VSmart: Design and Implementation of Dedicate Short-Range Communizations Testbed

Xuepeng Xu

Master of Science

Department

McGill University

Montreal,Quebec

2015-04-15

A thesis submitted to McGill University in partial fulfillment of the requirements of the degree of Master of Science

DEDICATION

This thesis is dedicated to the DSRC team of Cyber-Physical Systems lab in McGill University.

ACKNOWLEDGEMENTS

At first, I want to give my sincere gratitude to my supervisor, Professor Xue Liu for his great support on my thesis. I benefit a lot from his supervision and guidance as well as his profound knowledge. I am also grateful to all the members in Cyber-Physical Systems lab, especially Xi Chen, Linghe Kong and Qiao Xiang. Our discussions and experiments bring me the inspiration in my thesis. I would give my special thanks to Xi Chen, who help me review and proofread my thesis. Finally, I want to give my thanks to my family for their tremendous support on my study.

ABSTRACT

Dedicated Short-Range Communications (DSRC) technology is one of the most promising vehicular technologies to enhance road safety and traffic efficiency. The research and development of the DSRC relies heavily on tests and experiments, which require the support of a configurable testbed. VSmart testbed is designed for DSRC experiments in both indoor and outdoor scenarios. It utilizes Software Defined Radios (SDR), laptops and robots to emulate the real-world traffic. In addition, it provides user-friendly human interface to manipulate the testbed. There are three operating modes in the VSmart, simulation mode, stationary mode and full mode. The user can choose one of the modes to conduct the experiment based on available devices and experiment requirements. Detailed design and implementations of VSmart testbed are discussed in this thesis. Several use cases are provided to verify and demonstrate the usability of VSmart testbed.

ABRÉGÉ

La technologie des Communications dédiées à courte portée (Dedicated Short-Range Communications en anglais, ou DSRC) est une des technologies les plus prometteuses pour améliorer la sécurité routière ainsi que pour réduire le trafic. La recherche et le développement du DSRC reposent sur l'expérimentation, ce qui requière le support d'un banc de test configurable. Le banc de test VSmart est conçu pour des expérimentations de DSRC dans des scénarios intérieurs ainsi qu'extérieurs. It utilise des radios logicielles (Software Defined Radio en anglais, ou SDR), des ordinateurs portables et des robots pour simuler les trafics dans le monde réel. En plus, il a une interface conviviale et facile à utiliser et permet d'utiliser le banc de test sans difficulté. Dans VSmart, il y a trois modes d'utilisation: le mode simulateur, le mode stationnaire et le mode complet. Pour conduire son expérience, l'utilisateur peut choisir le mode qui lui convient, en se basant sur la disponibilité d'équipements et les exigences du test. La conception détaillée ainsi que l'implémentation du banc de test VSmart sont sujets de thèse. Certains cas d'utilisation sont fournis pour clarifier et démontrer l'utilité du VSmart.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
Introduction

1.1 Overview

According to the statistics from World Health Organization (WHO), the total number of global traffic fatalities is around 1.24 million per year [33]. Even in developed countries such as the United States and Canada, the number of vehicle crashes remains unacceptable. The Traffic Safety Facts Research Note by the U.S. Department of Transportation (DOT) shows that, the number of deaths and injuries caused by vehicle crashes are around 33 thousands and 2.3 millions respectively every year [40]. The total victims in Canada are also more than 166 thousands annually [7]. In order to avoid these tragedies, governments and the automotive industry have been actively developing new vehicular technologies. Among them, the Dedicated Short-Range Communication (DSRC) technology is one of the most promising ones.

DSRC is now under active development and deployment. In North America, U.S. DOT leads the Vehicle Safety Communications (VSC) project, in partnership with many vehicle manufactures and research groups, such as BMW of North America, LLC, General Motors Corporation, Toyota Technical Centre USA Inc., etc., to design and evaluate the DSRC standards [10]. U.S. DOT estimated that vehicle-to-vehicle (V2V) communication based on DSRC could address up to 82% of all crashes in U.S., which can save thousands of lives and billions of dollars [25].

Table 1–1: DSRC Applications[10]

Safety application	Non-safety application
Intersection Collision Avoidance	Traffic Management
Public Safety	Tolling
Sign Extension	Information from Other Vehicles
Vehicle Diagnostics and Maintenance	Other Potential Applications
Information from Other Vehicles	

The primary motivation to deploy DSRC is to enhance traffic safety. A vehicle equipped with DSRC devices can communicate with its neighbouring vehicles and roadside units (RSUs). The DSRC technology enables a communication range up to 1000 meters. A DSRC-equipped vehicle broadcasts safety messages multiple times per second. A safety message conveys basic information of the broadcasting vehicle, such as location, speed, and acceleration. Neighbouring vehicles can use the received messages to predict the collision threats. Besides, the RSU also distributes messages to inform the vehicles of the current geometry of the road, the state of the intersection and the other emergency situations. The exchanged messages between DSRC devices should follow DSRC standards. However, it is the responsibility of manufactures to implement the standards and the warning system.

In addition to safety messages, RSUs also deliver non-safety messages, which provide services, such as navigation, parking, and infotainment. Table 1-1 shows the different categories of each type of the applications [10].

The safety applications focus on the collision prevention. In [2], the U.S. DOT provides a list of top crash imminent scenarios:

- Lead Vehicle Stopped
- Control Loss without Prior Vehicle Action

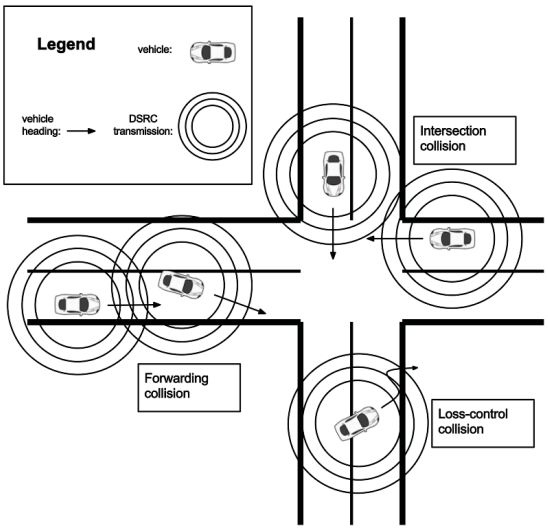


Figure 1–1: DSRC Collision

- Vehicle(s) Turning at Non-Signalized Junctions
- Straight Crossing Paths at Non-Signalized Junctions
- Lead Vehicle Decelerating
- Vehicle(s) Not Making a Maneuver - Opposite Direction
- Vehicle(s) Changing Lanes - Same Direction
- LTAP/OD at Non-Signalized Junctions

The above scenarios can be concluded into the three types of collisions: intersection collision, forwarding collision and loss-control collision. Figure 1-1 illustrates DSRC collision avoidance in the scenarios.

In order to test and evaluate the V2V and vehicle-to-infrastructure(V2I) applications, U.S. DOT deployed a testbed in Oakland County, Michigan. The Michigan testbed currently covers an area of 194 km^2 including 121 km of roadways [35]. The testbed allow researchers to conduct their vehicular onboard tests in the real world

Figure 1–2: Connected Vehicle testbed in Michigan[31]

scenarios. However, beforehand, prototyping and small scale tests are compulsory in the research and development path.

In fact, it is not practical to make a vehicular onboard tests for the group, which works on only one or a few aspects of DSRC research. Deploying a DSRC system to a vehicle costs not only money but also the efforts. On the other hand, some research groups are geographically far away from the testbed. It is difficult for them to utilize the infrastructure. Thus, a small scale and portable testbed can improve the DSRC research progress. To this end, this thesis designs and implements a testbed named VSmart.

1.2 VSmart Testbed

VSmart testbed is a small scale V2V and V2I testbed, which can be used to emulate the scenarios in the real-world traffic. It is easy to deploy and cost-effective. The testbed has two different units, vehicle unit and infrastructure unit. A vehicle unit is composed of a software-defined radio (SDR), a computer and a robot, while

an infrastructure unit is made up by a SDR and a computer. In the testbed, the SDR acts as communication module and the computer is the center processor. A robot in the vehicle unit is the executor, which is able to emulate a moving car. The testbed can be deployed in the indoor/outdoor environment whose area is between 25 m^2 to 100 m^2. The cost of a vehicle unit and an infrastructure unit are around 2,000 US dollars and 1,500 US dollars respectively. Compared with the real world testbed, the cost of the VSmart testbed is acceptable for most of academic groups.

In addition, the VSmart testbed is based on Software Define Radio (SDR), which makes the platform reconfigurable and programmable. Applications, algorithms and even standards can be easily implemented and deployed. Hence, the VSmart testbed has high reusability and extensibility. It reduces the efforts and time to develop a DSRC testing platform and helps users focus on the research.

1.3 Thesis Contributions

There are four main contributions of this thesis:

- The thesis proposes the design of the VSmart testbed and elaborates the implementation of the platform. It covers the description of both software and hardware design in detail.
- The thesis describes the extensibility of the VSmart testbed. It allows users to customize the functionalities.
- The thesis also introduces two use cases, collision avoidance and cooperative adaptive cruise control, to verify the usability of the VSmart testbed.
- The thesis provides a list of future works for the VSmart testbed.

1.4 Thesis Structure

The rest of the thesis will be organized as follows. Chapter 2 introduces the related works. It includes the overview of DSRC standards and an introduction of SDR. Chapter 3 is the architecture overview of the VSmart testbed. The software design is elaborated in Chapter 4. Chapter 5 covers two use cases of DSRC application. The thesis is concluded in Chapter 6.

CHAPTER 2
Background

2.1 The Overview of DSRC Standards

DSRC relies on standard-based interoperability among the devices by different manufactures. The implementations have to follow the protocols defined by standard institutes. It ensures that the equipment from different suppliers can communicate without gap.

DSRC technology has been developed worldwide for many years and several standards have been created in different regions, such as Europe, Japan and the United States. The standards are all based on IEEE 802.11p and the designs are similar. The thesis mainly focuses on the design of U.S. DSRC architecture, since the VSmart testbed is standard-independent. Note that the DSRC mentioned in the rest of thesis is in the U.S. standards.

There are four layers in the DSRC architecture, physical layer (PHY), data link layer or media access control layer (MAC), Network/Transportation layer and Application layer.

As illustrated in the Figure 2-1, PHY layer protocol and MAC layer protocol are defined in IEEE 802.11p Wireless Access in Vehicular Environments (WAVE) [18], which is an amendment to the IEEE 802.11 standard. Similar to IEEE 802.11a, the WAVE 802.11p utilized the Orthogonal Frequency Division Multiplexing (OFDM) modulation technique. In DSRC architecture, PHY layer and MAC layer compose

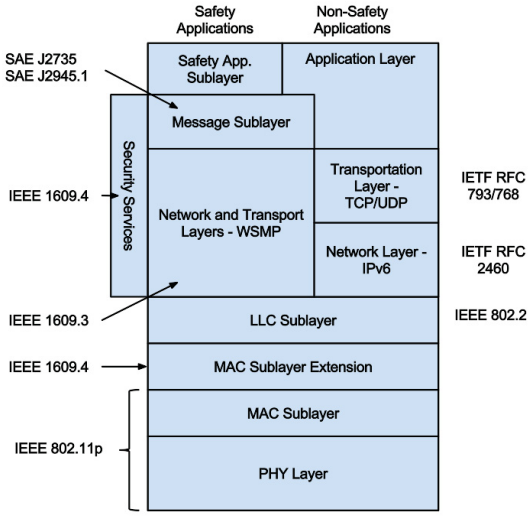


Figure 2–1: DSRC Architecture[25]

the bottom stack, which provides data exchange functionality to the middle stack. In order to accommodate the wireless communication to high-speed vehicles, three new features are added to WAVE protocols as extensions to the IEEE 802.11. First of all, a management frame for timing advertisement is added to synchronize the time reference. Secondly, receiver performance is improved by channel rejection requirements enhancement. Finally, to reduce interference, new channels are defined in the protocol. The WAVE 802.11p uses channels of 10MHz bandwidth in 5.9GHz band (5.850-5.925GHz).

