Toward Geometric Deep SLAM

Daniel DeTone
Magic Leap, Inc.
Sunnyvale, CA
ddetone@magicleap.com

Tomasz Malisiewicz
Magic Leap, Inc.
Sunnyvale, CA
tmalisiewicz@magicleap.com

Andrew Rabinovich
Magic Leap, Inc.
Sunnyvale, CA
arabinovich@magicleap.com

Abstract: We present a point tracking system powered by two deep convolutional neural networks. The first network, MagicPoint, operates on single images and extracts salient 2D points. The extracted points are "SLAM-ready" because they are by design isolated and well-distributed throughout the image. We compare this network against classical point detectors and discover a significant performance gap in the presence of image noise. As transformation estimation is more simple when the detected points are geometrically stable, we designed a second network, MagicWarp, which operates on pairs of point images (outputs of MagicPoint), and estimates the homography that relates the inputs. This transformation engine differs from traditional approaches because it does not use local point descriptors, only point locations. Both networks are trained with simple synthetic data, alleviating the requirement of expensive external camera ground truthing and advanced graphics rendering pipelines. The system is fast and lean, easily running 30+ FPS on a single CPU.

Keywords: Deep Learning, SLAM, Tracking, Geometry, Augmented Reality

1 Introduction

Much of deep learning success in computer vision tasks such as image categorization and object detection stems from the availability of large annotated databases like ImageNet and MS-COCO. However, for SLAM-like pose tracking and reconstruction problems, there instead exists a fragmented ecosystem of smaller device-specific datasets such as the Freiburg-TUM RGBD Dataset [1] based on the Microsoft Kinect, the EuRoC drone/MAV dataset [2] based on stereo vision cameras and IMU, and the KITTI driving dataset [3].

We frequently ask ourselves: *what would it take to build an ImageNet for SLAM?* Obtaining accurate ground-truth pose measurements for a large number of environments and scenarios is difficult. Getting accurate alignment between ground-truthing sensors and a standard set of Visual SLAM sensors takes significant effort; it is expensive and is difficult to scale across variations in cameras. Category labeling in a crowd-sourced or pay-per-label Amazon Mechanical Turk fashion, as is commonly done for ImageNet-like datasets, suddenly seems a lot more fun.

Photorealistic rendering is potentially useful, as all relevant geometric variables for SLAM tasks can be recorded with 100% accuracy. Benchmarking SLAM on photorealistic sequences makes a lot of sense, but training on such rendered images often suffers from domain adaptation issues. Our favorite deep nets seem to overfit. Datasets have been created with this intent in mind, but as the research community always demands results on real-world datasets, the benefit of photorealistic rendering for automatic SLAM training is still a dream. Since a public ImageNet-scale SLAM dataset does not exist today and photorealistic rendering brings its own set of new problems, how are we to embrace the data-driven philosophy of deep learning while building an end-to-end Deep SLAM system? Our proposed solution comes from a couple of key insights.

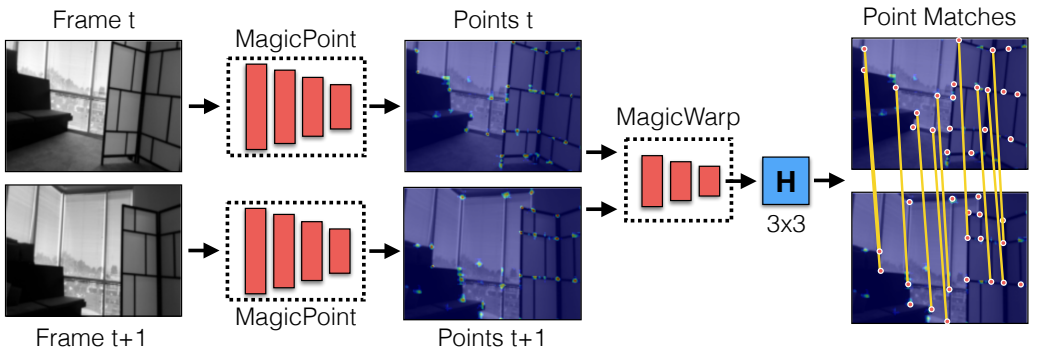

Figure 1: Deep Point-Based Tracking Overview. Pairs of images are processed by a convolutional neural network called MagicPoint which is trained to detect salient corners in the image. The resulting point images are then processed together by MagicWarp (another convolutional neural network) to compute a homography H which relates the points in the input images.

First, recent ego-motion estimation has shown that it is possible to train deep convolutional neural networks on the task of image prediction. Compared to direct supervision (i.e., regressing to the ground truth 6 DoF pose), supervision for frame prediction comes "for free." This new insight that the move away from strong-supervision might bear more fruits is rather welcoming for SLAM, an ecosystem already plagued with a fragmentation of datasets. Systems such as [4] perform full frame prediction, and we will later see that our flavor of the prediction problem is more geometric, as we focus on *geometric consistency*.

Second, SLAM models must be lean or they will not run at a large scale on embedded platforms such as those in robotics and augmented reality. Our desire to focus on geometric consistency as opposed to full frame prediction comes from a dire need to deploy such systems in production. While it is fulfilling to watch full frame predictions made by a deep learning system, we already known from previous successes in SLAM (e.g., [5] and [6]) that predicting/aligning points is sufficient for metric-level pose recovery. So why solve a more complex full frame prediction task than is necessary for SLAM?

2 Related Work

Individual components of SLAM systems have recently been tackled with supervised deep learning methods. The feature point detection and description stage was tackled in the work of [7], where a convolutional neural network was trained using image patches filtered through a classical Structure From Motion pipeline. Transformation estimation was shown to be done successfully by CNNs in [8], where a deep network was trained on a large dataset of warped natural images. The transformation estimation done in this work was direct, meaning that the convolutional neural network directly mapped pairs of images to their transforms. The work of [9] tackled dense optical flow. The problem of camera localization was also tackled with a CNN in [10], where a network was trained to learn a mapping from images to absolute 6DOF poses. A deep version of the RANSAC algorithm was presented in [11], where a deep network was trained to learn a robust estimator for camera localization. There have also been many works such as [12] in the direction of deep relocalization via metric learning, where two images with similar pose are mapped to a similar point on an embedding produced by a deep network.

A series of works have also tackled multiple SLAM problems concurrently. In [13], a CNN was trained to simultaneous estimate the depth and motion of a monocular camera pair. Interestingly, the models did not work well when the network was trained on the two tasks separately, but worked much better when trained jointly. This observation is consistent with natural regularization of multi task learning. This approach relies on supervised training data for both motion and depth cues.

There is also a new direction of research at the intersection of deep learning and SLAM where no or very few ground truth measurements are required. By formulating loss functions to maximize photometric consistency, the works of [4] and [14] showed an ego-motion and depth estimation system based on image prediction. Work in this direction show that there is much promise in moving away from strong supervision.

