「マルチメディア，分散，協調とモバイル
(DICOMO2021) シンポジウム」 令和3年6月-7月

自動運転を支援する時空間予測を用いた路側の協調経路計画システム

辻尾 康平[1] 平田 真唯[1] 奥村 圭祐[2] 田村 康将[2] 塚田 学[1] デファゴ クサヴィエ[2]

概要：近年，自動運転技術が発展する中で，協調型 ITS(Intelligent Transportation Systems) が注目されている．協調型 ITS は，自動運転車が周囲の自動運転車や路側機（RSU）と通信するために用いられる．このシステムを通して，自動運転車は自車のセンサで感知できない，自車の死角にいる障害物などの情報を得ることができる．ところが，現状の協調型 ITS では主にリアルタイムの情報は共有しているが，車両の行動計画などの未来のデータを共有していない．そのため，周囲の車両の車線変更を伴うシナリオや，交差点のシナリオなどの，走行経路が衝突する状況では，車載センサの情報に基づく経路計画が非効率になるという問題がある．そこで本研究では，車両の未来の行動計画/経路を定義し，複数の車両の未来の経路情報に基づく時空間予測を用いて，交差点における協調的な経路計画モデルを提案する．RSU は，共有された未来の経路情報を用いて，車両経路の衝突の可能性や加速の余地がある場合に，協調的に車両の速度を調整し経路の再生成を行う．提案手法を自律型自動運転のためのオープンソースソフトウェアである Autoware を用いて実装し，LGSVL シミュレータを用いて評価した．見通しの悪い交差点のシナリオで 2 台の車を用いて実験を行ったところ，それぞれの車が他車の行動計画を反映した経路計画を行うことで，安全に走行を行うことができることがわかった．さらに，路側機を導入することで，通信を行わない場合よりも，交差点を含む経路を，2 台の車がそれぞれ 23.0% と 28.1%，短い時間で通過することができた．

Kohei Tsujio[1] Mai Hirata[1] Keisuke Okumura[2] Yasumasa Tamura[2] Manabu Tsukada[1] Xavier Défago[2]

1. はじめに

近年，交通に関する様々なデータを用いて，交通事故や渋滞などの問題を解決したり，効率的な交通を実現する協調型 ITS(Intelligent Transportation Systems) が注目を集めている．車両に搭載されたセンサから得られる情報に基づいて実現される自動運転技術も協調型 ITS の一部である．多くの自動車関連企業は自動ブレーキやレーンキープアシスト，自動駐車機能といった技術の商用化を進めており [1,2]，今後もより安全で効率的な交通を実現するためのさらなる機能やサービスの導入が見込まれている．このような状況の中，車両や歩行者など交通参加者のデータを用いて様々な問題を解決する協調型 ITS はますます重要性が高まっていくと言える．代表的な協調型 ITS のアーキテクチャとして，欧州電気通信標準化機構 (ETSI) と国際標準化機構 (ISO) が定めるステーションインフラストラクチャがある [3,4]．この中で標準化の議論と実用化が進むメッセージ形式として CAM(Cooperative Awareness Message) が挙げられる．CAM は自車両に関する位置情報，速度，進行方向などを含んだリアルタイムな情報を共有するための車々間通信メッセージである．さらにその発展形として CPM(Collective Perception Message) も標準化の議論が進んでいる [5,6]．これは自車両のみならず，車両に搭載されたセンサで検出した周辺物体に関する情報を共有するためのメッセージである．これらのメッセージを用いることで車々間通信によって交通参加者は周辺認識能力を向上させ，より安全な交通流を実現することができる．

しかしながら，現在の協調型 ITS ではリアルタイム情報は共有されているものの，車両の次の行動などに関する未来に関するデータが共有されていないという課題が存在している．そのため，周辺車両が車線変更したり，交差点で右左折を行うようなシナリオでは，その周辺認識はリアルタイム情報にのみ基づいて行われ，実際に動き出してからしか行動の把握ができないという限定的なものになっている．そこで本研究では，交差点での時空間予測を用いた路側での協調経路計画システムの提案と，自動運転ソフトウェアとドライビングシミュレータ上での提案手法の実装及び評価を行う．自動運転車が，未来の経路情報を路側機 (RSU: RoadSide Unit) と共有し，RSU において複数車両の経路情報に基づく調停を行うことで，衝突を避けた効率的な経路計画を行う．

