


Pathfinder solver using MicroRato

RMI - Robótica Móvel Inteligente

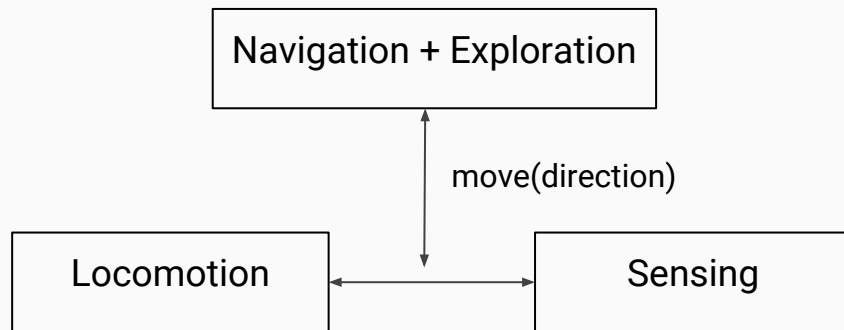
Assignment 3

Pedro Santos, 76533
Ruben Kins, 92078

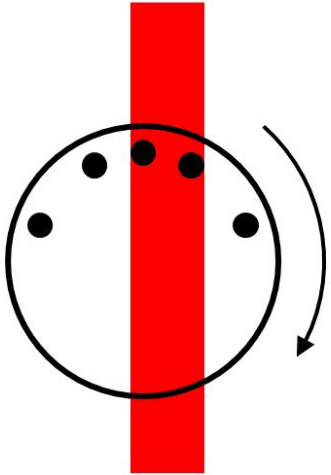


Subtasks

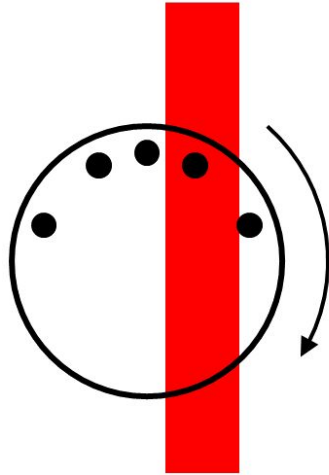
1. Locomotion
2. Sensing and Localization
3. Map building
4. Exploration
5. Pathfinding



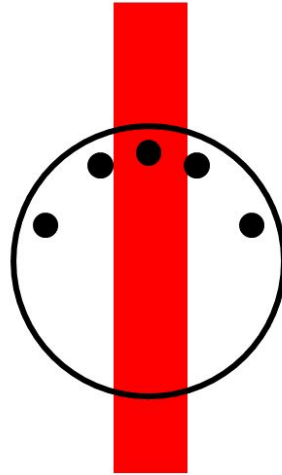
Locomotion



a) Slow turn (here: right)



b) Fast turn (here: right)



c) Straight ahead

Localization and Sensing

- Odometry becomes unreliable after a time -> only as local reference
- Discrete state: x, y coordinates, direction
- Direction in two domains: Relative (Front, back...), Absolute (North, East...)
- Stabilizing the sensor output for more reliable results
 - State-switch and buffer
 - Example: State 1, Buffer [1, 0, 0, 0] -> state 1; Next iteration Buffer [0, 0, 0, 0] -> State 0
 - Not used for locomotion due to delay

Localization and Sensing

- Possible node types:
 - Straight segment
 - Dead end
 - Intersection
 - Token (or cheese)
- Decision based on central and outer left/right sensor readings
- Algorithm next page

Localization and Sensing

```
1.  if (distance traveled > cell distance):
2.      if(frontSensor ==1):
3.          paths = relative2absolute(1001)      //straight section
4.      else:
5.          paths = relative2absolute(1000)      //dead end
6.          node = nextNode(direction)
7.  else if (leftSensor || rightSensor):
8.      while(not centered): center()
9.      if(leftSensor && !rightSensor):
10.         paths = relative2absolute(leftSensor<<3 | frontSensor)      // left turn with maybe front path
11.      if(leftSensor && rightSensor):
12.         if(distance travelled while measuring black > line thickness):
13.             paths = relative2absolute(10010)      //token found
14.         else: paths = relative2absolute(0101 | frontSensor)      //T-intersection or cross-intersection
```

