東京大学
情報理工学系研究科 創造情報学専攻
修士論文

路車間通信を利用した自動走行のための協調認識システムの提案と評価

Infrastructure-aided cooperative perception for connected autonomous vehicles

大井 貴晴
Takaharu Oi

指導教員　江崎 浩 教授

2020 年 1 月

概要

近年、自動車の自動運転への関心が高まると同時に、その開発も進んでいる。現在の開発分野は自律型自動運転と協調型自動運転に大別される。自律型自動運転は車両に取り付けられたセンサなどの自車両から得られる情報をもとに自動運転をする技術であり、協調型自動運転は通信により他の車両から送られてきた情報をもとに自動運転を行うものである。そして、これらは将来的に統合され、自車両で得られた情報と他車両から送られてきた情報をもとに自動運転をすることが想定されている。

自律型自動運転を実現するオープンソースソフトウェアとして Autoware が存在する。Autoware は自車両に取り付けられたセンサを用いて意思決定をし、車両制御をする。しかし、Autoware は自車両から見える範囲の情報しか認識することができず、それにより安全面での問題が発生することが考えられる。

そのような問題を解決するため、本研究では協調型オープンソースソフトウェアである OpenC2X を Autoware に統合し、Autoware の認識領域を拡張するシステムである AutoC2X を提案・実装する。インドアテスト及びフィールド実験による性能評価を行った結果、路側機で物体を検知してから CAM の生成周期である 100ms 以内で多くのパケットは離れた車両に送信することができることが明らかになった。

Abstract

Recently, more and more attention is paid to self-driving. Development of self-driving is divided into autonomous self-driving and cooperative self-driving. In autonomous self-driving, Cars use information from sensors attached to vehicle. On the other hand, in cooperative self-driving, cars use information from other cars via networks.

One of the autonomous self-driving software is Autoware. Autoware is open source software running in autonomous self-driving cars. Autoware uses sensor information collected by LiDAR and camera attached to ego-vechile, decides route, and controls ego-vehicle. However, Autoware only uses information that can be seen from ego-vehicle. Therefore, it may happen some accidents.

In this research, we propose to integrate cooperative self-driving software(OpenC2X) into Autoware and expand recognition area of Autoware to solve that problem. We call the system AutoC2X. Then, we evaluated the system by indoor test and outdoor test and proved that objects information could be sent mostly within 100ms which is CAM generation interval.

目次

第 1 章　序論 1
1.1 はじめに . . . 1
1.2 本研究の目的 . . . 1
1.3 本論文の構成 . . . 2

第 2 章　自律型自動運転と協調型自動運転 3
2.1 自律型自動運転 . . . 3
2.2 センサによる検知の限界 . . . 3
2.3 オープンソースソフトウェア Autoware . . . 4
2.4 協調型自動運転 . . . 6
2.5 ETSI による標準化 . . . 6
2.6 オープンソースソフトウェア OpenC2X . . . 8

第 3 章　関連研究 11
3.1 生センサデータの共有 . . . 11
3.2 Proxy CAM . . . 12
3.3 Autoware を用いた研究 . . . 12
3.4 OpenC2X を用いた研究 . . . 13

第 4 章　AutoC2X 14
4.1 要求事項 . . . 14
4.2 AutoC2X の概要 . . . 15
4.3 本手法の設計 . . . 16
4.4 座標変換 . . . 18

第 5 章　実装 21
5.1 実装で用いたソフトウェア・ライブラリについて . . . 21
5.2 システム概要 . . . 22
5.3 AutoC2X-AW の概要 . . . 22
5.4 AutoC2X-OC の概要 . . . 26

第 6 章 実験・評価 31
6.1 共通の実験環境 31
6.2 計測手法 32
6.3 インドアテスト 34
6.4 フィールド実験 36

第 7 章 結論 39
7.1 まとめ 39
7.2 今後の課題 39

発表文献と研究活動 41

参考文献 42

第 1 章

序論

1.1 はじめに

自動車の自動運転への社会的な関心が高まっている。その背景には自動運転が実現することによる安全な移動の実現などがある。しかし、Uber 社のテスト走行に代表されるように自動運転の実験における死亡事故も実際に発生している。これは自動運転技術が発展途上であり、研究の余地が大いに残されているということを示している。

自動運転における現在の開発分野は自律型自動運転と協調型自動運転に大別される。自律型自動運転は車両に取り付けられたセンサなどの自車両から得られる情報をもとに自動運転をする技術である。一方、協調型自動運転は通信により他の車両から送られてきた情報をもとに自動運転を行うものである。そして、これらは将来的に統合され、自車両で得られた情報と他車両から送られてきた情報をもとに自動運転をすることが想定されている。

自律型自動運転の開発は様々な組織で行われているが、オープンソースとして開発されているソフトウェアがいくつか存在する。その代表として Autoware Foundation による Autoware[1] や、Baidu 社による apollo[2] や、NVIDIA 社による NVIDIA Drive[3] などがある。

一方、協調型自動運転においては欧州、アメリカ、日本でそれぞれが通信方式などに関して標準化を推し進めている。欧州においては ETSI[4] が、アメリカにおいては IEEE[5] が、日本においては ARIB[6] が主導している。

現在開発されている自律型自動運転にはいくつかの通信における問題が存在する。それは安全性、拡張性、標準への準拠という点である。これらの問題は協調型自動運転の分野で標準化されている規格を適切に統合することにより解決することができる。

1.2 本研究の目的

自律型自動運転の問題点として先述の点以外にも、自車両から見える範囲外の状況を認識することができないという問題点が存在する。自律型自動運転の研究には、死角から人や車が飛び出してくる可能性を考慮して自動運転を行う確率論的な手法 [7] は存在するが、他車両の情

報を利用した方が自動運転の効率と安全性を向上させることができると考えられる。

そこで本研究では自律型自動運転ソフトウェアである Autoware が認識する領域を拡張することを目的とする。そのために協調型自動運転ソフトウェアである OpenC2X を Autoware に統合し、路側機が得た情報を車両に送信するシステムを提案し、評価する。

1.3　本論文の構成

本論文は次のように構成される。第 2 章では自律型自動運転と協調型自動運転それぞれの技術について背景を説明し、第 3 章では本研究で用いる手法の関連研究を紹介する。第 4 章では本研究の提案手法について説明し、第 5 章でその実装を紹介する。第 6 章では実装を元に行った実験及びその結果について説明し、第 7 章で本研究のまとめと今後の課題を述べる。

第 2 章

自律型自動運転と協調型自動運転

本章では現在の自動運転開発における両輪に当たる、自律型自動運転と協調型自動運転について述べる。自律型自動運転がニュースや産業界では取り上げられることが多いが、自律型自動運転特有の問題点も存在する。その問題点の解決を目標に本研究では協調型自動運転と自律型自動運転を統合するが、その背景について述べる。

2.1 自律型自動運転

自律型自動運転とは、自車両に取り付けられたカメラや LiDAR などのセンサ情報をもとに、自己位置推定や、自車両周辺状況を認識することで自動運転を行う技術である。

カメラの情報をもとに運転する場合は、画像認識技術の精度・速度向上により安全性が高められ、LiDAR の情報をもとに運転する場合は、得られた点群データのクラスタリングの精度・速度向上により安全性が高められる。

例えば、テスラ社の自動車にはオートパイロット [8] という部分的な自動運転機能が搭載されているが、これは車両に取り付けられた 6 つのカメラと超音波、レーダーによって周囲の状況を認識し、自動走行を行う。

2.2 センサによる検知の限界

先述の通り自律型自動運転は自車に取り付けられたセンサ情報のみを元に自動運転を行う技術である。したがって、自車両からセンサを通して見えない部分の情報に関しては得ることができない。本研究において用いたセンサ (Velodyne VLP-16) は直線距離では 100m 先まで検知範囲であるが、物体を物体として検知できるのは環境によると考えられるが、本研究の環境においては 20m 程度先の物体までしか認識することができなかった。

また、障害物があるとその後方を検知することはできない。公道においては、歩行者や自転車が死角から急に飛び出してくる可能性もあり、センサで検知できる情報のみを信頼して運転を行うと重大な事故につながる可能性がある。

死角から歩行者などが急に飛び出してきても安全な速度で走り、歩道との適切な距離を取る

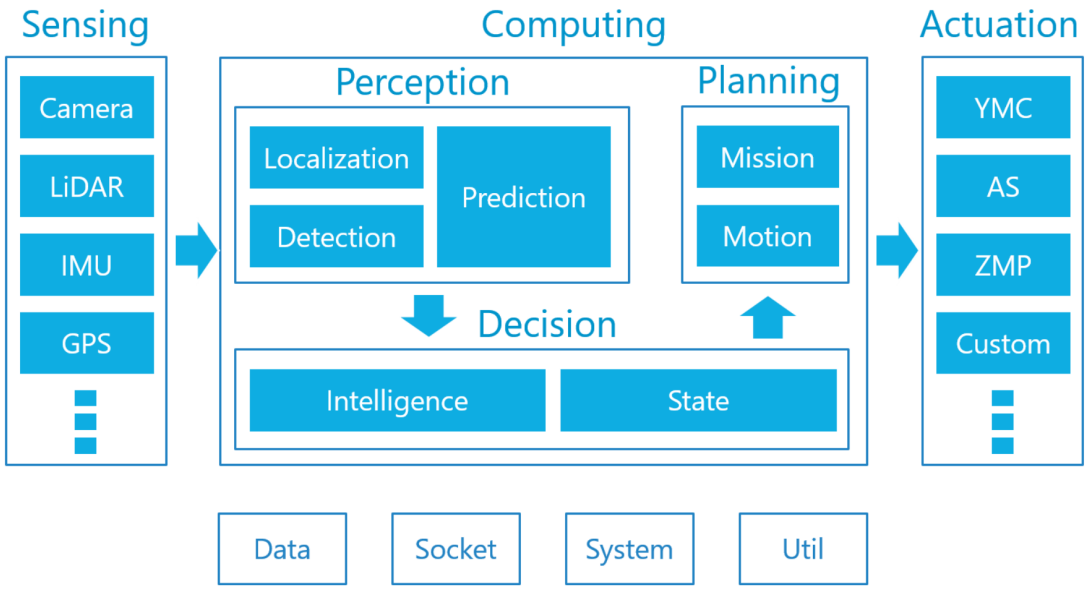


図 2.1. Autoware のシステム概要 (出典:[9])

ことで重大な事故は回避できると考えられる。しかし、可能性を考慮するあまり人間が運転する以上に慎重な運転を自動運転者がするようになり、交通が非効率になる状況も想定される。安全性は第一であるが、それによって効率性が損なわれるというトレードオフの状況が考えられる。

2.3　オープンソースソフトウェア Autoware

自律型自動運転を行うアプリケーションはいくつか存在するが、本研究では Autoware Foundation が中心となって開発されている Autoware[1] を用いて実験を行う。

Autoware は LiDAR やカメラを主な入力とし、周辺状況を認識し、意思決定を行い、経路計画によって生成された道を走行するように車両制御を行うアプリケーションである。周辺状況認識は自車両の位置を推定する自己位置推定、周辺の障害物や歩行者や車両を認識し、その動作を予測する機能で構成されている。(図 2.1)

2.3.1　ROS(Robot Operating System)

Autoware は ROS[10] と呼ばれるロボット制御に使われるシステムをベースとしたソフトウェアである。本研究では ROS の操作を多く行うため Autoware 内での ROS の動作について述べる。

ROS はノード同士がトピックを介して互いに情報を交換する。各ノードにはロボット内での役割が与えられており、Autoware 内では自己位置推定をしてその結果を出力するノードなどが存在する。各ノードは自身が計算により得た情報をトピックに対して発行する。トピック

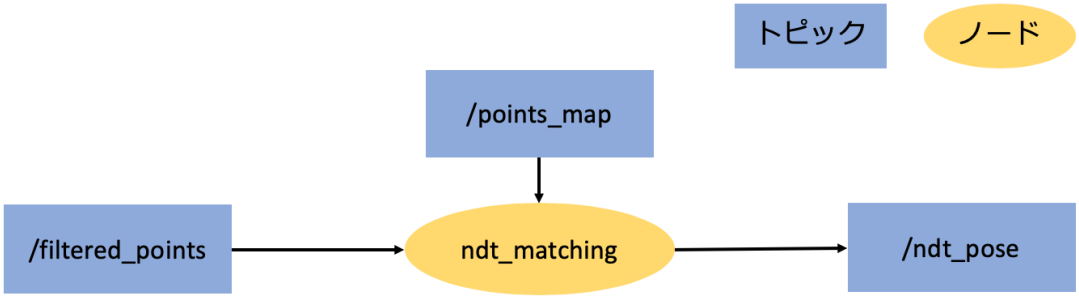


図 2.2. Autoware において自己位置推定に関するノードとトピックの関係

に対して発行された情報は他のノードが取得することができ、互いに必要な情報を交換することができる。

図 2.2 に Autoware において自己位置推定をする際のノードとトピックの関係を示す。Autoware の自己位置推定には ndt matching と呼ばれる方法が採用されている。この手法と同名の ndt_matching というノードが Autoware 内の自己位置推定を主に行っている。このノードは/filtered_points というリアルタイムのセンサデータが発行されているトピックと、/points_map というあらかじめ読み込んである 3 次元高精度地図の情報が発行されているトピックの情報を統合することにより自己位置推定をし、/ndt_pose というトピックに結果を出力することで、他のトピックが自己位置を利用することができるようになっている。

また、rosbag という機能を用いることで、トピックに発行されたメッセージを時間情報と共に収録することができ、収録されたファイルを再生することで、収録した実験を再現することができる。Autoware においてもこの rosbag の機能を使用することができ、車両に取り付けられたセンサが得た情報を時間と共に記録することができる。本研究においても過去の実験において収録された Autoware のデータを用いることでインドア実験を行った。

2.3.2 自己位置推定

自動運転のためには自己位置推定は必須の技術である。自己位置推定のためには GPS を用いた手法や、LiDAR から得られた点群データを元にしたスキャンマッチングなどの手法がある。

GPS は安価で導入コストが低いが、トンネルや屋内で位置情報を利用することができない。また、マルチパスの影響によりビルに囲まれている市街地などでは 10m 程度の誤差が生じることもある。

一方、LiDAR から得られた点群データによるスキャンマッチングは、LiDAR を設置する必要や、あらかじめ地図データを読み込んでおく必要があるというコストはあるが、数 cm 程度の推定誤差に収めることが可能である。

2.4　協調型自動運転

協調型自動運転は路側機や他の車両と通信することで周辺の状況を認識し、自動運転を行う技術である。このように通信により交通の安全化・効率化を目指すシステムのことを協調型 ITS(Cooperative Intelligent Transport System) と呼ぶ。また、車両と車両の通信を V2V 通信 (Vehicle to Vehicle)、車両と路側機の通信を V2I 通信 (Vehicle to Infrastructure) と呼ぶが、車両は様々なものと通信を行うためそれらを総称して V2X 通信 (Vehicle to Everything) と呼ぶ。

通信のためには標準化されたプロトコルが必要であり、欧州の ETSI[4]、アメリカの IEEE[5]、日本の ARIB[6] でそれぞれ標準化に向けた議論がなされている。

2.5　ETSI による標準化

ETSI により標準化されている、IEEE802.11p をベースにした V2X の規格を ITS-G5 と呼ぶ。ITS-G5 において車両や路側機などの通信を行う主体は ITS ステーション (ITS-S) と呼ばれており、ETSI により標準化されている ITS ステーションアーキテクチャーを図 2.3 に示す。ITS ステーションアーキテクチャーはアプリケーション層、ファシリティ層、ネットワーク&トランスポート層、アクセス層、マネージメント層、セキュリティ層によって構成されている。

