Real Time Road Lane Segmentation and Tracking System

Michael Person, Mathew Jensen, Anthony O. Smith, Nezamoddin Nezamoddini-Kachouie, Marius Silaghi

IGVC Spec 2 Team
College of Engineering and Computing
(1)College of Science
Florida Institute of Technology, Melbourne, Florida 32901
mperson2016@my.fit.edu, mjensen@fit.edu, anthonysmith@fit.edu, nezamoddin@fit.edu, msilaghi@fit.edu

ABSTRACT

Robust, low latency road lane detection is essential to the safe operation of an autonomous vehicle. Lane detection is performed by searching an image for lane markers in order to form a model of the lane that the vehicle can follow. Deep Learning CNNs are able to form robust lane models that are able to operate in poor conditions when classical computer vision algorithms will fail. However CNNs are computationally expensive models that are not yet able to detect lanes at the the necessary speed on a desktop machine let alone an embedded device. The concept being explored is how to leverage the visual power of CNNs but be able to obtain information at close to real time speeds. The proposed system is a segmentation CNN and a Bayesian tracking algorithm that is able to achieve the necessary speed but with no loss of information so that an autonomous vehicle can still safely drive within the lane. A segmentation CNN, trained on the KITTI autonomous driving dataset, outputs a dense pixelwise prediction of where the lane is. Blob detection is then performed to extract the most likely set of connected pixels. These predicted lane pixels are used to compute the lane state, which we define as the lane's inverse radius of curvature and is analogous to the lanes centerline. Since the state variable is continuous, it is nonlinearly discretized and then tracked with a particle filter on a GPU. The particle filter's output is available even when no image data has been processed by the CNN to provide information about where the lane is currently.

Keywords: Autonomous Vehicle Perception, Embedded GPU Computing, Convolutional Neural Networks, Particle Filtering

1. INTRODUCTION

For an autonomous vehicle to function safely and effectively, it must have precise knowledge about about it's surroundings. This encompasses but is not limited to detection of all other vehicles, pedestrians, cyclists, stop lights, road signs, and lastly the road itself. Before complex reasoning about the environment can be made, an autonomous vehicle's first task is to be able to identify and safely drive within the current lane. Lane detection by itself is a challenging problem with many different scenarios; there are combinations of solid, striped and white, yellow lines and sometimes there are no lane lines at all. To complicate things further, lane lines can be occluded by other vehicles, glare from the Sun can obscure the lines, or shadows on the road can provide false detections. Given how many possible situations an autonomous vehicle may encounter, it is advantageous to pursue the use of machine learning algorithms because of their power to accurately generalize to many different inputs.

Figure 1. Stanely, winner of the DARPA Grand Challenge in 2005 [1]

Once the lane the vehicle is traveling in, or travel lane, has been identified, the lane information must be mapped into a value which can be passed to the steering controller to keep the vehicle in the lane. Some such values are the Cross Track Error (CTE) in Frenet Coordinates or the lane's radius of curvature. The steering controller must be supplied with a Process Variable, the current state, and Set Point, the desired state, at a high rate of speed in order for the vehicle to be able to respond in it's dynamic environment. The controller's time restraint requires the perception system to identify the travel lane at higher rates of speed than are generally achievable by computer vision algorithms.

In this research the development of a real time lane tracking algorithm is proposed. In the algorithm, the travel lane is first segmented from the current time step's image using a high latency convolutional neural network (CNN) which is used to form the travel lane's current radius of curvature. The radius of curvature is then supplied as evidence to a particle filter which provides a Markovian prediction for the travel lane's radius of curvature at the desired speed. Finally, future works of completing the development of the lane tracking algorithm is proposed.

2. LITERATURE REVIEW

Ever since AlexNet in 2012, CNNs have become popular computer vision algorithms for a variety of tasks [2]. The first semantic segmentation, or pixelwise class labeling, network called Fully Convolutional Network was introduced in 2014 [3]. Since then CNN models have been developed specifically tailored for autonomous driving applications [4][5]. A computationally efficient

