東京大学
情報理工学系研究科 電子情報学専攻
修士論文

協調型自動運転のための機能抽象化した走行調停サービス

Maneuver coordination service with abstracted functions for cooperative autonomous driving

48-196443
水谷 将也
Masaya Mizutani

指導教員　江崎 浩 教授

2021 年 1 月

概要

近年、自律型自動運転の研究開発が盛んに行われている。自律型自動運転には死角の存在、全体最適な経路計画の限界など課題が存在する。これを解決するために、車車間通信を利用する協調型自動運転の研究も盛んである。協調型自動運転用のネットワークアーキテクチャやメッセージは各団体で標準化されているが、走行調停のためのメッセージは未だ議論の段階で、標準化されていない。本研究では、安全性・汎用性・コーナケースを考慮した走行調停用のプロトコルとサービスの設計を提案した。プロトコルは、コーナケースや安全性を考慮し、メッセージを 7 種類に分け、各ステーションを状態で管理する設計とした。また、サービスは、汎用性を持たせるために、各シナリオが共通して使用する機能を抽象化する設計とした。

実験により、提案した走行調停用プロトコルが有用であることを示すために、自律型自動運転用ソフトウェアである Autoware と協調運転用ソフトウェアの OpenC2X を拡張し、走行調停サービスを実装した。シミュレータを用いて、4 台での走行調停が実現できることを示し、通信量、各プロセスの実行時間、MCM による交通の快適性への効果を明らかにした。

Abstract

In recent years, there has been a lot of research and development on autonomous self driving. Autonomous self driving has some issues such as the existence of blind spots and the limitation of globally optimal route planning. In order to solve these problems, research on cooperative autonomous driving using inter-vehicle communication is also active. Network architectures and messages for cooperative automatic driving have been standardized by various organizations, but messages for maneuver coordination are still in the discussion stage and have not been standardized. In this study, we proposed a protocol and service design for maneuver coordination considering safety, generality, and corner cases. The protocol is divided into seven types of messages and designed to manage each station by its state in consideration of corner cases and safety. The service is designed to abstract the functions commonly used in each scenario in order to make it versatile.

In order to demonstrate the effectiveness of the proposed protocol for maneuver coordination through experiments, the maneuver coordination service was implemented by extending Autoware, the software for autonomous automatic driving, and OpenC2X, the software for cooperative driving. Using a simulator, we showed that maneuver coordination can be achieved with four vehicles, and clarified the effect of communication volume, execution time of each process, and MCM on the comfort of traffic.

目次

第 1 章　序論 1
1.1 はじめに 1
1.2 本研究の目的 2
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 Autoware 3
2.3 自律型自動運転の問題点と課題 5
2.4 協調型自動運転 5
2.5 MCM とその必要性 8
2.6 OpenC2X 9

第 3 章　関連研究 12
3.1 特定のシナリオに対する走行調停の研究 12
3.2 より汎用的な走行調停プロトコルの研究 14

第 4 章　MCM と MC Service の提案 16
4.1 要求事項 16
4.2 MCM 19
4.3 MC Service 24
4.4 MCM を使用するシナリオ 25

第 5 章　AutoMCM の実装 27
5.1 実装で用いたソフトウェア 27
5.2 実装の概要 27
5.3 各メッセージごとの実装 28

第 6 章　実験・評価 37
6.1 実験環境 37
6.2 実験概要 37

第 7 章 結論 46
7.1 まとめ 46
7.2 今後の課題 46

発表文献と研究活動 48

参考文献 49

第 1 章

序論

1.1 はじめに

近年、自動車の自動運転に関わる研究開発が数多く行われている。

その背景としては、乗客の快適性を高めるだけでなく、交通安全性を高めることが挙げられる。2003 年以降は DARPA チャレンジ [1] やつくばチャレンジ [2] などが開催され、自動運転車のさまざまな分野で研究が活発化している。しかし、テスラ社の半自動運転車両の公道における死亡事故や、Uber 社のテスト走行における死亡事故なども実際に発生している。これらの事故からも、自動運転分野は未成熟であり、課題が多く残されていることがわかる。

自動運転の中心となるプロセスは、周辺環境を感知し、車両の進路を計画し、その進路に沿って行動するという 3 つの要素に基づいている。以上 3 つの全てのプロセスで、様々な研究開発が行われている。これら全てのプロセスを自車に搭載されているセンサー郡、コンピュータのみで行う技術は自律型自動運転と呼ばれている。自律型自動運転には様々な問題点がある。例えば、自車で検知できる範囲が距離的に制限されたり、死角が存在したりする問題であったり、自動運転車に搭載できる計算資源やストレージなどに限界があるという問題がある。これらの問題点を解決するために、自車両が他の車両や路側機と通信する技術を、協調型自動運転と呼ぶ。現在は、完全に統合された技術は存在しないが、将来的に統合され、自車両で得られて情報と他車両から送られてきた情報を元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[3] や、Baidu 社による apollo[4]、NVIDIA 社による NVIDIA Drive[5] などがある。

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[6] が、アメリカにおいては IEEE[7] が、日本においては ARIB[8] が主導している。協調型自動運転では、すでに車両間、路車間での知覚情報の交換などに関するメッセージの標準化が進んでおり、メッセージフォーマットまで策定されているものもある。知覚情報の交換に関する研究は、シミュレーションによる検証から、実機でのフィールド実験まで、数多く行われている。

一方で、シナリオによっては、知覚の共有だけでは不十分であることもあり、車両の進路計

画の協調も必要である。欧州では進路計画の標準となるメッセージの検討が行われており、そのメッセージに関する研究も近年増えてきている。

1.2　本研究の目的

前節で述べたとおり、自律型自動運転の問題点があり、それを克服するために現在研究されている協調型自動運転にも不十分な点がある。具体的には、例えば、車両同士が合流をしなければならない場合や、車線変更をしなければならない場合である。以上の状況では、車車間で認識した現在の状況を伝達するだけでは不十分であり、将来の車両の軌道とそれを考慮した経路計画の交換が必要となる。

そこで、本研究では、調停用のメッセージである MCM のフォーマットを提案し、また、Maneuver Coordination Message(MCM) が自動運転に与える影響を評価することを目的とする。そのために、提案した MCM を協調型自動運転用ソフトウェアである OpenC2X によって実装する。また、車車間での協調をおこなうための、複数台での進路計画をおこなうシステムを提案し、自律型自動運転ソフトウェアである Autoware と OpenC2X を統合することによって実現し、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

前章で述べたとおり、自動運転は大きく自律型自動運転と協調型自動運転に分類される。本章では、まず、自律型自動運転について説明するとともに、その問題点、不十分な点について述べる。次に、その問題点や不十分な点を解決するために検討されている協調型自動運転について説明し、本研究で扱う MCM の立ち位置と詳細について述べる。

2.1 自律型自動運転

自律型自動運転とは、自車両に搭載されているカメラや LiDAR などのセンサ情報をもとに、自己位置推定、周辺状況の認識、経路計画、車両の制御の一連のプロセスをおこない、自動運転を行う技術である。

具体的には、カメラの情報は、周辺の車両や障害物認識に使用され、LIDAR の情報は、得られた点群データと予め用意しておいた PCD マップを比較することによって自己位置を推定することや、点群データのクラスタリングによる障害物認識に用いられる。

したがって、画像認識技術や、カメラの解像度、点群データのクラスタリング精度の向上によって、自動運転の安全性が高められる。

自動運転用のソフトウェアは数多くあり、前章で述べたオープンソースのソフトウェアだけでなく、日産プロパイロット 2.0 やテスラ車のオートパイロットなど、自動車業界の各社が独自の自動運転のソフトウェアの開発を行っている。

日産プロパイロット 2.0 やテスラ社のオートパイロットでは LIDAR は用いておらず、代わりに、ミリ波レーダーやソナー、カメラのみで周辺状況や自車両の位置を認識している。

2.2 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 が開発している Autoware[3] を用いる。したがって本節では Autoware について説明する。

Autoware における自動運転のプロセスの概要を図 2.1 に示す。まず、LIDAR やカメラな

どのセンサーで周辺環境をセンシングする （Sensing)。更に、得られた情報から自己位置を推定すると同時に、画像認識などで物体を認識、予測する（Localization、Perception)。そして、自車両周辺の地物情報を 3 次元マップから取得し、検知した障害物に衝突しないように、どの経路を通るかや一時停止をするかなどの経路決定を行う（Planning)。最後に、車両の各部位をどのように動かすかを決定し、指示を出す（Control, Actuation)。

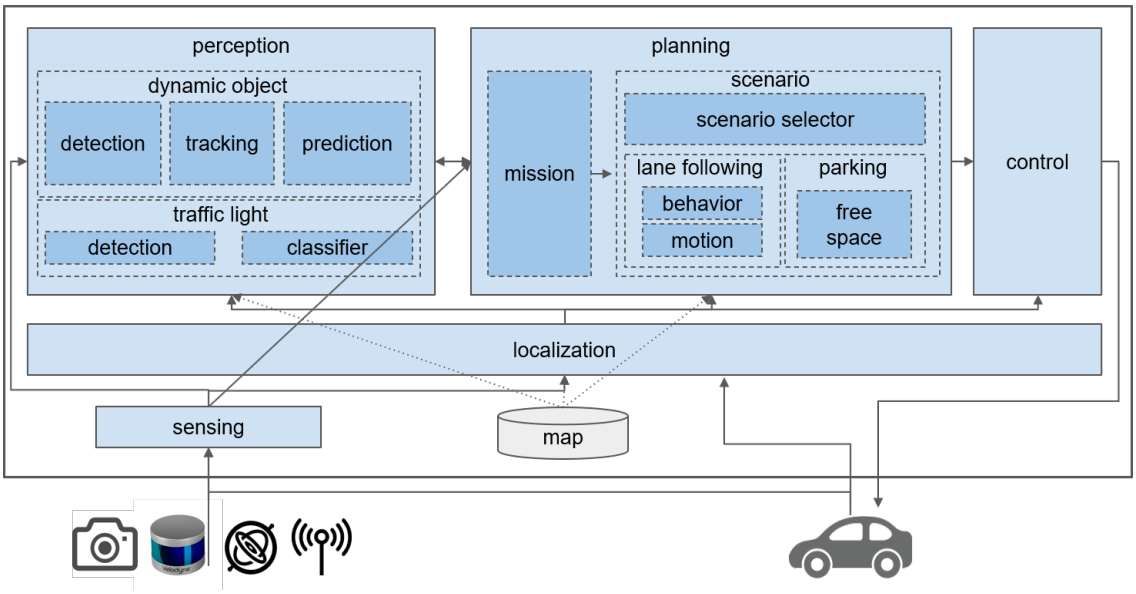


図 2.1: Autoware のシステム概要 (出典:[9])

2.2.1　ROS(Robot Operating System)

これらの機能を実現するために、Autoware は ROS[10] というミドルウェア上で実装されている。ROS はロボットの制御を行うために最適化されたミドルウェアであり、分散処理を行うために、各プロセスをノードで実装し、各ノードはトピックで情報を交換する。Autoware では、自己位置推定をするノード、物体検知をするノード、経路決定をするノードなど各プロセスごとにノードがある。自己位置推定で得られたデータはトピックに publish され、物体検知をするノードや経路決定をするノードはそのトピックを subscribe する。

ここでは、ROS における pub/sub の仕組みを図 2.2 に示す例を用いて説明する。図の楕円形、四角形はそれぞれ ROS ノード、トピックを表す。この例では、Publisher と Subscriber 1 から Subscriber n のノードがある。Publisher はトピック名/example というトピックを publish し、Subscriber 1 から Subscriber n は/example を subscribe している。したがって、Publisher が/example を publish すると n 個の Subscriber にデータが伝達される。このように、一つのノードが生成したデータを複数のノードに伝達することができる。また、/example はメッセージタイプ std_msgs/Bool というメッセージタイプで定義されている。これは、/example のデータの中身が {data: boolean} という形式であることを意味する。

ここで指定できるメッセージタイプは標準で要されているメッセージタイプだけでなく、自分で新たに定義したメッセージタイプも使用することができる。以上の仕組みによって、ROS はロボットの制御を分散的に行うことができる。

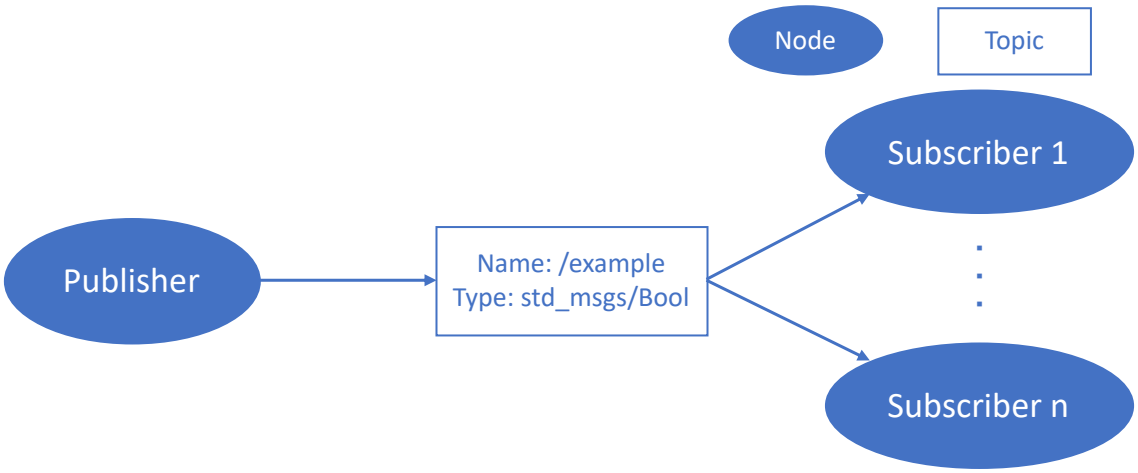


図 2.2: ROS ノードとトピックの関係の例

2.3 自律型自動運転の問題点と課題

前節で述べたとおり、自律型自動運転は自車両に搭載されたセンサやカメラの情報のみを使用して、自動運転を行う。それらのセンサ群には検知できる距離に限界がある。一般的な LIDAR は直線距離では約 100m 先まで検知範囲であるが、得られたセンサ情報から物体を認識できる範囲は更に短い距離までとなる。　また、障害物による死角も存在する。公道では、交通弱者 (Vulnerable Road User (VRU)) が死角から飛び出してくる可能性があり、自車両のセンサ群の情報のみでは、急な飛び出しによる重大な事故を防ぐことはできない。

一方、事故を防ぐために常に停止できる速度で走行し、交差点や死角の存在するところでは毎回停止をするなどの処置をすることで、これらの事故を防ぐことができるかもしれない。しかし、安全性を考慮するあまり、目的地に到着する時間が遅くなったり、交通渋滞を発生させたりしてしまうかもしれない。したがって、自律型自動運転では安全性と快適性のトレードオフになってしまう。

2.4 協調型自動運転

協調型自動運転は車車間、路車間通信 (V2X) を通じて、周辺状況を共有し、自動運転をより安全に、快適にする技術である。協調型自動運転を実現するためのシステムは、欧州の ETSI[6]、アメリカの IEEE[7]、日本の ARIB[8] で各々標準化に向けた議論がなされている。例えば、欧州の ETSI では、Cooporative Intelligent Transport System (C-ITS) として標準化されている。

2.4.1　C-ITS

C-ITS では、通信規格が標準化されており、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) と呼ばれている。ITS-S の通信部分のアーキテクチャは TCP/IP などとは別に、独自に標準化されている。ITS-S の通信部分のアーキテクチャーを図 2.3 に示す。

ITS-S の通信部分の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。

アクセス層は通信の物理的な機能を扱う層であり、CITS においては先述の IEEE802.11p がこれに該当するが、現在では携帯網の 5G なども検討されている。

ネットワーク&トランスポート層は主にルーティングを扱う層である。車車間通信はネットワークトポロジが頻繁に変化するため、主にアドホックで行われるとされている。さらに、情報の地理的特性があるため、CITS においては、ジオネットワーキングと呼ばれるプロトコルが標準とされている。ジオネットワーキングは GPS や自己位置推定などによって得られる地理情報をもとに通信を行う。

ファシリティ層は車々間通信の基本的機能を扱う層である。ここでは、Cooperative Awareness Message (CAM)[11] や Collective Perseption Message (CPM)[12] などの C-ITS で標準化されているメッセージプロトコルを操作する機能や、それらの情報を保存しておくデータベースである、Local Dynamic Map (LDM)[13] などの機能がある。

アプリケーション層は自動運転用のソフトウェアに代表される、車々間通信の情報を使用するアプリケーション群のことである。

今後自動車には、自動運転だけでなく Mobility as a Service (MAAS) などの発展に伴い、多くのアプリケーションが搭載されることになる。各アプリケーションが個別に通信すると、通信帯域の逼迫やセキュリティの問題などが生じる可能性がある。C-ITS ではこれらの問題を解決するために、全てのアプリケーションがファシリティ層を介して通信し、通信部分を一つのルータにまとめる設計となっている。

2.4.2　C-ITS 標準メッセージ

先述した通り、C-ITS では CAM や CPM など標準化されているメッセージがいくつかある。CAM は自車両の位置や速度、加速度などを周囲の車両や路側機などに伝達するメッセージである。このメッセージの目的は自車両が重大な影響を与える情報を周囲に伝達することである。この目的を達成するために、このメッセージの情報は地理的に近い周囲の車両にのみ伝達すれば良い。したがって、CAM は基本的にシングルホップブロードキャストで送信される。