以下、本研究において取り上げる標準化されたメッセージである CAM(Cooperative Awareness Message)[11] 及び、各車両が自車両および周辺情報を記録しておくデータベースである LDM(Local Dynamic Map)[12] について述べる。

2.5.1　CAM (Cooperative Awareness Message)

協調型 ITS において、車両や路側機は他の車両や物体などの位置情報・速度をはじめとしたデータを CAM によって交換する。CAM を発信することができる車両は自らの位置情報などをフォーマットに則った形で CAM として送信する。

ETSI により標準化されている CAM の構造を図 2.4 に示す。CAM は ITS PDU (Packet Data Unit) header[14]、Basic Container、HF (High Frequency) Container、LF (Low Frequency) Container、Special vehicle Contaier によって構成されている。

ITS PDU header にはプロトコルバージョン (現在は 1)、メッセージ ID(CAM は 2)、ステーション ID が含まれる。Basic Container にはそれぞれの ITS station の種類や位置情報などが含まれる。

全ての CAM には HF Container が含まれている必要があり、HF Contaier には車両の進行方向や速度などの動的な情報が含まれる。一方 LF Container は必須ではなく、車載灯などの静的または変化の少ない情報が含まれる。

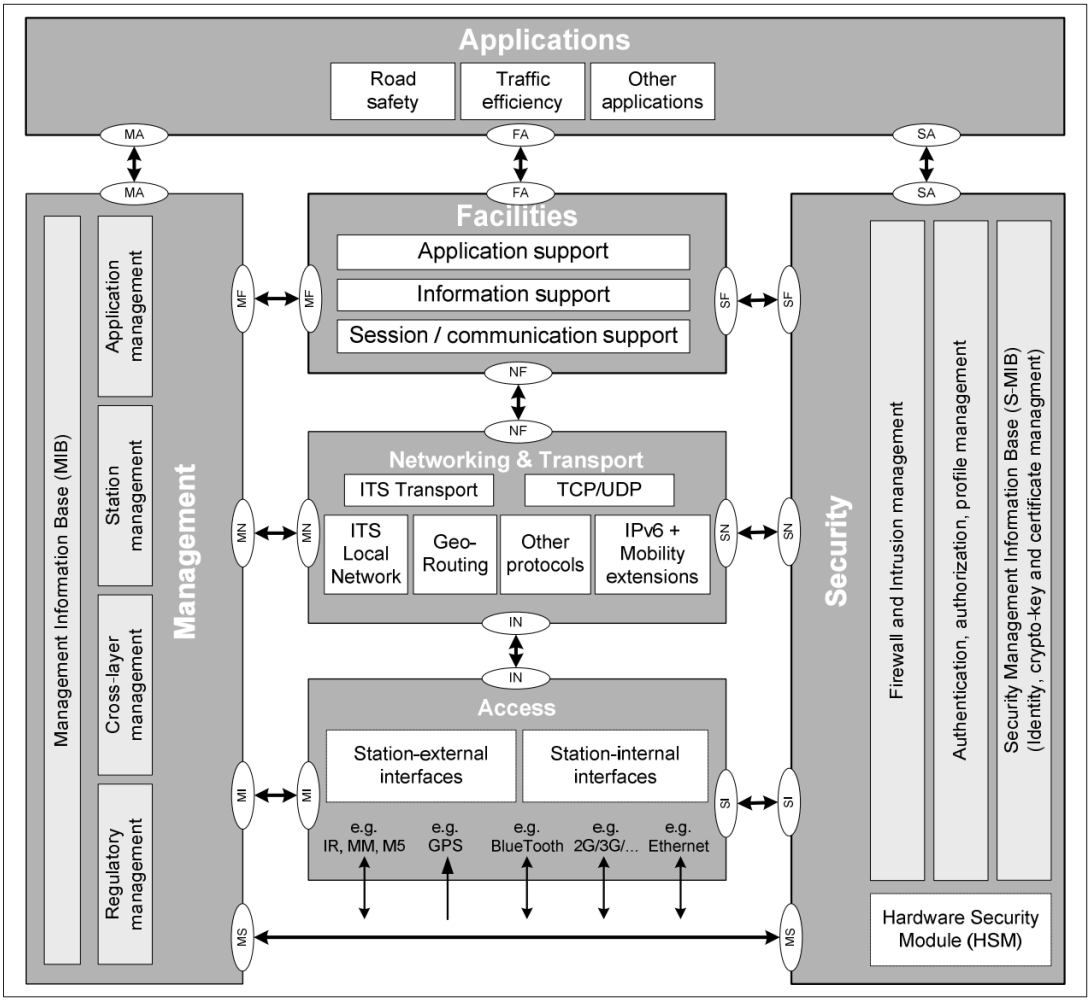


図 2.3. ETSI により標準化されている ITS ステーションアーキテクチャー構成 (出典:[13])

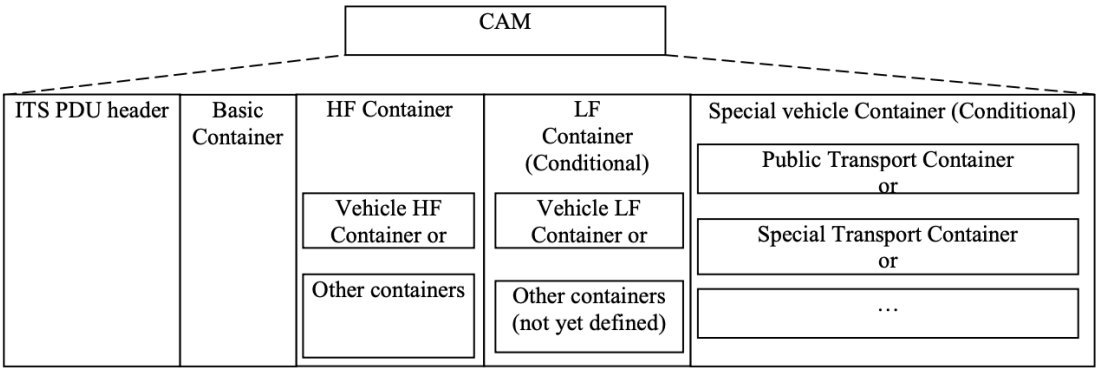


図 2.4. ETSI により標準化されている CAM の構造 (出典:[11])

2.5.2　LDM (Local Dynamic Map)

LDM は図 2.3 に示した ITS ステーションアーキテクチャーにおいてファシリティ層に位置する。LDM はアプリケーション層で動くアプリケーションに対し、周辺の物体情報を提供するためのデータベースの役割を持つ。

ITS ステーションの中で LDM に対して情報を提供するファシリティやアプリケーションのことを LDM Data Providers と呼び、LDM から情報をもらうアプリケーションのことを LDM Data Consumers と呼ぶ。これらはそれぞれ ITS ステーションの中で認証されたものでなくてはならない。認証に関わるデータ以外で LDM に格納されるべきデータは周囲の交通状況に異変が起こった場合に送信される DENM[15] や、先述の CAM が主である。

2.6　オープンソースソフトウェア OpenC2X

先述の ITS ステーションアーキテクチャーに対応した通信を実現できるオープンソースソフトウェアとして OpenC2X が存在する [16]。OpenC2X のモジュール構成を図 2.5 に示す。実線で示されたモジュールは既に実装されており、点線で示されている Security 層は未実装であり、ITS GeoNet. に関してはパケットに GeoNetworking ヘッダーをつけてはいるが、ルーティングに関しては未実装である。

また、OpenC2X にはブラウザでグラフィカルにその動作を確認する機能も備わっている 2.6。デフォルトの OpenC2X では OBD2 や GPS から得た情報を元に CAM や DENM を生成し、その情報を LDM に格納すると共に DCC(Decentralized Congestion Control) を通じて指定したインターフェースから MAC アドレス ff:ff:ff:ff:ff:ff に対してデータリンク層レベルでシングルホップブロードキャストされる。また、他の車両や路側機から送られてきた CAM や DENM を受け取り、LDM に格納する。実際に送信されているパケットを wireshark にて観測した様子が図 2.7 である。

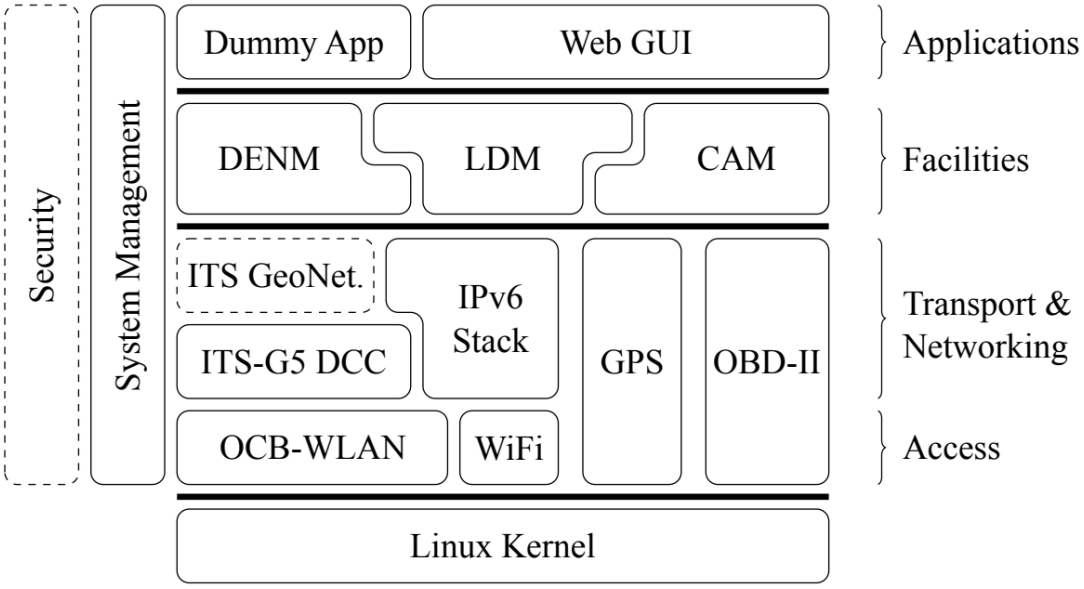


図 2.5. OpenC2X のモジュール構成 (出典:[17])

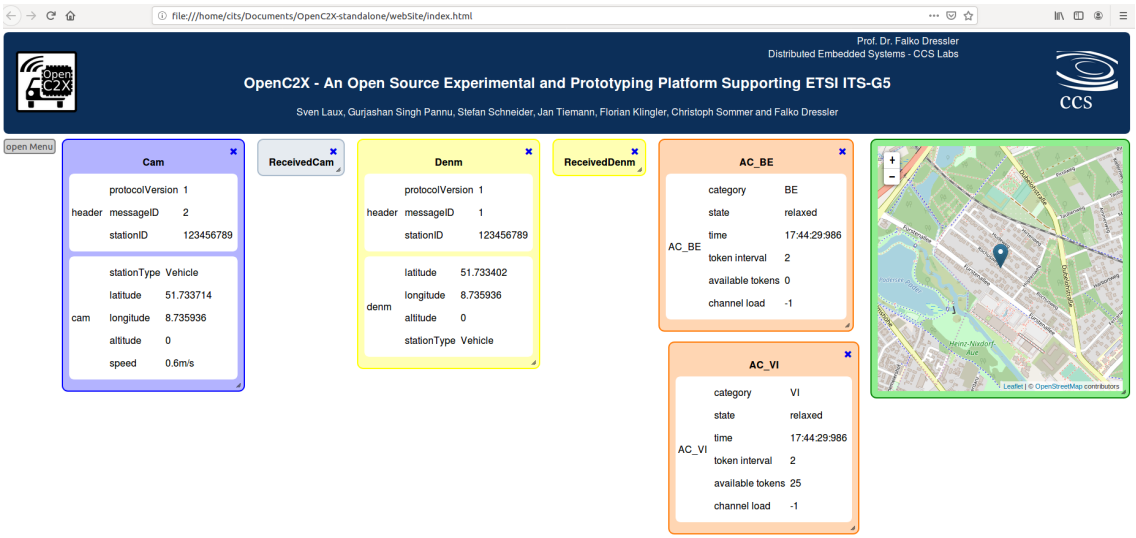


図 2.6. OpenC2X が送受信しているパケットを GUI で見た様子

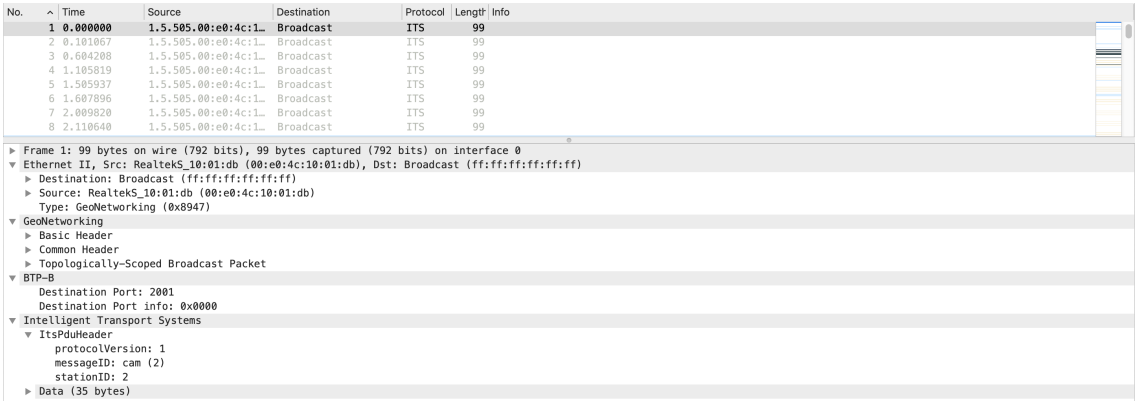


図 2.7. wireshark で OpenC2X が送信しているパケット観測している様子

第 3 章

関連研究

自車両以外の物体などの情報を共有する手法がすでにいくつか提案されている。本章ではそれらについて紹介するとともに、本研究において使用する Autoware と OpenC2X をそれぞれ用いた研究について紹介する。

3.1 生センサデータの共有

車両間の情報共有手法として、生のセンサデータを全て共有するものがある。CarSpeak[18] は空間を再帰的に立方体状に区切り、そのブロックごとに点群データを管理し、他の車両と共有するシステムである。

この研究では、Octree と呼ばれる手法で空間を再帰的に分割し、名前をつけることで効率的なデータ圧縮を可能にし、車のように絶えず変化するネットワーク環境に対してもパケットロスを抑えることができるという。さらに、MAC 層では場所に紐づいた情報を元にやりとりをし、場所ごとのリクエスト回数を記録しておくことで重要度の高い場所を優先的に通信することを可能にしている。

生データの共有よりも、CAM のような抽象化した情報を送るのが効率的な通信ができるのではないかという議論については、どのアプリケーションがどのような情報を必要としているかわからないため、生データを送るべきだという立場をとっている。

他にも生センサデータを共有する研究は存在する。Kim[19] らはカメラから得られたデータと LiDAR から得られたデータを共有することで互いに認識する手法を提案した。この研究では他の車両から見える情報と自車両から見える情報の角度のズレを算出することで座標変換を行い、他の車両から送られてきた速度情報を元にその車両を特定する。

この研究では LiDAR のみの情報を共有した場合、時速 100km の車両であっても平均推定誤差が 0.122m に収まることが示されている。また、遅延に関しては LiDAR データのみを送信した場合が最も遅延が小さく、カメラデータと組み合わせると遅延が大きく成っている。しかし、この実験は車両が数台での実験であるため、この台数が増えていくにつれて通信負荷がどのように誤差に影響を与えるのかは明らかではない。

3.2　Proxy CAM

自動運転の実用化が進む中で、道路を走る自動車が自動運転車と非自動運転車が混在するという過渡期が存在する。その中で、CAM を自ら発することができない車両や歩行者の代わりに、それらを認識した RSU(Road Side Unit) が代わりに CAM を発するシステムとして Proxy CAM[20] が提案されている。[20] では、RSU がカメラにより車両を検知し、そこから位置情報や速度などを推定し CAM を生成し、送信するプロトタイプを作成し、性能評価を行っている。

