A Lego robot for experimental benchmarking of robust exploration algorithms

Antonino Serri, serri@diee.unica.it

DIEE University of Cagliari -Piazza d'Armi, Cagliari, Italy 09123

Abstract— **A very simple autonomous robot based on the Lego programmable brick (RCX) is proposed to test the performances of robust exploration algorithms. The robot must explore an unknown area with arbitrary boundary detected by a unique light-sensor and has to find a little target by travelling within the search area. The control algorithm must keep the robot within the area and adjust the trajectory after each collision to the boundary. The mechatronic design may be optimized but attention is focused on the robusteness of the control strategy. Experimental benchmarking is presented for fixed and random turning. Programming details are discussed to further test other different algorithms.**

Keywords: mechatronic, robust control, autonomous robot

I. Introduction

REALIZATION of autonomous mobile robot represents a hard challenge involving both mechanical and control issues, a common mechatronic problem. There are many applications for this systems and technical literature reports a huge amount of projects referring to very different mechanical structures or simulating idealised robot behaviours. Moreover, the exploration of unknown environments represents a fundamental task for mobile robots and can be addressed as the basic control issue [1]. Scientific advance of this topic should benefit from the definition of some reference robot or referring to a benchmark problem.

This paper describes a simple autonomous robot suitable for testing real-time control algorithms. Aiming to benchmarking of the control algorithms, the robot has to be widely accessible and any laboratory prototype is unfeasible, while the recent Lego robotic components [2] may be useful to realise a mobile robot that can be easily built by any interested researcher. Details on the Lego robot will be reported in the following to highlight its simplicity and robustness.

The control algorithm to explore an unknown environment may lead to an excessive complexity requiring continuous self-localisation and construction of maps (SLAM-robot) that require the unreal hypothesis to avoid wheel slipping. A more general Artificial Intelligence perspective suggests to control the robot by defining an active learning, but the collection of training data may likely result in very inefficient operation.

More recently, random techniques, e.g. randomized motion planning (RMP), are increasing the researchers interest [3]. Many random techniques may be very efficient and benefit from recent probabilistic optimization techniques such as genetic algorithms [4]. Due to the lack of deterministic proof of completeness for this algorithms, it is very important to test their performances by using a physical robot. As a consequence of this approach, a benchmarking procedure is outlined and applied for a small set of random exploration algorithms.

To give an application perspective to this work, one can refers to the problem of locating a mine within a restricted area. Simplifying the mechanical model of this problem, the autonomous mobile robot has to be able to move, detect the area boundary and detect the mine. Robot localisation and map construction have poor reliability in this application where no landmarks are available and odometering in outdoor condition is highly imprecise due to wheel slipping. This frame, with trivial simplifications, can be directly translated in the proposed test bed using the Lego robot. The main issue is to perform very fast exploration without map construction. The resulting robot can be suitable for an extremely low-cost mine sweeper production.

From the control point of view, the scope of this preliminary paper is to deal with robust algorithms that avoid odometry techniques while chaotic behaviour is considered useful.

II. The Lego Robot

The proposed Lego robot represents the simplest realisation to meet the application requirements described in the introduction. Moreover, there is a budget issue to use Lego robots in scientific research: it may already be used for teaching purposes [4].

The required Lego parts are: the programmable brick RCX (robot command experiment), two motors directly