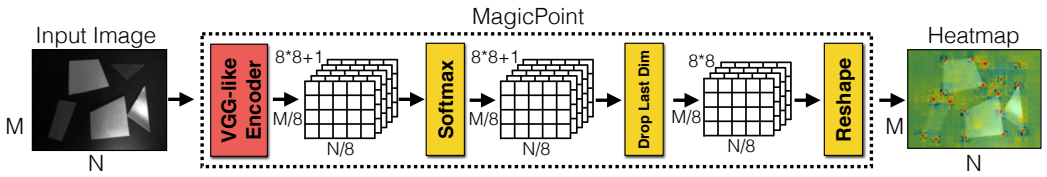

Figure 2: MagicPoint architecture. The MagicPoint network operates on grayscale images and outputs a "point-ness" probability for each pixel. We use a VGG-style encoder combined with an explicit decoder. Each spatial location in the final 15x20x65 tensor represents a probability distribution over a local 8x8 region plus a single dustbin channel which represents no point being detected $(8 * 8 + 1 = 65)$. The network is trained using a standard cross entropy loss, using point supervision from the 2D shape renderer (see examples in Figure 3).

3 Deep Point-Based Tracking Overview

The general architecture of our Deep Point-Based Tracking system is shown in Figure 1. There are two convolutional neural networks that perform the majority of computation in the tracking system: MagicPoint and MagicWarp. We discuss these two models in detail below.

3.1 MagicPoint Overview

MagicPoint Motivation. The first step in most sparse SLAM pipelines is to detect stable 2D interest point locations in the image. This step is traditionally performed by computing corner-like gradient response maps such as the second moment matrix [15] or difference of Gaussians [16] and detecting local maxima. The process is typically repeated at various image scales. Additional steps may be performed to evenly distribute detections throughout the image, such as requiring a minimum number of corners within an image cell [6]. This process typically involves a high amount of domain expertise and hand engineering, which limits generalization and robustness. Ideally, interest points should be detected in high sensor noise scenarios and low light. Lastly, we should get a confidence score for each point we detect that can be used to help reject spurious points and up-weigh confident points later in the SLAM pipeline.

MagicPoint Architecture. We designed a custom convolutional network architecture and training data pipeline to help meet the above criteria. Ultimately, we want to map an image I to a point response image P with equivalent resolution, where each pixel of the output corresponds to a probability of "corner-ness" for that pixel in the input. The standard network design for dense prediction involves an encoder-decoder pair, where the spatial resolution is decreased via pooling or strided convolution, and then upsampled back to full resolution via upconvolution operations, such as done in [17]. Unfortunately, upsampling layers tend to add a high amount of compute, thus we designed the MagicPoint with an explicit decoder[1] to reduce the computation of the model. The convolutional neural network uses a VGG style encoder to reduce the dimensionality of the image from 120x160 to 15x20 cell grid, with 65 channels for each spatial position. In our experiments we chose the QQVGA resolution of 120x160 to keep the computation small. The 65 channels correspond to local, non-overlapping 8x8 grid regions of pixels plus an extra dustbin channel which corresponds to no point being detected in that 8x8 region. The network is fully convolutional, using 3x3 convolutions followed by BatchNorm normalization and ReLU non-linearity. The final conv layer is a 1x1 convolution and more details are shown in Figure 2.

MagicPoint Training. What parts of an image are interest points? They are typically defined by computer vision and SLAM researchers as uniquely identifiable locations in the image that are stable across a variety of viewpoint, illumination, and image noise variations. Ultimately, when used as a preprocessing step for a Sparse SLAM system, they must detect points that work well for a given SLAM system. Designing and choosing hyper parameters of point detection algorithms requires expert and domain specific knowledge, which is why we have not yet seen a single dominant point extraction algorithm persisting across many SLAM systems.

There is no large database of interest point labeled images that exists today. To avoid an expensive data collection effort, we designed a simple renderer based on available OpenCV [19] functions. We render simple geometric shapes such as triangles, quadrilaterals, stars, lines, checkerboards, 3D

[1] Our decoder has no parameters, and is known as "sub-pixel convolution" [18] or "depth to space" inside TensorFlow.

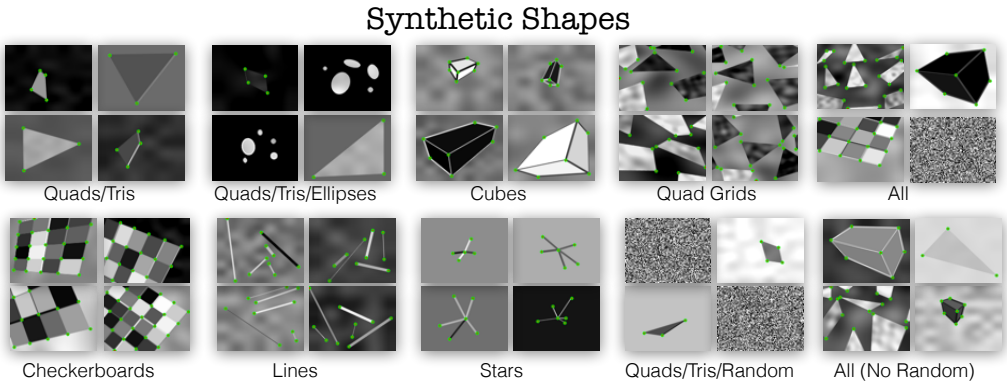

Figure 3: Synthetic Shapes Dataset. The `Synthetic Shapes` dataset consists of rendered triangles, quadrilaterals, lines, cubes, checkerboards, and stars each with ground truth corner locations. It also includes some negative images with no ground truth corners, such as ellipses and random noise images.

cubes, ellipses and random noise. For each image we know the ground truth corner locations. See Figure 3 for examples from our synthetic shapes renderer. Note the 2D ground truth locations need not correspond to local, high-gradient intersections of edges in the image, but can instead correspond to other low-level cues which require a larger local receptive field. We also trained MagicPoint networks that detect non-corner interest points such ellipse centers, 2D polygon face centers, and midpoints along edges. For simplicity, we only train on corners in this paper.

Once the shapes are rendered, we apply homographic warping to each image to augment the number of training examples and we apply high amounts noise in the form of brightness changes, shadows, blurring, Gaussian noise, and speckle noise. See Figure 8 for examples of the noise applied during training. The data is generated on the fly and no example is seen by the network twice. The network is trained using a standard cross entropy loss after the logits for each cell in the 15x20 grid are passed through a softmax function.

3.2 MagicWarp Overview

MagicWarp Motivation. Our second network, MagicWarp, produces a homography given a pair of point images as produced by Magic Point. Once the homography is computed, the points in one image are transformed into the other and the point correspondence is computed by assigning correspondence to close neighbors. In doing so, MagicWarp estimates correspondence in image pairs without interest point descriptors. Once correct correspondences are established, it is straightforward to compute 6DOF relative pose Rs and ts using either a homography matrix decomposition for planar scenes or a fundamental matrix decomposition for non-planar scenes, assuming the camera calibration matrix K is known. By designing the network to operate on (the space of point images $\times$ the space of relative poses) instead of (the space of all images $\times$ the space of relative poses), we do not have to worry about illumination, shadows, and textures. We no longer rely on photometric consistency assumption to hold. Plus, by reducing the problem dimensionality, the transformation estimation model can be small and efficient.