CAM のメッセージの構造を図 2.4 に示す。CAM は ITS PDU (Packet Data Unit) header[15]、Basic Container、HF (High Frequency) Container、LF (Low Frequency) Con-

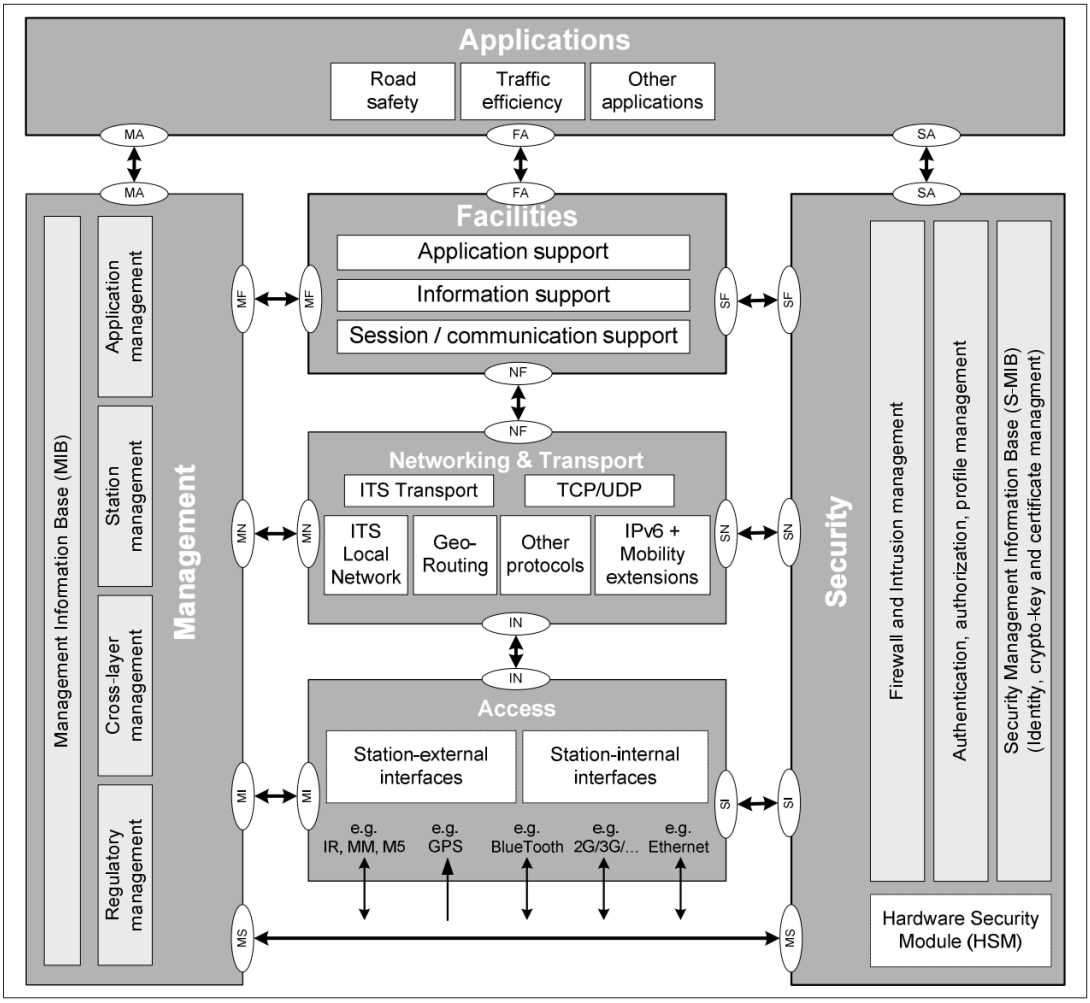


図 2.3: ETSI により標準化されている ITS ステーションアーキテクチャー構成 (出典:[14])

tainer、Special vehicle Container からなる。ITS PDU header にはプロトコルバージョンやメッセージ ID、周辺環境で一意なステーション ID が含まれる。現在、CAM のプロトコルバージョンは 1、メッセージ ID は 2 となっている。Basic Container にはそれぞれの ITS station の種類（車両、路側機、歩行者など）や位置情報の精度などが含まれる。HF Contaier には車両の位置や速度、方向などの動的な情報が含まれており、全ての CAM で必須の項目となっている。LF Container は車両の種類や軌跡などの静的または準静的な情報が含まれており、必須項目ではない。

CPM は自車両の情報だけでなく、自車両が検知した周辺状況を含めた情報を伝達するメッセージである。このメッセージの目的は、センサーや通信機器を搭載していないレガシー車や Vulnerable Road User(VRU) の知覚情報を補うことである。

CPM のメッセージの構造を図 2.5 に示す。CPM は CAM と同様に ITS PDU header があり、プロトコルバージョンやメッセージ ID などが含まれている。現在、CPM のメッセージ

ID は 1 となっている。その他に、自車両に搭載されているセンサーの情報を書き込む Sensor Information Container と、センサー郡で認識した物体の情報を書き込む Perceived Object Container がある。どちらも、128 個まで書き込み可能となっている。それに加えて、Free Space Addendum Container が 128 個用意されており、追加の情報を記述することができる。

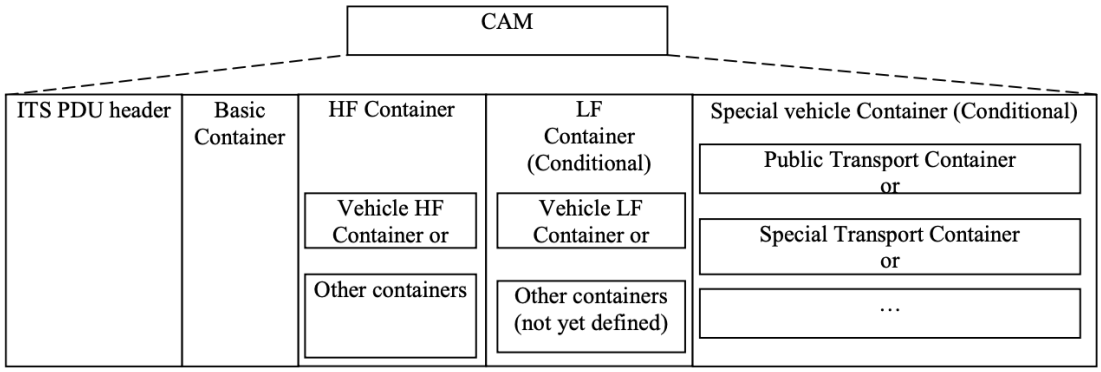


図 2.4: ETSI により標準化されている CAM の構造 (出典:[11])

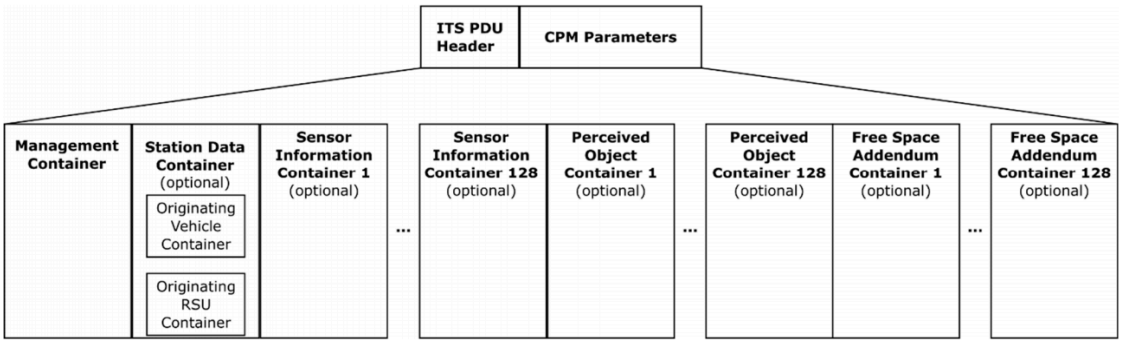


図 2.5: ETSI により標準化されている CPM の構造 (出典:[12])

2.5　MCM とその必要性

以上のメッセージにより、車両の死角を補填することができ、自律型自動運転における Perception の課題を解決することができる。さらに、レガシー車が混在する場面においても問題点を解決することができる。しかし、図 2.6、図 2.7 に示すシナリオにおいては不十分な点がある。

図 2.6 は、車線規制により車両 A が車線変更をする時、車線変更先の車線に並走している車両 B がいるシナリオを表している。この場合、先述の CAM によって相手車両の位置を知覚することができたとしても、車線変更することを認識することはできない。また、仮に車線変更することを認識することができたとしても、どちらが減速するか判断する必要がある。

図 2.7 は黄色い車両の四角に横断中の歩行者がいるシナリオである。このシナリオでは、黄色い車両から送られる CPM によって、青い車両は歩行者を検知できるが、歩行者の進行方向がわからないため、歩行者が横断中であるかどうかわからない。したがって、この場合も歩行者から車両に対して、減速の指示を出す必要がある。

以上のような場面で必要となるメッセージとして C-ITS では、MCM が標準化されている。欧州委員会 [16] の技術普及ロードマップによると、このメッセージプロトコルは 2045 年を目処に実用化が検討されている。更に、このメッセージプロトコルはまだ議論がされている段階で、具体的なメッセージプロトコルの構造などは決まっていないが、先述したメッセージの使用シナリオなどは多く議論されている。

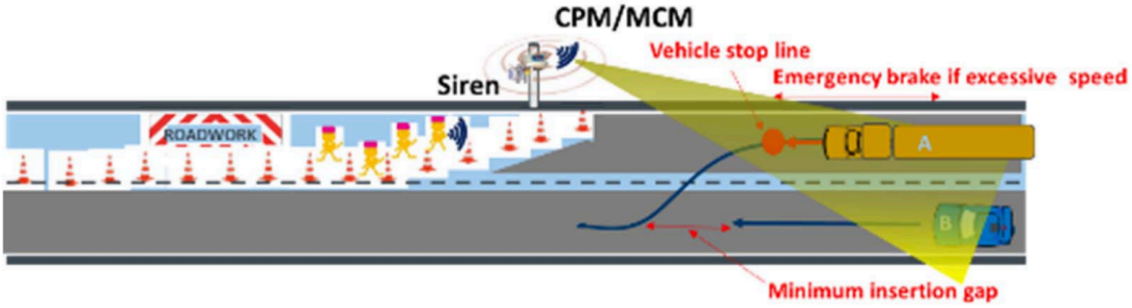


図 2.6: CAM や CPM では不十分なシナリオ 1(出典:[17])

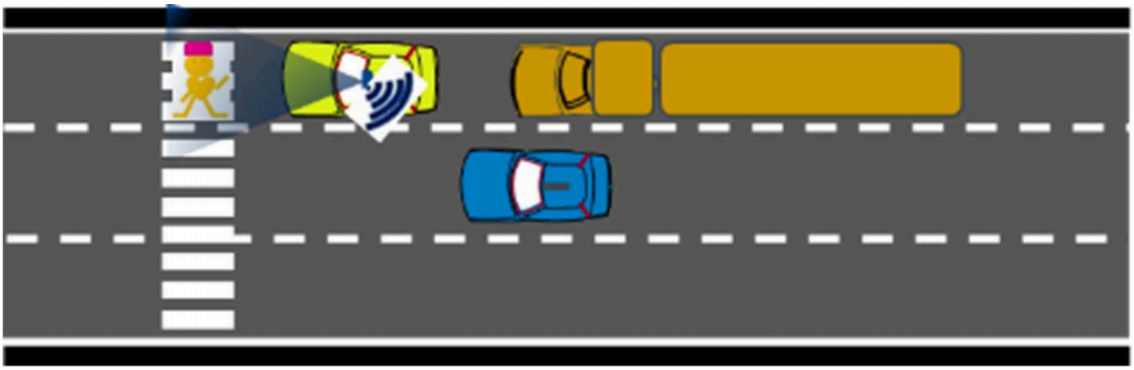


図 2.7: CAM や CPM では不十分なシナリオ 2(出典:[17])

2.6 OpenC2X

ETSI で標準化されている ITS ステーションでの通信メッセージを実装したオープンソースソフトウェアとして OpenC2X[18] がある。OpenC2X のモジュール構成を図 2.8 に示す。実線で示されたモジュールは既に実装されている。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 ヘッダーの実装のみで、ルーティ

ングは未実装である。

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 Decentralized Congestion Control(DCC) 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 Decentralized Environmental Notifi cation Message(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 2.9 である。図のプロトコルの欄に CAMv1 と書かれている通り、wireshark の最新バージョンでは ITS 標準の day1 メッセージである CAM を認識することができる。一方、本研究で新たに実装した MCM は標準にないメッセージであるため、プロトコルの欄にはネットワーク層のプロトコルである Basic Transport Protocol(BTP) が書かれている。このように、OpenC2X を使用して、GeoNetworking ヘッダーのついた CAM や MCM を送信することができる。

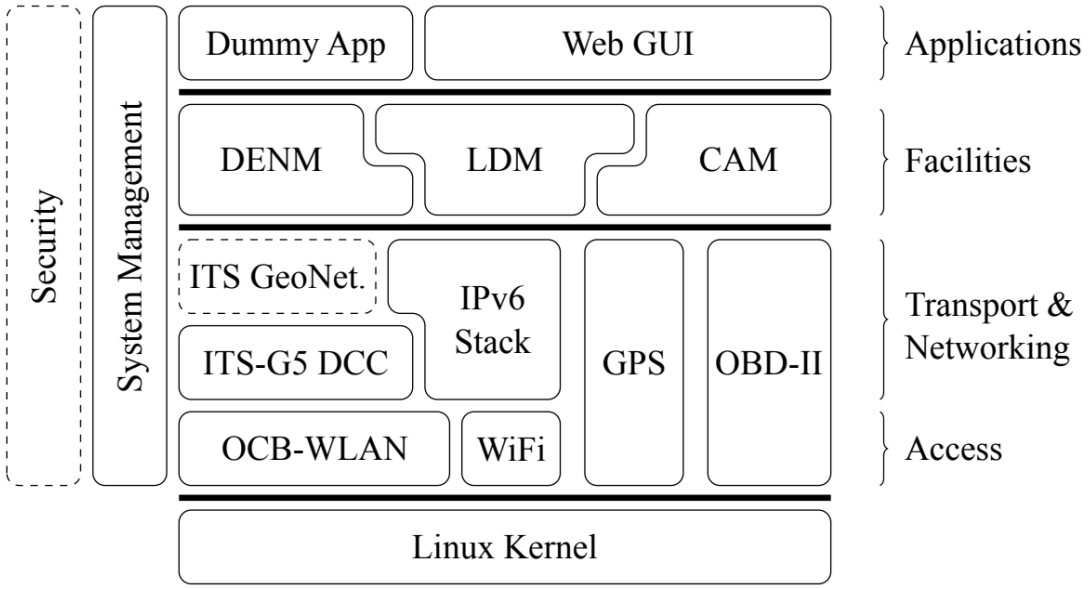


図 2.8: OpenC2X のモジュール構成 (出典:[19])

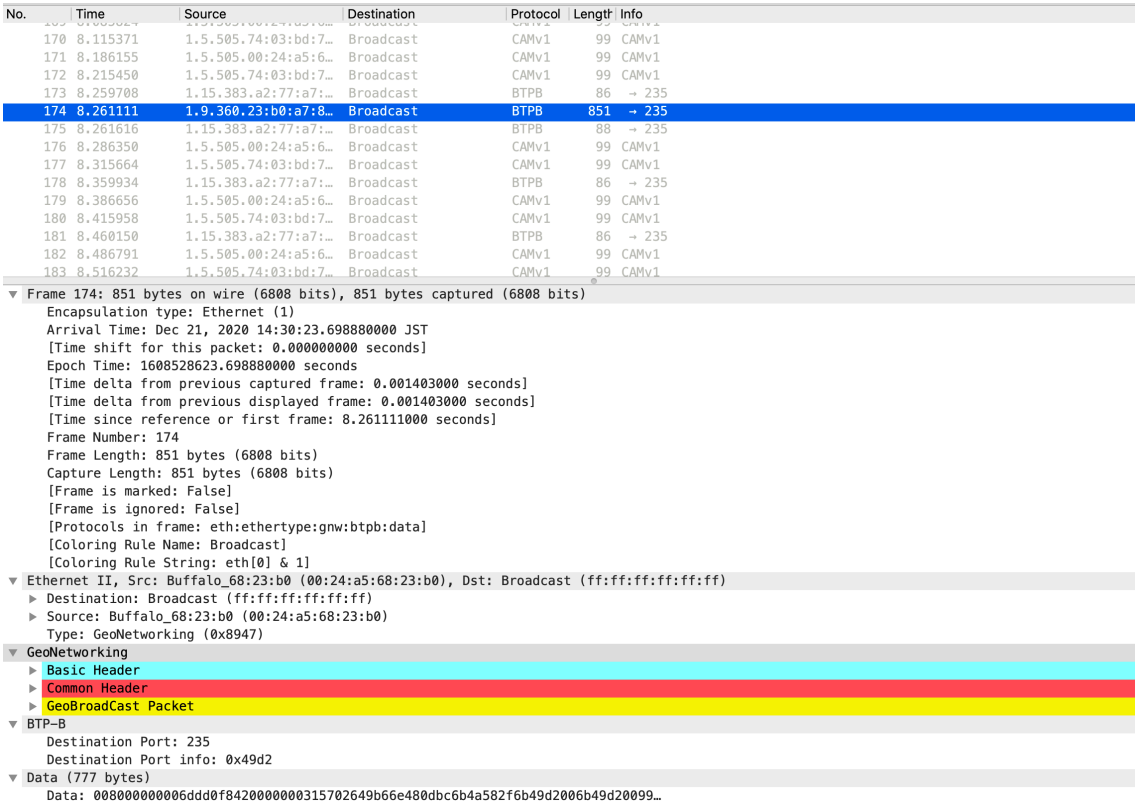


図 2.9: wireshark で OpenC2X が送信しているメッセージを観測している様子

第 3 章

関連研究

先述したとおり、MCM はメッセージのフォーマットも決まっていないが、走行調停に関する研究は多くされている。本章では、まず特定のシナリオに対する走行調停の研究を紹介し、次に汎用的なシナリオに使用できる走行調停プロトコルの研究を紹介する。

