Read me

1. Choreonoid 及び graspPlugin

環境構築を，**Ubuntu Linux** にて行う．Ubuntu OS が自分の PC に入っていない場合は，先輩にインストール Disk を借りて，自分の PC にインストールしておこう！以降の手順は，Windows ではなく Ubuntu 上で処理すること．

また，コマンドを実行するために”**Terminal**”というアプリケーションを用いるが，Ctrl+Alt+T で立ち上げることが出来る．

I. Choreonoid

i. 参照サイト URL

・ダウンロード

http://choreonoid.org/ja/download.html#windows-32bit-windows-8-windows-7

・インストール

http://choreonoid.org/ja/install/install.html

ii. ダウンロード

ⅰ.の参照サイト URL より，ソースパッケージ内（もしくは旧バージョン内）の

choreonoid-1.3.1.zip

をダウンロード．私が ver.1.3.1 を使っているので，このバージョンを例として以降で用いるが，最新版でも使える・・・はず．

iii. インストール

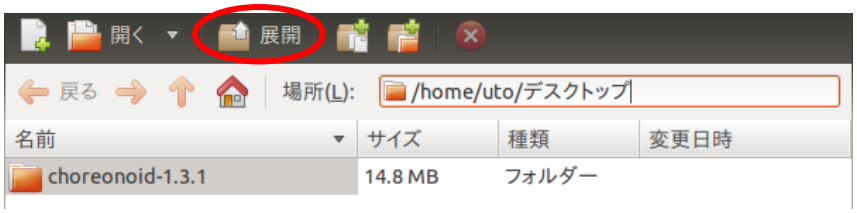
① Zip ファイルの展開

ⅰ.の参照サイト URL より，ソースコードからのインストール内の

ソースコードからのビルドとインストール（Ubuntu Linux 編）

をクリック．基本的にはここに書いてある処理を行う．

ⅱ.で DL した choreonoid-1.3.1.zip の展開をダブルクリックで行う．ダブルクリックすると，以下のウィンドウが出るので，展開．展開場所は home/(ユーザ名)/に展開することをお勧めする．



② 開発ツールと依存ソフトウェアのインストール

展開した後，Ctrl+Alt+T で”Terminal”を立ち上げ，以下のコマンドを実行する．

```
cd choreonoid-1.3.1
```

※cd は「チェンジディレクトリ」を行うコマンドで，Terminal で現在いるディレクトリの下層ディレクトリに移動するコマンド．Terminal を立ち上げた時は必ず

home/(ユーザ名)/のディレクトリにいる．そのため，choreonoid-1.3.1 を home/(ユーザ名)/のに展開しておくことで，Terminal を立ち上げた後，上のコマンド入力で即座に choreonoid-1.3.1 ディレクトリに移動できる．これが例えば，デスクトップ下に展開してしまった場合は，

```
cd デスクトップ/choreonoid-1.3.1
```

とコマンドを叩かなければならないので，正直めんどくさい．

Choreonoid-1.3.1 下に移動した後，開発ツールと依存ソフトウェアをインストールするために，以下のコマンドを叩く．

```
misc/script/install-requisties-ubuntu-11.10.sh
```

赤字の部分は ubuntu のバージョンを入れる．Ubuntu のバージョンの確認法としてホーム画面の右上にある 

ボタンを押し，システム設定をクリック．

こういう画面が出るので，さらに赤丸で囲まれたシステム設定をクリック．

赤のアンダーラインの箇所が，ubuntu のバージョンとなる．この数字を上記したコマンドに打ち込む．すると以下のように，ubuntu のエントリーパスワードが求められるので入力（打ち込んでも表示されない）．

```
uto@uto-Prime-Series:~/デスクトップ/choreonoid-1.3.1$ misc/script/install-requis
ites-ubuntu-11.10.sh
[sudo] password for uto:
```

数秒～数分ほどで，以下のように容量の確保が求められるので「y」を押してインストール完了．

```
アップグレード：2 個、新規インストール：0 個、削除：0 個、保留：503 個。
2,967 kB のアーカイブを取得する必要があります。
この操作後に 606 kB のディスク容量が解放されます。
続行しますか [Y/n]?
```

