CCoMa Documentation

CWRU SME Lab

Classes:

Cable	1
CableSet	1
generateCables()	2
generateDebugSliders()	2
readDebugSliders()	2
ActuatorMotor	2
setMotorControl()	3
stepSimulation()	3
ActuatorPneumatic	3
setMotorControl()	4
stepSimulation()	4
ContinuumManipulator	4
stepSimulation()	5
updateSegments()	5
addNewCableSet()	5
setSpringConstant()	5
setCableTensions()	6
showCables()	6
hideCables()	6
showSegmentColors()	6
hideSegmentColors()	6
showModel()	6
hideModel()	7
addSoftBody()	7
getMassMatrix()	7

Functions:

multiplyQuaternion()	8
scaleVector()	8
vectorAngle()	8
invertQuaternion()	9
magnitude()	9
vecProject()	9
normalize()	10
quaternionFromAxisAngle()	10
applyRotation()	10
axisAngleFromQuaternion()	10
jointEulerFromQuaternion()	11
generateURDF()	11

Classes:

Cable

The Cable is a singular cable that applies force to a manipulator. Each cable has a length, tension, and a list of [xyz] positions. These are relative to the local origin of each manipulator link affected by the cable.

Inputs:

required	numJointsAffected	int	Total number of links affected
required	linkPositions	list of vec3 [xyz] floats, length numJointsAffected	Local position of each link anchor point
required	initialTension	float	Initial tension in the cable
required	debugRenderColor	vec3 [RGB]	Color of the cable as shown in the debug view

CableSet

The CableSet is a grouping of cables. Each CableSet requires an initial and final link that all cables in the set anchor to. Models affected by the cable set will actuate all links between these two. The positions of each cable are relative to the local frame of the affected links. Models will recolor links according to the set segmentColor.

Inputs:

required	startLink	int	Initial link of the cables
required	endLink	int	End link of the cables
required	numCables	int	Number of cables
optional	cables	list of Cable objects	All Cable objects comprising the set (optional, can be generated automatically) Default value: None
optional	segmentColor	vec4 floats	RGBA color of links containing cables Default value: [0.5,0.5,0.5,1.0]

generateCables()

Automatically generates a set of Cable objects to use based on given parameters.

Inputs:

required	radius	float	Distance from the center of each link on local xy plane
required	startAngle	float	Initial radian angle of placement from local x axis
required	linkLength	float	Length of each manipulator link
required	twist	float	Radian offset of each subsequent link

generateDebugSliders()

Generates UI debug sliders for manually setting Cable tensions.

Inputs:

required	maxTension	float	Maximum tension of the slider

readDebugSliders()

Read the value of the UI debug sliders generated and update the Cable tensions affected

ActuatorMotor

A model of an electric motor, stepper, or servo. Applies tension to a Cable object as a function of torque and radial distance.

Inputs:

required	manipulator	ContinuumManipulator object	Affect Continuum Manipulator
required	cable	Cable object	Cable attached to motor shaft output
required	maxTorque	float	Maximum torque output of the motor

required	wheelRadius	float	Radius of pulley or motor shaft attachment
optional	PIDCoeff	vec3 floats	Positional PID controller coefficients Default value: [1.0,1.0,1.0]

setMotorControl()

Sets either the output torque directly or uses a PID, feedback, and a desired Cable length. Does not directly actuate motor, only sets control mode for stepSimulation().

Inputs:

required	controlMode	int	Chooses either torque control or positional control, uses Pybullet flags p.TORQUE_CONTROL or p.POSITION_CONTROL
required	torque	float	Either the maximum torque in position control or the applied torque in torque control
optional	targetLength	float	The target cable length in position control, can be ignored for torque control. Default value: None

stepSimulation()

Steps the simulation forces, calculates required torque for position control and calculates/applies forces to Cable.

ActuatorPneumatic (INCOMPLETE)

A model of an pneumatic hybrid actuator. Applies tension to a Cable object using air pressure and physical displacement.

