Project 1: Part 2
Simulated Light Painting with the PUMA 260

MEAM 520, University of Pennsylvania
Katherine J. Kuchenbecker, Ph.D.

October 21, 2014

This project component is due on **Tuesday, October 28, by midnight (11:59:59 p.m.)**, which is also the deadline for the final version of Part 1 of this project. Your code should be submitted on Canvas according to the instructions at the end of this document.

Late submissions will be accepted after these deadlines, but they will be penalized by 10% for each partial or full day late, with a maximum of 30%. Because our midterm is on Thursday, October 30, neither Wednesday nor Thursday of that week will not count against the lateness tally; assignments submitted Wednesday 10/29, Thursday 10/30, or Friday, 10/31, will all be penalized 10%. The ultimate late deadline (30% late penalty) for this part of the project is midnight on Sunday, November 2. After the ultimate late deadline, no further assignments may be submitted; post a private message on Piazza to request an extension if you need one due to a special situation such as illness.

You may talk with other students about this assignment, ask the teaching team questions, use a calculator and other tools, and consult outside sources such as the Internet. To help you actually learn the material, what you write down must be your team's own work, not copied from any other student, team, or source. Any submissions suspected of violating Penn's Code of Academic Integrity will be reported to the Office of Student Conduct. If you get stuck, post a question on Piazza or go to office hours!

Teamwork

You should work closely with your Project 1 teammates throughout this assignment. You will turn in one set of MATLAB files for which you are all jointly responsible, and you will receive the same baseline grade. Please follow the pair programming guidelines that have been shared before. After the project is over, we will administer a simple survey that asks each student to rate how well each teammate (including him/herself) contributed to the project; when teammates agree that the workload was not shared evenly, individual grades will be adjusted accordingly.

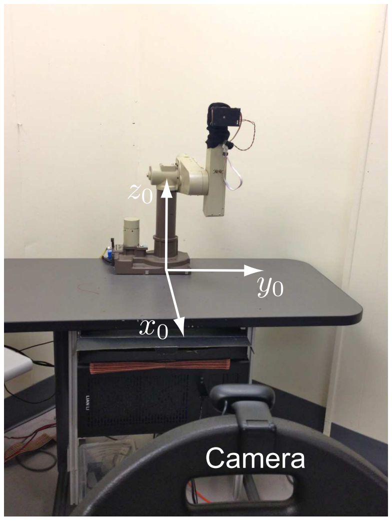
Light Painting

Project 1 is PUMA Light Painting. Each team of three students will write MATLAB code to make our PUMA 260 robot draw something interesting in the air with a colored light, which we will capture by taking a long-exposure photograph. Drawing precise, arbitrary shapes with a robot requires you to solve the robot's full inverse kinematics (IK), so that was the first component of this project. This is the second part – using your inverse kinematics to create the actual light painting in simulation. The third part will be recording your light painting with the actual robot.

Task Overview

Your task is to write two MATLAB functions that can be combined with your inverse kinematics function to enable our PUMA robot to create an original light painting. The first function defines the painting; it needs to be able to calculate the position, orientation, and color that the robot's light-emitting diode (LED) should have at any given time during the performance of the painting. At each time step, this position and orientation will be passed into your IK code, which will return sets of joint angles that will put the robot's LED in the desired pose. The identified IK solutions will then be passed into the second function, which will select the best solution from the options found, based both on the characteristics of the PUMA 260 robot and on the robot's current configuration. The robot is commanded to move to these selected joint angles with the specified LED color, and the cycle begins again. You will develop your light painting with a **simulated version of the PUMA**; after you turn in a successful simulated light painting, the teaching team will help you record your final performance with the real robot. If all goes well, the robot's painting will be captured in a long-exposure photograph for you and others to enjoy.

Inverse Kinematics This part of the project necessarily depends on the first project component, wherein you programmed the inverse kinematics for the PUMA 260. Some teams probably were not able to get all of the left-arm IK solutions fully working by the draft deadline. Others may discover problems within their IK as they program this part of the project. Thus, **all teams must submit the final version of their inverse kinematics code (team1XX_puma_ik.m)**, along with any subsidiary functions, in parallel with the submission of this component of the project. You must do this submission even if your IK code has not changed since the draft submission. Furthermore, you should not include your team's IK code with your submission for this project component.



