Tracking a suspect's vehicle in a high speed chase using machine and deep learning techniques to generate GPS datasets

Dylan Sirna, Louis Livingston

Abstract—From law enforcement officers, to fleeing suspects, to the general public, lengthy police pursuits poses a risk to all involved. With modern advances in machine vision, pursuits caught on camera can be analyzed with machine learning techniques. These techniques can then be used to produce datasets for future researchers to use in machine learning algorithms. This paper explores ways to produce those datasets using OpenCV, a machine vision library for Python, as well as GOTURN, a object tracker for OpenCV.

Index Terms—Machine Learning, High Speed Pursuits, Vehicle Tracking, Direction of Travel Detection, Computer Vision

1 Introduction

High speed car chases are a deadly occurrence. In 2018 the National Highway Traffic Safety Administration reported 427 deaths [1] as a result of a high speed chase. The number of deaths of both officers, bystanders and suspects, continues to increase every year, we propose that if we can extract turn data from these vehicles it will be a big first step in shortening the time of the chase, and also decreasing the likely hood of unnecessary death as a result. Object tracking, and vehicle tracking in particular has been a commonly researched topic, we hope to utilize these prebuilt tools and apply them to the use case of generating turn data from a high speed car chase, and ultimately take the first step in reducing the time these chases. Suspects evade the police in 35 percent of car chases, the reason for this is split relatively evenly between the police suspending the chase because it is too dangerous and on the other hand simply that the suspect evades them. The fact that high speed chases kill a person every day is unacceptable. We hope that this research is a step in the right direction towards decreasing the deaths as well as the escape statistics.

This research can also broadly be applied towards any machine vision situation, where the direction of a camera centered on an object being tracked is not known. This information is readily available to the creators of the footage however a researcher who is pulling a video from the web would not have access to this information. The direction of the camera moving to keep a tracked object centered is key information and however easily recognizable by humans is information not naturally extracted by machines [2]. We hope to enable machine vision researchers with this valuable frame direction data to use in any way they see fit, for example for our original problem statement to extract directional data from aerial police chase footage. This footage which is inherently being tracked gives little information given that the object is being actively centered [3], extracting turn data was not as straightforward as applying a tracker with a bounding box and finding the difference between the tracked points. We hope the ability to provide this directional data from videos filmed years ago under any number of conditions will provide a valuable data point for researchers going forward.

L. Livingston and D. Sirna are with the College of Computing and Software Engineering, Kennesaw State University, Marietta, GA, 30060 USA. Emails: lliving8@students.kennesaw.edu, dsirna@students.kennesaw.edu
Manuscript received xx, 2020; revised xx, 2020.

1.1 Research problem

Previous research into high speed vehicle pursuits using machine vision techniques has led to a great deal of data on the subject. Most of this data, however, is geared toward detecting the speed of the vehicle. Direction of travel of the pursuant vehicle during the chase has yet to be fully investigated. This paper hopes to rectify this gap in knowledge by demonstrating how to create datasets used to develop an algorithm based on high speed vehicle pursuits. These purposed datasets would demonstrate the direction of travel of the fleeing vehicle. This research problem is really a longstanding problem in the field of object tracking in computer vision. The problem of dealing with occlusion. The object we are proposing to track will likely be in an urban environment, it is no secret that the concentration of crime correlates with concentrations of people. Given that our object we wish to track is likely in an urban environment results in two unfavorable factors. Firstly the object will likely be regularly and completely lost from the frame of the image and few tracking solutions can hold a lock on the object for extended periods of time under these conditions.Secondly the nature of the urban environments requires that our camera angle, that is, the helicopter must at time fly in a sub optimal line to track our target, in an effort to avoid buildings and other factors, such as police helicopters. These factor contribute to a tracking environment that is incredibly noisy and ridden with varying degrees of occlusion. We need a tracking solution that can recover from prolonged periods of complete occlusion.

1.2 Purpose of the study

Ultimately, the result of this paper will be used to generate an overall algorithm for predicting the events of a police pursuit.

However, this paper will not itself generate said algorithm. Instead, this study will demonstrate how to generate datasets from aerial video of police pursuits. The datasets will detail the direction changes a suspect vehicle makes and how long in between those direction changes. In the future, these datasets will help to create machine and deep learning algorithms to assist law enforcement during high speed chases. We hope to use data from pervious pojects utilizing UAVs to accomplish this as well [4].

