Motion-aware Monocular Depth Completion for Aerial Vehicles with Deep Neural Networks

by

Jing Lin

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Author .....
Department of Electrical Engineering and Computer Science
May 12, 2020

Certified by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Motion-aware Monocular Depth Completion for Aerial Vehicles with Deep Neural Networks

by

Jing Lin

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2020, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Depth data is critical for autonomous robots like cars and aerial vehicles to understand their environments for obstacle avoidance and path planning. Classically, depth data for these robotics applications is obtained with stereo cameras, structured light cameras, or light detection and ranging (LIDAR) sensors. That is possible for autonomous vehicles which can be equipped with additional sensors but poses significant challenges for aerial vehicles: more sensors mean more weight which restricts mobility and flight-time. Furthermore, it is impossible to mount depth sensors on drones on the scale of 10-50 centimeters. To that end, we explore the depth completion problem with only a monocular camera which can be readily mounted on a drone. Our work builds on a prior state-of-the-art encoder-decoder network architecture for depth completion. Our model performs accurate depth completion on the Blackbird dataset, a drone dataset and adding scaled depth data from visual inertial odometry (VIO) further improves performance.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics

Acknowledgments

The work presented here would not have been possible without the continued help from my collaborators Fangchang Ma, Varun Murali and Winter Guerra. Thank you so much for everything. I have really enjoyed working together.

I really appreciate the valuable revisions provided by Varun Murali, Winter Guerra and Sertac Karaman on this thesis.

I am also very grateful to Sertac Karaman for always being supportive and thoughtful throughout the project. I am also very thankful to fellow lab members who have all created a fun, collaborative workspace: Andrea Henshall, Dave McCoy, Ezra Tal, Gilhyun Ryou, Ifueko Igbinedion, Igor Spasojevic, Jin Gao, John Aleman, Muyuan Lin, and Sebastian Quilter. I would also like to thank my friends and family for their continued support and encouragement.

Contents

1 Introduction **13**

1.1 Overview of Depth Sensors 13

1.2 VIO Depth Completion Approach 15

1.2.1 Feature Tracking 15

1.2.2 Pose Optimization And Depth Triangulation 17

1.3 Encoder Decoder Architectures Approach 19

2 Depth Completion with Sparse Depth **23**

2.1 Datasets For Depth Completion 23

2.2 The Blackbird Dataset 24

2.3 Related Work 24

2.4 Overview of Network Architecture 26

2.4.1 Overview of Sampling Strategy 27

2.4.2 Training Framework 29

2.4.3 Loss Function 30

2.5 Experiments on Half Moon 31

3 Motion-aware Monocular Depth Completion **35**

3.1 Related Work 35

3.2 Overview of Our VIO Setup 36

3.3 Experiments 37

3.3.1 Experiments on Blackbird Subset 37

3.3.2 Experiments on Blackbird 40

4 Other Experiments **45**

4.1 Experiments on VOID 45

4.2 Other Experiments on VOID 47

5 Conclusion **49**

5.1 Overview of Results 49

5.2 Future Work 50

5.2.1 Exploring Data 50

5.2.2 Improving the VIO Pipeline 51

5.2.3 Blackbird Official Release 52

A Figures **53**

List of Tables

1.1 Sensor Comparisons. 14

2.1 `blackbird` train and val errors. 31

3.1 `blackbird_subset` train and val errors. 37
3.2 Number of GFTT features by trajectory. 38
3.3 Validation error by trajectory for RGB and RGB_VIO. 39
3.4 `blackbird` train and val errors. 40
3.5 Validation error in `mm` by trajectory for RGB and RGB_VIO. 42
3.6 Validation error in `mm` by environment for RGB and RGB_VIO. . . . 42

5.1 `blackbird_subset` train and val errors. 49

List of Figures

1-1 Overview of our factor graph for Pose Estimation 18

1-2 Complete overview of our VIO system. 19

1-3 Overview of encoder-decoder neural network architectures 20

2-1 Figure reproduced from [5] depicting the Blackbird environments . . . 25

2-2 Figure reproduced from [5] depicting all the flight trajectories. 26

2-3 Examples of various RGB/depth data from Blackbird. 27

2-4 Diagram of our sparse to dense network on the HalfMoon trajectory in Blackbird 28

2-5 RMSE comparison on Half Moon between our RGB and RGB_RANDOM networks 32

2-6 Sample predictions of our depth prediction networks on the Half Moon trajectory 33

3-1 Examples of various images on oval. 40

3-2 Example RGB_VIO and RGB predictions on oval. 41

3-3 Some of our network predictions on the Blackbird dataset. 43

4-1 Some of our network predictions on the VOID dataset 46

5-1 Some RGB data with incorrect depth maps. Some from Museum Pillars and NYC Subway are shown here. 51

A-1 Sample predictions of RGB and RGB_VIO trained on `blackbird_subset` for the oval trajectory. 53

A-2 Sample predictions of RGB and RGB_VIO trained on `blackbird_subset` for all the environments. 54
A-3 Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Small Apartment environment 55
A-4 Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Hazelwood Loft environment 56
A-5 Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Museum Pillars environment. 57
A-6 Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Museum Sphinx Loft environment. 58
A-7 Sample predictions of RGB and RGB_VIO trained on `blackbird` for the NYC Subway environment 59

Chapter 1

Introduction

Depth data is important for autonomous robots to navigate their environments. It is important for obstacle avoidance and path planning [18] [17]. However, depth data can be hard to obtain due to sensor costs and the physical constraints of the robots. To that end, researchers have explored deep learning approaches for depth completion - taking sparse depth data and predicting depth data for the rest of the pixel locations.

Previously, researchers studied depth completion for autonomous vehicles [9, 10, 23, 29, 30]. Our work extends that work to autonomous aerial vehicles. Depth data is important for aerial vehicles to detect and maneuver around obstacles in their flight path [17, 1, 14]. In our work, we train sparse depth deep learning models to perform depth reconstruction. Then we explore extensions of our work with depth data from VIO (Visual Inertial Odometry) generated from successive monocular camera data. Overall, we show deep learning is a robust solution to depth completion.

In this introduction, we provide an overview of various depth sensors, explain VIO depth reconstruction and present a primer on encoder decoder network architectures.

1.1 Overview of Depth Sensors

There are a variety of depth sensors that differ in terms of accuracy, cost and size. Broadly, there are three types: LIDAR sensors, structured light sensors (e.g. Kinect) and stereo cameras. LIDAR sensors can provide rich depth information but can

be cost prohibitive. Velodyne's HDL (High Definition Real-Time 3D Lidar) sensors can provide depth information up to 120m at ±2cm resolution. However its best sensor costs $75,000. Structured light sensors provide accurate depth information at a shorter range and are more economical, but can be oversaturated in outdoor environments with strong illumination. The Microsoft Kinect tracks depth up to 5m at ±5cm resolution at 5m but only costs $400. Stereo cameras provide depth up to 10m with 5cm resolution and only costs $200. However, they incur heavy computation costs and are limited by camera intrinsics and the number of visual features, which can vary dramatically depending on the environment.

Another consideration is the physical size of the sensor. For drones on the order of 10-50cm, both LIDAR (17.8cm × 15.2cm) and structured light sensors (27.5cm × 7.5cm) are too big. Payload is also important and both types of sensors are heavy: 13.2kgs and 1.4kgs respectively. In contrast, stereo cameras can be 5cm × 5cm and 0.5kgs. Although camera based approaches are not as robust as structured light or LIDAR, they are the only viable approach that can be mounted on aerial vehicles.

One more consideration is the heavy computational costs of stereo approaches. The current state of the art in drone constrained hardware are embedded GPUs which include the Jetson TX series from Nvidia. The feature tracking, matching and depth predictions of stereo approaches can be performed at very high frame rates. Furthermore, prior work, namely FastDepth, optimized depth completion to run at 178 FPS on a Nvidia Jetson TX2 GPU and at 27 FPS when restricted to the TX2 CPU, with active power consumption under 10 W [42]. For aerial vehicle applications, those framerates are on par with off the shelf stereo depth sensors.

The various aspects of the sensors are summarized in Table 1.

Table 1.1: Sensor Comparisons.

Sensor	Accuracy	Robust	Cost	Weight	Dimension (cm)
LIDAR	120m±2.2cm	Yes	$75,000	13.2kg	17.8 × 15.2
S. Light	5m±5cm	No	$400	1.4kg	27.5 × 7.5
Stereo	10m±5cm	No	$200	0.5	5 × 5

Of the sensors listed, only stereo satisfies the weight and dimension constraints for drones.

1.2 VIO Depth Completion Approach

Our VIO approach contains two parts: feature tracking and factor graph based depth triangulation. On a high level, we find features on the first frame (or the key frame) in an input data sequence with Good Features To Track (GFTT) [36]. We then track those features across successive RGB frames with iterative Kanade–Lucas–Tomasi (KLT) optical flow [27] [39]. The IMU data between the start and the last frame is then computed as a pre-integrated factor and added as a constraint between the keyframes. The bundle adjustment problem is then solved to recover the poses. We apply Direct Linear Transform (DLT) with the poses to recover the depth values for each feature point.

1.2.1 Feature Tracking

Features should capture the interesting, distinct parts of a picture. They should ideally be invariant to changes in position, camera pose, lighting, illumination and scale that occur so they can be tracked in the successive frames. Corners are interesting because they are associated with objects and structures. They also have significant gradient changes in all directions.

We apply the Shi–Tomasi feature detector to detect gradient changes and find robust corners. It is very similar to the Harris corner detector [15]. It works by measuring the intensity change for a u, v shift over windows of x, y pixel locations. The weighted sums of squared differences $S(u, v)$ is defined as

$$S(u,v) = \sum_{x,y} w(x,y)[]I(x+u, y+v) - I(x,y)]^2$$

$$I(x+u, y+v) \approx I(x,y) + uI_x(x,y) + vI_y(x,y)$$

$$S(u,v) \approx \sum_{x,y} w(x,y)[uI_x(x,y) + vI_y(x,y)]^2$$

$$= \sum_{x,y} w(x,y)[u^2 I_x^2 + 2uvI_xI_y + v^2 I_y^2]$$

$$= \sum_{x,y} w(x,y)[u \;\; v]\begin{pmatrix} I_x^2 & I^xI^y \\ I^xI^y & I_y^2 \end{pmatrix}[u \;\; v]^T$$

$$M = \begin{pmatrix} I_x^2 & I^xI^y \\ I^xI^y & I_y^2 \end{pmatrix}$$

Let λ_1, λ_2 be the first two eigenvalues of the M. Their magnitude is directly correlated with the likelihood of a corner being present in the window. Shi-Tomasi performs thresholding on the minimum eigenvalue to ensure that we have detected corners, not edges or uniform pixels.

The detected features are first sorted in descending order according to their eigenvalues for quality t and distance d thresholding. Let f_1 be the first feature in the list and the response function be $q = \min(\lambda_1, \lambda_2)$. Any feature where f_i with $tq(f_1) \geq q(f_i)$ is removed. For the rest of the features, any feature which is closer than d distance to the best feature is removed. The current best feature is moved to the set of tracked features and the next best feature is selected from the remaining features. This continues iteratively until we run through all the points.

We then apply KLT optical flow on our tracked feature points. KLT optical flow depends on three assumptions: small motion (points don't move very far), brightness constancy (projection of the same point looks the same in every frame) and spatial coherence (points move like nearby pixels). Brightness constancy means that a point x, y on an image at time t $I(x, y, t)$ is displaced by a motion vector u, v.

$$I(x, y, t) = I(x + u, y + v, t + 1)$$

We can write $I(x + u, y + v, t + 1)$ in terms of its first order Taylor Series expansion and solve for u, v.

$$I(x + u, y + v, t + 1) \approx I(x, y, t) + I_x u + I_y v + I_t$$

$$I(x + u, y + v, t + 1) - I(x, y, t) = I_x u + I_y v + I_t$$

$$\nabla I[u \;\; v]^T + I_t = 0$$

There is one equation here and two unknowns. However, the spatial coherence assumption allows us to consider a $k \times k$ window around the point of interest. Let's assume a 3×3 window. Our equation can now be rewritten as a system of equations.

$$\begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_9) & I_y(p_9) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_t(p_1) \\ I_t(p_2) \\ \vdots \\ I_t(p_9) \end{bmatrix}$$

$$A = \begin{bmatrix} I_x(p_1) & I_y(p_1) \\ I_x(p_2) & I_y(p_2) \\ \vdots & \vdots \\ I_x(p_9) & I_y(p_9) \end{bmatrix}, b = \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum I_x I_t \\ \sum I_y I_t \end{bmatrix},$$

We can solve for u, v if $A^T A$ is invertible and its eigenvalues λ_1, λ_2 satisfy the same constraints described for good corners. On real-world data, the small motion assumption often breaks and KLT optical flow is implemented with a few tricks to account for that. The algorithm is often applied iteratively , and u, v are updated until one of two stop conditions is reached: the change between u_t, v_t and u_{t+1}, v_{t+1} is below some threshold ϵ or the algorithm reached a specified number of iterations.

KLT is a multi-level approach. Each pyramid level requires downsampling the image by a factor of 4^l where l is the number of levels. The optical flow is calculated at each level and provided as a first approximation at the next level.

1.2.2 Pose Optimization And Depth Triangulation

With the features available from the target frame to a nearby frame, we now explain our approach for pose estimation with the `gtsam` (python bindings) framework [11].

`gtsam` is a library that implements factor graphs for sensor fusion in robotics. Fac-

tor graphs are a set of probabilistic graph models. A factor graph is a bipartite graph with two types of nodes, `factors` and `variables` [8]. The variables are unknown random variables in the problem and the factors are probabilistic constraints on the variables, taken from sensor measurements or prior knowledge. In our experiments, the variables are the camera poses, and the factors are the IMU measurements and features constraints.

For the sake of robustness, we assume that each track contains three keyframes (the start, mid and last frame) which we will refer to as f_0, f_m, f_l. The feature tracks are added as a `SmartProjectionPose3` factor to the factor graph. We assume here that the IMU bias is precomputed and stored for every sequence and available. The bias is computed by averaging the first two seconds of data assuming that the vehicle is level. We also assume that the initial rotation of the target frame is available and add that as a `PriorFactor`. This assumption is necessary to account for gravity in the inertial measurements. The IMU data is added to the factor graph as a `PreintegratedIMUFactor` [12] between f_0 and x_m and between f_m and f_l. The poses are then recovered by running Levenberg Marquardt optimization on the factor graph. Then we triangulate depth values for our feature points with DLT on the optimized poses.