Position and Orientation of the Camera and the LED The painting that you create may be either two-dimensional or three-dimensional. All properties of the PUMA 260 robot were defined in the first component of this project. As pictured in the photo at right, the camera will be looking in the negative x_0 direction with positive z_0 up and positive y_0 to the right. Thus, two-dimensional artwork should be drawn in a plane parallel to the y_0-z_0 plane. The PUMA's tricolor LED is oriented in the positive z_6 direction; you should generally make it point in the positive x_0 direction so that it can be seen by the camera. Rather than putting your painting at an arbitrary location, you should think carefully about the PUMA's abilities and put your painting in an area of space that is easy for the robot to reach. In particular, you should probably avoid requiring transitions between several different arm configurations.

Task Specifics

We are providing a zip file of starter code to help you accomplish this task. Please download **puma_paint.zip** from our Piazza Course Page under Project Resources. As you are working on this project, please report any bugs, typos, or confusing behavior that you discover in the starter code by posting on Piazza. Corrections, clarifications, and/or new versions will be posted and announced as needed.

Each important file in the starter code is described below. Files that we anticipate you will modify are purposefully named to follow the pattern of **team100_filename.m**. Please change all of these to include your three-digit team number instead of **100**. You will also need to change some function declarations (first line of a function's own file) and some function calls inside the files themselves. Please comment your code so that it is easy to follow, and feel free to create additional custom functions if you need them; all files that you submit must begin with **team1XX_**, where **1XX** is your team number.

team100_get_poc.m This function defines the geometry and coloration of your light painting. Modify this function so that it **can calculate the position, orientation, and color (poc) for the PUMA's LED at any specified point in time**. This function takes in only the present time (`t`) in seconds. The light painting begins at t = 0, when the robot must be in the home pose (as shown in the photograph above: all joint angles equal to zero except θ_5, which must be at $-\pi/2$ radians).

When called without a value for `t`, this function initializes its internal variables and returns only its first output (`duration`). This function must be initialized in this way before it can be used.

This function was designed to return many items. The first output (`duration`) is the total duration of this light painting, in seconds. The calling function needs this information so it can tell when to stop painting. The duration calculation has already been programmed for you, assuming that your light painting contains only linear moves in Cartesian space at a single constant tip speed. If you add other types of motions, you will need to update the calculation of `duration`.

The next three outputs (`x`, `y`, `z`) are the coordinates where the PUMA's end-effector tip should be at this point in time, specified in inches in the base frame (frame 0). The fifth through seventh outputs (`phi`, `theta`, `psi`) represent the presently desired orientation of the PUMA's end-effector in the base frame using ZYZ Euler angles in radians. The last three outputs (`r`, `g`, `b`) are the red, green, and blue components of the color that the PUMA's LED should take on. Each value must range from 0 to 1; set all three color components to zero to turn off the LED.

The main function you will call to make the robot move is **pumaServo.p**, which takes the six commanded joint angles. Below are the motion restrictions that apply to both the simulated and the real PUMA robots. Your light painting must obey all of these rules.

- **Table Collisions** The origin of frame 6 must always be at least 3 inches above the table, and the center of the robot's wrist must always be at least 4 inches above the table.
- **Wall and Motor Collisions** Both the origin of frame 6 and the center of the robot's wrist must always have a positive x_0 coordinate.
- **Joint Angle Limits** None of the PUMA's joints should be commanded outside the range of their minimum and maximum angles, as specified in part 1 of this project.
- **Joint Velocity Limit** None of the PUMA's joints should be commanded to rotate faster than 1.0 rad/s in either direction.
- **Joint Increment Limit** None of the PUMA's joints should be commanded to rotate more than ± 0.5 rad between two successive calls to **pumaServo**.

