**XRhex Class: Properties**

|  |  |
| --- | --- |
| **Name**:  maxError | **Purpose**:  Stores the maximum allowed error for the position of the modules |
| **Type of Variable:**  Float | **Units:**  Radians |
| **Used For:**   * Determining if position error is within an acceptable range * Functions: checkPosError, checkPosAgainstGiven   + Within: moveLegsToPos, biDirMoveLegsToPos, forwardLeap, upwardLeap | |

|  |  |
| --- | --- |
| **Name**:  group | **Purpose**:  Stores which modules are in the robot, and their positions on the robot |
| **Type of Variable:** | **Units:** |
| **Used For:**   * Used to send commands to the robot and get feedback from the modules * Functions: almost all gait and trajectory functions | |

|  |  |
| --- | --- |
| **Name**:  fbk | **Purpose**:  Stores the absolute angular position of all the modules |
| **Type of Variable:**  1X6 Array | **Units:**  Radians |
| **Used For:**   * Getting the current position of all of the modules. * Functions: almost all gait and trajectory functions | |

|  |  |
| --- | --- |
| **Name**:  directionFlip | **Purpose**:  Reverses the angular position of the modules that are attached backwards (as on the right side of the robot) |
| **Type of Variable:**  1X6 Array | **Units:**  None |
| **Used For:**   * Always has value of [-1 -1 -1 1 1 1] * Functions: almost all gait and trajectory functions | |

**XRhex Class: Methods**

|  |  |
| --- | --- |
| **Name**:  robot = XRhex(group)  **CONSTRUCTOR** | **Purpose**:  Creates a new robot object |
| **Input**:  None | **Output**:  XRhex robot object |
| **Notes**:   * robot.group = group: the group of modules given are the new robot object’s modules * robot.fbk = robot.group.getNextFeedback():the robot’s current position vector is the current position of the modules | |

|  |  |
| --- | --- |
| **Name**:  checkPosError(robot, fbk) | **Purpose**:  Verifies that the position error is within an acceptable tolerance of the commanded position |
| **Input**:  Fbk: robot’s current position | **Output**:  Boolean: true if error is less than error tolerance, false if error is greater than error tolerance |
| **Notes**: | |

|  |  |
| --- | --- |
| **Name**:  checkPosAgainstGiven(robot, fbk, given) | **Purpose**:  Verifies that the position error is within an acceptable tolerance of the given position vector |
| **Input**:  Fbk: robot’s current position  Given: 1X6 vector of angles for each leg | **Output**:  Boolean: true if error is less than error tolerance, false if error is greater than error tolerance |
| **Notes**: | |

|  |  |
| --- | --- |
| **Name**:  moveLegsToPos(robot, pos) | **Purpose**:  Moves legs forward to given position with no constraints on time or torque, but verifying position error |
| **Input**:  Position vector: 1X6 vector of angle for each leg | **Output**:  None |
| **Notes**:   * When this function is used repeatedly, there is a long delay between each action. * Helpful for debugging gaits | |

|  |  |
| --- | --- |
| **Name**:  biDirMoveLegsToPos(robot, pos) | **Purpose**:  Moves the legs backward or forward, whichever is closer, to a given set of positions with no time or torque, but verifying position error |
| **Input**:  Position vector: 1X6 vector of angle for each leg | **Output**:  None |
| **Notes**:   * When this function is used repeatedly, there is a long delay between each action. * Does not incorporate as well into trajectories (less helpful for debugging gaits) | |

|  |  |
| --- | --- |
| **Name**:  generateLegTraj(robot, waypoints, times, speeds) | **Purpose**:  Generates a smooth trajectory of positions for the legs to pass through when given waypoints, at desired times, with desired speeds |
| **Input**:   * Waypoints; 1XN matrix where each entry is a 6X1 matrix of positions for each leg * Times: Time to get from each waypoint * Speeds: Speed to move each leg to reach each waypoint | **Output**:  Traj  Trajectory for the robot’s legs to follow in followLegTraj |
| **Notes**:   * Output of this function is passed to followLegTraj * Used for smooth gait creation * Uses createTraj5 to create the actual gait | |

|  |  |
| --- | --- |
| **Name**:  followLegTraj(robot, trajPoints, startPnt,endPnt) | **Purpose**:  Makes the legs of the robot follow a trajectory given. |
| **Input**:   * Trajectory * Start Point and End Point | **Output**:  None |
| **Notes**:   * Used after generateLegTraj at all times * Used for smooth gait creation | |

