spot_mini_mini Documentation

Release 1.0.0

Maurice Rahme

Jun 30, 2020

Contents:

1 main **1**
- 1.1 spotmicro package 1

2 Simulation **21**
- 2.1 Quickstart PyBullet 21
- 2.2 Kinematics 22

3 Reinforcement Learning Environment **23**
- 3.1 Quickstart Reinforcement Learning 25

4 Indices and tables **27**

Python Module Index **29**

Index **31**

CHAPTER 1

main

1.1 spotmicro package

1.1.1 Subpackages

spotmicro.GaitGenerator package

Submodules

spotmicro.GaitGenerator.Bezier module

class spotmicro.GaitGenerator.Bezier.**BezierGait**(*dSref=[0.0, 0.0, 0.5, 0.5], dt=0.01, Tswing=0.15*)

Bases: object

BezierPoint(*t*, *k*, *point*)

Calculate the point on the bezier curve based on phase (0->1), point number (0-11), and the value of the point itself

Parameters

- **t** – phase
- **k** – point number
- **point** – point value

Returns Value through Bezier Curve

BezierSwing(*phase*, *L*, *LateralFraction*, *clearance_height=0.04*)

Calculates the step coordinates for the Bezier (swing) period

Parameters

- **phase** – current trajectory phase

- **L** – step length
- **LateralFraction** – determines how lateral the movement is
- **clearance_height** – foot clearance height during swing phase

Returns X,Y,Z Foot Coordinates relative to unmodified body

Binomial(*k*)

Solves the binomial theorem given a Bezier point number relative to the total number of Bezier points.

Parameters **k** – Bezier point number

Returns Binomial solution

CheckTouchDown()

Checks whether a reference leg touchdown has occured, and whether this warrants resetting the touchdown time

GenerateTrajectory(*L, LateralFraction, YawRate, vel, T_bf_, T_bf_curr, clearance_height=0.06, penetration_depth=0.01, contacts=[0, 0, 0, 0], dt=None*)

Calculates the step coordinates for each foot

Parameters

- **L** – step length
- **LateralFraction** – determines how lateral the movement is
- **YawRate** – the desired body yaw rate
- **vel** – the desired step velocity
- **clearance_height** – foot clearance height during swing phase
- **penetration_depth** – foot penetration depth during stance phase
- **contacts** – array containing 1 for contact and 0 otherwise
- **dt** – the time step

Returns Foot Coordinates relative to unmodified body

GetFootStep(*L*, *LateralFraction*, *YawRate*, *clearance_height*, *penetration_depth*, *Tstance*, *T_bf*, *index*, *key*)

Calculates the step coordinates in either the Bezier or Sine portion of the trajectory depending on the retrieved phase

Parameters

- **phase** – current trajectory phase
- **L** – step length
- **LateralFraction** – determines how lateral the movement is
- **YawRate** – the desired body yaw rate
- **clearance_height** – foot clearance height during swing phase
- **penetration_depth** – foot penetration depth during stance phase
- **Tstance** – the current user-specified stance period
- **T_bf** – default body-to-foot Vector
- **index** – the foot index in the container
- **key** – indicates which foot is being processed

Returns Foot Coordinates relative to unmodified body

GetPhase(*index*, *Tstance*, *Tswing*)
Retrieves the phase of an individual leg

Parameters

- **index** – the leg's index, used to identify the required phase lag
- **Tstance** – the current user-specified stance period
- **Tswing** – the swing period (constant, class member)

Returns Leg Phase, and StanceSwing (bool) to indicate whether leg is in stance or swing mode

Get_ti(*index*, *Tstride*)
Retrieves the time index for the individual leg

Parameters

- **index** – the leg's index, used to identify the required phase lag
- **Tstride** – the total leg movement period (Tstance + Tswing)

Returns the leg's time index

Increment(*dt*, *Tstride*)
Increments the Bezier gait generator's internal clock

Parameters

- **dt** – the time step phase lag
- **Tstride** – the total leg movement period (Tstance + Tswing)

Returns the leg's time index

SineStance(*phase*, *L*, *LateralFraction*, *penetration_depth=0.0*)
Calculates the step coordinates for the Bezier (swing) period

Parameters

- **phase** – current trajectory phase
- **L** – step length
- **LateralFraction** – determines how lateral the movement is
- **penetration_depth** – foot penetration depth during stance phase

Returns X,Y,Z Foot Coordinates relative to unmodified body

StanceStep(*phase*, *L*, *LateralFraction*, *YawRate*, *penetration_depth*, *T_bf*, *key*, *index*)
Calculates the step coordinates for the Sine (stance) period using a combination of forward and rotational step coordinates initially decomposed from user input of L, LateralFraction and YawRate

Parameters

- **phase** – current trajectory phase
- **L** – step length
- **LateralFraction** – determines how lateral the movement is
- **YawRate** – the desired body yaw rate
- **penetration_depth** – foot penetration depth during stance phase
- **T_bf** – default body-to-foot Vector

- **key** – indicates which foot is being processed
- **index** – the foot index in the container

Returns Foot Coordinates relative to unmodified body

SwingStep(*phase*, *L*, *LateralFraction*, *YawRate*, *clearance_height*, *T_bf*, *key*, *index*)
Calculates the step coordinates for the Bezier (swing) period using a combination of forward and rotational step coordinates initially decomposed from user input of L, LateralFraction and YawRate

Parameters

- **phase** – current trajectory phase
- **L** – step length
- **LateralFraction** – determines how lateral the movement is
- **YawRate** – the desired body yaw rate
- **clearance_height** – foot clearance height during swing phase
- **T_bf** – default body-to-foot Vector
- **key** – indicates which foot is being processed
- **index** – the foot index in the container

Returns Foot Coordinates relative to unmodified body

reset()
Resets the parameters of the Bezier Gait Generator

spotmicro.GaitGenerator.Raibert module

Module contents

spotmicro.GymEnvs package

Submodules

spotmicro.GymEnvs.spot_bezier_env module

This file implements the gym environment of SpotMicro with Bezier Curve.

class spotmicro.GymEnvs.spot_bezier_env.**spotBezierEnv**(*distance_weight=1.0*, *rotation_weight=0.0*, *energy_weight=0.0*, *shake_weight=0.0*, *drift_weight=2.0*, *rp_weight=5.0*, *rate_weight=0.05*, *urdf_root='/home/docs/checkouts/readthedocs.org/use mini-mini/envs/v1.0.0/lib/python3.7/site-packages/pybullet_data'*, *urdf_version=None*, *distance_limit=inf*, *observation_noise_stdev=(0.0, 0.0, 0.0, 0.0, 0.0)*, *self_collision_enabled=True*, *motor_velocity_limit=inf*, *pd_control_enabled=False*, *leg_model_enabled=False*, *accurate_motor_model_enabled=False*, *remove_default_joint_damping=False*, *motor_kp=2.0*, *motor_kd=0.03*, *control_latency=0.0*, *pd_latency=0.0*, *torque_control_enabled=False*, *motor_overheat_protection=False*, *hard_reset=False*, *on_rack=False*, *render=True*, *num_steps_to_log=1000*, *action_repeat=1*, *control_time_step=None*, *env_randomizer=<spotmicro.spot_env_randomizer.Sp object>*, *forward_reward_cap=inf*, *reflection=True*, *log_path=None*, *desired_velocity=0.5*, *desired_rate=0.0*, *lateral=False*, *draw_foot_path=False*, *height_field=False*, *AutoStepper=True*, *action_dim=15*)

Bases: *spotmicro.spot_gym_env.spotGymEnv*

The gym environment for spot.

