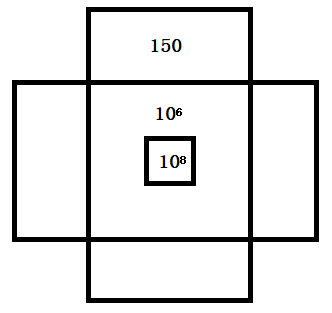
Path Scheduling Algorithm

1. Transform the arena from 2m\*2m square to a 20\*20 grid
2. Only 90 degrees turning is used, so the robot has only 4 possible orientations.
3. With the restriction above, the state of robot can be represented in (x, y, orientation) and x=0~19, y=0~19, orientation=0~3. There are 20\*20\*4 = 1600 different states in total.
4. We transform the problem into that: the robot only occupies 1\*1 cell, but an obstacle occupies roughly 5\*5 cells.
5. For the 20\*20 grid, we give each cell a penalty, based on how risky if we enter this cell.
   * The default penalty is 0
   * The cell that an obstacle locates has penalty 10^8
   * 8 cells around the obstacles have penalty 10^6
   * 12 cells around the obstacle have penalty 150 (see image on the right)
   * The cell at the boundary has penalty 150
6. For each operation that a robot can perform (go straight, turn left, etc.), calculate the change of the state, and the cells that will go through with this operation (exclude the start position). The change of state is represented in forms of (forward, left, rotate). For example:
   * Go forward 50cm: change = (5,0,0), go through = (1,0)(2,0)(3,0)(4,0)(5,0)图片包含 形状

     描述已自动生成
   * Go backward 20cm: change = (-2, 0, 0), go through = (-1,0)(-2,0)
   * Back turn left: change = (-3, 2, -1), go through = (-1,0)(-2,0)(-2,1)(-3,1)(-3,2) 蓝色的门

     中度可信度描述已自动生成
7. We build a graph based on the information above:
   * Each node in the graph represents a state of the robot (totally 1600 nodes)
   * Each edge in the graph represents one operation.
     1. If a robot with state S0 can change to another state S1 with one operation, then there is an edge connecting the corresponding nodes in the graph.
     2. The cost of edge = the distance of operation in cm + the sum of the penalty of the cells it goes through + the operation penalty
        + The operation penalty of turning is about 40
        + The operation penalty of straight movement is between 0~2. The shorter the movement, the larger the penalty
     3. Totally 10 different operations are used, so there are roughly 16000 edges.
8. Given the position of obstacles and the facing of image, the observation position is 3~5 cells away from the obstacle. The orientation of robot is opposite to the facing of image.
9. For each observation position, find out the corresponding node in the generated graph, perform a Dijkstra with that node as the destination. Then we can find out the shortest path and the corresponding path cost from all the cells in the map to each observation points with the output of Dijkstra, which imply that we know the minimal distance between each pair of observation points, and the distance from the starting position to each observation point.
10. If the distance of an observation point from the starting position is greater than 10^6, that means the shortest path going through a cell that is very close to an obstacle, then the corresponding image will be given up and removed from the list of images to be visited.
11. Given the list of images to be visited, to find out the optimal visiting order, we use brute force search. For all the possible visiting orders, we calculate the total cost. Given the result of step 9, calculating the cost of a certain order of length N has time complexity O(N). So the total time complexity is O(N \* N!). Since N≤8, the cost is acceptable.
12. Given the result from step 9 and 10, we can generate the full travelling path.

Error Correction

1. In the algorithm, it assumes that the offset of turning is a multiple of 10cm, but it is not reality. For example, the offset of back left turn is forward=-28.4cm, left=14.6cm. To reduce the error, we add straight movement before and after turning, so that it is roughly a multiple of 10cm. For back left turn, we add a 5cm backward movement after turning, so the offset is (-28.4cm, 19.6cm). A combination of these robot instructions is deemed as one operation in the algorithm.
2. Since our preferred observation position is about 50cm from the obstacle, the camera may capture multiple images at the same time, but the algorithm can only handle one image at a time. To find out the desired image:
   1. The algorithm knows all the instructions executed, the precise measured offset of each operation, the starting position, and the position of image, so it can estimate the position of image relative to the robot
   2. CV can measure the position of image relative to the robot if an image is detected.
   3. Algorithm will tell CV the expected position of desired image, then the CV will return only one image that is closest to the expected position if multiple images are detected.
3. We use CV to do further calibration in case of the actual offsets of operations are different from what we have measured. CV can return the position of image relative to the robot if an image is detected. Then, every time after the robot recognize a image, since the position of image is also known, we can update the position of robot, and then round the position of robot to the closest cell in the 20\*20 grid. Because when we perform the Dijkstra, we use fixed destination and variable sources, even if the state of robot changes after the calibration, we can still generate the shortest path quickly with the cached Dijkstra results.