RAPTOR Technical Report

Sam Kelly[1], Jeff Byers[2], David Aha[3]

[1]B.S. Dickinson College '14, 2014 NREIP Intern, Naval Research Laboratory

[2]Jeff Byers, Naval Research Laboratory, Optical Communications & Sensing Section Head

[3]David Aha, Naval Research Laboratory, Adaptive Systems Section Head

1. Introduction

In this technical report we present RAPTOR (Rapid Three-Dimensional Orientation Resolver), which is a novel pipeline for inferring solely from 2D image data the 3D position and orientation (pose) of known classes of rigid objects for which man-made 3D models are available. There are many existing systems and techniques that attempt to infer 3D meshes and scene information from 2D image or video data [1][2][3]. Without domain-specific knowledge it would be difficult to generate a proportionally accurate 3D mesh based on a single still image with no depth information [3]. Instead, RAPTOR takes advantage of the availability of existing highly accurate man-made 3D models for common rigid objects, such as vehicles, weapons, street signs, etc., to tackle the problem of 3D scene understanding in a class-specific fashion. In other words, RAPTOR leaves the problem of generating 3D meshes for terrain and uncommon or articulated objects to other systems and instead focus on finding the position and orientation of known, already-detected objects with a high degree of speed and accuracy.

Figure 1.1: A properly oriented 3D mesh of a Hamina-class missile boat shown superimposed over a real-world picture of said boat. Since the 3D model is properly oriented with respect to the picture, we can easily calculate non-trivial state information, such as movement speed, trajectory, potential collisions with and proximity to other objects, etc. The goal of RAPTOR is to be able to estimate object poses like this on the fly in a highly accurate, real-time fashion.

For this technical report we have planned three preliminary versions of the RAPTOR pipeline (Direct Learning, Pose Triangulation, and Adjustive Pose Convergence). In all three versions, the input to the pipeline is a pre-masked[1] region of a 2D image of an object whose class is known *a priori*. Direct Learning generates 2D covariance matrices from input images, and then uses class-specific artificial neural networks to try to learn the mapping of 2D covariance matrix to 3D pose parameters directly (possibly in tandem with stereo vision, and possibly also using a scaled-down version of the raw image pixels as an additional feature vector). Pose Triangulation uses a covariance matrix distance metric to measure the distance between the 2D covariance matrix for the input image and several reference poses. This information can then be used as a feature vector to an artificial neural network or as a key to a locality sensitive (or space partitioning) data structure or a clustered knowledgebase of input/output pairings. Adjustive Pose Convergence requires no training data and instead repeatedly generates renderings based on a man-made 3D model of the target object class, iteratively converging on the target pose using a procedure similar to binary search in which Pose Triangulation is used at each step.

In general, RAPTOR generates training input images by rendering (with full textures, lighting, etc.) arbitrary poses of the 3D model of the object class in question using a 3D rendering pipeline (for our experiments we use the raytracing engine from the popular Blender 3D model editor). Thus a major and so far untested assumption all RAPTOR pipeline versions make is that a system trained on 3D renderings will generalize readily to real world images. Arguably, though, this is really just a question of how photorealistic implementers decide to make the 3D rendering pipeline. Since we have full control over the 3D models and rendering pipeline, RAPTOR can generate as many or as few training instances as are deemed necessary at desired level of accuracy or generality.

1.1 Motivations

Artificial agents in modern 3D video games are tactically formidable because they are able to leverage a wealth of game state and 3D scene information that would be impossible or incredibly difficult to infer visually in an uninformed fashion. General techniques exist that attempt to infer full 3D depth information from 2D video or image data, however these techniques tend to be over-sensitive to lighting conditions and optical illusions, often leading to dramatic inaccuracies and deformations in the resulting 3D scene (e.g. see the error cases from [3]). While depth sensors and/or LIDAR can help alleviate this problem, there are many situations where accurate 3D pose estimation is required, but only 2D imagery is available. Existing techniques tend to focus on generating 3D meshes for surrounding terrain and buildings rather than figuring out the 3D position and orientation of common world objects such as vehicles, signs, weapons, etc., for which accurate man-made 3D models are readily available. We believe that next-generation tactical systems should be able to (1) infer from 2D imagery an accurate 3D understanding of the surrounding terrain and obstacles; (2) recognize and *label* nearby common

