ホワイトペーパー

モデル予測制御 (MPC) を高速化する 3 つの方法

MathWorks®

はじめに

モデル予測制御 (MPC) は、1980 年代からプロセス制御に使用されている高度な制御手法です。マイクロプロセッサの計算能力の向上に伴い、MPC の適用範囲は、自動車や航空宇宙産業分野のリアルタイム組み込みアプリケーションにも広がりつつあります。MPC では、入出力チャネルが相互干渉する多入力多出力 (MIMO) システムを扱うことができるため、制御ループの構造が簡略化されます。たとえば、PID 制御を用いて MIMO システムのコントローラーを設計しようとすると、制御ループどうしの応答が相互依存するため調整が難しくなる場合があります。さらに、MPC では入力、出力、および状態に関する制約を扱うことができます。実際のシステムでは考慮すべき物理的な制約があるため、これは重要な点です。線形 2 次レギュレーター (LQR) のような他の MIMO 技術では、制約を陽に扱うことができません。また、数歩先の未来に起こる (であろう) 振る舞いを予測しながら制御することで、性能を改善できる点も、MPC の利点の 1 つです。これが可能なのは、MPC が制御対象となるシステムの内部予測モデルを使用しているためです。さらに、MPC では経済、制御、安全などを考慮して、複数の目的を最適化することができます。

では、MPC にこれだけの利点があるのなら、PID のような従来の手法を使う理由はあるのでしょうか。MPC は PID とは異なり、制御帯域の広いフィードバック制御のアプリケーション、つまり制御ループの応答時間が短いアプリケーションへの適用には課題があります。MPC はオンラインで最適化問題を解くため、かなりの計算能力とメモリを必要とします。具体的には、MPC の仕組みは以下の通りです (図 1)。

1. 各タイムステップにおいて、ソルバーは制約付き最適化問題を解きます。プラントの状態、出力、操作量 (MV) の制約やプラントのダイナミクスを考慮して、制御要求 (軌道追従など) を表すコスト関数を最小化します。最適化問題の性質 (パネル 1) に応じて、MPC は線形と非線形に分けられます。線形 MPC では、出力や状態、操作量に対して、コスト関数は 2 次形式であり、内部予測モデルと制約は線形となります。非線形 MPC では、コスト関数、内部予測モデル、制約条件などが非線形となります。
2. 最適化により、指定された予測区間に渡り MV 動作の時系列が解として計算されます。これは開ループ制御としてみなされます (パネル 2)。
3. コントローラーは、この時系列の 1 ステップ目の MV 動作のみをシステムに適用し、残りは使用しません (パネル 3)。MPC が、フィードバック制御とみなされるのは、最適な制御動作を計算するためにプラントの現在の状態が必要となるからです。状態は、常に測定、あるいは、カルマンフィルターなどの状態推定器によって適切に推定します。
4. その後、次の時間ステップをシフトし、現在の状態の情報を取得します (パネル 4)。これらの手順を繰り返します。

MPC は、時間経過につれて予測区間の端が徐々に後退しているように見えるため、Receding Horizon 制御とも呼ばれています。

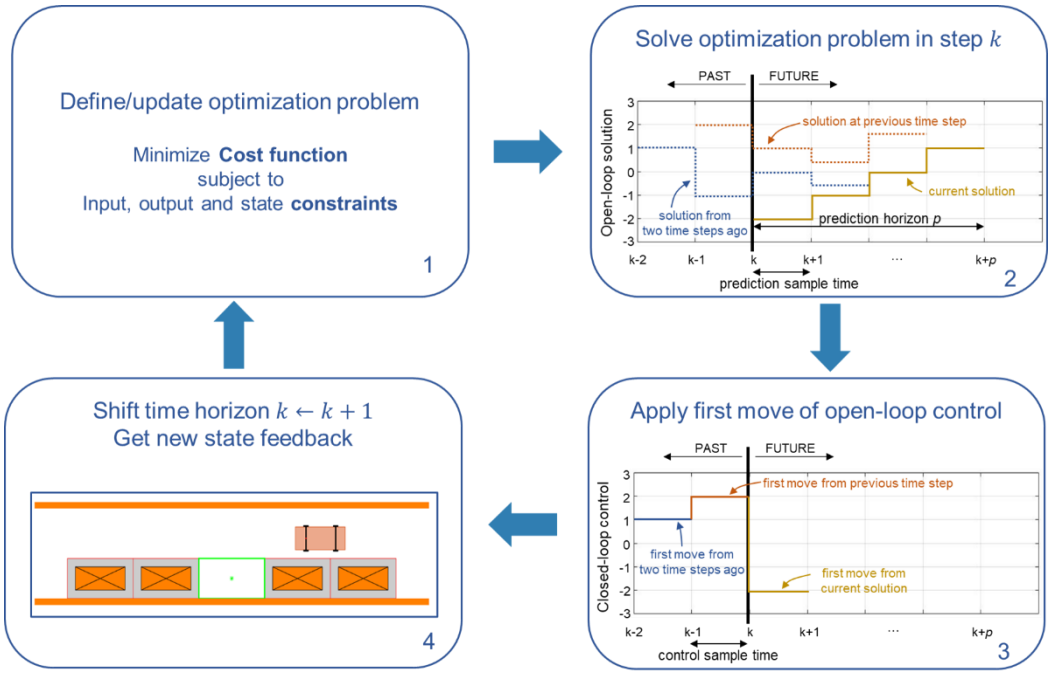

図 1.MPC の仕組み

MPC (線形または非線形) は設計する上で多くの選択肢があります。それらは最適化問題の複雑度に影響するため、結果的にコントローラーの実行速度にも影響します。共通の課題としては、たとえば、プラントのダイナミクスや制御目的に応じた適切な MPC のタイプ、構造、モデル複雑度の選択、あるいは、ハードウェア/ソフトウェアの制限に基づく適切な MPC の設定などが挙げられます。このホワイトペーパーでは、線形および非線形 MPC コントローラーの実行速度に影響を与える設計上の選択肢をまとめ、Model Predictive Control Toolbox™ を 使用して MPC コントローラーを高速実行するためのヒントを示すことを目的としています。以降のセクションでは、特に言及しない限り、MPC をフィードバック (閉ループ) 制御に適用することに焦点を当てていきます。フィードバック制御が実行速度向上の大きな恩恵を受けるためです。実際、Model Predictive Control Toolbox は、開ループのアプリケーション (軌道最適化技術など) にも使用することができます。軌道最適化にも、このホワイトペーパーで説明するトピックの多くを適用することができますが、軌道最適化は、通常オフラインで行われるか、あるいは遅いレートで実行されるため、フィードバック制御問題ほど速度が重視されない場合がほとんどです。