Figure 2. Cityscapes Test Set Input

Figure 3. Cityscapes Segmentation using ported PSPNet

CNN that is able to be deployed in real time called Efficient Neural Network (ENet) was able to achieve 58.3 class Intersection over Union (IoU) on the Cityscapes autonomous driving segmentation dataset [6]. In comparison Pyramid Scene Parsing Network (PSPNet) was able to achieve 80.2 class IoU but is not able to provide segmentation in real time. PSPNet utilizes a 101 convolutional layer Residual Network (ResNet101) to perform feature extraction which are then passed into a Pyramid Pooling Module. The Pyramid Pooling Module performs segmentation with the extracted features. ResNet101 has been extremely successful at image classification so it is known to provide representationaly powerful features [7]. The Pyramid Pooling Module takes the extracted feature maps from ResNet101 and uses a billinear interpolartion to resize them spatially to the the same size as the input image. A series of bottleneck convolutions, average pooling operations, and concatenation are then applied to the resized feature maps to create the final segmentation.

3. METHODOLOGY

In this section, the segmentation CNN and particle filter development is described. The algorithm is currently under development so it will be explicitly stated whether or not each section has been constructed or still needs to be built.

3.1 PSPNet Augmentation

PSPNet was first ported from its native Caffe implementation trained on Cityscapes to a Tensorflow implementation [8]. This included a trained weight conversion and also a network implementation.

After PSPNet was converted, a second Pyramid Pooling Module was added branching off from the ResNet101 output and initialized with the originally trained weights. All convolutional weights, biases, batch normalization reparameterization weights, and moving mean and variances were frozen to their original values in

Figure 4. KITTI Test Set Input

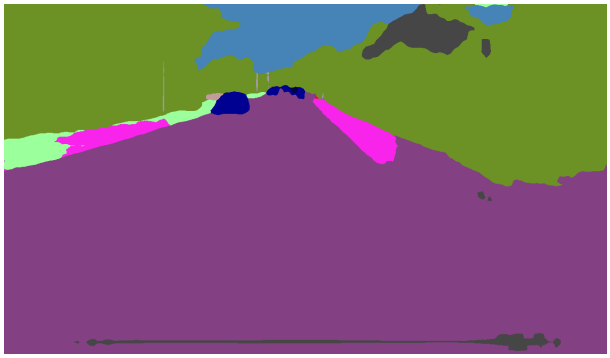

Figure 5. Original Pyramid Pooling Module Output

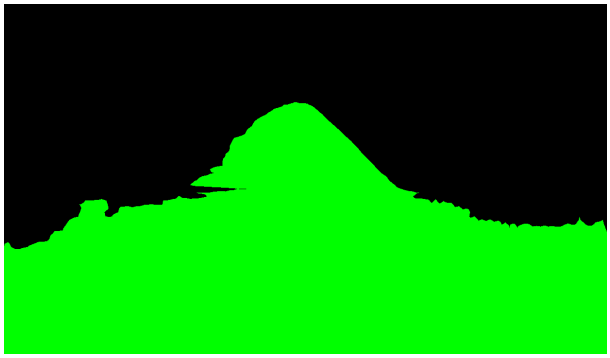

Figure 6. Lane Pyramid Pooling Module Output

ResNet101 and the original Pyramid Pooling Module while the added Pyramid Pooling Module values were left as trainable variables. The KITTI urban marked subsection of the lane detection dataset was used to train the added Pyramid Pooling Module parameters which consists of 94 labeled instances, the same data augmentation as the original PSPNet training was performed. The loss function that was optimized was the average pixel wise cross entropy and added l2 normalized weight regularization with decay value of 0.001. Training was performed for 30 epochs with a batch size of 3. The Yellowfin optimizer was used due to it's efficiency at optimizing residual-like connections [9]. After the trained model was obtained, the model was frozen and then kernel fusion was performed to create the deployment ready model for an embedded Nvidia Tegra system. All of the above mentioned pieces have been completed.

3.2 Segmentation Post Processing

After the lane had been segmented out, Gaussian Blur kernel is applied and then a connected component clustering search was performed [10]. The clustering algorithm searches for groups of spatially connected pixels. The group with the largest number of pixels was selected to be the most probable group of lane pixels.

