YOLinO: Echtzeitfähige Schätzung von linienförmigen Kartenelementen im Kontext des automatisierten Fahrens

Annika Meyer* und Christoph Stiller †

Zusammenfassung: Für viele Anwendungen im Bereich des automatisierten Fahrens fehlt es bisher an einer geeigneten, echtzeitfähigen Architektur, die Kartenmerkmale darstellen und damit eine Lokalisierung in Karten, kartenloses Fahren und auch Ende-zu-Ende-Ansätze unterstützen kann. Wir präsentieren daher YOLinO, eine neuronale Netzarchitektur, die mit mehr als 150 Bildern pro Sekunde Kartenmerkmale detektieren kann. Dies können neben Fahrstreifenrändern jegliche geometrischen Elemente einer Karte sein, die sich aus Liniensegmenten darstellen lassen, bspw. Markierungen und Bordsteine. Wir zeigen zudem, dass auch implizite Merkmale wie Mittellinien von Fahrstreifen schätzbar sind.

Schlüsselwörter: Liniendetektion, Deep Learning, Kartenmerkmale

1 Einleitung

Im automatisierten Fahren werden – wie in vielen anderen Bereichen – immer mehr Aufgaben durch tiefe neuronale Netze (NNs) übernommen. Während für viele Aufgaben, wie Objektdetektion oder pixelweise Segmentierung, bereits etablierte Repräsentationen existieren, fehlt eine passende Repräsentation für Linien und Linienzüge bislang.

Linienbasierte Objekte spielen aber für die Wahrnehmung und ein Umgebungsverständnis im automatisierten Fahren eine entscheidende Rolle: Nicht nur enthalten HD-Karten mit Fahrstreifenrändern, Markierungen oder Bordsteinen linienförmige Elemente [1]. Auch kartenlose, Ende-zu-Ende-gelernte Ansätze profitieren von einer geeigneten, kartenverwandten Repräsentation der Umgebung [2].

Aktuell werden Linien oft durch eine klassifizierte Menge von Pixeln repräsentiert. Diesen fehlt jedoch der geometrische Zusammenhang, den Linien aufweisen. Dies zeigt sich insbesondere bei der Repräsentation impliziter oder kreuzender Linien. Alternativ wird auf langsame, iterative Verfahren zurückgegriffen, die weit von Echtzeitfähigkeit entfernt sind, schlecht konvergieren und/oder nicht generisch anwendbar sind.

Dieser Beitrag stellt daher eine Linien-Repräsentation für tiefe neuronale Netze vor, die diese Probleme überwindet und eine echtzeitfähige Wahrnehmung und Darstellung von Linien und linienähnlichen Elementen erlaubt.

*Annika Meyer, M.Sc., ist wiss. Mitarbeiterin am Institut für Mess- und Regelungstechnik am Karlsruher Institut für Technologie, Kontakt: annika.meyer@kit.edu.

†Prof. Dr.-Ing. Christoph Stiller leitet das Institut für Mess- und Regelungstechnik am Karlsruher Institut für Technologie

2 Stand der Forschung

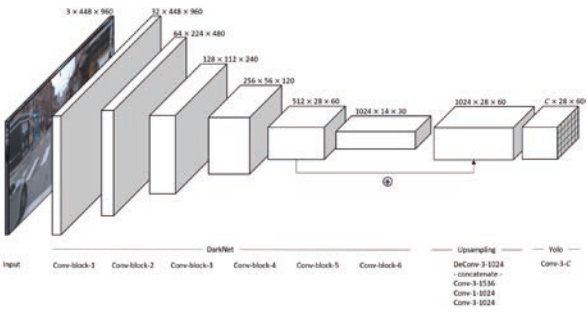
Häufig wird die Erkennung von Fahrstreifenrändern, Markierungen, der Fläche des Ego-Fahrstreifens [3] oder der Nachbar- und Gegenfahrstreifen [4] als *semantische Segmentierung* umgesetzt. Teilweise wird auch nur ein möglicher Fahrkorridor prädiziert [5]. Semantischer Segmentierung fehlt jedoch der geometrische Zusammenhang. Eine direkte Schätzung eines parametrischen Modells ist für das Problem deutlich zielführender und effektiver [6, 7]. Hierzu stellen [8] eine Architektur vor, die die Fahrstreifenränder nicht pixelweise klassifiziert, sondern als *Distanztransformation* zum nächsten Rand darstellt. Diese Form der Darstellung erleichtert eine Nachverarbeitung erheblich, ist jedoch in kreuzenden Szenarien nicht eindeutig.