1.3 Audience

In the future, we hope this paper will be used to create a larger algorithm from the datasets generated from the methods we propose. As such, our audience would generally be researchers investigating how to reduce the length of high speed vehicle pursuits. However, we also forsee the proposed methods could also be used by anyone wishing to track moving objects from an aerial view using openCV. They could most likely tailor the method to reflect their use case. As such, this paper is also geared toward those individuals or groups.

1.4 Contribution

In the end, this paper will demonstrate how to detect turning and generate datasets related to suspect vehicle change of direction during vehicle pursuits. Using the openCV Python library, a video of a police pursuit shot from an aerial perspective would be loaded into a python script. From there, the fleeing vehicle would be detected and tracked using openCV tracking api. The script will then track any cardinal direction change (North, South, East, West, etc.) as well as keep track of how long the vehicle stays on a particular road before turning. We also hope to be able to track the actual street names the suspect vehicle uses, but this may be unattainable due to time constraints. This last part may prove difficult due to the lack of algorithm out there for this topic [5].

1.5 Motivation

From 1996 to 2015, the United States averaged over 355 deaths per year as a by-product of a police pursuit, resulting in over 7000 deaths over a twenty year period [1]. Moreover, the longer the duration of a vehicle pursuit, the chance of a fatality occurring from the pursuit rises as well [6]. We hope reducing the length of vehicle pursuits will also reduce the risk of injury or death to law enforcement, the actors in the vehicle, and innocent bystanders. By creating a way to more easily generate datasets on police pursuits, we hope future researchers can develop policies and techniques to help reduce the length of the pursuits.

1.6 Paper goals and organization

This paper has two goals. First, we intend to use openCV and video of high speed pursuit scraped from the internet to detect when and what way a fleeing vehicle changes direction. Second, we hope to use this data to create an algorithm for future researchers to use. This algorithm can be used to detect change of direction in future videos and convert the result into a dataset.

Looking at the first goal, this research has two sub-goals for the datasets with a stretch sub-goal as well. First, the datasets will contain any direction change the suspect vehicle makes. Second, the duration between direction changes will be tracked by the dataset. Finally, as a bonus, we would like for the datasets to track the street names of the route the fleeing vehicle takes during the pursuit.

We will first summarize the work and research related to decreasing the length of vehicle pursuits. Next, we will discuss the techniques we used to create our dataset-generating method. Then, we will present the method itself. Later, we will display the results of the purposed method, including datasets generated from sample videos of police pursuits found on the Internet. Finally, we will discuss our conclusions on how well the method performed using these sample videos.

2 Related works

2.1 GOTURN

Related works, in applying tracking to the problem, of generation a tracking dataset for high speed chases are hard to find. Our work is closely related to the tools we are using. GOTURN is the closest parallel to the research problem we are trying to solve. GOTURN was created by David Held, Sebastian Thrum, Silvio Savarese in 2016, in the paper labeled "Learning to Track at 100FPS with Deep Regression Networks. This tracker is trained offline from single object tracking using images and videos with predefined bounding box labels. By training the model offline they are able to increase the performance of the tracker [7]. Another factor that improves its fps is the fact that the tracker, given the initial bounding box the object is not placed into any specific class allowing it to run fast and track novel objects. The fact that the model is trained on a general data set has produced its problems for us, as the tracker has a tendency to latch on to the wrong car if the our target vehicle is moving at a similar speed and orientation, however the tracker preforms better then all of our tested tracker options, especially with occlusion.

2.2 Adrian Rosebrock's OpenCV Track Object Movement

A novel idea discovered during our research came in the form of Adrian Rosebrock's "OpenCV Track Object Movement" article. In the article, he presents snippets of code which will track the movement of a tennis ball in a live webcam feed or a saved video. Written using OpenCV's Python library, it presents the user with a window showing the supplied video feed and displays the current location of the ball, the change in direction of the ball in terms of x and y coordinates, and the ball's direction of travel. What we were interested in was the ability to track the ball's direction of travel. If we could utilize this code to track the suspect vehicle's in our code, we could then keep track of when the direction of travel changes.

