Binary TTC: A Temporal Geofence for Autonomous Navigation

Abhishek Badki[1,2] Orazio Gallo[1] Jan Kautz[1] Pradeep Sen[2]
[1]NVIDIA [2]UC Santa Barbara

Abstract

Time-to-contact (TTC), the time for an object to collide with the observer's plane, is a powerful tool for path planning: it is potentially more informative than the depth, velocity, and acceleration of objects in the scene—even for humans. TTC presents several advantages, including requiring only a monocular, uncalibrated camera. However, regressing TTC for each pixel is not straightforward, and most existing methods make over-simplifying assumptions about the scene. We address this challenge by estimating TTC via a series of simpler, binary classifications. We predict with low latency *whether the observer will collide with an obstacle* within a certain time*, which is often more critical than knowing exact, per-pixel TTC. For such scenarios, our method offers a temporal geofence in* 6.4 *ms—over* 25× *faster than existing methods. Our approach can also estimate per-pixel TTC with arbitrarily fine quantization (including continuous values), when the computational budget allows for it. To the best of our knowledge, our method is the first to offer TTC information (binary or coarsely quantized) at sufficiently high frame-rates for practical use.*

1. Introduction

Path planning, whether for robotics or automotive applications, requires accurate perception, which in turn, benefits from depth information. Many modalities exist to infer depth. Strategies such as lidar estimates depth accurately but only at sparse locations, in addition to being expensive. Depth can also be estimated with strategies such as stereo [34], but these introduce issues such as calibration drift over time.

An alternative is to use a monocular camera—an attractive, low-cost solution, with light maintenance requirements. The motion of the camera induces optical flow between consecutive frames, which carries information on the scene's depth. Depth, however, can only be estimated in the constrained case of static scenes. For dynamic scenes, the 2D flow of a pixel is a function of its depth, its velocity, and the velocity of the camera. Disentangling these three components is an under-constrained and challenging problem.

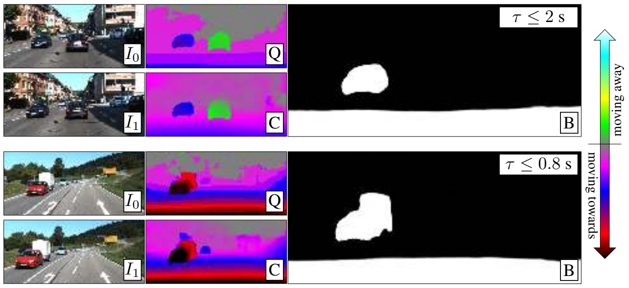


Figure 1: Given I_0 and I_1, our binary time-to-contact (TTC) estimation acts as a temporal geofence detecting objects that will collide with the camera plane within a given time, $\boxed{\text{B}}$. It only takes 6.4 ms to compute. Our method can also output quantized TTC, $\boxed{\text{Q}}$, and continuous TTC, $\boxed{\text{C}}$.

Previous approaches either ignore dynamic regions [35], or use strong scene priors [26, 44, 46, 43, 25] to hallucinate their depth. *Do we really need to disentangle them?* The role of perception is to inform decisions. An object moving towards the camera is more critical than another that is potentially closer, but moving away from the camera. Differently put, predicting the time at which an object would make contact with the camera may be more valuable than knowing its actual depth, velocity, or acceleration [22].

In fact, time-to-contact (TTC), the time for an object to collide with the camera plane under the current velocity conditions, is a traditional concept in psychophysics [38, 14] as well as computer vision [37, 5]. TTC can be estimated from the ratio of an object's depth and its velocity relative to the camera, even when the problem of regressing either one independently is ill-posed. However, TTC estimation has its own challenges, forcing most of the existing approaches to severely constrain their scope. For instance, they assume that the scene is static, or that a mask for dynamic objects is provided [16, 32]. The recent approach by Yang and Ramanan tackles some of these challenges by learning a mapping between optical flow and TTC directly, thus producing a per-pixel TTC estimate [42]. However, it relies on accurate optical flow estimation and inherits its limitations, including its heavy computational load.

Unlike most existing approaches, we side-step the need to explicitly compute the optical flow. Our learning-based approach estimates per-pixel TTC directly from images. We leverage the relationship between an object's TTC and the ratio of the size of its image in different frames [5, 15].

This work was done while A. Badki was interning at NVIDIA.
Project page: https://github.com/NVlabs/BiTTC

However, because regressing this scale factor exactly is challenging, we focus on whether the size of the object's image is increasing, indicating a collision at some time in the future, or decreasing, indicating that the object is moving away.

More concretely, inspired by Badki *et al*. [1], we perform a series of binary classifications with respect to different scale factors, each corresponding to a specific TTC. Each classification yields a binary TTC map with respect to the desired time threshold, Figure 1, insets $\boxed{\text{B}}$. Our binary map, efficient to compute, acts as a temporal geofence in front of the camera: it identifies objects within a given TTC, in 6.4 ms.[1] This is useful when a quick reaction time is important. We can also estimate per-pixel TTC with arbitrary quantization, as shown in Figure 1, insets $\boxed{\text{Q}}$, or continuous, Figure 1, insets $\boxed{\text{C}}$. These different levels of quantization, from binary to continuous, can be predicted with the same core network. In fact, quantization levels can be added, removed, or moved dynamically at inference time, based on the current needs of the autonomous agent. Given the scarcity of TTC ground truth data, to impose additional inductive bias to our network we also introduce binary optical flow estimation as an auxiliary task. We achieve competitive performance for TTC estimation against existing methods, even stereo-based methods, but we are from 25× to several orders of magnitude faster.

2. Related Work

While valuable for navigation, 3D information about the scene is challenging to gather with a monocular camera. Existing methods assume the scene to be static [9, 39, 17, 24], or estimate single-image *relative* depth, rather than *metric* depth [8, 11, 13, 10, 33]. Single-image relative depth can be "upgraded" to metric by computing the optical flow between multiple images [26, 44, 46, 43, 25], but this requires strong priors, and it is brittle for complex, large motions. Nevertheless, with depth maps and optical flow we can estimate scene flow, which captures both depth and velocity [36]. A recent approach by Hur and Roth elegantly combines these principles by learning scene priors to decompose the depth and the velocity directly from the estimated optical flow information [18].

