



UNIVERSITY OF GOTHENBURG

Analyzing the Quality of Sensor Data with Simulations combined  
with Automated Theorem Proving

*BACHELOR OF SCIENCE THESIS SOFTWARE ENGINEERING AND MANAGMENT*

MUHANAD NABEEL

JAMES OMOYA

University of Gothenburg  
Chalmers University of Technology  
Department of Computer Science and  
Engineering  
Göteborg, Sweden, June 2014

The Author grants to Chalmers University of Technology and University of Gothenburg the non-exclusive right to publish the Work electronically and in a non-commercial purpose make it accessible on the Internet. The Author warrants that he/she is the author to the Work, and warrants that the Work does not contain text, pictures or other material that violates copyright law.

The Author shall, when transferring the rights of the Work to a third party (for example a publisher or a company), acknowledge the third party about this agreement. If the Author has signed a copyright agreement with a third party regarding the Work, the Author warrants hereby that he/she has obtained any necessary permission from this third party to let Chalmers University of Technology and University of Gothenburg store the Work electronically and make it accessible on the Internet.

## Analyzing the Quality of Sensor Data with Simulations combined with Automated Theorem Proving

Muhanad Nabeel  
James Omoya

© Muhanad Nabeel, June 2014.

© James Omoya, June 2014.

Examiner: Matthias Tichy

University of Gothenburg  
Chalmers University of Technology  
Department of Computer Science and Engineering  
SE-412 96 Göteborg  
Sweden  
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering  
Göteborg, Sweden

June

2014

# ANALYZING THE QUALITY OF SENSOR DATA WITH SIMULATIONS COMBINED WITH AUTOMATED THEOREM PROVING

Olatunde James Omoya

Department of Computer Science and Engineering  
Software Engineering, Gothenburg, Sweden  
gusomoja@student.gu.se

Muhanad Nabeel

Department of Computer Science and Engineering  
Software Engineering, Gothenburg, Sweden  
Muhanad\_88@hotmail.se

## ABSTRACT

Self-Driving vehicles are still in the development process and will soon be part of our everyday life. There are companies working with this technology today and have already demonstrated a prototype of those self-driving vehicles, one of those companies is Google. Over the years ideas have been spread around in the world and many developers wanting to be part of the new technology. The DARPA Grand Challenge was created to gather skilled developers from around the world to compete with their automated cars. In this paper we focused on the efficiency part in automated parking by studying the sensors mounted on and around the vehicle. The sensors will be analyzed systematically by injecting noise data and also skipped sensor data. The vehicle will be tested with different parking scenarios in a simulating environment and the outcome of the tests will be verified by using an Automated Theorem Prover called “Vampire Theorem Prover” to draw conclusion according to the results. To determine the ground truth, we ran 100 test with different parking scenarios from which we got a subset of 58 scenarios at which the car parked successfully according to the specification while using 100% sensor quality. Selecting ten scenarios from the ground truth, we ran the tests with different noise levels and observe the parking accuracy. To achieve a parking accuracy of 90%, the sensor(s) used should have about 90% quality.

## KEYWORDS:

Sensor Quality, Automated Testing, Automated Theorem Proving, Automated Parking, Vampire theorem prover, Simulation, Self-Driving Vehicles, Lidar sensor, Laser scanner.

## 1. INTRODUCTION

Self-Driving vehicles [4] is fast becoming a reality and a major breakthrough was experienced in the field as a result of the series of Autonomous competition organized by The Defense Advanced Research Projects Agency (DARPA). The 2004/2005 DARPA Grand Challenge [6, 2, 7] saw autonomous vehicles competed in a desert environment with

rough terrain and the 2007 Urban Challenge saw autonomous vehicles compete in a city-like environment challenged to obey all traffic rules. There are companies and car manufacturers that are working with this technology and a few of them have already demonstrated a prototype of their self-driving vehicles. One of the companies is Google who have demonstrated a fully Autonomous Driving with their Google Car [5] and Volvo in their Drive Me project [17] have also been field testing their self-driving vehicle in the city of Gothenburg, Sweden and they aim to do a major testing in 2017 by rolling out 100 self-driving vehicles in Gothenburg, Sweden. For this new technology, the major issues that will concern the society and potential adopters of self-driving vehicles are how safe will these self-driving vehicles be and how reliable is the technology that makes driving decisions. Some of the most expensive and yet very important components that make up the self-driving vehicle are the Sensors use on the vehicle. These sensors are mounted on and around the vehicle to gather information about the vehicle’s environment which will then assist in making driving decisions. The Google self-driving car and some cars that competed in notable competitions like the DARPA Urban Challenge [2, 5, 6, 7], paraded an array of high-quality but expensive sensors on their autonomous vehicles one of these sensors is the Velodyne HDL-64E LIDAR Sensor [1, 2, 3]. The teams that came first and second in the competition both used the Lidar sensors [1, 13, 14] and complemented it with other sensors.

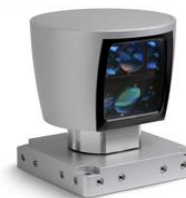


Figure 1: Velodyne HDL-64E LIDAR Sensor

The Lidar sensor boasts a  $360^{\circ}$  horizontal field of view (FOV) and about  $27^{\circ}$  vertical field of view (FOV). The sensor is equipped with 64 lasers and outputs over 1.3 million points/second. More information is shown in Appendix B.



Figure 2: Boss, the car that won the DARPA Urban Challenge[3]

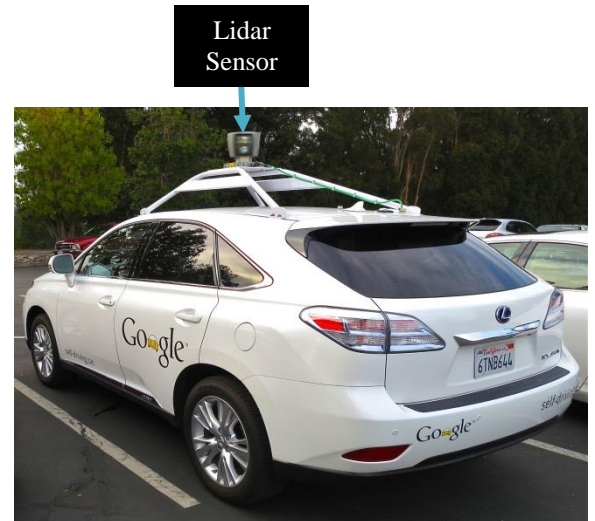


Figure 3: Google self-driving Car[5]

Using these expensive sensors will eventually make these self-driving vehicles very expensive which might then be out of reach of lots of people who might embrace the technology but won't be able to afford getting one.

