Simple Pure Pursuitモデルにおける旋回性能の改善に関する技術レポート

第I部: 既存システムの分析と問題の定式化

1.1. 調査結果と推奨事項の要約

本レポートは、提供されたC++コードで実装されたsimple_pure_pursuit（単純追従）モデルが、シケインのような高曲率区間でコースをショートカットしてしまう（極端に内側を走行する）という課題を解決するための技術的分析と具体的な改善策を提示するものです。
分析の結果、この現象は単純追従アルゴリズム、特に固定的なゲインに基づく前方注視距離（lookahead distance）の決定方法に起因する、予測可能かつ広く知られた挙動であることが確認されました。車両の速度のみに比例して前方注視距離を長くする現在の手法では、急カーブにおいて経路追従性が低下するのは必然的な結果です。
この課題を解決するため、本レポートでは以下の階層的な改善戦略を提案します。

1. 即時的な改善策（最優先）: 経路の曲率に応じて前方注視距離を動的に調整する曲率感応型適応前方注視制御を導入します。これは、実装の労力に対して最も高い改善効果が期待できる手法です。
2. 頑健性の強化: 速度、曲率、および横方向偏差（Cross-Track Error, CTE）を統合的に考慮する多因子適応前方注-視モデルへと進化させ、さらに横方向偏差（**CTE**）補償を制御則に組み込みます。
3. アーキテクチャの刷新: より高度な幾何学的制御手法である**Stanley**（スタンレー）法に置き換えます。これにより、特にアグレッシブな操舵が要求される場面で、追従性能が飛躍的に向上します。
4. 最先端技術の導入: 最高の性能を追求する場合、車両の未来の状態を予測し、システム全体の制約を考慮しながら最適な制御入力を計算する**モデル予測制御（Model Predictive Control, MPC）**の導入を検討します。ただし、これは実装の複雑性が大幅に増大します。

主要な推奨事項として、まずは**戦略1（曲率感応型適応前方注視制御）**から着手することを提案します。これにより、ユーザーが直面している特定の課題に対して、最も効率的に顕著な改善が見込めます。

1.2. 提供された**simple_pure_pursuit.cpp**の解体分析

提供されたsimple_pure_pursuit.cppソースコードは、ROS2ノードとして実装された単純追従コントローラーの典型的な例です。観測されている「コーナーカット」問題の直接的な原因を特定するため、onTimer()関数内の制御ロジックを詳細に分析します。

1.2.1. コードレビュー

制御の中核をなすロジックは、以下の要素で構成されています。

- 前方注視距離の計算:
  C++
  ```
  double lookahead_distance = lookahead_gain_ * target_longitudinal_vel +
  lookahead_min_distance_;
  ```

  この一行が、問題の核心です。前方注視距離（lookahead_distance）が、車両の目標速度（target_longitudinal_vel）の一次関数としてのみ決定されています。この実装では、経路の形状（曲率）が完全に無視されています。高速走行時には前方注視距離が長くなるため、カーブでは必然的に内側をショートカットする軌道を描くことになります [1]。
- 参照点:
  C++
  ```
  double rear_x = odometry_->pose.pose.position.x -
              wheel_base_ / 2.0 * std::cos(odometry_->pose.pose.orientation.z);
  double rear_y = odometry_->pose.pose.position.y -
              wheel_base_ / 2.0 * std::sin(odometry_->pose.pose.orientation.z);
  ```

  車両の参照点として後輪軸中心（rear_x, rear_y）が使用されています。これは、前方注視点の探索と操舵角の計算の両方において、古典的な単純追従アルゴリズムの標準的な定義に準拠しています [4]。
- 操舵角の計算:
  C++
  ```
  double alpha = std::atan2(lookahead_point_y - rear_y, lookahead_point_x - rear_x) -
              tf2::getYaw(odometry_->pose.pose.orientation);
  cmd.lateral.steering_tire_angle =
    steering_tire_angle_gain_ * std::atan2(2.0 * wheel_base_ * std::sin(alpha),
  lookahead_distance);
  ```

  この計算式は、前方注視点への角度alphaと前方注視距離lookahead_distanceを用いて、幾何学的な円弧を車両が追従するために必要な操舵角を算出するものです。これは、運動学的な自転車モデルに基づく単純追従の正確な数学的表現です [2]。
- 誤差項の欠如:
  この制御ロジック全体を俯瞰すると、経路からの横方向のずれ（横方向偏差、CTE）や、経路の向きと車両の向きの差（方位角偏差）を明示的に扱う項が存在しないことがわかります。制御は純粋に前方注視点への角度alphaのみに依存しています。この点が、Stanley法のような

より高度な幾何学的コントローラーとの決定的な違いです 6。

1.3. 単純追従制御におけるコーナーカットの幾何学的起源

単純追従アルゴリズムの「コーナーカット」現象は、バグではなく、その基本的な幾何学的原理に根差した本質的な特性です。

1.3.1. 「人参を追いかける馬」のアナロジー

このアルゴリズムは、しばしば「馬の鼻先に吊るされた人参」に例えられます [1]。車両（馬）は、常に経路上の少し先にある単一の目標点、すなわち前方注視点（人参）に向かって進もうとします。この目標点は、車両の移動に伴って経路に沿って前進していきます。

