Taskology: Utilizing Task Relations at Scale (Supplemental Material)

Yao Lu[1 2], Sören Pirk[1,2], Jan Dlabal[2], Anthony Brohan[1,2], Ankita Pasad[3]*,
Zhao Chen[4], Vincent Casser[4], Anelia Angelova[1,2], Ariel Gordon[1,2]
[1]Robotics at Google, [2]Google Research, [3]Toyota Technological Institute at Chicago, [4]Waymo LLC
{yaolug, pirk, dlabal, brohan}@google.com, ankitap@ttic.edu,
{zhaoch, casser}@waymo.com, {anelia, gariel}@google.com

1. Scene depth, segmentation and ego-motion

1.1. Modules and Interfaces

The interfaces of the three modules, depth, motion and semantic segmentation, are defined below:

Motion Prediction Network: Given two consecutive RGB frames, $I_1(i,j)$ and $I_2(i,j)$, of width w ($0 \leq j < w$) and height h ($0 \leq i < h$), the motion prediction network predicts the following quantities:

- $\delta t_{1\to 2}(i,j)$: For every pixel (i,j), $\delta t_{1\to 2}(i,j)$ estimates the movement of the point visible at the pixel (i,j) of frame 1, relative to the scene, which occurred between frame 1 and frame 2.
- $T_{1\to 2}$: The translation vector of the camera between frame 2 and frame 1.
- $R_{1\to 2}$: The rotation matrix of the camera between frame 2 and frame 1.

Similarly, the network predicts $\delta t_{2\to 1}(i,j)$, $T_{2\to 1}$, and $R_{2\to 1}$, which are defined as above, with (1) and (2) swapped.

Depth Prediction Network: Given an RGB frame, $I(i,j)$, the depth prediction network predicts a depth map, $z(i,j)$, for every pixel (i,j).

Semantic Segmentation Network: Given an RGB frame, $I(i,j)$, the semantic segmentation network predicts a logit map $l_c(i,j)$ for each class c. For each pixel (i,j), the class is given by $c(i,j) = \text{argmax}_c\ l_c(i,j)$.

1.2. Next frame warping

To construct the consistency losses for these tasks, as shown in the main paper, we need to derive the locations of each pixel from the first frame onto the next frame, which is also referred to as image warping from frame 1 to frame 2. We start with defining $m(i,j)$ to be the *movable mask*:

$$m(i,j) = \begin{cases} 1 & c(i,j) \in \mathcal{M} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$\mathcal{M}$ is the collection of all classes that represent movable objects. These are detailed below. For each pixel (i,j), $m(i,j)$ equals 1 if the pixel belongs to one of the movable object classes, and 0 otherwise.

Given two adjacent video frames, 1 and 2, a depth map of frame 1 $z_1(i,j)$, the camera matrix K, and a pixel position in homogeneous coordinates

$$p(i,j) = \begin{pmatrix} j \\ i \\ 1 \end{pmatrix}, \tag{2}$$

one can write the shift in p resulting from the rotation and a translation that occured between the two frames as:

$$\begin{aligned} z_1'(i,j)p_1'(i,j) &= KR_{1\to 2}K^{-1}z_1(i,j)p_1(i,j) \\ &+ K(m_1(i,j)\delta t_{1\to 2}(i,j) + T_{1\to 2}) \end{aligned} \tag{3}$$

where p_1' and z_1' are respectively the new homogeneous coordinates of the pixel and the new depth, projected onto frame 2, and K is the camera matrix. The above equation consists of the scene depth, as obtained by rigid motion of the scene and the additional changes obtained from the motions of the individually movable objects. Note that the motion mask is only applied to regions of potentially movable objects $m_1(i,j)$, determined by the semantic segmentation model. The movable mask $m_1(i,j)$ (of frame 1) restricts motion of objects relative to the scene to occur only at pixels that belong to movable objects.

1.3. Evaluation Protocol

In our experiment COCO served as the dedicated dataset for segmentation, and Cityscapes served as the unlabeled mediator dataset. Since the two datasets have different sets of labels, we had to create a mapping between the two. The mapping is shown in Table 1.

*Work done while at Robotics at Google

	Label ID		
Class	Ours	COCO 2017	Cityscapes
person/rider	1	1	24/25
bicycle	2	2	33
car	3	3	26
motorcycle	4	4	32
traffic lights	5	10	19
bus	6	6	28
truck	7	8	27
others	8	other labels	other labels

Table 1: Mapping between Cityscapes label IDs, COCO labels IDs, and the label IDs we defined for this experiment.

Only labels that represent movable objects are of interest for our experiment. We therefore restricted our label set to 7 classes, that are in the intersection of Cityscapes and COCO and represent movable objects. All other labels were mapped to label ID 8. When evaluating the segmentation on Cityscapes, we mapped the Cityscapes groundtruth labels and the COCO-trained model predictions to these 8 labels.

