Homework 4
Robot Autonomy
CMU 16-662, Spring 2016
Group 7 – Abhishek Bhatia (abhatia1), Lekha Mohan (lwalajap), and ShuKai Lin (shukail)

Note:

- Please comment/un-comment the appropriate lines in the function GetBasePoseForObjectGrasp in GraspPlanner to run Test Case 1/2/3 for Part 3. Currently the code is enabled to run for Test Case 3. Respective line numbers: 56-72 and 122-129.

1).

There are three states in the control space, w_l, w_r, and duration.

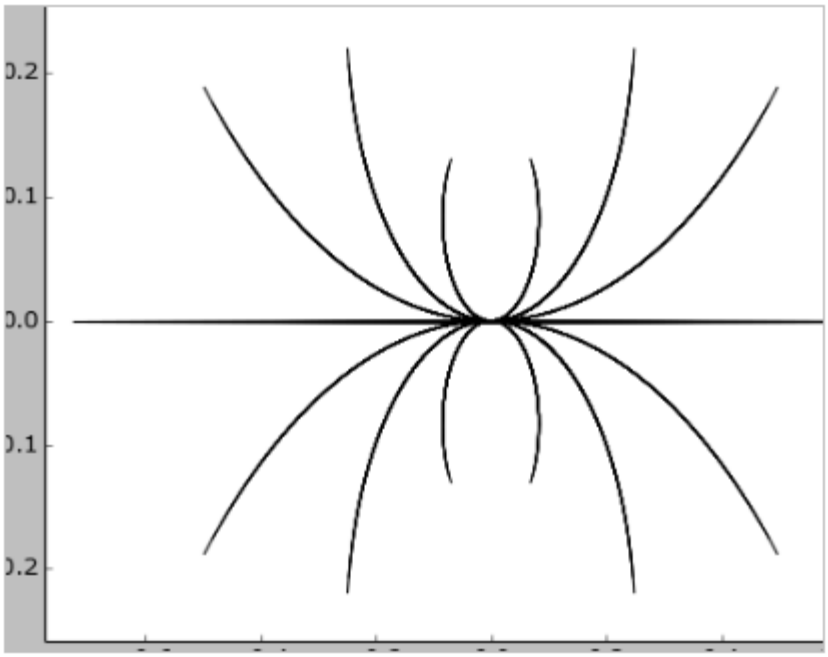
We use the following setting to design our control set:
$w_l = \{-1, -0.5, 0, 0.5, 1\}$

