Features in Extra Dimensions: Spatial and Temporal Scene Representations

Zhaoyuan Fang

CMU-RI-TR-22-39

July 2022

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Katerina Fragkiadaki, *chair*
Shubham Tulsiani
Adam Harley

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Copyright © 2022 Zhaoyuan Fang. All rights reserved.

Abstract

Computer vision models have made great progress in featurizing pixels of images. However, an image is only a projection of the actual 3D scene: occlusions and perspective distortions exist. To arrive at a better representation of the scene itself, extra dimensions are needed to learn spatial or temporal priors.

In this thesis, we propose two methods that introduce extra dimensions for modelling the scene space and time. The first method lifts features from the image plane onto the bird's eye view (BEV) plane for perception in autonomous driving. Features over the scene space enables our models to handle occlusion better, producing accurate BEV semantic representation. The second method introduces extra dimensions for modelling time, for better geometry-free point tracking. We track points through partial or full occlusions, using components that drive the current state-of-the-art in flow and object tracking, such as learned temporal priors, iterative optimization, and appearance updates. Features allocated over timesteps enables our models to track over long horizons and through occlusions, outperforming previous feature-matching and optical flow methods.

Acknowledgments

I would like to thank my advisor, Prof. Katerina Fragkiadaki. Your passion and energy for research inspired me to work hard on important problems in computer vision. Throughout the past two years, your guidance has pushed me to become a better researcher, and more importantly, a better person in many aspects.

Besides my advisor, I really appreciate the rest of my committee members. Thank you to Prof. Shubham Tulsiani. Your ideas were always constructive and insightful. Thank you to Dr. Adam Harley. It has been a great pleasure to work with you. I have learned so much from you, about coming up with ideas, writing good code, adjusting my mindset, and many many more.

Thank you to Prof. Saurabh Gupta, for being both an awesome mentor on both research and life choices. I've always enjoyed our conversations. Thank you to Prof. Ranjay Krishna, for showing extra care and support to Ph.D. admits. You helped me a lot when I was lost about my future. Thank you to Prof. David Held, for hosting me three summers ago as an intern. The first peek at CMU was very nice.

I would like to thank my collaborators in Katerina's Lab. Thank you to Ayush Jain, Fish Tung, Gabe Sarch, Hao Zhu, Jing Wen, Jingyun Yang, Mayank Singh, Mihir Prabhudesai, Nikos Gkanatsios, Paul Shydlo, Shamit Lal, Wen-Hsuan Chu, Xian Zhou, Yiming Zuo, and Yunchu Zhang. I will miss our chats in the lab and I believe the lab will continue to do great research. I would also like to thank the collaborators from outside the lab. Thank you to Dr. Jie Li and Dr. Rares Ambrus for the fruitful project discussions.

Thank you to all my friends in life. Thank you for bearing with my ranting when I feel down, joining me on trips to random places and making precious memories, unwinding together in video games and soccer. You made life worth living.

Last but definitely not least, I would like to thank my parents. The bitter moments are less unbearable with you behind my back. The best things in my life have been your unconditional love, care, and support, and I'm so proud to be your son.

Funding

This material is based upon work supported by Toyota Research Institute (TRI), US Army contract W911NF20D0002, a DARPA Young Investigator Award, an NSF CAREER award, an AFOSR Young Investigator Award, and DARPA Machine Common Sense. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Army or the United States Air Force.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Geometry-based Scene Representation for Bird's Eye View Perception 2
1.3 Geometry-free Scene Representation for Point Tracking . . . 2

2 Geometry-based Scene Representation for Bird's Eye View Perception **3**
2.1 Introduction . . . 3
2.2 Related Works . . . 5
2.3 A Simple Baseline for BEV Perception . . . 7
2.3.1 Setup and overview . . . 7
2.3.2 Architecture . . . 8
2.3.3 Implementation details . . . 11
2.4 Experiments . . . 11
2.4.1 Main results . . . 12
2.4.2 Ablation studies . . . 14
2.5 Conclusion . . . 18

3 Geometry-free Scene Representation for Point Tracking **19**
3.1 Introduction . . . 19
3.2 Related Work . . . 21
3.2.1 Optical flow . . . 21
3.2.2 Feature matching . . . 22
3.3 Persistent Independent Particles (PIPs) . . . 23
3.3.1 Setup and overview . . . 23
3.3.2 Extracting features . . . 24
3.3.3 Initializing each target . . . 24
3.3.4 Measuring local appearance similarity . . . 25
3.3.5 Iterative updates . . . 25
3.3.6 Supervision . . . 26
3.3.7 Test-time trajectory linking . . . 27
3.4 Implementation details . . . 28
3.5 Experiments . . . 29
3.5.1 Training data: FlyingThings++ . . . 30

3.5.2 Baselines 30
3.5.3 Trajectory estimation in FlyingThings++ 32
3.5.4 Trajectory estimation in KITTI 33
3.5.5 Trajectory estimation in CroHD 34
3.5.6 Keypoint propagation in BADJA 35
3.5.7 Ablation on visibility-aware linking 37
3.5.8 Experiment details 38
3.5.9 Limitations 39
3.6 Conclusion 39

4 Conclusion and Future Work **41**

Bibliography **43**

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

2.1 **A baseline for neural BEV mapping**. (A) Our sensor setup consists of multiple cameras and radar units. (B) We begin by featurizing each camera image with a ResNet-101. (C) We define a set of 3D coordinates around the ego-vehicle, project these coordinates into all images, and bilinearly sample features at the projected locations, yielding a 3D volume of features. We then concatenate a rasterized radar image, and reduce the vertical dimension of the volume to yield a BEV feature map. We process this BEV map with a Resnet-18. Finally, task heads produce semantic segmentation, a centerness heatmap that indicates which BEV locations correspond to object centers, and an offset field that indicates the closest object centroid from each BEV location. . 7

2.2 **Comparison of lifting techniques**. (a) [47] uses MLPs to first reduce the vertical dimension of the image and then expand the depth dimension; the polar BEV features are resampled with a Cartesian grid; (b) [48] uses transformers to translate a vertical strip in the image into a BEV ray, and then resamples the BEV features with a Cartesian grid; (c) [43] estimates a distribution over depth and place the features with a soft weight; (d) [32] uses deformable attention to let the 3D queries interact with the image features; (e) we perform a simple bilinear sampling in the projected location of each voxel in the feature volume. 10

2.3 Visualization of our best model's inputs (RGB and radar) and outputs (BEV vehicle segmentation) overlaid on a road visualization. We display LiDAR and ground truth for comparison. Please see the supplementary file for a video visualization. 14

3.1 **Persistent Independent Particles.** Our method takes an RGB video as input, and estimates trajectories for any number of target pixels. Top-left: target pixels are marked with dots; bottom-left: estimated trajectories. Right: estimated trajectories overlaid on the first frame of the input video. 21

3.2 **Architecture of Persistent Independent Particles (PIPs).** Given an RGB video as input, along with a location in the first frame indicating what to track, our model initializes a multi-frame trajectory, then computes features and correlation maps, and iteratively updates the trajectory and its corresponding sequence of features, with a deep MLP-Mixer model. 23
3.3 **Qualitative results in FlyingThings++** for RAFT (left column), and PIPs (our model, right column). Trajectories are drawn in a pink-to-yellow colormap, over ground-truth trajectories which are in blue-to-green, and in the background is the mean RGB of the sequence. Note that RAFT trajectories get "stuck" in the middle of the frame, due to occlusions there. The value displayed in the top left corner is the average error in the video. 32
3.4 **Qualitative results in FlyingThings++ (left), KITTI (middle), and CroHD (right).** We visualize a video with the mean of its RGB. We trace the estimates with pink-to-yellow trajectories, and show ground truth in blue-to-green. FlyingThings++ is chaotic, but training on this data allows our model to generalize. 34
3.5 **Comparison with baselines in BADJA, on videos with occlusions.** For each method, we trace the estimated trajectory with a pink-to-yellow colormap. The sparse ground truth is visualized with cyan x's. In the video on the first row, all methods perform fairly well, though DINO and RAFT drift slightly toward the horse's body. In the second video, the target (on the dog's tail) leaves the image bounds then returns into view. In the third video, the target (on the horse's leg) is momentarily occluded, causing RAFT to lose track entirely. For a more detailed view of these results, please watch the supplementary video. 36

List of Tables

2.1 State-of-the-art comparisons against *single-frame RGB-only* methods for vehicle segmentation IOU in the nuScenes validation set. 12
2.2 State-of-the-art comparisons against all top methods for vehicle segmentation IOU in the nuScenes validation set. Since our focus is on LiDAR-free methods, we put LiDAR-enhanced results in parentheses, including them only for reference. 13
2.3 Effect of input resolution. 15
2.4 Effect of batch size. 15
2.5 Effect of randomizing the reference camera. 16
2.6 Effect of camera dropout. 16
2.7 Using the meta-data associated with each point improves performance. 17
2.8 Turning off nuScenes' outlier-filtering strategy improves performance. 17
2.9 Aggregating radar data across multiple sweeps improves performance. 17

3.1 **Trajectory estimation error in FlyingThings++.** "Vis." evaluates pixels that stay visible and in-bounds; "OOB" evaluates pixels that fly outside the image bounds; "Occ." evaluates pixels that cross behind an occluder. PIP trajectories are more robust to targets moving out-of-bounds or becoming occluded. 33
3.2 **Trajectory estimation error in KITTI.** PIP and RAFT trajectories are similar; DINO lags behind both. 34
3.3 **Trajectory estimation error in CroHD.** PIPs achieves better accuracy, for both visible and occluded targets. 35
3.4 **PCK-T in BADJA.** In this evaluation, keypoints are initialized in the first frame of the video, and are propagated to the end of the video; PCK-T measures the accuracy of this propagation. In each column, we bold the best result, and underline the second-best. Above the middle bar, we give methods a spatial window (marked "Win.") to constrain how they propagate labels, which encodes domain knowledge about the span of plausible motions in the domain (which is a common strategy in existing work). Below the bar, we run each method in the unconstrained setting. Our method wins in most videos, but DINO performs well also. 37

3.5 **Effect of visibility-aware linking on keypoint propagation in BADJA.** Linking trajectories from locations estimated to be "visible" yields a small improvement in average PCK-T. 38

Chapter 1

Introduction

1.1 Motivation

Advances in computer vision brought methods that produce good representations for pixels of images. However, an image is only a projection of the scene. The projection is itself an imperfect representation of the actual scene: occlusions, scaling and perspective distortions exist, and most vision methods that featurize the image fail to handle these artifacts. To arrive at a better representation of the scene itself, features in extra dimensions are needed. Works have been exploring in two directions: learning spatial priors and temporal priors.

In this thesis, we propose two methods, one along each line discussed above. The first explicitly introduces extra dimensions for modelling the scene space, for better bird's eye view (BEV) perception for autonomous vehicles [18]. We use geometry-based methods to lift image features into the 3D space. This representation enables our models to handle occlusion better, producing accurate BEV semantic representation of the space surrounding the vehicle from multiple sensors. The second method explicitly introduces extra dimensions for modelling time, for better geometry-free point tracking. We re-build the classic "particle video" approach [51] to track through partial or full occlusions [17], using components that drive the current state-of-the-art in flow and object tracking, such as learned temporal priors, iterative optimization, and appearance updates. This representation enables our models to track over long time horizons and through occlusions, outperforming previous feature-matching and

optical flow methods.

1.2 Geometry-based Scene Representation for Bird's Eye View Perception

In Chapter 2, we use extra dimensions to model the scene space, for BEV Perception. The task of producing BEV semantic representation of the 3D space surrounding the vehicle from multi-view camera observations has drawn great interest, as the BEV representation captures useful information required for autonomous driving. Most previous works first lift 2D image features onto the BEV plane and then perform segmentation, and the main focus has been on innovating new "lifting" techniques. In contrast, using an architecture similar to [16], we propose a simple baseline model with a parameter-free lifting method [18], exceeding the performance of state-of-the-art models. With extensive experiments and ablations, we hope to understand what really matters for performance. We also show that with a simple fusion strategy for RGB and radar, our model obtains another performance boost of 8 points. We hope this invites people to reconsider the utility of radar in autonomous driving.

1.3 Geometry-free Scene Representation for Point Tracking

In Chapter 3, we use extra dimensions to model time, for point tracking. We revisit the "particle video" [51], where the video is represented with a set of particles that move across frames with permanence, leveraging long-range temporal priors while tracking the particles. Inspired by this approach, we build "deep particle video" [17], using components that drive the current state-of-the-art [62] in flow and object tracking, such as dense cost maps, iterative optimization, and learned appearance updates. We show that with the model trained entirely using point trajectories mined from synthetic optical datasets [38] with augmented occlusions, our method outperforms state-of-the-art feature-matching and optical flow methods on long-range correspondence tasks which consist of real videos [2, 12, 61].