[1] 東京大学大学院情報理工学系研究科
[2] 東京工業大学 情報理工学院

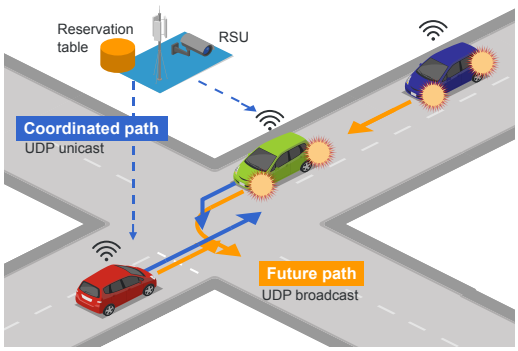


図 1　時空間予測に基づく協調経路計画

提案手法は，大きく分けて

- **時空間予測情報の交換**: 車両の未来の経路情報の交換
- **協調計画**: RSU における，未来の経路情報を格納する予約テーブルを利用した車両の調停アルゴリズム

の 2 つの段階に分けられる．提案手法の実装は，自動運転のためのオープンソースソフトウェアである Autoware [7] を用いた．Autoware を用いて協調計画を実装したのは本研究が初である．

図 1 は本研究の概念図である．自動運転車が，未来に走行する予定の経路である *future path* を定期的に周囲にブロードキャストする．future path を受け取った車両は，それを自車の物体認識の入力と統合し，未来の障害物情報として利用する．さらに RSU は，複数の車両の経路情報を集約し，予約テーブルに格納していく．このとき，ある経路が他車両の経路と重なることがあれば，衝突と判定して RSU はその衝突を回避するような経路を再生成して *coordinated path* として当該車両に送り返す．

本論文の構成は以下の通りである．2 章で関連研究を紹介し，3 章で提案手法について説明する．4 章で提案手法の実装の詳細を述べた後，5 章で実験概要とその結果を説明する．最後に 6 章で本稿のまとめと課題について述べる．

2. 関連研究

2.1 CAM の拡張

Renzler ら [8] は，標準化メッセージである CAM を拡張して future path を格納することで，未来の経路を共有することを提案した．隊列走行を行っている際の緊急時の対応のために使用され，周辺車両に危険をもたらすような回避行動を減らすことができることを示した．

2.2 センサデータの共有

ITS メッセージを含むセンサデータの共有技術は low, feature, track の 3 つのレベルに分けることができる [9].

- **low レベル**: 前処理のされていない生データを共有する．
- **feature レベル**: 前処理として生データから特定の特徴を抽出した後，その特徴を共有する．
- **track レベル**: それぞれのセンサが独立にトラッキングアルゴリズムを実行し，障害物を生成した後，そのリストを共有する．

このうち，track レベルのセンサデータ共有は通信帯域を圧迫しないという利点がある．以下 track レベルの共有を行っている関連研究を紹介する．Gabb ら [10] は，モバイルエッジコンピューティング (MEC) サーバから車に送る際のデータスキームを示し，さらに車載センサと路側の MEC サーバが生成した環境モデルのデータ融合に関するシステムも提案した．Tsukada ら [11, 12] は Autoware と，協調型 ITS のためのオープンソースの実験とプロトタイプ用のプラットフォームである OpenC2X を用いて，路側機で検出した物体情報を自動運転する車両と共有する AutoC2X を提案した．

2.3 交差点管理

交差点における自動運転車の管理に関する既存研究は分散型アプローチと集中型アプローチに分けられる．

2.3.1 分散型アプローチ

Azimi ら [13] は交差点をグリッドとしてモデル化する協調的な交差点管理アルゴリズムを提案した．それぞれのグリッド内のセルは一意な識別子が割り振られ，現在と未来の位置に関する識別子を各車両が共有し，状況に応じて減速や停車をすることで交差点のスループットを 87.82%上げた．Aoki ら [14] は，交通事故などを引き起こす可能性のある動的な交差点において，センサベースの認識情報と車々間通信を用いてどのように自動運転者が走行するかを検討した．