It simulates the locomotion of spot, a quadruped robot. The state space include the angles, velocities and torques for all the motors and the action space is the desired motor angle for each motor. The reward function is based on how far spot walks in 1000 steps and penalizes the energy expenditure.

metadata = {'render.modes': ['human', 'rgb_array'], 'video.frames_per_second': 50}

pass_joint_angles(*ja*)

For executing joint angles

return_state()

return_yaw()

step(*action*)
Step forward the simulation, given the action.

Args: action: A list of desired motor angles for eight motors. smach: the bezier state machine containing simulated

random controll inputs

Returns: observations: The angles, velocities and torques of all motors. reward: The reward for the current state-action pair. done: Whether the episode has ended. info: A dictionary that stores diagnostic information.

Raises: ValueError: The action dimension is not the same as the number of motors. ValueError: The magnitude of actions is out of bounds.

Module contents

spotmicro.Kinematics package

Submodules

spotmicro.Kinematics.LegKinematics module

class spotmicro.Kinematics.LegKinematics.**LegIK**(*legtype='RIGHT', hip_length=0.04, shoulder_length=0.1, leg_length=0.1, hip_lim=[-0.548, 0.548], shoulder_lim=[-2.17, 0.97], leg_lim=[-0.1, 2.59]*)

Bases: object

LeftIK(*x*, *y*, *z*, *D*)
Left Leg Inverse Kinematics Solver

Parameters

- **x,y,z** – hip-to-foot distances in each dimension
- **D** – leg domain

Returns Joint Angles required for desired position

RightIK(*x*, *y*, *z*, *D*)
Right Leg Inverse Kinematics Solver

Parameters

- **x,y,z** – hip-to-foot distances in each dimension
- **D** – leg domain

Returns Joint Angles required for desired position

get_domain(*x*, *y*, *z*)
Calculates the leg's Domain and caps it in case of a breach

Parameters **x,y,z** – hip-to-foot distances in each dimension

Returns Leg Domain D

solve(*xyz_coord*)
Generic Leg Inverse Kinematics Solver

Parameters **xyz_coord** – hip-to-foot distances in each dimension

Returns Joint Angles required for desired position

spotmicro.Kinematics.LieAlgebra module

spotmicro.Kinematics.LieAlgebra.**Adjoint**(*T*)
Computes the adjoint representation of a homogeneous transformation matrix

Parameters **T** – A homogeneous transformation matrix

Returns The 6x6 adjoint representation [AdT] of T

Example Input:

T = np.array([[1, 0, 0, 0], [0, 0, -1, 0], [0, 1, 0, 3], [0, 0, 0, 1]])

Output:

np.array([[1, 0, 0, 0, 0, 0], [0, 0, -1, 0, 0, 0], [0, 1, 0, 0, 0, 0], [0, 0, 3, 1, 0, 0], [3, 0, 0, 0, 0, -1], [0, 0, 0, 0, 1, 0]])

spotmicro.Kinematics.LieAlgebra.**RPY**(*roll*, *pitch*, *yaw*)
Creates a Roll, Pitch, Yaw Transformation Matrix

Parameters

- **roll** – roll component of matrix
- **pitch** – pitch component of matrix
- **yaw** – yaw component of matrix

Returns The transformation matrix

Example Input: roll = 0.0 pitch = 0.0 yaw = 0.0

Output:

np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])

spotmicro.Kinematics.LieAlgebra.**RotateTranslate**(*rotation*, *position*)
Creates a Transformation Matrix from a Rotation, THEN, a Translation

Parameters

- **rotation** – pure rotation matrix
- **translation** – pure translation matrix

Returns The transformation matrix

spotmicro.Kinematics.LieAlgebra.**RpToTrans**(*R*, *p*)
Converts a rotation matrix and a position vector into homogeneous transformation matrix

Parameters

- **R** – A 3x3 rotation matrix

- **p** – A 3-vector

Returns A homogeneous transformation matrix corresponding to the inputs

Example Input:

R = np.array([[1, 0, 0], [0, 0, -1], [0, 1, 0]])

p = np.array([1, 2, 5])

Output:

np.array([[1, 0, 0, 1], [0, 0, -1, 2], [0, 1, 0, 5], [0, 0, 0, 1]])

spotmicro.Kinematics.LieAlgebra.**TransInv**(*T*)
Inverts a homogeneous transformation matrix

Parameters **T** – A homogeneous transformation matrix

Returns The inverse of T

Uses the structure of transformation matrices to avoid taking a matrix inverse, for efficiency.

Example input:

T = np.array([[1, 0, 0, 0], [0, 0, -1, 0], [0, 1, 0, 3], [0, 0, 0, 1]])

Output:

np.array([[1, 0, 0, 0], [0, 0, 1, -3], [0, -1, 0, 0], [0, 0, 0, 1]])

spotmicro.Kinematics.LieAlgebra.**TransToRp**(*T*)
Converts a homogeneous transformation matrix into a rotation matrix and position vector

Parameters **T** – A homogeneous transformation matrix

Return R The corresponding rotation matrix,

Return p The corresponding position vector.

Example Input:

T = np.array([[1, 0, 0, 0], [0, 0, -1, 0], [0, 1, 0, 3], [0, 0, 0, 1]])

Output:

(np.array([[1, 0, 0],

[0, 0, -1], [0, 1, 0]]),

np.array([0, 0, 3]))

spotmicro.Kinematics.LieAlgebra.**TransformVector**(*xyz_coord*, *rotation*, *translation*)
Transforms a vector by a specified Rotation THEN Translation Matrix

Parameters

- **xyz_coord** – the vector to transform
- **rotation** – pure rotation matrix
- **translation** – pure translation matrix

Returns The transformed vector

spotmicro.Kinematics.LieAlgebra.**VecToso3**(*omg*)
Converts a 3-vector to an so(3) representation

Parameters **omg** – A 3-vector

Returns The skew symmetric representation of omg

Example Input: omg = np.array([1, 2, 3])

Output:

np.array([[0, -3, 2], [3, 0, -1], [-2, 1, 0]])

spotmicro.Kinematics.SpotKinematics module

class spotmicro.Kinematics.SpotKinematics.**SpotModel**(*hip_length=0.04, shoulder_length=0.1, leg_length=0.1, hip_lim=[-0.548, 0.548], shoulder_lim=[-2.17, 0.97], leg_lim=[-0.1, 2.59]*)

Bases: object

IK(*orn*, *pos*, *T_bf*)

Converts a desired position and orientation wrt Spot's home position, with a desired body-to-foot Transform into a body-to-hip Transform, of which the translational component can be fed into the LegIK solver.