MagicWarp Architecture. MagicWarp is designed to operate directly on the point detections outputs from MagicWarp (although it can operate on any traditional point detector). We found that the model works well on pairs of the semi-dense 15x20x65 images. At this small spatial resolution the network uses very little compute. After channel-wise concatenation of the inputs to form an input of size 15x20x130, there is a VGG style encoder consisting of 3x3 convolutions, max-pooling, BatchNorm and ReLU activations, followed by two fully connected layers which output the 9 values of the 3x3 homography H. See Figure 4 for more details. Note that MagicWarp can be applied iteratively, by using the network's first predicted H_1, applying it to one of the inputs, and computing a second H_2, yielding a final $H = H_1 * H_2$, which improves results. For simplicity we do not apply MagicWarp iteratively in this paper.

MagicWarp Training. To train the MagicWarp network, we generate millions of examples of point clouds rendered into two virtual cameras. The point clouds are generated from simple 3d

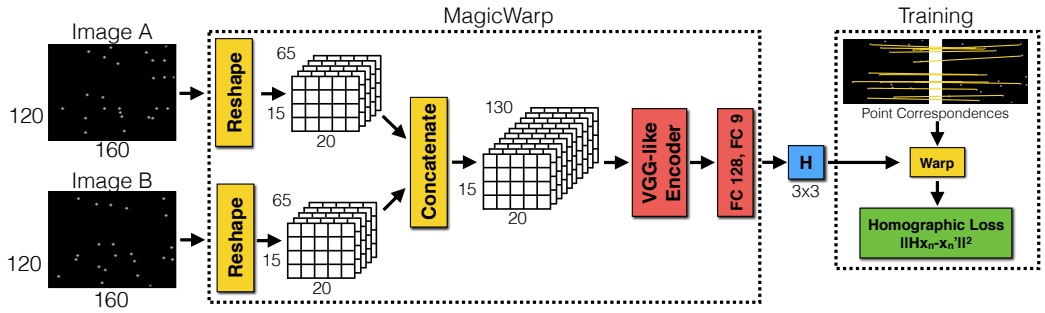

Figure 4: MagicWarp architecture. Pairs of binary point images are concatenated and then fed through a standard VGG-style encoder. The 3x3 homography H is output by a fully connected layer. H is then normalized such that its bottom right element is one. The loss is computed by warping points with known correspondence from one image into the other and measuring their distance to the ground truth correspondences.

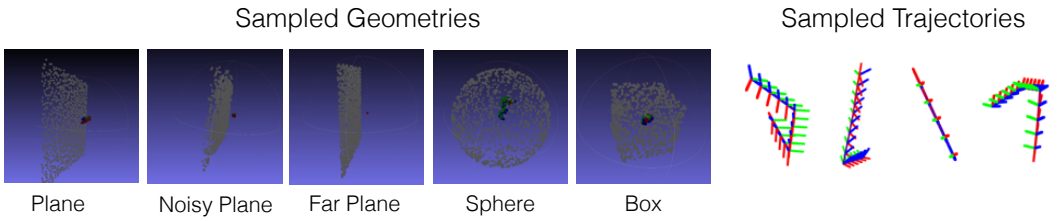

Figure 5: MagicWarp data generation. To generate 2D point set pairs, we create 3D point clouds of 3D geometries and render them to virtual cameras governed by simple 3D trajectories.

geometries, such as planes, spheres and cubes. The positions of the two virtual cameras are sampled from random trajectories which consist of piece-wise linear translation and rotations around random axes, as shown in Figure 5. We randomly sample camera pairs which have at least 30% visual overlap. Once the points are projected into the two camera frames, we apply point input dropout, to improve the network's robustness to spurious and missing point detections. We found that randomly dropping 50% of the matches and randomly dropping 25% of the points independently works well.

The loss is computed by measuring the Euclidean distance between the correct matches. For N matches in the left point image, each point x_n is multiplied by the the predicted H and compared to its match location in the right point image x'_n, shown in Equation 1.

$$L_{MagicWarp} = \sum_{n=1}^{N} \|Hx_n - x'_n\|^2 \tag{1}$$

We found that care must be taken to train the network to directly output the 3x3 matrix. Training worked best when the final FC layer bias is initialized to output the identity matrix, when the coordinates of the homography H are normalized to the range $[-1, 1]$, and when the H quantity is normalized such that the bottom right element is one, since the homography H has eight degrees of freedom and nine elements.

4 MagicPoint Evaluation

We evaluate the MagicPoint component of our system against traditional corner detection baselines like the FAST [20] corner detector, the Harris [15] corner detector, and the "Good Features to Track" or Shi [21] corner detector. For a thorough evaluation of classical corner detectors, see [22]. The deep baselines are a small (MagicPointS, 81KB) and large version (MagicPointL, 3.1MB) of MagicPoint where the larger version was trained with corner-related sidetasks and a bigger network.

The detectors are evaluated on both synthetic and real image data. Both types of data consists of simple geometry that a human could easily label with the ground truth corner locations. While

one could not build a fully functional SLAM system based on detectors which only work in these scenarios, we expect a good point detector to easily detect the correct corners in these scenarios. The added benefit of images with ground truth corner locations is that we can more rigorously analyze detector performance. In fact, we were surprised at how difficult the simple geometries were for the classical point detectors.

4.1 Evaluation Measures

Corner Detection Average Precision. We compute Precision-Recall curves and the corresponding Area-Under-Curve (also known as Average Precision), the pixel location error for correct detections, and the repeatability rate. For corner detection, we use a threshold $\varepsilon = 4$ to determine if a returned point location $\mathbf{x}$ is correct relative to a set of K ground-truth corners $\{\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_K\}$. We define the correctness as follows:

$$\texttt{Corr}(\mathbf{x}) = (\min_j ||\mathbf{x} - \hat{\mathbf{x}}_j||) \leq \varepsilon \tag{2}$$

The precision recall curve is created by varying the detection confidence and summarized with a single number, namely the Average Precision (which ranges from 0 to 1), and larger AP is better.

Corner Localization Error. To complement the AP analysis, we compute the corner localization error, but solely for the correct detections. We define the Localization Error as follows:

$$\texttt{LE} = \frac{1}{N} \sum_{i:\texttt{Corr}(\mathbf{x}_i)} \min_{j \in \{1,\ldots,K\}} ||\mathbf{x}_i - \hat{\mathbf{x}}_j|| \tag{3}$$

The Localization Error is between 0 and ε, and lower LE is better.