3.1 特定のシナリオに対する走行調停の研究

特定のシナリオ、特定の条件において、V2X を利用した走行調停を実現するための研究がある。

まず、platooning の調停に関する研究を紹介する。

Englund[20] らの研究では、platooning において想定される各シナリオに対応するための、V2X のプロトコルを提案した。その中で、2 つの platooning の列を一つの platooning に統合するために走行調停するプロトコルを考案した。その流れは、速度の同期、2 つの platooning から車両のペアを決定、前の車両から順に統合の準備、platooning の統合、である。このシナリオを検証するために Interactive Test Tool(ITT) を実装し、検証を行った。検証の結果、安全にシステムを実行するための自動運転車両、通信の要求スペックを明らかにした。

Oliver[21] らの研究では、platooning など、数台で協調走行を行う際の冗長なメッセージプロトコルを提案した。提案手法では、searching、vote、sync、in session という 4 つのセッションを設けている。まず、協調する車両を探すために beacon を一定周期で配信する (searching)。次に、各車両で協調するための情報を交換し、同意を取る（vote)。同意が取れない場合には、searching の状態に戻る。実行する前に、各車両の同期をとる（sync)。最後に、heartbeats を一定周期で送信し、シナリオを実行する（in session)。heartbeats を確認し、同期ができていない場合はもう一度 sync のセッションに戻り、同期を行う。ここでは、VSimRTI[22] というシミュレーションを用いて、2 台の車の platooning を行った際のセッションの持続性を検証した。検証の結果、メッセージの待ち時間を小さくするほど、セッションの持続時間が減少することがわかった。一方で、データの堅牢性を考えるとメッセージの待ち時間は小さい方がいいとしている。

以上のように platooning の特定のシナリオにおける、快適性を向上させるための調停メッ

セージは多く考えられている。

次に、合流、車線変更に限定した研究について紹介する。

Milanes[23] らの研究では、合流調停において、進入路からの車両をスムーズに合流させるためのシステムを提案した。提案手法は、合流箇所に RSU があり、進入路からの合流を検知すると、本道の車両の中で衝突する可能性のある車両を探索し、その車両に速度調節の指示を出すというシステムである。RSU からは、車両が安全な距離を保つために望ましい速度や位置を車両に送信し、受信した車両はその情報と自車両の情報をもとに、ファジィ制御を行う。望ましい速度や位置の計算には、Uno[24] らの提案した「仮想車両」を用いている。このシステムを ACC を搭載した車両を使用し、フィールド実験によってこのシステムを検証した。車線変更においてはこのシステムが安全性を向上させ、更に本道への影響を最適化することができることを示した。

Wang[25] らの研究では、車線変更を行うための調停メッセージとして Lane Change Message(LCM) を提案した。LCM には Lane change request と Lane change reply という二つのメッセージがあり、各メッセージを受信した際には Ack を返す。Lane change request は自車両の車線変更を伝達するメッセージであり、Lane change reply は request を受け入れるかどうかを示すメッセージである。request を受け入れる場合には、車線変更に干渉しないように自車両の trajectory を変更する。メッセージのフローは次の 2 つを検討している。一つは、Lane change request を受信した車両は無条件に車線変更を受け入れ、Ack のみを送信するパターンである。もう一つは、Lane change request を受信した車両は、車線変更を受け入れるかどうかを検討し、Lane change reply を返すパターンである。前者は、フローが簡潔で調停完了までの時間が短くなるが、request を受信した車両が無条件にメッセージを受け入れなければならない。一方、後者はフローが一つ増えるため、調停完了までの時間が長くなるが、request を受信した車両がメッセージを受け入れるかを決定することができる。各フローごとに、車両の密度に応じた車両の速度、メッセージの交換にかかる時間を検証し、パフォーマンスは前者のフローのほうが高く、LCM を使用しない場合と比較すると、どちらの場合も向上していることがわかった。

Eiermann[26] らは合流地点における調停メッセージとして Cooperative Maneuver Protocol(CMP) を提案していた。この研究では、trajectory をやり取りする MCM は帯域幅を逼迫するため、車間距離、相対速度などをやり取りする CMP を採用していた。プロトコルの流れは、合流地点 · 予想到着時刻 · 走行軌跡の把握 (situation analysis)、車間距離 · 相対速度の計算 (gap calculation)、正しい車間距離 · 相対速度の計算 (gap evaluation)、正しい車間距離 · 相対速度を満たすように走行 (action) である。この研究では、加減速だけでなく、合流するタイミングも可変として最適化を行っている。シミュレーションによって CMP のチャネルに対する負荷を計測し、50 台まで同時に調停できることを示している。

次に、Space Time Resevation Protocol(STRP) を使用した研究がいくつかある。[27, 28] STRP とは、一定時間、特定の場所を予約するメッセージである。そのメッセージを受信した他車両は、一定時間、予約された場所を通らないように軌跡を変更するというものである。

Daniel[27] らの研究では、車線変更の調停のプロトコルとして、STRP を提案している。

STRP の流れは、車線変更リクエスト、調停参加のリプライ、時刻 · 場所の予約、承認、である。このプロトコルを検証するために、フィールド実験を行っている。シミュレーションと実際の車両での走行を組み合わせることで、メッセージ生成までの計算時間、計算量、車両の速度の変化などを定量的に評価している。

以上で示したとおり、特定のシナリオに対する走行調停のメッセージプロトコルの研究は行われており、フィールド実験も行われている。一方で、使用用途が platooning や車線変更など限定的なシナリオに限られてしまう。

3.2　より汎用的な走行調停プロトコルの研究

より汎用的な V2X のメッセージに関する研究開発を目的とするプロジェクトとして、TransAID[29] がある。このプロジェクトでは協調型自動運転に関する研究が数多くされており、CAM、CPM などの C-ITS 標準メッセージに関しても研究を行っている。同様に MCM に関する研究も行われている。[30, 31]

Correa[30] らの研究では、より汎用的な MCM のメッセージフォーマットを提案した。[30] では、MCM での指示の種類として、車線変更や速度調整以外に Transition of Control(ToC)（自動運転から手動運転への切替）を考えた。自動運転が困難な場所に移る前に、RSU から ToC の指示を出すというシナリオを検証した。シミュレーションの結果、MCM を使用することで、衝突危険性が減少することが明らかになった。また、MCM の配信頻度がチャネルの使用率に与える影響も評価しており、動的に変化させることで、車両の密度が多い場合でも使用率を抑えることができることを示した。

Julian[31] らの研究では、C-ITS で標準化されている各メッセージセットを交換するための V2X のフレームワークを実装した。MCM では、[30] と同様に、ToC や車線変更をユースケースとして上げており、ここでは、いずれも RSU からの指示のみを考えている。[30] では、ToC や速度の調節が MCM の主な目的であり、より詳細な指示は送ることを考えていない。実際の車両を使用したフィールド実験により、想定されるシナリオ 15 個をテストした。

Richard[32, 33] らの研究では、アクセス層からファシリティ層まで、フルスタックの実装を行った。また走行調停のためのメッセージとして、MCM の他に Maneuver Recommended Message(MRM) を定義した。MRM は RSU から推奨される軌跡を各車両にアドバイスするためのメッセージである。MCM の主な使用シナリオとして、車線変更だけでなく、左折車両の譲りなどを考えた。メッセージの中身を定義しており、Target station id、Intention trajectory（自車両の将来の軌跡）、Requested trajectory（他車両への指示）に加えて、Duration（メッセージの有効期限）がある。検証では、プロトタイプを実機で動かし、車線変更のシナリオを行った。

Hafner[34] らの研究では、走行調停用プロトコルとして Complex Vehicular Interactions Protocol(CVIP) を提案した。CVIP では他で提案されている MCM と同じようにメッセージフローがある。まず走行調停の request を送信し、調停への参加を返信し、最後に調停に参加する車両に指示を送信する。ここでは、具体的にどのようなシナリオでメッセージを使用する

か、どのような指示をするのか、検討されていない。シミュレーションによって、各パケロス率によるプロトコルの成功確率を計測していた。

Daniele[35] らは、走行調停を行う際に、全車両が常に trajectory を配信していることを前提とした。また、自車両の通常の軌跡である planned trajectory と危険が伴う trajectory である desired trajectory を定義した。危険が伴う場合は desired trajectory を送信し、受信した車両は desired trajectory を回避するように最適な trajectory を計算する。ここでは、衝突回避の方法として、速度の加減速だけでなく、車線変更も含めていた。

Lehmann[36] らの研究では、MCM を使用して、自動運転車両の不確実性を減少させることを目的とする研究を計画している。[36] は、MCM をより一般的に使用する事ができるようプロトコルとして提案した。提案手法では、3 つのフェーズを設けている。まず、coordination が必要であるかを判定する、Detection phase。次に、車車間で交渉する Negotiation phase。更に、実行に移す、Execution phase である。メッセージの種類は planned trajectory と desired trajectory の 2 種類である。planned trajectory が自車両の将来の軌跡、desired trajectory が自車両の望ましい軌跡である。メッセージ交換の流れは以下の通りである。

1. planned trajectory を常に配信
2. 緊急の事象（自車両と他車両の planned trajectory が衝突）が起こる場合、desired trajectory を送信
3. desired trajectory を受信した車両はそれを許諾するか判断
4. （受け入れられる場合）planned trajectory を更新、実行する
5. （受け入れられない場合）desired trajectory を破棄

3 車線の幹線道路に 1 車線の道路から車両が合流するシナリオを SUMO[37] と OMNet++[38] というシミュレータを使用し、検証予定である。また、ここでは、MCM において重要なコーナーケースを挙げている。まず、desired trajectory の重複である。複数車両から同時に desired trajectory を送信した場合に、どちらかの trajectory を優先するための仕組みが必要である。もう一つが、cascading である。最初の desired trajectory が他車両の desired trajectory を引き起こし、連鎖的に maneuver coordination が発生してしまう場合である。いずれの場合についても、プロトコル設計に依存しないとして、詳細な解決策は触れられていない。

以上に示すとおり、MCM のメッセージフォーマットを提案し、チャネルに与える MCM の影響や、特定シナリオにおける MCM の有効性などの研究は行われている。一方で、汎用的な MCM に向けた議論は未成熟であり、かつ、ネットワーク遅延やパケットロスが走行の快適性に与える影響などを評価した研究はない。更に、オープンソースの自動運転用ソフトウェアを拡張し、C-ITS 標準に準拠したメッセージ交換による走行調停の検証も行われていない。最後に、コーナケースをメッセージプロトコル作成に組み込んだ研究もない。

第 4 章

MCM と MC Service の提案

C-ITS では ITS アーキテクチャの中で、MCM に関係する部分を Maneuver Coordination Service(MCS) と呼ぶ。MCS は Cooperation Awareness Service や Collective Perception Service などと同様にファシリティ層に位置する。前節でも述べた通り、MCM の使用用途やフォーマットなどについては議論が行われているだけで、標準化はされていない。したがって、MCS の機能についても定義されていない。しかし、MCM の目的は議論されており、自車両の将来の Trajectory を伝えること（Intention）と他車両に指示をだすこと（Prescription）が主な目的となる。したがって、本研究では、以上の目的を果たすための、MCM の要件を述べた上で、その要件を満たす MCM のフォーマット、MCS のプロセスを提案する。ただし、MC においては、指示を出すものと指示を出されるものの 2 つの役割があり、以後、前者を Prescriber、後者を Receiver とする。

4.1 要求事項

本研究では汎用的に使用できる MCM を提案し、実装し検証することが目的であるが、MCM の実現にあたって要求される事項がある。以下では、それぞれの要求事項を、安全性・汎用性・標準への準拠・帯域幅の節約・指示の重複・指示の連鎖に分類し、説明する。

4.1.1 安全性

MCM は自車両の将来の軌跡を送信し、他車両の将来の軌跡を指示するメッセージである。MCM の送信が正しくできないと、直接重大な事故につながるため、CAM や CPM よりもより安全性が必要となる。安全性という観点では要求事項は大きく分けて 4 つある。

1 つ目は、**メッセージをロスした場合でも正しくメッセージを伝達可能（R1）**である。自動車は比較的高速な移動体であるため、通信の安定性は保証できない。MCM のやり取りの途中で通信ができなかった場合、正しい指示を伝達できないこととなるが、Prescriber は指示が伝達されている前提で走行するため、重大な事故につながる危険性がある。したがって、メッセージロスした場合でも情報を正しく伝達できる仕組みが必要である。

2 つ目は、**メッセージが正しく送信できていることを確認できる必要性（R2）**である。先述の通り、MCM は走行の安全性において重要度が高いため、メッセージが送信できていない場合には再送などの処理が必要である。そのために、送信が正しくできたかを確認できる必要がある。

3 つ目は、**異常事態やシナリオ変更への対応可能（R3）**である。異常事態には大きく 2 つあり、通信の不具合と車両自体の異常による緊急停止である。どちらのケースに対しても、MCM によるシナリオを遂行することはできないため、シナリオを停止する必要がある。したがって、異常事態があった場合に、安全にシナリオを停止できることが必要となる。

4 つ目は、**危険な指示や運行の妨害をするような指示を拒否可能（R4）**である。通信の不具合で、間違った指示を受け取ってしまう場合や、不正な指示によって、運行を邪魔しようとする車両がいる場合もある。これらの指示を全て鵜呑みにし、走行を続けることは非常に危険である。したがって、危険だと判断される指示に関しては、許可しない仕組みが必要である。

4.1.2 汎用性

MCM の使用されるユースケースは、前節で述べたとおり、合流調停、車線変更に限らず、交差点でのスムーズな交通など、多岐に渡る。また、将来的には、車両だけでなく、RSU や歩行者の持っているモバイル端末も MCM をやり取りする。したがって、MCM は**様々なシナリオで使用でき、かつ様々なステーションタイプに対応可能（R5）**である必要がある。

4.1.3 快適性

MCM はよりスムーズな交通をすることが、ひとつの目的である。快適性能に影響するのはアプリケーションのアルゴリズムと異常事態やシナリオ変更への対応速度である。前者は、MCM の設計に非依存であるため、ここでは後者のみ説明する。異常事態やシナリオ変更に対応しなかった場合には、指示を受けた車両はその指示通り進行するため、危険であるだけでなく、快適性にも影響が生じる。例えば、調停の結果、速度を減速する指示を受けた場合、そのシナリオが途中で終了している場合でも、減速したまま走行し続けることになる。したがって、**異常事態やシナリオ変更への対応可能（R3）**である必要がある。

4.1.4 標準への準拠

協調型自動運転では欧州、アメリカ、日本でそれぞれ車々間通信に関する標準化がされている。C-ITS における標準化はメッセージを扱うファシリティ層だけでなく、その下のネットワーク・トランスポート層やアクセス層も標準化されている。例えば、ネットワーク・トランスポート層においては、GeoNet/BTP を標準としており、マルチホップの場合、ルーティングは地理情報をもとに行われる。そのため、ネットワーク・トランスポート層における各 Station の識別子は地理情報である。また、高速な移動体の通信においてはトランスポート層におけるフラグメントは好ましくないため、C-ITS では推奨されていない。以上の前提から、

MCM には次の 2 つの要求事項がある。

1 つ目は、**MCM が送信先の車両を指定可能（R6）**である。先述の通り、ネットワーク層では、基本的に地理情報による識別しか想定されておらず、通信 ID での送信先の指定ができない。したがって、ファシリティ層である MCM で送信先の指定をできる必要がある。

2 つ目は、**フラグメントしない（R7）**である。C-ITS では、トランスポート層でのフラグメントが推奨されていない。したがって、メッセージサイズを 1500Bytes 以下に抑える必要性がある。

4.1.5　帯域幅の節約

C-ITS では V2X でのアドホック通信でメッセージの交換を行う。また、各車両がメッセージをブロードキャストで送信するため、通信量は多くなる。MCM は自車両の位置や速度だけでなく、将来の軌跡をメッセージに含めるため、一つのメッセージの量は非常に大きくなる。以上の点から、MCM を一定間隔で送信し続けることは、帯域幅の逼迫につながる。したがって、MCM を適切なタイミングでのみ送信することで、**帯域幅の節約（R8）**をすることが必要である。

4.1.6　指示の重複

1 台の車両が MCM の指示を受信した後に、他の車両や RSU から別の指示を受け取る場合がある。したがって、**指示の重複に対応可能（R9）**である必要がある。この場合、考えられる対応としては 2 つある。1 つ目は、2 つ以上指示が来た場合には後から来た指示を全て断るパターンである。この場合、車両が、指示を受け取り実行していることを、状態として持っている必要がある。2 つ目は、後から来たメッセージの指示に上書きするパターンである。この場合、指示が上書きされ、破棄されてしまったことを前の指示を送信した車両に伝達する必要がある。したがって、この場合にも車両は、指示を受信し実行していることを状態として持っている必要がある。いずれの場合にも、状態で管理されている必要がある。本研究では、以下 1 つ目のパターンで考察を行う。

