PyBulletクイックスタートガイド[Erwin Coumans](https://twitter.com/erwincoumans), [Yunfei Bai](http://yunfei-bai.com)[デスクトップドキュメント](http://goo.gl/QwJnFX)、[フォーラム](https://pybullet.org/Bullet/phpBB3)、星の[Bulletをご覧](https://github.com/bulletphysics/bullet3), 2016-2020ください。

[序章 2](#_Toc60399659)

[Hello PyBullet World 3](#_Toc60399660)

[connect, disconnect 3](#_Toc60399661)

[setGravity 7](#_Toc60399662)

[loadURDF, loadSDF, loadMJCF 8](#_Toc60399663)

[saveState, saveBullet, restoreState 11](#_Toc60399664)

[createCollisionShape/VisualShape 12](#_Toc60399665)

[createMultiBody 16](#_Toc60399666)

[stepSimulation 17](#_Toc60399667)

[getBasePositionAndOrientation 19](#_Toc60399668)

[resetBasePositionAndOrientation 19](#_Toc60399669)

[Transforms: Position and Orientation 20](#_Toc60399670)

[ロボットの制御 22](#_Toc60399671)

[Base, Joints, Links 22](#_Toc60399672)

[getNumJoints, getJointInfo 23](#_Toc60399673)

[setJointMotorControl2/Array 25](#_Toc60399674)

[getJointState(s)、resetJointState 29](#_Toc60399675)

[enableJointForceTorqueSensor 31](#_Toc60399676)

[getLinkState(s) 32](#_Toc60399677)

[getBaseVelocity, resetBaseVelocity 34](#_Toc60399678)

[applyExternalForce/Torque 35](#_Toc60399679)

[getNumBodies, getBodyInfo, getBodyUniqueId, removeBody 35](#_Toc60399680)

[createConstraint, removeConstraint, changeConstraint 36](#_Toc60399681)

[getNumConstraints, getConstraintUniqueId 38](#_Toc60399682)

[getConstraintInfo/State 38](#_Toc60399683)

[getDynamicsInfo/changeDynamics 39](#_Toc60399684)

[setTimeStep 42](#_Toc60399685)

[setPhysicsEngineParameter 42](#_Toc60399686)

[resetSimulation 45](#_Toc60399687)

[startStateLogging/stopStateLogging 45](#_Toc60399688)

[変形性と布(FEM, PBD) 48](#_Toc60399689)

[loadSoftBody/loadURDF 48](#_Toc60399690)

[createSoftBodyAnchor 50](#_Toc60399691)

[合成カメラレンダリング 50](#_Toc60399692)

[computeView/ProjectionMatrix 51](#_Toc60399693)

[getCameraImage 52](#_Toc60399694)

[getVisualShapeData 54](#_Toc60399695)

[changeVisualShape, loadTexture 55](#_Toc60399696)

[衝突検出クエリ 56](#_Toc60399697)

[getOverlappingObjects, getAABB 56](#_Toc60399698)

[getContactPoints, getClosestPoints 57](#_Toc60399699)

[rayTest, rayTestBatch 59](#_Toc60399700)

[getCollisionShapeData 61](#_Toc60399701)

[Enable/Disable Collisions 61](#_Toc60399702)

[逆動力学、運動学 64](#_Toc60399703)

[calculateInverseDynamics(2) 64](#_Toc60399704)

[calculateJacobian, MassMatrix 65](#_Toc60399705)

[calculateInverseKinematics(2) 66](#_Toc60399706)

[強化学習ジム環境 69](#_Toc60399707)

[環境とデータ 69](#_Toc60399708)

[Stable Baselines & ARS, ES,... 73](#_Toc60399709)

[バーチャルリアリティ 76](#_Toc60399710)

[getVREvents,setVRCameraState 76](#_Toc60399711)

[Debug GUI, Lines, Text, Parameters 78](#_Toc60399712)

[addUserDebugLine, Text, Parameter 78](#_Toc60399713)

[addUserData 82](#_Toc60399714)

[configureDebugVisualizer 82](#_Toc60399715)

[get/resetDebugVisualizerCamera 83](#_Toc60399716)

[getKeyboardEvents, getMouseEvents 84](#_Toc60399717)

[プラグイン 86](#_Toc60399718)

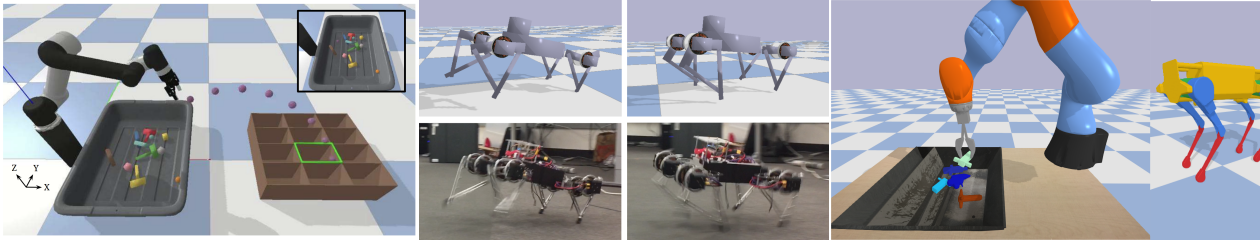
[loadPlugin,executePluginCommand 86](#_Toc60399719)

[PyBullet のビルドとインストール 87](#_Toc60399720)

[サポート、ヒント、引用 90](#_Toc60399721)

# 序章

PyBulletは、シムからリアルへの転送に焦点を当てた、ロボットシミュレーションと機械学習のための高速で使いやすいPythonモジュールです。PyBulletを使用すると、URDF、SDF、MJCF、その他のファイルフォーマットから多関節体をロードすることができます。PyBulletは、順方向力学シミュレーション、逆方向力学計算、順方向運動学および逆方向運動学、衝突検出、および光線交差クエリを提供します。[Bullet Physics SDKには](http://github.com/bulletphysics/bullet3)、シミュレーションされたミニタウロスの四足歩行、TensorFlow推論を使用したヒューマノイド、物体を把持するKUKAアームなどのPyBulletロボットの例が含まれています。還元座標マルチボディ、リジッドボディ、変形可能性は、本[論文](https://drive.google.com/file/d/0Bz3vEa19XOYGNWdZWGpMdUdqVmZ5ZVBOaEh4ZnpNaUxxZFNV/view?usp=sharing)と同様の統一されたLCP制約ソルバーによって処理されます。



物理シミュレーションの他に、レンダリングへのバインディングがあり、CPUレンダラ（TinyRenderer）とOpenGL 3.xのレンダリングと可視化、HTC ViveやOculus Riftなどのバーチャルリアリティヘッドセットへの対応などがあります。また、PyBulletは、衝突検出クエリ（最も近い点、重なり合うペア、レイの交差テストなど）を実行したり、デバッグレンダリング（デバッグラインやテキスト）を追加したりする機能も持っています。PyBulletは、共有メモリ、UDPおよびTCPネットワークのためのクロスプラットフォームの組み込みクライアント・サーバをサポートしています。そのため、WindowsのVRサーバーに接続してLinux上でPyBulletを実行することができます。

PyBullet は新しい [Bullet C-API](https://github.com/bulletphysics/bullet3/blob/master/examples/SharedMemory/PhysicsClientC_API.h) をラップしています。これは、基礎となる物理エンジンやレンダリングエンジンから独立して設計されているため、新しいバージョンの Bullet に簡単に移行したり、別の物理エンジンやレンダリングエンジンを使用したりすることができます。デフォルトでは、PyBullet は CPU 上で Bullet 2.x API を使用します。OpenCLを使用してGPU上で動作するBullet 3.xも公開する予定です。PyBulletに似たC++ APIもあります。

PyBulletは、TensorFlowやOpenAI Gymを使って簡単に利用することができます。[Google Brain](https://sites.google.com/corp/view/graspgan) [[1](https://arxiv.org/abs/1804.10332),[2](https://sites.google.com/corp/view/graspgan),[3](https://xcyan.github.io/geoaware_grasping/),[4](https://arxiv.org/abs/1712.07642)]、[X](https://sites.google.com/corp/view/multi-task-domain-adaptation)[[1](https://sim2real.github.io/assets/slides/bai-Learning_to_Grasp_Using_Simulation_Yunfei_Bai_Google_X.pdf),[2](https://www.linkedin.com/pulse/speeding-up-robot-learning-100x-simulation-mrinal-kalakrishnan/)]、Stanford AI Lab [[1](https://stanfordvl.github.io/ntp/?utm_content=buffer8b1fc),[2](http://gibsonenv.stanford.edu),[3](http://gibson.vision/)]、[OpenAI](https://blog.openai.com/roboschool/),INRIA [[1](https://openlab-flowers.inria.fr/t/state-representation-learning-for-reinforcement-learning-srl-for-rl/405)]など[の研究](https://scholar.google.com/scholar?oi=bibs&hl=en&cites=14155418983690238411,15548795680127842422,12750085261778447071)者がPyBulletを利用しています。研究でPyBulletを使用している場合は、[引用](#c5k8j1j6kny9)を追加してください。

PyBulletのインストールは、(sudo) pip install PyBullet (Python 2.x)、pip3 install PyBulletと簡単です。これでPyBulletモジュールとpybullet\_envs Gym環境が公開されます。

## Hello PyBullet World

ここではPyBulletの導入スクリプトを一歩一歩説明します。

import pybullet as p

import time

import pybullet\_data

physicsClient = p.connect(p.GUI)#or p.DIRECT for non-graphical version

p.setAdditionalSearchPath(pybullet\_data.getDataPath()) #optionally

p.setGravity(0,0,-10)

planeId = p.loadURDF("plane.urdf")

cubeStartPos = [0,0,1]

cubeStartOrientation = p.getQuaternionFromEuler([0,0,0])

boxId = p.loadURDF("r2d2.urdf",cubeStartPos, cubeStartOrientation)

for i in range (10000):

p.stepSimulation()

time.sleep(1./240.)

cubePos, cubeOrn = p.getBasePositionAndOrientation(boxId)

print(cubePos,cubeOrn)

p.disconnect()

## connect, disconnect

PyBulletモジュールをインポートした後、最初に行うべきことは物理シミュレーションへの「接続」です。PyBulletは、クライアントがコマンドを送信し、物理サーバーがステータスを返します。PyBulletには、いくつかの物理サーバーが組み込まれています。DIRECTとGUIである。GUI接続とDIRECT接続の両方とも、PyBulletと同じプロセスで物理シミュレーションとレンダリングを実行します。

DIRECT モードでは、"Virtual Reality"と"Debug GUI, Lines, Text, Parameters"の章で説明されているように、OpenGL と VR ハードウェア機能にアクセスできないことに注意してください。DIRECTモードでは、'getCameraImage' APIを使用して、内蔵のソフトウェアレンダラを使用して画像をレンダリングすることができます。これは、GPUのないサーバー上のクラウド上でシミュレーションを実行するのに便利です。

独自のデータファイルを提供することもできますし、PyBulletに同梱されているPyBullet\_dataパッケージを使用することもできます。そのためには、pybullet\_dataをインポートし、pybullet.setAdditionalSearchPath(pybullet\_data.getDataPath())でディレクトリを登録します。

getConnectionInfo

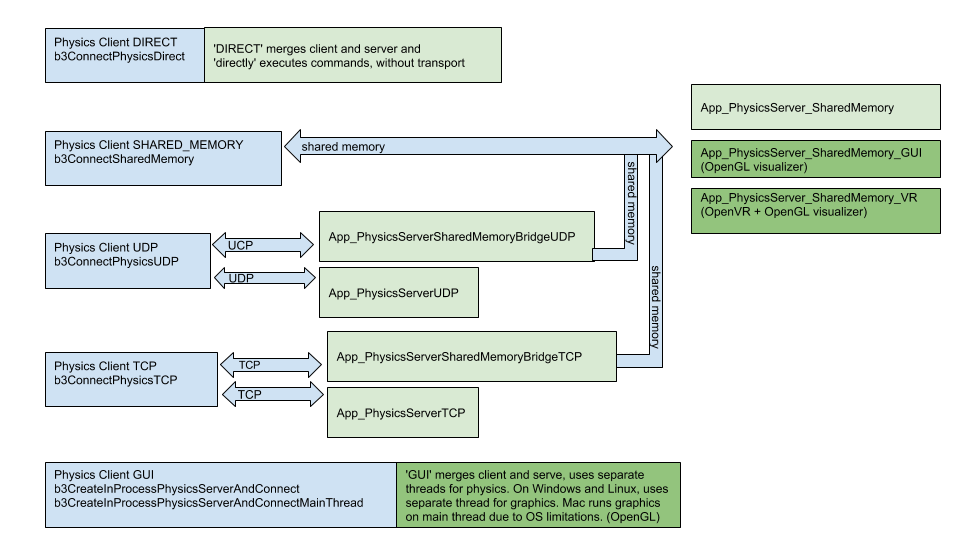
物理クライアントIDが与えられると、リスト[isConnected, connectionMethod]を返します。

isConnected

isConnected は接続されていれば true を返し、そうでなければ false を返します。

setTimeOut

特定のタイムアウト値内にコマンドがサーバーによって処理されない場合、クライアントは切断されます。この値を秒単位で指定するには、setTimeOut を使用します。



様々な物理クライアント(青)と物理サーバー(緑)のオプションを持つ図。濃い緑色のサーバーは、OpenGLデバッグの可視化を提供します。

connect using DIRECT, GUI

DIRECT 接続では、トランスポート レイヤーやグラフィック ビジュアライゼーション ウィンドウを使用せずに、物理エンジンに直接コマンドを送信し、コマンド実行後のステータスを直接返します。

GUI接続は、PyBulletと同じプロセス空間内に、3D OpenGLレンダリングの新しいグラフィカルユーザーインターフェース（GUI）を作成します。LinuxとWindowsでは、このGUIは別のスレッドで実行されますが、OSXではオペレーティングシステムの制限により同じスレッドで実行されます。Mac OSXでは、'stepSimulation'または他のPyBulletコマンドを実行するまで、OpenGLウィンドウで回転しているホイールが見えるかもしれません。

コマンドとステータスメッセージは、PyBulletクライアントとGUI物理シミュレーションサーバの間で、通常のメモリバッファを使用して送信されます。

また、SHARED\_MEMORY、UDP、TCP ネットワークを使用して、同じマシン上の別のプロセスやリモートマシン上の物理サーバーに接続することも可能です。詳細は共有メモリ、UDP、TCP についてのセクションを参照してください。

他のほとんどすべてのメソッドとは異なり、このメソッドは下位互換性のためにキーワード引数をパースしません。

connectの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | connection mode | integer: DIRECT, GUI, SHARED\_MEMORY, UDP, TCP  GUI\_SERVER, SHARED\_MEMORY\_SERVER, SHARED\_MEMORY\_GUI | DIRECTモードは新しい物理エンジンを作成し、それと直接通信します。GUI はグラフィカルな GUI フロントエンドを持つ物理エンジンを作成し、それと通信します。SHARED\_MEMORY は、同じマシン上の既存の物理エンジンプロセスに接続し、共有メモリを介して通信します。TCP または UDP は、TCP または UDP ネットワークを介して既存の物理サーバーに接続します。  GUI\_SERVERはGUIに似ていますが、外部のSHARED\_MEMORY接続を許可するサーバとしても動作します。SHARED\_MEMORY\_SERVER は DIRECT に似ていますが、外部の SHARED\_MEMORY 接続を許可するサーバとしても動作します。SHARED\_MEMORY\_GUI は DIRECT に似ていますが、表示のために外部のグラフィックサーバーへの接続を試みます。Bullet ExampleBrowserには、Physics ServerまたはGraphics Serverとして動作するオプションがあります。 |
| optional | key | int | SHARED\_MEMORY モードでは、オプションの共有メモリキーを使用します。ExampleBrowserやSharedMemoryPhysics\_\*を起動する際に、オプションのコマンドライン--shared\_memory\_keyでキーを設定することができます。これにより、同一マシン上で複数のサーバを動作させることができます。 |
| optional | hostName  (UDP and TCP) | string | IP アドレスまたはホスト名、例えば「127.0.0.0.1」または「localhost」または「mymachine.domain.com」。 |
| optional | port  (UDP and TCP) | integer | UDP ポート番号。デフォルトの UDP ポートは 1234、デフォルトの TCP ポートは 6667 です (サーバのデフォルトと一致します)。 |
| optional | options | string | コマンドラインオプションを GUI サーバに渡すことができます。現時点では --opengl2 フラグのみが有効になっています。デフォルトでは、Bullet は OpenGL3 を使用しますが、仮想マシンやリモートデスクトップクライアントなどの環境によっては OpenGL2 のみをサポートしています。現時点では、コマンドライン引数は1つしか渡すことができません。 |

connect は物理クライアント ID を、接続されていない場合は -1 を返します。物理クライアントIDは、他のほとんどのPyBulletコマンドのオプション引数です。GUI を除いて、複数の異なる物理サーバに接続することができます。

例えば

pybullet.connect(pybullet.DIRECT)

pybullet.connect(pybullet.GUI, options="--opengl2")

pybullet.connect(pybullet.SHARED\_MEMORY,1234)

pybullet.connect(pybullet.UDP,"192.168.0.1")

pybullet.connect(pybullet.UDP,"localhost", 1234)

pybullet.connect(pybullet.TCP,"localhost", 6667)

共有メモリを使用して接続する

共有メモリ接続を許可している物理サーバーはいくつかあります: App\_SharedMemoryPhysics、App\_SharedMemoryPhysics\_GUI、そしてBullet Example BrowserにはExperimental/Physics Serverの下に共有メモリ接続を許可している例が1つあります。これで物理シミュレーションとレンダリングを別プロセスで実行できるようになります。

また、HTC ViveやOculus RiftのようなヘッドマウントディスプレイやTouchコントローラーを搭載した6-dofのトラッキングコントローラーに対応したバーチャルリアリティアプリ「App\_SharedMemoryPhysics\_VR」に共有メモリを介して接続することもできます。Valve OpenVR SDKはWindows環境下でしか正常に動作しないため、App\_SharedMemoryPhysics\_VRはWindows環境下ではpremake（できれば）かcmakeを使ってしかビルドできません。

ＵＤＰまたはＴＣＰネットワークで接続する

UDP ネットワーキングのために、特定の UDP ポートをリッスンする App\_PhysicsServerUDP があります。これは信頼性の高い UDP ネットワーキングのためにオープンソースの [enet](http://enet.bespin.org) ライブラリを使用しています。これにより、物理シミュレーションとレンダリングを別のマシンで実行することができます。TCPについては、PyBulletは[clsocket](https://github.com/DFHack/clsocket)ライブラリを使用しています。これは、ファイアウォール内のマシンからロボットシミュレーションにSSHトンネリングを使用する場合に便利です。例えば、Linux上でPyBulletを使って制御スタックや機械学習を実行し、Windows上でHTC ViveやRiftを使ってバーチャルリアリティで物理サーバを実行することができます。

もう一つの UDP アプリケーションは App\_PhysicsServerSharedMemoryBridgeUDP アプリケーションで、既存の物理サーバとのブリッジとして機能する。同様の方法で、TCP バージョンもあります (UDP を TCP に置き換えてください)。

GRPC クライアントとサーバのサポートもありますが、これはデフォルトでは有効になっていません。premake4 のビルドシステムで --enable\_grpc オプションを使って試してみることができます (Bullet/build3/premake4 を参照してください)。

注意: 現時点では、クライアントとサーバーの両方が32bitか64bitのビルドである必要があります。

disconnect

物理サーバーから切断するには、connect コールで返された物理クライアント ID を使用します (負の値でない場合)。DIRECT」または「GUI」物理サーバはシャットダウンします。別の (プロセス外の) 物理サーバは実行を継続します。すべての項目を削除するには 'resetSimulation' も参照してください。

切断のパラメータ。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | physicsClientId | int | 複数の物理サーバーに接続している場合は、どれを選ぶことができます。 |

## setGravity

デフォルトでは、重力は有効になっていません。 *setGravityで*は、すべてのオブジェクトのデフォルトの重力を設定します。

setGravityの入力パラメータは以下の通りです。(戻り値なし)

|  |  |  |  |
| --- | --- | --- | --- |
| required | graX | float | Ｘ世界軸に沿った重力力 |
| required | gravY | float | Ｙ世界軸に沿った重力力 |
| required | gravZ | float | Ｚ世界軸に沿った重力力 |
| optional | physicsClientId | int | 複数の物理サーバーに接続している場合は、どれを選ぶことができます。 |

## loadURDF, loadSDF, loadMJCF

loadURDFは物理サーバにコマンドを送り、ユニバーサルロボット記述ファイル(URDF)から物理モデルをロードします。URDF ファイルは ROS プロジェクト (Robot Operating System) でロボットやその他のオブジェクトを記述するために使用されており、WillowGarage と Open Source Robotics Foundation (OSRF) によって作成されました。多くのロボットが公開されている URDF ファイルを持っています。説明とチュートリアルは [http://wiki.ros.org/urdf/Tutorials を参照してください。](http://wiki.ros.org/urdf/Tutorials)

重要な注意：ほとんどのジョイント（スライダ、リボリュート、連続）には、デフォルトでモータが有効になっており、自由な動きができないようになっています。これは、非常に高摩擦のハーモニック・ドライブを持つロボット・ジョイントに似ています。pybullet.setJointMotorControl2を使用して、ジョイントのモータ制御モードとターゲットの設定を行う必要があります。詳細はsetJointMotorControl2 APIを参照してください。

