2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

Unsupervised Motion Estimation of Vehicles Using ICP

Tom Roussel[1] Tinne Tuytelaars[1] Luc Van Eycken[1]

Abstract— **Anticipating the motion of dynamic objects is critical for making intelligent decisions navigating through an environment while avoiding collisions. In this work, we propose a CNN model that estimates 3D motion of objects using sequences of monocular images. We show that we can train this model without using any manual annotations by using Iterative Closest Points (ICP) to align pointclouds of an object at different points in time. We compare our unsupervised approach to a model that was trained using ground truth supervision, on the KITTI tracking dataset. We further improve our model by training our model on a larger dataset, which would otherwise not be possible due to the lack of ground truth data. We also compare our approach with a 3D object detector that estimates motion using a simple tracking scheme.**

I. Introduction

Quickly understanding what is happening around you is a crucial skill for survival. In the wild, anticipating the movement of others can be a great asset. For example, within ecology "ballistic interception" is an observed predatory behaviour in which the predator predicts the future location of its prey by observing its initial movement [1]. For robotics applications, this can be equally important. In applications such as autonomous navigation, it is useful to be able to anticipate how another car will move. This information can be used to plan a path to avoid the object and prevent any collisions. A closely related field is 3D object detection, which is usually tackled through machine learning. These models can be trained using annotated data. The same approach can be taken to estimate the 3D motion of an object. This does require the annotated data to consist of sequences, instead of single images.

In this work, we describe a method to train a model to predict the 3D motion of an object without using any manual 3D annotation. The lack of annotations requires us to somehow figure out how objects move in 3D space indirectly. One option could be tracking points over time and triangulating them, but this brings issues such as correspondence search and scale ambiguity. Instead, we look at using additional sensors to train the model.

In Autonomous Vehicle (AV) research, most data collecting vehicles are outfitted with a LiDAR sensor. We use the depth measurements provided by this sensor to train a model that estimates the 3D motion of other vehicles. The resulting model can then be used without the depth sensor, relying solely on the information in monocular camera images.

ICP, or Iterative Closest Points [2], is an algorithm that compares two 3D point clouds and finds the rotation and translation between them. ICP is a standard and well understood approach to register point clouds w.r.t. each other. If these 3D point clouds belong to the same object measured at different times, then the transformation between the two clouds is the 3D motion of that object. We use this to build a loss function to train a CNN model without supervision. In contrast with methods such as PnP, which match 2D points with a 3D reference point cloud, ICP finds a transformation between two 3D point clouds. This ensures the shape of the object is taken into account, not just its projection in the image plane. It is also reasonably resistant to noisy input data, as it is easy to integrate an outlier removal mechanism such as RANSAC into it.

We generate point clouds by using a pretrained image segmentation model, Mask-RCNN [3], to add semantic information to the measured LiDAR points. During training, we apply ICP on the segmented point clouds to estimate the 3D motion of all instances in the scene.

We consider this application from an AV point of view. This means that we must consider egomotion. The observed motion in the camera frame is comprised of two components: the actual motion of the object and the egomotion of the camera. For this work, we predict the observed motion. This simplifies the problem for our model, as it can focus on the motion of the object and ignore the moving background caused by the egomotion. Egomotion can be extracted using already established techniques such as visual odometry or SLAM.

This paper starts with a discussion of the related work in Section II. This is followed by an overview of the proposed unsupervised training method in Section III. We evaluate our model and compare it with some available baselines in Section IV. Finally, we conclude in Section V.

II. Related Work

A. 3D Object Detection

The goal of 3D Object Detection is to find objects and draw a 3D bounding box around them that encompasses the object. It is a combination of detecting instances, 6-DoF pose estimation of instances and estimation of its dimensions. It is a complex task that has only fairly recently received increased research attention due to its importance in monocular driving. The problem can be seen as a combination of detecting objects and accurately locating these objects.

A major distinction in the approaches is what data is used as input. There is a distinction to be made between methods that use a depth sensor such as LiDAR [4], [5], [6] and works that explore the more challenging setting of only using monocular images [7], [8], [9], [10], [11], [12].