The middle stack is Network/Transportation layer. This layer employs IEEE 1609, a family of standards for WAVE [32]. In the standards, IEEE 1609.2 defines secure message formats and processing, while IEEE 1609.3 defines network and transportation layer services to support the secure data exchange, and IEEE

1609.4 enhances IEEE 802.11 Media Access Control (MAC) to support multichannel WAVE operations. IEEE 1609.3 includes the WAVE Short Message Protocol (WSMP), which covers addressing and routing for the network layer. DSRC, on the other hand, can support TCP/IP protocol as well. Thus, an application can choose to use IEEE 1609 or TCP/IP protocol. However, since WSMP is bandwidth-efficient, it should be used in the safety applications. For non-safety applications, TCP/IP protocol may be a better choice because of its routing capability [25].

In the top layer, SAE J2735 DSRC Message Set Dictionary standard [22] is designed for the V2V and V2I applications. The protocol defines the data formats, which include message sets, data frames and data elements. It can be used in the applications such as vehicle safety, emergency vehicle notification, automated tolling, enhanced navigation and etc. Among the message set, Basic Safety Message (BSM), Roadside Alert (RSA) and Probe Vehicle Message (PVM) are the most typical. BSM conveys the basic vehicle information including location, speed, heading, brake status, vehicle size and etc. DSRC-equipped vehicle broadcasted the BSM 10 times per second. RSA is the message that sent from RSU to the vehicle to inform the users about the emergency operations. PVM contains the data gathered by the vehicle and transmitted to RSU. It includes traffic condition, weather and road surface condition. Traffic Management Center (TMC) and information service provider will collect these data. SAE J2945.1 specifies the minimum communication performance requirements of DSRC message sets [25]. It covers the topics such as BSM transmission rate and power, accuracy of data elements and channel congestion control. Besides, the

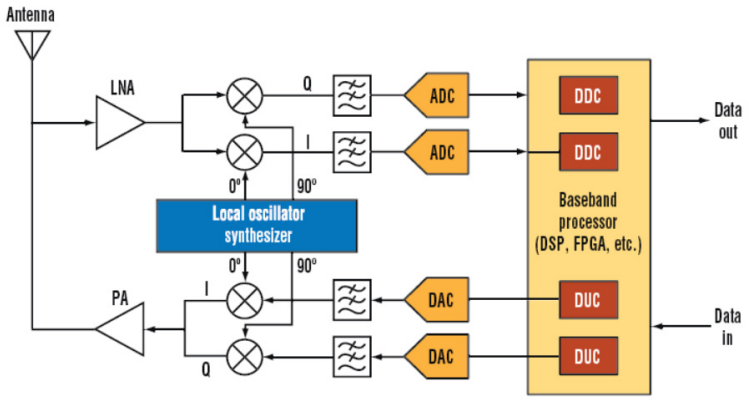


Figure 2–2: SDR Architecture[11]

application can utilize TCP/IP to define customized message like most of Internet applications.

2.2 Software Defined Radio

Software Defined Radio is the radio in which some or all of the physical layer functions are software defined [16]. A regular radio communication system implements the functions directly in hardware. In such a system, a special-purpose processor does the signal processing. Therefore, it has only limited functionalities. In opposite, SDR uses configurable components in the hardware design and leaves the definition of functions to software. It utilizes the general-purpose processor, such as Central Processing Unit (CPU) in personal computer (PC) , Microcontroller (MCU) in an embedded system, to process the signal. Figure 2-2 depicts a typical SDR architecture.

In the SDR architecture, an analog-to-digital converter (ADC) converts the analog signal, which is gathered from antenna, to the digital signal and hands over to a general-purpose processor. The rest of processing is defined in software. On the

Table 2–1: USRP product list [37]

Product	Frequency range	Real time bandwidth	Host Interface	Base price (US $)
B200	70 MHz-6 GHz	56 MS/s	USB 3.0	$675
B210	70 MHz-6 GHz	56 MS/s	USB 3.0	$1100
N200	DC-6 GHz	50 MS/s	Gigabit Ethernet	$1515
N210	DC-6 GHz	50 MS/s	Gigabit Ethernet	$1717
X300	DC-6 GHz	up to 200 MS/s	GbE, 10 GbE, PCIe	$3900
X310	DC-6 GHz	up to 200 MS/s	GbE, 10 GbE, PCIe	$4800

contrary, the software-defined functions can generate data and deliver to antenna by digital-to-analog converter (DAC) [28]. The architecture significantly improves the hardware reusability of the radio system and ensures that the DSR system can support varieties of applications.

The SDR developed rapidly in the last two decades. There are many commercial off-the-shelf (COTS) SDR products today. Among them, Universal Software Radio Peripheral (USRP) family is a range of the most competitive SDR products. It is designed and sold by Ettus Research. Table 2-1 is a list of main products in USRP family [37]. The implementation in the thesis uses the USRP B210.

In USRP, the baseband processor, as shown in Figure 2-2, consists of a millions gate-field programmable gate array (FPGA) and a programmable USB controller [5]. The baseband processor interacts with PC, in which, a software application is able to handle the data. Hence, to fully use the USRP board, the FPGA need to be programmed and the application on the PC side need to be developed. However, programming the FPGA takes a bit of skill, and mistakes can fry the board permanently [5]. On the other hand, it costs a lot of efforts to design and implement a

SDR application from scratch. GNU Radio takes care of both of the difficulties in the development.

2.3 GNU Radio

GNU Radio is a free software toolkit for building software radios [5]. It is an official GNU Project and funded by Eric Blossom. GNU Radio aims to create SDR application with external RF hardware or simulation-like environment. A GNU Radio application is a flow graph, which consists of numbers of digital signal processing (DSP) blocks. The signal in the application is treated as data flow, which flows through the connected blocks for processing. The block in the application can be implemented by using C++ or Python programming language. GNU Radio utilizes Simplified Wrapper and Interface Generator (SWIG) tool to integrate C++ and Python. Figure 2-3 shows a typical USRP transceiver flow graph.

With the help of GNU Radio, lots of wireless communication applications can be easily implemented. For example, GNU Radio application can implement IEEE 802.11 standards to make the USRP work as WiFi station; it can also implement GNSS receivers by following the GPS specification. In this thesis, GNU Radio is utilized to create the VSmart testbed.

GNU Radio can support a wide range of COTS SDR platforms besides USRP. These hardware platforms include Fairwaves UmTRX, Funcube Dongle, Great Scott Gadgets HackRF, Microtelecom Perseus, etc. [36]. It provides a various of hardware options to build the VSmart testbed. According to Eric Blossom in [5], the USRP is the preferred solutions. However, it depends on the DSRC research topic. An

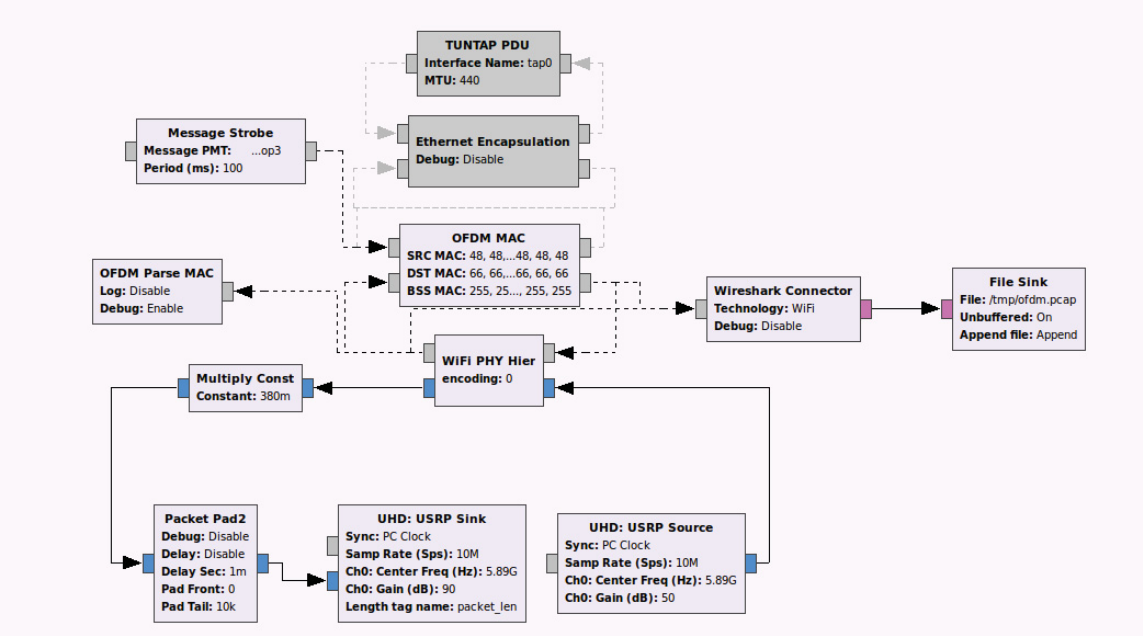


Figure 2–3: USRP transceiver flow graph in GNU Radio [4]

expensive hardware system can provide high quality communication performance, while a cheap one can reduce the cost of the research.

2.4 Related Work

In order to test the effectiveness and performance of DSRC application, the simulation or emulation work has to be performed. Many previous works focus on the simulation. Thus, a great deal of traffic simulation tools are designed and implemented [14, 9, 34, 15]. Most of them utilized NS-2, NS-3 or other network simulation tools. However, DSRC system is a platform works with hardware and real world scenarios. The hardware implementation is always less accurate than the simulation. Hence, the simulation tools are not enough to evaluate the DSRC performance, and testbed and prototype become necessary in the research experiments.

In the recent years, a few DSRC hardware testbed were proposed [29, 41, 13, 26, 42, 1]. Among them, NEC C2X [29] and Cohda MK2 [41] are two mature products, but they are bound to hardware constraints and not suitable for research, which may need dynamic modification of the protocols. On the contrary, [13] and [1] implement the testbed with SDR, since it is reconfigurable. In [13], the authors demonstrate that, in the algorithm evolution, to be more abstract means to be more accurate and vice versa. Thus, they proposed a hardware testbed which adopts Xilinx XtremeDSP-II kit as hardware platform and Mathworks Matlab as well as Simulink to implement the software models. In [1], the authors built their DSRC prototype on OpenAirInterface platform and introduced their software architecture. There are three blocks in their architecture, Linux 802.11 subsystem, IEEE 802.11p driver and IEEE 802.11p software-modem. Linux 802.11p subsystem contains an extension

for IEEE 802.11p. IEEE 802.11p driver bridges the Linux 802.11 subsystem and hardware. The soft-modem is connected the hardware via an IOCTL link and a dedicated OpenAirInterface driver.

However, the DSRC testbeds presented above are all about DSCR device itself. In the real world testing, the device has to be deployed in a vehicle and connected with the warning system. The testbed proposed in this thesis contains the entire DSRC system. Namely, the VSmart testbed framework covers the design from PHY layer to application layer. In the testbed, the USRP works as DSRC device, the robot acts as a vehicle and the PC is the central processor. The researchers can conduct the entire DSRC experiments in the laboratory with the help of VSmart testbed. It is intuitive to use and easy to extend. A new application or a new algorithm can be implemented in the platform with minimal efforts.

CHAPTER 3
Overview of VSmart

3.1 The Overview of the Architecture

The VSmart is composed of two types of units, vehicle units and infrastructure units. The vehicle unit has three subsystems, a *transceiver*, a *processor* and an *executor*, and the infrastructure unit consists of a *transceiver* and a *processor*. The *transceiver* is a SDR-based DSRC transceiver and has identical implementation in both units. It provides wireless communication ability necessary to support the functioning to the other subsystems. The *processor* in vehicle unit has three main responsibilities:

1. It generates the basic vehicle messages to report the vehicle information.
2. It handles the received messages for emergency actions.
3. It is responsible to control the *executor* and the *transceiver*.

The *executor* is the driver of the robot. It controls the actuators of the robot to fulfill the movement commands. The *processor* of the infrastructure unit probes the message from vehicle units and handles the message for monitoring and measuring purposes. In addition, it can send commands or information to the vehicle unit. Figure 3-1 outlines an example of the architecture of VSmart.