Repeatability. We compute the repeatability rate, which is the probability that a point gets detected in the next frame. We compute sequential repeatability (between frame t and $t+1$ only). For repeatability, we also need a notion of correctness that relies on a pixel distance threshold. We use $\varepsilon = 2$ for the threshold between points. Let's assume we have N_1 points in the first image and N_2 points in the second image. We define correctness for repeatability experiments as follows:

$$\texttt{Corr}(\mathbf{x}_i) = (\min_{j \in \{1,\ldots,N_2\}} ||\mathbf{x}_i - \hat{\mathbf{x}}_j||) \leq \varepsilon \tag{4}$$

Repeatability simply measures the probability that a point is detected in the second image.

$$\texttt{Rep} = \frac{1}{N_1 + N_2} (\sum_i \texttt{Corr}(\mathbf{x}_i) + \sum_j \texttt{Corr}(\mathbf{x}_j)) \tag{5}$$

For each sequence of images, we want to create a single scalar Repeatability number. We first create a Repeatability vs Number of Detections curve, then find the point of maximum repeatability. When summarizing repeatability with a single number, we use the point of maximum repeatability, and report the result as repeatability@N where N is the average number of detections at the point of maximum repeatability.

4.2 Results on Synthetic Shapes Dataset

We created an evaluation dataset with our synthetic shapes generator to determine how well our detector is able to localize simple corners. There are 10 categories of images, shown in Figure 3.

Mean Average Precision and Mean Localization Error. For each category, there are 1000 images sampled from the synthetic shapes generator. We compute Average Precision and Localization Error with and without added imaging noise. A summary of the per category results are shown in Figure 6 and the mean results are shown in Table 1. The MagicPoint detectors outperform the classical detectors in all categories and in the mean. There is a significant performance gap in mAP in all categories in the presence of noise.

Effect of Noise Magnitude. Next we study the effect of noise more carefully by varying its magnitude. We were curious if the noise we add to the images is too extreme and unreasonable for a point detector. To test this hypothesis, we linearly interpolate between the clean image ($s = 0$) and the noisy image ($s = 1$). To push the detectors to the extreme, we also interpolate between the noisy

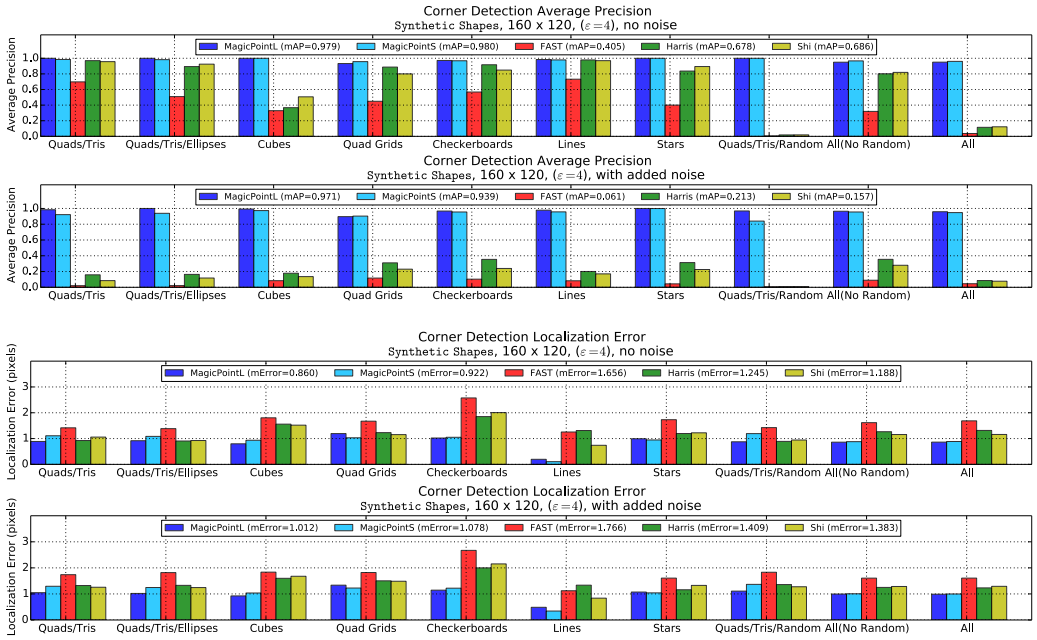

Figure 6: Synthetic Shapes Results Plot. These plots report Average Precision and Corner Localization Error for each of the 10 categories in the `Synthetic Shapes` dataset with and without noise. The sequences with "Random" inputs are especially difficult for the classical detectors.

Metric	Noise	MagicPointL	MagicPointS	FAST	Harris	Shi
mAP	no noise	0.979	**0.980**	0.405	0.678	0.686
mAP	noise	**0.971**	0.939	0.061	0.213	0.157
MLE	no noise	**0.860**	0.922	1.656	1.245	1.188
MLE	noise	**1.012**	1.078	1.766	1.409	1.383

Table 1: Synthetic Shapes Table Results. Reports the mean Average Precision (mAP, higher is better) and Mean Localization Error (MLE, lower is better) across the 10 categories of images on the Synthetic Shapes dataset. Note that MagicPointL and MagicPointS are relatively unaffected by imaging noise.

image and random noise ($s = 2$). The random noise images contain no geometric shapes, and thus produce an mAP score of 0.0 for all detectors. An example of the varying degree of noise and the plots are shown in Figure 7.

Effect of Noise Type. We categorize the noise we apply into eight categories. We study the effect of each of these noise types individually to better understand which has the biggest effect on the point detectors. Speckle noise is particularly difficult for traditional detectors. Results are summarized in Figure 8.

4.3 Results on 30 Static Corners Dataset

We next evaluate MagicPoint on real data. We chose scenes with simple geometry so that the ground truth corner locations can be easily labeled by a human. These sequences are about 1-2 minutes in length and are recorded using a static, commodity webcam. Since the camera is static, we only label the first frame with ground truth corner locations and propagate the labels to all the other frames in the sequence. Throughout each sequence, we vary the lighting conditions using a hand-held point source light and overall room lighting.

