In Defense of Classical Image Processing: Fast Depth Completion on the CPU

J.Ku, A.Harakeh, L. Waslander

Presenter : Mohammad Sadegh Abadijou

Presentaition Headlines

- Introduction
- Newer Approaches
- Paper Method
- Results

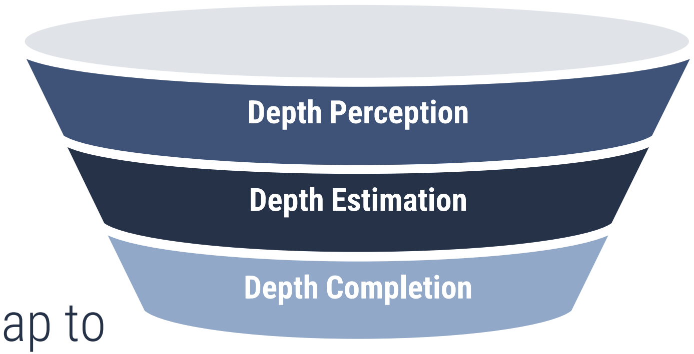
1

INTRODUCTION

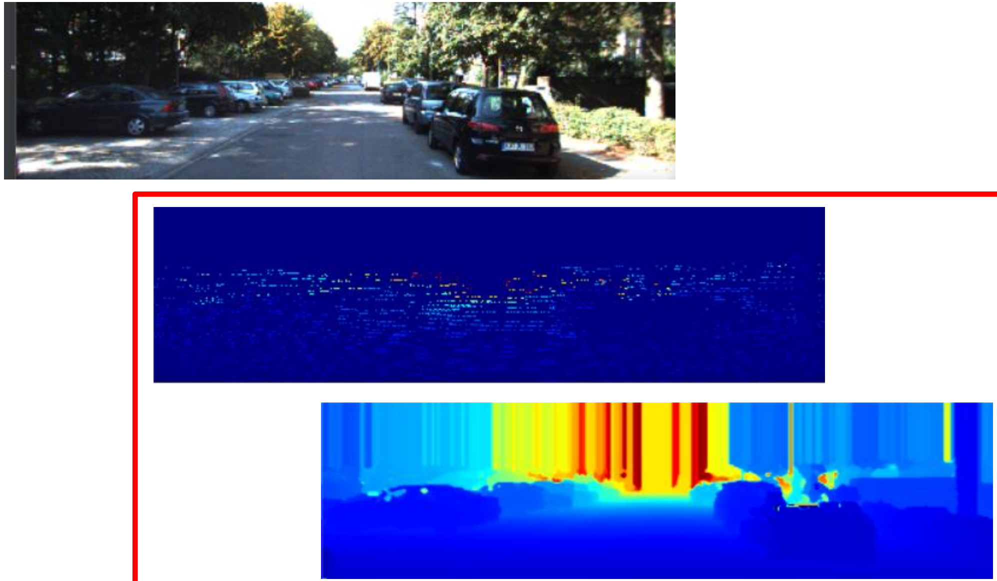
What is Depth Completion?

Introduction

- Depth Perception
  - The procedure to Find Depth in 2D
- Depth Estimation
  - Depth Completion
    - Converting Sparse DEPTH Dense map to Depth map



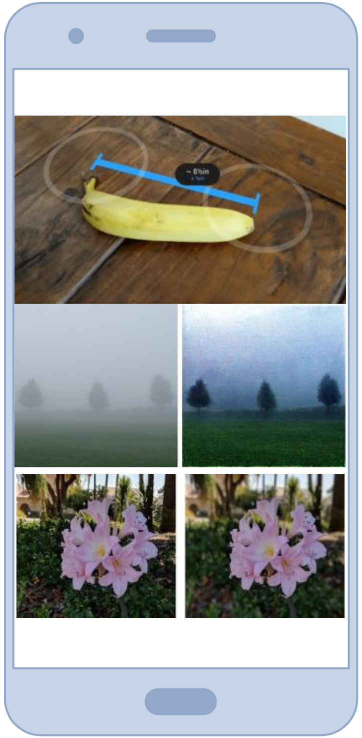
Introduction



- Depth Perception
- Depth Estimation
- Depth Completion

Application

- Augmented reality
- Robotics and object trajectory estimation
- Haze and Fog removal
- Portrait mode