この研究のプロトタイプでは優先道路から交差点に 1 台の車両が進入するシナリオを実装している。結果として交差点の 70m 手前で RSU が車両を検知し、100m 先まで CAM を送信することが可能であることが示されている。また、20Hz までならば CAM を正常に生成・送信でき車両情報の生成から CAM の受信まで 0.22 秒以内の遅延で収まることが示されている。

また、より広範囲で Proxy CAM を交換することで各車両がより大域的な道路情報を得るためのシステムとして Grid Proxy CAM[21] が提案されている。この研究では各交差点に設置された RSU が互いに有線接続され、シミュレータにより性能評価が行われた。また、より重要な情報の優先的に車両に送信する Distance Priority　Algorithm も提案されており、これにより自車両から近くの車両（これを重要度の高い車両と定義している）についてはパケットロスが少ないことが示されている。

3.3　Autoware を用いた研究

Autoware は複数の特徴的な要素技術から成っている。それらの要素技術を他のシステムに組み込む研究や、ベンチマークとして採用している研究が存在する。

Autoware では自己位置推定の際に NDT という点群マッチング手法が用いられているが、Shen[22] らは真相学習をベースとしたより効率的な dynamic local NDT という手法を提案し、この手法のベンチマークとして Autoware で用いられている NDT マッチングを採用している。

Sasaki[23] らは Autoware で自動走行する際の処理をエッジサーバとクラウドサーバにそれぞれ役割を分けて分散処理させることによる性能評価を行った。ここでエッジサーバとクラウドサーバは 5G 通信を想定した 1Gbit/s で通信を行っている。これによりエッジサーバとクラウドサーバを協調させることは、エッジサーバのみに処理をさせるよりも大幅に実行時間とタイムアウト率を改善させたが、クラウドサーバのみに処理をさせるよりもわずかに悪化したことが明らかになった。

Autoware をシミュレーションに用いる研究も存在する。Aparow[24] らは車両のダイナミクスを再現することができる CarMaker と呼ばれるシミュレータと、Autoware の経路計画機能を組み合わせてより物理的な再現度の高い自動運転シミュレータを作成した。また、Tokunaga[25] らは車両のシミュレーションで多く使われる MATLAB/Simulink と、

Autoware を組み合わせることでシミュレーションとフィールド実験を同時に行うことができることを示した。

以上のように Autoware の要素技術を他のソフトウェアと組み合わせる研究が存在しているが、Autoware により制御されている車両が ITS-G5 に則って通信を行う研究は存在していない。

3.4 OpenC2X を用いた研究

協調型 ITS の分野においては Artery[26], Veins[27], VSimRTI[28] といった通信環境を数多く考慮することができる V2X シミュレータが存在する。また、V2X 用の車両を多く用意することが困難であることからシミュレーションによるネットワークの性能評価に関する研究が多く存在する。しかし、実世界の通信では数多くの物理的な制約が存在し、それら全てをシミュレーションにおいて考慮することはできない。したがって、シミュレーションの結果を必ずしも実世界に当てはめることはできない。シミュレーションと同様のシナリオをフィールド実験で行い、結果を比較する必要がある。そのような問題に対応するため、OpenC2X はフィールド実験のためにいくつかの研究で用いられている。ここではそれらの研究について述べる。

Adrian[29] らは自動車の隊列走行において 700MHz 帯を用いた場合と 5.9GHz 帯を用いた場合の比較を行った。ブカレストの公道にて 2 台の車両による隊列走行を行い 700MHz の方が冗長性が高いため、5.9GHz 帯を補完する役割として用いるべきだという結果を得ている。

Francesco[30] らは 802.11p で通信する NIC(Network Interface Card) の性能評価を行うためのオープンソースフレームワークを作成し、MAC 層での処理に OpenC2X を用いている。

走行している車両の速度などといった様々な情報を FCD(Floating Car Data) と呼ぶ。Ion[31] らは FCD を共有するプロトコルとして BASELINE と DISCOVER を取り上げ、OpenC2X を用いてフィールド実験を行い、BASELINE に対する DISCOVER の有効性を示した。

以上のようにネットワークに関する手法の検証のために OpenC2X が用いられてきた。一方、自律型自動運転ソフトウェアとの統合に関してはまだ研究がなされていない。

第 4 章

AutoC2X

自律型自動運転と協調型自動運転が統合されることにより、各自動車の認識領域を拡張するためのシステムを本研究では提案する。本研究では自律型自動運転ソフトウェアとして Autoware を、協調型自動運転ソフトウェアとして OpenC2X を用い、それらを拡張することにより統合する。この統合するためのシステムを本研究では AutoC2X と呼ぶ。本章では AutoC2X について述べる。

4.1 要求事項

本研究の目的は協調型自動運転を自律型自動運転に統合することで、自律型自動運転の問題点を解消することである。そこで現状の自律型自動運転における問題点から達成すべき 4 つの要求事項が導かれる。その要求事項について述べる。

- 安全性
  自動運転車に複数の自律型自動運転アプリケーションがインストールされている場合、それぞれのアプリケーションが自由に通信をすると、どのアプリケーションがどのような通信をしているのかをまとめて把握することができない。特定のアプリケーションがウィルスに感染して、意図しない通信を大量に発生させるなどした場合、従来のアプリケーションの接続方法だとそれを検知することができない。
- 拡張性
  先述の通信の管理とも関連するが、各アプリケーションが別々の通信インターフェースを要求している場合、不都合が生じることがある。例えば、商業施設などで Wi-Fi が使用できる場合、あるアプリケーションが意図せずその Wi-Fi に接続してしまうということが考えられる。我々が普段スマートフォンを使用していて、意図しない Wi-Fi アクセスポイントに接続してしまうように、自動運転車においても意図しない接続というものが発生すると考えられる。
  また、仮に上記の問題が解決できたとしても各アプリケーションのために Wi-Fi や SIM カードなどの受信機を接続するのはコストがかかり、ネットワーク的に推奨され

ない。

- 標準への準拠
  すでに協調型自動運転においては欧州、アメリカ、日本でそれぞれ車々間通信における通信の標準化が進められていることは述べた。各アプリケーションが独自の通信をする場合、そのアプリケーションが得た情報を他のアプリケーションが使用するのは難しい。各アプリケーションが共通のフォーマットで通信をが行えば、お互いが得た情報を共有し運転に役立てることができる。
- 死角にある物体の認識
  本研究の目標は自律型自動運転アプリケーションの認識領域の拡張である。その目標の達成のためには、自車両から見えない位置にある物体をリアルタイムに認識することが必要である。

以下、これらの要求事項を実現する手法について述べる。

4.2 AutoC2X の概要

本研究では自律型自動運転ソフトウェアにより周辺の車両や物体を認識し、その情報を緯度経度情報に変換し、協調型自動運転ソフトウェアを用いてその情報を共有する。これにより自律型自動運転ソフトウェアのみでは認識できない領域の情報を取得することができ、加えて自ら情報を発信する手段を持たない自動運転非対応車や物体、歩行者の情報を共有することができる。

想定シナリオを図示すると図 4.1 のようになる。本研究の実験ではセンサを備えた 1 台の路側機が情報を発信する役割を持ったシナリオを設定しているが、この役割は路側機ではなく個々の車両が持っていても同じである。

送信側が物体を検知してから、その情報が受信側に認識されるまでの具体的な流れを以下に示す。

送信側の動作の流れは以下の通りである。

1. 自身が自動車の場合、自律型自動運転ソフトウェアが自車位置を算出する。路側機の場合、あらかじめ固定された位置、向きの情報を常に保持しておく
2. 周囲の物体を検知する
3. 算出された自車位置を元に、周囲の物体情報を緯度経度のデータに変換する
4. 緯度経度に変換されたデータを協調型自動運転ソフトウェアに送る
5. 協調型自動運転ソフトウェアは送られてきた情報を元に標準に従った形式でメッセージをブロードキャストする

受信側の動作の流れは以下の通りである。

1. 送られてきた情報を協調型自動運転ソフトウェアが自律型自動運転ソフトウェアに送る
2. 自律型自動運転ソフトウェアは送られてきた情報から緯度経度や stationID を取得し、

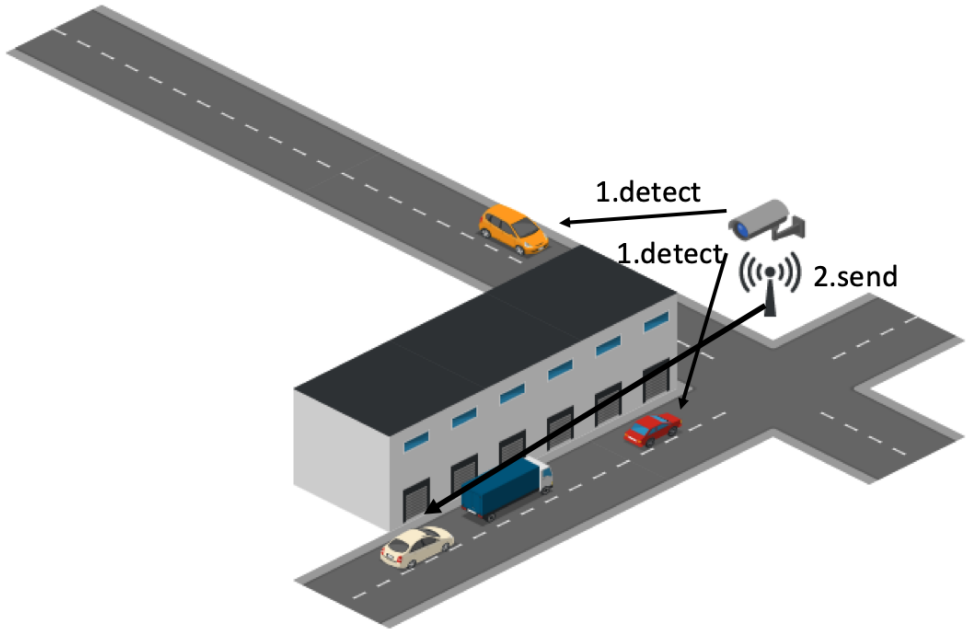


図 4.1. 提案手法の概要と想定シナリオ

座標変換することで自車両から見た物体の位置を算出する

4.3　本手法の設計

本研究では自律型自動運転ソフトウェアと協調型自動運転ソフトウェアを統合するが、統合のための設計を述べる。

4.3.1　ルータとホストの分離

本研究においては協調型自動運転ソフトウェアはルータで、自律型自動運転ソフトウェアは laptop PC で動く。つまり、ルータとホストが分離された状態で本手法は機能する。これによりルータに対して複数のホスト・アプリケーションを追加していくことができ、要求事項の拡張性が確保される。また、これにより通信の管理を行いやすくなり安全性の向上も達成される。本手法の概要を図 4.2 に示す。

また、ルータとホストの分離は協調型 ITS と自動運転の機能分担でもある。

ルータは外部との接続性を確保し、ETSI により標準化されている V2X の通信方式である ITS-G5 に則った通信を行うという機能を持つ。具体的には LDM に車両情報を保存し、CAM を生成して BTP/GN により通信をするなどのことである。ここで BTP(Basic Transport Protocol) は UDP と似た最小限の通信を行うプロトコルであり [32]、GN(GeoNetworking)

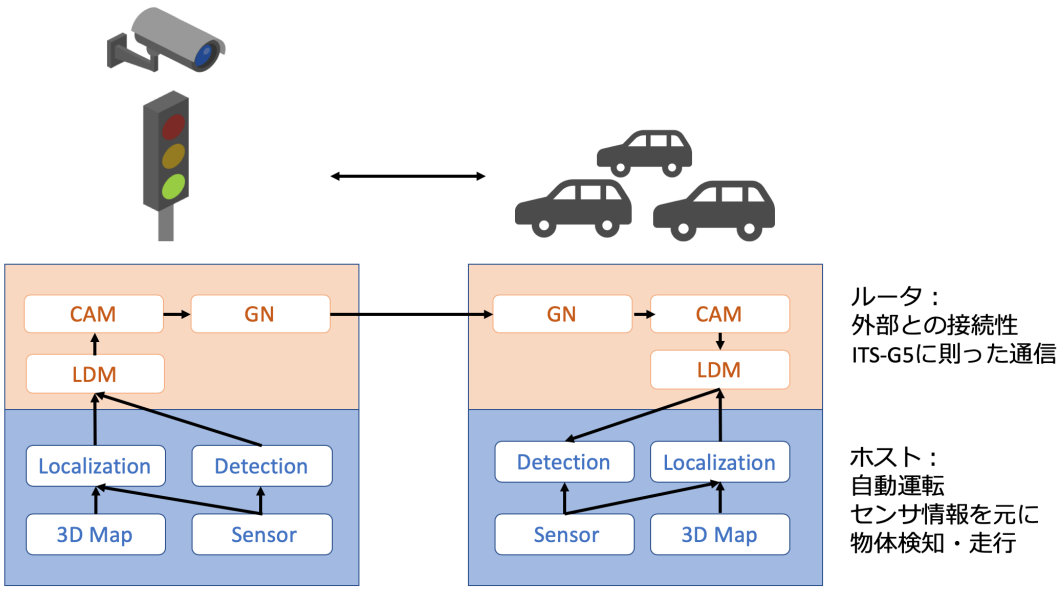


図 4.2. 提案手法の概要

は位置情報に基づいたルーティングを行うプロトコルを指す [33]。

一方、ホストはセンサ情報を元に物体を検知し、安全に走行するという役割を持つ。

4.3.2 Autoware と OpenC2X の統合

本研究では自律型自動運転ソフトウェアとして Autoware を、協調型自動運転ソフトウェアとして OpenC2X を採用し、これらを統合することにより提案したシステムを実現する。これらを用いることで要求事項にある標準への準拠と死角にある物体の認識を実現することができる。

4.3.3 自己位置推定の統一と精度向上

本来 OpenC2X には GPS モジュールが含まれており、GPS により自己位置推定を行うことが前提となって設計されていた。しかし、Autoware は高精度地図を用い、LiDAR により検知されたデータと結び付けることで自己位置を推定するため、GPS よりも自己位置推定の際の誤差を小さくすることができる。そこで本研究では OpenC2X に本来備わっている GPS モジュールではなく、Autoware による自己位置推定を行うように設計した。2.3.2 項で述べた通り、GPS による推定誤差よりも LiDAR で得られた点群データによるスキャンマッチングの方が推定誤差を遥かに小さくすることができるため、本来の OpenC2X よりもより正確な位置情報を共有することができる。

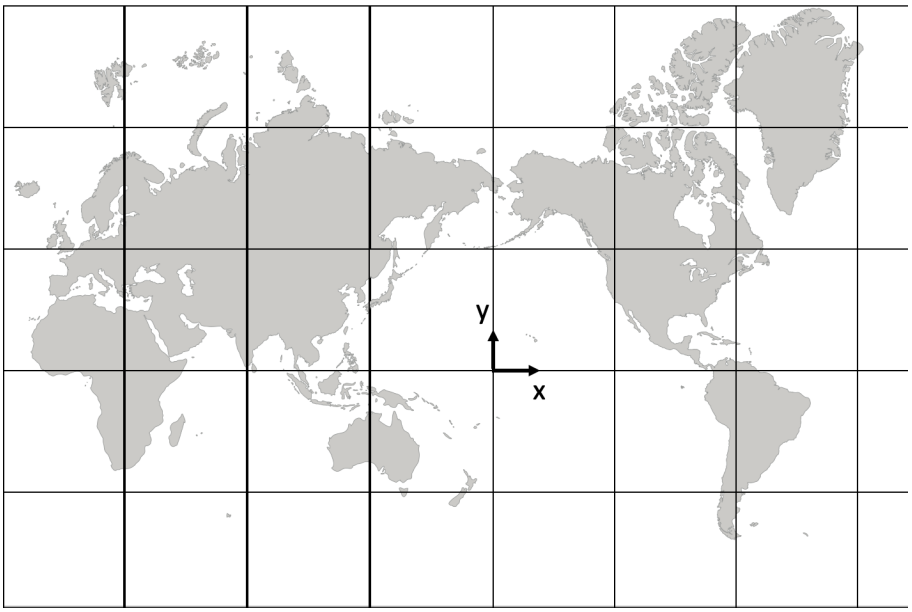


図 4.3. 地理座標系をセルに分割し、新たな座標系を定義する例

4.4　座標変換

自律型自動運転ソフトウェアと協調型自動運転ソフトウェアが利用する情報の大きな違いとして座標系の違いがある。自律型自動運転ソフトウェアはその車両から見た座標系で基本的な判断を行う。一方、協調型自動運転ソフトウェアは全ての車から見て共通の座標系、すなわち緯度経度を含んだメッセージを交換する。

