# PAPER ANALYSIS

Presented by Yannis He

-

Paper: **In Defense of Classical Image Processing: Fast Depth Completion on the CPU (IP-basic)**

Authors: Jason Ku, Ali Harakeh, and Steven L. Waslander | University of Waterloo

https://arxiv.org/abs/1802.00036

https://github.com/kujason/ip_basic

## INTRODUCTION

- Abstract (Background):
  - Trends to deep learning made people move away from handcrafted classical image processing algorithms.
  - With well designed algorithm, we are able to outperform neural network based method on depth completion
  - The process is
    - Simple and fast, and can be runs on the CPU.
    - Only relies on basic image processing operations to perform depth completion of sparse LIDAR depth data
    - Requires no training (data independent)
- Introduction:
  - Depth Completion: Inferring a **dense** depth map from image and **sparse** depth map inputs
    - An important task for machine vision and robotics
    - The sparse depth map limits both the performance and the operational range of many perception algorithm that rely on the depth as input
      - Ex. 3D object detection algorithm
  - Current approaches
    - Deep learning based are attractive
      - It requires minimal human design decisions due to the data-driven nature
      - Many consequences:
        - Finite computing power on embedded system (GPU are power hungry)
        - Deep learning (without understanding problems can lead to sub-optimal network design)

## INTRODUCTION (CONT')

- Contributions:
  - Show that on certain problems, deep learning based approaches can be outperformed by well designed classical image processing based algorithms.
    - Shown through depth completion that relies on image processing operations only
      - Runs at 90 Hz on CPU and ranks first among all published methods on the KITTI depth completion benchmark
      - Outperforms CNN based approaches by a wide margin

## RELATED WORK

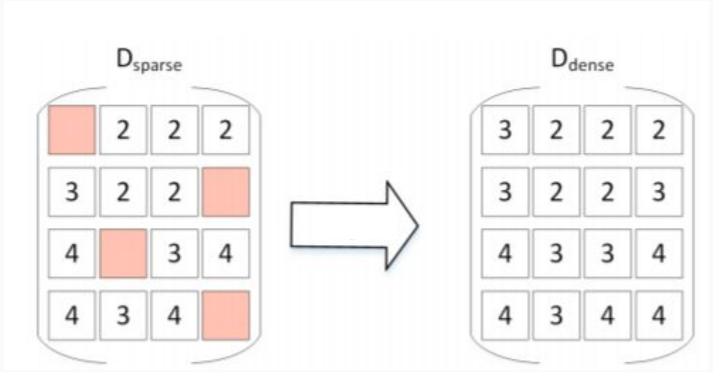
- Two main classes: Guided Depth Completion & Non-guided Depth Completion
- Guided Depth Completion:
  - Recent research focus
  - Rely on color images for guidance to perform depth map completion.
  - Ex.
    - Recent research focus
    - use joint bilateral filtering to perform "hole filling"
    - Median filter
  - Shown to produce higher quality depth maps
  - Data-driven, requiring large amounts of training data to generalize well
  - Suffer from a dependency on quality of the guiding color images
  - Assume operation on a regular grid (fail when applied to very sparse input)
- Non-Guided Depth Completion
  - Use only a sparse depth map to produce a dense one
  - Ex. Use repetitive structures to identify similar patches in 3D across different scales
  - Nadaraya-Watson kernel regression to estimate missing values for depth completion
  - CNNs trained on nearest neighbour

## PROBLEM FORMULATION

- The problem can be described as:
  - Given an image $I \in R^{M \times N}$ and a sparse depth map $D_{sparse} \in R^{M \times N}$
  - Find **f** that approximates a true function, $f: R^{M \times N} \times R^{M \times N} \rightarrow R^{M \times N}$ where $f(I, D_{sparse}) = D_{dense}$
  - I.e. :

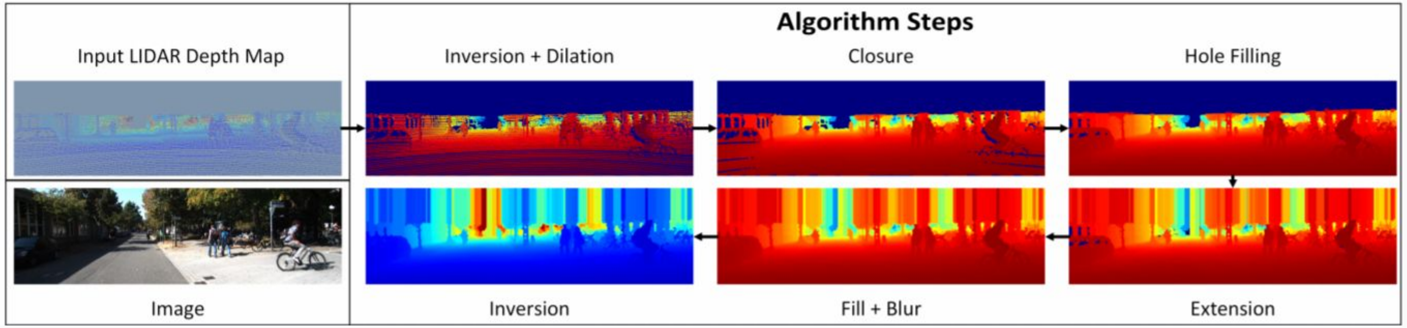
$$\min \| \mathbf{f}(I, D_{sparse}) - f(I, D_{sparse}) \|_F^2 = 0$$

  - Where:
    - $D_{dense}$ is the output dense depth map, and has the same size as $I$ and $D_{sparse}$
      - With empty values replaced by their depth estimate
  - * for non-guided depth completion, the formulation becomes independent of the image $I$



