Universidad de Sevilla

Escuela Politécnica
Superior de Sevilla

Trabajo Fin de Grado en Ingeniería Electrónica Industrial

Prototipo de control brazo robótico industrial con detección de objetos basado en Deep-Learning

ANEXO 3. CÓDIGO DE FUNCIONES EN MATLAB.

Autor: Roque Sánchez Ferrera

Tutores: Alejandro Linares Barranco y Enrique Piñero Fuentes

Fecha de Presentación: Julio 2023

ÍNDICE GENERAL

1 INTRODUCCIÓN.

En este Anexo 3, se muestra el código implementado en las funciones de la Scorbot toolbox de Matlab para el control del robot. Solamente se mostrarán aquellas que se han utilizado en el proyecto.

- ScorInit().
- ScorHome().
- ScorSetSpeed().
- ScorSetGripper().
- ScorWaitForMove().
- ScorSetXYZPR().
- ScorGoHome().
- ScorSafeShutdown().

Además se mostrará el uso y código de la función deg2rad() utilizada en el proyecto.

2 ScorInit().

```
function confirm = ScorInit()
% SCORINIT initialize ScorBot
%   SCORINIT initializes ScorBot by loading applicable DLLs, initializing
%   USB communication, and enabling control.
%
%   confirm = SCORINIT(___) returns 1 if successful and 0 otherwise.
%
%   See also: ScorHome ScorSafeShutdown
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function name "ScorInit.m"
%
%   C. Wick, J. Esposito, K. Knowles, & M. Kutzer, 10Aug2015, USNA
%
```

```
%   J. Donnal, 19Jun2017, USNA (64-bit Support)

% Updates
%   25Aug2015 - Updated correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   17Jul2019 - Updated to replace instances of "calllib.m" with
%               "ScorCallLib.m" to include J. Donnal 64-bit solution
%   17Jul2019 - Updated to differentiate between 32-bit and 64-bit
%               initialization

persistent osbits

%% Check operating system
if isempty(osbits)
    switch computer
        case 'PCWIN'
            % 32-bit OS
            osbits = 32;
        case 'PCWIN64'
            % 64-bit OS
            osbits = 64;
    end
end

%% Delete extra shutdown figure handles
ShutdownFig = 1845;
if ishandle(ShutdownFig)
    delete(ShutdownFig);
end

%% Define library alias
libname = 'RobotDll';

%% Create background figure to force ScorSafeShutdown when closing MATLAB
ShutdownFig = figure(ShutdownFig);
set(ShutdownFig,...
    'Color',[0.5,0.5,0.5],...
    'Name','ScorSafeShutdown',...
    'MenuBar','none',...
    'NumberTitle','off',...
    'ToolBar','none',...
    'Units','normalized',...
    'Position',[0.005,0.943,0.167,0.028],...
    'HandleVisibility','off',...
    'Visible','off',...
    'CloseRequestFcn',@ScorShutdownCallback);

%% Check if library is loaded
confirm = true;
wrn = {};
switch osbits
    case 32
        % 32-bit OS
        % -> [...] = calllib(libname,funcname,...);
        fprintf('Loading ScorBot library...');
        if ~libisloaded(libname)
            % Check for dll file
            if exist('RobotDll.dll','file') ~= 3 % MEX-file on MATLAB's
search path
                confirm = false;
                wrn{end+1} = '"RobotDll.dll" not found.';
            end
            % Check for header file
            if exist('RobotDll.h','file') ~= 2 % full pathname to a file on
MATLAB's search path
                confirm = false;
                wrn{end+1} = '"RobotDll.h" not found.';
            end
            % Return if files do not exist
            if ~confirm
                fprintf('FAILED\n');
                for i = 1:numel(wrn)
                    warning(wrn{i});
                end
                return
            end
            % Load library
            [notFound,warnings] =
loadlibrary('RobotDll','RobotDll.h','alias',libname);
            if ~isempty(notFound)
                confirm = false;
                fprintf('FAILED\n');
                warning('Failed to load library.');
                warning(warnings);
                return
            end
            % Check for success
            if libisloaded(libname)
                fprintf('SUCCESS\n');
            else
                confirm = false;
                fprintf('FAILED\n');
                warning('Library did not load successfully.');
                return
            end
        else
            fprintf('SKIPPED\n');
            fprintf('\tScorBot library "%s" is already loaded.\n',libname);
        end
    case 64
        % 64-bit OS
        % -> ScorServerCMD using ScorbotServer.exe
        fprintf('Starting ScorBot Server...');
        % Check if server is running
        isRunning = ScorServerIsRunning;
        if ~isRunning
            % Check if server exists
            if ~ScorServerExist
                confirm = false;
                wrn{end+1} = '"ScorbotServer.exe" not found.';
            end
            % Return if server executable does not exist
            if ~confirm
                fprintf('FAILED\n');
                for i = 1:numel(wrn)
                    warning(wrn{i});
                end
                return
            end
            % Try to run the server
            try
                ScorServerStart;
                t0 = tic;
                t_out = 30;
                while ~ScorServerIsRunning
                    % Wait for server to start
                    t = toc(t0);
                    if t > t_out
                        fprintf('TIMEOUT\n');
                        warning('Server did not start successfully.');
                        return
                    end
                end
                fprintf('SUCCESS\n');
            catch
                confirm = false;
                fprintf('FAILED\n');
                warning('Server did not start successfully.');
                return
            end
        else
            fprintf('SKIPPED\n');
            fprintf('\tScorBot Server is already running.\n');
        end
    otherwise
        error('OSBITS variable not set to known value.');
end

%% Initialize ScorBot
fprintf('\n');
switch osbits
    case 32
        fprintf('Initializing USB interface...');
        isCalled = ScorCallLib(libname,'RInitialize');
        if ~isCalled
            confirm = false;
            fprintf('FAILED\n');
            warning('Unable to initialize.');
            return
        end
        % Wait for initialization to complete
        iter = 0;
        while iter < 20 % pause(2)
            if mod(iter,4) == 0
                fprintf(char([8,8,8]));
            else
                fprintf('.');
            end
            iter = iter+1;
            pause(0.1);
        end
        % Wait for initialization completion to be confirmed
        while ScorCallLib(libname,'RIsInitDone') == 0
            if mod(iter,4) == 0
                fprintf(char([8,8,8]));
            else
                fprintf('.');
            end
            iter = iter+1;
            pause(0.1);
            %TODO - add a FAILED stop condition
        end
        fprintf( char(repmat(8,1,mod(iter-1,4))) );
        fprintf('...');
        fprintf('SUCCESS\n');
        fprintf('\t"POWER" light on the ScorBot Control Box should be
green.\n');
    case 64
        fprintf('Checking initialization status...');
        % Exception to "ScorCallLib"
        t0 = tic;
        t_out = 30;
        isDone = false;
        while ~isDone
            % Check initialization status
            try
                status = ScorServerCmd('RIsInitDone');
                isDone = true;
            catch
                pause(0.1);
            end
            % Throw timeout
            t = toc(t0);
            if t > t_out
                fprintf('CONNECTION REFUSED\n');
                warning('Server did not start successfully.');
                return
            end
        end
        % Check status
        switch lower(status)
            case 'error'
                fprintf('Error communicating with ScorbotServer, is it
running?\n')
                warning('Unable to initialize.');
                return
            otherwise
                % DO NOTHING
        end
        fprintf('SUCCESS\n');
        fprintf('\t"POWER" light on the ScorBot Control Box should be
green.\n');
    otherwise
```

```
        error('OSBITS variable not set to known value.');
end

%% Define USNA vector for sending points to the robot
fprintf('\n');
fprintf('Defining default waypoint vector...');
isCreated = ScorCreateVector('USNA',1000);
if ~isCreated
    confirm = false;
    fprintf('FAILED\n');
    warning('Unable to create waypoint vector location.');
    return
end
fprintf('SUCCESS\n\n');
fprintf('Initialization complete.\n');
fprintf('\t(1) Turn teach pendant to AUTO\n');
fprintf('\t(2) Home the ScorBot using "ScorHome"\n')
fprintf('\n'); % add line for cleaner display in the command prompt

%% Confirm and set hidden figure to green
confirm = true;
set(ShutdownFig,'Color',[0.0,1.0,0.0]);
```