[1] ESAT-PSI, KU Leuven.
firstname.lastname@esat.kuleuven.be

Though the performance of methods of the latter setting is typically much lower than methods using depth input, it has received significant attention as it avoids the costly LiDAR sensor.

A commonly employed pipeline for visual 2D object detection is comprised of two phases, the first being finding object proposals. The second phase is to look at each proposal and classify them as one of the trained classes or reject that proposal. Most 3D object detectors follow that same pipeline. Some works leverage the same technique as 2D object detectors and find regions of interest in the image, from which the 3D parameters are later extracted [8], [10], [12], [9].

M3D-RPN [11] distinguishes itself by finding 2D-3D proposals, instead of leveraging a 2D detector. These proposals take both the 3D and 2D bounding boxes into account. It regresses to the 3D parameters of each instance by combining both global features with a "depth-aware" convolution. These convolutions define unique kernels for different rows of the feature maps. E.g.: the top rows of the input feature map would have different filter kernels than the bottom rows.

Among the LiDAR based methods, the work of Han et al. [6] is particularly interesting for our work. Most object detectors use data from a single moment, but in their work they use a recursive neural network to detect objects from a continuous data stream. Though their work is evaluated on data belonging to a single frame, the authors mention that this can be easily extended to use data over longer periods of time. This could also be applied to tackle tracking problems, which brings this work close to our application.

B. Future Motion Prediction

Future motion prediction has seen some attention the last few years. There is extensive background on pixel level motion prediction [13], [14], but less so on the instance level [15], [16], [17]. There have been works where Newtonian physics is modeled to estimate the trajectory of rigid objects [15]. In the work of Mottaghi et al. [15], the authors model forces such as gravity, which influence the object and determine its trajectory. Other approaches try to extract the full trajectory of instances using an almost top-down view, as in [16]. They extract patches of the objects and train a Markov model to determine where the object moves towards.

A more recent approach is to relegate the motion estimation problem into a classification problem [17]. Kim et al. [17] train a model to estimate whether an object is moving slow, fast or not at all. It also estimates the direction it is moving in, quantized into 8 directions, parallel with the ground plane. To train their model, they manually annotate images from public datasets such as KITTI [18].

The methods mentioned above require ground truth information, in contrast to our method. For pixel level motion prediction, there are unsupervised methods available. For optical flow estimation, there is for example the work by Janai et al. [14]. In their work they train a model to estimate the optical flow between 3 consecutive RGB frames. They compute the optical flow to the future frame and the past frame separately, while also explicitly training a model to detect pixel-wise occlusions. In contrast to our method, this only gives information on the 2D motion of separate pixels, where we estimate the 3D rigid body motion on instance level. Hur et al. [13] propose an unsupervised training method for scene flow estimation, which estimates the 3D motion of pixels in a scene. They train their model using a photometric loss combined with a 3D reconstruction loss. They use a stereo sensor to resolve the scale ambiguity issue. Although this is more similar to our method, it estimates motion up to the pixel level, not the instance level. It is unclear if their method works well when estimating the motion of individual dynamic instances, as it is evaluated on the full scene. Errors due to a moving instance will not contribute as much to the global error metrics as errors caused by egomotion.

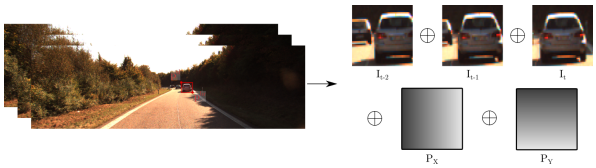

Fig. 1: Input to the network

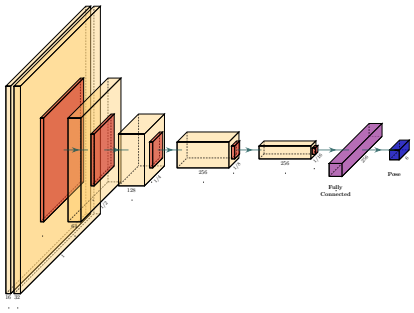

Fig. 2: Architecture used for our experiments.