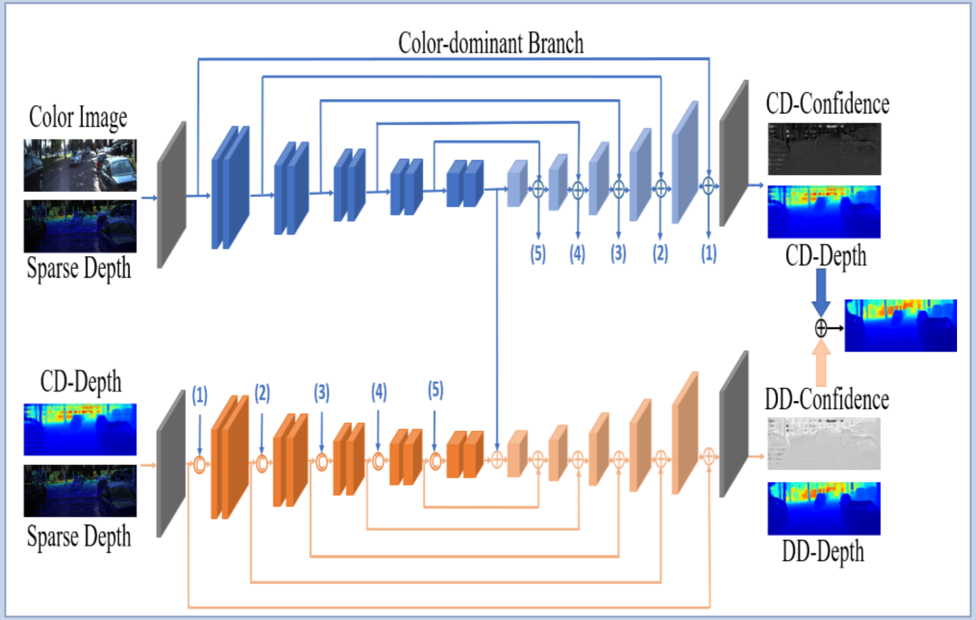
2

NEWER APPROACHES

Faster but on GPU

PENet-2020

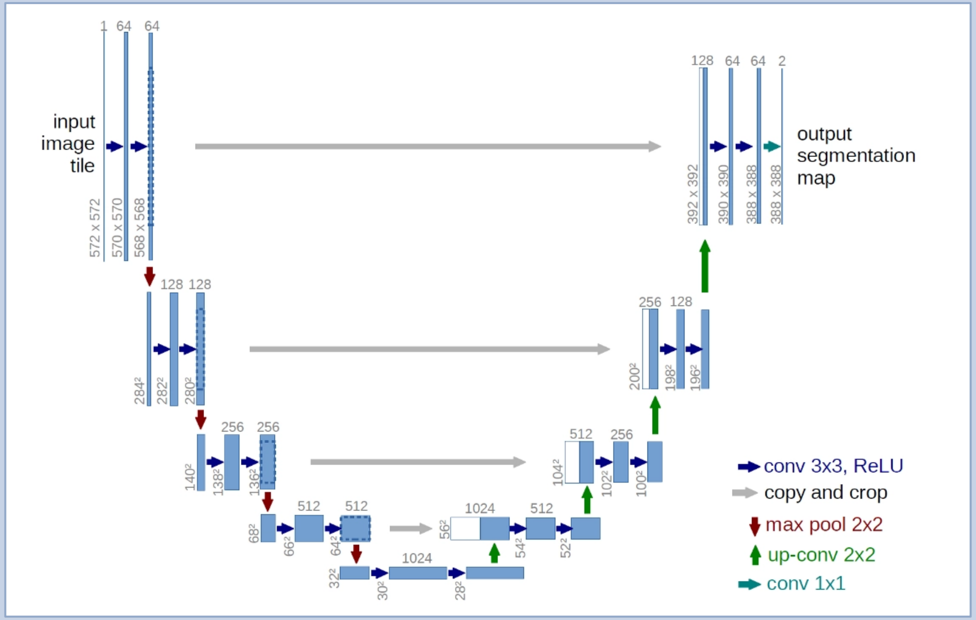
- More Accurate
- Faster (GPU)
- Supervised



	RMSE	MAE	Run Time
PENet[9]	730.08	210	0.04
Classic Method [1]	1288.46	302.6	0.011