[1] i.e. pixels that are not part of the object are excluded from the input image, and the input image is also cropped so that the object is centered and the image rectangle is dominated by the object

rigid objects, such as ground vehicles, ships, planes, weapons, ordinance, helmets, etc.; and (3) correctly *orient* and *embed* full 3D meshes of these objects in the resulting virtual 3D scene space if they are available. In principle, a system capable of meeting all three of these requirements in real time would have roughly the same wealth of scene information available to artificial agents in modern video games, potentially allowing such a system to be competitive with human-level scene understanding in real-world situations. RAPTOR, then, aims to accomplish only item (3) on this list, whereas previous and related work largely address only (1) and/or (2).

1.2 Related Work

Most prior work in this space either focuses on 3D tracking or template matching as an object recognition problem, or tries instead to solve the much harder problem of inferring per-pixel depth information from a 2D image in a class-insensitive fashion. Additionally, many papers exist that focus on 3D reconstruction, or rather, constructing 3D meshes based on numerous 2D images (or stereo-pair images) of an object or environment. That said, we can find no existing work that tries to take advantage of man-made 3D models to diminish or mitigate the difficulty of reconstructing a full 3D scene. In fact, there seems to be a wealth of existing work that attempts to solve (1) and (2), but very little if any that tackles (3), which boils down to inferring the 3D depth and orientation of known (already classified) objects based solely on 2D image data. A major reason for the scarcity of existing work in this space might be the lack of relevant training data – while it might be feasible to take thousands of images of a ball or a teapot from varying distances and angles, doing so would be much more difficult in the case of large real-world object classes such as cars, ships, and planes. RAPTOR addresses this scarcity problem by relying on computer-generated rather than real-world imagery.

OverFeat [4] combines an efficient scale-invariant feature extraction pipeline with convolutional neural networks to facilitate extremely accurate object detection in 2D images and video. The authors of [5] use a similar technique to perform object classification, also making use of CNNs in their pipeline since CNNs are particularly suitable for image data [6]. The authors of [7] use a unique melding of generative and discriminative layered deep belief networks to recognize 3D objects based on the NORB database of stereo-pair images (depth information is included in the input). LINE-MOD [5] provides a gradient-based approach for real-time 3D object recognition that does not rely on neural networks or machine learning. All of these approaches, however, concern themselves merely with object classification, and do not attempt to resolve object distance or 3D orientation as RAPTOR does.

Cornell's Make3D [3] provides impressive results on the difficult problem of generating a 3D mesh of a complex scene from a single monocular image. In [6], a similar technique for 3D scene reconstruction is used whereby semantic cues are detected in the 2D image as a preprocessing step to help take the burden of differentiating buildings from ground, and ground from sky, etc., off of the learning algorithm. The authors of [8] use a coded lens and exploit image focal length properties to obtain accurate per-pixel distance estimations from a conventional camera. Generally the existing work

in this category focuses on the extremely difficult problem of generating full 3D meshes or per-pixel depth information, rather than attempting to orient existing 3D models for already-identified instances of known object classes in a 3D scene as RAPTOR does. In fact as far as we can tell, RAPTOR is the first entry in the space of 3D-model-aided pose estimation.

2. RAPTOR Pipeline

RAPTOR is designed to take a region of a 2D image that corresponds with a known object class as input. For output, RAPTOR provides the 3D distance and 3D orientation of that object with respect to the camera (depending on which pipeline variant is being used, 3D orientation is represented either as an Euler rotation angle, a unit vector, or a 3D covariance matrix which implicitly encodes the orientation). To accomplish this, we intend to test three major families of variations of the basic RAPTOR pipeline, which will be outlined in sections 2.2 through 2.5.