III. Method

A. Model

The goal is to train a CNN to predict the 3D motion of an instance in a scene, using the camera input. As the goal is to estimate motion, it is beneficial to use sequences of images to exploit temporal information in the network. In more precise terms, we want to find the motion vector $m_{t\rightarrow t+1}$ of the instance, given images $I_{t-(N-1)}, I_{t-(N-2)} \dots I_t$. The motion vector is a 6 element vector containing the 3 translational coordinates and rotation is stored encoded in Euler angles (roll, pitch, yaw).

Using a pretrained instance segmentation network, in our case Mask-RCNN [3], we extract the bounding boxes of all the instances in the scene. To avoid the problems that come with instance tracking, we always use the bounding box at time t, the last element of our sequence of images. The full input images are cropped to the bounding box, for each image in the sequence. This allows the model to focus on how the instance moves. Cropping the image does remove

the location information of the instance, which can be an important cue to predict its motion. Adding 2D position information can help the model to learn priors related to the objects positioning. We encode this information for the network as 2 additional location channels, P_X, P_Y. Each element of these channels corresponds to respectively the original x and y coordinates of the corresponding pixel of the cropped image. Mathematically, P_X is expressed as follows:

$$P_X = \begin{bmatrix} p_x^{tl} & \dots & p_x^{tl} + \frac{i}{w}(p_x^{br} - p_x^{tl}) & \dots & p_x^{br} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ p_x^{tl} & \dots & p_x^{tl} + \frac{i}{w}(p_x^{br} - p_x^{tl}) & \dots & p_x^{br} \end{bmatrix}, \quad (1)$$

with w, h the width and height of the resized bounding box, and p^{tl}, p^{br} the top left and bottom right coordinates of the box respectively. i is the index of the columns of the channel. P_Y is defined analogously, this time varying along the rows instead of the columns. The cropped images, P_X, and P_Y are channel-wise concatenated before giving them to the CNN. A visual representation of the network input can be found in Figure 1.

As most standard architectures are designed for a single image input, we use a simple custom 7-layer CNN inspired by AlexNet [19] as shown in Figure 2. In this diagram, yellow blocks denote convolution layers, red blocks denote average pooling layers. We use ReLU as the activation function after each convolutional layer. The filter sizes of the first 2 layers are 7×7 and 5×5 respectively. The following layers have filters of size 3×3. The final layer is a fully connected layer, with a hyperbolic tangent ($\tanh$) as the activation function, multiplied by a predefined constant. This constrains the motion within plausible ranges. For our experiments, we feed 3 consecutive frames to the network. This model can be trained using both a supervised training method or an unsupervised training method.

B. ICP for Motion Prediction

The goal of this paper is to train the model specified above without supervision. To do so, we propose using the 3D information provided by a depth sensor. Say we have 2 point clouds, p_{t+1} and p_t, of an instance produced by this sensor, each measured at a different time. The transformation matrix T, which aligns one point cloud with the other, describes the apparent motion of the object instance. The ICP operation can be summarized as follows:

$$T_{\text{ICP}} = \text{ICP}(p_{t+1}, p_t) \rightarrow p_{t+1} \approx T_{\text{ICP}} \times p_t. \quad (2)$$

In our framework, we use ICP to measure the residual transformation, after applying the transformation estimated by our model, T_{estim}, to p_t. Ideally, this residual transformation should be zero, so it makes sense to use it to define the loss for training the model. Unfortunately, this operation is not differentiable. However, as shown by Mahjourian et al. [20], it is possible to use the output of the operation as a gradient w.r.t. T_{estim}. Using f_{ICP} to measure the misalignment of 2 point clouds w.r.t. T_{estim}, we get

$$f_{\text{ICP}}(T_{estim}, p_{t+1}, p_t) = \text{ICP}(p_{t+1}, T_{estim} \times p_t) \quad (3)$$

$$\frac{\partial f_{\text{ICP}}(T_{estim}, p_{t+1}, p_t)}{\partial T_{estim}} \approx -T_{\text{ICP}}. \quad (4)$$

Note that this "gradient" is not truly the gradient of the f_{ICP} function. It is instead the direction towards the minimum found by the ICP operation [20].