4.1.7　指示の連鎖

一つの車両が他車両に指示を出す際に、他車両がその指示を実行に移すためには、更に別の車両に指示を出さなければならない場合がある。そして、更にその車両が受け取った指示を実行に移すためにするために、他の車両に指示を出さなくてはならない場合もある。このように、指示が別の指示の連鎖を引き起こしてしまうことが考えられる。したがって、**指示の連鎖に対応可能（R10）**である必要がある。

4.2 MCM

以下、MCM のフォーマットと MCM を交換する流れについて説明する。まず、各要求事項を満たす設計方針を述べる。

4.2.1 要求事項を満たす設計方針

各要求事項ごとにそれを満たすような設計方針を述べる。

■基本的なメッセージセット

MC を実現するために最低限必要なメッセージの種類として、以下が挙げられる。

- **Intention**: 自車両の将来の軌跡を周囲に伝達するメッセージ
- **Prescription**: 他車両の将来の軌跡を操作するメッセージ
- **Acceptance**: Perception を受け入れるかどうかのメッセージ

まず、Intention は Prescription をする前に、他車両と衝突するかどうか（軌跡がかぶるかどうか）を正確に判断するために必要である。

そして、MCM の主たる目的は、車車間で調停をし、他車両の軌跡を変更することである。Prescription は MCM の主たる目的を実現するものであるため、自明に必要である。

Acceptance は Prescription を受信した車両が、その軌跡を受け入れるかを返答するメッセージである。その軌跡の通りに走行して、危険であった場合にはその指示を拒否することができる。これにより、**R4: 危険な指示や運行の妨害をする指示を拒否可能**を満たす。ここで、指示の連鎖のシナリオを考える。指示の連鎖は様々な MCM の研究で考えられているコーナーケースであるが、一般的な解決策は出ていない。本研究では指示の連鎖を許さないという制約を設ける。つまり、指示を受け取った車両が他の車両に指示を出さなくてはならない場合は、その指示を許可しないという制約を設ける。**R4** で、指示を拒否する仕組みを導入することをのべたので、**R10: 指示の連鎖への対応**を満たす。指示を拒否するか判断する、詳細なアルゴリズムなどはシナリオ、アプリケーションに依存するため、ここでは割愛する。

本研究では、以上のメッセージ以外に Advertisement というメッセージを考える。Advertisement は、車線変更などの MC が必要なイベントを周囲に告知するメッセージである。このメッセージを受信した車両は、Intention を返信する。これにより、常に、Intention を配信し続ける必要がなくなり、**R8: 帯域幅の節約**を満たすことになる。

■State での管理

以上のようにメッセージが複数あり、状況に応じて配信するメッセージが異なるため、state により現在の車両の状態を管理する。state の詳細は後述するが、大きく分けると、Advertising、Prescribing、Negotiating、Actuating である。Advertising が、イベントを周囲に告知している状態、Prescribing が指示を出している状態、Negotiating が指示を受け入

れるか判断している状態、Actuating が指示を実行している状態である。

更に、state で管理することにより、一度指示を受信し、その指示を実行している間は、他の指示を受け付けなくなる。すなわち、指示の上書きは起こらないため、**R9: 指示の重複への対応**を満たす。

■Ack メッセージによる応答確認

「一度だけ送信できれば良いが、送信できたか確認が必要なメッセージ」に関しては、以下のルールに従ってメッセージを交換する。上記のメッセージを受信した側は受信したことを伝達するメッセージ（Ack）を返信する。メッセージを送信する側は Ack が返ってくるまで、送信し続け、Ack が返ってきた時、メッセージの送信を停止する。このルールによって、メッセージを受信したことを認識でき、**R1: ロスした場合でもメッセージを正しく伝達可能**と **R2: メッセージが正しく送信できていることを確認可能**の二つの要件を満たす。

■シナリオの異常終了にまつわるルール

要件にもあったように、自車両の内部に異常を生じた場合や、途中で目的地が変わり衝突が起こらない場合には、シナリオを途中で終了するべきである。その場合には、Cancel メッセージを配信し、シナリオを終了させる。また、内部の異常の他に通信の異常が生じる場合がある。異常の場合、メッセージが他車両に伝わらないため、異常を検知することができない。したがって、相手車両から、所望のメッセージを一定時間以上受け取れなかった場合、そのシナリオを終了し、通常の走行に戻る。これらの、ルールによって、**R3: 異常事態への対応可能**を満たす。

■Target station id の指定

前述の通り、C-ITS ではネットワーク層で、車両を一意に特定する識別子は、地理情報だけである。地理情報は刻一刻と変化するため、送信先の誤認識を引き起こす可能性がある。また、MCM はメッセージの内容に地理的特性がないため、地理情報で送信先を指定することは向いていない。一方で、C-ITS で定義されているファシリティ層のメッセージには StationID という各ステーションを一意に特定できる ID が存在する。したがって、MCM では TargetStationID というパラメータを用意し、送信先を一意に特定できるようにする。受信側は、ファシリティ層で、TargetStaionID が自身の StaionID と一致しているかどうかを判断し、一致していれば、自身に当てたメッセージであると認識する。以上の仕組みにより、**R6: 送信先の車両を指定可能**を満たす。

■システムの設計方針

MC を使用するシナリオが複数あり、それぞれのシナリオ、アプリケーションで、共通部分と固有部分がある。共通部分としては、基本的なメッセージセット、メッセージフローなどが挙げられる。固有部分としては、Prescription の生成方法や、scenario trigger のタイミングなどが挙げられる。従って、共通部分をファシリティ層に機能抽象化し、固有部分と分離した設計にすることで、新たなアプリケーションや様々なシナリオに対応させることを可能にする。これにより、**R5: 様々なシナリオで使用でき、かつ様々なステーションタイプに対応可**

能を満たす。

4.2.2 MCM のフォーマット

本研究では前節で述べた設計方針に基づくメッセージを提案する。メッセージの構造を図 4.1 に示す。まず、全てのメッセージは CAM や CPM などと同様に、メッセージのデータ部分である MC Container の他に、itsPduHeader があり、送信元の Station ID や Station type など基本的な情報が記されている。そして、Maneuver Container には複数種類の Container があり、場合によって選択する。それぞれ、Advertisement、Intention、Prescription、Acceptance、Fin、Ack、Cancel である。以下で各メッセージについて詳細に説明する。

<table>
<tr><td rowspan="14">MCM</td><td colspan="5">ItsPduHeader</td></tr>
<tr><td rowspan="13">Maneuver Coordination</td><td colspan="4">GenerationDeltaTime</td></tr>
<tr><td rowspan="12">MCM Parameters</td><td colspan="3">BasicContainer(ReferencePosition + StationType)</td></tr>
<tr><td rowspan="11">Maneuver Container</td><td>Advertisement Container</td><td>Scenerio</td></tr>
<tr><td rowspan="2">Intention Container</td><td>Target station ID</td></tr>
<tr><td>Planned trajecoty</td></tr>
<tr><td rowspan="2">Prescription Container</td><td>Target station ID</td></tr>
<tr><td>Prescribed trajectory</td></tr>
<tr><td rowspan="3">Acceptance Container</td><td>Target station ID</td></tr>
<tr><td>Advice accepted</td></tr>
<tr><td>Selected trajectory</td></tr>
<tr><td rowspan="2">Fin/Ack/Cancel Container</td><td>Target station IDs</td></tr>
<tr><td>Control flag = Ack OR Fin OR Cancel</td></tr>
</table>

図 4.1: MCM の構造

- Advertisement
  車線変更や合流調停など他車両に対して調停を行いたい車両が最初に送信するメッセージである。このメッセージにはシナリオが含まれている。
- Intention
  Advertisement を受け取った車両が返信するメッセージである。周囲にいるということを伝えると同時に、Planned trajectory を含んでいる。また、C-ITS 標準のメッセージでは、基本的に位置情報以外で送信先を特定できないので、どの車両に返信しているかを明示的にするために、Target Station ID を含んでいる。
- Prescription
  Intention を返した車両に対して、指示を出すメッセージである。指示を出すために、

Prescribed Trajectory（自車両が希望する、相手車両の将来の軌跡）を含んでいる。Intention と同様に Target Station ID を含んでいる。

- Acceptance
  Prescrition を許可するかどうかを伝えるメッセージである。Target Station ID と Accepted（許可するかどうか)、Selected Trajectory を含んでいる。
- Fin
  シナリオの終了を伝えるメッセージである。
- Ack
  後述するメッセージ交換のフローでも説明するが、メッセージを受け取ったことを他車両に伝えるメッセージである。
- Cancel
  MCM のやり取りの最中にシナリオの変更や車両の異常停止など、途中でシナリオを停止しなければならないときに送るメッセージである。

ここで、各 trajectory の意味は以下のとおりである。

- **Planned Trajectory**: 車両の将来の軌跡
- **Prescribed trajectory**: 他車両を指示する軌跡
- **Selected trajectory**: 選択された軌跡

ただし、各 trajectory には各時刻の車両の座標が含まれている。

4.2.3　MCM を交換するフロー

本研究の提案手法では、MCM のやり取りに状態をもたせる。状態遷移図を図 4.2 に示す。状態遷移については、以下で具体的なシナリオの例をもとに説明する。

前述の通り、MCM を使用するシナリオは複数ある。MCM を交換するシナリオとして、ここでは車線変更を考えて説明する。シナリオの状況とメッセージのやり取りを図 4.3 に示す。

車両 A（ステーション ID: 1）が車線変更をし、車線変更先に車両 B（ステーション ID: 2）がいるというシナリオを考える。車両 A と車両 B はそのまま何もしないで進むと、車線変更するタイミングで衝突してしまう。したがって、車両 B は、車両 A と衝突しないために、十分に安全な車間を取る必要がある。以下、車両 A と車両 B がメッセージをやり取りし、実行に移すまでのプロセスを各フェーズごとに説明する。

1. Advertising
   始め、車両 A と車両 B はともに Waiting の状態である。車両 A は車線変更をする前に、車線変更の情報を自動運転用ソフトウェアなどから受け取り、Advertising の状態に遷移する。そして、Advertisement メッセージに自車両の Planned Trajectory と車線変更というシナリオを含ませる。Advertisement は一定時間、一定の周期でブロードキャストする。Advertisement を受け取った車両 B は Planned Trajectory、Target

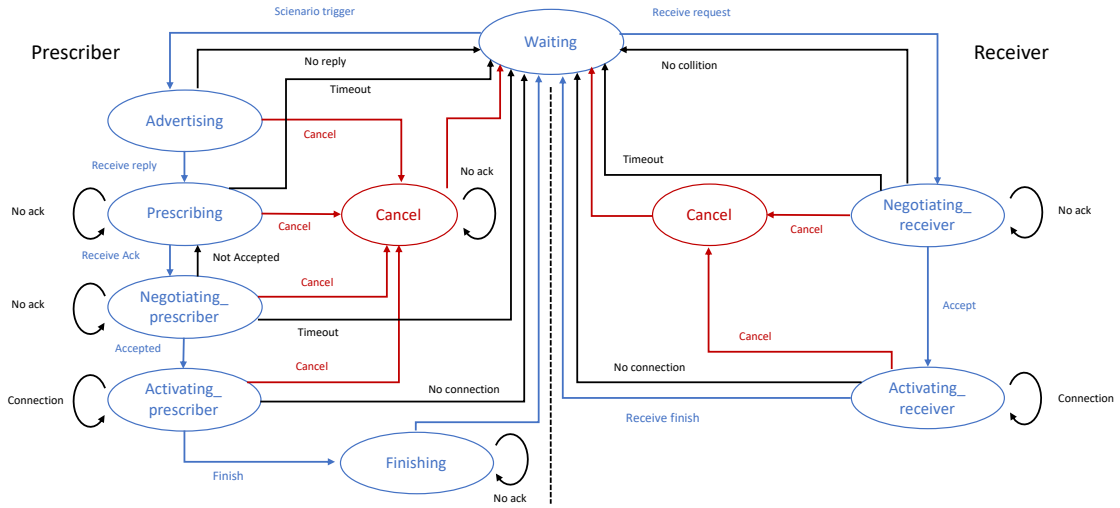


図 4.2: MCM の状態遷移

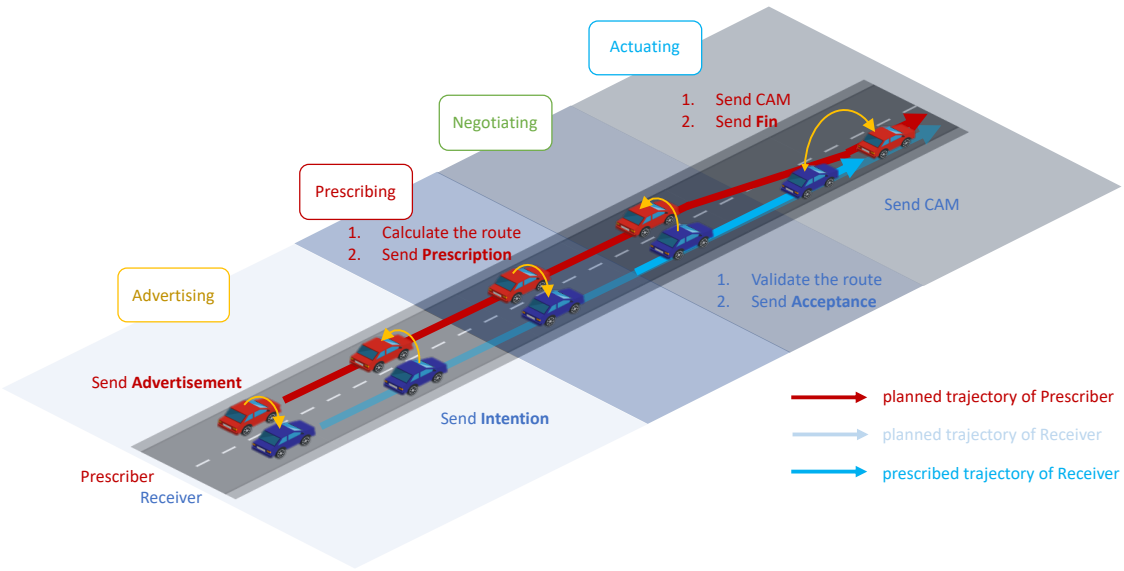


図 4.3: MCM 交換のフロー

Station ID = 1 を含んだ、Intention を送信する。Intention は受信先（この場合車両 A）から Ack がくるまで、一定周期で送信し、Ack を受信したら状態を Negotiating にする。この先の Prescription や Acceptance などについても同様に Ack が返ってくるまで一定周期で送信し、Ack を受け取ると次の状態に遷移する。

2. Prescribing
次に車両 A は一定時間以内に Intention を受信していた場合（今回であれば車両 B から受け取る）、状態を Prescribing にする。車両 B の Planned Trajectory を修正し、Prescribed Trajectory を計算する。そして車両 B の Prescribed Trajectory を含んだ、

Prescription を送信する。

3. Negotiating
   車両 A は送信できたことを確認すると Negotiating の状態に遷移する。車両 B は Prescription を受け取ると、Prescribed Trajectory が実現可能か（危険でないか、他車両を操作しなくても実現できるか）を検証し、Prescribed trajectory に従うかを Acceptance として送信する。もし車両 B が Prescrition に従わない場合は、車両 B の状態は Negotiating のままで、車両 A は状態を Prescribing に戻す。そして、車両 A は、もう一度 Prescribing を送信する。一定時間これを繰り返しても、Prescribed Trajectory が受け入れられない場合は、状態を Cancel の状態にし、Cancel を車両 B に送信した後送信が確認できたら、状態を Waiting にする。Cancel を受け取った車両 B も状態を Waiting にする。
4. Actuating
   車両 B が Prescription に従う場合は車両 B も車両 A も状態を Actuating にし、車両 B は受信した Prescribed trajetory をもとに走行する。Actuating の間、相手車両が正常に動作しているかどうかは受信した CAM によって判断する。最後に、車両 A はシナリオを終了するタイミングで、状態を Finishing にし、Fin を送信する。送信が確認できたら、状態を Waiting にする。車両 B も Fin を受信したら、状態を Waiting にする。

以上が正常系で動作した際の状態遷移である。一方で、途中でシナリオを変更したり、車両が緊急停止した場合は、Cancel という状態に遷移し、相手車両に Cancel を送信する。送信が確認できたら、状態を Waiting に戻す。また、通信系の異常で、一定時間パケットを受信できなかった場合は、状態を Waiting に戻す。

以上の仕組みにより、通信異常やシナリオ変更による、無駄な軌跡の変更を最小限に抑えると同時に、安全性も向上させることができる。かつ、車線変更以外にも交差点での調停や合流調停にも汎用的に使用できる仕組みとなっている。

4.3　MC Service

前節で説明した機能の実現をするサービスの設計を説明する。設計方針にもあった通り、走行調停のすべてのシナリオに共通する部分を MC Service の機能、固有部分を MC Application の機能とする。MC Service には、MC Application とデータをやり取りするためのインターフェースがあり、メッセージを受信すると、インターフェースを介してデータをアプリケーションに伝達する。また、アプリケーションで生成したデータをインターフェースを通じて受け取り、メッセージを生成、送信する。さらに、state を管理しており、state に応じて、アプリケーションにデータを生成するように指示をしたり、メッセージをドロップしたりする。state を管理する機能は、アプリケーション側にあっても良いが、本研究では、state の管理は MCM において根幹となる機能であり、かつ全てのアプリケーションで使用する機能であるた

め、MC Service の機能とした。

次に、アプリケーションの機能について説明する。一般的な自動運転用アプリケーションでは主に、Localization、Detection、Planning がある。Planning は Localization や Detection の結果を使用して行われるが、今回は直接関係のある Planning の機能について説明する。Planning において、MC Service を実現させるために必要十分な機能は以下の 5 つである。

