CS 133b Final Report
Names: Alexander Vazquez
Gael Moran

Project Overview

Our main goal was to compare several variations of RRT, finding which algorithms are best suited for specific environments. More specifically, the algorithms we are analyzing are RRT, RRT March, RRT with variable step size, RRT*, and Bi-directional RRT. Refer to the technical details section below for a more detailed description of these five algorithms.

Approach

The data we collected included the total number of nodes created, runtime, and the path length found across different scenarios for each variation of RRT mentioned in the project overview. The scenarios involved planning paths for a three-DOF planar arm, a four-DOF planar arm, and a 2D obstacle course, each with three maps to traverse. In other words, we ran each algorithm in nine different scenarios/grids. Each algorithm was run 500 times for each grid/scenario corresponding to the 2D point workspace, 200 times for each map corresponding to the three-DOF planar arm workspace, and 200 times for each map corresponding to the four-DOF planar arm workspace. It is important to note that we only counted and collected data on the number of successful runs, which are defined by the algorithm being able to find a path.

We chose these different workspaces as we wanted to analyze how each algorithm changes when the number of dimensions increases. Furthermore, we tested each algorithm on different scenarios for each workspace because we wanted to observe how well each algorithm performed within narrow spaces, complex spaces, and spaces without many obstacles. Refer to Appendix D, to see each scenario/grid for each workspace (2d point, three DOF planar, four DOF planar arm).

Results

RRT (rrt_reg):

This algorithm has a mediocre performance compared to all other algorithms. It is important to note that the regular RRT algorithm never obtained the best performance in any metric, but there were a few cases where it did have the worst. There are observable patterns that tell us RRT has better time measurements when there are more obstacles on the map. Overall, this algorithm can just be described as average, and an appropriate experimental control or base for comparison.

RRT March (rrt_march):

In general, this algorithm falls in the middle of the performance ranking in comparison to the others. It performs slightly better than RRT with variable step size, placing it in the upper half of the ladder. There are several cases where it has the best or second-best time

metric. When there are fewer obstacles in the map, this metric (runtime) tends to outperform most other algorithms.

RRT with variable step size (rrt_var):
Overall, this algorithm has an average performance in all metrics compared to the others. There are a few cases where it has both the best and worst nodes and runtime measurements, but distance always tends to fall somewhere in the middle. Whenever the path between the start and the goal becomes more narrow or has more obstacles, the general trend is that it will have worse runtimes. This makes sense, as the algorithm is designed to have shorter step sizes when approaching walls.

RRT* (rrt_star):
In all workspaces (2d point, three DOFs planar arm, four DOFs, planar arm) and scenarios/grids this algorithm tended to provide a path with a shorter distance than all the other algorithms. Furthermore, it also tended to have the worst average runtimes for all workspaces and scenarios/grids as well. However, it should be noted that it also tended to have the largest variation in the runtime. We believe that this is a result of the randomness of RRT*. The amount of nodes needed for RRT* varies across all workspaces and scenarios, but it was never the best (that is, it always needed more nodes to find a path than any of the other algorithms). Overall, RRT* tended to be best at exploring more of each grid. The tradeoff of this exploration is a longer runtime, but a tendency to find the shortest path among the algorithms tested.

Bi-directional RRT (rrt_bi):
In terms of the distance of the path found, it is often found to be in the lower end compared to the other algorithms. You can regularly observe it to have the worst distance metric in certain scenarios. However, when adding more obstacles to each workspace, the runtime metric significantly improves for this algorithm in comparison to the others, taking it to the top of the ladder (that is, it does not increase as much as the others do). We can conclude that there is a relationship between increasing complexity increasing distance and decreasing time. The number of nodes tends to follow a similar trend to that of distance, but not as severe. Therefore, bi-directional RRT seems to be the best at tracing a path from A to B in the shortest time possible, even if it is not the most optimal.

Conclusions
It is essential to always think about what you want your algorithm to accomplish and how complex your grid/map would be before choosing an appropriate variation of RRT. For example, if your desired goal is to get the shortest path then RRT* would be ideal. Alternatively, if the time tradeoff with RRT* is too much, then you can also use RRT with variable step size if the shortest path tends to be through narrow passages. RRT with

variable step size could also be a good option if you are looking for something close to the most optimal path but have more runtime constraints to worry about.