|  |  |
| --- | --- |
| **Name**:  createTraj5(~, theta0, thetaf, thetad0, thedaydf, thetadd0, thetaddf, tstart, tfina) | **Purpose**:  Creates a minimum jerk trajectory given position, velocity, and acceleration at different time steps  Creates the smoothest trajectory given the points |
| **Input**:   * Angles for all of the legs * Time starting and ending for the trajectory for speed and acceleration calculation | **Output**:  Trajectory |
| **Notes**:   * Used within generateLegTraj * Do not use besides in generateLegTraj | |

|  |  |
| --- | --- |
| **Name**:  holdPos(robot, time) | **Purpose**:  Holds the robot’s position for a certain period of time |
| **Input**:  Time – the amount of time in seconds that you want XRhex to hold its position for | **Output**:  NONE |
| **Notes**:   * Not often used | |

|  |  |
| --- | --- |
| **Name**:  standUp(robot) | **Purpose**:  Makes the robot stand up |
| **Input**:  NONE | **Output**:  NONE |
| **Notes**:   * Uses a trajectory to go from the robot’s current position to standing position | |

|  |  |
| --- | --- |
| **Name**:  standUpReverse(robot) | **Purpose**:  Makes the robot stand up upside-down |
| **Input**:  NONE | **Output**:  NONE |
| **Notes**:   * Uses a trajectory to go from the robot’s current position to standing position | |

|  |  |
| --- | --- |
| **Name**:  takeStep(robot, stepSize, stepTime, gaitName, upsideDown) | **Purpose**:  Makes the robot take one step in the selected gait |
| **Input**:   * stepSize: size of each step the robot will take * stepTime: amount of time per step (speed) * gaitName: type of gait to execute * upsideDown: true if upside-down, false if not | **Output**:  NONE |
| **Notes**:   * Shell function: actual step code is in the respective “takeStep[GaitName]” functions. This function is basically just a switch statement determining which step gait to run | |

|  |  |
| --- | --- |
| **Name**:  takeStepBackwards(robot, stepSize, stepTime, upsideDown) | **Purpose**:  Takes a step backwards in the tripod gait |
| **Input**:   * stepSize: size of each step the robot will take * stepTime: amount of time per step (speed) * upsideDown: true if upside-down, false if not | **Output**:  NONE |
| **Notes**:   * For the tripod gait, the front and back legs on one side (1), and the middle leg on the other side (2), will move backward. Then, the front and back legs on the other side (2), and the middle leg on one side (1) will move backward. This is one step. * If the step sizes are not equal, the robot will turn. * Creates a trajectory based on current position and two intermediate positions, which is used to create one, repeatable step backwards | |

|  |  |
| --- | --- |
| **Name**:  takeStepTripod(robot, stepSize, stepTime, upsideDown) | **Purpose**:  Takes a step forwards in the tripod gait |
| **Input**:   * stepSize: size of each step the robot will take: 1X2 matrix for each side of the robot * stepTime: amount of time per step (speed) * upsideDown: true if upside-down, false if not | **Output**:  NONE |
| **Notes**:   * For the tripod gait, the front and back legs on one side (1), and the middle leg on the other side (2), will move forward. Then, the front and back legs on the other side (2), and the middle leg on one side (1) will move forward. This is one step. * If the step sizes are not equal, the robot will turn. * Creates a trajectory based on current position and two intermediate positions, which is used to create one, repeatable step forwards | |

|  |  |
| --- | --- |
| **Name**:  takeStepTripod(robot, stepSize, stepTime, upsideDown) | **Purpose**:  Takes a step forwards in the tripod gait |
| **Input**:   * stepSize: size of each step the robot will take: 1X2 matrix for each side of the robot * stepTime: amount of time per step (speed) * upsideDown: true if upside-down, false if not | **Output**:  NONE |
| **Notes**:   * For the tripod gait, the front and back legs on one side (1), and the middle leg on the other side (2), will move forward. Then, the front and back legs on the other side (2), and the middle leg on one side (1) will move forward. This is one step. * If the step sizes are not equal, the robot will turn. * Creates a trajectory based on current position and two intermediate positions, which is used to create one, repeatable step forwards | |

|  |  |
| --- | --- |
| **Name**:  flipRobot(robot)  **EXPERIMENTAL- USE WITH CAUTION** | **Purpose**:  Flips the robot against a wall from right-side up to upside-down |
| **Input**:  NONE | **Output**:  NONE |
| **Notes**:   * Range: touching the wall when standing up to 18.25 inches away (from contact of leg to the ground) * One can change the time of flipping by changing stepTime within the function, but this is not advised. * The user will need to change the robot to Upside-Down mode if the flip is successful. * The robot flips quickly and moves automatically after using this function. Use with caution! | |