- シナリオトリガー
  MCM を送信するトリガーとなる機能である。車線変更などのイベントを検知し、メッセージを生成するために、ファシリティ層にシナリオを登録する。
- Planning trajectory の生成
  Station が車両である場合、MCM を行うために必要である、自車両の将来の軌跡である。逐次自車両の将来の軌跡を生成し、ファシリティ層に伝達する。
- Prescribed trajectory の計算
  MC Service を実現するためには、他車両に指示するための軌跡を計算する必要がある。他車両の Planned Trajectory がアプリケーションに伝達されると、逐次、Prescribed Trajectory を計算する。
- Prescribed trajectory の検証
  Prescribed trajectory は必ずしも安全な軌跡ではない。したがって、Prescribed trajectory 通りに走行して安全かどうかの判断をする必要がある。安全かどうかの判断は各アプリケーションによって異なる。
- Prescribed trajectory の読み込み
  MC Service を実現するためには、送られてきた Prescribed Trajectory を自車両の Planning に読み込む機能が必要である。アプリケーションは、自車両の Prescribed Trajectory がアプリケーションに格納されると、逐次、Prescribed Trajectory を読み込む。

4.4 MCM を使用するシナリオ

MCM を使用するシナリオとして、現状考えられているものは、車線変更、合流調停、交差点での路側機のアシスト、緊急車両の道譲り、歩行者の横断である。

先述のメッセージ交換のフローでは、車線変更を考えたが、その他のシナリオにおいても、同じメッセージセット、同じフローで MC を実現することができる。先述の MC Service の機能のうち、シナリオ、アプリケーションによらない部分は、メッセージの生成、state での管理である。一方、シナリオごとに、Scenario trigger のタイミング、Prescribed trajectory の計算方法は異なる。また、Prescribed trajectory の実行は自動運転用のソフトウェアに依存する。

例えば、車線変更であれば、scenario trigger は車両が車線変更するタイミングで発行され、一定時間 Advertisement を配信した後、State を遷移させる。また、Prescribed trajectory

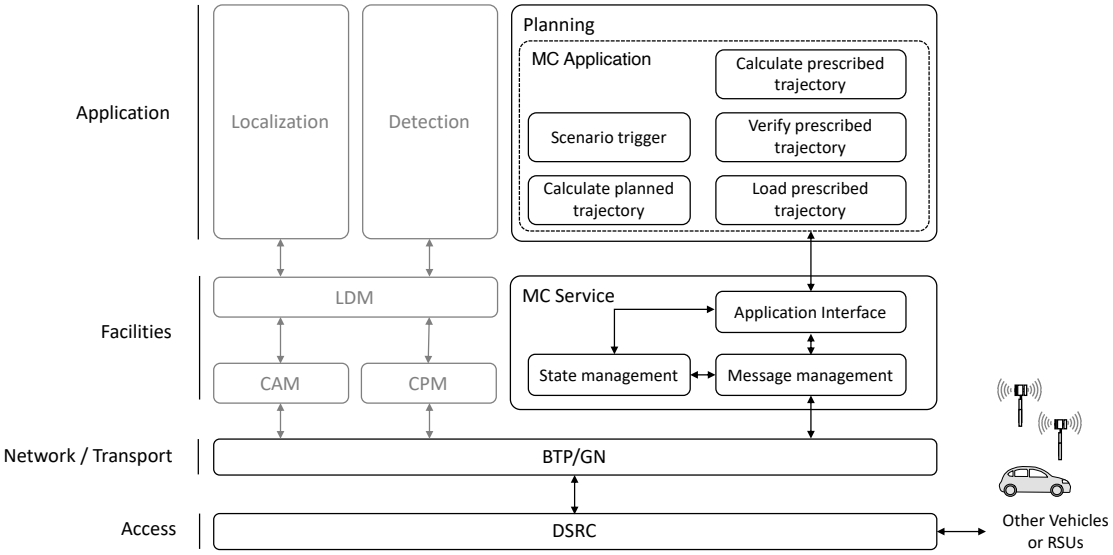


図 4.4: システムの概要図

は、Prescriber とぶつからない位置まで、相対的に車両を下げるような、trajectory になる。

一方、合流調停や交差点での調停では、交差点で常に調停を行うために、RSU が常時 Advertisement を配信している。したがって、scenario trigger はこの場合は必要ない。また、Prescribed trajectory は、車線変更の場合と同様に、衝突を回避するように車両を減速させるようことで求める。ただし、一方向が優先になってしまうことがないように、信号機のように交互に車両を通過させる必要がある。

第 5 章

AutoMCM の実装

本研究ではある特定のシナリオで、MCM を使用することで、自動運転車同士の交通をよりスムーズにすることを、実験によって示した。そのために、オープンソースの自動運転用ソフトウェアである Autoware を拡張し、MCM を送受信できるように実装した。メッセージの通信部分は、オープンソースの協調型運転用ソフトウェアである OpenC2X を拡張し、実装した。本研究では実装したソフトウェアを AutoMCM と呼ぶ。

5.1 実装で用いたソフトウェア

本研究で用いたミドルウェア・ソフトウェアのバージョンを表 5.1 に示す。

まず、Autoware は ROS というロボット制御のためのミドルウェア上で動作している。Autoware はバージョンによって、サポートしている ROS のバージョンが異なる。また、ROS は最新では ROS2[39] がリリースされており、ROS2 に対応している Autoware.Auto[40] というバージョンもある。しかし、Autoware.Auto は、機能が完全に移行できていないため、本研究では、ROS 上で動作する Autoware を使用する。

ROS 上で動作する Autoware には Autoware.AI という、一般に公開されているバージョンとは別に、Tier4.inc 社が独自に開発し、より先進的な機能を搭載する Autoware.IV というプロトタイプのバージョンがある。Autoware.IV で動作が安定し次第、順次 Autoware.AI に機能が移植される。Autoware.IV は一般には公開されていないが、本研究では、より機能の多い Autoware.IV の最新版を使用した。

OpenC2X は standalone 版 [41] と、OpenWrt からのフォークである embedded 版 [42] がある。機能的に大きな差はないため、本研究では、standalone 版の version1.5 を使用した。

Autoware と OpenC2X を連携するために、独自に拡張した部分は、全て C++ で実装した。

5.2 実装の概要

本研究で実装したシステムのうち、MCM に関連するプロセスの概要図を図 5.1 に示す。本研究では、図 4.3 のような車線変更のシナリオを想定し、実験を行う。したがって、図には、

表 5.1: 実装に使用したソフトウェア

名前	バージョン
ROS	melodic
Autoware	Autoware.IV v0.8.0
OpenC2X-standalone	1.5

車線変更のシナリオでのみ行うプロセスも含まれている。

各プロセスは、ROS Topic の pub/sub によってデータをやり取りする。ROS には rosbridge server[43] という websocket server が実装されており、ROS 上で動作しないアプリケーションも json を介してデータのやり取りをすることができる。OpenC2X は ROS 上で動作しないため、websocket client で rosbridge server に接続し、json 形式のデータをやり取りする。

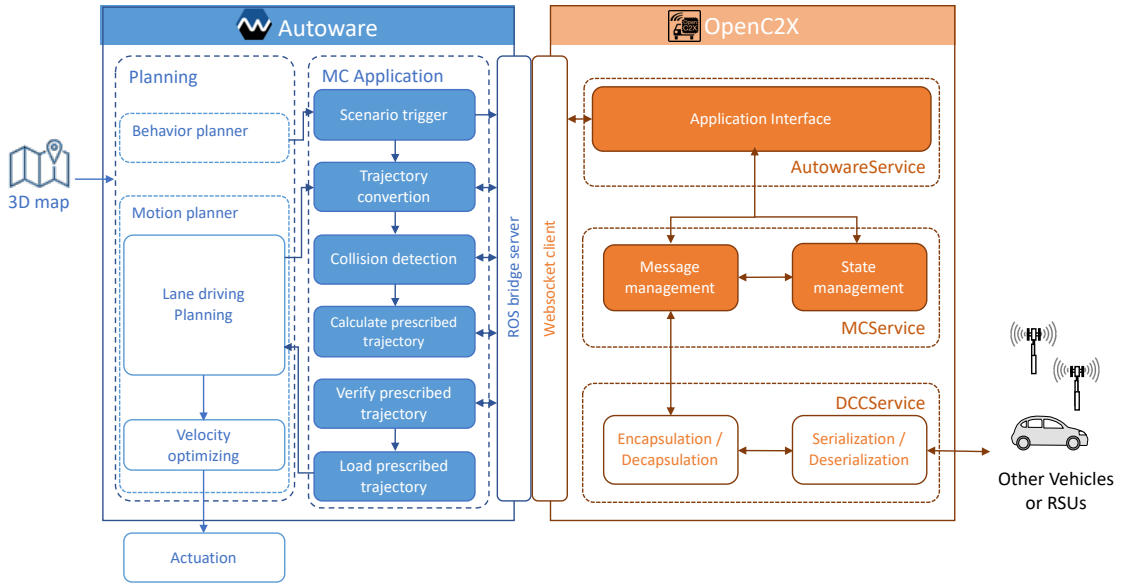


図 5.1: 本研究で実装したシステム概要

5.3　各メッセージごとの実装

各メッセージごとに各プロセスの詳細とデータの流れについて説明する。ただし、図 5.2, 図 5.4, 図 5.5, 図 5.8, 図 5.9 は、各メッセージ生成までの流れの概要である。Autoware 側の角丸四角形で表されているもの、矢印の上に書かれているものはそれぞれ、ROS のノード、トピックを表す。MC Application と書かれているものが一つのノードを表しており、その中の角丸四角形は各プロセスを表している。白抜きされているものと灰色の矢印で表されているものは既存の実装であり、色で塗りつぶされているもの、黒色の矢印は本研究で新たに実装した

部分である。OpenC2X 側では各機能ごとに一つの角丸四角形で表されており、機能間のデータのやり取りとそのデータの中身を矢印で表している。

5.3.1 Advertising

advertisement メッセージを生成する流れを図 5.2 に示す。メッセージ生成までの流れは以下の通りである。

1. Autoware の Lane Driving Planner というノードから、将来の軌跡を表す、*/trajectory* というトピックを subscribe し、逐次、時刻つき trajectory に変換し、*/ego_planned_trajectory* というトピックとして publish する。trajectory の変換については後述する。
2. Autoware の Turn Signal Decider というノードから、車両のウィンカーを意味する、*/turn_signal_cmd* というトピックを subscribe し、シナリオ実行中でないときに、メッセージを受信した場合、*/scenario_trigger* というトピックを publish する
3. Ros Bridge を経て、json 形式に変換されたデータを OpenC2X 側の State Manager が受信した時、State が Waiting であれば、Message Manager に Advertisement を配信するように指示する。
4. Message Manager は State Manager からの指示を受け取ると、ROS から受け取った最新の ego_planned_trajectory を Advertisement として生成し、シリアライズ化した後、DCC にデータを渡す。
5. DCC は受け取ったデータを GeoNet でカプセル化し、メッセージを配信する。

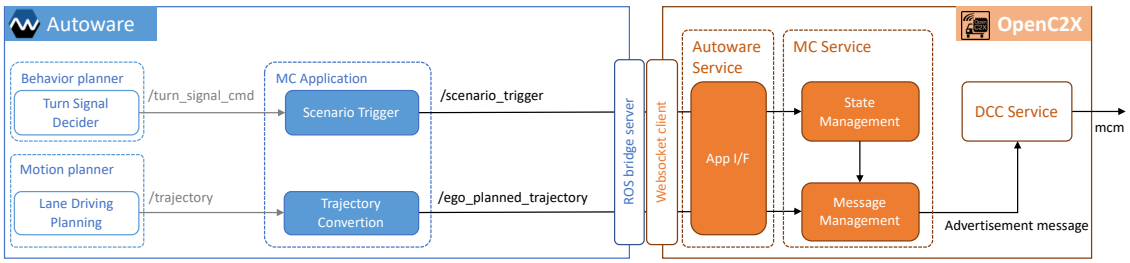


図 5.2: advertisement 生成までの実装と流れ

■trajectory の変換

ここで、trajectory の変換について詳しく説明する。Autoware 内で発行される trajectory は、図 5.3 のように、座標とその座標における速度の列で表されている。MC では衝突検知や、詳細な車両の操作のために、時刻と位置の関係が必要である。したがって、以上の位置と速度の関係から、各時刻の車両の位置を計算する。まず、trajectory の中で現在の車両の座標に最も近い点 x_0, v_0, t_0 を探索する。次に、現在の位置から次の位置までの速度を一定であると仮定し、次の位置に到着する時刻 t_1 を $\frac{(x_1-x_0)}{v_0} + t_0$ から求める。以後これを繰り返

し、trajectory の終端まで時刻を求める。これを、一般化すると現在地点 x_0, v_0, t_0 から始まる trajectory の n 番目の時刻は以下の式で表される。

$$t_n = \sum_{k=0}^{n} \frac{(x_n - x_{n-1})}{v_{n-1}} + t_0 \ (n \geq 1)$$

また、trajectory 全てをメッセージの中に書き込むと、1 フレームに収まらないため、本研究ではフラグメントが起こらないように、trajectory の点を 5 分の 1 に間引いた。

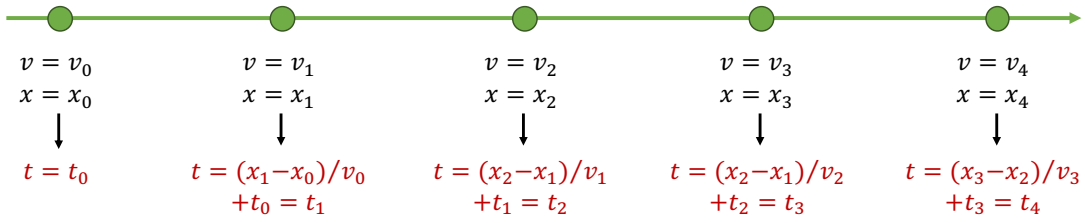


図 5.3: trajectory 変換の例

5.3.2　Intention

intention メッセージを生成する流れを図 5.4 に示す。メッセージ生成までの流れは以下の通りである。

1. advertisement のときと同様に、変換された時刻つき trajectory が Autoware から OpenC2X に定期的に送信され、OpenC2X 内の Message Manager で最新の trajectory を保存しておく。
2. DCC が他車両や RSU から advertisement を受信したとき、そのメッセージ部分を Message Manager に伝達する。
3. Message Manager は State Manager に advertisement を伝達すると同時に、advertisement の Station ID を保存しておく。
4. State Manager は State を確認し、State が Waiting であった場合、Message Manager に Intention を伝達するように指示を出す。
5. Message Manager は 3 で保存してある Station ID を Target Station ID に、最新の ego_planned_trajectory を Planned Trajectory にそれぞれ書き込み、Intention としてメッセージを生成し、DCC に伝達する。
6. DCC は受け取ったデータを GeoNet でカプセル化し、メッセージを配信する。

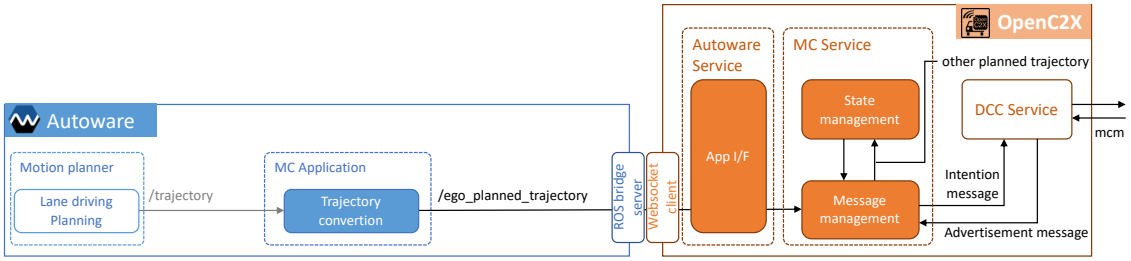


図 5.4: intention 生成までの実装と流れ

5.3.3 Prescription

prescription メッセージを生成する流れを図 5.5 に示す。メッセージ生成までの流れは以下の通りである。

1. advertising の状態で intention を受信していた場合、Message Manager が other planned trajectory を保存しておく。
2. State が Prescribing に遷移したとき、Message Manager に other planned trajectory が保存されていれば、State Manager が Message Manager から other planned trajectory のデータを読み取り、*/other_planned_trajectory* というトピックを publish する。
3. Collition Detector は*/other_planned_trajectory* を subscribe しており、トピックを受信した時、*/ego_planned_trajectory* と比較し、衝突検知を行う。
4. 衝突検知された車両の現在の位置を*/other_vehicles* というトピックに書き込み、publish する。
5. Lane Driving は、*/other_vehicles* というトピックを受信したとき、各車両の位置が車線変更先の車線内であるか検証し、その結果を*/other_vehicles/in_target_line* というトピックに書き込み、publish する。
6. Prescribed Trajectory Calculator は受信した*/other_vehicles/in_target_line* の中から、先頭の車両を見つける。
7. Prescribed Trajectory Calculator は先頭車両に対し、Prescribed trajectory を計算し、*/other_planned_trajectory* というトピックに書き込み、publish する。
8. Ros Bridge を経て、json 形式に変換されたデータを OpenC2X 側の Message Manager が受信後、車両ごとに Prescription を生成し、DCC に伝達する。
9. DCC は受け取ったデータを GeoNet でカプセル化し、メッセージを配信する。