2.3.2 集中型アプローチ

Liu ら [15] は，軌道計画に基づく交差点管理である，TP-AIM (a trajectory planning based autonomous intersection management mechanism) を提案した．車両は，管理者に関連するコンフリクトゾーンの占有時間の共有を要求する．管理者は車両をサービスグループに分けて，車両の種類，先頭車両，交差点までの距離によって決まる優先順位を与える．車両は前のグループ内にいる，自分と競合する関連車両が軌道計画プロセスを終了した場合に，コンフリクトゾーンの占有時間情報を取得する．この情報を得た車両は，窓探索アルゴリズムを利用して，軌道を計画する．軌道計画が成功すると，車両は管理者にメッセージを送信する．この手法を，MATLAB/Simulink と SUMO を用いて評価した．シミュレーションの結果，提案する TP-AIM メカニズムは，平均退避時間を大幅に短縮し，スループットをそれぞれ 60 秒以上，20% 以上向上させることを証明した．

Bashiri ら [16] は協調的な交差点管理問題に対する小隊ベースのアプローチを提案した．小隊リーダーが小隊全体

を代表してインフラストラクチャと通信することで，通信オーバーヘッドを削減することができる．シミュレータで実験した結果，PVM 法を用いることで，停止標識政策と比較して平均遅延を最大 40% 削減し，分散を 50% 削減した．

以上のように既存研究では，未来の車両の経路情報の交換手法や交差点における車両管理の研究が行われているものの，経路交換はまだ発展途上の分野であることに加え，実際の自動運転ソフトウェア上での実装や，それを用いた実験等まで行っている研究はごく僅かである．そこで本研究では，未来の経路交換手法と交差点における車両管理手法を提案し，それらを自動運転用のソフトウェアとドライビングシミュレータ上で実装し，評価を行った．

3. 時空間予測に基づく協調計画

この章では future path の共有手法と，coordinated path の生成手法に関して詳しく述べる．

3.1 future path の共有

以下に，future path の共有に関する説明で用いる主要な用語を紹介する．

- **trajectory**: 通る予定地点の情報と，その地点を通るときの速度情報の対が，連続的に並んだデータ構造を持つ．Autoware などを搭載した自動運転車は，一般的にこの trajectory に従って走行する．
- **future path**: 通る予定地点の情報と，その地点を通るときの時刻情報の対が，連続的に並んだデータ構造を持つ．現在時刻を t_0 とし，それぞれの地点を通過する時刻を trajectory から計算することで，生成可能である．future path は，車や路側機が衝突の検知をしやすいデータ構造である．future path は式 (1) を用いて trajectory から生成できる．

$$t_n = \sum_{k=0}^{n} \frac{x_k - x_{k-1}}{v_k} + t_0 \quad (n \geq 1) \qquad (1)$$

図 2 は一般的な自動運転システムの概要を表している．まず，perception で周囲をセンシングした結果を入力として物体検出などの処理を行う．その後，planning では前段の物体情報などを考慮して trajectory を生成し，それを control へと送る．そして control では，trajectory をもとにアクセルやブレーキなどの行動を実行する．本研究では，まず planning における出力である trajectory を，式 (1) を用いて future path に変換する．そして，生成された future path をブロードキャストする．このとき future path には ID，現在位置，速度，車両の形の情報も付与されている．

future path を受け取った車両はそれを動的な障害物として自車の perception の出力結果と統合する．この統合により未来の新たな動的障害物に関する情報が増えることになるが，場合によっては意図しない車両の減速などにつながる可能性もある．その問題を，次の章で詳しく説明する RSU が生成する coordinated path を用いて解決する．

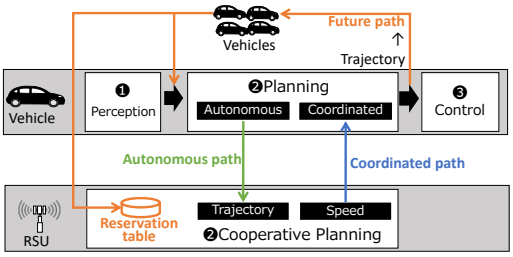


図 2 提案手法で用いる path の流れと予約テーブル

3.2 Coordinated path の生成

Coordinated path の生成に関する説明で用いる用語を紹介する．

- **予約テーブル**: 自動運転車から送られてくる future path を格納する RSU のデータベースである．予約テーブルを用いて RSU は車両間の衝突の検知や加減速などの可否を判定する．
- **autonomous path**: 自動運転車によって計算される目標地点までの経路である．coordinated mode の場合，車両は定期的に autonomous path を RSU にユニキャストで送信する．autonomous path は他車両のそれと衝突する場合もある．
- **coordinated path**: RSU によって速度調節された調停結果として，衝突が無く交通流を向上させるような新たな経路である．RSU は定期的にこの経路を車両にユニキャストし，車両はこの経路を優先的に使用する．

