

# Motion Planning HW 3: Trajectory Optimization using CasADi

Will Graham

## 1 REACHING DESIRED TASK SPACE POSITIONS

---

### 1.1

1.1 (15 pts) Implement the following optimization problem:

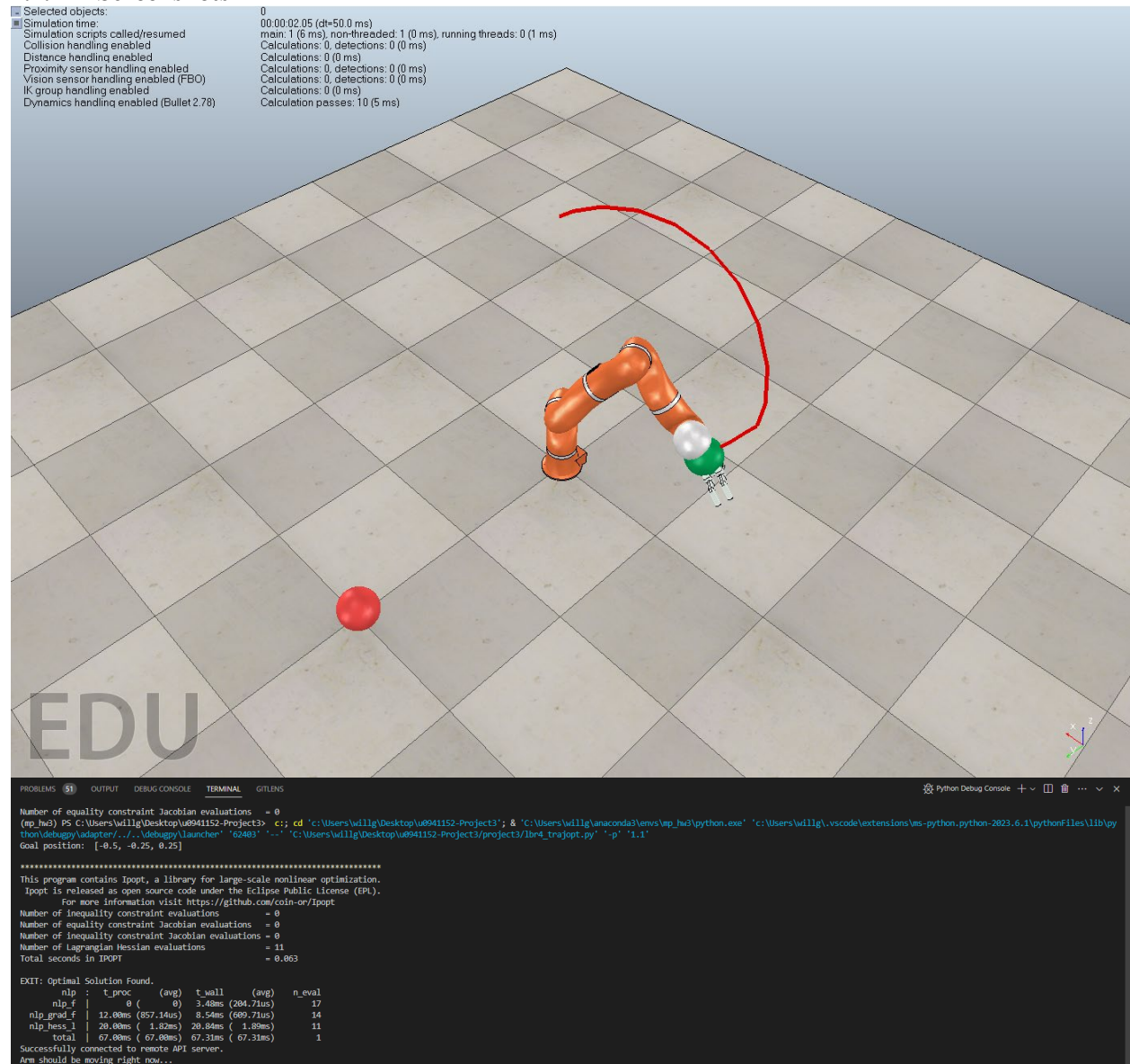
$$\begin{aligned} \min_{\theta} \quad & \sum_{t=1}^T \|FK(\theta_t) - x_g\|_2^2 \\ \text{s.t.} \quad & \theta_{lower} \leq \theta_t \leq \theta_{upper} \\ & t = 1, \dots, T. \end{aligned} \tag{3}$$

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?

#### 1.1.1 Analysis

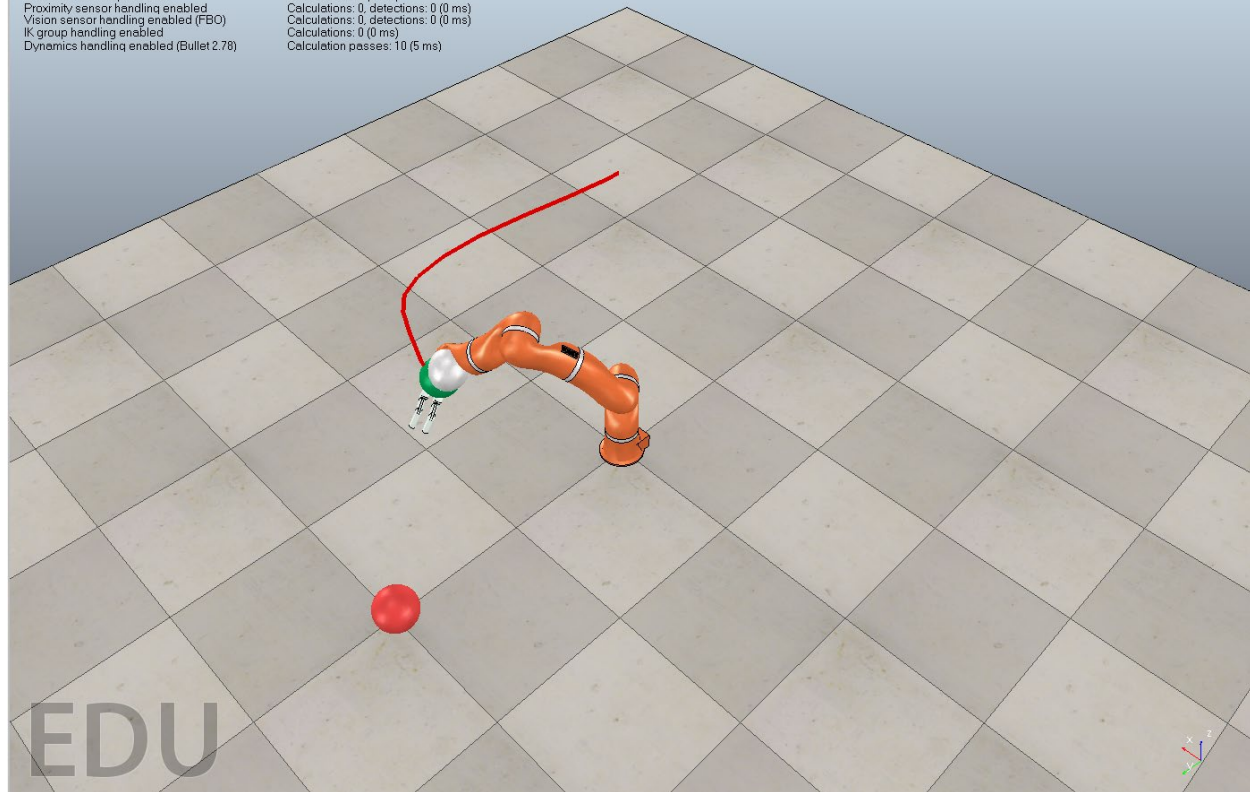
The trajectories seem to be a bit choppy, but overall they find the goal location. The costing function is minimizing the position of the end effector, not the smoothness of the operation. The priority here isn't the health of the robot, or the appearance of it's motion, but rather driving the gradient as quickly as possible towards a minimal cost. Additionally, these optimizations are taking very few time steps to solve.

