This is my preliminary idea for path finding algorithm, and I am working on the code now:

At the first 30 seconds or less, the robot should try to explore the entire field of our side and locate all stacks of cubes on our side.

An important note on the software design in general: There should be at least two non-blocking threads: computer vision and path finding, and a shared map data structure between these two threads representing the field. More threads may be added as necessary.

The computer vision thread is responsible to detect boundaries and cubes, and it writes to the map data structure. The path finding thread is responsible to decide the next **objective** (location) and plan a path to the objective, and it outputs controls to the actuator. Path finding **only** **reads** the map data structure (so synchronization won’t be a problem so far, since it has *eventually consistency*.)

Here is my design of the path finding algorithm:

There should be at least two phases in the game, the exploration phase, and the collection phase (There will be more phases, but these two will spend most of the time). The algorithm used in each phase is different.

Since the boundary will not be finalized until the actual game, the robot assumes *no information* regarding the field at the beginning. Initially it sets up a square lattice of points of interest that it is going to explore, and each lattice point is separated by a constant distance. The path finding algorithm sets the current objective to be the point of interest with the least estimated cost (\* explain later), and it computes a least-cost path to achieve the objective. Since the detected boundary is updated as it moves (because the computer vision thread is updating the map in real time), path finding updates the estimated cost of each unexplored point, and updates its objective to the point with the **current** lowest cost. After enough exploration, the detected boundary forms a closed loop and the path finding algorithm detects there is no path to the remaining vertices, so they can be removed and the exploration phase ends. (Note: this algorithm converges theoretically, assuming perfect detection, so it is impossible that the robot bounces between two points)

During the exploration phase the computer vision thread should be able to detect the locations of all stacks (Note: this gives a constraint on the lattice distance), so the algorithm is simpler in the collection phase: go to the nearest stack, and perform the “collect” action (Note: we haven’t discuss about how to perform collection, but this should be a mechanical/electrical issue, and personally I want to make collection done by pure hardware).