Chapter 2

Geometry-based Scene Representation for Bird's Eye View Perception

2.1 Introduction

There is great interest in building 3D-aware perception systems for autonomous vehicles, under various sensor platforms, which typically constitute multiple RGB cameras, multiple radar units, and optionally a LiDAR unit. Although LiDAR-based methods have made remarkable progress in highly accurate 3D perception [80, 81], the high costs of LiDAR sensors make it less appealing for large-scale deployment [30]. Recently, many works have been focusing on producing an accurate "bird's eye view" (BEV) semantic representation of the 3D space surrounding the vehicle, from only multi-view camera observations [32, 40]. The BEV representation captures the information required for driving-related tasks, such as navigation, obstacle detection, and moving-obstacle forecasting.

We have seen rapid progress in camera-based BEV perception: BEV vehicle semantic segmentation IoU improved from 23.9 [40] to 43.2 [32] in just two years. However, similar to other research areas [1, 71], BEV perception has seen methodological innovations and performance improvement, at the cost of system simplicity

and the risk of obscuring "what really matters" for performance. The main focus in those methods has been on innovating new techniques for "lifting" features from the 2D image plane(s) onto the BEV plane. For example, prior work has explored using homographies to warp features directly onto the ground plane [5], using depth estimates to place features at their approximate 3D locations [43, 52], using MLPs with various geometric biases[14, 20, 49], and most recently, using geometry-aware transformers [48]] and deformable attention across space and time [32]. In this work, we instead propose a simple baseline model where the "lifting" step is parameter-free, and does not rely on depth estimation: we simply define a 3D volume of coordinates over the BEV plane, project these coordinates into all images, and average the features bilinearly sampled from the projected locations. Surprisingly, our simple baseline exceeds the performance of state-of-the-art models, while also being faster and having fewer parameters. Our ablations show that batch size, data augmentation, and input resolution play a large part in performance.

As we establish a new simple baseline, we also take the opportunity to question the currently-popular paradigm of relying on multi-view camera observations alone, instead of fusing available metric information from radar. Radar sensors are not only cheap compared to LiDAR, but have been integrated in real vehicles for several years already [41]. While using RGB cameras alone may give the task a certain purity (requiring metric 3D estimates from 2D input alone), it does not reflect the reality of autonomous driving, where we can freely take advantage of noisy metric data, not only from radar but from GPS and odometry. The few recent works that discuss radar in the context of semantic BEV mapping have concluded that the data sometimes is too sparse to be useful [20, 30]. We identify that these prior works evaluated the use of radar alone, avoiding the multi-modal fusion problem, and perhaps missing the opportunity for RGB and radar to complement one another. We introduce a simple fusion strategy for RGB and radar (rasterizing the radar in BEV and concatenating it to the RGB features), and exceed the performance of all published BEV segmentation models, obtaining a score only 5 points behind a LiDAR-enabled system.

While this work does not a contribute new innovative architecture, it (1) provides a simple baseline for BEV semantic segmentation, with state-of-the-art results and extensive ablative analysis, and (2) invites the community to reconsider the utility of radar in autonomous driving perception systems. We also release code and

reproducible models to facilitate future research in the area.

2.2 Related Works

A major differentiator in prior work on dense BEV parsing is the precise operator for "lifting" 2D perspective-view features to 3D, or directly to the ground plane.

Parameter-free unprojection This strategy, pursued in a variety of object and scene representation models [7, 54, 65], uses the camera geometry to define a mapping between voxels and their projected coordinates, and collects features by bilinearly sampling at the projected coordinates. This places each image feature into multiple 3D coordinates, essentially tiling the feature along the ray's extent in the volume. This method of lifting is not typically used in bird's eye view semantic tasks.

Depth-based unprojection Several works estimate per-pixel depth with a monocular depth estimator, either pre-trained for depth estimation [34, 45, 52] or trained simply for the end-task [21, 43, 68], and used the depth to place features at their estimated 3D locations. This is an effective strategy, but note that if the depth estimation is perfect, it will only place "truck" features at the front visible surface of the truck, rather than fill the entire truck volume with features. We believe this detail is one reason that naive unprojection performs competitively with depth-based unprojection.

Homography-based unprojection Some works estimate the ground plane instead of per-pixel depth, and use the homography that relates the image to the ground to create a warp [5, 33, 35], transferring the features from one plane to another. This operation tends to produce poor results when the scene itself is non-planar (e.g., tall objects inevitably get spread out over a wide area after the homography).

MLP-based unprojection A popular approach is to convert a vertical-axis strip of image features to a forward-axis strip of ground-plane features, with an MLP [20, 31, 40]. An important detail here is that the initial ground-plane features are considered aligned with the camera frustum, and they are therefore warped into a

rectilinear space using the camera intrinsics. Some works in this category use multiple MLPs, dedicated to different scales [47, 49], or to different categories [14]. As this MLP is parameter-heavy, Yang et al. [79] propose a cycle-consistency loss (mapping backward to the image-plane features) to help regularize it.

Geometry-aware transformer-like models An exciting new trend is to transfer features using model components taken from transformer literature. Saha et al. [48] begin by defining a relationship between each vertical scan-line of an image, and the ground-plane line that it projects to, and show that transformer self-attentions can learn an effective "translation" function between the two coordinate systems. Defining this transformer at the line level helps provide inductive bias to the model, since the lifting task should be similar across lines. BEVFormer [32], which is concurrent with our work, proposes to use deformable attention operations to collect image features for a pre-defined grid of 3D coordinates. This is similar to the bilinear sampling operation in the parameter-free unprojection, but with approximately 10× more samples, learnable offsets for the sampling coordinates, and a learnable kernel on their combination.

Radar In the automotive industry, radar has been in use for several years already [41]. Since radar measurements provide position, velocity, and angular orientation, the data is typically used to detect obstacles (e.g., for emergency braking), and to estimate the velocity of moving objects (e.g., for cruise control). Radar is longer-range and less sensitive to weather effects than LiDAR, and substantially cheaper. Unfortunately, the sparsity and noise inherent to radar make it a challenge to use [36, 39, 55, 75]. Some early methods use radar for BEV semantic segmentation tasks much like in our work [36, 53, 55], but only in small datasets. Recent work within the nuScenes benchmark [4] has reported the data too sparse to be useful, recommending instead higher-density radar data from alternate sensor setups [20, 30]. Some recent works explore RGB-radar or RGB-LiDAR fusion strategies [33, 39], similar to our work, but targeting detection and velocity estimation rather than BEV semantic labelling.

2.3 A Simple Baseline for BEV Perception

2.3.1 Setup and overview

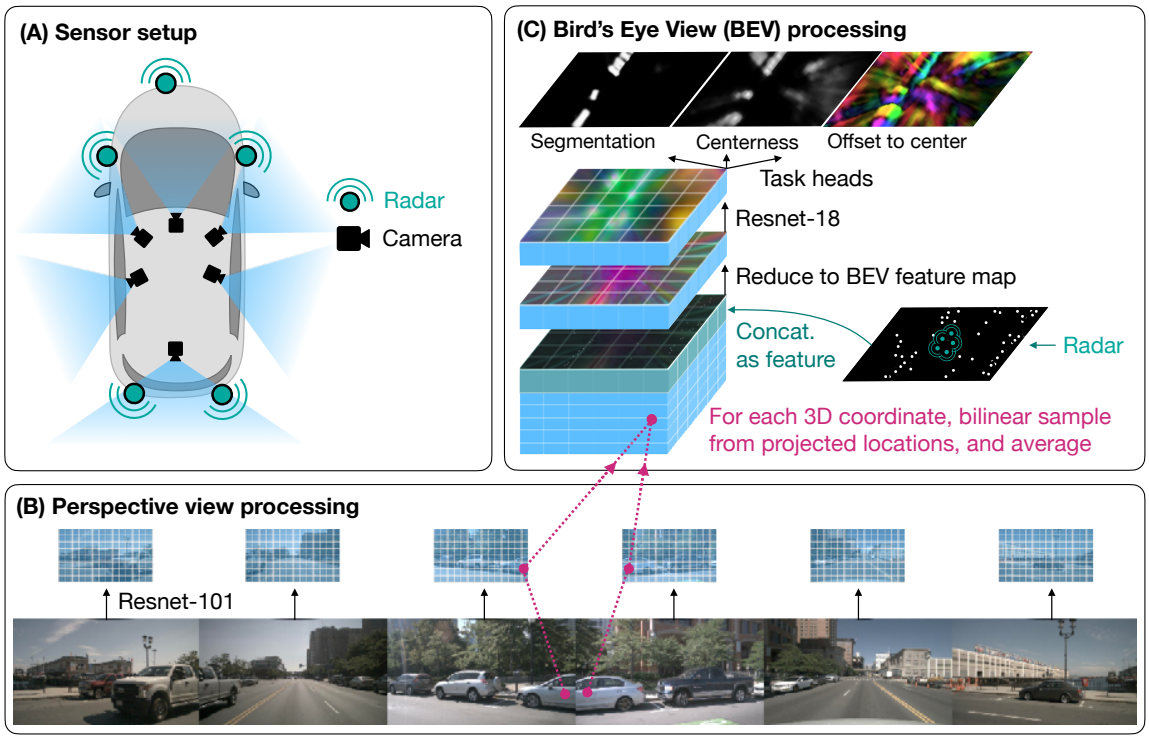

Figure 2.1: **A baseline for neural BEV mapping**. (A) Our sensor setup consists of multiple cameras and radar units. (B) We begin by featurizing each camera image with a ResNet-101. (C) We define a set of 3D coordinates around the ego-vehicle, project these coordinates into all images, and bilinearly sample features at the projected locations, yielding a 3D volume of features. We then concatenate a rasterized radar image, and reduce the vertical dimension of the volume to yield a BEV feature map. We process this BEV map with a Resnet-18. Finally, task heads produce semantic segmentation, a centerness heatmap that indicates which BEV locations correspond to object centers, and an offset field that indicates the closest object centroid from each BEV location.

The setup and architecture of our model are shown in Figure 2.1. Our model uses as input a flexible number of cameras, any number of radar units, and optionally even LiDAR. We assume that the data is synchronized across sensors. We assume that the intrinsics and extrinsics of each sensor are known.

The model has a 3D metric span, and a 3D resolution. Following the baselines in this task, we set the left/right and forward/backward span to $100m \times 100m$, discretized at a resolution of 200×200. We set the up/down span to $10m$, and discretize at a resolution of 8. This volume is centered and oriented according to a reference camera, which is typically the front camera. We denote the left-right axis with X, the up-down axis with Y, and the forward-backward axis with Z.

Our model first computes features from each camera image, as shown in Figure 2.1-B. Then, we populate our 3D volume with features, with bilinear sampling: starting from the 3D coordinates at the center of each discrete cell, we use the extrinsics and intrinsics to compute its 2D coordinates in each 2D feature map, and bilinearly sample the 2D feature map to obtain a feature. If its 3D coordinates land in the camera frustum of multiple views, we simply take an average of the features from all those "valid" views. If radar is provided, we rasterize its returns into a bird's eye view image, and concatenate this with the 3D feature volume. We then reduce the vertical dimension, and process the resulting feature map with a 2D convolutional net. Finally, task heads produce quantities of interest, such as segmentation, centerness scores, and an offset map, all in a 2D bird's eye view. The segmentation map contains a categorical distribution over semantic categories. The centerness map indicates probabilities of a grid cell being the center of an object. The offset map contains a vector field, where each vector points to the nearest object center. The 3D/BEV processing is illustrated in Figure 2.1-C.

2.3.2 Architecture

We featurize each input RGB image, shaped $3 \times H \times W$, with a ResNet-101 [19] backbone. We upsample the output of the last layer and concatenate it with the third layer output, and apply two convolution layers with instance normalization and ReLU activations [9], arriving at feature maps with shape $C \times H/8 \times W/8$ (one eighth of the image resolution).

We project our pre-defined volume of 3D coordinates into all feature maps, and bilinearly sample features there, yielding a 3D feature volume from each camera. We compute a binary "valid" volume per camera at the same time, indicating if the 3D coordinate landed within the camera frustum. We then take a valid-weighted

average across the set of volumes, reducing our representation down to a single 3D volume of features, shaped $C \times Z \times Y \times X$. We emphasize that our lifting step is parameter-free. We rearrange the 3D feature volume dimensions, so that the vertical dimension extends the channel dimension, as in $C \times Z \times Y \times X \rightarrow (C \cdot Y) \times Z \times X$, yielding a high-dimensional BEV feature map.

We next rasterize the radar information, to create another BEV feature map. We may use an arbitrary number of radar channels R (including $R = 0$, meaning not using radar). In nuScenes [4], each radar return consists of a total of 18 fields, with 5 of them being position and velocity, and the remainder being the result of built-in pre-processes (e.g., indicating confidence that the return is valid). We use all of this data, by using the position data to choose the nearest XYZ position in the volume (if in bounds), and using the 15 non-position items as channels, yielding a 3D feature volume shaped $R \times Z \times Y \times X$, with $R = 15$. Similar to the RGB feature volume, this radar feature volume is rearranged so that the vertical dimension extends the channel dimension, $(R \cdot Y) \times Z \times X$. If LiDAR is provided, we voxelize it to a binary occupancy grid shaped $Y \times Z \times X$, and use it in place of radar features (only for comparison).

