Project 4: Trajectory Optimization using CasADi

CS 6370 / ME EN 6225 Motion Planning
University of Utah

Overview

In this assignment, we will solve trajectory optimization problems using CasADi, a symbolic framework for nonlinear optimization. We will use V-REP as the simulation environment with a KUKA LBR4+ arm reaching to desired positions under varying constraints.

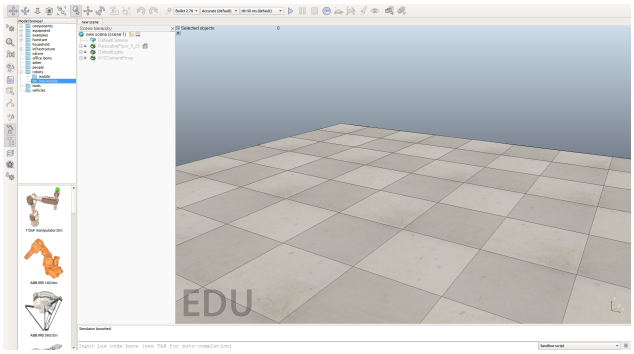
System Setup

You first need to download V-REP and get it running on your system. **If you encounter problems, please post them to the Discussion board**, because it is likely that others will have the same problem. Follow these steps:

1. Download V-REP Pro Edu for your operating system (Windows, Mac, or Linux) from their website. For Mac this is a zip file, for Linux it is a tarball, and for Windows it is a setup executable.

2. Install V-REP on your system. For Mac and Linux you simply have to expand the compressed file you downloaded to a directory of your choosing and you're ready to go. For Windows you have to run the setup executable and it will install where you tell it on your system.

3. Navigate to the V-REP root folder and start the application: `vrep.app` in Mac, `vrep.sh` in Linux, `vrep.exe` in Windows. You should see an empty environment come up:



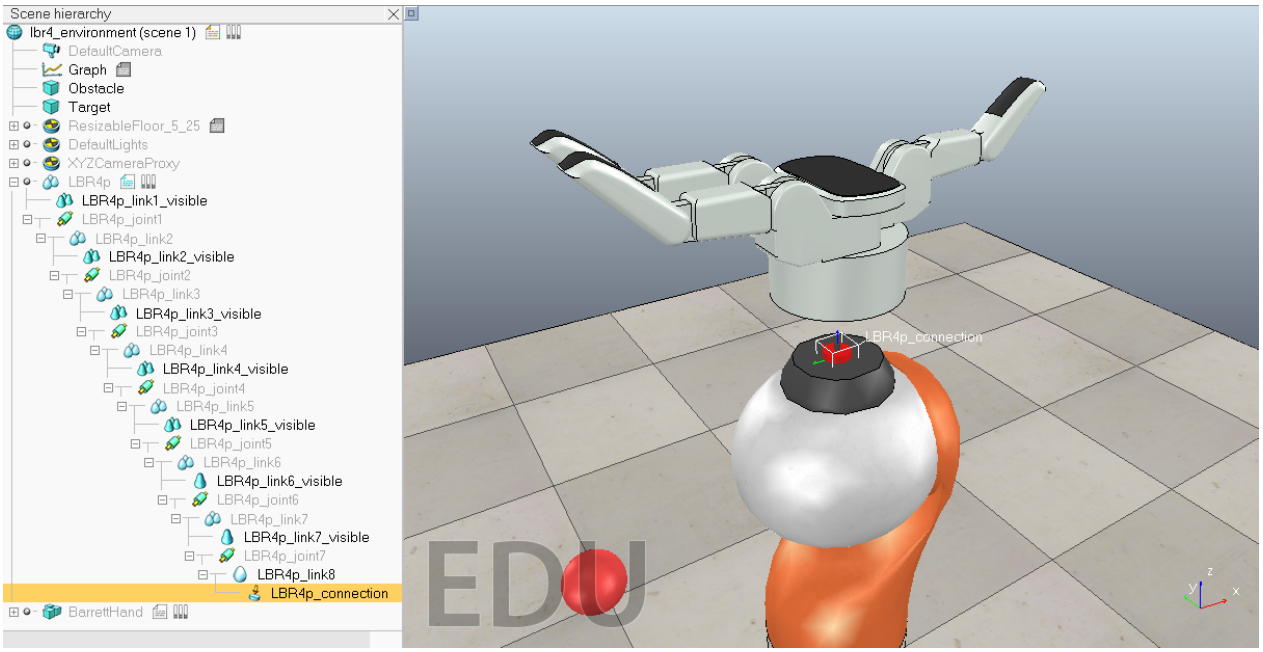
4. Install CasADi on your system as a Python library. If you use pip then it should be as easy as `pip install casadi`. It is also supported with conda for Mac and Linux. See these instructions if you're having trouble. If it's just not working for you, post a message on the discussion forum and we can try to host a virtual environment that you can use.