本手法では車両側で経路を生成し，RSU によってその速度調節をすることで再計画を行う．図 2 にあるように，RSU は車両の future path を受け取り，予約テーブルを更新する．このときもし衝突の可能性があると判定されれば，RSU は車両に対して，coordinated mode に移行するように指示するシグナルを送信する．RSU は衝突の可能性のある future path の速度を落とし，衝突が解消されるか，最悪の場合車両が停止するまで減速を行う．

一方，予約テーブルを参照し，衝突の可能性がない場合には，RSU は加速が可能かの判定を行う．もし可能であれば経路の速度情報を更新して加速された経路を生成する．

4. 実装

提案手法を Autoware.IV (v.0.6.0) を用いて実装した．この章では Autoware に関する説明をした後，提案手法の実装の詳細を述べる．

4.1 Autoware

Autoware は自動運転のためのオープンソースソフトウェ

アである [7]．ロボット用のミドルウェアである ROS を用いて実装されている [17]．ROS は Linux OS 上で動作し，複数のノードがトピックと呼ばれる論理チャネルを通してメッセージをやり取りすることで，複雑な処理を行うことができる．また RViz と呼ばれる 3D 描画ツールも搭載しており，ロボットの動く様子を確認する事ができる．

Autoware は自動運転に必要な localization, perception, planning, control という一連の機能を提供している．3D マップ，LiDAR，カメラ，GNSS などを使いながら localization と perception が行われる．perception が終わると検出された物体の情報が，planning モジュールへ渡される．planning は大きく分けて mission planning, scenario selection, behavior planning の 3 段階の処理からなる．mission planner は目的地点までの大域的な経路を静的な地図情報をもとに計算し，生成する．そして scenario selector が走行しているシナリオに応じてどの behavior planner を適用するべきか判定する．現状では通常の路上走行用と駐停車用の二種類の behavior planner が実装されている．そして最後に選択された planner が trajectory を計算し，それを加減速やステアリングを行う control モジュールに送る．

4.2 実装の概要

図 3 が実装の概要図である．白いボックスが Autoware として実装されている部分であり，青いボックスが今回提案手法で実装された部分である．

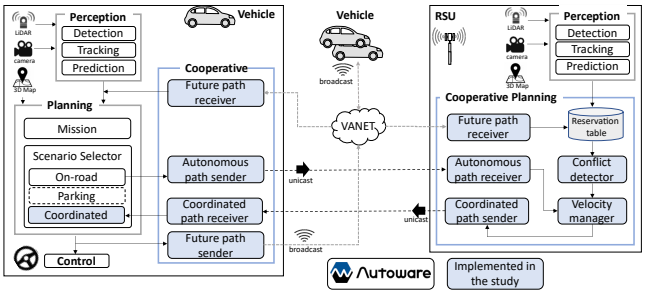


図 3 実装の概要

実装にあたり，路上走行用の on-road planner と駐停車用の parking planner に加えて，新たに coordinated planner の ROS ノードを追加した．scenario selector は後述するモードに応じて on-road と coordinated の planner を切り替える．planning module は trajectory を /planning/scenario_planning/trajectoryのトピックに送信する．**future-path sender** は trajectory を future path に変換し，UDP 通信で 10Hz の頻度でブロードキャストする．車両は，UDP 通信の最大パケットサイズによる制限から，最大で 150 ポイントの trajectory を送信する．trajectory の各ポイント間の最小距離は 0.1 m である．future path は JSON(JavaScript Object Notation) のフォーマット [18] で記述されている．

車両と RSU は future path を受け取り，**future-path receiver** が受け取った path を/perception/object_recognition/objects のトピックに送信する．これにより，perception モジュールが周囲の検出した物体を planning モジュールに送信したものと統合することができる．