以降のセクションでは、図 2 のように構成され、MPC コントローラーを高速化するための以下の 3 つの方法について説明していきます。

- MPC 問題に対して適切なパラメーター値を選択する。
- 適切なソルバーとソルバーオプションを選択する。
- 手法を変更する。

参考として、MPC コントローラーのメモリ使用量を小さくするためのガイドラインも掲載します。

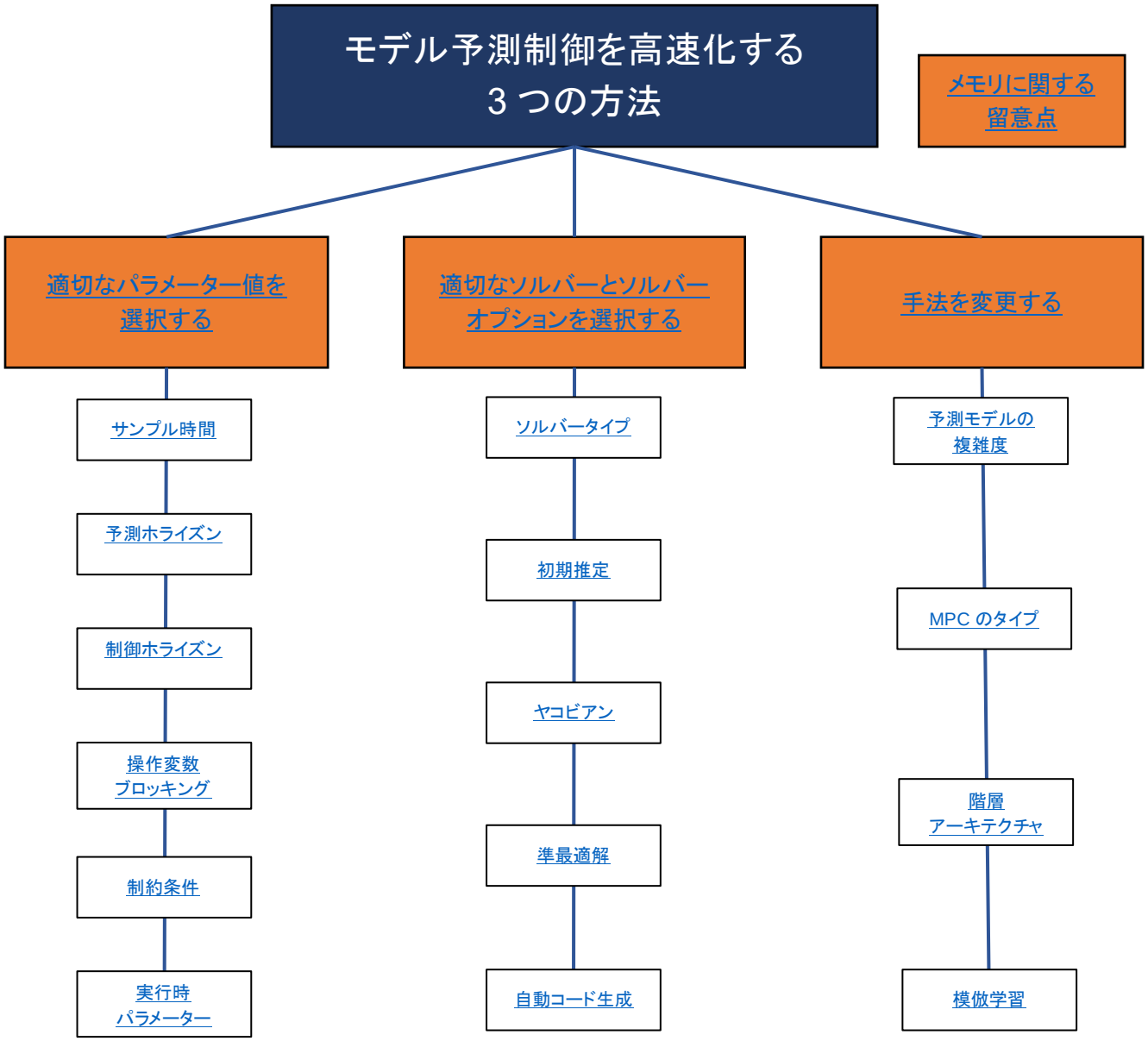

図 2.モデル予測制御の高速化手法
トピックを選択すると、ホワイトペーパーの該当セクションにジャンプします。

MPC 問題に対して適切なパラメーター値を選択する

線形 MPC、非線形 MPC にかかわらず、調整が必要な設計パラメーターは実質的に同じです。このセクションでは、MPC 問題の主要なパラメーターを取り上げ、それらが MPC 最適化問題の計算量にどのように影響するかを説明します。

サンプル時間

サンプル時間はモデル予測制御の重要な概念であり、予測サンプル時間と制御サンプル時間の 2 つに分けることができます。MPC 問題を設計する際に、予測サンプル時間と制御サンプル時間が等しくなるように設定し、1 つのパラメーターとして扱うことが多いのですが、この 2 つを区別し、性能にどのような影響を与えるかを知ることは重要です。

予測 *Ts*

予測 *Ts* は、内部予測モデルのサンプル時間です。これは各予測ステップのサンプル時間を定義し、各予測ステップの間、すべての MV 値は一定に保たれます (図 1 のパネル 2)。直感的には、予測サンプル時間が、達成可能な制御帯域幅の上限を決定します。予測ホライズン (次のセクションで説明します) と予測サンプル時間の積が、コントローラーの予測区間となります。つまり、コントローラーがどれだけ先の未来まで予測しているかを表します。予測区間は制御性能に影響するため、予測サンプル時間と予測ホライズンは共に選定されます。

予測 *Ts* の上限は、プラントのダイナミクスと制御応答時間によって決定されます。たとえば、予測 *Ts* が大きい場合、開ループの不安定なプラントを安定化させるのに十分な制御帯域幅が得られない場合があります。一方、予測 *Ts* が小さくなると、予測区間を一定に保つために予測ホライズンを長くしなければなりません。ただし、予測ホライズンのセクションで説明しているとおり、予測ホライズンが大きくなると最適化問題の決定変数や制約条件が増えるため、解くべき問題が大きく (メモリ使用量が増加します) 複雑になります。経験則として、開ループステップ応答の立ち上がり時間内で MV 動作が 10 ～ 20 ステップとなるように調整します。

制御 *Ts*

制御 *Ts* は、MPC コントローラーのサンプル時間を決定します。つまり、MPC 最適化問題が実行時にどのくらいの頻度で解かれるかを定義します (図 1 のパネル 3)。制御サンプル時間は通常、予測サンプル時間と等しくなるように選択しますが、より速くなるように設定することもできます (遅くはできません)。より小さな制御 Ts を設定すると、一般的に性能 (帯域幅) とロバスト性 (ゲイン余裕や位相余裕) がある程度向上します。また、制御 *Ts* が小さくなるにつれて、内部予測モデルと実際のプラントとのモデル化誤差も含め、未知な外乱に対する抑制効果が向上し、あるところで頭打ちになるのが一般的です。定性的には、環境の変化に対してコントローラーがより速く反応できるようになるため理にかなっているといえます。性能が上限に達する制御サンプル時間の値は、一般的にプラントの動特性に依存します。たとえば、ダイナミクスが遅いプロセスでは、モーター制御のような高速なアプリケーションとは異なり、制御サンプル時間を小さくしてもそれほどメリットはありません。

> 「私たちは、高速かつロバストに製品開発のプロトタイピングが行えるソリューションを探していました。コントローラー開発とコード生成には Simulink を採用し、開発作業の自動化には MATLAB を利用することを決めました。」
>
> *— Alan Mond, Voyage*

ただし、制御サンプル時間が小さくなると、MPC 最適化問題をより頻繁に解くことになるため、計算量が大幅に増加します。このため、性能と計算量のバランスをとることが重要となります。制御サンプル時間を決定するために、最初に、妥当な性能が得られるようなアグレッシブ度の低いコントローラー (制御 Ts を大きめにとり、予測 Ts と等しくする) をテストします。次に、制御 *Ts* を小さくし、コントローラーの実行時間をモニターします。リアルタイムアプリケーションの場合、通常のプラント動作条件のもとで、各サンプリング周期内に最適化が確実に完了するのであれば、制御 Ts をさらに小さくすることができます。詳細については次のセクションで説明しますが、最適解を見つけるために必要なソルバーの反復回数を予測する方法がないため、最適化問題の準最適解を使用することで、制御 Ts の調整が容易になります。MPC コントローラーが達成できる最小の制御 Ts はシステムに依存し、(リアルタイム) ハードウェアやシミュレーションプラットフォームによって異なることに留意してください。

ヒント

計算回数を小さくするために、予測 Ts と制御 Ts は、制御要求を満たすように十分に小さく(ただし、小さすぎないように)選択します。

MPC を高速に実行させるために、予測 Ts が制御 Ts より大きくなるように選択することを検討します。これによって、最適化問題を適切なサイズに抑え、制御サンプル時間よりも速く解けるようにします。

関連情報

サンプル時間の選び方

リアルタイム環境でのサンプリングレート

予測ホライズン

予測ホライズン p は、MV を最適化する際に、MPC コントローラーが (内部プラントモデルを使用して) 予測する未来の制御インターバルのステップ数です。各制御インターバルの長さは、予測サンプル時間によって決まります。上述のサンプル時間の説明と同様に、予測ホライズンの選択はプラントダイナミクスの特性に依存します。*p* の選択方法に関する主な指針は、対象となるシステムの予測区間 (p と予測 Ts の積) の要件を満たすことです。通常、遅いダイナミクスのシステムでは、操作変数 MV がコスト関数や出力に与える影響を MPC コントローラーが十分に予測できるように、長い予測区間を必要とします。このように、予測ホライズン *p* と予測 *Ts* の値は、ある意味密接に関係しています。