2.1 Training Data

Most systems in 3D reconstruction and object recognition rely on real-world images to facilitate most of their training, however in the case of RAPTOR it would be too difficult to acquire the number of images that would be needed, especially considering the large size of some of the objects we want to work with (i.e. naval ships). To combat this, we came up with the idea of training on renderings of 3D models of our objects at arbitrary distances and orientations. This means that for RAPTOR to be able to learn any given object, there must be a man-made 3D model available for that object. It remains to be shown, however, whether learning on renderings will generalize such that RAPTOR can make correct estimations based on real-world images. There are also obstacles that we will eventually have to deal with involving different camera focal lengths, etc. That said, there do exist expensive 3D rendering pipelines such as Eon Vue that are known for their ability to create seamlessly photo-realistic renderings including realistic oceans, waves, weather, foliage, etc. Likewise, filters could be used during rendering to simulate the focal properties of a particular camera, and this is actually a common practice in certain CAD environments. We do not have plans to work with any of these pipelines for this project, however, since this project is intended more as a proof-of-concept. If we are able to obtain decent results on purely rendered data, then we will also asses how well RAPTOR adapts to real-world image data, though doing so will require a good degree of work on our part – we will need to manually orient 3D models in 2D images and add the proper annotations, and this can take up to an hour per image.

Figure 2.1.1: Training data generated by RAPTOR.

Currently, RAPTOR uses a Python script to hook into Blender 3D's rendering pipeline (Blender is written entirely in Python and has an extensible design allowing for task automation using Python). This hook allows us to create high quality ray-traced renderings with realistic lighting and textures without having to write our own pipeline for doing so. Our python script also allows us to load a 3D model and randomize camera position, angle, and lighting properties (soon we will be able to do this with texture and material properties as well). Thus we can create a virtually infinite set of training data for any given 3D model. Distance and rotation are controlled by the script, so this information will be known a priori meaning testing will be incredibly easy (we won't have to spend hours manually annotating images). As mentioned before, we also plan to supplement this data with a sparse set of real-world image data to improve RAPTOR's ability to generalize.

2.2 Variant A1: Direct Learning

Since RAPTOR takes image data as input and produces four continuous values as output, it is tackling a function approximation problem rather than a classification problem. Direct Learning, then, is merely the naïve approach to solving the function approximation problem of mapping 2D input images to 3D pose parameters.

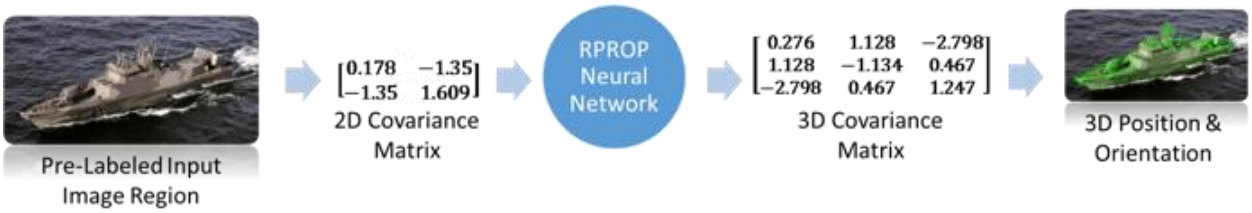

Figure 2.2.1: One possible version of the evaluation pipeline for Direct Learning.

Our proposed architecture for Direct Learning consists of a shallow (around three hidden layers)

resilient-propagation (RPROP) artificial neural network trained via supervised learning. We chose RPROP because it is extremely quick to train (only the sign of the error gradient needs to be calculated), highly resilient against local minima, immune to the vanishing gradient problem (see [9]), has no hyper-parameters that need configuring, and is widely recommended as a reliable general-purpose feedforward learning algorithm [6]. The input to this network will be a scaled down, grayscale version of the input image (preliminary results suggest that a 32 x 32 pixel input grid is sufficient for most models, however this number is subject to change). The input image is cropped and scaled up or down to fit the input grid. Each of the resulting 1024 pixel intensities is then normalized to a real number between 0.0 and 1.0 so it can be provided directly as input to the RPROP network. Since the input image was cropped and scaled, we also provide the network with the original crop and scaling information. Specifically, the network is presented with four real values representing the proportion by which the left, top, right, and bottom sides of the image was cropped, and an additional real value representing the scaling factor. Thus in theory the network has all the information it needs to make sense of the cropped and scaled input pixels.

While the input pixel intensities should contain all the information needed to conduct 3D pose estimation, we believe augmenting this information (or possibly replacing this information entirely) with a 2D covariance matrix representing the 2D input image might prove effective. Formally, the eigenvalues of an n-dimensional covariance matrix define an n-ellipse capturing the size, shape, and orientation of the original data, so the 2-ellipse formed from the 2D input image should contain the spatial information needed to identify a particular 3D pose depending on the shape of the model.