As a DSRC testbed, the communication structure in VSmart has four layers and can be divided into three stacks. In the bottom stack, PHY layer and MAC layer are

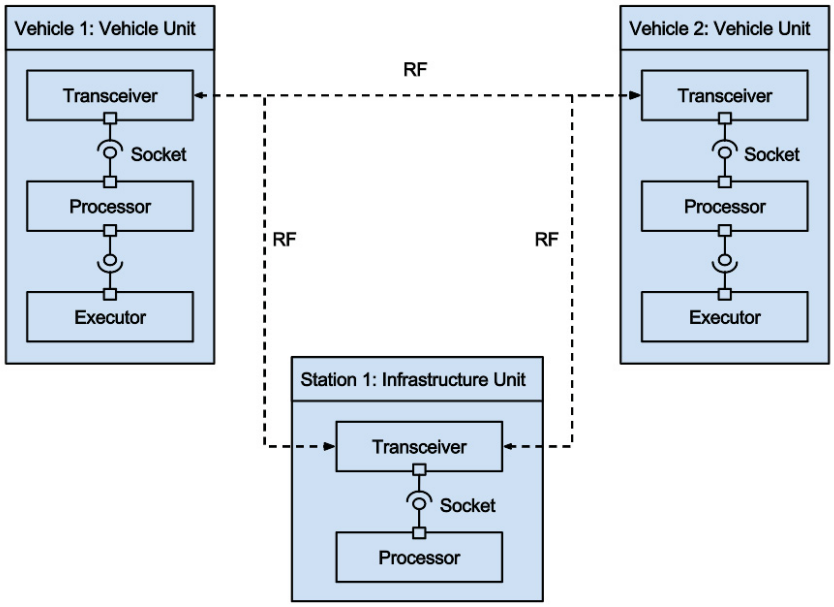


Figure 3–1: Testbed components

Table 3–1: VSmart testbed hardware list

Component	Hardware and model
Software Define Radios	USRP B210
Computers and laptops	Inspiron 660, Thinkpad X60s, Acer V5-171
Robots	iRobot Create

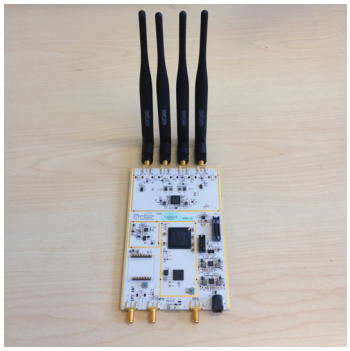
implemented in *transceiver*. The middle stack consisting of network layer, combined with top stack, which is the application layer, are all encapsulated in the *processor*.

3.2 Hardware setup

In VSmart, computers and laptops act as the brain for both vehicle units and infrastructure units. The SDR and the robot are functioning as two external hardware components, which connect to the computers and laptops by the cables. They are driven by the unit systems to accomplish different tasks. SDR is dedicated to the wireless communications. It keeps sending the messages generated by the system in the wireless channel. The SDR is under control of the *transceiver*. The robot is responsible to carry the entire vehicle unit to perform the moving actions and it is driven by the *executor*. As long as the vehicle unit needs to finish a moving task, the *processor* creates a corresponding job and utilizes *executor* to actuate the robot. Table 3-1 and Figure 3-2 show the hardware devices used in VSmart testbed.

SDR is a circuit board with wireless communication capability, which allows the customized communication protocols and stacks to be developed and deployed. It provides the researchers a fair platform to compare different algorithms of scheduling, routing, rate adaptation and power control. In VSmart, the USRP B210 is adopted as the communication board. It is connected to the computer thorough USB 3.0. Furthermore, the USRP B210 is also powered by USB port, which enables the board

(a) iRobot Create



(b) USRP B210

(c) Laptop

Figure 3–2: VSmart Testbed Hardware

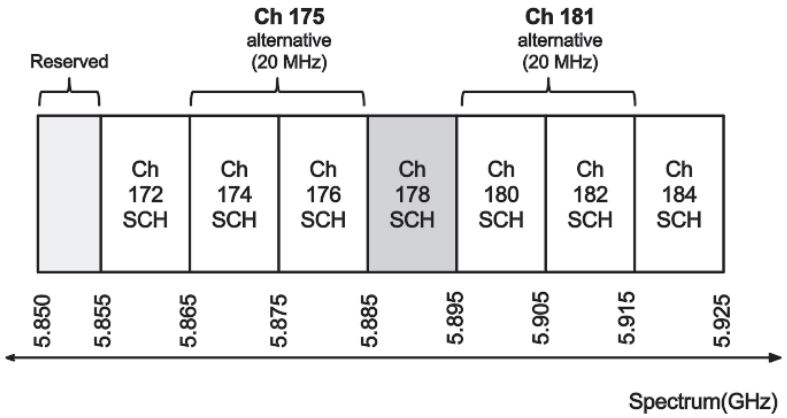


Figure 3–3: DSRC Band Plan channel designations [25]

to be carried around with robot freely. The basic setting for the USRP includes the frequency and sample rate. As described in the background, DSRC uses channels of 10MHz bandwidth in 5.9GHz band. The spectrum are divided into 7 10MHz channels and pairs of them can be combined into 20MHz channels. The channels, on the other hand, are designated either control channel (CCH) or service channels (SCH). Figure 3-3 illustrates the channel designations of DSRC [25]. Because of the limitation of the hardware, the VSmart testbed is not able to have access to Universal Coordinated Time (UTC). Hence, the channel switching is not implemented in the testbed and a fixed channel is assigned. In VSmart, the channel frequency is set to 5.89 GHz and the sample rate 200 KHz.

The options for the robot are not as much as SDR. Indeed, the selection of the robot is based on the experimental requirements. The basic configuration of VSmart includes the use of iRobot Create [24]. The iRobot Create has a serial port for the connection to the computer. The serial port is not convenient in the experiments with

laptops, since most of the laptops don't have the corresponding port. In VSmart, a converter is used to convert the serial port into USB port, which makes the robot more compatible with mainstream computers and laptops. The most important actuator in the iRobot Create is the wheel actuator. It enables the mobility of the vehicle unit. The maximum speed of the actuator can achieve $500mm/s$.

3.3 Modes

The VSmart testbed can work in three modes, simulation mode, stationary mode, and full mode. The simulation mode and stationary mode abandon the use of some subsystems and hardware devices. Table 3-2 shows the subsystems and devices used in each mode.

The simulation mode is used to simulate communications and movements of DSRC-enabled vehicles without any hardware support. Both infrastructure unit system and vehicle unit system are running in the same computer. In other words, the unit instances are running as processes in the computer and the *transceiver* is replaced by interprocess communication. The *executor* in the vehicle unit is virtual. The user can obtain the information of the vehicle as well as control the vehicle by using GUI applications in the infrastructure unit. The detail of these applications will be elaborated in Chapter 4. This mode is suitable for rapid tests and experiments of new vehicular communication technologies and protocols.

The difference between stationary mode and simulation mode is that the stationary mode utilizes the physical communication. Each unit has a SDR board for wireless communication and uses the *transceiver* to drive the SDR board. This mode

Table 3–2: The composition of different modes

Mode	Hardware	Subsystem
Simulation	Computer	*Processor*
Stationary	Computer, SDR	*Processor*, *Transceiver*
Full	Computer, SDR, Robot	*Processor*, *Transceiver*, *Executor*

is good for tests and experiments in real channels, but the movements of wireless nodes are neglected.

The full mode uses all the software and hardware components. In this mode, the vehicle unit has the property of mobility. The robot can move the entire system physically, which can change the transmission range between different units.

CHAPTER 4
Software Design

In full mode, each unit has two different processes, a transceiver process and a processor process. These two processes control the *transceiver* subsystem and the *processor* subsystem respectively. The connection between two processes utilizes the interprocess communication (IPC) technique. In the vehicle unit, the *executor* is also a part of processor process and the communication between the *processor* and the *executor* is function invocation. A *Processor* can explicitly control the robot via interfaces provided by the *executor* subsystem. Compared with full mode, the stationary mode only disables the *executor* while the rest keeps the same. In simulation mode, both *transceiver* and *executor* are disabled and the *processor* of different units are running in the same computer. In order to introduce the software design of VSmart concisely, this chapter mainly focuses on the full mode VSmart. However, it is easy to deduce the other two modes by removing the corresponding subsystems.

High modularity and reusability are the two basic requirements of software design in VSmart testbed. The DSRC technology is still in the research and early-deployment stage, and researchers may want to modify the protocols in their experiments. On the other hand, there are more than one DSRC standards, which may lead to different implementations of the bottom stack and the middle stack. However, the top layer functionalities should remain the same despite the change of the lower level protocols. Hence, it is important to separate different layers from

each other. The design in VSmart testbed modularizes the different layers and the functionalities to increase the reusability.

In practice, a researcher or an engineer may want to add new features to the testbed for his experiments. It requires the testbed to have very good extensibility. To meet the demand, the plugin design pattern is added to the testbed. It enables the extensibility of the new functionalities.

As mentioned in section 2.3, GNU Radio supports both C++ and Python languages. For a better consistency of the testbed, it is more elegant to use one of them as the primary language in the implementation. In this thesis, Python is selected as the main programming language because of its simplicity and short development cycle.

The rest of this chapter elaborates the software design of VSmart testbed.

4.1 Transceiver of vehicle unit and infrastructure unit

The subsystem *transceiver* is developed under GNU Radio development environment. It drives the USRP for wireless communications in the testbed. Figure 4-1 is a component diagram of *transceiver*. In the *transceiver*, the component *PHY and MAC module* implements IEEE 802.11 protocol for physical layer as well as MAC layer. In [4], Bastian Bloessl et al. provide an implementation of IEEE 802.11 a/g/p OFDM transceiver in GNU Radio. Besides, they release the open source project for Wi-Fi transceiver based on GNU Radio. This thesis applies their code and makes necessary modifications to meet the requirements of VSmart. Above the *PHY and MAC module*, there is a *controller* module. It is used to control the parameters, such as power, frequency, sample rate, and etc.. The modification of parameters is

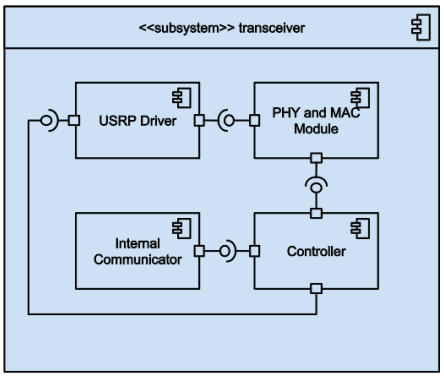


Figure 4–1: Transceiver Component Diagram

based on the request of the *processor* subsystem. *USRP Driver* is a built-in module in GUN Radio. It is responsible to control the USRP board, especially the wireless communication. The *internal communicator* is the implementation of the interprocess communication, which is discussed in section 4.2.

4.2 Interprocess communication in vehicle unit and infrastructure unit

Due to the multi-process property of the testbed, an IPC mechanism is required to exchange information between processes. There are many options for the IPC mechanisms, which may have different performance. It depends on the application to determine which IPC approach to adopt. The most frequently used IPC methods include pipe, signal, shared memory, message queue, and socket. Among them, the shared memory is the most efficient approach since the data exchange happens only in memory. However, in the VSmart testbed, the socket-based mechanism is employed because some experiments may require the infrastructure unit to provide services that work across different hosts, and only the socket-based mechanism can accomplish this job effectively and efficiently. Despite the detailed implementation of

IPC, the VSmart implements the IPC as a separate module, *internal communicator*. It can be replaced by any other implementation of IPC approach, which might be preferred in some other experiments. For instance, some experiments require IPC to be as fast as possible, and the *internal communicator* can be reimplemented using shared memory. As showed in both Figure 4-1 and Figure 4-2, the *internal communicator* exists in both *transceiver* and *processor*, so that transceiver process and processor process can exchange the message directly.