警告: デフォルトでは、PyBullet は読み込みを高速化するためにいくつかのファイルをキャッシュします。ファイルのキャッシュを無効にするには、setPhysicsEngineParameter(enableFileCaching=0) を使用します。

loadURDFの引数は

|  |  |  |  |
| --- | --- | --- | --- |
| required | fileName | string | 物理サーバのファイルシステム上の URDF ファイルへの相対パスまたは絶対パス。 |
| optional | basePosition | vec3 | 世界空間座標[X,Y,Z]の指定された位置にオブジェクトの基底を作成します。この位置はURDFリンクの位置であることに注意してください。慣性フレームが0以外の場合は、質量中心位置とは異なります。質量中心の位置/向きを設定するには、resetBasePositionAndOrientationを使用します。 |
| optional | baseOrientation | vec4 | 指定した方向にオブジェクトの基底をワールドスペースクォータニオン[X,Y,Z,W]として作成します。basePosition の注釈を参照してください。 |
| optional | useMaximalCoordinates | int | 実験用です。デフォルトでは、URDFファイル内のジョイントは縮小座標法を使用して作成されます。ジョイントは、Featherstone Articulated Body Algorithm (ABA, btMultiBody in Bullet 2.x)を使用してシミュレートされます。useMaximalCoordinatesオプションを指定すると、各リンクに対して6自由度の剛体が作成され、それらの剛体間の制約がジョイントのモデル化に使用されます。 |
| optional | useFixedBase | optional | ロードされたオブジェクトの基部を強制的に静的にします。 |
| optional | flags | optional | 以下のフラグは、ビット単位のOR、｜を使用して組み合わせることができます。  URDF\_MERGE\_FIXED\_LINKS: これはURDFファイルから固定リンクを削除し、結果のリンクをマージします。これは、様々なアルゴリズム（多関節体アルゴリズム、前方運動学など）が、固定関節を含む関節の数が線形の複雑さを持っているので、パフォーマンスの面では良いです。  URDF\_USE\_INERTIA\_FROM\_FILE: デフォルトでは、Bulletは衝突形状の質量と体積に基づいて慣性テンソルを再計算します。より正確な慣性テンソルを提供できる場合は、このフラグを使用してください。  URDF\_USE\_SELF\_COLLISION: デフォルトでは、Bullet はセルフコリジョンを無効にします。このフラグを使用して有効にすることができます。セルフコリジョンの動作は、以下のフラグを使ってカスタマイズすることができます。  URDF\_USE\_SELF\_COLLISION\_INCLUDE\_PARENT は、子と親の衝突を有効にします。URDF\_USE\_SELF\_COLLISION フラグと一緒に使う必要があります。  URDF\_USE\_SELF\_COLLISION\_EXCLUDE\_ALL\_PARENTS は、子リンクとその祖先(親、親の親、親の親、基底まで)との間の自己撞着を破棄します。URDF\_USE\_SELF\_COLLISIONと一緒に使う必要があります。  URDF\_USE\_IMPLICIT\_CYLINDER を指定すると、滑らかな暗黙の円柱を使用します。デフォルトでは、Bulletはシリンダーを凸状の外皮にテッセレーションします。  URDF\_ENABLE\_SLEEPING は、ボディがしばらく動かなかった後にシミュレーションを無効にします。アクティブな身体との相互作用により、シミュレーションを再び有効にします。  URDF\_INITIALIZE\_SAT\_FEATURES は、凸形状用の三角形メッシュを作成します。これにより可視化が改善され、GJK/EPA の代わりに分離軸テスト (SAT) を使用できるようになります。setPhysicsEngineParameter を使って SAT を有効にする必要があります。  URDF\_USE\_MATERIAL\_COLORS\_FROM\_MTL は、URDF ファイルからではなく、Wavefront OBJ ファイルからの RGB カラーを使用します。  URDF\_ENABLE\_CACHED\_GRAPHICS\_SHAPES は、図形をキャッシュして再利用します。これは、類似のグラフィックアセットを持つファイルの読み込み性能を向上させます。  URDF\_MAINTAIN\_LINK\_ORDER は、URDF ファイルのリンク順を維持しようとします。URDFファイルでは、以下のようになっています。ParentLink0, ChildLink1 (ParentLink0 に接続されている)、ChildLink2 (ParentLink0 に接続されている) となります。このフラグがなければ、順番は P0, C2, C1 となります。 |
| optional | globalScaling | float | globalScaling は、URDF モデルにスケールファクタを適用します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

loadURDFは、ボディユニークIDを返します。URDF ファイルを読み込むことができない場合、この整数は負の値となり、有効なボディユニーク ID とはなりません。

デフォルトでは、loadURDF はメッシュの衝突検出に凸型の外皮を使用します。静的な (質量 = 0, 移動しない) メッシュの場合、URDF にタグを追加することでメッシュを凹にすることができます。

<link concave="yes" name="baseLink"> [例](https://github.com/bulletphysics/bullet3/blob/master/data/samurai.urdf)はsamurai.urdfを参照してください。URDFフォーマットには他にもいくつかの拡張機能があります。PyBulletは、URDFファイルからすべての情報を処理するわけではありません。どのような機能がサポートされているかについては、例とURDFファイルを参照してください。通常、機能を制御するためのPython APIがあります。各リンクは単一のマテリアルしか持てないので、異なるマテリアルを持つ複数のビジュアルシェイプがある場合は、固定ジョイントで接続された別々のリンクに分割する必要があります。これを行うには、Bulletの一部であるOBJ2SDFユーティリティを使用することができます。

loadSDF, loadMJCF

また、.bullet、.sdf、.mjcfなどの他のファイルフォーマットからもオブジェクトをロードすることができます。 これらのファイルフォーマットは複数のオブジェクトをサポートしているので、戻り値はオブジェクトのユニークなIDのリストになります。SDF フォーマットについては <http://sdformat.org>で詳しく説明しています。loadSDFコマンドは、ロボットモデルとジオメトリに関連するSDFの一部の重要な部分のみを抽出し、カメラやライトなどに関連する多くの要素は無視します。loadMJCFコマンドは、OpenAI Gymで使用されているMuJoCo MJCF xmlファイルの基本的なインポートを行います。デフォルトのジョイントモーター設定に関連するloadURDFの下の重要な注意事項も参照して、setJointMotorControl2を必ず使用してください。

|  |  |  |  |
| --- | --- | --- | --- |
| required | fileName | string | 物理サーバのファイルシステム上の URDF ファイルへの相対パスまたは絶対パス。 |
| optional | useMaximalCoordinates | int | 実験的なものです。詳細は loadURDF を参照してください。 |
| optional | globalScaling | float | globalScalingはSDFとURDFでサポートされていますが、MJCFではサポートされていません。すべてのオブジェクトは、このスケールファクタを使用してスケーリングされます（リンク、リンクフレーム、ジョイントアタッチメント、リニアジョイントリミットを含む）。これは質量には影響せず、ジオメトリにのみ影響します。必要に応じて質量を変更するには changeDynamics を使用してください。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

loadBullet、loadSDF、loadMJCFはオブジェクト固有のidの配列を返します。

|  |  |  |
| --- | --- | --- |
| objectUniqueIds | list of int | リストには、読み込まれた各オブジェクトのオブジェクト固有のIDが含まれます。 |

## saveState, saveBullet, restoreState

以前に保存した状態に復元した後に決定論的なシミュレーションが必要な場合、接点を含むすべての重要な状態情報を保存する必要があります。これには、saveWorldコマンドだけでは不十分です。saveState（メモリ内）またはsaveBullet（ディスク上）を使用して撮影したスナップショットから復元するには、restoreStateコマンドを使用することができます。

saveStateコマンドは、オプションのclientServerIdを入力として受け取り、ステートIDを返すだけです。

saveBulletコマンドは、ディスク上の.bulletファイルに状態を保存します。

restoreState コマンドの入力引数は

|  |  |  |  |
| --- | --- | --- | --- |
| optional | fileName | string | saveBulletコマンドを使用して作成された.bulletファイルのファイル名。 |
| optional | stateId | int | saveStateが返す状態ID |
| optional | clientServerId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

ファイル名またはステート ID のいずれかが有効である必要があります。restoreState は、オブジェクトの位置と関節角度を保存された状態にリセットし、接触点情報を復元することに注意してください。restoreState を呼び出す前に、オブジェクトと制約が設定されていることを確認する必要があります。[saveRestoreState.py の](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/saveRestoreState.py)例を参照してください。

removeState

removeState は、以前に保存された状態をメモリから削除することができます。

saveWorld

サーバーに保存されているPyBullet Pythonファイルとして、現在の世界のおおよそのスナップショットを作成することができます。 saveWorldは基本的な編集機能として、例えばVRでロボットや関節角度、オブジェクトの位置や環境を設定するのに役立ちます。後でPyBullet Pythonファイルをロードするだけで、ワールドを再作成することができます。pythonのスナップショットには、関節角度やオブジェクトの変形の初期化とともにloadURDFコマンドが含まれています。すべての設定がワールドファイルに保存されているわけではないことに注意してください。

入力引数は

|  |  |  |  |
| --- | --- | --- | --- |
| required | fileName | string | filename of the PyBullet file. |
| optional | clientServerId | int | if you are connected to multiple servers, you can pick one |

## createCollisionShape/VisualShape

世界にあるものを作成するための推奨された最も簡単な方法はロード関数(loadURDF/SDF/MJCF/Bullet)を使用することですが、コリジョンやビジュアルシェイプをプログラムで作成し、createMultiBodyを使用してマルチボディを作成することもできます。Bullet Physics SDKの[createMultiBodyLinks.py](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createMultiBodyLinks.py)と[createVisualShape.pyの](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createVisualShape.py)例を参照してください。

createCollisionShapeの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | shapeType | int | GEOM\_SPHERE, GEOM\_BOX, GEOM\_CAPSULE, GEOM\_CYLINDER, GEOM\_PLANE, GEOM\_MESH, GEOM\_HEIGHTFIELD |
| optional | radius | float | デフォルト 0.5: GEOM\_SPHERE, GEOM\_CAPSULE, GEOM\_CYLINDER |
| optional | halfExtents | vec3 list of 3 floats | default [1,1,1]：GEOM\_BOX用 |
| optional | height | float | default: 1: GEOM\_CAPSULE, GEOM\_CYLINDERの場合 |
| optional | fileName | string | GEOM\_MESHのファイル名、現在はWavefront .objのみ。.objファイル内の各オブジェクト('o'とマークされたもの)に対して、凸状のハルを作成します。 |
| optional | meshScale | vec3 list of 3 floats | デフォルトでは1,1,1,1]、GEOM\_MESHの場合 |
| optional | planeNormal | vec3 list of 3 floats | デフォルトではGEOM\_PLANEの場合は[0,0,1]です。 |
| optional | flags | int | GEOM\_FORCE\_CONCAVE\_TRIMESH: GEOM\_MESH の場合、静的な三角形の凹面メッシュを作成します。これは動的 / 移動するオブジェクトには使用しないでください。 |
| optional | collisionFramePosition | vec3 | リンクフレームに対する衝突形状の並進オフセット |
| optional | collisionFrameOrientation | vec4 | リンクフレームに対する衝突形状の回転オフセット |
| optional | vertices | list of vec3 | の定義を参照してください。[heightfield.pyの](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/heightfield.py)例を参照してください。 |
| optional | indices | list of int | ハイトフィールドの定義 |
| optional | heightfieldTextureScaling | float | 高さフィールドのテクスチャのスケーリング |
| optional | numHeightfieldRows | int | ハイトフィールドの定義 |
| optional | numHeightfieldColumns | int | ハイトフィールドの定義 |
| optional | replaceHeightfieldIndex | int | 既存のハイトフィールドの置き換え(ハイトフィールドの高さを更新) (ハイトフィールドを削除して再作成するよりもはるかに速い) |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つを選ぶことができます。 |

戻り値は、衝突形状のための非負の int 一意の ID であり、呼び出しに失敗した場合は -1 となります。

createCollisionShapeArray

collisionShapeArray は createCollisionShape の配列版です。使い方は snake.py や createVisualShapeArray.py の例を参照してください。

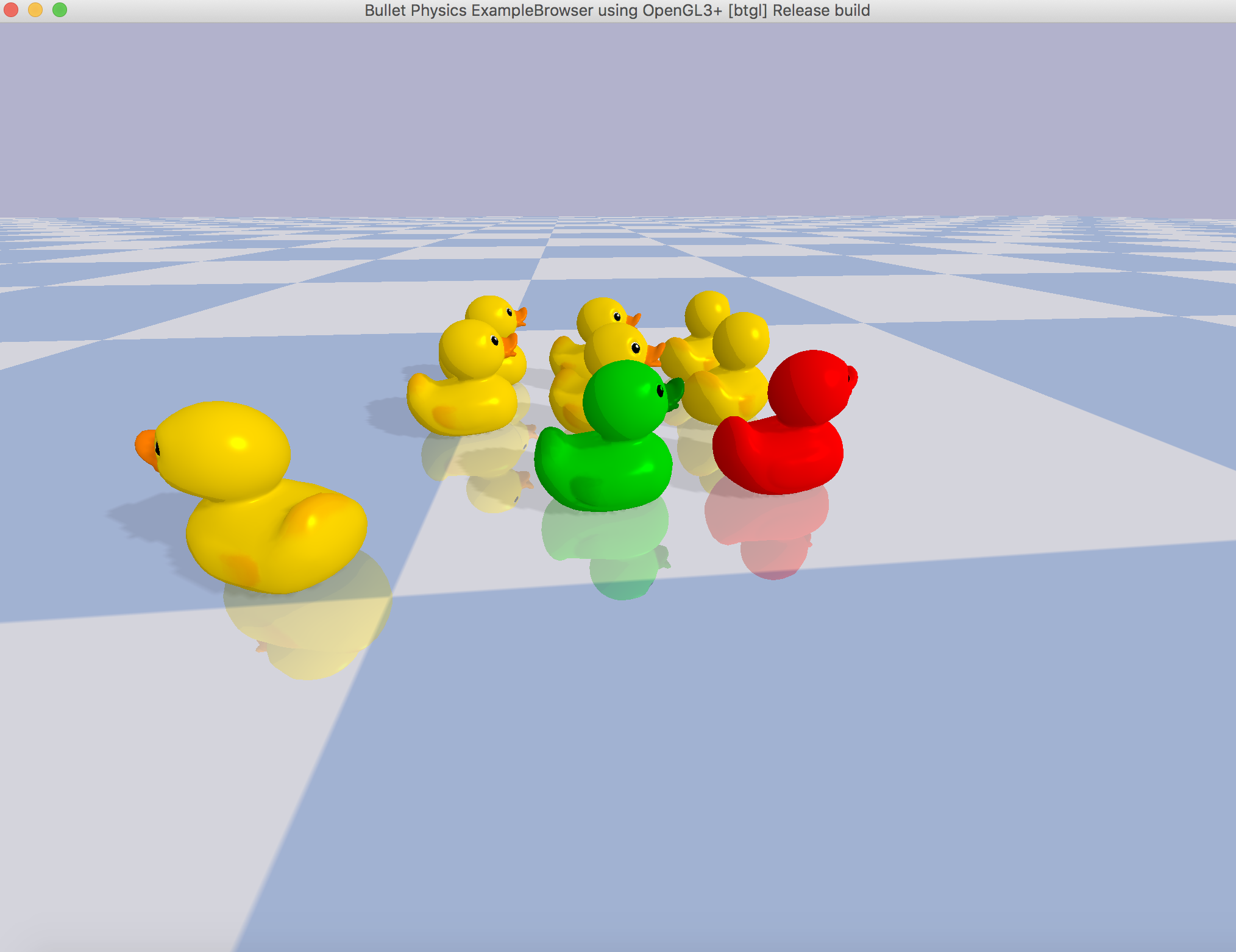
removeCollisionShape

removeCollisionShapeは、既存のコリジョン形状を、そのコリジョン形状固有のIDを使用して削除します。

createVisualShape

ビジュアル・シェイプは、コリジョン・シェイプの作成と同様の方法で作成できますが、拡散色やスペキュラー・カラーなどの視覚的な外観を制御するための引数を追加して作成することができます。GEOM\_MESHタイプを使用する場合、Wavefront OBJファイルを指定すると、ビジュアルシェイプがマテリアルファイル(.mtl)からいくつかのパラメータを解析し、テクスチャをロードします。大きなテクスチャ（1024x1024ピクセル以上）は、読み込みやランタイムのパフォーマンスが遅くなることに注意してください。

examples/pybullet/examples/addPlanarReflection.pyとcreateVisualShape.pyを参照してください。



入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | shapeType | int | GEOM\_SPHERE, GEOM\_BOX, GEOM\_CAPSULE, GEOM\_CYLINDER, GEOM\_PLANE, GEOM\_MESH |
| optional | radius | float | デフォルト 0.5: GEOM\_SPHERE, GEOM\_CAPSULE, GEOM\_CYLINDER のみ |
| optional | halfExtents | vec3 list of 3 floats | default [1,1,1,1]: GEOM\_BOXのみ |
| optional | length | float | default: 1: GEOM\_CAPSULE, GEOM\_CYLINDER (length = height) のみ。 |
| optional | fileName | string | GEOM\_MESHのファイル名、現在はWavefront .objのみ。.objファイル内の各オブジェクト('o'とマークされたもの)に対して、凸状のハルを作成します。 |
| optional | meshScale | vec3 list of 3 floats | デフォルトは1,1,1,1]、GEOM\_MESHのみ |
| optional | planeNormal | vec3 list of 3 floats | デフォルトでは0,0,1] GEOM\_PLANEのみ |
| optional | flags | int | 未使用／未定 |
| optional | rgbaColor | vec4, list of 4 floats | 赤, 緑, 青, アルファの色成分, それぞれの範囲は [0...1] です. |
| optional | specularColor | vec3, list of 3 floats | 鏡面反射色, 赤, 緑, 青成分の範囲 [0〜1] |
| optional | visualFramePosition | vec3, list of 3 floats | リンクフレームに対する視覚形状の並進オフセット |
| optional | vertices | list of vec3 | objファイルからメッシュを作成する代わりに、頂点、インデックス、UVS、法線を指定することができます。 |
| optional | indices | list of int | 三角形のインデックスは、3の倍数でなければなりません。 |
| optional | uvs | list of vec2 | 頂点のUVテクスチャ座標。テクスチャ画像を選択するには changeVisualShape を使用します。uvsの数は頂点の数と同じにする必要があります。 |
| optional | normals | list of vec3 | 頂点の法線、数は頂点の数と同じでなければなりません。 |
| optional | visualFrameOrientation | vec4, list of 4 floats | リンクフレームに対する視覚形状の回転オフセット |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つを選ぶことができます。 |

戻り値は、ビジュアルシェイプのための非負の int 一意の ID であり、呼び出しに失敗した場合は -1 となります。

[createVisualShape](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createVisualShape.py)、[createVisualShapeArray](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createVisualShapeArray.py)、[createTexturedMeshVisualShapeの](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createTexturedMeshVisualShape.py)例を参照してください。

createVisualShapeArray

createVisualShapeArray は createVisualShape の配列版です。createVisualShapeArray.pyの例を参照してください。

## createMultiBody

世界で一番簡単に物を作る方法はローディング関数(loadURDF/SDF/MJCF/Bullet)を使うことですが、createMultiBodyを使ってマルチボディを作ることができます。

Bullet Physics SDKの[createMultiBodyLinks.pyの](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createMultiBodyLinks.py)例を参照してください。createMultiBodyのパラメータは、URDFとSDFのパラメータに非常に似ています。

ジョイント/子リンクを設けずにベースだけでマルチボディを作成することもできますし、ジョイント/子リンクを設けてマルチボディを作成することもできます。リンクを提供する場合は、すべてのリストのサイズが同じであることを確認してください(len(linkMasses) == len(linkCollisionShapeIndices) など)。createMultiBodyの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | baseMass | float | mass of the base, in kg (if using SI units) |
| optional | baseCollisionShapeIndex | int | unique id from createCollisionShape or -1. You can re-use the collision shape for multiple multibodies (instancing) |
| optional | baseVisualShapeIndex | int | unique id from createVisualShape or -1. You can reuse the visual shape (instancing) |
| optional | basePosition | vec3, list of 3 floats | Cartesian world position of the base |
| optional | baseOrientation | vec4, list of 4 floats | Orientation of base as quaternion [x,y,z,w] |
| optional | baseInertialFramePosition | vec3, list of 3 floats | Local position of inertial frame |
| optional | baseInertialFrameOrientation | vec4, list of 4 floats | Local orientation of inertial frame, [x,y,z,w] |
| optional | linkMasses | list of float | List of the mass values, one for each link. |
| optional | linkCollisionShapeIndices | list of int | List of the unique id, one for each link. |
| optional | linkVisualShapeIndices | list of int | list of the visual shape unique id for each link |
| optional | linkPositions | list of vec3 | list of local link positions, with respect to parent |
| optional | linkOrientations | list of vec4 | list of local link orientations, w.r.t. parent |
| optional | linkInertialFramePositions | list of vec3 | list of local inertial frame pos. in link frame |
| optional | linkInertialFrameOrientations | list of vec4 | list of local inertial frame orn. in link frame |
| optional | linkParentIndices | list of int | Link index of the parent link or 0 for the base. |
| optional | linkJointTypes | list of int | list of joint types, one for each link. JOINT\_REVOLUTE, JOINT\_PRISMATIC, JOINT\_SPHERICAL and JOINT\_FIXED types are supported at the moment. |
| optional | linkJointAxis | list of vec3 | Joint axis in local frame |
| optional | useMaximalCoordinates | int | experimental, best to leave it 0/false. |
| optional | flags | int | similar to the flags passed in loadURDF, for example URDF\_USE\_SELF\_COLLISION. See loadURDF for flags explanation. |
| optional | batchPositions | list of vec3 | array of base positions, for fast batch creation of many multibodies.See [example](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createMultiBodyBatch.py). |
| optional | physicsClientId | int | If you are connected to multiple servers, you can pick one. |

createMultiBodyの戻り値は、負ではない一意のID、失敗した場合は-1です。例

cuid = pybullet.createCollisionShape(pybullet.GEOM\_BOX, halfExtents = [1, 1, 1])

mass= 0 #スタティックボックス

pybullet.createMultiBody(mass,cuid)

Bullet/examples/pybullet/examplesフォルダのcreateMultiBodyLinks.py、createObstacleCourse.py、createVisualShape.pyも参照してください。

getMeshData

getMeshDataは、三角形メッシュのメッシュ情報（頂点、インデックス）を返すための実験的な未公開APIです。

|  |  |  |  |
| --- | --- | --- | --- |
| ひつよう | ボディユニークアイディー | インスタント | ボディユニークＩＤ |
| 任意 | リンクインデックス | インスタント | リンクインデックス |
| 任意 | 衝突形状インデックス | インスタント | リンク内に複数の衝突形状がある場合の複合形状のインデックス (getCollisionShapeData を参照) |
| 任意 | 旗 | インスタント | デフォルトでは、PyBullet はグラフィックレンダリングの頂点を返します。法線が異なる頂点は複製されるので、元のメッシュよりも多くの頂点が生成される可能性があります。flags = pybullet.MESH\_DATA\_SIMULATION\_MESHでシミュレーション頂点を受け取ることができます。 |
| 任意 | 物理クライアントId | インスタント | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

## stepSimulation

stepSimulation は、衝突検出、制約解法、統合などのすべてのアクションを 1 つのフォワードダイナミクスシミュレーションステップで実行します。デフォルトのタイムステップは 1/240 秒ですが、setTimeStep または setPhysicsEngineParameter API を使って変更することができます。

stepSimulationの入力引数はオプションです。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

デフォルトでは、stepSimulation は戻り値を持ちません。

実験用/アドバンスト用のみ：setPhysicsEngineParameter APIでreportSolverAnalyticsを有効にした場合、以下の情報が詳細な島情報の一覧として返されます。

|  |  |  |
| --- | --- | --- |
| islandId | int | island unique id |
| numBodies | list of body unique ids | the body unique ids in this island |
| numIterationsUsed | int | the number of solver iterations used. |
| remainingResidual | float | the residual constraint error. |

物理サーバにリアルタイムクロックに基づいた順方向のダイナミクスシミュレーションを自動的に実行させるための setRealTimeSimulation も参照してください。

setRealTimeSimulation

デフォルトでは、明示的に'stepSimulation'コマンドを送らない限り、物理サーバはシミュレーションをステップさせません。これにより、シミュレーションの制御の決定性を維持することができます。setRealTimeSimulationコマンドを使って、物理サーバがリアルタイムクロック(RTC)に応じて自動的にシミュレーションをステップさせることで、リアルタイムでシミュレーションを実行することができます。リアルタイムシミュレーションを有効にすれば、'stepSimulation'を呼び出す必要はありません。

大和モードでは、setRealTimeSimulationは効果がないことに注意してください。大和モードでは、物理サーバとクライアントは同じスレッドで実行され、あなたがすべてのコマンドをトリガーします。GUI モード、Virtual Reality モード、TCP/UDP モードでは、物理サーバはクライアント (PyBullet) とは別のスレッドで動作し、setRealTimeSimulation により物理サーバのスレッドは stepSimulation への呼び出しを追加することができます。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | enableRealTimeSimulation | int | 0 to disable real-time simulation, 1 to enable |
| optional | physicsClientId | int | if you are connected to multiple servers, you can pick one. |

