

Supplementary Material:

Perspective Plane Program Induction from a Single Image

The supplementary material is organized as follows. First, in Appendix Section **A**, we formally defines the domain specific language for perspective plane programs. Next, in Appendix Section **B**, we discuss the implementation details of neural painting networks (NPNs) used in this paper for image manipulation tasks. Finally, in Appendix Section **C**, we supplement more experimental results for camera pose estimation, repeated pattern detection, and image manipulation.

A. Domain Specific Language of Perspective Plane Programs

We formally define the domain specific language (DSL) of perspective plane programs in Table 1. A perspective plane program consists of the primitive `Draw` command which places objects at specified positions, (possibly nested) `For`-loop and `Rotate`-loop statements for characterizing structures, and `SetCameraPose` commands for set camera poses.

| | |
|----------------|--|
| Program | → CameraProgram;WorldProgram |
| CameraProgram | → SetCameraPose(rx=Real, ry=Real) |
| WorldProgram | → For1Stmt |
| For1Stmt | → For (i in range(Integer, Integer)) { For2Stmt Rotate2Stmt } |
| For2Stmt | → For (i in range(Integer, Integer)) { CondDrawStmt } |
| Rotate2Stmt | → For (i in range(ExprI, ExprI)) { RotateDrawStmt } |
| CondDrawStmt | → If (Expr ≥ 0) { CondDrawStmt } |
| CondDrawStmt | → DrawStmt |
| DrawStmt | → Draw (x=Expr, y=Expr) |
| RotateDrawStmt | → Draw (x=REExprX, y=REExprY) |
| Expr | → Real * i + Real * j |
| ExprI | → Real * i |
| REExprX | → Real * Cos(Real * j) * i + Real |
| REExprY | → Real * Sin(Real * j) * i + Real |

Table 1: The domain-specific language (DSL) of perspective plane programs. Language tokens `For`, `If`, `Integer`, `Real`, and arithmetic/logical operators follow the Python convention.

B. Implementation Details of Neural Painting Networks

The program description of an image provides us with many possible ways of partitioning the image into patches. The most straightforward way may be simply generating the Voronoi diagram with the peaks described by the program. Despite simple, this approach is sufficient to aid downstream image manipulation tasks (performed by neural networks) significantly, as we show in both the main text and this supplementary document.

On a high level, neural networks are known to be good at local pixel manipulations, but less so at understanding more global structure, such as regularity, present in the input image. The program description allows us to aggregate the patches into a “source patch stack” and also align them with the center of the patch to inpaint. By doing so, we reduce a task that originally required understanding the image’s global structure into one that requires only inpainting local pixels using a stack of aligned source patches.

Specifically, we adopt the neural painting network (NPN) proposed in [6], a neural network guided by the image program to perform pixel manipulation. For simple illustration, we discuss how NPNs utilize programs in image inpainting only for the

grid programs, but the framework can be easily extended to rotational programs by add a patch rotation step, in addition to the patch translation step.

Here we recap the essence of NPNs for completeness. Interested readers are referred to the original paper [6].

Patch aggregation. We first construct a stack of source patches, from which the NPNs can smartly copy from to inpaint the missing pixels. This is achieved by first generating the Voronoi diagram given the object (loosely defined by the inducted program) centroids, and then breaking down the image into patches according to this Voronoi diagram (“source patches”). We then translate each source patch so that their centers (given by the program) all align with the center of the patch to inpaint. These aligned source patches, together with the corrupted patch with missing pixels, get consumed by a convolutional network and smartly combined into a complete patch with missing pixels filled.

Architecture. An NPN is a convolutional network with a U-Net [8] encoder-decoder architecture. Because there are variable numbers of source patches across different input images, it handles an arbitrary number of source patches (as the memory permits). In addition, since the patches form an unordered set, the network is also designed to be invariant to their ordering, producing the same result for different input orders. These two properties are achieved by designs inspired by Aittala et al. [1] and Qi et al. [7], and we refer the reader to the original paper [6] for details. The input of the network is the stack of the corrupted input image plus source patches, and the output of the network is the inpainted image. A detailed printout of the generator’s architecture can be found in the supplemental document.