We then concatenate the RGB features and radar features, and compress the extended channels down to a dimensionality of C, by applying a 3×3 convolution kernel. This achieves the reduction $(C \cdot Y + R \cdot Y) \times Z \times X \rightarrow C \times Z \times X$. At this point, we have a single plane of features, representing a bird's eye view of the scene. We process this with three blocks of a Resnet-18 [19], producing three feature maps, then use additive skip connections with bilinear upsampling to gradually bring the coarser features to the input resolution, and finally apply task-specific heads. Each head is two convolution layers with instance normalization, and ReLU after the norm in the first layer.

The model is trained on three tasks and is equipped with three corresponding task heads: segmentation, centerness, and offset. The segmentation head produces per-pixel vehicle/background segmentation. The centerness head produces a heatmap where high values indicate high probability that the grid cell is an object center. The offset head produces a vector field where, within each object mask, each vector points to the center of that object. We train the segmentation head with a cross entropy loss, and supervise the centerness and offset fields with an L1 loss. We compute

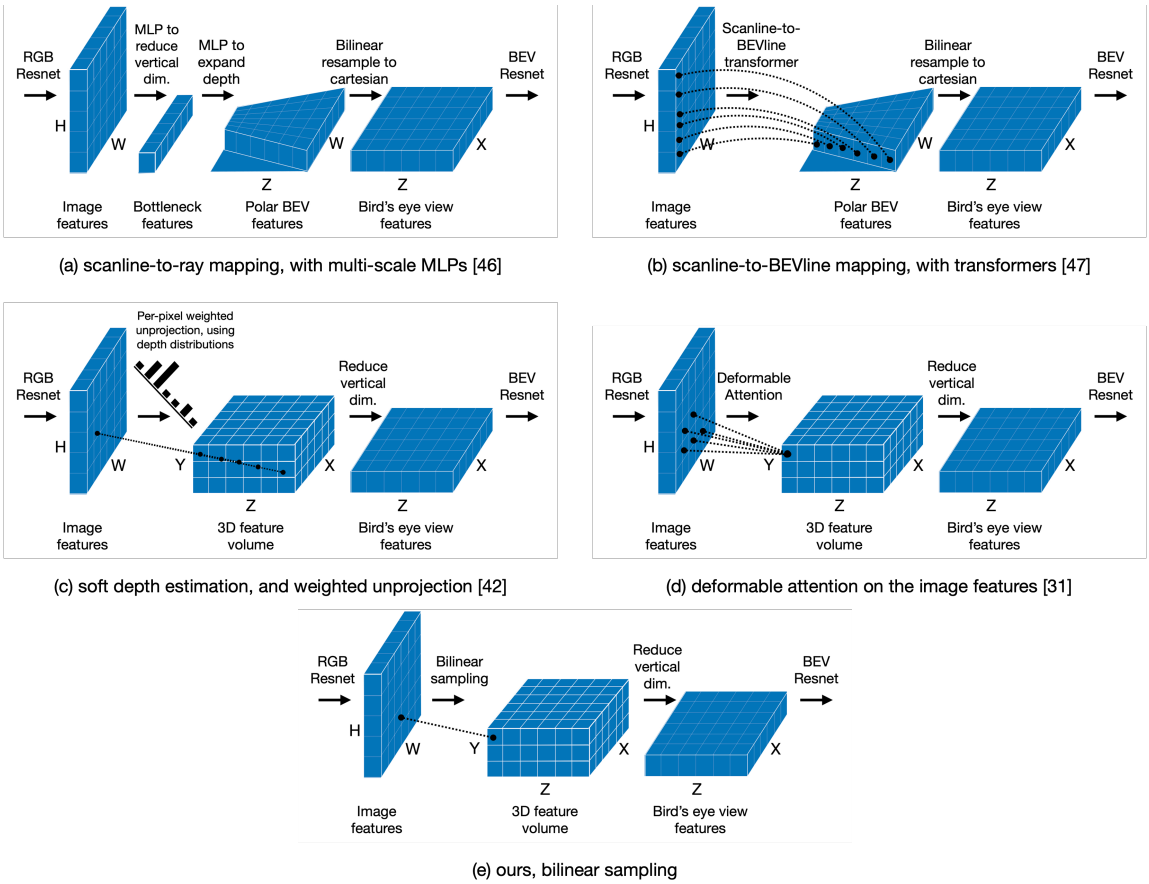

Figure 2.2: **Comparison of lifting techniques**. (a) [47] uses MLPs to first reduce the vertical dimension of the image and then expand the depth dimension; the polar BEV features are resampled with a Cartesian grid; (b) [48] uses transformers to translate a vertical strip in the image into a BEV ray, and then resamples the BEV features with a Cartesian grid; (c) [43] estimates a distribution over depth and place the features with a soft weight; (d) [32] uses deformable attention to let the 3D queries interact with the image features; (e) we perform a simple bilinear sampling in the projected location of each voxel in the feature volume.

the ground truth from 3D box annotations. We use an uncertainty-based learnable weighting [27] to balance the three losses.

Compared to related models, the architecture choices for the perspective-view and bird's-eye-view encoders are most similar to those in Lift-Splat [43] and FIERY [21]. Our multi-task setup is the same as FIERY [21].

Our model is "simpler" than related work particularly in the 2D-to-3D lifting

step, which is handled by (parameter-free) bilinear sampling. This replaces, for example, depth estimation [43], MLPs [20, 31, 40], or attention mechanisms [32, 48]. A qualitative comparison of the lifting step is shown in Figure 2.2. Besides the implementation simplicity, our model is also faster and has fewer parameters than the next-best existing model, as we will detail in the experiments.

2.3.3 Implementation details

Our model uses an RGB input resolution of 448×960. The ResNet-101 has a total stride of 8. We use a feature dimension (i.e., channel dimension) of 128. Our 3D resolution is $200 \times 8 \times 200$, and our final output resolution is 200×200. Our 3D metric span is $100m \times 10m \times 100m$. This corresponds to voxel lengths of $0.5m \times 1.25m \times 0.5m$ (in Z, Y, X order). The Resnet-101 is pre-trained for image classification on ImageNet [8]. The BEV Resnet-18 is trained from scratch.

At training time, we randomly select a camera to be the "reference" camera, which randomizes the orientation of the 3D volume (as well as the orientation of the rasterized annotations). We apply random cropping on the RGB input, by resizing the original images to 558×992, and taking a random 448×960 crop inside (and updating the intrinsics accordingly). At test time, we use the "front" camera as the reference camera, and take a center crop.

We train end-to-end for 25,000 iterations, with a batch size of 40, with the AdamW optimizer [37] at a constant learning rate of 3e-4. We accumulate gradients across eight V100 GPUs and five iterations, to obtain an effective batch size of 40 for each gradient step. The model takes 2-3 days to train.

2.4 Experiments

We train and test our model in the nuScenes [4] urban scenes dataset, which is publicly available for non-commercial use. The dataset has 6 cameras, pointing front, front-left, front-right, back-left, back, and back-right, and 5 radar units, pointing front, left, right, back-left, and back-right, as well as a LiDAR unit. We use LiDAR inputs only for comparison, and focus on using RGB and RGB+radar. The sensor setup is illustrated in Figure 2.1-A. We use the official nuScenes training/validation split,

which contains 28,130 samples in the training set, and 6,019 samples in the validation set. We use annotations from the "vehicle" superclass, which includes bicycle, bus, car, construction vehicle, emergency vehicle, motorcycle, trailer, and truck. We evaluate on vehicle/background segmentation, using the intersection-over-union (IOU) metric in a bird's eye view. (We follow related work [21] in treating centerness and offset purely as auxiliary tasks.)

2.4.1 Main results

In this section we present our BEV vehicle segmentation results on the nuScenes validation set, and compare with the state-of-the-art.

Method	IOU
FISHING [20]	30.0
PON [47]	31.4
Lift, Splat [43]	32.1
FIERY [21]	35.8
TIIM [48]	38.9
BEVFormer [32]	44.4
Ours	**47.2**

Table 2.1: State-of-the-art comparisons against *single-frame RGB-only* methods for vehicle segmentation IOU in the nuScenes validation set.

We first compare against single-frame RGB-only models, in Table 2.1. Our RGB-only method obtains 47.2 IOU, outperforming all other single-frame RGB-only models. Second-best is BEVFormer [32] (concurrent work) at 44.4. The main difference between these two methods is that BEVFormer uses a deformable attention-based strategy to lift features from 2D to BEV, in place of our bilinear sampling.

We next open the comparison to all methods of all modalities, in Table 2.2. The best model we are aware of is BEVFormer's temporal variant, which obtains 46.7 IOU—just under the accuracy of our single-frame RGB-only model.

Our RGB+radar model improves over this by 9 points, reaching 55.7 IOU. For reference, we also compute our model's performance using RGB+LiDAR, obtaining a new high of 60.8. High performance from LiDAR is consistent with related work in 3D object detection [80], but the gap between RGB+LiDAR and RGB+radar is

Method	Inputs	IOU
FISHING [20]	RGB	30.0
	LiDAR	(44.3)
FIERY [21]	RGB	35.8
	RGB+time	38.2
TIIM [48]	RGB	38.9
	RGB+time	41.3
BEVFormer [32]	RGB	44.4
	RGB+time	46.7
Ours	RGB	47.2
	RGB+radar	**55.7**
	RGB+LiDAR	(**60.8**)

Table 2.2: State-of-the-art comparisons against all top methods for vehicle segmentation IOU in the nuScenes validation set. Since our focus is on LiDAR-free methods, we put LiDAR-enhanced results in parentheses, including them only for reference.

smaller than might have been expected, since prior work conveyed negative results from RGB+radar fusion [20]. Note that integrating over time is orthogonal to the strengths of our model, so it should be possible to push results even higher using time.

Speed and complexity: Our model runs at 7.3 FPS on a V100 GPU. This is more than 3× faster than BEVFormer [32], which runs at 2.3 FPS. Our model also has fewer parameters: 42.0M, compared to BEVFormer's 68.7M. Most of our parameters (37.0M) come from the Resnet-101, and this is also the main speed bottleneck, due to the high RGB resolution.

Qualitative results: We show qualitative results in Figure 2.3. We also visualize corresponding LiDAR and radar data, to show the scene structure as captured by those sensors. Qualitatively, the radar is indeed sparse and noisy as noted in related work [20, 30], but we believe it gives valuable hints about the metric scene structure, which, when fused with information acquired from RGB, enables higher-accuracy semantic segmentation in the bird's eye view.

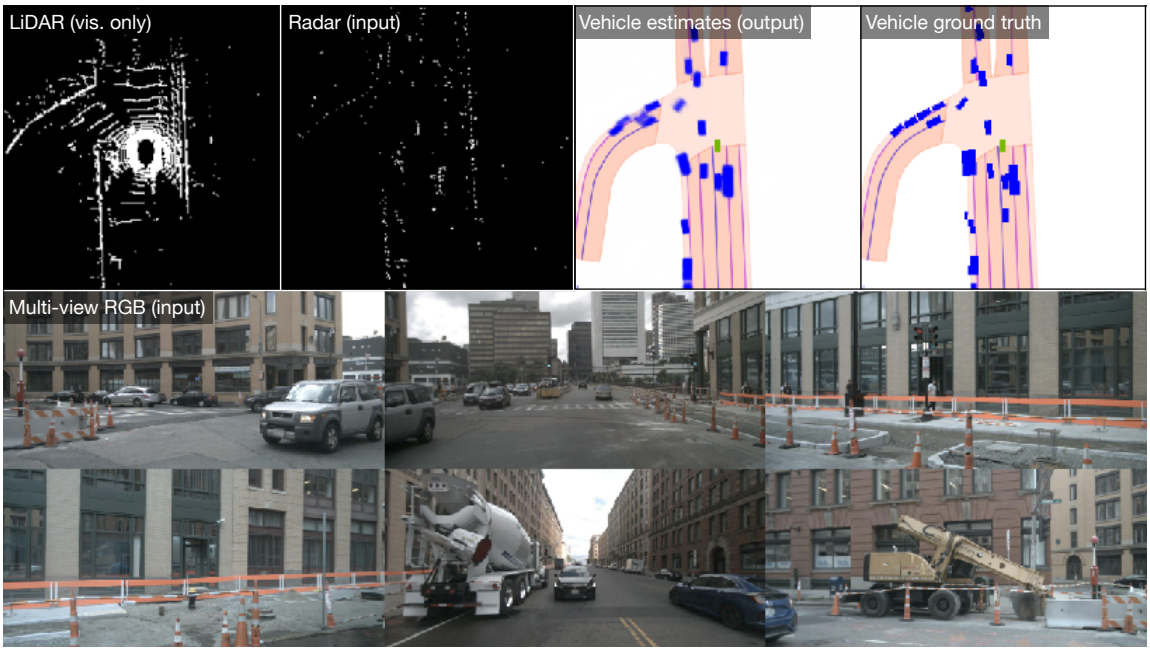

Figure 2.3: Visualization of our best model's inputs (RGB and radar) and outputs (BEV vehicle segmentation) overlaid on a road visualization. We display LiDAR and ground truth for comparison. Please see the supplementary file for a video visualization.