However, if your desired goal is to find a path in the shortest amount of time, then either Bi-directional RRT or RRT March would be the best choice.
In any case, it would be strongly advisable to never settle for a regular RRT algorithm. While it is an average performer, our experiments showed that RRT will never return the best measurements for nodes, distance, and time. It can perform the worst, however, which is why time must be dedicated to tailoring your algorithms to best fit your specific task.

Technical Details

RRT: We use a similar algorithm as presented in citation [1]. We only made two modifications. The first is in how we choose x_{rand}. We made it so that the probability of choosing the goal node is 5 percent. The second being that we stopped the algorithm once a path from the start node x_{init} to the goal node was found.

Algorithm 3: RRT.

$V \leftarrow \{x_{\text{init}}\}; E \leftarrow \emptyset;$
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i;$
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}});$
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}});$
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \cup \{x_{\text{new}}\}; E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\};$
return $G = (V, E);$

RRT March: We used the same algorithm as we did for RRT but with a modification. The modification is that instead of taking a single step towards the chosen target node, we take as many steps as we can (that is until we either reach the target node or until there is an obstacle in the way).

RRT with variable step size: We used the same algorithm as we did for RRT but with a modification. The step size changed depending on how close the grow node was to an obstacle. That is we choose the step size to be min(MIN_DSTEP, K * Dist) where MIN_DSTEP

is a minimum step size, and K * Dist is a factor of the distance between the grow node and the closest obstacle.

RRT*: We use a similar algorithm as presented in citation [1]. Like before, we only made two modifications. The first being in how we choose x_{rand}. We made it so that the probability of choosing the goal node as x_{rand} is 5 percent. The second being that we stopped the algorithm once a path from the start node x_{init} to the goal node was found.

Algorithm 6: RRT*.

$V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$;
for $i = 1, \ldots, n$ **do**
  $x_{\text{rand}} \leftarrow \texttt{SampleFree}_i$;
  $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G = (V, E), x_{\text{rand}})$;
  $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x_{\text{rand}})$;
  if $\texttt{ObtacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \texttt{Near}(G = (V, E), x_{\text{new}}, \min\{\gamma_{\text{RRT}^*}(\log(\text{card}(V))/\text{card}(V))^{1/d}, \eta\})$;
    $V \leftarrow V \cup \{x_{\text{new}}\}$;
    $x_{\text{min}} \leftarrow x_{\text{nearest}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{nearest}}) + c(\texttt{Line}(x_{\text{nearest}}, x_{\text{new}}))$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Connect along a minimum-cost path
      if $\texttt{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}})) < c_{\text{min}}$ **then**
        $x_{\text{min}} \leftarrow x_{\text{near}}$; $c_{\text{min}} \leftarrow \texttt{Cost}(x_{\text{near}}) + c(\texttt{Line}(x_{\text{near}}, x_{\text{new}}))$
    $E \leftarrow E \cup \{(x_{\text{min}}, x_{\text{new}})\}$;
    foreach $x_{\text{near}} \in X_{\text{near}}$ **do** // Rewire the tree
      if $\texttt{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge \texttt{Cost}(x_{\text{new}}) + c(\texttt{Line}(x_{\text{new}}, x_{\text{near}})) < \texttt{Cost}(x_{\text{near}})$ **then** $x_{\text{parent}} \leftarrow \texttt{Parent}(x_{\text{near}})$;
      $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$
return $G = (V, E)$;

Bi-directional RRT: We used a similar algorithm as presented in citation [2]. Like before, we only made two modifications. The first is in how we choose x_{rand}. We made it so that the probability of choosing the goal node as x_{rand} is 5 percent when growing the tree with the start node as the root, and the probability of choosing the start node as x_{rand} is 5 percent when growing the tree with the goal node as the root. The second being that we stopped the algorithm once a path from the start node x_{init} to the goal node was found.

RRT_BIDIRECTIONAL(x_{init}, x_{goal})
1 $\mathcal{T}_a$.init(x_{init}); $\mathcal{T}_b$.init(x_{goal});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 **if not** (EXTEND($\mathcal{T}_a, x_{rand}$) = *Trapped*) **then**
5 **if** (EXTEND($\mathcal{T}_b, x_{new}$) = *Reached*) **then**
6 Return PATH($\mathcal{T}_a, \mathcal{T}_b$);
7 SWAP($\mathcal{T}_a, \mathcal{T}_b$);
8 Return *Failure*

Citations

[1] Karaman S, Frazzoli E. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*. 2011;30(7):846-894. doi:10.1177/0278364911406761

