

# Autonomous Navigation and Collision Avoidance Robot

*Pengqi Cheng*



Electrical Engineering and Computer Sciences  
University of California at Berkeley

Technical Report No. UCB/EECS-2013-102

<http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-102.html>

May 17, 2013

Copyright © 2013, by the author(s).  
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

# Masters Project Paper

## Autonomous Navigation and Collision Avoidance Robot

Pengqi Cheng  
Electrical Engineering and Computer Sciences  
University of California, Berkeley

April 15, 2013

### Contents

<b>1</b>	<b>Abstract</b>	<b>2</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
<b>3</b>	<b>Literature Review</b>	<b>5</b>
3.1	Google Driverless Car . . . . .	5
3.2	Stanford's Junior . . . . .	6
3.3	Nissan's Mixed-mode Designs . . . . .	7
3.4	Other Implementations . . . . .	7
3.5	Our Solution . . . . .	7
3.6	Conclusion . . . . .	8
<b>4</b>	<b>Methodology</b>	<b>9</b>
4.1	GPS Choices . . . . .	9
4.2	GPS Driver Implementation . . . . .	11
4.3	Motor Encoder Driver Implementation . . . . .	12
<b>5</b>	<b>Discussion</b>	<b>13</b>
5.1	GPS Driver . . . . .	13
5.2	Motor Encoder Driver . . . . .	15
5.3	Evaluation . . . . .	16
<b>6</b>	<b>Conclusion</b>	<b>18</b>

# 1 Abstract

In the last 50 years, information technology has automated many jobs like word processing, mathematical calculation and data retrieval. However, as driving a car is much more difficult than such simple jobs, it has always been a dream to do so autonomously with a computer. With the development of related technologies, people have begun significant work on making this dream a reality.

For automakers, it is more profitable to invest in a mature market – semi-automatic vehicles. However, this project aims to develop a fully automated driving system that is profitable in the market. To accomplish this, this project uses a single personal computer instead of computer clusters as in experimental projects like Stanford’s Junior. This opens up the possibility of migrating the software into high-end smartphones due to their gradually increasing performance. In this case, we simplified existing algorithms and still keep the basic functionality, but we still have to make sure that our system is safe enough for passengers. In addition, this project only uses cheap sensors. For instance, the video cameras are low-end webcams and cameras in smartphones rather than high-end professional cameras. Thus, the definition of our sensors is lower and the responding time may be longer. So we have to use multiple sensors to reach redundancy and robustness.

This paper discusses a subset of the larger autonomous vehicle project. In particular, it includes the GPS driver module migration and Orca Robotics middleware integration.

## 2 Introduction

The number of motor vehicles has reached more than one billion one billion in the world, which has expanded for nearly ten times since the 1960s. [1]

Given people's increasing need for faster transportation, there will likely be even more vehicles in the future. As technology in the information industry has evolved a lot, it now becomes possible to utilize the computing power of machines to improve the experience of driving and organize the traffic flow in a more efficient way.

Currently, there are many driver assistance systems, including many that are mature enough for industry products. One example is Nissan's driver assistance technologies which can make driving safer and more comfortable. [2] One of these technologies is Nissan's Intelligent Pedal [3], which will automatically slow down the car to prevent collisions when there is insufficient gap or deceleration of vehicles in front. In addition, Nissan have much more technologies as shown on its website.

Moving one step further, we can change the driving process to be fully autonomous. Because this system needs no human interference in driving, people can occupy their time doing other things while on the road. In addition, there are many anticipated benefits of automated vehicles, which may include but are not limited to:

- Fewer traffic collisions [4] and higher speeds [5] because of better robustness and shorter reaction times of computers.
- Better traffic flow [6] (less congestion and higher capacity) due to less safety gaps and automatic system scheduling.
- Increased accessibility [7] as automated vehicles do not have physical limitations on the driver such as age, vision or health condition.
- Higher fuel efficiency [8] because the computer can work out the best route in current traffic conditions.
- Simplified physical infrastructure [9] like road signage.
- Less space for parking [10] as computers have more accurate control.

- No need of redundant passengers for driving [11] since the car can move without human driver.
- Less jobs needed such as driver, traffic police and toll staff.

Therefore, the project presented in this paper is part of a larger project which aims to implement such a fully automated driving system – Autonomous Navigation and Collision Avoidance Robot (ANCAR). As a driving system is a large industrial project, this paper will mainly focus on two submodules of this system – the GPS driver and the motor encoder driver.

This paper will first discuss the hardware and software architecture of the ANCAR system, and then move on the technical details of driver migration and integration into the ANCAR system. Then, this paper will talk about the results, possible applications, strengths and weaknesses.