Instead of regressing depth and velocity, we focus on estimating their ratio directly from images, which yields time-to-contact (TTC). We do this via multiple binary classifications. Here we discuss the state-of-the-art in terms of these two axes—TTC and regression via classification.

2.1. Time-to-Contact

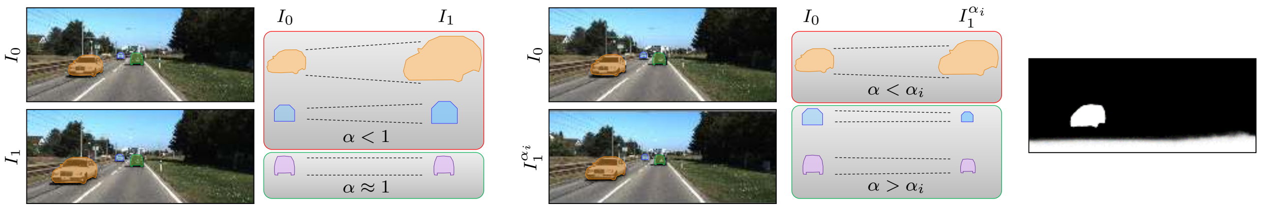
Time-to-contact (TTC) was studied in psychophysics and psychology, even before it attracted the attention of the computer vision community. Early work by Lee, for instance, suggested that TTC is sufficient for making decisions about braking, and is likely to be picked up by the driver faster than distance, speed, or acceleration of objects in the scene [22]. From a computational standpoint, TTC is appealing because it only depends on the ratio of depth and velocity, which can be estimated directly from images, even when estimating either one is an ill-posed problem [16]. Several traditional approaches have been proposed to estimate TTC from the estimated optical flow [31, 32, 3]. Horn *et al*. proposed a direct method for TTC estimation that only uses the constant brightness assumptions [15, 16]. While these approaches estimate the TTC for an object that is moving relative to the camera, they require masks for dynamic objects, which limits their practical impact.

We propose a learning-based approach for TTC estimation that handles multiple dynamic objects—without needing any segmentation—and estimates per-pixel TTC. The elegant, closely related work by Yang and Ramanan estimates optical flow, uses it to compute the scaling factor of objects, and maps it to TTC [42]. Xu *et al*. also estimate the scaling of objects, but do so by modeling the change of objects' size explicitly for optical flow estimation [41]. However, computing the full optical flow is time-consuming, and estimating TTC from optical flow inherits its limitations. Instead, following Horn *et al*. [16], we compute TTC directly from the input images, side-stepping optical flow computation altogether. Moreover, inspired by Badki *et al*. [1], we solve TTC estimation via a series of binary classifications. Each binary classification can be computed independently of the others at over 150 fps. This can be thought of as a temporal geofence, detecting pixels or objects within a given TTC. For existing methods, including the method by Yang and Ramanan [42], this can only be achieved by computing the TTC for all the pixels or objects in the scene, and then thresholding it.

2.2. 3D Inference as a Classification Problem

The idea of posing 3D regression as a classification task has a rich history. Several learning-based approaches estimate depth via a multi-class classification task [21, 4, 45, 20]. These are more accurate than other learning-based approaches that pose the problem as a regression task [29, 7]. Badki *et al*. introduced a method that allows us to control the trade-off between latency and accuracy [1]. Instead of posing depth as a multi-class classification problem, they solve it via multiple binary classifications. Each classification provides useful information about the scene at high frame-rates. Our approach is based on the same intuition. We are the first learning-based approach to estimate per-pixel TTC directly from the input images, and to pose it as a (binary) classification problem.

[1] On an NVIDIA Tesla V100 for 384×1152 images.



(a) Inputs (b) Scaled inputs (c) Temporal geofence

Figure 2: **Intuition.** Given two images of a dynamic scene, I_0 and I_1, we define a temporal geofence to detect objects expected to cross the camera plane before a given time τ_i from the time of capture of I_0. Compare I_0 and I_1, (a). The images of the orange and blue cars are smaller in I_0 ($\alpha < 1$), while the image of the purple car is roughly unchanged. This allows us to predict that only the first two cars will collide with the camera plane. We propose to perform this comparison after scaling the source image by a factor α_i (corresponding to TTC value τ_i). The orange car is still larger in $I_1^{\alpha_i}$ ($\alpha < \alpha_i$), indicating that it will cross the camera plane *before* the specified τ_i. On the other hand, the blue and purple cars will not. Rather than regressing the exact scale factor, we classify a pixel as making contact before or after τ_i by classifying if the objects scale up or down in $I_1^{\alpha_i}$ with respect to I_0. This yields a binary TTC probability map for τ_i, (c).

3. Method

Despite some time-to-contact (TTC) estimation methods dating back to the 1990s [37, 5], TTC never rose to the popularity of other techniques that are now mainstream, such as optical flow or stereo estimation. This is due in part to its intrinsic limitations and to the challenges it poses, which we discuss below. Note that in the rest of the paper we often attribute the properties of objects to the corresponding pixels, to simplify the discussion. For instance, we talk of "a pixel's velocity" to indicate the projection on the image plane of the velocity of the object imaged by that pixel.