3 ScorHome().

```
function confirm = ScorHome(varargin)
% SCORHOME homes the ScorBot
%   SCORHOME homes the ScorBot and enables control.
%
%   confirm = SCORHOME returns 1 if successful and 0 otherwise. If homing
%   has already been executed, a pop-up dialog will prompt the user to see
%   if re-homing is the desired course of action.
%
%   confirm = SCORHOME(true) will bypass the user dialog and execute the
%   homing sequence regardless of prior homing.
%
%   See also ScorInit
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function name "ScorHome.m"
%
%   C. Wick, J. Esposito, K. Knowles, & M. Kutzer, 10Aug2015, USNA
%
%   J. Donnal, 28Jun2017, USNA (64-bit Support)

% Updates
%   25Aug2015 - Updated correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   28Aug2015 - Updated to include move-in-place after successful homing
%               successfully setting control to "on". This should allow
%               "ScorIsMoving" to reflect a state of 0 once ScorBot has
%               homed.
%   01Sep2015 - Updated to include set to default speed of 50%
%   08Sep2016 - Updated to include check for previously homed ScorBot with
%               pop-up (suggested by MIDN Jaunich)
%   13Jan2017 - Updated documentation
%   25Sep2018 - Updated to include "last error" reset
%   02Oct2018 - Updated to include error logging
%   17Jul2019 - Updated to replace instances of "calllib.m" with
%               "ScorCallLib.m" to include J. Donnal 64-bit solution
%   17Jul2019 - Updated to differentiate between 32-bit and 64-bit
%               initialization

%% Define persistent variable for homing
persistent priorHome osbits

% Set default prior home value to false
if isempty(priorHome)
    priorHome = false;
end

% Check if ScorBot has been shutdown
ShutdownFig = 1845;
if ~ishandle(ShutdownFig)
    % Implies ScorBot has been shutdown
    priorHome = false;
end

%% Check operating system
if isempty(osbits)
    switch computer
        case 'PCWIN'
            % 32-bit OS
            osbits = 32;
        case 'PCWIN64'
            % 64-bit OS
            osbits = 64;
    end
end

%% Check inputs
% TODO - update error dialog for too many inputs
narginchk(0,1);
if nargin == 1
    bypassDialog = varargin{1};
    % TODO - Consider lightening this to include 0 and 1
    if ~isscalar(bypassDialog) || ~islogical(bypassDialog)
        error('ScorHome:BadInput',...
            'Input argument must be a scalar logical value (e.g. "true" or
"false").');
    end
else
    bypassDialog = false;
```

```
end

if bypassDialog
    % Run homing
    priorHome = false;
end

%% Pop-up to ask user if they really want to rehome the ScorBot
if priorHome
    cState = ScorGetControl;
    switch lower(cState)
        case 'on'
            % Define dialog message for "Control On" scenario
            dlgMsg = ...
                sprintf(...
                    ['ScorBot was previously homed and control appears to be
enabled. \n',...
                     '\n',...
                     '  Note: You can use "ScorGoHome" as a faster
alternative to the "ScorHome" \n',...
                     '        command if ScorBot has already been homed and
is operating \n',...
                     '        correctly. \n',...
                     '\n',...
                     'Would you like ScorBot to run the homing sequence?']...
                );
             % "Fast Option"
             btn3 = 'Execute "ScorGoHome"';
             fcn3 = 'ScorGoHome'; % NOTE: @ScorGoHome is a better option
        case 'off'
            % Define dialog message for "Control Off" scenario
            dlgMsg = ...
                sprintf(...
                    ['ScorBot was previously homed but control appears ',...
                     'to be disabled. \n',...
                     '\n',...
                     '  Note: You can use "ScorSetControl(''On'')" to attempt
to recover control \n',...
                     '        of ScorBot without running the entire homing
sequence. \n',...
                     '        -> If this does not work, re-homing is
required. \n',...
                     '\n',...
                     'Would you like ScorBot to run the homing sequence?']...
                );
             % "Fast Option"
             btn3 = 'Execute "ScorSetControl(''On'')"';
             fcn3 = 'ScorSetControl(''On'')';
        otherwise
            error('Unexpected response from ScorGetControl.');
    end
    % Prompt user
    choice = questdlg(dlgMsg,'Execute Homing
Sequence','Yes','No',btn3,'Yes');
    switch lower(choice)
        case 'yes'
```

```
            % Run homing
            priorHome = false;  % Reset prior home status
        case 'no'
            % Skip homing
            confirm = 1;        % Assume ScorBot is homed
            priorHome = true;   % Maintain prior home status
            return
        case lower(btn3)
            % Execute fast option
            eval( sprintf('confirm = %s;',fcn3) );
            priorHome = true;
            return
        otherwise
            % Action cancelled
            fprintf('Action Cancelled.\n');
            % Run homing
            priorHome = false;  % Reset prior home status
    end
end

%% Define library alias
libname = 'RobotDll';

%% Check library
switch osbits
    case 32
        isLoaded = libisloaded(libname);
        if ~isLoaded
            confirm = false;
            % Error copied from ScorIsReady
            errStruct.Code       = NaN;
            errStruct.Message    = sprintf('TOOLBOX: The ScorBot library "%s" 
has not been loaded.',libname);
            errStruct.Mitigation = sprintf('Run "ScorInit" to intialize 
ScorBot');
            errStruct.QuickFix   = sprintf('ScorInit;');
            ScorDispError(errStruct);
            return
        end
    case 64
        isRunning = ScorServerIsRunning;
        if ~isRunning
            confirm = false;
            % Error copied from ScorIsReady
            errStruct.Code       = NaN;
            errStruct.Message    = sprintf('TOOLBOX: The ScorBot server "%s" 
is not running.','ScorbotServer.exe');
            errStruct.Mitigation = sprintf('Run "ScorInit" to intialize 
ScorBot');
            errStruct.QuickFix   = sprintf('ScorInit;');
            ScorDispError(errStruct);
            return
        end
    otherwise
        error('OSBITS variable not set to known value.');
end
```

```
%% Set teach pendant to auto
isAuto = ScorSetPendantMode('Auto');
if ~isAuto
    confirm = false;
    warning('Failed to set ScorBot Teach Pendant to "Auto"');
    return
end

%% Home robot
fprintf('Homing ScorBot...');
isHoming = ScorCallLib(libname,'RHome',int8('A'));
if ~isHoming
    confirm = false;
    fprintf('FAILED\n');
    warning('Unable to execute homing.');
    % Check if an actual error has occured
    sError = ScorCallLib(1,'RIsError');
    % Update "last error" code
    ScorErrorLastSet(901);
    % Write to error log
    ScorErrorLogWrite(901);
    return
end

%% Check if robot is homed
isHome = ScorCallLib(libname,'RIsHomeDone');
if ~isHome
    % Check for ScorBot error
    sError = ScorCallLib(libname,'RIsError');
    errStruct = ScorParseErrorCode(sError);
    % Check special case errors
    switch sError
        case 0
            % No error was found, check home again
            isHome = ScorCallLib(libname,'RIsHomeDone');
        case 300
            % "Emergency on" was pressed at some point, check home again
            isHome = ScorCallLib(libname,'RIsHomeDone');
        case 301
            % "Emergency off" was pressed at some point, check home again
            isHome = ScorCallLib(libname,'RIsHomeDone');
        case 903
            % Special case for control disabled
            ScorGetControl('SetControl','Off');
        otherwise
            % Unforseen error, try checking home again
            isHome = ScorCallLib(libname,'RIsHomeDone');
    end
end

if ~isHome
    confirm = false;
    fprintf('FAILED\n');
    warning('Unable to reach home position.');
    % Update "last error" code
    ScorErrorLastSet(937);
    % Write to error log
    ScorErrorLogWrite(937);
    return
end

%% Enable robot
isOn = ScorSetControl('On');
if ~isOn
    confirm = false;
    fprintf('FAILED\n');
    warning('Failed to set ScorBot Control Mode to "On".');
    % Update "last error" code
    ScorErrorLastSet(903);
    % Write to error log
    ScorErrorLogWrite(903);
    return
else
    confirm = true;
    fprintf('SUCCESS\n');
    % Execute "move" to update ScorIsMoving to 0
    XYZPR = ScorGetXYZPR;
    [~] = ScorSetXYZPR(XYZPR);
    % Initialize speed
    [~] = ScorSetSpeed(50);
    % Set prior home value to true
    priorHome = true;
    % Reset "last error" code
    ScorErrorLastSet(0);
    % Write to error log
    ScorErrorLogWrite(0);
end
```