Our factor graph is shown in Figure 1.2. Variables representing camera poses are denoted by by x_i and factors representing IMU data and feature positions are denoted by f_i and l_i respectively.

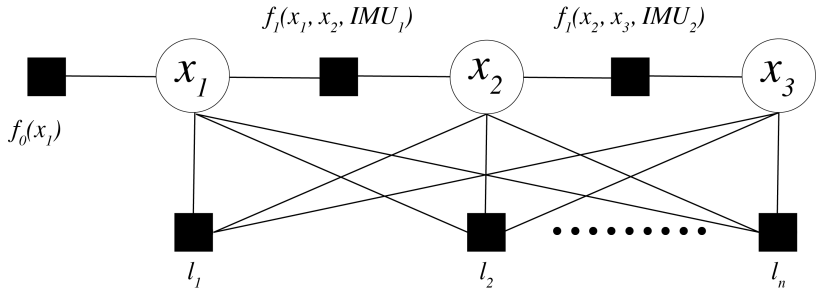

Figure 1-1: Black circles denote factors while white circles denote variables

The advantages of our approach are that the target frame is always a keyframe and keyframes in subsequent frames are randomized for learning.

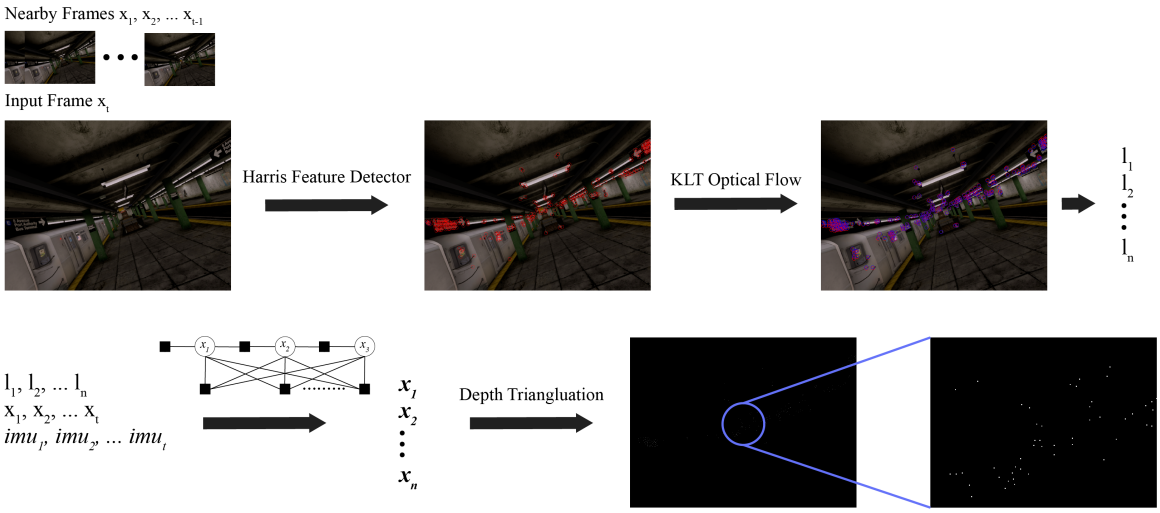

Figure 1-2: The input image is first run through our feature detector and tracker. The points are used to constrain the factor graph pose optimization. Then we triangulate feature points in 3D space to recover depth values. $\boldsymbol{x_i}$ is our optimized poses.

We provide a complete overview of our VIO system in Figure 1.3. The input frame is an actual example from our dataset. For this particular frame, there are around 200 feature points, and more than 150 sparse depth input points.

1.3 Encoder Decoder Architectures Approach

The encoder decoder network architecture contains two sub-networks: the encoder and the decoder. The encoder maps the input data to a low dimensional feature dense state. The decoder then unravels the state and predicts an output. The network architecture is illustrated in Figure 1.3. The architecture works on both for natural language processing (NLP) and computer vision tasks.

In NLP, encoder decoder architectures have been effectively applied for neural machine translation and more broadly, sequence to sequence problems [37]. In computer vision, they have been effectively applied to semantic segmentation [25] and more recently to depth completion [23, 29, 30].

The encoder compresses the input data. There are a few deep learning operations that do that: convolution, average/max pooling and weighted reductions. The operations are implemented in the convolution, average/max pooling and fully connected

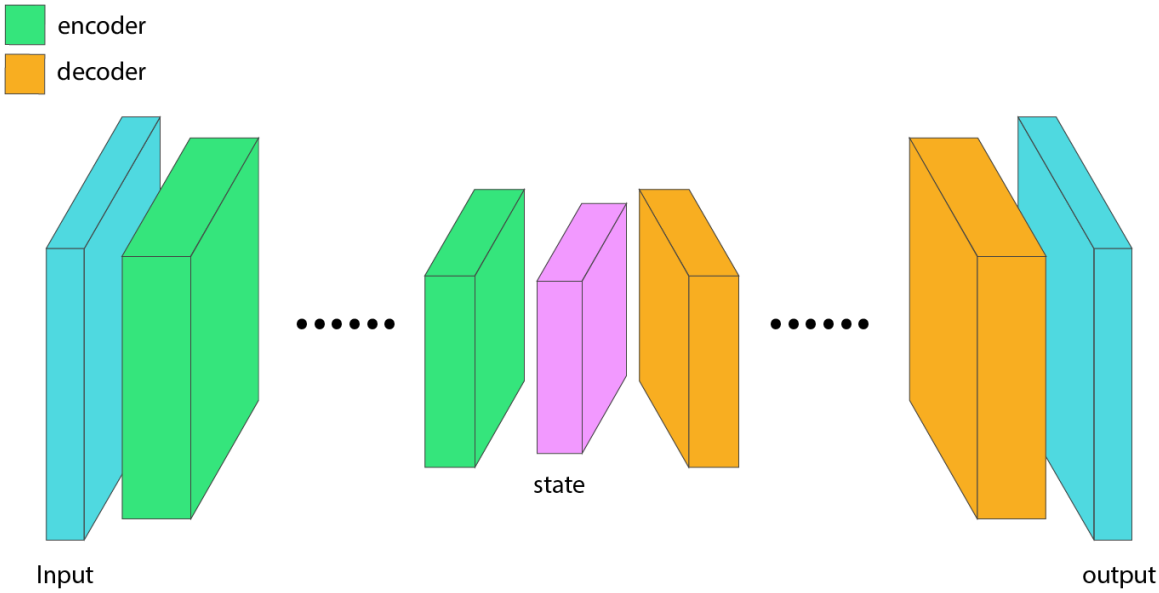

Figure 1-3: Overview of encoder-decoder neural networks. Here, the input is sent through a sequence of encoder layers which output a feature-rich compact state representation. That is then upsampled by a sequence of decoder blocks.

(FC) layers.

The convolution layer is specified by the filter dimensions $H \times W \times D$ where H, W are the filter height and width and D is the number of input data channels, and c is the number of output channels. The convolution layer convolves the input data with the filter f at each spatial location. Each filter creates one output channel, so for c output channels, there are c filters.

Let $x^l_{i,j,d}$ denote the input x from the l layer in the network, i, j be the pixel location and d be the channel. Let $y^{l+1}_{i,j,d}$ correspondingly denote the output y at the $l+1$ layer.

$$y^{l+1}_{i,j,d} = \sum_{i=0}^{H} \sum_{j=0}^{W} \sum_{d^l=0}^{D^l} f_{i,j,d^l,d} \times x^l_{i,j,d^l}$$

The dimensions of the output data can be described in the equation below where O is the output, P is the padding, and S is the stride.

$$O_w = \frac{I_w - F_w + P_w}{S} + 1$$

$$O_h = \frac{I_h - F_h + P_h}{S} + 1$$

The pooling layer is specified by its operation dimensions $H \times W$. It operates on each channel independently and the number of output channels is equal to the number of input channels. There are two types of pooling: average and max pooling that take the average and maximum value of the relevant spatial locations.

$$y^{l+1}_{i,j,d} = \max_{0 \leq i \leq H, 0 \leq j \leq W} x^{l}_{i \times H + i, j \times W + j, d}$$
$$y^{l+1}_{i,j,d} = \frac{1}{HW} \sum_{0 \leq i \leq H, 0 \leq j \leq W} x^{l}_{i \times H + i, j \times W + j, d}$$

FC layers can also be represented similarly as convolution layers, except H, W are over all I_W, I_H, I_D neurons of the input data. However FC layers are too computationally prohibitive to serve as encoder layers and typically are the output layers in neural architectures [22, 24, 38].

Pooling layers reduce the input dimensions by a factor of HW. In practice, pooling layers are 2×2 and so reduces the input by a factor of 4. Convolution filters are typically 3×3 and with $S = 2, P = 1$, the input dimensions is also reduced by a factor of 4. Although max pooling layers don't require parameter learning like convolution layers, convolution layers are preferred as encoders. They don't throw away data like max pooling layers and their filter weights are more data driven than the uniform weighting in average pooling.

Let's now look at decoder layers. Our networks predict depth maps on RGB input, so the resolutions of the RGB input and the depth maps have to match. To ensure that, our networks always have equal numbers of encoder and decoder layers. Some well known decoder layers are deconvolution, upconvolution and upprojection [23]. Previous work by [29] show network accuracies doesn't really change with the decoder layers.

Consequently, our experiments work with the deconvolution layer. To understand it, let's revisit convolution. Consider a 2×2 filter with weights w_{ij} acting on a 3×3 input x. Another way we can consider the operation is via matrix multiplication.

Let's unroll x into a column vector $\mathbf{x}$ and represent our filter as a sparse matrix C.

$$C = \begin{bmatrix} w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 & 0 & 0 \\ 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & 0 & w_{1,0} & w_{1,1} \end{bmatrix}$$

$$x = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2} \\ x_{1,0} & x_{1,1} & x_{1,2} \\ x_{2,0} & x_{2,1} & x_{2,2} \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} x_{0,0} & x_{0,1} & x_{0,2}x_{1,0} & x_{1,1} & x_{1,2}x_{2,0} & x_{2,1} & x_{2,2} \end{bmatrix}$$

Let's now apply the convolution matrix on $\mathbf{x}$ and check what happens.

$$\begin{aligned} y &= C\mathbf{x}^T \\ y_{0,0} &= w_{0,0}x_{0,0} + w_{0,1}x_{0,1} + w_{1,0}x_{1,0} + w_{1,1}x_{1,1} \\ y_{0,1} &= w_{0,0}x_{0,1} + w_{0,1}x_{0,2} + w_{1,0}x_{1,1} + w_{1,1}x_{1,2} \\ y_{0,2} &= w_{0,0}x_{1,0} + w_{0,1}x_{1,1} + w_{1,0}x_{2,0} + w_{1,1}x_{2,1} \\ y_{0,3} &= w_{0,0}x_{1,1} + w_{0,1}x_{1,2} + w_{1,0}x_{2,1} + w_{1,1}x_{2,2} \end{aligned}$$

All we have to do is reshape y back into a 2×2 matrix to match the dimensions we expect. Therefore, the convolution matrix is another way to write the convolution operation. The deconvolution operation is taken by multiplying the input with C^T. Here, if we multiply $C^T y$, our output is a 3×3 input. The transpose convolution of a 3×3 convolution with $S = 2, P = 1$ which downsamples the input by four would correspondingly upsample the input by four.

In our work, our encoder contains convolution layers and downsampling residual blocks from the ResNet architecture [16] and our decoder contains deconvolution layers.

Chapter 2

Depth Completion with Sparse Depth

This section describes our preliminary work for depth completion. First we provide an overview of various benchmark datasets for depth completion, detail our network architecture and training framework, and present findings from experiments on one trajectory in the Blackbird dataset.

Specifically we take a network architecture that was previously state of the art on two benchmark datasets (NYU Depth and Kitti) and apply it to a series of flightpaths on the Half Moon trajectory in Blackbird.

2.1 Datasets For Depth Completion

Researchers typically benchmark their depth completion approaches on a few common datasets: the SYNTHIA Dataset, the NYU Depth V2 Dataset and the Kitti Odometry Dataset [32] [31] [4]. The SYNTHIA (Synthetic Images for Semantic Segmentation of Urban Scenes) dataset is a synthetic dataset containing four video sequences of around 50K frames with corresponding depth maps. The data resolution is 960 $\times$ 720.

The NYU Depth dataset contains around 48K RGB and depth image pairs generated from 464 indoor scenes with a Microsoft Kinect. The depth samples are very dense for each input image with more than 95% coverage. The data resolution is 640 $\times$ 480 pixels.

In contrast, the Kitti Dataset contains around 46K RGB and depth image pairs generated from 22 outdoor road sequences with a Velodyne HDL-64E sensor. The depth samples are much sparser, with around 5% coverage. The depth resolution is also sparser at 912 $\times$ 228 pixels because the sensor scans from the ground up.

2.2 The Blackbird Dataset

For our experiments, we work with the Blackbird Dataset. Blackbird is rendered with FlightGoggles, a framework for photorealistic flight simulation with Unity and ROS [13]. Our experiments are conducted on an initial rendering of the dataset containing 15 different trajectories at varying maximum speeds with 162 flight paths overall. At the time of this writing, Blackbird was expanded to an official version with 18 trajectories and 176 flight paths [5]. The trajectories are rendered in 5 different Unity environments: Butterfly Apartment, Hazelwood Loft, Museum Pillars, Museum Sphinx, NYC Subway. The environments, flight paths, and sample RGB/depth data are displayed in Figures 1, 2 and 3.

Although the Blackbird Dataset is synthetic, the sensor data is real. For each flight path, a quadcopter with a suite of sensors is flown to obtain 100Hz IMU (inertial measurement unit) data. The quadcopter's position is determined to millimeter precision at 360Hz with an array of motion capture cameras. Then, the position data is applied in Unity to render photorealistic images and depth maps at 60Hz.

Due to its rich sensor and camera data, the Blackbird Dataset is uniquely suited to our depth completion experiments because it contains sequential RGB data and IMU data.