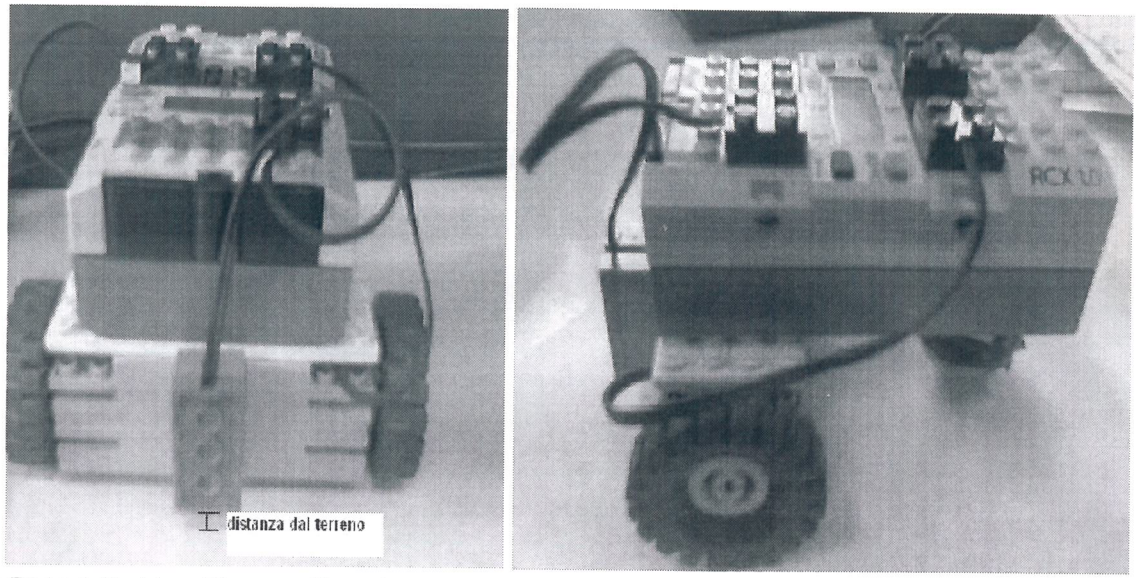

Fig. 1. Front and side picture of the proposed Lego robot.

connected to the traction wheels, one light sensors, one pivoting wheel plus cables and few spare parts to obtain mechanical connection and robustness.

Fig. 1 shows the assembled robot used for experiments. By so far, only few pieces of the commercial set of Lego (RIS: Robotics Inventory Set) are required but the cost to purchase these single parts is almost the same of the complete set (about 200US$). This robot has a little weight: 0.3Kg (mainly due to the six AA size batteries) and can travel at adjustable speed up to 0.2m/s. It is possible to extremely simplify this robot by replacing the pivoting wheel with a sliding support, but the resulting friction and losses lowers the maximum speed and reduce the battery life.

III. Elementary Random Exploration Algorithms

At first, a very simple exploration of an unknown environment may be carried on by straight moving within the area and simply reacting to its boundary while seeking for the target. It is widely accepted that a random search will be more efficient in finding the target than an exhaustive search. On the contrary, random search may fail the "completeness" and strange cycling behaviours may appears. Theoretical discussion of probabilistic properties is beyond the scope of this preliminary work and very simple, referred as "elementary", random exploration algorithms are outlined here for subsequent implementation. The control algorithm, while seeking for the target, changes the robot direction after each collision to the boundary following the control rule. Keeping in mind the robustness of the control and the very low hardware resources, only these simple rules are investigated in the present paper:

a) turn 90° clockwise

b) turn 90° counterclockwise or clockwise at random

c) turn θ° clockwise

The rotation θ can be defined dinamically and resorting to some heuristic rule, for example computing (1) in degrees using some kind of memory.

$$\theta = 90 + (\text{NormTime})*75 \qquad (1)$$

where NormTime is a positive variable less than one piecewise proportional to the time elapsed between the two last boundary collision. The (c) rule represents a heuristic to bias the trajectory in the direction of the larger free space while avoiding entrapment or cycling.

The complexity of the control algorithm can further grow within the memory and real-time capabilities of the robot microcontroller. It is worth noting that in the present work the attention is focused on the control robustness and on the implementation issues. If highly complex control algorithms are considered, it is much more appropriate to use a more powerful hardware with GPS connection, thus resulting in a completely different application.

IV. Numerical Simulation Of Robot Navigation

Each control law has to be discussed in order to evaluate the expected performances. Due to the non linear nature of the system, the analytical approach may likely fails and a numerical simulation must be computed introducing the appropriate modeling of the robot and of the environment.