C. More Experiments and Results

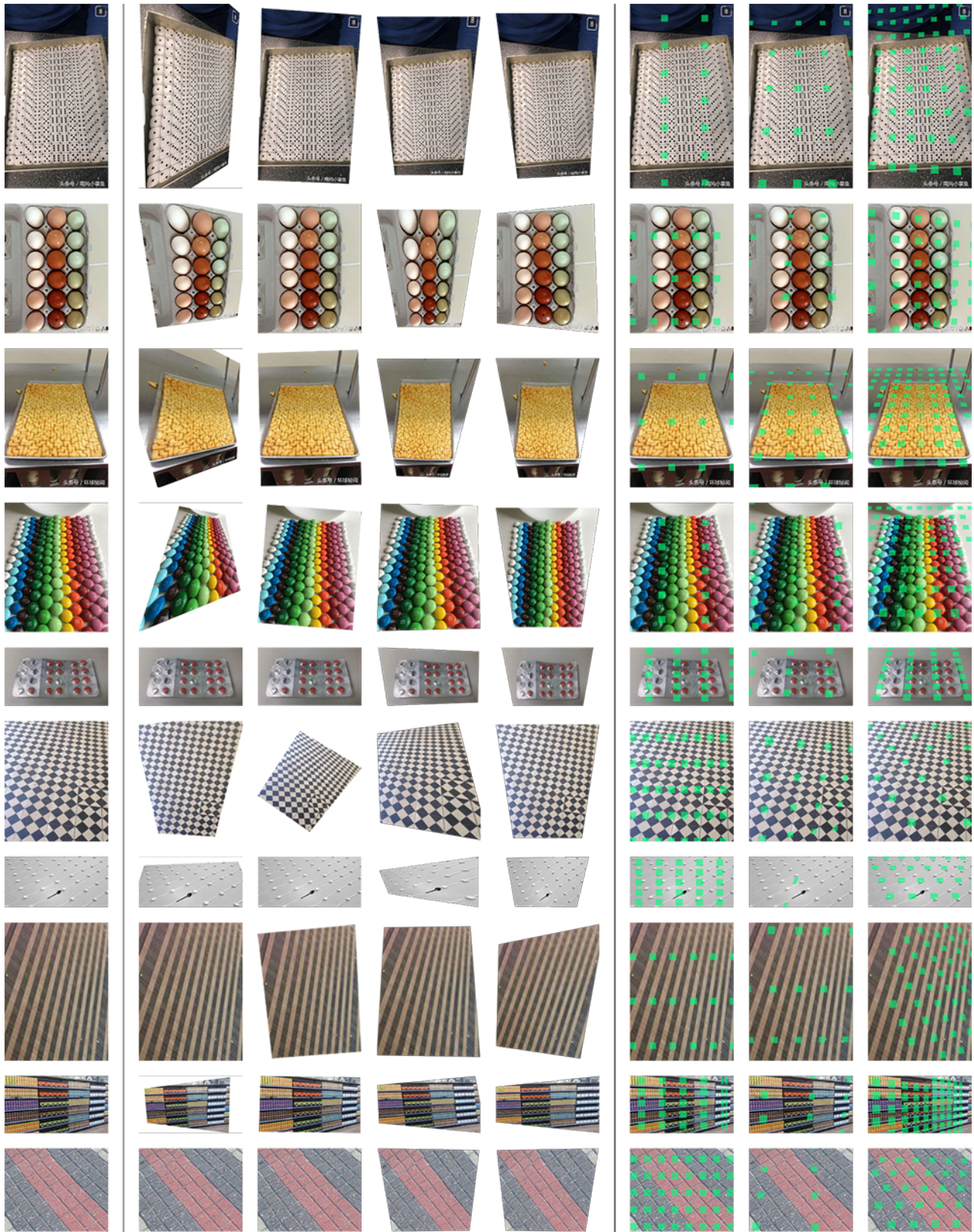
We show three groups of results in this section. First, in Fig. 1 we show the results of our camera pose estimation and repeated pattern detection. Our algorithm P3I outperforms both Lightroom and PlaneNet [5] in camera pose estimation. We find that for images with more salient straight line structures, Lightroom can produce more reasonable results. However, it still fails on most images in our dataset. In contrast, P3I succeeds on most images based on the global regularity cues. As for the repeated pattern detection, our model performs robust perspective-aware detection. Compared with it, the baseline RPD [4] fails when there is a large perspective angle. Adding a PlaneNet-based perspective correction does not show significant improvement (shown as RPD + PlaneNet).