1.3.2. 前方注視距離のトレードオフ

このアルゴリズムの性能は、前方注視距離l_dのチューニングに極めて敏感であり、そこには根本的なトレードオフが存在します。

- 長い前方注視距離: l_dを長く設定すると、車両は遠くの目標点を参照するため、制御が滑らかになり安定性が向上します。しかし、急なカーブに差し掛かると、目標点はカーブの頂点をはるかに超えた位置に設定されます。その結果、車両はその遠い目標点に向かって大きな半径の円弧を描こうとするため、カーブの内側をショートカットする「コーナーカット」現象が発生します [1]。
- 短い前方注視距離: l_dを短く設定すると、車両はすぐ近くの目標点を参照するため、経路への追従精度は向上します。しかし、小さな偏差にも過敏に反応し、操舵が頻繁に修正されるため、特に高速走行時には車両が蛇行する振動的な（オシレーション）挙動を引き起こし、不安定になりがちです [1]。

1.3.3. シケイン走行シナリオの分析

ユーザーが直面しているシケインでの問題は、このトレードオフが典型的に現れたものです。シケインは、高曲率のS字カーブが連続する区間です。提供されたコードでは、速度が上がるにつれて前方注視距離l_dが長くなります。車両が高速でシケインに進入すると、大きなl_dが設定され、前方注視点はシケインの最初の頂点（エイペックス）を大きく越えた先に置かれます。これにより、車両は最初のカーブを大きくショートカットします。そして、2つ目の頂点に対して適切に操舵を修正する前に、車両はすでに目標経路から大きく逸脱してしまっているのです。
この問題の根源は、単純追従アルゴリズムが車両と前方注視点の間の経路形状に関する情報を一切持たないことにあります。アルゴリズムは、車両の現在位置と前方注視点を結ぶ経路が、常に単一の滑らかな円弧で近似できると仮定しています [4]。しかし、シケインのように経路が複雑に曲がりく

ねっている場合、この仮定は成立しません。アルゴリズムは、前方注視点が実際に高曲率の領域に入るまで、目前に迫る急カーブを「予見」することができません。その時点では、多くの場合、適切に反応するには手遅れです。この「先見性の欠如」こそが、これから提案する全ての解決策が、異なるアプローチで克服しようとする根本的な課題なのです。適応制御は曲率に反応することで擬似的な先見性を与え、MPCは予測モデルを用いることで真の先見性を与えます。

第II部: 単純追従アルゴリズムの直接的な機能強化

提供されたコードの根本的な問題を解決するため、既存の単純追従アルゴリズムの枠組みの中で実現可能な、直接的な機能強化策を二段階で提案します。第一段階は、問題の直接的な原因である前方注視距離の決定ロジックを高度化することです。第二段階は、制御則自体に新たな誤差項を導入し、追従精度をさらに向上させることです。

2.1. 戦略1: 高度な適応前方注視制御

ユーザーが直面しているコーナーカット問題を解決するための最も直接的かつ効果的な変更は、前方注視距離を固定的なゲインではなく、走行状況に応じて動的に変化させる「適応前方注視制御」を導入することです。特に、経路の曲率を考慮に入れることが不可欠です。

2.1.1. 曲率感応型前方注視制御の実装

基本原理: 前方注視距離は、経路の曲率に対して逆比例の関係にあるべきです。すなわち、曲率が大きい（カーブが急な）場合は前方注視距離を短くしてカーブに密着させ、曲率が小さい（直線に近い）場合は前方注視距離を長くして安定性を確保します [11]。
経路曲率の計算: 経路は離散的なウェイポイントの集合として与えられるため、曲率を数値的に推定する必要があります。以下に代表的な手法を示します。

- 手法A: メンガーの曲率（3点円）
  最も単純な手法です。経路上の最近傍点、その一つ手前の点、一つ先の点の3点を通る唯一の円を定義し、その円の半径の逆数を曲率とします 17。この方法は実装が容易であり、迅速な改善に適しています。

  3点P_1, P_2, P_3が与えられたとき、曲率κは以下の式で計算できます。

  κ=|P1−P2||P2−P3||P3−P1|4·Area(P1,P2,P3)

  ここで、Areaは3点がなす三角形の面積、$|P_i - P_j|$は2点間の距離です。
- 手法B: 経路平滑化と有限差分
  経路データにノイズが多い場合、まず移動平均フィルタなどの平滑化処理を適用します（Autowareの実装でも平滑化パラメータが用意されています 20）。その後、平滑化された点列