ここで、衝突検知、特定の車線内の車両の発見、先頭車両の発見、Prescribed trajectory の計算、それぞれについて、以下で詳しく説明する。

■衝突検知

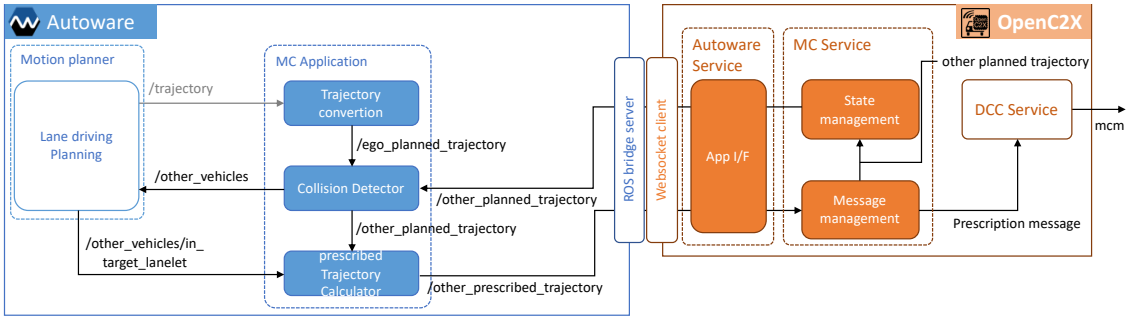


図 5.5: prescription 生成までの実装と流れ

各車両の Trajectory は一定間隔の時間ごとに 3 次元の位置情報が含まれているため、各時刻の車車間の距離を計算し、最も接近したときに、閾値以下であるかどうかで判断する。

■特定の車線内の車両の発見

衝突検知した車両の中には自車両の車線変更前の車線にいる車両も検知されてしまう。これを防ぐために、車線変更先の車線内に車両がいるかどうかを検知する仕組みが必要である。Autoware では Lanelet2 という 3 次元の地物情報を表すマップを内部で保持している。Lanelet2 には車線の情報も記されており、車線 ID を指定すると、その車線の地理情報がわかる。したがって、車線変更先の車線の地理情報も得られる。Lanelet2 の情報をもとに、車線を座標上の四角形で表し、車両の座標がその四角形の中にあるかどうかで判断できる。

■先頭車両の検出

以上の 2 つのプロセスによって、指示を出すべき車両が絞られるが、複数の車両がある可能性がある。本研究では、それらの車両の中で、先頭の車両のみに指示を出す。それより後ろの車両は、先頭車両の減速に合わせ、自律的に減速をする。したがって、先頭車両を発見する。

先頭車両の発見のためには、車車間の前後を認識する必要がある。車両の前後の計算例を図 5.6 に示す。まず、車両 B が車両 A に対して前か後ろか調べる場合について考える。図の例では、車両 A の向きと車両 A から車両 B への向きの角度 α が 0 以上 $\frac{\pi}{2}$ 未満であれば、車両 B のほうが前、$\frac{\pi}{2}$ 以上 π 未満であれば、車両 A のほうが前であるといえる。ここで、車両 A は自車両の向き、$\overrightarrow{d}$ を認識している。そして、各時刻の車両 A から車両 B へのベクトル $\overrightarrow{a}$ も Trajectory から計算することができる。したがって、2 つのベクトルの内積 $\overrightarrow{d} \cdot \overrightarrow{a}$ を取り、その正負を見ることで前後認識することができる。今回の例では、$\overrightarrow{d} \cdot \overrightarrow{a}$ の結果は正なので、車両 B は車両 A より前であることがわかる。同様に、車両 C についても、$\overrightarrow{d} \cdot \overrightarrow{a'}$ を取り、その結果が負であることから、車両 C が車両 A に対して後ろであることがわかる。これを繰り返すことでどの車両が先頭であるか発見することができる。

■Prescribed trajectory の計算

本研究では、Prescribed trajectory は Planned trajectory の速度のみを変更したものとし、安全な車間まで、相手の車両を相対的に下げることを目的とする。Planned trajectory と

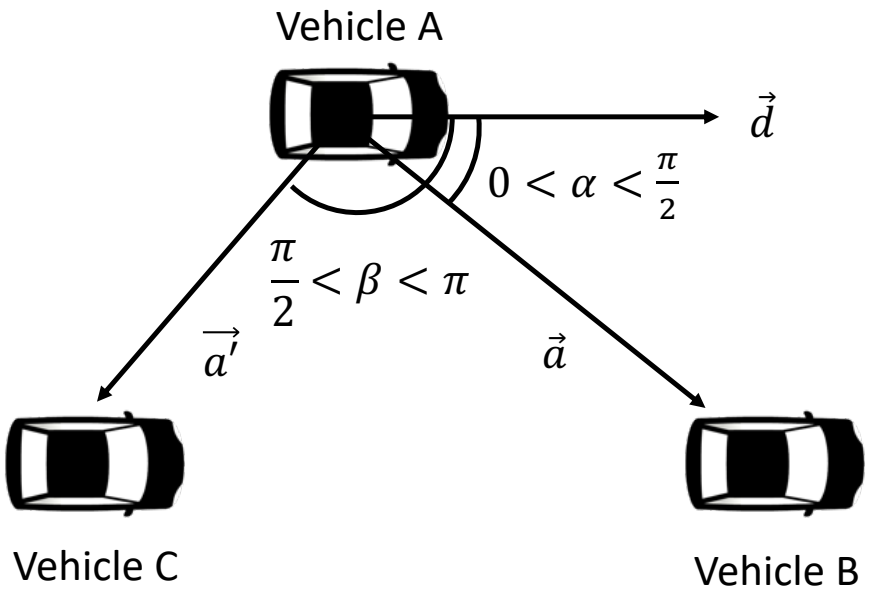


図 5.6: 車両の前後の認識例

Prescribed trajectory の関係を図 5.7 に示す。まず、Prescribed trajectory の計算に必要なパラメータは、車両を相対的に下げる距離 D、減速する速さ ΔV、減速するまでの時間 Δt_1 である。ここで、D は現在の車車間の符号付き距離 d と安全な車間 d_0 の和、$D = d_0 - d$ で表される。符号付き距離は、自車両よりも前であれば負、自車両よりも後ろであれば正である。また、ΔV と Δt_1 は変更可能なパラメータである。もとの速度に戻る時刻 Δt_2 は $\Delta t_2 = \Delta t_1 + T = \Delta t_1 + D/\Delta V = \Delta t_1 + \frac{(d+d_0)}{V}$ で表される。以上のパラメータによって Prescribed trajectory を求める。

ここで、生成される Prescribed trajectory は図の Reference velocity と書かれているものである。図の通り、速度が不連続であるが、受信した車両側で、現在の速度をフィードバック制御することで、速度が平滑化されるため、実行するときには Actual velocity と書かれたもののように、連続な値となる。

次に、先頭車両と自車両の最接近時の距離の計算について説明する。自車両と他車両の trajectory は事前に認識しているため、各時刻における車両の位置もわかる。したがって、各時刻における車車間の距離を求め、その中の最小を求める。更に、符号付き距離を求めるために、前述の車車間の前後を求め、自車両が相手車両より前であれば正、そうでなければ負の符号をつける。

以上で求めた、符号付き距離をもとに、Prescribed trajectory を求める。

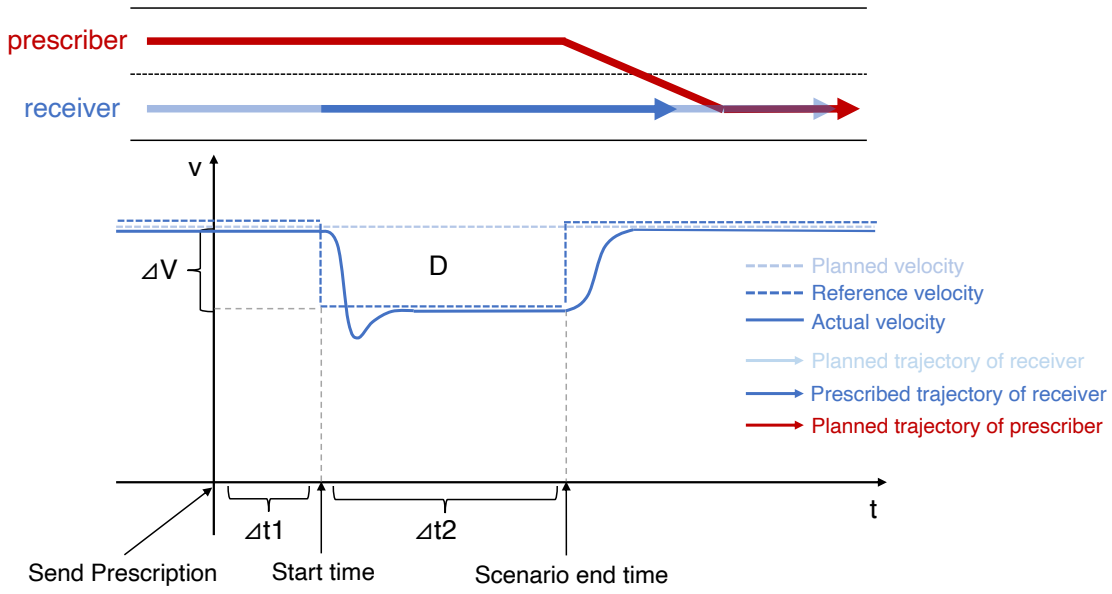


図 5.7: Planned Trajectory と Prescribed Trajectory

5.3.4　Acceptance と Prescribed trajectory の実行

acceptance メッセージの生成と Prescribed trajectory の実行までの流れを図 5.8 に示す。メッセージ生成、Prescribed trajectory の実行までの流れは以下の通りである。

1. DCC が他車両や RSU から prescription を受信したとき、そのメッセージ部分を Message Manager に伝達する。
2. Message Manager は State Manager に ego prescribed trajectory を伝達すると同時に、prescription の Station ID を保存しておく。
3. State Manager は State を確認し、State が Negotiating であった場合、ego prescribed trajectory を */ego_prescribed_trajectory* に書き込み publish する。
4. Verify Prescribed Trajectory は */ego_prescribed_trajectory* を subscribe しており、トピックを受信した際に、prescribed trajectory を実行しても安全であるか検証する。
5. 検証した結果、安全であれば、*/acceptance* というトピックに True を書き込み、publish する。一方、安全でなければ、*/acceptance* に False を書き込み publish する。
6. Message Manager は */acceptance* を subscribe しており、受信した際に、データが True であれば、State を Activating に、False であれば、Negotiating のままにしておく。また、*/acceptance* のデータを acceptance メッセージに書き込み、DCC に伝達する。
7. 検証した結果が安全であれば、Load Prescribed Trajectory が */ego_prescribed_trajectory* を実行に移す。

Prescribed Trajectory を実行する流れは以下の通りである。

1. */ego_prescribed_trajectory* から、各時刻における速度を求める。
2. trajectory の各点の時刻になった際に、その時刻の速度を*/max_velocity* というトピックに書き込み、publish する。
3. Lane Driving Planner は*/max_velocity* を subscribe しており、受信した際に、trajectory の各点の速度を*/max_velocity* にし、publish する。
4. Velocity Optimizer が Lane Driving Planner から受信した trajectory と、現在の速度や角速度を考慮し、速度の平滑化を行い、最終的な trajectory を publish する。
5. Fin メッセージが来るか、Prescribed trajectory の終端になったときに、*/max_velocity* の publish を終了し、State Manager は State を Waiting にする。

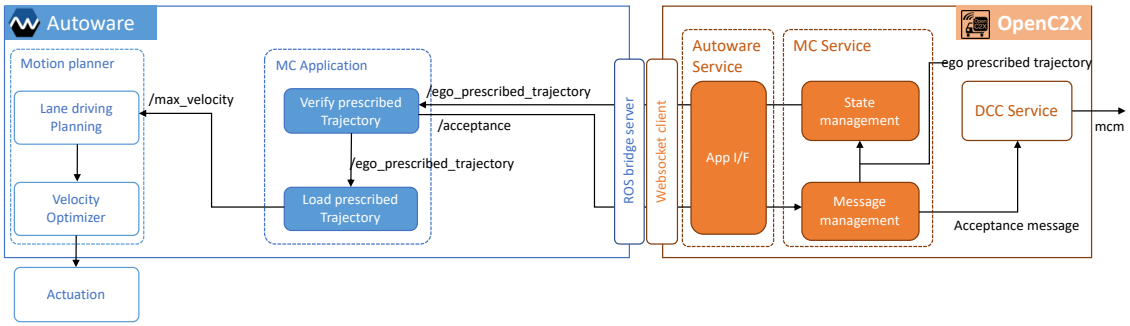


図 5.8: acceptance 生成までの実装と流れ

5.3.5 Fin

Fin メッセージを生成する流れを図 5.9 に示す。メッセージ生成までの流れは以下のとおりである。

1. Autoware の Turn Signal Decider というノードから、車両のウィンカーを意味する、*/turn_signal_cmd* というトピックを subscribe し、シナリオ実行中に、メッセージのデータが 0 であった場合、*/scenario_trigger_end* というトピックを publish する。
2. Ros Bridge を経て、json 形式に変換されたデータを OpenC2X 側の State Manager が受信した時、State が Activating であれば、State を Finishing にし、Message Manager に Fin を配信するように指示する。
3. Message Manager は State Manager からの指示を受け取ると、Fin メッセージを生成し、シリアライズ化した後、DCC にデータを渡す。
4. DCC は受け取ったデータを GeoNet でカプセル化し、メッセージを配信する。
5. DCC を介して Message Manager が Ack を受信した時、State Manager に伝達し、State を Waiting にする。

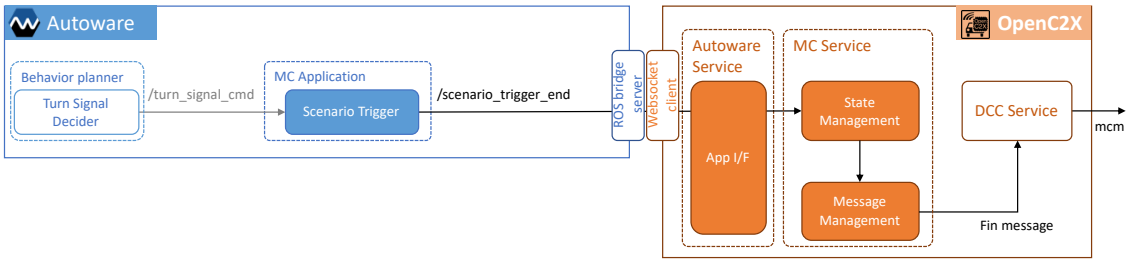


図 5.9: fin 生成までの実装と流れ

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 実験環境

最初に実験環境について述べる。

本研究で使用した 3 種類の PC4 つ（PC A × 2、PC B、 PC C）を表 6.1 に示す。なお、スペックの違いは意図的なものではなく、Autoware、OpenC2X が同一 PC 内で動作するような PC であれば良い。

表 6.1: 使用した PC のスペック

項目	PC A	PC B	PC C
CPU	Core i7-10875H 2.30GHz 8-core	Core i7-7700K 4.20GHz 8 core	Core i7-7700HQ 2.80GHz 8 core
GPU	NVIDIA GeForce RTX 2070 Super	NVIDIA GeForce GTX 1080	NVIDIA GeForce GTX 1070
RAM	64GB	32GB	
OS	Ubuntu 18.04.5 LTS	Ubuntu 18.04.3 LTS	
NVIDIA Driver	455.32.00		
CUDA	10.2		

実験に使用したネットワーク機器を表 6.2 に示す。

6.2 実験概要

本研究では、実験によって MCM の定量的な評価を行うことを目的としている。実際に走行させることができないため、Autoware に実装されている Planning simulator を用いて実

表 6.2: 実験に使用したネットワーク機器

役割	機器
V2X 用スイッチングハブ	NETGEAR GS108
スイッチングハブ (SW B)	BUFFALO WXR-1900DHP

験を行った。Planning simulator では、Planning の部分のみをシミュレートすることができる。したがって、Perception を加味した走行の実験を行うことができない。

そこで本実験では、MCM とは別に、Pub/Sub モデルによる、軽量な通信プロトコルである MQTT を使用して、擬似的に自車両の位置を他車両に伝達し、Perception の代わりとした。また、繰り返し行う実験の場合には、各 PC を事前に時刻同期をしておき、Autoware.IV を常時起動しておく。その上で、CRON によって、2 分に 1 回自動で AutoMCM、OpenC2X、実験用の python コード（MQTT クライアント、車両の初期位置、目的位置などを送信するメッセージを publish する ROS ノード）が立ち上がるようにした。

また、地図は西新宿にある直線を用いた。スタート位置、ゴール位置があり、その間に、車線変更を行う。スタート位置とゴール位置の距離は約 260m である。

インドア実験で行った測定は以下の 3 つである。

1. 走行テストと MCM の通信量
2. 各プロセスにかかった時間
3. MCM が車両の快適性に与える影響

6.2.1　実験環境構成

実験環境の構成を図 6.1 に示す。実験環境には Prescriber、Receiver A、Receiver B、Receiver C と SW A、SW B がある。まず、全ての PC に Autoware と OpenC2X をインストールし、それぞれ Ethernet で SW A と SW B に接続されている。各 PC は Ethernet ポートが 1 つであるため、もう一つのスイッチにはそれぞれ usb ethernet 変換アダプタを介して接続されている。図のとおり、SW A が V2X 用ネットワーク、すなわち CAM や MCM を通信するネットワークである。SW B は先述した MQTT や計測用のメッセージを通信するためのネットワークである。ただし、MQTT クライアントは python のライブラリである、paho-mqtt を使用した。また、Receiver B には Mosquitto（MQTT ブローカー）を起動してある。