2.3 Related Work

Researchers first approached the depth completion problem with RGB data. Saxena et al. applied a markov random field approach to learning depth at the local and global scale with multi-scale image patches [33]. They then refined their approach

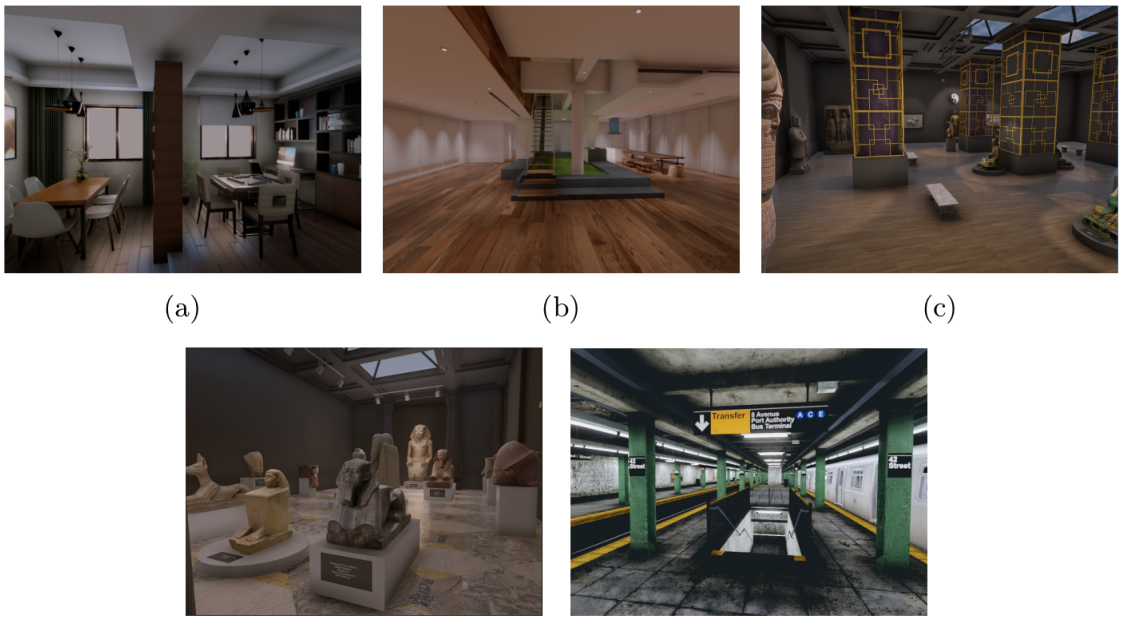

Figure 2-1: Figure reproduced from [5] depicting the Unity environments: (a) Butterfly Apartment, (b) Hazelwood Loft, (c) Museum Pillars, (d) Museum Sphinx, and (e) NYC Subway.

in [34] for general scenes. Others have taken non-parametric approaches with depth databases such as combining depths of images similar to the input image as a prediction. Similarity metrics include SIFT matching [20] or photometric content. [21].

Then, deep learning was successfully applied in various approaches. Eigen et al. first adopted deep learning to depth completion with two deep network stacks: one that makes a rough global prediction with the entire image, and another that refines the prediction locally [9]. Laina et al. later achieved higher accuracy with a deep residual network architecture [23].

Recently, some researchers started incorporating sparse depth samples. Ma et al. incorporated sparse depth samples into their residual architecture and improved performance [29]. They then integrated other objective functions such as the smoothness and photometric functions [30]. Other approaches involve novel convolution layers because convolution layers are not equipped to handle sparse inputs. The current state of the art learns content dependent and spatially variant filters [19]. The prior state

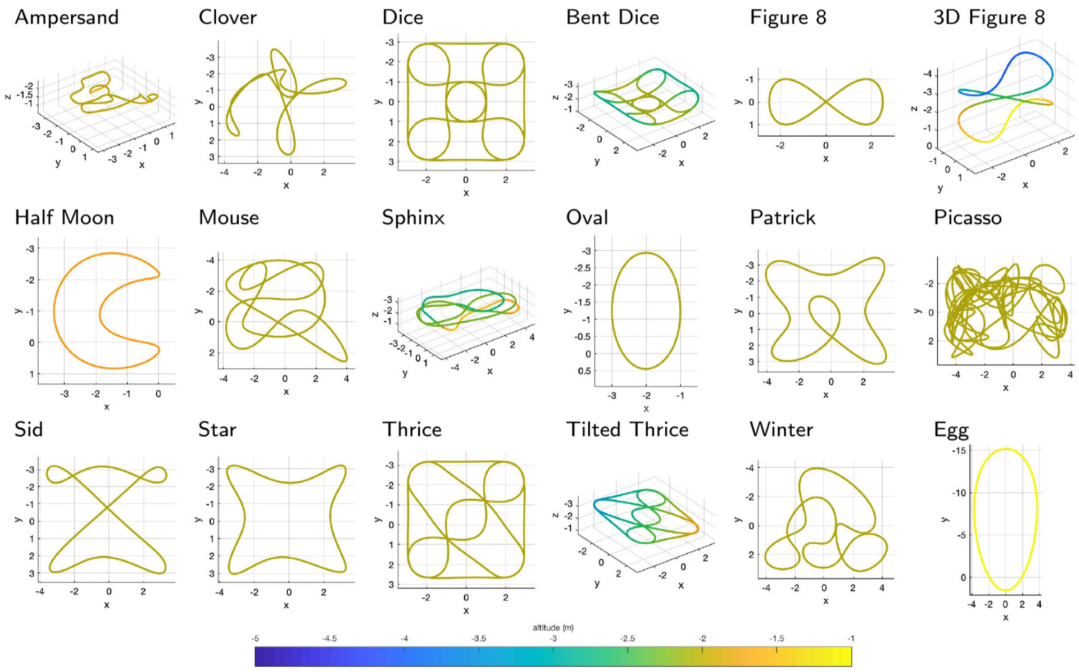

Figure 2-2: Figure reproduced from [5] depicting all the flight trajectories.

of the art was an approach with sparsity invariant convolution layers [40].

2.4 Overview of Network Architecture

Our network architecture is equivalent to the encoder-decoder architecture from [30]. Our network input is the RGB ($H \times W \times 3$) or RGB-d ($H \times W \times 4$) data. RGBd data is the concatenation of RGB and sparse depth points which are sampled from the target depth maps.

Our encoder contains two convolution layers and five downsampling layers. The two convolution layers separately convolve the RGB and depth data. The activations are then concatenated and downsampled five times. Each downsampling layer reduces the data by a factor of 4 while doubling the output filter channels. Four of the downsampling layers are from the ResNetX architecture (in this work, `resnet18|34` for computational costs). The fifth downsampling layer is a block of a 3×3 convolution with $S = 2, P = 1$, batchnorm and ReLU layers.

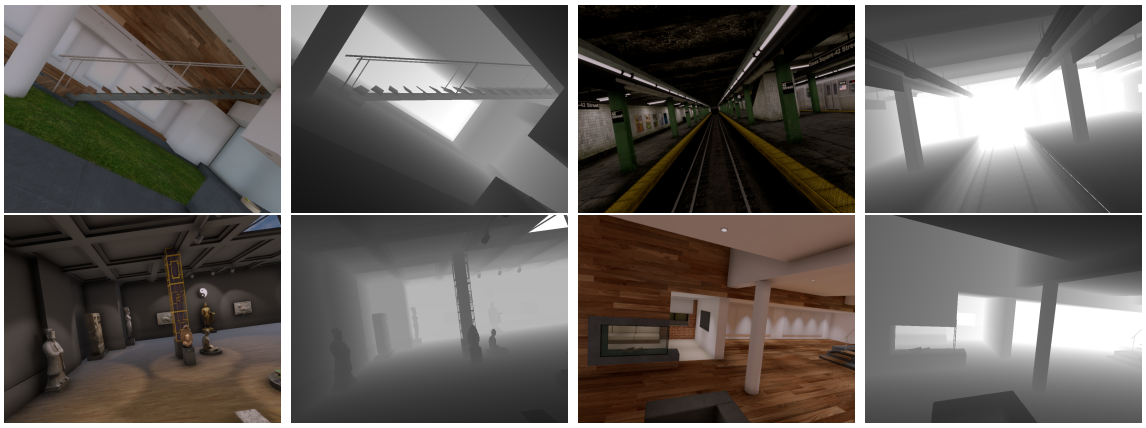

Figure 2-3: Examples of various RGB/depth data from Blackbird.

There are also five decoder layers that each upsamples their input representation, upsampling the data by a factor of 4 while halving the output filter channels. Every decoder layer is a 3×3 transposed convolution (`deconv3`) with $S = 2, P = 1$ followed by a batchnorm and ReLU . We also add skip connections between the encoder and decoder layers to boost accuracy. There are five skip connections and the corresponding activations are concatenated to the input of each decoder layer.

The network architecture is shown in Figure 4. The decoder blocks are coded orange, the encoder blocks are both purple and green. The relative sizes of the layers denote their upsampling and downsampling operations. When referring to networks, we concatenate the encoder and decoder sections so `resnet18-deconv3` refers to a network with ResNet18 for the encoder layers and 3×3 deconvolution layers.

2.4.1 Overview of Sampling Strategy

There are four sparse depth sampling strategies that we apply. They are NONE, UNIFORM, GFTT, VIO. Here, we provide an overview of each. Our experiments in Chapter 2 use UNIFORM and NONE sampling strategies. Our experiments in Chapter 3 predominantly use VIO and NONE, but also UNIFORM and GFTT.

None Sampling

None is simply not sampling any points and only working with RGB data. Our network architecture is a bit different than the one specified in Figure 3.4 for RGB

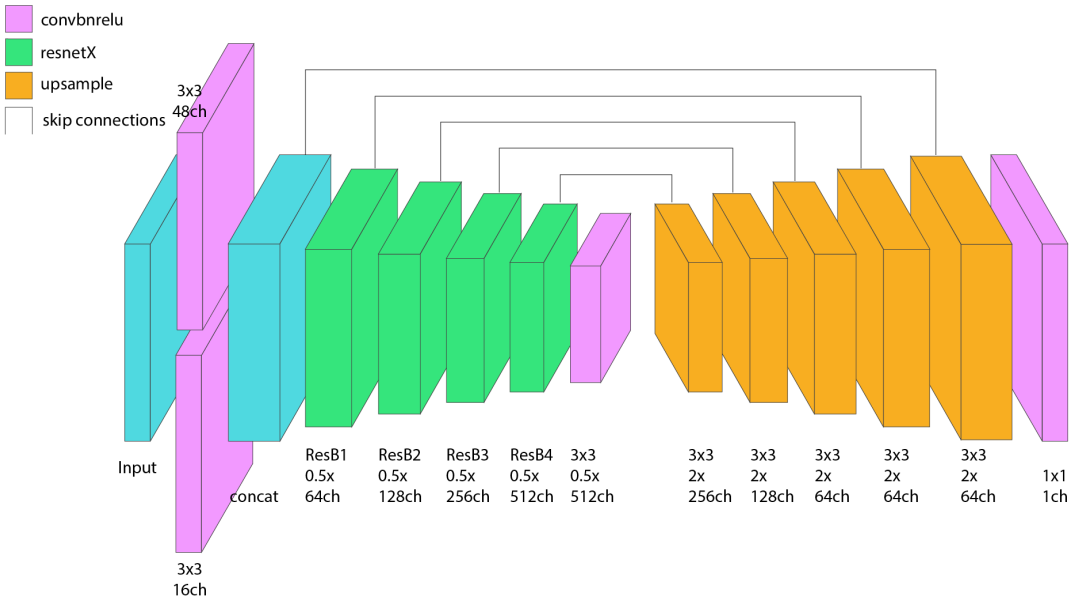

Figure 2-4: Architecture of our sparse to dense network. There are 4 encoder layers from the ResNetX architecture and one 3×3 conv+bn+relu encoder block. The 1×1 conv+bn+relu block after the five decoders reduces the output to one channel. Each pixel location in that channel represents a depth value.

networks. Rather than concatenating the output from two conv+bn+relu blocks (in purple) that separately learn the RGB and sparse depth data, we have only one conv+bn+relu block to learn the RGB data.

Uniform Sampling

For uniform sampling, depth points are randomly sampled from the target depth according to a Bernoulli distribution with probability $p = n/m$ where n is the number of sample points and m is the number of valid points. For a target depth map P with coordinates i, j, the input sparse map P^* is,

$$P^*(i,j) = \begin{cases} P(i,j) & \text{with probability } p \\ 0 & \text{otherwise} \end{cases}$$

In expectation, our sampling strategy returns n depth points. However, on each data point, the input depth points are randomized, forcing our network to be robust across a variety of sparse depth configurations.

We work with $n \in [200 \dots 1000]$ for a fair comparison with models trained with the GFTT and VIO sampling strategies.

GFTT Sampling

For GFTT sampling, we find features with the Shi-Tomasi feature detector. Let $f = (x, y)$ be the set of pixel locations that correspond to features. We then create an output mask o over the input image where

$$o(i,j) = \begin{cases} 1 & \text{if } (i,j) \in f \\ 0 & \text{otherwise} \end{cases}$$

Finally, our input sparse map $P^*(i,j)$ is

$$P^*(i,j) = \begin{cases} P(i,j) & \text{if } o(i,j) \\ 0 & \text{otherwise} \end{cases}$$

VIO Sampling

VIO sampling involves tracking features across successive RGB frames with Shi-Tomasi, creating a factor graph with the camera poses and IMU data in GTSAM and performing Direct Linear Transform (DLT) to recover scaled depth points.

2.4.2 Training Framework

Our network is trained with an Adam Optimizer with 0.9 momentum and 1×10^{-4} weight decay, learning rate of 1×10^{-3} , with a learning rate of 1×10^{-4}, batch size of 8 and n random depth samples (where $n \in [100, 500]$) over 10 epochs.

We also augment our input data with the following transformations:

1. *Scale*: RGB images are scaled by s and depth maps are divided by s for $s \in [1, 1.5]$.
2. *Color Jitter*: the brightness, contrast and saturation of the RGB images are scaled by $k \in [0.6, 1.4]$.

3. *Rotation*: RGB images and depth maps are rotated by $r \in [-5, 5]$.

4. *Flip*: RGB images and depth maps have a 50% chance of horizontal flips.

5. *Crop*: Center crop of the image is taken

2.4.3 Loss Function

Depth completion is a regression task for which ℓ_p norm functions are a common objective function. ℓ_p functions can be described as $\ell_p = (|y - \hat{y}|^p)^{\frac{1}{p}}$, where y and $\hat{y}$ are the target and predicted depth maps respectively. Depth completion models are evaluated with the ℓ_2 function, and we also optimize our networks with it.

We also optimize our networks with respect to the smoothness and photometric objective functions, ℓ_s and ℓ_p. For depth completion networks, the smoothness of the predicted depth map is important. Networks with smoother depth predictions have sharper object features and boundaries. The photometric reconstruction of a nearby RGB frame to the current frame also is an important signal about how accurate the depth map is.