### 3 Literature Review

Autonomous driving is not an easy task. It involves various subjects and technologies such as automation control [12], computer vision [13] and artificial intelligence [14]. Thus, a working system would be very complicated and comprehensive.

Although autonomous driving is quite challenging, with the development of related technologies, it is not completely unfeasible. There are already some companies and universities making approaches. If it comes true in the future, there will be a revolution of transportation. With more accurate control by computers, there would be many benefits as mentioned in introduction. Therefore, autonomous driving is important for improving humans' life. Also, some researches have shown the big potential market of autonomous cars [15].

Taking this big societal and marketing impacts into account, many universities and companies are doing a lot of work to automate driving process. In general, there are two types of approaches: automated mode, which does not need any human intervention, and mixed mode, which helps human driver in some limited scenarios. This section will talk about two major implementations of automated cars: the Google driverless car [16] and Stanford's Junior [17], and mixed-mode designs of Nissan.

#### 3.1 Google Driverless Car

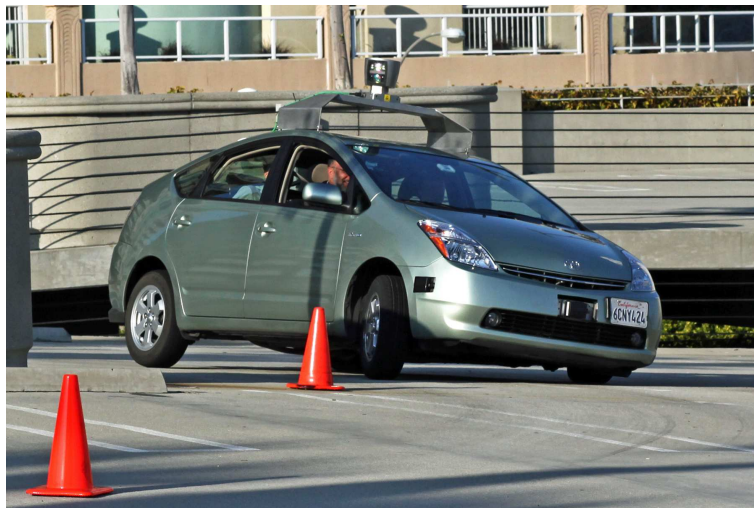


Figure 1 Google Driverless Car

The Google driverless car (Figure 1) is not just one car or one type of car. Instead, it is a project involving at least eight vehicles consisting of six Toyota Prius, an Audi TT and a Lexus RX450h [18], which is led by Google engineer Sebastian Thrun.

The Google driverless car consists of three main components: Google Maps, hardware sensors, and artificial intelligence [19]. Google Maps provides not only the map but also current traffic conditions. Sensors include GPS, LIDAR, video camera, position estimator and distance sensor. Due to the dynamic environment outside, all these sensors keep sending real-time data anytime. The artificial intelligence module decides turning and acceleration, to safely and legally take the passenger to its desired destination [20].

### 3.2 Stanford's Junior

Junior (Figure 2) got the second place in the 2007 DARPA Urban Challenge, which traveled 60 miles in about 4 hours and 29 minutes, at the speed of 13.7 mph [21]. It has many sensors, including laser sensors, INS modules, radar and DMI [17]. The software system consists of five modules: sensor interfaces, perception modules, navigation modules, drive-by-wire interface, support modules [17]. All computing and power equipment is in the vehicle. There are two Intel quad core computers inside to run all the software. Different modules communicate by IPC (Inter-Process Communication) [17].



Figure 2 Stanford's Junior



### 3.3 Nissan’s Mixed-mode Designs

Unlike the two automated system, Nissan leverages many technologies to make driving safer and more comfortable. An example is its “Intelligent Pedal” [3] as mentioned above, which will slow down the car in some dangerous cases. Also, using the radar in other directions than front, it can detect other moving objects, which is the “Moving Object Detection” technology [22]. Moreover, with the help of special infrastructure, it will know traffic conditions in advance to avoid some possible danger (Intelligent Transport Systems) [23].

### 3.4 Other Implementations

The previous sections talked about several typical and famous implementations of automatic driving system in both academia and industry. Since it is a popular topic, there are also many other research publications in this field. Prior to the 1980s, there were some approaches to autonomous driving such as [24], which is a vision system for autonomous vehicle navigation. Another example is [25], which is a car chasing system to follow another car in front. These systems are just small parts of a fully automatic system, which are far away from practical uses in traffic, but they made big approaches for later researches and implementations. In the 21st century, with the development of computational power, artificial intelligence, computer vision and other related fields, it became possible to make a fully automatic driving system without human interference. Early researches have made fully functional modules in the big system like lane detection [26] and control system [27]. Based on these results, later scientists and engineers made many usable and reliable systems. Except for those mentioned above, there are quite a lot similar systems like the system developed by the Ohio State University at the 2004 DARPA Grand Challenge [28] and the Spirit of Berlin [29].