に対して有限差分法を用いて経路の1階および2階微分を近似し、曲率を計算します。この手法はノイズに対してより頑健です。

- 手法C: クロソイド曲線フィッティング
  MATLABのreferencePathFrenetなどで用いられる、より高度で正確な手法です 21。ウェイポイント間を、曲率が弧長に対して線形に変化するクロソイド曲線で接続します。これにより、区分的に滑らかな
  C^2連続の経路が得られ、各点での正確な曲率が解析的に求まります。本件の解決策としては過剰性能かもしれませんが、完全なレポートとして言及する価値があります。

表**1:** 経路曲率の計算手法比較

手法名	原理	C++実装の複雑度	計算コスト	精度/ノイズ耐性
メンガーの曲率	連続する3点を通る円の半径の逆数から曲率を算出する。	低	低	ノイズに敏感。平滑化が推奨される。
平滑化と有限差分	移動平均などで経路を平滑化した後、差分法で微分を近似して曲率を計算する。	中	中	ノイズに強いが、平滑化により経路の鋭さが失われる可能性がある。
クロソイド曲線	ウェイポイント間をクロソイド曲線で補間し、解析的に曲率を求める。	高	高	非常に高い精度。C^2連続性を保証する。

この比較から、迅速な実装と効果を求めるならばメンガーの曲率が、より高い信頼性を求めるならば平滑化と有限差分が現実的な選択肢となります。

2.1.2. 頑健な多因子前方注視モデル

実用的な自動運転システムでは、単一の要素だけでなく、複数の要素を組み合わせて前方注視距離を決定します。ここでは、AutowareのPure Pursuitコントローラーの実装を参考に、より頑健なモデルを提案します [20]。

提案する前方注視距離の計算式:

前方注視距離l_dは、基本距離、速度、曲率、横方向偏差の4つの要素の重み付き和として定義します。

$l_d = l_{d_{min}} + k_v \cdot v + k_c \cdot \frac{1}{\kappa_{abs}} + k_e \cdot |e_{cte}|$

各項目の解説**:**

- $l_{d_{min}}$: 最小前方注視距離。ユーザーコードのlookahead_min_distance_に相当し、低速時や急カーブでの過度な接近を防ぎます [20]。
- $k_v \cdot v$: 速度比例項。ユーザーが既に実装している項で、高速走行時の安定性を確保します。
- $k_c \cdot \frac{1}{\kappa_{abs}}$: 曲率反比例項。これが今回の改善の核となる項です。

κ_{abs}は曲率の絶対値です。直線路（$\kappa \approx 0$）でのゼロ除算を避けるため、分母に微小値を追加するか、この項に上限値を設ける必要があります [17]。

- $k_e \cdot |e_{cte}|$ (オプション): 横方向偏差比例項。経路から大きく離れている場合に前方注視距離を伸ばし、より滑らかな経路復帰を促す効果があります [11]。

実装ガイダンス:

1. クラスに新しい関数double calculatePathCurvature(const size_t closest_idx) constを追加します。この関数は、指定されたインデックス周辺の3点（例：idx-1, idx, idx+1）を用いてメンガーの曲率を計算し、返します。境界条件（インデックスが0や終端の場合）の処理に注意が必要です。
2. onTimer()ループ内で、まずfindNearestIndexで最近傍点closet_traj_point_idxを求めます。
3. calculatePathCurvature(closet_traj_point_idx)を呼び出して、現在の車両位置に最も近い経路上の曲率κを計算します。
4. 上記の多因子モデル式を用いて、新しい適応的な前方注視距離lookahead_distanceを計算します。この際、計算されたlookahead_distanceがmin_lookahead_distanceとmax_lookahead_distanceの範囲内に収まるようにクランプ処理を行います。
5. 最後に、計算されたlookahead_distanceを用いて前方注視点を探索します。

C++コードスニペット（概念）:

C++

```
// in SimplePurePursuit::onTimer() after finding closet_traj_point_idx

// 1. Calculate path curvature
double curvature = calculatePathCurvature(closet_traj_point_idx);

// 2. Calculate adaptive lookahead distance
// Note: k_c is a new parameter for curvature gain
const double curvature_gain = 1.0; // Tune this parameter
double lookahead_distance = lookahead_gain_ * target_longitudinal_vel +
lookahead_min_distance_;
if (std::abs(curvature) > 1e-3) { // Avoid division by zero
  lookahead_distance += curvature_gain / std::abs(curvature);
}

// Clamp the lookahead distance
const double max_lookahead_distance = 15.0; // Example value
lookahead_distance = std::clamp(lookahead_distance, lookahead_min_distance_,
max_lookahead_distance);

//... proceed with searching for the lookahead point using the new lookahead_distance
```

前方注視点の探索方法には、2つの主要なアプローチが存在します。ユーザーのコードは「車両から

のユークリッド距離」で探索していますが、より頑健な方法は「経路上での弧長距離」で探索することです [12]。後者は、経路の事前処理として各ウェイポイントまでの累積弧長を計算しておく必要がありますが、ループ状の経路など複雑なケースで曖昧さがなくなり、より正確な追従が可能になります。シケインの問題では現在の方法でも十分機能しますが、将来的な拡張性を見据え、この弧長ベースの探索方法への切り替えも検討する価値があります。

