Simplified Robotic System

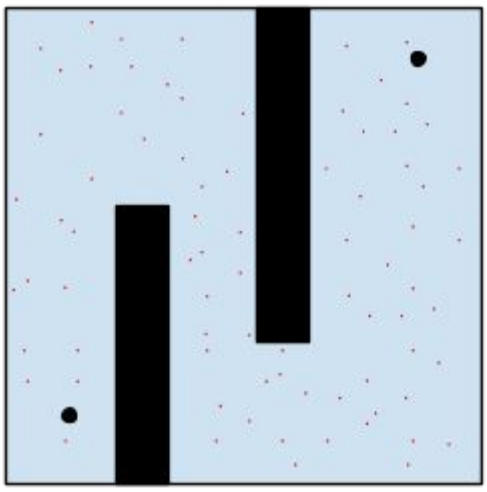
The goal of this system is to provide an approach for moving an object (the robot) with a multi segmented arm from a start location to a goal in a space contained by obstacles. This problem is derived from a real world example: a robot with arm moving from one location to another, avoiding obstacles in the process. The problem specifies an environment in which the world is static, continuous, and fully observable.

The problem can be broken into three modular components. Firstly, converting from a continuous to a discretised world space, called the universe process (or c-space in the literature). Secondly, creating relationships between arm configs form discrete space, called the graph process. Thirdly, a search algorithm must be performed, to find a solution from the start to the goal arm config, called the tree process.

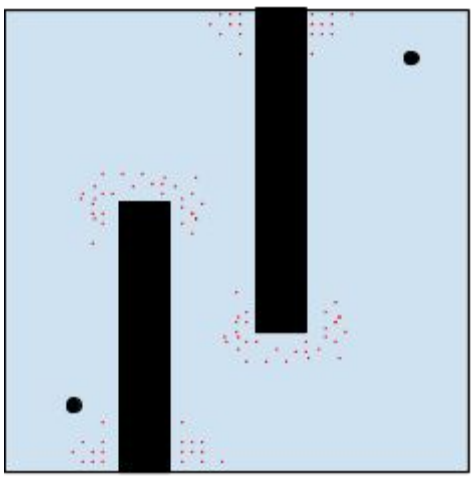
The universe process

From the environment specification, the world is continuous over [0,1] x [0,1]. But it is impossible to process a continuous space (because by definition there are infinitely many points in this space) therefore it must be discretised. Our team approached this problem by combining two algorithms. The first algorithm sampled 2000 random points in the sample space (a PRM approach). The second algorithm is more specific; the goal is to produce points at locations valuable to searching algorithm. Therefore, the algorithm focuses on producing arm configurations at corners of obstacles. This is done by sampling 100 random points, at locations around the corners of all obstacles.

In short, the goal of the universe is to produce all the possible (valuable) points, creating a discrete workspace, to be passed to the next component - the graph process.