しかし、*p* の値が大きくなると，最適化問題の決定変数や制約条件が多くなり (最適化の制約条件は制御インターバルごとに満たす必要がある)、問題のサイズが大きくなります。また、MPC 最適化問題の多くの行列の次元は p に比例し (詳細については表を参照)、実行時間が長くなり、メモリ使用量も増

加します。線形 MPC を例にとると、MV のみを最適化する場合 (状態や出力も含めて最適化する場合とは異なり)、決定変数の総数は p と MV の数の積となり、(p と MV の数の積) × (p と MV の数の積) ヘシアン行列が導出されます。同様に、MV 制約の数は、*2p と MV の数の積*です。一般的に、予測ホライズンを調整してコントローラーを調整することはそれほどありません。*p と Ts の積*は、コントローラーを内部安定にし、制約違反や環境変化を早期に予測して修正動作が取れるような値にする必要があります。これは、*p と Ts の積*を開ループステップ応答の立ち上がり時間、あるいは、過渡応答時間をカバーするように選択することで、達成されることがあります。もしプラントが開ループ不安定な場合、プラントの開ループステップ応答が発散するため、p をあまり大きくとることはできません。

例を用いた予測ホライズン、制御 Ts、予測 Ts の比較

望ましい予測区間 (*p* と予測 *Ts* の積) が 1 秒で、特定のハードウェア上で単一の MPC 最適化問題を解くための平均実行時間が (実験から) 分かっていると仮定します。

ケース 1: *p*=10 で、実行時間が 1 ms であることが分かっているとします。予測区間が 1 秒の場合、予測 *Ts* は 100 ms になります。予測 *Ts* は総実行時間よりも大きいので、コントローラーはオーバーランすることなくハードウェア上で実行することができます。性能を改善するために、1 kHz 未満であれば、制御 *Ts* を予測 *Ts* よりも高速にすることができます。

ケース 2: *p*=25 で、実行時間が 40 ms であることが分かっているとします。予測区間が 1 秒の場合、予測 *Ts* は 40 ms になります。予測 *Ts* は総実行時間に等しいので、これがハードウェア上でコントローラーをオーバーランすることなく実行させるために使用できる最小の予測 *Ts* (または最大の予測ホライズン *p*) となります。制御 *Ts* も 40 ms より小さくすることはできません。

上記のいずれのケースも良い性能が得られる場合、どちらを選択すべきでしょうか。ケースによっても異なりますが、*p* 値が大きいほどメモリ使用量が大きくなるため、メモリ容量が限られているハードウェアでは、ケース 1 の方が望ましいでしょう。

ヒント

予測ホライズンが大きくなると、コントローラーは将来のイベントをより正確に予測し、計画することができますが、コントローラーの計算時間とメモリ使用量が増加します。

関連情報

モデル予測コントローラーの設計方法 (3:00)

制御ホライズン

制御ホライズン *m* は、制御インターバル k で最適化される MV 動作ステップの数であり、1 と予測ホライズン p の間の値をとります (Model Predictive Control Toolbox では、既定の制御ホライズンの値として m = 2 を使用します)。操作変数ごとに、制御ホライズンの各ステップで MPC 解 (開ループ制御) の値が決定されます (図 3)。すなわち、制御入力の数や制御ホライズンの値に応じて、ソルバーで最適化する変数の数が増加します。たとえば、MV のみが決定変数である場合 (最適化ソルバーセクション参照)、最適化変数の数は *m と MV の数の積*となります。状態も決定変数として含まれる場合 (スパ

ースな問題の定式化など)、最適化変数の数は、m と MV の数の積と p と状態の数の積の和となります。MPC では、*m* をどのように選択しても、コントローラーの実際の動作には、MPC 解の 1 ステップ目の MV 動作のみを使用し (つまり、ホライズンの最初のみ) 、それ以外は破棄されます。実際に、制御ホライズンが大きいほど、最適化問題を解くのに時間がかかり、コントローラーが次のインターバルに移るときに廃棄される情報が多くなることになります。

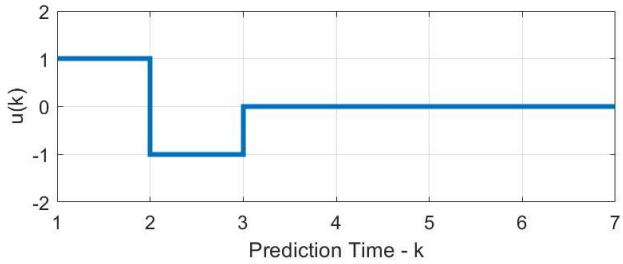

図 3.m=2、p=7 の場合のインターバル k における MPC 解 (開ループ制御) の例

ヒント

m<<*p* と選択すると、各制御インターバルで最適化する変数が少なくなるため、計算の高速化が促進されます。

関連情報

予測ホライズンの選び方

制御ホライズンの選び方

操作変数ブロッキング

*操作変数ブロッキング*は、制御ホライズンの概念に代わるもので、多くのメリットがあります。操作変数ブロッキングは、制御ホライズンをスカラー値で指定する代わりにブロックサイズのベクトルで指定することで、予測ホライズンを一連のブロック間隔に分割します。ブロックサイズの合計は、予測ホライズン *p* と一致しなければなりません。これにより、望ましい MV 動作ステップの数だけでなく、各 MV 動作ステップの長さ (予測サンプル時間の倍数) も指定することができます。図 4 は、制御ホライズン *m*=[2 3 2]、予測ホライズン *p*=7 の場合の操作変数ブロッキングの例を示しています。

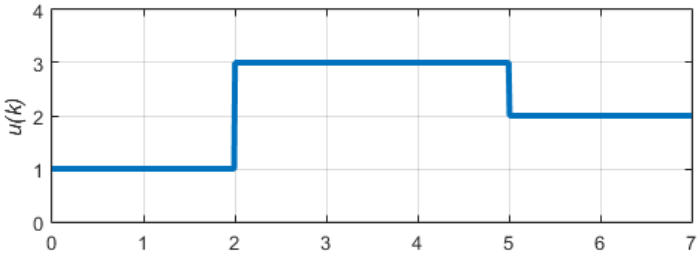

図 4.m=[2 3 2]、p=7 の場合の間隔 k における MPC 解

ヒント

制御ホライズンの概念と同様に、操作変数ブロッキングでは、*各制御インターバルにおける決定変数の数を指定することができるため、計算の高速化につながります。*

関連情報

操作変数ブロッキング

制約条件

先に述べたように、MPC は各タイムステップで制約付き最適化問題を解きます。一般的に、制約条件の数が多いほど、問題の複雑さが増し、解くのに時間がかかります。制約条件は、許容可能な制御の空間を制限するため、実行不可能問題 (解が存在しない問題) に遭遇するリスクを高めます。制約条件の数を必要最小限にすることで、最適化問題をより簡単に解くことができます。

Model Predictive Control Toolbox では、ハード制約とソフト制約の両方を定義して、最適化問題を設定することができます。ハード制約は確実に満たされなければなりませんが、ソフト制約は必要に応じて違反することも許容されます。ハード制約は通常、ソフト制約とは異なり、許容解の空間を制限するため、最適化プロセスの収束に時間がかかります。制約が MV の範囲のみであれば、それをハード制約にすることができます。MV の範囲のみであれば、実行不可能な問題は発生しません。制約が MV 変化量の範囲のみの場合も同様です。ただし、MV の範囲と MV 変化量の範囲を同時にハード制約にすると、制約過多の問題となり、実行不可能になる可能性があります。プラントが外乱の影響を受けやすく、出力のハード制約、あるいは、入出力混在のハード制約を含む場合も同様です。原則として、可能な限りソフト制約を選びます。

