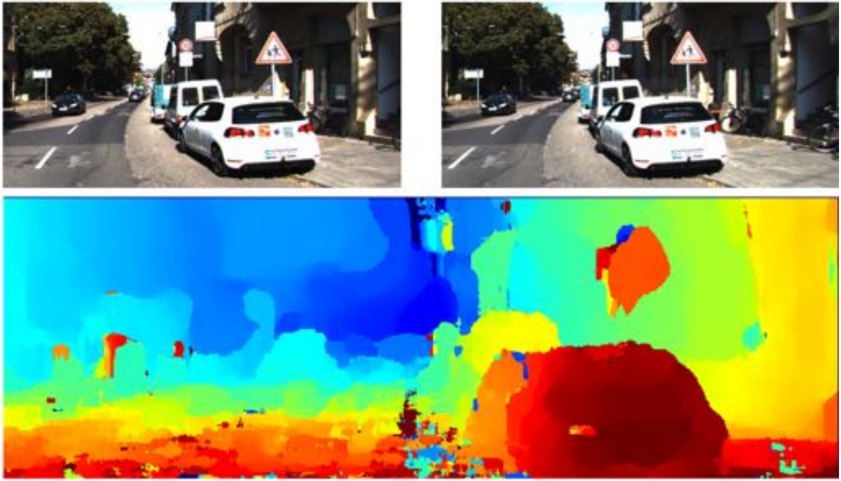


left image

Do this for all the points in the left image!

Stereo: Parallel Calibrated Cameras

- We get a disparity map as a result

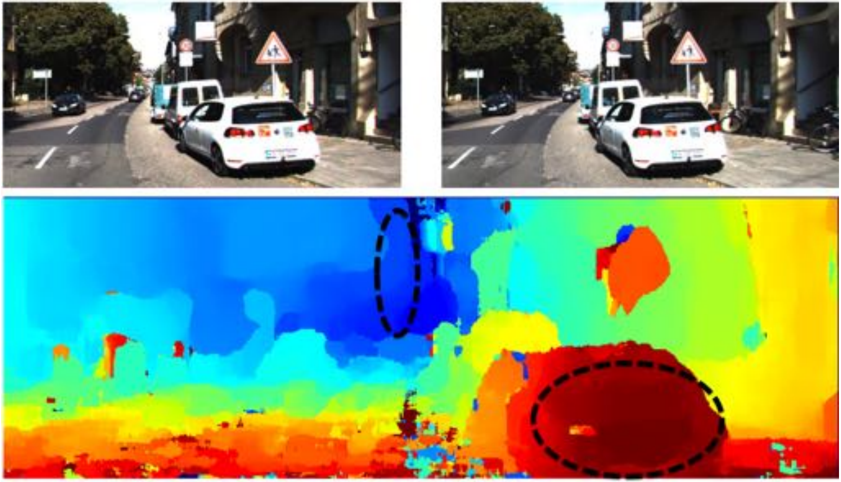


Result: **Disparity map**

(red values large disp., blue small disp.)

Stereo: Parallel Calibrated Cameras

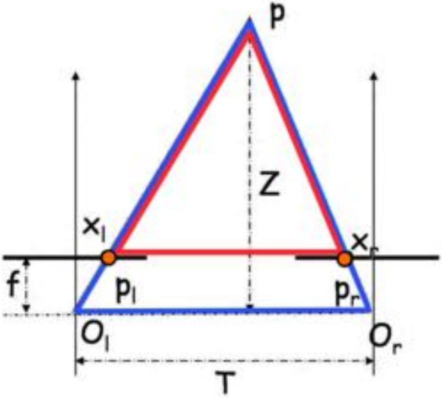
- We get a disparity map as a result



Things that are closer have **larger disparity** than those that are far away from camera. Why?