2.4.2 Ablation studies

Considering the simplicity of our method compared to prior work, we next aim to answer the question: what really matters for performance?

While prior work has focused on the details of the 2D-to-BEV lifting strategy, our 2D-to-BEV step is parameter-free, so we study other factors: input resolution, batch size, and augmentations. We first perform ablations using our RGB-only model, and then turn to evaluating the details of our radar processing. Each ablation involves re-training the model with a single specific difference with respect to the proposed model. In all ablations we report vehicle segmentation IOU in the nuScenes validation set (where the proposed model achieves 47.2).

Input resolution The nuScenes dataset provides high-resolution RGB images, which are 900×1600. While earlier work downsampled the RGB substantially before feeding it through the model (e.g., downsampling to 128×352 [43]), we note that

recent works have been downsampling less and less (e.g., most recently using the full resolution [32]). We believe this is an important factor for performance, and so we train and test our RGB-only model across different input resolutions. Table 2.3 summarizes the results. We find that our model obtains its best result at 448×960, which is approximately half of the total available resolution. It may be that when the images are too large or too small, the typical object scale is no longer consistent with the backbone's pre-training (on ImageNet [8]), leading to less-effective transfer.

RGB resolution	IOU
112×240	36.3
224×480	42.7
448×960	**47.2**
672×1440	44.0

Table 2.3: Effect of input resolution.

Batch size It has been reported in the image classification literature that higher batch sizes deliver superior results [57], but we have not seen batch size discussed in BEV literature. In Table 2.4 we explore the impact of batch size on our model's performance: each increase in batch size gives an improvement in accuracy, with diminishing (but sizeable) returns. Increasing the batch size from 2 to 40 gives a nearly 10-point improvement in IOU. It appears that increasing the batch size further might improve performance still, but this is beyond our current compute capacity.

Batch size	IOU
2	36.8
4	41.8
20	43.7
40	**47.2**

Table 2.4: Effect of batch size.

Augmentations When training our model, we randomize the camera selected to be the "reference" camera, which dictates the orientation of the 3D coordinate system.

To the best of our knowledge ours is the first work to do this.

We show the results of this augmentation in Table 2.5. Randomizing the reference camera provides approximately a 1 point boost in IOU. We believe that randomizing the reference camera helps reduce overfitting in the bird's eye view module. We have observed qualitatively that without this augmentation, the segmented cars have a slight bias for certain orientations in certain positions; with the augmentation added, this bias disappears.

Reference camera	IOU
"Front"	46.0
Random	**47.2**

Table 2.5: Effect of randomizing the reference camera.

Prior work has reported a benefit from randomly dropping 1 of the 6 available cameras in each training sample [43]. Interestingly, we find the opposite from the results shown in Tbale 2.6: using all cameras performs better. It may be that our reference-camera randomization provides enough regularization to make camera-dropout unnecessary.

Number of cameras	IOU
5/6	45.9
6/6	**47.2**

Table 2.6: Effect of camera dropout.

We have also experimented with color, contrast, and blur augmentations, and found that these worsened results. Such augmentations are known to be beneficial in image classification, but it may be that the model benefits from sensitivity to these factors in the current data.

Radar usage details Since ours is the first model to report strong results from RGB+radar fusion in this domain, we aim to reveal the important hyperparameter choices in the radar setup.

As shown in Table 2.7, our model benefits from accessing the meta-data associated with each radar point. This includes information such as velocity, which may help distinguish moving objects from the background. Removing this aspect lowers IOU by 2.5 points.

Input	IOU
Full return	**55.7**
Occupancy only	53.2

Table 2.7: Using the meta-data associated with each point improves performance.

As shown in Table 2.8, our model benefits from having *all* radar returns as input, achieved by disabling nuScenes' built-in outlier filtering strategy. The filtering strategy attempts to discard outlier points (produced by multipath interference and other issues), but potentially discards some true returns as well. Using the filtered data instead of the raw data results in a 2.3 point drop in performance.

Filtering	IOU
Off	**55.7**
On	53.4

Table 2.8: Turning off nuScenes' outlier-filtering strategy improves performance.

As shown in Table 2.9, our model benefits from aggregating multiple sweeps of radar as input. This means using radar from timesteps $(t, t-1, t-2)$ aligned to the coordinate frame of timestep t, rather than exclusively using the data from timestep t. Using a single sweep lowers performance by 2.4 points, likely because the model struggles with the extreme sparsity of the signal.

# Sweeps	IOU
3	**55.7**
1	53.3

Table 2.9: Aggregating radar data across multiple sweeps improves performance.

Discussion and limitations In this work, we propose a simple baseline architecture for BEV semantic parsing, and show its surprising effectiveness. Our work highlights that radar provides useful sparse metric information for BEV parsing, and this insight can be applied to other approaches. Similarly, our training techniques may lead to improvements for other models. Our work does not argue against the use of LiDAR in particular, but rather for the use of metric information whenever available, even if sparse and noisy.

We did not discuss any temporal integration strategies. Temporal integration is very natural to include in this setting, and previous works have shown it gives a performance boost of 3-4 points. We leave this for future work. Finally we note that training these models is costly in terms of both GPU time and carbon footprint [58]. We aim to release our models and encourage their re-use.

2.5 Conclusion

LiDAR-free BEV perception is a critical step toward low-cost autonomous vehicles. This paper proposes a simple baseline model with a parameter-free 2D-to-BEV lifting step that outperforms the state-of-the-art. We then reconsider the assumption that radar data is too sparse to be useful, and propose a simple strategy to fuse its noisy returns with the 3D representation acquired from RGB. The resulting radar-enhanced model exceeds the performance of all published models, including ones that integrate information across time, and is only 5 points behind the LiDAR-enhanced model in BEV vehicle segmentation. We will make our code publicly available. We hope that this simple model will serve as a useful baseline in the future.

Chapter 3

Geometry-free Scene Representation for Point Tracking

3.1 Introduction

In 2006, Sand and Teller [51] wrote that there are two dominant approaches to motion estimation in video: feature matching and optical flow. This is still true today. In their paper, they proposed a new motion representation called a "particle video", which they presented as a middle-ground between feature tracking and optical flow. The main idea is to represent a video with a set of particles that move across multiple frames, and leverage long-range temporal priors while tracking the particles.

Methods for feature tracking and optical flow estimation have greatly advanced since that time, but there has been relatively little work on estimating long-range trajectories at the pixel level. Feature correspondence methods [6, 70] currently work by matching the features of each new frame to the features of one or more source frames [29], without taking into account temporal context. Optical flow methods today produce such exceedingly-accurate estimates within pairs of frames [62] that the motion vectors can often be chained across time without much accumulation of error, but as soon as the target is occluded, it is no longer represented in the flow field, and tracking fails.

Particle videos have the potential to capture two key elements missing from

feature-matching and optical flow: (1) persistence through occlusions, and (2) multi-frame temporal context. If we attend to a pixel that corresponds to a point on the world surface, we should expect that point to exist across time, even if appearance and position and visibility all vary somewhat unpredictably. Temporal context is of course widely known to be relevant for flow-based methods, but prior efforts to take multi-frame context into account have yielded only small gains. Flow-based methods mainly use consecutive pairs of frames, and occasionally leverage time with a simple constant-velocity prior, which weakly conditions the current flow estimate on previous frames' flow [46, 62].

We propose Deep Particle Video (PIPs), a new particle video method, which takes a T-frame RGB video as input, along with the (x, y) coordinate of a target to track, and produces a $T \times 2$ matrix as output, representing the positions of the target across the given frames. The model can be queried for any number of particles, at any positions within the first frame's pixel grid. A defining feature of our approach, which differentiates it from both the original particle video method and modern flow methods, is that it makes an extreme trade-off between spatial awareness and temporal awareness. *Our model estimates the trajectory of every target independently.* Computation is shared between particles within a video, which makes inference fast, but each particle produces its own trajectory, without inspecting the trajectories of its neighbors. This extreme choice allows us to devote the majority of parameters into a module that simultaneously learns (1) temporal priors, and (2) an inference mechanism that *searches* for the target pixel's location in all input frames. The value of the temporal prior is that it allows the model to *fail* its correspondence task at multiple intermediate frames. As long as the pixel is "found" at some sparse timesteps within the considered temporal span, the model can use its prior to estimate plausible positions for the remaining timesteps. This is helpful because appearance-based correspondence is impossible in some frames, due to occlusions, moving out-of-bounds, or difficult lighting.

We train our model entirely in synthetic data, which we call FlyingThings++, based on the FlyingThings [38] optical flow dataset. Our dataset includes multi-frame amodal trajectories of various lengths, with challenging synthetic occlusions caused by moving and static objects. In our experiments on both synthetic and real video data, we demonstrate that our particle trajectories are more robust to occlusions

Figure 3.1: **Persistent Independent Particles.** Our method takes an RGB video as input, and estimates trajectories for any number of target pixels. Top-left: target pixels are marked with dots; bottom-left: estimated trajectories. Right: estimated trajectories overlaid on the first frame of the input video.

than flow trajectories—they can pick up an entity upon re-appearance—and also provide smoother and finer-grained correspondences than current feature-matching methods, thanks to its temporal prior. We also propose a method to link the model's moderate-length trajectories into arbitrarily-long trajectories, relying on a simultaneously-estimated visibility cue. Figure 3.1 displays sample outputs of our model on RGB videos from the DAVIS benchmark [44]. We plan to publicly release our code, model weights, and data.

3.2 Related Work

3.2.1 Optical flow

While earlier optical flow methods use optimization techniques to estimate motion fields between two consecutive frames [3, 60], recent methods learn such displacement fields supervised from synthetic datasets [10, 22]. Many recent works use iterative refinements for flow estimation by leveraging coarse-to-fine pyramids [59]. Instead of coarse-to-fine refinements, RAFT [62] mimics an iterative optimization algorithm, and estimates flow through iterative updates of a high resolution flow field based on 4D correlation volumes constructed for all pairs of pixels from per-pixel features. Inspired by RAFT, we also perform iterative updates of the position estimations

using correlations as an input, but unlike RAFT we additionally update features.

Ren *et al.* [46] propose a fusion approach for multi-frame optical flow estimation. The optical flow estimates of previous frames are used to obtain multiple candidate flow estimations for the current timestep, which are then fused into a final prediction by a learnable module. In contrast, PIPs explicitly reasons about multiframe context, and iteratively updates its estimates across all frames considered. Note that without using multiple frames, it is impossible to recover an entity after occlusion. Janai et al. [24] is closer to our method, since it uses 3 frames as multiframe context, and explicitly reasons about occlusions. That work uses a constant velocity prior [50] to estimate motion during occlusion. In contrast, PIPs devotes a large part of the model capacity to learning an accurate temporal prior, and iteratively updates its estimates across all frames considered, in search of the object's re-emergence from occlusion. Note that without using multiple frames, it is impossible to recover an entity after occlusion. Additionally, our model is the only work that aims to recover *amodal* trajectories that do not terminate at occlusions but rather can recover and re-connect with a visual entity upon its re-appearance.

3.2.2 Feature matching

Wang and Jabri et al. [23, 70] leverage cycle consistency of time for feature matching. This allows unsupervised learning of features by optimizing a cycle consistency loss on the feature space across multiple time steps in unlabelled videos. Lai et al. [28, 29] and Yang et al. [76] learn feature correspondence through optimizing a proxy reconstruction objective, where the goal is to reconstruct a target frame (color or flow) by linearly combining pixels from one or more reference frames. The combination weights are obtained by computing affinities between the features from the target frame and features from the reference frame(s).

Instead of using proxy tasks, supervised approaches [13, 25, 69, 72] directly train models using ground truth correspondences across images. Features are usually extracted per-image and a transformer-based processor locates correspondences between images. In this work, we reason about point correspondences over a long temporal horizon, incorporating motion context, instead of using pairs of frames like these works.

3.3 Persistent Independent Particles (PIPs)

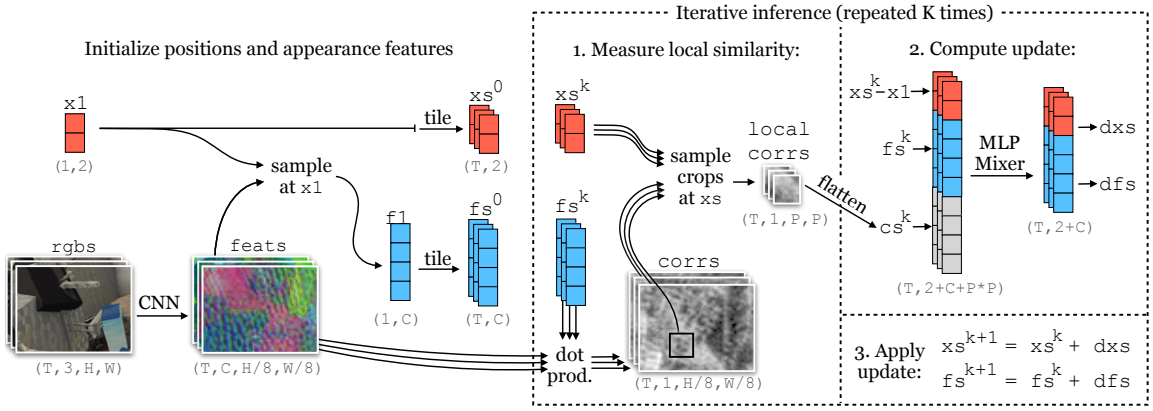

Figure 3.2: **Architecture of Persistent Independent Particles (PIPs).** Given an RGB video as input, along with a location in the first frame indicating what to track, our model initializes a multi-frame trajectory, then computes features and correlation maps, and iteratively updates the trajectory and its corresponding sequence of features, with a deep MLP-Mixer model.