2.2. 戦略2: 横方向偏差（CTE）補償の導入

単純追従の性能をさらに向上させるため、制御則に横方向偏差（Cross-Track Error, CTE）を補償する項を追加します。これは、車両が現在、目標経路からどれだけ横にずれているかを積極的に修正しようとするアプローチです。
原理: 単純追従の操舵角は、前方注視点への角度のみに基づいています。これに対し、CTE補償は、車両の現在の横ずれe_{cte}を直接測定し、それをゼロに近づけるような制御入力を追加します [23]。これにより、経路への収束が速まり、定常的なオフセットを低減できます。
実装:
幸いなことに、提供されたコードは既にtier4_autoware_utils::calcLateralDeviationをインクルードしており、これを使えばCTEを容易に計算できます。

1. **CTEの計算:** onTimer()内で、最近傍点closet_traj_pointと現在の車両姿勢odometry_->pose.poseを用いて、calcLateralDeviationを呼び出し、横方向偏差e_{cte}を取得します。
2. 制御則の変更: 算出されたCTEを用いて、最終的な操舵角を補正します。最も簡単な方法は、単純追従で計算された操舵角に、CTEに比例する項を加えることです。

修正された操舵則（概念）:
$\delta_{final} = \delta_{pp} + k_{cte} \cdot e_{cte}$
ここで、δ_{pp}は従来の単純追従で計算された操舵角、e_{cte}は横方向偏差、k_{cte}は新たに導入するCTE補償ゲインです。このゲインは、車両がどれだけ積極的に経路中心に戻ろうとするかを決定します。
この変更により、コントローラーは「前方注視点を追いかける」という本来の滑らかな挙動を維持しつつ、経路からのずれを修正する「押し戻す力」を持つハイブリッドな特性を獲得します。
このCTE補償の導入は、単なるパッチ適用以上の哲学的な意味を持ちます。これは、制御の焦点が純粋な「追従ベース」から、より高度な「誤差ベース」へと移行する第一歩です。単純追我は「どこへ行くべきか？」を問いますが、CTEを考慮した制御は「どこへ行くべきか？ かつ 現在の誤差をどう修正するか？」を問います。この論理こそが、次章で解説するStanley法やLQR/MPCといった現代制御理論の根幹をなすものです [6]。この小さな変更を実装することは、より高度な制御アーキテクチャへの理解を深める重要なステップとなります。

第III部: 代替となる経路追従アーキテクチャ

単純追従アルゴリズムの改良には限界があります。特に、アグレッシブな操舵と高い追従精度が同時に求められる状況では、アルゴリズムの構造自体を見直すことが有効です。ここでは、幾何学的ア

プローチの代表格である**Stanley**法と、最適制御に基づく最先端のアプローチである**モデル予測制御（MPC）**を紹介します。

3.1. Stanley法: アグレッシブな操舵に適した優れた幾何学的コントローラー

Stanley法は、スタンフォード大学のチームがDARPAグランドチャレンジで優勝した際に使用したことで知られる、非常に効果的な幾何学的経路追従コントローラーです。単純追従と比較して、特に急カーブでの追従性能に優れています。

3.1.1. 単純追従とStanley法の比較

Stanley法が単純追従と根本的に異なる点は、参照点と誤差の定義にあります。

- 参照点: Stanley法は、車両の前輪軸中心を制御の参照点とします。後輪軸中心を用いる単純追従とは対照的です [7]。
- 誤差補正: Stanley法の制御則は、明確に分離された2つの誤差項の和で構成されます [6]。
    1. 方位角偏差（**Heading Error, ψ**）: 車両の進行方向と、経路上で車両に最も近い点の接線方向との角度差。この項は、車両の向きを経路に沿わせるように作用します。
    2. 横方向偏差（**Cross-Track Error, e**）: 車両の前輪軸中心から、経路上の最近傍点までの横方向の距離。この項は、車両を経路の中心に引き戻すように作用します。

Stanley法の制御則:
Stanley法による操舵角$\delta(t)$は、以下の式で与えられます。
$\delta(t) = \psi(t) + \arctan\left(\frac{k \cdot e(t)}{v(t) + k_s}\right)$
ここで、$\psi(t)$は方位角偏差、$e(t)$は横方向偏差、$v(t)$は現在の車速です。kは横方向偏差に対するゲイン、k_sは低速時の制御を安定させるためのソフトニング定数（ゼロ除算防止）です。
シケイン問題の解決メカニズム:
単純追従が遠くの前方注視点に依存するのに対し、Stanley法は最も近い経路に対する自身の位置と向きの誤差を直接測定します。車両がシケインの最初の頂点に近づくと、経路の向きが急激に変わるため、方位角偏差ψが大きくなります。コントローラーは即座にこれを補正しようと大きな操舵角を指令し、車両の向きを経路に合わせようとします。同時に横方向偏差eも補正するため、結果としてカーブに吸い付くような、はるかにタイトな追従が実現されます 8。