RSU が受け取った path の情報から衝突が無いと判定できた場合は，当該 future path は予約テーブルに格納される．衝突がある場合には RSU は車両に対して coordinated mode に移行するよう指示するシグナルを送る．車両はその後, **autonomous path sender** から RSU に対して 100 ms 間隔のユニキャストで autonomous path を送信する．RSU は autonomous path receiver から autonomous path を受け取ると，velocity manager がその path の速度調整を行って coordinated path を生成する．velocity manager は衝突ある場合には，車両の速度を落とすように path を修正する．逆に加速の余地がある場合には車両の速度を上げるように path を修正する．車両は RSU から coordinated path を受け取ったら，coordinated モードで走行する．

RSU は Autoware を搭載しており，perception モジュールを利用することで交差点付近の，自動運転車以外の車両，自転車，歩行者などの動的な物体を検出する．そして /perception/object_recognition/objectsから配信されている検出物体の予測経路を予約テーブルに格納する．RSU はその予測経路も考慮に入れて衝突を起こさない coordinated path を生成する．

4.3 Coordinated path とモード管理

RSU は表 1 に示す 3 つ中のいずれかのモードを車両に指示する．*Auto* モードは車両が自車の生成した autonomous path に従って走行している場合である．C_{slow} モードは coordinated モードのうち車両が減速する場合である． C_{fast} は coordinated モードのうち車両が加速する場合である．RSU は図 4 のフローチャートに従って 3 つのモードを使い分ける．

初期段階では，すべての車両のモードが *Auto* にセットされている．RSU が最初の車両から受け取った future path が予約テーブルに格納される．以降の車両も衝突がない場合の future path は予約テーブルに格納されていく．受け取った複数の future path 内の各ポイントのうち，5 s 以内かつ 2.8 m 以内に別の future path のポイントが存在する

表 1 モードの種類

Status	Description
Auto	Autonomous mode
C_{slow}	Coordinated mode with slowing down
C_{fast}	Coordinated mode with speeding up

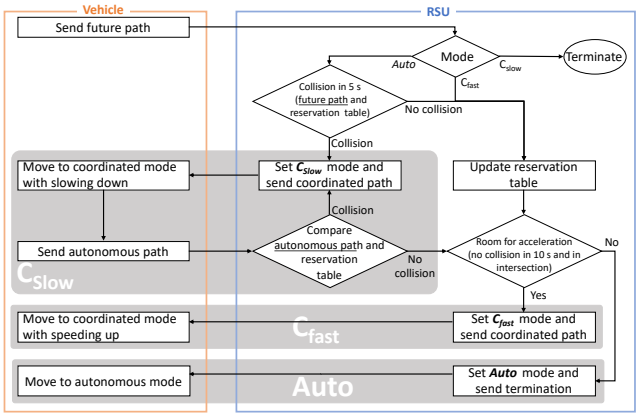


図 4　モード移行のフローチャート

場合には衝突と判定し，C_{slow} モードに移行する．モード移行の際には車両に移行を知らせるシグナルが送信される．車両から autonomous path を受け取った後，RSU は速度が 0 に置き換わった coordinated path を生成して送り返す．車両側では，衝突が解消されるまで Autoware の control module が coordinated path に従った減速を行い，最悪の場合車両は停車する．そして衝突が解消された後は，車両は加速の余地を確認し，*Auto* か C_{fast} のどちらかのモードに移行する．

一方で，衝突が無い場合には，RSU は加速の余地があるかどうか判定する．他車両の future path 内のポイントが，10 s 前までに 2.8 m 以内に無い状態で車両が交差点に進入する方向に走行している場合には加速の余地があると判定し，RSU は車両のモードを *Auto* から C_{fast} に変更する．変更時には，RSU は車両にモード変更のためのシグナルを送信する．そして RSU はその後衝突が発生しない限り交差点内は最大速度 (50 km/h) まで加速するような coordinated path を生成する．車両はその coordinated path に従って加速する．この時，加速がスムーズなものになるように Autoware の control モジュールが適切に管理してコントロールする．交差点を通過した後は，RSU がモードを *Auto* に戻し，終了シグナルを車両に送信する．以上のようにして，加減速が行われる．