By performing test automation while systematically manipulating the quality of sensors, we show the minimum quality that sensors used on self-driving vehicles should have in order to successfully actualize an automated parking.

We automatically generate different parking scenarios in the OpenDaVINCI simulator[9] using a script written in the Python programming language. Each generated scenarios have randomly distributed characteristics in terms of the number of available parking spaces and the respective positions.

The major contribution of this paper is to propose the minimum quality that the sensors mounted on self-driving vehicles should have in order to successfully achieve an automated parking. Considering the current state of research in the field of autonomous vehicles, self-driving vehicles are expected to be very expensive by the time they are made available to the public in the future. Our findings can help car manufacturers that are interested in building self-driving Vehicles to cut down on the cost of sensors thereby making autonomous vehicles available at reasonable prices.

Car manufacturers can choose between using very advanced and expensive sensors or make use of sensors that are not too expensive and which are also able to successfully achieve automated parking.

The rest of the paper is structured as follows: We present the related works in section 2 and then went further to explain the methodology we employed in section 3. We present the result of our findings in section 4 while section 5 and 6 represents our discussions and conclusions.

## 2. RELATED WORK / BACKGROUND

We selected four digital databases for our literature review which are: Springer Link, Science Direct, ACM Digital Library and IEEE Xplore Digital Library. We based our search on our keywords from which we came up with the search strings below which we then used on four databases that we have selected:

### 2.1 Search Strings

"Darpa Urban Challenge" OR "Sensor Quality" AND ("Autonomous Vehicle\*" OR "Autonomous Car\*" OR "Self-Driving Vehicle\*") OR ("Vampire" AND "Theorem prov\*")

### 2.2 Inclusion:

- I. Papers published between 2004 and 2014.
- II. Papers whose title or abstract captures a combination of the keywords that appears in our search string.

### 2.3 Exclusion:

- i. Non-English texts
- ii. Books or "Chapters in books"

Most of the relevant chapters we found belong to the book: "The DARPA Urban Challenge". We did not totally ignore chapters during the search. We applied the filtering criteria on both Springer Link and Science Direct because we couldn't get access to the chapters we found there since each chapter costs about 30 dollars. Using a snowballing approach, we then search for the chapters that we found relevant on "onlinelibrary.wiley.com" where we got access to the pdf files that we needed. [2, 3, 7, 18] are all chapters from the book: "The DARPA Urban Challenge".

- iii. Papers published before 2004
- iv. Papers focusing on a topic or field different from our area of our study

Table 4: Filtering out books and chapters

Our last filtering was done to identify and ignore papers that talks about topics that are not related to the domain of our work. Applying this criterion helped to filter out papers that is centered on fields like: Human Machine Interface, Modelling and design. It also filters out articles that are within Software Engineering but whose scope is beyond the scope of our work. Examples of these are articles that talked about: Unmanned Aerial Vehicles (UAVs), Image processing path planning and so on.

In order to carry out this particular filtering, we went through the title of all the 98 articles and on occasions where the title did not provide enough information about the domain or context of the paper we read briefly the Abstract or Introduction and also read the keywords to get a clearer view of the purpose of the article which then determined if we should include it or not. The outcome of this filtering is presented in Table 5 below. Using the snowball sampling [16] method, we also identify some relevant papers by looking into the reference section of articles that are relevant to our work and the number of the papers we found is added to the total in the table below.

Databases	Initial Value	Out of Scope	Remainder
Springer Link	63	58	5
Science Direct	35	29	6
ACM Digital Library	0		0
IEEE Xplore	0		0
Total			11
Snowball Sampling	8	0	4
Total + Snowball			15

Table 5 Ignoring papers that are out of scope

We divided the remaining papers into two categories: The first category which contains 8 papers includes articles that are based on the DARPA Challenges (Urban and Grand Challenge). The second category included papers that made use of or did an evaluation of the **Velodyne HDL-64E LIDAR Sensor** and other sensors outside the context of the DARPA Challenges.

**Category 1:**

Thrun et al. explained the implementation of their autonomous car named Stanley which won the DARPA Grand Challenge in 2005. Amongst the sensors used in Stanley are 5 laser sensors which were used to gather information about the cars environment [18].

Urmson et al. Explained the implementation of their autonomous car named Boss: that won the DARPA Urban Challenge in 2007 in addition to the very advanced and expensive **Velodyne HDL-64E LIDAR Sensor** they also used a number of other Lidar sensors and radar scanners for the car’s perception of its environment [3].

Hoffmann et al. Also explained the implementation of their autonomous car named “Junior” and it came 2<sup>nd</sup> in the DARPA Urban Challenge 2007. Like its counterpart, “Boss” that won the challenge Junior also made use of the **Velodyne HDL-64E LIDAR Sensor** and complemented it with other laser scanners [2].

Databases	Papers found
Springer Link	374
Science Direct	41
ACM Digital Library	0
IEEE Xplore	0
<b>Total</b>	<b>415</b>

Table 1: Total of papers found in all four databases

Table 1: represents the result of the initial search across the four databases with a combined total of 415 papers.

We applied a filter to exclude the papers that are not written in the English language and we narrowed the total papers from 415 to 409.

Table 2 below displays the breakdown of this filtering

Databases	Initial Value	Non English	Remainder
Springer Link	374	4	370
Science Direct	41	2	39
ACM Digital Library	0		0
IEEE Xplore	0		0
Total			<b>409</b>

Table 2: Filtering out Non- English literature

We then choose to restrict our search to include only articles ignoring others literatures like: Books, chapters in a book e.t.c.

We applied this filtering criterion on the remaining 409 results from the previous filtering and ended up with 104 articles. The breakdown of the search results is presented in Table 3 below.

Databases	Initial Value	Books/ “Chapters”	Remainder
Springer Link	370	301	69
Science Direct	39	4	35
ACM Digital Library	0		0
IEEE Xplore	0		0
Total			<b>104</b>

Table 3: Filtering out books and chapters

We decided to limit the scope of our search to articles that were published between the year 2004 and 2014. After applying this criterion, we were able to narrow the search down from 104 to 98. The breakdown of this filtering is captured in Table 4.

Databases	Initial Value	Before 2014	Remainder
Springer Link	69	6	63
Science Direct	35	0	35
ACM Digital Library	0		0
IEEE Xplore	0		0
Total			<b>98</b>

Rauskolb et al. explained the implementation of their autonomous car named “Caroline” which was among the 11 finalists in the DARPA Urban Challenge 2007 and their car also made use of several laser sensors and radars to enable the car perceive its environment [15].