Finally, the resultant joint angles are returned from the LegIK solver for each leg.

Parameters

- **orn** – A 3x1 np.array([]) with Spot's Roll, Pitch, Yaw angles
- **pos** – A 3x1 np.array([]) with Spot's X, Y, Z coordinates
- **T_bf** – Dictionary of desired body-to-foot Transforms.

Returns Joint angles for each of Spot's joints.

Module contents

spotmicro.OpenLoopSM package

Submodules

spotmicro.OpenLoopSM.SpotOL module

Open Loop Controller for Spot Micro. Takes GUI params or uses default

class spotmicro.OpenLoopSM.SpotOL.**BezierStepper**(*pos=array([0., 0., 0.])*, *orn=array([0., 0., 0.])*, *StepLength=0.03*, *LateralFraction=0.0*, *YawRate=0.0*, *StepVelocity=0.1*, *ClearanceHeight=0.03*, *PenetrationDepth=0.005*, *episode_length=2000*, *dt=0.01*, *num_shuffles=2*, *mode=0*)

Bases: object

COMBI()

Here, we can modify all the parameters

FB()
Here, we can modulate StepLength and StepVelocity

LAT()
Here, we can modulate StepLength and LateralFraction

ROT()
Here, we can modulate StepLength and YawRate

StateMachine()
State Machined used for training robust RL on top of OL gait.

STATES:

Forward/Backward: All Default Values. Can have slow changes to StepLength(+-) and Velocity

Lateral: As above (fwd or bwd random) with added random slow changing LateralFraction param

Rotating: As above except with YawRate

Combined: ALL changeable values may change! StepLength StepVelocity LateralFraction YawRate

NOTE: the RL is solely responsible for modulating Clearance Height and Penetration Depth

max_time = None
States 1: FWD/BWD 2: Lat 3: Rot 4: Combined

ramp_up()

reshuffle()

return_bezier_params()

which_state()

Module contents

spotmicro.util package

Subpackages

spotmicro.util.pybullet_data package

Module contents

spotmicro.util.pybullet_data.**getDataPath**()

Submodules

spotmicro.util.action_mapper module

spotmicro.util.bullet_client module

A wrapper for pybullet to manage different clients.

class spotmicro.util.bullet_client.**BulletClient**(*connection_mode=None*)
Bases: object

A wrapper for pybullet to manage different clients.

spotmicro.util.gui module

class spotmicro.util.gui.**GUI**(*quadruped*)
Bases: object

UserInput()

Module contents

1.1.2 Submodules

1.1.3 spotmicro.env_randomizer_base module

Abstract base class for environment randomizer.

class spotmicro.env_randomizer_base.**EnvRandomizerBase**
Bases: object

Abstract base class for environment randomizer.

An EnvRandomizer is called in environment.reset(). It will randomize physical parameters of the objects in the simulation. The physical parameters will be fixed for that episode and be randomized again in the next environment.reset().

randomize_env(*env*)
Randomize the simulated_objects in the environment.

Args: env: The environment to be randomized.

1.1.4 spotmicro.heightfield module

class spotmicro.heightfield.**HeightField**
Bases: object

UpdateHeightField(*heightPerturbationRange=0.08*)

1.1.5 spotmicro.motor module

This file implements an accurate motor model.

class spotmicro.motor.**MotorModel**(*torque_control_enabled=False*, *kp=1.2*, *kd=0*)
Bases: object

The accurate motor model, which is based on the physics of DC motors.

The motor model support two types of control: position control and torque control. In position control mode, a desired motor angle is specified, and a torque is computed based on the internal motor model. When the torque control is specified, a pwm signal in the range of [-1.0, 1.0] is converted to the torque.

The internal motor model takes the following factors into consideration: pd gains, viscous friction, back-EMF voltage and current-torque profile.

convert_to_torque(*motor_commands*, *current_motor_angle*, *current_motor_velocity*)
Convert the commands (position control or torque control) to torque.

Args:

motor_commands: The desired motor angle if the motor is in position control mode. The pwm signal if the motor is in torque control mode.

current_motor_angle: The motor angle at the current time step. current_motor_velocity: The motor velocity at the current time step.

Returns: actual_torque: The torque that needs to be applied to the motor. observed_torque: The torque observed by the sensor.

get_viscous_dampling()

get_voltage()

set_viscous_damping(*viscous_damping*)

set_voltage(*voltage*)

1.1.6 spotmicro.spot module

This file models a spot using pybullet.

spotmicro.spot.**MapToMinusPiToPi**(*angles*)
Maps a list of angles to [-pi, pi].

Args: angles: A list of angles in rad.

Returns: A list of angle mapped to [-pi, pi].

class spotmicro.spot.**Spot**(*pybullet_client*, *urdf_root='/home/docs/checkouts/readthedocs.org/user_builds/spot-mini-mini/checkouts/v1.0.0/spotmicro/util/pybullet_data'*, *time_step=0.01*, *action_repeat=1*, *self_collision_enabled=False*, *motor_velocity_limit=9.7*, *pd_control_enabled=False*, *accurate_motor_model_enabled=False*, *remove_default_joint_damping=False*, *max_force=100.0*, *motor_kp=1.0*, *motor_kd=0.02*, *pd_latency=0.0*, *control_latency=0.0*, *observation_noise_stdev=(0.0, 0.0, 0.0, 0.0, 0.0)*, *torque_control_enabled=False*, *motor_overheat_protection=False*, *on_rack=False*, *kd_for_pd_controllers=0.3*, *pose_id='stand'*, *np_random=<module 'numpy.random' from '/home/docs/checkouts/readthedocs.org/user_builds/spot-mini-mini/envs/v1.0.0/lib/python3.7/site-packages/numpy/random/__init__.py'>*)
Bases: object

The spot class that simulates a quadruped robot.