## getBasePositionAndOrientation

getBasePositionAndOrientationは、身体の基部（またはルートリンク）の現在の位置と向きをデカルト座標で報告します。方向は [x,y,z,w] 形式の四角形です。

getBasePositionAndOrientationの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | objectUniqueId | int | object unique id, as returned from loadURDF. |
| optional | physicsClientId | int | if you are connected to multiple servers, you can pick one. |

getBasePositionAndOrientationは、位置のリストを3つのフロートで、方位を4つのフロートで[x,y,z,w]の順に返します。必要に応じて、クォータニオンをオイラーに変換するには getEulerFromQuaternion を使用します。

オブジェクトの位置と向きをリセットするには、 resetBasePositionAndOrientation も参照してください。

これで最初のPyBulletスクリプトが完成しました。Bullet は Bullet/data フォルダにいくつかの URDF ファイルを同梱しています。

## resetBasePositionAndOrientation

各オブジェクトの基底（ルート）の位置と向きをリセットすることができます。このコマンドはすべての物理シミュレーションの効果を上書きしてしまうので、シミュレーションの開始時にのみ行うのがベストです。線速度と角速度はゼロに設定されます。resetBaseVelocityを使用して、ゼロではない線速度および/または角速度にリセットすることができます。

resetBasePositionAndOrientationへの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDFから返されるオブジェクトの一意のid。 |
| required | posObj | vec3 | オブジェクトの基底をワールドスペース座標[X,Y,Z]の指定された位置にリセットします。 |
| required | ornObj | vec4 | 指定された方向のオブジェクトの基底をワールドスペースクォータニオン[X,Y,Z,W]としてリセットします。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

戻り値の引数はありません。

## Transforms: Position and Orientation

オブジェクトの位置は、デカルト世界空間座標[x,y,z]で表現することができます。オブジェクトの向き（または回転）は、クォータニオン [x,y,z,w]、オイラー角 [yaw, pitch, roll]、または 3x3 行列を使って表現することができます。PyBulletには、クォータニオン、オイラー角、3x3行列の間で変換するためのヘルパー関数がいくつか用意されています。さらに、乗算や反転変換を行う関数もあります。

getQuaternionFromEuler と getEulerFromQuaternion

PyBullet API は、方向を表現するためにクォータニオンを使用します。クォータニオンは人にとってあまり直感的ではないので、クォータニオンとオイラー角の間で変換するための2つのAPIがあります。

getQuaternionFromEulerの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | eulerAngle | vec3: list of 3 floats | X,Y,Zのオイラー角はラジアン単位で、X軸周りのロール、Y軸周りのピッチ、Z軸周りのヨーを表す3つの回転を蓄積しています。 |
| optional | physicsClientId | int | 未使用ですが、APIの一貫性のために追加されました。 |

getQuaternionFromEulerは、4つの浮動小数点値[X,Y,Z,W]の四角形、vec4リストを返します。

getEulerFromQuaternion

getEulerFromQuaternionの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | quaternion | vec4: list of 4 floats | クォータニオンの形式は [x,y,z,w] です。 |
| optional | physicsClientId | int | 未使用ですが、APIの一貫性のために追加されました。 |

getEulerFromQuaternionは、3つの浮動小数点値のリスト、vec3を返します。回転順序は、ROSのURDF rpyの規約にあるように、最初にXの周りにロールし、次にYの周りにピッチし、最後にZの周りにヨーします。

getMatrixFromQuaternion

getMatrixFromQuaternionは、クォータニオンから3x3の行列を作成するためのユーティリティAPIです。入力はクォータニオンで、出力は行列を表す9つのフロートのリストです。

getAxisAngleFromQuaternion

getAxisAngleFromQuaternionは、指定されたクォータニオンの向きの軸と角度の表現を返します。

|  |  |  |  |
| --- | --- | --- | --- |
| required | quaternion | list of 4 floats | orientation |
| optional | physicsClientId | int | unused, added for API consistency. |

multiplyTransforms, invertTransform

PyBulletには、乗算変換や逆変換を行うためのヘルパー関数がいくつか用意されています。これは、ある座標系から別の座標系に座標を変換するのに役立ちます。

multiplyTransformsの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | positionA | vec3, list of 3 floats |  |
| required | orientationA | vec4, list of 4 floats | quaternion [x,y,z,w] |
| required | positionB | vec3, list of 3 floats |  |
| required | orientationB | vec4, list of 4 floats | quaternion [x,y,z,w] |
| optional | physicsClientId | int | unused, added for API consistency. |

戻り値は、位置(vec3)と向き(vec4、四角形x,y,x,w)のリストです。

invertTransform の入力パラメータと出力パラメータは次のとおりです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | position | vec3, list of 3 floats |  |
| required | orientation | vec4, list of 4 floats | quaternion [x,y,z,w] |

invertTransform の出力は，位置（vec3）と向き（vec4，四元系 x,y,x,w）です．

getDifferenceQuaternion

getDifferenceQuaternionは、開始方向から終了方向への補間を行うクォータニオンを返します。

|  |  |  |  |
| --- | --- | --- | --- |
| required | quaternionStart | list of 4 floats | start orientation |
| required | quaternionEnd | list of 4 floats | end orientation |
| optional | physicsClientId | int | unused, added for API consistency. |

アピバージョン取得

APIのバージョンを年月0日単位で問い合わせることができます。同じAPIバージョンの物理クライアント/サーバ間では、同じビット数(32ビット/64ビット)の物理クライアント/サーバ間でのみ接続できます。オプションで未使用の引数 physicsClientId がありますが、これは API の整合性のために追加されたものです。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | physicsClientId | int | unused, added for API consistency. |

# ロボットの制御

はじめに」では、PyBulletを初期化してオブジェクトをロードする方法を紹介しました。loadURDFコマンドのファイル名を"r2d2.urdf"に置き換えると、ROSチュートリアルのR2D2ロボットをシミュレートすることができます。このR2D2ロボットを制御して、移動したり、周りを見回したり、グリッパーを制御してみましょう。そのためには、ジョイントモーターにアクセスする方法を知っておく必要があります。

## Base, Joints, Links

質量中心慣性フレーム

ベース

質量中心慣性フレーム

リンク0

ジョイント0  
枠

質量中心慣性フレーム

リンクワン

ジョイントワン

枠

URDFファイルに記述されている模擬ロボットは、ベースと、オプションでジョイントで接続されたリンクを持っています。各ジョイントは、親リンクと子リンクを接続します。階層のルートには、ベースと呼ぶ単一のルート親があります。ベースは完全固定、自由度0、または自由度6の完全自由のいずれかです。各リンクは単一のジョイントで親に接続されているので、ジョイントの数はリンクの数と同じです。通常のリンクは、[0...getNumJoints()]の範囲のリンクインデックスを持ちます。 ベースは通常の「リンク」ではないので、そのリンクインデックスとして-1の慣例を使用します。ジョイントフレームは、親の質量中心慣性フレームに対して相対的に表現されるという慣例を使用しています。

## getNumJoints, getJointInfo

ロボットをロードした後、getNumJoints APIを使用して関節の数を問い合わせることができます。r2d2.urdfでは15を返します。

getNumJoints 入力パラメータ。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などによって返されるボディ固有の ID。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getNumJoints は、ジョイントの数を表す整数値を返します。

getJointInfo

各ジョイントに対して、名前やタイプなどの情報を問い合わせることができます。

getJointInfo 入力パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などによって返されるボディ固有の ID。 |
| required | jointIndex | int | 範囲内のインデックス [0 ... getNumJoints(bodyUniqueId)] を指定します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getJointInfoは、情報のリストを返します。

|  |  |  |
| --- | --- | --- |
| jointIndex | int | 入力パラメータと同じジョイントインデックス |
| jointName | string | URDF(またはSDFなど)ファイルで指定されたジョイントの名前 |
| jointType | int | これは、位置と速度の変数の数を意味しています。  JOINT\_REVOLUTE、JOINT\_PRISMATIC、JOINT\_SPHERICAL、JOINT\_PLANAR、JOINT\_FIXED。詳細は、ベース、ジョイント、リンクの項を参照してください。 |
| qIndex | int | この体の位置状態変数の最初の位置インデックス |
| uIndex | int | この体の速度状態変数の最初の速度指数 |
| flags | int | 控え目 |
| jointDamping | float | URDF ファイルで指定されたジョイントダンピング値 |
| jointFriction | float | URDF ファイルで指定されたジョイント摩擦値 |
| jointLowerLimit | float | スライダとリボリュート（ヒンジ）の関節の位置的な下限値。 |
| jointUpperLimit | float | スライダー関節、リボリュート関節の位置上限値です。上限値＜下限値の場合は無視されます。 |
| jointMaxForce | float | URDF で指定された最大の力（他のファイル形式の場合もあります） この値は自動的には使用されないことに注意してください。setJointMotorControl2' で maxForce を使用することができます。 |
| jointMaxVelocity | float | URDF で指定された最大速度。なお、現時点では実際のモータ制御コマンドでは最大速度は使用されていません。 |
| linkName | string | URDF (またはSDFなど) ファイルで指定されたリンクの名前。 |
| jointAxis | vec3 | ローカルフレーム内のジョイント軸 (JOINT\_FIXEDでは無視されます) |
| parentFramePos | vec3 | 親枠内関節位置 |
| parentFrameOrn | vec4 | 親フレーム内の関節方向 |
| parentIndex | int | 親リンクインデックス、ベースの場合は -1 |

## setJointMotorControl2/Array

注意: *setJointMotorControlは廃止され、setJointMotorControl2 APIに置き換えられました。(あるいは setJointMotorControlArray を使用するのが良いでしょう)。*

1つまたは複数の関節モータに対して所望の制御モードを設定することでロボットを制御することができます。stepSimulationの間、物理エンジンは、最大モータ力およびその他の制約の範囲内で到達可能な所定の目標値に到達するようにモータをシミュレートします。

重要：デフォルトでは、各レボリュートジョイントとプリズムジョイントは、速度モータを使用してモータ駆動されています。最大力を0にすることで、これらのデフォルトのモータを無効にすることができます。 これにより、トルク制御を行うことができます。

例えば

maxForce = 0

mode = p.VELOCITY\_CONTROL

p.setJointMotorControl2(objUid, jointIndex.

controlMode=mode, force=maxForce)

また、ジョイント摩擦を模倣するために、小さなゼロではない力を使用することもできます。

ホイールを一定の速度を維持したい場合は、最大の力で使用することができます。

maxForce = 500

p.setJointMotorControl2(bodyUniqueId=objUid,

jointIndex=0,

controlMode=p.VELOCITY\_CONTROL,

targetVelocity = targetVel,

force = maxForce)

setJointMotorControl2への入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDFなどから返されるボディ固有のID |
| required | jointIndex | int | 範囲内のリンクインデックス [0...getNumJoints(bodyUniqueId) (リンクインデックス==ジョイントインデックスであることに注意) |
| required | controlMode | int | POSITION\_CONTROL（実際にはCONTROL\_MODE\_POSITION\_VELOCITY\_PD）、VELOCITY\_CONTROL、TORQUE\_CONTROL、PD\_CONTROLがあります。安定した(暗黙の)PD制御のための実験的なSTABLE\_PD\_CONTROLもありますが、これは追加の準備が必要です。STABLE\_PD\_CONTROLの例はhumanoidMotionCapture.pyとbullet\_envs.deep\_mimcを参照してください)。TORQUE\_CONTROLは瞬時にトルクをかけますので、stepSimulationを明示的に呼び出した場合のみ有効です。 |
| optional | targetPosition | float | POSITION\_CONTROLでは、targetValueは関節の目標位置です。 |
| optional | targetVelocity | float | VELOCITY\_CONTROLとPOSITION\_CONTROLでは、targetVelocityはジョイントの希望する速度です。targetVelocityはジョイントの最大速度ではないことに注意してください。PD\_CONTROL と POSITION\_CONTROL/CONTROL\_MODE\_POSITION\_VELOCITY\_PD では、最終的なターゲット速度は以下を使用して計算されます。  kp\*(arp\*(erp\*(westedPosition-currentPosition)/dt)+currentVelocity+kd\*(m\_desiredVelocity - currentVelocity)。examples/pybullet/examples/pdControl.pyも参照してください。 |
| optional | force | float | POSITION\_CONTROLおよびVELOCITY\_CONTROLでは、目標値に到達するために使用される最大のモータ力です。TORQUE\_CONTROLでは、シミュレーションの各ステップで適用される力/トルクです。 |
| optional | positionGain | float | 以下の実装ノートを参照してください。 |
| optional | velocityGain | float | 以下の実装ノートを参照してください。 |
| optional | maxVelocity | float | POSITION\_CONTROLでは、これは速度を最大 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

注：ジョイントモータコントローラの実際の実装は、POSITION\_CONTROLとVELOCITY\_CONTROLの制約として、TORQUE\_CONTROLの外力として行われます。

|  |  |  |  |
| --- | --- | --- | --- |
| method | implementation | component | constraint error to be minimized |
| POSITION\_CONTROL | constraint | velocity and position constraint | error = position\_gain\*(desired\_position-actual\_position)+velocity\_gain\*(desired\_velocity-actual\_velocity) |
| VELOCITY\_CONTROL | constraint | pure velocity constraint | error = desired\_velocity - actual\_velocity |
| TORQUE\_CONTROL | external force |  |  |

一般的には、VELOCITY\_CONTROL または POSITION\_CONTROL から始めるのがベストです。正しい力のシミュレーションは、非常に正確なURDF/SDFファイルのパラメータとシステムの識別（正しい質量、慣性、質量の中心の位置、関節摩擦など）に依存しているので、TORQUE\_CONTROL（力の制御）を行うのははるかに困難です。

setJointMotorControlArray

各ジョイントに対して個別の呼び出しを行う代わりに、すべての入力に対して配列を渡すことで、呼び出しのオーバーヘッドを大幅に削減することができます。

setJointMotorControlArray は setJointMotorControl2 と同じパラメータを取ります。

setJointMotorControlArrayへの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDFなどから返されるボディ固有のID |
| required | jointIndices | list of int | index in range [0...getNumJoints(bodyUniqueId) (link index == joint indexに注意) |
| required | controlMode | int | POSITION\_CONTROL, VELOCITY\_CONTROL, TORQUE\_CONTROL, PD\_CONTROL.(安定した(暗黙の)PD制御のための実験的なSTABLE\_PD\_CONTROLもありますが、これは追加の準備が必要です。STABLE\_PD\_CONTROLの例はhumanoidMotionCapture.pyとbullet\_envs.deep\_mimcを参照してください) |
| optional | targetPositions | list of float | POSITION\_CONTROLでは、targetValueは関節の目標位置です。 |
| optional | targetVelocities | list of float | PD\_CONTROL, VELOCITY\_CONTROL, POSITION\_CONTROLでは、targetValueは関節の目標速度です。 |
| optional | forces | list of float | PD\_CONTROL、POSITION\_CONTROL、VELOCITY\_CONTROLでは、目標値に到達するために使用される最大のモータ力です。TORQUE\_CONTROLでは、シミュレーションの各ステップで適用される力/トルクです。 |
| optional | positionGains | list of float | 以下の実装ノートを参照してください。 |
| optional | velocityGains | list of float | 以下の実装ノートを参照してください。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

setJointMotorControlArrayの使用例はbullet/examples/pybullet/tensorflow/humanoid\_running.pyを参照してください。

setJointMotorControlMultiDof

setJointMotorControlMultiDofはsetJointMotorControl2と似ていますが、球状のジョイントをサポートしています。1つのfloatの代わりに、targetPosition、targetVelocity、force引数には、1つのfloatまたは3つのfloatのリストを指定することができます。

setJointMotorControlMultiDofへの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDFなどから返されるボディ固有のID |
| required | jointIndex | int | 範囲内のリンクインデックス [0...getNumJoints(bodyUniqueId) (リンクインデックス==ジョイントインデックスであることに注意) |
| required | controlMode | int | POSITION\_CONTROL（実際にはCONTROL\_MODE\_POSITION\_VELOCITY\_PD）、VELOCITY\_CONTROL、TORQUE\_CONTROL、PD\_CONTROLがあります。安定した(暗黙の)PD制御のための実験的なSTABLE\_PD\_CONTROLもありますが、これは追加の準備が必要です。STABLE\_PD\_CONTROLの例はhumanoidMotionCapture.pyとbullet\_envs.deep\_mimcを参照してください)。 |
| optional | targetPosition | list of 1 or 3 floats | POSITION\_CONTROLでは、targetValueは関節の目標位置です。 |
| optional | targetVelocity | list of 1 or 3 floats | VELOCITY\_CONTROLとPOSITION\_CONTROLでは、targetVelocityはジョイントの希望する速度です。targetVelocityはジョイントの最大速度ではないことに注意してください。PD\_CONTROL と POSITION\_CONTROL/CONTROL\_MODE\_POSITION\_VELOCITY\_PD では、最終的なターゲット速度は以下を使用して計算されます。  kp\*(arp\*(erp\*(westedPosition-currentPosition)/dt)+currentVelocity+kd\*(m\_desiredVelocity - currentVelocity)。examples/pybullet/examples/pdControl.pyも参照してください。 |
| optional | force | list of 1 or 3 floats | POSITION\_CONTROLおよびVELOCITY\_CONTROLでは、目標値に到達するために使用される最大のモータ力です。TORQUE\_CONTROLでは、シミュレーションの各ステップで適用される力/トルクです。 |
| optional | positionGain | float | 以下の実装ノートを参照してください。 |
| optional | velocityGain | float | 以下の実装ノートを参照してください。 |
| optional | maxVelocity | float | POSITION\_CONTROLでは、これは速度を最大 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

setJointMotorControlMultiDofArray

setJointMotorControlMultiDofArrayはsetJointMotorControlMultiDofArrayのより効率的なバージョンで、複数のコントロールターゲットを渡すことで、PythonとPyBullet C++エクステンション間の呼び出しオーバーヘッドを回避/削減します。例としてhumanoidMotionCapture.pyを参照してください。

setJointMotorControlMultiDofArrayの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDFなどから返されるボディ固有のID |
| required | jointIndices | list of int | 範囲内のリンクインデックス [0...getNumJoints(bodyUniqueId) (リンクインデックス==ジョイントインデックスであることに注意) |
| required | controlMode | int | POSITION\_CONTROL（実際にはCONTROL\_MODE\_POSITION\_VELOCITY\_PD）、VELOCITY\_CONTROL、TORQUE\_CONTROL、PD\_CONTROLがあります。安定した(暗黙の)PD制御のための実験的なSTABLE\_PD\_CONTROLもありますが、これは追加の準備が必要です。STABLE\_PD\_CONTROLの例はhumanoidMotionCapture.pyとbullet\_envs.deep\_mimcを参照してください)。 |
| optional | targetPositions | list of float or vec3 | POSITION\_CONTROLでは、targetValueは関節の目標位置です。 |
| optional | targetVelocities | list of float | VELOCITY\_CONTROLとPOSITION\_CONTROLでは、targetVelocityはジョイントの希望する速度です。targetVelocityはジョイントの最大速度ではないことに注意してください。PD\_CONTROL と POSITION\_CONTROL/CONTROL\_MODE\_POSITION\_VELOCITY\_PD では、最終的なターゲット速度は以下を使用して計算されます。  kp\*(arp\*(erp\*(westedPosition-currentPosition)/dt)+currentVelocity+kd\*(m\_desiredVelocity - currentVelocity)。examples/pybullet/examples/pdControl.pyも参照してください。 |
| optional | forces | list of float | POSITION\_CONTROLおよびVELOCITY\_CONTROLでは、目標値に到達するために使用される最大のモータ力です。TORQUE\_CONTROLでは、シミュレーションの各ステップで適用される力/トルクです。 |
| optional | positionGains | list of float | setJointMotorControl2 の実装ノートを参照してください。 |
| optional | velocityGains | list of float | setJointMotorControl2 の実装ノートを参照してください。 |
| optional | maxVelocities | list of float | POSITION\_CONTROLでは、これは速度を最大 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

## getJointState(s)、resetJointState

getJointStateを使用して、関節の位置、速度、関節反力、関節モータトルクなどの関節の状態変数を問い合わせることができます。

getJointState入力パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディユニークID |
| required | jointIndex | int | 範囲 [0..getNumJoints(bodyUniqueId)] のリンクインデックス |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getJointState出力

|  |  |  |
| --- | --- | --- |
| jointPosition | float | このジョイントの位置値。 |
| jointVelocity | float | この関節の速度値。 |
| jointReactionForces | list of 6 floats | これらは関節反力で、トルクセンサーが有効な場合は[Fx, Fy, Fz, Mx, My, Mz]となります。トルクセンサがない場合は[0,0,0,0,0,0,0,0]となります。 |
| appliedJointMotorTorque | float | これは、最後のステップSimulationで適用されたモータトルクです。これはVELOCITY\_CONTROLとPOSITION\_CONTROLでのみ適用されることに注意してください。TORQUE\_CONTROLを使用している場合、適用されたジョイント・モータ・トルクは提供されたものと同じなので、別途レポートする必要はありません。 |

getJointStates

getJointStates は getJointState の配列版です。単一のジョイントインデックスを渡すのではなく、ジョイントインデックスのリストを渡します。

getJointStateMultiDof  
関節のためのgetJointStateMultiDofもあります。

ゲットジョイントステート出力

|  |  |  |
| --- | --- | --- |
| jointPosition | list of 1 or 4 float | このジョイントの位置値（ジョニット角/位置またはジョイント方位四分儀として |
| jointVelocity | list of 1 or 3 float | この関節の速度値。 |
| jointReactionForces | list of 6 floats | これらは関節反力で、トルクセンサーが有効な場合は[Fx, Fy, Fz, Mx, My, Mz]となります。トルクセンサがない場合は[0,0,0,0,0,0,0,0]となります。 |
| appliedJointMotorTorque | float | これは、最後のステップSimulationで適用されたモータトルクです。これはVELOCITY\_CONTROLとPOSITION\_CONTROLでのみ適用されることに注意してください。TORQUE\_CONTROLを使用している場合、適用されたジョイント・モータ・トルクは提供されたものと同じなので、別途レポートする必要はありません。 |

getJointStatesMultiDof

getJointStatesMultiDofを使用すると、複数のジョイント状態（複数の球状のジョイントを含む）を問い合わせることができます。

resetJointState

ジョイントの状態をリセットすることができます。これは、シミュレーションを実行していない状態で、最初に行うのがベストです。なお、現時点では1自由度の電動ジョイント、スライディングジョイントまたはリボリュートジョイントのみをサポートしています。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディユニークID |
| required | jointIndex | int | 範囲内のジョイントインデックス [0..getNumJoints(bodyUniqueId)] |
| required | targetValue | float | 関節位置 |
| optional | targetVelocity | float | きょうどうそくど |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

resetJointState(s)MultiDof

球状の関節にはresetJointStateMultiDofもあります。resetJointStateMultiDofの例は[humanoidMotionCaptureを](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/humanoidMotionCapture.py)参照してください。また、複数の関節を一度にリセットするためのresetJointStatesMultiDofもあります。

## enableJointForceTorqueSensor

各ジョイントのジョイント力/トルクセンサーを有効にしたり、無効にしたりすることができます。有効にすると、stepSimulationを実行すると、'getJointState'は固定自由度でのジョイント反力をレポートします：固定ジョイントは6自由度のジョイント力/トルクをすべて測定します。固定ジョイントでは、6自由度のジョイント力/トルクをすべて測定します。ジョイントモータによる印加力は、getJointState の appliedJointMotorTorque で取得できます。

enableJointForceTorqueSensorへの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディユニークID |
| required | jointIndex | int | 範囲内のジョイントインデックス [0..getNumJoints(bodyUniqueId)] |
| optional | enableSensor | int | 1/Trueで有効、0/Falseでフォース/トルクセンサを無効にします。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

