ScorBot Toolbox Quick Reference Guide

Table of Contents

Units 3

Common Variables 3

Basic Function Naming Convention 4

ScorBot Hardware Interaction 4

ScorBot Conversions 4

ScorBot Simulation 4

Quick Reference for Commonly Used Functions 5

ScorBot Hardware Interaction 5

Initialization and Shutdown 5

General Utilities 5

Get Movement Speed 6

Set Movement Speed 6

Arm Measurements 7

Absolute Arm Movements 7

Relative Arm Movements 8

Gripper Measurements 8

Gripper Movements 9

Movement Utilities 9

Teach Pendant Utilities 10

ScorBot Conversions 10

BSEPR Conversions 10

XYZPR Conversions 11

End-effector Frame Conversions 11

General Conversions 12

ScorBot Simulation 12

Initialization 12

General Utilities 12

Arm Measurements 12

Absolute Arm Movements 13
Relative Arm Movements 13
Gripper Measurements 13
Gripper Movements 14
Simulation Teach User Interface 14
Update and Version Utilities 14
Basic Hardware Example 16
Basic Simulation Example 17

Units

Unless otherwise noted, the units used by the ScorBot Toolbox functions will be millimeters for linear measures, radians for angles, or unitless values (e.g. for elements of a rotation matrix).

Common Variables

The most widely used variables in the ScorBot Toolbox are:

- `BSEPR` - a 1x5 array containing the joint angles of ScorBot in radians.
  - `BSEPR = ...`
    `[BaseAngle,ShoulderAngle,ElbowAngle,WristPitch,WristRoll];`
  - `BSEPR(1)` - Base Joint Angle (radians)
  - `BSEPR(2)` - Shoulder Joint Angle (radians)
  - `BSEPR(3)` - Elbow Joint Angle (radians)
  - `BSEPR(4)` - Wrist Pitch Angle (radians)
  - `BSEPR(5)` - Wrist Roll Angle (radians)
- `XYZPR` - a 1x5 array containing the position and orientation of the ScorBot end-effector in millimeters (for position) and radians (for orientation).
  - `XYZPR = ...`
    `[EndEffectorX,EndEffectorY,EndEffectorZ,... % (mm)`
    `EndEffectorPitch,EndEffectorRoll]; % (radians)`
  - `XYZPR(1)` - X-position of the end-effector relative to the base frame (millimeters)
  - `XYZPR(2)` - Y-position of the end-effector relative to the base frame (millimeters)
  - `XYZPR(3)` - Z-position of the end-effector relative to the base frame (millimeters)
  - `XYZPR(4)` - End-effector pitch relative to the base frame (radians)
  - `XYZPR(5)` - End-effector wrist roll relative to a body-fixed frame* (radians)
    **NOTE: Wrist roll, and end-effector wrist roll are equal (`BSEPR(5) = XYZPR(5)`)*
- `Pose` or `H` - a 4x4 array containing translation and rotation information (in a homogeneous transformation) describing the position and orientation of the ScorBot end-effector
- `Grip` - scalar value describing the grip state of the ScorBot end-effector. Valid grip values range from 0 (fully closed) to 70 (fully open) millimeters
- `Confirm` - binary value ("`true`" or "`false`") indicating whether a specific function was executed successfully.

Basic Function Naming Convention

ScorBot Hardware Interaction

Most functions contained within the ScorBot Toolbox adhere to the following naming convention:

- `ScorGet*` - gets some measurement or quantity from the ScorBot (e.g. `ScorGetGripper` returns the amount the gripper is open in millimeters).
- `ScorSet*` - sets some measurement or quantity of the ScorBot (e.g. `ScorSetGripper(50)` sets the opening of the gripper to 50 millimeters).
- `ScorIs*` - returns a binary value describing some element of ScorBot (e.g. `ScorIsMoving` returns a "`true`" if ScorBot is moving, and a "`false`" otherwise).

ScorBot Conversions

Conversions related to ScorBot and ScorBot kinematics adhere to the following naming convention:

- `Scor*2*` - converts the first parameter to the second (e.g. `ScorBSEPR2XYZPR(BSEPR)` returns the `XYZPR` values associated with the input `BSEPR` values).