5.　評価

実装した提案手法を，ドライビングシミュレータである LGSVL シミュレータ [19] の 2020.06 のバージョンを用いて実験を行い，RSU による調停が車両の交差点を通過するまでの時間にどう影響を与えるか評価した．

5.1　実験構成とシナリオ

3 台のマシン *Car A*，*Car B*，*RSU* を用いた．*RSU* では LGSVL も動かしている．Autoware と LGSVL シミュレータを組み合わせることで perception, planning, control の機能を仮想空間上で実験することができる．LGSVL にプリインストールされている Shalun (Taiwan Car Lab Testing Facility) のマップを使用し，建物がある見通しの悪い交差点で実験を行った．3 台の PC は有線でルータを介して接続されている．

図 5 に示すように，*Car A* と *Car B* を交差点を縦横それぞれ直進するように配置し，出発地点と目標地点を設定した．RSU はシミュレータ上では停止している車両として配置している．なお，初期位置は，2 台とも独立に自動運転する場合に *Car A* が *Car B* より先に交差点を通過するような位置関係に設定してある．

実験では以下のシナリオで評価を行った．

- *Stand-alone*: 2 台がそれぞれ独立に自動運転を行う
- *Future path only*: 自動運転車が車々間で future path を交換する
- *Future path with RSU*: future path の交換と RSU による調停 (**提案手法**)

それぞれのシナリオで同地点からの発着に設定し，10 回ずつ実験を行った．

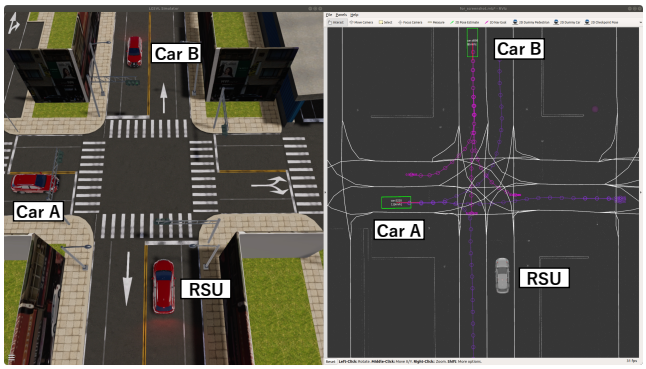


図 5　実験の様子 (左が LGSVL シミュレータ，右が Rviz の画面)

5.2　交差点の通過時間の測定

表 2 に，2 台の車両が交差点を通過するのにかかった時間を示す．

Stand-alone では *Car A* がすべての試行において先に交差点を通過した．これは初期位置の設定通りの挙動である．一方で，それぞれの試行において GUI からの操作によりわずかに出発タイミングをずらしながら測定を行った結果，*Car A* は future path only のシナリオでは 6 回の試行において，future path with RSU のシナリオでは 4 回の試行において，*Car B* より先に交差点を通過した．

表 2 が示すように，future path only のシナリオでは，36.51 s から 37.17 s に平均通過時間が 2 % 増加した．一方，提案手法 (future path with RSU) のシナリオでは，36.51 s から 27.05 s に 26 % 減少した．この結果より，単に future path を車々間で交換することにより，見かけ上の未来の障害物が増えることで交通流が非効率になることがわかった．一方提案手法のように，RSU による調停も行うこと

で，効果的に交通流を向上させることができた．

図 6 に 10 回の試行における通過時間の結果を示す．stand-alone では常に *Car A* が先に交差点を通過したため分散は 3 つのシナリオで最も小さい結果となった．車両が future path を交換する残り 2 つのシナリオでは，通過時間に幅がある結果となった．future path only のシナリオでは，10 回中 6 回が *Car A* の先着となって 4 回は *Car B* の先着となったため，*Car A* に着目すると 6.99 s 平均通過時間が長くかかった (22 % の増加)．一方，*Car B* は 5.65 s 平均通過時間が短くなった (13 % の短縮)．提案手法 (future path with RSU) のシナリオでは *Car A*，*Car B* どちらの平均通過時間も短縮され，stand-alone と比較してそれぞれ 23.0 % と 28.1 % の短縮となった．