③　ビルド設定

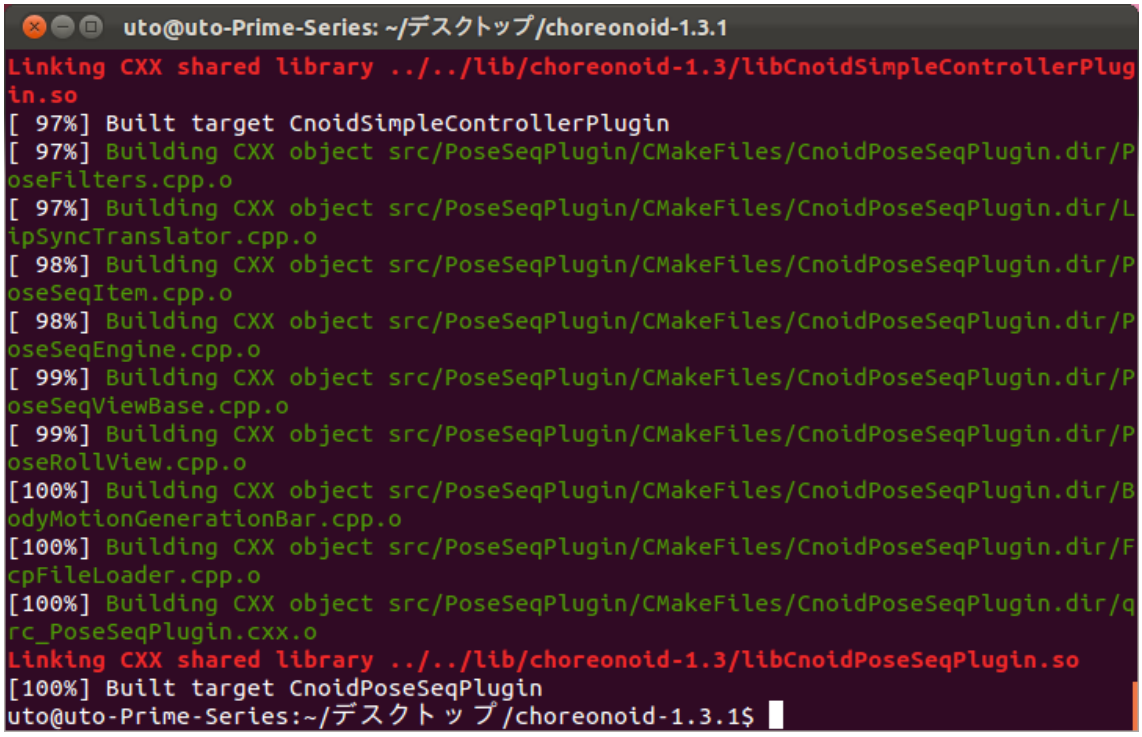
choreonoid-1.3.1 ディレクトリ下で，以下のコマンドを実行し，Makefile を作成する．

```
cmake .
```

ただし，ピリオドの前に半角スペースを空けることに注意．実行の完了が確認できたら，以下のコマンドを入力．

```
make –j4
```

これは 4 つのビルドプロセスを並列処理するコマンドで，コア数が多ければ 4 の箇所を増やし，更に高速処理が行える．Make 完了画面は以下の通り．



ｉ.の参考 URL にはこの後，「Qt スタイルの変更による描画速度の改善」と「Balancer プラグインについて」があるが，これの部分は割愛（多分やらなくても大丈夫なのではなかろうか）．

II. graspPulgin

i. 参照サイト URL

http://choreonoid.org/GraspPlugin/i/ja/node/2

ii. graspPlugin のインストールとビルド

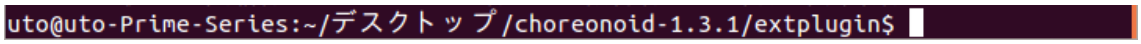
ｉ.の参照サイトと見比べながら，graspPlugin のインストールとビルドを行ってほしい．

① graspPlugin インストール

いきなりだが，graspPlugin のインストールは参照 URL のではなく，マーキュリアル

というサーバからコピーしてくる．現在 choreonoid-1.3.1 のディレクトリにいると思うので，以下のコマンドで，その下層ディレクトリ extplugin に移動する．

```
cd extplugin
```



そこで，マーキュリアルより graspPlugin をインストールするために以下のコマンドを実行．ただし，これにはマーキュリアルのアカウントを持っておく必要があるので，持ってない場合は辻先生か，他の先生方に相談してほしい．

```
hg clone http://subaru.ait.kyushu-u.ac.jp/hg/grasp-plugin-hg
```

もし，どうしてもアカウントが手に入らない場合は，参考 URL 通りにインストールするとよい．ただしこの場合，通常の GraspPlugin は使用できるが，以下の２．開発プログラムに示す関数等は用いることができない．

② PRM インストール

これは参照ＵＲＬと同様の処理を施す．以下のサイトにアクセス．

https://code.google.com/p/grasp-plugin/downloads/list

その後，以下のファイルをダウンロード．

PRM-1.3.zip

この zip ファイルを choreonoid と同様のやり方で，
choreonoid-1.3.1/extplugin/graspPlugin 以下に展開．

③ 必須パッケージインストール

graspPlugin に必要なパッケージをインストールするために，以下のコマンドを choreonoid-1.3.1/extplugin 以下で実行．

```
./graspPlugin/Grasp/install-requisities-ubuntu.sh
```