したがって、協調型自動運転ソフトウェアから得た情報を自律型自動運転ソフトウェアはそのまま利用することができず、緯度経度から座標変換をして情報を認識しなくてはならない。自律型自動運転ソフトウェアから協調型自動運転ソフトウェアに情報を送る場合は逆に、自車位置や自車から見た物体情報を緯度経度の形式に変換しなくてはならない。

全ての車両が同様に情報を認識できる座標系として緯度経度で表される地理座標系が存在する。さらに図 4.3 のように地理座標系を一定の規則に従ってセルに分割して、その中で座標系を定義する座標系も存在する。このような座標系の例として UTM 座標系 [34] などがある。分割された座標系は、同じ座標系で認識を行っている車両同士は同様に認識できる。より小規模な座標系になるにつれて位置関係をより正確に表すことができるが、その座標系から出てしまうと別の座標系に切り替えなくてはならない。

一方、自律型自動運転ソフトウェアにおいて周辺状況を認識する際、周辺の物体などは自車両を中心とした座標系で把握される。すなわち、全ての車両で別々の座標系が定義され、それをそのまま他の車両と共有することはできない。

そこで本研究では自律型自動運転ソフトウェアがそれぞれ独自に定義する座標系を、協調型自動運転ソフトウェアが利用できる形に座標変換を行う。図 4.4 は 2 車両が共に認識できる座

標系 (図においては xy 座標) と、それぞれの車両が周辺状況を認識する座標系 (図においては x'y' 座標、x"y" 座標) を表している。

Car A,B が Object 1 を認識する際、Car A は (x_1', y_1') という座標で認識し、Car B は (x_1'', y_1'') という座標で認識する。これをそのままの座標で共有してもお互いに利用することができないため、共通して利用できる xy 座標に直す必要がある。

ここで Car A は xy 座標に対して α 傾いているので、Object 1 の xy 座標における表し方は

$$\begin{aligned}\begin{pmatrix} X_1 \\ Y_1 \end{pmatrix} &= \begin{pmatrix} X_A \\ Y_A \end{pmatrix} + R(\alpha) \begin{pmatrix} x_1' \\ y_1' \end{pmatrix} \\ &= \begin{pmatrix} X_A \\ Y_A \end{pmatrix} + \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x_1' \\ y_1' \end{pmatrix} \\ &= \begin{pmatrix} X_A + x_1'\cos\alpha - y_1'\sin\alpha \\ Y_A + x_1'\sin\alpha + y_1'\cos\alpha \end{pmatrix}\end{aligned} \tag{4.1}$$

というように-α の回転行列をかけることで xy 座標に直すことができる。一方、この情報を受け取った Car B は以下の手順で自車から見た位置 (x_1'', y_1'') に変換する。

$$\begin{aligned}\begin{pmatrix} x_1'' \\ y_1'' \end{pmatrix} &= R(-\beta) \begin{pmatrix} X_1 - X_B \\ Y_1 - Y_B \end{pmatrix} \\ &= \begin{pmatrix} \cos(-\beta) & -\sin(-\beta) \\ \sin(-\beta) & \cos(-\beta) \end{pmatrix} \begin{pmatrix} X_1 - X_B \\ Y_1 - Y_B \end{pmatrix} \\ &= \begin{pmatrix} (X_1 - X_B)\cos(-\beta) - (Y_1 - Y_B)\sin(-\beta) \\ (X_1 - X_B)\sin(-\beta) - (Y_1 - Y_B)\cos(-\beta) \end{pmatrix}\end{aligned} \tag{4.2}$$

のようになり、Car B は Car A から送られてきた Object1 の情報を認識することができる。

Car A が Car B に Object 1 の位置情報を送る手順を図 4.4 に示した値を用いると以下の通りになる。

1. Car A が Object1 を (x_1', y_1') として認識する
2. (x_1', y_1') を式 4.1 に従って (X_1, Y_1) に変換する
3. (X_1, Y_1) を座標系に応じた変換方法により緯度経度に変換する
4. Car A が Car B に緯度経度を送信する
5. Car B は受け取った緯度経度を座標系に応じた変換方法により (X_1, Y_1) に変換する
6. (X_1, Y_1) を式 4.2 に従って (x_1'', y_1'') に変換する
7. Car B が Object1 を (x_1'', y_1'') として認識する

以上が自律型自動運転ソフトウェアと協調型自動運転ソフトウェアによる情報共有のための座標変換手法である。次に本研究において Autoware で得られた位置情報を OpenC2X で利用できる形に変換する過程を示す。

Autoware において今回用いたデータでの自己位置は横メルカトル図法によって表される (この座標系は Autoware で用いる高精度地図に依存する)。これを緯度経度に変換することで

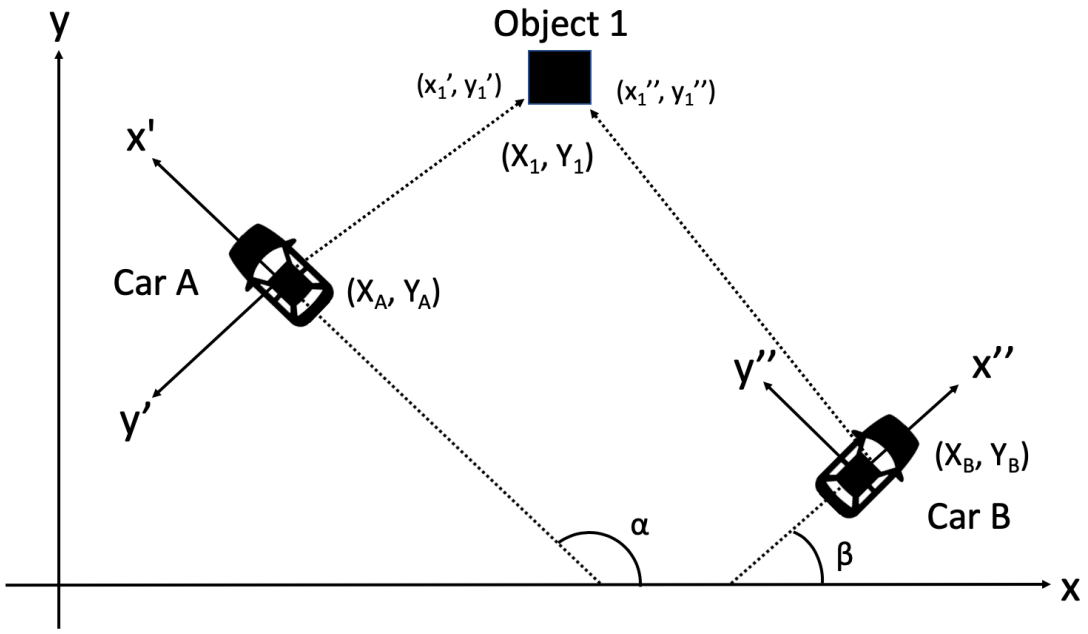


図 4.4. それぞれの車両独自の座標系と共通の座標系

OpenC2X で利用できる形となる。以下、横メルカトル図法によって表される座標系を横メルカトル座標系と表す。

Autoware において物体認識は車両の進行方向を x 軸とした座標系で行われる。また、Autoware では常に自車両で定義している座標系が横メルカトル座標系に対してどの程度傾いているのかをクォータニオンの形式で常に算出している。これをオイラー角に変換し、回転行列の形にすることで、横メルカトル座標系において物体が車両に対してどの位置に存在するかという形式にすることができる。

このようにして、Autoware において認識される自車両の位置及び物体の位置を緯度経度に変換することができる。

第 5 章

実装

本研究では先述の提案手法を評価するためのシステムを実装した。AutoC2X は Autoware と情報を交換する AutoC2X-AW と OpenC2X と情報を交換する AutoC2X-OC に分けることができる。

AutoC2X-AW は PC 側 (ホスト側) で動き、Autoware が扱う ROS ノードへの情報の発行及び読み出しを行う。また、協調型 ITS で用いられる緯度経度座標と Autoware で用いられる独自の座標系との変換を行う。

AutoC2X-OC はルータ側で動き、OpenC2X と AutoC2X-AW でそれぞれ使用される異なる形式のメッセージ型を変換する機能を持つ。さらに、CAM の生成周期に合わせて最新の情報を保持しておく機能を持つ。

5.1 実装で用いたソフトウェア・ライブラリについて

本研究で用いたソフトウェア及びライブラリのバージョンを表 5.1 に示す。AutoC2X に関しては C++ で実装している。

名前	バージョン
Autoware	1.11.0・1.11.1
OpenC2X-standalone	1.5
boost[35]	1.58
libproj[36]	4.9.2-2

OpenC2X に関しては、standalone 版 [37] と、OpenWrt からのフォークである embedded 版 [38] が存在するが、今回は standalone 版のバージョン 1.5 を独自に拡張したものを使用する。[39]

5.2　システム概要

本研究で実装したシステム概要を図 5.1 に示す。路側機 (RSU) が周辺の物体を認識して、それを車両が受け取るまでの流れは以下のようになる。

1. 路側機の Autoware において LiDAR から得た点群情報とあらかじめ読み込まれている高精度地図 (3D Map) を元に、自車位置、自車の向き、物体位置を検知する。
2. AutoC2X-AW がその情報を受け取り、座標変換をして各情報を緯度経度に直す。
3. ルータにメッセージをシリアライズして Ethernet 経由で送信する。
4. AutoC2X-OC においてメッセージをデシリアライズし、OpenC2X に対して Protocol Buffers の形で送信する。
5. OpenC2X は AutoC2X-OC から受け取った情報を元に CAM を生成し、DCC を通じて車両側に CAM を IEEE802.11b により送信する。本来 OpenC2X では GPS の情報を元に CAM を生成するが、本手法では Autoware で計算された精度の高い緯度経度を元に CAM を生成する。同時に LDM に位置情報、タイムスタンプなどを格納する。
6. 車両側の OpenC2X で CAM を受け取り、その情報を Protocol Buffers の形で AutoC2X-OC に送る。
7. AutoC2X-OC は受け取った情報をシリアライズし、Ethernet 経由で PC 側の AutoC2X-AW に送信する。
8. AutoC2X-AW は受け取ったメッセージをデシリアライズし、緯度経度情報を得る。
9. Autoware から自己位置、自車の向きを得て、その情報を元に受け取った物体情報を車両から見た座標系に変換する。

5.3　AutoC2X-AW の概要

本研究で実装した AutoC2X-AW の概要を図 5.8、に示す。各部について説明する。

5.3.1　Autoware において周辺情報を認識する方法

Autoware は主にカメラと LiDAR の情報をセンサーフュージョンすることで周囲の物体を認識する。具体的な手順は図 5.3 に示す通り、カメラ側では画像認識、LiDAR 側では点群データのクラスタリングにより物体を認識し、これらの情報を統合することでより正確な物体認識を行う。図において、/image_raw に対してはカメラ画像が入力され、/points_raw に対しては LiDAR の点群データが入力される。それぞれのデータが図に示した通りの処理をされ、range_vision_fusion ノードによりこれらのデータが統合され、最終的に認識された物体は/detection/objects トピックに発行される。認識された物体は naive_motion_predict ノードにより今後の動きを予測される。