Ein häufiger, jedoch stark vereinfachter Ansatz für Autobahnszenarien ist es, die Fahrstreifen im Kamerabild zu bestimmen, indem entlang bestimmter *Bildzeilen* [9, 10, 11, 12] oder *Ankerlinien* [13, 14] die Position von Punkten geschätzt wird, die auf einem Fahrstreifenrand liegen. Dies ist jedoch nicht auf Kreuzungen übertragbar. Eine Aufweichung der Anker stellt eine *Spline-Regression* dar, bei der ein Bild in Subbilder unterteilt und für jedes Subbild ein Punkt geschätzt wird, der Teil der Markierung bzw. des Fahrstreifenrands ist [15]. Diese Punkte werden dann als Kontrollpunkte für einen Spline verwendet. Für eine generische Darstellung von Fahrstreifenrändern eignen sich *gerichtete Graphen* [16], die von einem rekurrenten Netz prädiziert werden. Dabei wird in jeder Iteration ein Knoten geschätzt, der einem Randpunkt auf einem Fahrstreifen entspricht. Diese Repräsentation kann teilende und fusionierende Fahrstreifen darstellen, ist jedoch nicht einfach übertragbar auf Kreuzungen. [17] beschreiben diese Methodik für Kreuzungsränder, jedoch nicht für die Fahrstreifen.

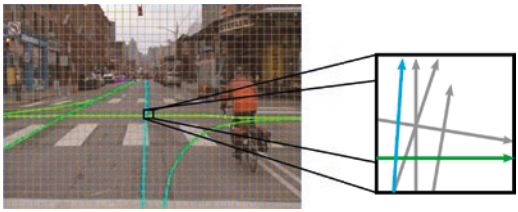
Die Forschung im Bereich der Fahrstreifenerkennung betrachtet häufig lediglich nichtkreuzende Bereiche der Straße und lässt die Komplexität von Kreuzungen unbeachtet. Zur Bestimmung von *Kreuzungsgeometrien* ist es wichtig, dass ein Algorithmus zusammenführende Fahrstreifen repräsentieren und trotz Verdeckungen Schlüsse ziehen kann. Viele Ansätze scheitern jedoch bereits daran, dass im Kamerabild horizontal verlaufende Linien nicht repräsentierbar sind. Von den bisher genannten zeigen lediglich [17] die Anwendbarkeit auf Kreuzungszenarien, wobei hier nur der Rand der Straße nicht der Rand der Fahrstreifen geschätzt wird und somit die Komplexitätsprobleme der Kreuzungen nicht gelöst werden. Kreuzungen werden eher mit Hypothesenschätzern detektiert, die jedoch ihrerseits auf Detektionsalgorithmen wie YOLinO angewiesen sind [18, 19].

3 Generische Liniensegmentschätzung

YOLinO nutzt das von der YOLO-Architektur [20] bereits bekannte Prinzip der Unterteilung eines Bildes in einzelne Zellen, für die dann jeweils mehrere Objekthypothesen geschätzt werden. Dabei wird ein vorwärtsgerichtetes, tiefes neuronales Netz eingesetzt, das in einem einzigen Prädiktionsschritt das gesamte Bild verwendet, um pro Zelle mehrere Prädiktoren zu schätzen. Mit YOLinO werden dabei Hypothesen zu Segmenten eines Linienzuges prädiziert, die in die jeweilige Zelle fallen. Abbildung 1b zeigt ein Beispiel.



(a) Architektur



(b) Zellstruktur

Abbildung 1: YOLinO-Prinzip: Für jede Zelle im Bild werden mehrere mögliche Liniensegmenthypothesen mit einer Konfidenz geschätzt. Zusätzlich kann für jedes Liniensegment implizit eine Richtung und eine Klassifikation prädiziert werden. In b) werden hohe Konfidenzen farbig dargestellt und niedrige in grau. Bildquelle: Argoverse-Datensatz [21].

3.1 Architektur

Der Backbone für YOLinO ist generisch austauschbar, solange es sich um eine komprimierenden Encoder handelt. Für die Experimente in dieser Arbeit wird ein Darknet-19 [22] verwendet. Die Architektur (vgl. Abbildung 1a) besteht aus 19 faltenden Schichten (engl. convolutional) mit fünf Pooling-Schichten. Ein spezieller Decoder berechnet hieraus die dichte Linienrepräsentation im Zellgitter mit Zellen von jeweils 16×16 Pixel.