3.1. A Review on Time-to-Contact

Consider two frames of a static scene captured by a moving camera. Pixel-level correspondences allow us to compute depth, if the camera information is known. In the more realistic case of dynamic scenes, however, the problem becomes ill-posed: the displacement of a pixel is the result of its depth, its velocity, and the camera velocity, all of which cannot be disambiguated without strong priors. In this case, the concept of time-to-contact (TTC) comes to the rescue. Given an object $\mathcal{O}$, the TTC τ, *i.e.*, the time at which object $\mathcal{O}$ will (or did) cross the camera plane, can be written as

$$\tau = -Z_{\mathcal{O}} \Big/ \frac{dZ_{\mathcal{O}}}{dt} = -Z_{\mathcal{O}}/\dot{Z}_{\mathcal{O}}, \tag{1}$$

where the origin is at the camera, and we assume that the current velocity conditions will continue. $Z_{\mathcal{O}}$ is the depth of the object from the camera plane, and $\dot{Z}_{\mathcal{O}}$ its relative velocity. Equation 1 shows the first appealing feature of TTC: even if the depth and the velocity of the object cannot be estimated independently, τ can be computed from their ratio.

However, we are interested in computing the TTC from pixel displacements alone. To do that we need an additional piece of information: the location of the focus-of-expansion (FOE). Given two frames of a static scene captured by translating the camera, all the pixels in the image move along lines originating from the FOE, the image of the point towards which the camera is moving. Under the same assumptions, the FOE coincides with the epipole. The relationship between the TTC, τ, and the velocity of the pixel is given by

$$\dot{x} = \frac{x - x_0}{\tau} \quad \text{and} \quad \dot{y} = \frac{y - y_0}{\tau}, \tag{2}$$

where (x_0, y_0) is the FOE. Equation 2 can be easily derived by differentiating the projection of a 3D point onto the image plane with respect to time [15]. Note that the velocity $(\dot{x}, \dot{y})$ can be computed from the optical flow (u, v), which allows us to write

$$\tau = \frac{x - x_0}{u} \cdot T \quad \text{and} \quad \tau = \frac{y - y_0}{v} \cdot T, \tag{3}$$

where T is the time elapsed between the two frames.

Equation 3 shows a second compelling reason to use TTC: its estimation does not require the camera calibration.[2] There are also challenges, however. If a rigid object is dynamic and translates with respect to the scene, its pixels move along lines centered around a different FOE. To estimate the TTC using Equation 3, then, we need to localize the FOE of *each* dynamic object. Moreover, if an object is deformable, the FOE varies with each pixel, and for general motion, we need to further compensate for rotations. This is why traditional approaches had to rely on oversimplifying assumptions.

However, we can compute the size of the image of a fronto-parallel, planar non-deformable object of size $S_{\mathcal{O}}$ at distance $Z_{\mathcal{O}}$ as [15]

$$s_{\mathcal{O}} = f S_{\mathcal{O}}/Z_{\mathcal{O}}, \tag{4}$$

where f is the focal length of the camera. Since f and $S_{\mathcal{O}}$ are constant, differentiating Equation 4 yields

$$Z_{\mathcal{O}}/\dot{Z}_{\mathcal{O}} = -s_{\mathcal{O}}/\dot{s}_{\mathcal{O}}, \tag{5}$$

[2] We assume a pin-hole camera model.

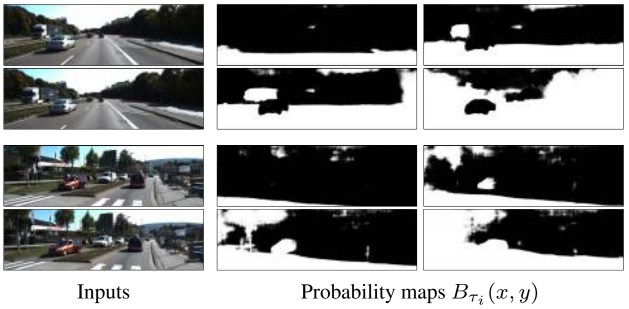


Figure 3: **Binary TTC maps.** Our method can directly identify the pixels that will contact the camera plane before a given time. From top to bottom, we show results for four TTC values for a case of highway driving and for a camera that rotates.

which, plugged into Equation 1, allows us to estimate the TTC using only information about the size of an object's image and its rate of change:

$$\tau = s_{\mathcal{O}} / \dot{s}_{\mathcal{O}}. \tag{6}$$

Note that both Equation 3 and 6 effectively look at scaling. However, the latter is independent of the point with respect to which the scaling is performed, whereas for the former, that point is the FOE. Of course, under more realistic conditions (*e.g.*, non-planar, fronto-parallel objects), Equation 6 becomes an approximation, which requires proper handling, as we show later.

3.2. TTC via Multiple Binary Classifications

In this section, we first provide the intuition motivating our method. We then describe binary TTC, the core of our method, which detects pixels predicted to collide with the camera plane within a given time. We further show how an arbitrary number of binary classifications can be combined to estimate, for each pixel, a quantized version of the TTC. To simplify the description, here we assume positive TTCs, *i.e.*, we focus on objects moving towards the camera rather than away from it. In Section 4, we describe a small adaptation of our method that allows us to seamlessly remove this distinction.

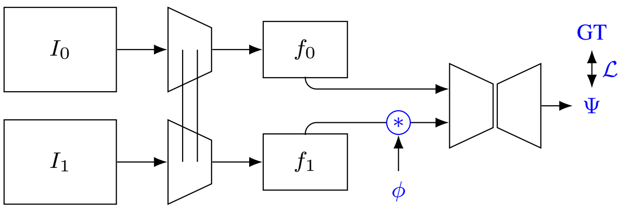
3.2.1 Intuition

Given two images captured at times t_0 and t_1, Equation 1 can be approximated as

$$\tau = -Z(t_0) \Big/ \left(\frac{Z(t_1) - Z(t_0)}{t_1 - t_0} \right) = \frac{t_1 - t_0}{1 - \frac{Z(t_1)}{Z(t_0)}}. \tag{7}$$

If we assume fronto-parallel and planar objects that do not rotate, we can combine Equations 7 and 4, thus expressing the TTC as a function of observations in image space:

$$\tau = \frac{t_1 - t_0}{1 - \frac{s(t_0)}{s(t_1)}} = \frac{t_1 - t_0}{1 - \alpha}, \tag{8}$$