### 3.5 Our Solution

For automakers, it is more profitable to invest in a more mature market – mixed-mode vehicles, which is different from our project, because we aim to develop a fully automated driving system. Compared with other experimental implementations mentioned above, we expect to make our system profitable in the market so that we take the cost of the system into consideration.

For example, Stanford's Junior uses two quad core computers as a cluster to support all the computation. As the performance of computers has evolved a lot, now we only use a single personal computer, and possibly migrate the software platform into high-end smartphones. In this case, we have to simplify some existed algorithms to deduct some calculation task and still remain the basic functionality. Even we lose some accuracy, we can still make sure that our system is safe and efficient enough for passengers.

In addition, we only use cheap sensors. For instance, the video cameras are low-end webcams and cameras in smartphones rather than high-end professional cameras. Thus, the definition of our sensors is lower and the responding time may be longer. So we have to use multiple sensors to reach redundancy for robustness.

### **3.6 Conclusion**

Autonomous driving has always been humans' dream. Although this task is hard, there are already some significant approaches. For most automakers, they are aiming to make automatic and manual mixed system, because it is more mature to make a profit. For technical companies and research institutions, they utilize the most advanced hardware and technologies to reach the best result as they can. Our project does care about short-term profitability so that we use low-cost sensors and computers to make a fully automated car. If we make a success, there would be a big potential market, and humans' life will improve a lot. However, there is still a long way to finally finish our prototype and then make it into the market.

## 4 Methodology

### 4.1 GPS Choices

In a navigation system, a GPS sensor is required to get location information. In the original design, the ideal way is to use both an internal GPS sensor as part of this system and an external sensor on the user's smartphone. This solution will reach a good robustness with some additional computational cost to deal with two different data sources. However, due to the limit of time, we feel that it is hard to make such a system. Therefore, we have to make a choice of either of these sensors.

After realizing that choosing a proper sensor has become a hard decision, we then analyzed both the advantages and disadvantages of both choices.

On one hand, as mentioned in the previous sections, ideally the system will be integrated with the driver's smartphone. In this case, reusing existing sensor can not only minimize the cost but also give the user a convenient interface to interact with the system by smartphone applications. If only the internal sensor is used, we have to implement some interfaces for the smartphone to get the information from the sensor and let the user to interact. In this point of view, choosing sensors on smartphones seems to be the best idea.

On the other hand, there are also significant disadvantages to develop this system based on external sensors. First of all, smartphone vendors are not willing to make all the resources utilizable to third-party applications, so they do not provide interfaces for development. What is worse, on most operating systems, it is necessary for the user to do some hacking to promote its user privileges as an administrator to get full access to the hardware, but it is possibly not able to do so on some systems. In addition, even we have resolved all these problems and taught users to get such privileges, the way to doing the same thing differs from devices to devices, as there is no specification of the GPS interfaces and the operating systems are always changing and updating. Instead, if the system only supports one kind of sensor integrated to itself, it will be more portable without changing interfaces for different types of phones. Considering the platform migration issues, we gave up the idea to develop based on smartphone sensors.

Since we have decided to use a mature GPS development kit, there still leaves one question –

what development kit do we use? In order to make the development easier, we want the kit to provide convenient development environment so that we only need minimal changes to adopt it to our system. First, we will run the system based on the Orca robotics system [30] under Linux operating system. Thus, if we do not expect to write code from the scratch, the hardware vendor must provide a native Linux driver. Second, as the demo system is controlled by a laptop, we need a USB port rather than a serial port for communication. We found that VectorNav’s VN200 can satisfy our requirements. Moreover, it has both GPS and INS (Inertial Navigation System) to get location information, and it will give the calibrated coordinates based on these two subsystems, which is quite useful when there is no signal from satellite for a short time, e.g. when the car goes through a tunnel.

Finally, we made our decision to choose the VN200 INS/GPS development kit. At first, the biggest challenge was just how to make the Linux driver work. On Windows, the sensor is better supported with an application with graphical interface (Figure 3). However, at the very beginning we could not get any output from the sensor on Linux. After we looked at the user manual of the sensor in detail, we realized we have to set some parameters like baud rate using the Windows interface before using it on Linux. Then, we were able to get the coordinates on Linux.