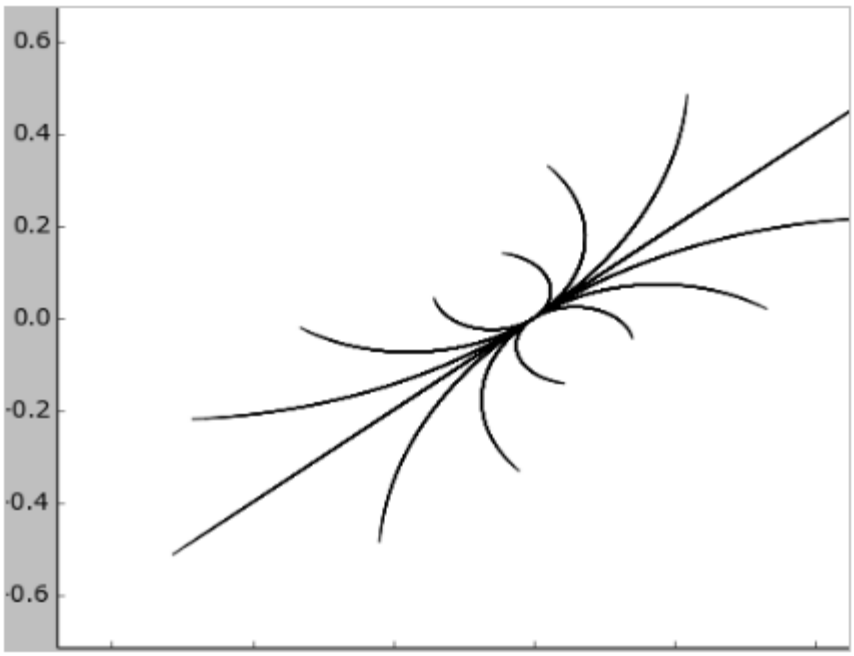
$$w_r = \{-1, -0.5, 0, 0.5, 1\}$$

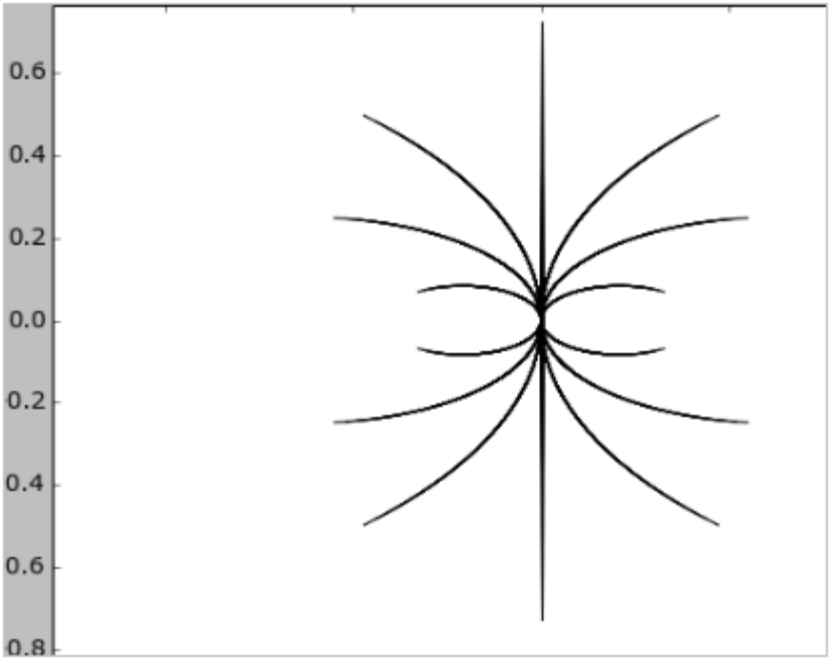
Duration = 0 to 3 seconds with 0.1 seconds interval

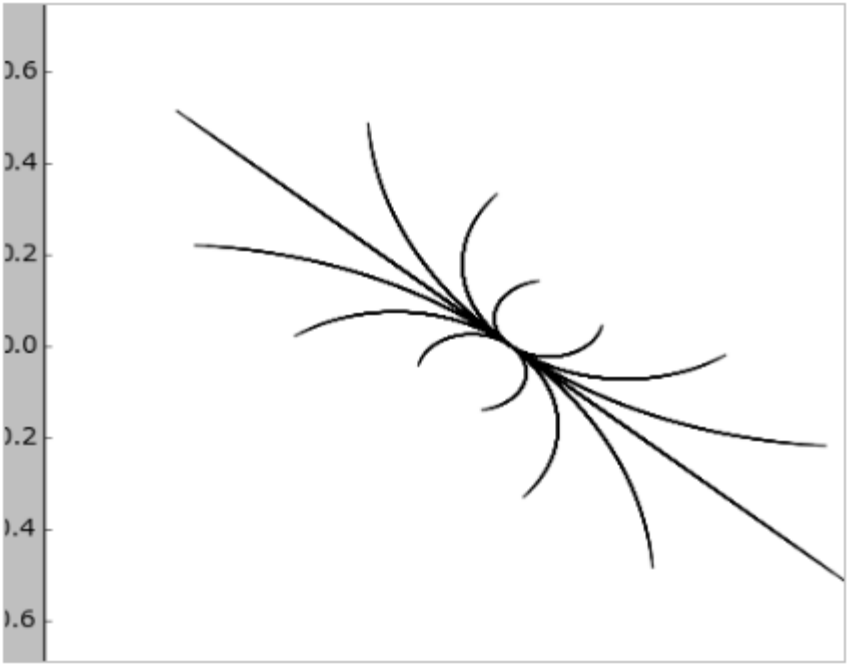
If the control set is too simple, it limits the maneuverability of the robot. However, if the control set is too complicated, it is hard to design evaluation metrics to select the optimal trajectory. The reason why we choose this setting is that we try to select a moderate size of the control set.