※「ins」くらいの時に Tab キーを押すと，ＰＣ側が大体の予測を立て，それ以降を表示してくれるから便利！

④ ビルド

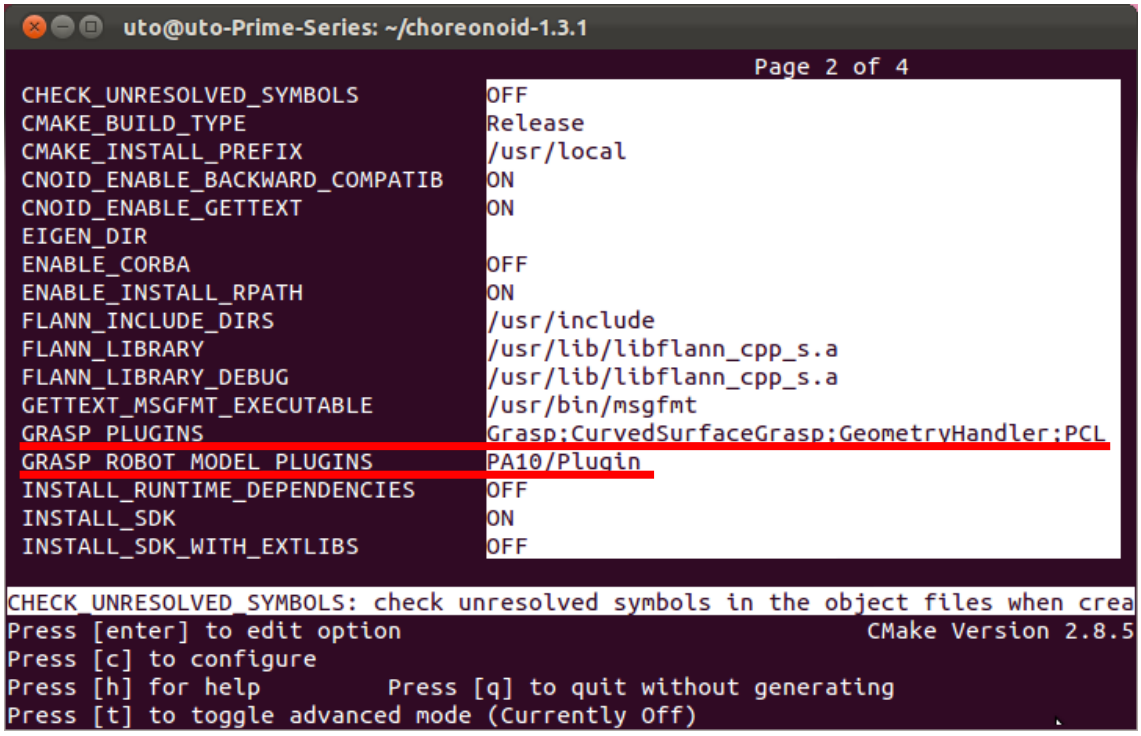
graspPlugin を choreonoid で使用できるようにするために，ccmake でビルド設定の変更を行う．まず，以下のコマンドで，choreonoid-1.3.1/extplugin から choreonoid-1.3.1 へ移動する．

```
cd ../
```

その後，以下のコマンドでビルド設定画面を開く．

```
ccmake .
```

ただし，ピリオドの前に半角スペースを空けることに注意．すると以下のようなウィンドウが現れるので，２度「ｃ」キーを押し，configure する．すると，下の方に GRASP_PLUGUINS と GRASP_ROBOT_MODEL_PLUGINS のつが編集できるようになる．



↓キーで GRASP_PLUGINS まで移動し, Enter で編集有効モードにする. 中に Grasp, CurvedSurfaceGrasp，GeometryHandle の 3 つを加え，Enter を押し終了.
GRASP_ROBOT_MODEL_PLUGINS には PA10/Plugin を加え，Enter で終了.
※PCL というライブラリを使うときは，PCL をインストールし，上に示すように GRASP_PLUGINS に PCL を加える. また, graspPlugin をインストールするときに，マーキュリアルからではなく，https://code.google.com/p/grasp-plugin/downloads/list から行っている場合は，CurvedSurfaceGrasp は使うことが出来ない.

その後，以下のコマンドにて make を実行する.

```
make –j4
```

make がとおれば成功！

2. Choreonoid 内の追加ボタン解説

choreonoid-1.3.1 下に移動し，以下のコマンドで Choreonoid を立ち上げることが出来る．

bin/choreonoid

立ち上げた後，私の研究で用いた PlannerBar に追加したボタンと，キーボード操作による二次曲面の情報ファイル生成について、以下で述べる．

I. キーボード操作

i. c キー

二次曲面近似を選択した状態で「c」キーをおすと，近似された二次曲面の式の係数情報と，そのクラスタの id が格納されたファイル「test.yaml」が choreonoid-1.3.1 下に生成される．ちなみに，これにより物体にはカラーリングが施されるが，二次曲面の形状を表示しているのではなく，一つの二次曲面で近似された領域が，一つの色で塗られた物体を表示しているだけである．

また，「c」キーを押すだけだと，設定された近似精度の閾値によって，二次曲面の数が決められているが，Geometry バーにある「Target number clustering」で，近似に用いる二次曲面の数をユーザが指定することが出来る．

ii. b キー

「c」キーを押した後に「b」キーを押すと，二次曲面の組み合わせから一葉双曲面と，２つの楕円体により構成されるくびれを把持するための情報が格納されたファイル「boundaryData.yaml」が choreonoid-1.3.1 下に生成されてる．具体的には以下のとおりである．

- boundaryNum：くびれ（共通切断面）の番号
- Common Cutting Plan：共通切断面の式の係数
- shapeNum：一つの共通切断面における境界の番号
- shape：くびれの種類
- id：くびれを生成しているクラスタの id