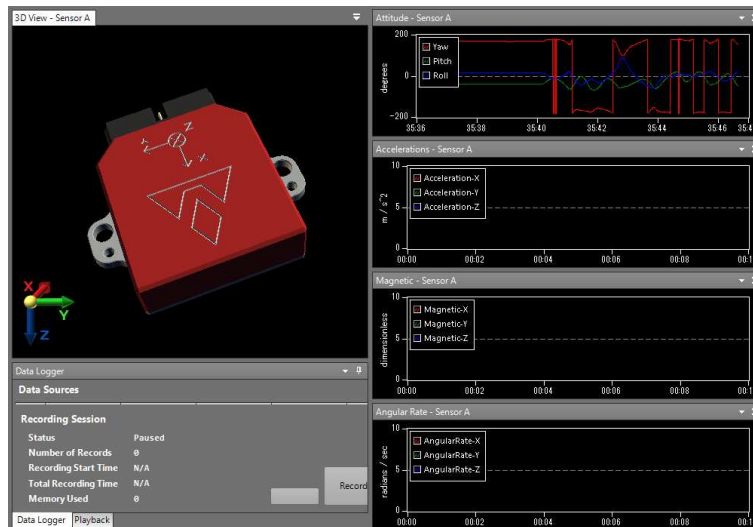


Figure 3 Windows Graphical Interface of VN200

## 4.2 GPS Driver Implementation

After making decision, we found that we still finally need to use the sensor on the Orca system, although the driver natively supports Linux. Because it is impractical to expect any sensor to natively support Orca, we have to write an Orca driver wrapper while utilizing the Linux driver. The Orca robotics system is mainly designed to help different components in a robotics system to communicate conveniently. In Orca's source directory, each component is a subdirectory under the directory of "components". For example, the code for the GPS navigation module (called `insgps` component in Orca) is under `src/components/insgps`. To support our sensor, we need to make some changes of this source tree.

Orca is written in C++. In each directory of Orca, there is a `CMakeLists.txt`, for generating the makefile. Looking at the `CMakeLists` in the original `insgps`, I found there were commands like `add_subdirectory(novatel)`. Novatel is another type of GPS sensor like VN200. There are two subdirectories in `insgps` related to Novatel, which are `novatel` and `novatelspan`. After reading the code inside these two subdirectories, I knew that `novatel` contains the basic Linux driver for Novatel and `novatelspan` is a wrapper for `novatel`. According to this design, I first copied the source files of VN200's Linux driver and make a subdirectory called `vn200` and then create a new `CMakeLists` in `vn200` just like that in `Novatel` except for names.

Then the next step is to build a `vn200span` like `novatelspan`. Similarly, the `CMakeLists` is almost the same as that in `novatelspan` except for names, and I also made a copy of Novatel-related commands in `insgps` to VN200 commands while changing names. The major files in `vn200span` are `vn200spandriver.h` and `vn200spandriver.cpp`. Like all the previous steps, they are built based on `novatelspandriver.*`. The control of Novatel is more complicated than that of VN200, as `vn200span` has already simplified most of the commands, so many of the routines and complex structures in `Novatel` are not going to be used in `vn200span`.

Based on `novatelspan`, the major changes include the following methods:

1. The constructor

Only gets the device name from the configuration file, which is usually `"/dev/ttyUSB0"`. It does not need to have any serial communication here.

2. State control methods: enable, disable, init, shutdown, reset

VN200 will automatically send data in a specified frequency after configuration. The driver does not need to keep a connection to send commands. Therefore, enable, disable, shutdown and reset are just empty. The init method only calls `vn200.connect` in `vn200` using the configuration read in the constructor.

3. `readMsgsFromHardware`

Just like other methods, this method also does not involve any manual connection. It only calls `vn200.getGpsSolution` and then does some conversion of the data format.

After changing all of this, what we need to do is just type “`cmake .`”, “`make`” and then “`make install`” in the root directory of the Orca source. Changing the default value to `Baud=230400`, `Device=/dev/ttyUSB0` and `Driver=vn200`, we can run “`insgps insgps.cfg`” to see the coordinates from the VN200 sensor.

### 4.3 Motor Encoder Driver Implementation

Similar to the GPS driver, we need to make a new interface in the component directory for our hardware. However, there is not an encoder module in the original Orca system. In order to make this new component, I first made a copy of the `laser2d` component for laser scanner, named as `arduino_enc`. Changing all the name prefixes `laser2d` into `arduino`, I got just the same new component except for name.

