Lab ISS | the project cautiousExplorer

Requirements

Design and build a software system that allow the robot described in *VirtualRobot2021.html* to exibit the following behaviour:

- the robot lives in a closed environment, delimited by walls that includes one or more devices (e.g. sonar) able to detect its presence;
- the robot has a **den** for refuge, located near a wall;
- the robot works as an *explorer of the environment*. Starting from its **den**, the robot moves (either randomly or - preferably - in a more organized way) with the aim to find the fixed obstacles around the **den**. The presence of mobile obstacles is (at the moment) excluded;
- since the robot is '*cautious*', it returns immediately to the **den** as soon as it finds an obstacle. Optionally, it should also return to the **den** when a sonar detects its presence;
- the robot should remember the position of the obstacles found, by creating a sort of 'mental map' of the environment.

Delivery

The customer requires to receive the completion of the analysis (of the requirments and of the problem) by **Friday 12 March**. Hopefully, he/she expects to receive also (in the same document) some detail about the project.
The name of the file (in pdf) should be:

```
cognome_nome_ce.pdf
```

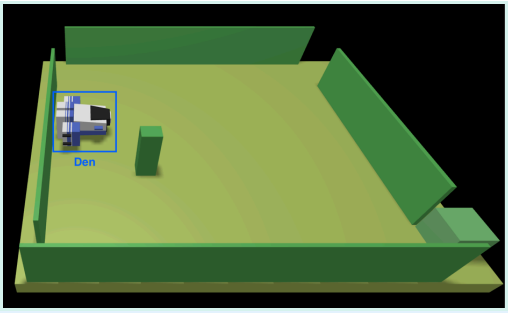
Requirement analysis

- Robot: the subject that will explore the room. Known robot's characteristics: *VirtualRobot2021.html*;
- Closed environment: the robot will move into a closed room. So its run will always be limited, due an obstacle or a wall;
- Devices able to detect its presence: a device able to notify to the robot caller if it has detect the presence of the robot;
- Den: a position near a wall where the robot starts its exploring and returns if it detects an obstacle;

- To refuge: the action done by the robot when it founds an obstacle. The customer said that, for this case, wall are similar to obstacles, so also when the robot founds a wall it has to come back to the den. This action terminates when the robot returns to its den;
- Explorer: the robot's goal is to explore the room to know the position of the obstacles;
- Fixed obstacles: obstacles that have a static position into the room;
- Mobile obstacles: obstacles that have a dynamic position into the room. Actually there aren't mobile obstacles;
- Cautious: the robot will return to its den immediately when it will find an obstacle (and optionally when a sonar detects its presence);
- To find an obstacle: the robot finds an obstacle when it has a collision with the obstacle;
- Optionally when a sonar detects its presence: the customer said that at the moment this features isn't important. It could be useful in the future, but this isn't sure;
- Should remember: the robot has to store a sort of mental map that says where are the obstacles found;
- A sort of mental map: the robot must be able to remember where are the obstacles found. There isn't a specific form for the map and the costumer specified that once stopped and restarted the robot exploration, its map restart from empty map.

User story

At the beginning, I put the robot in a position near the wall (called **den**). Then I start the system and the robot will start to explore the room independently. When it will find one obstacle (or a wall), it will return automatically to its den and restart its exploration following another path. The path choosed won't be completely random. Once started, the robot continues its exploration until I stop it. Once stopped, I can ask to the robot to give me the map representing where the obstacles found are located. Once restarted, the old map will be override with an empty map and the exploration will restart. If I don't stop the exploration, the robot will stop when it will know a complete map of the room (complete exploration).



Informal Test Plan

- Once found an obstacle, the robot must return to its den. The first half and the second half of the path must last the same time;
- Once found an obstacle, the robot must return to its den. The first half and the second half of the path must be exactly opposite;
- Changing the position of the den, the number of the obstacles found in a complete exploration must not change;
- Changing the position of the den, the position of the obstacles found in a complete exploration must not change (using only static obstacles).

Problem analysis

- The system isn't affected by the structure of the robot;
- The system can communicate with the robot using two ways:
  - HTTP POST on port 8090
  - websocket on port 8091
- There is a conceptual abstraction gap because there are two possible ways of communication but both requires a request-response schema. The nature of the problem suggests to prefer an asynchronous communication to retrieve in a better way what happens to the robot, so in this case is preferable the websocket mode.
- The problem suggests a logical architecture composed by:
  - a main service used to start the robot exploration, eventually to stop it and to retrieve the mental map representing the result of the exploration;
  - a service used by the robot to receive commands from the previous service and to send back feedbacks to it.