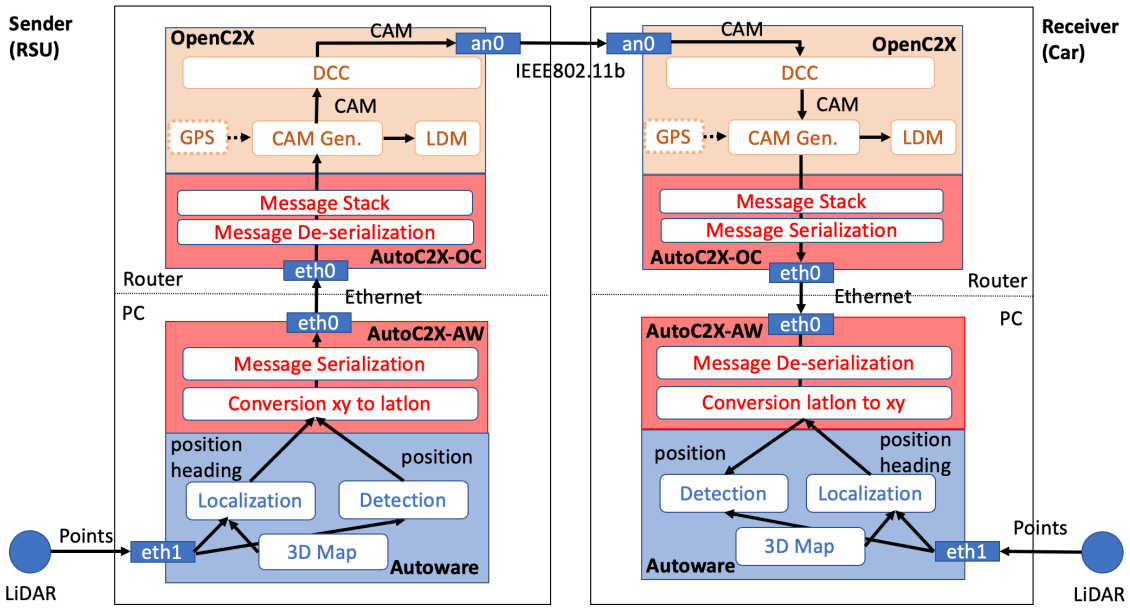


図 5.1. 本研究で実装したシステム概要

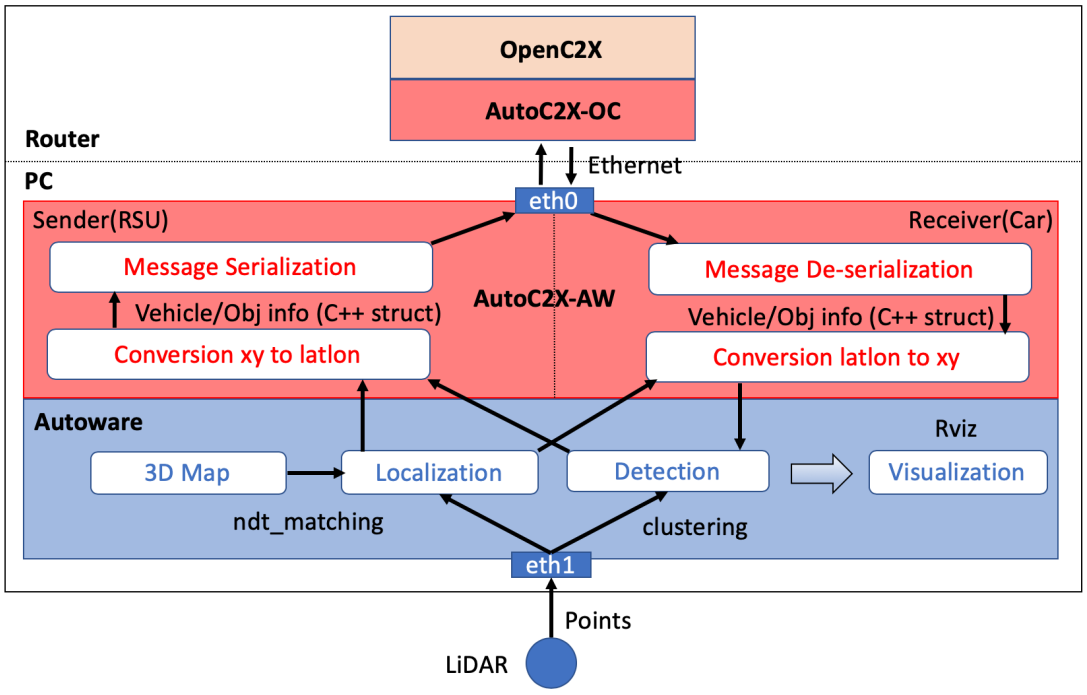


図 5.2. AutoC2X-AW の実装概要

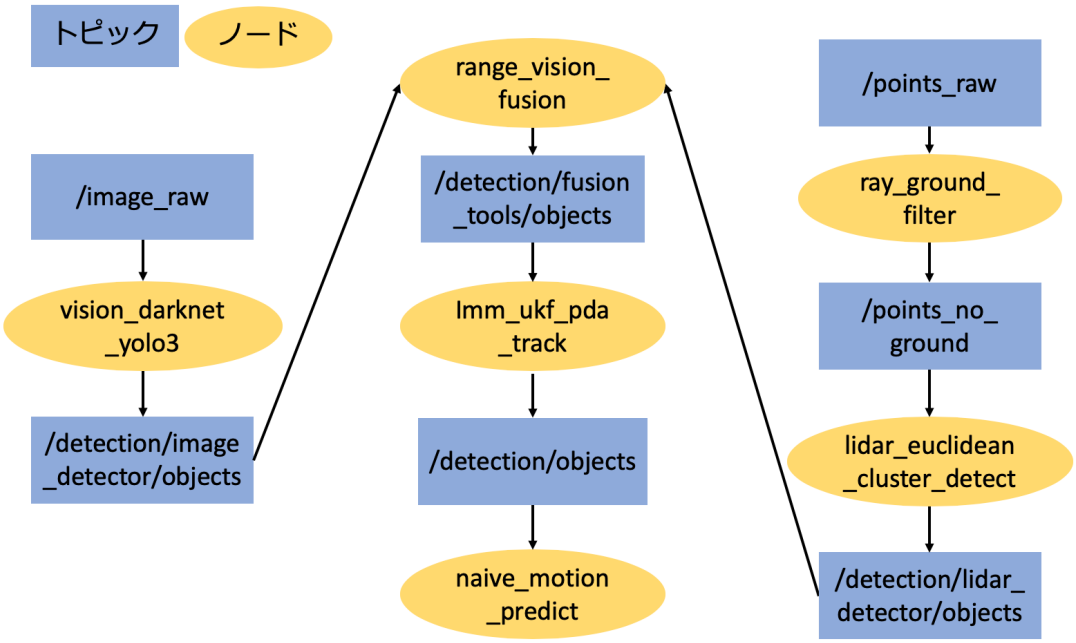


図 5.3. Autoware の物体認識の際に行われるセンサーフュージョンのトピックとノードの関係

しかし、本研究ではカメラを使用せず LiDAR の点群データのクラスタリングのみで物体認識を行う。したがって本研究における認識の手順は図 5.6 のようになっている。カメラ画像とのデータ統合がないため、range_vision_fusion ノードを介さずに物体認識を行っている。これを rviz という可視化ソフトウェア上で可視化すると図 5.5 のようになり、交差点の中央付近にいる歩行者の小さな点群も物体として認識できていることがわかる。/detection/objects トピックに対し物体情報が発行されるため、座標情報を AutoC2X-AW に送る。

5.3.2　座標変換

提案手法で述べた通り、Autoware においてそれぞれの車両は自車両から見た座標系を用いて周辺状況を認識しているので、そのまま他の車両と位置情報を交換することができない。そこで本研究では Autoware で得た位置情報を適切に座標変換することで、他の車両でも位置情報を利用できる形にしてから OpenC2X で情報共有する。以下、実装の上での具体的な手順を述べる。

送信側の AutoC2X-AW での座標変換であるが、Autoware において自車位置に関しては/ndt_pose トピックに対して発行されており、この値を用いる。この値は今回用いた本郷キャンパスの高精度地図では横メルカトル座標系で表されている。

Autoware において自車両の地理座標系に対しての傾きは/tf トピックに対して逐次発行されている。そこではクォータニオンの形式で傾きが表されており、これをオイラー角に変換する。/detection/objects トピックに発行された物体情報は車両の進行方向を x 軸とした座標系

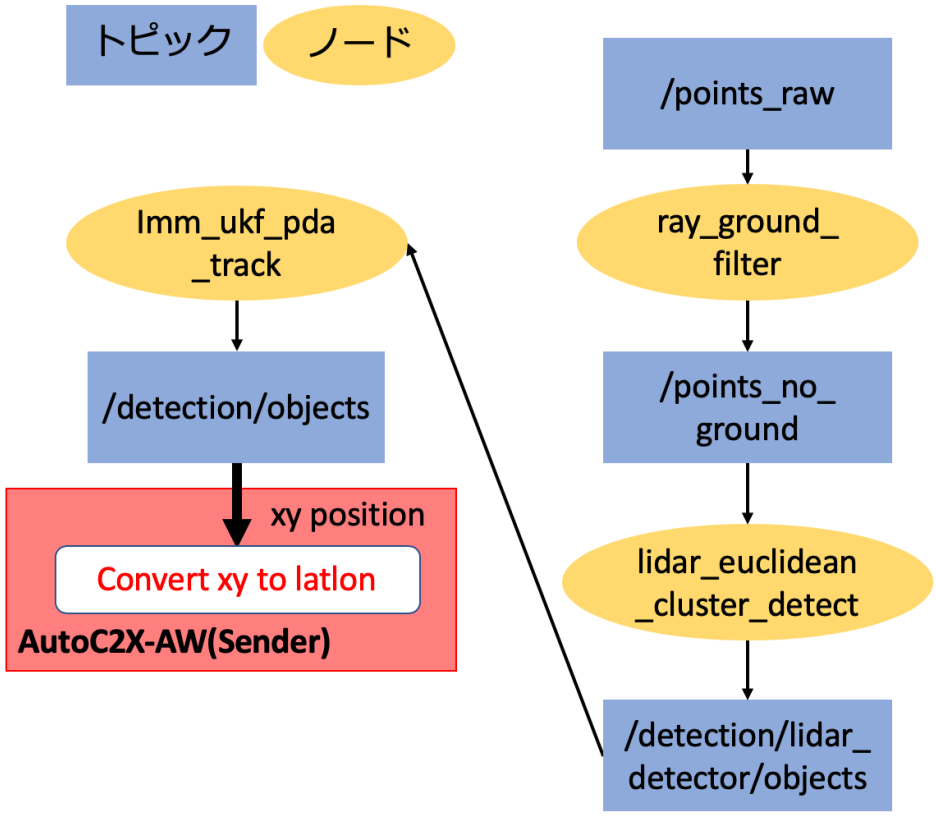


図 5.4. 本研究において路側機側の物体検知に使用したノードとトピックの関係。LiDAR のみの情報を用いて物体検知を行っている。そしてその情報を AutoC2X-AW に送る。

で表される。回転角、車両から見た物体の位置、自車位置を適切に提案手法で示した通り変換することで各物体の横メルカトル座標系での座標を求めることができる。

次に横メルカトル座標系で表された自車位置と物体位置を緯度経度に変換する。この変換は座標変換ライブラリの libproj によって行う。今回行う実験は全て東京大学本郷キャンパス内のデータを用い、本郷キャンパスは横メルカトル図法においては Japan Plane Rectangular CS IX というエリアに含まれる。したがって、libproj では以下のパラメータを用いることで本郷キャンパスにおいては横メルカトル図法と地理座標系を変換できる。

proj=tmerc lat_0=36 lon_0=139.8333333333333 k=0.9999 x_0=0 y_0=0 ellps=GRS80 units=m

次に受信側の AutoC2X-AW での座標変換であるが、AutoC2X-AW はルータから送られてきたメッセージをデシリアライズする。デシリアライズされたメッセージは路側機周辺の物体情報を緯度経度の形で表しているため、Autoware で認識するには座標変換を行う必要がある。以下、その際に行う座標変換を示す。

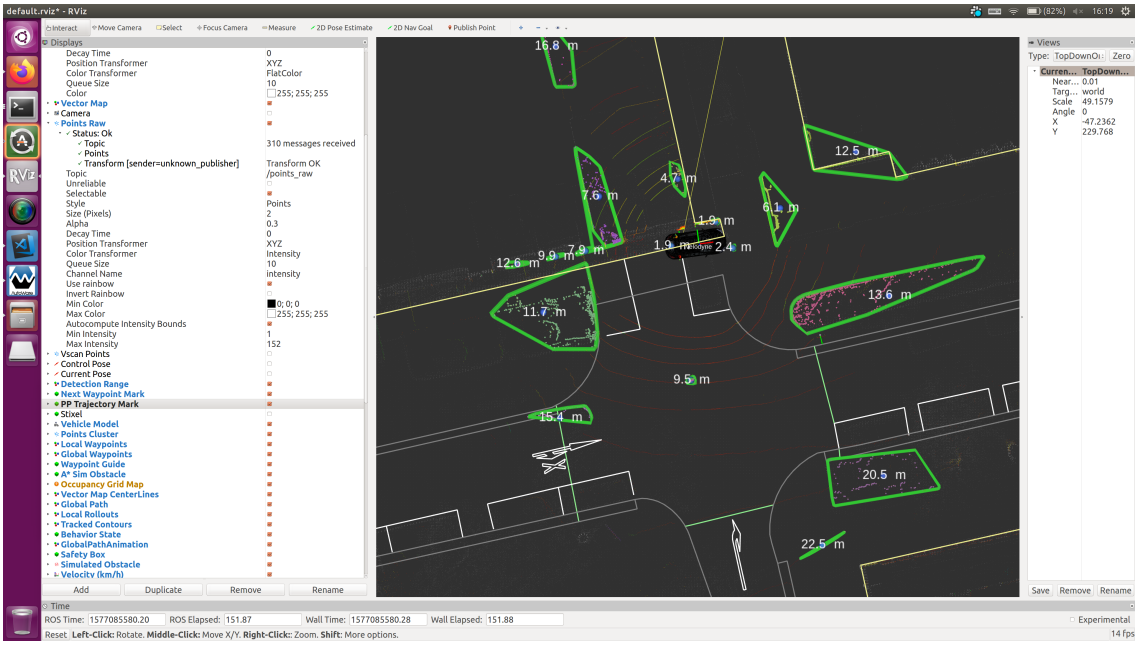


図 5.5. Autoware で点群情報を認識し、それをクラスタリングすることで周辺の物体や歩行者を認識している様子

最初に、受信側の Autoware が/ndt_pose トピックに発行する横メルカトル座標での自車位置を取得する。次に、/tf トピックに発行する横メルカトル座標に対する傾きを取得する。そして路側機から送られてきた緯度経度情報を libproj を用いることにより横メルカトル座標に変換し、これらの情報から送られてきた物体情報が自車から見てどの位置にあるのかを算出する。得られた物体位置は/detection/objects トピックに対して発行されることで Autoware が物体として認識できる。

5.3.3　AutoC2X-OC との通信

5.3.4　受信した物体情報の可視化

路側機側で認識した物体情報を離れた場所の車両が認識する様子を図 5.7 に示す。ここでは画像中央下部を受信側の車両が走行しており、画面上部にある路側機がその周辺の物体を認識している。図 5.5 とは元データが収録された時間は違うが、路側機の位置は同じである。画像上部に多くの黄緑色の正方形が表示されているが、それらが路側機より送信されてきた物体情報である。本研究では判別しやすいように送られてきた緯度経度を中心とした正方形で示しているが、大きさや車両タイプを CAM に含めることでより正確な情報を送ることができる。

5.4　AutoC2X-OC の概要

本研究で実装した AutoC2X-OC の概要を図 5.8、に示す。各部について説明する。

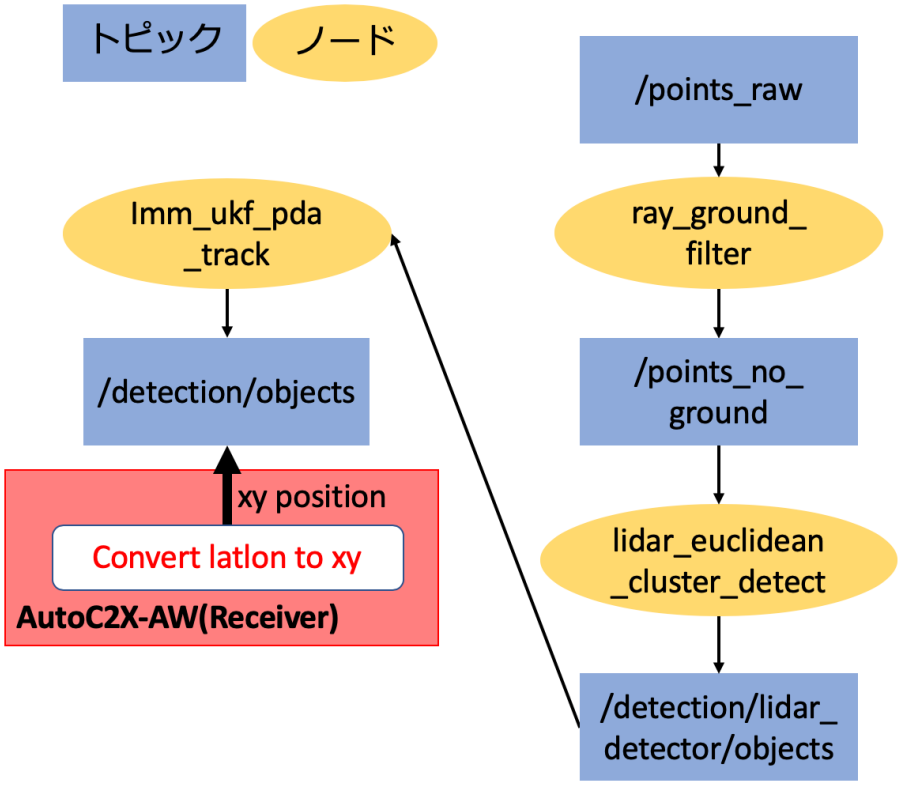


図 5.6. 本研究において車両側の物体検知に使用したノードとトピックの関係。AutoC2X-AW から得た情報を/detection/objects トピックに対して発行している。

5.4.1 路側機と車両の OpenC2X 同士の通信

OpenC2X は ETSI の標準に規定されている挙動を再現するために 8 つのモジュールに分けてソフトウェアが作られている。本研究においては自車両及び周辺の状況に関しては全て Autoware から得た情報を共有するため、gps と obd2 モジュールは使用していない。今回、Autoware の情報からは CAM のみを生成するので denm、denmApp も使用していない。OpenC2X はルータ上で動き、CUI 環境でのみの操作となるため httpServer も使用していない。また、AutoC2X の実装として新たに autoware モジュールを実装した。したがって、本研究において用いた OpenC2X のモジュールは以下の通りになる。

- cam
- dcc
- ldm
- autoware(AutoC2X-OC)