3.3.1 Setup and overview

We name our model Persistent Independent Particles (PIPs), taking conceptual inspiration from Particle Video [51]. Our model takes as input an RGB video with T frames, and the (x_1, y_1) coordinate of a single point on the first frame, indicating the target to track. As output, the model produces a $T \times 3$ matrix, containing T tracked coordinates (x_t, y_t) across time, along with visibility/occlusion estimates $v_t \in [0, 1]$. The model can be queried for N target points in parallel, and some computation will be shared between them, but the model does not share information between the targets' trajectories.

At training time, we query the model with points for which we have ground-truth trajectories and visibility labels. We supervise the model's (x_t, y_t) outputs with a regression objective, and supervise v_t with a classification objective. At test time, the model can be queried for the trajectories of any number of points.

We use the words "point" and "particle" interchangeably to mean the things we are tracking, and use the word "pixel" more broadly to indicate any discrete cell

on the image grid. Note that although the tracking targets are specified with single pixel coordinates, tracking successfully requires (at least) taking into account the local spatial context around the specified pixel, and therefore the model is somewhat sensitive to scale and resolution.

Our overall approach has four stages, somewhat similar to the RAFT optical flow method [62]: extracting visual features (Section 3.3.2), initializing a list of positions and features for each target (Section 3.3.3), locally measuring appearance similarity (Section 3.3.4), and repeatedly updating the positions and features for each target (Section 3.3.5). Figure 3.2 shows an overview of the method.

3.3.2 Extracting features

We begin by extracting features from every frame of the input video. In this step, each frame is processed independently with a 2D convolutional network (i.e., no temporal convolutions). The network produces features at 1/8 resolution.

3.3.3 Initializing each target

After computing feature maps for the video frames, we compute a feature vector for the target, by bilinearly sampling inside the first feature map at the first (given) coordinate, obtaining a feature vector f_1.

We use this sampled feature to initialize a trajectory of features, by simply tiling the feature across time, yielding a matrix $\mathcal{F}^0$ sized $T \times C$, where C is the channel dimension. This initialization implies an appearance constancy prior.

We initialize the target's trajectory of positions in a similar way. We simply copy the initial position across time, yielding a matrix $\mathcal{X}^0$, shaped $T \times 2$. This initialization implies a zero-velocity prior, which essentially assumes nothing about the target's motion.

During inference, we will update the trajectory of features, tracking appearance changes, and update the trajectory of positions, tracking motion.

3.3.4 Measuring local appearance similarity

We would like to measure how well our trajectory of positions, and associated trajectory of features, matches with the pre-computed feature maps. We compute visual similarity maps by correlating each feature f^t with the feature map of the corresponding timestep, and then obtain "local" scores by bilinearly sampling a crop centered at the corresponding position (x_t, y_t). This step returns patches of un-normalized similarity scores, where large positive values indicate high similarity between the target's feature and the convolutional features at this location. We denote the initial set of scores as $\mathcal{C}^0$, shaped $T \times P \cdot P$, where P is the size of the patch extracted from each correlation map.

Similar to RAFT [62], we find it is beneficial to create a spatial pyramid of these score patches, to obtain similarity measurements at multiple scales. This makes our score matrix $T \times P \cdot P \cdot L$, where L is the number of levels in the pyramid.

3.3.5 Iterative updates

The main inference step for our model involves updating the sequence of positions, and updating the sequence of features. To perform this update, we take into account all of the information we have computed thus far: the position matrix $\mathcal{X}^k$, the feature matrix $\mathcal{F}^k$, and the correlation matrix $\mathcal{C}^k$, indicated here with superscript k to emphasize that these inputs will change across iterative inference steps.

Instead of using absolute positions $\mathcal{X}^k$, we subtract the given position (x_1, y_1) from each element of this matrix, making it into a matrix of displacements. (On the first iteration, it will in fact be all zeros, since $\mathcal{X}^0$ is initialized from (x_1, y_1).) Using displacements instead of absolute positions makes all input trajectories appear to start at $(0, 0)$, which makes our model translation-invariant. We additionally concatenate sinusoidal position encodings of the displacements [67], motivated by the success of these encodings in vision transformers [11].

To process this broad set of inputs, we concatenate them all on the channel dimension, yielding a new matrix shaped $T \times D$, and process this with a 12-block MLP-Mixer [64], which is a parameter-efficient all-MLP architecture with design similarities to a transformer. As output, this module produces updates for the

sequence of positions and sequence of features, $d\mathcal{X}$ and $d\mathcal{F}$, which we apply with:

$$\begin{aligned} \mathcal{F}^{k+1} &= \mathcal{F}^k + d\mathcal{F}, \\ \mathcal{X}^{k+1} &= \mathcal{X}^k + d\mathcal{X}. \end{aligned} \tag{3.1}$$

After each update, we compute new correlation pyramids at the updated coordinates, using the updated features.

The update module is iterated K times. After the last update, the positions $\mathcal{X}^K$ are treated as the final trajectory, and the features $\mathcal{F}^K$ are sent to a linear layer to estimate per-timestep visibility logits $\mathcal{V}$.

3.3.6 Supervision

We supervise the model using the L_1 distance between the ground-truth trajectory and the estimated trajectory (across iterative updates), with exponentially increasing weights, similar to RAFT [62]:

$$\mathcal{L}_{\text{main}} = \sum_{k}^{K} \gamma^{K-k} ||\mathcal{X}^k - \mathcal{X}^*||_1, \tag{3.2}$$

where K is the number of iterative updates, and we set $\gamma = 0.8$. Note that this loss is applied even when the target is occluded, or out-of-bounds, which is possible since we are using synthetically-generated ground truth. This is the main loss of the model, and the model can technically train using only this, although it will not learn visibility estimation and convergence will be slow.

On the model's visibility estimates, we apply a cross entropy loss:

$$\mathcal{L}_{\text{ce}} = \mathcal{V}^* \log \mathcal{V} + (1 - \mathcal{V}^*) \log(1 - \mathcal{V}). \tag{3.3}$$

We find it accelerates convergence to directly supervise the score maps to peak in the correct location (i.e., the location of the true correspondence) when the target is visible:

$$\mathcal{L}_{\text{score}} = -\log(\exp(c_i) / \sum_{j} \exp(c_j)) 1\{\mathcal{V}^* \neq 0\}, \tag{3.4}$$

where c_j represents the match score at pixel j, and i is pixel index with the true

correspondence.

We average all losses across all targets within a batch. We set the coefficient of every loss to 1.

3.3.7 Test-time trajectory linking

At test time, it is often desirable to generate correspondences over longer timespans than the training sequence length T. To generate these longer trajectories, we may repeat inference starting from any timestep along the estimated trajectory, treating (x_t, y_t) as the new (x_1, y_1), and thereby "continuing" the trajectory up to (x_{t+T}, y_{t+T}). However, doing this naively (e.g., always continuing from the last timestep), can quickly cause tracking to drift.

It is first of all crucial to avoid continuing the trajectory from a timestep where the target is occluded. Otherwise, the model will switch to tracking the occluder. To avoid these identity switches, we make use of our visibility estimates, and seek the farthest timestep whose visibility score is above a threshold. Note that seeking farthest visible timestep allows the model to skip past frames where the target was momentarily occluded, as long as the temporal span of the occlusion is less than the temporal span of the model (T). Visibility estimates always begin near 1, since the model is trained assuming the provided (x_1, y_1) indicates the true target, so we exclude the first several timesteps from this selection process, forcing the model to choose a timestep from the later part of the trajectory. We initialize the threshold conservatively at 0.99, and decrease it in increments of 0.01 until a valid selection is found.

Even with visibility-aware trajectory linking, the model can "forget" what it was originally tracking, since the target initialization strategy involves bilinearly sampling at the new (x_1, y_1) location whenever a new link is added to the chain. We therefore employ a second strategy, which is simply to always initialize $\mathcal{F}^0$ using the features found at the very first timestep. Intuitively, this locks the model into tracking the "original" target. This does create a slight mismatch between the target features and the convolutional feature maps on the current frame, but we find that this is effectively compensated for by the internal update mechanism. We have also tried a strategy of initializing with the last $\mathcal{F}^K$ output by the model, but this leads to

unstable behavior.

3.4 Implementation details

CNN: Our CNN uses the "BasicEncoder" architecture from the official RAFT codebase [63]. This architecture has a 7×7 convolution with stride 2, then 6 residual blocks with kernel size 3×3, then a final convolution with kernel size 1×1. The CNN has an output dimension of $C = 256$.

Local correlation pyramids: We use three levels in our correlation pyramids, with radius 4. This translates to three 9×9 correlation patches per timestep.

MLP-Mixer: The input to the MLP-Mixer is a sequence of relative coordinates, features, and correlation pyramids. The coordinates are made "relative" by subtracting the first position, meaning the first element of this sequence is $(0, 0)$, and other elements of the sequence can be interpreted as flow vectors originating from the first timestep's absolute position. The per-timestep inputs are flattened, then treated as a sequence of vectors (i.e., "tokens") for the MLP-Mixer. We use the MLP-Mixer architecture exactly as described in the original paper; at the end of the model there is a mean over the sequence dimension, followed by a linear layer that maps to a channel size of $T \cdot (C + 2)$.

Updates: We reshape the MLP-Mixer's outputs into a sequence of feature updates and a sequence of coordinate updates. We apply the coordinate updates directly (with addition). We use separate linear layers to apply the updates for the instance-level and pixel-level features (i.e., ϕ in Equation 1 from the main paper). We train and test with 8 updates, but we find that performance is similar if we train with 4 (and still test with 8).

Visibility: We use a linear layer to map the last update iteration's pixel-level feature sequence into visibility logits.

Optimization: We train with a batch size of 4, distributed across four GPUs. At training time, we use a resolution of 368×496. For each element of the batch, we randomly sample 32 trajectories whose initial timestep is marked "visible" in the ground truth (so that the tracking targets always begin within-bounds and unoccluded). We train for 500,000 steps, with a learning rate of 1e-4 with a 1-cycle schedule [56], using the AdamW optimizer and clipping gradients to $[-1, 1]$. Training

takes approximately 5 days on four GeForce RTX 2080s.

Hyperparameters: We use $T = 8$ (timesteps considered by the MLP-Mixer), and $K = 8$ (update iterations). The model can in general be trained for any T, but we found that the model was more difficult to train at $T = 16$ and $T = 32$, likely because the complexity of trajectories grows rapidly with their length under our model, as there is no weight sharing across time. On the other hand, the temporal sensitivity allows our model to learn more complex temporal priors. We found that $K > 8$ performs similar to $K = 8$, and so use $K = 8$ because it is faster.

Complexity: *Speed:* When the number of targets is small enough to fit on a GPU (e.g., 128 targets for a 12G GPU), our model is faster than RAFT (340ms vs. 2000ms at 480×1024). RAFT is comparatively slow because (1) it is too memory-heavy to compute all frames' flows in parallel, so we must run it $T - 1$ times, and (2) much computation is spent on non-target pixels. *Memory:* Our model's memory scales with $T \cdot N$, where N is the number of particles being tracked, due to the iterated MLP-Mixer which consumes a T-length sequence of features per particle.

Code: PIPs is implemented in Pytorch [42]. We will publicly release the code for the model and training procedure.

3.5 Experiments

We train our model in a modified version of FlyingThings [38], which we name FlyingThings++ (discussed more below).

We evaluate our model on tracking objects in FlyingThings++, tracking vehicles and pedestrians in KITTI [12], tracking heads in a crowd in CroHD [61], and finally, propagating keypoints in animal videos in BADJA [2]. We visualize trajectory estimates in DAVIS videos in Figure 3.1, to illustrate the method's generality, and visualize the estimates against ground truth in Figures 3.4 and 3.5.

Please refer to the project page[1] for video visualizations of our results on these datasets.

[1] https://particle-video-revisited.github.io/

3.5.1 Training data: FlyingThings++

We created a synthetic dataset based on FlyingThings [38], which is a dataset typically used to train optical flow models (usually in combination with other datasets). We chose FlyingThings because (1) its visuals and motions are extremely complex, which gives hope of generalizing to other data, and (2) it provides 10-frame videos with ground-truth forward and backward optical flow (as opposed to 2-frame videos), from which we can mine multi-step trajectory ground truth.