- The mental map must be a way to store and "show" what the robot has discovered during its exploration. In some way it has to indicate where are the fixed obstacles found and it can represent the complete room or not. This mental map will be override when the robot restarts its exploration. It's not really important how the map is build,

but it's important the meaning of the informations. They can't be ambiguous;

- The room's dimensions aren't known;
- The number of the obstacles isn't known;
- The positions of the obstacles aren't known;
- Due to the three previous points, choose an intelligent way to explore the room isn't banal.

Test plans

1. ```
   @Test
   void testFirstHalfSecondHalfSameTime() {
      CautionsExplorer appl = new CautionsExplorer();
      appl.explore();
      String feedbacks = appl.getTrip(obstacleNumber);
      String firstHalf[] = separateFeedbacks(feedbacks, "first");
      String secondHalf[] = separateFeedbacks(feedbacks, "second");
      assertTrue(firstHalf.length == secondHalf.length);
   }
   ```
2. ```
   @Test
   void testFirstHalfSecondHalfEqualsAndOpposite() {
      CautionsExplorer appl = new CautionsExplorer();
      appl.explore();
      String feedbacks = appl.getTrip(obstacleNumber);
      String firstHalf[] = separateFeedbacks(feedbacks, "first");
      String secondHalf[] = separateFeedbacks(feedbacks, "second");
      for (int i=0; i<firstHalf.length; i++)
         assertTrue(firstHalf[i].equals(secondHalf[i]));
   }
   ```
3. ```
   @Test
   // To use only after complete explorations
   void testDifferentDenSameObstaclesNumber() {
      CautionsExplorer appl = new CautionsExplorer();
      appl.explore();
      String map1 = appl.getMap();
      int obstaclesNumber1 = readObstaclesNumber(map1);
      // change robot's den
      appl.explore();
      String map2 = appl.getMap();
      int obstaclesNumber2 = readObstaclesNumber(map2);
      assertTrue(obstaclesNumber1 == obstaclesNumber2);
   }
   ```
4. ```
   @Test
   // To use only after complete explorations
   void testDifferentDenSameObstaclesPostion() {
      CautionsExplorer appl = new CautionsExplorer();
      appl.explore();
      String map1 = appl.getMap();
      Position obstaclesPosition1[] = readObstaclesPosition(map1);
      // change robot's den
   ```

```
        appl.explore();
        String map2 = appl.getMap();
        Position obstaclesPosition2[] = readObstaclesPosition(map2);
        for (int i=0; i<obstaclesPosition1.length; i++)
            assertTrue(obstaclesPosition1[i].equals(obstaclesPosition2[i]));
    }
```

- explore() is the function to say to the robot to start the exploration;
- getTrip(obstacleNumber) is a function to read the trip done from den to an obstacle and from the obstacle to the den;
- separateFeedbacks(feedbacks, half) is a function to take the first half or the second half of a trip;
- getMap() is the function to retrieve the stored map;
- readObstaclesNumber(map) is a function that is able to read the number of the obstacles found starting from the exploration map;
- readObstaclesPosition(map) is a function that is able to read the the position of the obstacles found starting from the exploration map;
- Position is a class useful to represent where an obstacle found is located

Project

A first project abstract

The CautiousExplorer could be develop using Node.js because it is very useful for asynchronous programming. It also permit an easy websockets use.

CautiousExplorer contains the logic to explore the room, a possible idea is the following:

- Assumed the den is along the left "wall", the robot is looking "north" and one move forward consists of a walk of the same distance as the robot dimension;

1. It will start moving forward;
2. When it hit an obstacle or a wall it will return to the den;
3. It will turn right, move forward once and turn left;
4. Repeat from point 1.

- When it arrives on the right side, it will follow a similar strategy, moving to left instead of to right, but first it will walk once to "north";
- When it arrives on the left side, it will follow always this strategy but first it will walk once to "north";
- When it arrives on the top side, it will follow a similar strategy, moving to "south" instead of to "north", but first it will walk to the initial den "height" plus one walk to "south";
- So it will follow this logic until it reaches the bottom side.

At every step the CautiousExplorer must store the move and its result, as feedbacks.

Probably the map will be a matrix. Its dimensions will be the height and the width of the room in "robot dimension", this according to the fact that one walk will be equal to the

same distance as the robot dimension. When the robot will meet an obstacle, the CautiousExplorer will store a value (true? 1? obstacle?) into the cell that represents the position of the obstacle. In this way, it will be easy to read the number of obstacles and their positions.

By Daniele Colautti
email: daniele.colautti6@studio.unibo.it