ScorBot Simulation

Both simplified and advanced kinematic simulations of ScorBot are available within the ScorBot Toolbox and generally adhere to the following naming convention:

- `ScorSimGet*` - gets some measurement or quantity from the ScorBot simulation (e.g. `ScorSimGetBSEPR(simObj)` returns the `BSEPR` values associated with the simulation specified using the variable `simObj`).
- `ScorSimSet*` - sets some measurement or quantity of the ScorBot simulation (e.g. `ScorSimSetBSEPR(simObj,BSEPR)` sets the `BSEPR` values of the simulation specified using the variable `simObj`).

Quick Reference for Commonly Used Functions

ScorBot Hardware Interaction

Initialization and Shutdown

Function	Description and Syntax Example(s)
Initialization and Shutdown	

`ScorInit`	Loads DLLs, sets up USB communication, and enables control of ScorBot.
	
	`ScorInit;`
	`confirm = ScorInit;`

`ScorHome`	Home the ScorBot to calibrate absolute joint measurements.
	
	`ScorHome;`
	`confirm = ScorHome;`

`ScorSafeShutdown`	Move the ScorBot to the home position, disables control, and unloads libraries.
	
	`ScorSafeShutdown;`
	`confirm = ScorSafeShutdown;`

General Utilities

General Utilities	

`ScorGoHome`	Move ScorBot back to the home configuration.
	
	`ScorGoHome;`
	`confirm = ScorGoHome;`

`ScorGetControl`	Get the current control state of ScorBot ("`On`" is ScorBot's control is enabled or "`Off`" is ScorBot's control is disabled. `cState = ScorGetControl;`

`ScorSetControl`	Enable or Disable control of ScorBot.
	
	`ScorSetControl('On');`
	`ScorSetControl('Off');`
	`confirm = ScorSetControl(___);`

Get Movement Speed

Get Movement Speed	
`ScorGetSpeed`	Get the current joint speed as a percent of maximum (from 0 to 100). If a move time is set, this function returns an empty set. This remains fixed until a new speed or move time is declared. Note: Changes made to the speed using the Teach Pendant do not update the speed recorded in the toolbox. Speed must be updated using `ScorSetSpeed` for this function to work correctly. `Speed = ScorGetSpeed;`
`ScorGetMoveTime`	Get the time for moves (from start to finish) in seconds. If a speed is set, this function returns an empty set. This remains fixed until a new speed or move time is declared. Note: Changes made to the move time using the Teach Pendant do not update the move time recorded in the toolbox. Move time must be updated using `ScorSetMoveTime` for this function to work correctly. `MoveTime = ScorGetMoveTime;`

Set Movement Speed

Set Movement Speed	

`ScorSetSpeed`	Set the allowable joint speed to a percent of maximum (from 0 to 100). This remains fixed until a new speed or move time is declared.
	
	`ScorSetSpeed(Speed);`
	`confirm = ScorSetSpeed(___);`

`ScorSetMoveTime`	Set the total time for moves (from start to finish) to a value in seconds. This remains fixed until a new speed or move time is declared.
	
	`ScorSetMoveTime(MoveTime);`
	`confirm = ScorSetMoveTime(___);`

Arm Measurements

Arm Measurements	
`ScorGetBSEPR`	Get a 1x5 array containing the current joint angles of ScorBot in radians. `BSEPR = ScorGetBSEPR;`
`ScorGetXYZPR`	Get a 1x5 array containing the current end-effector position and orientation of ScorBot in millimeters (for position) and radians (for orientation). `XYZPR = ScorGetXYZPR;`
`ScorGetPose`	Get a 4x4 array containing the homogeneous transformation describing the end-effector position and orientation relative to the base frame of ScorBot. `H = ScorGetPose;`

Absolute Arm Movements

Absolute Arm Movements	

`ScorSetBSEPR`	Set the ScorBot joint configuration (in radians) to the values specified in a 1x5 array.
	