Preliminary results suggest that the Direct Learning approach is too difficult for a shallow feedforward network to learn, but we are still tweaking with different network sizes and input representations, so it is possible this approach might prove fruitful at some point in the future.

2.3 Variant A2: Direct Stereo Learning

If the other pipeline variants prove intractable, we might consider a version of Direct Learning that uses a sequential pair of input frames and stereo vision to generate a 3D covariance matrix that can be used as a feature vector to the normal Direct Learning pipeline. That said, we would like to avoid relying on stereo vision if we can because stereo vision requires video data and becomes less accurate when there is little movement in the video stream. All of our other pipeline variants work on still images, meaning RAPTOR could be applied to images pulled from the internet in addition to frames from videos if we do not use Direct Stereo Learning. There is also a large amount of existing work that uses stereo vision for similar purpose, which makes it a less attractive as a research topic.

2.4 Variant B: Pose Triangulation

2D covariance matrices provide a convenient way for encoding spatial properties of a 2D input image. The authors of [10] provide a powerful distance metric capable of measuring the length of the geodesic between two n-dimensional covariance matrices along the manifold of all possible n-

dimensional covariance matrices. For our purposes, this metric can be used to measure the approximate distance in pose space between pairs of 2D or 3D covariance matrices. The "distance" between two covariance matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is defined as follows:

$$\mathrm{d}(\boldsymbol{A},\boldsymbol{B}) = \sqrt{\sum_{i=1}^{n} \ln^2 \lambda_i(\boldsymbol{A},\boldsymbol{B})}$$

Pose Triangulation works by calculating the distance between the 2D covariance matrix generated from the input image and a number of reference poses that were pre-computed. These distances can then be used either as a feature vector to Direct Learning, or they can be averaged and used as a search heuristic to find the closest matching pose in the original set of training data.

In RAPTOR, a particular 3D pose can be uniquely represented merely as a combination of a distance parameter, a rotation-x parameter, a rotation-y parameter, and a rotation-z parameter. Thus (at least in RAPTOR) the space of all possible poses for a given 3D model is actually four-dimensional, and is bounded by the minimum and maximum rotation values (0 to 2π) and the minimum and maximum distance supported by the training data set.

Imagine for a moment that we are searching for 3D points within a predefined 3D bounding box. When performing this search, it would be useful to know the distance between our target point and each of the corners of the bounding box. In fact given the distance between our target point and any three of the eight corners of the bounding box, we can calculate the exact position of our target point using 3D triangulation. Pose Triangulation, then, is merely an analogue to this process, but instead of three dimensions, we are working in four-dimensional pose space, and instead of spatial distance, we use the covariance matrix distance metric defined in [10].

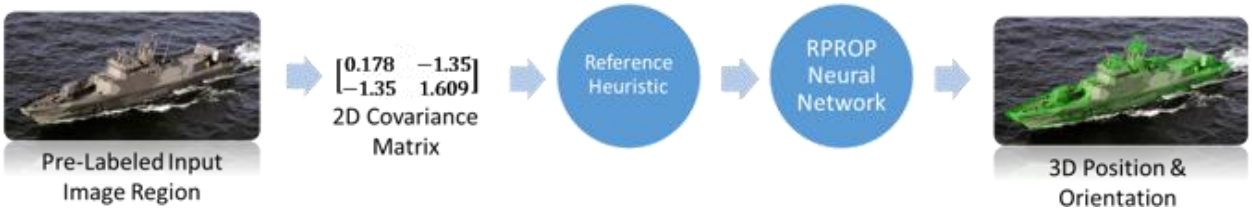

Figure 2.4.1: One possible version of the evaluation pipeline for Direct Learning.

The most straightforward approach would be to pre-compute 2D covariance matrices for each of the 16 "corner" poses in our four-dimensional pose space. During runtime, we would calculate the distance between the 2D input pose and the 16 corner poses. We would then either use this information as a feature vector to Direct Learning, or as a heuristic for a data-clustering approach (potentially nearest neighbor), a locality-sensitive hashing scheme, or a space partitioning data structure (probably something similar to the quadtrees used in [11]). If this doesn't work well, we could try resizing the input image to a fixed grid size before generating the 2D covariance matrix for this image. By removing distance from the equation (at least initially), we would instead be operating in the three-dimensional

space of all possible rotations, meaning there would only be 8 corner poses as in the imaginary 3D example from above. Once the correct rotation values are found, a simple linear matching algorithm could then be used to find the correct distance.