Mean Average Precision, Mean Localization Error and Repeatability. For each of the 30 sequences in the dataset, we compute Average Precision, Localization Error and Repeatability (metrics are described in detail in Section 4.1) with and without noise. The results are broken down by corner category in Figure 10. We are able to measure Repeatability in this dataset because we now have a sequence of images viewing the same scene in each frame. We see a similar story as we

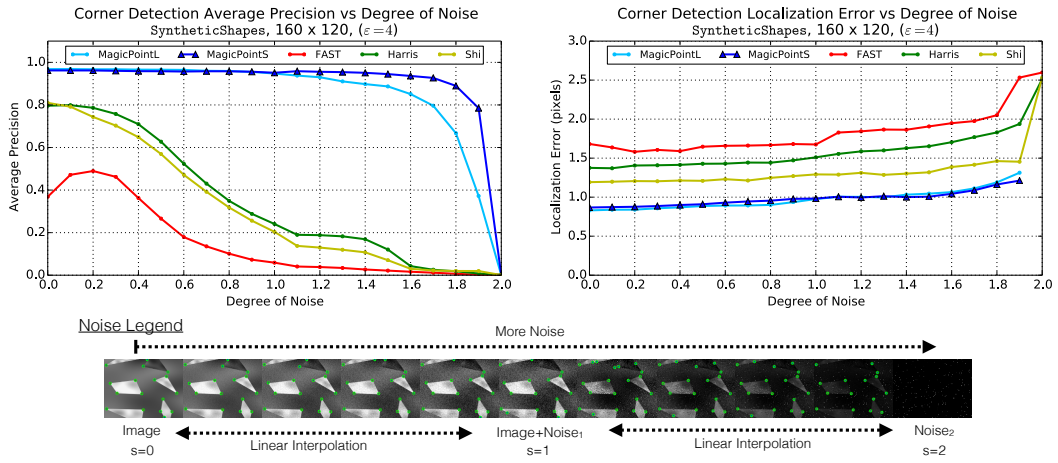

Figure 7: Synthetic Shapes Effect of Noise Magnitude. Two versions of MagicPoint are compared to three classical point detectors on the `Synthetic Shapes` dataset (shown in Figure 3). The MagicPoint models outperform the classical techniques in both metrics, especially in the presence of image noise.

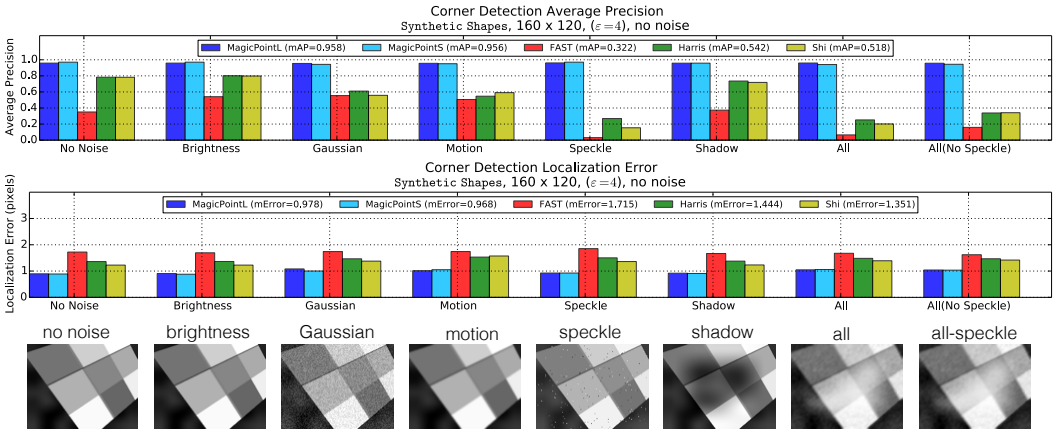

Figure 8: Synthetic Shapes Effect of Noise Type. The detector performance is broken down by noise category. Speckle noise is particularly difficult for traditional detectors.

did in the `Synthetic Shapes` evaluation. The MagicPoint detectors detect more corners more confidently and with better localization, especially in the presence of noise. The corners are also more repeatable across frames, showing their robustness to lighting variation.

Effect of Image Size. The experiments reported above were conducted at a 160x120 image resolution, which is the same resolution that we used to train the MagicPoint detector. This resolution is smaller than the resolution used in most modern SLAM systems and probably smaller than what the classical detectors were designed for. Since the MagicPoint detector is a fully-convolutional model, we can run it at different input resolutions. We repeated the above `30 Static Corners` experiments at the 320x240 resolution, and report the results in Table 2. The MagicPointL detector outperforms the other models in every setting except for no noise localization error, in which the Harris detector scores the best. However, in the presence of noise, MagicPointL and MagicPointS score the best. It is expected for manually designed corner detectors to work best in ideal conditions, however, engineered feature detectors prove to be too brittle in the presence of noise.

Compute Analysis. For an input image size 160x120, the average forward pass times on a single CPU for MagicPointS and MagicPointL are 5.3ms and 19.4ms respectively. For an input image size of 320x240, the average forward pass times on a single CPU for MagicPointS and MagicPointL are 38.1ms and 150.9ms respectively. The times were computed with BatchNorm layers folded into the convolutional layers.

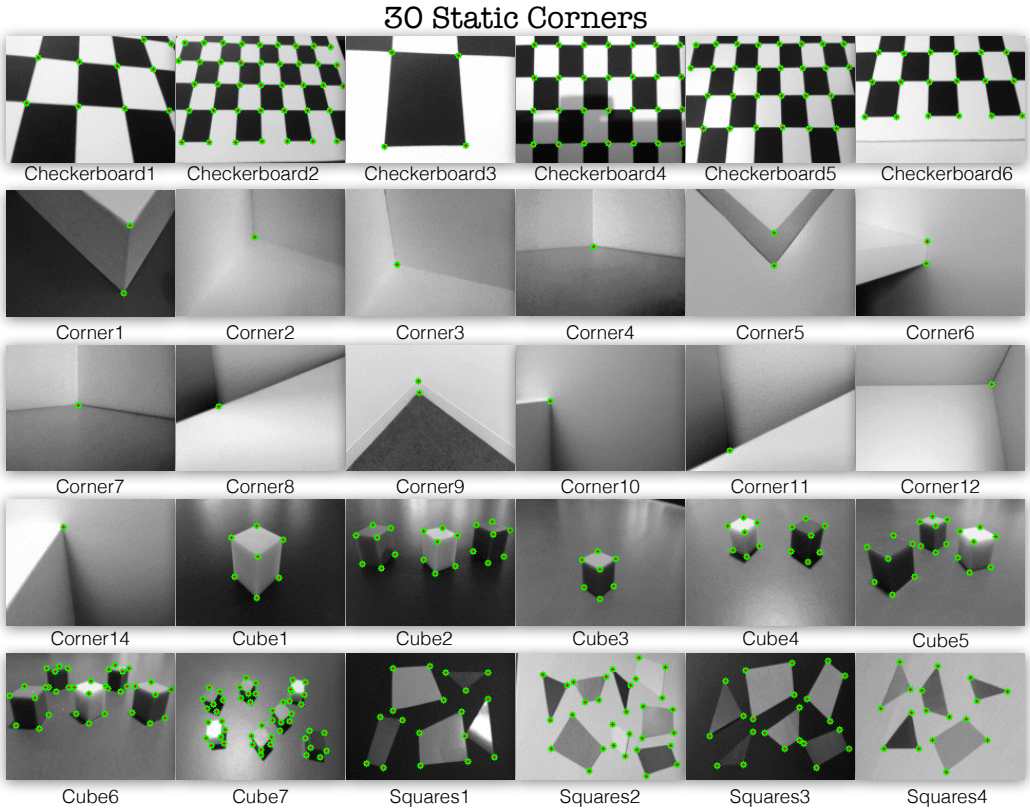

Figure 9: 30 Static Corners Dataset. We show example frames for each sequence in the `30 Static Corners` dataset alongside the ground truth corner locations. The sequences come from four different categories: checkerboards, isolated corners, cubes, and squares.