4 ScorSetSpeed().

```
function confirm = ScorSetSpeed(PercentSpeed)
% SCORSETSPEED changes the maximum speed of ScorBot to a percent of the
% maximum possible speed.
%   SCORSETSPEED(PercentSpeed) changes the maximum speed of ScorBot to
%   "PercentSpeed" of the maximum possible speed.
%       PercentSpeed - scalar integer value, 0 < PercentSpeed <= 100
%
%   confirm = SCORSETSPEED(___) returns 1 if successful and 0 otherwise.
%
%   NOTE: Changing the speed with the teach pendant will not change the
%   speed when setting waypoint using MATLAB.
%
%   NOTE: Speed remains fixed until a new speed or move time is declared.
%
%   Example:
%       %% Initialize and home ScorBot
%       ScorInit;
%       ScorHome;
%
%       %% Define two joint positions
%       BSEPR(1,:) = [0,pi/2,-pi/2,-pi/2,0];
%       BSEPR(2,:) = [0,pi/2,-0.10,-pi/2,0];
%       % Initialize arm configuration
%       ScorSetSpeed(100);
%       ScorSetBSEPR(BSEPR(2,:));
%       ScorWaitForMove;
%
%       %% Evaluate various speeds
%       for PercentSpeed = 10:10:100
%           tic;
%           ScorSetSpeed(PercentSpeed);
%           fprintf('Moving at %d%% of max speed.\n',PercentSpeed);
%           for i = 1:size(BSEPR,1)
%               ScorSetBSEPR(BSEPR(i,:));
%               ScorWaitForMove;
%           end
%           toc
%       end
%       ScorGoHome;
%
%   See also ScorGetSpeed ScorSetMoveTime
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function name "ScorSetSpeed.m"
%
%   C. Wick, J. Esposito, K. Knowles, & M. Kutzer, 10Aug2015, USNA
%
%   J. Donnal, 28Jun2017, USNA (64-bit Support)

% Updates
%   25Aug2015 - Updated correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   28Aug2015 - Updated to include ScorGetSpeed functionality
%   01Sep2016 - Updated help documentation
%   17Jul2019 - Updated to replace instances of "calllib.m" with
%               "ScorCallLib.m" to include J. Donnal 64-bit solution

%% Check ScorBot and define library alias
[isReady,libname] = ScorIsReady;
if ~isReady
    confirm = false;
    return
end

%% Check inputs
narginchk(1,1);

if PercentSpeed <= 0 || PercentSpeed > 100
```

```
    error('Percent speed must be greater than 0 and less than or equal to
100');
end

%% Set speed
PercentSpeed = round(PercentSpeed);
isSet = ScorCallLib(libname,'RSetSpeed',PercentSpeed);
if isSet
    confirm = true;
    ScorGetSpeed('SetSpeed',PercentSpeed);
    return
else
    confirm = false;
    if nargout == 0
        warning('Failed to set the specified speed.');
    end
    return
end
```

5 ScorSetGripper().

```
function confirm = ScorSetGripper(grip)
% SCORSETGRIPPER sets the gripper state in millimeters above fully closed.
%   SCORSETGRIPPER(grip) sets the gripper state in millimeters. State is
%   measured as the distance between the gripper fingers. A fully closed
%   gripper has a "grip" of 0 mm. A fully open gripper has a "grip" of
%   70 mm.
%       grip - scalar gripper state in millimeters
%
%   Binary (Open/Close) Commands:
%       SCORSETGRIPPER('Open') fully opens the gripper
%       SCORSETGRIPPER('Close') fully closes the gripper
%
%   confirm = SCORSETGRIPPER(___) returns 1 if successful and 0 otherwise.
%
%   See also ScorGetGripper
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function name "ScorSetGripper.m"
%
%   C. Wick, J. Esposito, K. Knowles, & M. Kutzer, 12Aug2015, USNA
%
%   J. Donnal, 28Jun2017, USNA (64-bit Support)

% Updates
%   25Aug2015 - Updated correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   28Aug2015 - Updated to maintain speed even after changing the gripper
%               state using the ScorGetSpeed/ScorGetMoveTime functionality
%   15Sep2016 - Updated to correct error text "closed" to "close"
%               Christian Jaunich
%   03Oct2018 - Updated to include error logging.
%   17Jul2019 - Updated to replace instances of "calllib.m" with
%               "ScorCallLib.m" to include J. Donnal 64-bit solution

%% Check ScorBot and define library alias
[isReady,libname] = ScorIsReady;
if ~isReady
    confirm = false;
    return
end

%% Check inputs
narginchk(1,1);

%% Confirm that ScorBot is in Auto
isAuto = ScorSetPendantMode('Auto');
if ~isAuto
    confirm = false;
    return
end

%% Set gripper state
if ischar(grip) % Binary Open/Close
    switch lower(grip)
        case 'open'
            isOpen = ScorCallLib(libname,'RGripOpen');
            if isOpen
                % Write gripper value to error log
                ScorErrorLogWrite('GripCommand',70);
                confirm = true;
            else
                confirm = false;
                if nargout == 0
                    warning('Failed to set the gripper to open.');
                end
            end
        case 'close'
            isClose = ScorCallLib(libname,'RGripClose');
            if isClose
                % Write gripper value to error log
                ScorErrorLogWrite('GripCommand',0);
                confirm = true;
            else
                confirm = false;
                if nargout == 0
                    warning('Failed to set the gripper to closed.');
                end
            end
        otherwise
            error('Binary gripper commands must be either "Open" or
"Close"');
    end
else % Metric grip state
```

```
    if grip > 70 || grip < 0
        error('Gripper value must be between 0 and 70 millimeters.');
    end
    grip = round(grip);
    isGrip = ScorCallLib(libname,'RGripMetric',grip);
    if isGrip
        % Write gripper value to error log
        ScorErrorLogWrite('GripCommand',grip);
        confirm = true;
    else
        confirm = false;
        if nargout == 0
            warning('Failed to set the gripper to %d mm.',grip);
        end
    end
end

%% Reset speed or move time
spd = ScorGetSpeed;
mTime = ScorGetMoveTime;
if ~isempty(spd)
    ScorSetSpeed(spd);
elseif ~isempty(mTime)
    ScorSetMoveTime(mTime);
else
    error('Unexpected ScorGetSpeed/ScorGetMoveTime state.');
end
```