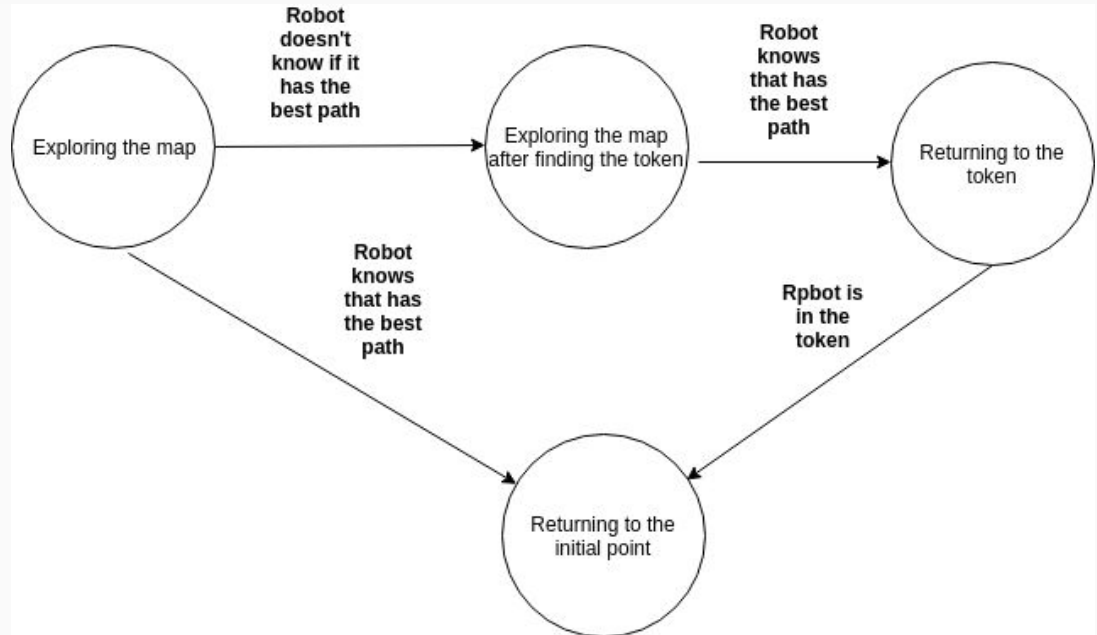
Deliberative Part

Challenge

- Explore a maze and find a token
- After finding the token the robot has to return to the initial position using the fastest path available.

States

- Exploring the map
- Exploring the map after finding the token
- Returning to the token
- Returning to the initial point



Map

- Bidimensional array containing elements of the type 'Node'.
- The initial size is 37 in each dimension because the map has at maximum 19 nodes
- The members of the structure 'Node':
 - `Int coor_x`
 - `Int coor_y`
 - `int paths[4]`
 - `int gCost;`
 - `Int hCost;`
 - `bool visited;`
 - `struct Node *parent;`

Map Building

- Function to initialize the map
- Function to update the paths and the member 'visited' of the current node is also set to 'true'
- Function to update the size of the map
- Function to map the wanted world node coordinates to the corresponding map array index.

State - Exploring Map

- UpdateMap()
- UpdateUnknownNodesHistory()
- Check if it found the token
 - Check if best path is available
- getDirectionToExploreMap()

getDirectionToExploreMap()

- Verifies if it is to follow the A*
- The robot will check if it has a neighbour that has only one path, but it never went there.
- if there isn't neighbours with this properties will go to the unknown neighbour.
- If the current node doesn't have neighbours that never went there with only one path or unknown neighbours it will be made A* to the nearest node with unknown neighbours. The nearest node with unknown neighbour is the last element of the array `updateUnknownNodesHistory`

State- Exploring after finding the token

- UpdateMap()
- Verifies if the last state was ExploringMap/Verifies if it is not following the A Star
 - Checks if the best path is available
- getDirectionToFindBestPath()

getDirectionToFindBestPath()

- verify if it is performing the A*
- verify if it is in a sub state
 - This sub state consists in calculating the node with unknown neighbours with lower fCost or if there is several minimums the node with a lower hCost of the closed set of the A* between the initial node and the target node.
 - At same time it is verified if the current node continues in the closed set and if it has unknown neighbours.
 - it will change the sub state and it will perform the A* to the node with unknown neighbours with lower costs.
- the robot will check for the unknown neighbour of the current node and will explore that node and the state it is changed again to go to analyse the closed set.

State - Returning to the token

- The agent will go to the token using the path from the A* algorithm
- When the robot it is in the node with the token will change the state to "Returning to the initial point".

State - Returning to the initial point

This is the last state. The robot it will go from the token to the initial position using the best path possible calculated by A*.

Results

- The project was an overall success.
- The robot can independently explore and navigate the maze.
- The additional challenge in this project was the fact that a real and not a simulated robot was used.
- Implementing the A* search algorithm proved to be much more challenging using C.

END