To create ground truth multi-frame trajectories, we chain the ground-truth flows forwards, and discard chains that fail a forward-backward consistency check (e.g., by landing on occluders). Through this process we create a sparse set of 4-frame and 8-frame trajectories. To this set we also add 2-frame trajectories from the raw flow fields.

The forward-backward consistency check ensures that the trajectories are accurate, but it leaves us with a library of trajectories where the target is visible on every timestep. Therefore, it is necessary to add *new occlusions* on top of the video. We do this on-the-fly during batching: for each FlyingThings video in the batch, we randomly sample an object from an alternate FlyingThings video, paste it directly on top of the current video, overwriting the pixels within its mask on each frame. We then update the ground-truth to reflect the new visibility in the occluded area on each frame, as well as update the trajectory list to include the trajectories of the added object.

Combining all videos with at least 32 valid 8-frame trajectories, we obtain a total of 4311 training videos, and 734 test videos. To expand the breadth of the training set, we augment the data on-the-fly with color and brightness changes, random scale changes, random crops which randomly shift across time, random Gaussian blur, and random horizontal and vertical flips.

3.5.2 Baselines

In our experiments we consider the following baselines.

Recurrent All-Pairs Field Transforms (RAFT) [62] represents the state-of-the-art in optical flow estimation, where a high resolution flow field is refined through iterative updates, based on lookups from a 4D cost volume constructed

between all pairs of pixels. Note that similar to our method, RAFT has been trained on FlyingThings (including occlusions and out-of-bounds motions), but only has a 2-frame temporal span. To generate multi-frame trajectories with RAFT at test time, we compute flow with all consecutive pairs of frames, and then compute flow chains at the pixels queried on the first frame. To continue chains that travel out of bounds, we clamp the coordinates to the image bounds and sample at the edge of the flow map.

DINO [6] is a vision transformer (ViT-S [11] with patch size 8) trained on ImageNet with a self-supervision objective based on a knowledge distillation setup that builds invariance to image augmentations. To use this model for multi-frame correspondence, we use the original work's code for instance tracking, which uses nearest neighbor between the initial frame and the current frame, as well as nearest-neighbor between consecutive frames, and a strategy to restrict matches to a local neighborhood around previous matches. We report results with and without this "windowing" strategy.

TimeCycle [70] learns correspondences between pixels from different frames by optimizing an objective that encourages correspondences to be cycle-consistent across time (i.e., forward-backward consistency), including across frame skips. This method tracks the same way as DINO, by computing feature affinity across frames and reporting nearest neighbors.

Contrastive Random Walk (CRW) [23] treats the video as a space-time graph, with edges containing transition probabilities of a random walk, and computes long-range correspondences by walking across the graph. Similar to TimeCycle [70], the model is trained with cycle-consistency.

Memory-Augmented Self-supervised Tracker (MAST) [29] learns correspondences between features by reconstructing the target frame with linear combinations of reference frames. At test time the correspondences are predicted autoregressively. The model is trained on OxUvA [66] and YouTube-VOS [74]

Video Frame-level Similarity (VFS) [73] learns an encoder that produces frame-level embeddings which are similar within a video, and dissimilar across videos. This model is trained on Kinetics-400 [26].

ImageNet ResNet [19] is a ResNet50 supervised for classification with ImageNet labels, and evaluated the same way as DINO.

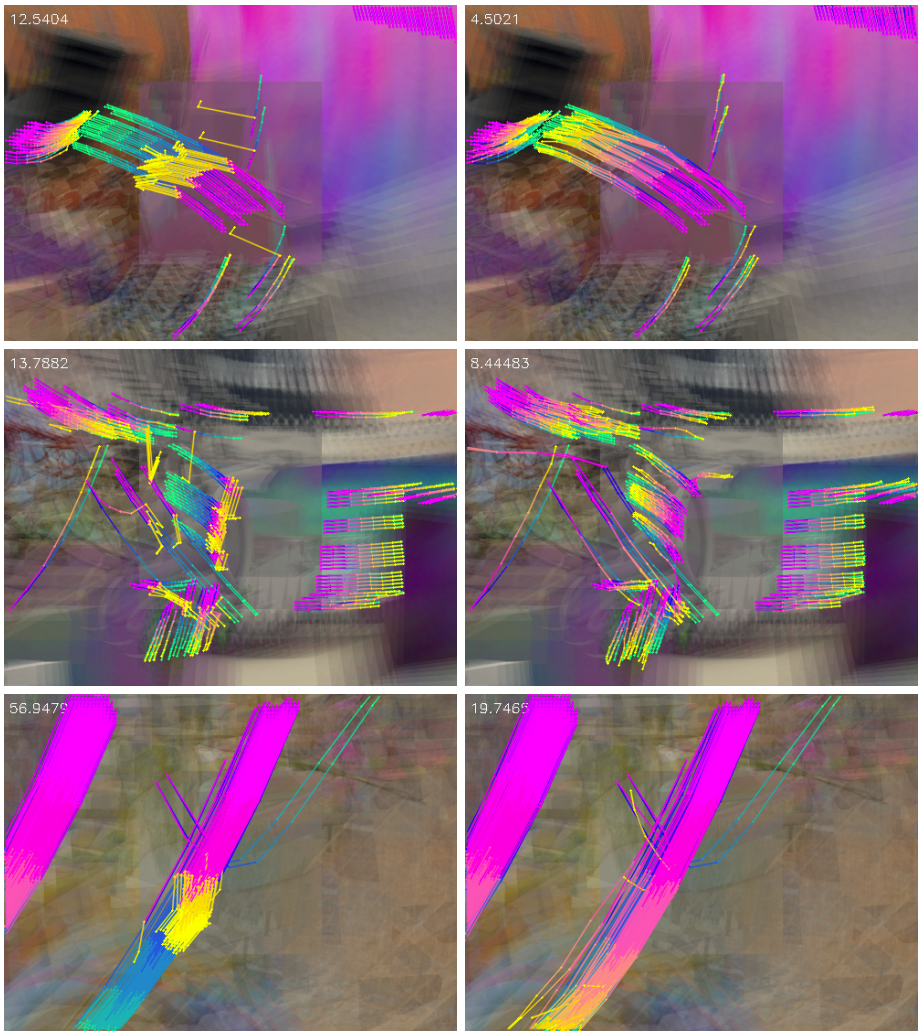

Figure 3.3: **Qualitative results in FlyingThings++** for RAFT (left column), and PIPs (our model, right column). Trajectories are drawn in a pink-to-yellow colormap, over ground-truth trajectories which are in blue-to-green, and in the background is the mean RGB of the sequence. Note that RAFT trajectories get "stuck" in the middle of the frame, due to occlusions there. The value displayed in the top left corner is the average error in the video.

3.5.3 Trajectory estimation in FlyingThings++

Using 8-frame videos from the FlyingThings++ test set as input, we estimate trajectories for all pixels for which we have ground-truth, and evaluate the Euclidean distance between each estimated trajectory and its corresponding ground truth, averaging

over all 8 timesteps. We are especially interested in measuring how our model's performance compares with the baselines when the target gets occluded or flies out of frame. To create controlled out-of-bounds and occlusion tests, we first take a central crop of each video, sized 368×496, providing us with some trajectories that fly out of the crop. Second, we implement a simple controllable occlusion strategy, where we replace a 200×200 square in frames 2-5 with gray pixels.

We compare our model against DINO [6], representing the state-of-the-art for feature matching, and RAFT [62], representing the state-of-the-art for flow. Table 3.1 shows the results across the different evaluations on the test set. DINO struggles across all splits and performs worse on occluded pixels than on visible ones. RAFT obtains high accuracy for visible pixels, but its errors increase as the occlusions become more difficult. Our model performs similarly to RAFT on visible pixels, but it is somewhat robust to occlusions. Figure 3.3 shows visualizations of our results on the FlyingThings++ test data, compared to RAFT and ground truth. Inspecting the results manually, we see that RAFT's trajectories become "stuck" in the region of the added occluder, which makes sense because flows there do not reflect the motion of the targets. Our model, in contrast, is able to locate the targets after they re-emerge from the occluder, and inpaint the missing portions of the trajectories.

Method	Vis.	Occ.
DINO [6]	40.68	77.76
RAFT [62]	24.32	46.73
PIPs(ours)	**15.54**	**36.67**

Table 3.1: **Trajectory estimation error in FlyingThings++.** "Vis." evaluates pixels that stay visible and in-bounds; "OOB" evaluates pixels that fly outside the image bounds; "Occ." evaluates pixels that cross behind an occluder. PIP trajectories are more robust to targets moving out-of-bounds or becoming occluded.

3.5.4 Trajectory estimation in KITTI

We additionally evaluate on an 8-frame point trajectory dataset that we created from the "tracking" subset of the KITTI [12] urban scenes benchmark. To create 8-frame trajectories, we sample a 3D box annotation that has at least 8 valid timesteps, select a LiDAR point within the box on the first timestep, transform it in 3D to

Figure 3.4: **Qualitative results in FlyingThings++ (left), KITTI (middle), and CroHD (right).** We visualize a video with the mean of its RGB. We trace the estimates with pink-to-yellow trajectories, and show ground truth in blue-to-green. FlyingThings++ is chaotic, but training on this data allows our model to generalize.

its corresponding location on every other step, and project this location into pixel coordinates.

Method	Vis.	Occ.
DINO [6]	13.33	13.45
RAFT [62]	**4.03**	6.79
PIPs(ours)	4.40	**5.56**

Table 3.2: **Trajectory estimation error in KITTI.** PIP and RAFT trajectories are similar; DINO lags behind both.

In Table 3.2 we see that RAFT and our method perform approximately on par with one another (RAFT is slightly better on visible, while our method is slightly better on occluded), but DINO's error is nearly twice this. We evaluate on vehicles and pedestrians. Qualitative results for our model are shown in Figure 3.4-middle.

3.5.5 Trajectory estimation in CroHD

We additionally evaluate on the Crowd of Heads Dataset (CroHD) [61], which consists of high-resolution (1920 x 1080) videos of crowds, with annotations tracking the heads of people in the crowd. We evaluate on 8-frame sequences extracted from

the dataset, using an FPS of 12.5. We filter out targets whose motion is below a threshold distance, and split the evaluation between targets that are visible and those that undergo occlusions. The results are shown in Table 3.3. In this data, PIPs outperforms RAFT and DINO by a wide margin, both visibility settings. DINO struggles overall, likely because the motions in this dataset are small, and DINO is only able to track at a coarse resolution. Qualitative results for our model are shown in Figure 3.4-right.

Method	Vis.	Occ.
DINO [6]	22.50	26.06
RAFT [62]	7.91	13.04
PIPs(ours)	**5.16**	**7.56**

Table 3.3: **Trajectory estimation error in CroHD.** PIPs achieves better accuracy, for both visible and occluded targets.

3.5.6 Keypoint propagation in BADJA

BADJA [2] is a dataset of animal videos with keypoint annotations. These videos overlap with the DAVIS dataset [44], but include keypoint annotations. Keypoint annotations exist on approximately 1/5 frames, and the standard evaluation is Percentage of Correct Keypoint-Transfer (PCK-T), where keypoints are provided on a reference image, and the goal is to propagate these annotations to other frames. A keypoint transfer is considered correct if it is within a distance of $0.2\sqrt{A}$ from the true pixel coordinate, where A is the area of the ground-truth segmentation mask on the frame.

We note that some existing methods test on a simplified version of this keypoint propagation task, where the ground-truth segmentation is available on every frame of the video (e.g., [77, 78]). Here, we focus on the harder setting, where the ground-truth mask is unknown. Similarly, we have found that feature-matching methods (e.g., [6]) constrain their correspondences to a local spatial window around the previous frame's match. We report results for these methods with the qualifier "Windowed", but focus again on the un-constrained version of the problem, where keypoints need to be propagated from frame 1 to every other frame, with no other knowledge about

motion or position.