Inputs:

required	manipulator	ContinuumManipulator object	Affect Continuum Manipulator
required	cable	Cable object	Cable attached to actuator surface
required	actuationArea	float	Area of surface where pressure is applied
required	minPressure	float	Minimum system pressure

required	maxPressure	float	Maximum system pressure
required	minLength	float	Minimum actuator length
required	maxLength	float	Maximum actuator length

setMotorControl()

Sets the system pressure.

Inputs:

required	pressure	float	Target pressure to actuate with

stepSimulation()

Steps the simulation forces. Calculates the applied force as a result of pressure, current cable length, current actuator length, minimum/maximum pressure, and minimum/maximum length.

ContinuumManipulator

The ContinuumManipulator is the primary manipulator class. It accepts a PyBullet loaded URDF model as the shape of the manipulator, and a list of CableSets contained within the model. It uses a damped oscillator model to simulate the physical properties of the continuum volume.

Inputs:

required	modelId	int	PyBullet object ID of the manipulator body, loaded from URDF
required	cableSets	list of CableSet instances	All sets of cables that affect the motion of the manipulator
required	springConstant	float	Spring constant of manipulator joints, uses Hooke’s law
optional	linkLength	float	Length of all links in manipulator URDF
optional	doSelfCollision	bool	Option to enable model self-collision, requires loadURDF

			flag p.URDF_USE_SELF_COLLISION Default value: True
optional	maxLinkVelocity	float	Maximum angular velocity of all links in manipulator Default value: 5.0
optional	twistConstant	float	z-axis spring constant scaling factor Default value: 1.0

stepSimulation()

Calculates the physical motion of all joints in the manipulator. Uses Hooke's spring law to simulate the stiffness and internal friction of the material. Also applies the forces from all cables to each joint affected. Applies the forces to all joints using PyBullet's JointMotorControlArray(). Automatically updates all Cable lengths.

updateSegments()

Creates a Look Up Table (LUT) that defines a list of CableSets that affects each joint. Used for optimization, required when adding new CableSets to a manipulator.

addNewCableSet()

Adds a new CableSet object to the manipulator.

Inputs:

required	newCableSet	CableSet object	CableSet object to be added to manipulator.

setSpringConstant()

Directly sets a new spring constant for use in joint force calculations.

Inputs:

required	newSpringConstant	float	Desired value of spring constant

setCableTensions()

Directly sets new tensions in a given set of cables, similar to updateCableForces but does not utilize actuator formulas.

Inputs:

required	cableSetIndex	int	Index of desired cable set to update from cableSets
required	newCableTensions	list of *numCables* floats	List of all cable tensions to update, must match number of cables in set

showCables()

Draws the cables relative to each link in the manipulator using PyBullet's addUserDebugLine().

hideCables()

Hides the cables drawn with showCables using PyBullet's removeUserDebugItem(). Requires showCables() to have been used first.

showSegmentColors()

Applies the RGBA colors specified by each CableSet object to the affect links. Blends colors for links affected by multiple sets.

hideSegmentColors()

Returns the manipulator link colors to a default grey. Can optionally specify new RGBA color to use.

Inputs:

optional	modelColor	vec4 floats	New color for all links Default value: [0.5,0.5,0.5,1.0]

showModel()

Wrapper for *hideSegmentColors()*, intended for displaying collision shapes.

hideModel()

Renders all model links as invisible.

addSoftBody()

Intended to add a soft body object for collision simulation. Does not work yet.

Inputs:

required	softBodyID	int	Pybullet object ID of soft body object.

getMassMatrix()

Calculates the mass matrix of model. May be very large matrix, depends on number of links in model.

Functions:

multiplyQuaternion()

Calculate the product of two quaternion inputs. Uses JPL Quaternion Notation [xyzw]. Uses left multiplication (p = q0 * q1)

Inputs:

required	q0	vec4 floats	Left quaternion.
required	q1	vec4 floats	Right quaternion.