2.5 Variant C: Adjustive Pose Convergence

Adjustive Pose Convergence relies on an iterative process by which we come up with a better guess for the target pose at each iteration. We start with the default neutral model pose as the initial guess. Next, we measure the distances between the (2D) covariance matrix for this guess pose and the 16 reference poses, whose covariance matrices were generated ahead of time. We call this sequence of distances D_{guess}. We also measure the distances between the covariance matrix for the target pose and the 16 reference poses, storing these in the sequence D_{target}. The differences between D_{guess} and D_{target} should now tell us whether we have over- or under-shot the target pose in terms of (2D) pose similarity to the 16 reference poses. We can then use a process similar to binary search to adjust the rotation and distance of our guess pose based on these differences. To converge on the target pose, we simply repeat this entire procedure iteratively until the differences go to zero. Assuming this technique does not suffer from local minima problems on certain classes, we hypothesize that this should be an effective, fast way of constructing the RAPTOR pipeline.

3. Conclusion

The aim of RAPTOR is to provide an accurate method for embedding already-created 3D models in a 3D scene based on 2D pixel data representing a known object class in such a way that distance to camera and 3D orientation are resolved. While similar problems have been studied extensively in the existing literature, to our knowledge no one has tried to tackle 3D scene reconstruction as a problem of embedding existing 3D models, and few have focused on 3D reconstruction in a class-sensitive fashion to begin with. Likewise, there are very few cases in the existing literature where researchers exploit renderings of 3D models to supplement an otherwise sparse set of 2D images. If RAPTOR works well, we will likely try to publish our results since this technique has the potential to positively impact the computer vision community and the deep learning community, not to mention the potential military, law enforcement (e.g. traffic violation detection), and surveillance applications RAPTOR could also be used for (e.g. a surveillance system could monitor a particular object's position in 3D, or trigger an alert if one class of object comes too close to another class of object, etc.). We are particularly optimistic about the research potential of the Pose Triangulation and Adjustive Pose Convergence families of pipeline variants, as these are radically different from existing techniques, and we hypothesize that they will work well even when faced with real-world data.

Works Cited

[1] J. Cheng, C. Leng, J. Wu, H. Cui, and H. Lu, “Fast and Accurate Image Matching with Cascade Hashing for 3D Reconstruction,” *CVPR*, 2014.

[2] B. Liu, S. Gould, and D. Koller, “Single image depth estimation from predicted semantic labels,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, 2010, pp. 1253–1260.

[3] A. Saxena, M. Sun, and A. Y. Ng, “Make3d: Learning 3d scene structure from a single still image,” *Pattern Anal. Mach. Intell. IEEE Trans.*, vol. 31, no. 5, pp. 824–840, 2009.

[4] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, “OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks,” *arXiv Prepr. arXiv1312.6229*, 2013.

[5] R. Socher, B. Huval, B. P. Bath, C. D. Manning, and A. Y. Ng, “Convolutional-Recursive Deep Learning for 3D Object Classification.,” in *NIPS*, 2012, pp. 665–673.

[6] J. Heaton, *Introduction to neural networks with Java*. Heaton Research, Inc., 2008.

[7] V. Nair and G. E. Hinton, “3D Object Recognition with Deep Belief Nets.,” in *NIPS*, 2009, pp. 1339–1347.

[8] A. Levin, R. Fergus, F. Durand, and W. Freeman, “Image and depth from a conventional camera with a coded aperture,” *ACM Trans. …*, 2007.

[9] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen Netzen,” *Master’s thesis, Inst. fur Inform. Tech. Univ. Munchen*, 1991.

[10] W. Förstner and B. Moonen, “A metric for covariance matrices,” *Geod. Chall. 3rd Millenn.*, 2003.

[11] H. Samet and R. Webber, “Storing a collection of polygons using quadtrees,” *ACM Trans. Graph.*, 1985.