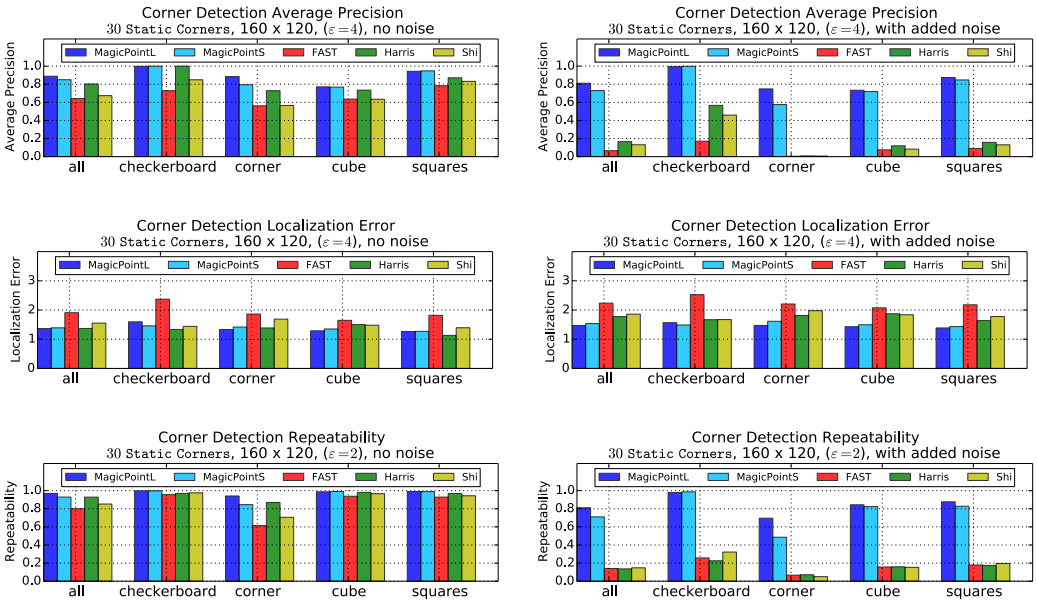

Figure 10: 30 Static Corners Results Plots. We report three metrics for the detectors on real video sequences with varying lighting conditions.

Metric	Noise	Resolution	MagicPointL	MagicPointS	FAST	Harris	Shi
mAP	no	160x120	**0.888**	0.850	0.642	0.803	0.674
mAP	yes	160x120	**0.811**	0.730	0.066	0.166	0.132
MLE	no	160x120	**1.365**	1.391	1.908	1.369	1.551
MLE	yes	160x120	**1.470**	1.537	2.236	1.775	1.858
R	no	160x120	**0.970**	0.929	0.800	0.929	0.852
R	yes	160x120	**0.811**	0.711	0.141	0.136	0.148
mAP	no	320x240	**0.892**	0.816	0.405	0.678	0.686
mAP	yes	320x240	**0.846**	0.687	0.018	0.072	0.077
MLE	no	320x240	1.455	1.450	1.914	**1.438**	1.592
MLE	yes	320x240	**1.533**	1.605	2.200	1.764	1.848
R	no	320x240	**0.959**	0.920	0.812	0.896	0.827
R	yes	320x240	**0.765**	0.675	0.099	0.081	0.104

Table 2: Static Corners Results Table. Reports the mean Average Precision (mAP, higher is better), Mean Localization Error (MLE, lower is better) and Repeatability (R, higher is better) across the 30 real data sequences.

5 MagicWarp Evaluation

MagicWarp is designed to operate on top of a fast, geometrically stable point detector running in a tracking scenario. In an ideal world, the underlying point detector would be so fast and powerful that a simple nearest neighbor approach would be sufficient to establish correspondence across frames. We believe that MagicPoint is step the right direction in this regard, but it is not yet perfect, and some logic is still required to clean up the mistakes made by the point detector and occlusions between detections.

On this premise, we devised an evaluation for MagicWarp in which points are randomly placed in an image and undergo four simple transformations. "Translation" is a simple right translation. "Rotation" is an in-plane rotation. "Scale" is a zoom-in operation. "Random H" is a more complex motion that samples a random homography in which the average displacement of points at the corners of the image is 30 pixels in a 160x120 image. Transformations are applied to various densities of point images and various amounts of extra random points added to to the point set x_j. The Nearest Neighbor baseline uses the 3x3 identity matrix I for H and MagicWarp uses the 3x3 matrix output from the network for H.

To measure the performance of MagicWarp, we compute a Match Correctness percentage. More specifically, given a set of points x_i in one image where $i \in \{1, \ldots, N_1\}$ we define a ground truth transformation $\hat{H}$ and the predicted transformation H. We also define the set of points in the second image as x_j where $i \in \{1, \ldots, N_2\}$. Match Correctness determines if a transformed point x'_i has a correct nearest neighbor.

$$\texttt{MatchCorr}(\mathbf{x}_i) = (\underset{j \in \{1,\ldots,N_2\}}{\arg\min} ||H\mathbf{x}'_i - \mathbf{x}_j||) == \hat{H}\mathbf{x}'_i \tag{6}$$

Match Repeatability counts the percentage of correct matches made.

$$\texttt{MatchRep} = 100 * \frac{1}{N_1}(\sum_i \texttt{MatchCorr}(\mathbf{x}_i)) \tag{7}$$

Table 3 aims to answer the question: how extreme of a transformation can the correspondence algorithm handle? To answer this question, we linearly interpolate between the identity transformation and each of the four transformations described above and measure the point at which the Match Repeatability drops less than 90%. We choose 90% because we believe that a robust geometric decomposition using the correspondences should be able to deal with 10% of incorrect matches. Unsurprisingly, the MagicWarp approach outperforms the Nearest Neighbor matching approach in all scenarios.