For ℓ_s, we penalize the norms of the gradients in the x, y directions.

$$\ell_s = \frac{1}{|p|} \sum_{x,y \in p} |\delta_X p(\hat{x}, y) + \delta_Y p(\hat{x}, y)| \tag{2.1}$$

For ℓ_p, we minimize the difference between the input image at time t, I_t and its reconstruction $\hat{I}_\tau$ from a nearby image I_τ for $\tau \in \{t - \alpha, t + \alpha\}$. For our experiments, $\alpha \in [3, \ldots 10]$.

$$\hat{I}_\tau(x, y) = I_\tau(\pi g_{\tau t} K^{-1} \overline{(x, y)} p(x, y)) \tag{2.2}$$

In the expression above, π is the perspective projection matrix, $g_{\tau t}$ is the change in camera pose from time t to τ, K is the camera intrinsics matrix, $\overline{(x, y)}$ is the position in homogeneous coordinates $[x, y, 1]^T$ and $p(x, y)$ is the predicted depth at pixel location (x, y).

Both I_t and $\hat{I}_\tau$ are converted to grayscale $I_{t_g}, \hat{I}_{\tau_g}$, and the ℓ_1 difference is calculated

at every nonzero pixel location in $\hat{I}_\tau$.

$$\ell_p = \frac{1}{|p|} \sum_{x,y \in p} \delta(p(x,y) > 0) |I_{t_g} - \hat{I}_{\tau_g}| \tag{2.3}$$

Our complete loss function is $\ell = w_2\ell_2 + w_s\ell_s + w_p\ell_p$ with weights w_2, w_s, w_p.

2.5 Experiments on Half Moon

We train `resnet34-deconv3` networks on all but one of the Half Moon trajectories in the Butterfly Apartment environment. The train set consists of four yaw constant flight paths with maximum speeds of 1, 2, 3, and 4 meters/second while the validation set consists of one forward yaw flight path with maximum speed of 2 meters/second.

Our data split avoids model overfitting by selecting the forward yaw flight path as the validation set. That ensures there are images in the validation set that are never in the training set. The forward yaw flight paths are also more different from the constant yaw flight paths than the constant yaw flight paths are from each other, which is why choosing one of the constant yaw flight paths as the validation set is not ideal. With our split, there are 18,464 images for training and 4,532 images for validation. Our training and validation curves are shown in Figure 5 and our best results are shown in Figure 2.1.

Table 2.1: `blackbird` train and val errors.

Model Type	Train (m)	Val (m)
RGB	0.54	1.32
RGB_RANDOM	0.39	0.52

Our best validation errors were `0.52m` and `1.32m` for our RGB_RANDOM and RGB networks respectively. Our findings show that our sparse depth samples significantly improve model accuracy.

Some example predictions are shown in Figure 6. Through inspection, both networks learn the depth values of tables, chairs, couches, walls, and windows well and

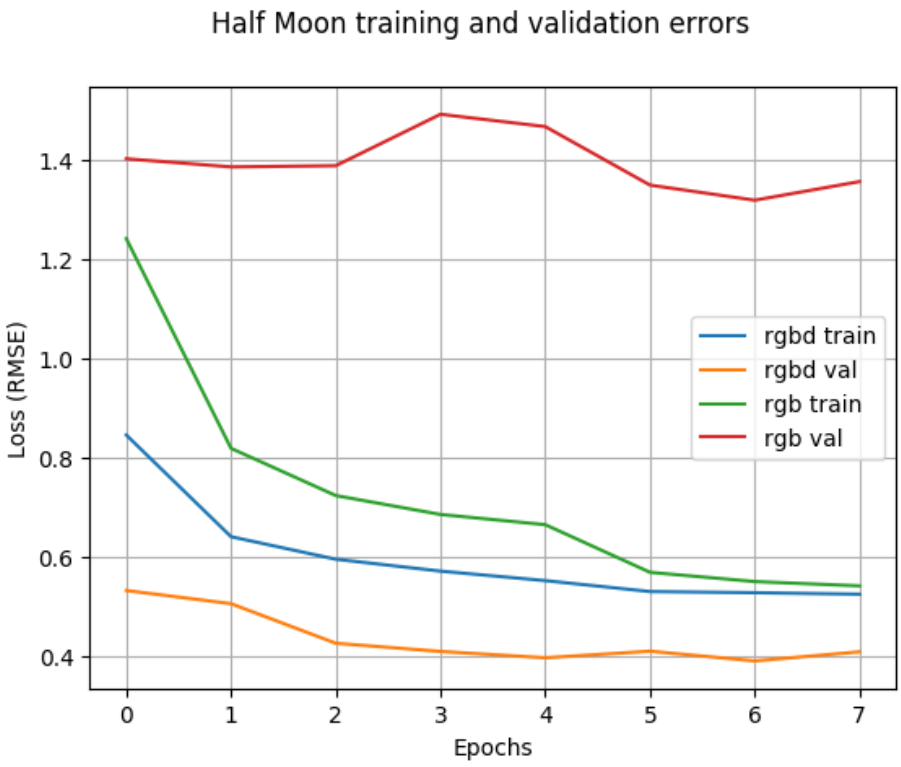

Figure 2-5: RMSE on Half Moon for both RGB and RGB_RANDOM networks. The validation error is significantly lower than the train error for RGB_RANDOM.

struggle with complex items that appear infrequently in the training data such as bookshelves and cabinets.

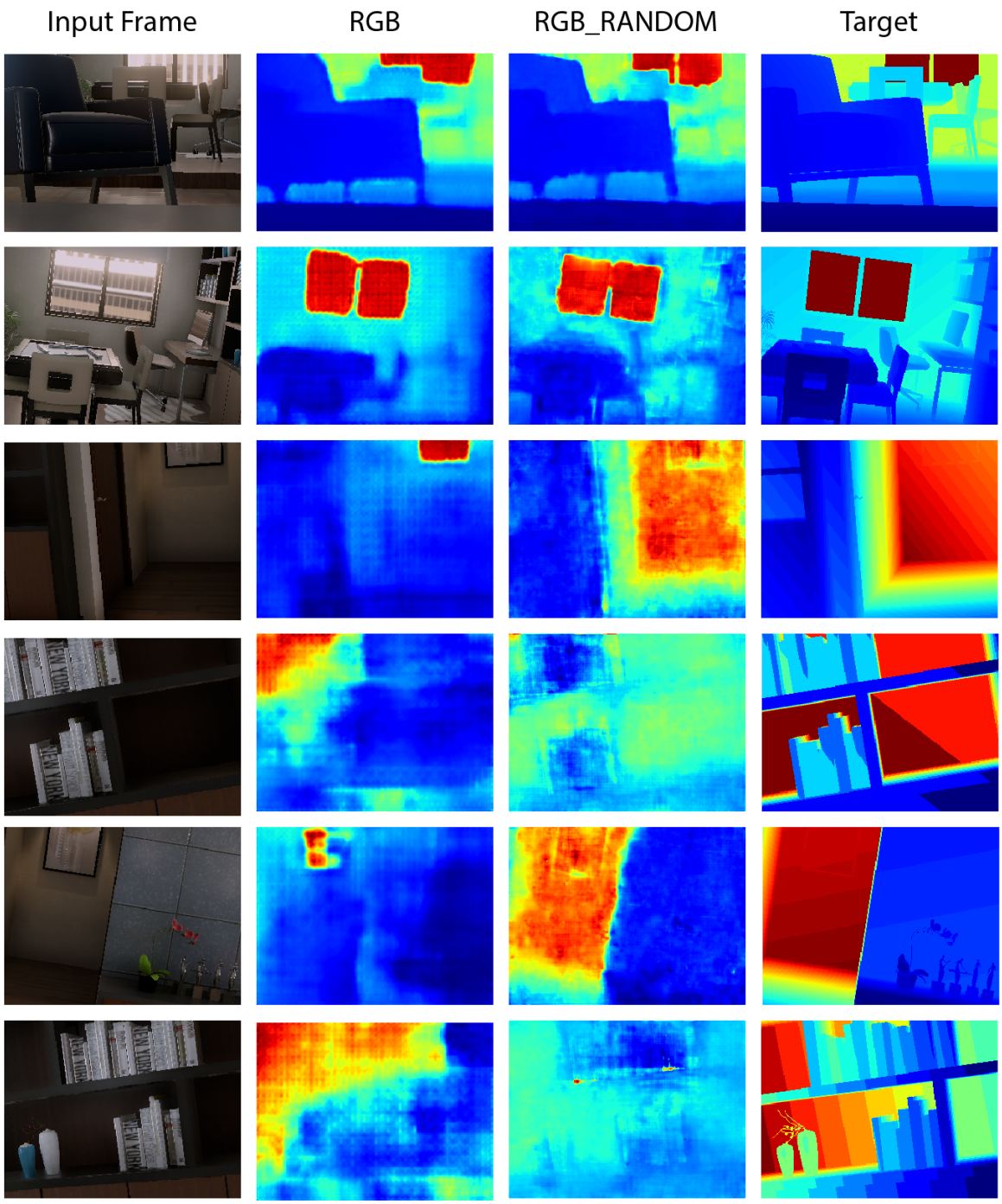

Figure 2-6: Predictions of our encoder-decoder architecture on the Half Moon trajectory. Our network learns the contours of walls, windows, tables and chairs, but struggles with more complex objects such as bookshelves.

Chapter 3

Motion-aware Monocular Depth Completion

This section describes various experiments on Blackbird. The previous chapter shows that our network architecture performs well with input sparse depth samples. However, it is not possible to sample sparse depth samples with a monocular camera. Aerial vehicles are also constrained by size and weight from carrying more sensors.

In this chapter, we explore extensions of our work with depth data from VIO generated with sequential monocular camera and IMU data. First, we present an overview of our VIO setup. Then we present findings from our experiments.

3.1 Related Work

Researchers have applied motion algorithms to depth completion in a variety of ways. One approach works with optical flow: concatenating it with RGB data to predict depth or learning it along with dense depth on unsupervised tasks [28] [43].

Other approaches involve predicting camera pose with algorithms like visual odometry or deep learning. For the latter, network architectures predict on a single RGB input or a sequence of RGB frames [2] [35]. Pose information forces the depth network towards more accurate predictions. Some works have applied it to refine network predictions by optimizing the photometric reconstruction error of the predicted depth to

nearby frames [6] [41].

Pose networks can also be applied directly to VIO algorithms to generate scaled sparse depth points as part of the input data [3]. That is the approach we also take in this work.

3.2 Overview of Our VIO Setup

There are two parts of our VIO setup: feature tracking and depth triangulation.

For each input RGB frame t, we select nearby frames $f_{t-\tau}, \tau \in [3 \dots 5]$. We run Good Features to Track (GFTT) on the first frame with 2000 features, a quality threshold of 0.05 and minimum feature distance of 5 [36]. If there are fewer than 5 features in the first frame, we don't continue.

Otherwise, at each following frame, we run Lukas-Kanade optical flow on the points we previously tracked with GFTT. The parameters for our optical flow 3 pyramid levels, a window size of 7, and a stop criterion of 5000 iterations or change of $< 1e-4$.

Features are rejected if their displacement between frames is more than 15 pixels. We then create a factor graph with each pose as a factor. We supply initial pose values from our tracked features, and constrain successive poses with IMU acceleration and orientation data. Our IMU and RGB data are both timestamped at different frequencies, and we linearly interpolate the IMU data for each RGB input. We solve our factor graph with `gtsam`'s `LevenbergMarquardt` optimizer to recover more accurate poses.

We then take the pose data and recover depth values by triangulating our features points in 3D space with DLT. Points with depth values greater than `19m` are rejected because our target depth maps only measure `10m`.

Our VIO system returns can upwards of 200 sparse depth points, but more often returns fewer than 50 points. There are also a lot of frames where no sparse depth points are found. That occurs when there are few visual features in the scene, or when the camera poses don't change significantly from successive frames. The former

leads to fewer sparse depth points and the latter leads to zero sparse depth points.

3.3 Experiments

Here, we describe our experiments. Let's first run through some terminology. There are five environments in Blackbird. `blackbird_subset` includes only data from Hazelwood Loft, Museum Pillars, Museum Sphinx and NYC Subway. `blackbird` includes data from all five environments.

For `blackbird_subset` and `blackbird`, one flight path from each trajectory is randomly chosen to be included in the validation set.

For all our experiments, we sample every tenth data point in our datasets so that every data point contains nearby frames with which to apply VIO and decrease training time.

3.3.1 Experiments on Blackbird Subset

First we train a series of network architectures on `blackbird_subset`. They include: RGB, RGB_GFTT, RGB_RANDOM and RGB_VIO. The _X here refers to the sparse depth sampling strategy. RGB_RANDOM (with 1,000 samples) refers to uniformly sampling sparse depth points from the target depth map.

RGB_GFTT applies GFTT with 1000 features, a quality threshold of 0.01 and a minimum distance of 10 to select distinct visual locations. The target depth map is then sampled at those locations. RGB_VIO is our VIO approach. Our results are shown in Table 3.1 below.

Table 3.1: `blackbird_subset` train and val errors.

Model Type	Train (mm)	Val (mm)
RGB	331.2	360.0
RGB_VIO	216.9	**287.9**
RGB_GFTT	211.3	327.3
RGB_RANDOM	171.2	184.4

Our findings show that the train and validation error decreases with more input sensor data (sparse depth information). That explains why in order of increasing accuracy, our architectures are RGB, RGB_GFTT, RGB_VIO, and RGB_RANDOM. VIO provides some scaled sparse depth points, and GFTT provides around 500-700 pixel locations to sample from the target depth map, while RANDOM provides in expectation 10000 sparse depth points from the target depth map. GFTT locates more features in some trajectories than others and a complete breakdown by trajectory is shown in Table 3.2 below.

Table 3.2: Number of GFTT features by trajectory.

Trajectory	# of Train Features	# of Val Features
ampersand	562.3	507
bentDice	676.8	625
clover	634.4	488.5
dice	700.4	676.7
halfMoon	587.3	667.4
mouse	564.1	761.2
oval	602.0	814.7
patrick	686.9	643.0
sid	983.9	807.8
sphinx	703.9	603.8
star	657.1	788.8
thrice	695.1	702.0
tiltedThrice	643.9	645.7
winter	715.6	691.2

It is impressive that the training error from RGB_VIO is close to that of RGB_GFTT (`216.9mm` and `211.3mm`) because GFTT samples points from the target depth maps which require depth sensors while VIO generates scaled depth points with only a monocular camera and IMU data. However, neither is comparable with RGB_RANDOM with a train error of `171.2mm`.

