RUNNING THE PUMA

The basic work flow when using the PUMA 260 arm is

1. Make sure the Emergency Stop (E-stop) button is engaged (pressed down).
2. Call `pumaStart('Hardware', 'on', 'Delay', 10)`, where the number following delay is the minimum allowable time, in milliseconds, between calls to `pumaServo`. This may be set to any value above 0.5ms. This will display a warning that the PUMA will return to the home position.
3. Ensure that the workspace is clear and manually move the PUMA close to the home position. You should especially avoid situations where the PUMA may hit the table or an object.
4. Type 'y' or 'yes' then hit Enter to continue.
5. Release the E-stop by pulling up on the button, at which time the PUMA will return to the home position.
6. Call `startFrameBuffer` to begin capturing video from the webcam.
7. Make a light painting, using `pumaServo` to command the robot to move. Remember to call `pumaLEDOn` to enable the LED and use `pumaLEDSet` to select the color.
8. Call `stopFrameBuffer` to finish capturing video.
9. Return the PUMA to the home position, if possible.
10. Call `pumaStop` to disable the controller.
11. Engage the E-stop by pressing the button down.
12. Use `makeVideoAndImage` to create a long-exposure picture and time-lapse video from the images you just captured. Optionally, you may wish to save the image files to a different location using `saveImagesToFolder` before starting a new video.

We recommend you test just the video capture system before running your whole light painting.

GENERAL PUMA FUNCTIONS

- **pumaStart('PropertyName', PropertyValue);**
  - Optional Inputs { DefaultValue }
    - 'Hardware' 'on' | {'off'} Toggle between simulation and using the actual hardware. Be careful to read the warning below before enabling hardware. If hardware is enabled then there is no visualization.
    - 'Delay' {0 (simulation), 10 (hardware)} Sets the minimum delay, in ms, between calls to the pumaServo function. In hardware mode, the minimum allowable delay is 0.5ms.
    - Simulation only options
      - 'Joints' {'on'} | 'off' Display the joints of the robot.
      - 'Jointaxis' 'on' | {'off'} Display the joint axes of the robot
      - 'Shadow' {'on'} | 'off' Plot the shadow of the robot on the ground plane.
      - 'Tool' {'on'} | 'off' Display the tool on the end effector
      - 'LEDMarkerStyle' {'*'} Set the style of the LED marker. See 'help linespec' for available styles.
      - 'LEDMarkerSize' {6} Set the size of the LED marker.
      - 'PlotEveryNFrames' {1} Decrease the number of times the PUMA is plotted in order to speed up the simulation. Must be an integer value. Use the function pumaRefreshPlot to update the visualization, bypassing the frame count check.
  - This function initializes the puma data structure along with the visualization (simulation only), controller (hardware only), and disengages the electromechanical brakes (hardware only). In both cases, the PUMA is set to the home position (note that this is **not** the same as the zero position).
  - WARNING: Be aware of the PUMA's position before calling this function. Remove any objects in the path between the current position and the home position. If the PUMA is likely to slam into the table surface or other immovable object, manually move the PUMA close to the home position before using this command. Ask a TA for help if you don't know how to do this.

- **pumaStop;**
  - In simulation, clears out any LED markers from the visualization. In hardware, stops the PUMA's controller, engages the electromechanical brakes, and shuts off the LED end effector. Please move the PUMA to the home position before using this command if possible. In both cases, pumaLEDOff is called.
- **[angles] = pumaAngles();**
  - Outputs: a 6x1 matrix of the current joint angles in radians.
- **pumaServo(theta1, theta2, theta3, theta4, theta5, theta6);**
  - Inputs: Joint angles as measured in radians. This can be 6 scalar inputs or a single 6 element vector.
  - Commands the PUMA's joints to the supplied angles. The difference between each new joint angle and the previous angle commanded for that joint must be no greater than 0.1 rad, and the commanded joint velocities must be less than 1 rad/sec. This function cannot be called again until time equal to the delay set in pumaStart has elapsed. Default values are 0ms for simulation and 10ms for hardware.
- **pumaMove(theta1, theta2, theta3, theta4, theta5, theta6);**
  - Inputs: Joint angles as measured in radians. This can be 6 scalar inputs or a single 6 element vector.
  - Simulation only. This functions the same way as pumaServo without the timing, angle, or angular velocity limits.

PUMA LED FUNCTIONS

- **pumaLEDOn;**
  - Turns on the controller for the LED end effector.
- **pumaLEDOff;**
  - Turns off the controller for the LED end effector.
- **pumaLEDSet(redValue, greenValue, blueValue);**
  - Inputs: Red, Green, and Blue color channels for the LED. This can be three scalar inputs or a single three-element vector.
  - Sets the color of the LED end effector to the RGB value specified in the input. Values are from 0 to 1. Setting all three values to 1 yields a white light, while setting them all to 0 turns the LED off completely. The pumaLEDOn function must already have been called for the color to take effect.
- **pumaRefreshPlot;**
  - In simulation, automatically updates the visualization, bypassing the frame count check when using the 'PlotEveryNFrames' option.

PUMA WEBCAM FUNCTIONS

- **startFrameBuffer**
  - Clears out any previous images in the image buffer then turns on the webcam and starts recording images to a buffer at 15 frames per second.
- **stopFrameBuffer;**
  - Turns off the webcam.
- **img = get_frame;**
  - Returns the most recent image in the buffer as an image matrix. This may be used for custom frame capturing if you want anything more complicated than startFrameBuffer.
- **testWebcam;**
  - Displays the most recent image from the frame buffer. Use this to test the positioning of the camera before you run your code.
- **makeVideoAndImage('aviName', 'imageName');**
  - Inputs: 'aviName' is the desired filename for the movie. 'imageName' is the desired filename for the long-exposure image.
  - Takes the contents of the buffer and creates a video and long exposure image. Files will be named with user supplied filenames. If a file with the same name already exists in the current directory, it will be replaced. Also includes instructions for compressing the video files, which MATLAB does not do automatically.
- **saveImagesToFolder('folderName');**
  - Inputs: 'folderName' is the desired folder to save the frames from the buffer to. Default is 'PUMArtImages'.
  - Saves the collected frames from the buffer (stored in /Desktop/vid) to a desired folder. This folder is automatically placed on the desktop. Note that calling the function twice with the same folder name will overwrite all previous data. Remember to save data to an external source if you want to keep it.