Stereo: Parallel Calibrated Cameras

- Depth and disparity are inversely proportional



Similar triangles:

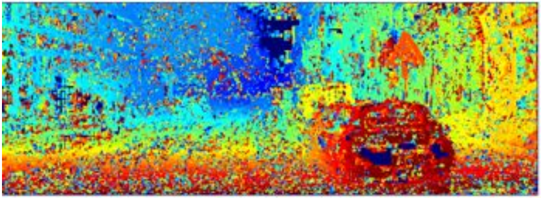
$$\frac{T}{Z} = \frac{T + x_l - x_r}{Z - f}$$

$$\boxed{Z = \frac{f \cdot T}{\boxed{x_r - x_l}}}$$

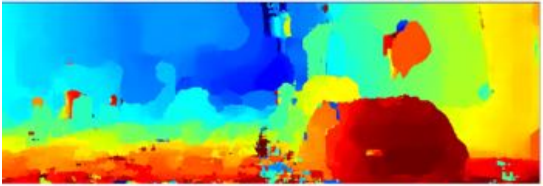
Depth (Z) and disparity are **inversely proportional**

Stereo: Parallel Calibrated Cameras

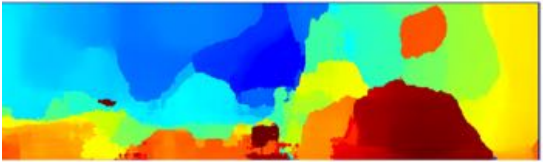
- Smaller patches: more detail, but noisy. Bigger: less detail, but smooth



patch size = 5



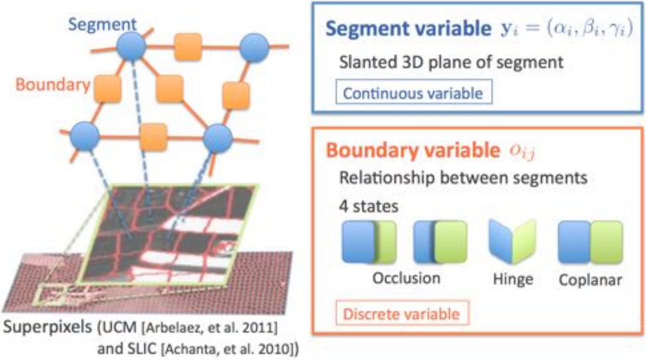
patch size = 35



patch size = 85

You Can Do It Much Better...

- With Energy Minimization on top, e.g., a Markov Random Field (MRF)



Segment variable $\mathbf{y}_i = (\alpha_i, \beta_i, \gamma_i)$

Slanted 3D plane of segment

Continuous variable

Boundary variable o_{ij}

Relationship between segments

4 states

Discrete variable

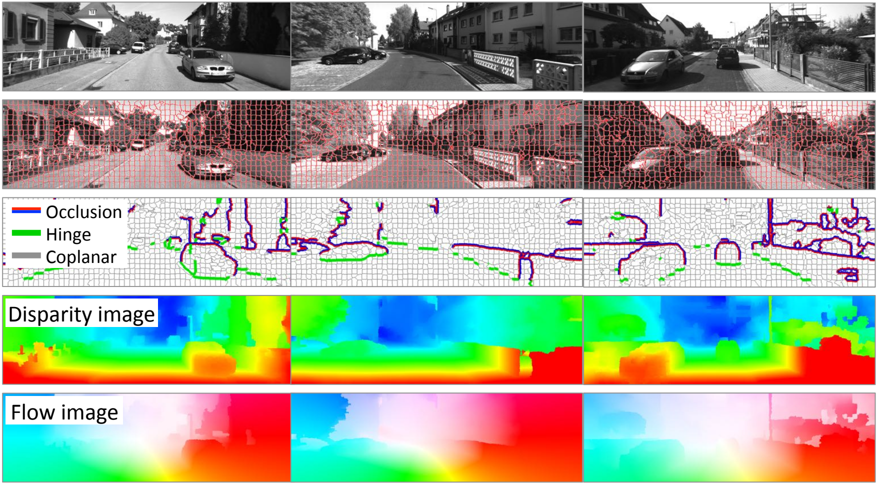
Superpixels (UCM [Arbelaez, et al. 2011] and SLIC [Achanta, et al. 2010])

