Home

**Final task 2016**

The context

A differential drive robot (called from now on **robot**) must reach an area (**B**) starting from a given point **A**. To reach the area **B**, the robot must cross an area equipped with **N (N>=1)** distance sensors (sonars). The signal emitted by each sonar is reflected by a wall put in front of it at a distance of approximately **90 cm**.



Moreover:

* The section of the wall in front of each sonar is painted with a different illustration.
* The robot is equipped with a distance sensor (sonar) and (optionally) with a Web Cam both positioned in its front. It owns also a Led
* The robot should move from **A** to **B** by travelling along a straight line, at a distance of approximately **40-50 cm** from the base-line of the sonars.

The sonar subsystem

Each sonar is handled by a RaspberryPi. Our factory already provides (project [it.unibo.tfce16.sonars](https://137.204.107.21/syskb/it.unibo.tfce16.sonars/)) a software layer that 'transforms' each sonar device into a source of events, as described by the following model expressed in the QActor language:

System sonarSensorEmitter

Event sonar : p( Distance, Angle )

Context ctxRadarTfCe16 ip [ host="192.168.251.1" port=8033 ] -standalone

Context ctxSensorEmitter ip [ host="192.168.251.118" port=8133 ]

QActor sensorsonar context ctxSensorEmitter {

Rules{

/\* THESE RULES ARE WRITTEN IN THE WorldTheory of the sensorsonar \*/

/\* CONFIGURATION \*/

onRaspberry.

mysonar( s1, coloryellow, "192.168.251.118" ).

//mysonar( s2, colorred, "192.168.251.121" ).

//mysonar( s3, colorgreen, "192.168.251.120" ).

/\* SIMULATION DATA \*/

p(80,0). p(85,20). p(90,40). p(85,60). p(85,80). p(80,100). p(75,120). p(70,140).p(65,160). p(70,180).

/\* RULES \*/

sonar(s1,30).

sonar(s2,90).

sonar(s3,150).

setmyposition :-

mysonar( SONAR, \_, \_ ), sonar(SONAR,POS), assert( position( POS ) ).

obstacledata( p(D,POS) ) :-

actorOpDone( \_,d(D) ), //set by actorOp getDistanceFromSonar (1)

position(POS).

}

Plan init normal

println("sensorsonar STARTS") ;

[ !? onRaspberry ] actorOp startSonarC ;

demo setmyposition ; //depends on mysonar

[ !? position(POS) ] println( position(POS) );

switchToPlan workSimulate ;

println("sensorsonar workReal") ;

[ !? onRaspberry ] switchToPlan workReal

Plan workSimulate resumeLastPlan

[ !? p(D, A) ] println(p(D, A)) else endPlan "bye";

[ ?? p(D, A) ] emit sonar : p(D,A) ;

delay time(500) ;

repeatPlan 9

Plan workReal

actorOp getDistanceFromSonar ; //(1)

[ !? obstacledata( VAL )] println( VAL ) ;

[ !? obstacledata( VAL )] emit sonar : VAL ;

[ ?? actorOpDone(OP,R)] println( R ) ;

repeatPlan

}

The usage of this software layer should be evaluated during the problem analysis phase. In particular, it should be clearly presented any movitation that leads to adopt it or to reject it in favor of another layer, that the analyst believes more appropriate to the current problem.

The work to do

Design and build a (prototype of a) software system such that:

* shows the sonar data on the GUI associated to a **console** running on a conventional PC. For example (see the project [it.unibo.qactor.radar](https://137.204.107.21/syskb/it.unibo.qactor.radar/" \t "qa)):



.

* evaluates the expression:

(sk + sk+1 + ... sN) / (N-k+1)

where k is the number of the first sensor not reached by the robot and sk is the value of the distance currently measured by that sensor. If the value of the expression is less than a prefixed value **DMIN**( e.g. **DMIN=70**), play an **alarm sound**.

* when the robot reaches the area in front of a sonar, it
  1. first stops
  2. then rotates to its left of approximately **90 degrees**
  3. starts *blinking a led* put on the robot
  4. takes a photo of the wall (in a simulated way only, if no WebCam is available) and sends the photo to console by using the **MQTT** protocol
  5. rotates to its right of approximately **90 degrees** to compensate the previous rotation
  6. stops the blinking of the led and continues its movement towards the area **B**
* when the robot leaves the area in front to the **last sonar**, it continues until it arrives at the area **B**
* stops the robot movement as soon as possible:
  1. when an **obstacle** is detected by the sonar in front of the robot
  2. when an alarm sound is played
  3. the user sends to the robot a proper command (e.g. **STOP**)
* makes it possible to restart the system (by manually repositioning the robot at point **A**) without restarting the software

The goal

The goal is to build a software system able to evolve from an initial proptotype (defined as the result of a problem analysis phase) to a final, testable product, by working in a team and by 'mixing' in a proper (pragmatically useful) way agile (SCRUM) software development with modelling.

Further requirement

The model of the system defined after the problem analysis phase should be sent to the product owner before starting the implementation phase. Any change introduced in the analysis model during project or implementation should be also communicated, together with proper motivations.

By AN Unibo-DISI