4.3 The processor of vehicle unit

Figure 4-2 depicts the design of the processor of vehicle unit. The design of *processor* utilizes several design patterns. At first, it uses a layer structure to handle the packets from the *transceiver*, which follows the four-layer structure in DSRC architecture. Secondly, the *processor* is also an event-driven system. It separates the communication layers from the top application functions. The incoming message is treated as a network event, which may trigger a series of operations in the unit system. In addition, a plugin design in the processor ensures that a new feature can be easily added. It significantly increases the reusability and extensibility of VSmart. In order to handle the job execution sequence issues, a job scheduler is introduced in the system. The scheduler utilizes batch-processing technique and interruptible job design. Besides the above design patterns, a self-positioning system is also designed to overcome the limitation in the hardware. The position of a vehicle unit can be self-evaluated with the help of the self-positioning system. The positioning calculation is based on the execution of the jobs. The following subsections will focus on the design of the *processor* in vehicle unit.

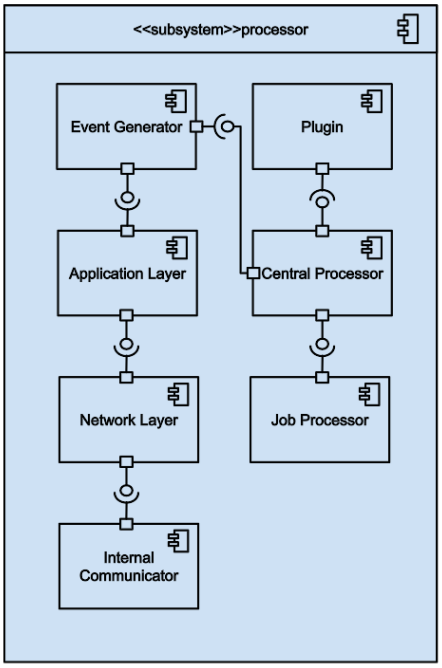


Figure 4–2: The processor components in vehicle unit

4.3.1 Layer Structure

As presented in the section 2.1, DSRC is enabled by a four-layer system. The VSmart testbed inherits the layer structure from DSRC. The *transceiver* covers the bottom two layers and the processor is responsible for the remaining two, network layer and application layer. In DSRC, network layer utilizes the WSMP and TCP/IP, and application layer employs SAE J2735. However, to minimize the efforts of development in the early deployment stage, the protocols are not fully implemented in VSmart.

In the current implementation, network layer uses the existing IP packet creation/parsing library [12] for the addressing and forwarding. The routing functions are not yet available. The application layer in the testbed adopts customized message formats. In the implementation, the messages are wrapped with JavaScript Object Notation (JSON) format and each message in the application layer can be viewed as a JSON object. Furthermore, users can define new message formats through the plugin design, which is covered in section 4.3.3.

4.3.2 Event-driven Design

Event-driven design is suitable for the layer-structured system. It allows a service to fire an event, which may activate the operations in another service. This design loosely coupled the software components. In the testbed, each layer can be regarded as a service. The lower layer can create an event and trigger the appropriate reaction in upper layer services. Thus, a combination of layer and event driven design pattern fit perfectly for the architecture of the testbed.

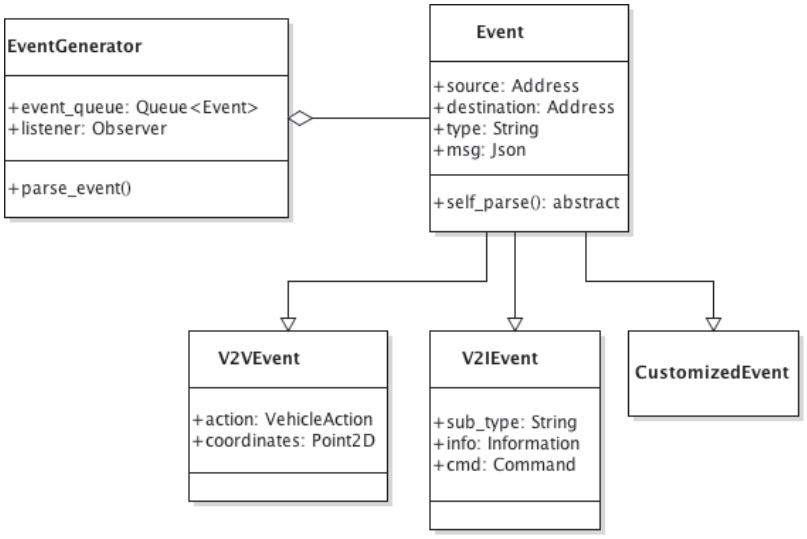


Figure 4–3: Class diagram of the event generator

In the system, an event generator module is built on top of the application layer. The module parses the messages from application layer and generates the corresponding events. The event will be then forwarded to the *central processor* module, as showed in Figure 4-2, for further disposition. The event-driven design adopts observer pattern as the event forwarding approaches. The *central processor* module acts as observer and the event generator is the subject.

In the object oriented programming (OOP), the event is designed as an abstract class. Every subclass matches up with a type of message. The subclass has to implement a *self_parse* method, which parses the messages from the application layer. Figure 4-3 is the class diagram of the event generator module.

The design of event-driven system decreases the coupling between observer and subject. As the observer, the *central processor* module has less knowledge about the

low level services. The modification of the bottom layers will not affect the *central processor*.

4.3.3 Plugin Design

In software architecture, a good design of the framework should allow users to add their functional modules. The plugin design enables a system to exploit new functionalities. To this end, the VSmart testbed is designed as an extensible framework. Users are encouraged to develop and deploy their own features and functionalities.

In VSmart, a customized plugin has three submodules, *customized executor*, *customized receiver* and *customized event*. In the *processor* subsystem, a background thread keeps invoking the same core procedures intermittently. These procedures are composed of the internal functions, such as message broadcasting and location updating, and external functions. The external functions are defined in the *customized executor* submodule, which is developed by users. As an event-driven system, the processing and generation of the event are the most elementary. An experiment platform should have the capability to customize the event processing and generation procedure. A customized receiver is able to handle the received event by a user-defined method. Similarly, users are also able to define a customized event. The customized event has to inherit the superclass event as described in Figure 4-3.

Every plugin module has to hook up with a corresponding configuration file, in which the name and the path of the submodules are specified. In addition, all the plugin modules are listed in the plugin initialization file, which guarantees that the plugin loader can load the modules appropriately at the runtime.

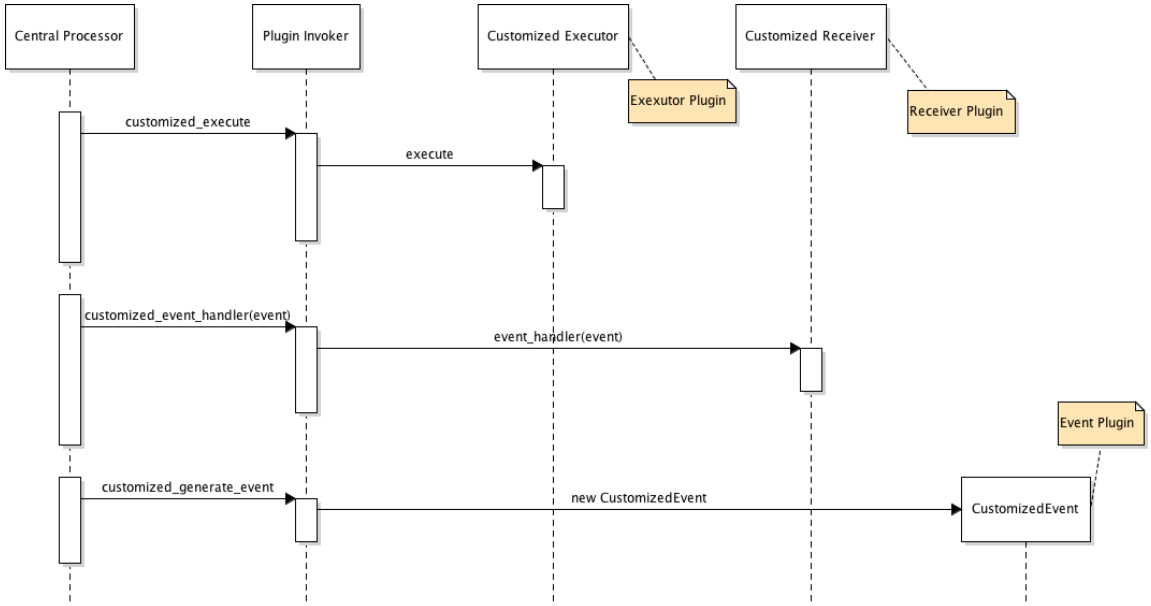


Figure 4–4: Sequence diagram of the plugin invocation procedures

Figure 4-4 depicts the sequence diagram of the invocation procedure in the plugin design. The plugin invoker plays a role of the intermediary interface. The *central processor* has very few knowledge about the implementation of customized submodules.

Based on this plugin design, many DSRC applications can be emulated by creating a new plugin module. For example, to emulate an elementary cooperative adaptive cruise control application, users can implement all three submodules in plugin design. The targeted vehicle needs to use *customized executor* to generate *customized event* messages. Upon receiving a *customized event* message from the targeted vehicle, the following vehicle utilizes *customized receiver* to follows target's action. Another example is the collision avoidance. The receiver can predict the

collision threat by calculating the relative distance between two vehicles when basic vehicle message is received. These examples will be elaborated in Chapter 5.

4.3.4 Job Scheduler

Interruptible Job. In the testbed, one of the most fundamental operations is to control the actuators, i.e., the robots. In the VSmart design, the *Job* class is designed to describe the robot's action execution. A *Job* has two important properties, *time*, which represents the duration of the execution, and *action*, which contains the two parameters, linear velocity and angular velocity, for the physical movement. During the execution, the parameters are sent to *executor* to drive the robot. In many scenarios, the robot movement has to be paused for some reason. For instance, in a collision avoidance scenario, the robot would be stopped if a collision threat is detected. The current job has to be suspended until the collision threat disappeared. Therefore, an interruptible design in the job scheduler is essential. To be interruptible, a job must have the capability of saving the current execution state. The most important states to save in the job are the remaining execution time and the parameters. It is easy to have the action parameters saved, but the remaining execution time has to be calculated. In fact, to actuate the robot, the driver only needs to send the command with speed and direction. The robot will execute the same command until the next received. Therefore, the execution time is managed by the *Job* class. In order to manage the execution time, the monitor design pattern [21] is utilized in the job execution. The technique is used in multithread synchronization. It allows the threads to have mutual exclusions and the ability to wait for a certain condition. When a job is executed, the job execution thread will wait

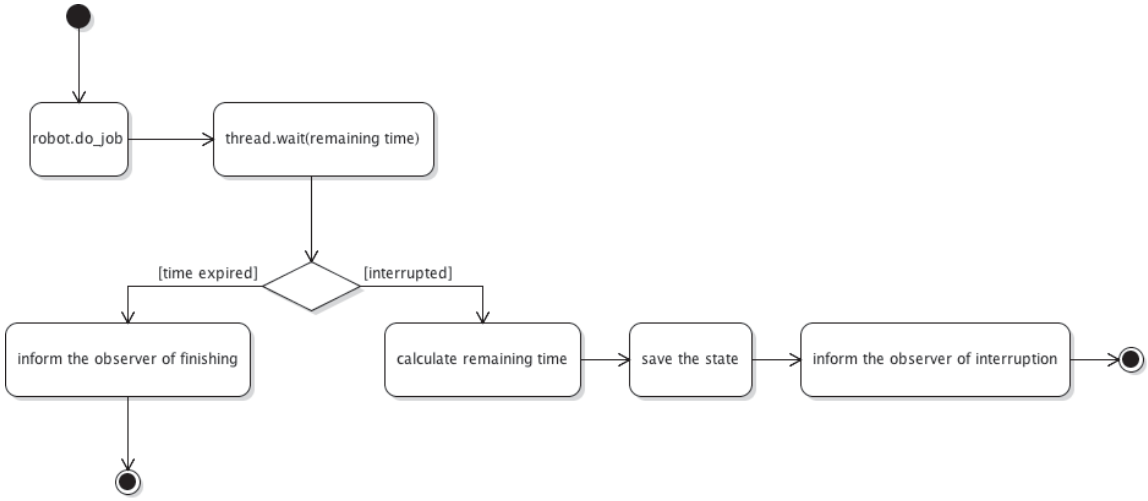


Figure 4–5: Activity diagram of job execution

until the execution time expired. To interrupt the execution, the waiting thread has to be waked up and calculate the executed time in order to save the current state of the job. In Python, the *Condition* in *threading* module is an implementation of monitor construct. Figure 4-5 is the activity diagram describing the procedure of job execution.

