Bharath Mahendra Raje Urs

USC ID – 5399443653

CS561 – Prof Laurent Itti

Assignment 1: Search and Path Planning

Part A: Search for optimal landing site.

The aim of this part was to find an optimal landing spot for the rover on the planet, which is defined as, *"the optimal landing spot is a location which has the shortest equal operational distance from all target locations on the surface of the planet."*

Algorithm:

Data Structure used: Min Heap.

Initially insert all the target nodes into the queue and start exploring all the target nodes until a particular node is explored the number of targets times, each from different source target. i.e If there are 10 targets, we explore the entire graph, until a particular node has been explored 10 times (from different targets).

```
// Initialize the queue with n TARGET nodes.
for each node in TARGET
  INSERT_QUEUE(QUEUE, node)
end for

// Continue till the queue is empty or GOAL_TEST is successful
while QUEUE is not empty
  minNode = removeMinimumNode(QUEUE)
  if GOAL_TEST( ) is true,
        break
  else
        explore(minNode)
  end if
end while
```

removeMinimumNode removes and returns the minimum node from the queue, which is then explored/subjected to Goal Test.

```
// Explore node.
for all the neighbors of node
```

if slope of neighbor falls below the threshold, then

// add neighbor to QUEUE, with the value set to the

//distance covered so far.

INSERT_QUEUE(QUEUE, neighbor)

end if

end for

// GOAL_TEST

If a node is reached and explored from ALL the TARGET nodes as their source, then

// Stop exploring, we found the optimal position.

return TRUE.

end if

Here, for Part A, we first start exploring all the TARGET nodes by adding all the neighboring passable cells into the queue as long as the slope lies within the threshold.
All the neighboring passable nodes are added to the queue by setting their value to the distance covered till that node. Now the nodes are arranged in such a way that we always get a node with minimum distance covered and explore it. This is implemented by using min heap data structure.
So, whenever a node needs to be explored, we delete the node from front of the queue.
Once, we get a node to be explored, we consider all its 8 passable neighbors whether they can be added to queue in order to be explored based on the slope calculation.
The node keeps track of information from which TARGET node it has been explored.
Once a node we explore satisfies the condition that it has been explored starting from all the TARGET nodes, we stop exploring the nodes and this node acts as the optimal landing position. Since, we cannot encounter any better node which is yet to be explored in the queue from certain TARGETS, whose distance will be least from all the TARGET nodes. i.e if there is a node later in the queue which is yet to be explored, then its distance from at least one of the targets will be more than distance from the node we already chose.

Optimality:

To prove: The first node which is explored from all the TARGET nodes is THE candidate for the optimal landing site.
According to the algorithm, the nodes are sorted based on the distance covered from the TARGET node to a particular node. (i.e node with least cost will be explored first).

Let node N be the first node to be reached/explored from all the TARGET sites.

Let us assume that there is optimal landing position node (P) which comes later than node (N) in the queue.

Now, from the algorithm we know that a node will be explored only when the distance from its TARGET is less when compared to all other nodes in the queue. So, if there is an optimal position node (P) in the queue which is later than node (N), then it takes longer path from at least one of the TARGET site to P. Thus, if there is an optimal landing position node, it cannot be explored later than N and HAS to be explored before N.

Thus, the landing site P cannot be optimal.
Hence, the first node that is explored from all the TARGET locations will be the correct candidate to be landing site. (Would have minimum distance from all the TARGET sites.)

Part B: Search for Optimal Path.

In this part, we need to find safest paths from optimal landing site to all the target sites. The maximum slope along that path defines safety of the path.
Thus in this part we need to give maximum weightage to the slope compared to the distance. We always chose a path which give less slopes.

We need to come up with the cost function and the heuristic function.

Heuristic function:

Here, I have considered the straight line distance from a node to the target as the heuristic, reason being that, there cannot be any distance from the node to target which takes less distance than the straight line distance. Thus we are not over estimating the heuristic function.

If a node is at (x1, y1) and target is at (x2, y2), then the heuristic function is calculated as the straight line distance from (x1, y1) to (x2, y2)

$$\text{heuristic } h = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

Cost function:

Here we need to give more weightage to slope, and given that we use min heap data structure, we need to make sure that the nodes added to the queue will have their values sorted in ascending order (i.e first element being the least).

So, we need to design a cost function such that when the nodes added into the queue, the nodes with less slope will be added towards beginning of the queue and nodes with relatively larger slopes towards the end.

Consider the scenario,

1. Slope: 0.3, distance covered: 10.
2. Slope: 0.6, distance covered: 5.

In this case, we need to choose case 1, as the slope is less than that of in case 2 even though the distance covered in case 1 is more.

Thus a simple addition of slope and distance would not work for cost function. Since a simple add would result in, *case 1:* 10.3, *case 2:* 5.6. Thus, the node in case 2 would be explored 1st which is not what we want.

We need to make sure that adding the distance covered would affect very less in the final cost.

Num of Rows (ROWS) x Num of Columns(COLUMNS) would give the maximum distance that the map can have. Thus, we multiply (ROWS x COLUMNS x 10) with the slope and then add the distance covered. Here we multiply 10 as well because it is given that the elevation changes in terms of 0.1

So, the cost function is (ROWS * COLUMNS * 10 * slope) + distance covered.

The value obtained from this equation will always sort the nodes giving more preference to slope and if in case of ties, distance covered would be taken into consideration.

If we consider the same scenario as above and assume ROWS = 20 and COLUMNS = 20.

Case 1:

Cost = (20 x 20 x 10 x 0.3) + 10 = 1210.

Case 2:

Cost = (20 x 20 x 10 x 0.6) + 5 = 2405.

Now when we add these two nodes with their cost into the queue, the node in case 1 would be explored first.

Thus choosing this cost function will give us the direction to find the “safest” path.