## 1.1.2 Screenshots



Selected objects:  
Simulation time:  
Simulation scripts called/resumed  
Collision handling enabled  
Distance handling enabled  
Proximity sensor handling enabled  
Vision sensor handling enabled (FBO)  
IK group handling enabled  
Dynamics handling enabled (Bullet 2.78)

0  
00:00:11.10 (dt=50.0 ms)  
main: 1 (5 ms), non-threaded: 1 (0 ms), running threads: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculations: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculations: 0 (0 ms)  
Calculation passes: 10 (5 ms)



PROBLEMS 51 OUTPUT DEBUG CONSOLE TERMINAL GITLENS

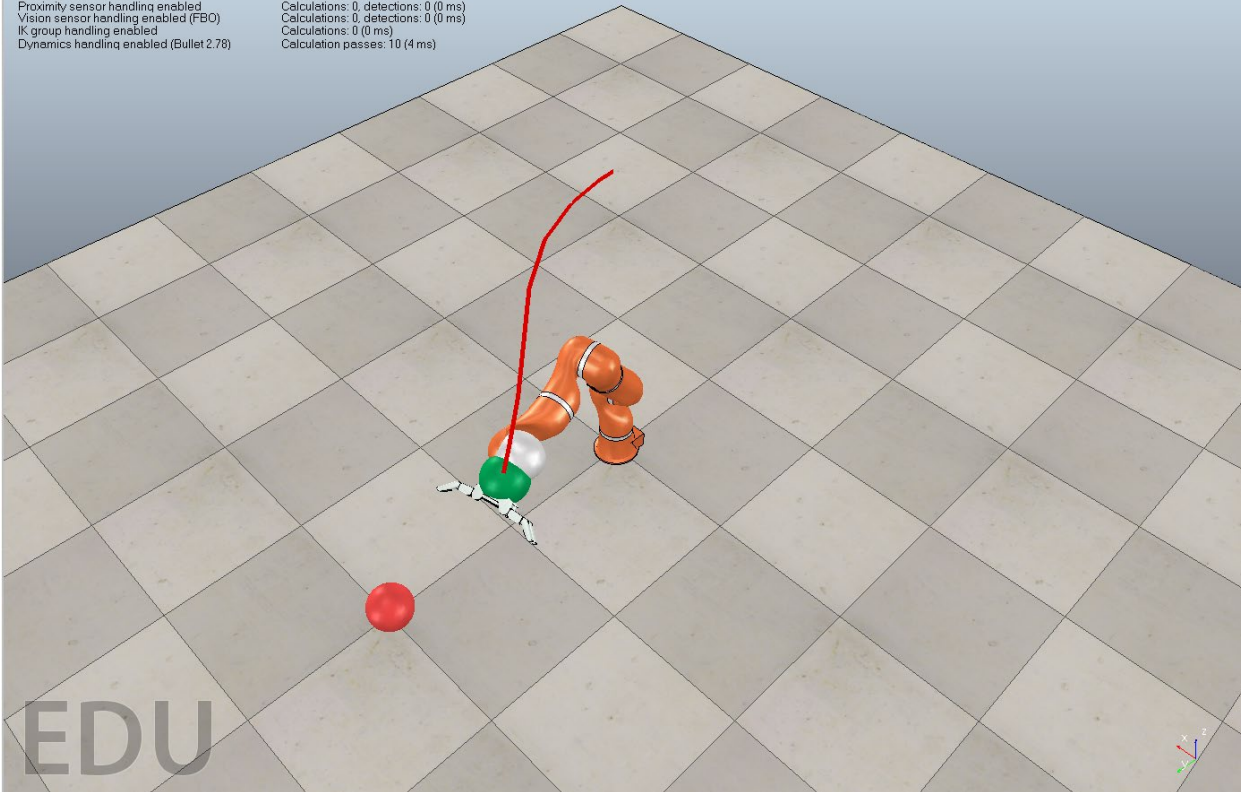
Python Debug Console + - [ ] ... v x

```
(mp_hu3) PS C:\Users\willg\Desktop\06941152-Project3> cd "C:\Users\willg\Desktop\06941152-Project3"; & "C:\Users\willg\anaconda3\envs\mp_hu3\python.exe" "C:\Users\willg\Desktop\06941152-Project3\project3\lib4_trajopt.py" "-p" "1.1"
Goal position: [[0.5533498]
[0.35903449]
[0.23048171]]
Number of equality constraint evaluations = 0
Number of inequality constraint evaluations = 0
Number of equality constraint Jacobian evaluations = 0
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations = 8
Total seconds in IPOPT = 0.050

EXIT: Optimal Solution Found.
      nlp : t_proc (avg) t_wall (avg) n_eval
      nlp_f | 2.00ms (222.22us) 2.40ms (266.33us) 9
      nlp_grad_f | 8.00ms (727.27us) 7.16ms (650.91us) 11
      nlp_hess_l | 16.00ms ( 2.00ms) 15.98ms ( 2.00ms) 8
      total | 52.00ms ( 52.00ms) 53.31ms ( 53.31ms) 1

Successfully connected to remote API server.
```

Selected objects: 0  
Simulation time: 00:00:01.90 (dt=50.0 ms)  
Simulation scripts called/resumed: main: 1 (5 ms), non-threaded: 1 (0 ms), running threads: 0 (0 ms)  
Collision handling enabled: Calculations: 0, detections: 0 (0 ms)  
Distance handling enabled: Calculations: 0 (0 ms)  
Proximity sensor handling enabled: Calculations: 0, detections: 0 (0 ms)  
Vision sensor handling enabled (FBO): Calculations: 0, detections: 0 (0 ms)  
IK group handling enabled: Calculations: 0 (0 ms)  
Dynamics handling enabled (Bullet 2.78): Calculation passes: 10 (4 ms)



EDU

PROBLEMS (5) OUTPUT DEBUG CONSOLE TERMINAL GITLENS

Python Debug Console

thor\debuggy\launcher\..\..\debuggy\launcher' '62434' '-' 'C:\Users\willg\Desktop\06941152-Project3\project3\lbr4\_trajopt.py' '-p' '1.1'  
Goal position: [[-0.20781425]  
[ 0.64523651]  
[ 0.53971922]]  
  
\*\*\*\*\*  
This program contains Ipopt, a library for large-scale nonlinear optimization.  
Ipopt is released as open source code under the Eclipse Public License (EPL).  
For more information visit <https://github.com/coin-or/ipopt>  
\*\*\*\*\*  
  
This is Ipopt version 3.14.11, running with linear solver MUMPS 5.4.1.  
  
Number of nonzeros in equality constraint Jacobian...: 0  
Number of nonzeros in inequality constraint Jacobian...: 0  
Number of nonzeros in Lagrangian Hessian...: 6279  
  
Total number of variables...: 2093  
variables with only lower bounds: 0  
variables with lower and upper bounds: 2093  
variables with only upper bounds: 0  
Total number of equality constraints...: 0  
Total number of inequality constraints...: 0  
inequality constraints with only lower bounds: 0  
inequality constraints with lower and upper bounds: 0  
inequality constraints with only upper bounds: 0  
  
Iter objective inf\_pr inf\_du lg(mu) ||d|| lg(rg) alpha du alpha pr ls  
0 2.5928886e+02 0.00e+00 1.12e+00 -1.0 0.00e+00 - 0.00e+00 0.00e+00 0  
1 3.8518998e+01 0.00e+00 8.25e-01 -1.0 0.91e-01 - 7.18e-01 1.00e+00f 1  
2 9.2389955e-01 0.00e+00 1.43e-01 -1.7 4.75e-01 - 8.59e-01 1.00e+00f 1  
3 3.0091995e-01 0.00e+00 2.54e-02 -2.5 8.83e-02 - 9.91e-01 1.00e+00f 1  
4 2.6084394e-02 0.00e+00 8.94e-04 -3.8 1.12e-01 -2.0 1.00e+00 1.00e+00f 1  
5 2.3001879e-05 0.00e+00 3.27e-05 -3.8 9.78e-03 -2.5 1.00e+00 1.00e+00f 1  
6 4.9725225e-09 0.00e+00 3.34e-06 -5.7 4.26e-03 - 1.00e+00 1.00e+00f 1  
7 7.5490439e-15 0.00e+00 4.46e-09 -8.6 1.57e-04 - 1.00e+00 1.00e+00f 1  
  