3.1.2. Stanleyコントローラーの実装ガイド

提供されたSimplePurePursuitノードの構造を参考に、StanleyControllerクラスをC++で実装するための手順を以下に示します。

1. クラス構造の定義:
   SimplePurePursuitと同様のROS2ノード構造を持つStanleyControllerクラスを作成します。パラメータとして、wheel_base_、横方向偏差ゲインk、ソフトニング定数k_sなどを宣言します。

2. 制御ループ **(onTimer)** の実装**:**
    - ステップ1: 前輪軸中心位置の計算
      現在の車両位置(x, y)と方位角θ_{yaw}から、前輪軸中心の位置$(front_x, front_y)$を計算します。
      $front\_x = x + \frac{wheel\_base}{2} \cdot \cos(\theta_{yaw})$
      $front\_y = y + \frac{wheel\_base}{2} \cdot \sin(\theta_{yaw})$
    - ステップ2: 経路上最近傍点の探索
      前輪軸中心位置$(front_x, front_y)$に最も近い経路上のウェイポイントを findNearestIndexを用いて見つけます。これが目標点となります。
    - ステップ3: 横方向偏差（e）の計算
      前輪軸中心と最近傍点を結ぶベクトルを計算します。このベクトルの長さが横方向偏差の大きさです。偏差の符号（経路の右側にいるか左側にいるか）は、車両の進行方向ベクトルと、最近傍点へのベクトルの外積を計算することで決定できます。正負の定義を明確にすることが重要です 6。
      C++
      ```
      // Conceptual code for signed cross-track error
      double path_dx = trajectory_->points[target_idx + 1].pose.position.x - trajectory_->points[target_idx].pose.position.x;
      double path_dy = trajectory_->points[target_idx + 1].pose.position.y - trajectory_->points[target_idx].pose.position.y;
      double error_vec_x = front_x - trajectory_->points[target_idx].pose.position.x;
      double error_vec_y = front_y - trajectory_->points[target_idx].pose.position.y;

      double cross_product = error_vec_x * path_dy - error_vec_y * path_dx;
      double cte = std::hypot(error_vec_x, error_vec_y);
      if (cross_product > 0) {
        cte = -cte;
      }
      ```
    - ステップ4: 方位角偏差（ψ）の計算
      最近傍点における経路の方位角θ_{path}を、最近傍点とその次の点から計算します。車両の現在の方位角θ_{yaw}との差が方位角偏差ψとなります。
      $\psi = \theta_{path} - \theta_{yaw}$
      計算結果は$-\pi$からπの範囲に正規化する必要があります。
    - ステップ5: Stanley制御則の適用
      ステップ3と4で求めたeとψを用いて、最終的な操舵角δを計算します。
      C++
      ```
      const double k_crosstrack = 1.0; // Crosstrack gain (tune this)
      const double k_softening = 1e-6; // Softening constant
      double current_velocity = odometry_->twist.twist.linear.x;

      double steering_crosstrack = std::atan2(k_crosstrack * cte, current_velocity + k_softening);
      ```

double steering_angle = normalize_angle(heading_error + steering_crosstrack);

- ステップ6: 制御指令の出力
  計算されたsteering_angleをAckermannControlCommandメッセージにセットしてパブリッシュします。

実装にあたっては、GitHubなどで公開されているオープンソースのStanleyコントローラー実装（例: [30]）が非常に参考になります。

3.2. 最先端技術の概要: モデル予測制御（MPC）

幾何学的手法が現在の誤差に基づいて反応的に制御を行うのに対し、モデル予測制御（MPC）は、未来を予測し、制約の中で最適な制御を行う、より高度な制御フレームワークです。
概念的フレームワーク: MPCは固定されたアルゴリズムではなく、以下の3つの要素から構成される最適化問題の繰り返し求解プロセスです [35]。

1. 予測モデル: 車両の運動学または動力学モデル。現在の状態$x(t)$と制御入力$u(t)$から、短い未来（予測ホライズン）の状態$x(t+1), x(t+2),...$を予測するために使用されます。
2. コスト関数: 最小化を目指す評価関数。通常、経路追従誤差（横方向偏差、方位角偏差）、制御入力の大きさ（エネルギー消費）、制御入力の変化率（乗り心地、アクチュエータへの負荷）などの項の重み付き和で定義されます [37]。
3. 制約条件: 車両の物理的な限界（最大操舵角、最大加速度など）や安全上の要求（障害物との距離など）を不等式で表現します。

単純追従/Stanley法に対する優位性:

- 真の先見性: MPCは、前方注視点のような単一の点ではなく、未来の経路セグメント全体を考慮して制御計画を立てるため、カーブに対して事前に準備することができます。
- 制約の明示的考慮: 最大操舵角や最大加速度といった物理的な制約を最適化問題に直接組み込むことができるため、アクチュエータの飽和を防ぎ、車両の限界内で最善の性能を引き出します。
- 最適性: 与えられたモデル、コスト関数、制約条件の下で、数学的に最適な制御入力系列を見つけ出します。