## getLinkState(s)

また、getLinkStateを使って、各リンクの質量中心の直交世界の位置と向きを問い合わせることもできます。また、重心のローカル慣性フレームをURDFリンクフレームに報告し、グラフィック/可視化フレームの計算を容易にします。

getLinkState入力パラメータ

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディユニークID |
| required | linkIndex | int | リンクインデックス |
| optional | computeLinkVelocity | int | 1に設定すると、デカルト世界速度が計算されて返されます。 |
| optional | computeForwardKinematics | int | 1 (またはTrue) に設定すると、直交世界の位置/向きが前方運動学を用いて再計算されます。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getLinkStateの戻り値

|  |  |  |
| --- | --- | --- |
| linkWorldPosition | vec3, list of 3 floats | 質量中心の直交位置 |
| linkWorldOrientation | vec4, list of 4 floats | 質量中心の直交方向、四元系[x,y,z,w]における |
| localInertialFramePosition | vec3, list of 3 floats | URDFリンクフレームで表される慣性フレーム（質量中心）の局所位置オフセット |
| localInertialFrameOrientation | vec4, list of 4 floats | URDF リンクフレームで表現された慣性フレームの局所方位 (クォータニオン [x,y,z,w] オフセット) |
| worldLinkFramePosition | vec3, list of 3 floats | URDFリンクフレームの世界位置 |
| worldLinkFrameOrientation | vec4, list of 4 floats | URDF リンクフレームの世界方向 |
| worldLinkLinearVelocity | vec3, list of 3 floats | 直交世界の速度.computeLinkVelocityが0でない場合にのみ返されます。 |
| worldLinkAngularVelocity | vec3, list of 3 floats | 直交世界の速度.computeLinkVelocityが0でない場合にのみ返されます。 |

URDFリンクフレームと質量中心フレーム（両方とも世界空間）の関係は次のようになります：urdfLinkFrame = comLinkFrame \* localInertialFrame.inverse()。リンクフレームと慣性フレームの詳細については、[ROSのURDFチュートリアルを](http://wiki.ros.org/urdf/Tutorials/Adding%20Physical%20and%20Collision%20Properties%20to%20a%20URDF%20Model)参照してください。

getLinkStates

getLinkStatesは複数のリンクの情報を返します。linkIndexの代わりに、linkIndicesをintのリストとして受け取ります。これにより、getLinkStateへの複数回の呼び出しのオーバーヘッドを減らすことができ、パフォーマンスを向上させることができます。

URDFリンクフレーム

質量中心フレーム（COM,

world space)

ローカル慣性フレーム

スクリプトの例 (古い可能性があります。実際のBullet/examples/pybullet/examplesフォルダを確認してください。)

|  |  |
| --- | --- |
| examples/pybullet/tensorflow/humanoid\_running.py | ヒューマノイドに負荷をかけ、訓練されたニューラルネットワークを使用して、OpenAIで訓練されたTensorFlowを使用して走行を制御します。 |
| examples/pybullet/gym/pybullet\_envs/bullet/minitaur.pyとminitaur\_gym\_env.py | OpenAI GYMとTensorFlowのためのMinitaur環境  python -m pybullet\_envs.examples.minitaur\_gym\_env\_exampleを使ってもミニツアーの動作を見ることができます。 |
| examples/pybullet/examples/quadruped.py | また、p.startStateLoggingを使用して状態をファイルに記録します。[動画を](https://www.youtube.com/watch?v=lv7lybtOzeo)参照してください。 |
| examples/quadruped\_playback.py | 四足歩行（ミニタウロス）を作成し、ログファイルを読み込み、運動制御対象として位置を設定します。 |
| examples/pybullet/examples/testrender.py | URDFファイルを読み込み，画像をレンダリングし，ピクセル（RGB，深度，セグメンテーションマスク）を取得し，MatPlotLibを用いて画像を表示します． |
| examples/pybullet/examples/testrender\_np.py | testrender.py に似ていますが、NumPy 配列を使ってピクセル転送を高速化しています。また、簡単なベンチマーク/タイミングも含まれています。 |
| examples/pybullet/examples/saveWorld.py | オブジェクトの状態（位置、向き）をpybullet Pythonスクリプトに保存します。主にVRでシーンを設定したり、初期状態を保存したりするのに便利です。すべての状態がシリアル化されているわけではありません。 |
| examples/pybullet/examples/inverse\_kinematics.py | Kuka ARMクロックを作成して、cracateInverseKinematicsコマンドを使用する方法を示します。 |
| examples/pybullet/examples/rollPitchYaw.py | スライダーGUIウィジェットの使い方を紹介 |
| examples/pybullet/examples/constraint.py | リンク間の制約をプログラムで作成します。 |
| examples/pybullet/examples/vrhand.py | VRコントローラーで追跡されたVRグローブを使って、手を操作する。[動画を](https://www.youtube.com/watch?v=0JC-yukK-jo)ご覧ください。 |

## getBaseVelocity, resetBaseVelocity

getBaseVelocity を使用して、体の底面の線速度と角速度にアクセスできます。入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | load\*メソッドから返されるボディ固有のID。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

これは、デカルト世界空間座標での線速度 [x,y,z] と角速度 [wx,wy,wz] を表す 2 つの vector3 値 (リスト内の 3 つのフロート) のリストを返します。

resetBaseVelocity を使用して、ボディの基底部の線速度や角速度をリセットすることができます。入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | objectUniqueId | int | load\*メソッドから返されるボディ固有のID。 |
| optional | linearVelocity | vec3, list of 3 floats | 直交世界座標での線速度 [x,y,z]。 |
| optional | angularVelocity | vec3, list of 3 floats | 角速度 [wx,wy,wz] をデカルト世界座標で指定します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

## applyExternalForce/Torque

体に力やトルクを加えるには、applyExternalForceとapplyExternalTorqueを使用します。このメソッドは、stepSimulationを使用して明示的にシミュレーションをステップさせた場合にのみ機能することに注意してください。各シミュレーションステップの後、外力はゼロにクリアされます。setRealTimeSimulation(1)を使用している場合、applyExternalForce/Torqueは未定義の動作をします（0、1、または複数の力/トルクの適用）。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | objectUniqueId | int | ロードメソッドによって返されるオブジェクト固有のID。 |
| required | linkIndex | int | リンクインデックス、またはベースに -1 を指定します。 |
| required | forceObj | vec3, list of 3 floats | 適用する力/トルクベクトル [x,y,z]。座標系のフラグを参照してください。 |
| required | posObj | vec3, list of 3 floats | 力が適用されるリンク上の位置を指定します。applyExternalForceのみ。座標系のフラグを参照してください。 |
| required | flags | int | 力/位置の座標系を指定します。直交世界座標の場合はWORLD\_FRAME、ローカルリンク座標の場合はLINK\_FRAMEのいずれかを指定します。 |
| optional | physicsClientId | int |  |

## getNumBodies, getBodyInfo, getBodyUniqueId, removeBody

getNumBodiesは物理サーバーにあるボディの総数を返します。

getNumBodies' を使用した場合は、'getBodyUniqueId' を使用してボディ固有の ID を問い合わせることができます。すべての API はすでにボディ固有の ID を返しているので、ボディ固有の ID を追跡しているのであれば、通常は getBodyUniqueId を使用する必要はないことに注意してください。

getBodyInfoは、URDF, SDF, MJCFなどのファイルから抽出したベース名を返します。

syncBodyInfo

syncBodyInfoは、1つの物理サーバーに接続された複数のクライアントがワールドを変更した場合(loadURDFやremoveBodyなど)に、ボディ情報(getBodyInfo)を同期させます。

removeBodyは、(loadURDFやloadSDFなどから)ボディ固有のIDでボディを削除します。

## createConstraint, removeConstraint, changeConstraint

URDF, SDF, MJCFは，多関節体をループのない木構造として指定する。createConstraint'を使うと、ループを閉じるためにボディの特定のリンクを接続することができます。Bullet/examples/pybullet/examples/quadruped.pyを参照してください。さらに、オブジェクト間や、オブジェクトと特定のワールドフレームの間に任意の制約を作成することができます。例については、Bullet/examples/pybullet/examples/constraint.pyを参照してください。

また、VRコントローラのように、アニメーションフレームによって駆動される物理オブジェクトの動きを制御するために使用することもできる。このような目的のためには、位置や速度を直接設定するのではなく、制約を使用した方が良いでしょう。

createConstraintは以下の入力パラメータを持っています。

|  |  |  |  |
| --- | --- | --- | --- |
| required | parentBodyUniqueId | int | しゅたいユニークアイディー |
| required | parentLinkIndex | int | 親リンクインデックス (ベースの場合は -1) |
| required | childBodyUniqueId | int | 子ボディ固有の ID、またはボディがない場合は -1 (ワールド座標で非動的な子フレームを指定します) |
| required | childLinkIndex | int | 子リンクインデックス、またはベース |
| required | jointType | int | ジョイントタイプ。JOINT\_PRISMATIC、JOINT\_FIXED、JOINT\_POINT2POINT、JOINT\_GEAR |
| required | jointAxis | vec3, list of 3 floats | ジョイント軸、子リンクフレーム内 |
| required | parentFramePosition | vec3, list of 3 floats | 質量フレームの親中心に対するジョイントフレームの位置。 |
| required | childFramePosition | vec3, list of 3 floats | 指定された子の質量中心フレーム（子が指定されていない場合はワールド原点）に対するジョイントフレームの位置 |
| optional | parentFrameOrientation | vec4, list of 4 floats | 親質量中心座標フレームに対するジョイントフレームの向き |
| optional | childFrameOrientation | vec4, list of 4 floats | 子の質量中心座標枠 （子が指定されていない場合はワールド原点枠）に対するジョイントフレームの向き。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

createConstraintは整数の一意なidを返し、これは制約を変更または削除するために使用できます。JOINT\_GEARの例はexamples/pybullet/examples/mimicJointConstraint.py、JOINT\_POINT2POINTの例はexamples/minitaur.py、JOINT\_FIXEDの例はexamples/pybullet/examples/constraint.pyを参照してください。

changeConstraint

changeConstraint は、既存の制約のパラメータを変更することができます。入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | userConstraintUniqueId | int | クリエイトコンストライントで返される一意のＩＤ |
| optional | jointChildPivot | vec3, list of 3 floats | 子ピボットを更新しました。 |
| optional | jointChildFrameOrientation | vec4, list of 4 floats | 子フレームの向きをクォータニオンに更新 |
| optional | maxForce | float | 拘束力 |
| optional | gearRatio | float | 歯車比 |
| optional | gearAuxLink | int | ディファレンシャルドライブなどの場合、第3の（補助的な）リンクが参照ポーズとして使用されることがあります。[racecar\_differential.pyを](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/racecar_differential.py)参照してください。 |
| optional | relativePositionTarget | float | 歯車間の相対位置目標オフセット |
| optional | erp | float | 制約誤差低減パラメータ |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

Bullet/examples/pybullet/examples/constraint.pyも参照してください。

removeConstraintは、一意のIDで与えられた制約を削除します。入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | userConstraintUniqueId | int | createConstraint で返される一意の ID |
| optional | physicsClientId | int | connect' で返される一意の ID |

## getNumConstraints, getConstraintUniqueId

createConstraint' を使用して作成された制約の総数を問い合わせることができます。オプションのパラメータはint physicsClientIdです。

getConstraintUniqueId

getConstraintUniqueIdは、範囲0.getNumConstraintsのシリアル・インデックスを取り、制約の一意のIDを報告します。制約を削除することができるので、制約のユニークIDは連続していない可能性があることに注意してください。入力は整数のシリアルインデックスとオプションで physicsClientId です。

## getConstraintInfo/State

制約情報を照会するには、制約の一意のIDを指定します。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | constraintUniqueId | int | createConstraint で返される一意の ID |
| optional | physicsClientId | int | connect' で返される一意の ID |

出力リストは

|  |  |  |
| --- | --- | --- |
| parentBodyUniqueId | int | createConstraint を参照してください。 |
| parentJointIndex | int | createConstraint を参照してください。 |
| childBodyUniqueId | int | createConstraint を参照してください。 |
| childLinkIndex | int | createConstraint を参照してください。 |
| constraintType | int | createConstraint を参照してください。 |
| jointAxis | vec3, list of 3 floats | createConstraint を参照してください。 |
| jointPivotInParent | vec3, list of 3 floats | createConstraint を参照してください。 |
| jointPivotInChild | vec3, list of 3 floats | createConstraint を参照してください。 |
| jointFrameOrientationParent | vec4, list of 4 floats | createConstraint を参照してください。 |
| jointFrameOrientationChild | vec4, list of 4 floats | createConstraint を参照してください。 |
| maxAppliedForce | float | createConstraint を参照してください。 |
| gearRatio | float | createConstraint を参照してください。 |
| gearAuxLink | int | createConstraint を参照してください。 |
| relativePositionTarget | float | createConstraint を参照してください。 |
| erp | float | createConstraint を参照してください。 |

getConstraintState

拘束力の一意なIDを与えると、直近のシミュレーションステップで適用された拘束力を問い合わせることができます。入力は拘束力のユニークID、出力は拘束力のベクトルで、その次元は拘束力によって影響を受ける自由度です（例えば、固定拘束力は6自由度に影響を与えます）。

## getDynamicsInfo/changeDynamics

ベースやリンクの質量、重心、摩擦などの情報を得ることができます。

getDynamicsInfoへの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるオブジェクト固有の ID。 |
| required | linkIndex | int | リンク (ジョイント) インデックス、またはベースに -1 を指定します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

返却情報には限りがありますので、必要に応じてより多くの情報を公開します。

|  |  |  |
| --- | --- | --- |
| mass | double | ｋｇの質量 |
| lateral\_friction | double | 摩擦係数 |
| local inertia diagonal | vec3, list of 3 floats | 局所的な慣性対角線。リンクとベースは質量中心を中心にして、慣性の主軸を揃えていることに注意してください。 |
| local inertial pos | vec3 | 慣性フレーム位置 |
| local inertial orn | vec4 | 慣性フレームの関節フレームの局所座標での向き |
| restitution | double | 返還係数 |
| rolling friction | double | 接触法線に直交する転がり摩擦係数 |
| spinning friction | double | 接触法線周りの回転摩擦係数 |
| contact damping | double | 利用できない場合は -1。 |
| contact stiffness | double | 利用できない場合は -1。 |
| body type | int | 1＝剛体、2＝マルチボディ、3＝ソフトボディ |
| collision margin | double | advanced/internal/unsupported info. collision margin of the collision shape. collision margins depend on the shape type, it is not consistent. |

changeDynamics

質量、摩擦、復元係数などのプロパティは、changeDynamicsを使用して変更することができます。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるオブジェクト固有の ID。 |
| required | linkIndex | int | リンクインデックス、またはベースの場合は -1 |
| optional | mass | double | リンクの質量を変更する (または linkIndex -1 の場合はベース) |
| optional | lateralFriction | double | 横接触摩擦 |
| optional | spinningFriction | double | 接触法線周りのねじり摩擦 |
| optional | rollingFriction | double | 接触法線に直交するねじり摩擦（この値をゼロに近づけないとシミュレーションが非常に非現実的になる可能性があります。 |
| optional | restitution | double | 接触の弾力性。1よりも少し小さく、好ましくは0に近い値に保つ。 |
| optional | linearDamping | double | リンクの線形減衰 (デフォルトでは 0.04) |
| optional | angularDamping | double | リンクの角度減衰 (デフォルトでは 0.04) |
| optional | contactStiffness | double | contactDampingと一緒に使用される接触制約の剛性。 |
| optional | contactDamping | double | このボディ/リンクの接触制約のダンピング.contactStiffnessと一緒に使用します。接触セクションのURDFファイルで指定されている場合は、この値を上書きします。 |
| optional | frictionAnchor | int | 摩擦アンカーの有効化・無効化：摩擦ドリフト補正（URDFコンタクトセクションで設定されていない限りデフォルトでは無効 |
| optional | localInertiaDiagnoal | vec3 | 慣性テンソルの対角線要素。ベースとリンクは質量中心を中心にして、慣性の主軸に沿って配置されているので、慣性テンソルには対角線から外れた要素がないことに注意してください。 |
| optional | ccdSweptSphereRadiu | float | 連続的な衝突検出を行うための球体の半径を指定します。例はBullet/examples/pybullet/examples/experimentalCcdSphereRadius.pyを参照してください。 |
| optional | contactProcessingThreshold | float | このしきい値以下の距離を持つコンタクトは、制約ソルバーによって処理されます。例えば、contactProcessingThreshold = 0の場合、距離0.01のコンタクトは制約として処理されません。 |
| optional | activationState | int | スリープが有効になっている場合、それに影響を与える他のすべてのオブジェクトもスリープの準備ができている場合、（閾値以下で）動かないオブジェクトはスリープとして無効になります。  pybullet.ACTIVATION\_STATE\_SLEEP  pybullet.ACTIVATION\_STATE\_ENABLE\_SLEEPING  pybullet.ACTIVATION\_STATE\_DISABLE\_WAKEUP  また、loadURDFでflags = pybullet.URDF\_ENABLE\_SLEEPINGを使用してスリープを有効にすることもできます。  sleeping.pyの例を参照してください。 |
| optional | jointDamping | double | 各ジョイントに適用されるジョイント減衰係数です。この係数は、URDFの関節減衰フィールドから読み込まれます。0に近い値にしてください。  ジョイント減衰力 = -damping\_coefficient \* joint\_velocity. |
| optional | anisotropicFriction | double | 異方性摩擦係数（anisotropicFriction coefficient）を使用して、異なる方向の摩擦のスケーリングを可能にします。 |
| optional | maxJointVelocity | double | 与えられたジョイントの最大ジョイント速度を、制約解法中に超過した場合はクランプされます。デフォルトの最大関節速度は100単位です。 |
| optional | collisionMargin | double | 形状の種類に依存しますが、コリジョン形状にパディングを追加したり、追加しなかったりします。 |
| optional | jointLowerLimit | double | ジョイントの下限値を変更するには、 jointUpperLimitも必要ですが、そうでなければ無視されます。現時点では、'getJointInfo'ではジョイントの下限値は更新されないことに注意してください! |
| optional | jointUpperLimit | double | ジョイントの上限を変更するには、 jointLowerLimitも必要です。現時点では、'getJointInfo'ではジョイントの上限は更新されないことに注意してください! |
| optional | jointLimitForce | double | 関節制限を満たすために適用される最大力を変更します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

## setTimeStep

警告: 多くの場合、timeStepはデフォルトの240Hzのままにしておいた方が良いでしょう。いくつかのパラメータはこの値を考慮して調整されています。例えば、ソルバーの反復回数や、接触、摩擦、非接触ジョイントのエラー低減パラメータ(ERP)は、タイムステップに関連しています。タイムス テップを変更した場合、これらの値、特に erp の値を調整し直す必要があるかもしれません。

stepSimulation'を呼び出す際に使用する物理エンジンのタイムステップを設定することができます。このメソッドはシミュレーションの開始時にのみ呼び出すのがベストです。このタイムステップは定期的に変更しないようにしてください。 setTimeStep は新しい setPhysicsEngineParameter API を使っても設定できます。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | timeStep | float | stepSimulation'を呼び出すたびに、'timeStep'で進みます。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

## setPhysicsEngineParameter

物理エンジンのパラメータを設定するには、setPhysicsEngineParameter API を使用します。以下の入力パラメータが公開されています。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | fixedTimeStep | float | 物理エンジンのタイムステップは秒単位で行われ、'stepSimulation'を呼び出すたびに、シミュレートされた時間はこの分だけ進みます。setTimeStep' と同じです。 |
| optional | numSolverIterations | int | 制約ソルバーの反復回数の最大値を選択します。solverResidualThresholdに達した場合、numSolverIterationsの前にソルバーを終了させることができます。 |
| optional | useSplitImpulse | int | 最大座標使用時のみの先進機能：位置制約解法と速度制約解法を2段階に分割し、巨大な貫通回復力を防ぐ。 |
| optional | splitImpulsePenetrationThreshold | float | useSplitImpulse」に関連しています。特定の接触制約の貫通度がこの指定されたしきい値よりも小さい場合、その接触に対してスプリットインパルスは発生しません。 |
| optional | numSubSteps | int | 物理シミュレーションのステップを 'numSubSteps' でさらに細分化します。これは精度よりも性能を優先します。 |
| optional | collisionFilterMode | int | デフォルトのコリジョンフィルタには0を使用します。(グループA&マスクB) AND (グループB&マスクA)。ORコリジョンフィルタに切り替えるには1を使用します。(グループA&maskB) OR (グループB&maskA) |
| optional | contactBreakingThreshold | float | このしきい値を超える距離の接触点は、LCPソルバーでは処理されません。さらに、AABBはこの数だけ拡張されます。Bullet 2.xではデフォルトは0.02です。 |
| optional | maxNumCmdPer1ms | int | 実験: 実行されたコマンドの数がこのしきい値を超えた場合、1msのスリープを追加します。 |
| optional | enableFileCaching | int | 0に設定すると、.obj波面ファイルの読み込みなどのファイルキャッシングを無効にします。 |
| optional | restitutionVelocityThreshold | float | 相対速度がこのしきい値以下であれば、復帰はゼロになります。 |
| optional | erp | float | 制約誤差低減パラメータ（非接触・非摩擦 |
| optional | contactERP | float | 接触誤差低減パラメータ |
| optional | frictionERP | float | 摩擦誤差低減パラメータ（位置摩擦アンカーが有効な場合 |
| optional | s | int | 0に設定すると、暗黙のコーン摩擦を無効にし、ピラミッド近似を使用します（コーンがデフォルトです）。 |
| optional | deterministicOverlappingPairs | int | 1 に設定すると有効、0 に設定すると重複ペアのソートを無効にします（下位互換設定）。 |
| optional | allowedCcdPenetration | float | contrinuous collision detection (CCD)が有効になっている場合、このしきい値以下であればCCDは使用されません。 |
| optional | jointFeedbackMode | int | Speficyジョイントフィードバックフレーム。  JOINT\_FEEDBACK\_IN\_WORLD\_SPACE  JOINT\_FEEDBACK\_IN\_JOINT\_FRAME |
| optional | solverResidualThreshold | double | 各制約の速度レベル誤差の最大値がこの閾値を下回る場合、ソルバーは終了します (ソルバーが numSolverIterations に到達しない限り)。デフォルト値は1e-7です。 |
| optional | contactSlop | float | この閾値以下ではコンタクトの位置補正が解消されず、より安定したコンタクトが可能になります。 |
| optional | enableSAT | int | true/1 の場合、分離軸定理に基づく凸衝突検出を有効にします。loadURDF で URDF\_INITIALIZE\_SAT\_FEATURES が必要です。satCollision.py の例を参照してください． |
| optional | constraintSolverType | int | Experimental (無視するのがベスト): Dantzigのような直接LCPソルバーの使用を許可します。switchConstraintSolverType.pyの例を参照してください。 |
| optional | globalCFM | float | 実験的（無視するのがベスト）なグローバルデフォルト制約力混合パラメータ。 |
| optional | minimumSolverIslandSize | int | 実験的に（無視するのがベスト）、独立した制約の非常に小さな島を避けるために、制約解決の島の最小サイズ。 |
| optional | reportSolverAnalytics | int | true/1 の場合、追加の解析が可能です。 |
| optional | warmStartingFactor | float | 初期ソルバーの初期化に使用される前フレームの力/インパルスの割合 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

setDefaultContactERPは、デフォルトのコンタクトパラメータ設定を設定するためのAPIです。setPhysicsEngineParameter APIにロールアウトされます。

getPhysicsEngineParameters

getPhysicsEngineParameters コマンドを使うと、オプションの 'physicsClientId' を使って現在の物理エンジンのパラメータを問い合わせることができます。これはパラメータの名前付きタプルを返します。