Monodepth-2019

- More Accurate
- Faster (GPU)
- Unsupervised
- Multi Loss



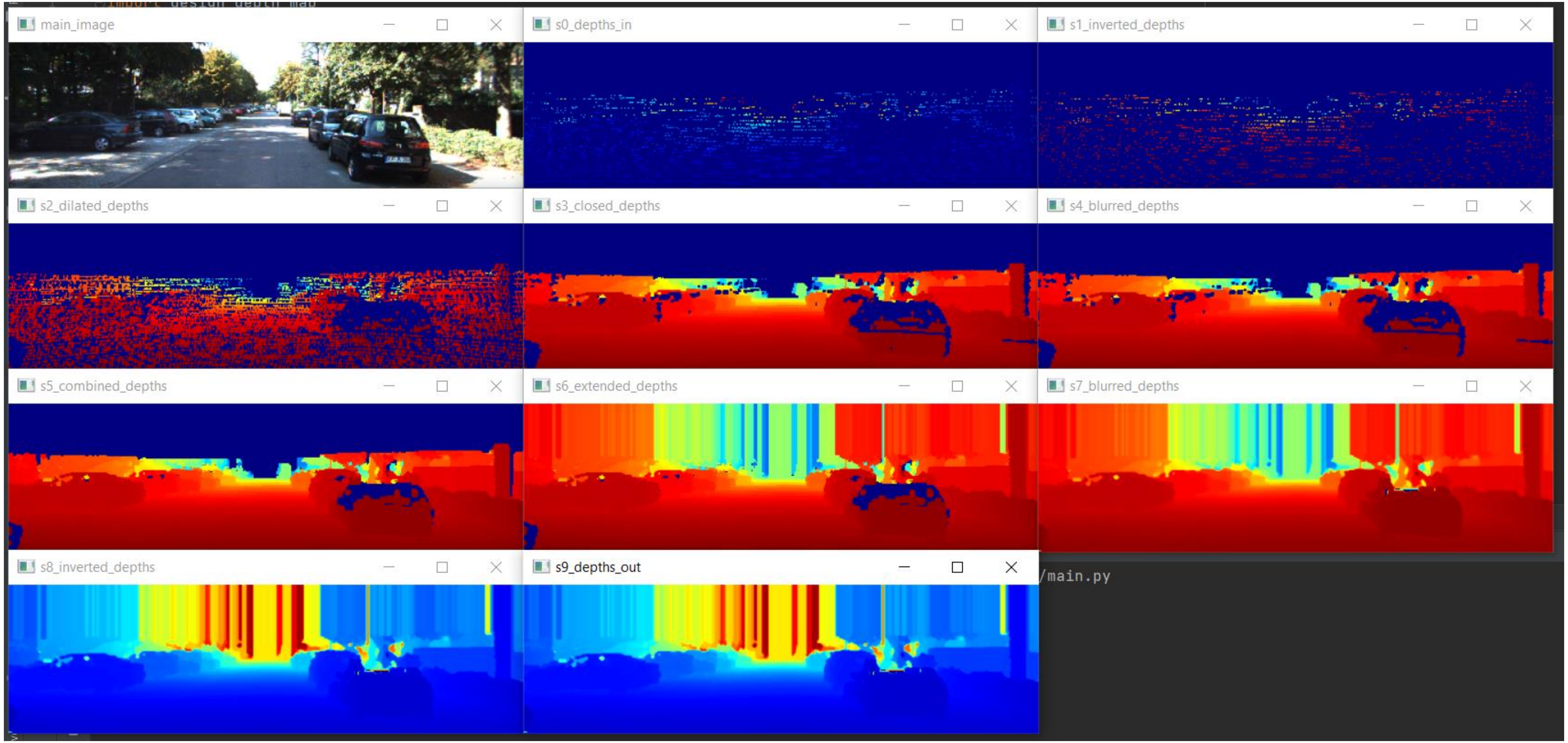
3

PAPER METHOD

Dive into Classical Image Processing.