RGB_VIO is more accurate than RGB on both the train and validation splits, `331.2` versus `219.4` and `216.9` versus `287.9` respectively. To understand our networks more, let's examine the validation error by trajectory for RGB and RGB_VIO in

Table 3.3.

Table 3.3: Validation error by trajectory for RGB and RGB_VIO.

Trajectory	Env	Yaw	Speed	RGB	RGB_VIO
ampersand	HL	forward	maxSpeed2p0	314.40	**293.54**
bentDice	MP	constant	maxSpeed1p0	276.41	**270.91**
clover	HL	forward	maxSpeed2p0	321.58	**271.93**
dice	MP	constant	maxSpeed2p0	261.85	271.76
halfMoon	HL	forward	maxSpeed3p0	339.05	**290.59**
mouse	NYC	forward	maxSpeed0p5	250.55	**219.60**
oval	HL	forward	maxSpeed4p0	1316.61	**373.64**
patrick	MP	forward	maxSpeed4p0	296.18	304.40
picasso	NYC	constant	maxSpeed4p0	427.38	**419.88**
sid	NYC	forward	maxSpeed1p0	254.19	**226.43**
sphinx	MS	constant	maxSpeed3p0	443.37	**435.43**
star	NYC	forward	maxSpeed1p0	257.70	**227.30**
thrice	MP	constant	maxSpeed6p0	262.45	264.759
tiltedThrice	MP	forward	maxSpeed0p5	238.95	**232.42**
winter	NYC	constant	maxSpeed2p0	309.46	**236.23**

The table shows that RGB_VIO outperforms RGB on 12 of the 15 trajectories. RGB_VIO learns the oval trajectory really well. For RGB, the validation error on oval is `1316.61mm` and for RGB_VIO, it's only `373.64`. That is impressive considering oval contains sequences of frames with very few distinct objects: there are stretches of images with smooth curtains, walls and pillars, some of which are shown in Figure 3.1.

Some predictions between RGB_VIO and RGB on oval and other trajectories are shown in Figure 3.2. RGB_VIO's predictions on oval are significantly more coherent than RGB's. It is more accurate with object contours such as curtains, floors, walls, and can understand complex objects like staircases. See Appendix A, Figure 1 for more images from the oval trajectory, and Figure 2 for a more detailed comparison of RGB_VIO and RGB by environment.

It is interesting to note that all three environments where RGB outperforms RGB_VIO are in the Museum Pillars environment. Some reasons why that might be are described in the next section.

Figure 3-1: Examples of various images on oval.

3.3.2 Experiments on Blackbird

We train RGB and RGB_VIO networks on `blackbird`. `blackbird` contains the Butterfly Apartment environment which was left out of `blackbird_subset`. Butterfly Apartment trajectories are added to the training and validation sets. The results are shown in Table 3.4.

Table 3.4: `blackbird` train and val errors.

Model Type	Train (mm)	Val (mm)
RGB	236.5	309.3
RGB_VIO	**225.5**	**286.6**

Overall, RGB_VIO outperforms RGB on the validation set (`309.mm` and `286.6mm`). Let's look at the metrics per trajectory and per environment in Tables 3.5 and 3.6. RGB_VIO outperforms RGB on every trajectory except bentDice and dice. The two trajectories take very similar flightpaths through the same environment which explains why the RMSE for RGB_VIO is similar on both.

RGB_VIO also outperforms RGB on every environment except Museum Pillars. That's surprising considering Museum Pillars is an environment with rich visual features for VIO. Table 3.5 also shows that the RMSE on Museum Pillar trajectories (bentDice, dice, patrick, thrice and tiltedThrice) don't really change between RGB and RGB_VIO.

The picture becomes clearer when we look at the 4th-6th columns in Table 3.6. The columns show the percentage of overall invalid frames and provide a breakdown by failures. For an input frame, our VIO fails when the pixel displacement for tracked

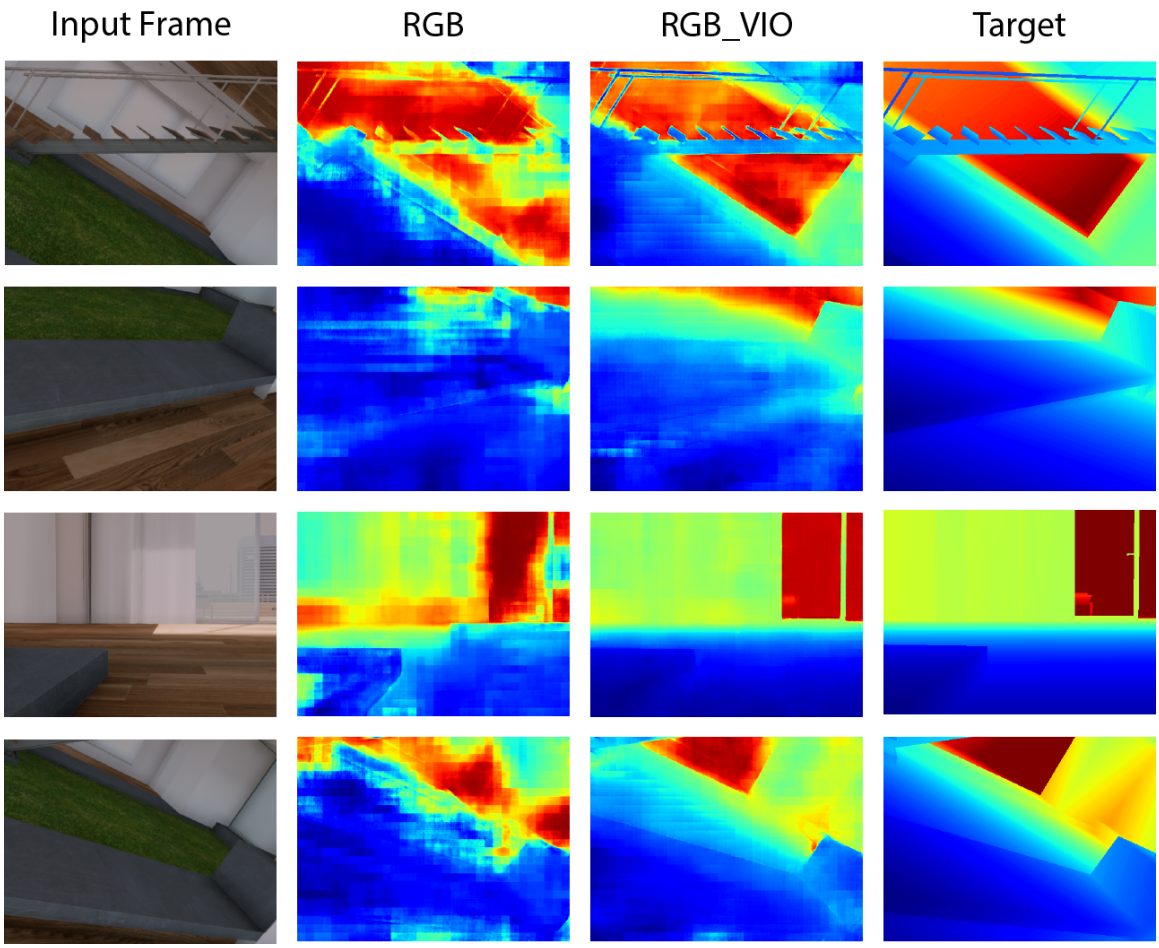

Figure 3-2: Example RGB_VIO and RGB predictions on oval.

features is lower than two pixels or when depth triangulation fails due to poor features. The environment with the highest number of invalid frames and IMU data is Museum Pillars. That explains why RGB_VIO doesn't learn well on it.

The pixel displacement is likely low at the start of every trajectory because the Blackbird drone is starting from a resting stationary position before takeoff. The lack of depth triangulations may be due to low feature count. For our training and validation, these frames are kept in, but a fairer comparison between RGB and RGB_VIO would probably exclude them. That would probably lower our VIO error values even more while keeping our RGB values around the same.

Some example predictions on each environment is shown in Figure 3.2. RGB_VIO is more aware of object boundaries than RGB. The tables, chairs, sofas, walls, pillars

Table 3.5: Validation error in mm by trajectory for RGB and RGB_VIO.

Trajectory	Env	Yaw	Speed	RGB	RGB_VIO
ampersand	HL	forward	maxSpeed2p0	316.31	**294.93**
bentDice	MP	constant	maxSpeed1p0	269.92	282.85
clover	HL	forward	maxSpeed2p0	282.51	**277.84**
dice	MP	constant	maxSpeed2p0	263.30	284.47
halfMoon	HL|BA	forward	maxSpeed3p0	338.66	**316.53**
mouse	NYC	forward	maxSpeed0p5	240.88	**210.70**
oval	HL|BA	forward	maxSpeed4p0	390.07	**362.157**
patrick	MP	forward	maxSpeed4p0	285.08	**281.84**
picasso	NYC	constant	maxSpeed4p0	439.17	**412.70**
sid	NYC	forward	maxSpeed1p0	249.07	**224.03**
sphinx	MS	constant	maxSpeed3p0	490.83	**369.648**
star	NYC	forward	maxSpeed1p0	254.16	**218.55**
thrice	MP	constant	maxSpeed6p0	252.87	**251.76**
tiltedThrice	MP	forward	maxSpeed0p5	251.84	**235.14**
winter	NYC	constant	maxSpeed2p0	263.98	**226.03**

Table 3.6: Validation error in mm by environment for RGB and RGB_VIO.

Env	RGB	RGB_VIO	Invalid	<2p	No Triangulations
Butterfly Apartment	324.71	**314.16**	10.1%	79.4%	20.6%
Hazelwood Loft	346.73	**321.67**	10.4%	80.4%	19.6%
Museum Pillars	264.52	266.96	**13.6%**	**21.5%**	**78.5%**
Museum Sphinx	490.83	**369.65**	4.9%	86.8%	13.2%
NYC Subway	285.63	**254.92**	10.7%	95.4%	4.6%

and sculptures have sharper boundaries for RGB_VIO than RGB. RGB_VIO also understands the scale of the depth more than RGB. Rows 2, 4, and 5 of Figure 3.2 illustrate that: the Hazelwood Loft walls are more blue (indicating a closer distance), and the space behind the pillar and beyond the subway tracks on NYC Subway are more red (indicating a farther distance).

Some more results, by environment are shown in the Appendix Figures, 3-7.

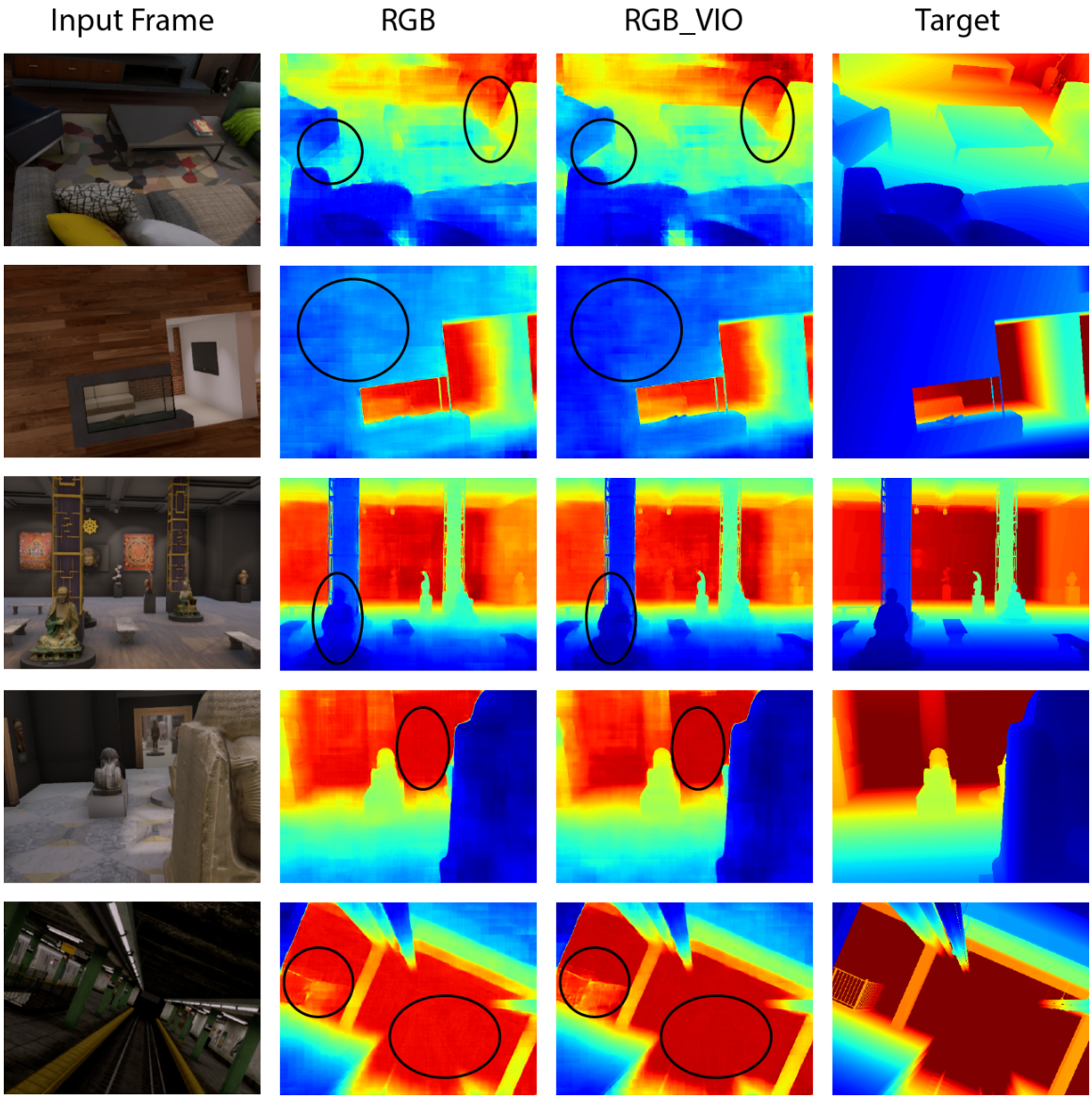

Figure 3-3: Some of our network predictions on the Blackbird dataset.

Chapter 4

Other Experiments

Here we describe experiments on other datasets, namely the Visual Odometry with Inertial and Depth (VOID) dataset [3].

4.1 Experiments on VOID

The VOID dataset contains synchronized RGB frames, sparse depth data and inertial measurements for depth completion [3]. There are three input sparse depth densities: 0.5%, 0.15% and 0.05%. On 0.5%, the authors obtain a RMSE of `169.79mm`. Our best result with a `resnet34-deconv3` architecture is `318.64mm`.