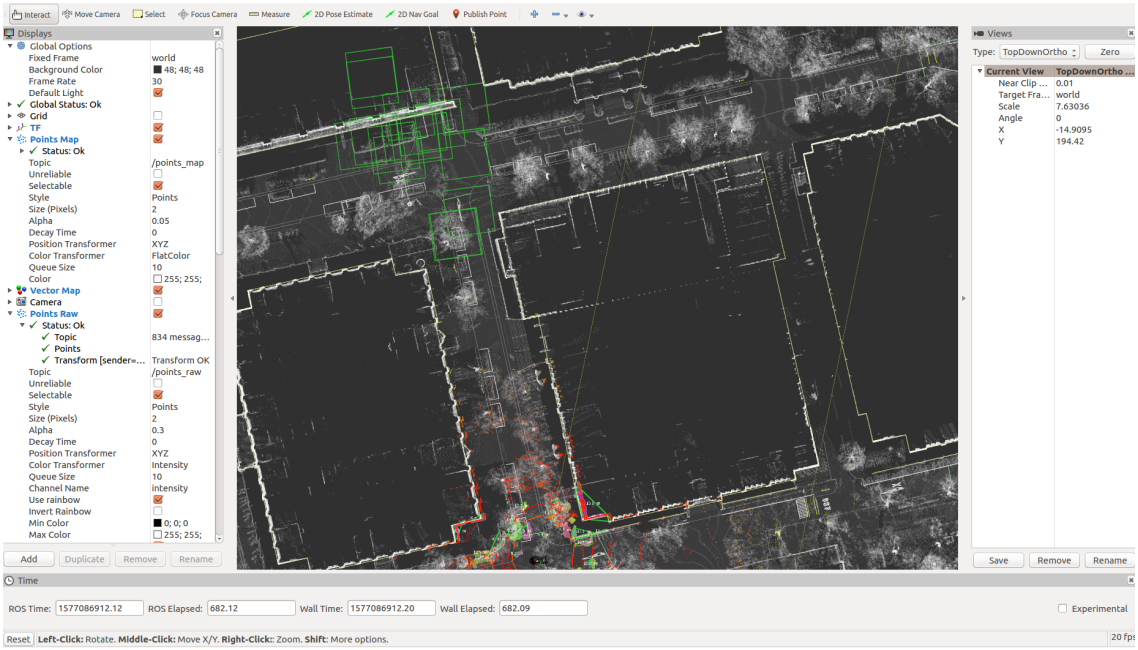


図 5.7. 車両側 (受信側) で死角にある物体を受信したメッセージを元に認識している様子

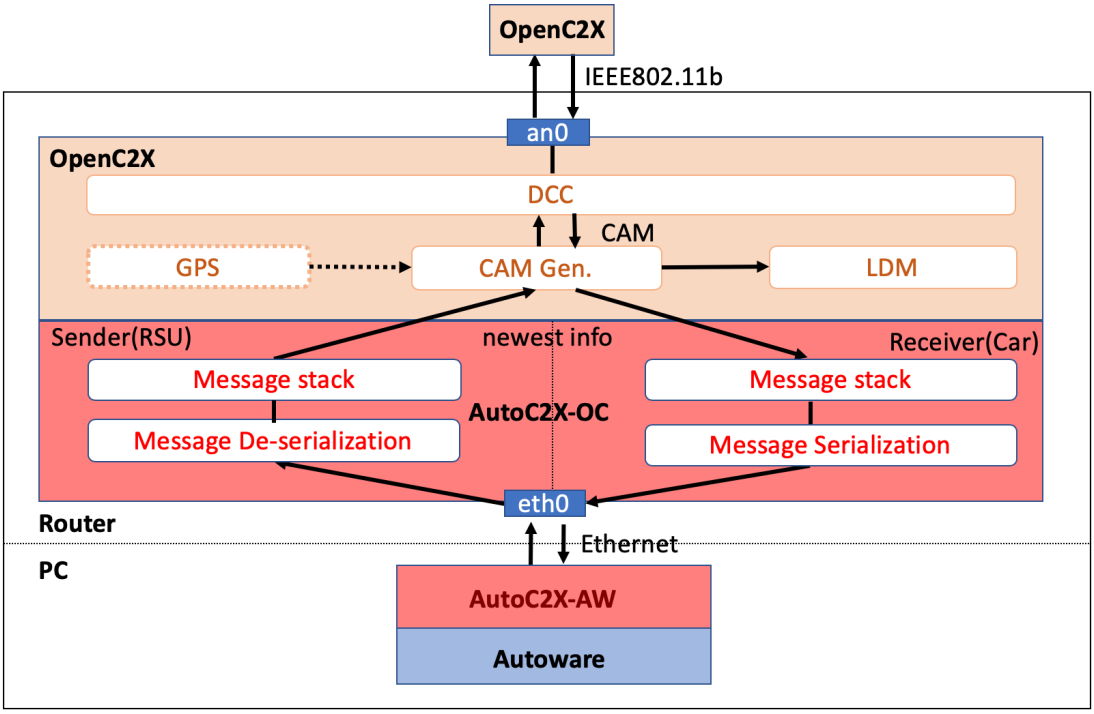


図 5.8. AutoC2X-OC の実装概要

路側機と車両のアンテナが IEEE802.11b を用いて通信し、OpenC2X が CAM を交換する。

5.4.2 AutoC2X-AW との通信

本研究では AutoC2X-AW は laptop PC で動き、AutoC2X-OC はルータで動作している。PC とルータは有線接続されており、TCP 通信によりこれらの間で情報が交換される。ここではその通信で交換されるメッセージについて述べる。

路側機側についてであるが、Autoware が/detection/objects トピックに物体情報を発行するたびに、AutoC2X-AW がその情報を座標変換をしてルータの AutoC2X-OC に送信する。それらの情報は C++ の構造体の形式で保持されており、AutoC2X-OC に送信される際に boost ライブラリのシリアライズの機能によりシリアライズされる。構造体のデータ形式は以下の通り。

- long timestamp
- std::vector<int> speed
- std::vector<int> latitude
- std::vector<int> longitude
- std::vector<int> time
- std::vector<int> stationid

timestamp はメッセージが生成された時点でのタイムスタンプを表しているため、各メッセージにつき 1 つである。それ以外の項目に関しては、認識された物体ごとの値を表すため物体の個数分の長さの vector 形式で表される。それぞれの値は ETSI が定めたデータ型で送られているため、OpenC2X では特に変換は行わずにそのままの値を用いて CAM を生成することができる。

また、本来 stationID は各 ITS station において一意であるが、今回は Autoware が物体を認識するごとに各物体にはランダムな stationID が割り振られる。すなわち、同一の物体に対しても別の stationID が割り振られるということになる。これは一つの物体に一意の stationID が与えられていた場合、遅延計測においてパケットロスが生じた際にどのパケットが落ちたのかを判別できなくなるためである。

このメッセージをデシリアライズすることで AutoC2X-OC でこの情報を利用できる。

次に車両側についてであるが、CAM を受け取った OpenC2X は Protocol Buffers の形式で AutoC2X-OC に路側機周辺の物体情報を送る。AutoC2X-OC は AutoC2X-AW にメッセージを送信するために、Protocol Buffers 形式のメッセージを C++ の構造体に変換し、boost ライブラリによりシリアライズして、Ethernet 経由で AutoC2X-AW に送信する。その構造体は以下の形式である。

- long timestamp
- float latitude

- float longitude
- std::vector<int> speed
- std::vector<int> latitude
- std::vector<int> longitude
- std::vector<int> time

このメッセージをデシリアライズすることで AutoC2X-AW がこの情報を利用できる。

5.4.3　OpenC2X との通信

AutoC2X-OC と OpenC2X とのメッセージ交換について述べる。

路側機側の AutoC2X-OC についてであるが、AutoC2X-AW から得たメッセージが到達するたびにデシリアライズした後 Protocol Buffers の形式に直してスタックに追加していく。CAM は 10Hz で生成されるが、その周期が来るたびにスタックに含まれる最新の情報のみが CAM の生成に利用される。

一方車両側の AutoC2X-OC についてであるが、OpenC2X から Protocol Buffers の形式で渡された CAM を 10Hz の周期ごとに区切り、先述の C++ の構造体の形式に直しシリアライズして AutoC2X-Aw に送信する。

5.4.4　GPS モジュールから ndt_matching への変更

4.3.3 項で述べた通り、本実装では自己位置推定を OpenC2X の GPS モジュールを Autoware の ndt_matching に置き換えることで、自己位置推定の精度をあげている。したがって、本来 GPS モジュールが cam モジュールに対して緯度経度情報を送っていた所を廃止し、autoware モジュール (AutoC2X-OC) が cam モジュールに対して緯度経度情報を送るように変更した。

第 6 章

実験・評価

本章では本研究において行ったインドアテスト及びフィールド実験について説明する。

6.1 共通の実験環境

最初にインドアテスト及びフィールド実験両方に共通する実験環境について述べる。
まず、送信機側と受信機側の Autoware が動く PC のスペックを表 6.1 に示す。

項目	送信側	受信側
CPU	Core i7-7700HQ 2.80GHz 8-core	Core i7-6700HQ 2.60GHz 8-core
RAM	16GB	32GB
HDD	512GB	256GB
OS	Ubuntu 16.04.5 LTS	Ubuntu 16.04.6 LTS

表 6.1. インドアテスト及びフィールド実験共通の実験環境

実験に使用したその他の機器を表 6.2 に示す。

役割	内容
ルータ	apu4c4[40]
Wi-Fi モジュール	Intel Wi-Fi 6 AX200 Module

表 6.2. その他、実験に使用した機器

また、実験の際に使用した路側機と車両はそれぞれ図 6.1、図 6.2 のようになっている。

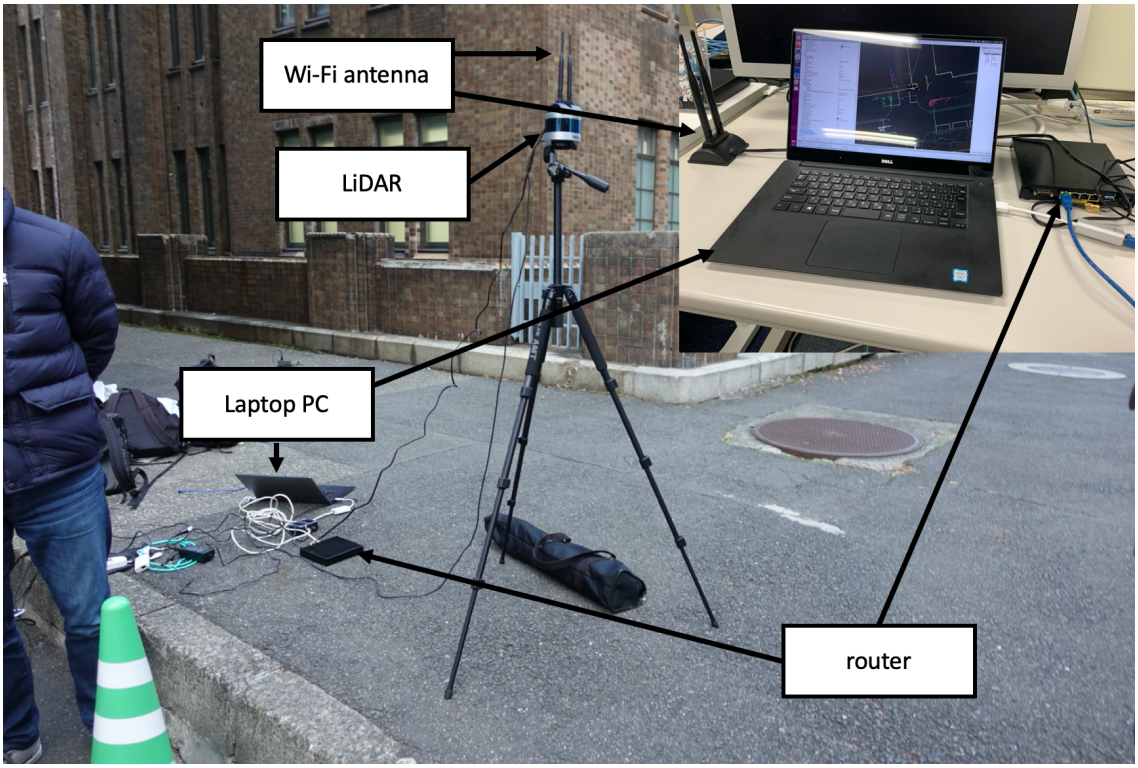


図 6.1. 実験に使用した路側機

6.2　計測手法

本システムにおいて路側機の Autoware が周辺の物体を検知し、それを車両の Autoware が受け取るまでの遅延を総遅延 T とする。総遅延は Autoware からの読み出し遅延 T_{pc}、ホストからルータまでの通信遅延 T_{hr}、ルータ間の無線通信における通信遅延 T_{rr}、ルータからホストまでの通信遅延 T_{rh} を用いて以下の式で求めることができる。

$$T = T_{pc} + T_{hr} + T_{rr} + T_{rh} \tag{6.1}$$

以下、これら 4 つの遅延の計測手法について説明する。

6.2.1　Autoware からの読み出し遅延 T_{pc}

Autoware が点群情報をクラスタリングし、物体を検出するまでは処理時間がかかる。本研究において認識された物体情報は/detection/objects トピックに発行された情報を用いるが、それらの情報にはタイムスタンプが記録されている。そのタイムスタンプは認識の際に使用した点群データが発行された時刻を示している。このタイムスタンプをルータにメッセージを送信する直前のタイムスタンプから引くことで Autoware からの自己位置の読み出し時間 T_{pc} を計測する。

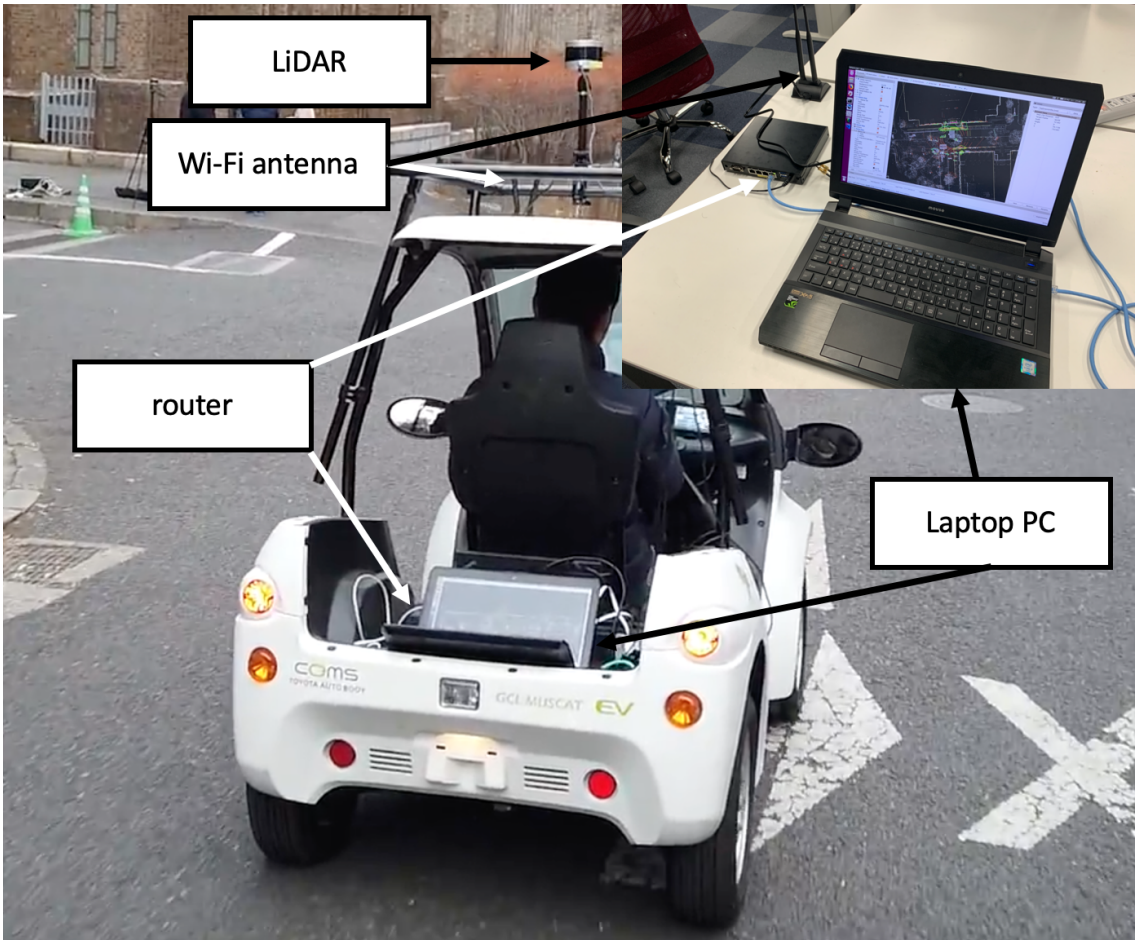


図 6.2. 実験に使用した車両

6.2.2 ホストからルータまでの通信遅延 T_{hr} とパケット到達率 R_{hr}

PC 側でメッセージを生成したシステム時間を**??**項にて述べた構造体の timestamp に記録する。これをルータに対して送信し、さらにルータ側からそのメッセージを再び PC に送り返しそれを PC 側で受け取ることで PC とルータの往復の遅延 T_{hr} が計算できる。また、パケット到達率 R_{hr} に関しては PC 側で記録されたタイムスタンプの一覧と、ルータ側で記録されたタイムスタンプの一覧を比較することで求めることができる。