We use a LiDAR sensor to construct the 3D point cloud. However, this point cloud does not contain only the instance we need to align. We need to sample only the points that belong to that instance. Given depth maps D_t and D_{t+1}, we can easily project these points using the camera calibration matrix, but we are only interested in the points belonging to the target instance. For this we use the instance segmentation masks produced by Mask-RCNN as described in Section III-A, which we will refer to as M_t and M_{t+1}. Constructing the two point clouds is then done as follows:

$$p_t = \left\{ D_t(i,j) \cdot K^{-1} \times [i,j,1]^T \mid \forall i,j : M_t(i,j) = \rho \right\} \quad (5)$$

$$p_{t+1} = \{ D_{t+1}(i,j) \cdot K^{-1} \times [i,j,1]^T \mid \forall i,j : M_{t+1}(i,j) \neq 0 \}, \quad (6)$$

with K being the intrinsic calibration matrix and ρ the index of the target instance. We do not apply tracking, which eliminates tracking errors as a source of noise during training. This means we do not know the relationship between the instances of the two segmentation masks. This means that the point cloud p_{t+1} represents the whole scene, not only the instance. We do however filter "background" pixels from the point cloud. These are points that do not belong to any instance of the segmentation mask.

Before applying ICP, we first apply a transformation T_{io} to both point clouds to make sure that all subsequent transformations are local to the object itself. Ideally, T_{io} would translate the object center to the origin of the coordinate system. We approximate this by using the median point of p_t as the object center. This is not a perfect assumption, but it does yield good results in practice.

Our unsupervised loss function is the weighted sum of the translation and rotation residuals found by ICP. The output of the ICP procedure is a 4×4 transformation matrix, so we convert this to a 6 element vector. This conversion is represented as $\Pi_{\text{T}\rightarrow\text{3DRPY}}$.

$$T_{\text{ICP}} = f_{\text{ICP}}\left(T_{estim}, T_{io} \times p_{t+1}, T_{io} \times p_t\right) \quad (7)$$

$$\begin{bmatrix} \boldsymbol{t} \\ \boldsymbol{\omega} \end{bmatrix} = \Pi_{\text{T}\rightarrow\text{3DRPY}}\left(T_{\text{ICP}}\right) \quad (8)$$

Our final loss function is then the following:

$$\mathcal{L}_{ICP} = \lambda_{trans} ||\boldsymbol{t}||_2 + \lambda_{rot} ||\boldsymbol{\omega}||_2 \quad (9)$$

The ICP-based training procedure is also shown in the form of a diagram in Figure 3.

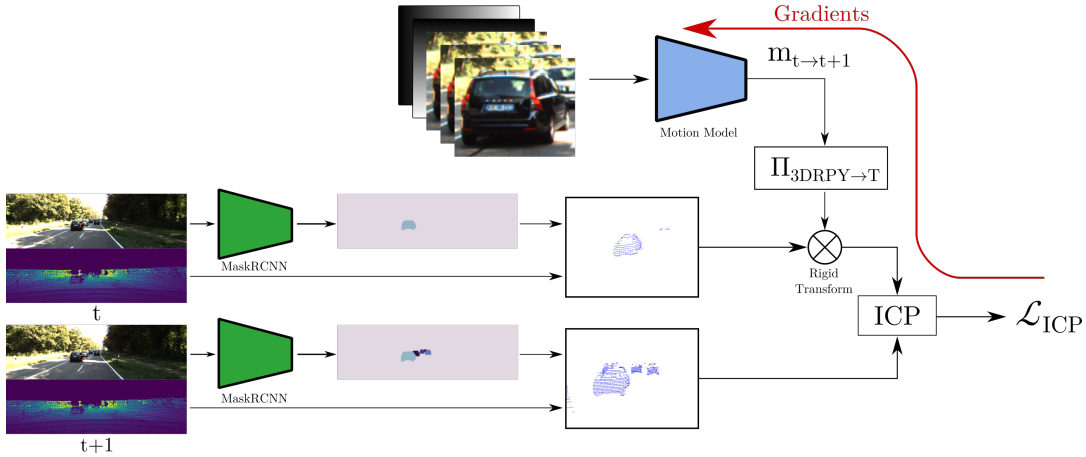

Fig. 3: Diagram showing the ICP training procedure

Unfortunately, the ICP procedure is not flawless. There are cases where point clouds are not aligned properly, which would give the model an incorrect supervision signal. Detecting situations such as this is difficult, but we found that cases where an abnormally large transformation was found this was due to misalignment. For each sample, we therefore apply a simple threshold τ_{thresh}. We discard a sample if for that sample: $\mathcal{L}_{\text{ICP}} > \tau_{thresh}$. As a future work, more advanced pointcloud registration techniques could be investigated to improve the quality of the alignment.