## resetSimulation

resetSimulationは、ワールドからすべてのオブジェクトを削除し、ワールドを初期状態にリセットします。

|  |  |  |  |
| --- | --- | --- | --- |
| 任意 | 旗 | インスタント | 実験用の旗だ 無視するのが一番だ  RESET\_USE\_SIMPLE\_BROADPHASE  RESET\_USE\_DEFORMABLE\_WORLD  RESET\_USE\_DISCRETE\_DYNAMICS\_WORLD |
| 任意 | 物理クライアントId | インスタント | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

これはオプションのパラメータとして、物理クライアント ID (複数の物理サーバ接続を作成した場合) を受け取ります。

## startStateLogging/stopStateLogging

状態ロギングでは、シミュレーションの状態をログに記録することができます。例えば、シミュレーションの各ステップ後（stepSimulationを呼び出すたびに、またはsetRealTimeSimulationが有効な場合は、シミュレーションの各ステップ後に自動的に）、1つまたは複数のオブジェクトの状態を記録することができます。これにより、オブジェクトの軌跡を記録することができます。また、ベースの位置や向き、関節の位置（角度）、関節の運動力などのボディの共通の状態をログに記録するオプションもあります。

startStateLogging を使用して生成されたすべてのログファイルは、C++ または Python スクリプトを使用して読み取ることができます。ログファイルを読み込む Python スクリプトについては quadruped\_playback.py と kuka\_with\_cube\_playback.py を参照してください。bullet3/examples/Utils/RobotLoggingUtil.cpp/hを使えば、C++でログファイルを読み込むことができます。

MP4動画の録画には、ロギングオプションのSTATE\_LOGGING\_VIDEO\_MP4を使用することができます。他にもVRコントローラの状態をロギングするなど、様々なタイプのロギングを実装していく予定です。

特殊なケースとして、Minitaurロボットのログを実装しました。PyBulletシミュレーションのログファイルは、実際のMinitaur四足歩行ロボットのログファイルと同じです。例はBullet/examples/pybullet/examples/logMinitaur.pyを参照してください。

重要: さまざまなロガーには、作成時にゼロから始まる独自の内部タイムスタンプが含まれています。つまり、同期を取るためには、すべてのロガーを同時に起動する必要があります。ロガーを起動する際に、シミュレーションがリアルタイムモードで実行されていないことを確認する必要があります: ロガーを作成する前に pybullet.setRealTimeSimulation(0) を使用してください。

|  |  |  |  |
| --- | --- | --- | --- |
| required | loggingType | int | 実装されているロギングには様々な種類があります。  STATE\_LOGGING\_MINITAUR: quadruped/quadruped.urdfとオブジェクトのユニークIDを読み込む必要があります。タイムスタンプ、IMUのロール/ピッチ/ヨー、8本の脚モーターの位置(q0-q7)、8本の脚モーターのトルク(u0-u7)、胴体の前進速度、モード(シミュレーションでは未使用)を記録します。  STATE\_LOGGING\_GENERIC\_ROBOTです。これは、すべてのオブジェクトまたは選択されたオブジェクト（objectUniqueIdsが提供されている場合）のデータのログを記録します。  STATE\_LOGGING\_VIDEO\_MP4: MP4ファイルを開き、FFMPEGパイプを使ってOpenGL 3Dビジュアライザーピクセルをファイルにストリーミングし始めます。これには ffmpeg がインストールされている必要があります。avconv (Ubuntuのデフォルト)を使用することもできますが、ffmpegがavconvを指すようにシンボリックリンクを作成するだけです。Windows では、ffmpeg にはいくつかの問題があり、ティアリングや色のアーチファクトが発生する場合があります。  STATE\_LOGGING\_CONTACT\_POINTS  STATE\_LOGGING\_VR\_CONTROLLERSです。  STATE\_LOGGING\_PROFILE\_TIMINGS  これはGoogle Chromeのabout://tracing LOADで開くことができるJSON形式のタイミングファイルをダンプします。 |
| required | fileName | string | ログファイルのデータを保存するファイル名(絶対パスまたは相対パス)。 |
| optional | objectUniqueIds | list of int | 空のままにしておくと、ロガーはすべてのオブジェクトをログに記録します。そうでなければ、ロガーはobjectUniqueIdsリストのオブジェクトをログに記録するだけです。 |
| optional | maxLogDof | int | ログを取るための最大の関節自由度数（基底dofsを除く）。STATE\_LOGGING\_GENERIC\_ROBOT\_DATAに適用されます。デフォルト値は12です。ロボットがこの数を超えると、全くログに記録されません。 |
| optional | bodyUniqueIdA | int | STATE\_LOGGING\_CONTACT\_POINTS に適用されます。指定されている場合、bodyUniqueIdAを含むコンタクトポイントのみをログに記録します。 |
| optional | bodyUniqueIdB | int | STATE\_LOGGING\_CONTACT\_POINTS に適用されます。指定されている場合、bodyUniqueIdBを含むコンタクトポイントのみをログに記録します。 |
| optional | linkIndexA | int | STATE\_LOGGING\_CONTACT\_POINTS に適用されます。指定した場合、bodyUniqueIdA の linkIndexA を含むコンタクトポイントのみをログに記録します。 |
| optional | linkIndexB | int | STATE\_LOGGING\_CONTACT\_POINTS に適用されます。指定した場合、bodyUniqueIdA の linkIndexB を含むコンタクトポイントのみをログに記録します。 |
| optional | deviceTypeFilter | int | deviceTypeFilterでは、ログに記録するVRデバイスを選択することができます。VR\_DEVICE\_CONTROLLER、VR\_DEVICE\_HMD、VR\_DEVICE\_GENERIC\_TRACKER、またはそれらの組み合わせ。STATE\_LOGGING\_VR\_CONTROLLERS に適用されます。デフォルト値は VR\_DEVICE\_CONTROLLER です。 |
| optional | logFlags | int | (次期 PyBullet 1.3.1)。STATE\_LOG\_JOINT\_TORQUES, ジョイントモーターによるジョイントトルクをログに記録します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

このコマンドは、stopStateLogging で使用できる非負の int loggingUniqueId を返します。

Todo: ロギングタイプごとにログに記録されるデータを文書化します。今のところ、ログ読み込みユーティリティを使って調べたり、[ロギング](https://github.com/bulletphysics/bullet3/blob/master/examples/SharedMemory/PhysicsServerCommandProcessor.cpp#L467)やPythonの[dumpLog.py](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/dumpLog.py)スクリプトの[C++ソースコードを](https://github.com/bulletphysics/bullet3/blob/master/examples/SharedMemory/PhysicsServerCommandProcessor.cpp#L467)チェックしてみてください。

stopStateLogging

loggingUniqueId を使用してロガーを停止することができます。

submitProfileTiming

submitProfileTimingは、Pythonのプロファイルコードに開始と停止のタイミングを挿入することができます。[profileTiming.pyの](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/profileTiming.py)例を参照してください。

PyBulletとBulletは多くの関数をインストルメントしているので、どこに時間が費やされているかを見ることができます。これらのプロファイルのタイミングをファイルにダンプすることができ、Google Chrome で about://tracing ウィンドウの LOAD 機能を使って表示することができます。GUIでは、'p'を押してプロファイルダンプを開始/停止することができます。場合によっては、クライアントコードのタイミングを計測したいこともあるでしょう。PyBulletを使ってプロファイルのタイミングを送信することができます。以下に出力例を示します。

# 変形性と布(FEM, PBD)

多関節マルチボディおよび剛体ダイナミクスの他に、PyBulletは、有限要素法（FEM）質量バネシステムを使用した変形可能なシミュレーションおよび布シミュレーションを実装しており、位置ベースダイナミクス（PBD）も使用しています。この実装を使用した研究論文には、[Sim-to-Real Reinforcement Learning for Deformable Object Manipulation](https://sites.google.com/view/sim-to-real-deformable)や[Assistive Gymなど](https://ras.papercept.net/proceedings/ICRA20/1572.pdf)があります。Assistive [Roboticsのための物理シミュレーションフレームワーク](https://ras.papercept.net/proceedings/ICRA20/1572.pdf)

PyBulletは、multibody/rigidbodyとdeformablesの間の双方向のカップリングを実装しています。双方向のカップリングは、手動で作成された添付ファイルだけでなく、衝突に対しても動作します（下記のcreateSoftBodyAnchorを参照してください）。

デフォルトでは、PyBulletは位置ベースのダイナミクス（PBD）を使用します。以下のようにワールドをリセットすることで、有限要素法（FEM）ベースのシミュレーションを有効にすることができます。

pybullet.resetSimulation(p.RESET\_USE\_DEFORMABLE\_WORLD)

例として [deformable\_torus.py](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/deformable_torus.py) も参照してください。

## loadSoftBody/loadURDF

変形可能なオブジェクトを作成する方法はいくつかありますが、布製でもボリューミーでも構いません。

loadSoftBodyを使用すると、VTKやOBJファイルから変形可能なオブジェクトをロードすることができます。

loadSoftBodyでは、以下の引数を使用できます。いくつかのパラメータはFEMモデルを使用することを前提としており、PBDシミュレーションには影響しないことに注意してください。

|  |  |  |  |
| --- | --- | --- | --- |
| required | fileName | string | デフォルメ可能なファイル名。Wavefront .obj形式またはVTK形式にすることができます。 |
| optional | basePosition | vec3 | 変形体初期位置 |
| optional | baseOrientation | vec4 | 初期方位 |
| optional | scale | double | 変形可能なサイズを変更するためのスケーリング係数 (デフォルト = 1) |
| optional | mass | double | 頂点の総質量 |
| optional | collisionMargin | double | 衝突マージンが変形可能な範囲を広げると、特に薄い（布製の）変形可能な場合には、貫通を回避するのに役立ちます。 |
| optional | useMassSpring | bool | 質量ばねを使って |
| optional | useBendingSprings | bool | 撓みを制御するために撓みバネを作る |
| optional | useNeoHookean | bool | ネオフクアンのシミュレーションを有効にする |
| optional | springElasticStiffness | double | 剛性パラメータ |
| optional | springDampingStiffness | double | ダンピングパラメタ |
| optional | springDampingAllDirections | double | ばね減衰パラメータ |
| optional | springBendingStiffness | double | 曲げ剛性のパラメータ |
| optional | NeoHookeanMu | double | ネオフックモデルのパラメータ |
| optional | NeoHookeanLambda | double | ネオフックモデルのパラメータ |
| optional | NeoHookeanDamping | double | ネオフックモデルのパラメータ |
| optional | frictionCoeff | double | 変形材の接触摩擦 |
| optional | useFaceContact | bool | 頂点だけでなく、面の内部での衝突を可能にします。 |
| optional | useSelfCollision | bool | 変形可能な自己衝突を有効にする |
| optional | repulsionStiffness | double | 貫通を回避するのに役立つパラメータです。 |

loadURDF

また、PyBullet は 'deformable' タグを使って変形可能なオブジェクトを指定するために URDF フォーマットを拡張しています。例えば [deformable\_torus.urdf を](https://github.com/bulletphysics/bullet3/blob/5233b72160df8e8915320b03793d85dd0c6792ba/data/torus_deform.urdf)参照してください。

<robot name="torus">

<deformable name="torus">

<inertial>

<mass value="1" />

<inertia ixx="0.0" ixy="0" ixz="0" iyy="0" iyz="0" izz="0" />

</inertial>

<collision\_margin value="0.006"/>

<repulsion\_stiffness value="800.0"/>

<friction value= "0.5"/>

<neohookean mu= "60" lambda= "200" damping= "0.01" />

<visual filename="torus.vtk"/>

</deformable>

</robot>

## createSoftBodyAnchor

デフォルメ可能なオブジェクトの頂点をワールドに固定したり、createSoftBodyAnchorを使ってデフォルメ可能なオブジェクトの頂点をマルチボディにアタッチしたりすることができます。これにより、制約の一意のidが返されます。この制約はremoveConstraint APIを使って削除することができます。

例として [deformable\_anchor.py](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/deformable_anchor.py) を参照してください。

getMeshData APIを使用して、変形可能な頂点にアクセスすることができます。

# 合成カメラレンダリング

PyBulletには、TinyRendererをベースにしたビルドインのOpenGL GPUビジュアライザーとビルドインのCPUレンダラーの両方があります。これにより、任意のカメラ位置から画像を非常に簡単にレンダリングすることができます。Linuxでは、Google Cloud Platformや[Colab](https://colab.sandbox.google.com/drive/1u6j7JOqM05vUUjpVp5VNk0pd8q-vqGlx)でのクラウドレンダリングなどのために、X11コンテキストなしでハードウェアアクセラレーションされたOpenGLレンダリングを有効にすることもできます。プラグイン」のセクションで説明されているように、使用方法は [eglRenderTest.py の](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/eglRenderTest.py)例を参照してください。

合成カメラは，ビュー行列とプロジェクション行列という4×4の2つの行列で指定されます．これらの行列は直感的ではないので，わかりやすいパラメータからビュー行列とプロジェクション行列を計算するヘルパーメソッドがあります．

OpenGLのカメラ情報へのリンクがある、本質的なカメラマトリックスに関する[記事を](http://ksimek.github.io/2013/08/13/intrinsic/)チェックしてみてください。

## computeView/ProjectionMatrix

computeViewMatrixの入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | cameraEyePosition | vec3, list of 3 floats | 直交座標における目の位置 |
| required | cameraTargetPosition | vec3, list of 3 floats | 目標点位置 |
| required | cameraUpVector | vec3, list of 3 floats | カメラのアップベクトル（直交世界座標での |
| optional | physicsClientId | int | 未使用、APIの一貫性のために追加 |

出力は，16個のフロートのリストとして格納された4x4のビュー行列です．

computeViewMatrixFromYawPitchRoll

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | cameraTargetPosition | list of 3 floats | 直交世界座標の目標焦点点 |
| required | distance | float | しゅうちゅうてんきょり |
| required | yaw | float | 上軸を中心とした左右方向のヨー角。 |
| required | pitch | float | ピッチを上下方向に設定しています。 |
| required | roll | float | 前方ベクトルを中心とした回転 |
| required | upAxisIndex | int | Y軸を1、Z軸を2のどちらかを上にします。 |
| optional | physicsClientId | int | 未使用ですが、APIの一貫性のために追加されました。 |

出力は，16個のフロートのリストとして格納された4x4のビュー行列です．

computeProjectionMatrix

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | left | float | 左画面座標 |
| required | right | float | 右画面座標 |
| required | bottom | float | 下画面座標 |
| required | top | float | トップ画面座標 |
| required | near | float | きんきょくめんきょり |
| required | far | float | えんめんきょり |
| optional | physicsClientId | int | 未使用ですが、APIの一貫性のために追加されました。 |

出力は，16個のフロートのリストとして格納された4x4の射影行列です．

computeProjectionMatrixFOV

このコマンドはまた、異なるパラメータを使用して、4x4の投影行列を返します。パラメータの意味については、OpenGLのドキュメントを参照してください。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | fov | float | 視野 |
| required | aspect | float | アスペクト比 |
| required | nearVal | float | near plane distance |
| required | farVal | float | far plane distance |
| optional | physicsClientId | int | 未使用ですが、APIの一貫性のために追加されました。 |

## getCameraImage

getCameraImage APIは、RGB画像、深度バッファ、および各ピクセルの可視オブジェクトのボディ固有のIDを持つセグメンテーションマスクバッファを返します。PyBullet は numpy オプションを使用してコンパイルできることに注意してください: numpy を使用すると、カメラピクセルを C から Python にコピーする際のパフォーマンスが向上します。注意: 古い renderImage API は廃止され、getCameraImage に置き換えられました。

getCameraImage入力パラメータ。

|  |  |  |  |
| --- | --- | --- | --- |
| required | width | int | 水平解像度 |
| required | height | int | 垂直解像度 |
| optional | viewMatrix | 16 floats | 4x4のビュー行列， computeViewMatrix\* を参照してください． |
| optional | projectionMatrix | 16 floats | 4x4の射影行列， computeProjection\* を参照してください． |
| optional | lightDirection | vec3, list of 3 floats | lightDirectionは光源のワールド位置を指定し、光源の位置からワールドフレームの原点までの方向を指定します。 |
| optional | lightColor | vec3, list of 3 floats | 範囲0～1の[RED,GREEN,BLUE]の方向性光色、ER\_TINY\_RENDERERのみ適用 |
| optional | lightDistance | float | 正規化されたlightDirectionに沿った光の距離、ER\_TINY\_RENDERERにのみ適用されます。 |
| optional | shadow | int | 影がある場合は1、影がない場合は0、ER\_TINY\_RENDERERにのみ適用されます。 |
| optional | lightAmbientCoeff | float | 光環境係数、ER\_TINY\_RENDERERのみに適用されます。 |
| optional | lightDiffuseCoeff | float | 光拡散係数、ER\_TINY\_RENDERERのみに適用されます。 |
| optional | lightSpecularCoeff | float | 光鏡面係数、ER\_TINY\_RENDERERにのみ適用されます。 |
| optional | renderer | int | ER\_BULLET\_HARDWARE\_OPENGLまたはER\_TINY\_RENDERERが必要です。なお、DIRECTモードにはOpenGLがないので、ER\_TINY\_RENDERERが必要です。 |
| optional | flags | int | ER\_SEGMENTATION\_MASK\_OBJECT\_AND\_LINKINDEX, segmentationMaskBufferの説明とコード例は以下を参照してください。  セグメンテーション・マスクの計算を避けるために ER\_NO\_SEGMENTATION\_MASK を使用します。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getCameraImageは、パラメータのリストを返します。

|  |  |  |
| --- | --- | --- |
| width | int | 幅画像解像度（水平ピクセル |
| height | int | 高さ画像の解像度（ピクセル単位 |
| rgbPixels | list of [char RED,char GREEN,char BLUE, char ALPHA] [0..width\*height] | R,G,B,A形式のピクセルカラーのリスト。 |
| depthPixels | list of float [0..width\*height] | デプスバッファを使用しています。BulletはOpenGLを使用してレンダリングを行います。https://stackoverflow.com/questions/6652253/getting-the-true-z-value-from-the-depth-buffer を参照してください。  far=1000. //投影行列に依存します。  near=0.01/投影行列による  depth = far \* near / (far - (far - (far - near) \* depthImg)//depthImg は Bullet 'getCameraImage' からの深度です。  [PyBullet](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/pointCloudFromCameraImage.py) https://github.com/bulletphysics/bullet[3/blob/master/examples/pybullet/examples/pointCloudFromCameraImage.py も](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/pointCloudFromCameraImage.py)参照してください。 |
| segmentationMaskBuffer | list of int [0..width\*height] | 各ピクセルについては、可視オブジェクト固有の ID。ER\_SEGMENTATION\_MASK\_OBJECT\_AND\_LINKINDEXを使用した場合、SegmentationMaskBufferは以下のようにオブジェクト固有IDとリンクインデックスを結合する。  value = objectUniqueId + (linkIndex+1)<<24.[例を](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/segmask_linkindex.py)参照してください。  そのため、関節/リンクのないフリーフローティングボディの場合、リンクインデックスが -1 であるため、セグメンテーションマスクはボディ固有の ID と等しくなります。 |

isNumpyEnabled

C/C++からPythonへのピクセルのコピーは、NumPyを使ってPyBulletをコンパイルしない限り、大きな画像の場合は本当に遅いことに注意してください。NumPyが有効になっているかどうかは、PyBullet.isNumpyEnabled()を使って確認することができます。

今のところ、getCameraImageだけはnumpyを使ってアクセラレーションしています。

## getVisualShapeData

getVisualShapeDataを使用して、視覚的な形状情報にアクセスすることができます。これを使用して、独自のレンダリングメソッドをPyBulletシミュレーションとブリッジさせ、各シミュレーションステップの後にワールドトランスフォームを手動で同期させることができます。また、特に変形可能なオブジェクトでは、getMeshDataを使用して頂点の位置に関するデータを受け取ることもできます。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | objectUniqueId | int | ロードメソッドによって返されるオブジェクトの一意のID。 |
| optionsl | flags | int | VISUAL\_SHAPE\_DATA\_TEXTURE\_UNIQUE\_IDS は textureUniqueId も提供します。 |
| optional | physicsClientId | int | connect' が返す物理クライアント ID |

出力はビジュアルシェイプデータのリストで、各ビジュアルシェイプは以下の形式になっています。

|  |  |  |
| --- | --- | --- |
| objectUniqueId | int | オブジェクトの一意のID、入力と同じ |
| linkIndex | int | リンクインデックス、またはベースの場合は -1 |
| visualGeometryType | int | ビジュアルジオメトリタイプ |
| dimensions | vec3, list of 3 floats | 幾何学の寸法 |
| meshAssetFileName | string, list of chars | 三角形メッシュへのパス。通常はURDF、SDF、MJCFファイルの場所からの相対パスですが、絶対パスにすることもできます。 |
| localVisualFrame position | vec3, list of 3 floats | リンク/ジョイントフレームに対するローカルビジュアルフレームの位置 |
| localVisualFrame orientation | vec4, list of 4 floats | リンク/ジョイントフレームに対するローカルビジュアルフレームの向き |
| rgbaColor | vec4, list of 4 floats | URDF の色 (指定されている場合) を赤/緑/青/アルファで指定します。 |
| textureUniqueId | int | (このフィールドは VISUAL\_SHAPE\_DATA\_TEXTURE\_UNIQUE\_IDS フラグを使用している場合にのみ存在します)形状のテクスチャ固有の ID，なければ -1 |

物理シミュレーションでは、getBasePositionAndOrientationとgetLinkStateで、直交世界変換の参照として質量中心を使用しています。独自のレンダリングを実装する場合は、質量中心の世界変換と（逆の）localInertialFrameを利用して、ローカルの視覚変換を世界空間に変換する必要があります。localInertialFrameにはgetLinkState APIを使ってアクセスできます。

## changeVisualShape, loadTexture

changeVisualShapeを使って、シェイプのテクスチャやRGBAの色などのプロパティを変更することができます。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadメソッドによって返されるオブジェクトのユニークなID。 |
| required | jointIndex | int | リンクインデックス |
| optional | shapeIndex | int | 内部使用のための実験的なもので、 shapeIndex を無視するか -1 のままにしておくことを推奨します。 URDF (および SDF など) はリンクごとに 1 つ以上の視覚的な形状を持つことができるので、修正する特定の形状インデックスを選択できるようにするためです。この shapeIndex は、getVisualShapeData が返すリストの順序と一致します。 |
| optional | textureUniqueId | int | loadTexture' メソッドによって返されるテクスチャ固有の ID。 |
| optional | rgbaColor | vec4, list of 4 floats | RED、GREEN、BLUE、ALPHAの色成分を指定します。アルファは、現時点では 0 (不可視) または 1 (可視) でなければなりません。TinyRendererは透過性をサポートしていませんが、GUI/EGL OpenGL3レンダラーは透過性をサポートしていることに注意してください。 |
| optional | specularColor | vec3 | 鏡面色成分であるRED、GREEN、BLUEは、0から大数(>100)まであります。 |
| required | physicsClientId | int | connect' が返す物理クライアント ID |

loadTexture

ファイルからテクスチャを読み込み、読み込みに成功した場合は、負ではないテクスチャ固有の ID を返します。このユニークIDはchangeVisualShapeで使用することができます。