2.3 Ambardekar's "Efficient Vehicle Tracking and Classification for an Automated Traffic Surveillance System"

Mr. Ambardekar, in his paper entitled "Efficient Vehicle Tracking and Classification for an Automated Traffic Surveillance System" produces research where he segments a foreground image of vehicles in a traffic scene using optical flow to detect the vehicles pose. They use this data to implement a model-based technique in classification and detection. Once the object has been segmented it is then classified and edge model is generated using a synthetic camera. This model is then used as a template to aid in the detection and classification of the vehicle in subsequent frames. To classify the object, Mr.Ambardekar uses color contour, as well as gradient matching. This work shed light onto our work, we too

used color contour-based matching to identify and subsequently track our object of importance, as it passes through our image space. This paper gave us direction in common ways of low-level color contour detection, this information turned out to be our method of choice, for tracking passing foliage in our problem scenario. [8]

3 Techniques used in the study

3.1 OpenCV

OpenCV offers four object tracking methods which could be valuable for tracking vehicles. All three methods have their pros and cons. With the CRST (Channel and Spatial Reliability Tracking) tracker, it offers a high degree of accuracy but is slow. KCF (Kernelized Correlation Filters) is less accurate but faster. MOSSE (Minimum Output Sum of Squared Error) is much faster than either CRST and KCF, but is also far less accurate than the other two methods. The fourth method considers is GOTURN, GOTURN is an object tracker trained in an offline manner.A few other of the models are based on online learning models these types of models work better than adaboost tracking however they are commonly slower, Due to the on the fly learning that it must compute. TLD and MEDIANFLOW belong to this category. TLD handles large changes in both occlusion and scale better than the other 6 trackers available however the OpenCV implementation provided is riddled with false positives and failure is poorly reported.

In contrast, MEDIANFLOW is worse at large scale motion, however tracking failure is reported better that all of the other trackers we have been testing, instead of latching onto another car or a tree the tracker rarely tracks the wrong object. [9]

3.2 GOTURN

GOTURN is the only one of the trackers listed that utilizes Convolutional Neural Nets. Due to the offline training, this tracker is capable of tracking at 100fps as stated in the original paper.This tracker in our testing was the most robust, with great changes in scale as well as with severe but not complete occlusion. So given the strengths of the tracker, we have use it as a starting point, however its downsides need to be addressed.The GOTURN tracker we use utilized a general caffe model, tested on people and cars and common objects.We could have trained a the tracker on a separate data set but we found that the general caffe model extensively trained on vehicles already.The generality of the model is the downfall of the tracker in out use case. The GOTURN tracker performs poorly in crowded feature spaces, and with similar objects in close proximity moving at similar speeds.The tracker has a tendency to jump to unwanted vehicles. We propose to pair GOTURN with an online object detection solution, we believe that in conjunction with an online learning object detector we could solve two of our most pressing challenges. Firstly the online object detector would help to reinitialize our GOTURN tracker when the object is lost in a situation of prolonged and complete occlusion.

Secondly the online object tracker could help our GOTURN tracker stay focused on the vehicle we are interested in, instead of jumping to nearby vehicles in close proximity. We intend to solve this problem by incorporating an object detector capable of re initializing the tracker when it fails. The second goal of our research is to extrapolate from the path data, whether or not our suspects vehicle has undergone a turn or not. We will first calculate the centroid of the vehicle we wish to track, then we will plot points as the object moves through the image, we will then connect these points with a line showing the vehicles past positions. We will then use this line to calculate the angle created to determine whether or not a turn has been taken. This method will not be affected by the orientation of the camera, given the objects initial direction we can determine weather a right or left turn has been taken. The code can then use past positional data to extract angular data with a certain threshold to detect turning, for example if a 90 degree turn is detected, that is positive indication of a turn. If an angle less than 30 degrees is detected, it can be ignored as usual car sway or a lane change without a road change.

3.3 Pandas Dataframes

Pandas, a popular Python library for data analysis, has a multitude of ways of converting Python dictionaries and arrays into meaningful and easy to reference structures [10]. One of these structures, known as a dataframe, is a table-like structure with indexed rows and labeled columns. A dataframes has two main advantages we will focus on.

The first relevant ability of a dataframe is it's ability to easily accept and convert a Python array of dictionaries. The conversion process makes each dictionary a row in the dataframe. The keys in the dictionaries become the columns of the dataframe and the values for the keys become the values for the respective dataframe columns. This will help us as we can track the direction changes and timestamps in an array of dictionaries. These dictionaries can then be converted to a dataframe.