適用性とトレードオフ:
MPCは最高の追従性能を発揮するポテンシャルを持ちますが、その代償として実装の複雑性と計算コストが大幅に増加します 38。予測モデルの構築、コスト関数のチューニング、そしてリアルタイムで非線形最適化問題を解くためのソルバーの導入など、専門的な知識と技術が要求されます。
ユーザーのシケイン問題に対しては、MPCは過剰性能である可能性が高いですが、Stanley法でも性能が不十分な場合や、将来的に車両の動的限界を考慮したより高度な制御を目指す場合には、MPCが論理的な次のステップとなります。これは、経路追従制御における「究極の」ソリューションと位置づけられます。

第IV部: 提案手法の統合的評価と実装ロードマップ

これまでに提案した複数の改善策と代替アーキテクチャについて、性能、実装コスト、計算負荷の観点から統合的に評価し、ユーザーが段階的にシステムを改善していくための具体的な実装ロードマップを提示します。

4.1. 提案手法の比較分析

各手法の特性をまとめた比較表を以下に示します。この表は、ユーザーが自身の目標とリソースに応じて最適な解決策を選択するための意思決定ツールとして機能します。

表2: 制御戦略の比較

戦略	中核原理	コーナリング性能	安定性	実装労力	計算負荷
単純追従 (ベースライン)	後輪軸から前方注視点を追従	低 (コーナーカット)	速度とl_dに依存	-	極低
適応型単純追従 (曲率感応)	曲率に応じてl_dを動的に変更	中～高 (大幅改善)	向上	低	低
単純追従 + CTE補償	操舵角に横方向偏差の補正項を追加	中 (精度向上)	向上	低	低
Stanley法	前輪軸を基準に方位角偏差と横方向偏差を補正	高 (優れた追従性)	高	中	低～中
モデル予測制御 (MPC)	未来の状態を予測し、コスト関数を最小化する最適制御	極高 (最適)	極高	高	高

この表から、いくつかの重要な示唆が得られます。

- 適応型単純追従は、実装労力が低く、計算負荷もほとんど変わらないにもかかわらず、コーナーカット問題に対して顕著な改善をもたらすため、コストパフォーマンスが最も高い選択肢です。
- Stanley法は、単純追従の枠組みを脱し、アーキテクチャを変更する必要があるため実装労力は増えますが、MPCのような高度な最適化ソルバーを必要とせず、計算負荷を低く抑えながら優れたコーナリング性能と安定性を両立できる、非常にバランスの取れた強力な選択肢です。
- MPCは、最高の性能を提供しますが、その複雑さと計算コストから、他の手法では要求仕様を満たせない場合の最終手段と考えるのが妥当です。

4.2. 推奨される段階的実装計画

ユーザーが効率的に問題を解決し、システムを段階的に強化していくための実装計画を3つのフェーズで提案します。

フェーズ**1:** クイックウィン（低労力・高効果）

このフェーズの目標は、最小限のコード変更で、現在直面しているコーナーカット問題を迅速に解決することです。

1. 経路曲率計算の実装**:** まず、表**1**で示したメンガーの曲率法を用いて、経路の曲率を計算する関数を実装します。これは3つの連続したウェイポイントから幾何学的に計算できるため、最も手軽です。
2. 前方注視距離の適応化**:** lookahead_distanceの計算式を修正し、計算した曲率に反比例する項を追加します（セクション**2.1.2**参照）。
3. ゲインの調整**:** 新たに導入した曲率ゲインk_cを調整し、シケインでの走行挙動が改善されることを確認します。
4. 期待される成果**:** シケイン走行時の大幅なコーナーカット現象が抑制され、より目標経路に忠実な走行が可能になります。

フェーズ**2:** 頑健なソリューション（中労力・高性能）

フェーズ1で性能が不十分な場合、またはより汎用性の高い高性能なコントローラーを目指す場合、Stanley法への移行を推奨します。

1. **Stanley**コントローラーの実装**:** セクション**3.1.2**のガイドに従い、新しいC++クラスとしてStanleyコントローラーを実装します。
2. 誤差計算の正確性**:** 特に、符号付きの横方向偏差（CTE）と方位角偏差の計算を正確に行うことに注力します。ここでの計算ミスは、制御の不安定性に直結します。
3. ゲインの調整**:** Stanley法のゲインk（横方向偏差ゲイン）とソフトニング定数k_sを、直線およびカーブでの走行を通じて調整します。
4. 期待される成果**:** シケインだけでなく、あらゆる走行条件下で単純追従を凌駕する追従性能と安定性を実現します。実装の手間はかかりますが、その価値は十分にあります。