Our network input is RGB data concatenated with interpolated sparse depth points. We applied a similar weighted criterion function to [3] of the ℓ_2 norm function ℓ_d, the smoothness function ℓ_s, and the photometric reconstruction function ℓ_p with weights w_d, w_s, w_p $1.0, 0.1, 0.1$ respectively. Some example predictions are provided Figure 4.1.

The second column in Figure 4.1 is our interpolated sparse depth points. The original sparse depth points are sampled from the target distribution so their interpolation also provides a reliable signal to our network architecture. Sometimes our network predicts the depth effectively like in rows 1 and 2. It also learns the object contours well as in rows 3-6 but doesn't always learn at the right scale. It thinks objects are closer when they are actually farther.

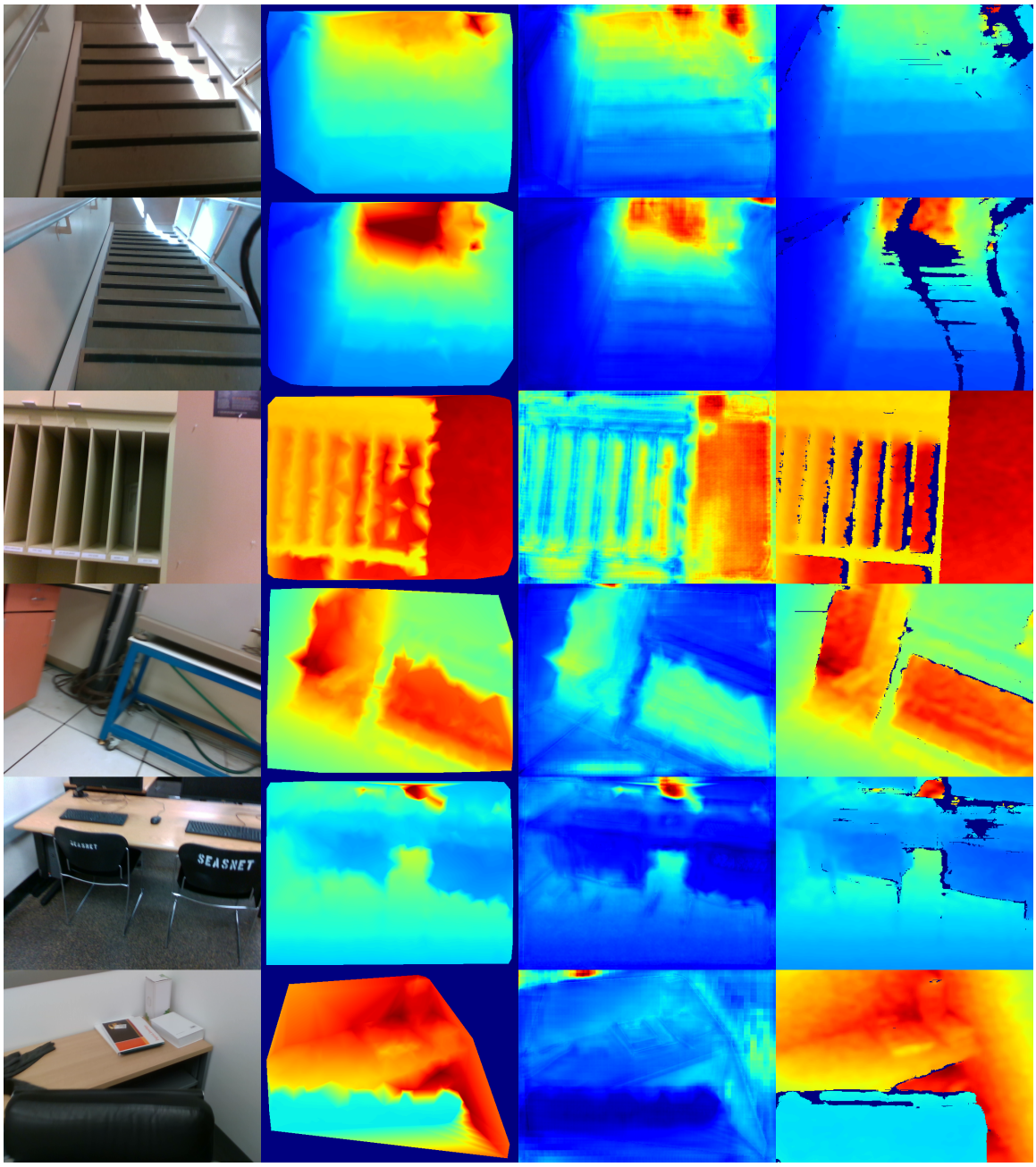

Figure 4-1: Some of our network predictions on VOID. From left to right, Our network often predicts the right object contours but doesn't always predict at the right scale.

4.2 Other Experiments on VOID

We conducted experiments to solve our scale problem. Our network architecture is an early fusion architecture in that the RGB and depth data are combined early on in the network. We created late fusion network architectures because the interpolated sparse depth points look close to the target map. However, that didn't work as well as our original network.

We also experimented with structural similarity index similarity (SSIM) for photometric consistency, and Minkowski sparse input architectures and didn't find significant improvements [7].

Chapter 5

Conclusion

We conclude here with an overview of our results, their significance and directions for future work.

5.1 Overview of Results

We have shown that encoder-decoder network architectures with sparse depth samples from VIO are more accurate than ones with only RGB data. Specifically, RGB_VIO outperforms RGB on both `blackbird_subset` and `blackbird`. The data is shown again in Table 5.1 below.

Table 5.1: `blackbird_subset` train and val errors.

Model Type	Dataset	Train (mm)	Val (mm)
RGB	`blackbird_subset`	331.240	360.0
RGB_VIO	`blackbird_subset`	**224.65**	**287.90**
RGB	`blackbird`	236.5	309.3
RGB_VIO	`blackbird`	**225.5**	**286.6**

Our work is significant because it shows that for aerial vehicles, a monocular camera may be all that is required for accurate depth completion. Furthermore, our system can be optimized to run at frame rates similar to off the shelf stereo depth sensors [42].

Our findings are more significant when considering that around 10% of our validation frames have zero sparse depth points due to low pixel displacement or no depth triangulations. Taking those frames out would probably lower the error for RGB_VIO while keeping RGB around the same.

Furthermore, having fewer sensors also helps aerial vehicles in other categories like manufacturing cost, maximum flight speed, maneuverability, and battery life.

5.2 Future Work

Some directions that are worth exploring are described below.

5.2.1 Exploring Data

It is important to understand our data more, and know how that relates to network accuracy. One question is why does depth triangulation fail 5x more on the Museum Pillars environment than on other environments? Furthermore, what is the significance of that on how RGB_VIO learns on Museum Pillars.

During our visualizations, we found that some depth samples in our environments that have poorly rendered depth. They might contribute adversely to network learning and evaluation. Some are shown in Figure 5.1. They don't look as detailed as other depth maps. They may come from our depth maps being encoded with only 8-bit precision which can lead to depth discretization.

It is likely harder for our networks to learn with discretized depth maps. Even though that should affect the RGB and RGB_VIO networks equally because both are evaluated with the same depth maps, it may be productive to remove those samples in future experiments.

It is also interesting to look at the densities of our depth triangulations per environment and trajectory. Which environments or trajectories have the highest densities? Are there ways to boost the densities?

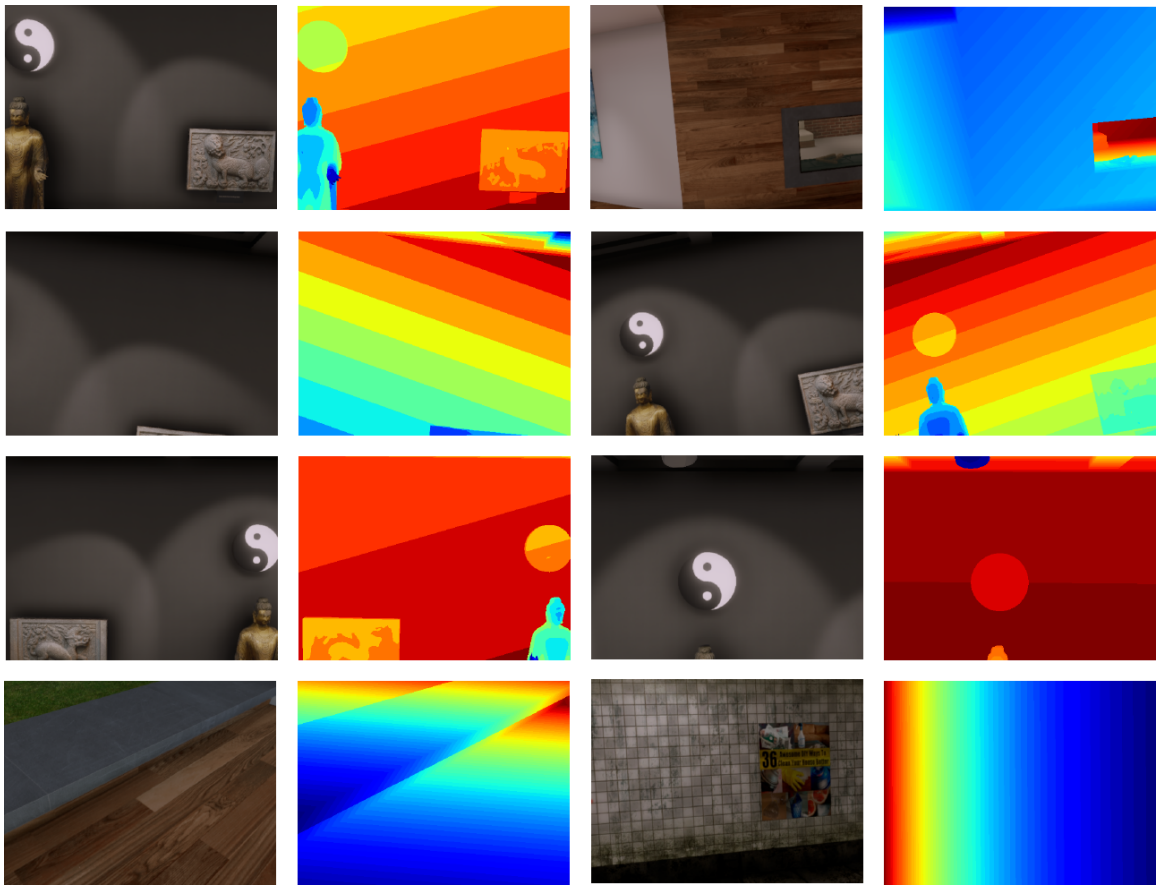

Figure 5-1: Some RGB data with incorrect depth maps. Some from Museum Pillars and NYC Subway are shown here.

5.2.2 Improving the VIO Pipeline

Our VIO setup requires camera pose information to triangulate visual features. Right now, camera pose is calculated from the perspective transformation across RGB frames of the tracked features. However, the accuracy of the pose is contingent on the quality of the tracked features.

Researchers have had success with separate pose network architectures that take an RGB frame or a sequence of frames to predict pose [2] [3]. That is worth exploring and may significantly improve accuracy on frames with fewer visual features where we can't recover pose data. For our setup, the pose network would replace our factor graph optimization approach for pose prediction. We would then apply DLT with the pose information to extract sparse depth values for our feature points.

Our VIO pipeline also depends on finding and tracking robust features. It could also be productive to apply other classical computer vision strategies like Scale Invariant Feature Transform (SIFT) for feature matching [26] or neural architectures for feature tracking and matching.

5.2.3 Blackbird Official Release

We conducted our experiments on initial renderings of the Blackbird Dataset. It could be interesting to run more comprehensive experiments and analyses on the final renderings.

The final renderings also encode depth maps as 16-bits. That may help with our depth discretization problem.

Appendix A

Figures

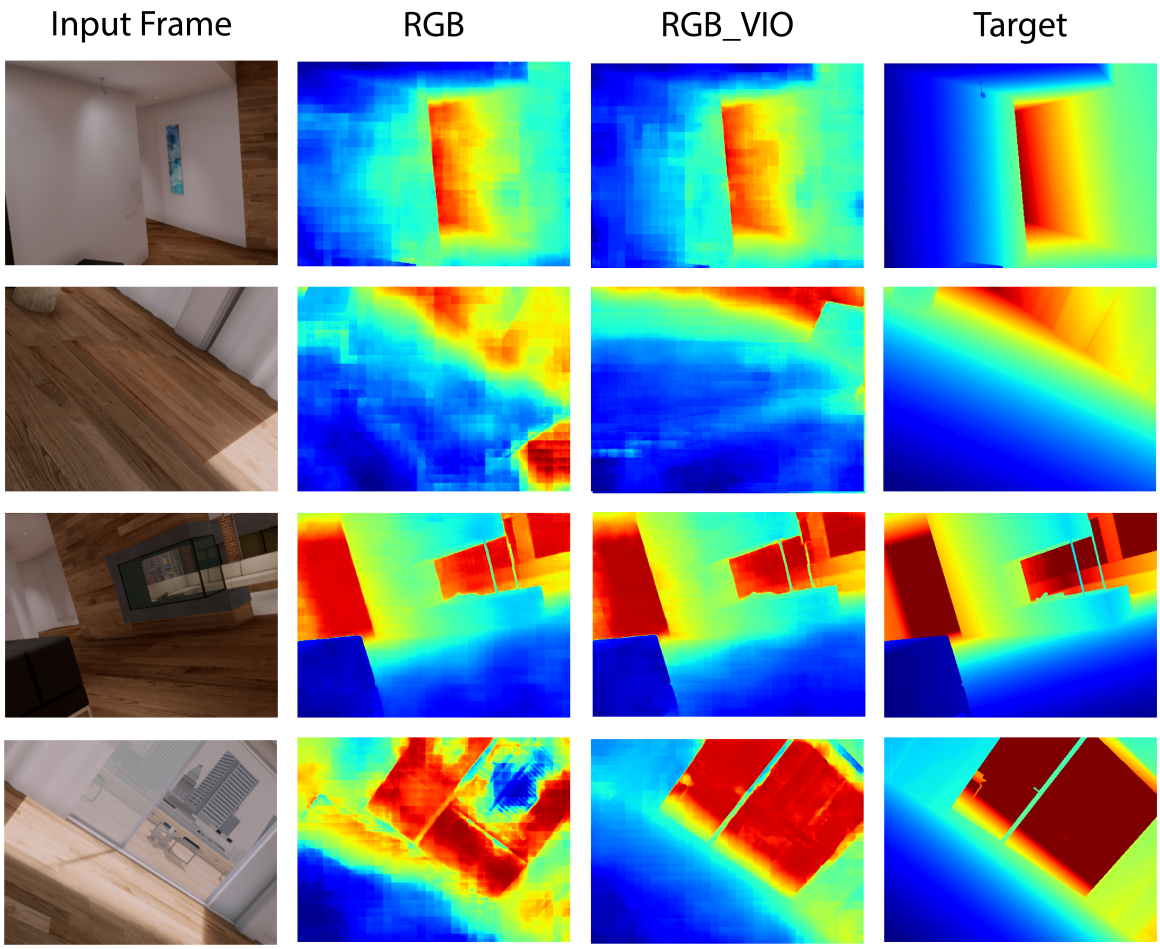

Figure A-1: Sample predictions of RGB and RGB_VIO trained on `blackbird_subset` for the oval trajectory.