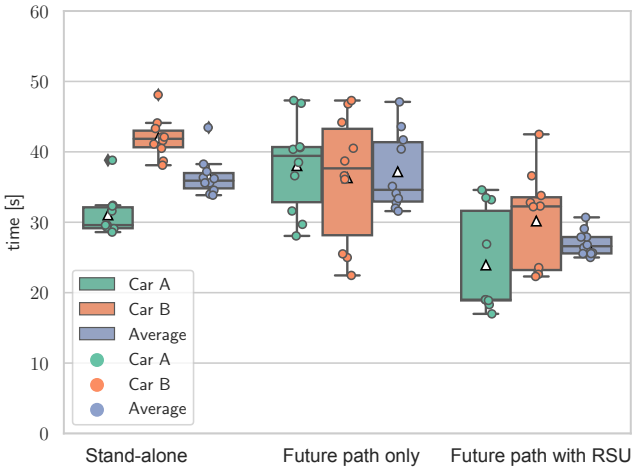


図 6　10 回の試行における交差点通過時間

図 7 に交差点付近での 2 台の車両の速度を示す．緑色が *Car A*，オレンジ色が *Car B* を表す．x 軸 y 軸どちらも大きい値の方に車両の走行方向であり，$(x, y) = (0, 0)$ が交差点である．半透明の線がそれぞれの試行の結果を表し，通常の線が 10 回の試行を平均した速度を表している．

図 7(a) に示すように，stand-alone のシナリオでは，*Car B* が常に交差点の前で停止しており，*Car A* が通過するのを待つ結果となった．図 7(b) に示すように，future path only のシナリオでは，*Car A* と *Car B* はどちらもお互いを認識して stand-alone のシナリオより早い段階で減速を行っている結果となった．

図 7(c) が示すように，future path with RSU のシナリオでは，*Car A* と *Car B* どちらも交差点前で停止することなく通過している結果となった．これは，1 台目の車両は RSU から coordinated path を受け取って加速の余地があるため加速し，2 台目の車両は 1 台目の車両を考慮した coordinated path を受け取って早めに減速を行ったことを示している．

表 2　各シナリオでの交差点の通過時間

	Stand-alone	Future Paths only	Future path with RSU
Car A passes first	10	6	6
Car B passes first	0	4	4
Car A passing time	31.04 s	38.03 s	**23.93** s
Car B passing time	41.97 s	36.32 s	**30.18** s
Average passing time	36.51 s	37.17 s	**27.05** s

6. まとめと今後の課題

協調型 ITS では自動運転車が車々間通信を利用することでリアルタイム情報を交換している．しかし，リアルタイムでの経路計画は複数車両の経路が衝突するような場合には非効率となる．そこで本研究では時空間予測に基づく交差点における協調的な経路計画システムを提案した．見通しの悪い交差点にある RSU で 2 台の自動運転車両の経路を調停する手法を，自動運転のためのオープンソースソフトウェアである Autoware とドライビングシミュレータである LGSVL を用いて実装し，実験と評価を行った．結果として，提案手法である RSU による調停を行った場合に，2 台の車両が独立に自動運転を行う場合よりも交差点の平均通過時間が 26 % 短縮されることを確認できた．

今後の課題としては，車両の数を増やす，無線通信を用いる，歩行者等の自動運転車両以外の動的障害物を取り入れるなど，より現実に即した環境での実験と評価が必要である．実際の無線通信またはネットワークシミュレータを組み合わせて実験を行ったり，実車両での提案手法の検証なども行っていく予定である．また，さらなる効率化のためにも車両の優先度を管理するアルゴリズム面での検討も必要になると考えられる．

参考文献

[1] Honda Motor Co ., Ltd. Honda receives type designation for level 3 automated driving in japan. `https://global.honda/newsroom/news/2020/4201111eng.html`, 11 2020. (Accessed on 02/04/2021).

[2] Toyota bringing advanced its technology to mass-market models — toyota motor corporation official global website. `https://global.toyota/en/detail/9676551`. (Accessed on 02/04/2021).

[3] ISO 21217:2020 Intelligent transport systems — Station and communication architecture , December 2020.