1.4. Hardware configuration

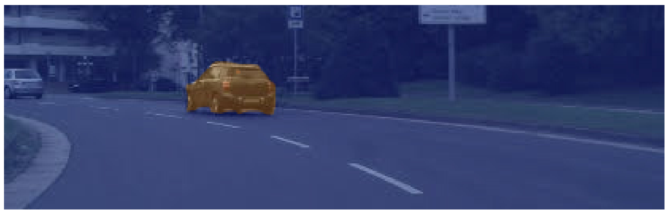
The three models in this experiment had different computational costs. Table 2 shows the duration of a training step for each of the three models on 8 NVIDIA p100 GPU, for a batch of 32. Placing the segmentation model on a TPU node reduced its training step time to be closer to the other modules. This way the convergence was not gated on the Segmentation model. Being able to train each module on a different hardware configuration is one of the strengths of our method.

Model	Hardware	Step time
Depth	GPU	0.81s
Motion	GPU	0.83s
Segmentation	GPU	2.18s
Segmentation	TPU	1.42s

Table 2: Time per training step in milliseconds for each of the three modules in Sec. 3.2.1 in the main paper, on various hardware platforms. The batch size is 32 in all cases. GPU denotes 8 NVIDIA p100 GPU, and TPU denotes a Google Cloud TPU v2-8 unit.

1.5. Failure cases

Consistency improves correctness, but does not guarantee it. A set of predictions can be consistent with one another, but not correct. A simple example is a misdetection of


(a) Failure example 1, frame 1

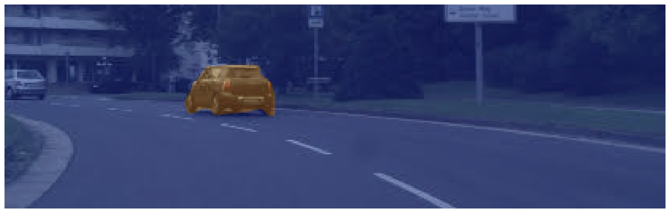
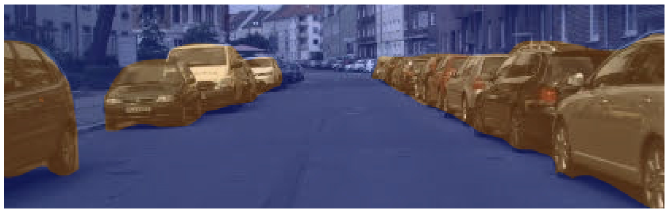

(b) Failure example 1, frame 2

Figure 1: Failure example 1: Segmentation network fails to segment out a white car on the left edge on two consecutive frames.


(a) Failure example 2, frame 1

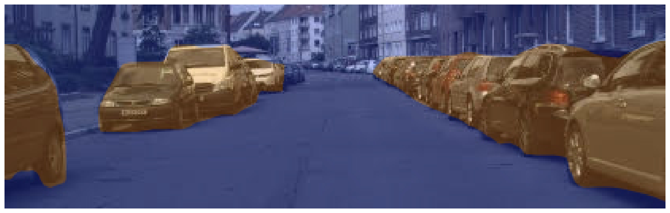

(b) Failure example 2, frame 2

Figure 2: Failure example 2: Segmentation network fails to segment out some cars at the end of the road on two consecutive frames.

a static object. If the segmentation network fails to segment out the same object on two consecutive frames, the consistency loss will not penalize this failure. Fig.1 and Fig.2 shows some examples of failure cases that consistency was unable to fix.

2. Depth and Surface Normals

To compute a consistency loss for the collective training of depth and normal prediction models we compute surface normals from the *predicted depth map*, and penalize their deviation from the *predicted normal map*, closely following the method in Ref. [1]. We first convert the the depth map to a 3D point cloud, using the inverse of the intrinsics matrix:

$$\vec{r}_{ij} \equiv \begin{pmatrix} x_{i,j} \\ y_{i,j} \\ z_{i,j} \end{pmatrix} = z_{i,j} \cdot \begin{pmatrix} 1/f_x & 0 & -x_0/f_x \\ 0 & 1/f_y & -y_0/f_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} j \\ i \\ 1 \end{pmatrix} \tag{4}$$

where f_x and f_y denote the focal length, and x_0 and y_0 the principal point offset, i and j are the pixel coordinates along the height and the width of the image respectively, and z_{ij} is the depth map evaluated at the pixel coordinates (i, j). $\vec{r}_{ij}$ is a point in 3D space, in the camera coordinates, corresponding to pixel (i, j).