6 ScorWaitForMove().

```
function varargout = ScorWaitForMove(varargin)
% SCORWAITFORMOVE waits for current move to complete.
%   SCORWAITFORMOVE waits for current move to complete. If no inputs or
%   outputs are specified, a progress message will appear in the command
%   prompt.
%
%   NOTE: For an accurate BSEPR or XYZPR value when the robot has completed
%   its move, it is recommended to use a "pause(2)" following the
%   completion of ScorWaitForMove.
%
%   NOTE: To minimize the wait time between movements in applications that
%   do not require data acquisition during or after a movement, consider
%   using "while ScorIsMoving; end" instead of ScorWaitForMove.
%
%   SCORWAITFORMOVE('PropertyName',PropertyValue)
%
%   NOTE: Setting one or more plot or data paramter to 'On' disables the
%   command line progress display "Executing move..."
%
%       Property Name: {PropertyValues}
%           XYZPRPlot: {'On' ['Off']}
%           BSEPRPlot: {'On' ['Off']}
%      RobotAnimation: {'On' ['Off']}
%          PlotHandle: Structured array containing plot information
%                      PlotHandle.XYZPRPlot - XYZPRPlot handles
%                      PlotHandle.BSEPRPlot - BSEPRPlot handles
%                      PlotHandle.RobotAnimation - Robot animation handles
%        CollectData: {'On' ['Off']}
%
%       XYPRPlot - plot XYZPR parameters as a function of time as ScorBot
%           executes a move.
%       BSEPRPlot - plot BSEPR parameters as a function of time as ScorBot
%           executes a move.
%       RobotAnimation - plot ScorBot DH frames and end-effector pose
%           evolution as ScorBot executes a move.
%       PlotHandle - specify a struct containing the plot handles as
%           specified above. This is primarily used to avoid creating
%           multiple figures for recursive calls of ScorWaitForMove with
%           plots or animations enabled.
%       CollectData - collect time stamped XYZPR and BSEPR data into a
%           structured array.
%
%   NOTE: When using plots, animations, or collecting data, a "pause(2)" is
%   automatically used to collect the trailing data points following
%   ScorBot's execution of a move.
%
%   confirm = SCORWAITFORMOVE(___) returns 1 if successful and 0 otherwise.
%
%   NOTE: Specifying one or more outputs for this function disables the
%   command line progress display "Executing move..."
%
%   [confirm,PlotHandle] = SCORWAITFORMOVE(___) returns binary confirming
%   success, and the plot handle structured array.
%
%   [confirm,PlotHandle,CollectedData] = SCORWAITFORMOVE(___) returns
%   binary confirming success, the plot handle structured array, and data
%   collected during the move.
%       CollectedData.tXYZPR - Nx6 array containing [timeStamp (sec), XYZPR]
%       CollectedData.tBSEPR - Nx6 array containing [timeStamp (sec), BSEPR]
%
%   Examples:
%
%   Setup for Examples:
%       % Initialize and home ScorBot
%       ScorInit;
%       ScorHome;
%
%   Example 1: ScorWaitForHome without plots or progress display
%       % Define two joint positions
%       BSEPR(1,:) = [0,pi/2,-pi/2,-pi/2,0];
%       BSEPR(2,:) = [0,pi/2,-0.10,-pi/2,0];
%       % Initialize arm configuration
%       ScorSetSpeed(100);
%       ScorSetBSEPR(BSEPR(1,:));
%       ScorWaitForMove;
%
%       % ScorWaitForHome without plots or progress display
%       fprintf('Demonstrating without command line progress display.\n');
%       ScorSetBSEPR(BSEPR(1,:));
%       confirm = ScorWaitForMove;
%
%   Example 2: ScorWaitForHome with progress display
%       % Define two joint positions
%       BSEPR(1,:) = [0,pi/2,-pi/2,-pi/2,0];
%       BSEPR(2,:) = [0,pi/2,-0.10,-pi/2,0];
%       % Initialize arm configuration
%       ScorSetSpeed(100);
%       ScorSetBSEPR(BSEPR(2,:));
%       ScorWaitForMove;
%
%       % ScorWaitForMove with command line progress display
%       fprintf('Demonstrating command line progress display.\n');
%       for i = 1:size(BSEPR,1)
%           ScorSetBSEPR(BSEPR(i,:));
%           ScorWaitForMove;
%       end
%
%       % ScorWaitForMove with command line progress display and sampled
%       %   waypoint
%       fprintf('Demonstrating command line progress display.\n');
%       for i = 1:size(BSEPR,1)
%           ScorSetBSEPR(BSEPR(i,:));
%           ScorWaitForMove;
%           pause(2);         % Pause to allow ScorBot to fully stop
%           q = ScorGetBSEPR; % Sampled waypoint
%           fprintf('q = [%0.3f,%0.3f,%0.3f,%0.3f,%0.3f]\n',q);
%       end
%
%   Example 3: ScorWaitForHome with all plots
%       % Define two joint positions
%       BSEPR(1,:) = [0,pi/2,-pi/2,-pi/2,0];
%       BSEPR(2,:) = [0,pi/2,-0.10,-pi/2,0];
%       % Initialize arm configuration
%       ScorSetSpeed(100);
%       ScorSetBSEPR(BSEPR(2,:));
%       ScorWaitForMove;
%
%       % ScorWaitForMove with XYZPR, BSEPR, and Animation Plots
%       h = []; % initialize variable for plot handle
%       fprintf('Demonstrating XYZPR, BSEPR, and Animation Plots.\n');
%       for i = 1:size(BSEPR,1)
%           ScorSetBSEPR(BSEPR(i,:));
%           [~,h] = ScorWaitForMove(...
%                       'XYZPRPlot','On',...
%                       'BSEPRPlot','On',...
%                       'RobotAnimation','On',...
%                       'PlotHandle',h);
%           pause(1);
%       end
%
%   Example 4: ScorWaitForHome data collection
%       % Define two joint positions
%       BSEPR(1,:) = [0,pi/2,-pi/2,-pi/2,0];
%       BSEPR(2,:) = [0,pi/2,-0.10,-pi/2,0];
%       % Initialize arm configuration
%       ScorSetSpeed(100);
%       ScorSetBSEPR(BSEPR(2,:));
%       ScorWaitForMove;
%
%       % ScorWaitForMove with data collection
%       fprintf('Demonstrating Collect Data functionality:\n');
%       for i = 1:size(BSEPR,1)
%           ScorSetBSEPR(BSEPR(i,:));
%           [~,~,sData(i)] = ScorWaitForMove('CollectData','On');
%       end
%
%   Example 4.1: Combining and plotting data from multiple moves
%       % Combine data
%       tBSEPR = sData(1).tBSEPR;
%       tXYZPR = sData(1).tXYZPR;
%       tBSEPR = [tBSEPR;...
%           bsxfun(@plus,tBSEPR(end,1),sData(2).tBSEPR(:,1)),... % append
time
%           sData(2).tBSEPR(:,2:end)];
%       tXYZPR = [tXYZPR;...
%           bsxfun(@plus,tXYZPR(end,1),sData(2).tXYZPR(:,1)),... % append
time
%           sData(2).tXYZPR(:,2:end)];
%       plotColors = 'rgbmk';
%       % Plot appended BSEPR data
%       fig = figure('Name','Appended BSEPR Plot');
%       axs = axes('Parent',fig);
%       hold on
%       xlabel('Time (s)');
%       ylabel('Angle (Radians)');
%       for i = 1:5
%           plot(axs,tBSEPR(:,1),tBSEPR(:,i+1),plotColors(i));
%       end
%       legend(axs,'Base Angle','Shoulder Angle','Elbow Angle','Wrist
Pitch','Wrist Roll');
%       % Plot appended XYZ data
%       fig = figure('Name','Appended XYZ Plot');
%       axs = axes('Parent',fig);
%       hold on
%       xlabel('Time (s)');
%       ylabel('Position (Millimeters)');
%       for i = 1:3
%           plot(axs,tXYZPR(:,1),tXYZPR(:,i+1),plotColors(i));
%       end
%       legend(axs(1),'x-pos','y-pos','z-pos');
%
%   See also ScorSetMoveTime ScorSetXYZPR ScorSetBSEPR
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function names:
%               "ScorWaitUntilDone.m"
%               "ScorBlockUntilMotionComplete.m"
```

```
%
%   M. Kutzer, 13Aug2015, USNA

% Updates