	`ScorSetBSEPR(BSEPR);`
	`ScorSetBSEPR(BSEPR,'MoveType','LinearTask');`
	`ScorSetBSEPR(BSEPR,'MoveType','LinearJoint');`
	`confirm = ScorSetBSEPR(___);`

`ScorSetXYZPR`	Set the ScorBot end-effector position (in millimeters) and orientation (in radians) to the values specified in a 1x5 array.
	
	`ScorSetXYZPR(XYZPR);`
	`ScorSetXYZPR(XYZPR,'MoveType','LinearTask');`
	`ScorSetXYZPR(XYZPR,'MoveType','LinearJoint');`
	`confirm = ScorSetXYZPR(___);`

`ScorSetPose`	Set the ScorBot end-effector position and orientation using a 4x4 homogeneous transformation specified relative to the ScorBot base.
	
	`ScorSetPose(H);`
	`ScorSetPose(H,'MoveType','LinearTask');`
	`ScorSetPose(H,'MoveType','LinearJoint');`
	`confirm = ScorSetPose(___);`

Relative Arm Movements

Relative Arm Movements	

`ScorSetDeltaBSEPR`	Set the ScorBot joint configuration (in radians) relative to the current joint configuration.
	
	`ScorSetDeltaBSEPR(dBSEPR);`
	`ScorSetDeltaBSEPR(dBSEPR,'MoveType','LinearTask');`
	`ScorSetDeltaBSEPR(dBSEPR,'MoveType','LinearJoint');`
	`confirm = ScorSetDeltaBSEPR(___);`

`ScorSetDeltaXYZPR`	Set the ScorBot end-effector position (in millimeters) and orientation (in radians) relative to the current end-effector position and orientation.
	
	`ScorSetDeltaXYZPR(dXYZPR);`
	`ScorSetDeltaXYZPR(dXYZPR,'MoveType','LinearTask');`
	`ScorSetDeltaXYZPR(dXYZPR,'MoveType','LinearJoint');`
	`confirm = ScorSetDeltaXYZPR(___);`

`ScorSetDeltaPose`	Set the ScorBot end-effector position and orientation using a 4x4 homogeneous transformation specified relative to the current position and orientation of the ScorBot end-effector.
	
	`ScorSetDeltaPose(dH);`
	`ScorSetDeltaPose(dH,'MoveType','LinearTask');`
	`ScorSetDeltaPose(dH,'MoveType','LinearJoint');`
	`confirm = ScorSetDeltaPose(___);`

Undo Arm Movements

Gripper Measurements	

`ScorSetUndo`	Set the ScorBot to the previously set joint configuration.
	
	`ScorSetUndo;`
	`confirm = ScorSetUndo;`

Gripper Measurements

Gripper Measurements	
`ScorGetGripper`	Get the current gripper state of the ScorBot in millimeters. The gripper state is measured by the distance between the gripper fingers. `grip = ScorGetGripper;`
`ScorGetGripperOffset`	Get the approximate offset between the gripper fingertip and the end-effector frame along the z-axis. Note: This value changes as the gripper is opened and closed due to the four-bar linkage design of the gripper fingers. `gOffset = ScorGetGripperOffset;`

Gripper Movements

Gripper Movements	

`ScorSetGripper`	Set the current gripper state of the ScorBot in millimeters. The gripper state is measured by the distance between the gripper fingers.
	
	`ScorSetGripper(grip);`
	`ScorSetGripper('Open');`
	`ScorSetGripper('Close');`
	`confirm = ScorSetGripper(___);`

Movement Utilities

Movement Utilities	
`ScorIsMoving`	Check if ScorBot is currently executing a move. Returns a "true" if ScorBot is moving, and a "false" otherwise. `bin = ScorIsMoving;`

`ScorWaitForMove`	Wait for ScorBot to complete a move.
	
	NOTE: This is a very powerful function that enables data collection and/or visualization during movements of the ScorBot in addition to blocking MATLAB from executing commands while ScorBot is moving.
	
	NOTE: When collecting waypoint information using `ScorGetBSEPR` or `ScorGetXYZPR`, adding `pause(2)` following `ScorWaitForMove` will ensure the ScorBot has fully executed the move and come to rest prior to acquiring the point.
	