ヒント

制約の数は少なくし、可能な限りハード制約ではなくソフト制約を選択します。

関連情報

線形 MPC の制約

非線形 MPC の制約

終端制約

時変制約

実行時パラメーター

Model Predictive Control Toolbox では、MPC 問題のある特定の設計パラメーターを実行時に変化させることができますたとえば、コスト関数の重みを調整するために、コントローラーを動作させながら、各予測ステップに対して重みを変化させて、オフラインまたはオンラインで修正することができます。同様に、試作時に制御ホライズンや予測ホライズンを調整したり、運用時にプラントのダイナミクスが変化した場合に、これらのパラメーターをオンラインで調整したりすることもできます。また、動作条件の変化に対応するために、たとえば、制約条件の上下限値をさせることも可能です。実行時のパラメーター調整は状況によっては有用ですが、複雑度が増し、必要な計算量が増える可能性があります。したがって、性能と計算量のバランスをとることが重要となります。

ヒント

各タイムステップで必要な計算回数を抑制するために、実行時パラメーターの変更回数を少なくします。

関連情報

実行時の重みの調整

実行時の制約の更新

実行時のホライズンの調整

非線形 MPC の制約の指定

適切なソルバーとソルバーオプションを選択する

MPC のように数値最適化に大きく依存する手法では、適切にソルバーとソルバー設定を選択することが性能に大きく影響します。たとえ同じ MPC パラメーターのセットであっても、より良いソルバーを選ぶことで、より速く解が得られる可能性があります。このセクションでは、Model Predictive Control Toolbox で利用できるさまざまなソルバーオプションと、ソルバーの選択に関わらず、より速く解を得るためのヒントを紹介します。

最適化ソルバー

MPC を適用するアプリケーションでは、リアルタイムに最適化を実行するために、品質の高いソルバーが求められます。一般的な性能要件には、実行時間、メモリ使用量、精度、ロバスト性などがあります。適切なソルバーを選択することで、MPC の他のパラメーターを変更しなくても、実行時間を大幅に短縮できる可能性があります。

ソルバーの中には、制御サンプル時間が速く、メモリ容量が限られたリアルタイム アプリケーション向けに特化して設計されたものがあります。これらは「組み込み」ソルバーと呼ばれ、大きく以下の 2 つに分類されます。

- 使用するアルゴリズム (例: 有効制約法、内点法、拡張ラグランジュ関数)
- 行列のスパース性 (例: 密、疎)

Model Predictive Control Toolbox では、ビルトインのソルバーやサードパーティーツールとの連携により、密(非スパース)や疎(スパース)の定式化による有効制約法や内点法を利用できます(表 1)。ソルバーのアルゴリズムや定式化方法で問題になるケースはそれほどありませんが、高速な MPC アプリケーションでは、これらの選択がリアルタイム性能の決め手になる可能性があります。

有効制約法アルゴリズム。有効制約法は、小規模や中規模の最適化問題に対して、高速かつロバストな性能を発揮することができます。有効制約法ソルバーは、制約条件の数が増えると著しく速度が低下します (長い予測ホライズンに制約条件が適用される MPC 問題を想像してください)。そのため、最悪実行時間では、ハードウェア上でのタスクオーバーランを生じやすくなります。

内点法アルゴリズム。内点法は、長い予測ホライズンや制御ホライズンに制約が課されるような、大規模な最適化問題に対して優れた性能を発揮します。また、内点法が収束するのに必要な反復回数は、おおよそ均一です (つまり、制約条件の数に依存しません)。そのため、組み込みシステムでの最悪実行時間をより簡単に推測することができます。

密な問題。MPC の観点では、決定変数が操作変数 MV のみの場合、密な最適化問題となります。密な定式化は、スパースな定式化に比べて決定変数が少なくなり、特に操作変数ブロッキングや小さな制御ホライズンを使用する場合には効率的となります。ただし、内部プラントモデルが不安定の場合、密な問題の行列は、予測ホライズンが大きくなるにつれて特異行列に近づく可能性があります。

スパースな問題。MPC の観点では、操作変数、状態、および出力が決定変数となる場合、最適化問題はスパースとなります。スパースな問題は、密な問題と比べて決定変数の数が多くありますが、ゼロの要素を多く備えた対角行列に近い行列が生成されます。このような行列のスパース構造を使用することで、計算時間とソルバーのメモリ使用量を削減することができます。密な定式化とは異なり、スパースな問題では、特に内部プラントモデルが不安定な場合に、より適切な条件数の行列が生成されます。

表 1.Model Predictive Control Toolbox のソルバーオプション

	ビルトインのソルバー	サードパーティ製のソルバーオプション

線形	• 有効制約法ソルバー – KWIK (密)
	• 内点法ソルバー (密)
		• カスタムソルバー
		  ○ カスタム実装
		  ○ サードパーティソルバー
		• FORCES PRO プラグイン (FORCES PRO ライセンスが必要)
		  ○ 内点法ソルバー (スパース)

非線形	• Optimization Toolbox の SQP アルゴリズムを備えた関数 `fmincon` (スパース、有効制約法に類似)	• カスタムソルバー
		  ○ カスタム実装
		  ○ サードパーティソルバー
		• FORCES PRO プラグイン (FORCES PRO ライセンスが必要)
		  ○ 内点法ソルバー (密、スパース)
		  ○ SQP ソルバー (密、スパース)

表 1 に Model Predictive Control Toolbox で使用できるソルバーオプションをまとめました。線形 MPC 向けに、このツールボックスでは「密な」2 つのビルトインの QP ソルバーをサポートしています。KWIK アルゴリズムを使用した有効制約法ソルバーと、Mehrotra 型予測子修正子法による主双対アルゴリズムを使用した内点法ソルバーです. 一方、非線形 MPC の最適化問題を解くために、Model Predictive Control Toolbox では Optimization Toolbox™ を使用します。具体的には、SQP アルゴリズムを備えた関数 `fmincon` を使用します。また、(線形または非線形の) MPC コントローラーにカスタムソルバーを指定することもできます。カスタムソルバーは、制御インターバルごとにビルトインソルバーの代わりに呼び出されます。さらに、Embotech 社が開発したリアルタイムの組み込み最適化ソフトウェアツールである FORCES PRO を使用して、Model Predictive Control Toolbox で設計された MPC コントローラーをシミュレーションし、自動コード生成できるようになりました。FORCES PRO ソルバーと Model Predictive Control Toolbox との併用については、線形 MPC および 非線形 MPC に関する FORCES PRO のドキュメンテーションを参照してください。

まとめると、アプリケーションに応じてソルバーを選択・設定する際に、以下の点を考慮する必要があります。

- 操作変数、レート、出力、制約条件の総数が多い場合 (つまり、予測ホライズンや制御ホライズンが大きく、数百を超える場合) は、内点法ソルバーの使用を検討します。
- 内部プラントが開ループ不安定な場合は、スパースな問題の定式化を検討します。
- さもなければ、密な有効制約法ソルバーが実用的な選択肢となります。

さまざまなソルバーを使用して複数のシナリオでコントローラーをシミュレーションすることで、アプリケーションに適したソルバーを検討することができます。

> 「従来のアプローチでは、MPC のような複雑なコントローラーの開発には約 1 年を要したでしょう。モデルベースデザインでは、約 6 か月でプロトタイプの開発ができました。」
>
> 「QP ソルバー用に生成されたコードは非常に効率的だったので、他のソルバーを探し求める必要はありませんでした。Embedded Coder を使用したので、コントローラーの機能を確認した後は、組み込みプロセッサに実装するのにほとんど時間がかかりませんでした。」
>
> *— Taku Takahama, Hitachi Automotive Systems*

ヒント

MPC 問題のサイズや構成は、利用できるソルバーの相対的な性能に影響を与えます。さまざまなソルバーを使用して複数のシナリオでコントローラーをシミュレーションすることで、アプリケーションに適したソルバーを検討することができます。

関連情報

QP ビルトインソルバーとカスタムソルバー

Optimization Toolbox の制約付き非線形最適化アルゴリズム

カスタムソルバーを用いた非線形 MPC による結核治療の最適化

Model Predictive Control Toolbox 向け FORCES PRO プラグイン

初期推定

数値最適化では、決定変数の初期値は、最適化アルゴリズムが探索のコースを決める上で大きな役割を果たします。たとえば、適切に構成された標準的な線形 MPC 最適化問題は一意解を持ちますが、非線形 MPC 最適化問題では複数解 (局所最適解) を持つことがあります。このような場合では、可能な限り (大域的な) 最適解の近くによい開始点を指定することが重要です。