%   25Aug2015 - Updated to correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   25Aug2015 - Updated RobotAnimation to include previous trajectories in
%               plot.
%   28Aug2015 - Updated help documentation to include a note about 2 second
%               pause that is added when data is collected or if
%               plots/animations are used.
%   28Aug2015 - Updated error handling
%   15Sep2015 - Added timeout to execute quick move, and execute regular
%               move
%   13Oct2015 - Bad property value identification in error
%   20Oct2015 - Updated "quick move" to include a wait for the change in
%               joint angles to drop below a threshold.
%   08Jan2016 - Error fix to set general "showProgress" default
%   08Jan2016 - Error fix to set general "iter" default
%   23Aug2016 - Corrected typo in help documentation.
%   01Sep2016 - Added comments to examples
%   01Sep2016 - Added rapid movement note, "while ScorIsMoving; end"
%   23Sep2023 - Added 'MoveType' 'Instant' to ScorSimSetBSEPR when using
%               "robOn" flag

% Known Issues
%   15Sep2015 - Running ScorWaitForMove immediately following a
%               ScorSetTeachPendant('Auto') when the teach pendant is in
%               'Teach' will result in a "TIMEOUT"

%% Start timer
t_swfm = tic;

%% Check number of outputs
nargoutchk(0,3);

%% Initialize plot handle
h.XYZPRPlot = [];
h.BSEPRPlot = [];
h.RobotAnimation = [];

%% Initialize data structure
CollectedData.tXYZPR = [];
CollectedData.tBSEPR = [];

%% Check for ScorBot Error
isReady = ScorIsReady;
if ~isReady
    if nargout > 0
        varargout{1} = false;
    end
    if nargout > 1
        varargout{2} = h;
    end
    if nargout > 2
```

```
        varargout{3} = CollectedData;
    end
    return
end

%% Initialize flags
posOn = false;   % XYZPR Plot flag
jntOn = false;   % BSEPR Plot flag
robOn = false;   % Robot Animation flag
getData = false; % Collect XYZPR and BSEPR data flag

%% Process inputs
% Check number of inputs
n = nargin;
if n/2 ~= round(n/2)
    error('Inputs must be specified as Property Name, Property Value
pairs.');
end
% Parse inputs
for i = 1:2:n
    switch lower(varargin{i})
        case 'xyzprplot'
            switch lower(varargin{i+1})
                case 'on'
                    posOn = true;
                case 'off'
                    posOn = false;
                otherwise
                    error('Unexpected property value for "%s".',varargin{i});
            end
        case 'bseprplot'
            switch lower(varargin{i+1})
                case 'on'
                    jntOn = true;
                case 'off'
                    jntOn = false;
                otherwise
                    error('Unexpected property value for "%s".',varargin{i});
            end
        case 'robotanimation'
            switch lower(varargin{i+1})
                case 'on'
                    robOn = true;
                case 'off'
                    robOn = false;
                otherwise
                    error('Unexpected property value for "%s".',varargin{i});
            end
        case 'plothandle'
            h = varargin{i+1};
        case 'collectdata'
            switch lower(varargin{i+1})
                case 'on'
                    getData = true;
                case 'off'
                    getData = false;
                otherwise
                    error('Unexpected property value for "%s".',varargin{i});
            end
        otherwise
            error(sprintf('Unexpected Property Name "%s".',varargin{i}));
    end
end

%% Execute quick move when no data or plots are required
iter = 0; % set default value
showProgress = false; % set default value
if ~posOn && ~jntOn && ~robOn && ~getData
    % Show progress in command window if no outputs are declared
    if nargout == 0
        showProgress = true;
    else
        showProgress = false;
    end
    if showProgress
        fprintf('Executing move...');
    end
    confirm = true;
    iter = 0;
    BSEPR(1,:) = ScorGetBSEPR;
    while ScorIsMoving
        % Update current joint state
        BSEPR(2,:) = ScorGetBSEPR;
        % Check movement
        if size(BSEPR,1) > 1 && toc(t_swfm) > 5
            dBSEPR = diff(BSEPR((end-1:end),:),1,1);
            if norm(dBSEPR) < 1e-8
                [isReady,~,errStruct] = ScorIsReady;
                if ~isReady
                    if showProgress
                        fprintf( char(repmat(8,1,mod(iter,4))) );
                        fprintf('...');
                        fprintf('FAILED\n');
                        ScorDispError(errStruct);
                    else
                        ScorDispError(errStruct);
                    end
                    confirm = false;
                    % Package output
                    if nargout > 0
                        varargout{1} = confirm;
                    end
                    if nargout > 1
                        varargout{2} = h;
                    end
                    if nargout > 2
                        varargout{3} = CollectedData;
                    end
                    return
                else
                    if toc(t_swfm) > 8
                        if showProgress
```

```
                            fprintf( char(repmat(8,1,mod(iter,4))) );
                            fprintf('...');
                            fprintf('TIMEOUT\n');
                            ScorDispError(errStruct);
                            return
                        end
                    end
                end
            end
        end
        % Show progress in command window
        if showProgress
            if mod(iter,4) == 0
                fprintf(char([8,8,8]));
            else
                fprintf('.');
            end
            iter = iter+1;
        end
        if showProgress
            pause(0.10);
        else
            pause(0.01);
        end
        % Update initial joint state
        BSEPR(1,:) = BSEPR(2,:);
    end
    if showProgress
        fprintf( char(repmat(8,1,mod(iter-1,4))) );
        fprintf('...');
        fprintf('SUCCESS\n');
    end
    % Package output
    if nargout > 0
        varargout{1} = confirm;
    end
    if nargout > 1
        varargout{2} = h;
    end
    if nargout > 2
        varargout{3} = CollectedData;
    end
    % Wait for move to actually complete
    dq = inf(1,5);
    q(1,:) = ScorGetBSEPR;
    while norm(dq) > 1e-8
        pause(0.05);
        q(2,:) = ScorGetBSEPR;
        dq = diff(q,1);
        q(1,:) = q(2,:);
    end
    return
end

%% Create figures
% check plot handle structure fields
```

```
if ~isfield(h,'XYZPRPlot')
    h.XYZPRPlot = [];
end
if ~isfield(h,'BSEPRPlot')
    h.BSEPRPlot = [];
end
if ~isfield(h,'RobotAnimation')
    h.RobotAnimation = [];
else
    wipeRob = false;
    if ~isfield(h.RobotAnimation,'Sim')
        wipeRob = true;
    else
        if ~isfield(h.RobotAnimation.Sim,'Joints')
            wipeRob = true;
        end
    end
    if ~isfield(h.RobotAnimation,'Plot')
        wipeRob = true;
    end
    if ~isfield(h.RobotAnimation,'Waypoint')
        wipeRob = true;
    end
    if wipeRob
        h.RobotAnimation = [];
    end
end

% check plot handles
for i = 1:5
    if ~isempty(h.XYZPRPlot)
        if ~ishandle(h.XYZPRPlot(i))
            h.XYZPRPlot = [];
        end
    end
    if ~isempty(h.BSEPRPlot)
        if ~ishandle(h.BSEPRPlot(i))