ApplyAction(*motor_commands*)
Set the desired motor angles to the motors of the minitaur. The desired motor angles are clipped based on the maximum allowed velocity. If the pd_control_enabled is True, a torque is calculated according to the difference between current and desired joint angle, as well as the joint velocity. This torque is exerted to the motor. For more information about PD control, please refer to: https://en.wikipedia.org/wiki/PID_controller. Args:

motor_commands: The eight desired motor angles.

ApplyMotorLimits(*joint_angles*)

ConvertFromLegModel(*action*)

GetActionDimension()
Get the length of the action list.

Returns: The length of the action list.

GetBaseInertiasFromURDF()
Get the inertia of the base from the URDF file.

GetBaseMassFromURDF()
Get the mass of the base from the URDF file.

GetBaseMassesFromURDF()
Get the mass of the base from the URDF file.

GetBaseOrientation()
Get the orientation of spot's base, represented as quaternion.

Returns: The orientation of spot's base.

GetBasePosition()
Get the position of spot's base.

Returns: The position of spot's base.

GetBaseRollPitchYaw()
Get the rate of orientation change of the spot's base in euler angle.

Returns: rate of (roll, pitch, yaw) change of the spot's base.

GetBaseRollPitchYawRate()
Get the rate of orientation change of the spot's base in euler angle.

This function mimicks the noisy sensor reading and adds latency. Returns:

rate of (roll, pitch, yaw) change of the spot's base polluted by noise and latency.

GetBaseTwist()
Get the Twist of minitaur's base. Returns:

The Twist of the minitaur's base.

GetControlInput(*controller*)
Store Control Input as Observation

GetControlLatency()
Get the control latency.

Returns:

The latency (in seconds) between when the motor command is sent and when the sensor measurements are reported back to the controller.

GetExternalObservations(*TrajectoryGenerator*, *controller*)
Augment State Space

GetLegInertiasFromURDF()
Get the inertia of the legs from the URDF file.

GetLegMassesFromURDF()
Get the mass of the legs from the URDF file.

GetLegPhases(*TrajectoryGenerator*)
Leg phases according to TG from 0->2 0->1: Stance 1->2 Swing

GetMotorAngles()
Gets the eight motor angles at the current moment, mapped to [-pi, pi].

Returns: Motor angles, mapped to [-pi, pi].

GetMotorGains()
Get the gains of the motor.

Returns: The proportional gain. The derivative gain.

GetMotorTorques()
Get the amount of torque the motors are exerting.

Returns: Motor torques of all eight motors.

GetMotorVelocities()
Get the velocity of all eight motors.

Returns: Velocities of all eight motors.

GetNumKneeJoints()

GetObservation()
Get the observations of minitaur. It includes the angles, velocities, torques and the orientation of the base. Returns:

The observation list. observation[0:8] are motor angles. observation[8:16] are motor velocities, observation[16:24] are motor torques. observation[24:28] is the orientation of the base, in quaternion form. NOTE: DIVERGES FROM STOCK MINITAUR ENV. WILL LEAVE ORIGINAL COMMENTED For my purpose, the observation space includes Roll and Pitch, as well as acceleration and gyroscopic rate along the x,y,z axes. All of this information can be collected from an onboard IMU. The reward function will contain a hidden velocity reward (fwd, bwd) which cannot be measured and so is not included. For spinning, the gyroscopic z rate will be used as the (explicit) velocity reward. This version operates without motor torques, angles and velocities. Erwin Coumans' paper suggests a sparse observation space leads to higher reward

NOTE: use True version for perfect data, or other for realistic data

GetObservationDimension()
Get the length of the observation list. Returns:

The length of the observation list.

GetObservationLowerBound()
Get the lower bound of the observation.

GetObservationUpperBound()
Get the upper bound of the observation. Returns:

The upper bound of an observation. See GetObservation() for the details of each element of an observation.

NOTE: Changed just like GetObservation()

GetTimeSinceReset()

INIT_POSES = {'liedown': array([-0.4, -1.5, 6. , 0.4, -1.5, 6. , -0.4, -1.5, 6. , 0.4

RealisticObservation()
Receive the observation from sensors.

This function is called once per step. The observations are only updated when this function is called.

Reset(*reload_urdf=True*, *default_motor_angles=None*, *reset_time=3.0*)
Reset the spot to its initial states.

Args:

reload_urdf: Whether to reload the urdf file. If not, Reset() just place the spot back to its starting position.

default_motor_angles: The default motor angles. If it is None, spot will hold a default pose for 100 steps. In torque control mode, the phase of holding the default pose is skipped.

reset_time: The duration (in seconds) to hold the default motor angles. If reset_time <= 0 or in torque control mode, the phase of holding the default pose is skipped.

ResetPose(*add_constraint*)
Reset the pose of the spot.

Args: add_constraint: Whether to add a constraint at the joints of two feet.

SetBaseInertias(*base_inertias*)
Set the inertias of spot's base. Args:

base_inertias: A list of inertias of each body link in CHASIS_LINK_IDS. The length of this list should be the same as the length of CHASIS_LINK_IDS.

Raises:

ValueError: It is raised when the length of base_inertias is not the same as the length of self._chassis_link_ids and base_inertias contains negative values.

SetBaseMass(*base_mass*)

SetBaseMasses(*base_mass*)
Set the mass of spot's base.

Args:

base_mass: A list of masses of each body link in CHASIS_LINK_IDS. The length of this list should be the same as the length of CHASIS_LINK_IDS.

Raises:

ValueError: It is raised when the length of base_mass is not the same as the length of self._chassis_link_ids.

SetBatteryVoltage(*voltage*)

SetControlLatency(*latency*)
Set the latency of the control loop.

It measures the duration between sending an action from Nvidia TX2 and receiving the observation from microcontroller.

Args: latency: The latency (in seconds) of the control loop.

SetFootFriction(*foot_friction=100.0*)
Set the lateral friction of the feet.

Args:

foot_friction: The lateral friction coefficient of the foot. This value is shared by all four feet.

SetFootRestitution(*link_id*, *foot_restitution=1.0*)
Set the coefficient of restitution at the feet.

Args:

foot_restitution: The coefficient of restitution (bounciness) of the feet. This value is shared by all four feet.

SetJointFriction (*joint_frictions*)

SetLegInertias (*leg_inertias*)
Set the inertias of the legs.

Args: leg_inertias: The leg and motor inertias for all the leg links and motors.

Raises: ValueError: It is raised when the length of inertias is not equal to the number of links + motors or leg_inertias contains negative values.

SetLegMasses (*leg_masses*)
Set the mass of the legs. Args:

leg_masses: The leg and motor masses for all the leg links and motors.

Raises:

ValueError: It is raised when the length of masses is not equal to number of links + motors.

SetMotorGains (*kp*, *kd*)
Set the gains of all motors.

These gains are PD gains for motor positional control. kp is the proportional gain and kd is the derivative gain.