MPC をフィードバック制御 (閉ループ制御) に使用する場合は、(非線形) ソルバーをウォームスタートさせるのが最善の方法です。そのためには、前回の制御インターバルで予測された状態と操作変数の最適解を、現在の制御インターバルの初期推定値として使用します。このアプローチの根拠は、一連の制御インターバルの間で解がそれほど大きく変化することがないことに基づきます。これにより、よい初期探索方向が与えられ、ソルバーによる計算リソースの浪費を回避できるため、最適化を高速化します。最初の制御インターバルでは、初期推定として使用するための過去の解がないため、特に有効制約法ソルバーでは、初期推定をランダムに設定するよりも、直線などのシンプルな解軌跡を選択する方が効率的であると考えられています。

ヒント

フィードバック制御では、特に非線形 MPC の場合、実行可能な推定値を使用してソルバーをウォームスタートさせるのが最善の方法です。

関連情報

非線形 MPC での初期推定

線形 MPC での初期推定

解析ヤコビアン (非線形 MPC)

線形 MPC とは異なり、非線形 MPC では、プラントのダイナミクスや制約条件が非線形になります。さらに、コスト関数は、決定変数に関する非二次 関数になりえます。前述の通り、非線形 MPC における最適化問題を解くために、Model Predictive Control Toolbox では Optimization Toolbox を使用し、具体的には SQP アルゴリズムを備えた関数 `fmincon` をソルバーとして使用します。SQP 法は、MPC の各タイムステップにおいて、(プラントのダイナミクスを含む) 線形化された制約条件と二次のコスト関数による部分問題を解きます。

SQP ソルバーを使用する際に、プラント、(カスタム) 制約条件、あるいは、(カスタム) コスト関数の解析的なヤコビアンを指定しなければ、コントローラーは数値摂動法を用いてヤコビアンを数値的に自動計算します。その場合、各タイムステップで複数回ヤコビアンを計算するため、処理時間がかかります。計算効率を上げるためには、ヤコビアンを解析的に指定するのが最善の方法です。Symbolic Math Toolbox™ を使用すれば、記号変数を使用した数式からヤコビアンを自動的に計算することができるので、面倒な手動計算をする必要がありません。以下のリンク先のクアッドローターの例で、この方法を紹介しています。厳密なヤコビアンを計算するのが難しい場合は、近似値の利用でも十分かもしれません。非線形 MPC による縦列駐車に近似ヤコビアンを使用する方法をご覧ください。ヤコビアンは最適化の探索方向を指定するために使用されるため、近似ヤコビアンであっても最適化を正しい方向に導くことができます。

また、自動微分を用いて必要なヤコビアンを計算する方法もあります。自動微分 (autodiff)は、アルゴリズムに従って微分を数値的に計算する一連の手法です。解析的にヤコビアンを与えるほど効率的ではありませんが、autodiff は数値摂動法よりも効率的で、実行時間を短縮することができます。たとえば、Model Predictive Control Toolbox をサードパーティ製の autodiff ツールと併用する、あるいは、自動微分を使用した FORCES PRO プラグイン (表 1) を使用することができます。

ヒント

解析ヤコビアンと自動微分により、非線形 MPC における最適化の計算時間を短縮します。

関連情報

コスト関数のヤコビアン

カスタム制約のヤコビアン

予測モデルのヤコビアン

非線形モデル予測制御を用いたクアッドローターの制御

準最適 MPC 解

(線形または非線形の) 制約条件のある MPC アプリケーションに対して、最適解を探索するために必要なソルバーの反復回数を予測する方法はありません。また、リアルタイムアプリケーションでは、使用するソルバーアルゴリズム (有効制約法など) に応じて、制御インターバルごとに反復回数が大きく変化することがあります。このような場合、最悪実行時間が、ハードウェアプラットフォームや制御サンプル時間によって決定される制限を超えてしまうことがあります。

最適化の反復回数が指定された最大値を超えてしまう場合に、準最適解を適用することで、MPC コントローラーの最悪実行時間を保証することができます。最悪実行時間を設定するには、最初にコントローラーをノミナル条件で実験して、最適化を一度反復するのに必要な時間を決定します。次に、制御インターバルごとの反復回数に上限を設定します。たとえば、ハードウェアの各反復計算に約 1 ms かかり、制御サンプル時間が 10 ms の場合、最大反復数を 10 以下に設定します。先に述べたように、内点法ソルバーの場合、収束までの反復回数はほぼ一定であるため、最悪実行時間を簡単に推定することができます。

最終反復後にたどり着く解は最適解ではありませんが、指定されたすべての制約は満足されます。

ヒント

最適化の反復回数が指定された最大値を超えてしまう場合、準最適解を適用することで、MPC コントローラーの最悪実行時間を保証することができます。

関連情報

高速な MPC アプリケーションでの準最適解の使用

非線形 MPC を用いたロボット マニピュレーター計画への準最適解の使用

自動コード生成

最適化問題やソルバーのパラメーターを調整することなく、MPC のシミュレーションを高速化する最も負担が少ない方法は、自動コード生成です。Model Predictive Control Toolbox のすべてのビルトインソルバーは、コード生成をサポートしています。MATLAB® (MATLAB Coder™ を使用) または Simulink® (Simulink Coder™ または Embedded Coder™ を使用) で設計した MPC の C コードを自動生成し、任意の実装ターゲットに展開することができます。

コード生成ターゲットを MATLAB 実行ファイルに設定し、MATLAB で MPC コントローラーの MEX 関数を生成・呼び出すことで、シミュレーション結果の再現や性能の評価が可能になります。この手法は MPC の計算が大幅に高速化されるため、シミュレーション時間が短縮され、設計した MPC コントローラーに対して複数のシミュレーションを実行する場合に役立ちます。コード生成ターゲットを、C のスタティック ライブラリ、ダイナミック ライブラリ、実行ファイルなどに変更することができます。

ヒント

MPC コントローラーの C/C++ コードを自動生成し、シミュレーションや展開を加速することができます。

関連情報

自動コード生成による非線形 MPC を用いた縦列駐車

Simulink Coder を用いたシミュレーションとコード生成

MATLAB でコードを生成して最適な MPC の動作を算出

手法を変更する

ここまでの提案方法で最適化がなかなか速くならない場合は、少し立ち止まって別のアプローチを検討する必要があるかもしれません。たとえば、線形コントローラーと同等の性能でありながら高級な非線形 MPC コントローラーを使用する意味はあまりありません。このセクションでは、モデル低次元化、さまざまな MPC 手法の複雑度、また MPC 問題をリアルタイムで解くことを避けるために利用できる機械学習法である模倣学習について説明します。

予測モデルの複雑度

前述のとおり、MPC は物理的なプラントの内部予測モデルに基づいて動作します。予測モデルを用いて、システムのダイナミクスを考慮して数タイムステップ先までの計画を立てる、非線形 MPC のために線形化する、あるいは、状態推定に利用することができます。内部の MPC モデルで冗長な状態が 1 つでもあると、制約、微分計算、積分、あるいは、出力や推定状態が増え、結果的に計算のオーバーヘッドが増大します。操作変数を 1 つ追加すると、最適化変数が m (制御ホライズン) 追加されます。このように、予測モデルの複雑度、特に操作変数の数、状態の数、出力の数が、計算や全体の最適化速度に影響を与えます。

一般的に、操作変数や出力の数は固定であり、問題設定や制御目標によって決まります。ただし、たとえば、プラントの中に非干渉化された入出力を含む場合は、これらの入出力に対して個別の制御ループを設計し、MPC 問題の出力数と最適化変数を減らすことも検討可能です。また、制御エンジニアにはモデルの状態数を減らすための柔軟な方法と選択肢があります。選択肢の 1 つは、ダイナミクスに大きく寄与しない状態を削減するようなモデル低次元化の手法を使用することです。Control System Toolbox™ を使用したモデルの低次元化手法の詳細については、こちらを参照してください。