Second, we provide more examples of the induced perspective plane programs from our dataset NRPP. This includes programs for both images with lattice patterns and ones with circular patterns. P3I infers these programs in a unified framework.

Finally, we show more results on the program-guided image inpainting for images with structural regularity and perspective. In general, PatchMatch [2] produces blurry results when operating on non-fronto-parallel images, because this breaks the stationary assumption used by PatchMatch. Both Image Quiting [3] and GatedConv [9] fail to preserve the structural regularities in the image.

References

- [1] Miika Aittala and Frédo Durand. Burst Image Deblurring Using Permutation Invariant Convolutional Neural Networks. In *ECCV*, 2018. 2
- [2] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan Goldman. PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing. *ACM TOG*, 28(3):24, 2009. 2, 7
- [3] Alexei A. Efros and William T. Freeman. Image Quilting for Texture Synthesis and Transfer. In *SIGGRAPH*, 2001. 2, 7
- [4] Louis Lettry, Michal Perdoch, Kenneth Vanhoey, and Luc Van Gool. Repeated Pattern Detection Using CNN Activations. In *WACV*, 2017. 2, 5
- [5] Chen Liu, Jimei Yang, Duygu Ceylan, Ersin Yumer, and Yasutaka Furukawa. PlaneNet: Piece-Wise Planar Reconstruction from a Single RGB Image. In *CVPR*, 2018. 2, 5
- [6] Jiayuan Mao, Xiuming Zhang, Yikai Li, William T. Freeman, Joshua B. Tenenbaum, and Jiajun Wu. Program-Guided Image Manipulators. In *ICCV*, 2019. 1, 2
- [7] Charles R. Qi, Li Yi, Hao Su, and Leonidas J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NeurIPS*, 2017. 2
- [8] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In *MICCAI*, 2015. 2
- [9] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Free-Form Image Inpainting with Gated Convolution. In *ICCV*, 2019. 2, 7



Input Images

AutoRectify

Lightroom

PlaneNet

P3I (Ours)