**Category 2:**

Glennie and Lichti [1] did an analysis and a static calibration of the **Velodyne HDL-64E LIDAR Sensor**. Their work proposes another alternative calibration method to the standard calibration method used on the Lidar in order to achieve improved performance. The remaining papers in this category presented how the LIDAR sensor can be used to create a Map of the environment it’s used in [13, 14].

If autonomous vehicle is going to be available to civilians in the future then a lot more work needs to be done in its development so as to make the cars available at a reasonable price. Using these multiple sensors will eventually increase the total cost spent on the cars and it also means that the cars needs to be equipped with computers with enough processing power in order to perform a good sensor fusion [19] which is a very crucial activity when working with multiple sensors. The autonomous cars in most of the papers we found relied heavily on lots of sensors to make the car aware of its environment. But our paper takes a different turn and our main focus is to determine the minimum quality that sensors should possess in order to be able to successfully achieve automated parking.

**3. METHODOLOGY**

**3.1 Research Questions**

**RQ 1.** What is the minimum quality required in sensors used in self-driving vehicles in order to successfully achieve an automated parking.

**3.2 Experiment Variables**

For the self-driving vehicle experiment to be valid, some variables needed to be defined which are independent variables, controlled variables [12] and dependent variables.

- **Independent Variable:** Independent variable is the sensors noise data and skipped data, and a combination of both noise and skip.[20, 21]. We chose three fault models because we believe they can be a representation of the characteristics that are found in some of the sensors that were used in the Autonomous vehicles that competed in the DARPA Urban Challenge [2, 3]. Two of the sensors with their characteristics are presented in Appendix B.

I. **Noise:** For this fault, we inject values to the actual distance that is passed to the System under Test (SUT). For instance, applying a noise value of “4” will add “4” to the actual distance that is passed to the System under Test (SUT). If the actual distance between the

car and an obstacle is 10cm then 4 will be added thereby returning a distance of 14cm. Figure 4 represents the range of numbers for the noise fault model. The range is between 1 and 60. Using a value of 1 leads to 100% success rate and using a value of 60 leads to 0% success rate.

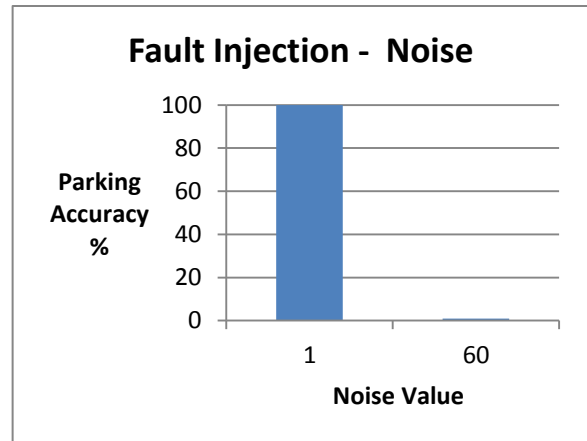


Figure 4: Fault model for noise data injection

II. **Skip:** For this fault, we intentionally skip some of the distances that are passed to the System under Test (SUT). If a skip value of 0.1 is used, then one out of every 10 distances is skipped while using the skip value of 0.5 will lead to the skipping of 5 out of 10 distances.

**Figure 5** represents the range of numbers for the skipped data fault model. The range is between 0.0 and 1.0. Using a value of 0.0 leads to 100% percent success rate and a value of 1.0 leads to 0% success rate.

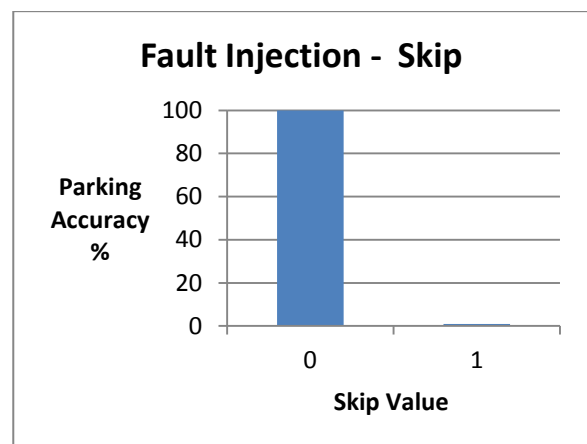


Figure 5: Fault model for Skipped data injection

III. **Combination of Noise and Skip,** this fault model is a combination of both the noise and the skip presented above. We combine noise value from the range of 1 to 60 with skip value from the range of 0.0 to 1.0. A combination of noise value “4” and skip value “0.1” will represents an overall reduction of 16% in the sensor quality(6% noise and 10% skip) using the formula below:

$$(\text{Actual Noise Value} / \text{Highest Possible Noise Value}) * 100$$

A combination of noise value of 1 and skip value of 0.1 leads to 100% parking accuracy while a combination of noise value of 60 and a skip value of 1.0 leads to a parking accuracy of 0%.

- Controlled Variable:** One of the controlled variables is the parking scenarios that the vehicle is tested on. The parking scenarios which are subsets of scenarios from our ground truth will be the same for all the sensors tested in the simulator. This way we can make sure that the tests have the same condition. The second controlled variable is the System under Test (SUT). We execute the test using the same parking algorithms for all sensor levels.
- Dependent Variable:** The dependent variable is the parking accuracy or success rate which results from the application of different fault injections to the sensors in the simulation environment. The parking accuracy is recorded for each sensor noise from which a graph will be plotted. From the application of different fault injections to the sensors in the simulation environment.

For our experiment, we made use of the Simulating environment in the OpenDaVINCI Framework [9] and an Automated Theorem Prover called Vampire. The environment is a simple straight-road setting with a sideways box-parking layout. As depicted in Figure 6. The box parking scenario can have a total of 21 available parking spaces which means all the parking spaces can be free.

When a parking position is already taken for example: Box\_3, then a box will be in that position and the car is not allowed to park or crash into the spot and the parking positions that are not taken for example Box\_4 and Box\_13 is signified by an empty space and the car is allowed to park in the space. The Parking scenario is encoded into an "SCNX" file containing the information about the properties of the scenario. The "SCNX" holds the information for a parking layout which can then be visualized using the "Cockpit" component in the OpenDaVINCI Platform [9]. Figure 6 represents an edited version of an "SCNX" file. A typical ".SCNX" file include: Lane Markings and their positions, Boxes and their positions, starting position of the car usually at position (0,0). The scenarios are then modified using a script written in the Python Programming language.