フェーズ**3:** 究極の目標（高労力・最適性能）

最高の性能を追求し、車両の動的な限界まで考慮した制御が必要な場合の最終目標です。

1. **MPC**ライブラリの調査**:** ROS2と互換性のあるMPCライブラリ（例：acadosなど）や、既存のMPCコントローラーパッケージ（例：Autowareのmpc_follower）の調査から始めます。
2. モデルとコスト関数の定義**:** 車両の運動学モデル（または動力学モデル）、追従誤差や制御入力を評価するコスト関数、そして車両の物理的制約を定義します。これはMPCの性能を決定する最も重要な作業です。
3. 期待される成果**:** あらゆる条件下で、定義されたコスト関数と制約の下で最適な走行を実現します。ただし、このフェーズは制御理論と最適化に関する深い知識を要する、大規模な開発プ

ロジェクトとなります。
この段階的なアプローチにより、ユーザーはまず目の前の課題を効率的に解決し、その後、必要に応じてより高度で頑健なシステムへと計画的に発展させることが可能となります。

引用文献

1. Pure Pursuit Controller - MATLAB & Simulink - MathWorks, 7月 31, 2025にアクセス、https://www.mathworks.com/help/nav/ug/pure-pursuit-controller.html
2. autonomous-driving-book/book/3-trajectory-tracking/lateral-control/pure-pursuit.md at main - GitHub, 7月 31, 2025にアクセス、https://github.com/YangyangFu/autonomous-driving-book/blob/main/book/3-trajectory-tracking/lateral-control/pure-pursuit.md
3. Pure Pursuit - Linear and angular velocity control commands - Simulink - MathWorks, 7月 31, 2025にアクセス、https://www.mathworks.com/help/nav/ref/purepursuit.html
4. Implementation of the Pure Pursuit Path Tracking Algorithm - Carnegie Mellon University Robotics Institute, 7月 31, 2025にアクセス、https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf
5. Control strategies for autonomous vehicle path tracking: A comparative study of PID, Pure-Pursuit, and Stanley methods - ResearchGate, 7月 31, 2025にアクセス、https://www.researchgate.net/publication/393200397_Control_strategies_for_autonomous_vehicle_path_tracking_A_comparative_study_of_PID_Pure-Pursuit_and_Stanley_methods
6. Three Methods of Vehicle Lateral Control: Pure Pursuit, Stanley and MPC, 7月 31, 2025にアクセス、https://www.shuffleai.blog/blog/Three_Methods_of_Vehicle_Lateral_Control.html
7. Three Methods of Vehicle Lateral Control: Pure Pursuit, Stanley and MPC | by Yan Ding, 7月 31, 2025にアクセス、https://dingyan89.medium.com/three-methods-of-vehicle-lateral-control-pure-pursuit-stanley-and-mpc-db8cc1d32081
8. Stanley Method - Steven Gong, 7月 31, 2025にアクセス、https://stevengong.co/notes/Stanley-Method
9. Pure Pursuit | FTCLib Docs, 7月 31, 2025にアクセス、https://docs.ftclib.org/ftclib/pathing/pure-pursuit
10. controllerPurePursuit - Create controller to follow set of waypoints - MATLAB - MathWorks, 7月 31, 2025にアクセス、https://www.mathworks.com/help/nav/ref/controllerpurepursuit-system-object.html
11. Pure pursuit geometry diagram showing the path to be tracked(left) and the calculated steering curvature(right). - ResearchGate, 7月 31, 2025にアクセス、https://www.researchgate.net/figure/Pure-pursuit-geometry-diagram-showing-the-path-to-be-trackedleft-and-the-calculated_fig2_318856824
12. ACCURATE PATH TRACKING BY ADJUSTING LOOK-AHEAD POINT IN PURE PURSUIT METHOD, 7月 31, 2025にアクセス、