5. Navigate to Canvas in `Files > Project4` and download the zip file for this project. It contains a folder `project4` with all the code and V-REP Python bindings you need.

6. Go to your running instance of V-REP and open the environment we provide by going to `File > Open Scene...` and navigating to the file `project4/lbr4_environment.ttt`. You should see an LBR4 robot arm with a Barrett Hand and two colored spheres come up.

7. Run the test program `project4/lbr4_test.py`. You should see the LBR4 arm in V-REP move through a pre-programmed sequence of joint positions with a line tracing the end-effector position. If the arm doesn't move, something is wrong and you should post a message to the discussion forum.

Note that the end-effector link is `LBR4p_connection` as seen in the figure below. The attached hand was added as an afterthought just to look cool.



Quick CasADi Overview

CasADi is a symbolic framework for nonlinear optimization and algorithmic differentiation. It is targeted for use with solving optimal control problems, but it's also a relatively easy way to get access to nonlinear optimization solvers for other problems. The solver we are using takes problems of the form

$$\begin{aligned} \underset{x}{\text{minimize:}} \quad & f(x, \boldsymbol{p}) \\ \text{subject to:} \quad & x_{\text{lb}} \leq x \leq x_{\text{ub}} \\ & g_{\text{lb}} \leq g(x, \boldsymbol{p}) \leq g_{\text{ub}} \end{aligned} \tag{1}$$

where x is the decision variable you're optimizing (e.g. in our case joint angle positions), $\boldsymbol{p}$ are relevant input terms (e.g. desired goal position or position of obstacle), f is the objective (cost)

function, and g is a function of the input values that you want to constrain (e.g. a joint velocity constraint). Note we also can have constraints on the decision variable itself (e.g. joint position limits).

We have written all of the code to set up the optimization problems and solve them (we didn't want you to have to learn all of CasADi just to do this assignment). You will need to write the cost functions and add optimization constraints. It's important to note that these need to be written using **symbolic** variables and CasADi symbolic functions. We have created all the symbolic variables we think you'll need, so you should only have to use the ones already defined, but if you need others you can define them yourself.

We recommend you first read through the examples on the home page at `https://web.casadi.org` and click on the Python tabs to see how they solve those simple problems. You will probably want the L2-norm function at your disposable – in CasADi this is the `norm_2` function and is already imported in `lbr4_trajopt.py`

Reaching Desired Task Space Positions (45 pts)

We want to find joint space trajectories that allow the robot to move its end-effector from an initial position to specified goal positions. We will explore different constraints and objective functions to allow the robot to achieve this behavior. The general form of the optimization problem is

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sum_{t=1}^{T} f(\boldsymbol{\theta}_t, \boldsymbol{x}_g) \\ \text{s.t.} \quad & g_i(\boldsymbol{\theta}_t) \leq 0, i = 1, \ldots, K \\ & t = 1, \ldots, T \end{aligned} \tag{2}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_T\}$ is the joint space trajectory to be optimized, $\boldsymbol{x}_g$ is a desired task space goal position for the end-effector to reach, f is the objective function, g_i are inequality constraints (Note: that you can make equality constraints by combining two inequality constraints, setting upper and lower bounds to be equal).

All of the following problems have corresponding functions in `lbr4_trajopt.py` (e.g. problem 1.1 has a function `compute_trajectory_1_1`). These are self-contained so that for each problem, you only need to edit that one function. Most of it is written for you, you just need to do the blocks commented off with **TODO** tags. You should also look at the file `lbr4_kinematics.py` as it contains some useful functions you will need (e.g. a forward kinematics function).

Please see the file `project4/README.txt` for examples on how to execute `lbr4_trajopt.py`. It is setup already with a command line option interface.