### 3.3 Experiment Design

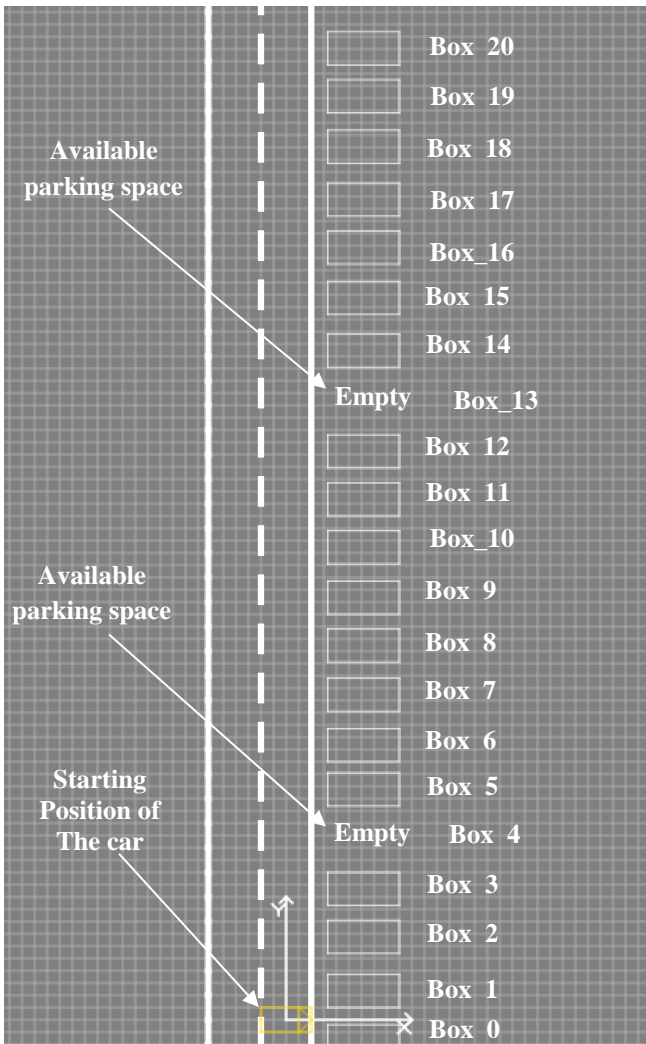


Figure 6: Box Parking scenario in simulation environment

Modification of the scenario is done for each of the test using the main scenario file. The main scenario file Figure 8 contains all the boxes, which mean that all the parking spaces are not available from the beginning. In the python script, the main scenario is read and determines how many parking space(s) should be made available using a random number from the range of 0 to 10. Generating a number of 0 means that no modification is made to the scenario and generating a number of 10 means that 10 parking spaces will be made available. After the number of parking spaces is known, another set of random number(s) from 0 till 20 are generated to determine the position where the available parking spaces should be placed. The scenario in Figure 6 depicts the outcome of the modification done to the main scenario with two available parking spaces created. For the scenario, a random number was generated in the python script, which in this case is the number 2. This number represents the number of parking space to be made available. After which two random numbers are generated in this case they are number two and number eleven and they represents the positions where the two parking spaces should be placed. As seen in Figure 6, Box\_4 and Box\_13 are now made available as a valid parking space.

## Ground Truth

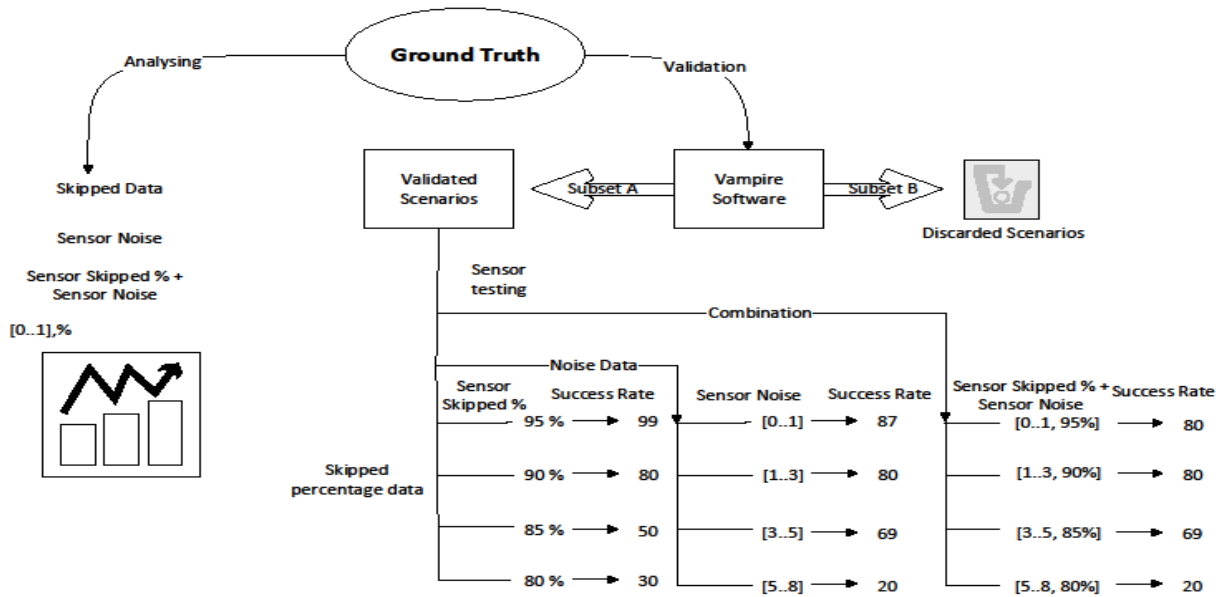


Figure 7: Design of the Ground Truth

**Figure 7** represents the structure of our experiment.

Verification was done using the Vampire software to find the Ground Truth for our experiment i.e the set of scenarios where the System under Test (SUT) performs as expected while making use of 100% sensor quality. The System under Test (SUT) is run on different parking scenarios and the results of the tests are verified using the Vampire theorem prover from which we get a subset of the scenarios where the SUT behaved according to specification.