単純に詳細度の低いモデルを使用するという方法もあります。MPC は、制御目標を達成するために不可欠なダイナミクスを内部プラントがうまく捉えていれば、モデル化誤差の補償は状態フィードバックの能力に頼ることができます。対象となるシステムのダイナミクスをどこまで捉える必要があるかを検討することが重要です。たとえば、図 5 では、ロボットマニピュレーターが衝突回避するように、MPC コントローラーを用いてジョイントの軌道計画を行った例を示しています。このシナリオでは、すべてのジョイントをシンプルな二重積分器としてモデル化するだけで、この問題を解決することができます。

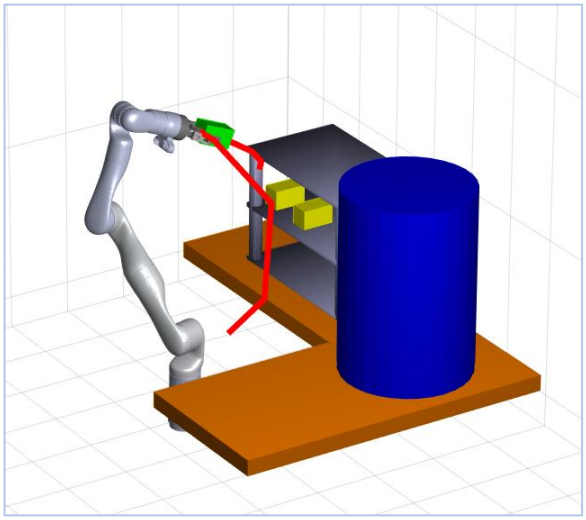

図 5.MPC によるピックアンドプレース アプリケーション向け衝突回避の軌道計画

ヒント

MPC 最適化問題を簡略化するために、内部プラントモデルの入力/操作変数、状態、および出力の数をできるだけ減らします。

関連情報

モデルの低次元化手法

非線形 MPC を用いたロボット マニピュレーターのピックアンドプレース・ワークフロー

MPC のタイプ

図 6 では、Model Predictive Control Toolbox でサポートされる MPC コントローラーのタイプを示しています。問題に応じて、以下のように適切な MPC 手法を選択することで、実行速度を大幅に向上させることができます。

- 線形時不変 (LTI) MPC では、内部プラントは線形で、時間ステップや予測ホライズンに渡って一定となります。
- 適応 MPC コントローラーは、内部プラントが時間ステップごとに更新されるため (ただし、予測ホライズンに渡って変化することはありません)、LTI MPC よりも実行時の計算量が必要になります。また、適応 MPC では、モデルを更新する処理も実装する必要があり、計算量が多くなる可能性があります。
- 線形時変 (LTV) MPC は、適応 MPC に似ていますが、各時間ステップと予測ホライズンに渡って内部プラントを更新します。
- 陽的 MPC は、オンラインで計算を行う従来の手法とは異なり、オフラインでの計算により、状態空間を分割し、各領域に線形アフィンな状態フィードバック制御則を割り当てます。これらの事前に計算された制御則が保存され、実行時には最適化問題を解く代わりに使用されます。
- ゲインスケジュール MPC は、事前に用意された複数の LTI MPC コントローラーあるいは陽的 MPC コントローラーを状況に応じて切り替え、幅広い動作条件に対応しながら非線形プラントを制御します。ゲインスケジュール MPC を実現するためには、まず、各動作点に対して LTI MPC あるいは陽的 MPC コントローラーを設計し、次に、実行時にコントローラーを切り替えるスケジューリング信号を設計する必要があります。
- 非線形 MPC は、非線形の制約条件やプラント、場合によっては二次ではないコスト関数を扱うことができます。

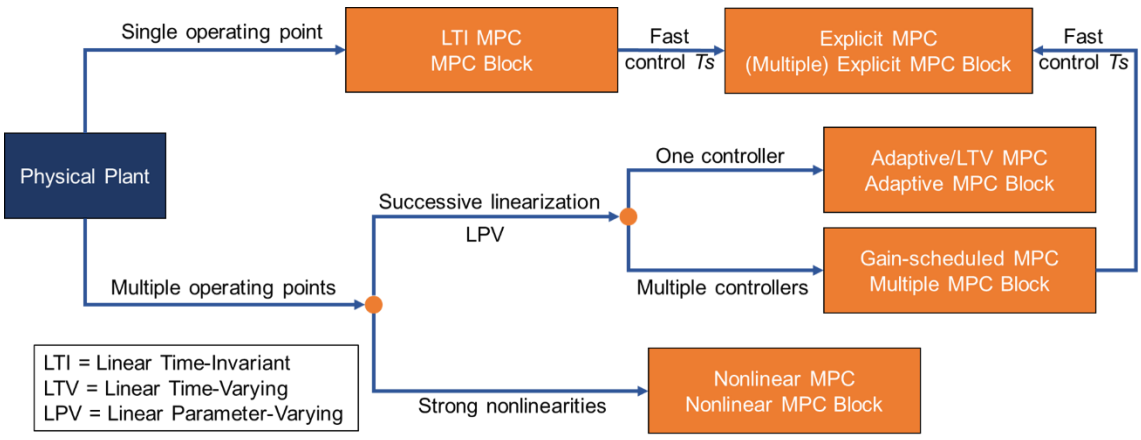

図 6.Model Predictive Control Toolbox がサポートする MPC のタイプ

図 7 では、では、さまざまな MPC 手法を実行速度とメモリの観点で比較する方法を示しています。陽的 MPC は最高速で実行することを可能にしますが、メモリ使用量が最も大きく、また他の手法と比較して機能が限定されています。非線形 MPC は、MIMO システム向けに最も高性能で汎用的な手法です。ただし、これは最も計算コストの高い手法でもあり、メモリ使用量は陽的 MPC に次いで大きくなります。ゲインスケジュール (陽的) MPC のメモリ要件は、設計されたコントローラーの数に依存し、これは図 7 にも示されています。

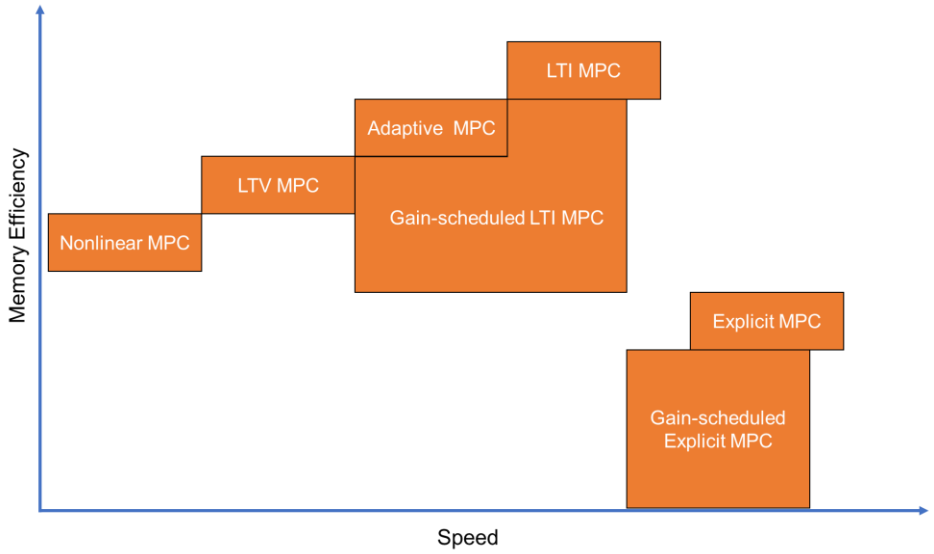

図 7.Model Predictive Control Toolbox における
さまざまな MPC 手法の実行速度とメモリ効率の比較

最速のランタイム実行を実現するには、許容範囲の性能が得られる最もシンプルな MPC 手法を使用します。たとえば、次のようなものがあります。

- 線形または線形化されたプラントモデルに対しては、LTI MPC から開始して、性能が期待どおりになるまで、必要に応じてゲインスケジュール、適応、または LTV MPC に徐々に移行していきます。
- 一般的には、可能な限り適応および LTV MPC をゲインスケジュール MPC よりも優先します。これは、適応および LTV MPC の手法では、設計が必要なコントローラーは 1 つのみであり、スケジューリングによる切り替えの影響を受けにくいためです。
- 高速な MPC アプリケーションでは、可能な限り陽的 MPC の使用を検討してください。
- 物理プラントの非線形性が高い場合は、非線形 MPC コントローラーの設計から始めて、LTV や適応 MPC で同じ性能が得られるかどうかを評価すると良いでしょう。Model Predictive Control Toolbox では、`nlmpc` オブジェクトに RunAsLinearMPC オプションを設定することで、非線形 MPC コントローラーを線形 (適応または時変) として簡単に実行することができます。