For this new component, I then changed its `fakedriver` into a real driver to read data from serial ports. In `fakedriver/fakedriver.cpp`, I changed `FakeDriver::read` into reading only one digit from the serial port. Also, `MainLoop::readData` in `mainloop.cpp` should be changed to adopt to the new data format.

The work done is quite similar to that for the GPS driver. Since the component is much simpler, the change in the code is less than that in the `insgps` component.

## 5 Discussion

### 5.1 GPS Driver

This section will first show the results of the GPS driver and make some comparison and explanation.

Before making any changes and/or doing software development, the first step is to just use the Official VectorNAV driver and application to show if it works. As mentioned above, there is an official application with graphical interface to show the current coordinates, speed, acceleration and other information of the GPS sensor (Figure 3). When I just plugged the sensor into the USB port on my computer, the application did not show any information other than zeroes. After looking at the user manual in detail, I reset some parameters like baud rate (which should be 230400) to make this application to correctly show its altitude, latitude and longitude.

After successfully tuned parameters for the Windows application, I then moved onto the official Linux driver. The driver provided by VectorNAV is just a library. When I just typed “make” at first, the compiler showed some error information about undefined reference. After looking up the error message on the internet, I got that we needed to add a parameter “-lrt” in the command line. Then the compilation step worked well and it generated the driver file, which is a static library.

In order to utilize this library, we must have some program which calls functions in the library to show necessary information. I made a program which shows the coordinate based on the sample program provided by VectorNAV. Fortunately, the program can display correct answer without any problem (Figure 4).

```
INS Solution:
GPS Time:      110349.238186
GPS Week:      1716
INS Status:    0004
YPR.Yaw:      +177.98
YPR.Pitch:    -0.35
YPR.Roll:     +4.16
LLR.Latitude: +37.87
LLR.Longitude: -122.27
LLR.Altitude: +22.15
Velocity.North: +0.48
Velocity.East: -0.11
Velocity.Down: +0.19
Attitude.Uncertainty: +7.78
Position.Uncertainty: +0.98
Velocity.Uncertainty: +0.18

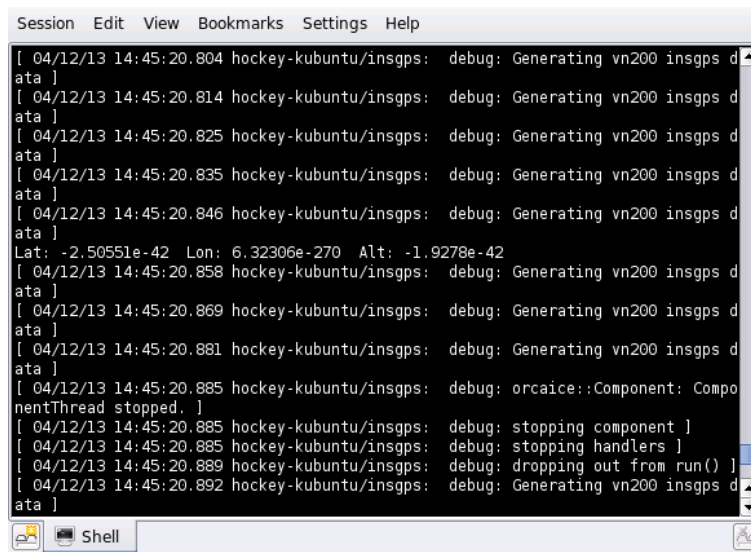
root@hockey-kubuntu:~/vn200/samples/vn200_linux#
```

Figure 4 Linux Sample Program of VN200

Therefore, there is just one more step left – integrate this VN200 driver into the Orca system. With these steps before, what we need to do is just write a driver wrapper for this sensor. Just as talked above, files and folders need to be changed include CMakeLists.txt, insgps/vn200/ and insgps/vn200span/.

The vn200 is easy to make by copying the official driver. However, there are additional works like writing a CMakeLists. Also, there was the library rt problem at first, and it was fixed while adding a command “TARGET\_LINK\_LIBRARIES rt”. This work is also similar to vn200span, in which we need to use similar commands to link with this vn200, otherwise the compiler will return some errors about undefined reference.