According to the specification:

- I. If there is only one available parking space, then the car should park in that space.
- II. If there are more than one available parking space, then the car should park in the first available parking space. For instance, if there are three parking space: “position 4”, “position 9” and “position 12”, the car is expected to park in the first position which is “position 4”

We ran the test with 100% sensor quality 100 times with randomly generated scenarios to find the ground truth. For each test, a “.TPTP”(Thousands of problems for theorem prover) files was generated which encodes the parking scenario and information about the vehicle behaviour using first-order logic. An example of a “.tptp” generated for one of the scenarios is presented in Appendix A. In each “.TPTP” file, the axioms remain unchanged, but the hypothesis and conjecture are specific to the scenarios that are being tested. “.TPTP” is a library used for Automated Theorem Proving and its syntax is very similar to first-order logic formulas. After all 100 tests have been executed we verified the

“.TPTP” files using the Vampire software [8]. Out of the total 100 tests ran, there were 58 scenarios where the car

behaved as expected using 100% sensor quality and we used these subsets for our ground truth. When we observed the parking scenarios that could not be verified by the Vampire Theorem Prover based on the requirement we modelled in the “.TPTP” file, we discovered that the System under Test (SUT) is not designed to handle those particular scenarios. An example of such scenario is when there are two available spaces next to each other like having Box\_4 and Box\_5 available in figure 6 and according to the specification the car is supposed to park in the Box\_4 but car always park in the second available spot which in this case is Box\_5. Another example is when there is just one parking space available being the last position: Box\_20, but due to some limitations in the SUT, the car always fail to park in the last position and the same goes for an available parking space at the beginning. When there is only one parking space and it is at the very beginning, then the System under Test(SUT) fails to park in that position. The remaining 42 scenarios that were not verified by the Vampire theorem prover was ignored.

Finding the grounded truth before manipulating the sensor quality allowed us to find the limitations of the System under Test (SUT) which is very crucial to the credibility of the tests. For example if the car does not behave according to specification for a specific scenario while using 100% sensor quality, then it is irrelevant and can lead to erroneous conclusions if we execute a test with reduced sensor quality using the same scenario. We randomly select ten scenarios from the 58 scenarios that was verified by the Vampire theorem prover which are then used to test the cars parking behaviour under different sensor quality.



**Figure 8** shows one of the scenarios where the car behaves as expected. If there are more than one parking space available, then we expect the car to park in the first available space. As shown in Figure 8, there were two available parking space Box\_4 and Box\_13 and the car parked in the first space as expected.

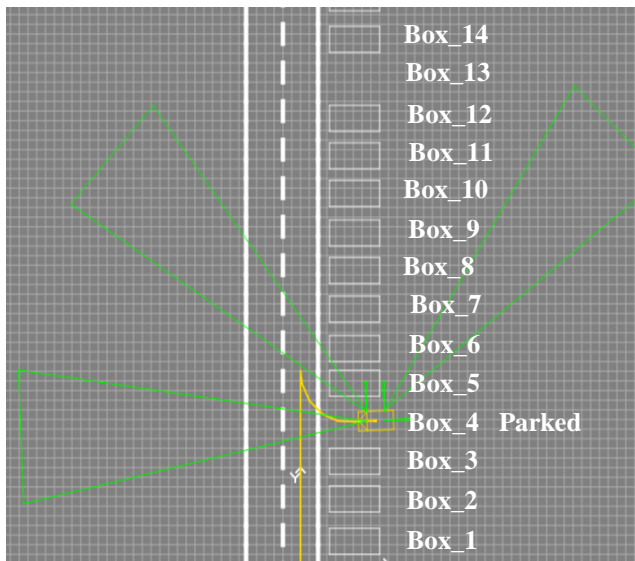


Figure 8: Ground truth scenario for sensor quality

### 3.4 Data Collection

The data are collected when testing the sensors mounted on the vehicle in the simulator, by executing the test written in C++ code. The test contains one of the scenarios generated from the Python code at a time.

Next step was to check if the vehicle has parked in the available space by checking the last position of the vehicle and calculate if it is inside the boundaries of the available parking space. Once the vehicle has parked, the results are written to a file named "Final.txt". The file contains:

- i. The scenario number.
- ii. The name of the parking space that the vehicle is expected to park in.
- iii. Parking space number
- iv. The total number of available parking space.

This is done for each scenario and if the vehicle does not park in the first parking space available in the scenario, the data will be considered invalid due to the failure of parking in the right parking space and there for the data is from the experiment. The data written into the file is important for the analyses, describing how the vehicle behaved during each scenario and taken into consideration when proceeding into the experiment. For each scenario generated from the Python code the test is triggered and executing that scenario. Those tests are recorded and can be run in the simulation environment for visualization purposes.

The experiment proceeds further with the subset of the scenarios that have successfully been validated in the simulator and the Vampire software. At this part we tested those scenarios with different sensors mounted on the

vehicle. We injected different data noise values, being four noise data, seven noise data, 27 noise data, 59 noise data and 60 noise data, into the sensors by changing the sensor parameter in the test suite. For each of those noise values we run the scenarios in the test suite and gathered the results. Further we tested the sensors with skipped data percentage with the same subset of scenarios and gathered the results. Proceeding with the experiment we combined the injection of data noises and skipped data for the sensors and run the tests, the results will be introduced in the result section.

### 3.5 Data Analysis

For this research paper we have chosen systematic analysis [11], it is an analysis suitable for our experiments throughout the procedures in terms of the data collection and the experiment design. The data we collected are from tests done on the sensor with different noise data and skipped percentage from the sensor data, these two tests are analyzed separately first. This was done systematically by increasing the noise data for the sensor and observes the behavior of the vehicle in terms of parking in the correct parking space or not, this is done for each scenario. Another test was to analyze if the vehicle parks in the scenarios with different percentage skipped values for the sensor. The analysis of those two scenarios ended when the vehicle received data it cannot comprehend and park with. The data is then aggregated for each test and introduced in a form of graph to be analyzed; viewing the data the vehicle successfully parked in, in terms of noise or skipped data percentage. This made our experiment systematic and led up to a final experiment. The Final experiment to be analyzed was the combination of the noise data and the skipped percentage until the vehicle was not able to park anymore in the Parking scenario it was setup in. This analysis captured the values in common for both data and introduced them in a graph viewing which data values combined can allow the vehicle to park. Theses analysis are introduced and explained in the result section.

## 4. RESULTS

This section will introduce the results gathered from the tests done in the simulator, with different noise data and skipped data percentage for the sensors. Each chart is based on ten scenarios we have chosen from the validated Subset mentioned in the previous sections.

### A. Sensor Noise

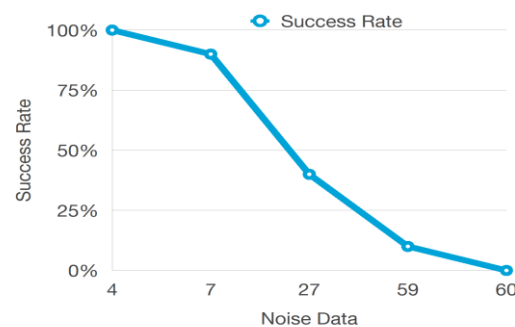


Figure 9: Noise Data Chart

In Figure 9, the chart explains the noise data applied on ten scenarios. For those scenarios the test was triggered ten times to make sure the results are valid. 100% in the X-axis in this chart means ten of the chosen scenarios have passed the same noise data. The values on the Y-axis are the noise injected into the sensors to test with.