	Basic Functionality:
	`ScorWaitForMove;`
	`confirm = ScorWaitForMove;`
	
	Basic Syntax:
	`ScorWaitForMove('PropertyName','PropertyValue');`
	
	Plotting/Visualization:
	`ScorWaitForMove('XYZPRPlot','On');`
	`ScorWaitForMove('BSEPRPlot','On');`
	`ScorWaitForMove('RobotAnimation','On');`
	`ScorWaitForMove('XYZPRPlot','On',...`
	`'BSEPRPlot','On','RobotAnimation','On');`
	`[~,h] = ScorWaitForMove(___,'PlotHandle',h);`
	
	Data Collection:
	`[confirm,~,data] = ...`
	`ScorWaitForMove('CollectData','On');`

Teach Pendant Utilities

Teach Pendant Utilities	
`ScorGetPendantMode`	Get the current teach pendant mode (`'Auto'` or `'Teach'`) `pMode = ScorGetPendantMode;`

`ScorSetPendantMode`	Set teach pendant mode (through a user prompt).
	
	`ScorSetPendantMode('Auto');`
	`ScorSetPendantMode('Teach');`
	`confirm = ScorSetPendantMode(___);`

ScorBot Conversions

BSEPR Conversions

Function	Description and Syntax Example(s)
BSEPR Conversions	
`ScorBSEPR2XYZPR`	Convert a 1x5 array defining a ScorBot joint configuration (in radians) to a 1x5 array defining the associated ScorBot end-effector position (in millimeters) and orientation (in radians). `XYZPR = ScorBSEPR2XYZPR(BSEPR);`
`ScorBSEPR2Pose`	Convert a 1x5 array defining a ScorBot joint configuration (in radians) to a 4x4 homogeneous transformation defining the associated ScorBot end-effector position (in millimeters) and orientation. `H = ScorBSEPR2Pose(BSEPR);`

XYZPR Conversions

XYZPR Conversions	

`ScorXYZPR2BSEPR`	Convert a 1x5 array defining a ScorBot end-effector position (in millimeters) and orientation (in radians) to one or more 1x5 arrays defining the associated ScorBot joint configuration (in radians).
	
	NOTE: The default returned solution is associated with the elbow-up configuration of ScorBot.
	
	NOTE: Multiple solutions are returned as a cell array.
	
	`BSEPR = ScorXYZPR2BSEPR(XYZPR);`
	`BSEPR = ScorXYZPR2BSEPR(___,'ElbowUpSolution');`
	`BSEPR = ScorXYZPR2BSEPR(___,'ElbowDownSolution');`
	`BSEPRs = ScorXYZPR2BSEPR(___,'AllSolutions')`

`ScorXYZPR2Pose`	Convert a 1x5 array defining a ScorBot end-effector position (in millimeters) and orientation (in radians) to one or more 4x4 homogeneous transformation defining the associated ScorBot end-effector position (in millimeters) and orientation.
	
	NOTE: Multiple solutions are returned as a cell array.
	
	`H = ScorXYZPR2BSEPR(XYZPR);`
	`H = ScorXYZPR2BSEPR(___,'AllSolutions')`

End-effector Frame Conversions

End-effector Frame Conversions	

`ScorPose2BSEPR`	Convert a 4x4 homogeneous transformation defining the associated ScorBot end-effector position (in millimeters) and orientation to one or more 1x5 arrays defining the associated ScorBot joint configuration (in radians).
	
	NOTE: The default returned solution is associated with the elbow-up configuration of ScorBot.
	
	NOTE: Multiple solutions are returned as a cell array.
	
	`BSEPR = ScorPose2BSEPR(H);`
	`BSEPR = ScorPose2BSEPR(___,'ElbowUpSolution');`
	`BSEPR = ScorPose2BSEPR(___,'ElbowDownSolution');`
	`BSEPRs = ScorPose2BSEPR(___,'AllSolutions')`

`ScorPose2XYZPR`	Convert a 4x4 homogeneous transformation defining the associated ScorBot end-effector position (in millimeters) and orientation to one or more 1x5 arrays defining a ScorBot end-effector position (in millimeters) and orientation (in radians)
	
	NOTE: Multiple solutions are returned as a cell array.
	