## PROPOSED ALGORITHM

- Tools:
  - Python, OpenCV, Numpy
- Idea: use larger pixel values to overwrite lower pixel values
  - → spasticity can be addressed by selecting appropriate operations to fill in empty pixels.
- 8 steps pipeline (Overview):
  1. Depth Inversion
  2. Custom Kernel Dilation
  3. Small Hole Closure
  4. Small Hole Fill
  5. Extension to Top of Frame
  6. Large Hole Fill
  7. Median & Gaussian Blur
  8. Depth Inversion



## PROPOSED ALGORITHM

1. Depth Inversion:
   a. OpenCV morphological transformation operation (main sparsity handling mechanisms)
   b. Overwrite smaller pixel values with larger ones
   c. Applying dilation operation on the original depth map would result in larger distances overwriting smaller distance
      i. I.e. loss of edge information
      ii. To resolve that, they modify the pixel depth's definition: $D_{inverted} = 100 - D_{input}$
2. Custom Kernel Dilation
   a. Start by filling empty pixels nearest to valid pixels, as they are more likely to share close depth values
      i. A custom kernel for an initial dilation of each valid depth pixel are designed by considering:
         1. Sparsity of projected points
         2. Structure of the LiDAR scan lines
      ii. I.e. the most likely pixels with the same values are dilated to the same value
   b. A 5 x 5 diamond kernel is used
3. Small Hole Closure:
   a. Step 2 doesn't fulfill all holes
      i. Since there are areas contain no depth values
   b. Idea after considering structure of objects in the environments:
      i. Nearby patches of dilated depths can be connected to form edges of objects
      ii. Done by a morphological close operation with a 5 x 5 full kernel
         1. Connect nearby depth values

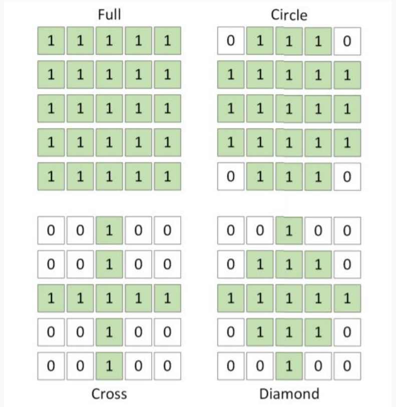


Figure 3: Different kernels used for comparison.

Presented by Yannis He

## PROPOSED ALGORITHM

1.
2.
3.
4. Small Hole Fill:
   a. After first two steps, there are still small to medium sized holes to be filled
   b. A mask of empty pixels is first calculated, followed by a 7 x 7 full kernel dilation operation
      i. → results in only the empty pixels is being filled while keeping valid pixel unchanged
5. Extension to Top of Frame:
   a. To account for tall objects outside the farm (ex. Trees, poles, buildings)
   b. The top value along each column is extrapolated to the top of the images, providing a denser depth map output
6. Large Hole Fill:
   a. The final filling step takes care of larger holes in the depth map that are not fully filled from previous steps
   b. The depth value are extrapolated from nearby values
   c. Dilation operation with a 31 x 31 full kernel is used to fill in any remaining empty pixels
7. Median & Gaussian Blur
   a. We now have a dense depth map. But the outliers exist in the depth map as a by-products of the dilation operation.
   b. To remove the outlier, we use a 5 x 5 kernel median blur (denoising)
      i. Removing outliers while maintaining local edges
8. Depth Inversion
   a. Revert back to the original depth encoding from the inverted depth values used in step 1:
      i. $D_{output} = 100 - D_{inverted}$

## RESULT

- The algorithm’s performance is tested on the depth completion task in the KITTI depth completion benchmark
  - Evaluated using the following metrics
    - Root Mean Squared Error (RMSE) → this metric is used to for the benchmark ranking
    - inverse Root Mean Squared Error (iRMSE)
    - Mean Average Error (MAE)
    - Inverse Mean Average Error (iMAE)
  - Performance:
    - Rank 1st on both RMSE & MAE metrics
    - Runs at 90 Hz on an Intel Core i7 7700K Process
      - 2nd place: 50 Hz on GPU
      - 3rd Place: 100 Hz on GPU
  - Due to the Euclidean calculation of the RMSE metric, a Gaussian blur reduces RMS errors significantly by minimizing the effect of outlier pixel depths

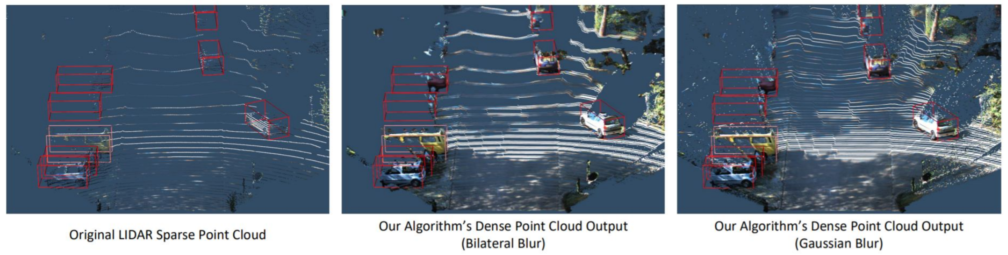

Original LIDAR Sparse Point Cloud
Our Algorithm’s Dense Point Cloud Output (Bilateral Blur)
Our Algorithm’s Dense Point Cloud Output (Gaussian Blur)

Figure 5: Qualitative results of the application of our algorithm on LIDAR point clouds using both bilateral and Gaussian blur kernels. Points have been colourized using the RGB image for easier visualization. It can be seen that ground truth object detection bounding boxes in **red** have much more points in the dense point cloud.