IV. Experiments

A. Dataset

We evaluate our results on the tracking dataset of KITTI [21]. This dataset contains 21 sequences of images, in which the 3D location of all vehicles are annotated. This is a subset of the "raw" KITTI dataset. The annotators have labeled each instance with a consistent ID throughout the sequence, which allows us to determine the ground truth 3D motion of the cars in a scene. The locations are annotated relative to the current position of the ego vehicle. This means we do not have to compensate for the egomotion while evaluating. We focus on predicting the 3D motion one frame into the future, which for the KITTI dataset means 0.1 second.

We create our train/validation/test split by randomly selecting subsequences of length 20 frames and dividing them in each category. The relative size of the splits is 75/12.5/12.5% for train, validation and test sets respectively. We extract the cropped images from the dataset before training using a pretrained Mask-RCNN [3] model. These cropped images are resized to 224×224 to be compatible with our network architecture. Each image has a variable amount of instances in it, which results in a slight imbalance in the data splits. The training set has 16 696 instances, the validation set has 3 202, and the test set contains 2 687 samples.

As our proposed method is unsupervised, we do not have to limit ourselves to data which has annotations. In other words, we can use the full KITTI dataset, not just the tracking subset. We simply refer to this larger dataset as the "raw" dataset. We remove the sequences that are also in the tracking subset. This ensures that there is no overlap between the raw dataset and any of the splits of the tracking dataset. The raw dataset contains 172 148 samples. Note that the raw dataset is not split into training/validation/test, as there is no ground truth to evaluate on.

Fig. 4: Example of the KITTI tracking dataset. Colors within the same sequences denote the same tracking IDs. Best viewed digitally.

Some examples of the KITTI tracking dataset can be found in Figure 4.

B. Baselines

As far as we know, at the time of writing there are no other unsupervised 3D motion estimation methods. In fact, most of the literature focuses on 3D object detection, not motion estimation. To fairly evaluate our method we show how it performs versus supervised training. We show results using the same training/validation/test split and the same architecture for both supervised and unsupervised training.

We evaluate the problem on 2 trivial estimators: "zero", a setting in which we estimate that the object remains motionless and "mean estimation", in which we assume that each object has the mean velocity of all objects in the training set.

As an additional baseline, we run an off-the-shelf 3D object detector on the test dataset. We use this object detector to estimate the motion between $t-1$ and t and assume that this velocity does not change. We chose to use M3D-RPN [11], as this architecture reaches state-of-the-art performance and the code and model weights are freely

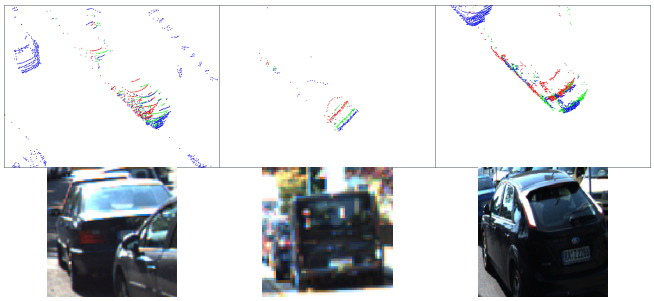

Fig. 5: Motion estimation results shown in 3D point clouds from a top-down view. The red is the initial position of the instance, the blue points show the future position and the green points show the initial points moved along the motion estimated by the model. The instance being matched is shown below the point cloud.

available. This architecture outputs both 2D and 3D bounding boxes. To find object motions, we need to track the detections over time. We use a simple solution: we compute the IoUs of the bounding boxes in consecutive frames, the instances that overlap the most, are considered to be the same instance. These are then matched with a ground truth annotation, again by finding the match with the highest IoU.