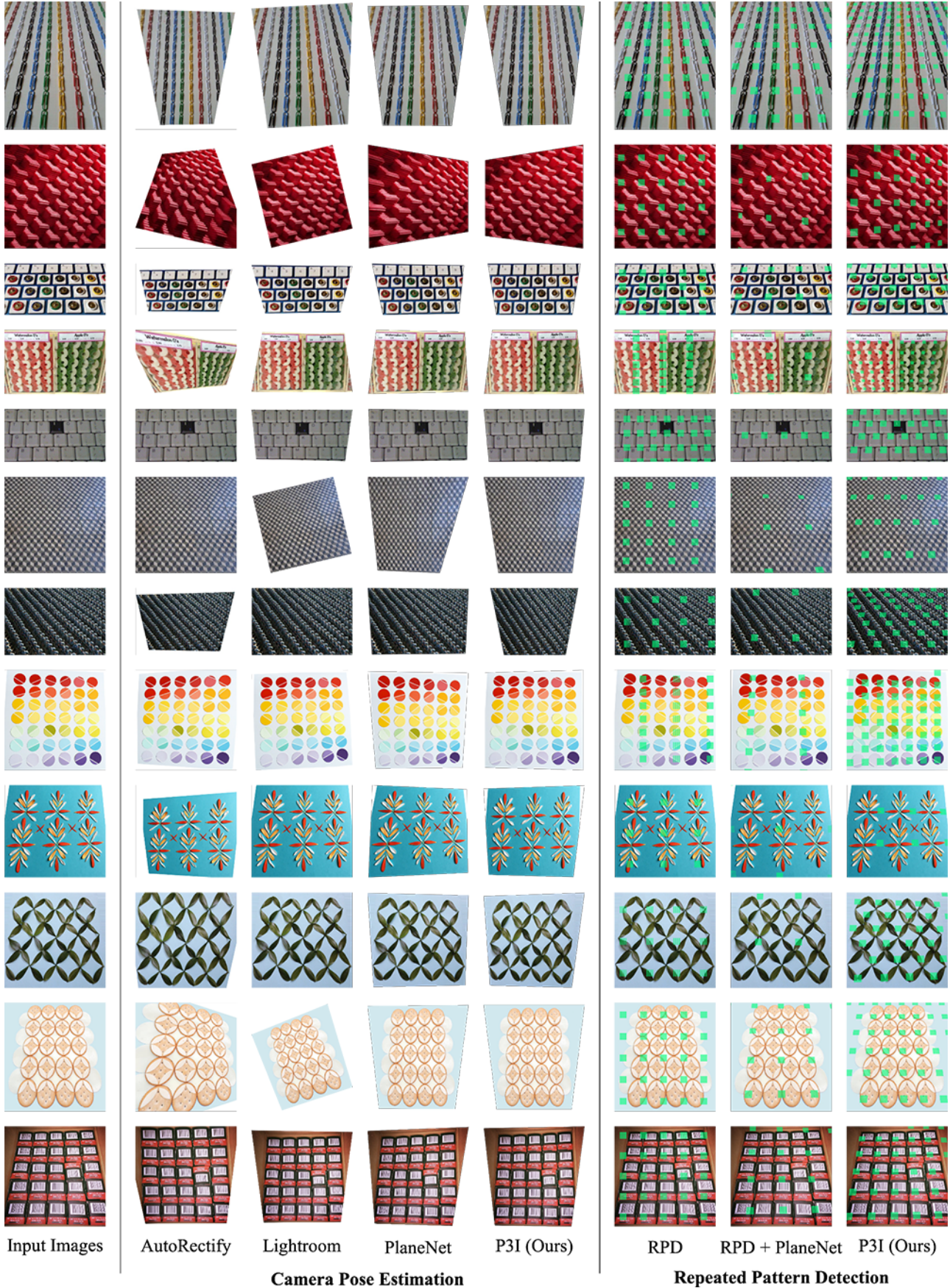
Camera Pose Estimation

RPD

RPD + PlaneNet

P3I (Ours)

Repeated Pattern Detection



Input Images

AutoRectify

Lightroom

PlaneNet

P3I (Ours)

Camera Pose Estimation

RPD

RPD + PlaneNet

P3I (Ours)