**B. Skipped Data**

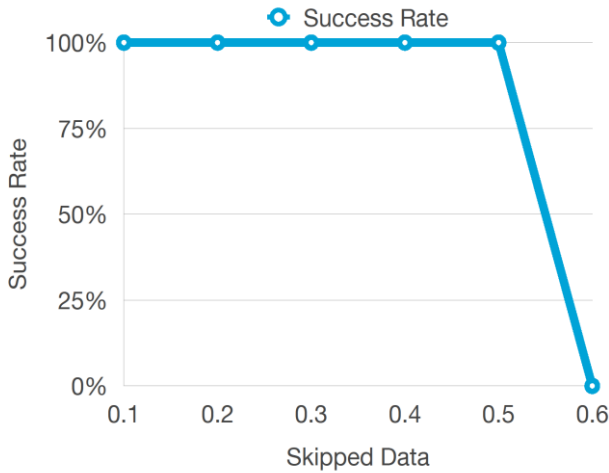


Figure 10: Skipped Data Chart

In figure 10, the chart explains the skipped sensor data from the tests done on the ten scenarios chosen. For each scenario we triggered the test ten times with the same skipped data value, the reason was to make sure the vehicle parks every time with the same value. In the X-axis of the chart the values represent the skipped data percentage, 0.1 being 10% skipped data from the sensors. In the Y-axis the percent values introduces the rate of success for all scenarios.

**C. Noise & Skipped data**

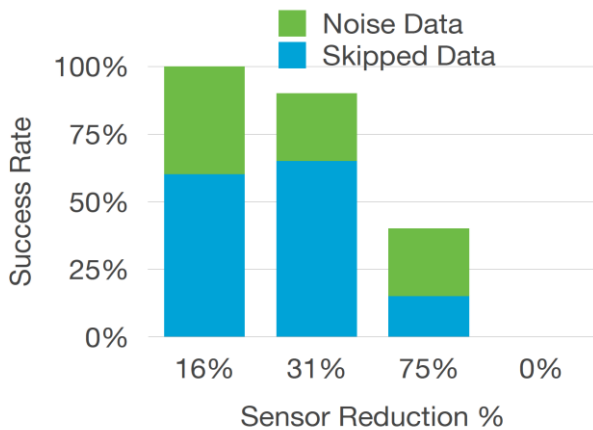


Figure 11: The combination of injecting Noise data and Skipped data percentage

The combination of injecting Noise data and Skipped data percentage is viewed in figure 11. The blue bar represents the

skipped data percentage and the green bar represents the injection of Noise data. The X-axis in the chart shows the percentage in reduction combining Noise Data and Skipped Data when tested on each scenario. By dividing the Noise value to test, with the biggest Noise data reached and multiply it by 100, to get the percentage in difference. Then combining this percentage value to the skipped data percentage, to get a total of 16 % reduction in the sensors for example. The values on the Y-axis represent the percentage rate of success for all ten scenarios.

## 5. DISCUSSIONS

### 5.1 Interpretation and Evaluate findings

For the noise fault model we tested the System under Test (SUT) by applying the range of numbers from 4 to 60. The tests were conducted using ten different scenarios selected from the subset that has been verified by the Vampire theorem prover. At the noise level of 4, the car parked successfully in all ten scenarios. Using the following formula:

$$(\text{Number of Successfully Parked} / \text{Total attempts}) * 100$$

Parking accuracy leads to a parking accuracy of 100%.

At the noise level of 7, the car parked successfully in 9 out of 10 scenarios, which leads to the success rate of 90%. At the noise level of 27 the car parked in 4 out of 10 scenarios, which leads to the success rate of 40%. At the noise level of 59 the car parked in 1 out of 10 scenarios, which leads to the success rate of 10%. The car failed to park successfully in any of the 10 scenarios when the noise level of 60 is applied to the sensors.

For the skipped sensor data fault model we tested the System under Test (SUT) by applying the range of numbers from 0.0 to 1.0. At the skipped data noise in the range of 0.1(10% reduction) to 0.5(50% reduction) the car successfully parked in all 10 scenarios, which leads to 100% success rate. However, when we applied the skipped data of 0.6, which translates to 60% reduction, the car failed to park in any of the scenarios, which leads to 0% success rate.

As depicted in Figure 11. We testes the System under Test using a combination of the two fault injection models (noise data and skipped noise data). Noise values within the range 1 to 60 were combined with skip values within the range 0.0 to 1.0.

Using a noise value of 4 combined with a skipped data of 0.1 representing an overall 16% reduction in sensor quality(6% noise and 10% skip), the car successfully parked in all 10 scenarios, which leads to one 100% parking accuracy using the formula below:

$$(\text{Number of Successfully Parked} / \text{Total attempts}) * 100$$

Using a noise value of seven combined with a skipped data of 0.2, the car parked in 9 out of 10 scenarios, which leads to a parking accuracy of 90%.

Using a noise value of 27 combined with a skipped data of 0.3 the car parked in 4 out of 10 scenarios, which leads to a parking accuracy of 40 %.

Using the noise value of 59 combined with a skipped data of 0.4, the car parked in only 1 scenario out of the 10 scenarios, which leads to a success rate of only 10% and the car failed to park in any of the 10 scenarios when a noise value 60% is combined with a skipped data of 0.5.

According to our tests, to achieve a minimum parking accuracy of 90% only a 10% reduction in the sensor quality is allowed when the sensor used can pick up noises without skipping any of the readings. This evaluates to only 10% percent error rate.

If the sensor used does not return error or corrupted data but skips some of the readings then a reduction of 50% can still be used to achieve a minimum parking accuracy of 90% (Car successfully parked 9 out of 10 times).

For the combination of both noise and skip value, we discovered that up to 31% reduction (11% noise and 20% skip) can still achieve 90% parking accuracy (Car successfully parked 9 out of 10 times).

While we were testing the sensor quality with the noise data, we discovered some patterns in some of the scenarios with regards to the car's behavior with the use of the same noise value. With the application of noise value of 8, the car failed to park for 5 of the scenarios and in all 5 scenarios, the parking space at position zero was already taken so the car is not allowed to park there. When the noise value of 28% is applied, the car failed to park in two scenarios and what both scenarios share is that the parking position zero is available for parking.

## 5.2 Validity Threats

Threats to **Construct Validity** are not critical. We will use the same parking scenario for all the tests that will be run against different sensor qualities. For manipulating the quality of the sensors in the simulation environment, we employed a systematic strategy, which makes use of three fault models. The first model is the addition of noise to the travelled distance of the sensors and the second model is the skipping of some data, while the third is the combination of the first two models.