# 衝突検出クエリ

最後の'stepSimulation'の間に存在したコンタクトポイントの情報を問い合わせることができます。コンタクトポイントを取得するには、'getContactPoints' API を使用します。getContactPoints' はコンタクトポイント情報を再計算しないことに注意してください。

## getOverlappingObjects, getAABB

このクエリは、指定された軸合わせバウンディングボックスと軸合わせバウンディングボックスが重なっているオブジェクトの一意のIDをすべて返します。このクエリは保守的であり、実際のAABBが重複していないオブジェクトも返される可能性があることに注意してください。これは、加速度構造には、AABBを少し拡大するヒューリスティックな機能があるためです(余白を増やして速度ベクトルに沿って押し出す)。

getOverlappingObjectsの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | aabbMin | vec3, list of 3 floats | minimum coordinates of the aabb |
| required | aabbMax | vec3, list of 3 floats | maximum coordinates of the aabb |
| optional | physicsClientId | int | if you are connected to multiple servers, you can pick one. |

getOverlappingObjectsは、オブジェクトのユニークなIDのリストを返します。

getAABB

オブジェクトの一意の ID と、オプションでリンクインデックスを指定して、軸を揃えたバウンディングボックス（ワールドスペース）を問い合わせることができます（リンクインデックスを渡さない場合や -1 を使用する場合は、ベースの AABB を取得します）。(リンクインデックスを渡さない場合や-1を使用した場合は、ベースのAABBを取得します)。

入力パラメータは

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | 作成メソッドで返されるオブジェクト固有のID。 |
| optional | linkIndex | int | 範囲 [0..getNumJoints(....]] のリンクインデックス |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

戻り値の構造体は、ワールドスペース座標のvec3、aabbMin(x,y,z)、aabbMax(x,y,z)のリストです。

[getAABB.py の](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/getAABB.py)例も参照してください。

## getContactPoints, getClosestPoints

getContactPoints APIは、stepSimulationの直近の呼び出し時に計算された接触点を返します。stepSimulationの後にシミュレーションの状態を変更した場合、'getContactPoints'は更新されず、無効になる可能性があることに注意してください。その入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | bodyA | int | bodyAに関係する接触点のみを報告する |
| optional | bodyB | int | 重要: bodyBを提供する場合は、有効な身体Aを持っている必要があります。 |
| optional | linkIndexA | int | bodyA の linkIndexA を含むコンタクトポイントのみを報告する。 |
| optional | linkIndexB | int | bodyB の linkIndexB を含むコンタクトポイントのみを報告する。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getContactPointsはコンタクトポイントのリストを返します。各コンタクトポイントには以下のフィールドがあります。

|  |  |  |
| --- | --- | --- |
| contactFlag | int | reserved |
| bodyUniqueIdA | int | body unique id of body A |
| bodyUniqueIdB | int | body unique id of body B |
| linkIndexA | int | link index of body A, -1 for base |
| linkIndexB | int | link index of body B, -1 for base |
| positionOnA | vec3, list of 3 floats | contact position on A, in Cartesian world coordinates |
| positionOnB | vec3, list of 3 floats | contact position on B, in Cartesian world coordinates |
| contactNormalOnB | vec3, list of 3 floats | contact normal on B, pointing towards A |
| contactDistance | float | contact distance, positive for separation, negative for penetration |
| normalForce | float | normal force applied during the last 'stepSimulation' |
| lateralFriction1 | float | lateral friction force in the lateralFrictionDir1 direction |
| lateralFrictionDir1 | vec3, list of 3 floats | first lateral friction direction |
| lateralFriction2 | float | lateral friction force in the lateralFrictionDir2 direction |
| lateralFrictionDir2 | vec3, list of 3 floats | second lateral friction direction |

getClosestPoints

stepSimulationとは独立して、最も近い点を計算することも可能です。これにより、任意の離間距離を持つオブジェクトの最近接点を計算することもできます。このクエリでは、法線力は報告されません。

getClosestPoints 入力パラメータ。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyA | int | オブジェクト 一意のオブジェクト(A)のためのID |
| required | bodyB | int | オブジェクト第２のオブジェクトに固有のＩＤ（Ｂ |
| required | distance | float | オブジェクト間の距離がこの最大距離を超えると、ポイントは返されません。 |
| optional | linkIndexA | int | オブジェクト A のリンクインデックス (ベースの場合は -1) |
| optional | linkIndexB | int | オブジェクトBのリンクインデックス(ベースの場合は-1) |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

getClosestPoints は、getContactPoints と同じ形式で最も近い点のリストを返します (ただし、この場合の normalForce は常に 0 です)。

## rayTest, rayTestBatch

1回のレイキャストを実行して、最初にヒットしたオブジェクトの交点情報を見つけることができます。

rayTestの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | rayFromPosition | vec3, list of 3 floats | 世界座標における光線の開始点 |
| required | rayToPosition | vec3, list of 3 floats | 世界座標における光線の終点 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

raytestクエリは、交点がある場合には以下の情報を返します。

|  |  |  |
| --- | --- | --- |
| objectUniqueId | int | オブジェクト ヒットしたオブジェクトのユニークID |
| linkIndex | int | ヒットしたオブジェクトのリンクインデックス。 |
| hit fraction | float | レイに沿った範囲[0,1]のレイに沿ったヒット率。 |
| hit position | vec3, list of 3 floats | 直交座標でのヒット位置 |
| hit normal | vec3, list of 3 floats | 直交世界座標で正常に当たる |

rayTestBatch

これは rayTest に似ていますが、より高速に実行するために光線の配列を提供することができます。rayFromPositions' のサイズは 'rayToPositions' のサイズと同じにする必要があります。交差がなくても、レイごとに 1 つのレイの結果を得ることができます: レイが何かにヒットしたかどうかをチェックするために objectUniqueId フィールドを使用する必要があります: objectUniqueId が -1 の場合はヒットしていません。バッチあたりのレイの最大数は pybullet.MAX\_RAY\_INTERSECTION\_BATCH\_SIZE です。

rayTestBatch の入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | rayFromPositions | list of vec3, list of list of 3 floats | 各光線の開始点のリスト (ワールド座標) |
| required | rayToPositions | list of vec3, list of list of 3 floats | 世界座標における各光線の終了点のリスト |
| optional | parentObjectUniqueId | int | 親オブジェクトのローカル空間にレイの発着がある |
| optional | parentLinkIndex | int | 親オブジェクトのローカル空間にレイの発着がある |
| optional | numThreads | int | レイテストを計算するために複数のスレッドを使用します (0 = 利用可能なすべてのスレッドを使用します。正の数 = 正確にこの数のスレッド、デフォルト =-1 = シングルスレッド) |
| optional | reportHitNumber | int | の代わりに、n 番目のヒットを報告することができます。 |
| optional | collisionFilterMask | int | は、 collisionFilterMask とボディ衝突フィルタグループの間のビット幅とビット幅が 0 以外の場合にのみテストを行います。ボディフィルタのマスク/グループを変更する方法については setCollisionFilterGroupMask を参照してください。 |
| optional | fractionEpsilon | float | reportHitNumberを使用している場合にのみ有用です: 同じ体に当たった場合、そのフラクションがこのフラクション内の既存のヒットと類似している場合、重複したヒットを無視します。例えば、ある光線が1つの体の多くの同一平面上の三角形に当たった場合、そのうちの1つのヒットにしか興味がないかもしれません。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

出力は、上記の rayTest クエリと同じ情報を持つ、入力レイごとのレイの交点の結果です。使い方は batchRayTest.py の例を参照してください。

## getCollisionShapeData

このクエリを使用して、既存のボディベースやリンクの衝突形状情報を取得することができます。getVisualShapeDataと非常によく似た動作をします。

getCollisionShapeDataの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | objectUniqueId | int | loadURDFなどから受信したオブジェクト固有のID |
| required | linkIndex | int | リンクインデックス、またはベースの場合は -1 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

戻り値は以下の内容のリストです。

|  |  |  |
| --- | --- | --- |
| object unique id | int | オブジェクトユニークアイディー |
| linkIndex | int | リンクインデックス、またはベースの場合は -1 |
| geometry type | int | ジオメトリタイプ。GEOM\_BOX、GEOM\_SPHERE、GEOM\_CAPSULE、GEOM\_MESH、GEOM\_PLANE |
| dimensions | vec3 | GEOM\_BOXの場合はextents、GEOM\_SPHEREの場合はdimensions[0] = radius、GEOM\_CAPSULEとGEOM\_CYLINDERの場合はdimensions[0] = height (length)、dimensions[1] = radiusとなります。GEOM\_MESHの場合、dimensionsはスケーリングファクタです。 |
| filename | string | GEOM\_MESHの場合のみ：衝突メッシュアセットのファイル名（とパス |
| local frame pos | vec3 | 質量/慣性フレームの中心を基準とした衝突フレームの局所位置。 |
| local frame orn | vec4 | 慣性フレームを基準とした衝突フレームの局所的な向き。 |

## Enable/Disable Collisions

デフォルトでは、異なる動的移動体間の衝突検出は有効になっています。同じボディのリンク間の自己衝突は、loadURDF の 'URDF\_USE\_SELF\_COLLISION' フラグのようなフラグを使って有効にすることができます (詳細は loadURDF コマンドを参照してください)。

オブジェクトのグループ間の衝突検出は、setCollisionFilterGroupMask APIを使用して有効・無効にすることができます。

V-HACD

PyBulletには、Khaled MamouによるVolumetric Hierarchical Approximate Decomposition (vhacd)の実装が含まれています。これは、凹型のWavefront .objファイルをインポートし、凸型分解した部分を含む新しいWavefront objファイルをエクスポートすることができます。これをPyBulletで使用することで、凹型の移動ジオメトリを効率的に扱うことができます。

静的な（動かない）凹型三角形メッシュ環境では、URDF ファイルのタグ（<link concave="yes" name="baseLink">）を使用して、三角形メッシュを凹型としてタグ付けするか、または flags=p.GEOM\_FORCE\_CONCAVE\_TRIMESH を指定して createCollisionShape を使用してください。

|  |  |  |  |
| --- | --- | --- | --- |
| required | fileNameIn | string | source (concave) Wavefront obファイル名 |
| required | fileNameOut | string | destination (convex decomposition) Wavefront objファイル名 |
| required | fileNameLog | string | ログファイル名 |
| optional | concavity | double | 最大許容凹み量（デフォルト=0.0025、範囲=0.0-1.0） |
| optional | alpha | double | 対称平面に沿ったクリッピングの偏りを制御します（デフォルトは0.05、範囲は0.0～1.0）。 |
| optional | beta | double | 回転軸に沿ったクリッピングの偏りを制御します（デフォルト=0.05、範囲=0.0～1.0）。 |
| optional | gamma | double | マージ段階で許容される凹みの最大値を制御します（デフォルト=0.00125、範囲=0.0-1.0）． |
| optional | minVolumePerCH | double | 生成された凸殻の適応的サンプリングを制御します(default=0.0001, range=0.0-0.01) |
| optional | resolution | int | ボクセル化段階で生成されるボクセルの最大数（default=100,000、範囲=10,000-16,000,000 |
| optional | maxNumVerticesPerCH | int | 凸殻あたりの三角形の最大数を制御します (default=64, range=4-1024) |
| optional | depth | int | クリッピングステージの最大数。各分割ステージの間、ユーザーが定義したしきい値よりも高い凹みを持つパーツは、最良のクリッピングプレーン（default=20、範囲=1～32）に従ってクリッピングされます。 |
| optional | planeDownsampling | int | \"best" clipping planeの検索の粒度を制御します。 |
| optional | convexhullDownsampling | int | クリッピングプレーン選択段階での凸版生成処理の精度を制御します(default=4、範囲=1～16) |
| optional | pca | int | 凸分解を適用する前にメッシュを正規化することを有効/無効にします（default=0, range={0,1}）． |
| optional | mode | int | 0: ボクセルベースの近似凸分解, 1: 四面体ベースの近似凸分解 (default=0, range={0,1}) |
| optional | convexhullApproximation | int | 凸殻計算時の近似の有効・無効(default=1, range={0,1}) |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。  注: vhacd の分解は現在クライアント側で行われています。 |

使用例

import pybullet as p

import pybullet\_data as pd

import os

p.connect(p.DIRECT)

name\_in = os.path.join(pd.getDataPath(), "duck.obj")

name\_out = "duck\_vhacd2.obj"

name\_log = "log.txt"

p.vhacd(name\_in, name\_out, name\_log)

setCollisionFilterGroupMask

それぞれの身体はグループの一部です。グループとマスクが一致していれば他の身体と衝突し、逆にグループとマスクが一致していれば他の身体と衝突します。以下のチェックは、関係する二つのボディのグループとマスクを使用して行われます。衝突フィルタのモードに依存します。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | 設定されるボディの bodyUniqueId |
| required | linkIndexA | int | 構成される本体のリンクインデックス |
| required | collisionFilterGroup | int | フィルタのビット単位のグループ、説明は以下を参照してください。 |
| required | collisionFilterMask | int | フィルタのビット単位のマスク、説明は以下を参照してください。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

特定のリンクのペア間の衝突検出をより細かく制御することができます。setCollisionFilterPair API を使用して、コリジョン検出を有効にしたり無効にしたりすることができます。

setCollisionFilterPair

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueIdA | int | フィルタリングされるボディ A の bodyUniqueId |
| required | bodyUniqueIdB | int | フィルタリングされるボディBのbodyUniqueId, A==Bは自己衝突を意味します。 |
| required | linkIndexA | int | 本体Aのリンクインデックス |
| required | linkIndexB | int | 本体Bのリンクインデックス |
| required | enableCollision | int | 1で衝突を有効にし、0で衝突を無効にします。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

独自のコリジョンフィルタリング[実装](https://github.com/bulletphysics/bullet3/tree/master/examples/SharedMemory/plugins/collisionFilterPlugin)を書くためのプラグインAPIもあります。

# 逆動力学、運動学

## calculateInverseDynamics(2)

calculateInverseDynamicsは、指定された関節位置と速度から、指定された関節加速度に到達するために必要な力を計算します。逆力学は、再帰的ニュートンオイラーアルゴリズム（RNEA）を使用して計算されます。

calculateInverseDynamicsの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディ固有の ID。 |
| required | objPositions | list of float | 各自由度（DoF）の関節位置（角度）。固定関節の自由度は0であることに注意してください。ベースはすべての場合（フローティングベースと固定ベース）でスキップ/無視されます。 |
| required | objVelocities | list of float | 各自由度の関節速度 |
| required | objAccelerations | list of float | 各自由度(DoF)に対する所望の関節加速度 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

calculateInverseDynamicsは、各自由度の関節力のリストを返します。

マルチドーフ（球状）ジョイントを使用する場合、calculateInverseDynamicsは異なるコードパスを使用し、少し遅くなることに注意してください。また、計算InverseDynamicsはジョイント/リンクのダンピングを無視しますが、フォワードダイナミクス（stepSimulation）はダンピング項を含んでいることにも注意してください。そのため、インバースダイナミクスとフォワードダイナミクスを比較したい場合は、changeDynamicsでジョイントダンピングを使用してダンピング項をゼロに設定し、linearDampingとangularDampingを使用してリンクダンピングを設定するようにしてください。

## calculateJacobian, MassMatrix

calculateJacobianは、リンク上の点の並進と回転のヤコビアンを計算します。返されるジャコビアンは、ルートリンクが固定かフローティングかによって若干異なります。フローティングの場合，ヤコビアンはルートリンクの自由度に対応する列を含み，固定の場合，ヤコビアンは関節に関連する列のみを持ちます．関数呼び出しは運動学的状態の完全な記述を取ります。これは、実際には計算InverseDynamicsが最初に呼び出され、目的のジャコビアンがこれから抽出されるためです。

計算ヤコビアンの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディ固有の ID。 |
| required | linkIndex | int | ヤコビアンのリンクインデックス |
| required | localPosition | list of float | 指定されたリンク上の点を指定し，その重心を中心としたリンクローカル座標でヤコビアンを計算します． |
| required | objPositions | list of float | 関節位置 |
| required | objVelocities | list of float | 関節速度 |
| required | objAccelerations | list of float | 所望の関節加速度 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

computJacobian returns.

|  |  |  |  |
| --- | --- | --- | --- |
| required | linearJacobian | mat3x  ((dof), (dof), (dof)) | 並進ヤコビアン、x\_dot = J\_t \* q\_dot. |
| required | angularJacobian | mat3x  ((dof), (dof), (dof)) | 回転ヤコビアン、r\_dot = J\_r \* q\_dot. |

[calculateMassMatrix](#_s0atq8gpssmy)

calculateMassMatrixは、関節位置を与えられた多関節体のシステム慣性を計算します。複合剛体アルゴリズム（CBRA）は、質量行列を計算するために使用されます。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF などで返されるボディ固有の ID。 |
| required | objPositions | array of float | 各リンクの jointPositions。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

結果は，次元が dofCount \* dofCount の正方質量行列で，各行は dofCount の質量行列要素のリストとして格納されます．

マルチドーフ（球面）ジョイントが含まれている場合，computateMassMatrixは少し遅い別のコードパスを使用することに注意してください．

逆運動学

エンドエフェクタが与えられた目標位置に到達するための関節角度を直交世界空間で計算することができます。Bulletは内部的には、Samuel Buss Inverse Kinematicsライブラリの改良版を使用しています。現時点では、Damped Least Squaresメソッドのみが公開されており、Null Space制御の有無に関わらず、単一のエンドエフェクタのターゲットを指定することができます。オプションでエンドエフェクタのターゲットの向きを指定することもできます。さらに、ヌルスペースを使用して関節制限やレストポーズを指定するオプションもあります。このオプションのヌル空間サポートは、4つのリスト（lowerLimits、 upperLimits、 jointRanges、 restPoses）すべてを必要とし、それ以外の場合は通常のIKが使用されます。詳細はBullet/examples/pybullet/examplesフォルダのinverse\_kinematics.pyの例を参照してください。

## calculateInverseKinematics(2)

calculateInverseKinematicsの入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF によって返されるボディ固有の ID |
| required | endEffectorLinkIndex | int | エンドエフェクターリンクインデックス |
| required | targetPosition | vec3, list of 3 floats | エンドエフェクタのターゲット位置（質量中心座標ではなくリンク座標）。デフォルトでは、currentPositionの関節角度を指定しない限り、これはデカルト世界空間です。 |
| optional | targetOrientation | vec3, list of 4 floats | 直交世界空間におけるターゲットの向き、四分儀 [x,y,w,z]。指定がない場合は、純粋な位置 IK が使用されます。 |
| optional | lowerLimits | list of floats [0..nDof] | オプションのヌル空間 IK は、4 つのリストすべて（lowerLimits、 upperLimits、 jointRanges、 restPoses）を必要とします。それ以外の場合は、通常の IK が使用されます。制限を持つジョイントのみを提供する（固定ジョイントはスキップ）ので、長さは自由度の数になります。lowerLimits, upperLimits, jointRanges は、IK ソリューションにおいて競合や不安定性を引き起こしやすいことに注意してください。最初は休息のポーズだけで、広い範囲と制限を使ってみてください。 |
| optional | upperLimits | list of floats [0..nDof] | オプションのヌル空間 IK は、4 つのリストすべて（lowerLimit、 upperLimit、 jointRanges、restPoses）を必要とします。それ以外の場合は通常のIKが使用されます。 |
| optional | jointRanges | list of floats [0..nDof] | オプションのヌル空間 IK は、4 つのリストすべて（lowerLimits、 upperLimits、 jointRanges、 restPoses）を必要とします。それ以外の場合は、通常のIKが使用されます。 |
| optional | restPoses | list of floats [0..nDof] | オプションのヌル空間 IK は、4 つのリストすべて (lowerLimits, upperLimits, jointRanges, restPoses) を必要とします。それ以外の場合は通常のIKが使用されます。指定されたレストポーズに近いIKソリューションを使用します。 |
| optional | jointDamping | list of floats [0..nDof] | jointDampingは、ジョイントダンピング係数を使用してIKソリューションを調整することができます。 |
| optional | solver | int | p.IK\_DLSまたはp.IK\_SDLS、Damped Least SquaresまたはSelective Damped Least Squares、Samuel Bussの論文"Selectively Damped Least Squares for Inverse Kinematics"に記載されているように、Damped Least SquaresまたはSelective Damped Least Squaresです。 |
| optional | currentPosition | list of floats [0..nDof] | 関節位置のリスト。デフォルトでは、PyBulletはボディの関節位置を使用します。指定されている場合、targetPositionとtargetOrientationはローカル空間にあります! |
| optional | maxNumIterations | int | ターゲットと実際のエンド エフェクターの位置の間の距離がこのしきい値を下回るか、maxNumIterations に達するまで IK ソリューシ ョンを精密化します。デフォルトは 20 回です。 |
| optional | residualThreshold | double | ターゲットと実際のエンドエフェクターの位置の間の距離がこのしきい値を下回るか、maxNumIterations に達するまで IK ソリューションを絞り込みます。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

calculateInverseKinematicsは、各自由度のジョイント位置のリストを返します。例はBullet/examples/pybullet/inverse\_kinematics.pyを参照してください。

デフォルトでは、ターゲットエンドエフェクタと実際のエンドエフェクタの間の距離が残差閾値（1e-4）を下回るか、または反復回数の最大値に達するまで、IKは解を絞り込みます。

calculateInverseKinematics2

calculateInverseKinematicsと似ていますが、エンドエフェクターインデックスとそのターゲット位置のリストを受け取ります（現時点では方向性はありません）。

|  |  |  |  |
| --- | --- | --- | --- |
| required | bodyUniqueId | int | loadURDF によって返されるボディ固有の ID |
| required | endEffectorLinkIndices | list of int | エンドエフェクターリンクインデックス |
| required | targetPositions | list of vec3 | エンドエフェクタのターゲット位置（質量中心座標ではなくリンク座標）。デフォルトでは、currentPositionの関節角度を指定しない限り、これはデカルト世界空間です。 |
| ... | その他の引数については、 calculateInverseKinematics を参照してください。 |  |  |

# 

# 強化学習ジム環境

一連のRLジム環境は、"pip install pybullet"の間にインストールされます。これには、蟻、ホッパー、ヒューマノイド、ウォーカーなどのOpenAIジム環境のPyBulletバージョンが含まれます。また、Ghost RoboticsのMinitaur四足歩行ロボット、MITのレースカー、KUKAのロボットアーム把持環境のように、実際のロボットだけでなくシミュレーションにも適用される環境もあります。

pybullet, pybullet\_envs, pybullet\_dataのソースコードと例はこちらです。

<https://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym>。

DQN, PPO, TRPO, DDPGなどのRL学習アルゴリズムで環境を学習することができます。いくつかの訓練済みの例が用意されていますので、このように楽しむことができます。

pip install pybullet, tensorflow, gym

python -m pybullet\_envs.examples.enjoy\_TF\_HumanoidBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.kukaGymEnvTest

## 環境とデータ

sudo pip install pybulletを実行すると、pybullet\_envsとpybullet\_dataパッケージが利用可能になります。pybullet\_envsパッケージをインポートすると、OpenAI Gymに自動的に環境が登録されます。