Task		$\circledast$	ϕ	Ψ	$\mathcal{L}$	GT
TTC	binary	scaling	α_i	B_{τ_i}	BCE	$\mathbb{1}_{\tau_* > \tau_i}$
	quantized	scaling	$\{\alpha_i\}$	$\{B_{\tau_i}\}$	BCE	$\{\mathbb{1}_{\tau_* > \tau_i}\}$
	continuous	scaling	$\{\alpha_i\}$	$\{B_{\tau_i}\}, \tau$	BCE, SL1	$\{\mathbb{1}_{\tau_* > \tau_i}\}, \tau_*$
Optical flow	horizontal	shifting	u_i	B_{u_i}	BCE	$\mathbb{1}_{u_* > u_i}$
	vertical	shifting	v_i	B_{v_i}	BCE	$\mathbb{1}_{v_* > v_i}$

Figure 4: **Architecture.** We perform all of our tasks with minor modifications to the same backbone. We preprocess the input images by extracting features and applying a task dependent operation, $\circledast$, to the features of the second image. The input parameter ϕ, the loss, $\mathcal{L}$, the ground truth, GT, and the output, Ψ for each task are listed in the table. Note that $\{\cdot\}$ indicates a set, and $\mathbb{1}$ the indicator function.

where s is the size of the image of an object or region in the scene. In other words, with Equation 8, the TTC can be computed from the scale factor α. This simplifies our task, compared with having to estimate per-pixel FOEs and optical flow. However, regressing α for each pixel explicitly, which requires defining the size of objects or image regions and tracking its change over time, is not straightforward. Instead, we consider a pair of images, I_0 and I_1, and compute $I_1^{\alpha_i}$, a version of I_1 scaled by a factor α_i. This scaling factor α_i corresponds to a unique TTC τ_i. The regions whose size matches between I_0 and $I_1^{\alpha_i}$ will collide with the camera plane exactly at τ_i. Furthermore, we can expect regions that are larger (or smaller) in $I_1^{\alpha_i}$ to collide with the camera before (or after) τ_i, see Figure 2.

Instead of regressing the scale factor α directly, then, we propose to train a neural network to take such pairs of images and classify regions in $I_1^{\alpha_i}$ as being larger or smaller than the corresponding regions in I_0, where the correspondence is learned implicitly. Our approach is inspired by the work of Badki *et al.* [1] for stereo. They also learn to classify the disparity of a pixel as being larger or smaller than a given disparity, instead of regressing the disparity directly.

3.2.2 Binary TTC

The inputs to our method are two images and scale factor α_i corresponding to the TTC we want to analyze. We first extract features f_0 and f_1 from both images, and scale f_1 by α_i, to obtain $f_1^{\alpha_i}$. Rather than classifying the objects as scaling up or down between f_0 and $f_1^{\alpha_i}$, we train a lightweight network to directly classify whether each pixel's TTC is larger or smaller than τ_i, using a binary cross-entropy loss. That is, the network predicts a probability map

$$B_{\tau_i}(x, y) = p(\tau(x, y) > \tau_i; f_0, f_1^{\alpha_i}), \tag{9}$$

which can be binarized by simple thresholding. We train directly to predict binary TTC instead of explicitly detecting if the size of the image of objects is getting larger or smaller with respect to α_i for two reasons. First, ground truth data for the scale factor α is challenging to even define beyond a small neighborhood of pixels, unless objects move rigidly and only translate. Moreover, as discussed before, Equation 8 is an approximation in the common case of objects that are not planar, and for non-rectilinear motions. Equation 7, which establishes the relationship between the change in depth and the TTC, allows us to generate the ground truth data from existing datasets, as we explain in Section 4.1. A network trained to classify the TTC directly can learn the necessary priors to compensate for the approximations introduced by Equation 8. Our core architecture is shown in Figure 4. This very architecture is also used for all the tasks we describe below, with the caveat that the terms in blue differ for each task.

3.2.3 Quantized TTC

Our core approach naturally extends to estimating a coarsely quantized TTC for each pixel, which may be more useful than binary in certain scenarios. We note that Equation 9 is a complementary cumulative distribution function. Therefore, for two time-to-contact values, $\tau_j > \tau_i$, we can compute

$$p(\tau_i < \tau(x,y) \leq \tau_j) = B_{\tau_i}(x,y) - B_{\tau_j}(x,y). \qquad (10)$$

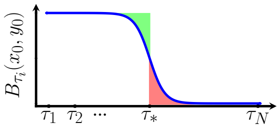
Consider a set of TTC values $\{\tau_i\}_{i=1:N}$, and assume they are in increasing order. After computing Equation 9 for each of the N TTC values, we can estimate the quantization bin in which the TTC of a pixel falls as

$$Q(x,y) = \arg\max_i \Big(p\big(\tau_i < \tau(x,y) \leq \tau_{i+1}\big)\Big). \qquad (11)$$

The different TTC values can be spaced non-uniformly.

3.2.4 Continuous and Selective TTC

While our method is specifically designed for binary and quantized TTC estimation, it can also estimate per-pixel, continuous TTC. In principle, we can approximate continuous values by predicting quantized TCC (Section 3.2.3) with a larger set of TTC values $\{\tau_i\}_{i=1:N}$. This, however, fails to exploit the relationship between the binary classifications for different τ_i's, for a pixel. We slightly modify the approach while still preserving its binary classification core, as shown in Figure 4. Specifically, instead of taking consecutive pairs of probability maps and applying Equation 10, we stack them.