6.2.3 ルータ間の無線通信における通信遅延 T_{rr} とパケット到達率 R_{rr}

OpenC2X から OpenC2X の遅延に関しては CAM に含まれている generationDeltaTime を用いる。本来 CAM の生成時刻は 2004 年 1 月 1 日 0 時 0 分 0 秒からのミリ秒単位経過時間を 65535 で割った剰余として表すこととなっている。この値を遅延計測に用いることでミリ秒単位の経過時間を計測することができる。

具体的な流れを示す。送信側の OpenC2X において CAM の生成時にミリ秒単位のシステム時間を 65536 で割り、その剰余を CAM の generationDeltaTime に記録する。その CAM が受信側の OpenC2X に達すると同時にその CAM は送信側の OpenC2X に返される。送信側で返ってきた CAM に含まれる generationDeltaTime とその時のミリ秒単位のシステム時間を 65536 で割った剰余との差を計算することで遅延 T_{rr} を計測することができる。また、オリジナルの OpenC2X の generationDeltaTime の実装に誤りが見られたため、訂正して実装した。パケット到達率 R_{rr} に関しては送信側の OpenC2X の LDM に保存されている項目と、受信側の OpenC2X の LDM に保存されている項目を比較することで求めることができる。

6.2.4　ルータからホストまでの通信遅延 T_{rh} とパケット到達率 R_{rh}

受信側では、ルータが CAM を受け取ってから PC でそれを受け取るまでの通信遅延を計測する。具体的な流れとしては OpenC2X が Autoware に対してシリアライズされたデータを送信する瞬間のシステム時間を timestamp としてメッセージに代入し、受け取った Autoware 側の PC はパケットを受け取るとともに、そのパケットをルータに送り返す。このようにすることでルータと PC 間の往復の通信遅延 T_{rh} を計測することができる。パケット到達率 R_{rh} に関してはルータ側でパケット送信時に記録したタイムスタンプの一覧と、PC 側でパケット受信時に記録したタイムスタンプの一覧を比較することで計測することができる。

6.3　インドアテスト

6.3.1　実験環境

インドアテストにおいては過去に本郷キャンパス内で走行した際に収録した rosbag データを用いた。路側機側の周辺情報のデータは 2019 年 12 月 12 日に収録されたものであり、車両側の走行データは 2017 年 4 月 19 日に収録されたものである。車両側の走行経路は図 6.3 の黒い矢印のとおりになっており 4 分 35 秒で一周する。図中央上部に示された工学部 8 号館付近の赤い星印の部分には路側機を設置した。

実際に使用した機器は図 6.1、図 6.2 の右上のように実際の機器から PC、ルータ、アンテナを抜き出し、机上に設置したものになっており、これらの機器を研究室内で約 2.7m 離して実験を 5 回計測を行った。

6.3.2　各部の計測結果

Autoware からの読み出し遅延 T_{pc}、ホストからルータまでの通信遅延 T_{hr}、ルータ間の無線通信における通信遅延 T_{rr}、ルータからホストまでの通信遅延 T_{rh} をそれぞれ分けて遅延を計測した結果が図 6.4 である。また、ホストからルータまでのパケット到達率 R_{hr}、ルータ間の無線通信におけるパケット到達率 R_{rr}、ルータからホストまでのパケット到達率 R_{rh} を表 6.3 に示す。

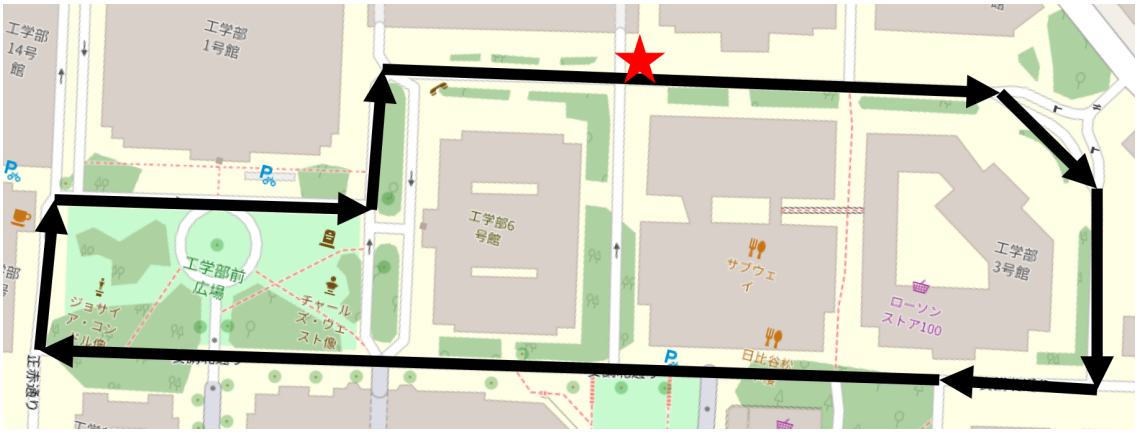


図 6.3. インドアテストにて使用した車両走行データの走行経路 (黒い矢印) 及び路側機の設置場所 (赤い星印)

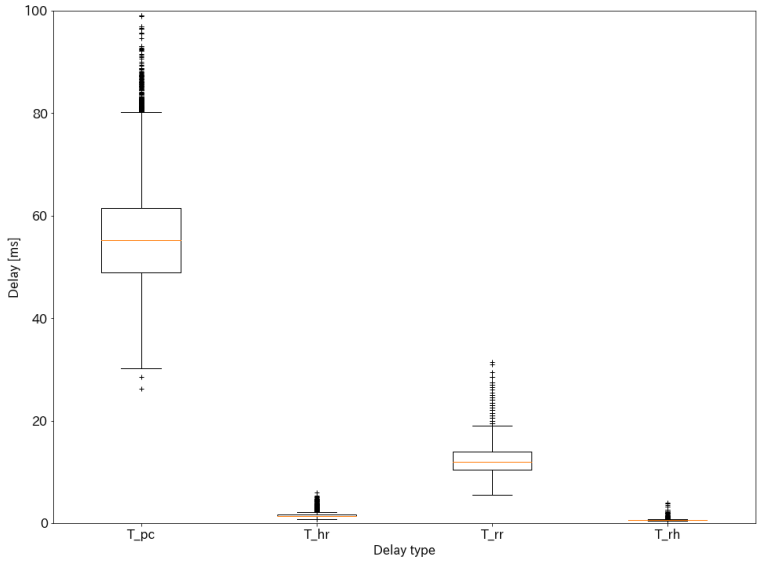


図 6.4. 各部の遅延計測結果の比較

T_{pc} に関しては概ね 80ms 以内の読み出し遅延に収まっている。100ms 以上読み出しに時間がかかっている点も存在するが、これは CPU 使用状況に依存して物体検出などの処理に時間がかかっていることが考えられる。

T_{hr} に関しては概ね 3ms 以内、T_{rr} は概ね 20ms 以内、T_{rh} は概ね 3ms 以内に収まる結果となった。

項目	パケット到達率
R_{hr}	100%
R_{rr}	97.2%
R_{rh}	100%

表 6.3. 各部のパケット到達率の比較

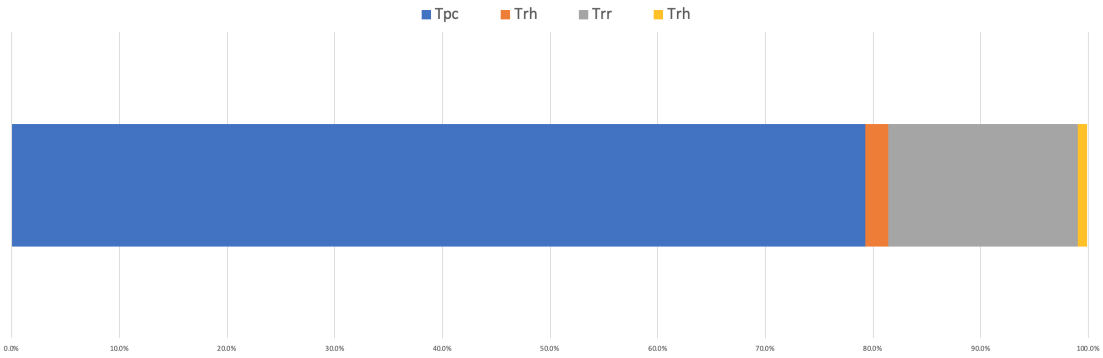


図 6.5. 総遅延 T に占める各遅延の割合

6.3.3　総遅延 T

以上の結果より総遅延 T を求めると平均して 70.7ms で路側機の Autoware から車両の Autoware に情報を送ることができる。また、総遅延に占める各遅延の割合は図 6.5 のようになる。路側機側の PC における Autoware の読み出し遅延の割合が総遅延 80% 程度を占めていることが明らかになった。

6.4　フィールド実験

6.4.1　実験環境

フィールド実験においては本郷キャンパス内で路側機を設置し、車両の位置を変えながら計測を行った。実験の際には図 6.6 において黒い線の部分を時速 3km 程度で複数回走行した。RSU の位置は赤い矢印に示す通り、インドアテストで用いた場所と同じである。また、実験に使用した路側機と車両はそれぞれ図 6.1、図 6.2 である。また、実験は 1000 秒程度であり、概ね 50 60 個 (平均 57 個/秒) のメッセージが送られていた。

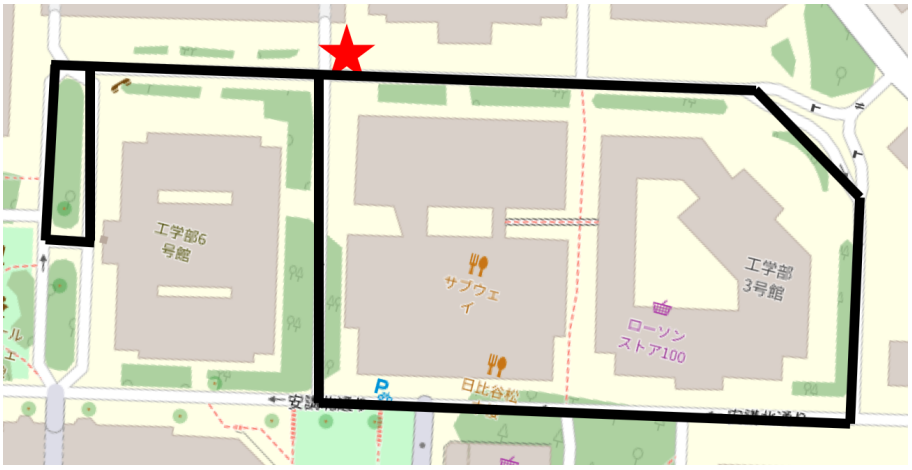


図 6.6. フィールド実験にて走行した経路。

6.4.2 ルータ間の無線通信における遅延 T_{rr} とパケット到達率 R_{rr}

フィールド実験において得られたルータ間の通信に関する結果を図 6.7、図 6.8 に示す。これらはそれぞれ遅延と場所の関係を表している図、パケット到達率と場所の関係を表している図である。黄色い星印の部分に RSU は設置されている。

図 6.7 において遅延を表すタイルが存在していない部分は路側機が発した CAM が 1 つも車両を経由して返ってこなかったことを表す。図 6.8 においてパケット到達率を表すタイルが存在していない部分は路側機が発した CAM が 1 つも車両に届いていないことを示す。パケット到達率は受信側にパケットが届くことによって計測することができるが、遅延は送信したパケットが受信側から返されないと計測ができないため、遅延の計測結果よりもパケット到達率の方がより広い範囲で計測結果が出ている。

図において RSU から見て左の部分は遠ざかるほど遅延が段階的に大きくなっているが、右の部分は一部不規則に遅延が大きくなっている部分がある。これは RSU の右側部分に茂みがあり、見通しが悪く不規則な結果になっていると考えられる。

パケット到達率に関しても同様の傾向が見られる。RSU に対して見通しの良い左側 30m 程度まではパケット到達率が 80% 以上となっている。また、図において RSU から見て下方の道は凹みとなっており、A 地点よりも B 地点の方が RSU からの距離は大きいがパケット到達率はそれ以上となっている。