1.1 (15 pts) Implement the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & \sum_{t=1}^{T} \|FK(\boldsymbol{\theta}_t) - \boldsymbol{x}_g\|_2^2 \\ \text{s.t.} \quad & \boldsymbol{\theta}_{lower} \leq \boldsymbol{\theta}_t \leq \boldsymbol{\theta}_{upper} \\ & t = 1, \ldots, T. \end{aligned} \tag{3}$$

Run the optimization three times, each time with a different valid goal location, and provide a screenshot of the completed trajectories in V-REP (i.e. after you have executed the trajectory and the end-effector is at the goal location with the trace of the end-effector path visible). What do you observe about the trajectory the robot takes, and why do you think it behaves like that?

1.2 (20 pts) Add an inequality constraint to the optimization problem of 1.1 that limits the instantaneous joint space velocity, i.e. $-\gamma \leq \theta_{i,t} - \theta_{i,t-1} \leq \gamma$ for each joint θ_i. Also, add two constraints at the final timestep T:

1. An equality constraint that forces the end-effector position to coincide with the goal position at the final timestep.
2. An equality constraint that forces the joint velocity to be zero at the final timestep.

Try three different values for γ that result in noticeably different trajectories and again take screenshots of the execution results. As a reference, I found $\gamma = 0.008$ to be a good nominal value. How do the trajectories differ as you change γ? How do the trajectories compare to those from 1.1? (Note: You may also have to play with the `self.num_timesteps` parameter in `LBR4TrajectoryOptimization` if you find you're not reaching the goal in the end).

1.3 (10 pts) Modify the optimization problem of 1.1 by changing the objective function to have a smoothness term:

$$f(\boldsymbol{\theta}_t, \boldsymbol{x}_g) = \|FK(\boldsymbol{\theta}_t) - \boldsymbol{x}_g\|_2^2 + \alpha\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}\|_2^2 \tag{4}$$

Note that for this problem you should drop the velocity constraint from 1.2. Try three different values of α that generate noticeably different trajectories and again provide screenshots of the execution results. How do the resulting trajectories compare to those you generated in 1.2?

Obstacle Avoidance (40 pts)

We would like the robot to not only reach to desired task space positions, but also avoid running into other objects in the environment. This section explores modifications to the optimization problem of the previous section to allow the robot to avoid running into things.

2.1 (20 pts) Design and implement an objective function that allows the robot end-effector to reach a desired task space position $\boldsymbol{x}_g$ subject to velocity constraints, while also avoiding contact with an obstacle position $\boldsymbol{x}_{obst}$ (this is the centroid of the object in V-REP). You

should design it to have an open parameter β that modulates the influence of the obstacle-avoidance term (similar to how α modulated the influence of the smoothness term in 1.3).

2.2 (10 pts) Run the optimization with three different values of β that produce noticeably different trajectories and provide screenshots of the execution results from V-REP. How does changing the value of β affect the trajectory?

2.3 (10 pts) Your objective function from 2.1 only considered the robot end-effector avoiding the centroid of an object. This approach ignores the rest of the robot (and also the rest of the object), so even if the robot's end-effector does not collide with the object, one of the other links still might. What concepts from class would you use to resolve this issue? Describe in a few sentences how you would design your optimization problem to handle this. (Note: there is no code for this problem.)

Self Analysis (5 pts)

3.1 (1 pt) What was the hardest part of the assignment for you?

3.2 (1 pt) What was the easiest part of the assignment for you?

3.3 (1 pt) What problem(s) helped further your understanding of the course material?

3.4 (1 pt) Did you feel any problems were tedious and not helpful to your understanding of the material?

3.5 (1 pt) What other feedback do you have about this homework?

Submission Instructions

All solutions should be uploaded via the assignment on Canvas.

Submission: Please submit two **separate** files: a **pdf** report and a zip file.

PDF Report: Place all written answers to questions as well as any images or other output data used to explain your responses in a single PDF document. This should be clearly named Proj<*number*>-answers.pdf, where number is the Project number (i.e. 4 for this assignment). Please make sure to write your name and uid at the top of the document!

Zip File: Your zip file should be named in the format <*uid*>.zip where <*uid*> is your Utah uid. The first level of the zip should contain a single folder with your uid as the name <*uid*>. Please put the README.txt, code and other necessary files (e.g. map or environment files) in the zip file. The README.txt should be in the root directory of your submitted zip file which explains how to run the code submitted.