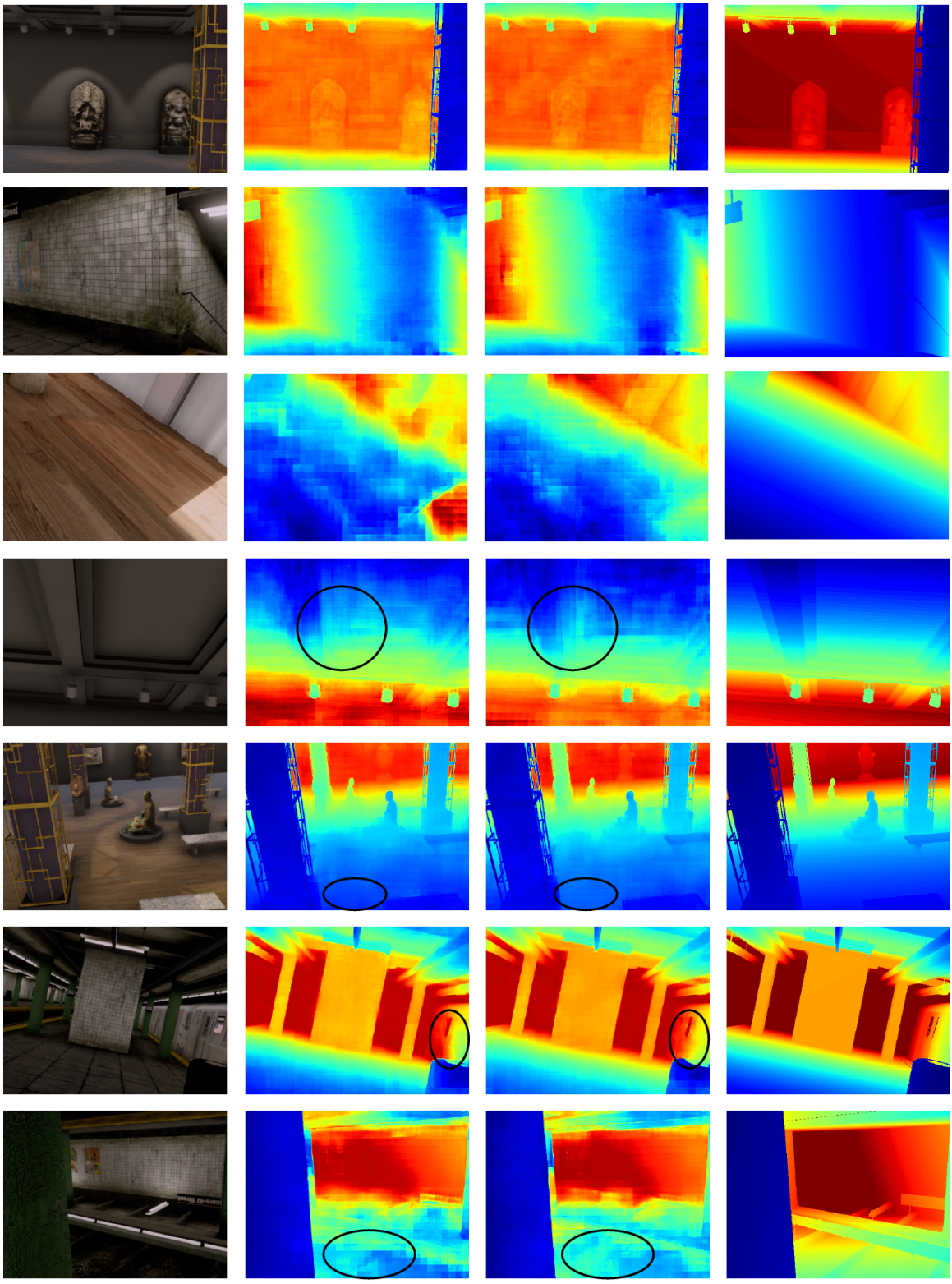

Figure A-2: Sample predictions of RGB and RGB_VIO trained on `blackbird_subset` for all the environments.

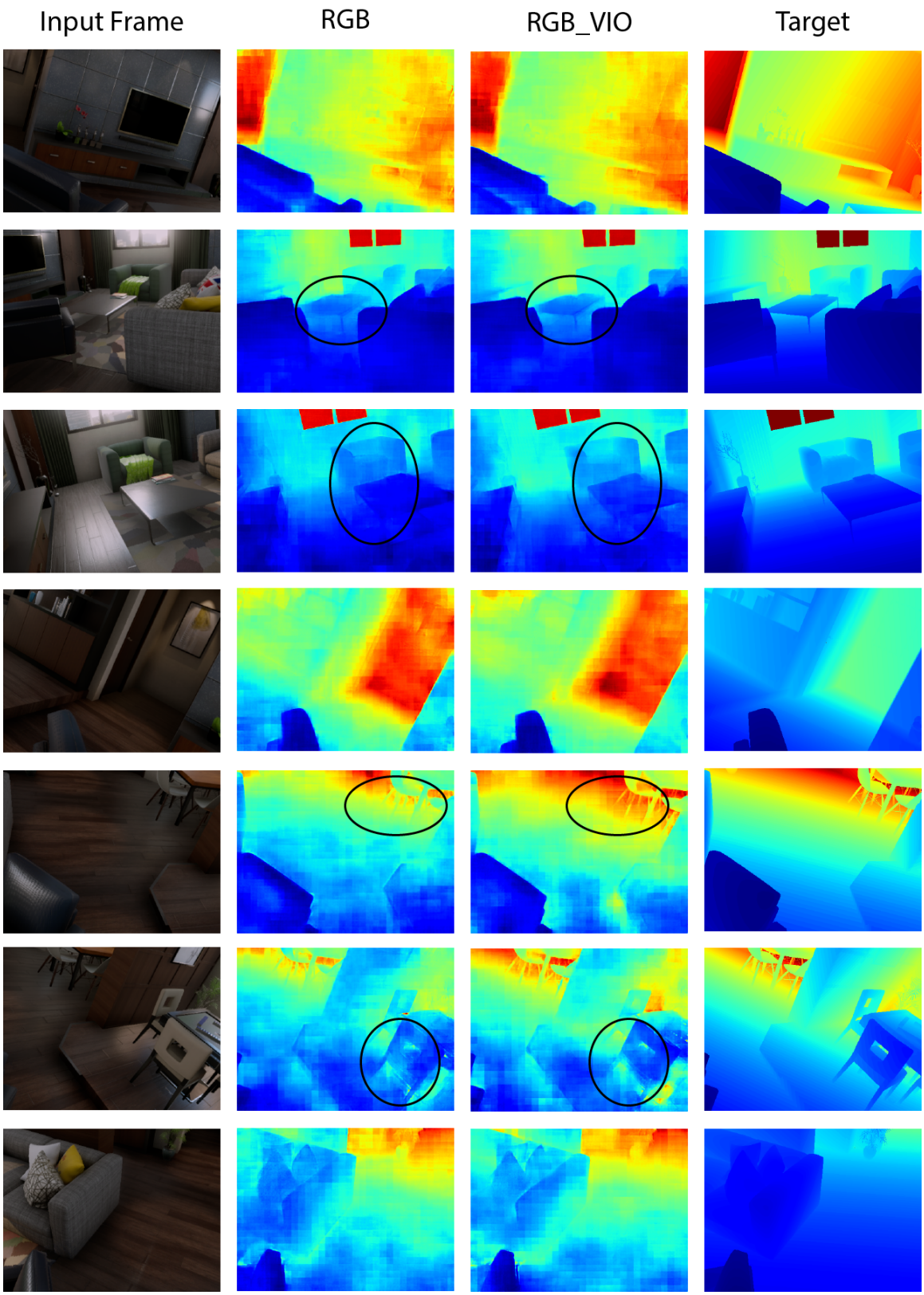

Figure A-3: Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Small Apartment environment

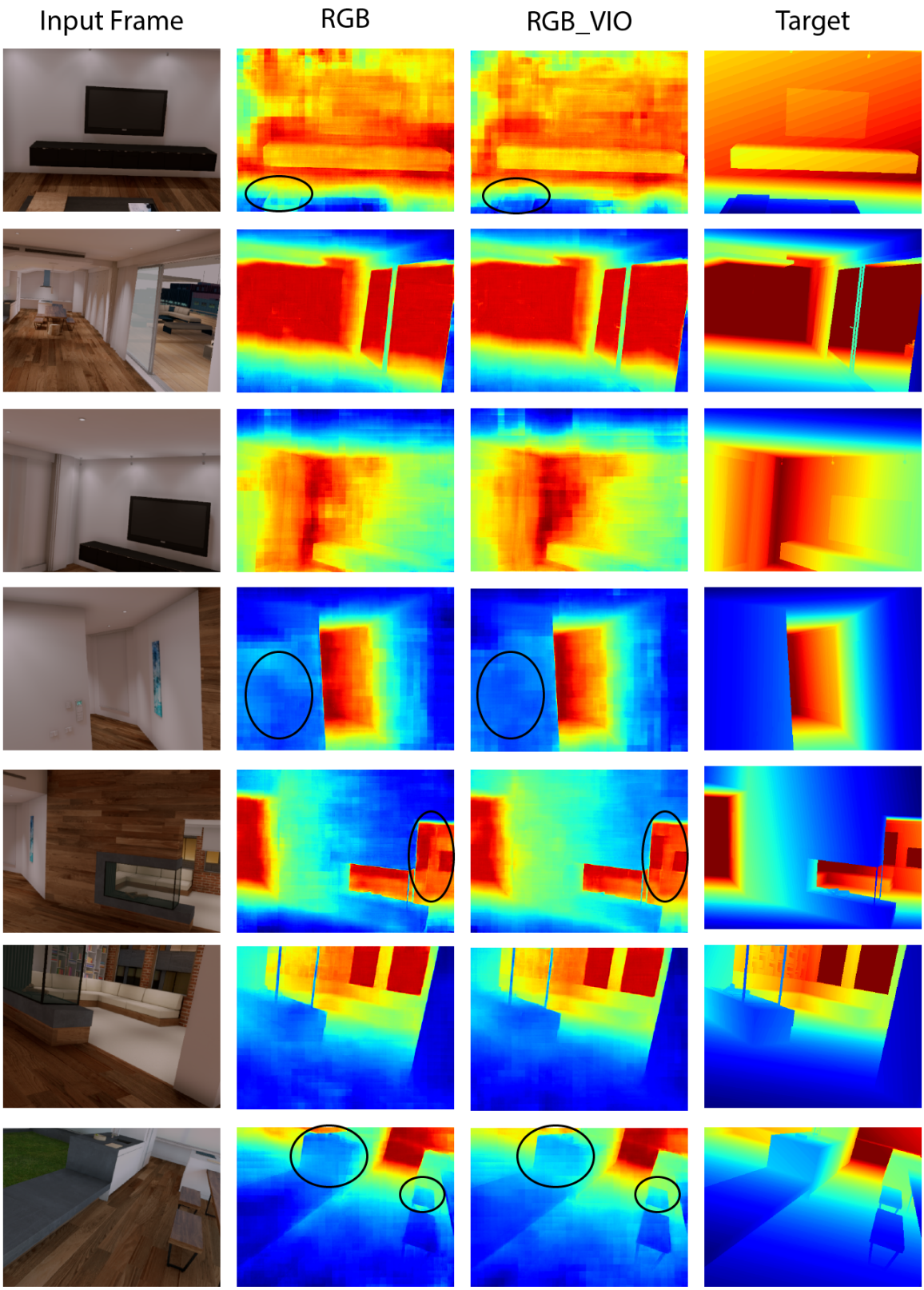

Figure A-4: Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Hazelwood Loft environment

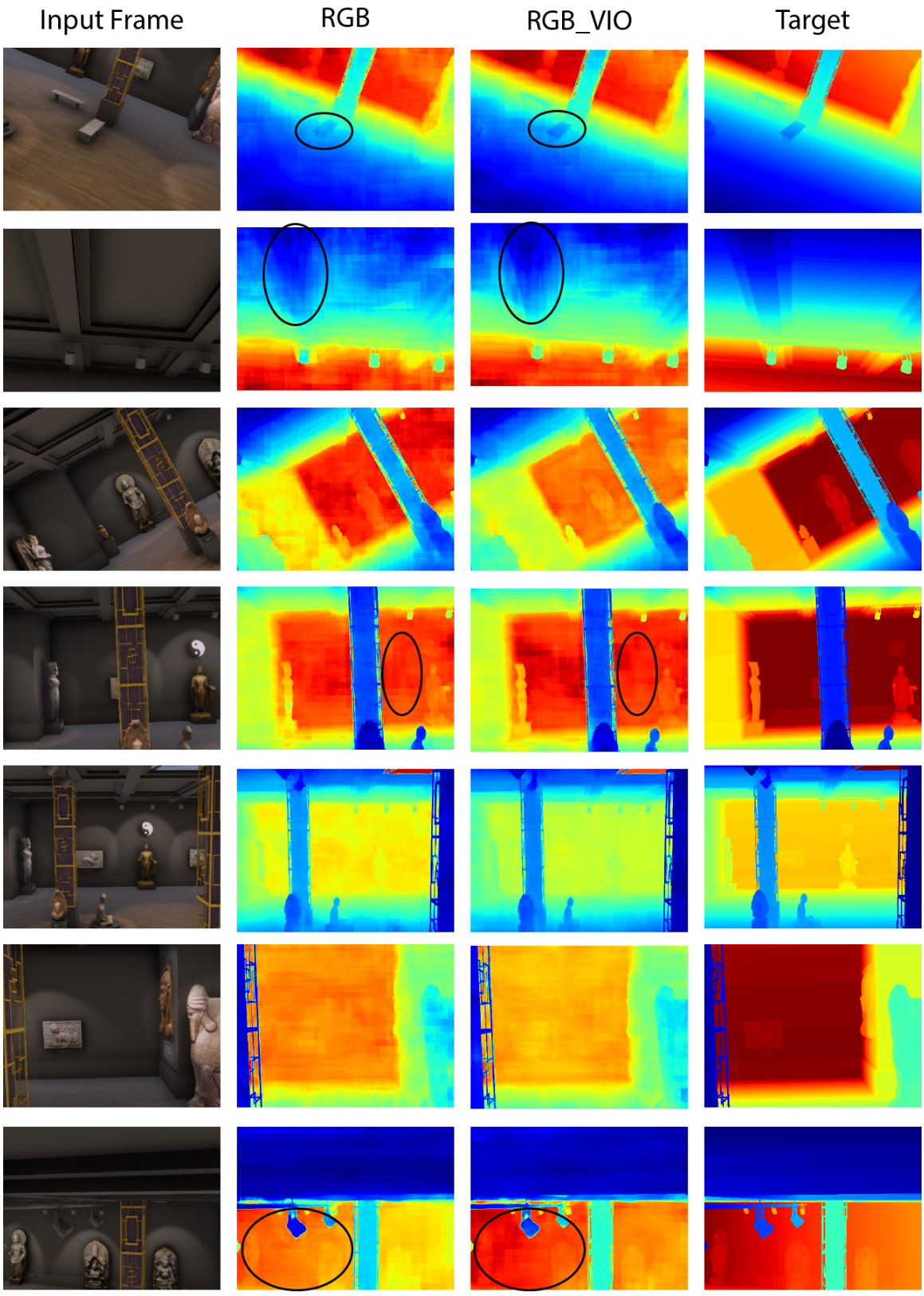

Figure A-5: Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Museum Pillars environment.