C. Training Settings

We investigate different training strategies in the form of an ablation study. Our ideal case is a model trained on both the large "raw" dataset and the tracking dataset. This is referred to as "*finetuned*". We then systematically change the training setting. We train a model on only the larger "raw" dataset, meaning that the model must generalize to the test set of the tracking dataset, we call this "*only raw*". We also show how the model performs when only training on the smaller tracking dataset, ignoring the large set. In this case, the model has less data to learn from, but the data is closer to the test set. We refer to this as "*only tracking*". We also show the effect of the ICP threshold by disabling it, referred to as "*no threshold*".

All the results of the aforementioned training settings were found by selecting the best performing model (on the validation set) after a random grid search over the hyper parameters. The results of these experiments can be found in Table I.

D. Results and Discussion

a) Main results and ablation study: The results of the different settings can be found in Table I. We show both the mean as the median errors of both the translation as the rotation errors. We also provide some qualitative results of our method in Figure 5. Our unsupervised approach does not reach the same accuracy as the supervised upper bound, though it gets relatively close with a median error of 0.245 m, compared with the supervised result of 0.192 m. Furthermore, even the training setting that performs the worst outperforms the zero and mean estimation baselines.

The ICP threshold constraint does help the performance of the model after training. This suggests that this simple constraint helps filter bad point cloud registrations during training. When training on the much larger "raw" dataset, we found that the median error decreases significantly, supporting the idea that more data helps the network generalize better to the test set.

TABLE I: Results on the KITTI tracking dataset

Model	Translation [m]		Rotation [°]	
	Mean	Median	Mean	Median
Ours – supervised	0.301	0.192	0.859	0.614
Zero	0.750	0.772	**0.398**	**0.0608**
Mean Estimation	0.525	0.438	0.498	0.190
Ours – unsupervised – no threshold	0.418	0.317	0.487	0.201
Ours – unsupervised – only tracking	0.396	0.289	0.610	0.303
Ours – unsupervised – only raw	0.392	0.253	0.783	0.451
Ours – unsupervised – finetuned	**0.363**	**0.245**	0.624	0.316

Notably, the rotation errors are larger for each method than simply estimating no rotation. We believe this is due to the rotations between frames for any instance being small in magnitude. We found that estimating rotations that are less than 1 degree accurate has proved difficult. An alternative explanation is that the annotations are not accurate enough for these small angle differences, as they were manually annotated. Unfortunately, the KITTI dataset does not provide an estimation of up to what scale their ground truth is accurate.

b) 3D object detection baseline: When comparing with the pretrained 3D object detector baseline, we noticed that the detector does not detect the same instances as Mask-RCNN. On the KITTI tracking dataset, it detected fewer instances, which were usually vehicles that are more clearly visible. To keep the evaluation fair, the metrics in Table II are only computed on samples detected by both methods. These results indicate that the unsupervised method on average performs similarly to the 3D object detector when looking at the translation error, though the median error is lower for M3D-RPN. Note that our unsupervised model has never seen any 3D annotations, unlike M3D-RPN. Both our supervised and unsupervised models clearly outperform M3D-RPN in estimating the rotation difference, beating it in both the mean and median errors.

The M3D-RPN baseline has a small issue, in that it has been trained on the KITTI object detection dataset. There may be some overlap with the object detection training set and our motion estimation test set. Nevertheless, this baseline is still worth considering to show that object detection and motion estimation are not equivalent fields. Our methods exploit the temporal information of several frames, which results in more accurate results.

We also evaluate how our model performs in relation to the distance of the object. We show the performance of our model, both supervised and unsupervised, in a set of boxplots in Figure 6. The unsupervised model in this figure is the aforementioned *finetuned* model. The plots show the distribution of errors by our model for different ranges of distances. For our unsupervised model, the distance to the

TABLE II: Comparison with pretrained baseline on the KITTI tracking dataset. Only evaluated with mutually detected samples.