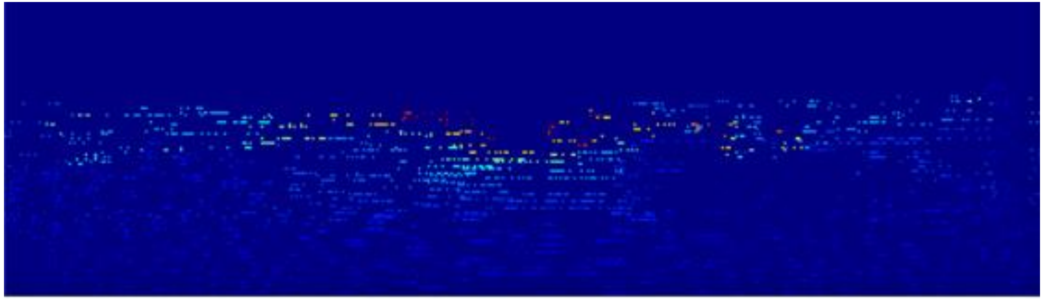
8 STEPS TO REDEMPTION

The Authors have used morphology many times

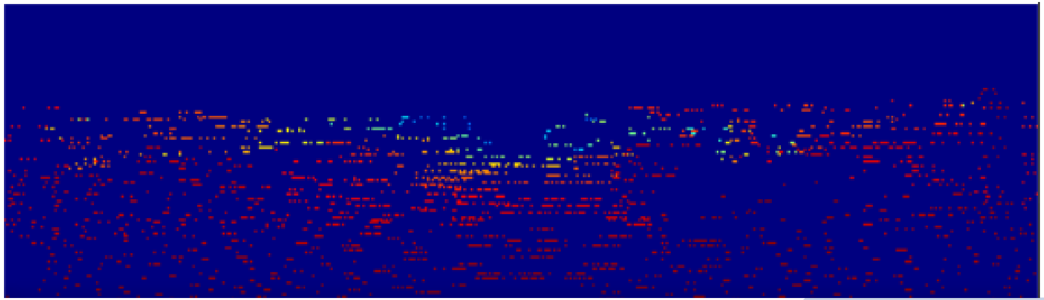

main_image
s0_depths_in
s1_inverted_depths
s2_dilated_depths
s3_closed_depths
s4_blurred_depths
s5_combined_depths
s6_extended_depths
s7_blurred_depths
s8_inverted_depths
s9_depths_out
/main.py

Step 1 – Depth Inverted

Input



Result



How?

Result = 100 - Input

Why?

Reduce the difference between Valid pixels and Empty pixels to prevent negative impact of dilation on edges.

Step 1 – Depth Inverted

Code

```
s1_inverted_depths = np.copy(depths_in)
valid_pixels = (s1_inverted_depths > 0.1)
s1_inverted_depths[valid_pixels] = \
    max_depth - s1_inverted_depths[valid_pixels]
```

Step 2 – Dilating With Custom Kernel

How?

Dilating with 3*3 ,5*5, 7*7 diamond filter

We can use cross filter here

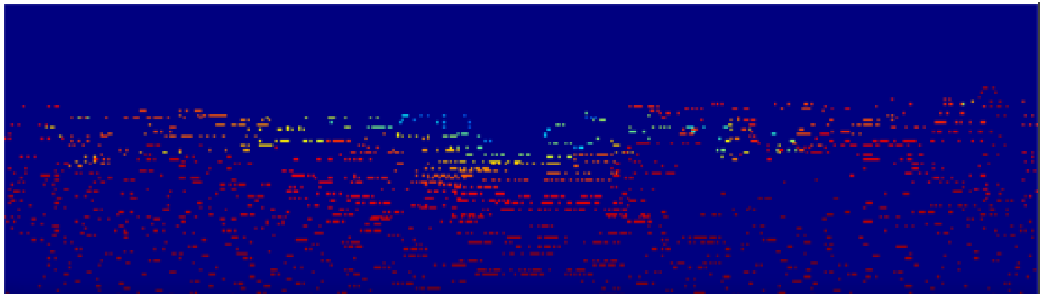
0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

Diamond

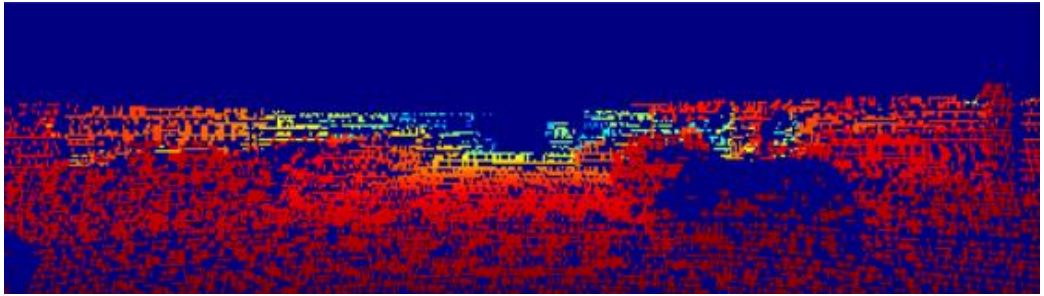
Why?