ヒント

最速のランタイム実行を実現するには、許容範囲の性能が得られる最もシンプルな MPC 手法を使用します。

階層アーキテクチャ

内部プラントモデルと最適化を用いて予測しながら制御する MPC は、計画問題にも利用することができます。たとえば、MPC を直接ローレベルの制御に使用する代わりに、最適化問題の解を内部プラントモデルに適用して、別のコントローラーが追従するための目標出力軌道の生成に使用することもできます。図 8 に 2 つのアーキテクチャの概要を示します。アーキテクチャ I には、独立したプランナーがないため、ある意味で、計画と制御の両方に MPC を使用しています。アーキテクチャ II は計画と制御を切り離し、目標軌道を生成するプランナー (MPC を使用する/しないのいずれも可能) と、これらの目標軌道に追従するコントローラー (MPC など) を備えています。この 2 つ目のアーキテクチャでは、環境が静的であれば、計画部分はオフラインでも実行可能です。一方、たとえば、移動する障害物を回避しなければならないような場合は、プランナーをループに入れた動的な計画が必要となります。一般にアーキテクチャ II では、外側のループ (計画) は内側のループ (追従制御) ほど高速に実行する必要はありません。

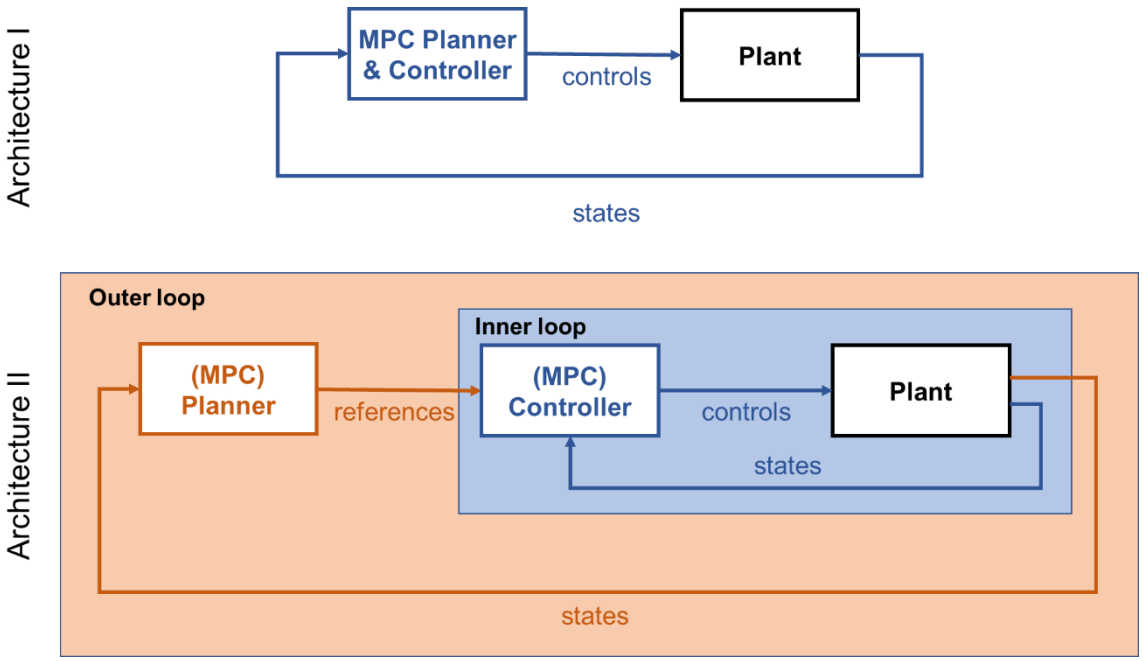

図 8.制御システム アーキテクチャにおけるさまざまな手法

計画とローレベルの制御を分離することで、制御エンジニアは制御システムの設計をより柔軟に行うことができます。たとえば、MPC がアーキテクチャ I に含まれているものの、最適化の速度がリアルタイムのアプリケーションには十分でないケースを想像してみます。アーキテクチャ II を用いると、内側の

ループは引き続き同じ (速さの) サンプルレートで実行しなければなりません。ただし、計画と制御が分離されることで、この場合の MPC の最適化問題はよりシンプルになり、より速い制御レートを実現できる可能性があります。MPC を用いて内側のループの改善が十分でない場合は、外側のループが有効な選択肢となります。前述のとおり、外側のループは内側のループよりも一般的に遅い速度で実行されるため、最適化がリアルタイムの要件を満たしやすくなります。この場合、内側のループでの軌道追従は、PID 制御などで実現することができます。なお、外側のループのプランナーは MPC の代わりに、RRT などの広く利用されているプランナーを使用することもできます。

> 「少人数のエンジニアチームが、既製のハードウェアと、モデルベースデザインにより開発し実装した制御アルゴリズムを使用して、自律型車両を完成させました。このシステムはすぐに量産可能な状態ではありませんが、実用的な設計手法により重要な設計コンセプトを実証することができました。」
>
> — *Dr. Mark Tucker, TMETC*

一例として、Tata Motors European Technical Centre (TMETC) がアーキテクチャ II を使用して、Tata Hexa SUV の自動運転ソフトウェアを開発し展開しました。チームは Model Predictive Control Toolbox を使用して、目標点を追従する横方向および縦方向のコントローラーを開発しました。

ヒント

階層化された 2 段階の制御システム アーキテクチャを使用することで、計画と (ローレベルの) 制御が分離され、MPC をいずれか一方あるいは両方の階層に適用できるように、最適化問題が簡略化される可能性があります。

模倣学習

近年の計算機の処理能力向上により、機械学習モデルの学習時間は大幅に短縮されました。1 つの方法として、機械学習を使用して MPC コントローラーを高速化することもできます。MPC コントローラーは、基本的に状態を入力して操作変数の指令値を出力する関数であり、最適化ベースの計算を行うため、解析的に状態から操作指令を計算するモデルがあるわけではありません。そこで、機械学習モデルを使用して、MPC コントローラーの挙動を近似、あるいは模倣する方法が考えられます。

模倣学習は実質的に回帰問題 (教師あり学習) であるため、学習データ (MPC の入出力) が必要になります。学習時間は、データセットのサイズ、使用するサロゲートモデルの種類、解くべき MPC 問題の複雑度に依存します。模倣学習では、MPC の計算をオフラインで実施して学習データセットを作成します。サロゲートモデルの動作する領域やデータをうまく表現できるように、学習データセットを慎重に構築することが重要です。モデルの性能は学習データに依存します。図 9 は、Model Predictive Control Toolbox を使用して設計した、車線維持支援 (LKA) 用 MPC コントローラーをニューラル ネットワーク近似した場合の性能を示しています。ニューラル ネットワークと元の MPC コントローラーの挙動はほぼ同じです。