3.2 Linien-Repräsentation

Für jede Zelle wird eine durch die Architektur bestimmte Anzahl Prädiktoren geschätzt. Jeder Prädiktor $L = (g, c, p)$ ist definiert durch eine geometrische Beschreibung g, eine Klassenverteilung $c \in [0, 1]^{|C|}$ über alle Klassen C und eine Konfidenz $p \in [0, 1]$. Die Anzahl Prädiktoren pro Zelle bestimmt somit auch die maximale Anzahl an Liniensegmenten pro räumlichen Bereich.

Zur Repräsentation der Liniensegmente bestehen mehrere Optionen [23]. Ein Liniensegment wird bspw. über je einen Start- und einen Endpunkt als $L := (s, e)$ mit $s, e \in [0, 1]^2$ in kartesischen Koordinaten definiert. Der Ursprung des Koordinatensystems liegt dabei auf der oberen linken Ecke jeder Zelle. Das Koordinatensystem der Zelle wird dabei normiert, sodass die Zellbreite (und -höhe) genau 1 entsprechen. Durch die explizite

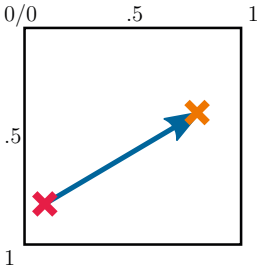


Abbildung 2: Segment mit Start- und Endpunkt

Formulierung als Start- und Endpunkt hat jedes Liniensegment eine implizite Richtung, die bspw. für Mittellinien als Fahrtrichtung interpretiert werden kann. Abbildung 2 zeigt die Repräsentation grafisch. Diese Darstellung bietet eine hohe Flexibilität, da Liniensegmente frei im Raum definiert werden können. Die Distanz zwischen zwei Liniensegmenten wird als euklidische Distanz definiert:

$$d(L, \hat{L}) := \|s - \hat{s}\|_2 + \|e - \hat{e}\|_2 \,. \tag{1}$$

3.3 Verantwortlichkeit

Ein wichtiges Konzept für direkte, einstufige Prädiktionsverfahren mit neuronalen Netzen, bei denen mehrere Hypothesen in Form von Prädiktoren geschätzt werden, ist die Verantwortlichkeit dieser Prädiktoren [20]. Für ein optimales Ergebnis werden Prädiktoren innerhalb einer Zelle für unterschiedliche Typen von Linien trainiert. Das kann z.B. bedeuten, dass ein Prädiktor für horizontale Linien und ein anderer für vertikale Linien trainiert wird. Diese unterschiedliche Verantwortlichkeit wird durch ein geschicktes Design der Kostenfunktion ermöglicht, jedoch nicht explizit modelliert.

Die Verantwortlichkeit eines Prädiktors wird über die Distanz d zur Ground Truth (GT) bestimmt. Hat ein Prädiktor L_{ik} die kürzeste Distanz zu einer GT-Linie GT_{ij} in derselben Gitterzelle G_i so ist er für diese verantwortlich. Dies wird im Folgenden über eine Indikatorfunktion beschrieben:

$$\mathbb{1}^r_{ijk} = \begin{cases} 1 & \text{wenn } L_{ik} \text{ verantwortlich für } GT_{ij} \\ 0 & \text{sonst} \end{cases} \tag{2}$$

3.4 Kostenfunktion für Liniensegmente

Gleichungen 3-5 zeigen die für Liniensegmente angepasste Kostenfunktion aus [20]. Dabei werden in jeder Gitterzelle G_i für jede prädizierte Linie L_{ik} Kosten in Bezug zu den Liniensegmenten in der GT berechnet.

$$\mathcal{L}_{loc} = \sum_{i=1}^{|G|} \sum_{j=1}^{|GT_i|} \sum_{k=1}^{|L_i|} \mathbb{1}_{ijk}^{r} d(g_{ij}, g_{ik}) \tag{3}$$

$$\mathcal{L}_{conf} = \sum_{i=1}^{|G|} \sum_{k=1}^{|L_i|} (p_{ik} - \mathbb{1}_{ik}^{r})^2 \tag{4}$$

$$\mathcal{L}_{class} = \sum_{i=1}^{|G|} \sum_{j=1}^{|GT_i|} \sum_{k=1}^{|L_i|} \mathbb{1}_{ijk}^{r} (c_{ik} - \hat{c}_{ij})^2 \tag{5}$$