To Extend Efficient Pixels

Input



Result



Step 2 – Dilating With Custom Kernel

Code

```
dilated_far = cv2.dilate(
    np.multiply(s1_inverted_depths, valid_pixels_far),
    dilation_kernel_far)
dilated_med = cv2.dilate(
    np.multiply(s1_inverted_depths, valid_pixels_med),
    dilation_kernel_med)
dilated_near = cv2.dilate(
    np.multiply(s1_inverted_depths, valid_pixels_near),
    dilation_kernel_near)

# Find valid pixels for each binned dilation
valid_pixels_near = (dilated_near > 0.1)
valid_pixels_med = (dilated_med > 0.1)
valid_pixels_far = (dilated_far > 0.1)

# Combine dilated versions, starting farthest to nearest
s2_dilated_depths = np.copy(s1_inverted_depths)
s2_dilated_depths[valid_pixels_far] = dilated_far[valid_pixels_far]
s2_dilated_depths[valid_pixels_med] = dilated_med[valid_pixels_med]
s2_dilated_depths[valid_pixels_near] = dilated_near[valid_pixels_near]

    dilation_kernel_near)
```

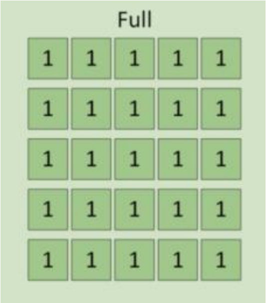
Step 3 – Small Hole Closure

How?

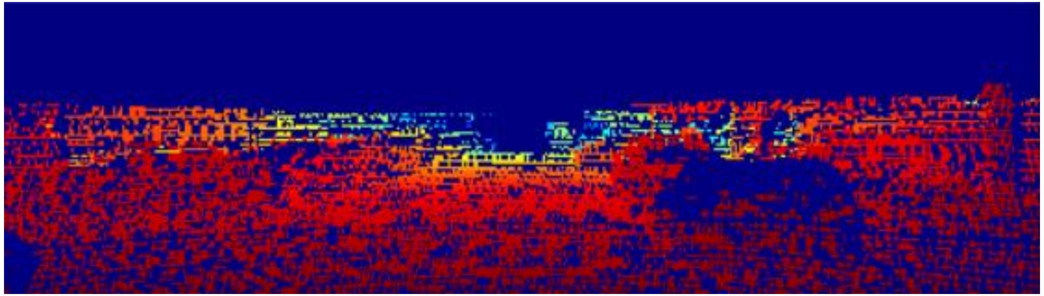
1- Closing with 5*5 full filter

2- Using Median filter for denoising

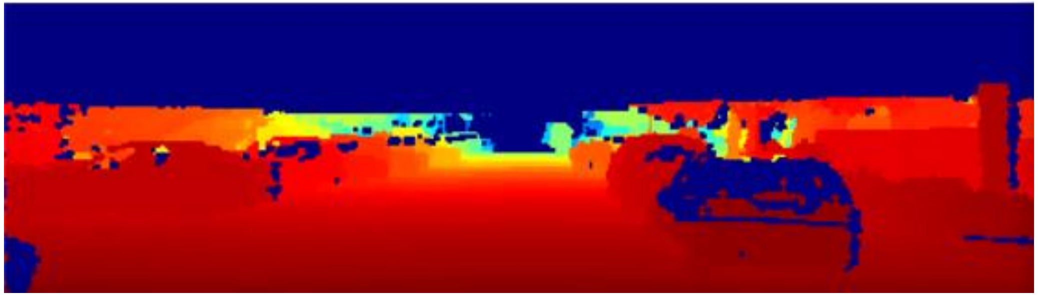
7*7 and 3*3 decrease RMSE



Input



Result



Why?