Job scheduler. The executor may need to process a series of jobs and each of them must be executed at a specific time. Thus, a scheduler is required to keep track of jobs and invoke them appropriately. This design is called scheduled task design pattern. It is used frequently in the real time system. There are many scheduling algorithms, but only a few fits the vehicle task scheduling. First in first served (FIFS) is the most straightforward approach. It is very similar to the batch processing. The jobs are kept in a queue and executed in sequence. Another available strategy is the multilevel queue. It can be used in path selection application. However, it may

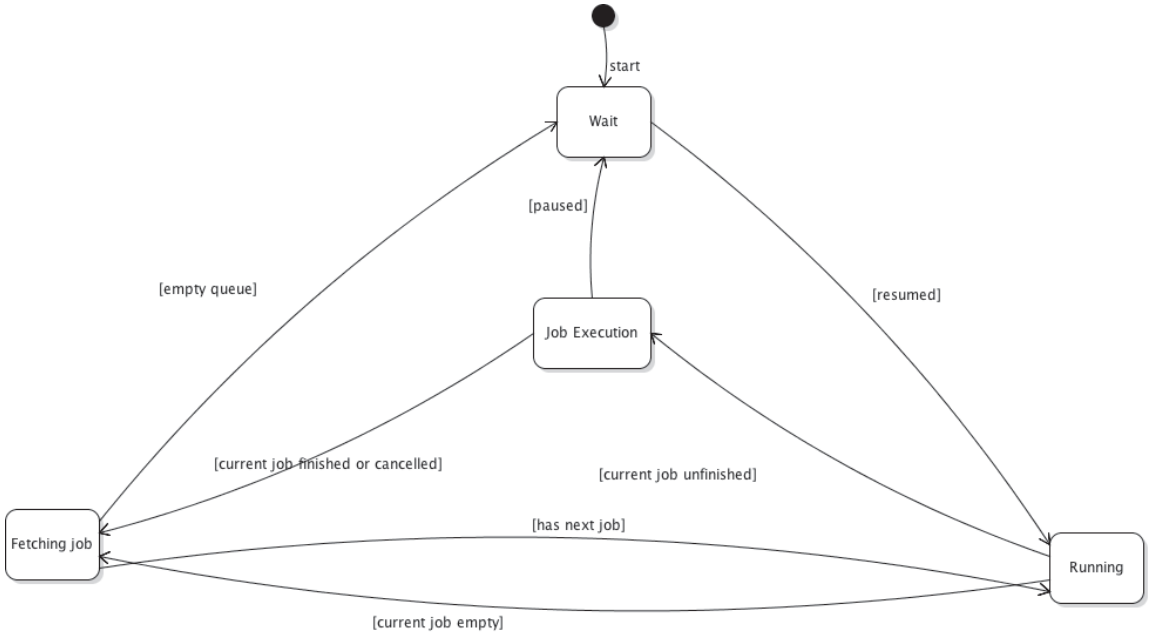


Figure 4–6: State diagram of job scheduler

take a lot of efforts to design the scheduler since it is more complex. In VSmart, the FIFS is adopted as the job scheduling strategy. To be able to schedule the jobs, the scheduler must be interruptible as well. The pause of scheduler should not only suspend the scheduling process but also the execution of the current job. In addition, the resuming of the scheduler should handle the current unfinished job first and then the rest. In the other hand, the scheduler also provides the job cancellation APIs. It can be used to cancel the current job or even clear the queue. Similar to the design of job execution, monitor pattern is also applicable in the interruption design of scheduler. Figure 4-6 is the state diagram of the job scheduler, where the job execution is described in Figure 4-5.

4.3.5 Self-positioning

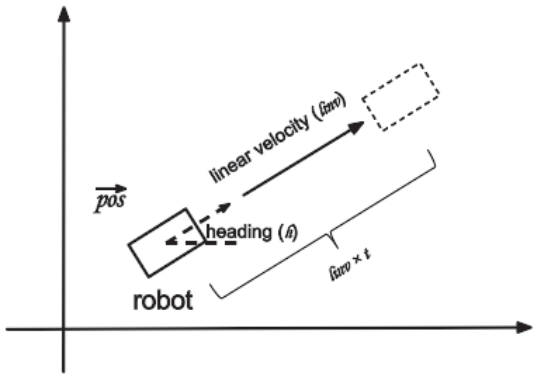
The primary purpose of the development of DSRC is to enhance safety and reduce collisions. In the collision avoidance applications, the key information is the location of the vehicles. It is used to predict the collision threats. In fact, most of DSRC safety applications and even some non-safety applications are location-based. For instance, in the approaching emergency vehicle warning application, the position of an emergency vehicle is reported to alert the driver; in the enhanced route guidance and navigation application, the location information is used in the navigation system. It is indisputable that the positioning module is crucial in the V2V system.

The VSmart testbed targets both indoor and outdoor environments. In the indoor environment, the satellite navigation system, such as Global Positioning System (GPS), Beidou Navigation Satellite System (BDS), is inapplicable. There are many accurate indoor positioning schemes, for instance, smartLOCUS [6], SpotON

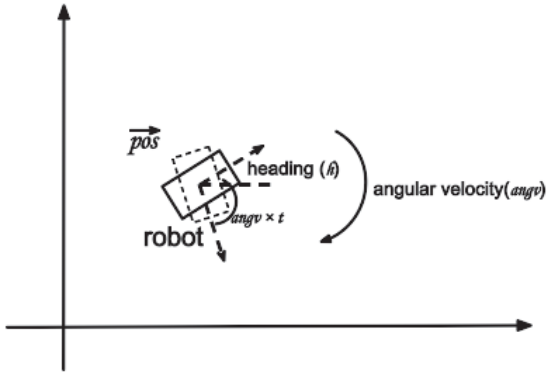
[20], LANDMARC [30], etc.. Unfortunately, they all require the assists of extra infrastructures. In other words, if the users of the testbed have the related hardware accessible, it is a good idea to utilize the above localization schemes as the positioning module in the testbed. The current implementation of positioning system is integrated within the processor of vehicle unit. It can be modified to invoke existing localization schemes by overwriting the functions *update_primary* and *update_secondary*.

In this thesis, a self-positioning scheme is designed to locate the vehicle unit. Indeed, many state-of-the-art self-positioning schemes have been proposed [3, 8, 19]. The schemes take advantage of the target's movements and locate it by calculating the moving direction and distance. In VSmart testbed, the speed and direction of the movement are a part of command to actuate the executor. They are applied to implement a self-positioning module. As mentioned in the section 4.3.4, a job, which is used to drive the robot from one location to another, keeps track of the execution time and two parameters, linear velocity and angular velocity. If the previous location is aware, the new position can be calculated by using the following algorithm.

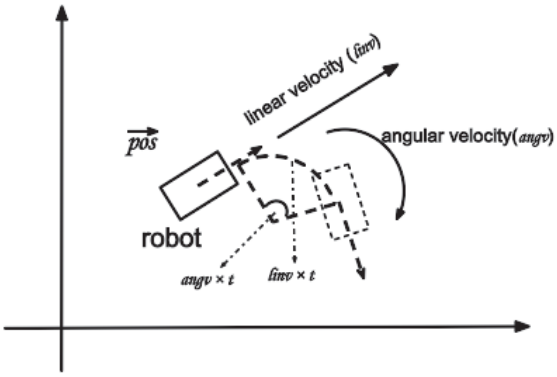
Algorithm. This algorithm is related to the robot used in the current implementation of the testbed. The robot adopted in this thesis is iRobot Create. According to the documentation of the iRobot Crate, there are three cases in the iRobot actuator command, which are illustrated in Figure 4-7. This algorithm keeps track of the current position and heading information.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 4–7: Scheme

Case 1: The angular velocity is 0. As illustrated in Figure 3-8 (a), the vehicle moves linearly and only position information is updated:

$$\vec{pos} = \vec{pos} + (linv \times cos(h), linv \times sin(h)) \times t, \tag{4.1}$$

where $\vec{pos}$ is the position of the vehicle in 2D space, $linv$ is the linear velocity, h is the current heading, t is the job execution time.

Case 2: The linear velocity is 0. As showed in Figure 4-7(b), only the heading is refreshed in this case:

$$h = (h + angv \times t)\%(2 \times \pi), \tag{4.2}$$

where $angv$ is the angular velocity.

Case 3: Neither the linear velocity nor angular velocity is 0. This case is a little bit complicated. As illustrated in Figure 4-7(c), the path of the vehicle is a circular arc. The heading of the vehicle can be obtained by the same method in case 2. To be able to get a new position of the vehicle, the coordinates are first transformed from cartesian coordinate system to polar coordinate system, since the polar coordinate is suitable for the angle and arc. In the polar coordinate system, the new position can be obtained by calculating the new radial coordinate and angular coordinate. Then, the position of the vehicle is transformed back to the cartesian system. It is worth noting that, in the polar coordinate, the original position is chosen as the pole and the original heading is the

polar axis. Thus, the transformation to cartesian system is composed of rotation and shift.

The following equation is used to calculate the vehicle coordinate in polar system:

$$radius = linv \div angv, \tag{4.3}$$

$$\gamma = angv \times t, \tag{4.4}$$

$$\begin{cases} \phi = \gamma/2, \\ r = 2 \times radius \times sin(\phi), \end{cases} \tag{4.5}$$

where $radius$ is the radius of the arc, γ is the angular rotation of the heading, (r, ϕ) is the coordinate of the polar system.

The next step is the transformation:

$$\varphi = \phi + h, \tag{4.6}$$

$$\vec{pos} = \vec{pos} + (r \times cos(\varphi), r \times sin(\varphi)), \tag{4.7}$$

where Equation (4.6) is the rotation and Equation (4.7) is the shift and convert. Figure 4-8 illustrates the coordinate transformation in case 3.

Periodical update. Indeed, there are two copies of location information in the VSmart self-positioning module, feedback copy and periodicity copy. Feedback copy is calculated based on the feedback of the job execution. In the job scheduler, the observer design pattern is utilized to inform the self-positioning module of the position and heading information once a job is finished or interrupted. In most of the experiments, the execution time of a job usually lasts for a few seconds. It means

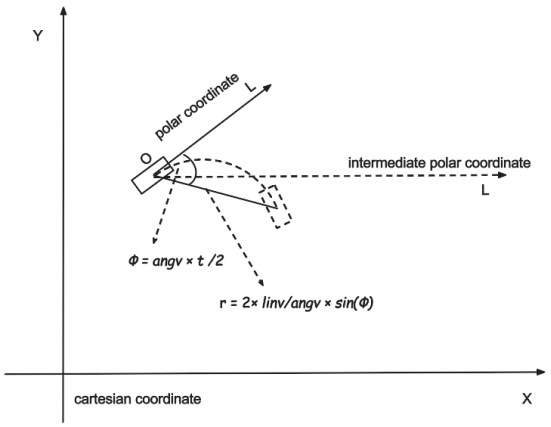


Figure 4–8: Coordinate Transformation in Case 3

that the feedback copy is updated a few seconds a time. It is not acceptable in the safety related scenarios such as collision avoidance. The use of periodicity copy can settle this issue. The periodicity copy is updated periodically at a specific rate and the rate can be adjusted based on the requirements. In VSmart, the rate is set to 10 times per second. The update is proceeded in a separate thread, periodicity thread. In the thread, the self-positioning module probes the job scheduler for the currently executing job and calculate the position with the algorithm described in last paragraph. The execution time for the algorithm is the interval of the probe, that is $1/rate$. Unfortunately, the periodical update has an accuracy issue. Figure 4-9 is one of the cases in which the calculation of periodicity copy results an inaccurate position. In the example, there are two jobs in the job scheduler. *Job1* has a linear velocity with -20 mm/s and is executed for 0.45 seconds. In *Job2*, the linear velocity is 20mm/s and the execution time is 0.5 seconds. The arrows mark the moments when the periodicity thread probes the currently executing job. The probe happens

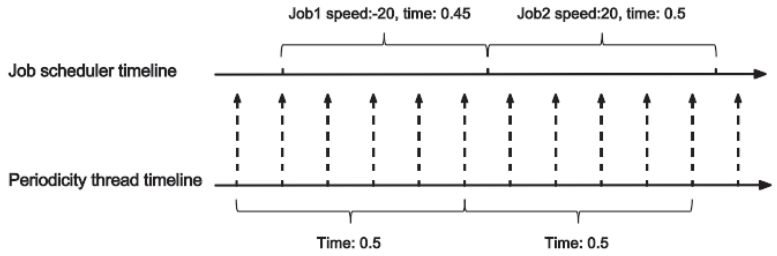


Figure 4–9: An example of the timeline of periodicity thread and job scheduler

every 100 milliseconds. It is easy to find out that, the vehicle should move 1 mm after the execution of both *Job1* and *Job2*, but the periodicity thread results 0 mm in the end. It is obvious that either feedback copy or periodicity copy has its limitation. Fortunately, they can make up for each other. In VSmart, the feedback copy can be used to refine the periodicity copy and the periodicity copy can provide the position information to the other modules more effectively and timely.