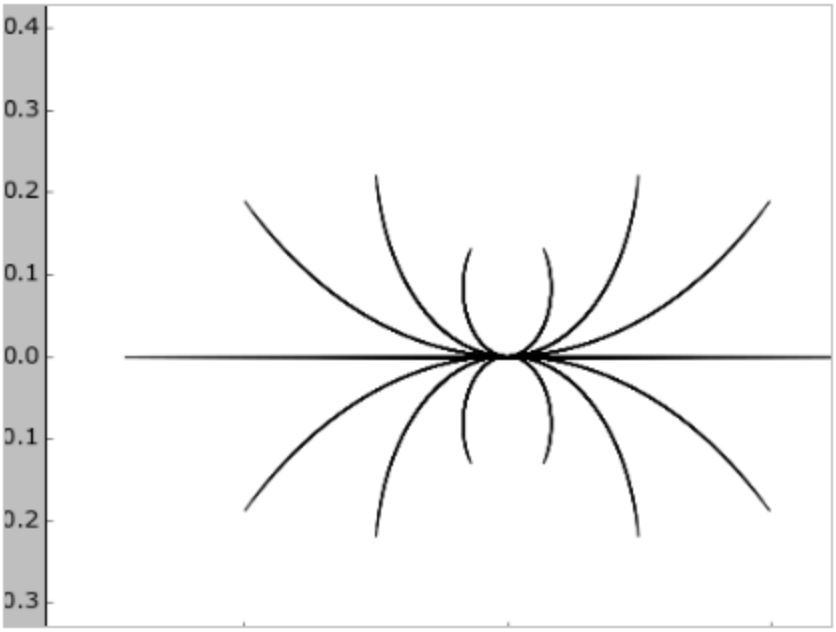
The figures below show the trajectories of our control set in different orientation.