MagicWarp is very efficient. For an input size of 20x15x130 (corresponding to an image size of 160x120), the average forward pass time on a single CPU is 2.3 ms. For an input size of 40x30x130

		Nearest Neighbor				MagicWarp			
Point Density	Noise	Trans	Rot	Scale	RandH	Trans	Rot	Scale	RandH
Low [5,25]	0%	8.41px	9.42°	1.20×	13.89px	**24.00px**	**21.45°**	**1.32×**	**32.83px**
	20%	9.05px	8.87°	1.24×	11.76px	**24.06px**	**21.25°**	**1.31×**	**29.78px**
	40%	7.15px	7.70°	1.20×	11.59px	**22.64px**	**19.65°**	**1.20×**	**28.84px**
Medium [25,50]	0%	5.19px	5.93°	1.11×	8.03px	**20.20px**	**20.01°**	**1.23×**	**26.52px**
	20%	4.82px	5.41°	1.11×	7.68px	**18.29px**	**18.07°**	**1.21×**	**24.84px**
	40%	4.66px	4.50°	1.10×	6.56px	**17.08px**	**18.03°**	**1.19×**	**24.47px**
High [100,200]	0%	3.49px	3.49°	1.07×	4.87px	**15.10px**	**15.38°**	**1.17×**	**17.51px**
	20%	3.39px	3.34°	1.06×	4.72px	**12.97px**	**13.97°**	**1.13×**	**15.35px**
	40%	3.27px	3.13°	1.06×	4.26px	**10.92px**	**11.17°**	**1.10×**	**12.13px**

Table 3: Matching Algorithm 90% Breakdown Point Experiment. This table compares matching ability of MagicWarp to a Nearest Neighbor matching approach. Each table entry is the magnitude of transformation that results in fewer than 90% Match Repeatability across a pair of input points. Higher is better and the MagicWarp approach performs best in all scenarios. Values are averaged across 50 runs.

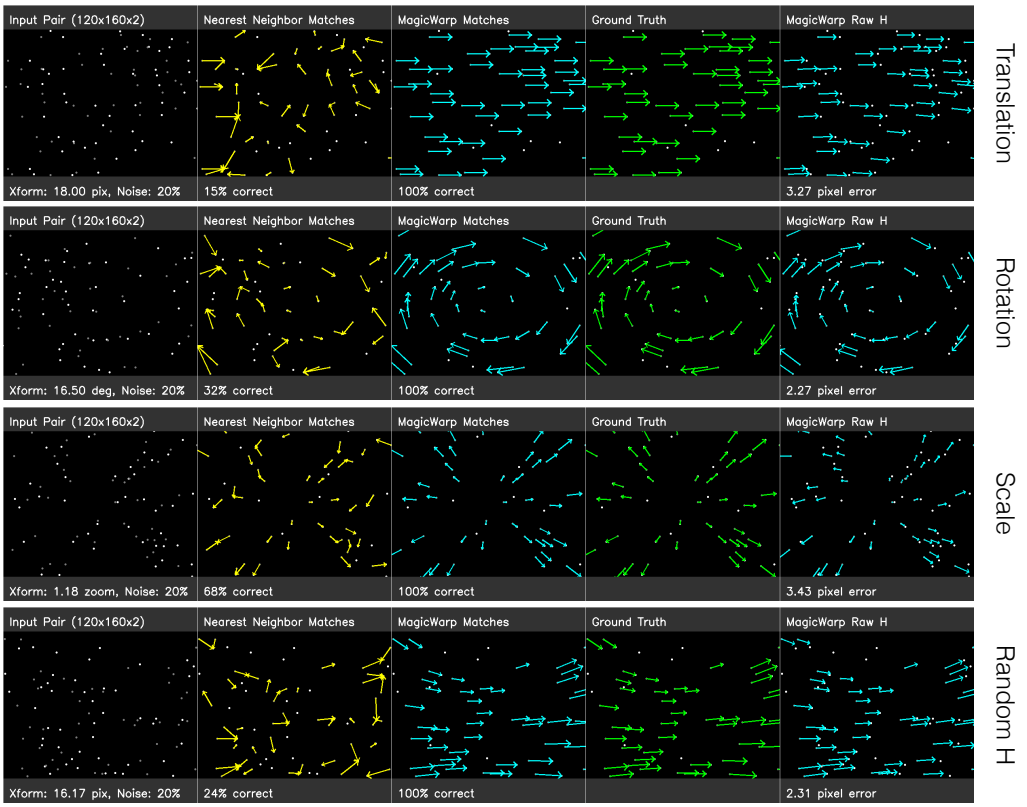

Figure 11: MagicWarp In Action. Examples of MagicWarp for each of the four transformation types summarized in Table 3. The left-most column shows the point image input pair overlayed onto a single image. The right-most column shows the MagicWarp's raw predicted homography applied to the gray point set. The middle column shows the MagicWarp result which applies nearest neighbor to this raw predicted homography, which snaps the arrows to the correct points.

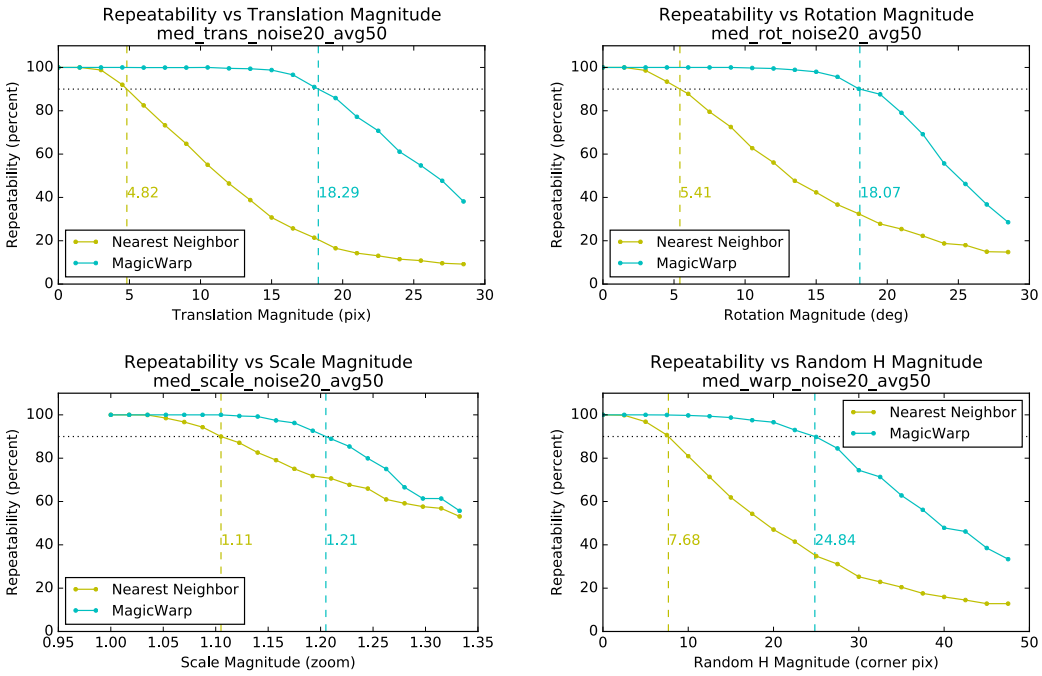

Figure 12: MagicWarp Average Match Repeatability. Match Repeatability is compared versus transformation magnitude for four types of transformations. The point image pairs have medium density and 20% noise added. The vertical dashed lines show the breakdown points at 90%, which are summarized for different configurations in Table 3

(corresponding to an image size of 320x240), the average forward pass time on a single CPU is 6.1 ms. The times were computed with BatchNorm layers folded into the convolutional layers.