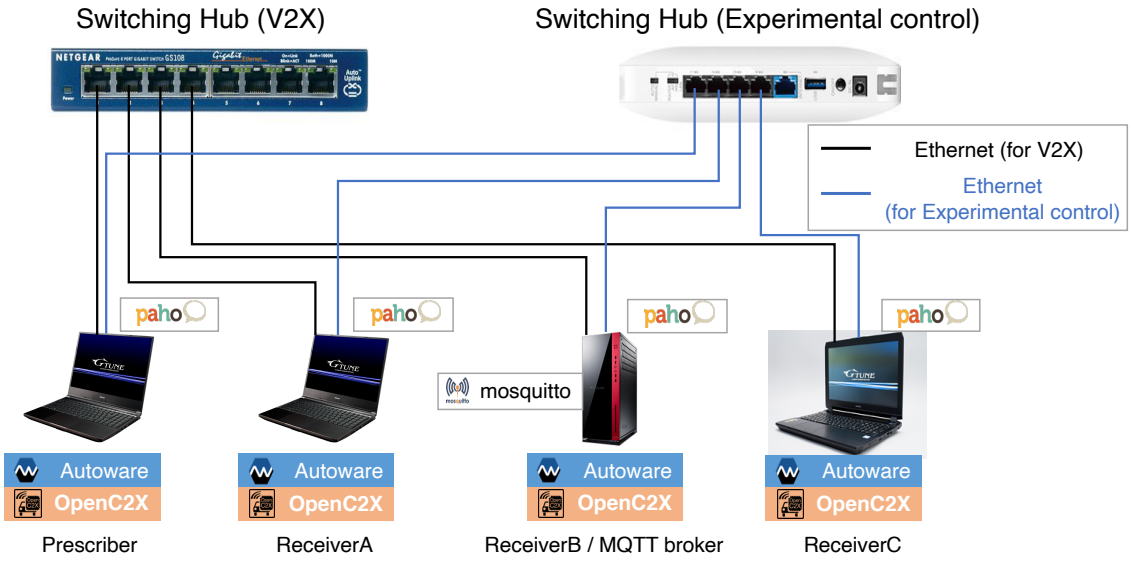


図 6.1: 実験環境の構成

6.2.2 実験環境変数

実験に関わる重要な変数を表 6.3 にまとめる。ただし、走行テストでは車両の速度を 20km/h としたため、Deceleration width は 10km/h とした。また、MCM のタイムアウトの時間は快適性に与える影響の実験でのみ 0s, 1.0s, 2.0s の 3 つのパラメータで実験を行い、その他の実験では 2.0s とした。

表 6.3: 実験環境変数

Type	Valiable name	Value
MCM	Time to timeout ($t_{timeout}$)	0s, 1.0s, 2.0s
	Time to resend (Δt_{resend})	0.1s
	Generation frequency (f)	10Hz
Prescribed trajectory	Prescribed vehicle distance ($D_{prescribed}$)	20m + d_0
	Deceleration width (ΔV)	20km/h
	Δt_1	$t_{timeout}$
	Δt_2	$D/\Delta V = 3.6$s

6.2.3 シミュレーションによる走行テストと MCM の通信量

ここでは、車線変更のシナリオで、提案手法が正しく動作するかの検証を行った。その様子を RViz という ROS の可視化ツールで可視化したものを図 6.2 に示す。図の通り、車線変更

を行う Prescriber と車線変更先を走行する Receiver A、B、C がいる。このシナリオにおいて、MCM をしようすることで、適切な車両に減速の指示を行い、スムーズな走行を行うことができるかの検証を行った。

図 6.3 に各時刻における車両の位置を示す。まず、Receiver A は Prescriber よりも前方に位置し、減速の指示を受けていないため、速度が一定であることがわかる。次に、Receiver B は Prescriber から指示を受け、減速していることがわかる。また、一度速度が Prescribed trajectory の速度である 10km/h よりも小さくなっていることがわかる。これは Autoware の速度平滑化の特性で、安全性の観点から trajetory の速度が現在の速度より小さい場合、一度目標の速度よりも速度を小さくし、その後速度を trajectory の速度に調節する。最後に、Receiver C は Receiver B の減速に合わせて減速している様子がわかる。Receiver C は前方車両の速度を検知するまでに少し遅延が生じるため、徐々に減速している。

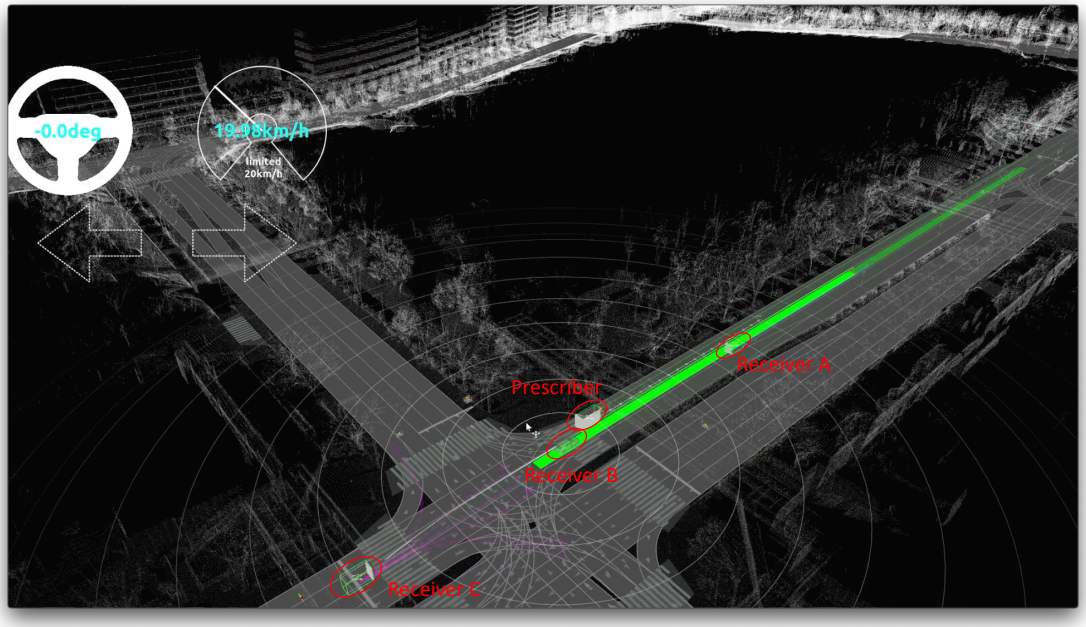


図 6.2: 実験したシナリオの RViz による可視化

次に、以上の実験を行っている際の 100ms ごとの通信量を計測した。実験では、MCM だけでなく標準メッセージである CAM を配信した。CAM、MCM の Advertisement は 10Hz であり、その他の MCM は全て一回で到達するものとした。図 6.4 が 2 台の調停での 100ms ごとの通信量、図 6.5 が 4 台の調停での 100ms ごとの通信量を表している。図より、trajectory が含まれる Intention と Prescription、Acceptance が CAM に対して 10 倍ほどの大きさであることがわかる。この結果から、Intention を全車両が常に配信することは帯域幅の圧迫に繋がるため、Event driven でメッセージを配信する本手法が帯域幅の節約に有効であることがわかる。また、図 6.5 において、Intention の総データ量が図 6.4 の 3 倍になっているのは、3 台が Intention を送信しているからである。一方、実際に指示を出すのは Receiver B のみで

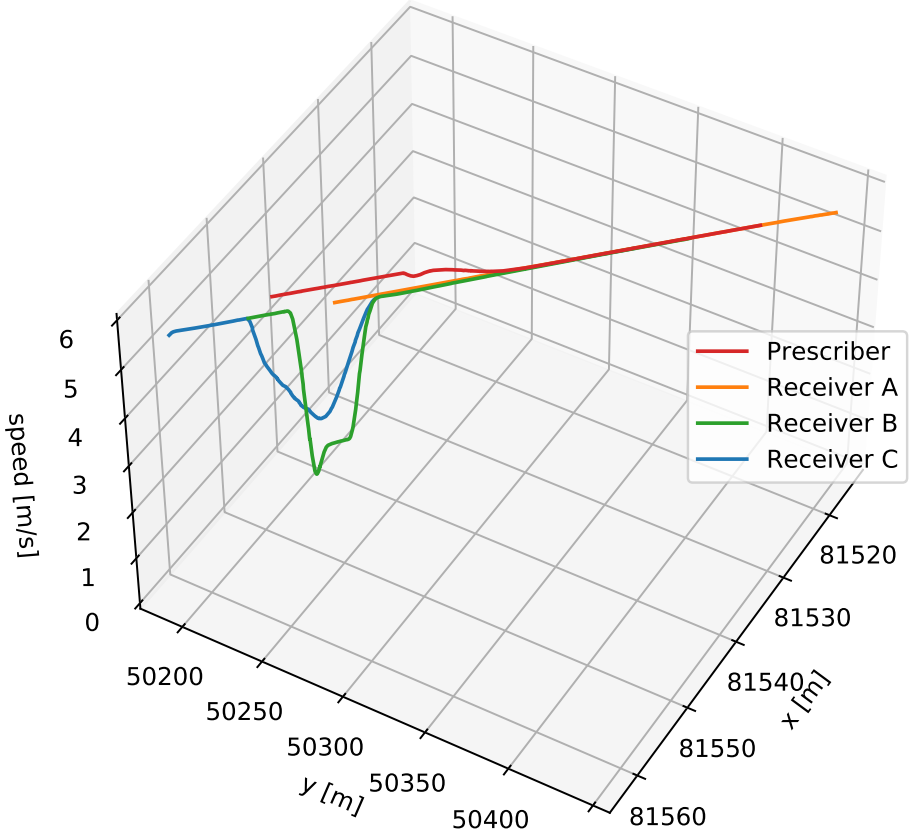


図 6.3: 各車両の位置と速度

あるため、Prescription、Acceptance の総データ量は図 6.4 と変わらないこともわかる。

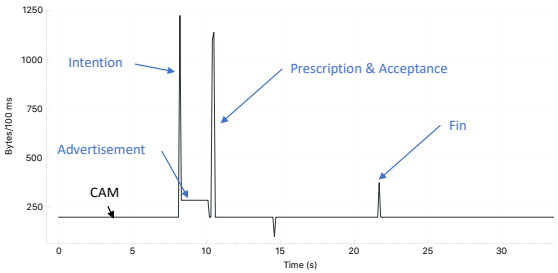


図 6.4: 2 台の調停での 100ms ごとの通信量

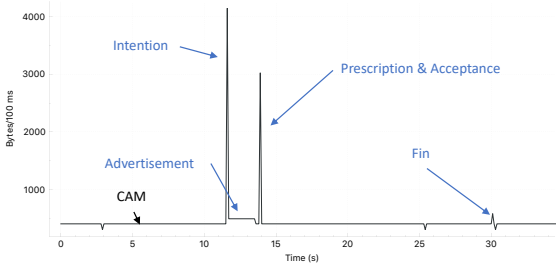


図 6.5: 4 台の調停での 100ms ごとの通信量

6.2.4 各プロセスにかかった時間

走行調停を実現する際のメッセージの流れを図 6.6 に示す。
図の通り、走行調停を実現するまでには、以下に示す 10 種類の遅延がある。

T1: Scenario trigger を検知してから Advertisement を生成するまでの時間

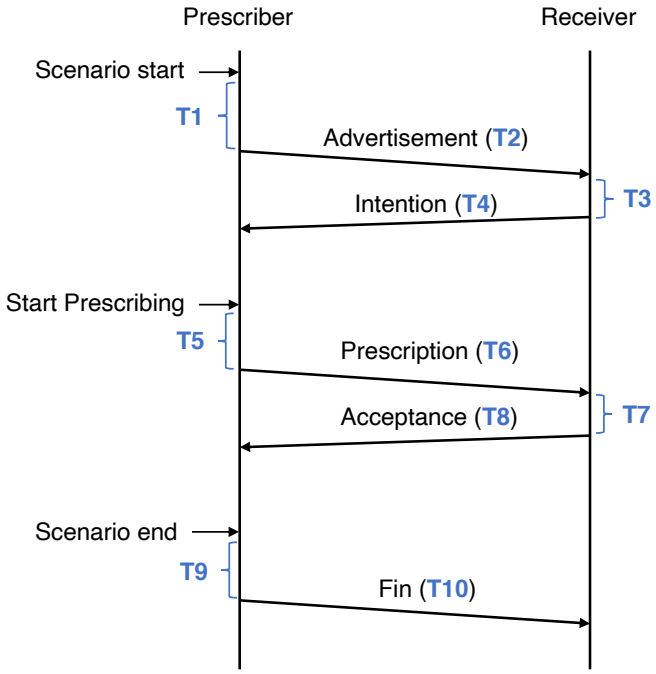


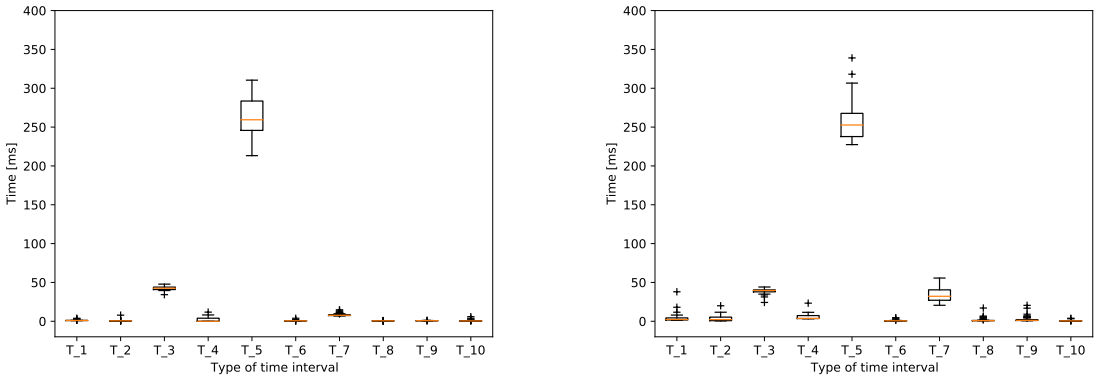
図 6.6: メッセージの流れとプロセス

T2:　Advertisement の通信時間
T3:　Advertisement を受信してから Intention を生成するまでの時間
T4:　Intention の通信時間
T5:　Prescription を生成する時間
T6:　Prescription の通信時間
T7:　Prescription を受信してから Acceptance を生成するまでの時間
T8:　Acceptance の通信時間
T9:　Scenario end trigger を検知してから Fin を生成するまでの時間
T10:　Fin の通信時間

各プロセスにかかった時間を計測するために、各イベントが起きたときに、MQTT で time メッセージを送信し、MQTT Broker の動いている PC でそのパケットをキャプチャする。各イベントの時刻の差分を取ることで、プロセスにかかった時間を計測することができる。この手法を使用する理由として、メッセージの通信時間などは 10ms を下回るため、PC 間の時刻の誤差が結果に影響するためである。

以上の手法を用いて、2 台の調停、4 台の調停で各プロセスにかかる時間を計測した。図 6.7 にその結果を示す。6.7a から、通信時間（T2、T4、T6、T8、T10）と Advertisement、Fin の生成時間（T1、T9）はおおよそ 1ms 以下であることがわかる。また、Acceptance の生成時間（T7）が次に短く、おおよそ 10ms 以下であり、Intention の生成時間（T3）はおおよそ 50ms 以下に収まっていることがわかる。一方で、Prescription の生成にかかる時間が約 250ms、最

大で 300ms を超えることから、他のプロセスにかかる時間よりはるかに大きいことがわかる。



(a) 2 台の調停　　(b) 4 台の調停

図 6.7: 各プロセスにかかる時間

一方で、2 台の場合と 4 台の場合で各プロセスにかかる時間はほとんど変わらないことがわかった。ここで、Prescription を生成するまでの時間にほとんど違いがない理由を考察するために、更に、2 台での調停で Prescription の生成までの各プロセスにかかる時間を計測した。Prescription 生成までの時間は以下の通りである。

T5_1: 状態が Prescripting に遷移してから、MC Application に ROS Bridge を介してトピックを送信する時間

T5_2: 衝突検知の計算時間

T5_3: 車線の確認にかかる時間

T5_4: Prescribed trajectory の生成時間

T5_5: MC Service に ROS Bridge を介してトピックが届き、メッセージを生成するまでの時間

ここでは 2 台での調停を計測したため、先頭車両の発見のプロセスは省略する。

各プロセス実行時間の結果を表 6.4 に示す。この結果から、ROS Bridge のデータの転送時間が遅延時間の 98% 以上を締めていることがわかる。

ここで、ROS Bridge のデータ転送の時間は車両台数に比例せず、衝突検知・車線の確認は車両台数に比例する。また、計測結果から、ROS Bridge のデータ転送にかかる時間と比べて衝突検知・車線の確認にかかる時間が少ないことから、台数が増えたとしても Prescription の生成時間に変わりがなかったということがわかった。従って、台数が増えたとしても遅延時間にほとんど影響がないため、走行調停が可能であることがわかる。

