4 0)

check_collision.m

This function is checking if any given configuration (q) places the arm links in collision with the obstacles in workspace. The function is first calculating the translational vectors w.r.t to the base frame. These translational vectors represents the edge points on each link. The difference between these two vectors on a link will give a direction vector in their direction. ~~of~~ Then, a set of points which are linearly spaced (in this case 9) points b/w start & end points of a link. For each of these points, norm between the spherical obstacles centres and the point itself are calculated. If the norm is greater than the sum of link radius & spherical obstacle's radius then the ~~link~~ ~~p~~ point on link is not in collision with the obstacle. This is performed for all the points on links to determine if the arm is in collision with any obstacles in the workspace.

check_edge.m : This function takes start configuration & goal configuration as inputs and determines if the trajectory between these configurations is in collision with the spherical obstacles. First, the function generates linearly spaced waypoints (in this case 9) between the start and goal configurations. Each of these waypoint configurations are passed to check_collision.m function to check if any link in collision. If all the waypoints are not in collision with any obstacle, then the edge connecting the start & goal configuration is collision free.

Issues with these algorithms:

1. These algorithms always consider linearly spaced points to check if they are collision free. There is a chance that points ~~between~~ that are not among selected points but lie on the links and in collision with the obstacles. If the spacing between the linearly spaced points is small, then this problem can be avoided.

2. These algorithms only work iff the obstacles are spherical, for other shaped obstacles there is a high chance of collision. One way to make this algorithm still work on nonspherical obstacle is to enclose obstacles in imaginary spheres.

3. check_edge function always considers, that only linear moments are made between the start & goal configurations. In some cases, this may not give the shortest path.

C0.

(a) The two axis of configuration space are

Cspace x-axis (configurations robot 'B' can take in (x-y workspace)

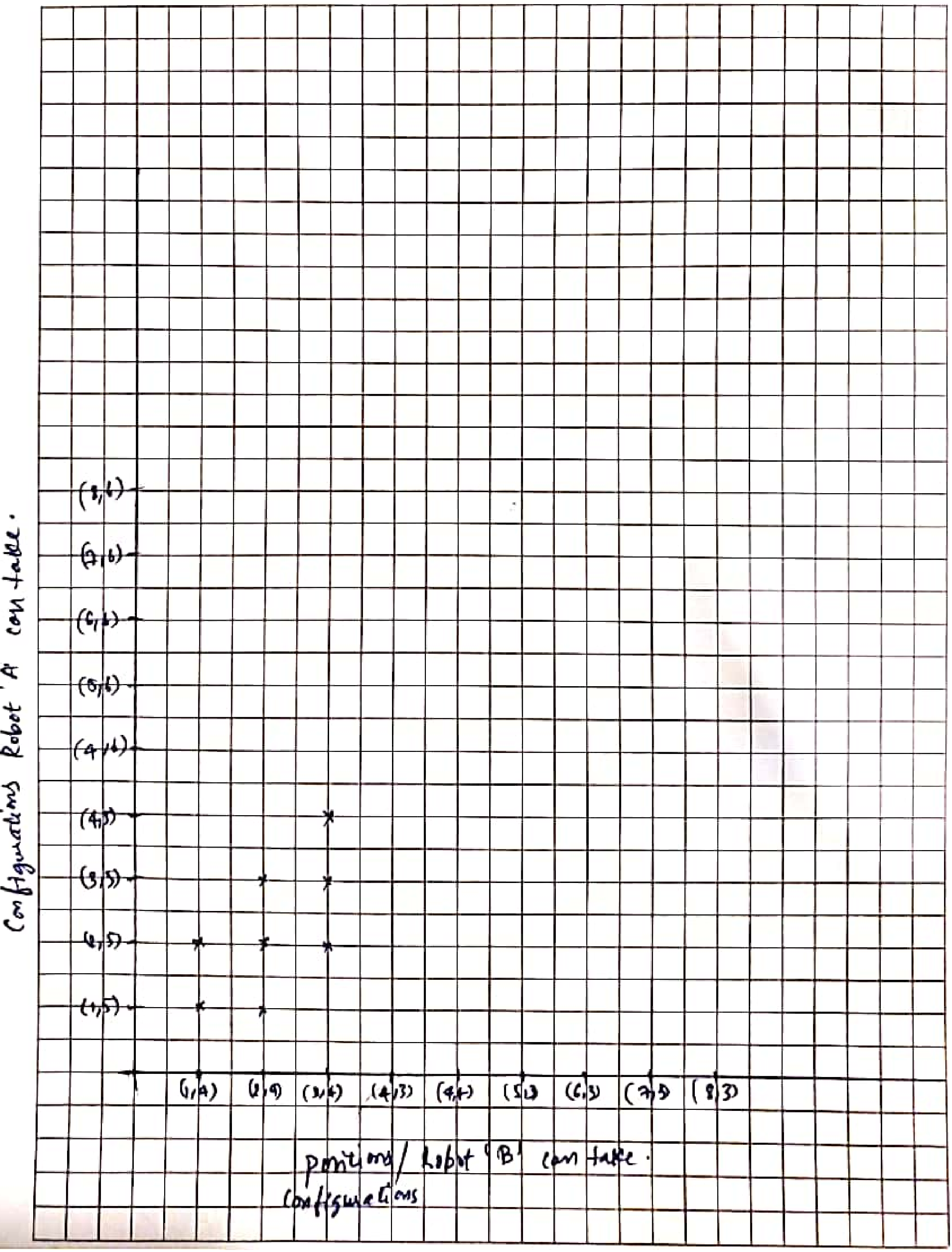
Cspace y-axis (configurations robot 'A' can take in (x-y) workspace)

Limits of c-space x axis are (1,4) & (12, 4)

limits of c-space y-axis are (1,5) & (12,5)

(b) ⓧ → cross in the plot represents the configurations robot 'A' & 'B' cannot take at the same time. ~~Axis~~

Configurations like (5,4), (8,4) on cspace x-axis and (5,5), (8,5) on cspace y-axis are avoided or not represented in the plot as these are not achieveable because of the obstacle in workspace c O



Configurations Robot 'A' can take.
(8,6)
(7,6)
(6,6)
(5,6)
(4,6)
(4,5)
(3,5)
(2,5)
(1,5)
(1,4)
(2,4)
(3,4)
(4,3)
(4,4)
(5,3)
(6,3)
(7,3)
(8,3)
positions/ Robot 'B' can take.
configurations