	`XYZPR = ScorPose2XYZPR(H);`
	`XYZPRs = ScorPose2XYZPR(___,'AllSolutions')`

General Conversions

General Conversions	
`rad2deg`	Convert a scalar value or array of values from radians to degrees. `deg = rad2deg(rad);`
`deg2rad`	Convert a scalar value or array of values from degrees to radians. `rad = deg2rad(deg);`

ScorBot Simulation

Initialization

Function	Description and Syntax Example(s)
Initialization	
`ScorSimInit`	Initialize the basic ScorBot kinematic simulation containing a basic representation of the manipulator and relevant reference frames. `simObj = ScorSimInit;`

General Utilities

General Utilities	
`ScorSimPatch`	Add a visualization of ScorBot to the basic ScorBot simulation. `ScorSimPatch(simObj);`
`ScorSimGoHome`	Move the ScorBot simulation to the home position. `ScorSimGoHome(simObj);`

Arm Measurements

Arm Measurements	
`ScorSimGetBSEPR`	Get a 1x5 array containing the current joint angles of the ScorBot simulation in radians. `BSEPR = ScorSimGetBSEPR(simObj);`
`ScorSimGetXYZPR`	Get a 1x5 array containing the current end-effector position and orientation of the ScorBot simulation in millimeters (for position) and radians (for orientation). `XYZPR = ScorSimGetXYZPR(simObj);`
`ScorSimGetPose`	Get a 4x4 array containing the homogeneous transformation describing the end-effector position and orientation relative to the base frame of ScorBot. `H = ScorSimGetPose(simObj);`

Absolute Arm Movements

Absolute Arm Movements	
`ScorSimSetBSEPR`	Set the ScorBot simulation joint positions (in radians) to the values specified in a 1x5 array. `ScorSimSetBSEPR(simObj,BSEPR);`
`ScorSimSetXYZPR`	Set the ScorBot simulation end-effector position (in millimeters) and orientation (in radians) to the values specified in a 1x5 array. `ScorSimSetXYZPR(simObj,XYZPR);`
`ScorSimSetPose`	Set the ScorBot simulation end-effector position and orientation using a 4x4 homogeneous transformation specified relative to the ScorBot base. `ScorSimSetPose(simObj,H);`

Relative Arm Movements

Relative Arm Movements	
`ScorSimSetDeltaBSEPR`	Set the ScorBot simulation joint positions (in radians) relative to the current joint positions. `ScorSimSetDeltaBSEPR(simObj,dBSEPR);`
`ScorSimSetDeltaXYZPR`	Set the ScorBot simulation end-effector position (in millimeters) and orientation (in radians) relative to the current end-effector position and orientation. `ScorSimSetDeltaXYZPR(simObj,dXYZPR);`
`ScorSimSetDeltaPose`	Set the ScorBot simulation end-effector position and orientation using a 4x4 homogeneous transformation specified relative to the current position and orientation of ScorBot. `ScorSimSetDeltaPose(simObj,dH);`

Gripper Measurements

Gripper Measurements	
`ScorSimGetGripper`	Get the current gripper state of the ScorBot simulation in millimeters. The gripper state is measured by the distance between the gripper fingers. `grip = ScorGetGripper(simObj);`
`ScorSimGetGripperOffset`	Get the approximate offset between the gripper fingertip and the end-effector frame along the z-axis. Note: This value changes as the gripper is opened and closed due to the four-bar linkage design of the gripper fingers. `gOffset = ScorSimGetGripperOffset;`

Gripper Movements

Gripper Movements	

`ScorSimSetGripper`	Set the current gripper state of the ScorBot simulation in millimeters. The gripper state is measured by the distance between the gripper fingers.
	