6 Discussion

In conclusion, our contributions are as follows. We formulated two SLAM subtasks as machine learning problems, developed two simple data generators which can be implemented in a few hundred lines of code, designed two simple convolutional neural networks capable of running in real-time, and evaluated them on both synthetic and real data.

Our paper was motivated by two burning questions: 1.) *What would it take to build an ImageNet-scale dataset for SLAM?* and 2.) *What would it take to build DeepSLAM?* In this paper, we have shown that our answers to both questions are intimately related. It would be wasteful to build a massive dataset first, only to learn a year later that the best algorithm does not even use the labels you worked so hard to procure. We started with a mental framework for Deep Visual SLAM that must solve two separate subtasks, which can be combined into a point tracking system. By moving away from full frame prediction and focusing solely on geometric consistency, our work has hopefully shown that the day of ImageNet-sized SLAM datasets might not need to come, after all. We believe that the day of massive-scale deployment of Deep-Learning powered SLAM systems is not far.

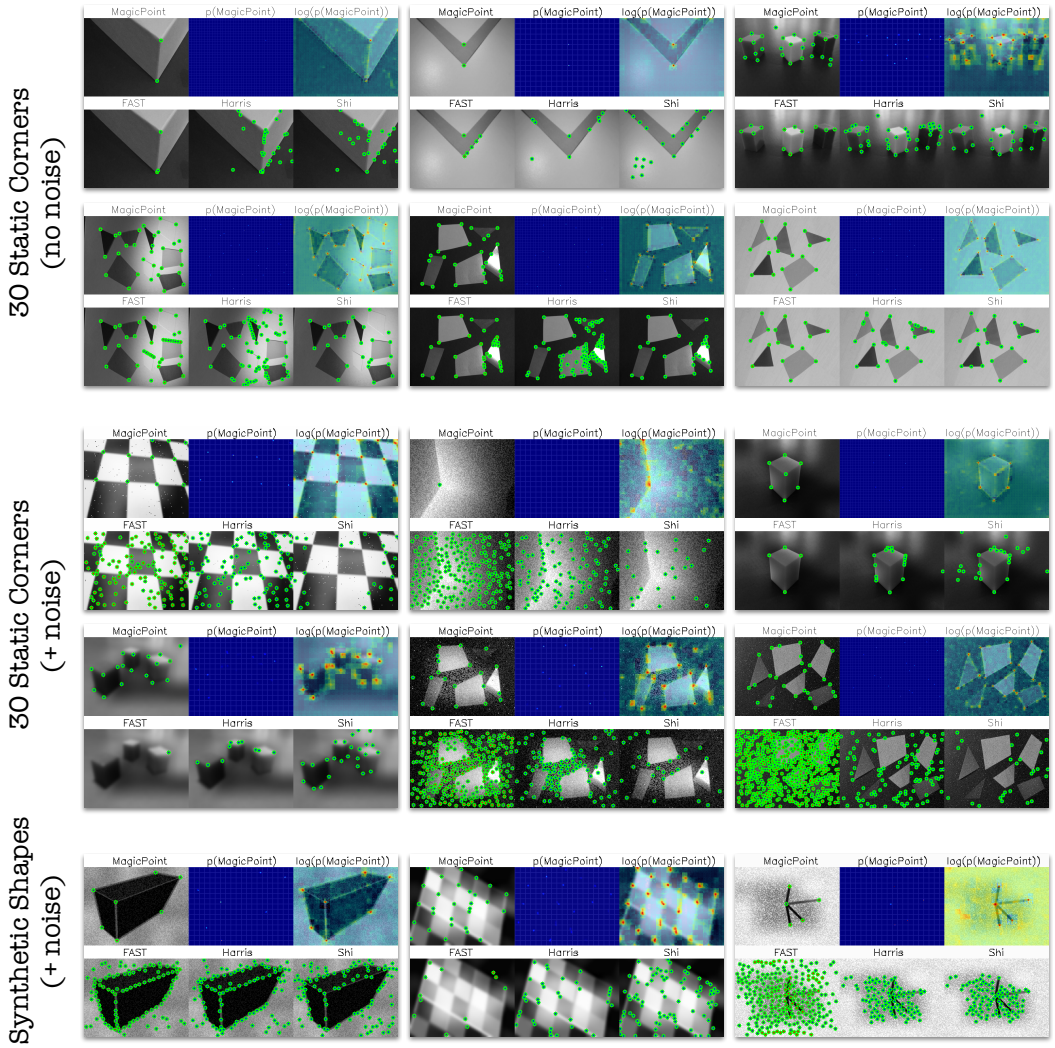

Figure 13: MagicPoint in Action. This figures show 15 example results for MagicPointS vs traditional corner detection baselines. For each figure, we display the MagicPointS output, the output probability heatmap, the overlayed log(probability) heatmap (to enhance low probabilities), as well as FAST, Harris, and Shi. The top examples are from `3O Static Corners` with no noise. The middle examples are from `3O Static Corners` with noise. The bottom examples are from `Synthetic Shapes` with noise. Note that our method is able to cope with large amounts of noise and produces meaningful heatmaps that can be thresholded in an application-specific manner.

References

[1] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[2] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[3] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[4] T. Zhou, M. Brown, N. Snavely, and D. G. Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, 2017.

[5] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality (ISMAR'07)*, November 2007.

[6] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[7] K. M. Yi, E. Trulls, V. Lepetit, and P. Fua. LIFT: Learned Invariant Feature Transform. In *Proceedings of the European Conference on Computer Vision*, 2016.

[8] D. DeTone, T. Malisiewicz, and A. Rabinovich. Deep image homography estimation. *CoRR*, abs/1606.03798, 2016. URL http://arxiv.org/abs/1606.03798.

[9] P. Fischer, A. Dosovitskiy, E. Ilg, P. Häusser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, 2015.

[10] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *ICCV*, pages 2938–2946, 2015.

[11] E. Brachmann, A. Krull, S. Nowozin, J. Shotton, F. Michel, S. Gumhold, and C. Rother. DSAC - differentiable RANSAC for camera localization. In *CVPR*, 2017.

[12] R. Gomez-Ojeda, M. Lopez-Antequera, N. Petkov, and J. Gonzalez-Jimenez. Training a convolutional neural network for appearance-invariant place recognition. *CoRR*, abs/1505.07428, 2015.

[13] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. In *CVPR*, 2017.

[14] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In *CVPR*, 2017.

[15] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK, 1988.

[16] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2): 91–110, Nov. 2004. ISSN 0920-5691.

[17] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[18] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, 2016.

[19] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[20] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *ECCV*, 2006.

[21] J. Shi and C. Tomasi. Good features to track. In *CVPR*, 1994.

[22] S. Gauglitz, T. Höllerer, and M. Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *International journal of computer vision*, 94(3):335–360, 2011.