Die geometrischen Lokalisierungskosten $\mathcal{L}_{loc}$ fallen dabei nur an, wenn ein Prädiktor verantwortlich ist. Mit den beiden Konfidenzkostentermen $\mathcal{L}_{conf}$ wird eine hohe Konfidenz entweder belohnt oder bestraft, je nachdem ob ein Prädiktor verantwortlich bzw. nicht verantwortlich ist. Die Klassifikationskosten des Prädiktors $\mathcal{L}_{class}$ werden mit der quadratischen Distanz zwischen den prädizierten Pseudo-Wahrscheinlichkeiten c und dem One-Hot-Encoding[1] der GT $\hat{c}$ berechnet. Die Gewichtung der einzelnen Kostenterme wurde empirisch bestimmt.

4 Non-Maximum Suppression

In direkten, einstufigen Prädiktionsverfahren mit neuronalen Netzen wird eine Vielzahl von Hypothesen inklusive einer Konfidenz für jede Hypothese geschätzt. Um nur die geeigneten Hypothesen zu erhalten, werden daher nur hinreichend kondfidente Prädiktionen ausgegeben.

Die YOLinO-Netzarchitektur prädiziert zur Laufzeit zu einem gegebenen Bild die passenden Prädiktionen als Zellstruktur. Da jede Prädiktion als $L = (g, c, p)$ definiert ist, kann über $p > \tau_p$ gefiltert werden, sodass nur sicher geschätzte Prädiktionen verarbeitet werden. Die gefilterten Prädiktionen werden anschließend mit einem dichtebasierten Verfahren (DBSCAN [24]) geclustert und je Cluster ein Repräsentant berechnet.

Für den Clusteralgorithmus werden die Prädiktoren zunächst in einen geeigneten Raum transformiert. Hierfür wird jedes Liniensegment über $\tilde{L} = (m_u, m_v, l, d_u, d_v)^\intercal$ beschrieben, wobei m_u, m_v der Mittelpunkt des Liniensegments in Bildkoordinaten, l die Länge des Segments und d_u, d_v die normierte Richtung darstellt. Dazu wird zunächst die geometrische Beschreibung des Liniensegments in Bildkoordinaten $\tilde{g}_s, \tilde{g}_e \in [0, M] \times [0, N]$ umgerechnet und Gleichung 6-8 angewendet. Die Vorfaktoren λ_d, λ_ℓ und λ_m skalieren die Koordinaten und werden empirisch für jeden Datensatz bestimmt.

$$\begin{pmatrix} m_u \\ m_v \end{pmatrix} = \lambda_m \frac{\tilde{g}_e + \tilde{g}_s}{2} \tag{6}$$

$$\ell = \lambda_\ell \left\| \tilde{g}_e - \tilde{g}_s \right\| \tag{7}$$

$$\begin{pmatrix} d_u \\ d_v \end{pmatrix} = \lambda_d \frac{\tilde{g}_e - \tilde{g}_s}{\ell} \tag{8}$$

[1] Vektor mit einer 1 an der Stelle der richtigen Klasse und ansonsten 0

Die resultierenden Cluster werden anschließend zu einem Repräsentanten zusammengefasst, der sich aus dem nach Konfidenz gewichteten Mittel der Clusterelemente ergibt. Die Konfidenz des Repräsentation wird aus dem Maximum des Clusters bestimmt.

5 Ergebnisse und Zusammenfassung

Für die Evaluation werden die prädizierten Linien abgetastet und pixelweise in den Raum (x, y, α) transformiert, der jeden Punkt mit der Pixelposition (x, y) und zudem die Richtung des Liniensegments α beschreibt. Mit dieser Darstellung werden alle Punkte als richtig-positiv betrachtet, die innerhalb eines Radius r als Nächstes zu einer Ground-Truth-Linie liegen. Daraus ergeben sich Recall, Precision und der F1-Wert.

Für den Vergleich mit dem TuSimple-Benchmark werden die prädizierten Liniensegmente nachverarbeitet. Dazu schlagen wir vor zunächst eine Connected-Component-Analyse durchzuführen, um die Instanzen zu identifizieren und einen Baum zu konstruieren. Auf jeder Ebene dieses Baums berechnen wir den mittleren Repräsentanten und legen einen kubischen Spline hindurch. Der Benchmark berechnet dann eine Genauigkeit (Acc.), Falsch-Positiv- (FP) sowie Falsch-Negativ-Rate (FN).