	`ScorSimSetGripper(simObj,grip);`
	`ScorSimSetGripper(simObj,'Open');`
	`ScorSimSetGripper(simObj,'Close');`

Simulation Teach User Interface

Simulation Teach User Interface	
`ScorSimTeachBSEPR`	Set the ScorBot simulation to BSEPR teach mode. This enables joint-level control of the ScorBot simulation using a prescribed set of keys. `ScorSimTeachBSEPR(simObj);`
`ScorSimTeachXYZPR`	Set the ScorBot simulation to XYZPR teach mode. This enables control of the ScorBot simulation end-effector position and orientation using a prescribed set of keys. `ScorSimTeachXYZPR(simObj);`

ScorBot Remote Operation

Initialization

Function	Description and Syntax Example(s)
Initialization	
`ScorInitSender`	Initializes a UDP server for transmitting ScorBot information to a remote client. The server is specified using a port (recommended between 31000 and 32000) and IP address. `udpS = ScorInitSender(port,IP);`
`ScorInitReceiver`	Initializes a UDP receiver to receive ScorBot information from a remote server. The receiver is tied to a designated port that must match the port of the corresponding sender. This currently uses a default IP of 0.0.0.0 allowing data to be accepted from any remote IP address. `udpR = ScorInitReceiver(port);`

General Utilities

General Utilities	

`ScorSendBSEPRG`	Sends the current or a specified BSEPR value and gripper state to a designated UDP server.
	
	`ScorSendBSEPRG(udpS);`
	`ScorSendBSEPRG(udpS,BSEPR);`
	`ScorSendBSEPRG(udpS,BSEPR,grip);`

`ScorReceiveBSEPRG`	Receives a BSEPR value and gripper state from a designated UDP receiver object. `[BSEPR,grip] = ScorReceiveBSEPRG(udpR);`

Update and Version Utilities

Update and Version Utilities	
`ScorUpdate`	Update the ScorBot Toolbox to the latest version using an active internet connection. Note: This function requires that MATLAB be run in Administrator mode. Note: The update process includes a basic motion test of the ScorBot for operating systems that support ScorBot Controller interaction. `ScorUpdate;`

`ScorVer`	Check the current version of the ScorBot Toolbox.
	
	`ScorVer;`
	`verStruct = ScorVer;`

Basic Hardware Example

```
%% Initialize and home ScorBot
% Note: You only need to run this once! If you already ran ScorInit and
% ScorHome, you do not need to run them again.
ScorInit;
ScorHome;

%% Define desired waypoints as end-point XYZPR positions/orientations
XYZPRs(1,:) = [500.000,-200.000,570.000,0.000,0.000];
XYZPRs(2,:) = [500.000, 200.000,570.000,0.000,0.000];
XYZPRs(3,:) = [500.000, 200.000,270.000,0.000,0.000];
XYZPRs(4,:) = [500.000,-200.000,270.000,0.000,0.000];
XYZPRs(5,:) = XYZPRs(1,:);

%% Convert XYZPR waypoints to BSEPR joint configurations
for wpnt = 1:size(XYZPRs,1)
    BSEPRs(wpnt,:) = ScorXYZPR2BSEPR(XYZPRs(wpnt,:));
end

%% Set speed and initialize arm configuration
ScorSetSpeed(100);
ScorSetXYZPR(XYZPRs(1,:));
ScorWaitForMove;

%% Move through end-point XYZPR positions/orientations
h = []; % initialize variable for plot handle
fprintf('Demonstrating XYZPR move with Animation Plots.\n');
title(h.RobotAnimation.Sim.Axes,'Movements using ScorSetXYZPR');
for wpnt = 1:size(XYZPRs,1)
    ScorSetXYZPR(XYZPRs(wpnt,:));
    [~,h] = ScorWaitForMove('RobotAnimation','On','PlotHandle',h);
end
plot3(h.RobotAnimation.Sim.Axes,XYZPRs(1:4,1),XYZPRs(1:4,2),XYZPRs(1:4,3),'*k');

%% Move through BSEPR joint configurations
h = []; % initialize variable for plot handle
fprintf('Demonstrating BSEPR move with Animation Plots.\n');
title(h.RobotAnimation.Sim.Axes,'Movements using ScorSetBSEPR');
for wpnt = 1:size(BSEPRs,1)
    ScorSetBSEPR(BSEPRs(wpnt,:));
    [~,h] = ScorWaitForMove('RobotAnimation','On','PlotHandle',h);
end
plot3(h.RobotAnimation.Sim.Axes,XYZPRs(1:4,1),XYZPRs(1:4,2),XYZPRs(1:4,3),'*k');

%% Safe shutdown
% Note: You only need to run this when you are finished using MATLAB or
% finished using ScorBot! If you run ScorSafeShutdown and still need to use
% ScorBot, you will need to reinitialize using ScorInit, and rehome using
% ScorHome.
ScorSafeShutdown;
```