The second relevant ability of a dataframe is it's ability to output it's data to JSON (Javascript Object Notation) and CSV (Comma Separated Values) file formats. Since both of these formats are commonly used to store datasets, we can convert our tracked direction changes and timestamps to dataframes and output them as a common form of a dataset.

4 Proposed method

In terms of how we would approach writing our code, we knew we had to choose our variable of importance carefully. If we had incorporated an object detector to draw bounding boxes on passing cars, having multiple trackers and detector would have greatly slowed down our program and was over-engineering our problem. We decided to explore another much simpler option, to pass a Hue, Saturation, and Value threshold or simply an HSV threshold to detect different passing colors in the frame. When trying to figure out an object or color present in almost all of our frames we came to the conclusion that in most of these police chase videos there is in some form or another foliage, we attempt to use this foliage as our indicator for direction. We used a Gaussian blur filter and an HSV threshold to detect the color green in our frame, we passed lower and upper HSV values to include most shades of the color green. When the HSV values are detected within the range we attempt to find the contours of the area base, differentiated by the HSV threshold. Once contours are found we then draw and enclosing min circle around the largest contour. Once this circle is drawn, we calculate its centroid and begin storing the centroid as in a queue data structure that updates each frame, as the color contour shifts through the pane. Once we have this movement, we can then take its opposite to determine the direction that our vehicle is heading.

Since the code repeats in a loop for each frame, we can keep track of the direction of travel as a variable. When the direction changes, the variable will be updated and the new direction will be added to an array as a Python dictionary. This dictionary will also include the timestamp for the direction change. Once the code has finished analyzing the entire video, it will use the array of direction change dictionaries to create a Panda's dataframe. Since dataframes can be converted to JSON and CSV data, we can then save the dataframes to those file types. These files will then serve as our intended datasets.

4.1 Code Setup

We begin by instantiating the variables used throughout the code. This includes our change of direction array, the variable which keeps track of the vehicle's current direction and the upper and lower range of green HSV values. The code then imports the video file and reads it's first frame using the OpenCV VideoCapture and read methods. From there, the program begins to loop over the frames of the video.

Inside the loop, we begin to manipulate the frame. This includes setting a Gaussian blur to the frame and then converting it's color space to HSV. The program then begins to make a mask over the image which searches for the color green. This is done using the upper and lower values of green we set at the beginning. The point behind this mask is to search and locate green objects in the frame. This is how he are tracking the foliage in the frames. Next, the code grabs the contours of the foliage and locates their centers in the image. Using the center, a circle is drawn around the largest foliage and shown on the screen by utilizing the OpenCV circle function.

The next steps are the crucial points in tracking the cardinal direction of movement. As the largest piece of foliage was tracked in the previous frame, we can then use it's center to calculate the change it's X and Y coordinates inside the frame. If it has moved in one direction, the code then notates this by taking the opposite direction and converting that into cardinal directions. This cardinal direction is set to the current direction variable.

The next step is one of the main points in which our code differs from the baseline code. At this point, the new current direction is checked against the current direction in the previous frame. If it is found to be different, a new dictionary is created and the new direction and timestamp of the change are added to the dictionary. The new dictionary is then pushed to the array of direction changes to be used later.

At this point, the code adds the cardinal direction and change in X and Y to the screen. It then increments a counter and checks if an escape key is pressed. If an escape key has not been pressed, it continues with the loop.

After the program finishes looping over all the frames of the video, the program begins to execute another main change to the baseline code. At this point, the array of direction changes is converted to a Pandas dataframe. This converts the array to a table where the rows are direction changes and the columns are the cardinal direction of change and the timestamp of the change. From there, the dataframe is converted to JSON and CSV data structures using Pandas built-in functionality. Finally, the JSON and CSV data structures are written and saved as files in their respective file formats. They can then be used as datasets in future projects.

A dataset of the videos we used can be found here.

5 Results and Discussions

Some of our initial iterations proved not to track the vehicles cardinal change in direction very well. We soon realized, however, that there was a flaw in our logic. Adrian Rosebrock's code to track the cardinal movement of objects within the frame did just as it advertised. We were trying to track the suspect vehicle, but since the camera's focus was primarily on the vehicle the whole time, so it stayed relatively centered within the frame. How could the baseline code track the direction changes of the vehicle then?