We then compute the spatial derivatives of the depth map:

$$\begin{aligned} (\partial x, \vec{r})_{i,j} &= \vec{r}_{i,j+1} - \vec{r}_{i,j-1} \\ (\partial y, \vec{r})_{i,j} &= \vec{r}_{i+1,j} - \vec{r}_{i-1,j} \end{aligned} \tag{5}$$

To exclude depth discontinuities, we invalidate pixels where the spatial gradient of the depth relative to the depth itself is greater than a certain threshold β. To this end, we define a validity mask $v_{i,j}$:

$$V_{i,j} = (V_x)_{i,j} \cdot (V_y)_{i,j}, \tag{6}$$

where

$$\begin{aligned} (V_x)_{i,j} &= \begin{cases} 1 & (\partial_x, z)_{i,j} < z_{i,j} \cdot \beta \\ 0 & \text{otherwise} \end{cases} \\ (V_y)_{i,j} &= \begin{cases} 1 & (\partial_y, z)_{i,j} < z_{i,j} \cdot \beta \\ 0 & \text{otherwise} \end{cases} \end{aligned} \tag{7}$$

The validity mask $V_{i,j}$ is used to zero out the spatial gradients at depth discontinuities:

$$\begin{aligned} (\partial x, \vec{r'})_{i,j} &= (\partial x, \vec{r})_{i,j} \cdot V_{i,j} \\ (\partial y, \vec{r'})_{i,j} &= (\partial y, \vec{r})_{i,j} \cdot V_{i,j} \end{aligned} \tag{8}$$

We then compute the average spatial derivatives over a window of size $N \times N$ pixels around each pixel (i, j):

$$\begin{aligned} \overline{(\partial x, \vec{r})}_{i,j} &= \frac{1}{N^2} \sum_{i',j}^{N} (\partial x, \vec{r'})_{i-\frac{N}{2}+i', j-\frac{N}{2}+j'} \\ \overline{(\partial y, \vec{r})}_{i,j} &= \frac{1}{N^2} \sum_{i',j}^{N} (\partial y, \vec{r'})_{i-\frac{N}{2}+i', j-\frac{N}{2}+j'} \end{aligned} \tag{9}$$

Note that we normalize by N^2, whereas the proper normalization would be by $\sum_{i',j'} V_{i-\frac{N}{2}+i', j-\frac{N}{2}+j'}$. However since the direction of the surface normal is insensitive to the norms of $\overline{(\partial x, \vec{r})}_{i,j}$ and $\overline{(\partial y, \vec{r})}_{i,j}$, as opposed to their directions, this is immaterial.

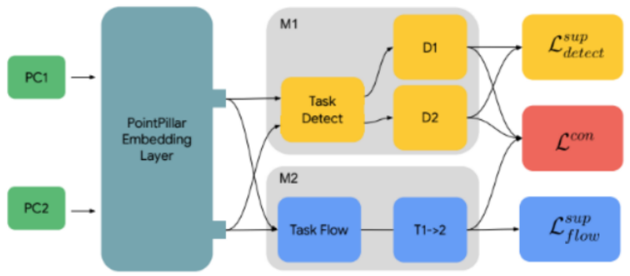

PC1
PC2
PointPillar Embedding Layer
M1
Task Detect
D1
D2
M2
Task Flow
T1->2
$\mathcal{L}^{sup}_{detect}$
$\mathcal{L}^{con}$
$\mathcal{L}^{sup}_{flow}$

Figure 3: 3D Detection and flow estimation joint training: we use parts of Waymo Open Dataset [3] to supervise the training of a single-frame 3D detector (M1) and multi-frame flow estimator (M2) and apply a motion consistency loss on a set of unlabeled data.

To obtain the surface normal, we calculate the cross product of $\overline{(\partial x, \vec{r})}_{i,j}$ and $\overline{(\partial y, \vec{r})}_{i,j}$,

$$(\vec{n_d})_{i,j} = \overline{(\partial x, \vec{r})}_{i,j} \times \overline{(\partial y, \vec{r})}_{i,j} \tag{10}$$

and then normalize it, to obtain the normalized surface normal:

$$(\hat{n_d})_{i,j} = (\vec{n_d})_{i,j} / \left\| (\vec{n_d})_{i,j} \right\| \tag{11}$$

The consistency is then computed as

$$\mathcal{L}_{consistency} = \text{cosine\_distance}(\hat{n_d}, \hat{n_p}), \tag{12}$$

where $\hat{n_d}$ is the computed surface normals from the inferred depth as described in this section, and $\hat{n_p}$ is the normal map predicted from the normal prediction network.

3. 3D Object Detection in Point Clouds in Time

To add more detail, we visualize the joint training of 3D Object detection and flow in Figure 3. Two consecutive Point Clouds are considered PC1 and PC2, which are processed by a Point Pillar embedding layer [2]. The first module (M1) provides the 3D detections D1 and D2 for the two point clouds, respectively, whereas the second module (M2) computes the flow $T1 \to 2$, which is limited to boxes only. Each of these modules applies its respective supervision, whereas the consistency loss imposes that the detected boxes from the static point cloud should match the detections from the previous static point cloud as transformed by the predicted flow (Figure 3).

References

[1] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2684–2689, 2012. 3

[2] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, pages 12697–12705, 2019. 3

[3] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020. 3