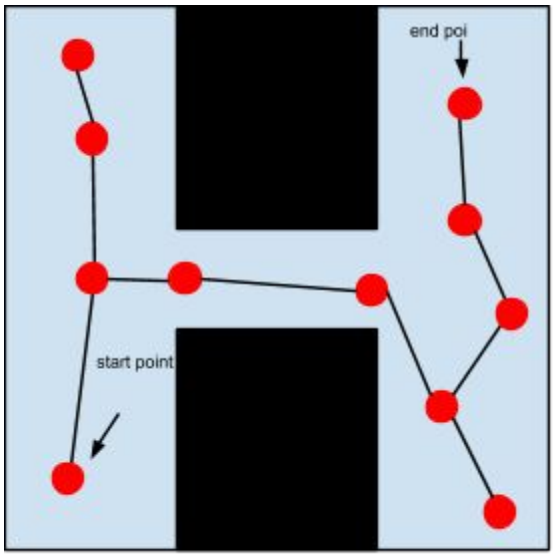
Random Algorithm



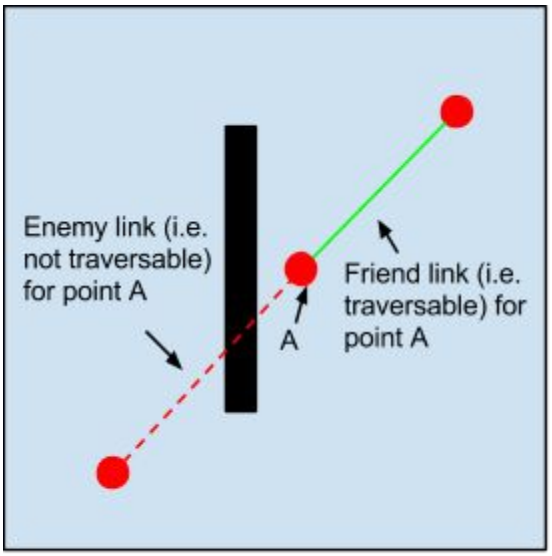
Corner Algorithm

The graph process

This part of the algorithm creates relationships to points, this is the precursor to the next component and does lots of its processing. By definition a graph is a representation of a set of objects where some pairs of objects are connected by links. This is exactly the case here. The points from the universe are given, links, just need to be generated between them. The algorithm generates several type of links for each point, these are: friend, the list of points this point can get to directly, and enemies, the list of point which cannot be traversed directly.
So for example, Given this sample domain:



a representation of the graph generated



a representation of the friend and enemy relation to point A

Once the graph is generated, A* can now be performed.

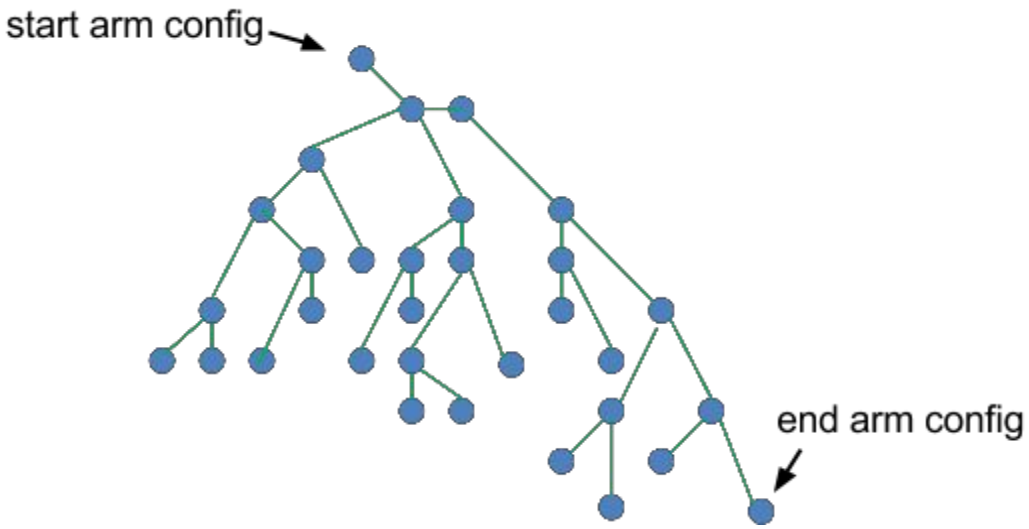
Local Planner

The local planner is an important tool for the creation of the graph. It determines whether or not two nodes are “friends” (can reach each other) or “enemies” (the opposite). To do this we need a local planner to say if we can get from point A to point B.
The local planner module takes the sections of movement taken from the path finding module and attempts to move along the path using different arm configurations. If the local planner module is unable to find a viable path the the end of the section then it returns null to path finding which considers the node to be unreachable. As this module relies on the idea that the section can be moved in a straight line by a robot with no arms it only focuses on the arm configurations and speed of movement, not direction.

The local planner module first attempts a series of base cases for the path segment, simple movement is calculated before advanced movement to lower the computational overhead for movements which are relatively easy. The simple movement base cases are naive movement, base first, arm first and the shotgun approach. The naive movement consists of rotating the starting arm configuration to match the goal arm configuration as it moves along the path. Base first and arm first move the arm and base separately, moving either the base or the arm first respectively. The shotgun approach samples random arm configurations and attempts to move to the goal using arm first movement.

The Tree Process

Once a graph is generated, A solution from the start to then end points must now be performed. To be completed by the A* search algorithm, which finds the shortest path from the start to the goal point. A*, can be thought of as generating a tree from the start to the goal arm configs. The tree can be thought of as an abstract tree, which is essentially independent from the workspace.



Limitations

The random element of the algorithm was quite exhaustive and a threshold had to be added so that it could run in less than one minute. The threshold combined with the random generation of nodes caused the algorithm to occasionally fail, on this event it would keep taking new samples from the universe until the goal had been reached. With tests that have a large number of arms

the A* search linkage would break resulting in false positives, this was an error with the implementation and not the concept.

Automated Testing

We used an automated test script to check our algorithm to ensure it was always solving each base case. We also checked the solution files against the provided Tester function.

```
#1/bin/bash
count=1
bashcmd="./a1-3702"
soln="../result/solution"
filetyp="txt"
for file in ../test/*.txt
do
    echo "now testing "$file
    $bashcmd $file $soln$count.$filetyp
    ((count++))
done
```