分割主鏡シミュレーター (第3版バージョン7) — 設計

岡山新技術望遠鏡グループ

平成 23 年 3 月 29 日

目 次

1 概要

シミュレーション中のアクチュエータ制御量分布を可視化したり、さまざまな設定をインタラクティブに変更しながらシミュレーションを行えるようにするため、シミュレータに Gtk ライブラリを利用した GUI モジュールを組み込んだ。また、シミュレーションにおけるアクチュエータ制御ルーチンを切り替えられるようにするため、データ構造と制御モジュールを分割した。

なお、シミュレータのデータの取り扱いなどの基本的な部分はバージョン 6 から変更されていないので、アクチュエータとギャップセンサー間の値の変換などについてはバージョン 6 の設計文書を参照すること。

2 データ構造

アクチュエータとギャップセンサーの現在値、変化量などの計算、相互変換は、一つのモジュールとして取り扱う。

2.1 内部データ

内部データは、アクチュエータ制御量、ギャップセンサー値、セグメントシフト量の 3 種類が必要となる。それぞれに対し、理想位置などでいくつかの分類が必要となる。パラメータの命名規約は<種別>_<分類>とする。

- 種別
  - actu : アクチュエータ制御量
  - gapsns : ギャップセンサー値
  - segshift : セグメントシフト量
- 分類
  - ideal : 現在理想位置 (コマンドが完全に実行済である場合の値)
  - real : 現実の制御位置もしくは読み出される値
  - noise : 直近のノイズ (ギャップセンサー) もしくはトラス変形による固定点シフト量 (アクチュエータ、セグメントシフト)
  - correct : 直近の補正ステップによる (理想位置) 変化量
  - resid : アクチュエータ制御残差 (時定数導入時用)

これらの間の相互関係は以下のようになる。

- actu_real = actu_ideal - actu_resid
- gapsns_ideal = SVD(actu_real)
- gapsns_real = gapsns_ideal + gapsns_noise

なお、アクチュエータ制御値にのみ、将来的な拡張用として制御残差 (アクチュエータの駆動時定数) のパラメータである actu_resid を定義する。制御モジュールによって設定された補正制御量 actu_correct の反映方法として、actu_ideal と actu_resid の両方にまず加算し、その後、駆動時定数による actu_resid の変更 (補正駆動が無遅延で完了する場合はこの段階で actu_resid は全て 0 に設定されることになる) を行うメソッドを呼ぶ。この後で、上記の相互関係の式により actu_real を設定する。

ギャップセンサーの温度係数に関してはとりあえず想定しない。これは、温度分布の状態を考慮する必要があり、現在のシミュレータでは想定していない範囲の変換を取り扱わなければならないからである。

2.2 シミュレーションのステップにおける計算の流れ

シミュレーション中の各ステップにおける計算の流れは以下のように規定する。基本的には、ステップ終了時がギャップセンサー読み出し操作の直前の状態になるようなループ上の停止位置になっている。

- 制御モジュールの補正制御量設定メソッドを呼ぶ (SimulatorBase::DoNextIteration; actu_correct がモジュールにより設定される)
  - 現在のギャップセンサー読み出し値を取得する
  - SimulatorBase の内部データ領域に、現在のギャップセンサー読み出し値の特異ベクトルに対する分布量、特異ベクトル分布の残差を設定する。
  - 制御モジュールの CalcActuatorCorrection(std::vector<double> &gapsns, std::vector<double> &actu) メソッドが呼ばれ、gapsns データから補正制御量データ actu を作成する。(制御量は補正"されるべき"量であり、実際の補正制御での移動量の-1 倍であることに注意。)
- *_noise 配列に値を設定する
- シミュレータデータオブジェクトのステップ更新メソッドを呼ぶ (SimulData::DoExecNextStep)
  - actu_correct の処理 (SimulData::DoExecNextActuator)
  - actu_resid の処理 (SimulData::CalcActuatorResidual)
  - actu_real を計算
  - セグメントシフトの処理 (SimulData::DoExecNextSegshift; 現状は何もしない)
  - ギャップセンサー読み出し値の計算 (SimulData::DoUpdateGapsns)

なお、前回ステップの計算結果である現在ステップでのギャップセンサー読み出し値による補正結果のみを取り出す場合、actu_noise の分を引き算しなければいけないことに注意する。本来ならば、この影響が入っていないアクチュエータ量として提供されたほうがいいものではあるが、この影響が入っていないアクチュエータ量に対応するギャップセンサー読み出し値をステップで算出してしまうと補正系の試験にならない[1]。