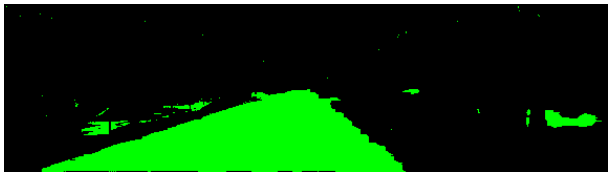

Figure 7. Example of a Poorly Segmented Lane

Figure 8. Outputted Connected Components

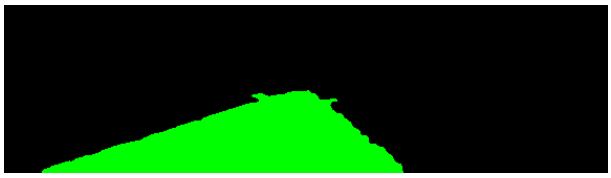

Figure 9. Selected Lane Pixel Grouping

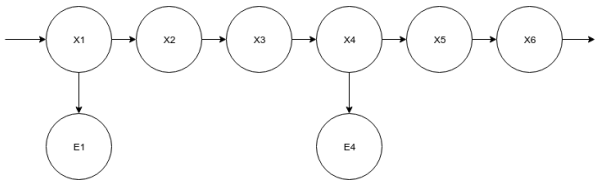

Figure 10. First Order Markov Chain

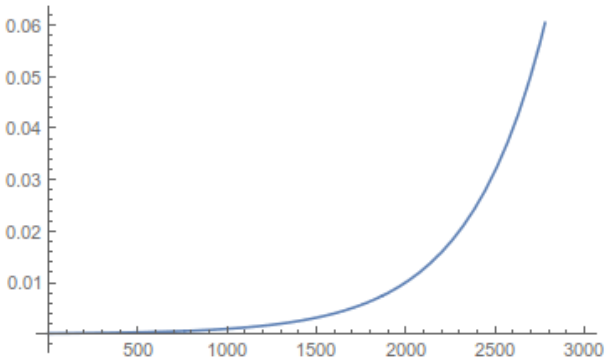

Figure 11. Separation of Discrete State Values

The poorly segmented lane can be seen in Figure 7, the output of the Gaussian Blur and connected component search can been seen in Figure 8, and the final selected lane grouping can be seen in Figure 9. All of the above mentioned functions have been implemented.

After the grouping of lane pixels has been selected, a second order polynomial will be fit using least squares to the height and width of each pixel location. The radius of curvature of the travel lane is then computed using Equation 1 [11]. The radius of curvature's sign determines whether the lane is curving left or right and an infinite radius of curvature indicates a straight travel lane. The polynomial and curve fitting has not been implemented yet.

$$R = \frac{(1+(\frac{dx}{dy})^2)^{\frac{3}{2}}}{\frac{d^2x}{d^2y}} \tag{1}$$

3.3 State Formulation

The radius of curvature of the travel lane is modeled to be a continuous random variable, ranging from negative to positive infinity, with evidence not available at every time step in a typical first order Markov Chain, seen in Figure 10. In order to ease the computational complexity of this problem the state variable is discretized with an absolute range of 10m to 10,000m. This range was chosen because an average car is not able to follow a sharper curve and above 10,000m is a large enough value to approximate a straight line. However there are two sets of these ranges; one from -10,000m to -10m and 10m to 10,000m. These ranges oppose one another because -10,000m and 10,000 meters both represent going straight and -10m and 10m both represent turning away from one another. In order to solve this, the inverse radius of curvature is used as the state variable instead [12]. The range then becomes -.1 to .1 where 0 represents going straight.

The separation between the discretized values became a challenge. If one assumed a 0.1m separation between states to allow for fine control for sharper curves then roughly 200,000 different values would need to be included in each state. Instead it was noted that closer to the -10m and 10m values a a fine level of resolution is needed but at -10,000m and 10,000m a lower resolution is acceptable. Therefore an exponentially growing separation between states was used, seen in Equation 2 and Figure 11. A was determined to be $\frac{1}{10,000}$, B was determined to be 6.90776, and n is the number of values in the discretization. These values were determined by enforcing the boundary conditions of a 0.1m separation at -.1 and .1 and 150m separation at 0. Equation 2 allows for an arbitrary large discretization while providing finer resolution where is is needed and coarser resolution where it is not. The state formulation is not yet complete.