The provided starter code uses the approach described below to generate the light painting. You are welcome to update or completely rewrite this function; if you do so, please include comments to explain how your code works. As provided, **team100_get_poc.m** is based on via points that are defined in Cartesian space and are not timed. The MATLAB data file **team100.mat** was created to hold a single matrix named `painting`, defined according to the following rules:

- Each row of the `painting` matrix defines a single via point (position, orientation, and color) that the robot's LED should accomplish, along with the type of trajectory to be used between this point and the next one. The via points are listed in the order in which they should be performed.
- We purposefully do not specify a particular time for each via point so that it is easier to cause the robot to move through the via points at **constant tip speed** (giving the light painting line approximately constant brightness and thickness). The via point times necessary to move the tip with constant speed are calculated during initialization. Speed up or slow down the robot by changing the value of `tipspeed`, specified in inches per second. A value around 2 or 3 inches per second is good.
- Columns 1, 2, and 3 of the `painting` matrix contain the x, y, and z coordinates of the LED relative to frame 0, expressed in inches.
- Columns 4, 5, and 6 contain the ϕ, θ, and ψ ZYZ Euler angles that define the orientation of the LED's frame (frame 6) relative to frame 0, in radians.
- Columns 7, 8, and 9 contain the red, green, and blue color component values for the LED. Each of these values should range from zero to one, with one being brightest. Set all three to zero to cause the LED to be off at a certain via point.

- Column 10 contains an integer that specifies the type of trajectory that should be executed as the robot moves from this via point to the next one. As written, the starter code defines and uses only one trajectory type: linear interpolation on position, orientation, and color with the timing chosen to achieve a constant Cartesian tip speed of 2 in./s. This trajectory type is defined to be number `0`, and it's implemented via the provided function **linear_trajectory_kuchenbe.m**. You are welcome to use only this provided trajectory type, or you may edit it or program your own. Note that you can use the provided approach to draw curves by creating many small line segments in the **team100_get_poc.m** function rather than having to type the directly into `team100.mat`.
- You may directly edit your team's `painting` matrix by typing `load team1XX` at the command line, where `1XX` is your team number. Double-click on the variable `painting` to modify its values, and then save it back to the disk by running the command `save team1XX painting`. Alternatively, you may also write a script that calculates all of the values of this matrix and saves it to the disk. Or you might even get rid of the mat file completely and merely specify all the characteristics of your via points inside the **team1XX_get_poc.m** function.

After the via point coordinates are available in the function, note that you may scale, shift, or otherwise transform them before returning the present values. This option is useful for finding a good location for your painting in the robot's workspace. The painting in the provided starter code is a rainbow-colored square with a diagonal line from the upper-right corner to the lower-left corner. You are required to change the light painting to something more interesting than the rainbow square.

team100_puma_ik.m A slightly modified version of the function by the same name that was provided for the starter code of the first component of this project. Your version should correctly implement the **full inverse kinematics of our PUMA** according to the guidelines shared before. A new version of this function is being provided so that you can run the **team100_test_painting** and **team100_puma_paint.m** scripts before you have a fully functional version of your own IK. The provided version simply calculates θ_1 from the y coordinate of the desired position, and it calculates θ_2 from the z coordinate, keeping all other joints at their home angles.

team100_choose_solution.m Modify this function so that it chooses the **best inverse kinematics solution** from all of the solutions passed in. This decision should be based both on the characteristics of the PUMA 260 robot and on the robot's current configuration. If your IK code always returns solutions in a given order, it is sufficient to simply pick the solution that matches the robot's home pose (left-arm, elbow-down, $\theta_5 < 0$), as long as your light painting takes place entirely within the robot's workspace using this set of solutions.

The first input (`allSolutions`) is the matrix returned by the IK function; it contains the joint angles needed to place the PUMA's end-effector at the desired position and in the desired orientation. The first row is θ_1, the second row is θ_2, etc., so it has six rows. The number of columns is the number of inverse kinematics solutions that were found; each column should contain a set of joint angles that place the robot's end-effector in the desired pose. These joint angles are specified in radians according to the order, zeroing, and sign conventions described in the documentation. If the IK function could not find a solution to the inverse kinematics problem, it will pass back `NaN` (not a number) for all of the joint angles.