Methode	Acc	FP	FN	fps
LineCNN [9]	**.969**	.044	.020	17
PINet [10]	.958	.059	.033	40
LaneATT [12]	.967	.036	**.018**	250
ResNet-18 [11]	.961	**.019**[2]	.040[2]	**312**
YOLinO	.942	.188	.076	187

Tabelle 1: Unser Ansatz im Vergleich zu ausgewählten Arbeiten, die auf dem TuSimple Datensatz präsentiert wurden. Obwohl YOLinO nicht auf diese spezielle Aufgabe zugeschnitten ist, erreichen wir vergleichbare Ergebnisse. Im Hinblick auf die Inferenzzeit ist YOLinO unter den besten Beiträgen.

In [23] wurde eine ausführliche Parameterstudie präsentiert. Es kann gezeigt werden, dass auch mit 187 fps hohe Detektionsgenauigkeiten erreicht werden können. Auf dem *TuSimple* Datensatz [25] erreicht YOLinO State-of-the-Art-Ergebnisse (siehe Tabelle 1), ist aber gleichzeitig generalisierbar auf Anwendungen im Innenstadtbereich und insbesondere kreuzende Fahrstreifen. Auf dem Argoverse-Datensatz [21] zeigen wir Ergebnisse für eine *Mittellinienschätzung* mit einer Präzision von 41 %, Recall von 69 %, was zu einem F1-Wert von 52 % führt. Diese Art der Detektion wurde bisher in keiner anderen Arbeit gezeigt. Auch in einem *Luftbilddatensatz* [26] ist die Detektion und Klassifikation von Markierungen möglich. Hier erreichen wir einen F1-Wert 89 %, einen Recall von 90 % eine Präzision von 88 %. Beispiele sind in den Abbildungen 3, 4 und 5 dargestellt.

Der präsentierte Ansatz ermöglicht die Detektion von Kartenelementen, die als Linienzüge dargestellt sind. YOLinO approximiert beliebige linienförmige Darstellungen stückweise linear und prädiziert pro Zelle den Teil des Linienzuges, der in diese Zelle fällt. Dadurch wird eine dichte Darstellung ermöglicht, die keine Doppeldeutigkeit in der Zuordnung mit sich bringt und beliebige Linien approximieren kann. Im Gegensatz zu verwandten Arbeiten ermöglicht YOLinO, erstmals echtzeitfähig Kreuzungsgeometrien in Innenstädten zu beschreiben.

(a) Ground Truth **(b)** Prädiktion **(c)** Splines

Abbildung 3: Ergebnisse auf dem TuSimple-Datensatz [25]. a) Ground Truth, b) Die Prädiktion nach der NMS. c) Gesampelte Splines, die in die YOLinO Prädiktion geschätzt wurden. Die Farben in (a) and (c) bezeichnen Instanzen. In (b) visualisiert die Farbe die Orientierung der prädizierten Liniensegmente.

Abbildung 4: Ergebnisse der Mittellinienschätzung auf dem Argoverse-Datensatz [21]. Die Farben zeigen hier die Orientierung der Liniensegmente und somit die Fahrtrichtung der Linien an. Besonders für Mittellinien ist es wichtig, dass YOLinO mehrere Linien pro Zelle schätzen und somit Kreuzungen repräsentieren kann.

Abbildung 5: Ergebnisse der Markierungsdetektion und -klassifikation in Luftbildaufnahmen [26]. Luftbilder: © Stadt Karlsruhe | Liegenschaftsamt

Literatur

[1] F. Poggenhans, J.-H. Pauls, J. Janosovits, S. Orf, M. Naumann, F. Kuhnt, and M. Mayr, "Lanelet2: A high-definition map framework for the future of automated driving," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1672–1679, 2018.

[2] J. Hawke, R. Shen, C. Gurau, S. Sharma, D. Reda, N. Nikolov, P. Mazur, S. Micklethwaite, N. Griffiths, A. Shah, and A. Kendall, "Urban Driving with Conditional Imitation Learning," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 251–257, 2020.

[3] G. L. Oliveira, W. Burgard, and T. Brox, "Efficient deep models for monocular road segmentation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4885–4891, 2016.

[4] A. Meyer, N. O. Salscheider, P. F. Orzechowski, and C. Stiller, "Deep semantic lane segmentation for mapless driving," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 869–875, IEEE, 2018.