  一葉双曲面なら 1 つの id と 0，楕円体によるくびれなら 2 つの id が表示
- targetPoint：ハンドをアプローチさせる時のターゲットとなる点の座標
- graspVec：グリッパの開閉軸とハンドのアプローチ軸

boundaryNum と shapeNum の具体図

iii. v キー

「c」キーを押した後に「v」キーを押すと，二次曲面の組み合わせから，複数の二次曲面を把持するための情報が格納されたファイル「DepartData.yaml」が choreonoid-1.3.1 下に生成されてる．具体的には以下のとおりである．

- 二次曲面の組み合わせ
- Middle point：ハンドをアプローチさせる時のターゲットとなる点の座標

- approachVec：ハンドのアプローチ軸
- fingerVec：グリッパの開閉軸

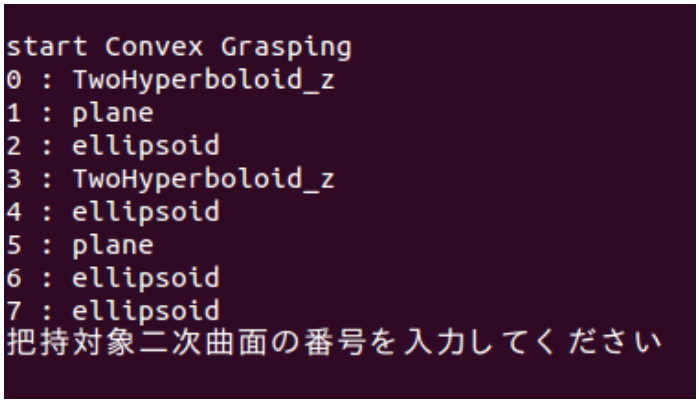
iv. x キー

void ObjectAnalysis::LmitedInitialClustersFromOneFace()にて定義されるある限定的な範囲に対して二次曲面近似を行う．

II. Planner ボタン

i. QuadricSurfaceGrasp

楕円体，楕円柱を把持するためのボタン． choreonoid-1.3.1 ディレクトリ内に「test.yaml」ファイルがある状態で QuadricSurfaceGrasp ボタンを押すと，以下の図のように，どの二次曲面を把持するかの選択画面が出る．



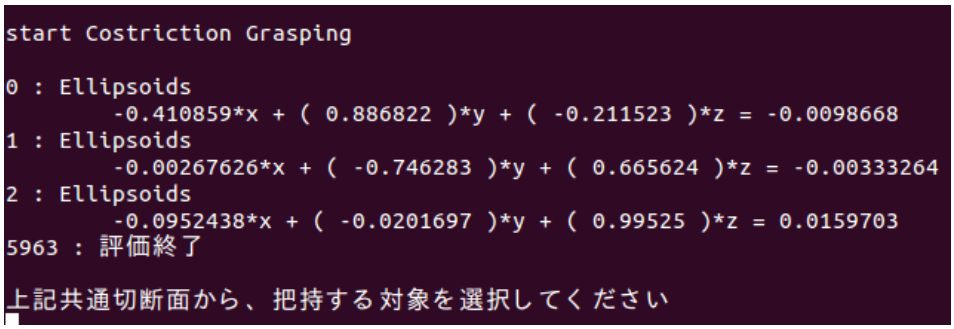
楕円体と楕円柱を対象としたボタンなので,「ellipsoid」「cylindrical surface_x(,y,z)」のみが有効．上の図ならば，2,4,6,7 番のみが有効．その番号を入力してエンターを押せば把持が始まる．

また楕円体を把持する場合は，どの二次曲面との境界を把持するかの選択を行う．

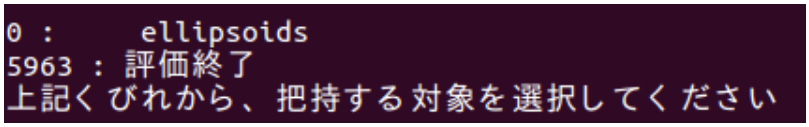
ii. ConstrictionGrasp

一葉双曲面と，２つの楕円体から生成されるくびれを把持するためのボタン．

choreonoid-1.3.1 ディレクトリ内に「boundaryData.yaml」ファイルがある状態で ConstrictionGrasp ボタンを押すと，以下の図のように，どの共通切断面に対して把持を行うのかを選択する（共通切断面については後述）．



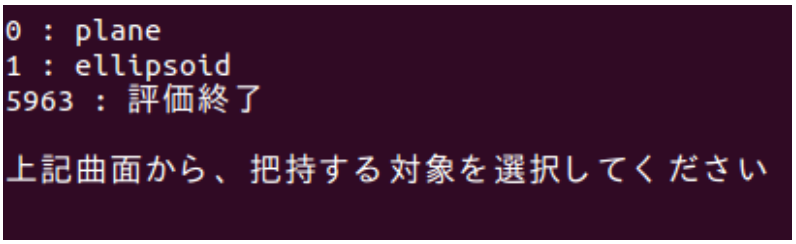
その後，一葉双曲面か，くびれかを選択する．



ただし，選択を失敗した場合，表示されている番号ではなく「5963」を入力すると，把持計画が強制終了できる．

iii. DepartSurfaceGrasp

楕円体-平面，楕円柱-平面，平面-平面を把持するためのボタン．choreonoid-1.3.1 ディレクトリ内に「DepartData.yaml」ファイルがある状態で DepartSurfaceGrasp ボタンを押すと，以下の図のように，１つめの二次曲面を選択できる．