Number of iterations.....: 7  
  
(scaled) (unscaled)  
Objective.....: 7.549043998918664e-15 7.549043998918664e-15  
Dual infeasibility.....: 4.4581852980933824e-09 4.4581852980933824e-09  
Constraint violation.....: 0.000000000000000e+00 0.000000000000000e+00  
Variable bound violation: 0.000000000000000e+00 0.000000000000000e+00  
Complementarity.....: 2.6864395799767895e-09 2.6864395799767895e-09  
Overall NLP error.....: 4.4581852980933824e-09 4.4581852980933824e-09  
  
Number of objective function evaluations: = 8  
Number of objective gradient evaluations: = 8  
Number of equality constraint evaluations: = 0  
Number of inequality constraint evaluations: = 0  
Number of equality constraint Jacobian evaluations: = 0  
Number of inequality constraint Jacobian evaluations: = 0  
Number of Lagrangian Hessian evaluations: = 7  
Total seconds in IPOPT: = 0.038  
  
EXIT: Optimal Solution Found.  
nlp : t\_proc (avg) t\_wall (avg) n\_eval  
nlp\_f | 1.00ms (125.00us) 1.88ms (230.12us) 8  
nlp\_grad\_f | 13.00ms ( 1.30ms) 6.58ms (628.10us) 10  
nlp\_hess\_l | 13.00ms ( 1.80ms) 13.05ms ( 1.80ms) 7

## 1.2

1.2 (20 pts) Add an inequality constraint to the optimization problem of 1.1 that limits the instantaneous joint space velocity, i.e.  $-\gamma \leq \dot{\theta}_{i,t} - \dot{\theta}_{i,t-1} \leq \gamma$  for each joint  $\theta_i$ . 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  $\gamma$  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  $\gamma$ ? 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).

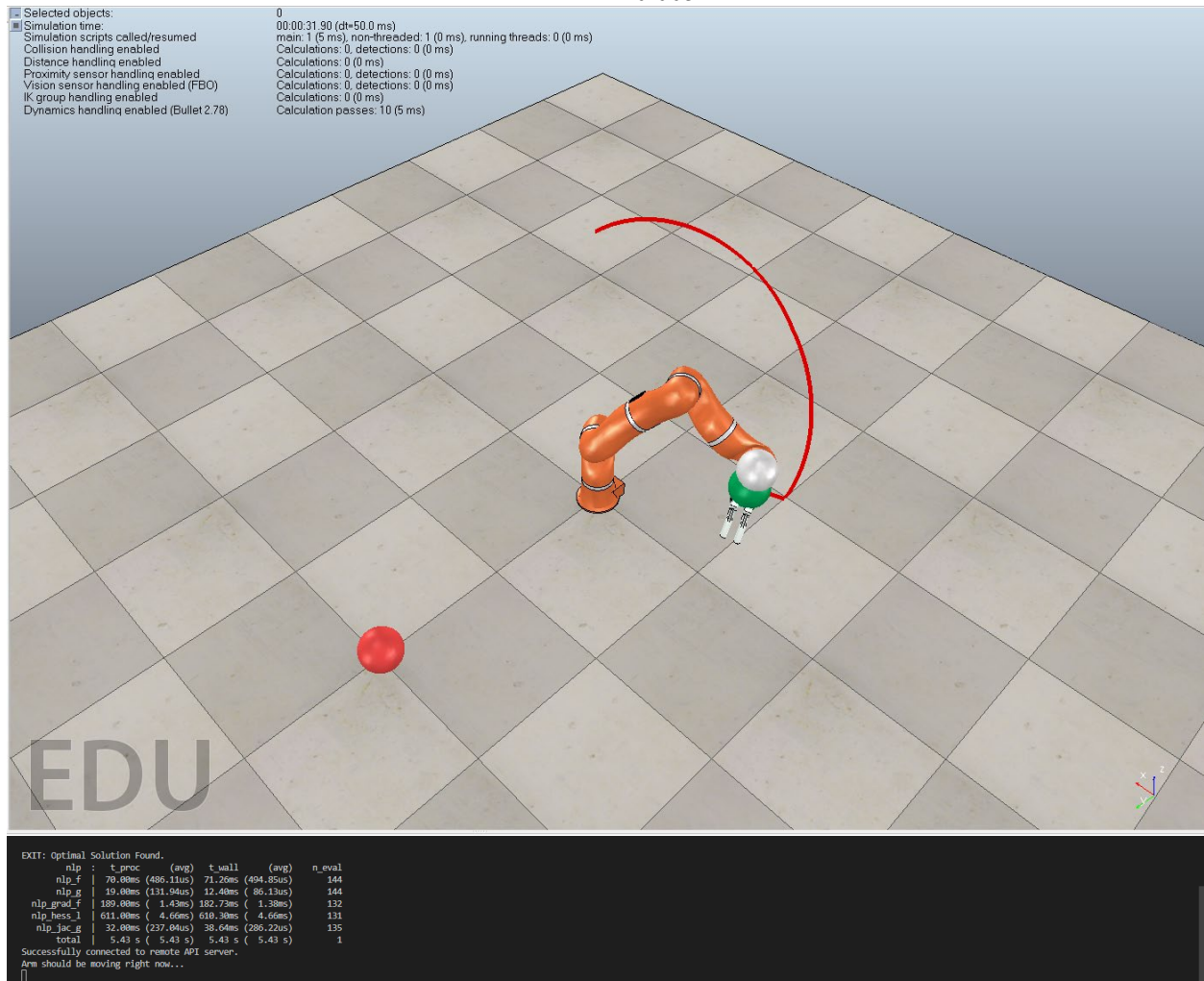
### 1.2.1 Analysis

The lower the gamma, the smoother the path. As the joint velocity is constrained, the end effector trajectory becomes much less jagged and choppy. When this gamma parameter is loosened, the joint velocities can move faster than before, leading to a much faster convergence, at the cost of smooth trajectories. The tighter the gamma constraints, the longer the optimization took to complete. When gamma was 0.05 or lower, we had ~115 iterations for a solution. At Gamma = 0.5, we only had 86 iterations for an optimization solution.