For a specific pixel (x_0, y_0) we generally see a progression as in the plot on the right. That is, $B_{\tau_i}(x_0, y_0)$ (the probability that the object will collide *after* τ_i) is consistently high for $\tau_i \ll \tau_*$, and low for $\tau_i \gg \tau_*$, where τ_* is the correct TTC value. In the transition region, the network is uncertain, which is why aggregating information across multiple τ_i is beneficial. Badki *et al.* [1], who obtain a similar curve for disparity, propose to estimate the transition point, τ_* in our case, by computing the area under the curve (AUC),

$$\text{AUC}(x,y) = \sum_i (\tau_{i+1} - \tau_i) \cdot B_{\tau_i}(x,y). \qquad (12)$$

To understand why Equation 12 yields the desired result, consider the case of a step function, *i.e.*, τ_* aligns with a quantization boundary: AUC$= \tau_* \cdot 1 = \tau_*$. This relationship holds for the more common case of a smooth transition region: because the transition is generally symmetric around τ_*, the red and green areas in the plot have similar extent and compensate for each other. Because the AUC is differentiable, we can use it to fine-tune our network so that combining a set of probability values using the AUC operation yields continuous TTC values.

3.2.5 A Note on Inference-Time Tradeoff

Binary, quantized, and continuous TTC estimation yield increasingly rich information for navigating an environment. We note that, when estimating quantized and continuous TTC, the multiple binary classifications can be run in parallel, as they are independent of each other. Therefore we can compute quantized and continuous TTC at roughly the same frame-rate as for binary quantization—just above 150 fps. If multiple binary classifications cannot be run in parallel due to hardware limitations, the cost for computing quantized and continuous TTC grows linearly with the number of levels. In such cases, one can decide the number of quantization levels dynamically to best leverage the trade-off between accuracy and latency. For situations where a fast response time is critical, such as highway driving, binary TTC may be sufficient. For slower navigation (*e.g.*, for robotics or for parking lot driving), our method can be adapted to trade latency for a finer quantization *at inference time*.

3.3. Dealing with the Lack of Training Data

As for any learning-based method, having access to a large amount of data is critical to properly train our network. Unfortunately, there are no datasets with TTC ground truth data and few scene flow datasets that we can use to infer it (Section 4.1). To increase the training data, we leverage a closely related task: binary optical flow estimation. We use the same binary approach, but we shift the features (horizontally and vertically) rather than scaling them. We then

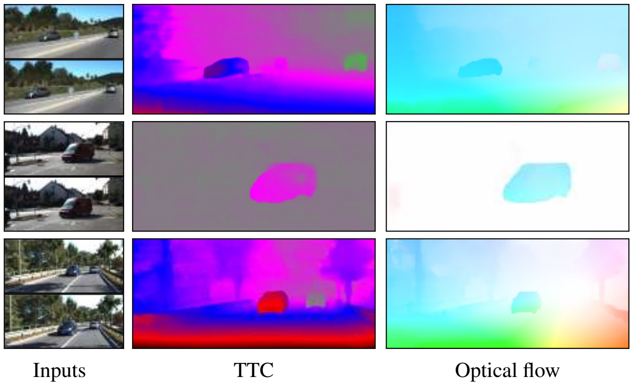


Figure 5: **Optical flow as an auxiliary task.** We estimate optical flow as an auxiliary task to improve our TTC estimation. Like for continuous TTC, we estimate optical flow via a series of binary classifications. While optical flow is not a goal for us, this figure shows that our prediction is reasonable.

classify the direction of the shift (left/right or up/down). For horizontal shifts, we seek to predict a probability map relative to a given shift u_i

$$B_{u_i}(x,y) = p(u(x,y) > u_i; f_0, f_1^{u_i}), \tag{13}$$

where $f_1^{u_i}$ are the features extracted from the second image and shifted by u_i. The equation for the vertical shift v_i is analogous. Using optical flow to pre-train our network and continuing using it as an auxiliary task yields a stronger inductive bias, as we discuss in Section 5. Although optical flow is an auxiliary task, we show results in Figure 5 to allow for a visual evaluation.

4. Implementation Details

4.1. TTC Data Generation

A dataset providing per-pixel TTC ground truth does not exist. However, we can use ratio of the depth in different frames to compute the TTC with Equation 7. This ratio can be computed from datasets offering per-pixel scene flow $[X(t_0), Y(t_0), Z(t_0)] \rightarrow [X(t_1), Y(t_1), Z(t_1)]$. To train for TTC estimation we use the Driving dataset from the SceneFlowDatasets [28]. We also use the KITTI15 [29] dataset to train for TTC estimation. We split the training dataset of KITTI15 into train and validation split, and show analysis on the validation part of the dataset. To pretrain our network for the task of binary optical flow, we use the FlyingChairs2 [7, 19] and the FlyingThings3D [28] datasets. We also use optical flow data available from Driving [28] and KITTI15 [29], and train our network for both binary optical flow and TTC estimation.

4.2. Working in the Inverse TTC Domain

We are interested in objects predicted to collide within τ_i seconds. However, $\tau \in (-\infty, \infty)$, with positive and negative values indicating objects moving towards and away from the camera, respectively. That is, some of the pixels whose TTC is smaller than τ_i will never collide with the camera plane. In practice, then, we seek to identify the pixels such that

$$(\tau(x,y) \leq \tau_i) \;\; \cap \;\; (\tau(x,y) > 0), \tag{14}$$

which introduces an additional complication. Moreover, the effect of TTC in the image space in not linear. A simple solution is to work with the ratio-of-depths, the inverse of the TTC domain:

$$\eta = \frac{Z(t_1)}{Z(t_0)} = 1 - \frac{t_1 - t_0}{\tau}. \tag{15}$$

Yang and Ramanan termed it motion-in-depth [42]. Thanks to Equation 15, the effect of TTC is linear in image space and Equation 14 simply reduces to

$$\eta(x,y) \leq \eta_i. \tag{16}$$

For binary TTC, then, we simply scale the features of the source image by a factor $\alpha_i = \eta_i$.

Similarly, for continuous estimation, we sample planes uniformly in the inverse TTC domain. We apply uniformly spaced scale factors to the features of the source image: $\{\alpha_i\}_{i=1:N} = \{\alpha_0 + i\Delta\alpha\}_{i=1:N}$. The AUC of the resulting segmentation gives us the map $\eta(x,y)$, which we then covert to a TTC map.