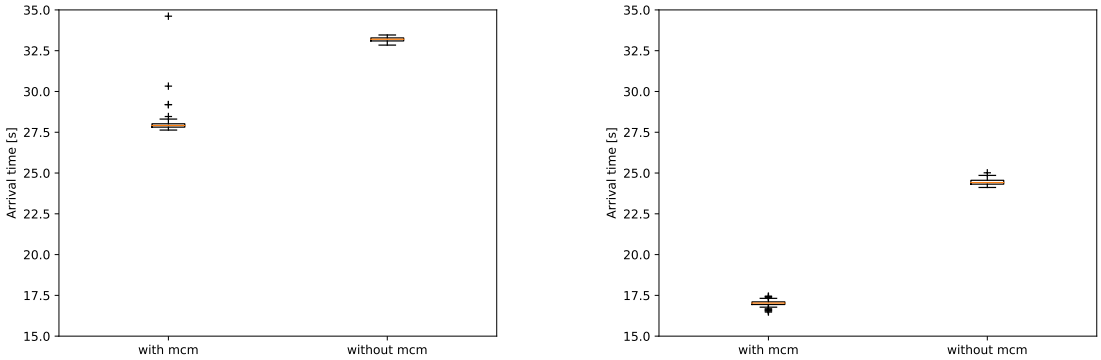
表 6.4: Prescription 生成までの各プロセスにかかる時間の割合

Type	T5_1	T5_2	T5_3	T5_4	T5_5
Delay ratio [%]	98.31	0.76	0.02	0.04	0.87

6.2.5　MCM が車両の快適性に与える影響

まず、MCM が与える快適性への影響を車両の速度が 30km/h のときと 50km/h のときの 2 つの場合について検証した。ここでは、快適性の指標として、車線変更を計画してから目標地点に到着するまでの時間を計測した。本実験では、2 台での調停を行い、測定した。

図 6.8 にその結果を示す。図より、30km/h、50km/h、どちらも MCM の使用により、到着時間が短くなっていることがわかる。MCM を使用しない場合は、車線変更の直前まで車線変更に気が付かない。従って、車線変更して初めて衝突検知を行い、急減速する。また、急減速した後に停止し、十分な車間距離を確保できると、再発進する。一方、MCM を使用する場合は、車線変更の前に十分な車間距離を取るために減速する。従って、車線変更のときには、ほとんど減速することなく走行する。この差が到着時間の差に影響している。

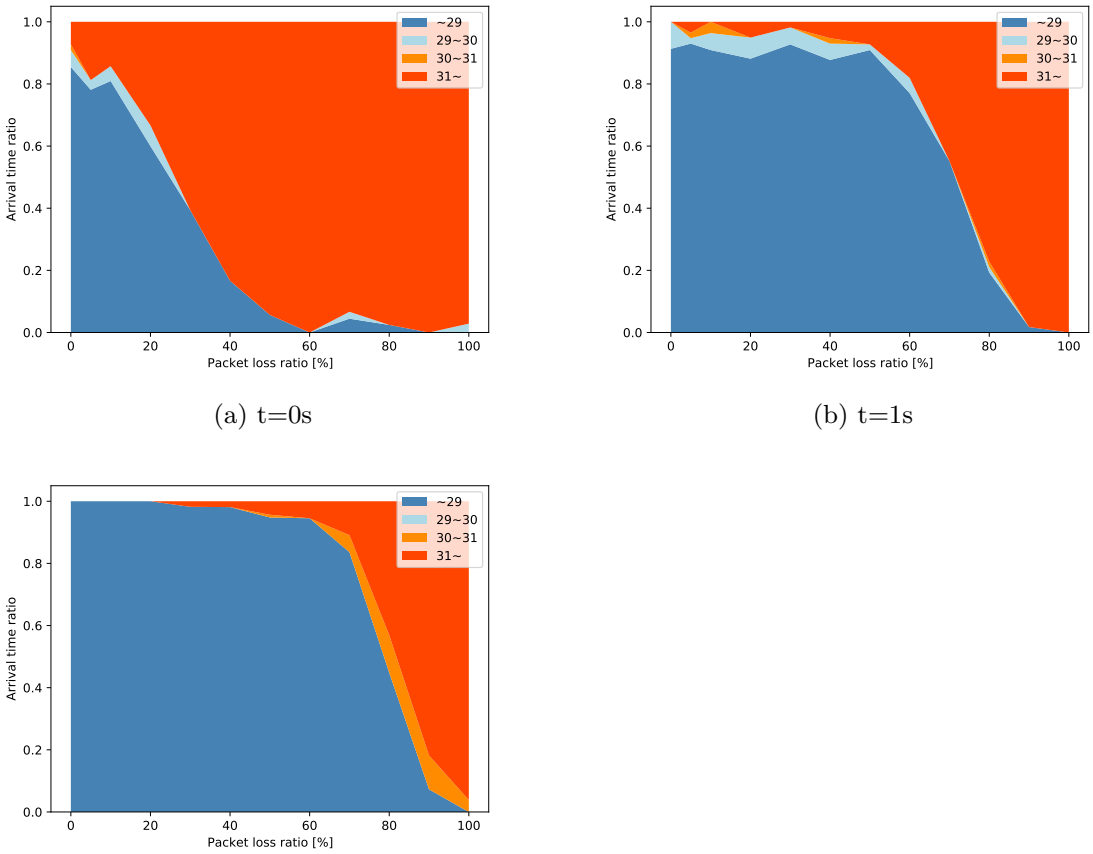


(a) 30km/h　　(b) 50km/h

図 6.8: 車線変更を計画してから Receiver が目標地点に到着するまでの時間

次に、パケットロス率が MCM に与える影響を検証した。ここでは、先述の快適性を評価した。実験では、2 台の車両を 30km/h で走行させ、車線変更による調停を行った。また、tc コマンドでパケットロス率を 0%~100% に変更し、Receiver の到着時間を測定した。ただし、各メッセージの再送の時間は全て 0.1s、最大再送時間を $t = 0\text{s}, 1\text{s}, 2\text{s}$ の場合について測定した。図 6.9 にその結果を示す。この図は、到着時間を 29 秒以下、29~30 秒、30~31 秒、31 秒以上に分類し、割合で示したものである。図より、$t = 0\text{s}$ のときは 10% から、$t = 1\text{s}$ のときは 60% から、$t = 2\text{s}$ のときは 70% から、急激に到着時間が長くなっていることがわかる。ここで、車線変更のシナリオは、見通しの良い直線路を想定しているため、V2X の間に壁など

の大きな障害物が入ることは考えられない。V2X の実践的な評価を行った研究 [44, 45] によると、壁などの障害物が入らないところでは、パケットロス率が 20% 以下に抑えられていることがわかっている。従って、$t = 1\text{s}, 2\text{s}$ ではパケットロスに対して一定の冗長性を持っていることがわかる。



(a) t=0s

(b) t=1s

(c) t=2s

図 6.9: パケットロス率を変化させたときの目標地点への到着時間

第 7 章

結論

7.1 まとめ

本研究では、自動運転における走行調停用メッセージである MCM の必要性を述べ、その要求事項を整理した。要求事項は安全性・汎用性・快適性・標準への準拠・帯域幅の節約・指示の重複・指示の連鎖に分類され、各要求事項を満たす MCM プロトコルの設計を提案した。具体的には、メッセージを 7 つの種類に分け、各ステーションは状態によって管理されようなプロトコル設計とした。また、汎用性を高めるために、様々なシナリオ・アプリケーションに共通する部分と固有な部分を MC Service と MC Application に分けた。MC Service では、Application とのデータのやり取り、状態管理、メッセージ管理を行い、MC Application ではシナリオトリガーや、trajectory の計算などを行う。

これらの手法を、自律型自動運転用ソフトウェアである Autoware と協調運転用ソフトウェアである OpenC2X を拡張することで、提案手法を実現した。

実験では、車線変更のシナリオでの調停を行った。実験では、走行テストと通信量の測定、各プロセスの実行時間の測定、MCM による快適性への影響の 3 つの測定を行った。まず 4 台での調停を行う実験で、4 台での調停を行うことができ、MCM のメッセージ量が CAM に対して非常に大きいため、提案手法の帯域幅節約が有効であることがわかった。次に、各プロセスの処理時間は、Prescription の生成プロセスに他のプロセスの 10 倍近くの時間がかかることがわかった。これは、ROS Bridge に依存する遅延であることがわかった。従って、調停に参加する車両が増えても、プロセスの実行時間に影響はないことがわかった。最後に、MCM を使用すると、5 秒から 7 秒到着時間が減少し、速度が早いほど、MCM の効果が大きいことがわかった。

7.2 今後の課題

本研究の提案手法は、様々なシナリオで使用できることを目的としている。従って、交差点での調停や、合流調停、歩行者の横断についても、アプリケーションを作成し、MCM の有効性を示す必要がある。今後の課題としては、これらのアプリケーションを追加で実装し、評価

を行うことである。

また、車線変更のシナリオについても、2 車線以外の調停については評価できていない。更に、2 車線でも、連続で車線変更する場合、どの程度の車間であれば、問題なく実現できるかの検証も必要である。

最後に、本研究ではシミュレータを使用した検証しか行うことができなかったため、2 台の実機を使用した検証を行うことで、さらなる評価が可能である。

発表文献と研究活動

(1) 水谷将也, 塚田学, 江崎浩, 飯田祐希, ”リアルタイムでの Point Cloud Data Map の配信による自動運転の支援とその評価”, マルチメディア，分散，協調とモバイル（DICOMO2020）シンポジウム, 高知県安芸郡, 2020.

(2) Masaya Mizutani, Manabu Tsukada, Yuki Iida, Hiroshi Esaki, ”3D maps distribution of self-driving vehicles using roadside edges”, 13th International Workshop on Autonomous Self-Organizing Networks (ASON) held in conjunction with CANDAR 2020, Okinawa, Japan, 2020.

参考文献

[1] The grand challenge. https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles. (Accessed on 01/24/2021).

[2] つくばチャレンジ 2020. https://tsukubachallenge.jp/2020/. (Accessed on 01/24/2021).

[3] 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.

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

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

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

[7] 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).

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

[9] Autowarearchitectureproposal/overview.md at master · tier4/autowarearchitectureproposal · github. https://github.com/tier4/AutowareArchitectureProposal/blob/master/design/Overview.md. (Accessed on 01/08/2021).

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

[11] Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Part 2: Specification of Cooperative Awareness Basic Service, December 2014. ETSI EN 302 637-2 V1.3.2 (2014-11).

[12] Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Analysis of the Collective Perception Service (CPS);, December 2019. ETSI

TR 103 562 V2.1.1 (2019-12).

[13] Intelligent Transport Systems (ITS); Vehicular Communications; Basic Set of Applications; Local Dynamic Map (LDM), September 2014. ETSI EN 302 895 V1.1.1 (2014-09).

[14] Intelligent Transport Systems (ITS); Communications Architecture, September 2010. ETSI EN 302 665 V1.1.1 (2010-09).

[15] Intelligent Transport Systems (ITS); Users and applications requirements; Part 2: Applications and facilities layer common data dictionary, September 2014. ETSI TS 102 894-2 V1.2.1 (2014-09).

[16] Connecting europe facility — innovation and networks executive agency. `https://ec.europa.eu/inea/en/connecting-europe-facility`. (Accessed on 01/08/2021).

[17] Intelligent Transport System (ITS); Vulnerable Road Users (VRU) awareness; Part 1: Use Cases definition;, September 2019. ETSI TR 103 300-1 V2.1.1 (2019-9).

[18] Openc2x - distributed embedded systems. `https://www.ccs-labs.org/software/openc2x/`. (Accessed on 01/11/2021).

[19] 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.

[20] Cristofer Englund, Lei Chen, Jeroen Ploeg, Elham Semsar-Kazerooni, Alexey Voronov, Hoai Hoang Bengtsson, and Jonas Didoff. The grand cooperative driving challenge 2016: boosting the introduction of cooperative automated vehicles. *IEEE Wireless Communications*, Vol. 23, No. 4, pp. 146–152, 2016.

[21] O Sawade, M Schulze, and I Radusch. Robust communication for cooperative driving maneuvers. *IEEE Intell. Transp. Syst. Mag.*, Vol. 10, No. 3, pp. 159–169, 2018.

[22] Simulation of vehicle-2-x communication - dcaiti - tu berlin. `https://www.dcaiti.tu-berlin.de/research/simulation/`. (Accessed on 01/11/2021).

[23] V Milanes, J Godoy, J Villagra, and J Perez. Automated On-Ramp merging system for congested traffic situations. *IEEE Trans. Intell. Transp. Syst.*, Vol. 12, No. 2, pp. 500–508, June 2011.

[24] Atsuya Uno, Takeshi Sakaguchi, and Sadayuki Tsugawa. A merging control algorithm based on inter-vehicle communication. In *Proceedings 199 IEEE/IEEJ/JSAI International Conference on Intelligent Transportation Systems (Cat. No. 99TH8383)*, pp. 783–787. IEEE, 1999.

[25] L Wang, R F Iida, and A M Wyglinski. Coordinated lane changing using V2V communications. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pp. 1–5, August 2018.

[26] Lucas Eiermann, Oliver Sawade, Sebastian Bunk, Gabi Breuel, and Ilja Radusch. Co-

operative automated lane merge with role-based negotiation. *2020 IEEE Intelligent Vehicles Symposium (IV)*, October 2020.

[27] Daniel Heb, Ray Lattarulo, Joshue Perez, Julian Schindler, Tobias Hesse, and Frank Koster. Fast maneuver planning for cooperative automated vehicles. *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[28] Matthias Nichting, Daniel Hess, Julian Schindler, Tobias Hesse, and Frank Koster. Space time reservation procedure (STRP) for V2X-Based maneuver coordination of cooperative automated vehicles in diverse conflict scenarios. *2020 IEEE Intelligent Vehicles Symposium (IV)*, October 2020.

[29] Julian Schindler, Robert Markowski, Daniel Heß, Daniel Wesemeyer, Clarissa Böker, Baldomero Coll Perales, Miguel Sepulcre, Javier Gozalvez, Michele Rondinone, and Dominik Matheis. TransAID deliverable 7.2: System prototype demonstration (iteration 1). 2019.

[30] Alejandro Correa, Robert Alms, Javier Gozalvez, Miguel Sepulcre, Michele Rondinone, Robbin Blokpoel, Leonhard Lücken, and Gokulnath Thandavarayan. Infrastructure support for cooperative maneuvers in connected and automated driving. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 20–25. IEEE, 2019.

[31] Alejandro Correa, Franz Andert, Robbin Blokpoel, Nicolai Wojke, Gokulnath Thandavarayan, Baldomero Coll Perales, Miguel Sepulcre, Javier Gozalvez, Anton Wijbenga, Julian Schindler, et al. Transaid deliverable 5.2: V2x-based cooperative sensing and driving in transition areas. 2019.

[32] Richard Jacob, Matthias Gay, Markus Dod, Sven Lorenz, Alexander Jungmann, Lars Franke, Michael Philipp, Michael Kloeppel-Gersdorf, Mathias Haberjahn, Erik Gruschka, and Gerhard Fettweis. IVS-KOM: A reference platform for heterogeneous ITS communications.

[33] Rico Auerswald, Roman Busse, Markus Dod, Richard Fritzsche, Alexander Jungmann, Michael Klöppel-Gersdorf, Josef F Krems, Sven Lorenz, Franziska Schmalfuß, Sabine Springer, et al. Cooperative driving in mixed traffic with heterogeneous communications and cloud infrastructure. In *VEHITS*, pp. 95–105, 2019.

[34] Bernhard Hafner, Josef Jiru, Karsten Roscher, Jorg Ott, Georg A Schmitt, and Yagmur Sevilmis. CVIP: A protocol for complex interactions among connected vehicles. *2020 IEEE Intelligent Vehicles Symposium (IV)*, October 2020.

[35] Charles Wartnaby Daniele Bellan. Decentralized cooperative collision avoidance for automated vehicles: a real-world implementation. *2020 IEEE Intelligent Vehicles Symposium (IV)*, October 2020.

[36] B Lehmann, H Günther, and L Wolf. A generic approach towards maneuver coordination for automated vehicles. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pp. 3333–3339, November 2018.

[37] Eclipse sumo - simulation of urban mobility. https://www.eclipse.org/sumo/. (Accessed on 01/11/2021).

[38] Omnet++ discrete event simulator. https://omnetpp.org/. (Accessed on 01/11/2021).

[39] Ros 2 overview. https://index.ros.org/doc/ros2/. (Accessed on 01/10/2021).

[40] Autoware.auto. https://www.autoware.auto/. (Accessed on 01/10/2021).

[41] florianklingler/openc2x-standalone: Open source experimental and prototyping platform supporting etsi its-g5. https://github.com/florianklingler/OpenC2X-standalone. (Accessed on 01/11/2021).

[42] 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 01/11/2021).

[43] ja/rosbridge - ros wiki. http://wiki.ros.org/ja/rosbridge. (Accessed on 12/01/2020).

[44] Manabu Tsukada, Takaharu Oi, Akihide Ito, Mai Hirata, and Hiroshi Esaki. AutoC2X: Open-source software to realize V2X cooperative perception among autonomous vehicles. In *The 2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, Victoria, B.C., Canada, 2020.

[45] Manabu Tsukada, José Santa, Satoshi Matsuura, Thierry Ernst, and Kazutoshi Fujikawa. On the experimental evaluation of vehicular networks: Issues, requirements and methodology applied to a real use case. *arXiv preprint arXiv:1504.04426*, 2015.

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。

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大井貴晴氏、グループの同期として励まし合った平田真唯氏、増田英孝氏、辻尾康平氏、廣瀬稔彦氏、伊藤彰秀氏、有井慎平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、励まし合ったソンイクイ氏、加藤慎氏、徳備彩人氏、小林日向氏、政宗達矢氏、オンカンチャナポラパット氏、上田草馬氏、張智晴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。