Model	Translation [m]		Rotation [°]	
	Mean	Median	Mean	Median
M3D-RPN [11]	0.294	0.152	10.517	0.813
Ours - supervised	**0.204**	**0.142**	0.717	0.622
Ours - unsupervised	0.293	0.241	**0.401**	**0.220**

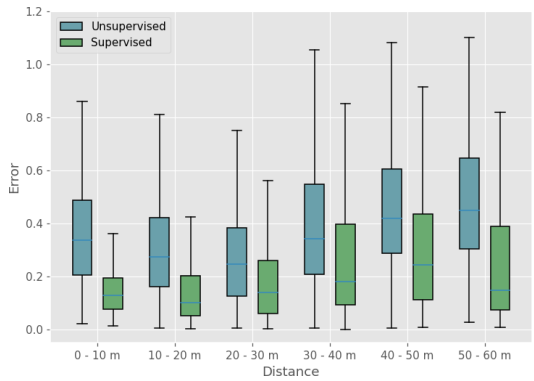

Fig. 6: Boxplots showing model errors w.r.t. the distance to the instance

instance clearly affects its performance, with larger errors for objects that are close to the camera and for those that are far.

E. ICP Loss Evaluation

To better understand the previous observation, we investigate whether the ICP loss somehow behaves differently when applied to objects at different distances. We created a similar boxplot evaluating the ICP error in Figure 7. Here, we find that this trend continues. The lowest errors are found in the 20m-30m range.

The sources of errors are different when the object is far from the camera and when it is close to the camera. When considering instances far from the camera there are a few problems. As they are far away, the instances sometimes become too small to be detected by the instance segmentation model. This causes the point cloud at time $t+1$ to incorrectly categorize the instance as being part of the background. Another problem is related to the resolution of the depth

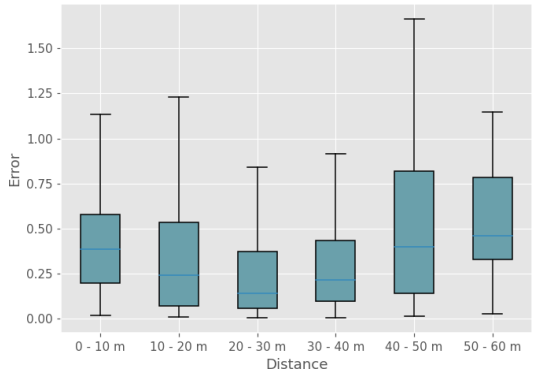

Fig. 7: Boxplot showing ICP errors w.r.t. the distance to the instance

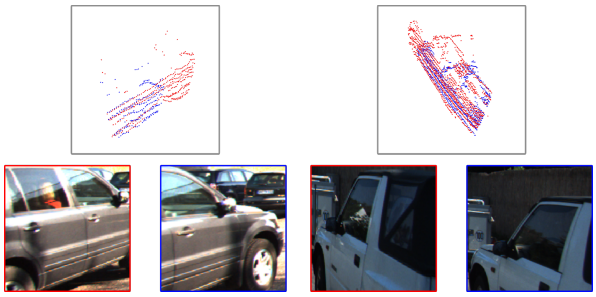

Fig. 8: Two examples of pointclouds at the edge of the frame. Point clouds at two points in time are shown in blue and red. Note that the length of the clouds are different as less of the instance is visible.

sensor. When objects are far away there are fewer beams sent by the LiDAR sensor that land on the surface of the instance. This results in point clouds with very few points.

On the other hand, instances closer to the camera suffer more from partial occlusions. The vehicles closest to the camera are usually in the process of passing the recording vehicle. This causes the front part of the object to not be visible in the point cloud at $t+1$, which leads to the compared point clouds looking very different. A visual example of this can be seen in Figure 8, here point clouds at t and $t+1$ are shown. At $t+1$, part of the object is not visible leading to a smaller point cloud. This also leads to inaccurate alignment by ICP.

V. Conclusions

In this work we showed that it is possible to use geometric constraints to train a model to predict the future 3D motion of objects without needing any annotations. Our proposed method does require depth measurements by a sensor such as LiDAR, which many vehicles for autonomous navigation research have equipped. We verified our method by training and evaluating on the KITTI tracking dataset, comparing between supervised and unsupervised methods.

The performance of our approach gets close to the performance of a supervised model without needing any ground truth data. We were able to close the gap between supervised and unsupervised training by using a wider dataset, that would otherwise require significant effort to manually annotate.

We evaluated the ICP loss function to find the source of the discrepancy between the two training settings. We found a correlation between the registration error and the distance from the camera. Instances that are far from the camera, usually only had very few points to accurately match with and also had a high probability to be occluded between two frames. Instances very close to the camera were often instances that were in the process of moving out of the frame. This caused occlusion problems decreasing the accuracy of the ICP alignment, which reflected on the performance of the unsupervised model.