Threats to **Internal Validity** are not considered to be critical. To ensure that there is a clear relationship between the Automated Parking accuracy and the sensor quality, we ran our tests on one computer. Executing the tests on just one computer will help us to avoid changes that might occur in the dependent variable as a result of poor performance incase more than one computers were used.

Threats to **External Validity** are considered critical since the parking algorithm that we used might not be fully representative of the parking algorithms that are used in most self-driving Vehicles. Another threat to external validity is the parking scenarios that we used in the simulator, for the

tests, we used a box-parking scenario, which might not be well representative of other forms of parking like: Parallel Parking and Diagonal Parking.

**Conclusion validity** can pose a threat to our work but we consider it to be handled in most of the cases. We measured the parking accuracy in percentage, which we derive by executing 10 tests for each noise value we applied. To determine the percentage level of the reduction made to the sensor quality we applied the following formula:

**(Actual Noise Value / Highest Possible Noise Value) \* 100.**

If the car successfully parked 9 times out of the 10 tests then we conclude that the parking accuracy is 90% and if a car failed to park in any of the ten scenarios then we concluded that the parking accuracy is zero percent. For manipulating the sensor data using the two fault models (noise and Skipped data) we tested the sensors to know the value at which the car parked in all ten scenarios and the value at which the car failed to park in all ten scenarios. For the Skipped data fault model, the range of numbers is between 0.1 and 1.0.

As specified in the above formula, determining the percentage level of an applied noise value is derived by dividing the "applied noise value" by the "maximum noise value" and then multiply the results by 100. Using the formula to the skipped data, a reduction of 0.1 to the quality of the sensor translates to 10% reduction while a reduction of 0.5 translates to 50% reduction in the sensor quality for the noise data. For the noise data fault model, the range of numbers is between 1 and 60; we applied the same formula specified above. A noise level of 7 will translate to 12% reductions and a noise level of 27 will translate to 45%.

## CONCLUSIONS

We automated the testing of the parking functionality in self-driving vehicles by combining tests carried out in the simulation with theorem proving using the Vampire theorem prover. Being able to automatically generate multiple scenarios, we were able to execute 100 tests from which we derived our ground truth which are the scenarios where the System under Test performed according to specification using 100% sensor quality.

We systematically and gradually reduce the quality of the sensor in the simulation environment by employing two fault injection models which are noise injection and skipped data injection so as to simulate the quality that are common to low quality or inexpensive sensors.

In the future, we intend to run more test for each noise level so as to have a better conclusion in terms of the parking accuracy. To determine the percentage level of the parking accuracy we applied the following formula:

**(Number of Successfully Parked / Total attempts) \* 100.**

Based on the above formula, if the car successfully parked in 6 out of 10 attempts then the success rate becomes 60%.

In our findings, a 90% sensor quality is required to achieve a parking accuracy of 90% when using a sensor which sometimes return noise or invalid sensor data and for a sensor that skips sometimes skips to return sensor data, 50% quality is still able to achieve a parking accuracy of 90%. We

however believe that inexpensive or low quality sensors can be complimented with improved and robust algorithm to achieve a better parking accuracy.

## ACKNOWLEDGEMENT

We would like to thank Ana Magazinius for the guidelines and course materials she provided for us. A big thank to our supervisor Christian Berger for all his help and time supporting us and pointing us to the right direction throughout this thesis work. We also want to thank our supervisor Laura Kovács for helping us to get started with Vampire Software and Thousands of Problems for Theorem Prover (TPTP).

## REFERENCES

- [1] C. Glennie, D. D. Lichti. Static Calibration and Analysis of the Velodyne HDL-64E S2 for High Accuracy Mobile Scanning. 2(6), pages 1610-1624, 2010.
- [2] G. Hoffmann, B. Huhnke, D. Johnston, S. Klumpp, D. Langer, A. Levandowski, J. Levinson, J. Marcil, D. Orenstein, J. Paefgen, I. Penny, A. Petrovskaya, M. Pflueger, G. Stanek, D. Stavens, A. Vogt, S. Articial, S. Cs, D. Hähnel, and S. Thrun. Junior: The Stanford Entry in the Urban Challenge. In The DARPA Urban Challenge, pages 91-123, 2009.
- [3] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. N. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, M. Gittleman, S. Harbaugh, M. Hebert, T. M. Howard, S. Kolski, A. Kelly, M. Likhachev, M. McNaughton, N. Miller, K. Peterson, B. Pilnick, R. Rajkumar, P. Rybski, B. Salesky, Y. W. Seo, S. Singh, J. Snider, A. Stentz, W. Whittaker, Z. Wolkowicki, J. Ziglar, H. Bae, T. Brown, D. Demitrish, B. Litkouhi, J. Nickolaou, V. Sadekar, W. Zhang, J. Struble, M. Taylor, M. Darms, D. Ferguson. Autonomous Driving in Urban Environments: Boss and the Urban Challenge. *Journal of Field Robotics*, 25(8), pages 425–466, 2008.
- [4] B. Ulmer. VITA - An Autonomous Road Vehicle (ARV) for Collision Avoidance in Traffic. Proceedings of the Intelligent Vehicles '92 Symposium, pages 36-41, 1992.
- [5] M. Gerla, L. Eun-Kyu, G. Pau, L. Uichin: Internet of vehicles: From intelligent grid to autonomous cars and vehicular clouds. In Internet of Things (WF-IoT), 2014 IEEE World Forum on, pages 241 – 246, 2014
- [6] B. Augusto, A. Ebadighajari, C. Englund, U. Hakeem, N. V. Irukulapati, J. Nilsson, A. Raza, and R. Sadeghitabar. Technical Aspects on Team Chalmers Solution to Cooperative Driving. Technical report, Chalmers University of Technology, 2011.
- [7] F. W. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Eertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, J.-M. Wille, P. Hecker, T. Nothdurft, M. Doering, K. Homeier, J. Morgenroth, L. Wolf, C. Basarke, C. Berger, T. Gülke, F. Klose, and B. Rumpe. Caroline: An Autonomously Driving Vehicle for Urban Environments. *Journal of Field Robotics*, 25(9):674-724, 2008.
- [8] L. Kovács and A. Voronkov. First-Order Theorem Proving and Vampire. In Proceedings of International Conference on Computer Aided Verification, pages 1-35, 2013.
- [9] OpenDaVINCI, webpage-  
<http://www.christianberger.net/OpenDaVINCI>, June 2013.
- [10] A. Platzer and J.-D. Quesel. KeYmaera: A Hybrid Theorem Prover for Hybrid Systems. In Proceedings of the Fourth International Joint Conference on Automated Reasoning, pages 171-178, 2008.
- [11] A. G. Tuckett. Applying Thematic Analysis Theory to Practice: A researcher's experience. *Contemporary Nurse*, 19(2): pages 75-87, 2005.
- [12] N. Juristo and AM. Moreno, Basics of Software Engineering Experimentation, pages 45-121, 2001.
- [13] S. Becker and N. Haala. Grammar Supported Facade Reconstruction from Mobile LiDAR Mapping. *International Archive of Photogrammetry and Remote Sensing*, XXXVIII Pa(2009): pages 229–234, 2009.
- [14] N. Haala, M. Peter, J. Kremer, and G. Hunter. Mobile LiDAR mapping for 3D point cloud collection in urban areas: a performance test. In Proceedings of the 21st International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS08, volume 37, pages 1119–1124, Beijing, China, 2008.
- [15] F. Rauskolb, K. Berger, C. Lipski, M. Magnor, K. Cornelsen, J. Effertz, T. Form, F. Graefe, S. Ohl, W. Schumacher, J.-M. Wille, P. Hecker, T. Nothdurft, M. Doering, K. Homeier, J. Morgenroth, L. Wolf, C. Basarke, C. Berger, T. Gülke, F. Klose, and B. Rumpe, “Caroline: An Autonomously Driving Vehicle for Urban Environments,” in The DARPA Urban Challenge, 2009, pp. 441-508.
- [16] L. A. Goodman. Snowball Sampling. *The Annals of Mathematical Statistics*, 32(1):148-170, 1961.
- [17] N. Fleming. Driverless cars: heading down a safer, faster highway? 206(2754), pages 34-37, 2010.
- [18] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L. E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. Niekerk, E. Jensen, P.

Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, P. Mahoney. Stanley: The Robot that Won the DARPA Grand Challenge. *Journal of Field Robotics* 23(9), pages 661–692, 2006.

[19] S. Parasuraman. Sensor Fusion for Mobile Robot Navigation: Fuzzy Associative Memory. 41, pages 251-256, 2012.

[20] A. L. Christensen, R.O'Grady, M. Birattari, M. Dorigo. Fault detection in autonomous robots based on fault injection and learning.24(1), pages 49-67, 2008

[21] R.L.O. Moraes, E. Martins. Fault Injection Approach Based on Architectural Dependencies. 3549, pages 300 – 321, 2005.

## Appendix A

One of the Generated “TPTP” files that is passed as input to the Vampire Theorem Prover

The Axioms remains unchanged for all test scenarios(Line 1 to 44):

The hypotheses and conjecture are specific for each scenarios, in the example below(line 45 to 54).

In the scenario below, six parking spaces are made available which are in position: 5, 8, 11, 17, 18, 20 and we assign the value of “0” to them respectively(line 46 to 51), while setting all other parking positions to the value of “1” to signify that they are not available “52”. According to the specification, if there are more than one parking available, we expect the car to park in the first available space. In this example, we expect the car to park in “position 5” (line 54).

```
1 %%%%
2 %type declarations
3 %%%%
4
5 %size of parking slot (as well as time)
6 tff(n_type, type, n: $int).
7 tff(parkingPos_type, type, parkingPos: $int).
8
9 %function f(i,pos) modeling the car f is parked at time i at position pos
10 tff(f_type, type, f: $int*$int>$int).
11
12 %function parking(x) modeling the parking space (0 if parking slot x is free, and 1 if it is taken)
13 tff(parking_type, type, parking: $int >$int).
14
15 %%%%
16 %axioms of parking theory
17 %%%%
18
19 tff(ax0_n, axiom, $lesseq(1,n)).
20
21 tff(ax0_f, axiom, ![I,J:$int]: (f(I,J) = 0 | f(I,J)=1)).
22
23 tff(ax0_parking, axiom, ![P:$int]: (parking(P) = 0 | parking(P)=1)).
24
25 tff(ax1, axiom, parking(0)=1).
26 tff(ax2, axiom, parking($sum(n,1))=0).
27
28 %moving further in time. Might be that lesseq is needed instead of less in f
29
30 tff(ax3, axiom, ![P:$int]: (($lesseq(0,P) & lesseq(P,$sum(n,1)))=>
31 ![I:$int, J:$int]: ($less(J,I) => $lesseq(f(J,P),f(I,P))))).
32
33 tff(ax4, axiom, ![P:$int]:(
34 ($lesseq(0,P) & $lesseq(P,$sum(n,1))) =>
35 ((parking(P)=0 & (![J:$int]: (($lesseq(1,J) & $less(J,P))=>parking(J)=1)))
36 =>
37 ( f(P,P)=1 & parkingPos=P & (![K:$int]: ($lesseq(0,K) & $lesseq(K,$sum(n,1)) & K!=P =>f(P,K)=0))
38 & (![J:$int]: ($less(P,J) => (![X:$int]: ($lesseq(0,X) & $lesseq(X,$sum(n,1)) => f(J,X) = f(P,X))))))
39 ))).
40
41 tff(ax5, axiom, f($sum(n,1), $sum(n,1))=1 => parkingPos = $sum(n,1)).
42
43 tff(ax6, axiom, f($sum(n,1), $sum(n,1))=0 => $less(0,parkingPos) & $lesseq(parkingPos,$sum(n,1))).
44
45 tff(hyp0, hypothesis, n=20).
46 tff(hyp1, hypothesis, parking(5)=0).
47 tff(hyp2, hypothesis, parking(8)=0).
48 tff(hyp3, hypothesis, parking(11)=0).
49 tff(hyp4, hypothesis, parking(17)=0).
50 tff(hyp5, hypothesis, parking(18)=0).
51 tff(hyp6, hypothesis, parking(20)=0).
52 tff(hyp7, hypothesis, ![P:$int]: (($lesseq(0,P) & $less(P,n) & ~(P=5) & ~(P=8) & ~(P=11) & ~(P=17) & ~(P=18) & ~(P=20)) => parking(P)=1)).
53
54 tff(conj, conjecture, parkingPos=5).
```

## Appendix B

Two of the Lidar sensors that were used in most of the cars that competed in the DARPA Urban Challenge 2007.



HDL-64E LIDAR

- Price: \$75,000
- 64 Laser diodes
- 360 degree – Horizontal FOV
- 26.8 degree – Vertical FOV
- 1.3million points per second



SICK Laser Rangefinder

- Price: \$6,000
- 1 Laser diode
- 180 degree – FOV
- 6,000 points per second

Source: <http://www.hizook.com/blog/2009/01/04/velodyne-hdl-64e-laser-rangefinder-lidar-pseudo-disassembled>