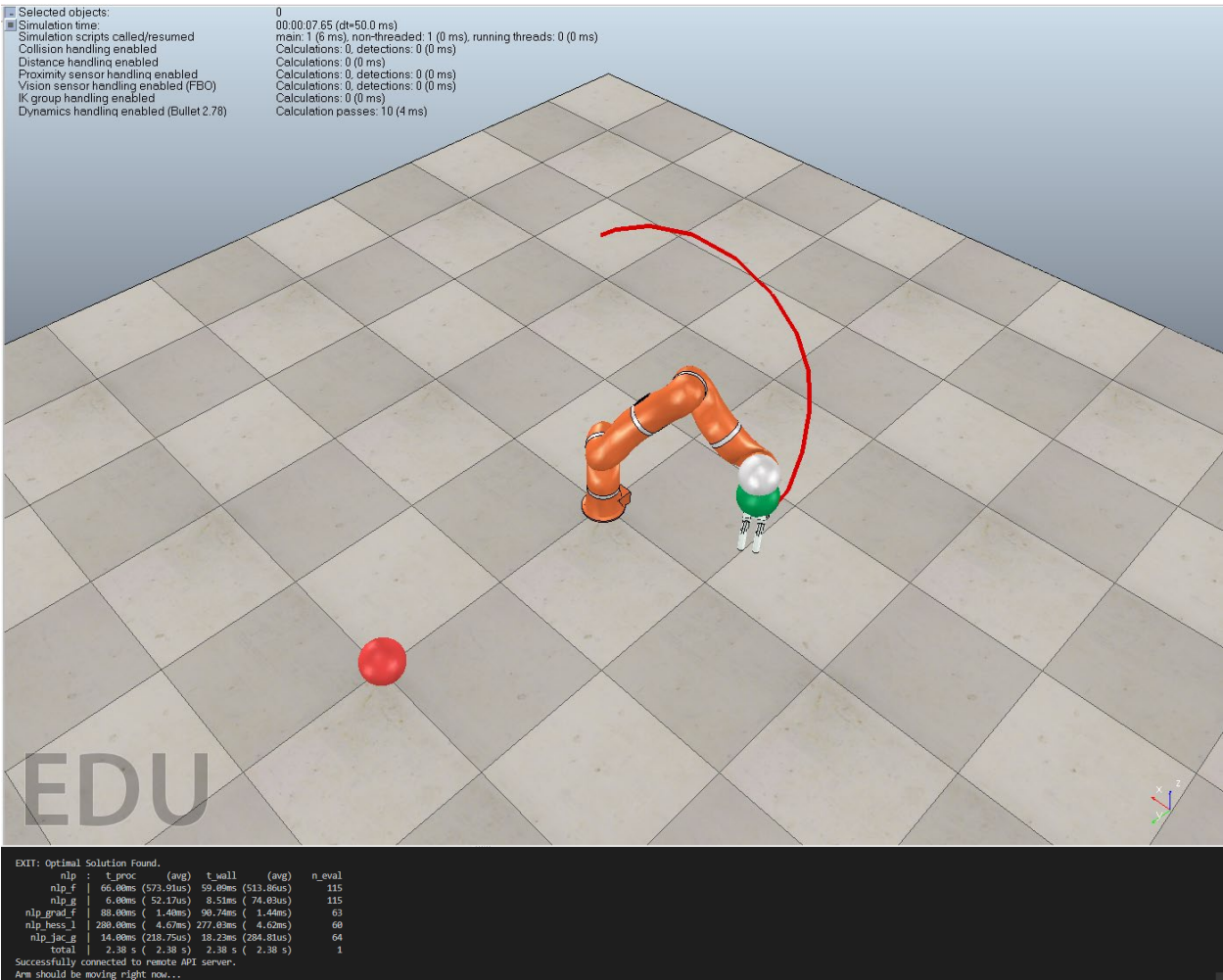


## 1.2.2 Plots

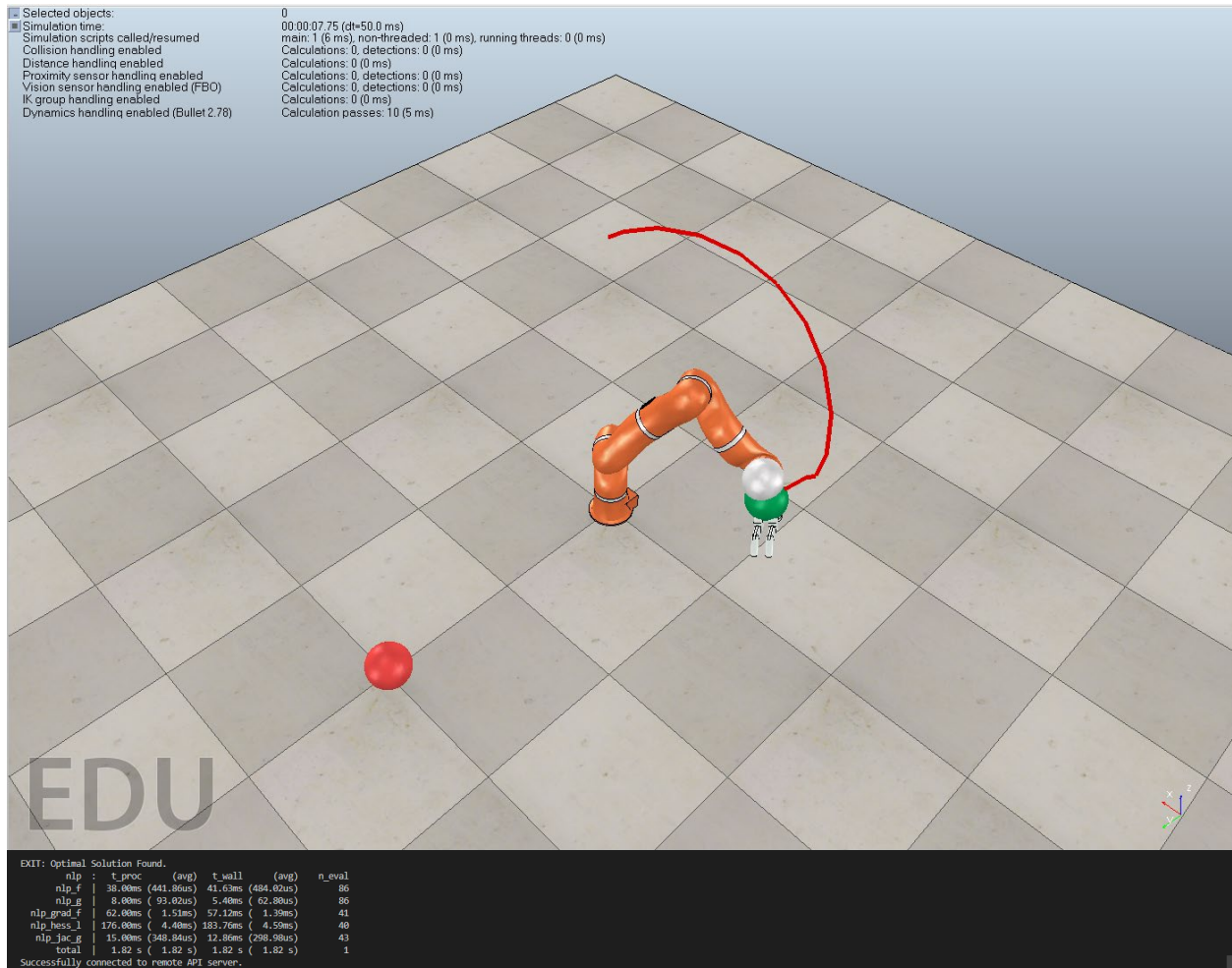
Gamma = 0.005



Gamma = 0.15



Gamma = 1





## 1.3

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

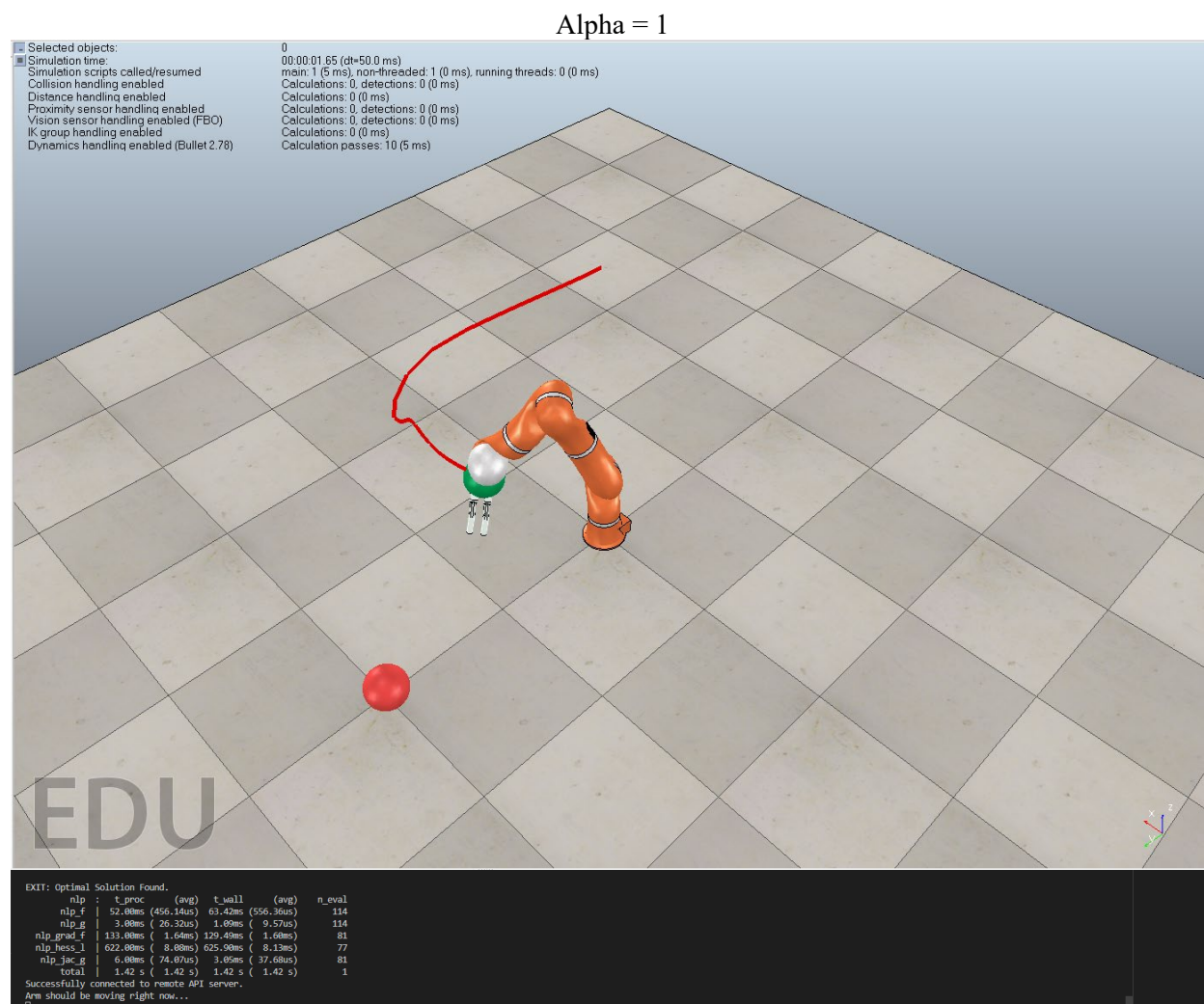
$$f(\theta_t, x_g) = \|FK(\theta_t) - x_g\|_2^2 + \alpha \|\theta_t - \theta_{t-1}\|_2^2 \quad (4)$$

Note that for this problem you should drop the velocity constraint from 1.2. Try three different values of  $\alpha$  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?