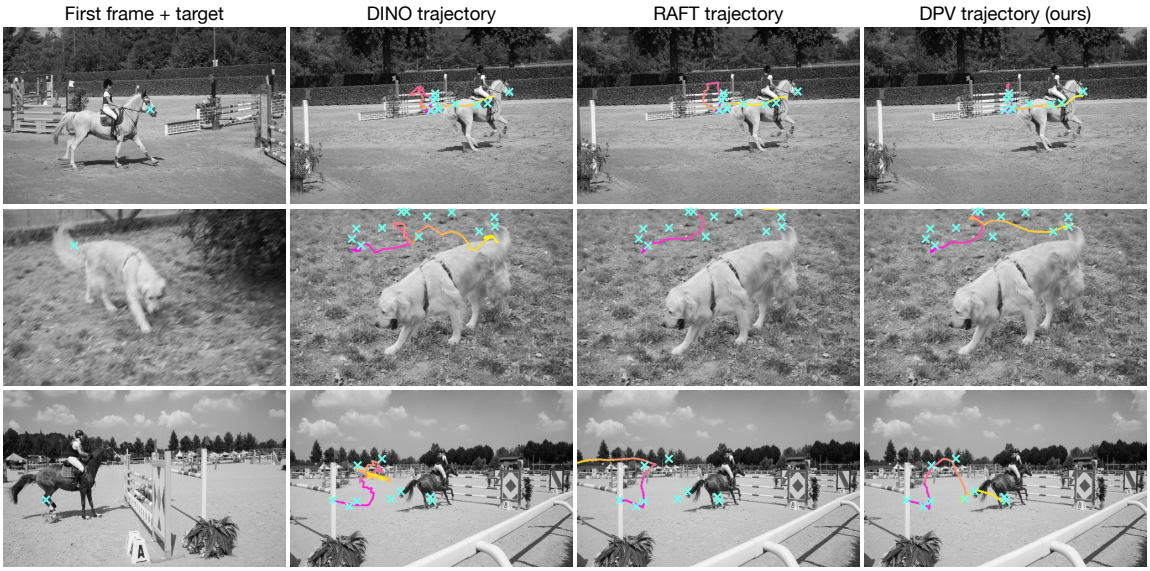

Figure 3.5: **Comparison with baselines in BADJA, on videos with occlusions.** For each method, we trace the estimated trajectory with a pink-to-yellow colormap. The sparse ground truth is visualized with cyan x's. In the video on the first row, all methods perform fairly well, though DINO and RAFT drift slightly toward the horse's body. In the second video, the target (on the dog's tail) leaves the image bounds then returns into view. In the third video, the target (on the horse's leg) is momentarily occluded, causing RAFT to lose track entirely. For a more detailed view of these results, please watch the supplementary video.

Table 3.4 shows the results of the BADJA evaluation. On four of the seven videos, our model produces the best keypoint tracking accuracy, as well as the best on average, by a margin of 9 points. DINO [6] obtains the best accuracy in the remaining videos, though its widest margin over our model is just 5 points. Interestingly, windowing helps DINO (and other baselines) in some videos but not in others, perhaps because of the types of motions in DAVIS. We note that DAVIS has an object-centric bias (i.e., the target usually stays near the center of the frame), which translation-sensitive methods like DINO can exploit, since their features encode image position embeddings; RAFT and PIPs track more generally. In Figure 3.5 we visualize DINO, RAFT, and PIP trajectories on targets that undergo momentary occlusions, illustrating how DINO tracks only coarsely, and how RAFT loses track after the occlusion.

Method	bear	camel	cows	dog-a	dog	horse-h	horse-l	Avg.
Win. DINO [6]	**77.9**	69.8	**83.7**	17.2	46.0	29.1	50.8	53.5
Win. ImageNet ResNet [19]	70.7	65.3	71.7	6.9	27.6	20.5	49.7	44.6
Win. TimeCycle [70]	13.6	10.0	8.0	3.4	9.8	7.9	13.1	9.4
Win. CRW [23]	63.2	75.9	77.0	6.9	32.8	20.5	22.0	42.6
Win. VFS [73]	63.9	74.6	76.2	6.9	35.1	27.2	40.3	46.3
Win. MAST [29]	35.7	39.5	42.0	10.3	8.6	12.6	14.7	23.3
Win. RAFT [62]	64.6	65.6	69.5	3.4	38.5	33.8	28.8	43.5
DINO [6]	75.0	59.2	70.6	10.3	**47.1**	35.1	56.0	50.5
ImageNet ResNet [19]	65.4	53.4	52.4	0.0	23.0	19.2	27.2	34.4
TimeCycle [70]	13.6	8.4	14.4	3.4	5.7	13.2	13.6	10.3
CRW [23]	66.1	67.2	64.7	6.9	33.9	25.8	27.2	41.7
VFS [73]	64.3	62.7	71.9	10.3	35.6	33.8	33.5	44.6
MAST [29]	51.8	52.0	57.5	3.4	5.7	7.3	34.0	30.2
RAFT [62]	64.6	65.6	69.5	13.8	39.1	37.1	29.3	45.6
PIPs (ours)	76.3	**81.6**	83.2	**34.2**	44.0	**57.4**	**59.5**	**62.3**

Table 3.4: **PCK-T in BADJA.** In this evaluation, keypoints are initialized in the first frame of the video, and are propagated to the end of the video; PCK-T measures the accuracy of this propagation. In each column, we bold the best result, and underline the second-best. Above the middle bar, we give methods a spatial window (marked "Win.") to constrain how they propagate labels, which encodes domain knowledge about the span of plausible motions in the domain (which is a common strategy in existing work). Below the bar, we run each method in the unconstrained setting. Our method wins in most videos, but DINO performs well also.

3.5.7 Ablation on visibility-aware linking

In this section we evaluate the effect of visibility-aware trajectory linking instead of naive linking. Our trajectory linking strategy relies on visibility estimates produced by the model. Without using these estimates, we may still link trajectories naively (i.e., greedily), by chaining the 8-frame trajectories end-to-end. We evaluate this choice in BADJA keypoint propagation, and show the result in Table 3.5. Naive linking indeed gives worse PCK-T, but the margin is only 0.2 points. This may suggest that the visibility estimator is not generalizing well to the new video domain. Alternatively, this may be showing a small effect because the metric is dominated by trajectories that stay visible for the full duration of the video, as evidenced by DINO

obtaining 50.5 PCK-T with no strategy for managing occlusions at all (see Table 3.4).

Method	Average PCK-T
Without visibility-aware linking	59.1
With visibility-aware linking	**59.3**

Table 3.5: **Effect of visibility-aware linking on keypoint propagation in BADJA.** Linking trajectories from locations estimated to be "visible" yields a small improvement in average PCK-T.

3.5.8 Experiment details

KITTI: We preprocess the KITTI [12] data by resizing the original images (of slightly varying resolution) to 512×320. The data is at 10 FPS, and we use this framerate as-is. We use videos from sequences 0000-0009, which include mostly vehicles, as well as 0017 and 0019, which include mostly pedestrians. We filter the data to only include targets that undergo an occlusion by another object. We do this by checking if (1) the target's box has an IOU greater than 0.5 with another annotated box, and (2) the target's box is behind (i.e., has a larger depth-axis value than) the intersecting box. Note that since the annotations are only at the box level, this data does not evaluate fine-grained motions within the objects, such as the movement of the legs or arms of a pedestrian.

CroHD: We preprocess the CroHD [61] data by cropping the original images of size 1920×1080 into crops of size 512×320. To make the data harder, we subsample the frames in the original dataset so that the FPS is reduced by a factor of 3. Since the dataset provides the ground truth for the head bounding boxes throughout the video and their visibility information, we define the ground truth point trajectory for a head as the trajectory of the center of its bounding box. In addition, we only test on heads that move (excluding ones that move less than 150px over the 8 frames). We test on sequences of 8 frames.

Runtime: On an 8-frame video with resolution 480×1024, the CNNs take approximately 6 ms, and the iterative inference for a batch of trajectories takes approximately 330 ms. If we require more trajectories than fit on the GPU simultaneously (e.g., if the GPU has 12G memory and we require more than 128 trajectories),

we simply split the set into multiple batches, and compute one batch at a time. Note it is possible to share the CNN features across these batches. Also note that when a small number of trajectories is required, our model runs much faster than RAFT, since RAFT always computes results densely (at approximately 2000 ms per 8-frame video). That is, as long as the number of trajectories is small enough to fit on the GPU in parallel, then our total time is less than 340 ms; but time will extend linearly as we exceed GPU capacity and demand more computation in serial.

3.5.9 Limitations

Our model has two main limitations. First is our unique extreme tradeoff, of spatial awareness for temporal awareness. Although this maximizes the power of the temporal prior in the model, it sacrifices potentially valuable information that could be shared between trajectories. We are indeed surprised that single-particle tracking performs as well as it does, considering that spatial smoothness is known to be essential for accurate optical flow estimation. Extending our architecture to concurrent estimation of multiple point trajectories is a direct avenue for future work.

Our second main limitation stems from the MLP-Mixer. Due to this architecture choice, our model is not recurrent across time. Although longer trajectories can be produced by re-initializing our inference at the tail of an initial trajectory, our model will lose the target if it stays occluded beyond the model's temporal window. We have tried models that are convolutional across time, and that use self-attention across the sequence length, but these did not not perform as well as the MLP-Mixer on our FlyingThings++ tests. Taking advantage of longer and potentially varying temporal context would help the model track through longer periods of ambiguity, and potentially leverage longer-range temporal priors.

3.6 Conclusion

We propose Pesistent Independent Particles (PIPs), a method for multi-frame point trajectory estimation through occlusions. Our method combines cost volumes and iterative inference with a multi-frame temporal deep network, which jointly reasons about location and appearance of visual entities across multiple timesteps. We

argue that optical flow, particle videos, and feature matches cover different areas in the spectrum of pixel-level correspondence tasks. Particle videos benefit from temporal context, which matching-based methods lack, and can also survive multi-frame occlusions, which is missing in flow-based methods. Given how tremendously useful optical flow and feature matching have been for driving progress in video understanding, we hope the proposed multi-frame trajectories will spark interest in architectures and datasets designed for longer-range fine-grained correspondences.

Chapter 4

Conclusion and Future Work

This thesis proposes two methods that introduce features in extra dimensions for modelling space and time. The first method uses features in the scene space for better BEV perception for autonomous vehicles. The second method attaches features over time to a point of interest for better tracking through occlusions. We show that introducing spatial and temporal priors results in better representation of the scene, and consequently better performance on the end tasks (e.g. segmentation, tracking).

For future work for our BEV segmentation baseline in Chapter 2, we are considering several directions: (1) our model currently only handles the task of BEV segmentation and we would like to see if the same architecture works well for 3D detection, forecasting, etc; (2) our model is currently implemented for a single time step only, and it is natural to extend it to also model time; (3) we showed that some factors (e.g. resolution, batch size) are more important than feature-lifting techniques, so it's worth trying new architectures (e.g. transformers); (4) to be more useful for autonomous driving applications, it would be interesting to optimize a behavioral end-task [15], where plans generated from model predictions are compared against plans generated by ground-truth information.

For future work for our point tracking method in Chapter 3, there are some potential directions too: (1) our model currently only tracks points, while tracking at multiple granularity levels together (e.g. points, parts, objects) might improve over tracking any single granularity level alone; (2) our model tracks every point independently, and it would be nice to allow entities of interest to communicate

and "support" each other to track everything better; (3) currently the "appearance" updates in our model is based on the correlation maps and attend directly to the features of the rest of the pixels, while it makes more sense to update the features in the neighborhood as well; (4) our model is trained for a fixed number of time steps, and we currently use visibility estimates to re-initialize point tracks in order to track over long time horizon; it would be nice to make the model able to handle videos of arbitrary lengths.

Another natural future direction is to explore more methods combining both the spatial and temporal priors, to get closer to a wholistic scene representation, where the foreground objects and the background are disentangled and propagated through time.

Bibliography

[1] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE international conference on image processing (ICIP)*, pages 3464–3468. IEEE, 2016. 3

[2] Benjamin Biggs, Thomas Roddick, Andrew Fitzgibbon, and Roberto Cipolla. Creatures great and SMAL: Recovering the shape and motion of animals from video. In *Asian Conference on Computer Vision*, pages 3–19. Springer, 2018. 2, 29, 35

[3] Thomas Brox and Jitendra Malik. Large displacement optical flow: Descriptor matching in variational motion estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:500–513, 2011. 21

[4] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. *arXiv:1903.11027*, 2019. 6, 9, 11

[5] Yigit Baran Can, Alexander Liniger, Ozan Unal, Danda Paudel, and Luc Van Gool. Understanding bird's-eye view of road semantics using an onboard camera. *IEEE Robotics and Automation Letters*, 7(2):3302–3309, 2022. 4, 5

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021. 19, 31, 33, 34, 35, 36, 37

[7] Ricson Cheng, Ziyan Wang, and Katerina Fragkiadaki. Geometry-aware recurrent neural networks for active visual recognition. In *NIPS*, 2018. 5

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009. 11, 15

[9] Victor Lempitsky Dmitry Ulyanov, Andrea Vedaldi. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, 2017. 8

[10] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas,

Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *ICCV*, pages 2758–2766, 2015. 21

[11] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021. 25, 31

[12] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 2, 29, 33, 38

[13] Hugo Germain, Vincent Lepetit, and Guillaume Bourmaud. Visual correspondence hallucination: Towards geometric reasoning. In *arXiv Preprint*, 2021. 22

[14] Nikhil Gosala and Abhinav Valada. Bird's-eye-view panoptic segmentation using monocular frontal view images. *IEEE Robotics and Automation Letters*, 2022. 4, 6

[15] Yiluan Guo, Holger Caesar, Oscar Beijbom, Jonah Philion, and Sanja Fidler. The efficacy of neural planning metrics: A meta-analysis of PKL on nuscenes. *CoRR*, abs/2010.09350, 2020. URL https://arxiv.org/abs/2010.09350. 41

[16] Adam Harley, Shrinidhi Kowshika Lakshmikanth, Paul Schydlo, and Katerina Fragkiadaki. Tracking emerges by looking around static scenes, with neural 3d mapping. In *ICLR*, 2020. 2