After writing and debugging all these code and configuration files, I finally made the driver connected to the Orca system as a component interacting with other components (Figure 5). Under the Kubuntu 8.04 platform given by Nokia, the changed driver can display the real-time coordinates of the sensor in every 0.1 second.



```
Session Edit View Bookmarks Settings Help
[ 04/12/13 14:45:20.804 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.814 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.825 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.835 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.846 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
Lat: -2.50551e-42 Lon: 6.32306e-270 Alt: -1.9278e-42
[ 04/12/13 14:45:20.858 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.869 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.881 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
[ 04/12/13 14:45:20.885 hockey-kubuntu/insgps: debug: orcaice::Component: Compo
nentThread stopped. ]
[ 04/12/13 14:45:20.885 hockey-kubuntu/insgps: debug: stopping component ]
[ 04/12/13 14:45:20.885 hockey-kubuntu/insgps: debug: stopping handlers ]
[ 04/12/13 14:45:20.889 hockey-kubuntu/insgps: debug: dropping out from run() ]
[ 04/12/13 14:45:20.892 hockey-kubuntu/insgps: debug: Generating vn200 insgps d
ata ]
Shell
```

Figure 5 Screenshot of VN200 Driver

The results above demonstrate that I have made a working GPS sensor driver for this system. From the official website of the Orca system, we can see although there is a basic interface of INS/GPS sensor provided by Orca, there is not any specific type of sensor being officially supported. The system only provides a fake driver as an example of an actual driver, which pushes out some



random data for testing. Thus, it makes me hard to compare this driver with other mature platforms.

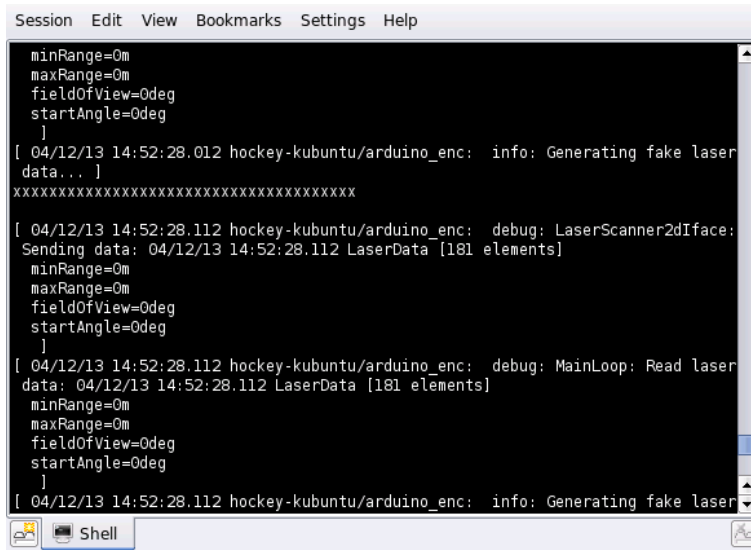
The only other actual driver that we can compare with is Novatel. Since we do not have the hardware, we can only compare them by their source codes. In the novatel folder, we can see there is only one cpp file for CRC32 checking, and the final library file libnovatel.a only contains functions for this. Other functions defined in other header files for command specification and logging are not included as part of the binary library. In the vn200 folder, the library gives us an easy access to each of the command listed in its user manual, so we only need to first initialize a vn200 struct and then pass necessary arguments as in the user manual.

The difference between novatel and vn200 makes the code of novatelspan and vn200span very distinct from each other. In novatelspan, it is clear that the driver programs does much work in detail, like setting baud rate, defining the serial port access and the format of commands. Therefore, the program looks complicated as it has to deal with so many details like mode switching, sensor disconnection, etc. For vn200span, the driver program is simpler as vn200 has a good encapsulation ready for development. My driver program only needs a direct function call when it needs to read data from the sensor. Other sections of the source code are only designed for maintaining the state of the state of the sensor, like open, close, reset and read.

Besides, the hardware design of VN200 is more advanced so that it has more functionalities than that of Novatel. For example, VN200 has an on-board chip to calculate calibrated coordinates of the GPS sensor and the IMU sensor. This can give us a better accuracy and we do not need to get these two results separately and calculate calibration on our platform. This functionality will help us to make the development even much easier.

## 5.2 Motor Encoder Driver

The motor encoder driver is much simpler, which only gives the current speed to the Orca system. This driver is simplified and modified from the laser2d component. The driver can display the speed of the motor correctly (Figure 6). Both laser2d and this arduino\_enc components are just simple modules to get data from devices. They are well designed for the Orca system to cooperate with other depending components.



```
Session Edit View Bookmarks Settings Help
minRange=0m
maxRange=0m
fieldOfView=0deg
startAngle=0deg
}
[ 04/12/13 14:52:28.012 hockey-kubuntu/arduino_enc: info: Generating fake laser
data.. ]
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
[ 04/12/13 14:52:28.112 hockey-kubuntu/arduino_enc: debug: LaserScanner2dIface:
Sending data: 04/12/13 14:52:28.112 LaserData [181 elements]
minRange=0m
maxRange=0m
fieldOfView=0deg
startAngle=0deg
}
[ 04/12/13 14:52:28.112 hockey-kubuntu/arduino_enc: debug: MainLoop: Read laser
data: 04/12/13 14:52:28.112 LaserData [181 elements]
minRange=0m
maxRange=0m
fieldOfView=0deg
startAngle=0deg
}
[ 04/12/13 14:52:28.112 hockey-kubuntu/arduino_enc: info: Generating fake laser
```

Figure 6 Screenshot of Motor Encoder Driver

### 5.3 Evaluation

It is hard to give out a quantitative evaluation of this project, since the work is mostly software development rather than optimization of some problems. As a development project, I finally made this project integrated into the system, so it should be considered as a confirmation of our previous work.