Fill small holes which is remaining from previous step

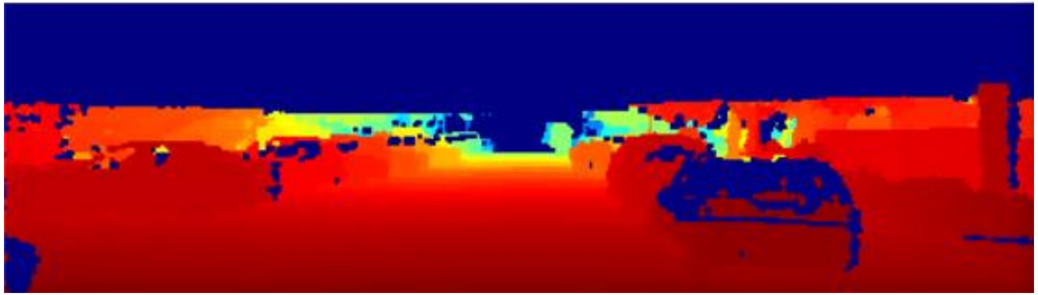
Step 3 – Small Hole Closure

Code

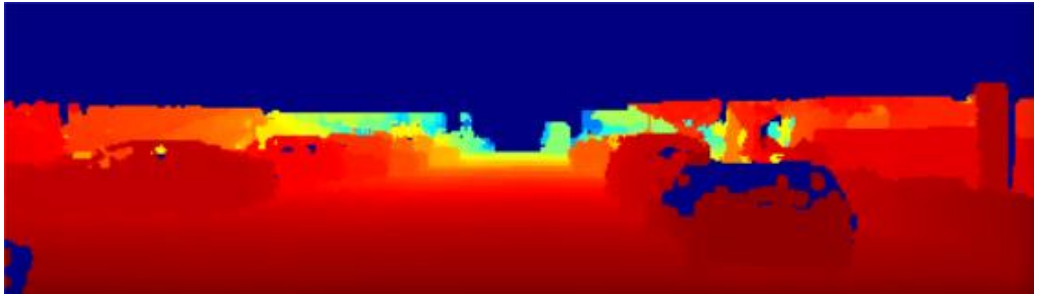
```
s3_closed_depths = cv2.morphologyEx(
    s2_dilated_depths, cv2.MORPH_CLOSE, kernels.FULL_KERNEL_5)

s4_blurred_depths = np.copy(s3_closed_depths)
blurred = cv2.medianBlur(s3_closed_depths, 5)
valid_pixels = (s3_closed_depths > 0.1)
s4_blurred_depths[valid_pixels] = blurred[valid_pixels]
```

Step 4 – Medium Hole Fill

Input



Result



How?

1- Dilating on unchanged pixel with
7*7 Full filter

Why?

Fill Medium Holes

Step 4 – Small Hole Fill

Code

```
top_mask = np.ones(depths_in.shape, dtype=np.bool)
for pixel_col_idx in range(s4_blurred_depths.shape[1]):
    pixel_col = s4_blurred_depths[:, pixel_col_idx]
    top_pixel_row = np.argmax(pixel_col > 0.1)
    top_mask[0:top_pixel_row, pixel_col_idx] = False

valid_pixels = (s4_blurred_depths > 0.1)
empty_pixels = ~valid_pixels & top_mask

dilated = cv2.dilate(s4_blurred_depths, kernels.FULL_KERNEL_7)
s5_dilated_depths = np.copy(s4_blurred_depths)
s5_dilated_depths[empty_pixels] = dilated[empty_pixels]
```

Step 5 – Extension to Top of Frame

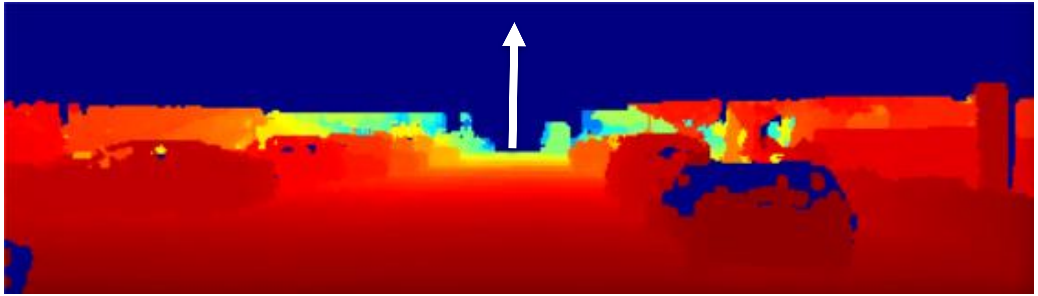
How?

the top value along each column is extrapolated to the top of the image, providing a denser depth map output.

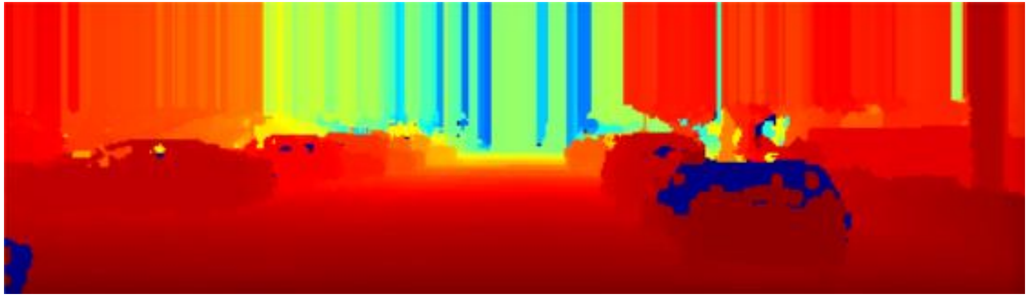
Why?