2.3 制御モジュールの作成

制御モジュールは全て後述の SimulatorBase クラスのオーバーライドクラスとして作成する必要がある。

制御モジュールからの表示としては、GUI メインウィンドウから立ち上げることができるステータス画面 (文字列のみ) が利用可能であり、また、オーバーライドクラスとして実装が必要な項目は以下のものである。なお、これら全てはスーパークラスである SimulatorBase オブジェクトの仮想関数を継承して実装する。

- 設定オプションの取り扱いメソッド
- 内部データリセットメソッド
- アクチュエータ補正量の計算メソッド

2.4 インデックスモジュールの作成

シミュレーションによるデータの補正を評価するインデックスを算出する部分も切り替え可能なようになっている。この実装は、後述の SimulatorIndex クラスの継承として実装する必要がある。

このインデックスは、ある時点におけるアクチュエータなどによるセグメントの移動量のスナップショットのみから算出される値として定義される。このため、履歴や差分などの機能は (現状では) 提供されていない。

インデックスモジュールからの出力はメインパネルにグラフとして表示される。将来的には、GUI より履歴ファイル出力などの機能を追加することも検討する。

3 モジュール設計

3.1 モジュール分割

シミュレータのモジュールはデータと制御部分を分割するという目的から、大きく 4 種類に分割する。なお、このバージョンでは PSF 演算モジュールはシミュレータに組み込まれていない。

- アクチュエータ・ギャップセンサー演算モジュール群 (行列演算)
- シミュレーションデータモジュール
- 制御モジュール群
- GUI モジュール群

それぞれのモジュール群は、対外インターフェースを定義する基本モジュールと、それをオーバーライドする拡張モジュールで構成する。これは、将来的に機能拡張を行う際にモジュールの入れ替えを簡単にすることと、モジュール数が大量になり相互関係が複雑になるシミュレータの中でモジュール間の関連を単純化したインターフェース経由にするためである。これらの目的から、上述のうちの上 3 つについて以下のようなモジュールを定義している。なお、GUI モジュール群については、Gtk を利用した GUI を構築するためにさまざまなクラスを定義する必要があるため、ここでは詳述しない。リストの先頭はモジュール (クラス) 名であり、() 内はそれぞれがオーバーライドしているスーパークラスを示し、() つきの場合はそのクラスはサブクラスとして定義されている。

- アクチュエータ・ギャップセンサー演算モジュール群 (行列演算)

[1] とはいえ、別な領域で保持して提供するという手もあるのだが。

  - GapSVD : SVD 行列演算の基本クラス
  - Gapsensor (public GapSVD) : ギャップセンサー配置を読み込んで SVD 行列を作成する機能を追加する
  - Segment : セグメント状態を保持する基本クラス
  - SegmentFile (public Segment) : ファイルからセグメント状態を読み込む機能を追加する
- シミュレーションデータモジュール
  - SimulData : シミュレーションデータの基本クラス
  - SimulatorConfig (private astro_inst::config::Config) : シミュレータ設定ファイルを扱うクラス
- 制御モジュール群
  - SimulatorBase : 制御モジュールの基本クラス
  - Simul01 (public SimulatorBase) : サンプル制御モジュールで、単純に SVD 逆行列計算した結果で制御する

なお、GUI モジュールの基本モジュールとしては以下のものが定義される。

- SimulDialog (public Gtk::Dialog, public astro_inst::system::thread::Thread) : GUI のメイン制御を行うクラスで、シミュレーションデータクラスなどをこのモジュールに設定する
- SimulStatus (public Gtk::Dialog, public astro_inst::system::thread::Thread) : ステータス画面表示用クラス
- SimulDraw (public Gtk::DrawingArea) : アクチュエータ制御量可視化クラスで、Cairo を利用して描画する

3.2 インターフェース定義

ここでは、シミュレータの機能モジュール (制御・インデックス) を実装するのに必要になるクラスのインターフェース定義をまとめる。

3.2.1 class GapSVD

SVD 行列演算を取り扱うクラスで、各種の変換を行うメソッドを提供する。シミュレータの制御モジュールでは、このクラスが提供する機能を利用して SVD の演算を行うことが想定されている。GapSVD クラスは内部で持つ変換行列を相互コピーする必要があるため、オブジェクトのコピーを行うためのインターフェースを持つ。

初期化メソッド

- bool DoCopyObject(GapSVD *) : 引数のオブジェクトの情報をコピーしてくる
- bool DoCopyMatrix(gsl_matrix *) : 内部の変換行列を引数の行列へコピーする
- bool DoCopyActuatorMap(std::vector<long> &) : アクチュエータ固定状況の情報を引数の配列へコピーする