If we want to deviate this project into a border area, I feel it is possible in three dimensions.

First, currently we have only GPS drivers for VN200 and Novatel. Gained experience form the development of VN200, it would be much easier to develop driver wrappers for other GPS sensors possibly not for engineering development, such as the sensors on smartphones. As we talked about the sensor selection before, it is much more difficult to make a driver for those sensors, because they do not provide the full access to all the interfaces and functionalities. We have to do some hacking in order to make a new driver, but this method is not scalable both for more inexperienced users and more smartphone models. If all smartphone vendors can make an agreement on their interfaces, it would be great for many software developers.

Second, a fully-functioning embedded system needs many components, not just GPS sensors and motor encoders, but also radar sensors, infrared sensors, motion controlling components, etc. For

a better communication and real-time controlling, embedded subsystems like Orca are designed. Usually, hardware vendors might not write a specific interface for such subsystems. For utilizing this hardware components, we have to write some wrapper or even driver if there is no driver for the base operating system. The work for these components is not quite different from this project. After getting a understanding of the hardware specification and the communication protocol, it is not hard to make a component work in a similar way.

Third, there are many communication middleware platforms like Orca. One example is RC2. In this project, we have only done some work in the Orca system. Under other platforms, the work does not change a lot. If we spend some time on the communication protocol of these new platforms, doing the same thing under a new platform will not take much effort as this project, since we did not have experience before.

In short, this project is successful and it gives me valuable experience to do similar development under other platforms or for other components.

## 6 Conclusion

I learned quite a lot from this capstone project. In the technical part, I learned how to develop drivers and applications in an embedded system like robot. The layout of development is different from traditional programming for generic computers. In the Orca system, we could utilize some standard interfaces for inter-process communication, as well as real-time control, which saves us quite a lot time to design different behaviours for each application to communicate. In the teamwork part, as a team, we five people did much work together and helped each other with personal expertise and skills. I learned a lot about how to communicate and cooperate during work, which is very helpful for my future career.

From this project, we now have a clear understanding of the Orca framework and partially finished this driving system. Additional work may include: testing in real traffic and in a larger scale rather than an experimental platform.

We have shown that the feasibility of a fully autonomous driving system with our platform. Our platform only uses low-cost sensors, which is different from advanced platforms like the Google Car. It is obvious that we have a much lower cost so that it is more attractive to consumers. But these low-cost sensors cannot give a high definition so that we need redundancy to make sure of robustness. This implementation will definitely increase the complexity of the system. With further development of this system into real scenario, it is possible that the cost of software development and integration would expand. To avoid this problem as much as possible, we have to make a well-modularized system so that adding/removing components would not lead to a great change of the system.

If this system is finally developed into a ready-to-use stage, the area of applications is very wide. For example, senior and disabled people might feel hard or even unable to drive by themselves, so that they want this autonomous car to help them to move around. Also, commuters may not want to waste time for driving, thus our system can free them to do other jobs meaningful to them. Moreover, it would change the way of transportation – there would be no human drivers on buses and taxis. People can get a car to move and do not care about parking spaces. These are just several examples. With the development of technology, the concept of “driving” will be finally changed by autonomous driving systems, and then we will have a more efficient world.