To account for tall objects such as trees, poles, and buildings To account for tall objects such as trees, poles, and buildings that extend above the top of LIDAR points

Input



Result



Step 5 – Extension to Top of Frame

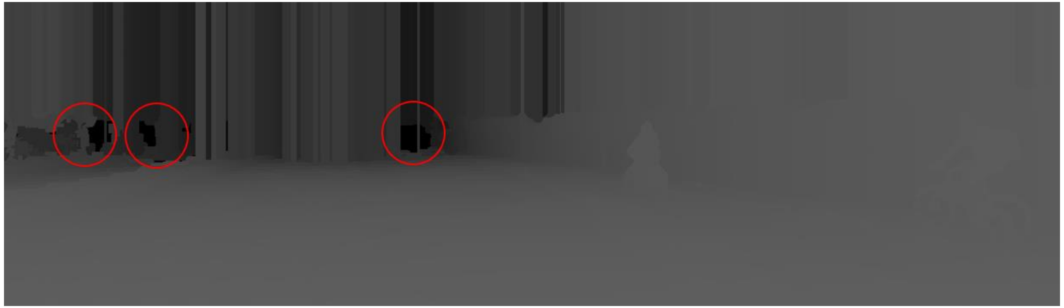
Code

```
s6_extended_depths = np.copy(s5_dilated_depths)
top_mask = np.ones(s5_dilated_depths.shape, dtype=np.bool)

top_row_pixels = np.argmax(s5_dilated_depths > 0.1, axis=0)
top_pixel_values = s5_dilated_depths[top_row_pixels,
                                      range(s5_dilated_depths.shape[1])]
for pixel_col_idx in range(s5_dilated_depths.shape[1]):
    s6_extended_depths[0:top_row_pixels[pixel_col_idx],
                       pixel_col_idx] = top_pixel_values[pixel_col_idx]
```

Step 6 – Large Hole Fill

How?

Why?



Step 6 – Large Hole Fill

Code

```
s7_blurred_depths = np.copy(s6_extended_depths)
for i in range(6):
    empty_pixels = (s7_blurred_depths < 0.1) & top_mask
    dilated = cv2.dilate(s7_blurred_depths, kernels.FULL_KERNEL_31)
    s7_blurred_depths[empty_pixels] = dilated[empty_pixels]
```

Step 7 – Median and Gaussian Blur

How?

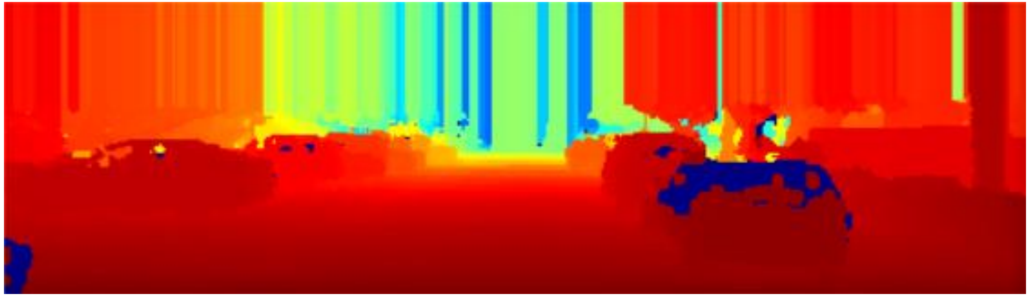
1- 5 × 5 kernel median blur

2- 5 × 5 kernel Gaussian blur

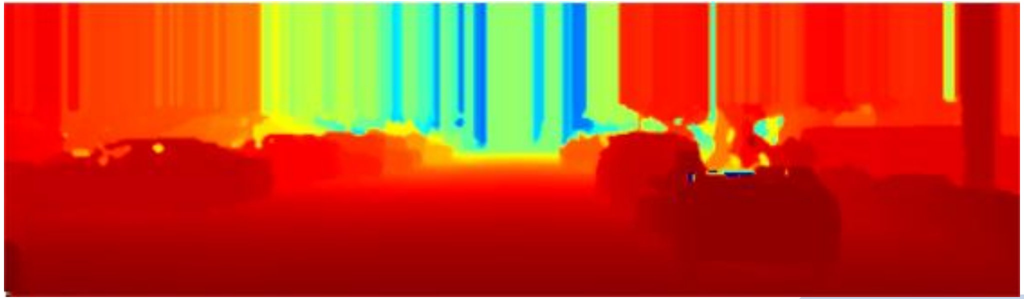
Why?