[2] Faiza Gul and Wan Rahiman 2019 *IOP Conf. Ser.: Mater. Sci. Eng.* **697** 012022

Code:

https://drive.google.com/drive/folders/1qCZl6Sorf-6iBc_WY_9tEh-i5hsvLhtb?usp=sharing

Example Command: python rrt_robot_three_all.py grid2 1 rrt_star

Videos

Three Links:
https://drive.google.com/file/d/10OhFAsbOiCnUuiXAh3UGbojgVDH0xV91/view?usp=sharing

Four Links:
https://drive.google.com/file/d/1v1zidQ4IquRh0OEbSAV3SDCi-1A3RXcj/view?usp=sharing

2D Map:
https://drive.google.com/file/d/1lxMjRp1UyDd_rGe0Hyam64MNR9EQYHwG/view?usp=sharing

APPENDIX A: FOUR LINKS DATA

Grid 1 Nodes:

algorithm	mean	std	range	median
rrt_reg	97.54	58.423	(9, 286)	84.0
rrt_march	94.235	60.590	(9, 287)	79.5
rrt_var	92.215	56.217	(10, 323)	84.0
rrt_star	91.575	55.935	(7, 349)	85.5
rrt_bi	108.845	54.997	(14, 340)	103.0

Grid 1 Dist:

algorithm	mean	std	range	median
rrt_reg	14.614	2.846	(9.536, 29.447)	14.179
rrt_march	15.127	3.083	(9.223, 28.969)	14.354
rrt_var	14.940	2.656	(9.825, 25.468)	14.569
rrt_star	11.278	55.935	(8.856, 14.730)	11.245
rrt_bi	18.081	3.556	(10.728, 28.862)	18.069

Grid 1 Time:

algorithm	mean	std	range	median
rrt_reg	0.429	0.260	(0.059, 1.975)	0.376
rrt_march	0.442	0.256	(0.064, 1.302)	0.376
rrt_var	0.444	0.251	(0.064, 1.542)	0.410
rrt_star	69.132	85.106	(0.207, 565.932	40.420
rrt_bi	0.504	0.226	(0.092, 1.500)	0.483

Grid 2 Nodes:

algorithm	mean	std	range	median
rrt_reg	133.32	140.085	(3, 877)	91.0
rrt_march	123.19	146.990	(3, 1037)	72.5
rrt_var	130.06	140.943	(4, 797)	91.5
rrt_star	129.72	129.593	(4, 797)	84.0
rrt_bi	58.78	48.725	(4, 374)	44.0

Grid 2 Dist:

algorithm	mean	std	range	median
rrt_reg	13.329	3.970	(7.626, 28.174)	12.255
rrt_march	13.151	3.268	(7.525, 26.632)	12.699
rrt_var	13.612	3.776	(7.328, 30.103)	13.032
rrt_star	9.350	1.343	(6.987, 19.224)	9.218
rrt_bi	17.031	5.808	(7.693, 37.740)	15.663

Grid 2 Time:

algorithm	mean	std	range	median
rrt_reg	0.627	0.602	(0.024, 4.005)	0.441
rrt_march	0.516	0.571	(0.021, 4.479)	0.341
rrt_var	0.568	0.569	(0.020, 3.428)	0.431
rrt_star	71.442	148.516	(0.056, 1214.404)	15.066
rrt_bi	0.427	0.298	(0.032, 2.189)	0.342

Grid 3 Nodes:

algorithm	mean	std	range	median
rrt_reg	31.245	30.027	(4, 211)	22.0
rrt_march	28.71	26.380	(4, 145)	20.0
rrt_var	33.52	34.312	(4, 223)	24.5
rrt_star	39.95	43.395	(4, 377)	24.0
rrt_bi	39.835	21.092	(6, 123)	37.0

Grid 3 Dist:

algorithm	mean	std	range	median
rrt_reg	12.699	3.297	(8.138, 27.235)	11.680
rrt_march	12.755	3.274	(8.343, 26.698)	12.022
rrt_var	12.925	3.637	(7.760, 31.425)	11.897
rrt_star	10.291	1.814	(6.828, 20.741)	10.001
rrt_bi	16.158	4.613	(8.594, 32.114)	15.220

Grid 3 Time:

algorithm	mean	std	range	median
rrt_reg	0.220	0.152	(0.024, 0.980)	0.185
rrt_march	0.211	0.143	(0.033, 0.817)	0.176
rrt_var	0.307	0.222	(0.025, 1.350)	0.261
rrt_star	4.885	13.468	(0.059, 171.004)	1.302
rrt_bi	0.308	0.133	(0.061, 0.877)	0.294