4.4 Executor of vehicle unit

The executor subsystem is the driver of the actuators in the testbed. It directly controls the movement of the robot. The implementation of the executor heavily depends on the use of the hardware. Different robots may have different programming interfaces. Therefore, it is hard to have a universal implementation of the executor subsystem. However, it is possible to provide a universal interfaces to the other subsystems or modules. In this thesis, iRobot Create [24] is used as the actuators of the vehicle units and the implementation of the subsystem has to follow the iRobot Create Open Interface [23]. In fact, according to [23], the iRobot Create has five types of actuators: wheels, speaker, LEDS, digital outputs and low side driver outputs. Among them, the wheels actuator is the most important one in VSmart testbed. The

command to trigger the actuators in iRobot Create includes two parts, the opcode and the data bytes. The opcode is the operation code, which represents the start of a specific command. The data bytes are the parameters for the execution details of the command. The command for wheels actuator starts with the opcode 137 and is followed by four data bytes. The first two bytes represent the velocity of the robot and the rest two represents the radius of the moving path. However, to make the command more readable and provide a universal interface, the wheels actuator command is wrapped in the API, *go(linear_velocity, angular_velocity)*. There are three cases in this API, which have been elaborated in the section 4.3.5. The map of the parameters between the API and the command is described as follow:

$$command.velocity = API.linear\_velocity, \tag{4.8}$$

$$command.radius = API.linear\_velocity/API.angular\_velocity \tag{4.9}$$

There are some ongoing projects, which have implemented the open interface of iRobot Create in Python language. Pycreate [27] is one of the best implementations. The VSmart testbed utilizes the pycreate code in the executor subsystem.

Similar to the other modules, the executor subsystem can be reimplemented as long as it follows the same interfaces

4.5 The processor of infrastructure unit

As introduced in the previous sections, the VSmart testbed has another important unit, the infrastructure unit. In this section, the processor in the infrastructure unit is discussed. Similar to the vehicle unit, the processor is the brain of the infrastructure unit. The applications as well as some data processing functionalities

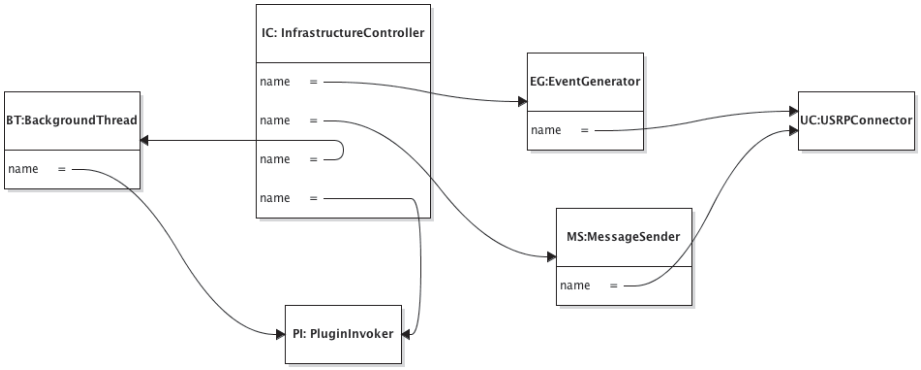


Figure 4–10: Object Diagram of the processor in the infrastructure unit

are implemented in this subsystem. There are many similarities between the processors in the vehicle unit and the infrastructure unit. First of all, they both have the layered structure to handle the packets received from *transciver*. Secondly, they are all event-driven systems and use the same design and interfaces in the implementation. Finally, the infrastructure unit also has the plugin design for the customized functionalities. These have been covered in the section 4.3.

Figure 4-10 shows the object diagram of the processor of infrastructure diagram. It has three important components, *USRPConnector*, *PluginInvoker* and the *BackgroundExecutor*. *USRPConnector* is used to connect the processor with *transceiver* subsystem. It can receive the packets from *transceiver* as well as forward the message. It is wrapped by *EventGenerator* and *MessageSender*. The *EventGenerator* can parse the packets and conduct the event generation. *MessageSender* is used in the other modules of the processor to broadcast the message. In the processor, the *USRPConnector* and *EventGenerator* are running in two separated threads for the real time purpose. Another important component is the *BackgroundExecutor*. As an essential part of the DSRC system, the infrastructure conveys the useful information

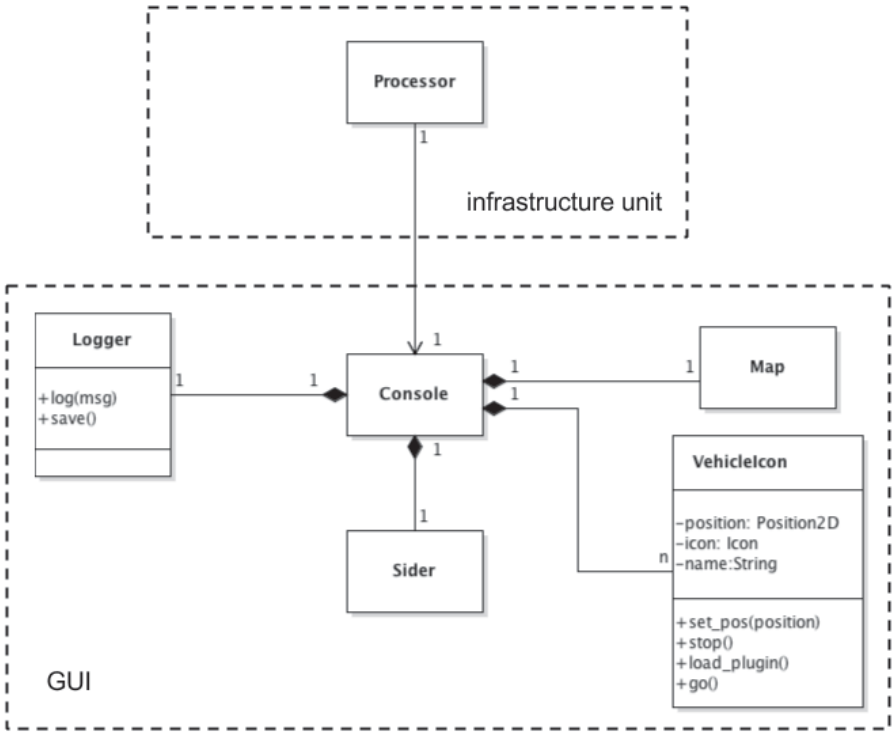


Figure 4–11: Class Diagram of the processor in the infrastructure unit

to the vehicles. This procedure should be running in the background and executed continuously. Therefore, a background thread is required in the processor. The functions executed in the thread can be customized by the plugin design, which is similar to the processor in the vehicle unit. *PluginInvoker* is used to invoke the customized functions. The details of the plugin design is covered in the section 4.3.3.

4.6 GUI Design

Despite the implementation of the framework of DSRC, some Graphic User Interface (GUI) applications are implemented to simplify the process of the DSRC experiments. In the current implementation, these functions include virtual map, logger, remote control, the batch job editor and some exclusive commands, such

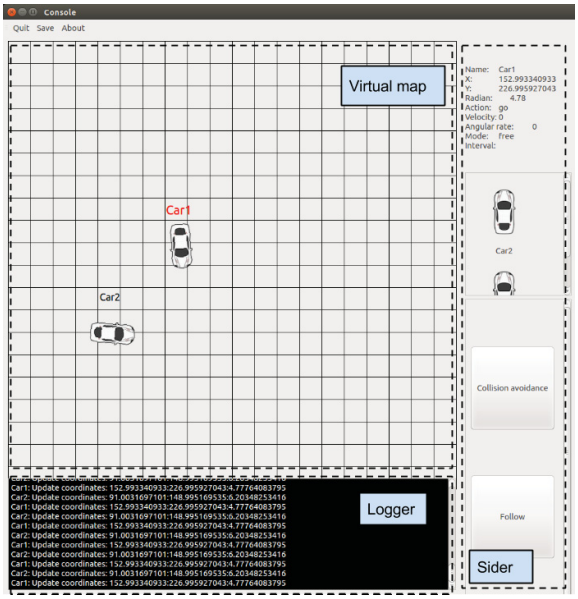


Figure 4–12: Console in the infrastructure unit

as the mode switch for the robot, plugin loading and position reset. The GUI applications are hosted in the infrastructure unit as central controller of the entire VSmart testbed. Most of these functions require communications with vehicle units. Apparently, the messages for GUI applications are not defined in DSRC standards. Therefore, the exclusive message formats are defined for the communications between vehicle and infrastructure units. The rest of the section elaborates the detail of these exclusive functions and the related GUI design.

Console. The console is the platform where the user can interact with the processor. Figure 4-11 is the class diagram of the console. It depicts the relationship between console and the processor in the infrastructure unit. Figure 4-12 is a screenshot of a console. In the console, there are three elements, virtual map, logger, and sider.

The virtual map can track and display movements of the vehicle units. As long as the infrastructure unit is within the transmission range of the vehicle unit, the basic vehicle messages can be probed. After extracting the information from the messages, the position is known and the vehicle can then be drew at the appropriate location in the virtual map. With the help of the virtual map, users can visually conduct and monitor experiments, even without the participation of the executor subsystem.

Logger is a tool, which keeps track of received events and user's actions. The use of the logger can help the users to find the issues in the experiments more conveniently. The users can save the log to files for further exploration and comparison.

The sider is separated into three parts. The panel at the top is the status display panel. It displays the information for the currently selected vehicle. The information includes the name, the coordinate, and the action. The panel in the middle is a list of vehicles within the transmission range and the bottom panel is a plugin controller for the selected vehicle. As discussed in vehicle unit, users can implement customized functions as plugins. In practice, a user may adopt several plugins in a single experiment. The plugin controller allows the user to switch between different plugins.

Remote Control. In VSmart testbed, the general way to control a vehicle unit is to use the keyboard on the laptop. However, the operability is not satisfactory, especially, when there is more than one vehicle running simultaneously. Therefore, the remote control in the console is designed for manipulating the vehicles. There are two ways to control the vehicle units from the console. The first one is to use the

Figure 4–13: Remote control panel

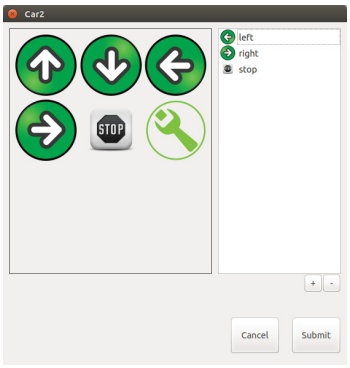


Figure 4–14: Batch job editor

control panel. Figure 4-13 is the vehicle control panel. It is very straightforward and operable. Another way is the Drag and Drop (DnD). The user can drag and drop the vehicle unit icon in the virtual map to control the corresponding vehicle.

Batch job editor. Sometimes, an experiment may require the vehicle to execute a list of predefined jobs. The processor in the infrastructure provides a function to edit a batch of jobs. The user can easily define a list of ordered jobs and submit to the designated vehicle unit. The vehicle will then process the jobs in job scheduler. Figure 4-14 is the GUI of batch job editor.