Repeated Pattern Detection

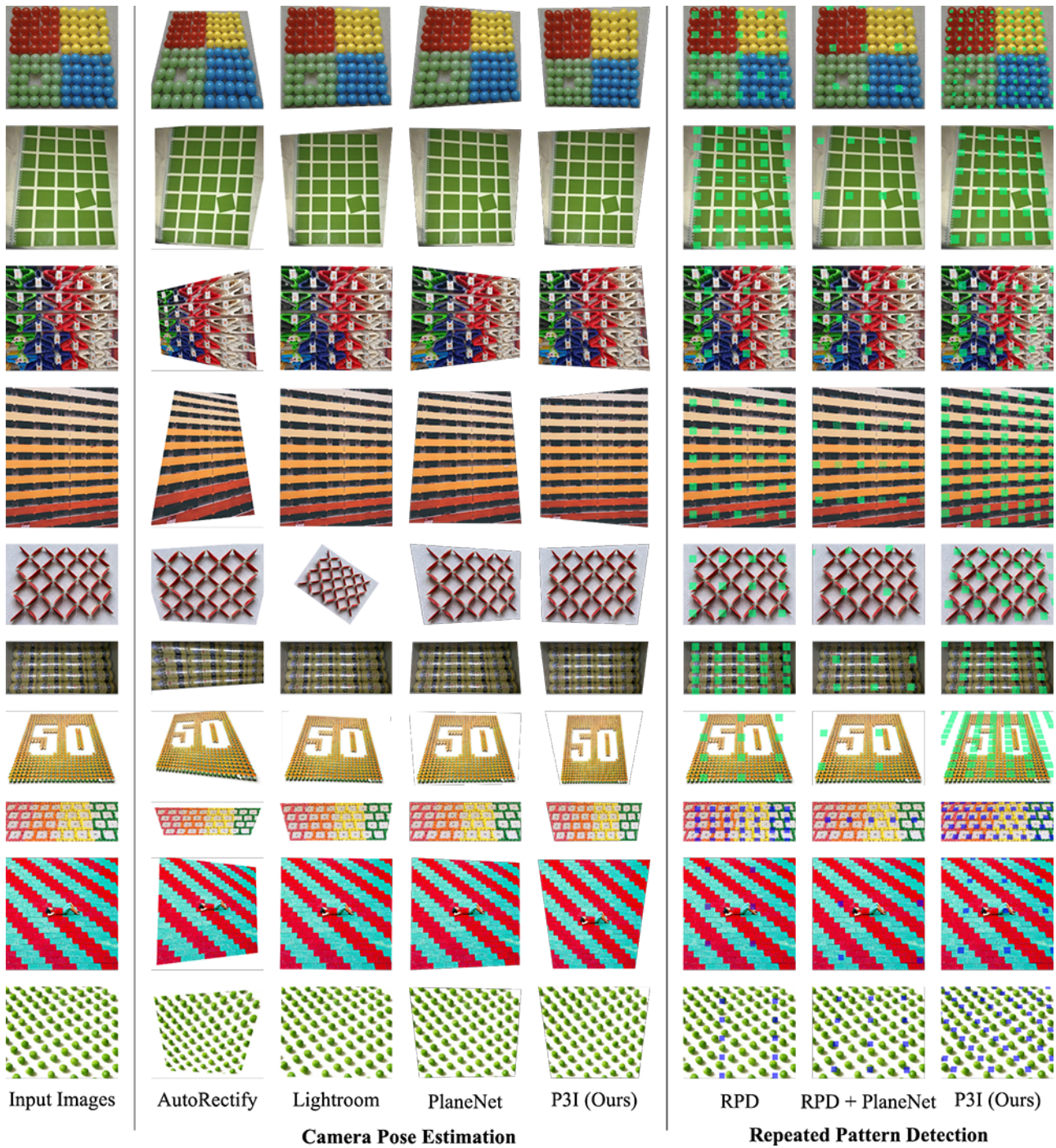
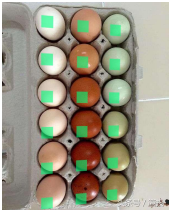

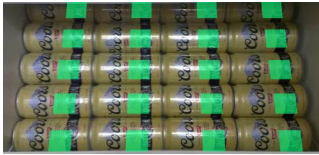





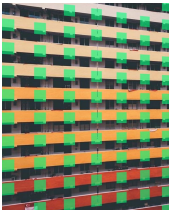





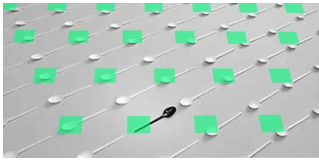







Figure 1: Qualitative results for camera pose estimation and repeated pattern detection. On the left, P3I outperforms both Lightroom and PlaneNet [5] on camera pose estimation. Results are visualized by performing a perspective correction based on the estimated parameters. On the right, we show that P3I can perform perspective-aware repeated pattern detection, which outperforms the baseline methods RPD [4] and RPD + PlaneNet.