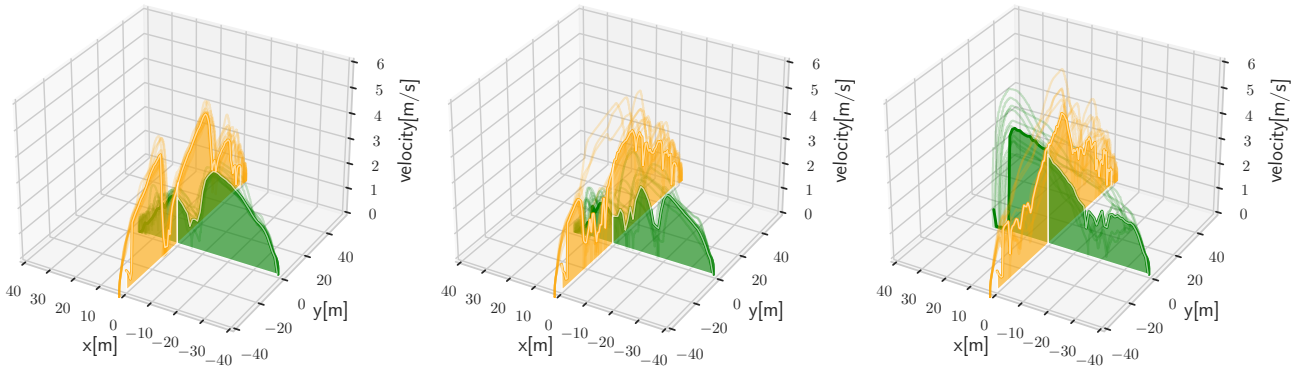
[4] Intelligent Transport Systems (ITS); Communications Architecture, September 2010. ETSI EN 302 665 V1.1.1 (2010-09).

[5] Intelligent Transport Systems (ITS); Cooperative Perception Services (CPS), December 2019. ETSI TS 103 324 V0.0.14 (2019-10).

[6] Intelligent Transport Systems (ITS);Vehicular Communications; Basic Set of Applications; Analysis of the Collective Perception Service (CPS); Release 2 , December 2019. ETSI TR 103 562 V2.1.1 (2019-12).

[7] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada. An open approach to autonomous vehicles. *IEEE Micro*, Vol. 35, No. 6, pp. 60–68, Nov 2015.

[8] T. Renzler, M. Stolz, and D. Watzenig. Looking into the



(a) Stand-alone (b) Future path only (c) Future path with RSU

図 7 2 台の車両の交差点付近の速度

path future: Extending cams for cooperative event handling. In *2020 IEEE 92nd Vehicular Technology Conference (VTC2020-Fall)*, pp. 1–5, Nov 2020.

[9] Michael Aeberhard and Nico Kaempchen. High-level sensor data fusion architecture for vehicle surround environment perception. In *Proc. 8th Int. Workshop Intell. Transp*, Vol. 665, 2011.

[10] M. Gabb, H. Digel, T. Müller, and R. Henn. Infrastructure-supported perception and track-level fusion using edge computing. In *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1739–1745, June 2019.

[11] 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.

[12] Manabu Tsukada, Takaharu Oi, Masahiro Kitazawa, and Hiroshi Esaki. Networked roadside perception units for autonomous driving. *MDPI Sensors*, Vol. 20, No. 18, 2020.

[13] R. Azimi, G. Bhatia, R. R. Rajkumar, and P. Mudalige. Stip: Spatio-temporal intersection protocols for autonomous vehicles. In *2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS)*, pp. 1–12, 2014.

[14] S. Aoki and R. Rajkumar. Dynamic intersections and self-driving vehicles. In *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 320–330, 2018.

[15] Bing Liu, Qing Shi, Zhuoyue Song, and Abdelkader El Kamel. Trajectory planning for autonomous intersection management of connected vehicles. *Simulation Modelling Practice and Theory*, Vol. 90, pp. 16 – 30, 2019.

[16] M. Bashiri and C. H. Fleming. A platoon-based intersection management system for autonomous vehicles. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pp. 667–672, 2017.

[17] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, Vol. 3, p. 5. Kobe, Japan, 2009.

[18] Alexey Melnikov and Ian Fette. The WebSocket Protocol. RFC 6455, December 2011.

[19] G Rong, et al. LGSVL simulator: A high fidelity simulator for autonomous driving. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*.