## References

- [1] Plunkett Research. 2012. Automobiles and Truck Trends.  
<http://www.plunkettresearch.com/automobiles-trucks-market-research/industry-trends>
- [2] Nissan Corporate. 2013. Nissan Technology.  
<http://www.nissan-global.com/JP/TECHNOLOGY/>
- [3] Nissan Corporate. 2007. Intelligent Pedal.  
<http://www.nissan-global.com/JP/TECHNOLOGY/OVERVIEW/dcas.html>
- [4] A. Eidehall, J. Pohl, F. Gustafsson and J. Ekmark. 2007. Toward autonomous collision avoidance by steering. *Intelligent Transportation Systems*.
- [5] U. Franke, D. Gavrilu, S. Gorzig, F. Lindner, F. Puetzold and C. Wohler. 1998. Autonomous driving goes downtown. *Intelligent Systems and Their Applications*.
- [6] P. Fernandes and U. Nunes. 2010. Platooning of autonomous vehicles with intervehicle communications in SUMO traffic simulator. *Intelligent Transportation Systems*.
- [7] L. Han, Q. Do, C. Guo and S. Mita. 2011. Optimal motion generation for autonomous vehicle in maze-like environment. *Intelligent Transportation Systems*.
- [8] K. Bullis. 2011. How Vehicle Automation Will Cut Fuel Consumption. *MIT Technology Review*.
- [9] T. Au, M. Quinlan, and P. Stone. 2012. Setpoint Scheduling for Autonomous Vehicle Controllers. *IEEE International Conference on Robotics and Automation*.
- [10] H. Horatiu. 2010. BMW Remote Controlled Parking. *BMW Blog*.
- [11] M. Arth. 2010. Democracy and the Common Wealth: Breaking the Stranglehold of the Special Interests. *Golden Apples Media*.
- [12] U. Kiencke and L. Nielsen, 2000. *Automotive Control Systems: For Engine, Driveline, and Vehicle*. *Measurement Science and Technology*.

- [13] F. Dellaert, W. Burgard, D. Fox and S. Thrun. 1999. Using the CONDENSATION Algorithm for Robust, Vision-based Mobile Robot Localization. Proceedings of IEEE International Conference on Computer Vision and Pattern Recognition.
- [14] S. Thrun, D. Fox, W. Burgard and F. Dellaert. 2001. Robust Monte Carlo localization for mobile robots. Artificial Intelligence.
- [15] KMPG. 2012. Self-driving Cars: The Next Revolution.  
<https://www.kpmg.com/US/en/IssuesAndInsights/ArticlesPublications/Documents/self-driving-cars-next-revolution.pdf>
- [16] S. Thrun. 2012. Google's Driverless Car.
- [17] S. Thrun, M. Montemerlo and J. Becker. 2008. Junior: The Stanford Entry in the Urban Challenge. Journal of Field Robotics.
- [18] D. Lavrinc. 2012. Exclusive: Google Expands Its Autonomous Fleet with Hybrid Lexus RX450h. Wired.
- [19] A. Michael, S. Santhikrishna. 2012. Google Driverless Car.
- [20] J. Markoff. 2010. Google Cars Drive Themselves, in Traffic. The New York Times.
- [21] Stanford Racing Team. 2007. Race Log.  
<http://cs.stanford.edu/group/roadrunner/racelog.html>
- [22] Nissan Corporate. 2010. Moving Object Detection.  
<http://www.nissan-global.com/JP/TECHNOLOGY/OVERVIEW/mod.html>
- [23] Nissan Corporate. 2009. Intelligent Transport Systems.  
[http://www.nissan-global.com/JP/TECHNOLOGY/OVERVIEW/its\\_dsss.html](http://www.nissan-global.com/JP/TECHNOLOGY/OVERVIEW/its_dsss.html)
- [24] M. Turk, D. Morgenthaler, K. Gremban and M. Marra. 1988. VITS-A Vision System for Autonomous Land Vehicle Navigation. IEEE Transactions on Pattern Analysis and Machine Intelligence.

- [25] R. Sukthankar. 1992. Raccoon: A Real-Time Autonomous Car Chaser Operating Optimally at Night. CMU-RI-TR-92-13, Carnegie-Mellon University Pittsburgh PA Robotics Inst.
- [26] Y. Yim and S. Oh. 2003. Three-Feature Based Automatic Lane Detection Algorithm (TFALDA) for Autonomous Driving. IEEE Transactions on Intelligent Transportation Systems.
- [27] T. Li, S. Chang and Y. Chen. 2003. Implementation of Human-Like Driving Skills by Autonomous Fuzzy Behavior Control on an FPGA-based Car-like Mobile Robot. IEEE Transactions on Industrial Electronics.
- [28] Q. Chen, U. Ozguner and K. Redmill. 2004. Ohio State University at the 2004 DARPA Grand Challenge: Developing a Completely Autonomous Vehicle. Intelligent Systems.
- [29] J. Rojo, R. Rojas, K. Gunnarsson, M. Simon, F. Wiesel, F. Ruff and D. Droeschel. 2007. Spirit of Berlin: An Autonomous Car for the DARPA Urban Challenge-Hardware and Software Architecture. Technical semifinalist paper of DARPA Urban Challenge.
- [30] Orca Robotics. 2009. Orca: Components for Robotics.  
<http://orca-robotics.sourceforge.net/>