(a) Image with objects placed in lattice patterns.

| | | | |
|---|---|--|--|
|  | <pre>SetCameraPose(rx = 3, ry = 15) for i in range(0, 6): for j in range(0, 3): draw( x = 11.38 * i, y = 11.38 * j)</pre> |  | <pre>SetCameraPose(rx = -24, ry = 0) for i in range(0, 5): for j in range(0, 4): draw( x = 26.31 * i, y = 13.54 * j)</pre> |
|  | <pre>SetCameraPose(rx = -27, ry = -3) for i in range(0, 12): for j in range(0, 12): draw( x = 7.00 * i, y = 8.00 * j)</pre> |  | <pre>SetCameraPose(rx = -36, ry = -3) for i in range(0, 4): for j in range(0, 9): draw( x = 18.00 * i + 9.69 * j, y = 16.15 * j)</pre> |
|  | <pre>SetCameraPose(rx = 0, ry = 15) for i in range(0, 10): for j in range(0, 6): draw( x = 9.31 * i, y = 7.00 * j)</pre> |  | <pre>SetCameraPose(rx = -42, ry = 0) for i in range(0, 8): for j in range(0, 7): draw( x = 19.00 * i + 6.00 * j, y = 9.92 * j)</pre> |
|  | <pre>SetCameraPose(rx = -27, ry = 3) for i in range(0, 5): for j in range(0, 5): draw( x = 10.69 * i, y = 12.15 * j)</pre> |  | <pre>SetCameraPose(rx = -45, ry = 3) for i in range(0, 4): for j in range(0, 7): draw( x = 21.62 * i + 12.69 * j, y = 22.62 * j)</pre> |
|  | <pre>SetCameraPose(rx = -18, ry = 3) for i in range(0, 10): for j in range(0, 10): draw( x = 6.00 * i, y = 6.00 * j)</pre> |  | <pre>SetCameraPose(rx = 24, ry = 0) for i in range(0, 6): for j in range(0, 9): draw( x = 9.00 * i, y = 9.23 * j)</pre> |

(b) Image with objects placed in circle patterns.

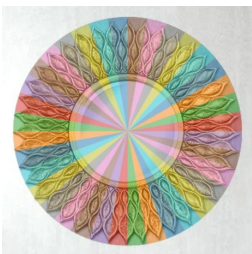

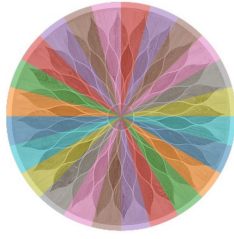

| | | | |
|---|--|--|--|
|  | <pre>SetCameraPose(rx = 0, ry = 0) for i in range(0, 36): draw( x = 28.0 + 13.5 * cos(2π / 36 * j), y = 28.0 + 13.5 * sin(2π / 36 * j))</pre> |  | <pre>SetCameraPose(rx = 0, ry = 0) for i in range(0, 25): draw( x = 28.0 + 13.5 * cos(2π / 25 * j), y = 28.0 + 13.5 * sin(2π / 25 * j))</pre> |
|---|--|--|--|

Figure 2: Examples of the induced perspective plane programs from our dataset NRPP. We overlay the centroids of the repeated objects as green squares on the original images for better visualization. We show the induced programs for both images with lattice patterns and images with circular patterns.