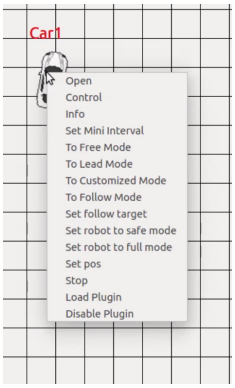


Figure 4–15: Command and shortcuts to manipulate the vehicle

Command. A few common commands and shortcuts are also implemented in the processor to manipulate the vehicle, such as, stop vehicle, reset vehicle, set the position, load vehicle plugin, disable vehicle plugin, etc. In addition, users can also define the customized commands in the infrastructure plugin. It will be automatically loaded.

CHAPTER 5
DSRC Use Cases

With the help of VSmart testbed, many popular use cases can be tested and demonstrated. In this chapter, collision avoidance and cooperative adaptive cruise control are introduced to illustrate the usability of the testbed.

The experimental setup of both use cases contains two vehicle units and an infrastructure unit. Each of the vehicles is composed of a USRP, a laptop and an iRobot. The infrastructure unit consists of a USRP and a desktop computer. The vehicles are placed in a $5m \times 5m$ space. The infrastructure unit is responsible for remote control and monitoring.

5.1 Collision Avoidance

Collision avoidance is one of the most important applications in V2V systems. There are many previous researches [38, 39] working on the design and analysis of the collision avoidance application. In this section, an effective strategy and its implementation are introduced.

To detect the collision, a vehicle has to know the position and the movement information of its neighbours. A vehicle can obtain the information from basic vehicle messages broadcast by its neighbours.

In Figure 5-1, $\vec{V}1$ and $\vec{V}2$ are the velocity of the *Car1* and *Car2* respectively. p is the intersection point. $L1$ and $L2$ are the distance between p and the cars respectively. With the following equations, the intersection point p can be found:

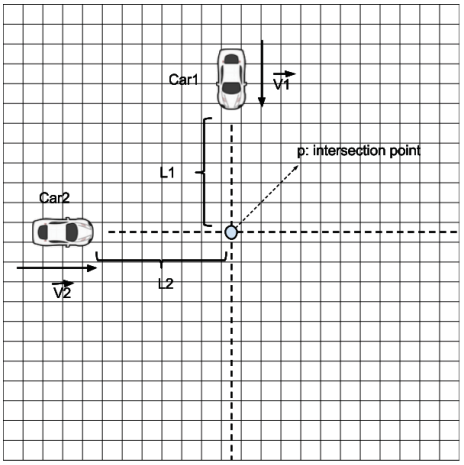


Figure 5–1: Collision Detection

$$\begin{cases} \vec{OP} = \vec{OC_1} + t_1\vec{V1} \\ \vec{OP} = \vec{OC_2} + t_2\vec{V2} \end{cases} \tag{5.1}$$

where $\vec{OP}$ is the coordinate of the intersection point p, $\vec{OC_1}$ and $\vec{OC_1}$ donate the coordinates of the cars, $\vec{V1}$ and $\vec{V2}$ represent the velocity vector of the cars, t_1 and t_2 donate the time to move the vehicle to p from $\vec{OC_1}$ and $\vec{OC_1}$ respectively. In the equations, the $\vec{OP}$, t_1 and t_2 are unknown and need to be solved.

The solution to the equations has five cases. Figure 5-2 illustrates five cases of the solution. In the first three cases, the equations has either no solution or many solutions. It indicates that $\vec{V1}$ and $\vec{V2}$ are parallel. In these scenarios, the distance between the paths of the cars has to be determined to find out if there is a collision. The distance can be calculated in the following equations:

$$\vec{C_2C_1} = \vec{OC_1} - \vec{OC_2}, \tag{5.2}$$

$$\phi = arccos(\vec{C_2C_1} \cdot \vec{V1}/(\left|\vec{C_2C_1}\right| \cdot \left|\vec{V1}\right|)), \tag{5.3}$$

$$d\_path = \left|\vec{C_2C_1}\right| \times sin(\phi), \tag{5.4}$$

where $\vec{C_2C_1}$ is the vector from *Car2* and *Car1*, ϕ is the angle between $\vec{C_2C_1}$ and the velocities, *d_path* is the distance between two paths.

With the help of the above equations, the cases can be analyzed as follow:

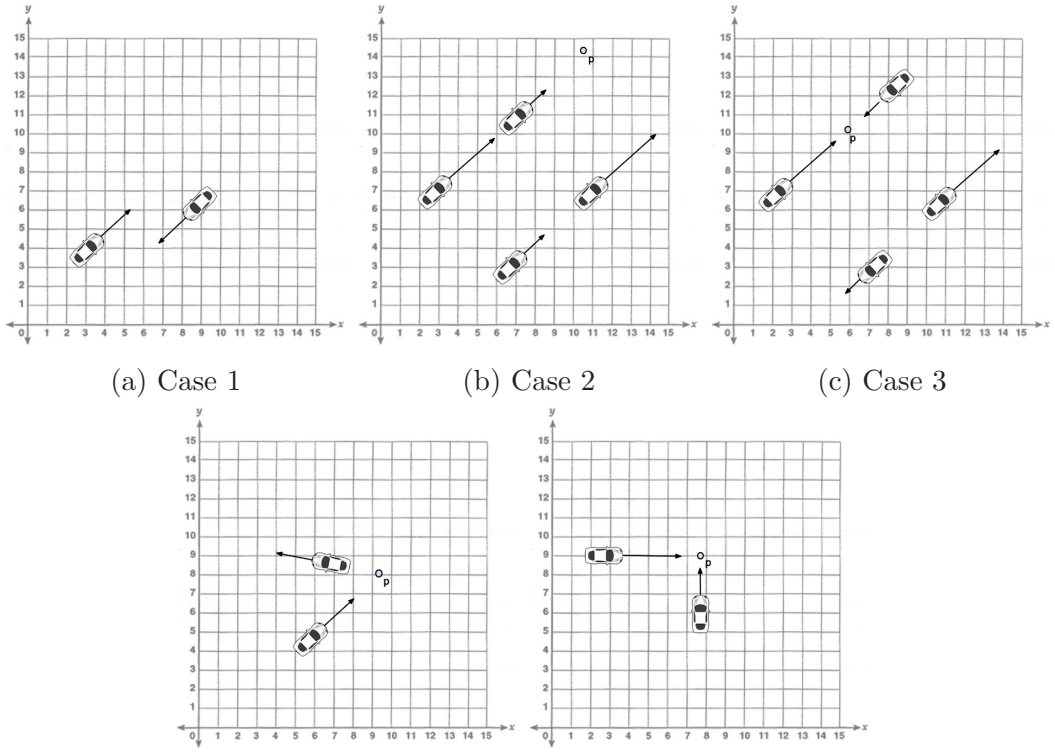
Case 1: $\vec{V1} = k \cdot \vec{V2}$ and $d\_path > w$, where w is the width of a car. In this case, the path of *Car1* is parallel to the path of *Car2* and the *d_path* is larger than the width of a car, which is considered as the safe distance between two paths. Thus, there is no intersection point.

Case 2: $\vec{V1} = k \cdot \vec{V2}$, $d\_path < w$ and $k > 0$. In this case, the *d_path* is smaller than the safe distance and the vehicles move in the same direction. If the vehicle ahead has a smaller speed, then a collision is expected; otherwise, there is no collision.

Case 3: $\vec{V1} = k \cdot \vec{V2}$, $d\_path < w$ and $k < 0$. In this case, the *d_path* is smaller than the safe distance and the vehicles move in the opposite direction. If the vehicles has passed the intersection point, there is no collision; otherwise, a collision is expected.

Case 4: The Equation (5.1) has only one solution and either t_1 or t_2 is less than 0. It means at least one of the vehicle has passed the intersection point and it is safe for both vehicles to move forward.

Case 5: The Equation (5.1) has only one solution and both t_1 and t_2 are greater than or equal to 0. Definitely, this case implies the possibility of the



(a) Case 1 (b) Case 2 (c) Case 3

(d) Case 4 (e) Case 5

Figure 5–2: The cases in collision avoidance

collision. However, if the difference between t_1 and t_2 is larger than a threshold, the vehicle is safe to go; otherwise, a collision is expected.

As long as the vehicle knows where and when the collision would happen, it can take action to avoid it. For instance, the vehicle can slow down or pause the actuator for a while. In this use case, the vehicle first slows down and then stops fully if the threat still exists. The use case is implemented as a plugin in the processor of the vehicle unit, called *CAPlugin*. As mentioned in section 4.3.3, a plugin is composed by three modules. Since the use case utilizes the message from the neighbours, the *CustomizedReceiver* has to be implemented. As the basic vehicle

messages are of a built-in message type, the *CustomizedExecutor* and *CustomizedEvent* could be neglected. In *CustomizedReceiver*, there is a abstracted method, *customized_event_handler(processor, event)*, which is the interface defined for *PluginInvoker*. It has to be implemented in order to be called by the invoker. The following pseudo code depicts the core procedure of the collision avoidance plugin.

```
stop_sign = False
slow_sign = False
def customized_event_handler(my_speed, my_position, event):
    neigh_speed = event.get_speed()
    neigh_position = event.get_position()
    intersect_position = find_collision_point(my_position
                                              neigh_position,
                                              my_speed,
                                              neigh_speed)
    if intersect_position exists:
            time_normal = calc_moving_time(my_position,
                                           intersect_position,
                                           my_speed)
            time_slow = calc_moving_time(my_position,
                                         intersect_position,
                                         my_speed/2)
            time_neigh = calc_moving_time(neigh_position,
                                          p,
```

```
                                        neigh_speed)
    if abs(time_slow - time_neigh) <= threshold:
        if not stop_sign:
            stop()
            stop_sign = True
    elif abs(time_normal - time_neigh) <= threshold:
        if not slow_sign:
            slow_down(my_speed/2)
            slow_sign = True
    else:
        if stop_sign or slow_sign:
            resume()
            stop_sign = False
            slow_sign = False
```

In the implementation, if the collision point p exists, the vehicle will slow down or stop based on the time difference between itself and its neighbour. The experiment shows that this use case works perfect in the testbed. However, if another vehicle has the same strategy, both of them will stop and keep still since each will treat the other as a threat.

5.2 Cooperative Adaptive Cruise Control

Cooperative Adaptive Cruise Control (CACC) is another very popular use case of the DSRC. CACC allows a vehicle to follow the preceding one and maintain a

fixed distance. The mechanism is that the vehicle can detect the action of the preceding vehicle and drive in a similar way. In fact, the CACC system is complicated. There are many factors that need to be considered, for instance, platoon length, fast traveling time, platoon merging behaviour, and damping behaviour in strong acceleration and deceleration situations [17]. However, the main focus of this thesis is not on the design of CACC systems or protocols. In this thesis, a simplified use case is implemented as plugin. In the implementation, the vehicle strictly follows the action of the preceding vehicle. Even though the use case is a simplified version, it is enough to demonstrate the usability of the VSmart testbed. Users of VSmart can extend the current plugin to a CACC system with full functionality..

The slave vehicle, which follows the preceding one, has to listen to the messages that are broadcasted by the preceding vehicle. Thus, the *CustomizedExecutor* module in the plugin of the leading vehicle has to be implemented. It keeps broadcasting the leading messages, which include the information of location, action and timing. Similar to collision avoidance, the slave vehicle receives the messages and passes them to the *CustomizedReceiver* for processing. The implementation of *CustomizedReceiver* module, in this case, handles the leading messages and actuates the vehicle unit to follow the actions.

The following pseudo code is the core implementation of *CustomizedExecutor* module in the plugin of the leading vehicle. The function *execute* is called by the processor continuously based on the interval defined by the user. *_generate_leading_message* is the function which generates the customized leading message. The message format is defined in the *CustomizedEvent* module.

```
def execute():
    msg = _generate_leading_message()
    send_to_USRP(msg)
```

The pseudo code below is the most elementary implementation of *CustomizedReceiver* in the slave vehicle. It retrieves the action information from the message, and add it to the job scheduler.

```
def customized_event_handler(event):
    action = event.action
    new_job = Job(action.name, action.arg1, action.arg2)
    send_to_job_processor(new_job)
```

With the help of the plugins, the slave vehicle can follow the action of the preceding one.