To remove these outliers

Input



Result



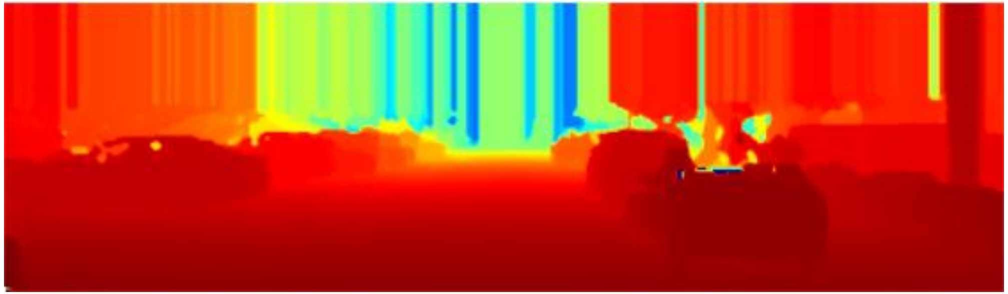
Step 7 – Median and Gaussian Blur

Code

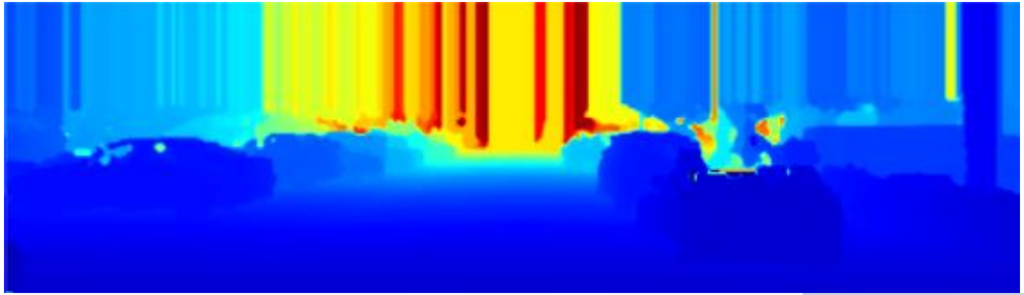
```
s7_blurred_depths = np.copy(s6_extended_depths)
for i in range(6):
    empty_pixels = (s7_blurred_depths < 0.1) & top_mask
    dilated = cv2.dilate(s7_blurred_depths, kernels.FULL_KERNEL_31)
    s7_blurred_depths[empty_pixels] = dilated[empty_pixels]
blurred = cv2.medianBlur(s7_blurred_depths, 5)
valid_pixels = (s7_blurred_depths > 0.1) & top_mask
s7_blurred_depths[valid_pixels] = blurred[valid_pixels]
blurred = cv2.GaussianBlur(s7_blurred_depths, (5, 5), 0)
valid_pixels = (s7_blurred_depths > 0.1) & top_mask
s7_blurred_depths[valid_pixels] = blurred[valid_pixels]
```

Step 8 – Depth Inversion

Input



Result



How?

Result = 100 - Input

Why?

Step 8 – Depth Inversion

Code

```
s8_inverted_depths = np.copy(s7_blurred_depths)
valid_pixels = np.where(s8_inverted_depths > 0.1)
s8_inverted_depths[valid_pixels] = \
    max_depth - s8_inverted_depths[valid_pixels]
depths_out = s8_inverted_depths
```

4

RESULT

Result

Method	iRMSE (1/km)	iMAE (1/km)	RMSE (mm)	MAE (mm)	Runtime (s)
NadarayaW	6.34	1.84	1852.60	416.77	0.05
SparseConvs	4.94	1.78	1601.33	481.27	**0.01**
NN+CNN	**3.25**	**1.29**	1419.75	416.14	0.02
Ours (IP-Basic)	3.78	**1.29**	**1288.46**	**302.60**	0.011

	Dataset	RMSE	MAE
Paper Implementation	KITTI Test Set	1288.46	302.6
My Implementation	KITTI Validation Cropped	1651.34	696.7

An Issue