以下のPythonの行を使って、ジムのBullet環境の一覧を取得することができます。

print(

|  |  |  |
| --- | --- | --- |
| MinitaurBulletEnv-v0 | HumanoidDeepMimic\*BulletEnv-v1 | CartPoleContinuousBulletEnv-v0 |
| HumanoidBulletEnv-v0 | AntBulletEnv-v0 | HopperBulletEnv-v0 |
| KukaBulletEnv-v0 | HalfCheetahBulletEnv-v0 | Walker2DBulletEnv-v0 |

|  |  |
| --- | --- |
| 環境名 | 説明 |
| MinitaurBulletEnv-v0 | Ghost Robotics Minitaurの平地での四足歩行シミュレーション  この環境は、RSS 2018「Sim-to-Real: Learning Agile Locomotion For Quadruped Robots」で使用されたもので、[arxiv](https://arxiv.org/abs/1804.10332)の論文を参照してください。  移動距離に応じた報酬Gymを使ってクラスを作成します。  env = gym.make('MinitaurBulletEnv-v0')  または、クラスを使って直接、パラメータを指定して環境を作成します。  import pybullet\_envs.bullet.minitaur\_gym\_env as e  env = e.MinitaurBulletEnv(render=True) |
| HumanoidDeepMimicBackflipBulletEnv-v1とHumanoidDeepMimicWalkBulletEnv-v1    [ビデオを](https://www.youtube.com/watch?v=aiWxIjtMMFI)ご覧ください。  とHuman 3.6データセットを使用した[例です。](https://github.com/bulletphysics/bullet3/tree/master/examples/pybullet/gym/pybullet_envs/deep_mimic/mocap) | DeepMimic論文をPyBulletで再実装したもの: 参照運動を模倣したヒューマノイドのシミュレーション。この実装では、参照モーションを選択できるようになっています。バックフリップとウォークの参照モーションをジム環境にしています。  PyBullet の一部として、様々な運動の事前学習済みモデルが利用できます。  には Tensorflow 1.x (1.14) が必要です。  python3 -m pybullet\_envs.deep\_mimic.testrl --arg\_file  run\_humanoid3d\_backflip\_args.txt  python3 -m pybullet\_envs.deep\_mimic.testrl --arg\_file  run\_humanoid3d\_walk\_args.txt  また、付属のDeepMimic MPI並列化PPO実装を使用してトレーニングすることもできます。  python3 mpi\_run.py --arg\_file train\_humanoid3d\_walk\_args.txt --num\_workers 16  (お使いのマシンに依存してコア数16個を変更)  訓練されたポリシーを再生するには、ウェイトをコピーする必要があります。 |
| RacecarBulletEnv-v0 | MIT RCレースカーのシミュレーション。ランダムに配置されたボールまでの距離に応じた報酬。観測はカメラフレーム内のボールの位置(x,y)です。環境のアクション空間は離散的（DQNの場合）または連続的（PPO, TRPO, DDPGの場合）です。  import pybullet\_envs.bullet.racecarGymEnv as e  env = e.RacecarGymEnv(isDiscrete=False ,renders=True)  env.reset() |
| RacecarZedBulletEnv-v0 | RacecarBulletEnv-v0と同じですが、観測はカメラピクセルです。 |
| KukaBulletEnv-v0 | KUKA Iiwaのロボットアームを使って、トレイの中の物体を把持するシミュレーションです。主な報酬は、一定の高さ以上の物体を掴むことができたときに発生します。報酬は、各ステップごとに発生します。観察には、物体のx,y位置が含まれます。  注：この環境では、現在のところトレーニングに問題があり、それを調査しています。 |
| KukaCamBulletEnv-v0 | KukaBulletEnv-v0と同じですが、観測はカメラピクセルです。 |

[ロボスクール環境を](https://blog.openai.com/roboschool/)pybulletに移植しました。ロボスクールの環境はMuJoCoジムの環境よりも難しいです。

|  |  |
| --- | --- |
| AntBulletEnv-v0 | アリは重く、それは一般的に地面に2つ以上の足を持っていることを奨励しています。 |
| HalfCheetahBulletEnv-v0 |  |
| ヒューマノイドBulletEnv-v0 | ヒューマノイドは、報酬からエネルギーコスト（＝トルク×角速度）を差し引いた、より現実的なエネルギーコストの恩恵を受けています。 |
| ホッパーBulletEnv-v0 |  |
| ウォーカー2DBulletEnv-v0 |  |
| InvertedPendulumBullet-v0 |  |
| InvertedDoublePendumBulletEnv-v0 |  |
| InvertedPendulumSwingupBulletEnv-v0 |  |

また、URDF/SDFロボットアセット、Wavefront .OBJファイルなどのデータをpybullet\_dataパッケージからアクセスすることも可能です。以下にその方法の一例を示します。

import pybullet

import pybullet\_data

datapath = pybullet\_data.getDataPath()

pybullet.connect(pybullet.GUI)

pybullet.setAdditionalSearchPath(datapath)

pybullet.loadURDF("r2d2.urdf",[0,0,1])

あるいは、loadURDF/SDFコマンドでファイル名にデータパスを手動で追加します。

## Stable Baselines & ARS, ES,...

HalfCheetah（HalfCheetahBulletEnv\_v0）、Ant（AntBulletEnv\_v0）、（Hopper）HopperBulletEnv\_v0、CartPoleContinuousBulletEnv\_v0のような連続制御のジム環境では、[Stable Baselinesを](https://github.com/hill-a/stable-baselines)使用することができます。以下に例を示します。

pip3 install stable\_baselines --user

pip3 install pybullet --user

python3 -m pybullet\_envs.stable\_baselines.train --algo sac --env HalfCheetahBulletEnv-v0

学習環境を楽しむために、重みファイルをsac\_HalfCheetahBulletEnv-v0.zipにコピー/リネームしてください(\_bestの部分を削除してください)。

python3 -m pybullet\_envs.stable\_baselines.enjoy --algo sac --env HalfCheetahBulletEnv-v0 --n-episodes 5

[安定したBaselines Zooは](https://github.com/araffin/rl-baselines-zoo)、事前に訓練されたPyBullet環境を提供します。

また、Google ColabのノートブックでStable Baselinesを使ってPyBullet環境をトレーニングしたり、楽しんだりすることもできますが、この[Colabの例ではカートポールをトレーニングしています](https://colab.sandbox.google.com/drive/15JSROMJbeiqxcUwifPR2NYeeFBKmyIlX#scrollTo=E2eWDjPZsQc5)。

## 

Train and Enjoy: DQN, PPO, ES

KukaBulletEnv-v0やRacecarBulletEnv-v0のような離散的なジム環境では、[OpenAIのBaselines](https://github.com/openai/baselines) DQNを使って離散的な動作空間を使ってモデルを訓練することができます。これらの離散的な環境をどのように学習して楽しむか、いくつかの例を紹介します。

python -m pybullet\_envs.baselines.train\_pybullet\_cartpole

python -m pybullet\_envs.baselines.train\_pybullet\_racecar

OpenAI Baselinesは、モデルが改善されたときに指定された間隔で.PKLファイルを保存します。この.PKLファイルは、エンジョイスクリプトで使用されます。

python -m pybullet\_envs.baselines.enjoy\_pybullet\_cartpole

python -m pybullet\_envs.baselines.enjoy\_pybullet\_racecar

PyBulletには、アウトオブボックスを楽しめる事前学習済みのモデルもあります。以下は、事前にトレーニングされた環境を楽しむためのリストです。

python -m pybullet\_envs.examples.enjoy\_TF\_AntBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_HalfCheetahBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_AntBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_HopperBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_HumanoidBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_InvertedDoublePendumBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_InvertedPendulumBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_InertedPendulumSwingupBulletEnv\_v0\_2017may

python -m pybullet\_envs.examples.enjoy\_TF\_Walker2DBulletEnv\_v0\_2017may

TensorFlowとPyTorchを使ったトレーニング

TensorFlow [Agents PPOを](https://github.com/tensorflow/agents)使って様々なpybullet環境をトレーニングすることができます。まず、必要なPythonパッケージをインストールします: pip install gym, tensorflow, agents, pybullet, ruamel.yaml

それからトレーニング用に。

python -m pybullet\_envs.agents.train\_ppo --config=pybullet\_pendulum --logdir=pendulum

Agentの設定では、以下の環境が利用可能です。

pybullet\_pendulum

pybullet\_doublependulum

pybullet\_pendulumswingup

pybullet\_cheetah

pybullet\_ant

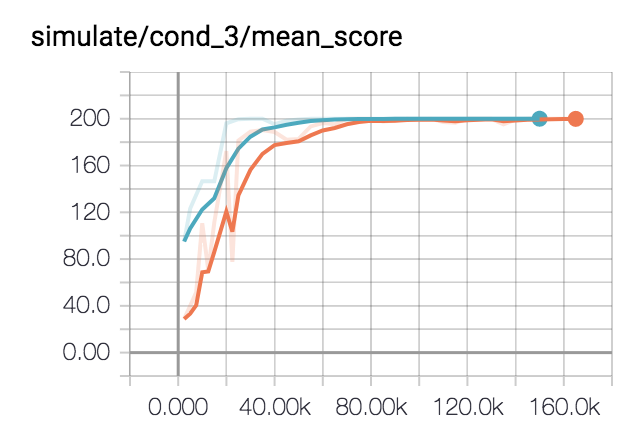
pybullet\_racecar

pybullet\_minitaur

tensorboardを使ってトレーニングの進捗状況を見ることができます。

tensorboard --logdir=pendulum --port=2222

Webブラウザを開き、localhost:2222のページにアクセスしてください。振り子のトレーニングのためのTensorboardのグラフの例です。



トレーニング後、トレーニングされたモデルをビジュアライズして動画を作成したり、物理サーバーを使ってビジュアライズしたりすることができます(python -m pybullet\_envs.examples.runServerまたは物理サーバーモードまたはVirtual RealityでExampleBrowserを使用します)。ローカルGUIの物理サーバーを起動すると、ビジュアライザー(bullet\_client.py)が自動的に接続し、OpenGLハードウェアレンダリングを使用してビデオを作成します。それ以外の場合は、代わりにCPUのTinyrendererを使用します。ビデオを生成するには

python -m pybullet\_envs.agent.visualize\_ppo --logdir=pendulum/xxxxx --outdir=pendulum\_video

同様の方法で、ミニタウロスのロボットを訓練し、視覚化することができます。

python -m pybullet\_envs.agent.train\_ppo --config=pybullet\_minitaur --logdir=pybullet\_minitaur

ミニタウロスの歩行の例のビデオはこちらです: <https://www.youtube.com/watch?v=tfqCHDoFHRQ>

Evolution Strategies (ES)

David Ha (hardmaru) さんのブログ記事で、Evolution Strategies を使った PyBullet 環境のトレーニング方法が <http://blog.otoro.net/2017/11/12/evolving-stable-strategies>にあります。

PyTorch PPOを使ったトレーニング

PyTorch と pybullet を使い始めるまでの説明を追記します。それまでの間は、こちらのリポジトリを参照してください: https://github.com/ikostrikov/pytorch-a2c-ppo-acktr

# バーチャルリアリティ

[vrBulletクイックスタートガイドも](https://docs.google.com/document/d/1I4m0Letbkw4je5uIBxuCfhBcllnwKojJAyYSTjHbrH8/edit)参照してください。 

VR物理サーバーは、HTC ViveとOculus Rift Touchのコントローラーに対応したOpenVR APIを使用しています。OpenVRは現在Windowsで動作していますが、Valveは[Linux版にも](https://github.com/ValveSoftware/openvr/issues/213)取り組んでいます。

また、共有メモリ、UDPまたはTCP接続を介してPyBulletを使用して完全に制御できるBulletの一部であるVRの例のビデオの例については、[https://www.youtube.com/watch?v=VMJyZtHQL50 を](https://www.youtube.com/watch?v=VMJyZtHQL50)参照してください。

WindowsでVRを使用する場合は、Microsoft Visual Studio (MSVC)を使用してBullet Physics SDKをコンパイルすることをお勧めします。build\_visual\_studio\_vr\_pybullet\_double.bat」スクリプトを実行してMSVCプロジェクトファイルを生成します。この小さなスクリプトをカスタマイズして、Pythonなどの場所を指すようにします。必ずMSVCの「リリース」設定に切り替えて、App\_PhysicsServer\_SharedMemory\_VR\*.exeをビルドして実行してください。デフォルトでは、このVRアプリケーションは、トラッカー/コントローラがある場合は空の世界を表示します。

## getVREvents,setVRCameraState

getVREvents は、最後に getVREvents を呼び出してから状態が変化した選択された VR デバイスのイベントのリストを返します。deviceTypeFilter を指定しない場合、デフォルトでは VR\_DEVICE\_CONTROLLER の状態のみを報告します。VR\_DEVICE\_CONTROLLER、VR\_DEVICE\_HMD (Head Mounted Device)、VR\_DEVICE\_GENERIC\_TRACKER (HTC Vive Tracker など) などのデバイスの任意の組み合わせを選択できます。

VR\_DEVICE\_HMDとVR\_DEVICE\_GENERIC\_TRACKERは、位置と向きのイベントのみを報告することに注意してください。

getVREventsには以下のパラメータがあります。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | deviceTypeFilter | int | デフォルトはVR\_DEVICE\_CONTROLLER  .VR\_DEVICE\_HMD を選択することもできます。  または VR\_DEVICE\_GENERIC\_TRACKER、またはそれらの組み合わせ。 |
| optional | allAnalogAxes | int | 全てのアナログ軸に1、1軸のみの場合は0 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

出力パラメータは

|  |  |  |
| --- | --- | --- |
| controllerId | int | コントローラインデックス (0～MAX\_VR\_CONTROLLERS) |
| controllerPosition | vec3, list of 3 floats | コントローラの位置、世界空間での直交座標 |
| controllerOrientation | vec4, list of 4 floats | 世界空間におけるコントローラ方位四元系 |
| controllerAnalogueAxis | float | アナログ軸値 |
| numButtonEvents | int | 最後にgetVREventsを呼び出してからのボタンイベント数 |
| numMoveEvents | int | 最後にgetVREventsを呼び出してからの移動イベント数 |
| buttons | int[64], list of button states (OpenVR has a maximum of 64 buttons) | 各ボタンのフラグを指定します。VR\_BUTTON\_IS\_DOWN（現在ダウンしている）、VR\_BUTTON\_WAS\_TRIGGERED（最後にgetVREventsを呼び出してから少なくとも一度はダウンした）、VR\_BUTTON\_WAS\_RELEASED（最後にgetVREventsを呼び出してから少なくとも一度はリリースされた）の各ボタンのフラグ。VR\_BUTTON\_IS\_DOWN のみが実際の現在の状態を報告することに注意してください。例えば、ボタンが下と上に行った場合、IS\_DOWN が false のままであっても、RELEASE/TRIGGERED フラグを見ればわかります。ログファイルでは、これらのボタンは1つの整数（ボタンごとに3ビット）で10個のボタンでパックされていることに注意してください。 |
| deviceType | int | デバイスのタイプ。VR\_DEVICE\_CONTROLLER、VR\_DEVICE\_HMDまたはVR\_DEVICE\_GENERIC\_TRACKER |
| allAnalogAxes (only if explicitly requested!) | list of 10 floats | 現在、MAX\_VR\_ANALOGUE\_AXISは、各軸x、yの値が5になっています。 |

VR描画の例はBullet/examples/pybullet/examples/vrEvents.pyを、HMDとジェネリックトラッカーを追跡するためのBullet/examples/pybullet/examples/vrTracker.pyを参照してください。

setVRCameraState

setVRCameraStateでは、カメラのルートトランスフォームのオフセット位置と向きを設定することができます。これにより、仮想世界におけるVRカメラの位置を制御することができます。また、VR カメラに車両などのオブジェクトを追尾させることも可能です。

setVRCameraStateには以下の引数があります（戻り値はありません）。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | rootPosition | vec3, vector of 3 floats | カメラ根元位置 |
| optional | rootOrientation | vec4, vector of 4 floats | 四角形[x,y,z,w]形式のカメラルートの向き。 |
| optional | trackObject | vec3, vector of 3 floats | トラックするオブジェクト固有の ID |
| optional | trackObjectFlag | int | flags.VR\_CAMERA\_TRACK\_OBJECT\_ORIENTATION（有効な場合、位置と向きの両方がトラッキングされます。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、その中から1つを選ぶことができます。 |

# Debug GUI, Lines, Text, Parameters

PyBulletには、シミュレーションのデバッグ、可視化、チューニングを容易にするための機能がいくつかあります。この機能は、GUIモードのような3D可視化ウィンドウがある場合や、別の物理サーバー（'Physics Server'モードのExample Browserや、OpenGL GUIのスタンドアロン物理サーバーなど）に接続されている場合にのみ有用です。

## addUserDebugLine, Text, Parameter

3次元の始点(from)と終点(to)、色[red,green,blue]、線幅、秒単位で指定した3次元線を追加することができます。addUserDebuglineの引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | lineFromXYZ | vec3, list of 3 floats | 直交座標線の始点 |
| required | lineToXYZ | vec3, list of 3 floats | 直交座標線の終点 |
| optional | lineColorRGB | vec3, list of 3 floats | RGBカラー [赤、緑、青] 各成分の範囲 [0...1] |
| optional | lineWidth | float | 線幅 (OpenGLの実装によって制限される) |
| optional | lifeTime | float | 永久的なライン、または秒の肯定的な時間のために 0 を使用して下さい（その後ラインはとの自動的に取除かれます |
| optional | parentObjectUniqueId | int | new in upcoming PyBullet 1.0.8: 親オブジェクト/リンクのローカル座標に線を引く。 |
| optional | parentLinkIndex | int | new in upcoming PyBullet 1.0.8: 親オブジェクト/リンクのローカル座標に線を引く。 |
| optional | replaceItemUniqueId | int | 既存の行を置き換える (パフォーマンスを向上させ、remove/addのちらつきを避けるため) [f10\_racecar.pyの](https://github.com/bulletphysics/pybullet_robots/blob/master/f10_racecar.py)例も参照してください。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

addUserDebugLineは、removeUserDebugItemを使用してその行を削除できるようにする非負のユニークなIDを返します('replaceItemUniqueId'を使用する場合はreplaceItemUniqueIdを返します)。( 'replaceItemUniqueId' を使用している場合は replaceItemUniqueId を返します)。

addUserDebugText

色とサイズを指定して、特定の場所に3Dテキストを追加することができます。入力引数は

|  |  |  |  |
| --- | --- | --- | --- |
| required | text | text | text represented as a string (array of characters) |
| required | textPosition | vec3, list of 3 floats | 直交世界座標におけるテキストの3D位置 [x,y,z] |
| optional | textColorRGB | vec3, list of 3 floats | RGBカラー [赤、緑、青] 各成分の範囲 [0...1] |
| optional | textSize | float | 文字サイズ |
| optional | lifeTime | float | 永久的なテキストのために 0 を使用して下さい、または秒の肯定的な時間（その後のテキストは自動的に取除かれます |
| optional | textOrientation | vec4, list of 4 floats | デフォルトでは、デバッグテキストは常にカメラの方を向き、自動的に回転します。テキストの向き（四角形）を指定することで、ワールド空間またはローカル空間（親が指定されている場合）で向きが固定されます。カメラに面したテキストには、異なる実装/シェーダが使用され、外観が異なることに注意してください：カメラに面したテキストはビットマップフォントを使用し、指定された向きのテキストはTrueTypeフォントを使用します。 |
| optional | parentObjectUniqueId | int | new in upcoming PyBullet 1.0.8: 親オブジェクト/リンクのローカル座標に線を引く。 |
| optional | parentLinkIndex | int | new in upcoming PyBullet 1.0.8: 親オブジェクト/リンクのローカル座標に線を引く。 |
| optional | replaceItemUniqueId | int | 既存のテキスト項目を置き換える(remove/addのちらつきを避けるため) |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

addUserDebugTextは負ではない一意のIDを返します。pybullet/examples/debugDrawItems.pyも参照してください。

addUserDebugParameter

addUserDebugParameter は、パラメータを調整するためにカスタムのスライダーやボタンを追加することができます。これは一意の ID を返します。これにより、readUserDebugParameter を使用してパラメータの値を読み取ることができます。addUserDebugParameter の入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | paramName | string | パラメータ名 |
| required | rangeMin | float | 最小値を指定します。最小値 > 最大値の場合、スライダーの代わりにボタンが表示されます。 |
| required | rangeMax | float | 最大値 |
| required | startValue | float | 開始値 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

readUserDebugParameterの入力パラメータは以下の通りです。

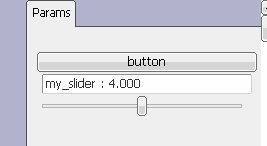
|  |  |  |  |
| --- | --- | --- | --- |
| required | itemUniqueId | int | addUserDebugParameter) が返す一意の ID |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

戻り値は、スライダの場合、パラメータの最新の読み込み値です。ボタンの場合、ボタンを押すたびにボタンに対するgetUserDebugParameterの値が1増加します。

Example:

p.addUserDebugParameter("button",1,0,1)

p.addUserDebugParameter("my\_slider",3,5,4)



removeAllUserParameters

これにより、すべてのスライダーとボタンが削除されます。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

removeUserDebugItem/All

ユーザデバッグ行、テキストを追加する関数は、成功すると負ではない一意のIDを返します。このユニークIDを使用してデバッグ項目を削除するには、removeUserDebugItemメソッドを使用します。

|  |  |  |  |
| --- | --- | --- | --- |
| required | itemUniqueId | int | 削除するデバッグ項目の一意の ID (行、テキストなど) |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

removeAllUserDebugItems

このAPIは、すべてのデバッグ項目(テキスト、行など)を削除します。

setDebugObjectColor

内蔵の OpenGL ビジュアライザには、ワイヤーフレームのデバッグレンダリング機能があります。ワイヤーフレームには、いくつかのデフォルトの色があります。特定のオブジェクトの色をオーバーライドしたり、setDebugObjectColorを使ってリンクしたりすることができます。入力パラメータは以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| required | objectUniqueId | int | オブジェクトのユニークID |
| required | linkIndex | int | リンクインデックス |
| optional | objectDebugColorRGB | vec3, list of 3 floats | デバッグカラーを [Red,Green,Blue] で指定します。指定しない場合、カスタムカラーは削除されます。 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

## addUserData

簡単に言えば、ボディの任意のリンクに添付されたテキスト文字列のユーザーデータを追加、削除、クエリすることができます。使い方は[userData.pyの](https://github.com/erwincoumans/bullet3/blob/master/examples/pybullet/examples/userData.py)例を参照してください。userDataIdを返します。urdfファイルにユーザデータを追加することもできます。

getUserData

getUserDataは、addUserDataによって返されたuserDataIdが与えられたユーザデータを受信します。使用例は userData.py を参照してください。

syncUserData

syncUserDataは、複数のクライアントがユーザデータを変更した場合(addUserDataなど)に備えて、ユーザデータ(getUserData)を同期化します。

removeUserData

removeUserData は、userDataId を指定して、以前に追加したユーザデータを削除します。

getUserDataId and getNumUserData

getNumUserDataは、bodyUniqueIdが与えられたユーザデータエントリの数を返します。

getUserDataInfo

getUserDataInfo は、ユーザデータのキーと識別子を (userDataId, key, bodyUniqueId, linkIndex, visualShapeIndex) として取得します。

## configureDebugVisualizer

ワイヤフレーム、シャドウ、GUIレンダリングの有効化や無効化など、内蔵OpenGLビジュアライザーのいくつかの設定を設定することができます。これは、ラップトップやデスクトップGUIの中には、OpenGL 3ビジュアライザーのパフォーマンスに問題があるものがあるので、便利です。

|  |  |  |  |
| --- | --- | --- | --- |
| required | flag | int | The feature to enable or disable, such as COV\_ENABLE\_WIREFRAME, COV\_ENABLE\_SHADOWS,COV\_ENABLE\_GUI, COV\_ENABLE\_VR\_PICKING, COV\_ENABLE\_VR\_TELEPORTING,  COV\_ENABLE\_RENDERING, COV\_ENABLE\_TINY\_RENDERER, COV\_ENABLE\_VR\_RENDER\_CONTROLLERS, COV\_ENABLE\_KEYBOARD\_SHORTCUTS, COV\_ENABLE\_MOUSE\_PICKING, COV\_ENABLE\_Y\_AXIS\_UP (Z is default world up axis),COV\_ENABLE\_RGB\_BUFFER\_PREVIEW, COV\_ENABLE\_DEPTH\_BUFFER\_PREVIEW, COV\_ENABLE\_SEGMENTATION\_MARK\_PREVIEW |
| required | enable | int | 0か1か |
| optional | lightPosition | vec3 | ビジュアライザーの光の位置 |
| optional | shadowMapResolution | int | シャドウマップテクスチャのサイズ、通常多くのGPUでは2の累乗です。デフォルトは4096です。最近のGPUは16384または32768以上を扱うことができます。 |
| optional | shadowMapWorldSize | int | 世界空間のシャドウマップのサイズ (単位はメートル、デフォルトは10) |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

例。

pybullet.configureDebugVisualizer(pybullet.COV\_ENABLE\_WIREFRAME,1)

## get/resetDebugVisualizerCamera

警告: getDebugVisualizerCameraの戻り値の引数は、resetDebugVisualizerCameraとは順序が異なります。将来のAPIリビジョン(メジャー新バージョン)で修正されます。

resetDebugVisualizerCamera

3D OpenGLデバッグビジュアライザーのカメラ距離（目とカメラターゲット位置の間）、カメラのヨーとピッチ、カメラターゲット位置をリセットすることができます。

|  |  |  |  |
| --- | --- | --- | --- |
| required | cameraDistance | float | distance from eye to camera target position |
| required | cameraYaw | float | camera yaw angle (in degrees) left/right |
| required | cameraPitch | float | camera pitch angle (in degrees) up/down |
| required | cameraTargetPosition | vec3, list of 3 floats | cameraTargetPosition is the camera focus point |
| optional | physicsClientId | int | if you are connected to multiple servers, you can pick one |

Example: pybullet.resetDebugVisualizerCamera( cameraDistance=3, cameraYaw=30, cameraPitch=52, cameraTargetPosition=[0,0,0])

getDebugVisualizerCamera

このコマンドを使用して、カメラの幅と高さ（ピクセル単位）、ビューと投影行列を取得できます。入力パラメータはオプションのphysicsClientIdです。出力情報は

|  |  |  |
| --- | --- | --- |
| width | int | カメラ画像の幅（ピクセル単位 |
| height | int | カメラ画像の高さ（ピクセル単位 |
| viewMatrix | float16, list of 16 floats | カメラのビューマトリックス |
| projectionMatrix | float16, list of 16 floats | カメラの投影行列 |
| cameraUp | float3, list of 3 floats | カメラの上軸（デカルト世界空間座標での |
| cameraForward | float3, list of 3 floats | カメラの前軸（デカルト世界空間座標 |
| horizontal | float3, list of 3 floats | TBDです。これは、光線を生成するために使用できる水平方向のベクトルです（マウスピッキングや単純なレイトレーサーの作成などに使用します）。 |
| vertical | float3, list of 3 floats | TBD.これはレイを生成するために使用できる垂直ベクトルです（マウスピックや単純なレイトレーサーの作成など）。 |
| yaw | float | カメラのヨー角（デカルト局所空間座標での |
| pitch | float | カメラのピッチ角（デカルト局所空間座標での |
| dist | float | カメラとカメラターゲットの距離 |
| target | float3, list of 3 floats | カメラの目標（デカルト世界座標 |

## getKeyboardEvents, getMouseEvents

前回 'getKeyboardEvents' を呼び出してから発生したすべてのキーボードイベントを受け取ることができます。各イベントはキーコードとステートを持っています。ステートは KEY\_IS\_DOWN, KEY\_WAS\_TRIGGERED, KEY\_WAS\_RELEASED のビットフラグの組み合わせです。キーが「上」から「下」に向かっている場合は、KEY\_WAS\_TRIGGERED と同様に KEY\_IS\_DOWN を受信します。キーが押されてリリースされた場合は、KEY\_IS\_DOWNとKEY\_WAS\_RELEASEDの状態になります。

いくつかの特殊なキーが定義されています。B3G\_F1 ... B3G\_F12、B3G\_LEFT\_ARROW、B3G\_RIGHT\_ARROW、B3G\_UP\_ARROW、B3G\_DOWN\_ARROW、B3G\_PAGE\_UP、B3G\_PAGE\_DOWN、B3G\_PAGE\_END、B3G\_HOME、B3G\_DELETE、B3G\_INSERT、B3G\_ALT、B3G\_SHIFT、B3G\_CONTROL、B3G\_RETURN。

getKeyboardEventsの入力はオプションのphysicsClientIdです。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

出力されるのは、キーコード'key'とキーボード状態'value'の辞書です。

例えば

qKey = ord('q')

keys = p.getKeyboardEvents()

if qKey in keys and keys[qKey]&p.KEY\_WAS\_TRIGGERED:

break;

getMouseEvents

getKeyboardEvents と同様に、最後に getMouseEvents を呼び出してから発生したマウスイベントを取得することができます。すべてのマウス移動イベントは、最新の位置を持つ単一のマウス移動イベントにマージされます。さらに、指定されたボタンのすべてのマウスボタンイベントがマージされます。ボタンが下と上に移動した場合、状態は 'KEY\_WAS\_TRIGGERED' になります。マウスボタンの状態は、KEY\_WAS\_TRIGGERED /KEY\_IS\_DOWN /KEY\_WAS\_RELEASEDを再利用します。

getMouseEventsへの入力引数は以下の通りです。

|  |  |  |  |
| --- | --- | --- | --- |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

出力されるのは、以下の形式のマウスイベントのリストです。

|  |  |  |
| --- | --- | --- |
| eventType | int | MOUSE\_MOVE\_EVENT=1, MOUSE\_BUTTON\_EVENT=2 |
| mousePosX | float | マウスポインタのx座標 |
| mousePosY | float | マウスポインタのy座標 |
| buttonIndex | int | マウスの左/中/右ボタンのボタンインデックス |
| buttonState | int | フラグ KEY\_WAS\_TRIGGERED /KEY\_IS\_DOWN /KEY\_WAS\_RELEASED |

マウスイベントの例としては、[createVisualShape.pyを](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/createVisualShape.py)参照してください。

# プラグイン

PyBulletでは、CやC++でプラグインを書いてカスタマイズ機能を追加することができる。PyBulletのいくつかのコア機能は、PD制御、レンダリング、gRPCサーバー、コリジョンフィルタリング、Virtual Realityシンクなどのプラグインとして記述されている。PyBulletのコア部分であるほとんどのプラグインは、デフォルトで静的にリンクされているので、手動でロードしたりアンロードしたりする必要はない。

Linuxでは、デフォルトでPyBulletに同梱されているプラグインの一例として、eglPluginがあります。これを有効にすることで、例えばGoogle Cloud Platform上のクラウドレンダリングのために、X11コンテキストなしでハードウェアOpenGL 3.xレンダリングを使用することができます。使用方法は [eglRenderTest.py の](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/eglRenderTest.py)例を参照してください。

PyBulletには、zipファイルから直接ファイルを読み込むことができ、ファイルキャッシュを可能にするfileIOPluginも付属しています。使用方法は、[fileIOPlugin.pyの](https://github.com/erwincoumans/bullet3/blob/master/examples/pybullet/examples/fileIOPlugin.py)例を参照してください。

## loadPlugin,executePluginCommand

PyBullet プラグインをロードするには、loadPlugin コマンドを使用します。

|  |  |  |  |
| --- | --- | --- | --- |
| required | pluginPath | string | path, プラグインを見つけるディスク上の場所 |
| required | postFix | string | 各APIに追加されるプラグインのpostfix名 |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

loadPlugin は pluginUniqueId の整数値を返します。この pluginId が負の値の場合、プラグインはロードされません。プラグインがロードされると、プラグインにコマンドを送信するには

executePluginCommand:

|  |  |  |  |
| --- | --- | --- | --- |
| required | pluginUniqueId | int | loadPlugin が返すプラグインの一意の ID。 |
| optional | textArgument | string | プラグインによって解釈されるオプションのテキスト引数 |
| optional | intArgs | list of int | プラグインで解釈される整数のオプションリスト |
| optional | floatArgs | list of float | プラグインで解釈されるフロートのオプションリスト |
| optional | physicsClientId | int | 複数のサーバーに接続している場合は、1つのサーバーを選ぶことができます。 |

unloadPlugin

プラグインをアンロードするには pluginId を使用します。

プラグインAPIはPyBulletと同じ基礎となるC APIを共有しており、PyBulletと同じ機能を持っています。

PyBulletの[プラグイン実装を](https://github.com/bulletphysics/bullet3/tree/master/examples/SharedMemory/plugins)ブラウザで閲覧することで、何が可能かを知ることができます。

# PyBullet のビルドとインストール

Windows、Mac OSX、LinuxにPyBulletをインストールする方法はいくつかあります。我々はPython 2.7とPython 3.5.2を使用していますが、ほとんどのPython 2.xとPython 3.xのバージョンが動作するはずです。PyBulletを動作させる最も簡単な方法は、pipかpython setup.pyを使うことです。

Pythonのpipを使う

Pythonとpipがインストールされていることを確認してから実行します。

pip install pybullet

*sudo pip install pybullet*か*pip install pybullet --userを*使う必要があるかもしれません。

pipを使ってPyBulletをインストールした場合、C++ Bullet Physics SDKもインストールすることが有益であることに注意してください: データファイル、物理サーバー、PyBulletに便利なツールが含まれています。

また、Bullet Physics SDK のルートで 'python setup.py build' と 'python setup.py install' を実行することもできます (SDK は <http://github.com/bulletphysics/bullet3>から入手してください)。

[https://pypi.python.org/pypi/pybullet も](https://pypi.python.org/pypi/pybullet)参照してください。

また、premake (Windows) や cmake を使ってソースコードから PyBullet をインストールすることもできます。

Windows用のpremakeを使用する

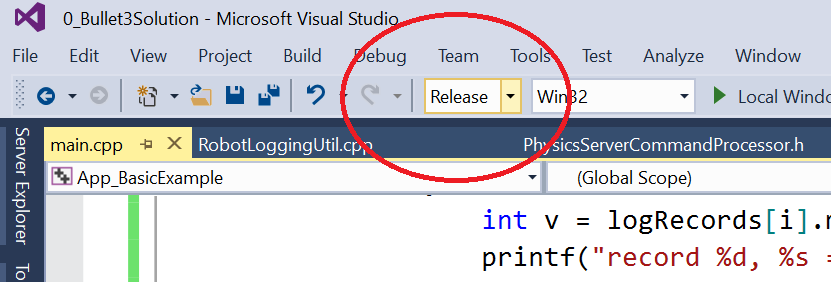
c:\python-3.5.2 (または他のバージョンのフォルダ名) に Python のバージョンがインストールされていることを確認してください。

まず、githubからソースコードを取得します。

git clone <https://github.com/bulletphysics/bullet3>

build\_visual\_studio\_vr\_pybullet\_double.batをクリックし、Visual Studioで0\_Bullet3Solution.slnプロジェクトを開き、必要に応じてプロジェクトを変換します。

Releaseモードに切り替え、'pybullet'プロジェクトをコンパイルします。



それから、Pythonインタプリタでpybulletをインポートするためのいくつかのオプションがあります。

1. pybullet\_vs2010.dllをpybullet.pydにリネームし、bullet/binをカレントワーキングディレクトリとしてPython.exeインタプリタを起動します。デバッグ用のオプション: bullet/bin/pybullet\_vs2010\_debug.dllをpybullet\_d.pydにリネームし、python\_d.exeを起動します。)
2. Rename bullet/bin/pybullet\_vs2010..dll to pybullet.pyd and use command prompt:  
   set PYTHONPATH=c:\develop\bullet3\bin (Bulletがある実際のフォルダに置き換える)か、Windows GUIを使ってこのPYTHONPATH環境変数を作成します。
3. 管理者プロンプト(cmd.exe)を作成し、以下のようにシンボリックリンクを作成します。

cd c:\python-3.5.2dlls

mklink pybullet.pyd c:\develop Develop Develop Develop Develop Develop Develop Develop Developbullet3bin\pybullet\_vs2010.dll

その後、python.exeを実行してpybulletをインポートすると動作するはずです。

Linux と Mac OSX で cmake を使用する

推奨される方法は sudo pip install pybullet (または pip3) であることに注意してください。cmakeやpremake、その他のビルドシステムを使うのは、自分が何をしているか知っている開発者だけが使うもので、一般的にはサポートされていません。

まず、githubからソースコードを取得します。

git clone <https://github.com/bulletphysics/bullet3>

1. cmake のダウンロードとインストール
2. Bullet:build\_cmake\_pybullet\_double.shのルートでシェルスクリプトを実行します。
3. Pythonがpybullet.soモジュールを見つけたことを確認してください。

export PYTHONPATH = /your\_path\_to\_bullet/build\_cmake/examples/pybullet

これだけです。python インタープリタを実行して pybullet をテストし、'import pybullet' と入力してモジュールがロードされるかどうかを確認します。もしロードできたら、Bullet/examples/pybulletにあるpybulletスクリプトで遊ぶことができます。

考えられるMac OSXの問題

* Mac OSXでpybulletをインポートする際に問題がある場合、正しいPythonインタプリタを実行し、-DPYTHON\_INCLUDE\_DIRと-DPYTHON\_LIBRARY(cmakeを使用)で設定されたinclude/librariesにマッチするようにしてください。homebrewを使っている場合など、複数のPythonインタプリタがインストールされている可能性があります。例として[このコメントを](https://github.com/bulletphysics/bullet3/issues/830#issuecomment-278015707)参照してください。
* CFLAGS='-stdlib=libc++' pip install pybulletで試してみてください。

考えられるLinuxの問題

* OpenGLがインストールされていることを確認してください。
* Anaconda を Python ディストリビューションとして使用する場合は、'GLIBCXX' が見つかるように libgcc をインストールしてください [(http://askubuntu.com/questions/575505/glibcxx-3-4-20-not-found-how-to-fix-this-error を参照)。](http://askubuntu.com/questions/575505/glibcxx-3-4-20-not-found-how-to-fix-this-error)
* Anaconda を Python ディストリビューションとして使用している場合、cmake が python libs を見つけられない可能性があります。./build\_cmake/CMakeCache.txtファイルにアクセスして、次の行を変更することで手動で追加することができます: 'PYTHON\_LIBRARY:FILEPATH=/usr/lib/python2.7/config-x86\_64-linux-gnu/libpython2.7.so'

OpenGL 3を欠いたGPUまたは仮想マシン

* デフォルトでは、PyBulletはOpenGL 3を使用します。 一部のリモートデスクトップ環境やGPUはOpenGL 3をサポートしていないため、アーチファクト（灰色の画面）が発生したり、クラッシュしたりすることがあります。これは完全にサポートされているわけではありませんが、シーンを表示する方法を提供してくれます。
  + pybullet.connect(pybullet.GUI,options="--opengl2")
* あるいは、物理サーバをリモートマシン上でUDPまたはTCPブリッジで実行し、ローカルラップトップからリモートサーバにUDPトンネリングで接続することもできます。

(todo: ステップを詳細に記述する)

# サポート、ヒント、引用

Question: どこに支援に行くのか、課題を報告するのか。

[http://pybullet.org/Bullet](https://pybullet.org/Bullet/phpBB3/viewforum.php?f=24) <https://github.com/bulletphysics/bullet3> and an issue tracker  
 に[ディスカッションフォーラムが](https://pybullet.org/Bullet/phpBB3/viewforum.php?f=24)Answer: あります。

Question: 論文中にPyBulletの引用を追加するにはどうすればいいですか？

回答: @MISC{coumans2020,

author = {Erwin Coumans and Yunfei Bai},

title = {PyBullet, a Python module for physics simulation for games, robotics and machine learning},

howpublished = {\url{[http://pybullet.org](http://pybullet.org/)}},

year = {2016--2020}

Question: PyBulletはGoogle Colabで使えるのか？

Answer: Colab で使用できるコンパイル済みの manlinux ホイールを提供しています。

また、EGLを使ってGPYレンダリングも動作します。Colabの例は[こちら](https://colab.research.google.com/drive/1u6j7JOqM05vUUjpVp5VNk0pd8q-vqGlx?authuser=1)。

Question: What happens to Bullet 2.x and the Bullet 3 OpenCL implementation?  
Answer: PyBulletはBullet [C-APIをラッピングして](https://github.com/bulletphysics/bullet3/blob/master/examples/SharedMemory/PhysicsClientC_API.h)います。Bullet 3のOpenCL GPU APIを入れていきます。

(および将来のBullet 4.x API)の背後には、このC-APIがあります。つまり、PyBullet や C-API を使用していれば、将来的には問題ありません。Bullet 2.x C++ APIと混同されることはありません。

Question: トルク/フォース制御と速度/位置制御モードのどちらを使うべきか？

一般的には、位置制御や速度制御から始めるのが良いでしょう。

力/トルク制御を確実に機能させるには、より多くの努力が必要になります。

Question: 物体の速度は思ったよりも小さいようです。PyBulletは適用されますか？

いくつかのデフォルトのダンピング？また、速度は100単位を超えないようになっています。

Answer: Yes, PyBullet applies some angular and linear damping to increase stability. You  
 can modify/disable this damping using the 'changeDynamics' command, using  
 引数にlinearDamping=0とangularDamping=0を指定しています。

最大線速/角速度は安定性を考慮して100単位でクランプしています。

Question: ロボットの一部（アームなど）だけ重力をオフにするには？

Answer:

現時点では、これは公開されていませんので、すべてのオブジェクトの重力加速度をオンにして、必要なオブジェクトに手動で重力を適用する必要があります。あるいは、実際のロボットのように、積極的に重力補正力を計算することもできます。Bulletは完全な制約システムを持っているので、反重力力を計算するのは簡単です。2つ目のシミュレーションを実行して（PyBulletでは複数の物理サーバに接続することができます）、ロボットを重力下に配置し、関節の位置制御を設定して希望する位置を維持し、「反重力」の力を収集します。そして、それらの力をメインのシミュレーションに適用します。

Question: オブジェクトをスケールアップ/ダウンするには？

Answer: loadURDFのオプション引数としてglobalScaleFactorの値を使用して

loadSDFを使用してください。そうでなければ、ビジュアルシェイプやコリジョンシェイプのスケーリングは、URDFやSDFのようなほとんどのファイルフォーマットに含まれています。現時点では、オブジェクトのスケールを変更することはできません。

Question: モデルにテクスチャを入れるにはどうすればいいですか？

Answer: Wavefront .objファイル形式を使用することができます。マテリアルファイル(.mtl)に対応します。

Bullet/data フォルダには、テクスチャを使用した様々な例があります。既存のテクスチャオブジェクトのテクスチャを変更するには、'changeTexture' APIを使用します。

Question: PyBulletで有効なテクスチャファイル形式は？

Answer: Bulletはテクスチャファイルを読み込むためにstb\_imageを使用しており、PNG、JPG、TGA、GIFなどを読み込みます。

詳細は [stb\_image.h](https://github.com/bulletphysics/bullet3/blob/master/examples/ThirdPartyLibs/stb_image/stb_image.h) を参照してください。

Question: 衝突検出の性能と安定性を向上させるにはどうしたらいいですか？

Answer: 例えば、最適化する方法はたくさんあります。

シェイプタイプ

1. 凸型や凹型の三角形メッシュを使用する代わりに、箱、球体、カプセル、円柱などのプリミティブなコリジョン形状を1つまたは複数選択してオブジェクトを近似します。
2. どうしても三角メッシュを使用する必要がある場合は、階層近似凸分解（v-HACD）を使用して凸分解を作成します。[test\_hacd ユーティリティは](https://github.com/bulletphysics/bullet3/blob/master/Extras/VHACD/test/src/premake4.lua)、OBJ ファイル内の凸型三角形メッシュを、複数の凸型ハルオブジェクトを持つ新しい OBJ ファイルに変換します。例えば、Bullet/data/teddy[2\_VHACD\_CHs.objを](https://github.com/bulletphysics/bullet3/blob/master/data/teddy2_VHACD_CHs.obj)指すBullet/data/teddy\_vhac[d.urdfや、duck\_vhacd.](https://github.com/bulletphysics/bullet3/blob/master/data/teddy_vhacd.urdf)objを指すduck\_vhacd.urdfを参照してください。
3. 三角形メッシュの頂点数を減らします。例えば、Blender 3Dには、メッシュの単純化の結果をインタラクティブに見ることができる素晴らしいメッシュデシメーションモディファイアがあります。
4. <link><contact>ノード内の<rolling\_friction>ノードと<spinning\_friction>ノードを使用して、球体やカプセル、ロボットグリッパーなどの丸い物体の転がり摩擦（例えば0.01）と回転摩擦に小さな正の値を使用します。例としてBullet/data/sphere2.urdfを参照してください。
5. URDF <link><contact> xmlノード内の<stiffness value="30000"/> <damping value="1000"/>を使用して、ホイールに少量のコンプライアンスを使用します。例えば、Bullet/data/husky/husky.urdf vehicleを参照してください。
6. Bulletの倍精度ビルドを使用すると、接触安定性と衝突精度の両方に優れています。制約ソルバーの設定とタイムステップを選択してください。
7. 物理シミュレーションをグラフィックスから切り離します。PyBulletは、GUIと様々な物理サーバーに対してこれを既に行っています: OpenGLグラフィックスの可視化は、物理シミュレーションとは独立した独自のスレッドで実行されます。

Question: 摩擦対応のオプションは何がありますか？

答え: デフォルトでは、BulletとPyBulletはクーロン摩擦モデルに正確な暗黙の円錐摩擦を使用します。さらに、<link><contact>ノードの中に<rolling\_friction>と<spinning\_friction>ノードを追加することで、転がり摩擦と回転摩擦を有効にすることができます（[例として](https://github.com/bulletphysics/bullet3/blob/master/data/sphere2.urdf)Bullet/data/sphere2.urdfを参照してください）。円錐体摩擦の代わりに、錐体近似を有効にすることができます。

Question: Bulletの中にある、高速を不可能にする定数や閾値は何ですか？

答えは次のとおりです。デフォルトでは、Bulletは貫通回復と同時に離散的な衝突検出を行います。純粋に離散的な衝突検出に頼るということは、1つのタイムステップ内でオブジェクトが自分の半径よりも速く移動してはいけないということを意味します。PyBulletでは、デフォルトのタイムステップとして1./240を使用しています。1./60よりも大きなタイムステップは、様々な理由で不安定になる可能性があります (ディープペネトレーション、数値積分器)。Bulletには、1つのタイムステップ内で半径よりも速く移動するオブジェクトの衝突を連続的に検出するオプションがあります。残念ながら、この連続的な衝突検出は、独自の問題（パフォーマンスや非物理的な応答、復帰の欠如）を引き起こす可能性があるため、この実験的な機能はデフォルトでは有効になっていません。この機能を有効にするには、ex[perimentalCcdSphereRadius.pyのサンプルを](https://github.com/bulletphysics/bullet3/blob/master/examples/pybullet/examples/experimentalCcdSphereRadius.py)チェックしてください。

Question: ドキュメント化されていないAPIがあります。通常、これは(1)クイックスタートガイドをまだ更新していないか、(2)その機能が実験的すぎて文書化できないということを意味します。ドキュメント化されていない特定のAPIについて本当に知りたい場合は、トラッカーに問題を提出することができます。