Args: kp: proportional gain of the motors. kd: derivative gain of the motors.

SetMotorStrengthRatio (*ratio*)
Set the strength of all motors relative to the default value.

Args: ratio: The relative strength. A scalar range from 0.0 to 1.0.

SetMotorStrengthRatios (*ratios*)
Set the strength of each motor relative to the default value.

Args: ratios: The relative strength. A numpy array ranging from 0.0 to 1.0.

SetMotorViscousDamping (*viscous_damping*)

SetTimeSteps (*action_repeat*, *simulation_step*)
Set the time steps of the control and simulation.

Args:

action_repeat: The number of simulation steps that the same action is repeated.

simulation_step: The simulation time step.

Step (*action*)

chassis_link_ids

1.1.7 spotmicro.spot_env_randomizer module

Randomize the spot_gym_env when reset() is called.

class spotmicro.spot_env_randomizer.**SpotEnvRandomizer**(*spot_base_mass_err_range=(-0.2, 0.2)*, *spot_leg_mass_err_range=(-0.2, 0.2)*, *battery_voltage_range=(14.8, 16.8)*, *motor_viscous_damping_range=(0, 0.01)*)

Bases: *spotmicro.env_randomizer_base.EnvRandomizerBase*

A randomizer that change the spot_gym_env during every reset.

randomize_env(*env*)

Randomize the simulated_objects in the environment.

Args: env: The environment to be randomized.

1.1.8 spotmicro.spot_gym_env module

This file implements the gym environment of SpotMicro.

spotmicro.spot_gym_env.**convert_to_list**(*obj*)

class spotmicro.spot_gym_env.**spotGymEnv**(*distance_weight=1.0*, *rotation_weight=1.0*, *energy_weight=0.0005*, *shake_weight=0.005*, *drift_weight=2.0*, *rp_weight=0.1*, *rate_weight=0.1*, *urdf_root='/home/docs/checkouts/readthedocs.org/user_builds/spot-mini-mini/envs/v1.0.0/lib/python3.7/site-packages/pybullet_data'*, *urdf_version=None*, *distance_limit=inf*, *observation_noise_stdev=(0.0, 0.0, 0.0, 0.0, 0.0)*, *self_collision_enabled=True*, *motor_velocity_limit=inf*, *pd_control_enabled=False*, *leg_model_enabled=False*, *accurate_motor_model_enabled=False*, *remove_default_joint_damping=False*, *motor_kp=2.0*, *motor_kd=0.03*, *control_latency=0.0*, *pd_latency=0.0*, *torque_control_enabled=False*, *motor_overheat_protection=False*, *hard_reset=False*, *on_rack=False*, *render=True*, *num_steps_to_log=1000*, *action_repeat=1*, *control_time_step=None*, *env_randomizer=<spotmicro.spot_env_randomizer.SpotEnvRandomizer object>*, *forward_reward_cap=inf*, *reflection=True*, *log_path=None*, *desired_velocity=0.5*, *desired_rate=0.0*, *lateral=False*, *draw_foot_path=False*, *height_field=False*, *AutoStepper=False*)

Bases: gym.core.Env

The gym environment for spot.

It simulates the locomotion of spot, a quadruped robot. The state space include the angles, velocities and torques for all the motors and the action space is the desired motor angle for each motor. The reward function is based on how far spot walks in 1000 steps and penalizes the energy expenditure.

DrawFootPath()

configure(*args*)

env_step_counter

get_objectives()

get_spot_base_orientation()
Get the spot's base orientation, represented by a quaternion.

Returns: A numpy array of spot's orientation.

get_spot_motor_angles()
Get the spot's motor angles.

Returns: A numpy array of motor angles.

get_spot_motor_torques()
Get the spot's motor torques.

Returns: A numpy array of motor torques.

get_spot_motor_velocities()
Get the spot's motor velocities.

Returns: A numpy array of motor velocities.

ground_id

is_fallen()
Decide whether spot has fallen.

If the up directions between the base and the world is larger (the dot product is smaller than 0.85) or the base is very low on the ground (the height is smaller than 0.13 meter), spot is considered fallen.

Returns: Boolean value that indicates whether spot has fallen.

metadata = {'render.modes': ['human', 'rgb_array'], 'video.frames_per_second': 50}

objective_weights
Accessor for the weights for all the objectives.

Returns: List of floating points that corresponds to weights for the objectives in the order that objectives are stored.

pybullet_client

render(*mode='rgb_array'*, *close=False*)
Renders the environment.

The set of supported modes varies per environment. (And some environments do not support rendering at all.) By convention, if mode is:

- human: render to the current display or terminal and return nothing. Usually for human consumption.
- rgb_array: Return an numpy.ndarray with shape (x, y, 3), representing RGB values for an x-by-y pixel image, suitable for turning into a video.
- ansi: Return a string (str) or StringIO.StringIO containing a terminal-style text representation. The text can include newlines and ANSI escape sequences (e.g. for colors).

Note:

Make sure that your class's metadata 'render.modes' key includes the list of supported modes. It's recommended to call super() in implementations to use the functionality of this method.

Args: mode (str): the mode to render with

Example:

class MyEnv(Env): metadata = {'render.modes': ['human', 'rgb_array']}

def render(self, mode='human'):

if mode == 'rgb_array': return np.array(...) # return RGB frame suitable for video

elif mode == 'human': ... # pop up a window and render

else: super(MyEnv, self).render(mode=mode) # just raise an exception

reset(*initial_motor_angles=None*, *reset_duration=1.0*, *desired_velocity=None*, *desired_rate=None*)
Resets the state of the environment and returns an initial observation.

Returns: observation (object): the initial observation.

seed(*seed=None*)
Sets the seed for this env's random number generator(s).

Note: Some environments use multiple pseudorandom number generators. We want to capture all such seeds used in order to ensure that there aren't accidental correlations between multiple generators.

Returns:

list<bigint>: Returns the list of seeds used in this env's random number generators. The first value in the list should be the "main" seed, or the value which a reproducer should pass to 'seed'. Often, the main seed equals the provided 'seed', but this won't be true if seed=None, for example.

set_env_randomizer(*env_randomizer*)

set_time_step(*control_step*, *simulation_step=0.001*)
Sets the time step of the environment.

Args:

control_step: The time period (in seconds) between two adjacent control actions are applied.

simulation_step: The simulation time step in PyBullet. By default, the simulation step is 0.001s, which is a good trade-off between simulation speed and accuracy.

Raises: ValueError: If the control step is smaller than the simulation step.

step(*action*)
Step forward the simulation, given the action.

Args: action: A list of desired motor angles for eight motors.

Returns: observations: The angles, velocities and torques of all motors. reward: The reward for the current state-action pair. done: Whether the episode has ended. info: A dictionary that stores diagnostic information.