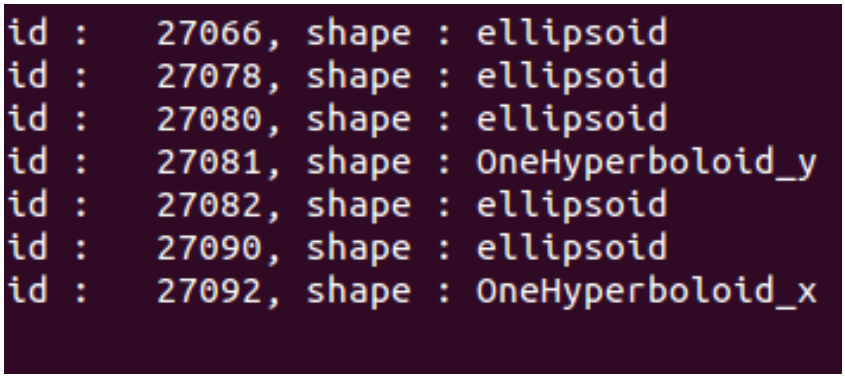
選択すると，もう一つの二次曲面を選択することが出来る．選択を失敗した場合，表示されている番号ではなく「5963」を入力すると，把持計画が強制終了できる．

iv. check

現在は何にも使われていない（はずの）ボタン．関数の挙動をチェックする用に設置している．このボタンの中身は，後述するが，graspPlugin/CurvedSurfaceGrasp/EllipsoidGraspController.cpp 内の checkMotion() という関数が反映されているため，この関数をいじるとよい．

v. check_shape

test.yaml に格納されている二次曲面の形状を表示するボタン．押すと以下のように，二次曲面の id と形状を表記してくれる．



3. 開発プログラム

私が開発したプログラムが格納されているファイルは, 大きく以下の３つである. ただし, すべての cpp ファイル, h ファイルは choreonoid-1.3.1/extplugin/graspPlugin/以下にある.

- CurvedSurfaceGrasp/EllipsoidGraspController.h/cpp
- CurvedSurfaceGrasp/SurfaceForceClosure.h/cpp
- GeometryHandler/GeometryAnalysis.h/cpp

これらに作った関数を以下で説明していく．なお，省略している関数はあまり重要ではない(もしくは作っただけで使ってない)ので，気にしなくてもいい・・・と思う.

I. EllipsoidGraspController.h/cpp

- void calcConffience

  二次曲面の係数 VectorXd co を引数として，二次曲面の半径(co_rad)，中心座標(co_gap)，回転行列(co_evec)，形状(co_shape)を導出する.

- virtual void graspConvexShape

  Planner バーの QuadricSurfaceGrasp ボタンが押されると実行する関数. test.yaml に格納されている二次曲面の式から，二次曲面の半径(rad)，中心座標(gap)，回転行列(evec)，形状(shape)等を計算(この部分は calcCofficience で計算される). その後，形状が ellipsoid，cylindrical surface x(y，z)の二次曲面に対して把持姿勢を生成(createGraspPosForEllipsoid or createGraspPosForEllipse)，評価(calcEllValue or calcCylValue)までを行う．ちなみに QuadricSurfaceGrasp ボタンを押したときに選ぶ数字が格納されている変数は cnt2

- virtual void graspConstriction

  Planner バーの ConstrictionGrasp ボタンが押されると実行する関数. boundaryData.yaml に格納されている凹部分に対する把持姿勢生成情報(targetPoint, graspVec)から，hyperboloid，elliposoids に対する把持姿勢を生成(createGraspPosForEllipse)し，安定性の評価(calcConstValue)を行う.

  この過程で，グリッパが凹部分を挟み込むために，グリッパを微調整する ConstrictionApproachFinger や，接触点の数を判別する checkContactNum などを行う.

- virtual void graspDepartSurface

  Planner バーの DepartSurfaceGrasp ボタンが押されると実行する関数. DerpartData.yaml に格納されている曲面と平面の組み合わせに対する把持姿勢生成情報(Middle point, approachVec, fingVec)をもとに，把持姿勢を生成(createGraspPosForDepartSurface)し，安定性の評価(calcDepartValue)を行う.