Basic Simulation Example

```
%% Initialize simulation and visualize ScorBot
% Note: Each time you run this, you will create a new simulation figure
simObj = ScorSimInit;
ScorSimPatch(simObj);

%% Define desired waypoints as end-point XYZPR positions/orientations
XYZPRs(1,:) = [500.000,-200.000,570.000,0.000,0.000];
XYZPRs(2,:) = [500.000, 200.000,570.000,0.000,0.000];
XYZPRs(3,:) = [500.000, 200.000,270.000,0.000,0.000];
XYZPRs(4,:) = [500.000,-200.000,270.000,0.000,0.000];
XYZPRs(5,:) = XYZPRs(1,:);

%% Convert XYZPR waypoints to BSEPR joint configurations
for wpnt = 1:size(XYZPRs,1)
    BSEPRs(wpnt,:) = ScorXYZPR2BSEPR(XYZPRs(wpnt,:));
end

%% Interpolate between waypoint for animation
n = 50;
XYZPR_all = [];
BSEPR_all = [];
for jnt = 1:size(XYZPRs,2)
    for wpnt = 1:size(XYZPRs,1)-1
        XYZPR_all(n*(wpnt-1)+1:n*(wpnt-1)+n,jnt) = ...
            linspace(XYZPRs(wpnt,jnt),XYZPRs(wpnt+1,jnt),n);
        BSEPR_all(n*(wpnt-1)+1:n*(wpnt-1)+n,jnt) = ...
            linspace(BSEPRs(wpnt,jnt),BSEPRs(wpnt+1,jnt),n);
    end
end

%% Move through end-point XYZPR positions/orientations
plt = plot3(simObj.Axes,0,0,0,'.m'); % initialize waypoint plot handle
clear xData yData zData
fprintf('Demonstrating simulated XYZPR move.\n');
title(simObj.Axes,'Movements using ScorSimSetXYZPR (Magenta)');
for ipnt = 1:size(XYZPR_all,1)
    ScorSimSetXYZPR(simObj,XYZPR_all(ipnt,:));
    XYZPR = ScorSimGetXYZPR(simObj);
    xData(ipnt) = XYZPR(1);
    yData(ipnt) = XYZPR(2);
    zData(ipnt) = XYZPR(3);
    set(plt,'xData',xData,'yData',yData,'zData',zData);
end
plot3(simObj.Axes,XYZPRs(1:4,1),XYZPRs(1:4,2),XYZPRs(1:4,3),'ok');

%% Move through BSEPR joint configurations
plt = plot3(simObj.Axes,0,0,0,'.c'); % initialize waypoint plot handle
clear xData yData zData
fprintf('Demonstrating simulated BSEPR move.\n');
title(simObj.Axes,'Movements using ScorSimSetBSEPR (Cyan)');
for ipnt = 1:size(BSEPR_all,1)
    ScorSimSetBSEPR(simObj,BSEPR_all(ipnt,:));
    XYZPR = ScorSimGetXYZPR(simObj);
    xData(ipnt) = XYZPR(1);
    yData(ipnt) = XYZPR(2);
    zData(ipnt) = XYZPR(3);
    set(plt,'xData',xData,'yData',yData,'zData',zData);
end
plot3(simObj.Axes,XYZPRs(1:4,1),XYZPRs(1:4,2),XYZPRs(1:4,3),'+k');
```