特異ベクトル・特異値　特異ベクトル・特異値を取得する、もしくはそれに関するパラメータの取得・設定のために以下のインターフェースを持つ。

- bool SetEigenLimit(double) : 特異値への制限を設定
- double GetCurEigenLimit() : 特異値への制限を取得
- bool GetActuatorEigenVector(std::vector<double> &, size_t) : size_t で指定した ID の特異ベクトルを配列に格納して返す (アクチュエータ側)
- bool GetGapsnsEigenVector(std::vector<double> &, size_t) : size_t で指定した ID の特異ベクトルを配列に格納して返す (ギャップセンサー側)
- bool GetEigenValues(std::vector<double> &) : 特異値を配列に格納して返す
- bool DoSVD() : SVD の演算を実行する

内部パラメータ　　パラメータを取得・設定するために以下のインターフェースを持つ。

- size_t GetActuatorsCount() : 全アクチュエータ数
- size_t GetActuMoveCount() : 駆動されるアクチュエータ数
- size_t GetActuFixedCount() : 固定されているアクチュエータ数
- size_t GetGapsnsCount() : ギャップセンサー数
- bool GetInnerRing() : 内環のありなし
- double GetEigenValue(size_t) : 指定した ID の特異値
- bool SetGapsnsCount(size_t) : ギャップセンサー数設定
- bool SetInnerRing(bool) : 内環の有り無し設定
- bool SetActuatorMap(std::vector<long> &) : アクチュエータ固定・駆動定義配列を設定
- bool SetFixedActuator(std::vector<size_t> &) : 固定するアクチュエータを設定

SVD 演算　　特異ベクトルに対する分布配列を計算、もしくは SVD 変換を行うために以下のインターフェースをもつ。

- bool GetActuatorEigenDeploy(std::vector<double> &actu, std::vector<double> &deploy) : actu に指定したアクチュエータ配列の特異ベクトルに対する分布配列を deploy に計算する
- bool GetActuatorFromDeploy(std::vector<double> &actu, std::vector<double> &deploy) : deploy に指定した分布配列から作られるアクチュエータ配列を actu に計算する
- bool GetGapsnsEigenDeploy(std::vector<double> &gapsns, std::vector<double> &deploy, std::vector<double> &resid) : gapsns に指定したギャップセンサー配列の特異ベクトルに対する分布配列を deploy に計算し、ギャップセンサー配列での残差を resid に格納する
- bool ApplyActuatorEigenFactor(std::vector<double> &actu, std::vector<double> &factor) : actu に指定したアクチュエータ配列を特異ベクトルに分布させ、各分布量に対して factor をかけた分布から再度アクチュエータ配列を作成して actu に入れる
- bool CalcGapsnsFromActu(std::vector<double> actuator, std::vector<double> &gapsns) : アクチュエータ配列 actuator からギャップセンサー配列 gapsns に行列変換
- bool CalcActuFromGapsns(std::vector<double> gapsns, std::vector<double> &actuator) : ギャップセンサー配列 gapsns からアクチュエータ配列 actuator に SVD 逆行列変換

3.2.2 class SimulData

シミュレーションにおける各ステップでのデータを保持し、ステップ進行を行うクラスである。このクラスは、アクチュエータ・ギャップセンサーの変換系や、セグメント並進・回転に関する影響を計算するオブジェクトを持ち、それらのオブジェクトを利用することで、ステップシミュレーションに必要なデータ変換をまわす。

ステップにおける内部でのデータ変換の流れは前述 (2 節) を参照。

内部データ初期化メソッド

- bool DoInitObjects(SimulatorConfig *config) : 内部オブジェクトの初期化
- bool DoTerminateObjects() : 内部オブジェクトの開放
- void DoResetDataVector() : 内部データ初期化
- void SetObjRandom(astro_inst::gsl::Random *obj) : 乱数生成器設定
- void SetObjRandomSeed(unsigned long seed) : 乱数生成器のシード設定、再初期化 (同じシードを入れると同じ乱数列が出てくる)

外部に提供する行列演算の標準メソッド

- void CalcStat(std::vector<double> &stat, std::vector<double> &value) : 平均・分散の統計量を返す

- void CalcStat(std::vector<double> &stat, std::vector<double> &value, std::vector<double> &bias) : 平均・分散の統計量を返すがデータは (value - bias)
- bool CalcVector(std::vector<double> &target, std::vector<double> &source, bool is_add) : target の各要素に source の各要素を加算 (is_add == true) もしくは減産 (is_add == false) する
- bool CalcNewNoise(std::vector<double> &noise, double value) : value を制御値とした乱数列で埋める