http://dyros.snu.ac.kr/wp-content/uploads/2021/02/Ahn2021_Article_AccuratePathTrackingByAdjustin-1.pdf
13. Adaptive Lookahead Pure-Pursuit for Autonomous Racing, 7月 31, 2025にアクセス、https://par.nsf.gov/servlets/purl/10320139
14. Regulated Pure Pursuit for Robot Path Tracking - arXiv, 7月 31, 2025にアクセス、https://arxiv.org/pdf/2305.20026
15. Vehicle Path Tracking Control Using Pure Pursuit with MPC-based Look-ahead Distance Optimization - ACL, 7月 31, 2025にアクセス、https://acl.kaist.ac.kr/wp-content/uploads/2023/2023TVT_KST.pdf
16. Curvature Sensitive Modification of Pure Pursuit Control1 | Letters ..., 7月 31, 2025にアクセス、https://asmedigitalcollection.asme.org/lettersdynsys/article/4/2/021002/1194486/Curvature-Sensitive-Modification-of-Pure-Pursuit
17. Introduction to Pure Pursuit Tracking Algorithm | Vines' Log, 7月 31, 2025にアクセス、https://vinesmsuic.github.io/robotics-purepersuit/
18. Unscented Transform-Based Pure Pursuit Path-Tracking Algorithm Under Uncertainty - SciTePress, 7月 31, 2025にアクセス、https://www.scitepress.org/Papers/2024/129814/129814.pdf
19. Calculate curvature for 3 Points (x,y) - Stack Overflow, 7月 31, 2025にアクセス、https://stackoverflow.com/questions/41144224/calculate-curvature-for-3-points-x-y
20. Pure Pursuit Controller - Autoware Universe Documentation, 7月 31, 2025にアクセス、https://autowarefoundation.github.io/autoware_universe/main/control/autoware_pure_pursuit/
21. referencePathFrenet - Smooth reference path fit to waypoints - MATLAB - MathWorks, 7月 31, 2025にアクセス、https://de.mathworks.com/help/nav/ref/referencepathfrenet.html
22. Path pursuit algorithm - Google Groups, 7月 31, 2025にアクセス、https://groups.google.com/g/diyrovers/c/Ry8kZyktHvI
23. Control, 7月 31, 2025にアクセス、https://rpal.cs.cornell.edu/foundations/control.pdf
24. Constrained Optimization Path Following of Wheeled Robots in Natural Terrain, 7月 31, 2025にアクセス、https://www.ri.cmu.edu/pub_files/pub4/howard_thomas_2006_1/howard_thomas_2006_1.pdf
25. Outdoor Tests of Autonomous Navigation System Based on Two Different Reference Points of PurePursuit Algorithm for 10-ton Articulated Vehicle - DiVA portal, 7月 31, 2025にアクセス、https://www.diva-portal.org/smash/get/diva2:1830119/FULLTEXT01.pdf
26. Understanding Geometric Path Tracking Algorithms — Stanley Controller - Medium, 7月 31, 2025にアクセス、https://medium.com/roboquest/understanding-geometric-path-tracking-algorithms-stanley-controller-25da17bcc219
27. Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller

Design, Experimental Validation and Racing - Stanford AI Lab, 7月 31, 2025にアクセス、https://ai.stanford.edu/~gabeh/papers/hoffmann_stanley_control07.pdf
28. Comparing the performance of lateral control algorithms on long rigid vehicles in urban environments - Journal of Emerging Investigators, 7月 31, 2025にアクセス、https://emerginginvestigators.org/articles/22-104/pdf
29. [Stanley Controller] Incorrect lateral (crosstrack) error term · Issue #161 · AtsushiSakai/PythonRobotics - GitHub, 7月 31, 2025にアクセス、https://github.com/AtsushiSakai/PythonRobotics/issues/161
30. br5555/stanley - GitHub, 7月 31, 2025にアクセス、https://github.com/br5555/stanley
31. Atharva-05/stanley_control: This repository contains an implementation of the geometric path tracking "Stanley" controller for traversal of smooth continuous curved paths. - GitHub, 7月 31, 2025にアクセス、https://github.com/Atharva-05/stanley_control
32. armando-genis/stanley_controller_cpp: stanley controller in c++ - GitHub, 7月 31, 2025にアクセス、https://github.com/armando-genis/stanley_controller_cpp
33. stanley-lateral-controller · GitHub Topics, 7月 31, 2025にアクセス、https://github.com/topics/stanley-lateral-controller
34. The submission contains a model to show the implementation of Stanley controller on a vehicle moving in a scene. - GitHub, 7月 31, 2025にアクセス、https://github.com/mathworks/vehicle-stanley-controller
35. Model Predictive Control With Learned Vehicle Dynamics for Autonomous Vehicle Path Tracking - ResearchGate, 7月 31, 2025にアクセス、https://www.researchgate.net/publication/354588244_Model_Predictive_Control_With_Learned_Vehicle_Dynamics_for_Autonomous_Vehicle_Path_Tracking
36. Model predictive control - Wikipedia, 7月 31, 2025にアクセス、https://en.wikipedia.org/wiki/Model_predictive_control
37. MPC Algorithm - Autoware Universe Documentation, 7月 31, 2025にアクセス、https://autowarefoundation.github.io/autoware.universe_planning/pr-5583/control/mpc_lateral_controller/model_predictive_control_algorithm/
38. A critical evaluation of Pure Pursuit, MPC and MPCC: Balancing simplicity, performance and constraints - MATEC Web of Conferences, 7月 31, 2025にアクセス、https://www.matec-conferences.org/articles/matecconf/pdf/2024/18/matecconf_rapdasa2024_04013.pdf
39. Comparison of Linear Time Varying Model Predictive Control and Pure Pursuit Control for Autonomous Vehicles - kth .diva, 7月 31, 2025にアクセス、https://kth.diva-portal.org/smash/get/diva2:1850279/FULLTEXT02.pdf
40. Model Predictive Control vs Pure Pursuit - General Discussions - RobotShop Community, 7月 31, 2025にアクセス、https://community.robotshop.com/forum/t/model-predictive-control-vs-pure-pursuit/46512