$$s(x) = Ae^{\frac{Bx}{n}} \tag{2}$$

3.4 Transition and Sensor Model Formulation

In order to form the transition model, the way the state of the travel lane evolves of time needed to be determined. Qualitatively it was determined that the lane will most likely stay at it's current state and will, with decreasing probability, deviate to the left or right. The probability for one state to transition to another state is set to be a Gaussian distribution with the mean centered at each state. This means that each state is most likely to transition to the same state in the next time step and with ever decreasing probability transition to a state to either the left or right. Since a Gaussian is a continuous distribution that ranges between $-\inf$ and inf, a maximum transition range of 10% of the total states are populated with non-zero probabilities. After these probabilities are assigned, each row is normalized. An example transition matrix with 10 states can be seen in Equation 3.

$$\begin{bmatrix} 0.9 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.8 & 0.1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0.9 \end{bmatrix} \quad (3)$$

The sensor model is implemented in much the same way as the transition model. The found state from the image segmentation is associated with the nearest state. Then a Gaussian centered at that state's index is used to create a vector of probabilities for each state assuming only 10% of the states get a non-zero value. After the Gaussian is applied, the vector is normalized. An example sensor model vector with 10 states and found evidence at index 3 can be seen in Equation 4.

$$\begin{bmatrix} 0 \\ 0 \\ 0.1 \\ 0.8 \\ 0.1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad (4)$$

3.5 Particle Filter Implementation

It is qualitatively assumed that to accurately predict the state of the lane, a discretization of 3,000 values is needed. It is also qualitatively assumed that for an accurate approximation of the posterior, 3,000 particles is needed. Even simply an application of the transition model with a 3,000 by 3,000 matrix is not able to be done at 30Hz on a desktop CPU, let alone the CPU of a Nvidia Tegra embedded system. To solve this problem, a GPU implementation of the particle filter is developed that exploits the natural parallelization of matrix operations and the particle filtering algorithm.

The first major development is that the transition model must be applied to the ith particle at time step $t-1$ to propagate it forward one unit in time, seen in Equation 5. However, this is inefficient because it requires each particle to be done in series. In order to parallelize this operation, a particle matrix is formed, seen in Equations 6 and 7. This matrix operation now computes the transition of every particle simultaneously. This matrix multiplication is still slow using a CPU though. A speed test was performed using the Eigen library in C++, the Numpy library in Python, and the cuBLAS library in CUDA. Both CPU implementations took over 4 seconds while the cuBLAS version took less than 0.1 seconds. TODO: POPULATE W/ REAL NUMBERS

$$\vec{\mathbf{p}}_t^{(i)} = \mathbf{T}\vec{\mathbf{p}}_{t-1}^{(i)} \quad (5)$$

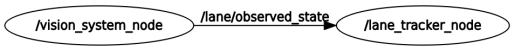

Figure 12. ROS Compute Graph

$$\mathbf{P}_t = \begin{bmatrix} \vec{\mathbf{p}}_t^{(0)} & \vec{\mathbf{p}}_t^{(1)} & \dots & \vec{\mathbf{p}}_t^{(N-1)} \end{bmatrix} \quad (6)$$

$$\mathbf{P}_t = \mathbf{P}_{t-1}\mathbf{T} \quad (7)$$

Another challenge that was overcome was the inefficiency of normalizing the particle matrix. In order to efficiently normalize each particle stored within the particle matrix, a vector of ones is generated and the vectorized summation is performed using Equation 8 using the cuBLAS library. After the summation $\vec{\alpha}_t$ is found, a CUDA kernel was written to divide each element in every column of $\mathbf{P}_t$ to perform the normalization. This GPU implementation allows for quick normalization even with large matrices.

$$\vec{\alpha}_t = \vec{1}\mathbf{P}_t \quad (8)$$

The library cuRAND is used to randomly initialize the prior distribution of the particle matrix. The last implemented function is to find the most probable state of the lane. In order to do this, the sum over all particle's states is formed in the same manner as Equation 8 and then that column sum is summed to normalize the final particle. After the normalized, average particle is found the maximum index is found and mapped back to a given radius of curvature of the travel lane.

The entire system is implemented in the Robotic Operating System (ROS) [13]. The ROS framework was chosen in order to be able to deploy this lane tracking system in real world applications. There are two ROS nodes in the system, one Python node to compute image segmentation and form the evidence and a second C++ and CUDA node to perform the particle filtering, seen in Figure 12. Currently, the particle filter publishes a predicted state at it's maximum rate and updates the particle matrix whenever evidence is computed from the segmentation process. The segmentation process occurs as fast as possible and when it is done, publishes a single floating point value of the computed state.