The answer is it would be unable to track these changes. We then had to think of a way to continue to use this code in a meaningful way. Our solution was simple. Instead of tracking the vehicle, we could instead track the objects around the vehicle and determine how much they move relative to the vehicle frame by frame. For example, we can see in Fig. 1 how the red line is tracking the direction the trees have moved from the previous frame. We then checked if it is a different direction of movement from the previous frame and record it in an array if it is.

One area that can be improved even farther is fine tuning this method of tracking the objects around the vehicle. While it worked for us to a certain degree, it sometimes had the tendency to jump to objects irrelevant objects. This then skewed our data by adding an incorrect change in direction. We believe future researchers will be able to modify the code in a meaningful way to prevent skewed data from occurring.

As can be seen in Fig. 2, once all of the frames of the video have been analyzed, the data array is converted into a table and can be used as a dataset. For the sake of brevity, the table has been shortened to ten rows. The example shows rows of direction changes and timestamps. As the extended form of this data is saved to both JSON and CSV files, future researchers could easily convert these tables to Pandas dataframes. From there, they would be able to use them in machine learning algorithms.

One interesting area to note is how the Gaussian blur looks when it locates the foliage. This process can be seen in Fig. 5 and Fig. 6. In the first image, the frame is normal and has located the foliage in the upper right hand corner in the previous frame. The Guassian blur is then applied and finds the previously tracked foliage by utilizing a HSV color space. This is essential in tracking the movement of the foliage in relation to the vehicle.

To give some context on how our code was run, we have provide the computer specifics for the computer it was run on in Fig. 3. The CPU it was run on was a AMD Ryzen 7 2700x CPU with a clock speed of 3.7 GHz. The computer's GPU was a Radeon RX 580 with eight gigabytes of memory. It's RAM consisted of 32 gigabytes of memory at a speed of 3200MHz.

6 Conclusions

Machine vision has made great advances since it's inception [11]. It is great for simple object detection and facial recognition as long as the background stays fairly static around the object being detected. Our goal, however, was to track a vehicle with an ever changing set variables. For the most part, OpenCV's trackers were quite erratic in their efficiency. Some greatly lacked in speed and most failed to track the vehicle after only a couple of seconds. GOTURN suffered from these shortfalls as well, but to a lesser degree. This leads use to conclude that these trackers are close, but still need to be improved before they are a reliable source to track fleeing vehicles. With this improvement, the accuracy of our data would increase as well.

Fig. 1: Tracking Surrounding Objects

Direction	Timestamp
West	0.5666666667
South-West	0.7
South	3
North-West	3.5
South-West	4.533333333
South	4.6
South-West	4.966666667
South	5
East	6.066666667
South-East	6.1

Fig. 2: Resulting Dataset Example

6.1 Room For improvement

At the moment our solution suffers from false positives, the direction of the tracked centroid points can be thrown off when the new green contours are being drawn on the frame for example if a green bush starts at the bottom of the frame and travels upwards, when that specific green contour exits the screen and there is another green bush on the right side of the frame the x, y of the tracked points jumps across the screen and can read a direction change of left to right when in reality the change in direction is north to south. The Program can be greatly improved in this regard, the problem of getting a clean re-initialization as previous

CPU	AMD Ryzen 7 2700x
Clock Speed	3.7 GHz
GPU	Radeon RX 580 Series
GPU Memory	8GB GDDR5
RAM	32 GB DDR4 3200
RAM Memory Speed	3200 MHz