ユーティリティー

- bool DoCopyGapSVD(GapSVD *target) : 引数で渡された GapSVD の変換行列を内部で持つ行列のコピーに設定する

ステップ実行・設定

- bool DoExecNextStep() : ステップ 1 段実行
- void SetCalcGapsnsReal(bool do_real) : ギャップセンサーの値導出を行列変換で行う (false) か、全て計算する (true) か
- bool SetNextActuatorCorrection(std::vector<double> &correction) : アクチュエータ補正量の設定
- bool SetNextActuatorCorrection(std::vector<double> &correction, bool rev) : アクチュエータ補正量の設定
- bool SetNoiseActuatorByRandom(double random) : アクチュエータの変動を乱数で設定
- bool SetNoiseGapsnsByRandom(double random) : ギャップセンサー読み出しノイズを乱数で設定
- bool SetNoiseSegshiftByRandom(double random) : セグメントシフト変化量を乱数で設定

データ取得　引数に std::vector<double>の値をとり、関数名で指定される各データを引数の配列に格納する。

- アクチュエータ関係
  - void GetActuIdeal
  - void GetActuReal
  - void GetActuNoise
  - void GetActuCorrect
  - void GetActuResid
- ギャップセンサー関係
  - void GetGapsnsIdeal
  - void GetGapsnsReal
  - void GetGapsnsNoise
  - void GetGapsnsCorrect
- セグメント並進・回転関係 (現在利用されていない)
  - void GetSegshiftIdeal
  - void GetSegshiftReal
  - void GetSegshiftNoise

3.2.3 class SimulatorBase

シミュレータの制御値計算モジュールの既定クラスで、シミュレータから呼ぶ際の手続きや必要なメソッド (および共通部分のコード) を定義している。

手続きの定義 (仮想メソッド)　ここに含まれる (仮想) メソッドは、必ず継承するモジュールで定義する必要がある。
パラメータ設定に関するメソッドについては、ConfigGetList で取得される文字列がそれぞれのパラメータのキーとなり、かつ GUI に表示される。なお、パラメータは全て double の実数である。

- void ConfigGetList(std::vector<std::string> &list) : パラメータ ID のリスト

- double ConfigGetValue(std::string config) : 指定されたパラメータ ID に対する現在の設定値を返す
- bool ConfigSetValue(std::string config, double value) : 指定されたパラメータ ID の値を設定する
- bool CalcActuatorCorrection(std::vector<double> &gapsns, std::vector<double> &actu) : ステップを実行する。gapsns で渡された値が現在のギャップセンサー読み出し値、そこから変換したアクチュエータ制御量を actu に入れる。
- bool StatusGetString(SimulData *object, std::string &text) : ステップ実行後にステータス画面に表示する文字列を text に格納する。戻り値が true の場合は追記になる。
- void DoResetData() : 内部データを全てリセットする。(ただし、SVD 行列オブジェクトなどのオブジェクトは除く。)

内部データ　処理に利用できる内部データをいくつか自動的に設定している。いくつかの値については設定した回数の履歴を保存しているので利用可能である。ただし、履歴は std::list であり、かつ最新のものが先頭に保存されているので注意。

- std::string simul_name : モジュール名 (コンストラクタで必ず設定)
- bool SetHistSize(size_t size) : 保存する履歴の回数
- std::list< std::vector<double> > hist_actu_correct : アクチュエータ補正量として返した値
- std::list< std::vector<double> > hist_eigen_dist : ギャップセンサー読み出し値の特異ベクトル分解値
- std::list< std::vector<double> > hist_eigen_resid : ギャップセンサー読み出し値の特異ベクトル分解値の残差

シミュレータ用メソッド　ここで定義されているメソッドをオーバーライドしてもシミュレータからは利用されないので注意。なお、シミュレータに設定する前にこれらのメソッドを呼んで (特に SVD 行列のオブジェクトは) 初期化する必要がある。

- オブジェクト設定・内部初期化
  - bool SetConfigObject(SimulatorConfig *source)
  - bool SetSVDObject(GapSVD *source)
  - bool SetSVDObject(SimulData *source)
- bool DoNextIteration(SimulData *object) : ステップ実行
- void BumpHistList() : 履歴データ保存
- std::string GetSimulatorName() : モジュール名を返す

3.2.4 class SimulatorIndex

インデックスを計算するモジュールの基底クラスで、全てこのクラスの継承とする必要がある。

手続きの定義 (仮想メソッド)

- double GetCurrentIndex() : インデックス値を返す

初期化

- bool SetDataObject(SimulData *source) : データオブジェクト設定
- std::string GetIndexName() : モジュール名を返す
- std::string index_name : モジュール名 (コンストラクタで設定)