Raises: ValueError: The action dimension is not the same as the number of motors. ValueError: The magnitude of actions is out of bounds.

1.1.9 Module contents

CHAPTER 2

Simulation

This section contains information on the most up-to-date simulation solutions.

2.1 Quickstart PyBullet

I have deployed a 12-point Bezier Curve gait to make Spot walk:

PyBullet

This example can be found in this repository. You can optionally use a Game Pad:

```
pip3 install numpy
pip3 install pybullet
pip3 install gym

cd spot_bullet/src

./env_tester.py
```

Optional Arguments

```
-h, --help          show this help message and exit
-hf, --HeightField  Use HeightField
-r, --DebugRack     Put Spot on an Elevated Rack
-p, --DebugPath     Draw Spot's Foot Path
-ay, --AutoYaw      Automatically Adjust Spot's Yaw
```

If you decide to use a controller, you can achieve some fairly fluid motions!

Changing Step Length:

PyBullet

Yaw in Place:

PyBullet

2.2 Kinematics

In this repository, there is a working IK solver for both Spot's legs and its body:

IK

CHAPTER 3

Reinforcement Learning Environment

We now have a Reinforcement Learning Environment which uses Pybullet and OpenAI Gym! It contains a variety of optional terrains, which can be activated using **heightfield=True** in the environment class constructor.

If you try to launch the vanilla gait on fairly difficult terrain, Spot will fall very quickly:

PyBullet

By training an Augmented Random Search agent, this can be overcome:

PyBullet

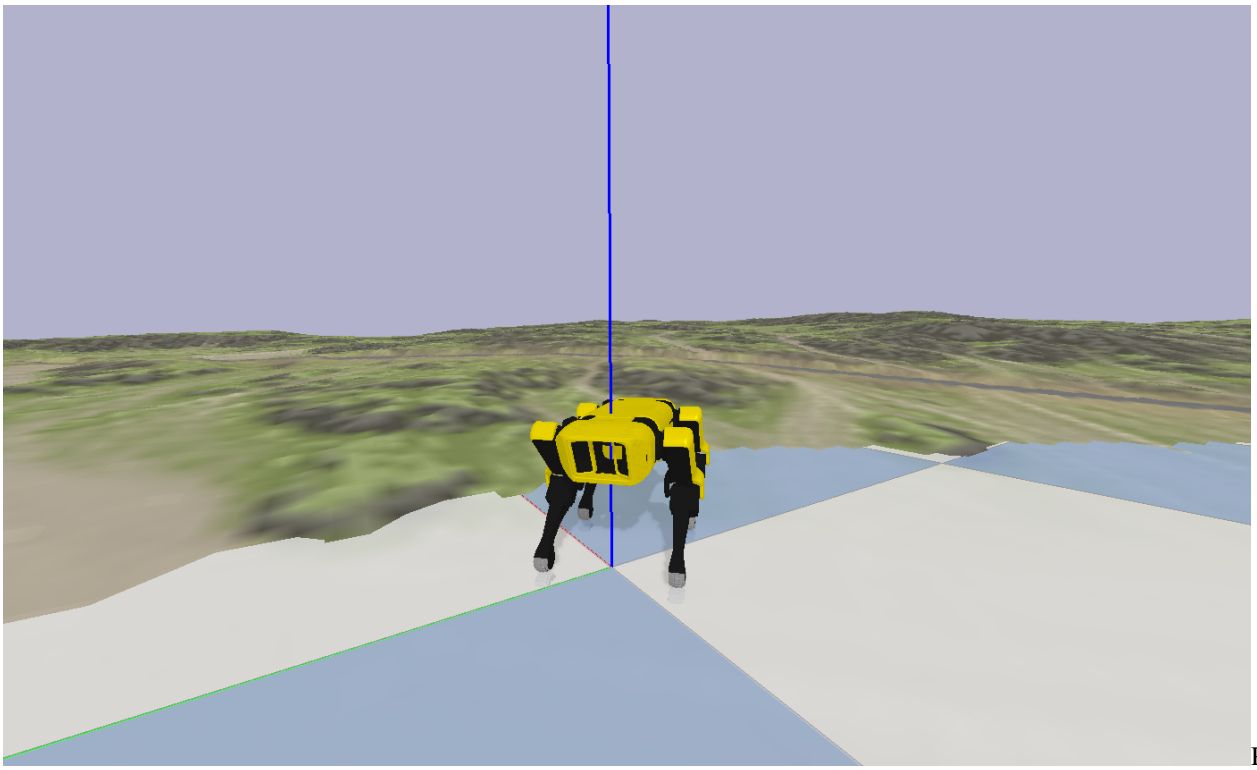
If you are new to RL, I recommend you try a simpler example. Notice that if we choose non-ideal parameters for the generated gait, the robot drifts over time with a forward command:

PyBullet

You should try to train a policy which outputs a yaw command to eliminate the robot's drift, like this:

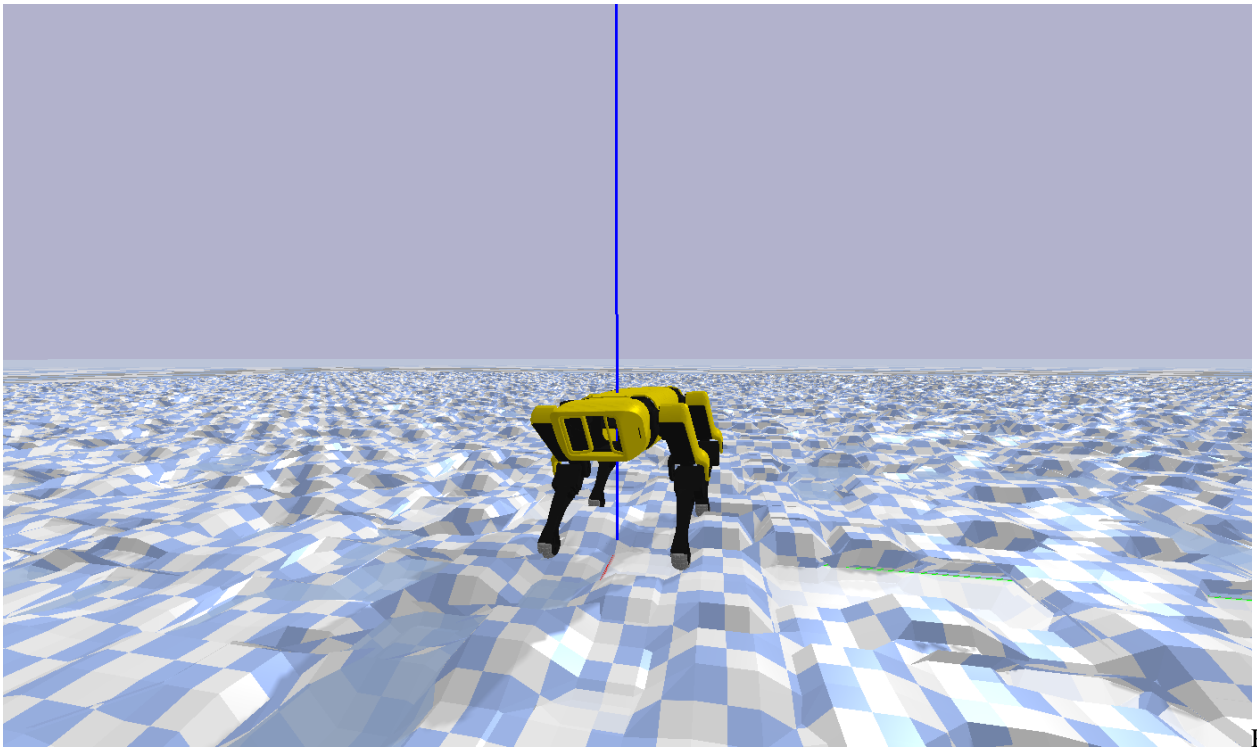
PyBullet

You can choose a PNG-generated terrain:



PyBullet

Or, for more control, you can choose a programmatically generated heightfield:



PyBullet

Notice that when the simulation resets, the terrain changes. What you cannot see is that the robot's link masses and frictions also change under the hood for added training robustness:

PyBullet

3.1 Quickstart Reinforcement Learning

```
pip3 install numpy
pip3 install pybullet
pip3 install gym

cd spot_bullet/src

./spot_ars_eval.py
```