- void closeFingers_move

  指を閉じる関数．引数として mode，fingVec がある．mode は，４種類あり，そ

れぞれ以下のとおりである．

- mode=0：一度グリッパを最大まで開き，物体と指が干渉したら終了
- mode=1：一度グリッパを最大まで開き，物体と指が干渉し，さらに press(グローバル変数)分だけ，物体にめり込む把持を行う．
- mode=2：現在のグリッパの位置からグリッパを閉じ始め，物体と指が干渉したら終了
- mode=3：現在のグリッパの位置からグリッパを閉じ始め，さらに press(グローバル変数)分だけ，物体にめり込む把持を行う．

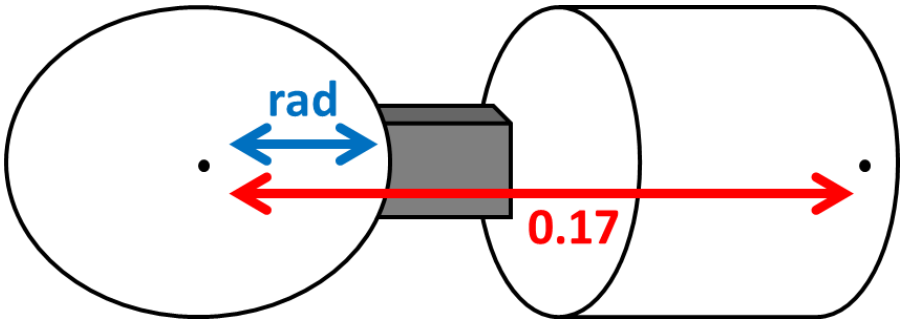
グリッパと指の干渉判定は calcContactPoint 関数にて行っている．この関数の引数にあたる以下の３つのベクトル

- 物体からみた時のグリッパの相対位置（めり込む前）
- 謎
- 物体から指への法線ベクトル

が，各指毎に fingVec に格納されている．

- void createGraspPosForEllipsoid

３軸が分かっている場合の把持姿勢生成関数．具体的には，引数である ApproachSelect よりハンドのアプローチ姿勢を，CloseFingSelect よりグリッパの開閉姿勢を生成する．他の引数 rad は，物体からハンドをどの程度離すか（0.17-rad という箇所）で用いる．Ell_p，Ell_R は把持対象の二次曲面の位置，姿勢．



- void createGraspPosForEllipse

２軸が分かっている場合の把持姿勢生成関数．引数は createGraspPosForEllipsoid と同じ．

- void calcObjFingPosAndNor

安定性評価の関数に用いる ObjFing_p と ObjFing_n を導出する関数．引数の p_sub はグリッパが物体にめり込む前の，物体から見た時のグリッパの相対位置が格納されている．

- void calcCylArea

楕円柱とグリッパの接触面積を導出する関数. 対象となる楕円柱の式の係数 qc と，楕円柱の半径 rad を用いる．グリッパと楕円柱の式から解析的に面積を導出する場合は，cpp 上部にある#define calcContactAreaAnalytical を，干渉メッ

シュ情報から面積を導出する場合は# calcContactAreaUsingMeshes をコメントインする．こうして接触面を構成する２つのパラメータ b,L を導出し，calcEn にて en を導出し，格納する．

- double calcEllValue

  楕円体把持における安定性評価関数．calcObjFingPosAndNor より，ObjFing_p と ObjFing_n を導出する. その後解析的, または干渉メッシュより面積を導出し(使い分けは calcCylArea と同様)，calcEn より en を導出する．その後，SurfaceForceClosure::forceClosureTestEllipsoidSoftFinger_uto2 より，安定性の評価を行う．

  また引数の time は，cpp 上部の#define DEBUG_TIME をコメントインすることで，評価にかかった時間が格納される変数である．

- double calcCylValue

  楕円柱把持における安定性評価関数．calcObjFingPosAndNor より，ObjFing_p と ObjFing_n を導出，calcCylArea より en を導出し，SurfaceForceClosure::forceClosureTestEllipsoidSoftFinger_uto2 より，安定性の評価を行う．

- void ConstrictionApproachFinger

  凹部分を把持する場合の，グリッパの微調整関数．BoundaryCata.yaml の情報だけでは，凹部分を綺麗に挟み込むことは困難であるため，この関数を作成．Cpp 上部の#define controlFingerUpDown をコメントインすると, ハンドの上下のみで微調整を行う．一方#define controlFingerUpDownandRoll をコメントインすると，ハンドの上下に加え，ハンドの回転も行い，微調整を行う．

  ちなみに controlFingerUpDown のみで結構精度よく微調整できた．

- void checkContactNum

  凹部分を把持する際の，グリッパと把持対象との接触点数を判別する関数．判別手法は修論に記載しているので，そちらを参照すること

- void calcEllipsoidsEn

  複数の楕円体により生成されるくびれを把持する場合の en を導出する関数. 接触点数 contactNum 毎に接触面を構成するパラメータ b,L の導出法が異なる．また，これはグリッパと物体の干渉メッシュ情報を用いた b,L の導出を行っておらず，解析的に b,L を導出している．

  calcEn を用い，b,L から en を導出する．

- void calcHypeboloidEn

  一葉双曲面を把持する場合の en を導出する関数. 説明としては calcEllipsoidsEn とほぼ同様.