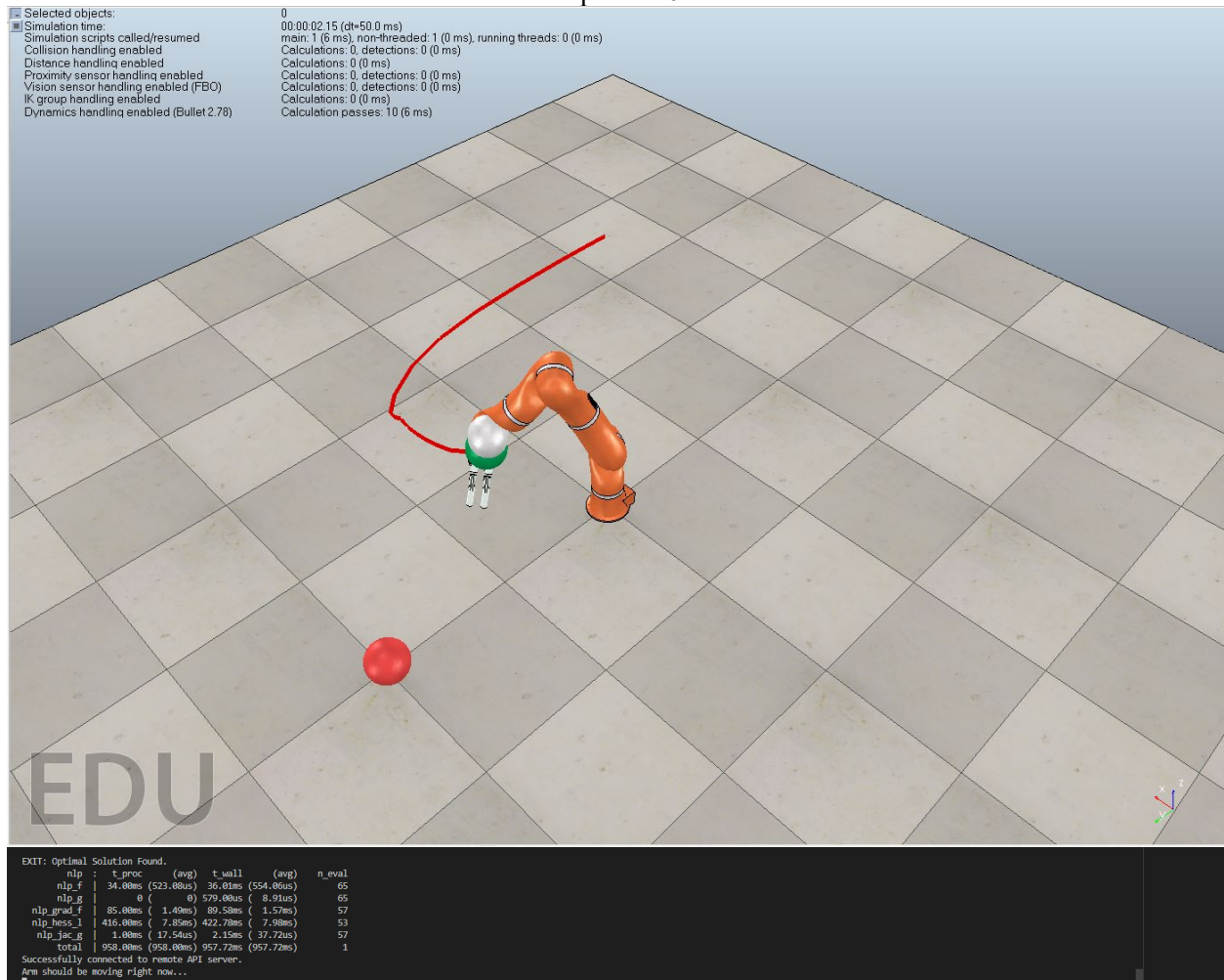
### 1.3.1 Analysis

The smoothness cost function acted very similarly to our gamma term. In problem 1.2, the joint velocity constraint seems to do a better job of ensuring a smoother end-effector path. We needed to increase our alpha to much higher parameters to get similar performance. The other effect was that our trajectories took much longer when approaching the goal. I'm sure it had lower acceleration when approaching, but it seemed to have an effect of lengthening the time required to successfully execute the generated trajectory.

### 1.3.2 Screenshots



Alpha = 10



Alpha = 35

Selected objects:

Simulation time:

Simulation scripts called/resumed

Collision handling enabled

Distance handling enabled

Proximity sensor handling enabled

Vision sensor handling enabled (FBO)

IK group handling enabled

Dynamics handling enabled (Bullet2.78)

0

00:00:10.05 (dt=50.0 ms)

main: 1 (6 ms), non-threaded: 1 (0 ms), running threads: 0 (0 ms)

Calculations: 0, detections: 0 (0 ms)

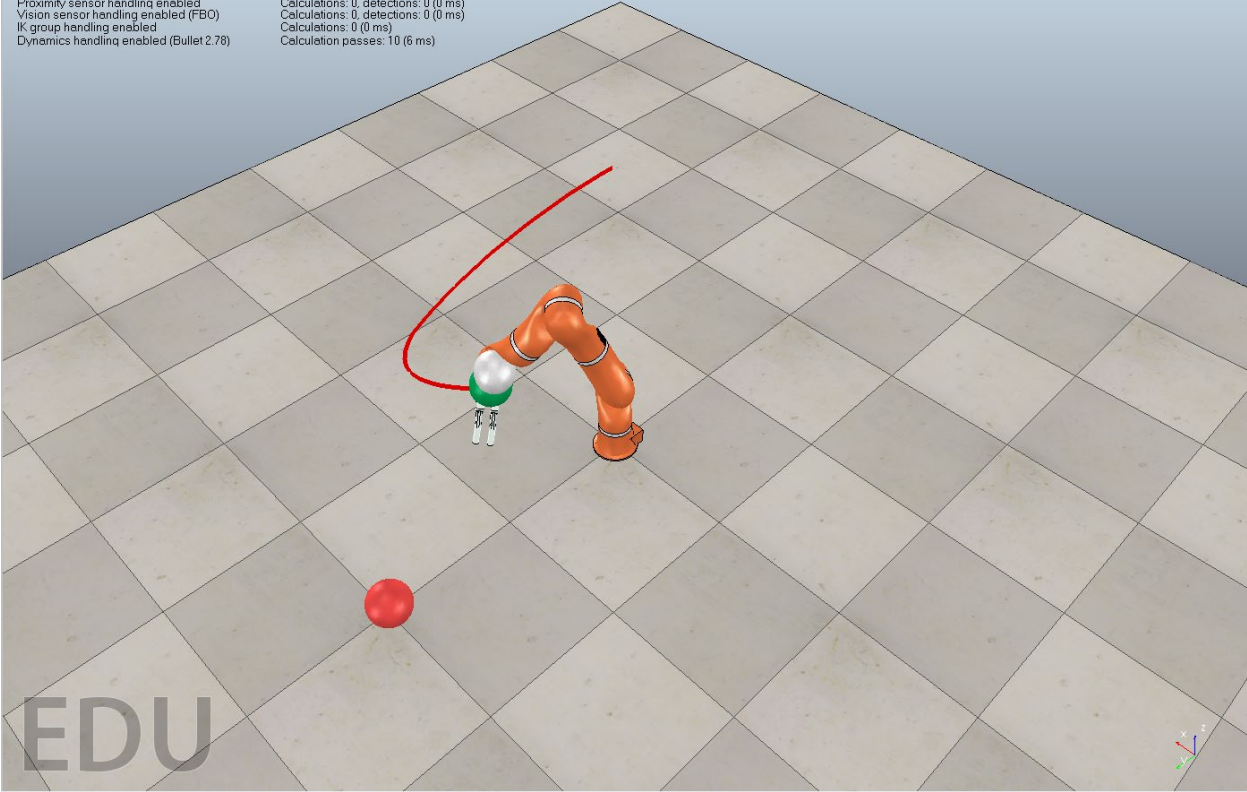
Calculations: 0 (0 ms)