K. Yamaguchi, D. McAllester, R. Urtasun, *Efficient Joint Segmentation, Occlusion Labeling, Stereo and Flow Estimation*, ECCV 2014

Paper: `http://www.cs.toronto.edu/~urtasun/publications/yamaguchi_et_al_eccv14.pdf`
Code: `http://ttic.uchicago.edu/~dmcallester/SPS/index.html`

You Can Do It Much Better...

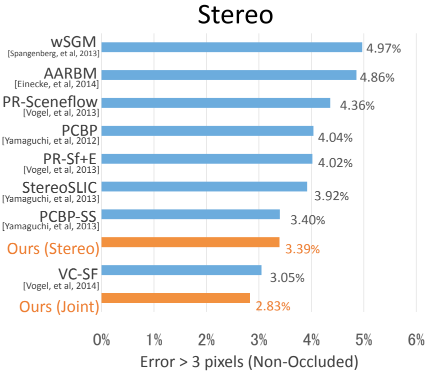
[K. Yamaguchi, D. McAllester and R. Urtasun, ECCV 2014]

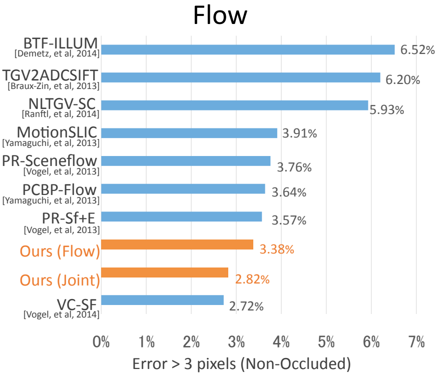


Look at State-of-the-art on KITTI

Where “Ours” means: [K. Yamaguchi, D. McAllester and R. Urtasun, ECCV 2014]

- How can we evaluate the performance of a stereo algorithm?





- Autonomous driving dataset KITTI: `http://www.cvlibs.net/datasets/kitti/`

From Disparity We Get...

- Depth: Once you have disparity, you have 3D

Figure: K. Yamaguchi, D. McAllester and R. Urtasun, ECCV 2014

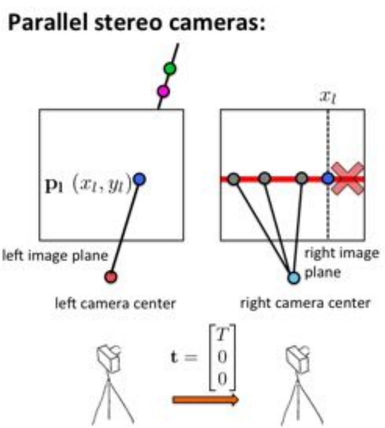
From Disparity We Get...

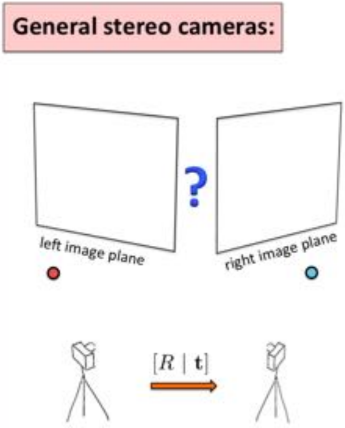
- Money ;)

Stereo

Epipolar geometry

- Case with two cameras with parallel optical axes
- General case ← **Next time**


Parallel stereo cameras:
x_l
$\mathbf{p}_l$ (x_l, y_l)
left image plane
right image plane
left camera center
right camera center
$\mathbf{t} = \begin{bmatrix} T \\ 0 \\ 0 \end{bmatrix}$


General stereo cameras:
?
left image plane
right image plane
$[R \mid \mathbf{t}]$