4.3. Architecture

Figure 4 shows our architecture. The feature extraction module uses spatial pyramid pooling layers as PSMNet [4]. The resulting 32-channel feature maps are at one-third the input image resolution. We then apply a task-dependent operator, $\circledast$, parametrized by ϕ, to f_1 and generate f_1^{ϕ}. For instance, when estimating the horizontal component of binary optical flow, $\circledast$ shifts the features by a horizontal shift u_i, and for binary TTC, it scales them by α_i. The full list for each task is in the table in Figure 4. To avoid cropping out features, we zero-pad f_0 and f_1^{ϕ} to $1.5\times$ the feature map resolution. The concatenation of f_0 and f_1^{ϕ} is fed to a 2D encoder-decoder network with skip connections. This module has three heads—one for binary TTC, one for binary horizontal optical flow, and one binary vertical optical flow. We apply a sigmoid to each output to obtain the respective probability maps. Intuitively, this network tells us if the features in f_1^{ϕ} are scaling up/down, shifting right/left, or shifting up/down with respect to the corresponding features in f_0.

4.4. Training

We pre-train our network for the binary optical flow task on the FlyingChairs2 dataset and train it further on the FlyingThings3D dataset [28] using a binary cross-entropy

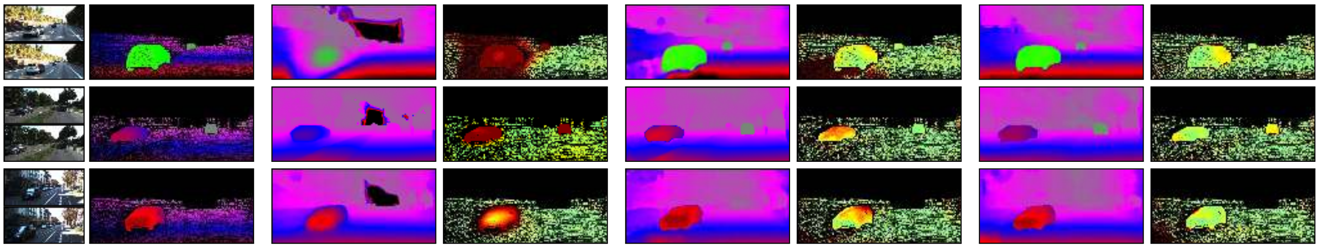


Figure 6: **Comparison on continuous TTC estimation.** We show the predicted TTC and the error map (red and green indicate high and low error, respectively) for all methods. Our method and Yang and Ramanan's method explicitly train for TTC estimation and provide better results than the monocular scene flow method of Hur and Roth. Note the lower error in the TTC estimation for our method.

(BCE) loss with respect to a thresholded version of the ground truth. The TTC head is left unsupervised in this stage. We then fine-tune our network for estimating binary TTC first on the Driving [28] and then on the KITTI15 [29] datasets. Since both datasets also offer optical flow data, in this second stage we train for both binary optical flow and binary TCC. In Section 5 we discuss the impact of this choice on the quality of the results. We use relative weights of 0.8 and 0.2 for binary TTC and binary optical flow tasks respectively. For training on the KITTI15 dataset, we use the same split as Yang and Ramanan [42].

For the continuous version, we pre-train the network as before, and fine-tune it on FlyingThings3D for continuous optical flow. For each training image pair we uniformly sample horizontal and vertical shifts and stack the maps corresponding to each shift. We use the resulting volumes to compute continuous optical flow via the AUC operation described in Section 3.2.4. We can then continue training the network using a BCE loss on the individual probability maps, and a Smooth-L1 (SL1) [12] regression loss on the output of the AUC module. We use relative weights of 0.1 and 0.9 respectively. To fine-tune our network for continuous TTC estimation, we follow a similar strategy as for the binary segmentation training. We continue training the network for the task of continuous optical flow and continuous TTC estimation on the Driving and the KITTI15 datasets. However, this time we form three volumes, two corresponding to optical flow and one to TTC. As discussed in Section 4.2, we work in the inverse TTC domain and uniformly sample scale factors. The network trained on the entire KITTI15 dataset is used for scene flow estimation on KITTI15 benchmark images, as explained in Section 5. We refer the reader to our Supplementary for the additional training details.

5. Evaluation and Results

In this section we discuss qualitative and quantitative results for our method. First, to validate the importance of the training strategy described in Section 3.3, we compare three training strategies: (1) we train our network only to estimate binary TTC, (2) we pre-train it to estimate binary optical flow (OF) and then binary TTC, and (3) we pre-train with binary OF, and then continue training with both binary OF and binary TTC (our method). As shown in Table 1, our final method achieves a percentage error of 1.013 and a mean intersection-over-union (mIOU) of 0.9525. If we only train for TTC estimation after pre-training for OF, we observe an increase in percentage error to 1.174 and a drop in mIOU to 0.9453. Removing the binary OF estimation altogether, leads to an additional increase in percentage error to 2.670 and an additional drop in mIOU to 0.8808.

We also thoroughly validate our approach numerically on the KITTI15 validation set. We compare to Yang and Ramanan [42], who proposed the only existing approach that computes per-pixel TTC from a monocular camera, under practical assumptions. They too look at how the size of objects changes, but they estimate it explicitly (and locally) from the optical flow between frames. We also measure against PRSM [40] and OSF [29], both of which perform 3D scene flow estimation using stereo cameras. They represent images as a collection of planar super-pixels and jointly solve for geometry and 3D motion, which informs about the change of depth over time (Section 4.1). This can be used to estimate motion-in-depth for each pixel, $\eta(x, y)$, using Equation 15, which can then be thresholded. Our final comparison is against Hur and Roth [18], a state-of-the-art monocular scene flow estimation approach. We compare against their best model, which uses a combination of self-supervised learning on the KITTI15 raw dataset and supervised learning on the entire KITTI15 training dataset. Note that this approach is already fine-tuned on our validation set.