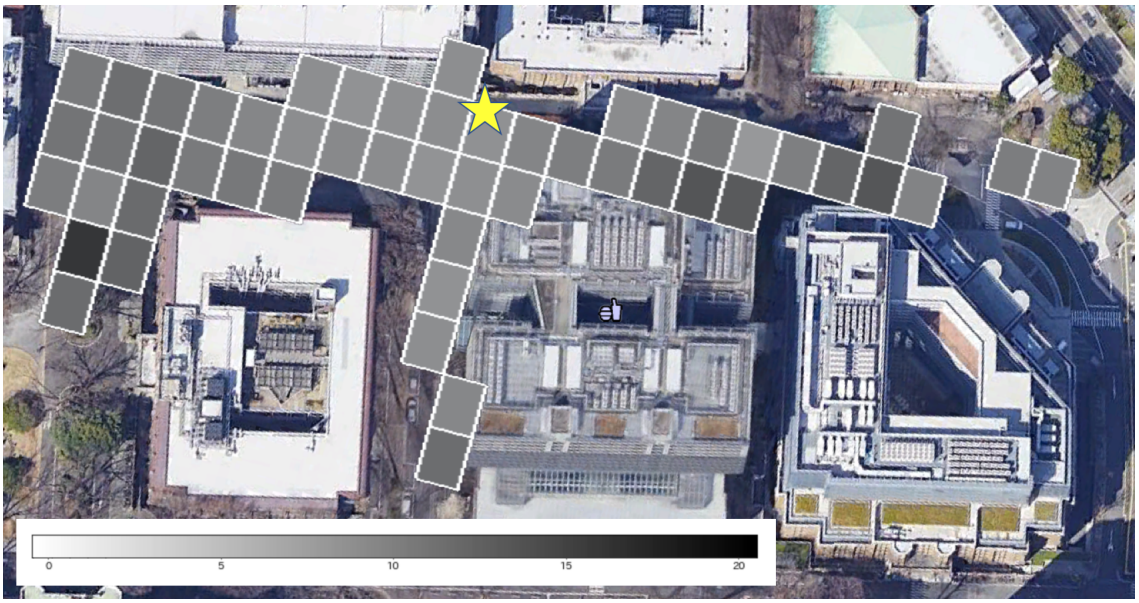


図 6.7. ルータ間の無線通信における通信遅延 T_{rr}

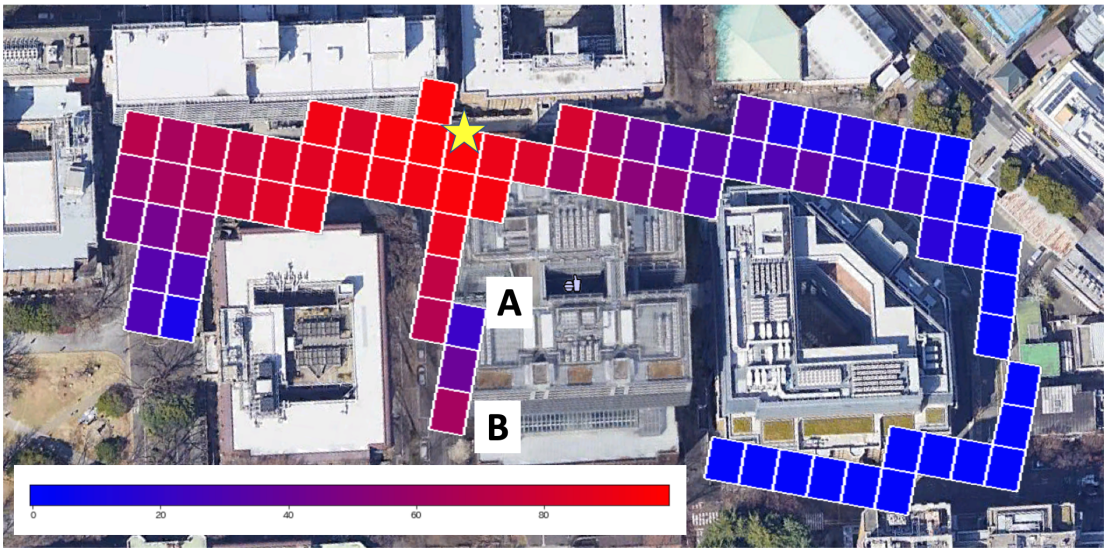


図 6.8. ルータ間の無線通信におけるパケット到達率 R_{rr}

第 7 章

結論

7.1 まとめ

本研究では現状の自律型自動運転ソフトウェアの持つ問題点を挙げ、要求事項として定義した。それらは通信の安全性、拡張性、標準への準拠、そして認識領域の拡張である。この要求事項を達成するために協調型自動運転ソフトウェアを自立型自動運転ソフトウェアに統合することを提案した。

本研究では自律型自動運転ソフトウェアとして Autoware を用い、協調型自動運転ソフトウェアとして OpenC2X を用いた。これらはすでにいくつかの研究で使用されているものである。また、自律型自動運転ソフトウェアの情報を協調型自動運転ソフトウェアを通じて共有する場合、適切な座標変換が必要であり、今回用いた座標変換の手法を紹介した。

実験においては、路側機の Autoware が周辺の物体を認識し、その情報を離れた場所にいる車両に対して OpenC2X を介して送信し、車両側の Autoware で送られてきた物体を認識することを確認すると共に、遅延及びパケット到達率を測定した。実験は研究室におけるインドアテスト、本郷キャンパス内におけるアウトドアテストの 2 種類行った。

実験結果として、路側機側で認識された情報を標準化された CAM の形式で車両側に送信し、それを車両側で認識できたことを確認した。また、Autoware の物体認識から CAM 生成までの時間が 100ms 以内となり、CAM の生成周期よりも短い間隔で Autoware が得た情報を更新することができることが明らかになった。

フィールド実験の結果として遅延、パケット到達率ともに路側機と車両が同じ距離であっても建物や木などが間にあるなどの見通しによって計測結果が悪化することが明らかになった。

7.2 今後の課題

2.6 節で述べた通り、OpenC2X は ETSI-G5 でなされているセキュリティ層と GeoNetworking のルーティング部分の実装がされていない。2.5.2 節の通り、LDM は認証されたアプリケーションのみデータベースにアクセスすることができる。認証自体は頻繁に行われるわけではないと考えられるが、アクセス制御によるオーバーヘッドが生じると考えられる。

今回の実験では路側機 1 台から車両 1 台に対してシングルホップでメッセージを送信するシナリオで実験を行った。したがって、GeoNetworking によるルーティングの必要はなかったが、車両台数を増やしてマルチホップが必要になった場合、追加実装の必要がある。

車両側で受け取った物体情報は定量評価は行なっていないが、位置の誤差が生じていた。これは車両が横メルカトル座標に対して絶えず向きを変えており、座標変換の際に車両の回転に対して敏感に反応して誤差が生じたと考えられる。特に車両が方向転換をしている時は数十 ms 程度の遅延であっても位置の誤差が大きくなると考えられる。

発表文献と研究活動

(1) 大井貴晴, 落合秀也, 江崎浩．オープン DNS リゾルバの現状把握手法の提案と評価．マルチメディア，分散協調とモバイルシンポジウム 2018, 2018.07.05.

(2) 大井貴晴, 北沢 昌大, 塚田 学, 落合 秀也, 江崎 浩．ネットワーク化された路側拠点による車両認識メッセージの広域送信．マルチメディア，分散協調とモバイルシンポジウム 2019, 2019.07.04.

参考文献

[1] Shinpei Kato, Shota Tokunaga, Yuya Maruyama, Seiya Maeda, Manato Hirabayashi, Yuki Kitsukawa, Abraham Monrroy, Tomohito Ando, Yusuke Fujii, and Takuya Azumi. Autoware on board: Enabling autonomous vehicles with embedded systems. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296. IEEE, 2018.

[2] Apolloauto/apollo: An open autonomous driving platform. `https://github.com/ApolloAuto/apollo`. (Accessed on 11/15/2019).

[3] Nvidia drive - software — nvidia developer. `https://developer.nvidia.com/drive/drive-software`. (Accessed on 01/04/2020).

[4] Etsi - welcome to the world of standards! `https://www.etsi.org/`. (Accessed on 11/15/2019).

[5] Ieee - the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity. `https://www.ieee.org/`. (Accessed on 11/15/2019).

[6] 一般社団法人　電波産業会. `https://www.arib.or.jp/`. (Accessed on 11/15/2019).

[7] M Kurdej, J Moras, V Cherfaoui, and P Bonnifait. Map-Aided Evidential Grids for Driving Scene Understanding. *IEEE Intell. Transp. Syst. Mag.*, Vol. 7, No. 1, pp. 30–41, 2015.

[8] Autopilot — tesla. `https://www.tesla.com/autopilot`. (Accessed on 01/04/2020).

[9] Autoware foundation / autoware.ai / autoware · gitlab. `https://gitlab.com/autowarefoundation/autoware.ai/autoware`. (Accessed on 12/23/2019).

[10] Ros wiki. `http://wiki.ros.org/`. (Accessed on 12/20/2019).

[11] ETSI. En 302 637-2 - v1.3.2 - intelligent transport systems (its); vehicular communications; basic set of applications; part 2: Specification of cooperative awareness basic service. `https://www.etsi.org/deliver/etsi_en/302600_302699/30263702/01.03.02_60/en_30263702v010302p.pdf`, 2014. (Accessed on 11/29/2019).

[12] ETSI. En 302 895 - v1.1.1 - intelligent transport systems (its); vehicular communications; basic set of applications; local dynamic map (ldm). `https://www.etsi.org/deliver/etsi_en/302800_302899/302895/01.01.`

01_60/en_302895v010101p.pdf, 2014. (Accessed on 1/4/2019).

[13] En 302 665 - v1.1.1 - intelligent transport systems (its); communications architecture. https://www.etsi.org/deliver/etsi_en/302600_302699/302665/01.01.01_60/en_302665v010101p.pdf. (Accessed on 11/29/2019).

[14] Ts 102 894-2 - v1.2.1 - intelligent transport systems (its); users and applications requirements; part 2: Applications and facilities layer common data dictionary. https://www.etsi.org/deliver/etsi_ts/102800_102899/10289402/01.02.01_60/ts_10289402v010201p.pdf. (Accessed on 11/29/2019).

[15] ETSI. En 302 637-3 - v1.2.2 - intelligent transport systems (its); vehicular communications; basic set of applications; part 3: Specifications of decentralized environmental notification basic service. https://www.etsi.org/deliver/etsi_en/302600_302699/30263703/01.02.02_60/en_30263703v010202p.pdf, 2014. (Accessed on 1/4/2020).

[16] Openc2x - distributed embedded systems. https://www.ccs-labs.org/software/openc2x/. (Accessed on 11/08/2019).

[17] Sven Laux, Gurjashan Singh Pannu, Stefan Schneider, Jan Tiemann, Florian Klingler, Christoph Sommer, and Falko Dressler. Openc2x—an open source experimental and prototyping platform supporting etsi its-g5. In *2016 IEEE Vehicular Networking Conference (VNC)*, pp. 1–2. IEEE, 2016.

[18] Swarun Kumar, Lixin Shi, Nabeel Ahmed, Stephanie Gil, Dina Katabi, and Daniela Rus. Carspeak: a content-centric network for autonomous driving. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pp. 259–270. ACM, 2012.

[19] Seong-Woo Kim, Zhuang Jie Chong, Baoxing Qin, Xiaotong Shen, Zhuoqi Cheng, Wei Liu, and Marcelo H Ang. Cooperative perception for autonomous vehicle control on the road: Motivation and experimental results. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5059–5066. IEEE, 2013.

[20] Tomoya Kitazato, Manabu Tsukada, Hideya Ochiai, and Hiroshi Esaki. Proxy Cooperative Awareness Message: An Infrastructure-Assisted V2V Messaging. In *The Ninth International Conference on Mobile Computing and Ubiquitous Networking*, Kaiserslautern,, Germany, October 2016.

[21] Manabu Tsukada, Masahiro Kitazawa, Takaharu Oi, Hideya Ochiai, and Hiroshi Esaki. Cooperative awareness using roadside unit networks in mixed traffic. In *IEEE Vehicular Networking Conference (VNC) 2019*, pp. 9–16, Los Angeles, California, 2019.

[22] Zebang Shen, Yichong Xu, Muchen Sun, Alexander Carballo, and Qingguo Zhou. 3d map optimization with fully convolutional neural network and dynamic local ndt. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 4404–4411.

IEEE, 2019.

[23] Yu Sasaki, Tomoya Sato, Hiroyuki Chishiro, Tasuku Ishigooka, Satoshi Otsuka, Kentaro Yoshimura, and Shinpei Kato. An edge-cloud computing model for autonomous vehicles.

[24] Vimal Rau Aparow, Apratim Choudary, Giridharan Kulandaivelu, Thomas Webster, Justin Dauwels, and Niels de Boer. A comprehensive simulation platform for testing autonomous vehicles in 3d virtual environment. In *2019 IEEE 5th International Conference on Mechatronics System and Robots (ICMSR)*, pp. 115–119. IEEE, 2019.

[25] Shota Tokunaga, Yuki Horita, Yasuhiro Oda, and Takuya Azumi. Idf-autoware: Integrated development framework for ros-based self-driving systems using matlab/simulink. In *Workshop on Autonomous Systems Design (ASD 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[26] riebl/artery: Omnet++ v2x simulation framework for etsi its-g5. `https://github.com/riebl/artery`. (Accessed on 01/04/2020).

[27] Veins. `http://veins.car2x.org/`. (Accessed on 01/04/2020).

[28] Simulation of vehicle-2-x communication - dcaiti - tu berlin. `https://www.dcaiti.tu-berlin.de/research/simulation/`. (Accessed on 01/04/2020).

[29] Adrian Abunei, Ciprian R Comşa, Constantin F Caruntu, and Ion Bogdan. Redundancy based v2v communication platform for vehicle platooning. In *2019 International Symposium on Signals, Circuits and Systems (ISSCS)*, pp. 1–4. IEEE, 2019.

[30] Francesco Raviglione, Marco Malinverno, and Claudio Casetti. Open source platform for ieee 802.11 p nics evaluation. In *2019 IEEE 20th International Symposium on" A World of Wireless, Mobile and Multimedia Networks"(WoWMoM)*, pp. 1–3. IEEE, 2019.

[31] Ion Turcanu, Florian Adamsky, and Thomas Engel. Experimental evaluation of floating car data collection protocols in vehicular networks. In *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pp. 1–6. IEEE, 2019.

[32] En 302 636-5-1 - v1.2.0 - intelligent transport systems (its); vehicular communications; geonetworking; part 5: Transport protocols; sub-part 1: Basic transport protocol. `https://www.etsi.org/deliver/etsi_en/302600_302699/3026360501/01.02.00_20/en_3026360501v010200a.pdf`, 2013. (Accessed on 01/16/2020).

[33] En 302 636-1 - v1.2.1 - intelligent transport systems (its); vehicular communications; geonetworking; part 1: Requirements. `https://www.etsi.org/deliver/etsi_en/302600_302699/30263601/01.02.01_60/en_30263601v010201p.pdf`, 2014. (Accessed on 01/06/2019).

[34] John Parr Snyder. *Map projections–A working manual*, Vol. 1395. US Government Printing Office, 1987.

[35] Boost c++ libraries. `https://www.boost.org/`. (Accessed on 12/24/2019).

[36] Proj — proj 6.2.1 documentation. https://proj.org/. (Accessed on 12/24/2019).

[37] florianklingler/openc2x-standalone: Open source experimental and prototyping platform supporting etsi its-g5. https://github.com/florianklingler/OpenC2X-standalone. (Accessed on 12/24/2019).

[38] florianklingler/openc2x-embedded: This repository is a mirror of https://git.openwrt.org/openwrt/openwrt.git it is for reference only and is not active for check-ins or for reporting issues. we will continue to accept pull requests here. they will be merged via staging trees then into openwrt.git. all issues should be reported at: https://bugs.openwrt.org. https://github.com/florianklingler/OpenC2X-embedded. (Accessed on 12/24/2019).

[39] esakilab/openc2x-standalone: Open source experimental and prototyping platform supporting etsi its-g5. https://github.com/esakilab/OpenC2X-standalone. (Accessed on 12/24/2019).

[40] Pc engines apu4c4 product file. https://www.pcengines.ch/apu4c4.htm. (Accessed on 12/24/2019).

謝辞

本論文の執筆にあたり多くの方々にご指導とご協力をいただきました。ここに心より感謝申し上げます。

指導教官として研究の方向性・発表に関するアドバイスなど大域的な観点から指導をしてくださり、温かい目で見守ってくださった江崎浩教授に感謝を申し上げます。

協調型 ITS について基礎からご教授いただき、本研究の細部まで指導をしてくださった塚田学准教授に感謝を申し上げます。

学部生の頃からお世話になり、研究の組み立て方、進め方、研究生活の指導をしてくださった落合秀也准教授に感謝を申し上げます。研究者として、先輩として、学部生の頃から励まし、研究者としての姿を間近で見せてくださった小林諭博士に感謝いたします。

協調型 ITS のグループにおいて先輩として指導・協力してくださった北沢昌大氏、グループの同期として励まし合った長嶋秀幸氏、実験協力・設備準備などの協力をしてくださった平田真唯氏、水谷将也氏、増田英孝氏、辻尾康平氏、伊藤彰秀氏に感謝いたします。本研究で使用した Autoware への深い理解をする手助けとなってくださった株式会社ティアフォーの飯田祐希氏に感謝いたします。

研究室の同期として 3 年の月日を共に過ごし、切磋琢磨し、励まし合った山城裕陽氏、松藤央氏、新津雄大氏に感謝いたします。研究生活の様々な面でバックアップをしてくださった岩井愛映子秘書に感謝いたします。数々の場面で共に過ごした研究室の皆様、ご卒業された方々に感謝いたします。

そして私を育ててきてくれ、支えてきてくださった両親に感謝いたします。本当にありがとうございます。