Optional Arguments

```
-h, --help           show this help message and exit
-hf, --HeightField   Use HeightField
-r, --DebugRack      Put Spot on an Elevated Rack
-p, --DebugPath      Draw Spot's Foot Path
-a, --AgentNum       Agent Number To Load
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

spotmicro, 19
spotmicro.env_randomizer_base, 11
spotmicro.GaitGenerator, 4
spotmicro.GaitGenerator.Bezier, 1
spotmicro.GaitGenerator.Raibert, 4
spotmicro.GymEnvs, 6
spotmicro.GymEnvs.spot_bezier_env, 4
spotmicro.heightfield, 11
spotmicro.Kinematics, 9
spotmicro.Kinematics.LegKinematics, 6
spotmicro.Kinematics.LieAlgebra, 7
spotmicro.Kinematics.SpotKinematics, 9
spotmicro.motor, 11
spotmicro.OpenLoopSM, 10
spotmicro.OpenLoopSM.SpotOL, 9
spotmicro.spot, 12
spotmicro.spot_env_randomizer, 16
spotmicro.spot_gym_env, 17
spotmicro.util, 11
spotmicro.util.action_mapper, 10
spotmicro.util.bullet_client, 10
spotmicro.util.gui, 11
spotmicro.util.pybullet_data, 10

Index

A

Adjoint() (*in module spotmicro.Kinematics.LieAlgebra*), 7
ApplyAction() (*spotmicro.spot.Spot method*), 12
ApplyMotorLimits() (*spotmicro.spot.Spot method*), 12

B

BezierGait (*class in spotmicro.GaitGenerator.Bezier*), 1
BezierPoint() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 1
BezierStepper (*class in spotmicro.OpenLoopSM.SpotOL*), 9
BezierSwing() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 1
Binomial() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 2
BulletClient (*class in spotmicro.util.bullet_client*), 10

C

chassis_link_ids (*spotmicro.spot.Spot attribute*), 16
CheckTouchDown() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 2
COMBI() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 9
configure() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
convert_to_list() (*in module spotmicro.spot_gym_env*), 17
convert_to_torque() (*spotmicro.motor.MotorModel method*), 11
ConvertFromLegModel() (*spotmicro.spot.Spot method*), 12

D

DrawFootPath() (*spotmicro.spot_gym_env.spotGymEnv method*), 17

E

env_step_counter (*spotmicro.spot_gym_env.spotGymEnv attribute*), 18
EnvRandomizerBase (*class in spotmicro.env_randomizer_base*), 11

F

FB() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 9

G

GenerateTrajectory() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 2
get_domain() (*spotmicro.Kinematics.LegKinematics.LegIK method*), 6
get_objectives() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
get_spot_base_orientation() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
get_spot_motor_angles() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
get_spot_motor_torques() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
get_spot_motor_velocities() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
Get_ti() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 3

get_viscous_dampling() (*spotmicro.motor.MotorModel method*), 12
get_voltage() (*spotmicro.motor.MotorModel method*), 12
GetActionDimension() (*spotmicro.spot.Spot method*), 13
GetBaseInertiasFromURDF() (*spotmicro.spot.Spot method*), 13
GetBaseMassesFromURDF() (*spotmicro.spot.Spot method*), 13
GetBaseMassFromURDF() (*spotmicro.spot.Spot method*), 13
GetBaseOrientation() (*spotmicro.spot.Spot method*), 13
GetBasePosition() (*spotmicro.spot.Spot method*), 13
GetBaseRollPitchYaw() (*spotmicro.spot.Spot method*), 13
GetBaseRollPitchYawRate() (*spotmicro.spot.Spot method*), 13
GetBaseTwist() (*spotmicro.spot.Spot method*), 13
GetControlInput() (*spotmicro.spot.Spot method*), 13
GetControlLatency() (*spotmicro.spot.Spot method*), 13
getDataPath() (*in module spotmicro.util.pybullet_data*), 10
GetExternalObservations() (*spotmicro.spot.Spot method*), 13
GetFootStep() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 2
GetLegInertiasFromURDF() (*spotmicro.spot.Spot method*), 13
GetLegMassesFromURDF() (*spotmicro.spot.Spot method*), 13
GetLegPhases() (*spotmicro.spot.Spot method*), 13
GetMotorAngles() (*spotmicro.spot.Spot method*), 13
GetMotorGains() (*spotmicro.spot.Spot method*), 14
GetMotorTorques() (*spotmicro.spot.Spot method*), 14
GetMotorVelocities() (*spotmicro.spot.Spot method*), 14
GetNumKneeJoints() (*spotmicro.spot.Spot method*), 14
GetObservation() (*spotmicro.spot.Spot method*), 14
GetObservationDimension() (*spotmicro.spot.Spot method*), 14
GetObservationLowerBound() (*spotmicro.spot.Spot method*), 14
GetObservationUpperBound() (*spotmicro.spot.Spot method*), 14
GetPhase() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 3
GetTimeSinceReset() (*spotmicro.spot.Spot method*), 14
ground_id (*spotmicro.spot_gym_env.spotGymEnv attribute*), 18
GUI (*class in spotmicro.util.gui*), 11

H

HeightField (*class in spotmicro.heightfield*), 11

I

IK() (*spotmicro.Kinematics.SpotKinematics.SpotModel method*), 9
Increment() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 3
INIT_POSES (*spotmicro.spot.Spot attribute*), 14
is_fallen() (*spotmicro.spot_gym_env.spotGymEnv method*), 18

L

LAT() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10
LeftIK() (*spotmicro.Kinematics.LegKinematics.LegIK method*), 6
LegIK (*class in spotmicro.Kinematics.LegKinematics*), 6

M

MapToMinusPiToPi() (*in module spotmicro.spot*), 12
max_time (*spotmicro.OpenLoopSM.SpotOL.BezierStepper attribute*), 10
metadata (*spotmicro.GymEnvs.spot_bezier_env.spotBezierEnv attribute*), 5
metadata (*spotmicro.spot_gym_env.spotGymEnv attribute*), 18
MotorModel (*class in spotmicro.motor*), 11

O

objective_weights (*spotmicro.spot_gym_env.spotGymEnv attribute*), 18

P

pass_joint_angles() (*spotmicro.GymEnvs.spot_bezier_env.spotBezierEnv method*), 5
pybullet_client (*spotmicro.spot_gym_env.spotGymEnv attribute*), 18

R

ramp_up() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10

randomize_env() (*spotmicro.env_randomizer_base.EnvRandomizerBase method*), 11
randomize_env() (*spotmicro.spot_env_randomizer.SpotEnvRandomizer method*), 17
RealisticObservation() (*spotmicro.spot.Spot method*), 14
render() (*spotmicro.spot_gym_env.spotGymEnv method*), 18
reset() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 4
Reset() (*spotmicro.spot.Spot method*), 14
reset() (*spotmicro.spot_gym_env.spotGymEnv method*), 19
ResetPose() (*spotmicro.spot.Spot method*), 15
reshuffle() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10
return_bezier_params() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10
return_state() (*spotmicro.GymEnvs.spot_bezier_env.spotBezierEnv method*), 6
return_yaw() (*spotmicro.GymEnvs.spot_bezier_env.spotBezierEnv method*), 6
RightIK() (*spotmicro.Kinematics.LegKinematics.LegIK method*), 6
ROT() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10
RotateTranslate() (*in module spotmicro.Kinematics.LieAlgebra*), 7
RpToTrans() (*in module spotmicro.Kinematics.LieAlgebra*), 7
RPY() (*in module spotmicro.Kinematics.LieAlgebra*), 7

S

seed() (*spotmicro.spot_gym_env.spotGymEnv method*), 19
set_env_randomizer() (*spotmicro.spot_gym_env.spotGymEnv method*), 19
set_time_step() (*spotmicro.spot_gym_env.spotGymEnv method*), 19
set_viscous_damping() (*spotmicro.motor.MotorModel method*), 12
set_voltage() (*spotmicro.motor.MotorModel method*), 12
SetBaseInertias() (*spotmicro.spot.Spot method*), 15
SetBaseMass() (*spotmicro.spot.Spot method*), 15
SetBaseMasses() (*spotmicro.spot.Spot method*), 15
SetBatteryVoltage() (*spotmicro.spot.Spot method*), 15
SetControlLatency() (*spotmicro.spot.Spot method*), 15
SetFootFriction() (*spotmicro.spot.Spot method*), 15
SetFootRestitution() (*spotmicro.spot.Spot method*), 15
SetJointFriction() (*spotmicro.spot.Spot method*), 16
SetLegInertias() (*spotmicro.spot.Spot method*), 16
SetLegMasses() (*spotmicro.spot.Spot method*), 16
SetMotorGains() (*spotmicro.spot.Spot method*), 16
SetMotorStrengthRatio() (*spotmicro.spot.Spot method*), 16
SetMotorStrengthRatios() (*spotmicro.spot.Spot method*), 16
SetMotorViscousDamping() (*spotmicro.spot.Spot method*), 16
SetTimeSteps() (*spotmicro.spot.Spot method*), 16
SineStance() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 3
solve() (*spotmicro.Kinematics.LegKinematics.LegIK method*), 7
Spot (*class in spotmicro.spot*), 12
spotBezierEnv (*class in spotmicro.GymEnvs.spot_bezier_env*), 4
SpotEnvRandomizer (*class in spotmicro.spot_env_randomizer*), 16
spotGymEnv (*class in spotmicro.spot_gym_env*), 17
spotmicro (*module*), 19
spotmicro.env_randomizer_base (*module*), 11
spotmicro.GaitGenerator (*module*), 4
spotmicro.GaitGenerator.Bezier (*module*), 1
spotmicro.GaitGenerator.Raibert (*module*), 4
spotmicro.GymEnvs (*module*), 6
spotmicro.GymEnvs.spot_bezier_env (*module*), 4
spotmicro.heightfield (*module*), 11
spotmicro.Kinematics (*module*), 9
spotmicro.Kinematics.LegKinematics (*module*), 6
spotmicro.Kinematics.LieAlgebra (*module*), 7
spotmicro.Kinematics.SpotKinematics (*module*), 9
spotmicro.motor (*module*), 11
spotmicro.OpenLoopSM (*module*), 10
spotmicro.OpenLoopSM.SpotOL (*module*), 9
spotmicro.spot (*module*), 12
spotmicro.spot_env_randomizer (*module*), 16

spotmicro.spot_gym_env (*module*), 17
spotmicro.util (*module*), 11
spotmicro.util.action_mapper (*module*), 10
spotmicro.util.bullet_client (*module*), 10
spotmicro.util.gui (*module*), 11
spotmicro.util.pybullet_data (*module*), 10
SpotModel (*class in spotmicro.Kinematics.SpotKinematics*), 9
StanceStep() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 3
StateMachine() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10
step() (*spotmicro.GymEnvs.spot_bezier_env.spotBezierEnv method*), 6
Step() (*spotmicro.spot.Spot method*), 16
step() (*spotmicro.spot_gym_env.spotGymEnv method*), 19
SwingStep() (*spotmicro.GaitGenerator.Bezier.BezierGait method*), 4

T

TransformVector() (*in module spotmicro.Kinematics.LieAlgebra*), 8
TransInv() (*in module spotmicro.Kinematics.LieAlgebra*), 8
TransToRp() (*in module spotmicro.Kinematics.LieAlgebra*), 8

U

UpdateHeightField() (*spotmicro.heightfield.HeightField method*), 11
UserInput() (*spotmicro.util.gui.GUI method*), 11

V

VecToso3() (*in module spotmicro.Kinematics.LieAlgebra*), 8

W

which_state() (*spotmicro.OpenLoopSM.SpotOL.BezierStepper method*), 10