Fig. 3: Computer Specifics

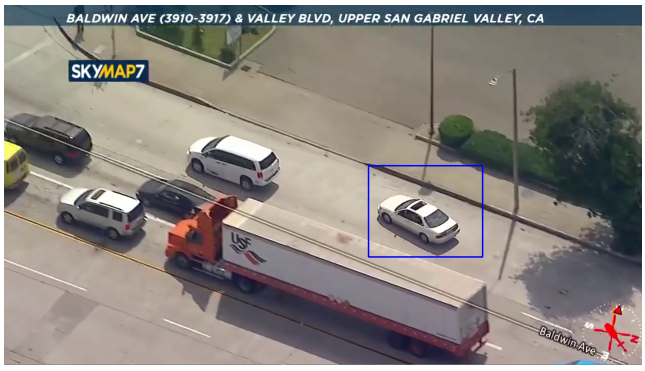

Fig. 4: Selecting the ROI

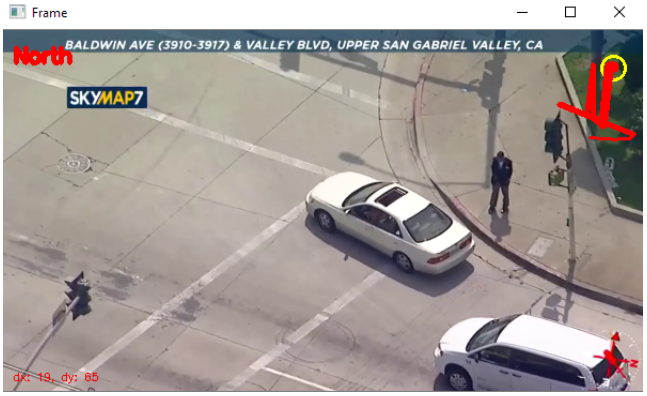

Fig. 5: Frame Before Filter is Applied

contours leave the frame is the greatest roadblock. A possible fix is to treat each detected contour as a unique object, the program currently treats all contours as the nothing of a single object which is fundamentally flawed. A possible solution it to give the detected contours some distance change threshold, which thanks to our relatively smooth transitions experienced in our police helicopter footage would present a favorable situation. The ability to differentiate between separate tracked contours would greatly decrease the false direction statements. Another possible solution would track multiple green contours to calculate a summation of the changes in x and y points, in an effort to water down the false positives. The possibility of passing contours of differing HSV values has been explored and we have found green to be the

Fig. 6: Frame After Filter is Applied

best options in terms of recognizing distinct contours as well as frequency of instance. The white HSV color space had a problem differentiating contours and the min enclosing circles drawn were often the size of half the frame.

References

[1] B. Reaves, *Police Vehicle Pursuits, 2012-2013*. US Department of Justice, Office of Justice Programs, Bureau of Justice˜. . . , 2017.

[2] D. Held, S. Thrun, and S. Savarese, "Learning to track at 100 fps with deep regression networks," in *European Conference Computer Vision (ECCV)*, 2016.

[3] R. Narmadha, R. Madhav, D. Barath, S. Kiruthika, and J. Keerthana, "Smart Moving Vehicle Detection System," *Journal of Computational and Theoretical Nanoscience*, vol. 17, no. 4, pp. 1758–1763, 2020.

[4] G. Guido, V. Gallelli, D. Rogano, and A. Vitale, "Evaluating the accuracy of vehicle tracking data obtained from Unmanned Aerial Vehicles," *International Journal of Transportation Science and Technology*, vol. 5, no. 3, pp. 136–151, 2016.

[5] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk, "A comparison and evaluation of map construction algorithms using vehicle tracking data," *GeoInformatica*, vol. 19, no. 3, pp. 601–632, 2015.

[6] C. Lum and G. Fachner, "Police Pursuits In An Age Of Innovation And Reform The IACP Police Pursuit Database," *The IACP police pursuit database. Alexandria, VA: International Association of Chiefs of Police*, no. September, pp. 1–115, 2008.

[7] S. Yang and M. Baum, "Tracking the Orientation and Axes Lengths of an Elliptical Extended Object," *IEEE Transactions on Signal Processing*, vol. 67, no. 18, pp. 4720–4729, 2019.

[8] A. A. Ambardekar, *Efficient vehicle tracking and classification for an automated traffic surveillance system*. University of Nevada, Reno, 2007.

[9] A. Mordvintsev and K. Abid, "OpenCV-Python Tutorials Documentation," *OpenCV Python Documentation*, pp. 1–269, 2017. [Online]. Available: https://media.readthedocs.org/pdf/opencv-python-tutroals/latest/opencv-python-tutroals.pdf

[10] W. McKinney *et al.*, "pandas: a foundational python library for data analysis and statistics," *Python for High Performance and Scientific Computing*, vol. 14, no. 9, 2011.

[11] E. D. Dickmanns, "The development of machine vision for road vehicles in the last decade," in *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 1. IEEE, 2003, pp. 268–281.