APPENDIX B: 2D Point

Grid 1 Nodes:

Algorithm	Mean	Std	Range	Median
rrt_var	341.55	69.308	(203,551)	338.0
rrt_bi	458.84	74.235	(267,769)	451.0
rrt_star	508.132	95.211	(251,831)	499.0
rrt_reg	500.956	90.126	(274,809)	498.0
rrt_march	245.65	74.053	(102,481)	238.5

Grid 1 Dist:

Algorithm	Mean	Std	Range	Median
rrt_var	20.695	1.254	(18.212,25.842)	20.548
rrt_bi	21.027	1.107	(18.519,24.638)	20.884
rrt_star	17.605	0.479	(16.812,19.874)	17.516
rrt_reg	20.578	1.034	(18.234,23.904)	20.465
rrt_march	20.767	1.214	(17.821,24.928)	20.721

Grid 1 Time:

Algorithm	Mean	Std	Range	Median
rrt_var	0.076	0.023	(0.035,0.164)	0.073
rrt_bi	0.108	0.03	(0.037,0.245)	0.104
rrt_star	0.236	0.078	(0.07,0.611)	0.221
rrt_reg	0.122	0.039	(0.043,0.316)	0.118
rrt_march	0.014	0.01	(0.002,0.067)	0.012

Grid 2 Nodes:

Algorithm	Mean	Std	Range	Median
rrt_march	196.602	87.428	(60,521)	181.0
rrt_reg	252.832	52.241	(161,524)	242.5
rrt_star	260.472	51.429	(162,492)	250.0
rrt_bi	124.552	26.59	(77,261)	118.0
rrt_var	112.532	37.63	(42,261)	108.5

Grid 2 Dist:

Algorithm	Mean	Std	Range	Median
rrt_march	19.243	2.593	(14.51,29.36)	18.825
rrt_reg	17.854	0.871	(16.229,22.244)	17.735
rrt_star	15.745	0.649	(14.736,18.378)	15.594
rrt_bi	18.027	1.03	(15.883,22.179)	17.839
rrt_var	18.718	1.69	(15.731,26.602)	18.394

Grid 2 Time:

Algorithm	Mean	Std	Range	Median
rrt_march	0.006	0.004	(0.001,0.024)	0.005
rrt_reg	0.022	0.008	(0.011,0.07)	0.02
rrt_star	0.053	0.017	(0.025,0.152)	0.049
rrt_bi	0.007	0.003	(0.003,0.025)	0.006
rrt_var	0.01	0.004	(0.003,0.028)	0.009

Grid 3 Nodes:

Algorithm	Mean	Std	Range	Median
rrt_march	381.328	151.684	(52,841)	380.0
rrt_reg	742.846	141.106	(391,1390)	740.5
rrt_star	748.676	148.769	(424,1337)	739.0
rrt_bi	270.118	58.932	(81,502)	266.0
rrt_var	1078.968	202.83	(607,2022)	1067.5

Grid 3 Dist:

Algorithm	Mean	Std	Range	Median
rrt_march	27.398	2.758	(22.524,41.855)	26.877
rrt_reg	27.567	1.227	(24.677,32.931)	27.535
rrt_star	24.735	0.721	(23.208,27.175)	24.64
rrt_bi	27.228	1.429	(24.575,38.531)	27.123
rrt_var	27.56	1.45	(24.075,38.791)	27.392

Grid 3 Time:

Algorithm	Mean	Std	Range	Median
rrt_march	0.046	0.034	(0.001,0.211)	0.038
rrt_reg	0.18	0.063	(0.054,0.575)	0.174
rrt_star	0.331	0.125	(0.113,0.954)	0.314
rrt_bi	0.039	0.014	(0.012,0.113)	0.037
rrt_var	0.468	0.156	(0.17,1.386)	0.448

APPENDIX C: THREE LINKS

Grid1 Nodes:

Algorithm	Mean	Std	Range	Median
rrt_bi	43.1	22.865	(6,147)	40.0
rrt_march	37.125	23.74	(5,158)	35.0
rrt_reg	37.17	26.152	(4,159)	31.0
rrt_star	38.305	23.558	(5,124)	33.5
rrt_var	35.47	24.363	(5,118)	29.0

Grid1 Dist:

Algorithm	Mean	Std	Range	Median
rrt_bi	12.266	2.191	(7.66,22.172)	11.878
rrt_march	10.995	1.984	(7.542,19.84)	10.61
rrt_reg	10.906	1.831	(7.265,18.985)	10.435
rrt_star	8.991	0.945	(6.463,11.926)	9.027
rrt_var	10.714	1.888	(7.662,17.657)	10.391

Grid1 Time:

Algorithm	Mean	Std	Range	Median
rrt_bi	0.143	0.065	(0.021,0.417)	0.131
rrt_march	0.108	0.06	(0.021,0.42)	0.101
rrt_reg	0.11	0.062	(0.017,0.38)	0.098
rrt_star	7.288	9.481	(0.035,64.297)	4.024
rrt_var	0.143	0.081	(0.028,0.414)	0.121

Grid2 Nodes:

Algorithm	Mean	Std	Range	Median
rrt_bi	27.945	24.881	(5,250)	21.0
rrt_march	48.955	47.122	(4,224)	30.5
rrt_reg	47.6	49.472	(4,235)	31.0
rrt_star	44.575	40.548	(4,213)	30.0
rrt_var	43.175	53.379	(4,486)	26.5

Grid2 Dist:

Algorithm	Mean	Std	Range	Median
rrt_bi	10.158	2.843	(6.267,22.615)	9.541
rrt_march	9.973	2.811	(5.566,21.587)	9.258
rrt_reg	9.313	1.951	(6.161,20.024)	9.066
rrt_star	7.766	1.014	(5.96,12.292)	7.608
rrt_var	9.57	2.521	(6.283,20.954)	8.84

Grid2 Time:

Algorithm	Mean	Std	Range	Median
rrt_bi	0.165	0.105	(0.036,0.914)	0.14
rrt_march	0.203	0.15	(0.027,0.733)	0.157
rrt_reg	0.199	0.156	(0.022,0.743)	0.155
rrt_star	5.437	9.585	(0.042,70.02)	1.534
rrt_var	0.23	0.215	(0.025,2.109)	0.171

Grid3 Nodes:

Algorithm	Mean	Std	Range	Median
rrt_bi	33.165	22.705	(5,151)	29.0
rrt_march	48.895	47.517	(4,306)	33.0
rrt_reg	61.52	59.055	(4,320)	41.0
rrt_star	56.07	55.682	(4,301)	38.0
rrt_var	56.325	55.722	(4,302)	37.0

Grid3 Dist:

Algorithm	Mean	Std	Range	Median
rrt_bi	10.331	1.947	(7.399,17.487)	9.813
rrt_march	10.032	2.037	(6.978,20.956)	9.593
rrt_reg	9.925	1.772	(7.291,15.514)	9.433
rrt_star	8.163	0.647	(6.846,11.02)	8.125
rrt_var	9.961	1.964	(7.169,20.116)	9.577

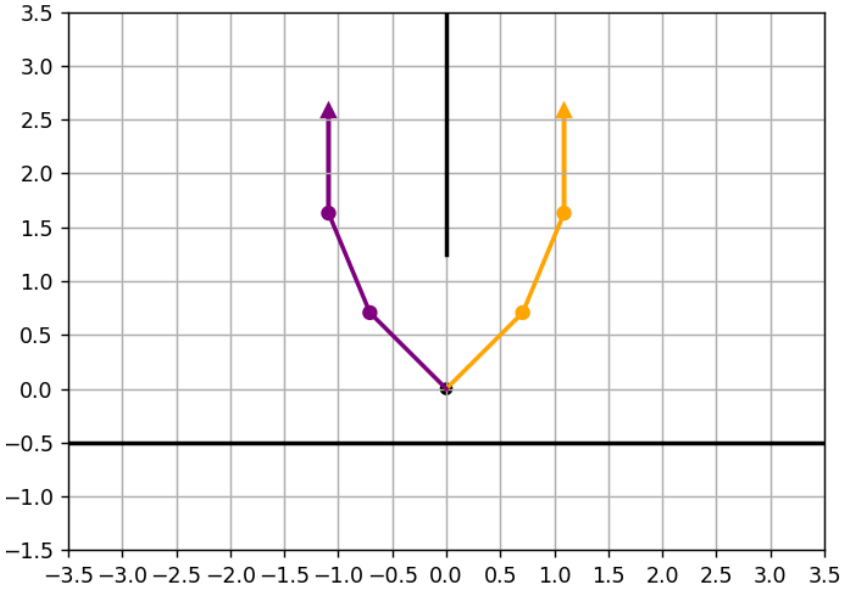
Grid3 Time:

Algorithm	Mean	Std	Range	Median
rrt_bi	0.154	0.091	(0.021,0.692)	0.141
rrt_march	0.16	0.135	(0.017,0.906)	0.117
rrt_reg	0.178	0.144	(0.02,0.82)	0.13
rrt_star	10.671	20.301	(0.047,133.043)	3.179
rrt_var	0.208	0.165	(0.018,0.891)	0.162

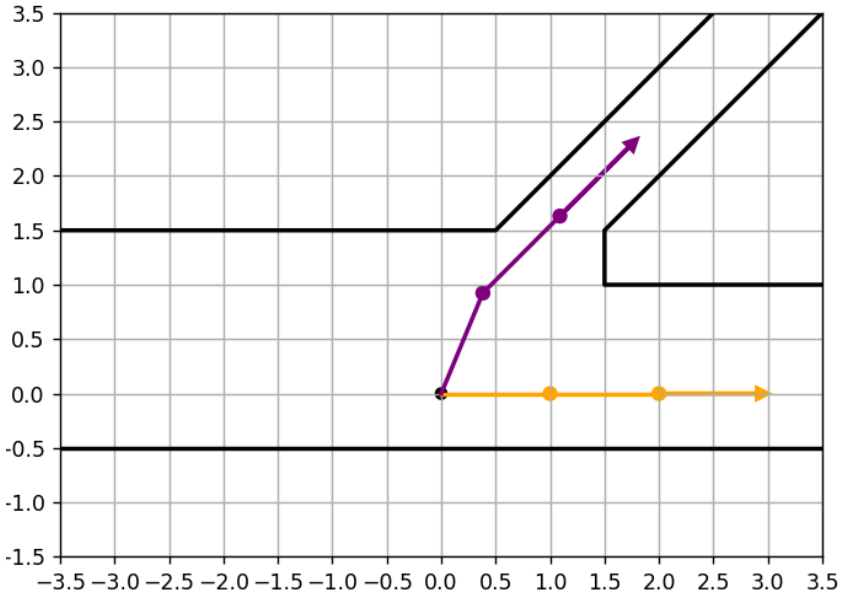
APPENDIX D: Grids/Scenario For Each Workspace

Three DOFs Planar Arm:

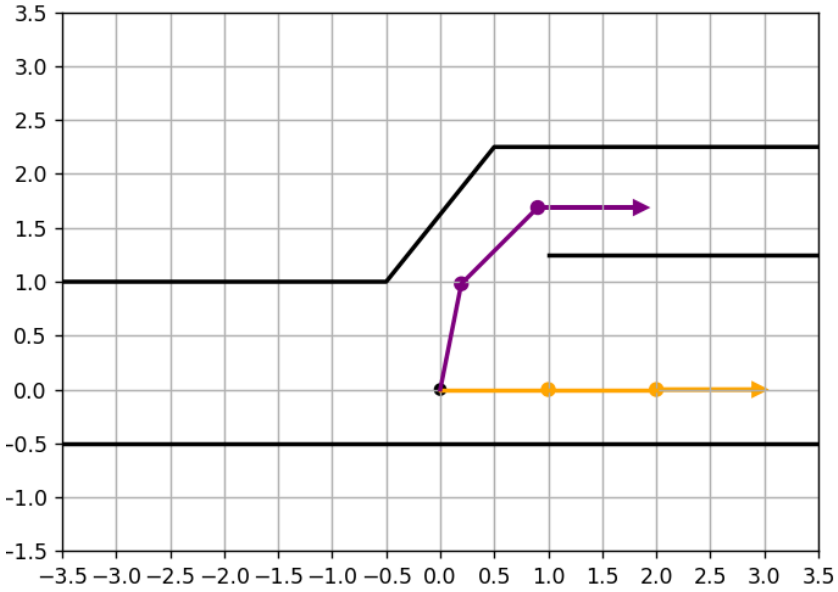
Grid 1:



Grid 2:

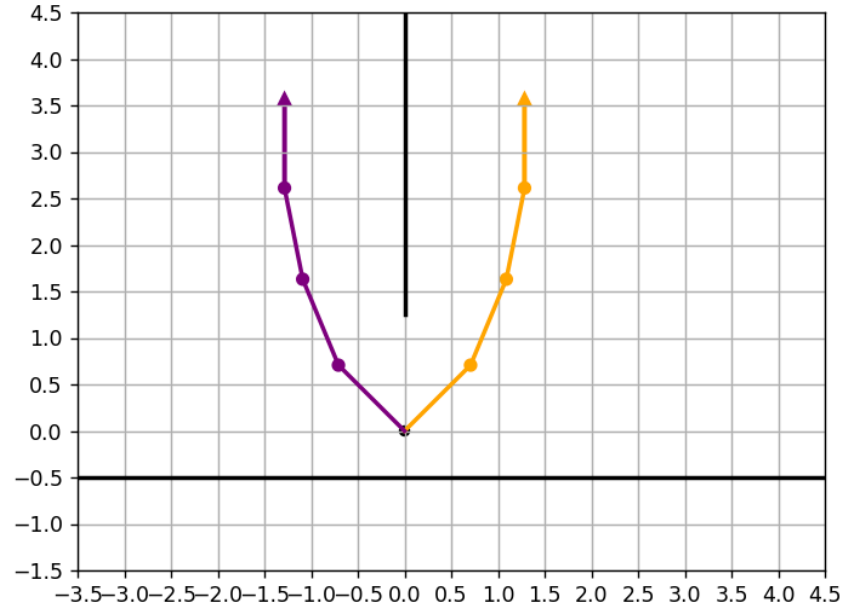


Grid 3:

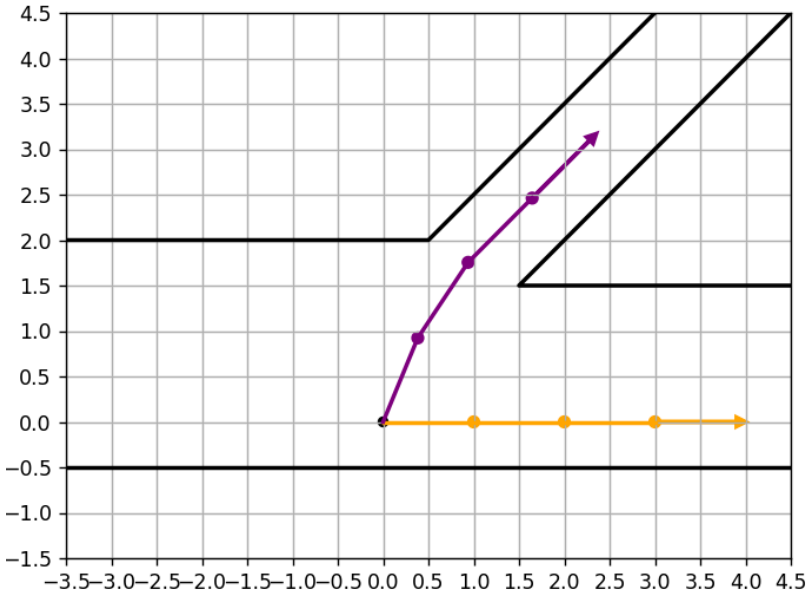


Four DOFs Planar Arm:

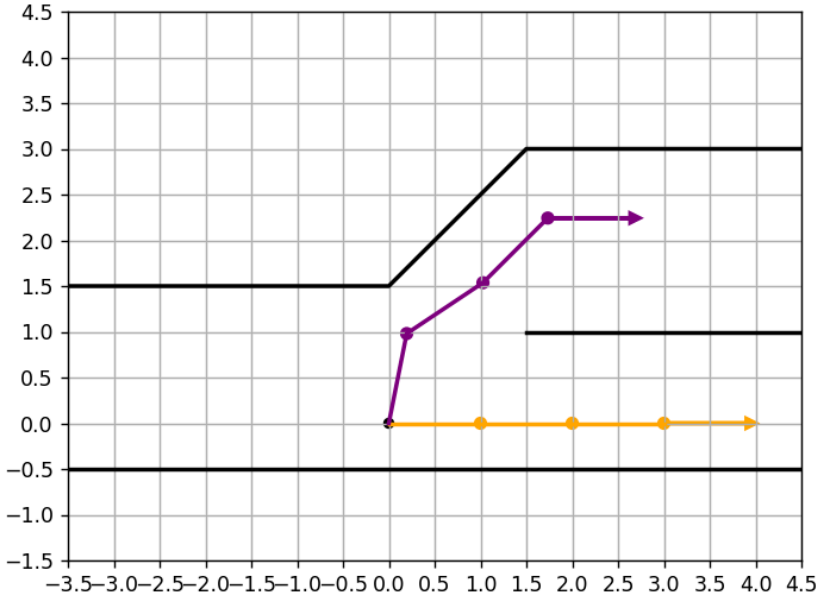
Grid 1:



Grid 2:

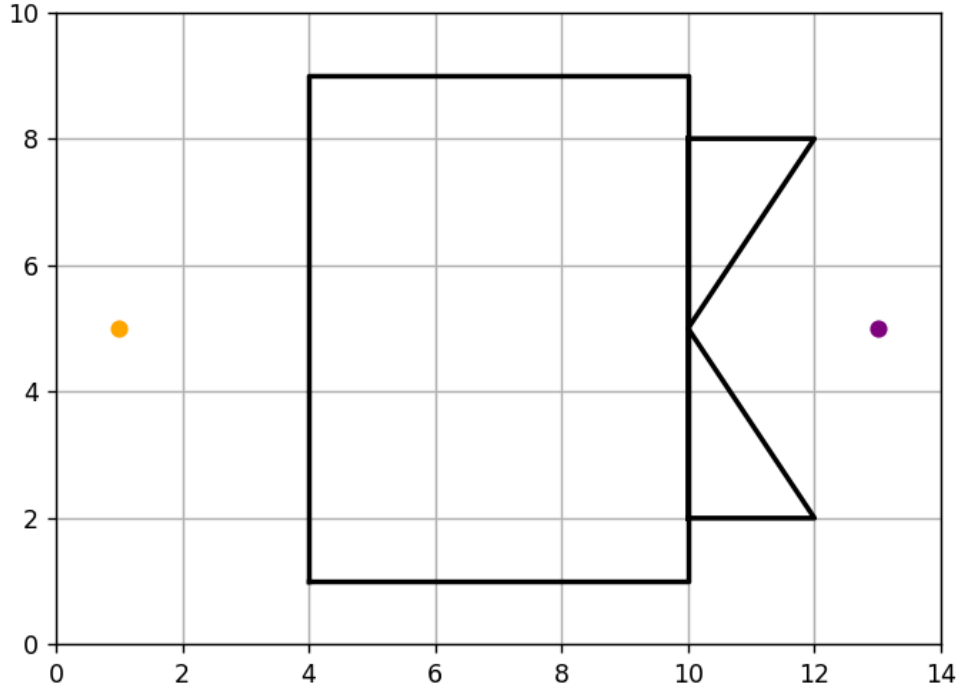

4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3.5 −3.0 −2.5 −2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5

Grid 3:

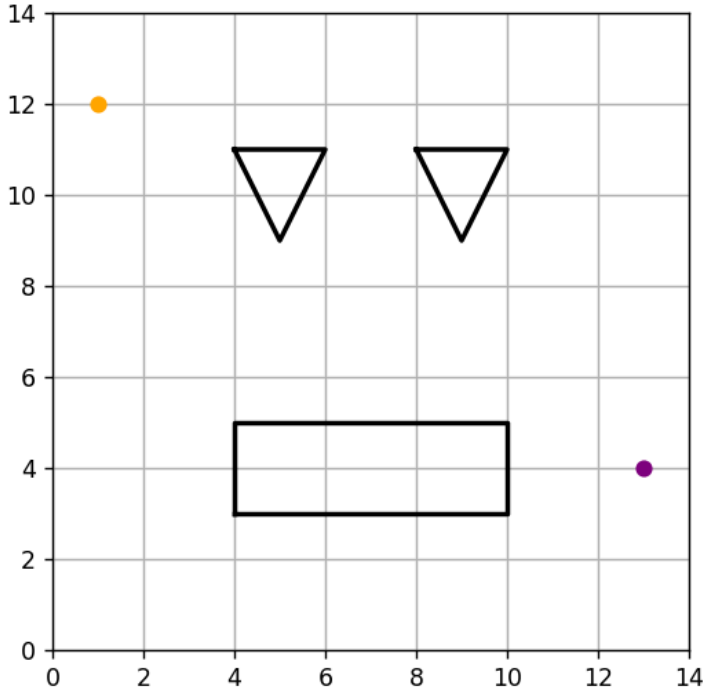

4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3.5 −3.0 −2.5 −2.0 −1.5 −1.0 −0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5

2D Obstacles:

Grid 1:



Grid 2:



Grid 3: