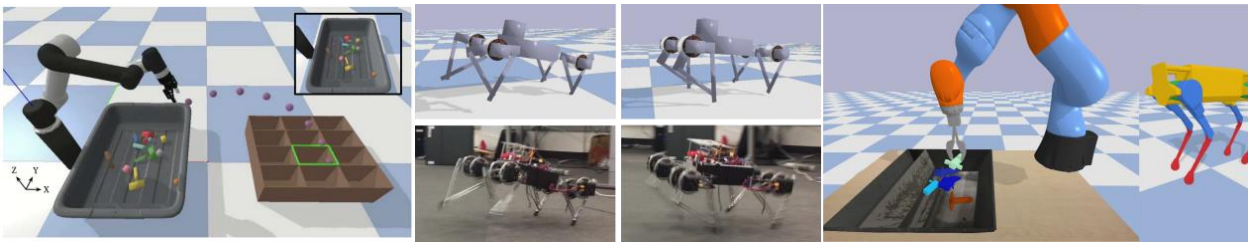
PyBullet クイックスタートガイド

Erwin Coumans,

Yunfei Bai デスクトップドキュメント、フォーラム、星の Bullet をご覧, 2016-2020 ください。

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、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 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 [1,2,3,4]、X[1,2]、Stanford AI Lab [1,2,3]、OpenAI,INRIA [1]などの研究者が 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

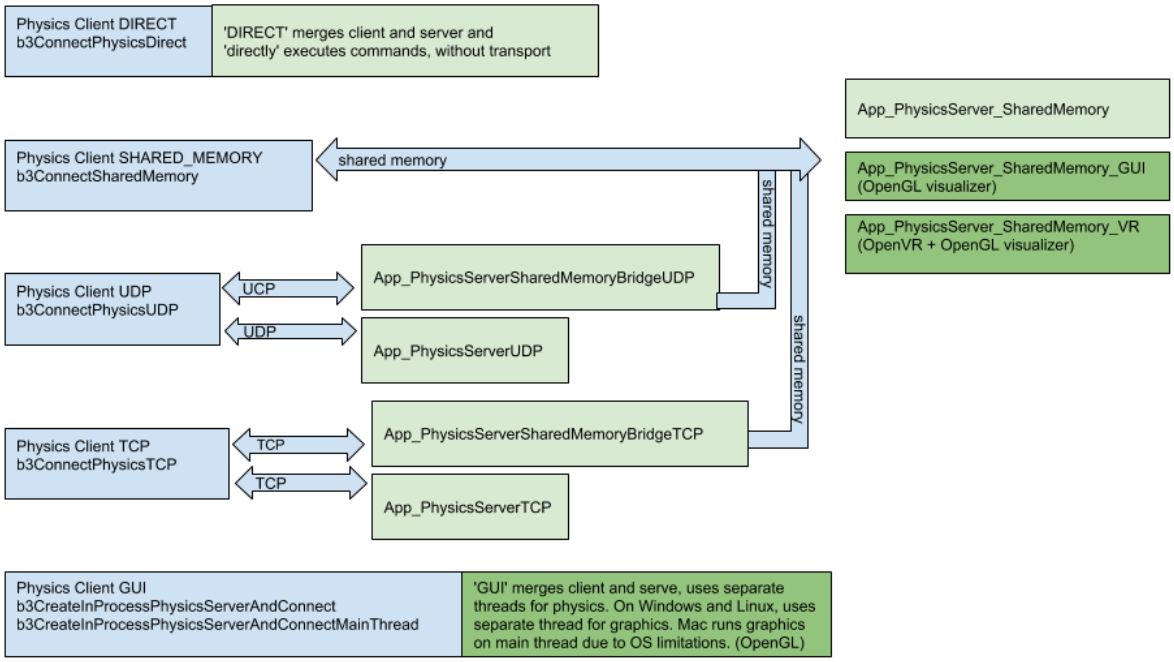
物理クライアント ID が与えられると、リスト[isConnected, connectionMethod]を返します。

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 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 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。

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 と createVisualShape.py の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。

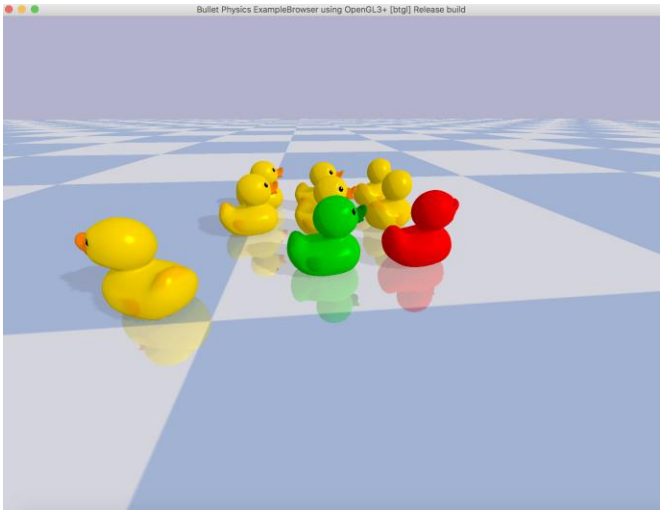
removeCollisionShape

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、createVisualShapeArray、createTexturedMeshVisualShape の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.
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.

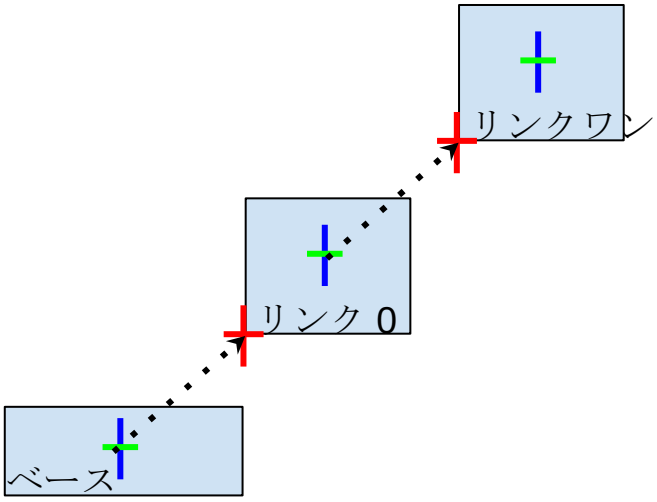
アピバージョン取得
API のバージョンを年月 0 日単位で問い合わせることができます。同じ API バージョンの物理クライアント/サーバ間では、同じビット数(32 ビット/64 ビット)の物理クライアント/サーバ間でのみ接続できます。オプションで未使用の引数 physicsClientId がありますが、これは API の整合性のために追加されたものです。

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



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]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

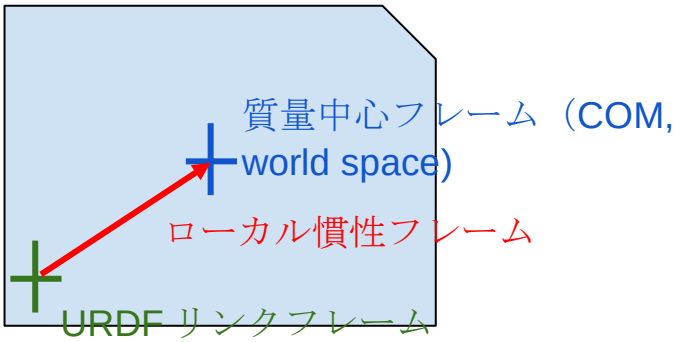
linkWorldPosition	vec3, list of 3 floats	質量中心の直交位置
linkWorldOrientation	vec4, list of 4 floats	質量中心の直交方向、四元系[x,y,z,w]における
localInertialFramePosition	vec3, list of 3 floats	URDF リンクフレームで表される慣性フレーム（質量中心）の局所位置オフセット

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。



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、ロギングや Python の dumpLog.py スクリプトの C++ ソースコードを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 や Assistive Gym など 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

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 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 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 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 も参照してください。
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

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。

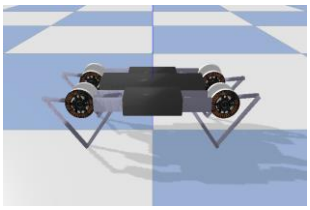
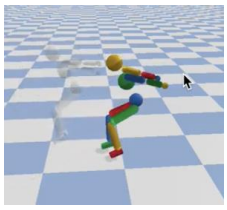
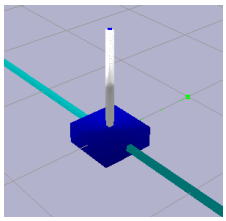
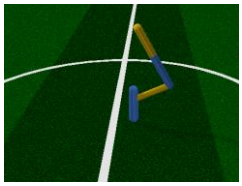
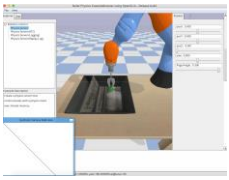
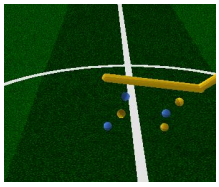
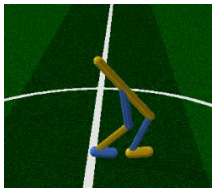
DQN, PPO, TRPO, DDPG などの RL 学習アルゴリズムで環境を学習することができます。いくつかの訓練済みの例が用意されていますので、このように楽しむことができます。

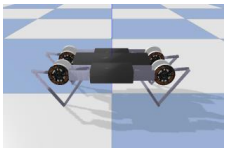
pip install pybullet, tensorflow, gym
python -m pybullet_envs.examples.enjoy_TF_HumanoidBulletEnv_v0_2017may
python -m pybullet_envs.examples.kukaGymEnvTest

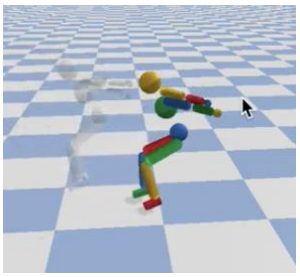
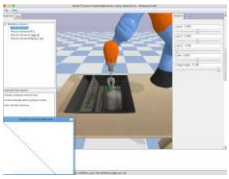
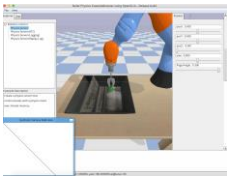
環境とデータ

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。

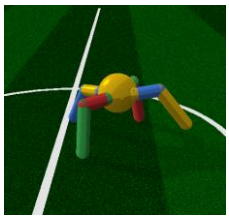
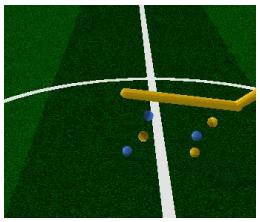
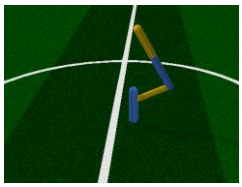
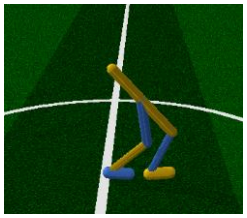
以下の Python の行を使って、ジムの Bullet 環境の一覧を取得することができます。
print(

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 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	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。

 ビデオをご覧ください。 と Human 3.6 データセットを使用した例です。	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 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、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。

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 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

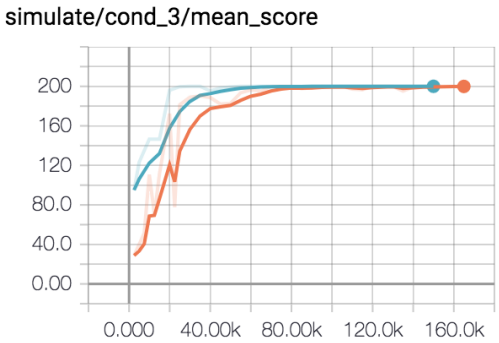
TensorFlow Agents PPO を使って様々な pybullet 環境をトレーニングすることができます。まず、必要な Python パッケージをインストールします: pip install gym, tensorflow, agents, pybullet, ruamel.yaml

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。

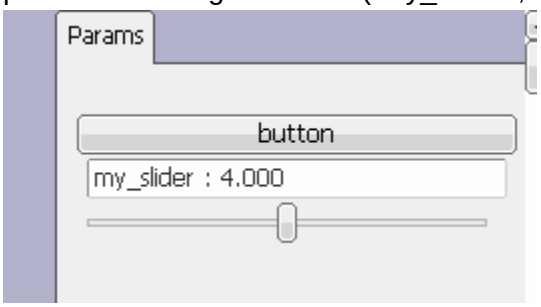
required	paramName	string	パラメータ名
required	rangeMin	float	最小値を指定します。最小値 > 最大値の場合、スライダーの代わりにボタンが表示されます。
required	rangeMax	float	最大値
required	startValue	float	開始値
optional	physicsClientId	int	複数のサーバーに接続している場合は、1 つのサーバーを選ぶことができます。

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。

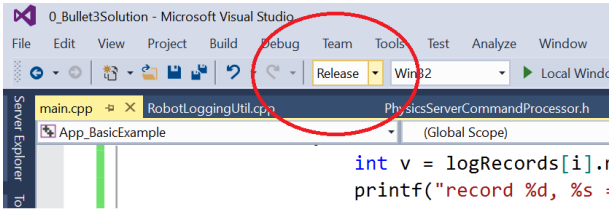
また、premake (Windows) や cmake を使ってソースコードから PyBullet をインストールすることもできます。

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)。
- 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://github.com/bulletphysics/bullet3 and an issue tracker
にディスカッションフォーラムが 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}},
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。

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 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、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/teddy2_VHACD_CHs.obj を指す Bullet/data/teddy_vhacd.urdf や、duck_vhacd.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 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、experimentalCcdSphereRadius.py のサンプルを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。