            h.BSEPRPlot = [];
        end
    end
    if ~isempty(h.RobotAnimation)
        if ~ishandle(h.RobotAnimation.Sim.Joints(i)) ||
~ishandle(h.RobotAnimation.Plot)
            h.RobotAnimation = [];
        end
    end
end
% create XYZPR figure
if posOn && isempty(h.XYZPRPlot)
    fig = figure('Name','XYZPR Data');
    set(fig,'Units','Normalized',...
        'Position',[0.0036,0.5200,2/3-0.008,0.4000],...
        'NumberTitle','Off');
    axs(1) = subplot(1,2,1,'Parent',fig);
    hold on
    axs(2) = subplot(1,2,2,'Parent',fig);
    hold on
    h.XYZPRPlot(1) = plot(axs(1),0,0,'r');
    h.XYZPRPlot(2) = plot(axs(1),0,0,'g');
    h.XYZPRPlot(3) = plot(axs(1),0,0,'b');
    h.XYZPRPlot(4) = plot(axs(2),0,0,'c');
    h.XYZPRPlot(5) = plot(axs(2),0,0,'k');
    legend(axs(1),'x-pos','y-pos','z-pos');
    legend(axs(2),'pitch','roll');
    xlabel(axs(1),'Time (s)');
    ylabel(axs(1),'Position (millimeters)');
    xlabel(axs(2),'Time (s)');
    ylabel(axs(2),'Angle (radians)');
end
% create BSEPR figure
if jntOn && isempty(h.BSEPRPlot)
    fig = figure('Name','BSEPR Data');
    set(fig,'Units','Normalized',...
        'Position',[2/3+0.005,0.5200,1/3-0.01,0.4000],...
        'NumberTitle','Off');
    axs = axes('Parent',fig);
    hold on
    h.BSEPRPlot(1) = plot(axs,0,0,'r');
    h.BSEPRPlot(2) = plot(axs,0,0,'g');
    h.BSEPRPlot(3) = plot(axs,0,0,'b');
    h.BSEPRPlot(4) = plot(axs,0,0,'c');
    h.BSEPRPlot(5) = plot(axs,0,0,'k');
    legend(axs,'Base Angle','Shoulder Angle','Elbow Angle','Wrist
Pitch','Wrist Roll');
    xlabel(axs,'Time (s)');
    ylabel(axs,'Angle (radians)');
end
% create robot animation figure
if robOn && isempty(h.RobotAnimation)
    h.RobotAnimation.Sim = ScorSimInit;
    h.RobotAnimation.Plot = plot3(h.RobotAnimation.Sim.Axes,0,0,0,'m');
    h.RobotAnimation.Waypoint = [];
end

%% Wait for motion
confirm = true;
posT = [];
BSEPR = [];
jntT = [];
XYZPR = [];
newWaypoint = true;
while ScorIsMoving
    % Get XYZPR
    posT(end+1) = toc(t_swfm);
    tmp = ScorGetXYZPR;
    if ~isempty(tmp)
        XYZPR(end+1,:) = tmp;
    else
        posT(end) = [];
    end
    % Get BSEPR
    jntT(end+1) = toc(t_swfm);
    tmp = ScorGetBSEPR;
    if ~isempty(tmp)
        BSEPR(end+1,:) = tmp;
    else
        jntT(end) = [];
    end
    % Update BSEPR and XYZPR Plots
    if ~isempty(BSEPR) && ~isempty(XYZPR)
        for i = 1:5
            if posOn

set(h.XYZPRPlot(i),'xData',posT,'yData',transpose(XYZPR(:,i)));
            end
            if jntOn

set(h.BSEPRPlot(i),'xData',jntT,'yData',transpose(BSEPR(:,i)));
            end
            if robOn
                ScorSimSetBSEPR(h.RobotAnimation.Sim,BSEPR(end,:),...
                    'MoveType','Instant');
                % Plot starting waypoint
                if size(XYZPR,1) == 1 && newWaypoint
                    h.RobotAnimation.Waypoint(end+1) = ...
                        plot3(h.RobotAnimation.Sim.Axes,...
                        XYZPR(1,1),XYZPR(1,2),XYZPR(1,3),'og');
                    newWaypoint = false;
                end
                % Get previous plot data
                xData = get(h.RobotAnimation.Plot,'xData');
                yData = get(h.RobotAnimation.Plot,'yData');
                zData = get(h.RobotAnimation.Plot,'zData');
                % Remove initialized point
                if numel(xData) == 1
                    if xData == 0 && yData == 0 && zData == 0
                        xData = [];
                        yData = [];
                        zData = [];
                    end
                end
                % Append new data
                xData(end+1) = XYZPR(end,1);
                yData(end+1) = XYZPR(end,2);
                zData(end+1) = XYZPR(end,3);
                % Update trajectory plot
                set(h.RobotAnimation.Plot,...
                    'xData',xData,...
                    'yData',yData,...
                    'zData',zData);
            end
        end
    end
    % Drawnow
    if posOn || jntOn || robOn
        drawnow
    end
    % Check movement
    if size(BSEPR,1) > 1 && toc(t_swfm) > 5
        dBSEPR = diff(BSEPR((end-1:end),:),1,1);
        if norm(dBSEPR) < 1e-8
            [isReady,~,errStruct] = ScorIsReady;
            if ~isReady
                ScorDispError(errStruct);
                confirm = false;
                break
            else
                if toc(t_swfm) > 8
                    if showProgress
                        fprintf( char(repmat(8,1,mod(iter,4))) );
                        fprintf('...');
                        fprintf('TIMEOUT\n');
                        ScorDispError(errStruct);
                        return
                    end
                end
            end
        end
    end
end

%% Add final points to collected data and plots
t_dwell = 1.2; % dwell time following ~ScorIsMoving
t_dwell_strt = toc(t_swfm);
dBSEPR = ones(1,5);
while norm(dBSEPR) > 1e-8 || (toc(t_swfm)-t_dwell_strt) < t_dwell
    if size(BSEPR,1) > 1
        dBSEPR = diff(BSEPR((end-1:end),:),1,1);
    end
    % Get XYZPR
    posT(end+1) = toc(t_swfm);
    tmp = ScorGetXYZPR;
    if ~isempty(tmp)
        XYZPR(end+1,:) = tmp;
    else
        posT(end) = [];
    end
    % Get BSEPR
    jntT(end+1) = toc(t_swfm);
    tmp = ScorGetBSEPR;
    if ~isempty(tmp)
        BSEPR(end+1,:) = tmp;
    else
        jntT(end) = [];
    end
    % Update BSEPR and XYZPR Plots
    if ~isempty(BSEPR) && ~isempty(XYZPR)
        for i = 1:5
            if posOn

set(h.XYZPRPlot(i),'xData',posT,'yData',transpose(XYZPR(:,i)));
            end
            if jntOn
```

```
set(h.BSEPRPlot(i),'xData',jntT,'yData',transpose(BSEPR(:,i)));
            end
            if robOn
                ScorSimSetBSEPR(h.RobotAnimation.Sim,BSEPR(end,:),...
                    'MoveType','Instant');
                % Get previous plot data
                xData = get(h.RobotAnimation.Plot,'xData');
                yData = get(h.RobotAnimation.Plot,'yData');
                zData = get(h.RobotAnimation.Plot,'zData');
                % Remove initialized point
                if numel(xData) == 1
                    if xData == 0 && yData == 0 && zData == 0
                        xData = [];
                        yData = [];
                        zData = [];
                    end
                end
                % Append new data
                xData(end+1) = XYZPR(end,1);
                yData(end+1) = XYZPR(end,2);
                zData(end+1) = XYZPR(end,3);
                % Update trajectory plot
                set(h.RobotAnimation.Plot,...
                    'xData',xData,...
                    'yData',yData,...
                    'zData',zData);
            end
        end
    end
    % Drawnow
    if posOn || jntOn || robOn
        drawnow
    end
end

% Add final waypoint
if robOn
    % Plot final waypoint
    if size(XYZPR,1) >= 1
        h.RobotAnimation.Waypoint(end+1) = ...
            plot3(h.RobotAnimation.Sim.Axes,...
            XYZPR(end,1),XYZPR(end,2),XYZPR(end,3),'xr');
        drawnow
    end
end

%% Package data
if getData
    CollectedData.tXYZPR = [transpose(posT),XYZPR];
    CollectedData.tBSEPR = [transpose(jntT),BSEPR];
end

%% Package output
if nargout > 0
    varargout{1} = confirm;
```

```
end
if nargout > 1
    varargout{2} = h;
end
if nargout > 2
    varargout{3} = CollectedData;
end
```