4. FUTURE WORK

This research project is only in it's infancy with many improvements to be made and many steps still to be completed. To complete the system, the particle filter likelihood weighting needs to be implemented as does the particle resampling. Once these are completed, an analysis of the convergence rate of the particle filter needs to be performed. The proposed method to do so is by generating artificial evidence of a straight lane with added Gaussian noise and the number of steps to convergence to the correct state will be counted. This process will also be performed with left and right curving lanes. The results of these three analyses will influence the search for the initialization of the particle matrix.

Accuracy results for the travel lane segmentation needs to be performed as well. The proposed method for this is to form separate the initial 94 labeled instances into a training and test set, retrain the

model using the new training set, and form the RMSE of the radius of curvature of the segmented lane versus the provided label in the test set.

Lastly the overall accuracy of the tracker needs to be determined. Since the KITTI dataset does not include sequences of images, a unique dataset collected at 30Hz will need to be collected including both straight and curved lanes. After which the segmentation will be performed on each image and stored. Next the images will be published at 30Hz and the output of the particle filter using evidence from the segmentation whenever possible will be be stored as well. Finally the image segmentation radius of curvature and the output of the particle filter will be used to form a single RMSE value.

5. ACKNOWLEDGMENTS

We are thankful for personal funding from the SMART Scholarship, which is funded by: USD/R&E, National Defense Education Program / BA-1, Basic Research. We would also like to thank both Magna and Polaris for funding Florida Tech's IGVC team.

6. REFERENCES

References

[1] Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, John Gale, Morgan Halpenny, Gabriel Hoffmann, Kenny Lau, Celia Oakley, Mark Palatucci, Vaughan Pratt, Pascal Stang, Sven Strohband, Cedric Dupont, Lars-Erik Jendrossek, Christian Koelen, Charles Markey, Carlo Rummel, Joe van Niekerk, Eric Jensen, Philippe Alessandrini, Gary Bradski, Bob Davies, Scott Ettinger, Adrian Kaehler, Ara Nefian, and Pamela Mahoney. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, 23(9):661–692, September 2006.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[3] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[4] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *CoRR*, abs/1612.01105, 2016.

[5] Adam Paszke, Abhishek Chaurasia, Sangpil Kim, and Eugenio Culurciello. Enet: A deep neural network architecture for real-time semantic segmentation. *CoRR*, abs/1606.02147, 2016.

[6] Marius Cordts, Mohamed Omran Sebastian Ramos, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, Bernt Schiele, Daimler Ag R, Tu Darmstadt, Mpi Informatics, and Tu Dresden. The cityscapes dataset.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Jian Zhang, Ioannis Mitliagkas, and Christopher Ré. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.

[10] Andreas Bieniek and A Moga. An efficient watershed algorithm based on connected components. 33:907–916, 06 2000.

[11] M. Bourne. Radius of curvature, March 2018.

[12] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[13] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[14] Muhammad Aizzat Zakaria, Hairi Zamzuri, and Saiful Amri Mazlan. Dynamic curvature steering control for autonomous vehicle: Performance analysis. *IOP Conference Series: Materials Science and Engineering*, 114(1):012149, 2016.

[15] Christian Rathgeber, Franz Winkler, Xiaoyu Kang, and Steffen MÃijller. Optimal trajectories for highly automated driving. 9(6):696, 2015.

[16] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[17] A Geiger, P Lenz, C Stiller, and R Urtasun. Vision meets robotics: The kitti dataset. *Int. J. Rob. Res.*, 32(11):1231–1237, September 2013.

[18] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[19] Guoliang Liu, Florentin WÃűrgÃűtter, and I MarkeliÄĞ. Combining statistical hough transform and particle filter for robust lane detection and tracking, 07 2010.

[20] Marcos Nieto, Andoni Cortés, Oihana Otaegui, Jon Arróspide, and Luis Salgado. Real-time lane tracking using rao-blackwellized particle filter. *Journal of Real-Time Image Processing*, 11(1):179–191, Jan 2016.

[21] ZuWhan Kim. Robust lane detection and tracking in challenging scenarios. 9:16 – 26, 04 2008.

[22] N Apostoloff and A Zelinsky. Robust vision based lane tracking using multiple cues and particle filtering. pages 558 – 563, 07 2003.

[23] Heidi Loose, Uwe Franke, and Christoph Stiller. Kalman particle filter for lane recognition on rural roads, 07 2009.