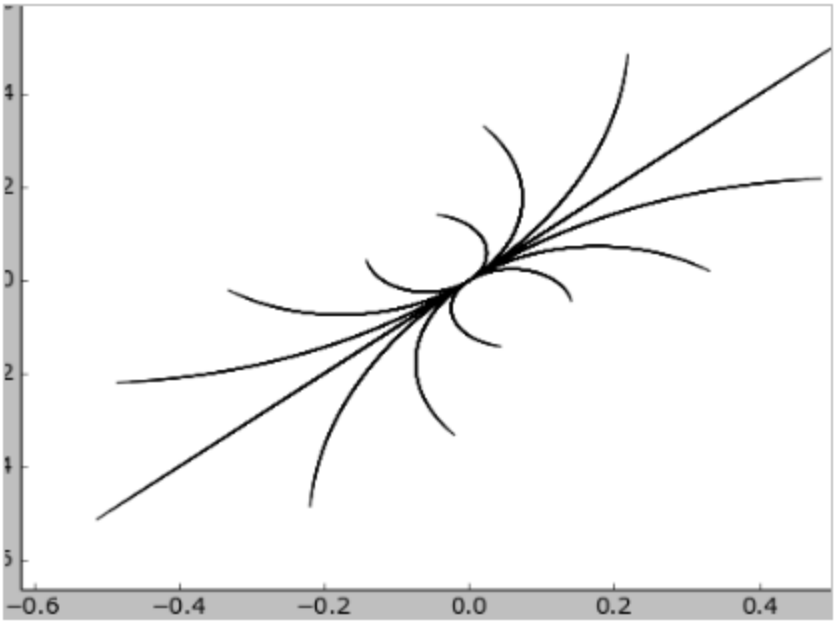

0.2
0.1
0.0
0.1
0.2

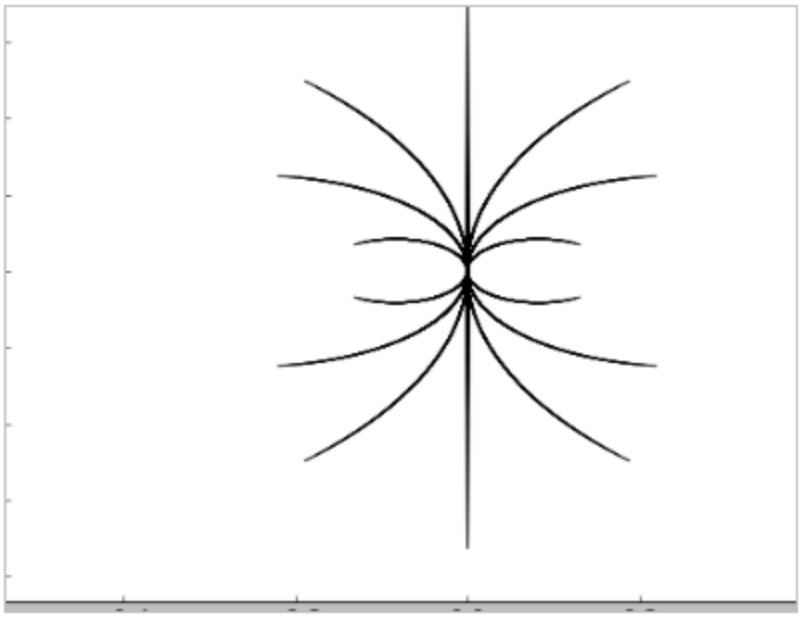

0.6
0.4
0.2
0.0
-0.2
-0.4
-0.6

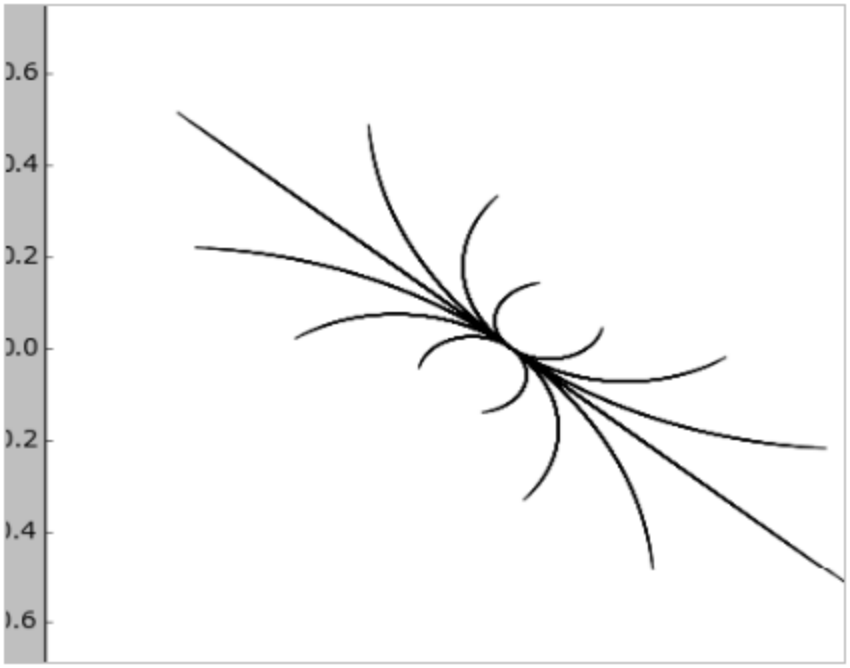

0.6
0.4
0.2
0.0
0.2
0.4
0.6
0.8


0.6
0.4
0.2
0.0
0.2
0.4
0.6


0.4
0.3
0.2
0.1
0.0
0.1
0.2
0.3


−0.6
−0.4
−0.2
0.0
0.2
0.4




0.6
0.4
0.2
0.0
0.2
0.4
0.6

2).

Configuration 1:
start_config = [-0.5 ,0 ,numpy.pi/2]
goal_config = [-0.5 ,2 ,0]

Configuration 2:
start_config = [-0.5 ,0 ,numpy.pi/2]
goal_config = [2 ,2 ,0]

Configuration 3:
start_config = [-0.5 ,0 ,-numpy.pi/2]
goal_config = [-0.5 ,-2 ,0]

For this part, we modified the “ComputeDistance“ function and “ComputeHeuristicCost” funciton.

In “ComputeDistance“, instead of using Euclidean distance metric, we add a weighting number on orientation trying to let it find the correct orientation as soon as possible.

In “ComputeHeuristicCost”, we add a weighting number on distance of x and y coordinates trying to let it go close to the goal position.

The current result is acceptable.

3).

We tried a couple of different methods to try and determine the base pose:

- First we tried the inverse reachability functionality built into the Open-RAVE to determine the base pose. One difficulty that we faced this approach was during the database generation. It was taking a huge time to generate the complete inverse reachability and reachability database. For this reason, we did not invest much time in implementing this method and moved on to a more experimental method.
- We manually determine the robot base pose next to the robot from where the robot can successfully grasp the object.

Although a better method to determine the base pose is definitely using the inverse reachability functionality, method 2 also gives acceptable results.

An extension of method 2 can be to determine multiple base poses all around the table and generate the grasp config from each of the base pose, finally select the one from where the robot is successfully able to determine grasp config. This is kind of an automation for method 2 to experimentally determine the base pose.