7 ScorSetXYZPR().

```
function confirm = ScorSetXYZPR(varargin)
% SCORSETXYZPR moves the ScorBot end-effector to a specified x,y,z
% position, and pitch and roll orientation.
%   SCORSETXYZPR(XYZPR) moves the ScorBot to the 5-element task-space
%   vector containing the end-effector x,y,z position, and end-effector
%   pitch and roll orientation.
%       XYZPR - 5-element vector containing end-effector position and
%       orientation.
%           XYZPR(1) - end-effector x-position in millimeters
%           XYZPR(2) - end-effector y-position in millimeters
%           XYZPR(3) - end-effector z-position in millimeters
%           XYZPR(4) - end-effector pitch in radians
%           XYZPR(5) - end-effector roll in radians
%
%   SCORSETXYZPR(...,'MoveType',mode) specifies whether the movement is
%   linear in task space or linear in joint space.
%       Mode: {['LinearTask'] 'LinearJoint'}
%
%   confirm = SCORSETXYZPR(___) returns 1 if successful and 0 otherwise.
%
%   Note: Wrist pitch angle of BSEPR does not equal the pitch angle of
%   XYZPR. BSEPR pitch angle is body-fixed while the pitch angle of XYZPR
%   is calculated relative to the base.
%
%   See also ScorGetXYZPR ScorSetBSEPR ScorGetBSEPR
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function name "ScorCartMove.m"
%
%   M. Kutzer, 10Aug2015, USNA

% Updates
%   25Aug2015 - Updated correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   28Aug2015 - Updated error handling
%   23Dec2015 - Updated to clarify errors.
%   25Aug2016 - Updated to correct default movement type.
%   28Sep2018 - Updated to include error logging
```

```
%% Set global for ScorSetUndo
global ScorSetUndoBSEPR

%% Check inputs
% This assumes nargin is fixed to 1 or 3 with a set of common errors:
%   e.g. ScorSetXYZPR(X,Y,Z,Pitch,Roll);

% Check for zero inputs
if nargin < 1
    error('ScorSet:NoXYZPR',...
        ['End-effector position and orientation must be specified.',...
        '\n\t-> Use "ScorSetXYZPR(XYZPR)".']);
end
% Check XYZPR
if nargin >= 1
    XYZPR = varargin{1};
    if ~isnumeric(XYZPR) || numel(XYZPR) ~= 5
        error('ScorSet:BadXYZPR',...
            ['End-effector position and orientation must be specified as a 5-
element numeric array.',...
            '\n\t-> Use "ScorSetXYZPR([X,Y,Z,Pitch,Roll])".']);
    end
end
% Check property designator
if nargin >= 2
    pType = varargin{2};
    if ~ischar(pType) || ~strcmpi('MoveType',pType)
        error('ScorSet:BadPropDes',...
            ['Unexpected property: "%s"',...
            '\n\t-> Use "ScorSetXYZPR(XYZPR,''MoveType'',''LinearJoint'')"
or',...
            '\n\t-> Use
"ScorSetXYZPR(XYZPR,''MoveType'',''LinearTask'')".'],pType);
    end
    if nargin < 3
        error('ScorSet:NoPropVal',...
            ['No property value for "%s" specified.',...
            '\n\t-> Use "ScorSetXYZPR(XYZPR,''MoveType'',''LinearJoint'')"
or',...
            '\n\t-> Use
"ScorSetXYZPR(XYZPR,''MoveType'',''LinearTask'')".'],pType);
    end
end
% Check property value
mType = 'LinearTask';
if nargin >= 3
    mType = varargin{3};
    switch lower(mType)
        case 'linearjoint'
            % Linear move in joint space
        case 'lineartask'
            % Linear move in task space
        otherwise
            error('ScorSet:BadPropVal',...
                ['Unexpected property value: "%s".',...
                '\n\t-> Use
"ScorSetXYZPR(XYZPR,''MoveType'',''LinearJoint'')" or',...
                '\n\t-> Use
"ScorSetXYZPR(XYZPR,''MoveType'',''LinearTask'')".'],mType);
    end
end
% Check for too many inputs
if nargin > 3
    warning('Too many inputs specified. Ignoring additional parameters.');
end

%% Check for very small movement(s)
q0 = ScorGetBSEPR;
q1 = ScorXYZPR2BSEPR(XYZPR);

degLIM = 0.25;
dq = abs( q1 - q0 );
if max(dq) < deg2rad(degLIM)
    fprintf('Movement is less than %.2f degrees.\n',degLIM);
    confirm = 1;
    return
end

%% Set point
isSet = ScorSetPoint(XYZPR,'Mode','Absolute');
if ~isSet
    confirm = false;
    return
end

%% Set the ScorSetUndo waypoint
% TODO - add error checking
ScorSetUndoBSEPR = ScorGetBSEPR;

%% Goto point
isMove = ScorGotoPoint('MoveType',mType);
if isMove
    % Write movement command to error log
    BSEPR = ScorXYZPR2BSEPR(XYZPR);
    ScorErrorLogWrite(0,mType,BSEPR);
    % Set confirm flag
    confirm = true;
    return
else
    confirm = false;
    return
end
```