The second input is a vector of the PUMA robot's current joint angles (`thetasnow`) in radians. This information should be used to enable this function to choose the solution that is closest to the robot's current pose in joint space. There are also other reasons why one solution might be better than the others, including whether it violates or obeys the robot's joint limits. Note that some of the PUMA's joints wrap around. Your solutions probably include angles only from $-\pi$ to $+\pi$ or 0 to 2π radians. If a joint wraps around, there can be multiple ways for the robot to achieve the same IK solution (the given angle as well as the given angle plus or minus $2n\pi$). Be careful about this point.

team100_test_painting.m This file is a modified version of the testing function that was provided for the starter code of the first component of the project. Change all mentions of **team100** to your own team number. This script uses your **team1XX_get_poc.m** function to load the positions and orientations specified for your light painting. It uses your **team1XX_puma_ik** function to find IK solutions, and then it uses your **team1XX_choose_solution** function to chose among them. This script lets you check your IK on a robot model that does not include the same limits as the simulator. You should make sure your light painting looks correct here before running it in the simulator.

plot_puma_kuchenbe.m, puma_fk_kuchenbe.m, and dh_kuchenbe.m These functions are included so that you can easily run the **team10_test_painting.m** script. The provided versions of these functions are the same as were provided for the first component of this project.

team100_puma_paint.m This is the script that **runs the performance of the light painting**. At the top of the file, define your team number and the names of your team members. Then change all mentions of **team100** to your own team number.

You can choose which part of your light painting to test by setting the variables `tstart` and `tstop` at the top of the file. Chose the size and style of the LED marker on line 46. The code sets up the robot and the plot window in which the simulation is graphed. It then moves the robot into the position where it should be at the start time you have chosen. Once there, it sets the LED to the correct color and begins painting. For each step in the loop, the code uses `toc` to measure the elapsed time and check if the performance is done. If it's not, it calls **team1XX_get_poc.m** to obtain the new position, orientation, and color for the current time. The LED is set to the correct color, and then **team1XX_puma_ik.m** is called on the position and orientation to obtain all of the possible solutions to the inverse kinematics problem. These options are winnowed down via **team1XX_choose_solution**. The time and joint angles are stored in history matrices, and the robot is commanded to move to the calculated joint angles. At the end, the code turns off the LED and stops the PUMA robot. The image below shows the simulator after drawing the default light painting. Aside from updating your names and team number, we don't anticipate that you will need to change much in this file.

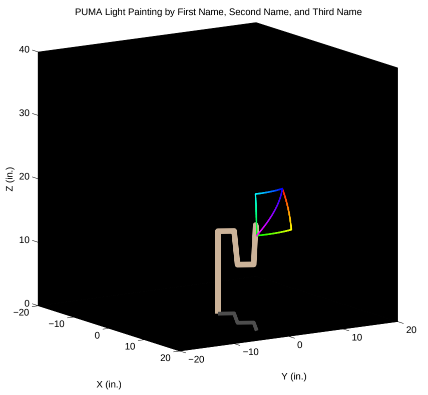
If you're having trouble getting the simulator to run your light painting, you can temporarily replace all calls to **pumaServo** with calls to **pumaMove**, which does not include the robot restrictions listed earlier in this document. However, you must get your code to work with **pumaServo** because it is the only command that works on the real robot.

Simulator Files The rest of the files provided in the starter code are for the PUMA simulator.

Submitting Your Code

Submit your correctly named MATLAB files to the **Project 1.2** assignment on the MEAM 520 Canvas site, which is available at `http://upenn.instructure.com/`. You should submit the following files:

- **team1XX_get_poc.m**
- **team1XX.mat**
- **team1XX_choose_solution.m**
- **team1XX_puma_paint.m**
- plus any additional files you have created or modified


PUMA Light Painting by First Name, Second Name, and Third Name
Z (in.)
X (in.)
Y (in.)
40
30
20
10
0
−20
−10
0
10
20
−20
−10
0
10
20

Do not submit **team1XX_puma_ik.m**, **team1XX_test_painting.m**, **linear_trajectory_kuchenbe.m**, **plot_puma_kuchenbe.m**, **puma_fk_kuchenbe.m**, or **dh_kuchenbe.m**. Be sure to upload your **.m** files, not temporary **.m**~ files. Only one member of your team should submit your team's code. You are welcome to resubmit your code before the deadline if you want to make corrections. Post any general comments and confusions on Piazza.