Table 1 shows that our continuous TTC estimation, yields the lowest motion-in-depth error:

$$\text{MiD} = ||\log(\eta) - \log(\eta_{GT})||_1 \cdot 10^4. \tag{17}$$

However, our fundamental innovation is the ability to define a temporal geofence, *i.e.*, detecting pixels with a TTC smaller than a given τ_i *without the need to estimate the full TTC first*. On this task we perform on par with Yang and Ramanan, but our binary TTC is around 26× faster. OSF performs better than our approach in terms of binary TTC, though, again, it uses a richer input. Moreover, OSF and PRSM, implemented on a CPU, take 390 s and 300 s respectively. The approach by Hur and Roth struggles despite fine-tuning the model on the validation set. However, unlike ours and Yang and Ramanan's approaches, they do not use a synthetic dataset to pre-train, nor train for a better-posed TTC estimation task. Instead, they focus on the more difficult task of monocular scene flow estimation.

We further evaluate our method on the KITTI15 benchmark [30] with the same strategy as Yang and Ramanan [42]: we use our motion-in-depth estimation to compute scene flow, the 3D motion $[X(t_0), Y(t_0), Z(t_0)] \rightarrow [X(t_1), Y(t_1), Z(t_1)]$ corresponding to each pixel. The first two scene flow components can be estimated by back-projecting the optical flow for each pixel. For the Z component, traditional approaches use stereo information. We can estimate the third component from the motion-in-depth ratio if we are given the depth in one frame. After computing the depth at t_0 with GANet [45] (evaluated by D1 in Table 2) and combining it with our TTC estimate, we can compute the depth for each pixel at time t_1. Therefore, D2 in Table 2 effectively measures the quality of our results. The other numbers in the table evaluate other components and are reported for completeness. While our method is not designed to estimate scene flow, it performs on par, or slightly better than methods specifically optimized for it.

In certain scenarios, binary TTC can be more informative than depth. For instance, the top row of Figure 1 shows a car driving away from the camera, and one that is farther, but driving towards the camera. The latter, detected by our binary TTC, is potentially more impactful—in the true sense of the word!—for the ego vehicle path planning. The last row of Figure 3 shows that, as discussed in Section 3.2.2, our method can compensate for camera rotation, even if that breaks some of our assumptions. This is also visible for continuous TTC in the first row of Figure 5.

We show quantized TTC estimation results in Figure 1 $\boxed{Q}$. Note that even just 9 TTC quantization levels (8 binary classifications) provide a meaningful representation of the scene. Moreover, the underlying binary classifications can be run in parallel as they are independent of each other. Therefore, quantized TTC can be run at roughly the same frame-rate as the binary TTC.

We show a visual comparison with Hur and Roth [18], and Yang and Ramanan [42] on continuous TTC for KITTI15 images in Figure 6. Note the lower for our approach. In Figure 7 we show qualitative result of our approach on the Citiscapes dataset [6]. We show two failure cases on this dataset, one due to the sudden vertical motion caused by a road bump and another due to an object rotating significantly. Broadly, our method struggles for motions under-represented in the training dataset.

		Binary (200 ms – 2 s)		Continuous
		mIOU (↑)	% error (↓)	MiD (↓)
Stereo	PRSM [40]	0.9365	1.339	124.0
	OSF [29]	**0.9556**	**0.941**	115.0
Mono	Hur & Roth [18]	0.9418	1.233	115.13
	Yang & Ramanan [42]	0.9525	1.012	75.00
	Ours	0.9525	1.013	**73.55**

Table 1: Comparison on the validation set of KITTI15 for both binary TTC (averaged over a set $\{\alpha_i\}$ uniformly sampled in the interval $\tau \in [0.02\text{s}, 2\text{s}]$) and continuous TTC. Note that PRSM and OSF both use richer input data (stereo vs mono). MiD, motion-in-depth, directly evaluates TTC.

	D1-all	D2-bg	D2-fg	D2-all	Fl-all	SF-all
UberATG-DRISF [27]	2.55	2.90	9.73	4.04	4.73	6.31
ACOSF [23]	3.58	3.82	12.74	5.31	5.79	7.90
ISF [2]	4.46	4.88	11.34	5.95	6.22	8.08
Yang&Ramanan [42]	1.81	3.39	8.54	4.25	6.30	8.12
Ours	**1.81**	**3.84**	**9.39**	**4.76**	**6.31**	**8.50**

Table 2: Top 5 published methods on the KITTI scene flow benchmark. Our method performs reasonably well, despite not being designed for scene flow, see text.

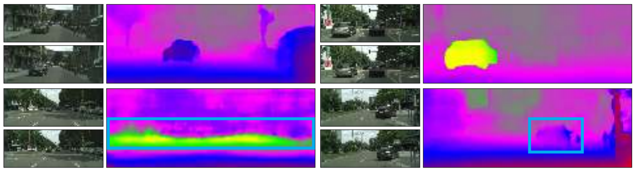


Figure 7: **Unseen dataset.** Our predicted TTC map on Citiscapes [6]. The first row shows an example of a car moving towards us and another where the car in the adjacent lane is speeding away. The bottom two rows show failures due to a drastic rotation and a road bump, respectively.

6. Conclusions

In certain scenarios, time-to-contact (TTC) information can be more useful than depth. However, existing TTC estimation methods either make impractical assumptions, or cannot be run in real time. We presented a framework to estimate time-to-contact (TTC) from a monocular input. In just 6.4 ms, our approach computes a temporal geofence to detect objects predicted to collide with the camera plane within a given TTC. By computing a number of such geofences, it can also estimate TTC with arbitrary quantization, including continuous TTC. We show that our method achieves competitive performance for TTC estimation—even when other methods use richer input data.

Acknowledgments

The authors would like to thank Stan Birchfield for the inspiring discussions on the theory of time-to-contact, Gengshan Yang and Junhwa Hur for their kind help in the comparisons with previous works, and anonymous reviewers and ACs for their helpful suggestions.