Calculations: 0, detections: 0 (0 ms)

Calculations: 0, detections: 0 (0 ms)

Calculations: 0 (0 ms)

Calculation passes: 10 (6 ms)



EXIT: Optimal Solution Found.

nlp	:	t_proc	(avg)	t_wall	(avg)	n_eval
nlp_f		31.00ms	(500.00us)	31.89ms	(514.32us)	62
nlp_g		0	(0)	523.00us	(8.44us)	62
nlp_grad_f		70.00ms	(1.35ms)	75.03ms	(1.44ms)	52
nlp_hess_l		367.00ms	(7.55ms)	365.94ms	(7.62ms)	48
nlp_jac_g		0	(0)	1.62ms	(31.23us)	52
total		839.00ms	(839.00ms)	837.88ms	(837.88ms)	1

Successfully connected to remote API server.

Arm should be moving right now...

## 2 OBSTACLE AVOIDANCE

---

### 2.1

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

You should design it to have an open parameter  $\beta$  that modulates the influence of the obstacle-avoidance term (similar to how  $\alpha$  modulated the influence of the smoothness term in 1.3).

I wasn't sure how to answer this, other than by saying that I did 4 plots of beta instead of 3. It works lol

### 2.2

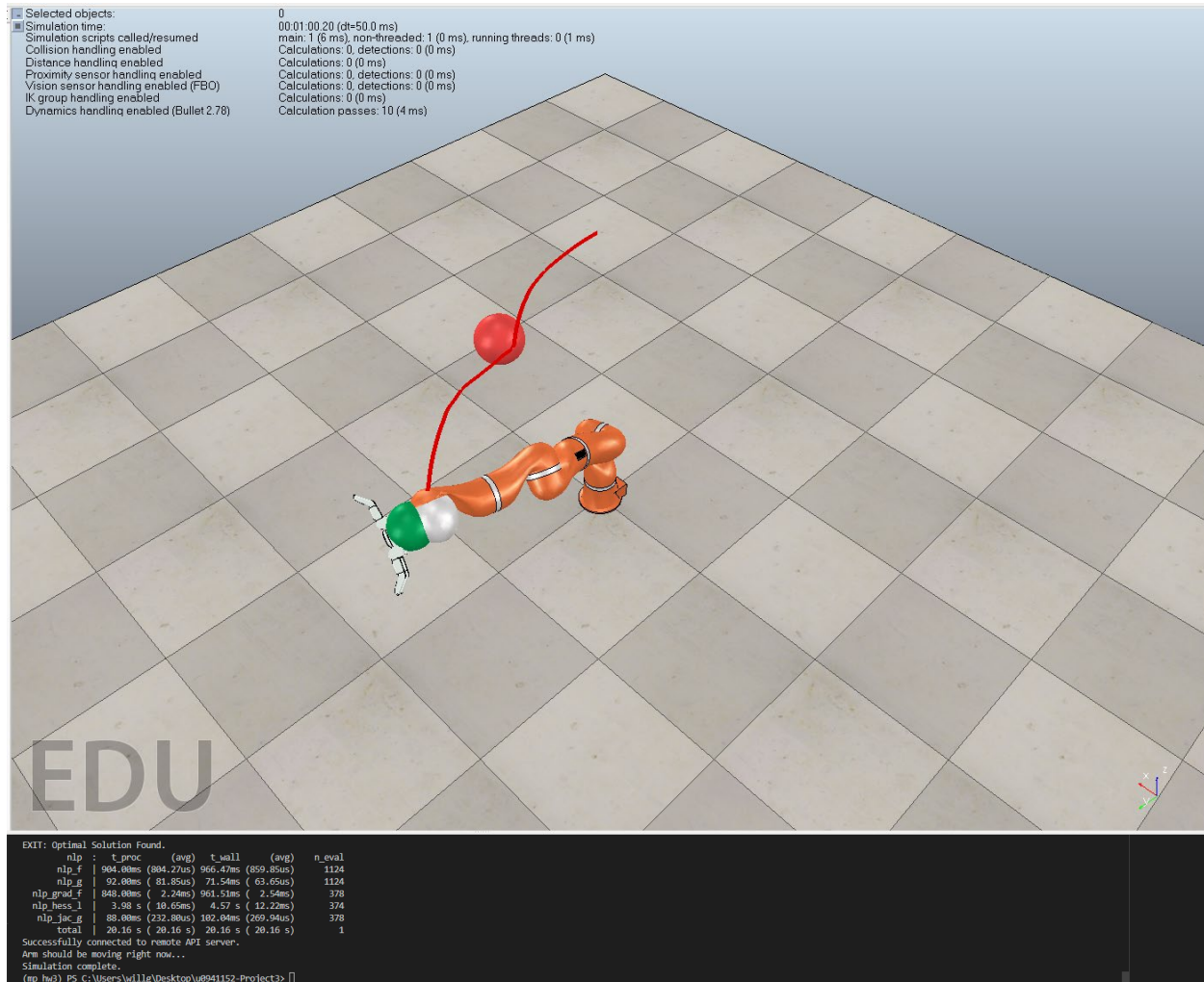
- 2.2 (10 pts) Run the optimization with three different values of  $\beta$  that produce noticeably different trajectories and provide screenshots of the execution results from V-REP. How does changing the value of  $\beta$  affect the trajectory?

#### 2.2.1 Analysis

The beta was a MUCH more sensitive parameter when compared with earlier attempts. In development of the algorithm, I couldn't square the norm\_2, because the response was much too large. Instead, beta was linearly combined for better results. There is still quite a bit of jerk going on, but the beta made a huge difference in determining if the trajectory would slightly avoid the obstacle or sweep in an entirely different direction. I ran a few additional tests on this one, as the results became progressively more exaggerated.