[5] D. Barnes, W. Maddern, and I. Posner, "Find your own way: Weakly-supervised segmentation of path proposals for urban autonomy," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 203–210, 2017.

[6] S. M. Azimi, P. Fischer, M. Körner, and P. Reinartz, "Aerial LaneNet: Lane-Marking Semantic Segmentation in Aerial Imagery Using Wavelet-Enhanced Cost-Sensitive Symmetric Fully Convolutional Neural Networks," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 57, pp. 2920–2938, May 2019.

[7] M. Ghafoorian, C. Nugteren, N. Baka, O. Booij, and M. Hofmann, "EL-GAN: Embedding Loss Driven Generative Adversarial Networks for Lane Detection," in *The European Conference on Computer Vision (ECCV)*, pp. 256–272, 2019.

[8] M. Bai, G. Mattyus, N. Homayounfar, S. Wang, S. K. Lakshmikanth, and R. Urtasun, "Deep Multi-Sensor Lane Detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3102–3109, 2018.

[9] X. Li, J. Li, X. Hu, and J. Yang, "Line-CNN: End-to-end traffic line detection with line proposal unit," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 1, pp. 248–258, 2020.

[10] Y. Ko, Y. Lee, S. Azam, F. Munir, M. Jeon, and W. Pedrycz, "Key points estimation and point instance segmentation approach for lane detection," *IEEE Transactions on Intelligent Transportation Systems*, pp. 1–10, 2021.

[11] Z. Qin, H. Wang, and X. Li, "Ultra fast structure-aware deep lane detection," in *The European Conference on Computer Vision (ECCV)*, vol. 16, pp. 276–291, Springer, 2020.

[12] L. Tabelini, R. Berriel, T. M. Paixão, C. Badue, A. F. De Souza, and T. Oliveira-Santos, "Keep your eyes on the lane: Attention-guided lane detection," *arXiv 2010.12035*, 2020.

[13] N. Garnett, R. Cohen, T. Pe'er, R. Lahav, and D. Levi, "3D-LaneNet: End-to-end 3D multiple lane detection," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2921–2930, 2019.

[14] T. Suleymanov, P. Amayo, and P. Newman, "Inferring road boundaries through and despite traffic," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 409–416, 2018.

[15] Y. Huang, S. Chen, Y. Chen, Z. Jian, and N. Zheng, "Spatial-Temproal Based Lane Detection Using Deep Learning," in *Artificial Intelligence Applications and Innovations*, IFIP Advances in Information and Communication Technology, pp. 143–154, 2018.

[16] N. Homayounfar, W.-C. Ma, J. Liang, X. Wu, J. Fan, and R. Urtasun, "DAGMapper: Learning to Map by Discovering Lane Topology," in *IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 2911–2920, 2019.

[17] J. Liang, N. Homayounfar, W.-C. Ma, S. Wang, and R. Urtasun, "Convolutional Recurrent Network for Road Boundary Extraction," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9512–9521, 2019.

[18] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun, "3D Traffic Scene Understanding From Movable Platforms," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, pp. 1012–1025, May 2014.

[19] A. Meyer, J. Walter, M. Lauer, and C. Stiller, "Anytime Lane-Level Intersection Estimation Based on Trajectories of Other Traffic Participants," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[20] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 779–788, 2016.

[21] M.-F. Chang, J. Lambert, P. Sangkloy, J. Singh, S. Bak, A. Hartnett, D. Wang, P. Carr, S. Lucey, D. Ramanan, and J. Hays, "Argoverse: 3D Tracking and Forecasting With Rich Maps," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 8748–8757, 2019.

[22] J. Redmon and A. Farhadi, "YOLO9000," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 7263–7271, 2017.

[23] A. Meyer, P. Skudlik, J.-H. Pauls, and C. Stiller, "Yolino: Generic single shot polyline detection in real time," in *IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, pp. 2916–2925, 2021.

[24] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *kdd*, vol. 96, pp. 226–231, 1996.

[25] TuSimple, "Lane Detection Challenge." abgerufen am 12.10.2021 von https://github.com/TuSimple/tusimple-benchmark/tree/master/doc/lane_detection, 2017.

[26] J.-H. Pauls, T. Strauss, C. Hasberg, M. Lauer, and C. Stiller, "Can We Trust Our Maps?," in *IEEE International Conference on Intelligent Transportation Systems (ITSC)*, pp. 2639–2644, 2018.