Outputs:

q2	vec4 floats	Product quaternion.

scaleVector()

Scales a vector by an arbitrary type numerical scalar.

Inputs:

required	S	numerical type object	Scalar value.
required	v0	vector of floats	Vector to be scaled.

Outputs:

v1	vector of floats	Scaled vector.

vectorAngle()

Computes the angle between vectors using a cross product.

Inputs:

required	v0	vector of floats	First vector.
required	v1	vector of floats	Second vector.

Outputs:

angle	float	Angle between vectors, in radians.

invertQuaternion()

Inverts a quaternion by negating only the [xyz] values.

Inputs:

required	q0	vec4 of floats	Quaternion to be inverted.

Outputs:

q1	vec4 of floats	Inverted quaternion.

magnitude()

Calculates the length of the vector.

Inputs:

required	vec	vector of floats	Vector of unknown magnitude.

Outputs:

magnitude	float	Magnitude of vector.

vecProject()

Projects the first vector onto the second vector.

Inputs:

required	v0	vec3 of floats	Vector projected from.
required	v1	vec3 of floats	Vector projected onto.

Outputs:

V2	vec3 of floats	Projection of first vector onto second.

normalize()

Normalizes a vector to magnitude of 1.

Inputs:

required	v	vector of floats	Vector to be normalized.

Outputs:

v1	vector of floats	Normalized vector.

quaternionFromAxisAngle()

Computes a quaternion that represents a rotation around a given [xyz] axis vector of a given radian angle.

Inputs:

required	axis	vec3 of floats	Axis of rotation.
required	angle	float	Radian angle of rotation.

Outputs:

quat0	vec4 of floats	Quaternion [xyzw].

applyRotation()

Applies a quaternion rotation to an [xyz] vector.

Inputs:

required	v	vec3 of floats	Vector to be rotated.
required	q	vec4 of floats	Quaternion representing rotation.

Outputs:

v	vec3 of floats	Rotated vector.

axisAngleFromQuaternion()

Computes the axis-angle representation of a quaternion rotation.

Inputs:

required	quat0	vec4 of floats	Quaternion representing rotation.

Outputs:

axis	vec3 of floats	Axis of rotation as [xyz] vector.
angle	float	Angle of rotation in radians.

jointEulerFromQuaternion()

Computes the local Euler angles that represent a quaternion rotation. Uses the order [rpy] ([xyz]), negates the yaw component for use in ContinuumManipulator [xyz] 3DOF joints.

Inputs:

required	q	vec4 of floats	Quaternion representing rotation.

Outputs:

angles	vec3 of floats	Euler angle representation of rotation as [rpy] vector.

generateURDF()

This function generates a .urdf file of a continuum manipulator for use with the ContinuumManipulator class in PyBullet. Generated models consist of a number of cylindrical links, specified by numLinks, with a virtual 3DOF [xyz] joint on each end. These consist of overlayed spheres with individual x, y, and z axis revolute joints. The values for length and mass can either be interpreted as per individual link or overall manipulator length/mass. This is specified with automaticSegments = True or False. Generated URDF files will be in the same directory as the script in which this function is called.

Inputs:

required	fileName	string	Name of the generated URDF file, relative file path can be specified as part of name
required	robotName	string	Name of robot within URDF file
required	radius	float	Radius of manipulator body

required	numLinks	int	Number of links in body
required	length	float	Length of either individual links or overall robot
required	mass	float	Mass of either individual links or overall robot
optional	automaticSegments	bool	Option to use length/mass for individual links or overall manipulator, True uses per robot and False uses per link. Default value: False
optional	linkInertia	vec3 of floats	The I_{XX}, I_{YY}, and I_{ZZ} inertias of all cylindrical links. Only necessary if custom link inertias are desired. Default value: [0.0,0.0,0.0]
optional	damping	float	The joint damping factor in the URDF file. Values between 0.01 – 0.15 are recommended, determine via experimentation. Default value: 0.1