CHAPTER 6
Conclusion and Future Works

6.1 Conclusion

This thesis analyzes the importance of the small scale DSRC testbed and discusses the design and implementation of the VSmart testbed in details. It describes the hardware requirements for the platform construction as well as three subsystems in the software design. The VSmart testbed has two kinds of units, vehicle unit and infrastructure unit, and can be used in three different modes, i.e., simulation mode, stationary mode, and full mode. Simulation mode works with *processor* subsystem only, while stationary mode enables the *transceiver* on the base of simulation mode. Full mode adds *executor* upon stationary mode. In the software design, the architecture of the VSmart testbed is a combination of layered architecture and event-driven architecture. In addition, plugin design pattern is integrated in the *processor* subsystem, which significantly improves the extensibility of the entire platform. The job scheduler design enables batch processing in the *executor*. It improves the efficiency of the experiments by allowing users to predefine the jobs. Furthermore, to overcome the shortage of lack of positioning system, the self-positioning module is designed in the *processor*. It can precisely estimate the current location of the vehicle unit. Beyond that, some experiment-oriented applications are designed in the infrastructure unit. A GUI is designed in the console, which allows the researcher to conduct graphical manipulation over the testbed. The virtual map design increases the visual

impression, because users can directly obtain the vehicle information over the map. The remote control, logger and other applications in the infrastructure improve the usability in the experiments.

6.2 Future Works

Implementation of the standards. In the thesis, the layered structure of the testbed is introduced. The VSmart follows design of DSRC communication stacks. Currently, the VSmart does not fully implement the standards, especially, the middle stack and the top stack. The implementation of the standards can verify the performance of the DSRC protocols.

Robustness of the transceiver. The VSmart utilizes the USRP B210 as the hardware of the transceiver. The wireless communication over the USRP board is sometimes unstable. There are many factors which can affect the performance of the transceiver. For instance, both placement of the board and orientation of the antenna can cause instability. It is acceptable in V2V and V2I communications, since the units are continuously broadcast the messages. The lost of a few packages may not cause a big issue in some experiments. However, the instability impact the use of some experiment-oriented applications. One way to enhance the robustness of the transceiver is to utilize the retransmission mechanism.

A scalable experiment-oriented testbed. Even though the purpose of the design of VSmart is to build a small scale testbed, it is till possible to use multiple vehicle units and infrastructure units in the same experiment. The experiment-oriented applications are limited within a specific infrastructure unit because of the transmission range of the SDR. Therefore, for the experiment with multiple infrastructure

units, the use of some experiment-oriented applications could be complicated. However, a central control station may overcome the difficulty. The central control station can gather the information from all the infrastructure units and present to the researchers. On the other hand, it can also control the entire testbed by a centralized method. The central control station is one way to increase the scalability of the testbed.

References

[1] Philippe Agostini, Raymond Knopp, J Harri, and Nathalie Haziza. Implementation and test of a DSRC prototype on OpenAirInterface SDR platform. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 510–514. IEEE, 2013.

[2] F Ahmed-Zaid, F Bai, S Bai, C Basnayake, B Bellur, S Brovold, G Brown, L Caminiti, D Cunningham, H Elzein, et al. Vehicle Safety Communications-Applications (VSC-A) Final Report. *Report No. DOT HS*, 811, 2011.

[3] Stephane Beauregard and Harald Haas. Pedestrian dead reckoning: A basis for personal positioning. In *Proceedings of the 3rd Workshop on Positioning, Navigation and Communication*, pages 27–35, 2006.

[4] Bastian Bloessl, Michele Segata, Christoph Sommer, and Falko Dressler. An IEEE 802.11a/G/P OFDM Receiver for GNU Radio. In *Proceedings of the Second Workshop on Software Radio Implementation Forum*, SRIF '13, pages 9–16, New York, NY, USA, 2013. ACM.

[5] Eric Blossom. GNU radio: tools for exploring the radio frequency spectrum. *Linux journal*, 2004(122):4, 2004.

[6] Cyril Brignone, Tim Connors, Geoff Lyon, and Salil Pradhan. Smartlocus: An autonomous, self-assembling sensor network for indoor asset and systems management. *Mobile Media Syst. Lab., HP Laboratories, Palo Alto, CA, Tech. Rep*, 41, 2003.

[7] Transport Canada. Canadian motor vehicle traffic collision statistics. *Ottawa, Ontario*, 2011.

[8] Jussi Collin, Oleg Mezentsev, Gérard Lachapelle, et al. Indoor positioning system using accelerometry and high accuracy heading sensors. In *Proc. of ION GPS/GNSS 2003 Conference*, pages 9–12, 2003.

[9] Hugo Conceição, Luís Damas, Michel Ferreira, and João Barros. Large-scale simulation of V2V environments. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 28–33. ACM, 2008.

[10] CAMP Vehicle Safety Communications Consortium et al. Vehicle safety communications project: task 3 final report: identify intelligent vehicle safety applications enabled by DSRC. *National Highway Traffic Safety Administration, US Department of Transportation, Washington DC*, 2005.

[11] Louis E. Frenzel Electronic Design. The Elusive Software-Defined Radio. `http://electronicdesign.com/communications/elusive-software-defined-radio`, 2006.

[12] dugsong. dpkt. https://code.google.com/p/dpkt/, 2013.

[13] Jared Dulmage, Minko Tsai, Mike Fitz, and Babak Daneshrad. COTS-based DSRC Testbed for Rapid Algorithm Development, Implementation, and Test. In *Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, WiNTECH '06, pages 113–114, New York, NY, USA, 2006. ACM.

[14] Ricardo Fernandes, Pedro M d'Orey, and Michel Ferreira. DIVERT for realistic simulation of heterogeneous vehicular networks. In *MASS*, pages 721–726, 2010.

[15] Ricardo Fernandes, Francisco Vieira, and Michel Ferreira. VNS: An integrated framework for vehicular networks simulation. In *Vehicular Networking Conference (VNC), 2012 IEEE*, pages 195–202. IEEE, 2012.

[16] Software Defined Radio Forum. SDRF Cognitive Radio Definitions. `http://www.sdrforum.org/pages/documentLibrary/documents/SDRF-06-R-0011-V1_0_0.pdf`, 2007.

[17] Andreas Geiger, Martin Lauer, Frank Moosmann, Benjamin Ranft, Holger Rapp, Christoph Stiller, and Julius Ziegler. Team AnnieWAY's entry to the 2011 Grand Cooperative Driving challenge. *Intelligent Transportation Systems, IEEE Transactions on*, 13(3):1008–1017, 2012.

[18] IEEE 802.11 Working Group et al. IEEE Standard for Information Technology–Telecommunications and information exchange between systems–Local and metropolitan area networks–Specific requirements–Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications

Amendment 6: Wireless Access in Vehicular Environments. *IEEE Std*, 802:11p, 2010.

[19] Dominik Gusenbauer, Carsten Isert, and J Krosche. Self-contained indoor positioning on off-the-shelf mobile devices. In *Indoor positioning and indoor navigation (IPIN), 2010 international conference on*, pages 1–9. IEEE, 2010.

[20] Jeffrey Hightower, Roy Want, and Gaetano Borriello. SpotON: An indoor 3D location sensing technology based on RF signal strength. *UW CSE 00-02-02, University of Washington, Department of Computer Science and Engineering, Seattle, WA*, 1, 2000.

[21] C. A. R. Hoare. Monitors: An Operating System Structuring Concept. *Commun. ACM*, 17(10):549–557, October 1974.

[22] SAE International. DSRC Implementation Guide: A guide to users of SAE J2735 message sets over DSRC. *DSRC Committee*, 2010.

[23] iRobot Corporation. iRobot Create Open Interface. http://www.irobot.com/filelibrary/pdfs/hrd/create/Create2006.

[24] iRobot Corporation. iRobot-Create 2. http://www.irobot.com/About-iRobot/STEM/Create-2.aspx, 2015.

[25] John B Kenney. Dedicated Short-range Communications (DSRC) standards in the United States. *Proceedings of the IEEE*, 99(7):1162–1182, 2011.

[26] Chih-Neng Liang and Bo-Chiuan Chen. A study of DSRC jammer in vehicle safety application testbed. In *ITS Telecommunications (ITST), 2012 12th International Conference on*, pages 216–219. IEEE, 2012.

[27] mgobryan. PyCreate Project. https://github.com/mgobryan/pycreate, 11 2010.

[28] Joe Mitola. The software radio architecture. *Communications Magazine, IEEE*, 33(5):26–38, 1995.

[29] NEC. NEC Car2X Communication SDK. `http://www.research2standards.net/`.

[30] Lionel M Ni, Yunhao Liu, Yiu Cho Lau, and Abhishek P Patil. LANDMARC: indoor location sensing using active RFID. *Wireless networks*, 10(6):701–710, 2004.

[31] U.S. Department of Transportation. Testing Connected Vehicle Technologies in a Real-World Environment. http://www.its.dot.gov/connected_vehicle/pdf/DOT_CVBrochure.pdf, 2011.

[32] U.S. Department of Transportation. IEEE 1609 - Family of Standards for Wireless Access in Vehicular Environments (WAVE). http://www.standards.its.dot.gov/Factsheets/Factsheet/80, 2013.

[33] World Health Organization et al. *WHO Global Status Report on Road Safety 2013: Supporting A Decade of Action*. World Health Organization, 2013.

[34] Saurabh D Patil, DV Thombare, and Vaishali D Khairnar. DEMO: Simulation of Realistic Mobility Model and Implementation of 802.11 p (DSRC) for Vehicular Networks (VANET). *arXiv preprint arXiv:1304.5190*, 2013.

[35] Patrick Ponticel. Michigan's V2V test bed touted. http://articles.sae.org/9682/, 2011.

[36] GNU Radio. A Quick Guide to Hardware and GNU Radio. https://gnuradio.org/redmine/projects/gnuradio/wiki/Hardware.

[37] Ettus Research. Product Categories. http://www.ettus.com/product, 2015.

[38] Antony Tang and Alice Yip. Collision avoidance timing analysis of DSRC-based vehicles. *Accident Analysis & Prevention*, 42(1):182–195, 2010.

[39] Raymond Tatchikou, Subir Biswas, and Francois Dion. Cooperative vehicle collision avoidance using inter-vehicle packet forwarding. In *Global Telecommunications Conference, 2005. GLOBECOM'05. IEEE*, volume 5, pages 5–pp. IEEE, 2005.

[40] National Highway Traffic Safety Administration U.S. Department of Transportation. Traffic Safety Facts Research Note. *Report No. DOT HS*, 812:101, 2014.

[41] Cohda Wireless. CohdaMobility MK2 Cooperative-ITS Module. http://cohdawireless.com/Portals/0/PDFs/CohdaWirelessMK2.pdf.

[42] Weidong Xiang, Yue Huang, and Sudhan Majhi. The design of a wireless access for vehicular environment (WAVE) prototype for intelligent transportation system (ITS) and vehicular infrastructure integration (VII). In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–2. IEEE, 2008.

List of Symbols

BDS	Beidou Navigation Satellite System
CCH	control channel
COTS	commercial off-the-shelf
CPU	Central Processing Unit
DnD	Drag and Drop
DSP	digital signal processing
DSRC	the dedicated short-range communication
FIFS	First in first served
GPS	Global Positioning System
GUI	Graphic User Interface
IPC	inter-process communication
JSON	JavaScript Object Notation
MAC	medium access control layer
MCU	Microcontroller
OFDM	Orthogonal Frequency Division Multiplexing
PC	personal computer
PHY	physical layer
RSU	roadside unit
SCH	service channel

SWIG	Simplified Wrapper and Interface Generator
USDOT	the U.S. Department of Transportation
USRP	Universal Software Radio Peripheral
UTC	Universal Coordinated Time
V2V	vehicle-to-vehicle
VSC	the Vehicle Safety Communications
WAVE	Wireless Access in Vehicular Environments
WHO	World Health Organization