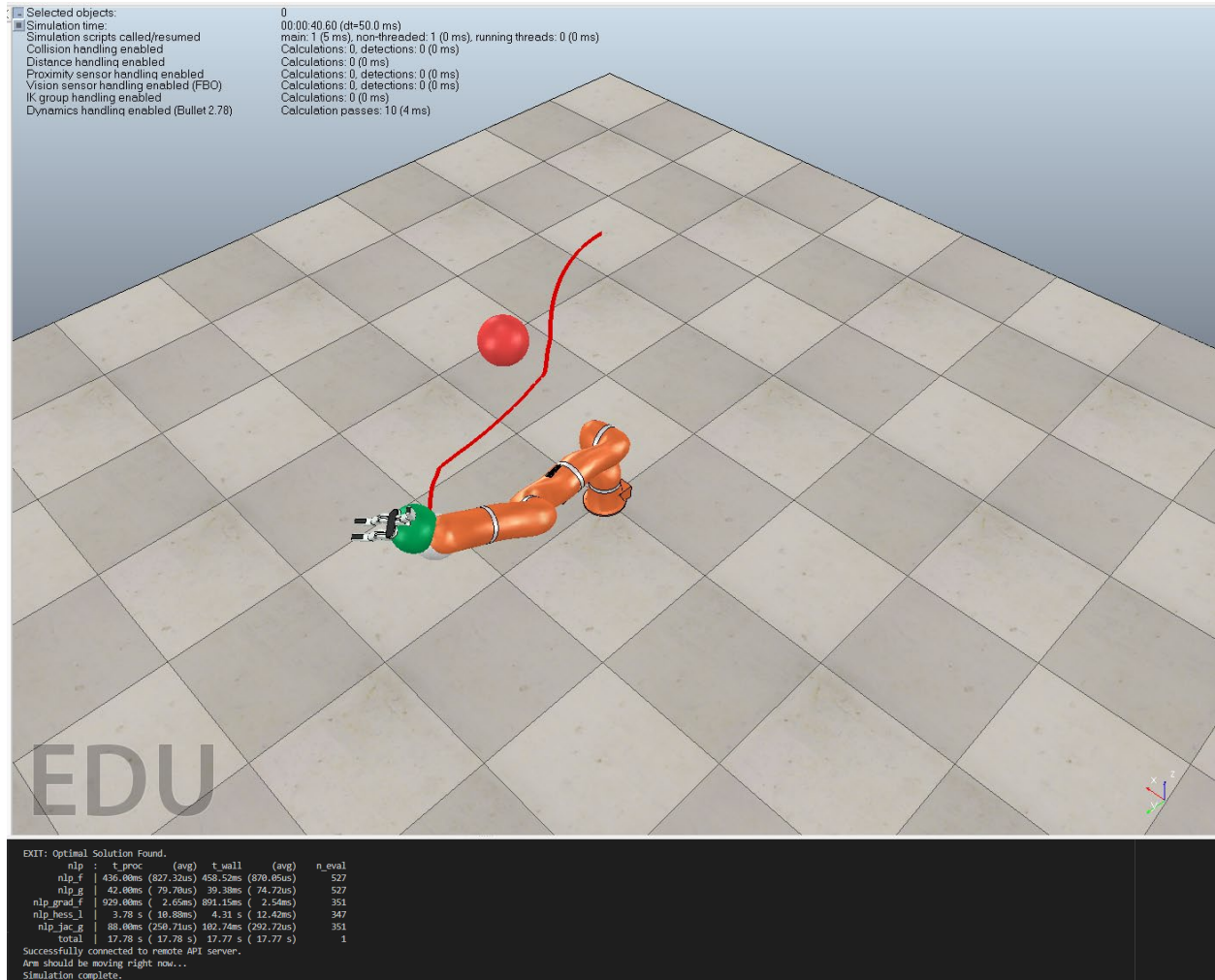
## 2.2.2 Screenshots

Beta = .005





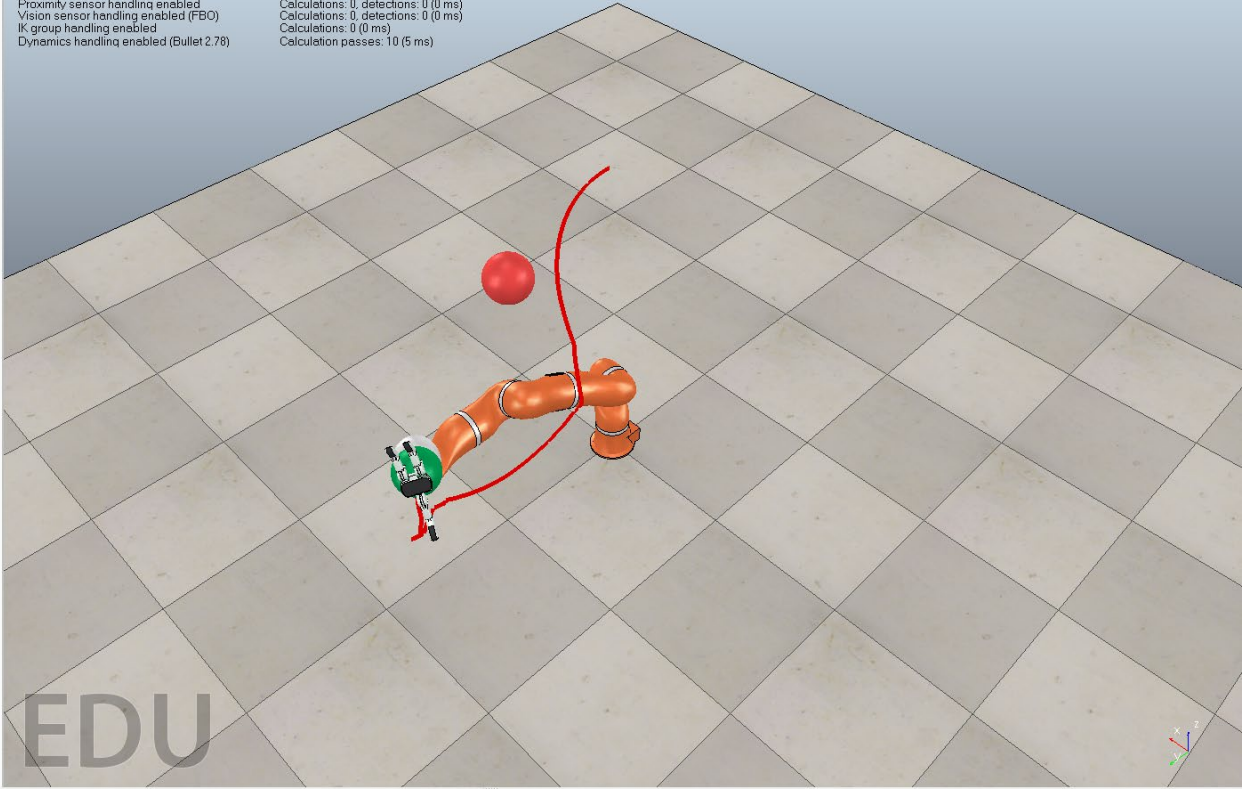
Beta = 0.05



Beta = 0.5

Selected objects:  
Simulation time:  
Simulation scripts called/resumed  
Collision handling enabled  
Distance handling enabled  
Proximity sensor handling enabled  
Vision sensor handling enabled (FBO)  
IK group handling enabled  
Dynamics handling enabled (Bullet 2.78)

0  
00:00:40.25 (dt=50.0 ms)  
main: 1 (6 ms), non-threaded: 1 (0 ms), running threads: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculations: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculations: 0 (0 ms)  
Calculations: 0, detections: 0 (0 ms)  
Calculation passes: 10 (5 ms)