8 ScorGoHome().

```
function confirm = ScorGoHome()
% SCORGOHOME moves ScorBot to the home position
%   SCORGOHOME moves ScorBot to the home position. This function does not
%   run the homing sequence.
%
%   confirm = SCORGOHOME(___) returns 1 if successful and 0 otherwise.
%
%   See also ScorHome
%
%   M. Kutzer, 10Aug2015, USNA

%% ScorBot XYZPR home position [mm, mm, mm, rad, rad]
XYZPRhome = [169.300,0.000,504.328,-1.10912,0.00000];

%% Go home
fprintf('Moving ScorBot to home position...');
isMovedHome = ScorSetXYZPR(XYZPRhome,'MoveType','LinearJoint');
if isMovedHome
    isComplete = ScorWaitForMove;
else
    isComplete = false;
end

%% Check possible errors
if ~isMovedHome || ~isComplete
    fprintf('FAILED\n');
    confirm = false;
    if nargout == 0
        warning('ScorBot failed to move to the home position.');
    end
    return
else
    fprintf('SUCCESS\n');
    confirm = true;
end
```

9 ScorSafeShutdown().

```
function confirm = ScorSafeShutdown()
% SCORSAFESHUTDOWN runs all processes to safely shutdown ScorBot.
%   SCORSAFESHUTDOWN moves the ScorBot to the home position, disables
%   control, and unloads the ScorBot library. This function should be run
%   prior to closing MATLAB.
%
%   confirm = SCORSAFESHUTDOWN(___) returns 1 if successful and 0
%   otherwise.
%
%   See also ScorInit ScorHome
%
%   References:
%       [1] C. Wick, J. Esposito, & K. Knowles, US Naval Academy, 2010
%           http://www.usna.edu/Users/weapsys/esposito-
old/_files/scorbot.matlab/MTIS.zip
%           Original function name "ScorSafeShutdown.m"
```

```
%
%   C. Wick, J. Esposito, K. Knowles, & M. Kutzer, 10Aug2015, USNA
%
%   J. Donnal, 28Jun2017, USNA (64-bit Support)

% Updates
%   25Aug2015 - Updated correct help documentation, "J. Esposito K.
%               Knowles," to "J. Esposito, & K. Knowles,"
%               Erik Hoss
%   25Aug2015 - Updated to make unload library "SUCCESS" occur after
%               library is actually unloaded.
%   28Aug2015 - Updated to delete ScorGetSpeed/ScorGetMoveTime file
%   31Aug2015 - Updated to delete ScorGetControl file
%   31Aug2015 - Updated to reorder turning off digital out and deleting the
%               ScorGetControl file
%   01Sep2015 - Added ScorWaitForMove prior to entering shutdown
%   15Sep2015 - Added ScorSetPendantMode('Auto') prior to ScorWaitForMove.
%   28Nov2017 - Updated to override ScorHome prompt
%   25Sep2018 - Updates to include error logging
%   05Oct2018 - Updated to account for safe shutdown if/when library is not
%               loaded.
%   17Jul2019 - Updated to replace instances of "calllib.m" with
%               "ScorCallLib.m" to include J. Donnal 64-bit solution
%   17Jul2019 - Updated to differentiate between 32-bit and 64-bit
%               initialization

persistent osbits

%% Check operating system
if isempty(osbits)
    switch computer
        case 'PCWIN'
            % 32-bit OS
            osbits = 32;
        case 'PCWIN64'
            % 64-bit OS
            osbits = 64;
    end
end

%% Check if ScorBot is initialized
[~,libname,~] = ScorIsReady;
switch osbits
    case 32
        isLoaded = libisloaded(libname);
        if ~isLoaded
            confirm = false;
            % Error copied from ScorIsReady
            errStruct.Code       = NaN;
            errStruct.Message    = sprintf('TOOLBOX: The ScorBot library "%s"
has not been loaded.',libname);
            errStruct.Mitigation = sprintf('Run "ScorInit" to intialize
ScorBot.');
            ScorDispError(errStruct);
            return
        end
    case 64
        isRunning = ScorServerIsRunning;
        if ~isRunning
            confirm = false;
            % Error copied from ScorIsReady
            errStruct.Code       = NaN;
            errStruct.Message    = sprintf('TOOLBOX: The ScorBot server "%s" 
is not running.','ScorbotServer.exe');
            errStruct.Mitigation = sprintf('Run "ScorInit" to intialize 
ScorBot');
            errStruct.QuickFix   = sprintf('ScorInit;');
            ScorDispError(errStruct);
            return
        end
    otherwise
        error('OSBITS variable not set to known value.');
end

%% Define shutdown figure handle
ShutdownFig = 1845;

%% Setup confirmation flags array
confirm = [];

%% Close and send error log
ScorErrorLogClose;

%% Set Teach Pendant to Auto Mode
isAuto = ScorSetPendantMode('Auto');
if ~isAuto
    confirm(end+1) = false;
    warning('ScorBot must be in Auto mode during shutdown.');
end

%% Wait for any/all previous moves
ScorWaitForMove;

%% Check ScorBot and define library alias
[isReady,libname] = ScorIsReady;
if ~isReady
    ScorSetControl('On');
end

%% Open/close gripper
%ScorSetGripper('Open');
%ScorWaitForMove;
%ScorSetGripper('Close');
%ScorWaitForMove;

%% Home ScorBot
ScorSetSpeed(50);
isMovedHome = ScorGoHome;
ScorWaitForMove;
pause(2);
if ~isMovedHome
    isHome = ScorHome(true);
    if ~isHome
        confirm(end+1) = false;
        warning('Unable to home ScorBot.');
    else
        confirm(end+1) = true;
    end
else
    confirm(end+1) = true;
end

%% Move gripper to neutral state
%ScorSetGripper(35);
%ScorWaitForMove;

%% Delete speed and move time file
ScorGetSpeed('DeleteSpeed');

%% Turn off digital outputs
fprintf('Turning off digital outputs...');
isDig = ScorSetDigitalOutput(zeros(1,8));
if ~isDig
    fprintf('FAILED\n');
    confirm(end+1) = false;
    warning('Unable to set ScorBot digital outputs to "Off".');
else
    fprintf('SUCCESS\n');
    confirm(end+1) = true;
end

%% Disable robot
fprintf('Setting ScorBot Control Mode to "Off"...');
isOff = ScorSetControl('Off');
if ~isOff
    fprintf('FAILED\n');
    confirm(end+1) = false;
    warning('Unable to set ScorBot Control Mode to "Off".');
else
    fprintf('SUCCESS\n');
    confirm(end+1) = true;
end

%% Delete control file
ScorGetControl('DeleteControl');

%% Unload library
switch osbits
    case 32
        fprintf('Unloading "%s" library...',libname);
        try
            unloadlibrary(libname);
            fprintf('SUCCESS\n');
            confirm(end+1) = true;
            fprintf('ScorBot library has been unloaded.\n')
            fprintf('\tRun "ScorInit" and "ScorHome" to continue using
ScorBot\n');
        catch
```

```
            fprintf('FAILED\n');
            confirm(end+1) = false;
            warning('Unable to unload "%s".',libname);
        end
    case 64
        fprintf('Terminating the ScorBot server...');
        try
            ScorServerStop;
            fprintf('SUCCESS\n');
            confirm(end+1) = true;
            fprintf('ScorBot server has been terminated.\n')
            fprintf('\tRun "ScorInit" and "ScorHome" to continue using
ScorBot\n');
        catch
            confirm = false;
            fprintf('FAILED\n');
            warning('Server did not terminate successfully.');
            return
        end
    otherwise

end

%% Define final confirmation state
confirm = min(confirm);

%% Delete "ShutdownFig"
if ishandle(ShutdownFig)
    delete(ShutdownFig);
end
```

10 deg2rad().

```
function angleInRadians = deg2rad(angleInDegrees)
% DEG2RAD Convert angles from degrees to radians.
%   DEG2RAD(X) converts angle units from degrees to radians for each
%   element of X.
%
%   See also RAD2DEG.

% Copyright 2015 The MathWorks, Inc.

if isfloat(angleInDegrees)
    angleInRadians = (pi/180) * angleInDegrees;
else
    error(message('MATLAB:deg2rad:nonFloatInput'))
end
```