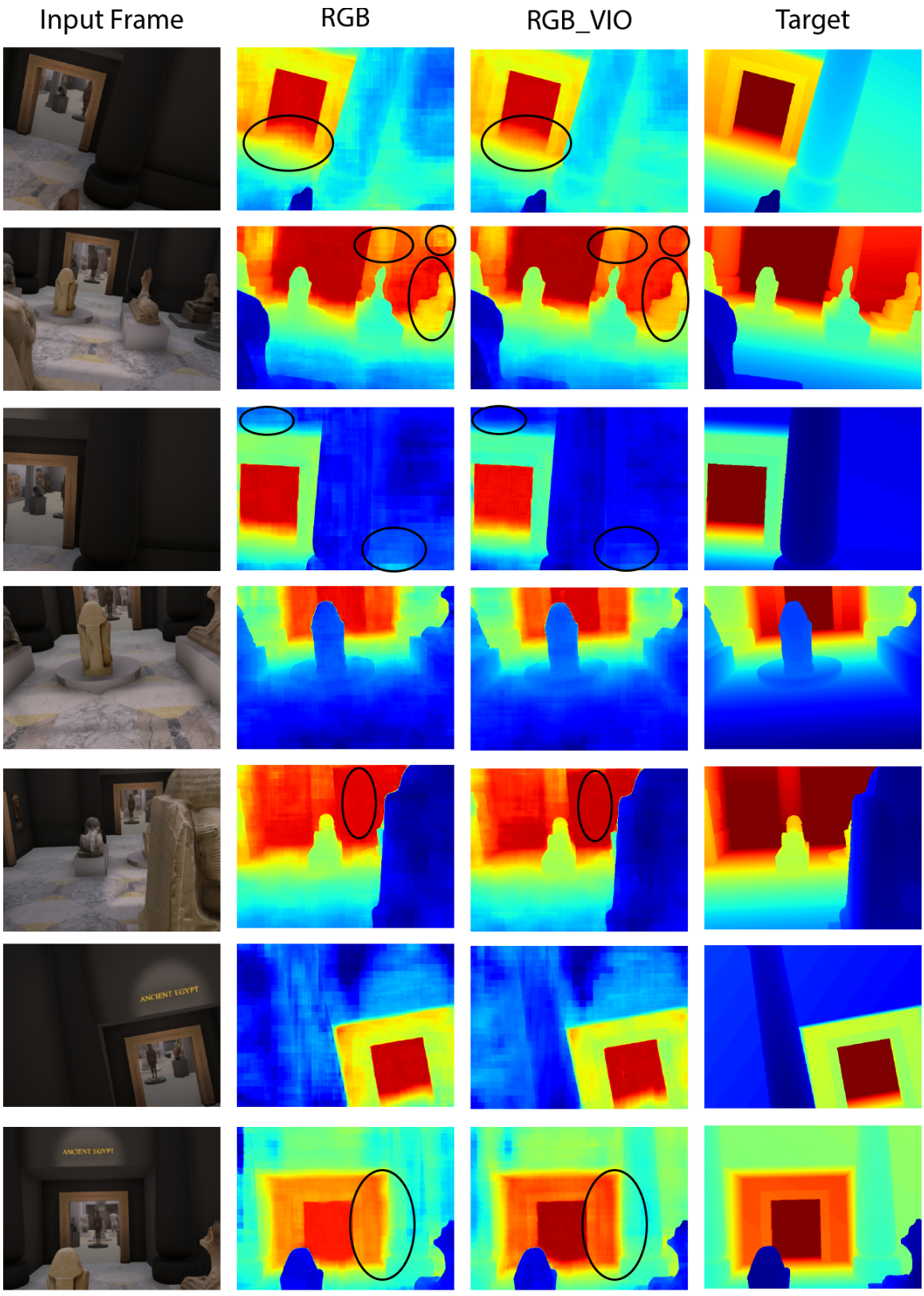

Figure A-6: Sample predictions of RGB and RGB_VIO trained on `blackbird` for the Museum Sphinx Loft environment.

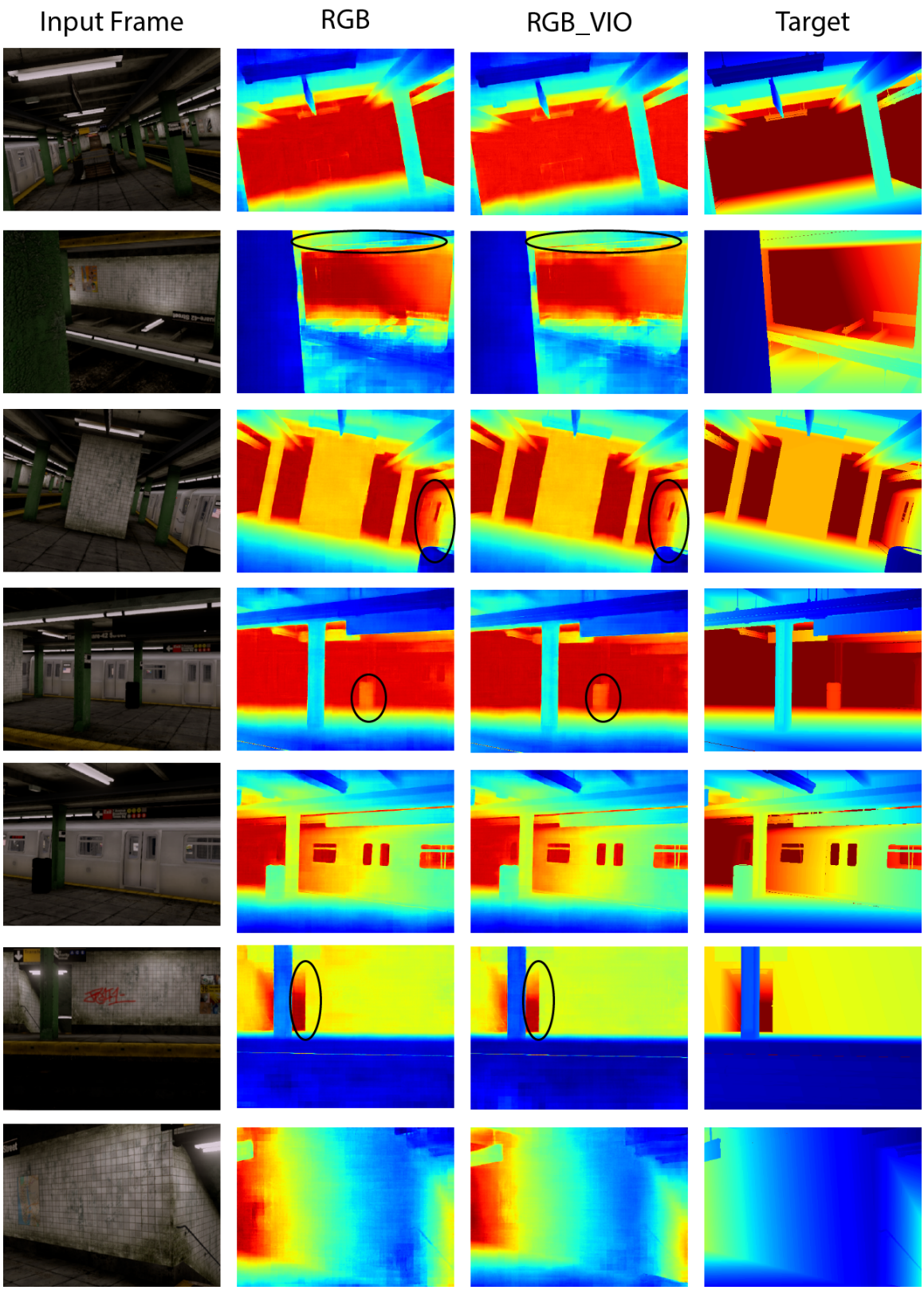

Figure A-7: Sample predictions of RGB and RGB_VIO trained on `blackbird` for the NYC Subway environment

Bibliography

[1] Abdulla Al-Kaff, Fernando Garcia, David Martín Gómez, Arturo de la Escalera, and J.M. Armingol. Obstacle detection and avoidance system based on monocular camera and size expansion algorithm for uavs. *Sensors*, 17:1061, 05 2017.

[2] Kendall Alex, Grimes Matthew, and Cipolla Roberto and. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[3] Wong Alex, Fei Xiaohan, Tsuei Stephanie, and Stefano Soatto. Unsupervised depth completion from visual inertial odometry. In *IEEE ROBOTICS AND AUTOMATION LETTERS*, 2020.

[4] Christoph Stiller Andreas Geiger, Philip Lenz and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research*, 2013.

[5] Amado Antonini, Winter Guerra, Varun Murali, Thomas Sayre-McCord, and Sertac Karaman. The blackbird dataset: A large-scale dataset for uav perception in aggressive flight. In *2018 International Symposium on Experimental Robotics (ISER)*, 2018.

[6] Rui Zhu Simon Lucey Chaoyang Wang, Jose Miguel Buenaposada. Learning depth from monocular videos using direct methods. *Arxiv preprint*, 2017.

[7] Christopher Choy, JunYoung Gwak, and Silvio Savarese. 4d spatio-temporal convnets: Minkowski convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[8] Frank Dellaert and Michael Kaess. *Factor Graphs for Robot Perception*. Now Publishers Inc., Hanover, MA, USA, 2017.

[9] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *2014 Neural Information Processing Systems (NIPS)*, 2014.

[10] David Eigen, Christian Puhrsch, and Fergus Rob. Depth map prediction from a single image using a multi-scale deep network. In *2018 Neural Information Processing Systems (NIPS)*, 2014.

[11] Dellaert F. Factor graphs and gtsam: A hands-on introduction. Technical report, GT RIM, 2012.

[12] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. 07 2015.

[13] Winter Guerra, Ezra Tal, Varun Murali, Gilhyun Ryou, and Sertac Karaman. FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, November 2019.

[14] Akkas Uddin Haque and Ashkan Nejadpak. Obstacle avoidance using stereo camera. *Arxiv preprint*, 2017.

[15] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[16] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[17] C. Häne, T. Sattler, and M. Pollefeys. Obstacle detection for self-driving cars using only monocular cameras and wheel odometry. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5101–5108, 2015.

[18] Yinda Zhang Xingdi Zhang Shuaicheng Liu Bing Zeng Jiaxiong Qiu, Zhaopeng Cui and Marc Pollefeys. Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. *Arxiv preprint*, 2018.

[19] Wei Feng Jian Li Ping Tan Jie Tang, Fei-Peng Tian. Learning guided convolutional network for depth completion. *Arxiv preprint*, 2019.

[20] K. Karsch, Liu C., and S. B. Kang. Depth extraction from video using non-parametric sampling. In *2014 European Conference on Computer Vision (ECCV)*, 2014.

[21] J. Konrad, Wang M., and P. Ishwar. 2d-to-3d image conversion by learning depth from examples. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2012 IEEE Computer Society Conference on.*, 2014.

[22] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[23] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In *2017 3D Vision (3DV)*, 2017.

[24] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[25] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[26] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004.

[27] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, page 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[28] Mancini M., Costante G., Valigi P., and Ciarfuglia T. A. Fast robust monocular depth estimation for obstacle detection with fully convolutional networks. In *RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 4296–4303, 2016.

[29] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[30] Fangchang Ma, Guilherme Venturelli, and Sertac Karaman. Self-supervised sparse-to-dense: Self-supervised depth completion from lidar and monocular camera. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[31] Derek Hoiem Rob Fergus Nathan Silberman, Pushmeet Kohli. Indoor segmentation and support inference from rgbd images. *ECCV*, 2012.

[32] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[33] A. Saxena, S. H. Chung, and A. Y. Ng. Learning depth from single monocular images. In *2006 Neural Information Processing Systems (NIPS)*, 2014.

[34] A. Saxena, Sun M., and A. Y. Ng. Make3d: Learning 3d scene structure from a single still image. In *2009 IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2009.

[35] Hongkai Wen Niki Trigoni Sen Wang, Ronald Clark. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. *Arxiv preprint*, 2017.

[36] Jianbo Shi and Carlo Tomasi. Good features to track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994.

[37] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc., 2014.

[38] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[39] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, CMU CS, 1991.

[40] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant networks. In *2017 3D Vision (3DV)*, 2017.

[41] Casser V., Pirk S., Mahjourian R., and Angelova A. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[42] Diana Wofk*, Fangchang Ma*, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. Fastdepth: Fast monocular depth estimation on embedded systems. In *2019 IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[43] Jianping Shi Zhichao Yin. Geonet: Unsupervised learning of dense depth, optical flow and camera pose. *Arxiv preprint*, 2018.