- double calcConstValue

  凹部分を把持する場合の安定性評価を行う関数．ObjFing_p と ObjFing_n の導出法だが，めんどくさいので解説は省略する．頑張って読み解いてほしい．その後，calcEllipsoidsEn や calcHypeboloidEn で en を導出し，SurfaceForceClosure::forceClosureTestEllipsoidSoftFinger_uto2 より，安定性の評価を行う．

- double calcEn

  接触面を構成するパラメータ b,L より en を導出する関数．Mode が８つ準備されており，それぞれが以下の応力分布を示している．

  - mode=0：回転放物面（楕円体とか）
  - mode=1：放物線柱面（楕円柱とか）
  - mode=2：双極放物面―状態１（くびれとか）
  - mode=5：双極放物面―状態２（くびれとか）
  - mode=6：直方体（平面とか）

- void createGraspPosForDepartSurface

  曲面と平面の組み合わせに対して把持姿勢を生成する関数．Tmp_point に向かって tmp_approacVec 方向からハンドをアプローチさせ，tmp_fingerVec 方向に指を閉じる．

- double calcDepartValue

  曲面と平面の組み合わせに対して安定性の評価を行う．calcContactPoint より ObjFing_p と ObjFing_n を導出し，calcEllPlaEn，calcCylPlaEn，calPlaPlaEn より，把持対象に対応した en を導出する．その後 SurfaceForceClosure::forceClosureTestEllipsoidSoftFinger_uto2 より，安定性の評価を行う．

  ただし引数である exchange_check は，把持対象となる 2 つの二次曲面の id が昇順であれば 0，降順であれば 1 となる．

- void calcEllPlaEn

  楕円体と平面の組み合わせを把持する際の en を導出する関数．２本の指のどちらがどちらの曲面と干渉しているかの判別を行い，楕円体側は解析的に，平面側はグリッパとの干渉メッシュ情報を用いて，接触面を構成するパラメータ b,L を導出している．それらの b,L から calcEn をもちいて en を導出する．

- void calcCylPlaEn

  楕円柱と平面の組み合わせを把持する際の en を導出する関数．２本の指のどちらがどちらの曲面と干渉しているかの判別を行い，楕円柱，平面共に干渉メッシュ情報を用いて，接触面を構成するパラメータ b,L を導出している．それらの b,L から calcEn をもちいて en を導出する．

- void calcPlaPlaEn

  平面と平面の組み合わせを把持する際の en を導出する関数．２本の指のどちらがどちらの平面と干渉しているかの判別を行い，干渉メッシュ情報を用いて，接触面を構成するパラメータ b,L を導出している．それらの b,L から calcEn をもちいて en を導出する．

- Bool checkCollisionUsingMeshes

  物体とグリッパが干渉しているかのチェックを行う関数．具体的にはグリッパ表面のメッシュと，物体のメッシュが交差していれば true を返し，交差していなければ false を返す．

- void calcContactRegionParameters

  グリッパと物体の接触領域を構成するパラメータ b,L を導出する関数．
  calcEllPlaEn, calcCylPlaEn, calcPlaPlaEn に対してこの関数使えば，もっとスマートになるかも？

- void calcContactRegionPoints

  グリッパと物体の接触領域の境界点 collision_point を導出する関数．Mode が２つ用意されており，それぞれが以下のとおりである．
  - mode=0：実際のメッシュ情報のみをもちいた境界点群
  - mode=1：mode=0 より導出された点群間を補完し，新たに collision_point[2]と[3]に格納する．

- void double_vec3_QSort

  double 型の配列と，それに対応した Vector3 型の配列のクイックソートを行う関数．ネットで拾ってきたやつなので，信頼度は高いと思う．

- void double_Swap

  double 型の要素を交換する関数．

- void vec3_Swap

  Vector3 型の要素を交換する関数．

- void PCAforPoints

  引数である点群 pos に対して PCA をかけ，その時の主軸３軸 mainAxis を導出する関数．mainAxis の昇順に第一，第二，第三主軸

- double calcAngle

  引数として渡される２つのベクトルがなす角度を 0～π の範囲で導出する関数．

- double calcAngleFull

引数として渡される２つのベクトルがなす角度を-π～πの範囲で導出する関数.

- virtual void checkShape
  test.yaml に格納されている二次曲面の形状を判別する関数．Planner バーの check_shape ボタンが押されると実行する関数.

- static double calcContactPoint
  グリッパと物体との最小距離を求める関数．得られる引数として，以下の３つがある.
  - Po：物体からみた時のグリッパの相対位置
  - Pf：謎
  - ObjN：物体から指への法線ベクトル

- define DEBUG_TIME
  コメントインすると，把持計画全体の時間を図ることが出来る.

- define DEBUG_CloseFinger
  コメントインすると，グリッパを閉じる動作を確認することが出来る.

- “define controlFingerUpDown” and “define controlFingerUpDownandRoll”
  凹部分を挟む場合，どちらの方法で微調整を行うか選べる．前者はハンドを上下させるのみで調整，後者はハンドの回転も行い調整を行う.

- “define calcContactAreaAnalytical” and “calcContactAreaUsingMeshes”
  グリッパと物体の接触領域を構成するパラメータを導出する方法を選べる．前者は，グリッパ表面の式と二次曲面の式を用いて解析に解く．後者は干渉メッシュ情報を用いて解く.