[17] Adam W. Harley, Zhaoyuan Fang, and Katerina Fragkiadaki. Particle videos revisited: Tracking through occlusions using point trajectories, 2022. URL https://arxiv.org/abs/2204.04153. 1, 2

[18] Adam W. Harley, Zhaoyuan Fang, Jie Li, Rares Ambrus, and Katerina Fragkiadaki. A simple baseline for bev perception without lidar, 2022. URL https://arxiv.org/abs/2206.07959. 1, 2

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 8, 9, 31, 37

[20] Noureldin Hendy, Cooper Sloan, Feng Tian, Pengfei Duan, Nick Charchut, Yuesong Xie, Chuang Wang, and James Philbin. Fishing net: Future inference of semantic heatmaps in grids. *arXiv preprint arXiv:2006.09917*, 2020. 4, 5, 6, 11, 12, 13

[21] Anthony Hu, Zak Murez, Nikhil Mohan, Sofía Dudas, Jeffrey Hawke, Vijay Badrinarayanan, Roberto Cipolla, and Alex Kendall. Fiery: Future instance

prediction in bird's-eye view from surround monocular cameras. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15273–15282, 2021. 5, 10, 12, 13

[22] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016. 21

[23] Allan Jabri, Andrew Owens, and Alexei A Efros. Space-time correspondence as a contrastive random walk. *Advances in Neural Information Processing Systems*, 2020. 22, 31, 37

[24] Joel Janai, Fatma G"uney, Anurag Ranjan, Michael J. Black, and Andreas Geiger. Unsupervised learning of multi-frame optical flow with occlusions. In *European Conference on Computer Vision (ECCV)*, volume Lecture Notes in Computer Science, vol 11220, pages 713–731. Springer, Cham, September 2018. 22

[25] Wei Jiang, Eduard Trulls, Jan Hosang, Andrea Tagliasacchi, and Kwang Moo Yi. COTR: Correspondence Transformer for Matching Across Images. In *ICCV*, 2021. 22

[26] Will Kay, Joao Carreira, Karen Simonyan, Brian Zhang, Chloe Hillier, Sudheendra Vijayanarasimhan, Fabio Viola, Tim Green, Trevor Back, Paul Natsev, et al. The kinetics human action video dataset. *arXiv:1705.06950*, 2017. 31

[27] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018. 10

[28] Z. Lai and W. Xie. Self-supervised learning for video correspondence flow. In *BMVC*, 2019. 22

[29] Zihang Lai, Erika Lu, and Weidi Xie. MAST: A memory-augmented self-supervised tracker. In *CVPR*, 2020. 19, 22, 31, 37

[30] Peizhao Li, Pu Wang, Karl Berntorp, and Hongfu Liu. Exploiting temporal relations on radar perception for autonomous driving. *arXiv:2204.01184*, 2022. 3, 4, 6, 13

[31] Qi Li, Yue Wang, Yilun Wang, and Hang Zhao. Hdmapnet: A local semantic map learning and evaluation framework. *arXiv preprint arXiv:2107.06307*, 2021. 5, 11

[32] Zhiqi Li, Wenhai Wang, Hongyang Li, Enze Xie, Chonghao Sima, Tong Lu, Qiao Yu, and Jifeng Dai. Bevformer: Learning bird's-eye-view representation from multi-camera images via spatiotemporal transformers. *arXiv preprint*

arXiv:2203.17270, 2022. xi, 3, 4, 6, 10, 11, 12, 13, 15

[33] Teck-Yian Lim, Amin Ansari, Bence Major, Daniel Fontijne, Michael Hamilton, Radhika Gowaikar, and Sundar Subramanian. Radar and camera early fusion for vehicle detection in advanced driver assistance systems. In *Machine Learning for Autonomous Driving Workshop at the 33rd Conference on Neural Information Processing Systems*, volume 2, page 7, 2019. 5, 6

[34] Buyu Liu, Bingbing Zhuang, Samuel Schulter, Pan Ji, and Manmohan Chandraker. Understanding road layout from videos as a whole. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4414–4423, 2020. 5

[35] Buyu Liu, Bingbing Zhuang, and Manmohan Chandraker. Weakly but deeply supervised occlusion-reasoned parametric layouts. *arXiv preprint arXiv:2104.06730*, 2021. 5

[36] Jakob Lombacher, Kilian Laudt, Markus Hahn, Jürgen Dickmann, and Christian Wöhler. Semantic radar grids. In *2017 IEEE intelligent vehicles symposium (IV)*, pages 1170–1175. IEEE, 2017. 6

[37] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017. 11

[38] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *CVPR*, 2016. 2, 20, 29, 30

[39] Michael Meyer and Georg Kuschk. Deep learning based 3d object detection for automotive radar and camera. In *2019 16th European Radar Conference (EuRAD)*, pages 133–136. IEEE, 2019. 6

[40] Bowen Pan, Jiankai Sun, Ho Yin Tiga Leung, Alex Andonian, and Bolei Zhou. Cross-view semantic segmentation for sensing surroundings. *IEEE Robotics and Automation Letters*, 5(3):4867–4873, 2020. 3, 5, 11

[41] Michael Parker. Chapter 20 – automotive radar. In Michael Parker, editor, *Digital Signal Processing 101 (Second Edition)*, pages 253–276. Newnes, second edition edition, 2017. ISBN 978-0-12-811453-7. doi: https://doi.org/10.1016/B978-0-12-811453-7.00020-2. URL https://www.sciencedirect.com/science/article/pii/B9780128114537000202. 4, 6

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox,

and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. 29

[43] Jonah Philion and Sanja Fidler. Lift, splat, shoot: Encoding images from arbitrary camera rigs by implicitly unprojecting to 3d. In *European Conference on Computer Vision*, pages 194–210. Springer, 2020. xi, 4, 5, 10, 11, 12, 14, 16

[44] Jordi Pont-Tuset, Federico Perazzi, Sergi Caelles, Pablo Arbeláez, Alexander Sorkine-Hornung, and Luc Van Gool. The 2017 DAVIS challenge on video object segmentation. *arXiv:1704.00675*, 2017. 21, 35

[45] Cody Reading, Ali Harakeh, Julia Chae, and Steven L. Waslander. Categorical depth distribution network for monocular 3d object detection. In *CVPR*, 2021. 5

[46] Zhile Ren, Orazio Gallo, Deqing Sun, Ming-Hsuan Yang, Erik B Sudderth, and Jan Kautz. A fusion approach for multi-frame optical flow estimation. In *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2019. 20, 22

[47] Thomas Roddick and Roberto Cipolla. Predicting semantic map representations from images using pyramid occupancy networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11138–11147, 2020. xi, 6, 10, 12

[48] Avishkar Saha, Oscar Mendez Maldonado, Chris Russell, and Richard Bowden. Translating images into maps. *arXiv preprint arXiv:2110.00966*, 2021. xi, 4, 6, 10, 11, 12, 13

[49] Avishkar Saha, Oscar Mendez, Chris Russell, and Richard Bowden. Enabling spatio-temporal aggregation in birds-eye-view vehicle estimation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5133–5139. IEEE, 2021. 4, 6

[50] Agustín Salgado and Javier Sánchez. Temporal constraints in large optical flow estimation. In *International Conference on Computer Aided Systems Theory*, pages 709–716. Springer, 2007. 22

[51] P. Sand and S. Teller. Particle video: Long-range motion estimation using point trajectories. In *CVPR*, volume 2, pages 2195–2202, 2006. 1, 2, 19, 23

[52] Samuel Schulter, Menghua Zhai, Nathan Jacobs, and Manmohan Chandraker. Learning to look around objects for top-view representations of outdoor scenes. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 787–802, 2018. 4, 5

[53] Ole Schumann, Markus Hahn, Jürgen Dickmann, and Christian Wöhler. Semantic segmentation on radar point clouds. In *2018 21st International Conference on Information Fusion (FUSION)*, pages 2179–2186. IEEE, 2018. 6

[54] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2437–2446, 2019. 5

[55] Liat Sless, Bat El Shlomo, Gilad Cohen, and Shaul Oron. Road scene understanding by occupancy grid learning from sparse radar clusters using semantic segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision Workshops*, pages 0–0, 2019. 6

[56] Leslie N Smith and Nicholay Topin. Super-convergence: Very fast training of neural networks using large learning rates. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, volume 11006, page 1100612. International Society for Optics and Photonics, 2019. 28

[57] Samuel L Smith, Pieter-Jan Kindermans, Chris Ying, and Quoc V Le. Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*, 2017. 15

[58] Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019. 18

[59] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In *CVPR*, 2018. 21

[60] Narayanan Sundaram, Thomas Brox, and Kurt Keutzer. Dense point trajectories by GPU-accelerated large displacement optical flow. In *ECCV*, 2010. 21

[61] Ramana Sundararaman, Cedric De Almeida Braga, Eric Marchand, and Julien Pettre. Tracking pedestrian heads in dense crowd. In *CVPR*, pages 3865–3875, 2021. 2, 29, 34, 38

[62] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. In *European Conference on Computer Vision*, pages 402–419. Springer, 2020. 2, 19, 20, 21, 24, 25, 26, 30, 33, 34, 35, 37

[63] Zachary Teed and Jia Deng. RAFT: Recurrent all-pairs field transforms for optical flow. `https://github.com/princeton-vl/RAFT`, 2020. 28

[64] Ilya O. Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. MLP-mixer: An all-mlp architecture for vision. *ArXiv*, abs/2105.01601, 2021. 25

[65] Hsiao-Yu Fish Tung, Ricson Cheng, and Katerina Fragkiadaki. Learning spatial common sense with geometry-aware recurrent networks. *CVPR*, 2019. 5

[66] Jack Valmadre, Luca Bertinetto, Joao F Henriques, Ran Tao, Andrea Vedaldi,

Arnold WM Smeulders, Philip HS Torr, and Efstratios Gavves. Long-term tracking in the wild: A benchmark. In *ECCV*, pages 670–685, 2018. 31

[67] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017. 25

[68] Hengli Wang, Peide Cai, Yuxiang Sun, Lujia Wang, and Ming Liu. Learning interpretable end-to-end vision-based motion planning for autonomous driving with optical flow distillation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13731–13737. IEEE, 2021. 5

[69] Qianqian Wang, Xiaowei Zhou, Bharath Hariharan, and Noah Snavely. Learning feature descriptors using camera pose supervision. In *Proc. European Conference on Computer Vision (ECCV)*, 2020. 22

[70] Xiaolong Wang, Allan Jabri, and Alexei A. Efros. Learning correspondence from the cycle-consistency of time. In *CVPR*, 2019. 19, 22, 31, 37

[71] Xinshuo Weng, Jianren Wang, David Held, and Kris Kitani. 3d multi-object tracking: A baseline and new evaluation metrics. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 10359–10366. IEEE, 2020. 3

[72] Olivia Wiles, Sebastien Ehrhardt, and Andrew Zisserman. Co-attention for conditioned image matching. In *CVPR*, 2021. 22

[73] Jiarui Xu and Xiaolong Wang. Rethinking self-supervised correspondence learning: A video frame-level similarity perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 10075–10085, October 2021. 31, 37

[74] Ning Xu, Linjie Yang, Yuchen Fan, Jianchao Yang, Dingcheng Yue, Yuchen Liang, Brian Price, Scott Cohen, and Thomas Huang. Youtube-vos: Sequence-to-sequence video object segmentation. In *ECCV*, pages 585–601, 2018. 31

[75] Bin Yang, Runsheng Guo, Ming Liang, Sergio Casas, and Raquel Urtasun. Radarnet: Exploiting radar for robust perception of dynamic objects. In *European Conference on Computer Vision*, pages 496–512. Springer, 2020. 6

[76] Charig Yang, Hala Lamdouar, Erika Lu, Andrew Zisserman, and Weidi Xie. Self-supervised video object segmentation by motion grouping. In *ICCV*, 2021. 22

[77] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Huiwen Chang, Deva Ramanan, William T Freeman, and Ce Liu. LASR: Learning articulated shape reconstruction from a monocular video. In *Proceedings of the*

IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 15980–15989, 2021. 35

[78] Gengshan Yang, Deqing Sun, Varun Jampani, Daniel Vlasic, Forrester Cole, Ce Liu, and Deva Ramanan. Viser: Video-specific surface embeddings for articulated 3d shape reconstruction. In *NeurIPS*, 2021. 35

[79] Weixiang Yang, Qi Li, Wenxi Liu, Yuanlong Yu, Yuexin Ma, Shengfeng He, and Jia Pan. Projecting your view attentively: Monocular road scene layout estimation via cross-view transformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15536–15545, 2021. 6

[80] Tianwei Yin, Xingyi Zhou, and Philipp Krahenbuhl. Center-based 3d object detection and tracking. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11784–11793, 2021. 3, 12

[81] Tianwei Yin, Xingyi Zhou, and Philipp Krähenbühl. Multimodal virtual point 3d detection. *Advances in Neural Information Processing Systems*, 34, 2021. 3