References

[1] T. Y. Moore and A. A. Biewener, "Outrun or Outmaneuver: Predator-prey interactions as a model system for integrating biomechanical

studies in a broader ecological and evolutionary context," *Integrative and Comparative Biology*, vol. 55, no. 6, pp. 1188–1197, 2015.

[2] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, p. 698–700, Sep 1987.

[3] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, p. 2961–2969.

[4] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun 2019, p. 12689–12697. [Online]. Available: https://ieeexplore.ieee.org/document/8954311/

[5] J. Ngiam, B. Caine, W. Han, B. Yang, Y. Chai, P. Sun, Y. Zhou, X. Yi, O. Alsharif, P. Nguyen, and et al., "Starnet: Targeted computation for object detection in point clouds," *arXiv:1908.11069 [cs]*, Dec 2019, arXiv: 1908.11069. [Online]. Available: http://arxiv.org/abs/1908.11069

[6] W. Han, Z. Zhang, B. Caine, B. Yang, C. Sprunk, O. Alsharif, J. Ngiam, V. Vasudevan, J. Shlens, and Z. Chen, "Streaming object detection for 3-d point clouds," *arXiv:2005.01864 [cs]*, May 2020, arXiv: 2005.01864. [Online]. Available: http://arxiv.org/abs/2005.01864

[7] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3d object detection for autonomous driving," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun 2016, p. 2147–2156. [Online]. Available: http://ieeexplore.ieee.org/document/7780605/

[8] A. Kundu, Y. Li, and J. M. Rehg, "3d-rcnn: Instance-level 3d object reconstruction via render-and-compare," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Jun 2018, p. 3559–3568. [Online]. Available: https://ieeexplore.ieee.org/document/8578473/

[9] B. Xu and Z. Chen, "Multi-level fusion based 3d object detection from monocular images," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Jun 2018, p. 2345–2353. [Online]. Available: https://ieeexplore.ieee.org/document/8578347/

[10] A. Mousavian, D. Anguelov, J. Flynn, and J. Kosecka, "3d bounding box estimation using deep learning and geometry," *arXiv:1612.00496 [cs]*, Apr 2017, arXiv: 1612.00496. [Online]. Available: http://arxiv.org/abs/1612.00496

[11] G. Brazil and X. Liu, "M3d-rpn: Monocular 3d region proposal network for object detection," *arXiv:1907.06038 [cs]*, Aug 2019, arXiv: 1907.06038. [Online]. Available: http://arxiv.org/abs/1907.06038

[12] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau, "Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image," *arXiv:1703.07570 [cs]*, Mar 2017, arXiv: 1703.07570. [Online]. Available: http://arxiv.org/abs/1703.07570

[13] J. Hur and S. Roth, "Self-supervised monocular scene flow estimation," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun 2020, p. 7394–7403. [Online]. Available: https://ieeexplore.ieee.org/document/9157146/

[14] J. Janai, F. Güney, A. Ranjan, M. Black, and A. Geiger, *Unsupervised Learning of Multi-Frame Optical Flow with Occlusions*, ser. Lecture Notes in Computer Science. Springer International Publishing, 2018, vol. 11220, p. 713–731.

[15] R. Mottaghi, H. Bagherinezhad, M. Rastegari, and A. Farhadi, "Newtonian image understanding: Unfolding the dynamics of objects in static images," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Jun 2016, p. 3521–3529. [Online]. Available: http://ieeexplore.ieee.org/document/7780752/

[16] J. Walker, A. Gupta, and M. Hebert, "Patch to the future: Unsupervised visual prediction," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, Jun 2014, p. 3302–3309. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6909818

[17] K.-R. Kim, W. Choi, Y. J. Koh, S.-G. Jeong, and C.-S. Kim, "Instance-level future motion estimation in a single image based on ordinal regression," in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Oct 2019, p. 273–282. [Online]. Available: https://ieeexplore.ieee.org/document/9009521/

[18] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[19] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Communications of the ACM*, vol. 60, no. 6, p. 84–90, May 2017.

[20] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

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