|  |  |
| --- | --- |
| **Name**:  takeStepWave(robot, stepSize, stepTime)  **EXPERIMENTAL- USE WITH CAUTION** | **Purpose**:  Moves the robot one step forward in the wave gait |
| **Input**:   * stepSize: size of each step the robot will take: 1X2 matrix for each side of the robot * stepTime: amount of time per step (speed) | **Output**:  NONE |
| **Notes**:   * For the wave gait, the front legs move one step, then the middle legs move one step (which the front legs push forward), then the back legs move one step (while the middle legs push forward) . This is one step. * If the step sizes are not equal, the robot will turn. * Does not work upside-down * Has strange errors: use with caution! | |

|  |  |
| --- | --- |
| **Name**:  dynamicRun(robot, stepSize, speed)  **EXPERIMENTAL- USE WITH CAUTION** | **Purpose**:  Makes the robot’s legs move in a square wave.  Allows the user to stop the robot in the middle of a step |
| **Input**:   * stepSize: size of each step the robot will take: 1X2 matrix for each side of the robot * speed: decimal from 0-1, used to calculate the maximum and minimum velocity | **Output**:  NONE |
| **Notes**:   * Sometimes has jerky movement | |

|  |  |
| --- | --- |
| **Name**:  forwardLeap(robot)  **EXPERIMENTAL- USE WITH CAUTION** | **Purpose**:  Makes the robot crouch and jump forward using all six legs |
| **Input**:  None | **Output**:  None |
| **Notes**: | |

|  |  |
| --- | --- |
| **Name**:  upwardLeap(robot)  **EXPERIMENTAL- USE WITH CAUTION** | **Purpose**:  Makes the robot crouch and jump upward using all six legs |
| **Input**:  None | **Output**:  None |
| **Notes**: | |

**XRhex Class: Programming Notes**

* How to Make a Gait Upside-Down
  + Switch the step sizes of the left and right sides
  + Add pi\*[1 -1 1 -1 1 -1]’ or pi\* [-1 1 -1 1 -1 1]’ to each position in the gait
  + This will take some experimentation but should be relatively fast to complete.
* Converting Discretized Movements to Trajectories
  + Problem: generateLegTraj takes a 6X1 column vector, moveLegsToPos takes a 1X6 row vector
  + Delete all of “moveLegsToPos” code
  + Either make sure that all of your position vectors are column vectors, with each entry separated by a semicolon, or transpose all of your position vectors by adding an apostrophe (X -> X’)
  + Add these lines of code: Underlined italics need to be replaced by the programmer
  + stepPoints = [*each position goes here, separated by a space*];
* stepTimes = linspace(0,stepTime,size(stepPoints,2));
* speeds = zeros(6,*NumberOfStepPoints*);
* walkTraj = robot.generateLegTraj(stepPoints,stepTimes,speeds);
* robot.followLegTraj(walkTraj,1,size(walkTraj,2));
* How to Find the Robot’s Current Position:
  + Add these lines of code: curPos will now store the robot’s current position
  + robot.fbk = robot.group.getNextFeedback();
* curPos = robot.fbk.position'.\*robot.directionFlip';

**Controller Script**

The controller script allows the user to run the robot using a basic video-game controller.

The experimental controller script has all of the same capabilities as controller script, with additional experimental commands. This should only be run by experienced users.

|  |  |  |
| --- | --- | --- |
| Button | Resulting Action | Notes |
| 1 | Flips robot against wall |  |
| 2 | Toggles upside-down | Will be replaced when pose control is implemented |
| 3 | Controller Script:  Moves legs backward  Experimental Controller Script: Initializes stair climbing |  |
| 4 | Points legs down |  |
| 5 | Decreases step time by .25 s | If the current step time is the minimum step size, step time does not change |
| 6 | Increases step time by .25 s | If the current step time is the maximum step size, step time does not change |
| 7 | Decreases step size by pi/12 | If the current step size is the minimum step size, step size does not change |
| 8 | Increases step size by pi/12 | If the current step size is the maximum step size, step size does not change |
| 9 | Changes gait from tripod to wave, or vice versa |  |
| 10 | Quits the controller code |  |
| 11 | Prints instructions in console |  |
| 12 | Robot stands up |  |
| D-Pad | Drives robot forward, arc turn, or point turn in chosen gait  Experimental Controller Script:  if stair climbing is chosen, stair climbs forward |  |
| Stick 1 | Unmapped |  |
| Stick 2 | Unmapped |  |

* To avoid duplicate button presses, the Matlab stopwatch timer is used. Toc(lastX) is the amount of time that X was last changed. If the value of toc is greater than human reaction time (stored in buttonIgnore), lastX = tic, the current time, and the button is triggered.
  + Documentation: <https://www.mathworks.com/help/matlab/ref/tic.html>, <https://www.mathworks.com/help/matlab/ref/toc.html>