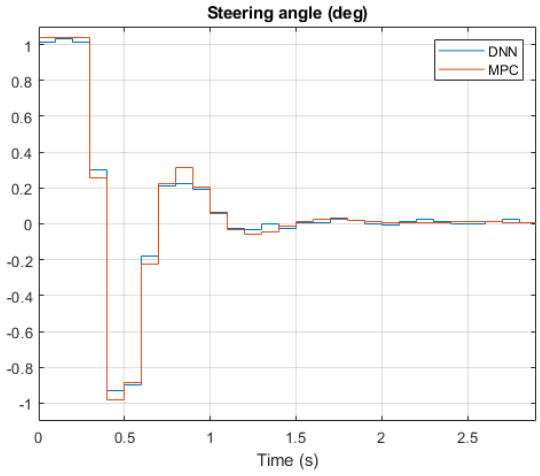

図 9.元の MPC コントローラーとニューラル ネットワーク近似 (DNN) の比較

模倣学習では、対応する最適化問題がオフラインで解かれ、学習済みモデルによるフォワードパスのみで状態から操作指令を計算できるため、MPC の実行速度が向上します。ルックアップテーブルを使用して、分割された状態空間の領域ごとに線形状態フィードバック則を切り替える陽的 MPC と比較して、模倣学習では、サロゲートモデルが状態に対する非線形性を表現できるため、柔軟性がより高くなります。理論上は、区分的に線形な陽的 MPC よりも、単一の非線形の強いサロゲートモデルが MPC コントローラーの応答をより正確に近似できることが期待されます。たとえば、(ディープ) ニューラル ネットワークは、適切な層のアーキテクチャを使用することで、複雑な非線形マッピングを学習できる優れた関数近似器です。また、陽的 MPC とは異なり、模倣学習では状態空間の分割による多面体領域を生成する必要がないため、メモリ使用量を削減できる可能性があります。

機械学習モデルには多くの利点があるにも関わらず、なぜ模倣学習がリアルタイム MPC の標準的な手法ではないのかと疑問に思われるかもしれません。ニューラル ネットワークなどの機械学習モデルは、パラメーターの数が多く、モデル出力を生成する内部の仕組みを直感的に理解することが困難です。ニューラル ネットワークのようなサロゲートモデルがブラックボックスのように扱われることが多いのは、このような説明可能性の欠如が大きな理由です。さらに、従来の制御手法とは異なり、機械学習モデルの性能を検証することが難しく、徹底的なシミュレーションが必要となります。特にセーフティク

リティカルなアプリケーションに対してこのような制限は重大です。しかしながら、アプリケーションがうまくフィットすれば、模倣学習も MPC コントローラーを高速化する良い手段となりえます。

ヒント

模倣学習では、ニューラル ネットワークなどの機械学習モデルで MPC コントローラーの動作を近似することができます。適切な近似を得るためには、学習に使用するデータセットが、MPC が動作する状態空間の領域をうまく捉えることが必要です。

関連情報

車線維持支援のための MPC コントローラーの模倣

飛行ロボット用非線形 MPC コントローラーの模倣

メモリ要件

MPC コントローラーを設計する際に考慮すべき、もう一つの重要な性能はメモリ使用量です。通常、組み込みデバイスのメモリ容量は限られているため、実装時には特に重要となります。このホワイトペーパーでは、MPC コントローラーの実行時間に焦点を当てていますが、前述の設計パラメーターの多くは、コントローラーのメモリ要件にも影響します。

MPC 問題のパラメーター。この表では、前のセクションで説明したいくつかのパラメーターと、Model Predictive Control Toolbox が生成する線形 MPC 問題の行列の次元を関連付けています。予測ホライズン、制御ホライズン、操作変数の数、最適化変数の数、状態と出力の数、および制約条件の数などのパラメーターは、(線形) MPC コントローラーのメモリ使用量を増加させます。制約条件、重み、プラントモデル、およびホライズンのオンライン更新機能は、最適化に使用されるいくつかの中間行列が一定でなくなるため、より多くの RAM を必要とします。

ソルバー。また、ソルバーの選択や設定も重要な役割を果たします。たとえば、操作変数の数や制御ホライズンが大きい場合、スパースな QP ソルバーはヘシアン行列のメモリ使用量を削減します。

MPC のタイプ。図 7 が示すように、メモリ要件は MPC のタイプによっても異なり、たとえば、LTI MPC のメモリ使用量は小さく、一方、(ゲインスケジュール) 陽的 MPC は大きくなります。

データ型。シミュレーションとコード生成の両方で、コントローラーのデータ型を単精度に設定すると、メモリ使用量が小さくなります。

以上より、実行速度を最適化するために MPC のパラメーターを調整する際、その選択がコントローラーのメモリ使用量に与える影響を考慮する必要があります。

ヒント

組み込みアプリケーションでは、メモリ使用量を少なくする必要があります。実行速度を最適化するために MPC のパラメーターを調整するときには、その選択がコントローラーのメモリ使用量に与える影響を考慮する必要があります。

関連情報

生成された C コードのための QP 問題の構成

MPC コントローラーの単精度でのシミュレーションおよびコード生成

まとめと次のステップ

このホワイトペーパーでは、MPC コントローラーを高速化するためのさまざまな方法をご紹介しました。ご紹介したヒントとコツの概要については、表 2 をご参照ください。

表 2.モデル予測制御を高速化するためのヒントとコツ

適切なパラメーター値を選択する	計算回数を小さくするために、予測 Ts と制御 Ts は、制御要求を満たすように十分に小さく(ただし、小さすぎないように)選択します。
	MPC を高速に実行させるために、予測 Ts が制御 Ts より大きくなるように選択することを検討します。これによって、最適化問題を適切なサイズに抑え、制御サンプル時間よりも速く解けるようにします。
	予測ホライズンが長くなると、コントローラーは将来のイベントをより正確に予測し、計画することができますが、コントローラーの計算時間とメモリ使用量が増加します。
	$m<<p$ と選択すると、各制御インターバルで最適化する変数が少なくなるため、計算の高速化が促進されます。
	制御ホライズンの概念と同様に、操作変数ブロッキングでは、各制御インターバルにおける決定変数の数を指定することができるため、計算の高速化につながります。
	制約の数は少なくし、可能な限りハード制約ではなくソフト制約を選択します。
	各タイムステップで必要な計算回数を抑制するために、実行時パラメーターの変更回数を少なくします。

適切なソルバーとソルバーオプションを選択する	MPC 問題のサイズや構成は、利用できるソルバーの相対的な性能に影響を与えます。さまざまなソルバーを使用して複数のシナリオでコントローラーをシミュレーションすることで、アプリケーションに適したソルバーを検討することができます。
	フィードバック制御では、特に非線形 MPC の場合、実行可能な推定値を使用してソルバーをウォームスタートさせるのが最善の方法です。
	解析ヤコビアンと自動微分により、非線形 MPC における最適化の計算時間を短縮します。
	最適化の反復回数が指定された最大値を超えてしまう場合、準最適解を適用することで、MPC コントローラーの最悪実行時間を保証することができます。
	MPC コントローラーの C/C++ コードを自動生成し、シミュレーションや展開を加速することができます。
手法を変更する	MPC 最適化問題を簡略化するために、内部プラントモデルの入力/操作変数、状態、および出力の数をできるだけ減らします。
	最速のランタイム実行を実現するには、許容範囲の性能が得られる最もシンプルな MPC 手法を使用します。
	階層化された 2 段階の制御システム アーキテクチャを使用することで、計画と (ローレベルの) 制御が分離され、MPC をいずれか一方あるいは両方の階層に適用できるように、最適化問題が簡略化される可能性があります。
	模倣学習では、ニューラル ネットワークなどの機械学習モデルで MPC コントローラーの動作を近似することができます。適切な近似を得るためには、学習に使用するデータセットが、MPC が動作する状態空間の領域をうまく捉えることが必要です。

MPC の高速化に向けて Model Predictive Control Toolbox と共に次のステップに進みましょう。

探す

モデル予測制御を理解する (7 ビデオ) – ビデオシリーズ

モデル予測コントローラーの設計方法 (3:00) – ビデオ

モデル予測コントローラーの実装方法 (3:09) – ビデオ

実際の事例を見る

Tata Motors European Technical Centre が、モデルベースデザインで自律車両制御アルゴリズムの開発を加速 – ユーザー事例

自動運転タクシーの縦方向の制御技術の開発 – ユーザー事例

日立 Astemo、モデルベースデザインによる車間距離制御装置 (ACC) 用のモデル予測コントローラーを開発 – ユーザー事例

モデル予測制御手法による渋滞時の実用的な車間距離制御装置 (ACC) の設計 –技術情報

ダウンロード

Model Predictive Control Toolbox 評価版ソフトウェア