References

[1] Abhishek Badki, Alejandro Troccoli, Kihwan Kim, Jan Kautz, Pradeep Sen, and Orazio Gallo. Bi3D: Stereo depth estimation via binary classifications. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[2] Aseem Behl, Omid Hosseini Jafari, Siva Karthik Mustikovela, Hassan Abu Alhaija, Carsten Rother, and Andreas Geiger. Bounding boxes, segmentations and object coordinates: How important is recognition for 3D scene flow estimation in autonomous driving scenarios? In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[3] Ted Camus. Calculating time-to-contact using real-time quantized optical flow. *Technical Report*, 1995.

[4] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[5] Roberto Cipolla and Andrew Blake. Surface orientation and time to contact from image divergence and deformation. In *European Conference on Computer Vision (ECCV)*, 1992.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[7] 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 *IEEE International Conference on Computer Vision (ICCV)*, 2015.

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

[9] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular slam. In *European Conference on Computer Vision (ECCV)*, 2014.

[10] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[11] Ravi Garg, Vijay Kumar Bg, Gustavo Carneiro, and Ian Reid. Unsupervised cnn for single view depth estimation: Geometry to the rescue. In *European Conference on Computer Vision (ECCV)*, 2016.

[12] Ross Girshick. Fast R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, 2015.

[13] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[14] Heiko Hecht and Geert Savelsbergh. *Time-to-contact*. Elsevier, 2004.

[15] Berthold KP Horn, Yajun Fang, and Ichiro Masaki. Time to contact relative to a planar surface. In *IEEE Intelligent Vehicles Symposium*, 2007.

[16] Berthold KP Horn, Yajun Fang, and Ichiro Masaki. Hierarchical framework for direct gradient-based time-to-contact estimation. In *IEEE Intelligent Vehicles Symposium*, 2009.

[17] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. DeepMVS: Learning multi-view stereopsis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[18] Junhwa Hur and Stefan Roth. Self-supervised monocular scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[19] Eddy Ilg, Tonmoy Saikia, Margret Keuper, and Thomas Brox. Occlusions, motion and depth boundaries with a generic network for disparity, optical flow or scene flow estimation. In *European Conference on Computer Vision (ECCV)*, 2018.

[20] Sunghoon Im, Hae-Gon Jeon, Stephen Lin, and In So Kweon. DPSNet: End-to-end deep plane sweep stereo. *International Conference on Learning Representations (ICLR)*, 2019.

[21] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[22] David N Lee. A theory of visual control of braking based on information about time-to-collision. *Perception*, 1976.

[23] Congcong Li, Haoyu Ma, and Qingmin Liao. Two-stage adaptive object scene flow using hybrid cnn-crf model. In *International Conference on Pattern Recognition (ICPR)*, 2020.

[24] Chao Liu, Jinwei Gu, Kihwan Kim, Srinivasa G Narasimhan, and Jan Kautz. Neural RGB $\rightarrow$ D sensing: Depth and uncertainty from a video camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[25] Chenxu Luo, Zhenheng Yang, Peng Wang, Yang Wang, Wei Xu, Ram Nevatia, and Alan L. Yuille. Every Pixel Counts ++: Joint learning of geometry and motion with 3D holistic understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[26] Xuan Luo, Jia-Bin Huang, Richard Szeliski, Kevin Matzen, and Johannes Kopf. Consistent video depth estimation. In *ACM Transactions on Graphics (SIGGRAPH)*, 2020.

[27] Wei-Chiu Ma, Shenlong Wang, Rui Hu, Yuwen Xiong, and Raquel Urtasun. Deep rigid instance scene flow. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[28] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[30] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[31] François Meyer and Patrick Bouthemy. Estimation of time-to-collision maps from first order motion models and normal flows. In *IEEE International Conference on Pattern Recognition (ICPR)*, 1992.

[32] François G Meyer. Time-to-collision from first-order models of the motion field. *IEEE Transactions on Robotics and Automation (TRA)*, 1994.

[33] René Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 2020.

[34] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision (IJCV)*, 2002.

[35] Johannes Lutz Schönberger and Jan-Michael Frahm. Structure-from-motion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[36] René Schuster, Oliver Wasenmüller, and Didier Stricker. Dense scene flow from stereo disparity and optical flow. *arXiv preprint arXiv:1808.10146*, 2018.

[37] Muralidhara Subbarao. Bounds on time-to-collision and rotational component from first-order derivatives of image flow. *Computer Vision, Graphics, and Image Processing*, 1990.

[38] James R Tresilian. Empirical and theoretical issues in the perception of time to contact. *Journal of Experimental Psychology: Human Perception and Performance*, 1991.

[39] Benjamin Ummenhofer, Huizhong Zhou, Jonas Uhrig, Nikolaus Mayer, Eddy Ilg, Alexey Dosovitskiy, and Thomas Brox. DeMoN: Depth and motion network for learning monocular stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[40] Christoph Vogel, Konrad Schindler, and Stefan Roth. 3D scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision (IJCV)*, 2015.

[41] Li Xu, Zhenlong Dai, and Jiaya Jia. Scale invariant optical flow. In *European Conference on Computer Vision (ECCV)*, 2012.

[42] Gengshan Yang and Deva Ramanan. Upgrading optical flow to 3D scene flow through optical expansion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[43] Zhichao Yin and Jianping Shi. GeoNet: Unsupervised learning of dense depth, optical flow and camera pose. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[44] Jae Shin Yoon, Kihwan Kim, Orazio Gallo, Hyun Soo Park, and Jan Kautz. Novel view synthesis of dynamic scenes with globally coherent depths from a monocular camera. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[45] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip H.S. Torr. GA-Net: Guided aggregation net for end-to-end stereo matching. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[46] Yuliang Zou, Zelun Luo, and Jia-Bin Huang. DF-Net: Unsupervised joint learning of depth and flow using cross-task consistency. In *European Conference on Computer Vision (ECCV)*, 2018.