II. SurfaceForceClosure.h/cpp

- double forceClosureTestEllipsoidSoftFinger_uto2
  面接触かつ, 摩擦円錐を楕円体近似したときの ForceClosure に基づいた安定性評価を行う関数．引数は以下の通り.
  - wrench：レンチ
  - Pc：物体からみた時のグリッパの相対位置
  - Nc：物体からグリッパへの法線ベクトル
  - Point：指の数（凹部分を把持するとき，接触点が増えても，２本指であればここは２）
  - Mu：静止摩擦係数
  - F_max：最大荷重

- En：最大摩擦モーメントと最大摩擦力の比

- double NormalForceClosureTest_uto

  面接触における ForceClosure にもとづいた安定性評価を行う関数.
  forceClosureTestEllipsoidSoftFinger_uto2 とは，行列Gの中身が異なる.

III. GeometryAnalysis.h/cpp

- void calcApproximatedPlane

  ２つの曲面の境界の点群を格納している boundaryList より, その境界点群を平面で近似したときの平面の式を導出する関数．点群数が 10 以下の場合は，共通切断面の傾きの信頼度が低くなるため，共通切断面を導出しない．10 以上の場合は，その点群に PCA をかけ，法線ベクトルを導出し，共通切断面の式を導出する.
  引数の listNum はなぜか関数の中で用いられていないので、無視しましょう・・・

- void createLineEllPla

  楕円体-平面に対する把持姿勢を生成するための approachVec，fingVec を導出する関数．導出法は修論を参考にすること.

- void createLineCylPla

  楕円柱-平面に対する把持姿勢を生成するための approachVec，fingVec を導出する関数．導出法は修論を参考にすること.

- void createLinePlaPla

  平面-平面に対する把持姿勢を生成するための approachVec，fingVec を導出する関数．導出法は修論を参考にすること.

- void createBoundaryData

  b キーを押したときに実行され boundaryData.yaml を生成する関数. 曲面同士の境界情報 boundaryList を作成し，それらの境界に対して共通切断面を生成しておく．calcConstriction より，凹部分にたいする把持姿勢を生成する targetPoint, graspVec を導出し，くびれを生成する二次曲面情報と共に，boundaryData.yaml に書き込む.

- void sortBoundary

  boundarydata の中身を整理する関数. 境界を構成する二次曲面の組み合わせでかぶっているものがあれば, それを boudarydata から削除したり, 境界を構成する二次曲面の id を昇順にするなどの整理を行う.

- void calcConstriction

  一葉双曲面や，２つの楕円体より構成されるくびれの情報を計算し，targetPoint

や graspVec 等を導出する関数. 一葉双曲面が二次曲面の組み合わせ内にある場合, calcHyperboloid を用いて，targetPoint と graspVec を導出する.

一方２つの楕円体が境界を生成している場合，calcBoundaryArea で共通切断面との各々の接触面の面積と，targetPoint，graspVec を導出する．面積と checkConstriction によりくびれ判定を行い，くびれていると判定された場合は，targetPoint と graspVec を boudarydata に格納する.

- int checkQuadricSurface

  test.yaml における二次曲面の形状を判別する関数.

- void calcBoundaryArea

  2 つの楕円体が境界を構成する場合, それらと共通切断面(係数は引数の co)の接触領域の面積 S と，把持姿勢生成に用いる targetPoint，graspVec を導出する.

- void calcHyperboloid

  二次曲面の組み合わせの中に一葉双曲面があれば，それを把持するための targetPoint と graspVec を導出する関数.

- bool checkConstriction

  2 つの楕円体と，それらの共通切断面との接触領域の面積より，それら２つの楕円体がくびれを生成しているかの判別を行う関数．判別の方法は修論を参考にすること.

- void createDepartData

  v キーを押したときに実行され DepartData.yaml を生成する関数. 二次曲面の id と形状を全て把握し，それらを２つづつの組み合わせにしたリスト departData を生成する. その後, それぞれの組み合わせ毎に grasp_point, approachVec, fingerVec を導出し，DepartData.yaml に書き込む.

- void calcBoundaryMainAxis

  boundaryList に格納されている境界点群に対して PCA をかけ，第一主軸と第二主軸を導出し，v1，v2 に格納する関数.

- cnoid::Vector3 calcMeshеCenter

  clusterNode に格納されている１つの二次曲面で近似されたメッシュ群の重心を計算する関数．引数の id は cluster 番号を指す.

- cnoid::Vector3 calcBoundaryCenter

  １つの二次曲面に近似されたクラスタと，他のクラスタとの境界点群の重心を導出する関数.

- void LimitedInitialClustersFromOneFace()

  ある限定的な範囲で initalClusterFromFace を行う関数．この initialClusterFromFace は，初期状態で，メッシュにクラスタを振り分ける関数．LimitedInitialClusterFromOneFace で，クラスタを振り分けるメッシュの範囲を指定し，二次曲面近似を行うと、その範囲のみの二次曲面近似が可能となる．

  この限定する範囲は、lim_x, lim_y, lim_z により現在は手動で与えている．