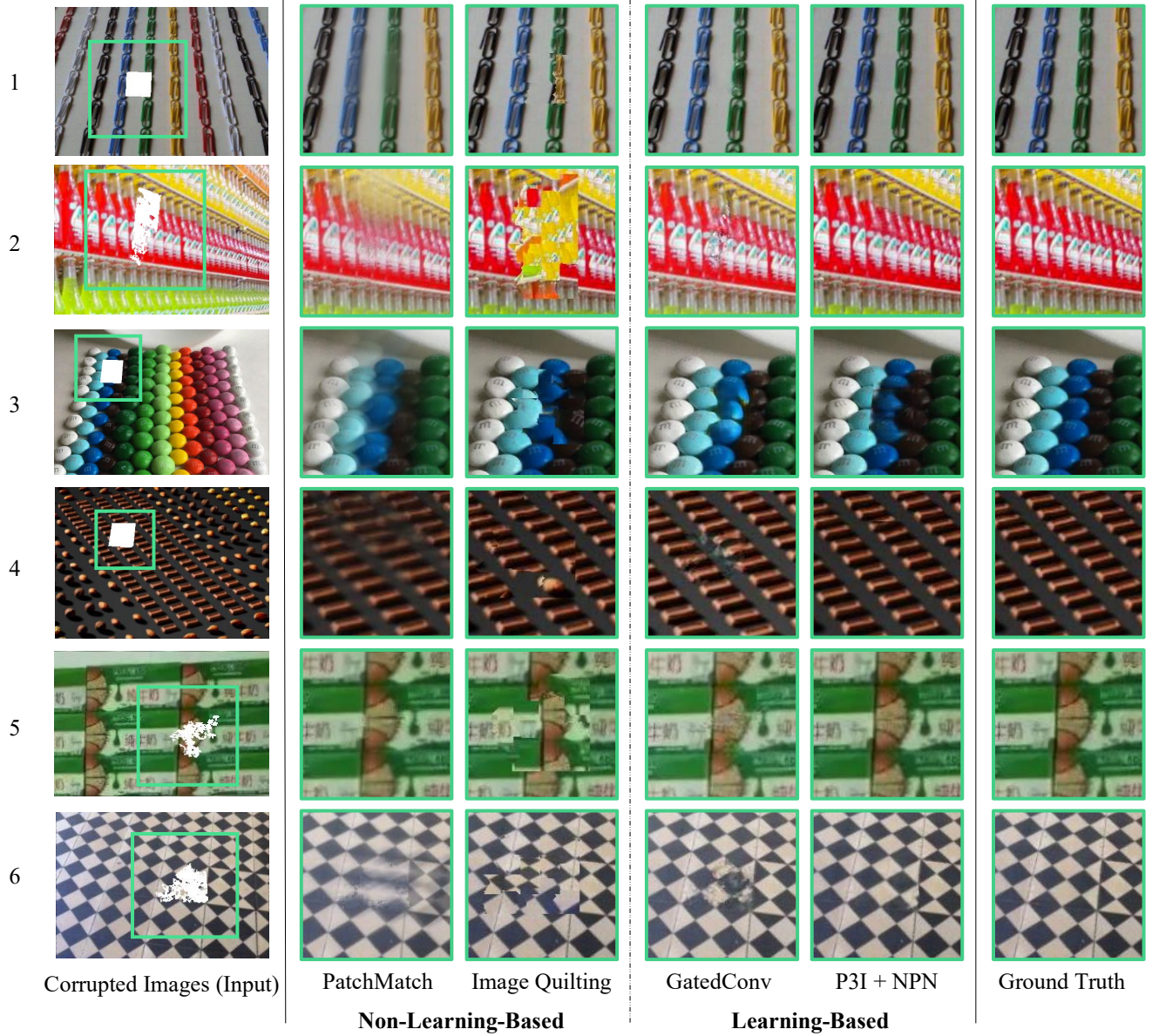


Figure 3: P3I-guided NPNs perform image inpainting in a perspective-aware fashion. Results produced by NPNs are sharp and consistent with the global structure. We compare our results with both non-learning-based methods (PatchMatch [2] and Image Quilting [3]) and a learning-based method (GatedConv [9]).