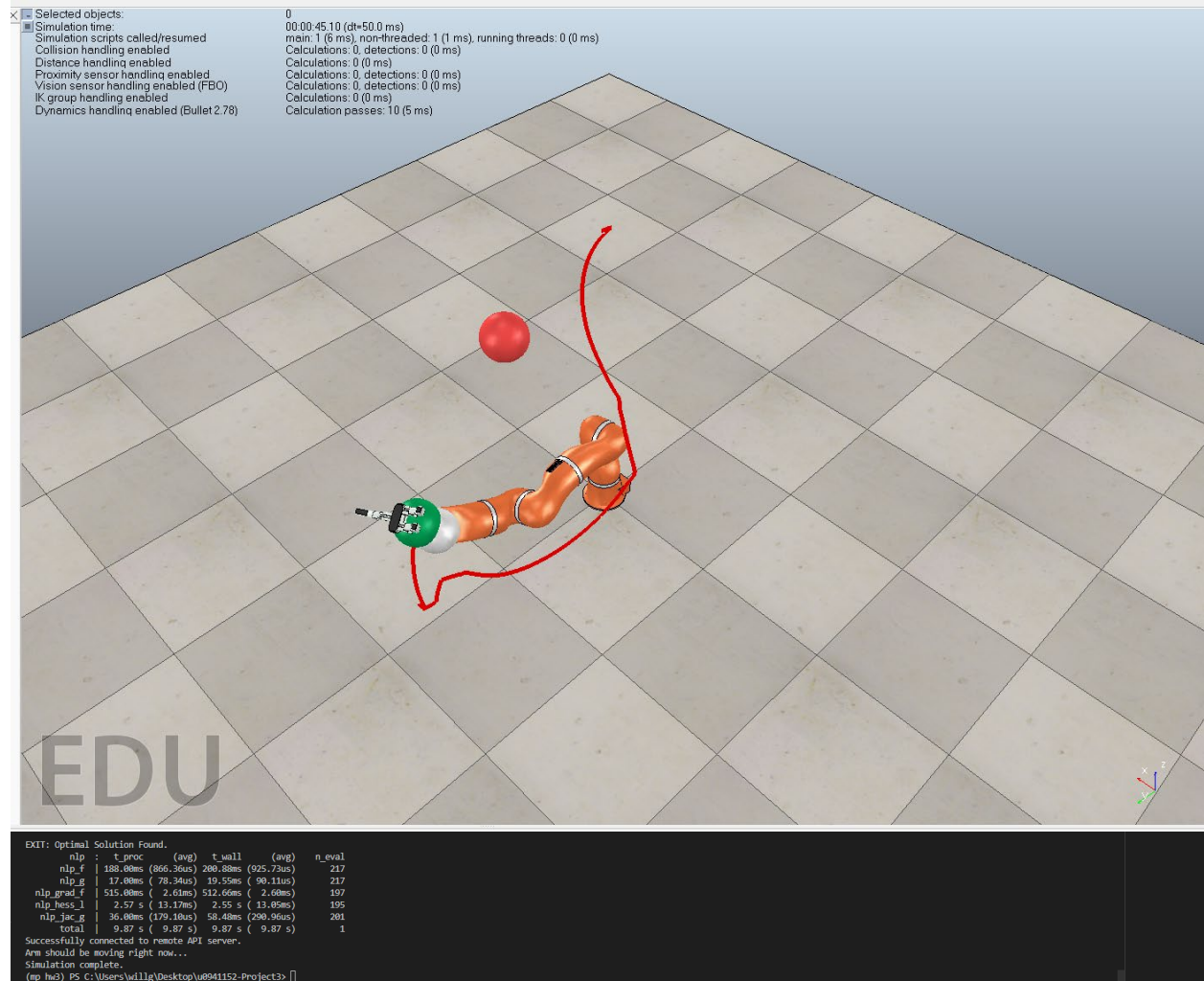
EDU

EXIT: Optimal Solution Found.

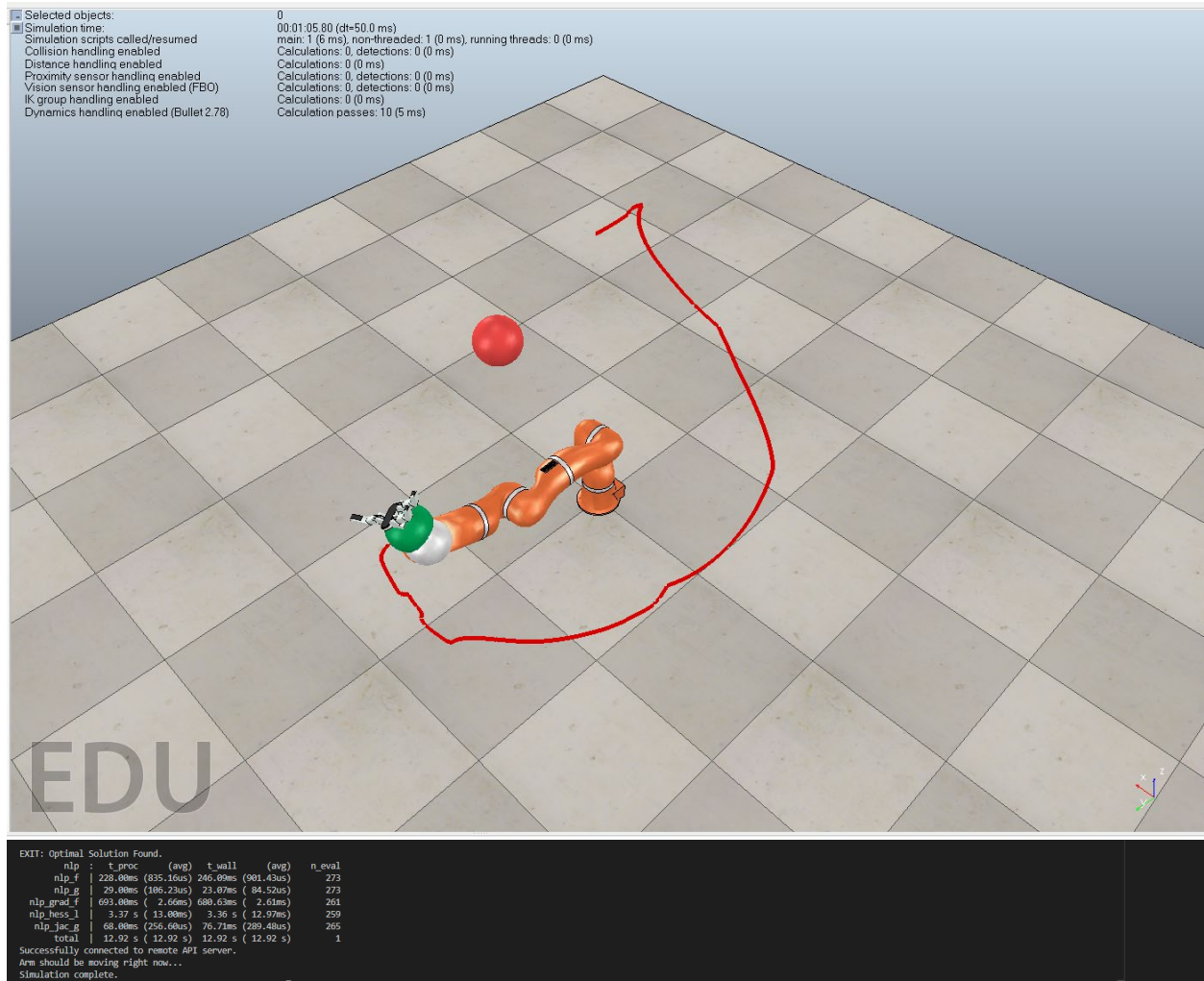
nlp	t_proc	(avg)	t_wall	(avg)	n_eval
nlp_f	202.00ms	( 1.13ms)	195.98ms	( 1.10ms)	178
nlp_g	28.00ms	(157.30us)	18.02ms	(101.26us)	178
nlp_grad_f	504.00ms	( 3.00ms)	577.31ms	( 3.44ms)	168
nlp_hess_1	2.44 s	( 17.14ms)	2.79 s	( 16.62ms)	166
nlp_jac_g	54.00ms	(311.95us)	55.74ms	(324.10us)	172
total	9.36 s	( 9.36 s)	9.37 s	( 9.37 s)	1

Successfully connected to remote API server.  
Arm should be moving right now...  
Simulation complete.

Beta = 1



Beta = 5



## 2.3

**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.)

One way to do this would be to calculate the norm from the centroid of each joint in relation to the object, and penalize each of those objects with either a different term, or the same beta term. This would require additional capability from the inverse kinematics, going backwards in the derivations. We could also use a 3d mesh (voxel grid) and pass each trajectory through a collision checker. To check collisions, we could use an algorithm like: RAPID, V-Collide, I-Collide, or flexible-collision-library. Ideally, some form of a voxel costmap would be implemented to ensure collisions don't occur.

I think the costing function based on the centroid of each joint is a bit more elegant than voxel grids, so I'd likely go with that. Of course, I'd perform time complexity studies to make sure that I'm not sacrificing computing time for the sake of a pretty-looking algorithm.



## 3 SELF ANALYSIS

---

### 3.1 WHAT WAS THE HARDEST PART OF THE ASSIGNMENT?

This is extremely dumb, but I didn't realize there was a `norm_2` function within casadi. I was trying to do all of that through indexing, and it was a nightmare. The project moved much more quickly once I discovered there was that functionality.

### 3.2 WHAT WAS THE EASIEST PART OF THE ASSIGNMENT FOR YOU?

Interestingly, it was adding the constraints. I thought that would be an extremely difficult aspect, but I never had an issue.

### 3.3 WHAT PROBLEM HELPED FURTHER YOUR UNDERSTANDING OF THE COURSE MATERIAL?

I loved the last problem, where there wasn't an algorithm given. I struggled with that for a bit, but ended up getting semi-decent results. I think that struggle helped me to learn better.

### 3.4 DID YOU FEEL ANY PROBLEMS WERE TEDIOUS AND NOT HELPFUL TO YOUR UNDERSTANDING OF THE MATERIAL?

No, this was extremely helpful, and I honestly enjoyed doing the project.

### 3.5 WHAT OTHER FEEDBACK DO YOU HAVE ABOUT THIS HOMEWORK

I'd love to see mobile robots introduced here too. I know that might be too much in this setting, but I'd really love to see how to implement these optimizations within an environment that isn't a 6-axis arm.