During this work the Matlab environment has been used

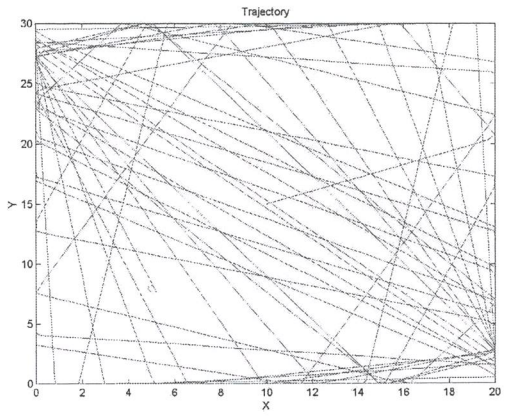

Fig. 2. Matlab simulation of the trajectory neglecting robot size

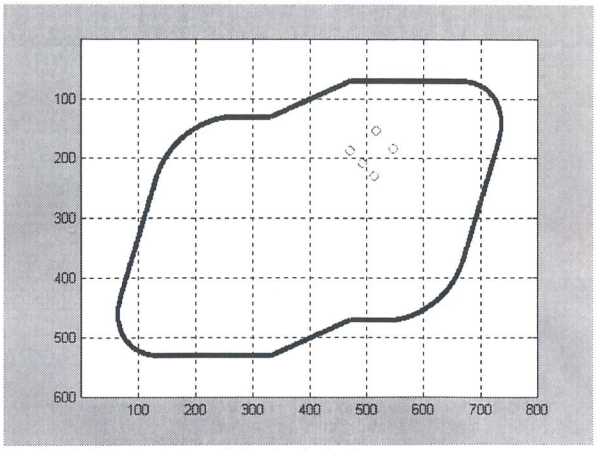

Fig. 3. Matlab simulation of the physical robot.

to obtain preliminary results useful to refine the control law. Two numerical simulations resulting from different robot modeling are presented.

First, the robot dimensions are neglected and a simple point has been considered moving within the boundary and its trajectory is changed at each collision according to the control law. This simplified model allows also for an analytical description, but this is out of the scope of the present work. For example, the robot trajectory for a rectangular area under (b) control is shown in Fig.2. The target has been found after 83 boundary collisions.

Is is worth noting that some interesting comments related to non linear dynamics and deterministic chaotic behavior may be developed. In this simplified numerical simulation, one has to choose the initial direction of the robot; this choice may lead to cycling trajectory, thus failing in the complete exploration of the area.

Another more detailed simulation has been developed to consider the robot geometry in order to verify its turning capabilities and sensor position. This simulation allows for refining the mechatronic design. Fig. 3 shows the map considering a robot with two motors and two light sensors. Numerical simulations confirmed that only one light sensor is really needed to keep the robot within the boundary. An additional light sensor can be useful to both increase the robot scope and to better detect the area boundary.

Both these numerical simulations are required to start the mechatronic design. After the definition of the physical size of the robot and its control law, the designer has to choose the most appropriate hardware and software platforms.

The main result of these simulation is the definition of the high-level description of the control algorithm to be implemented on the robot processor. Table I lists the pseudocode expressed as natural language for the subsequent code translation depending on the choosen hardware (mainly maximum RAM availability) and software (firmware and application programs) platforms.

V. Programming Environments Suitable For Real-Time Operation

8-bit/16-bit microcontrollers are suitable to build the robot and the Lego Mindstorms series are very attractive because a lot of free programming environments for the H8/3790 are available. The graphical programming of Robolab (based on the LabVIEW interface) is very effective [5], the virtual instrument for the elementary exploration (c) is reported in Fig. 3. In addition, a Java based language has been used to compare the different implementation of the algorithm from the memory usage point of view [6]. Table II lists the LeJOS source code for the same control algorithm.

VI. Benchmarking

The robot exploration can be evaluated considering a set of different areas and repeating the target search within them for a statistically representative number of times. Until now, a simple rectangular area (A4 sized) and an "U-shaped" area (obtained with three A4 sheets) are considered. The results of this procedure are still in progress and will be reported during the conference and submitted for publication after the inclusion of further control laws and comparing them with at least with a self localization algorithm.

TABLE I
Pseudocode

```
Initialize the robot memory,
activate robot sensor
run the following tasks:
T1: run motors forward until stop criteria is matched (max time)
T2: when boundary is detected:
        stop and backward motors to remain within boundary
        change direction according to control law
        update memory (timers and counters)
T3:when target is detected:
        stop the robot exploration
```

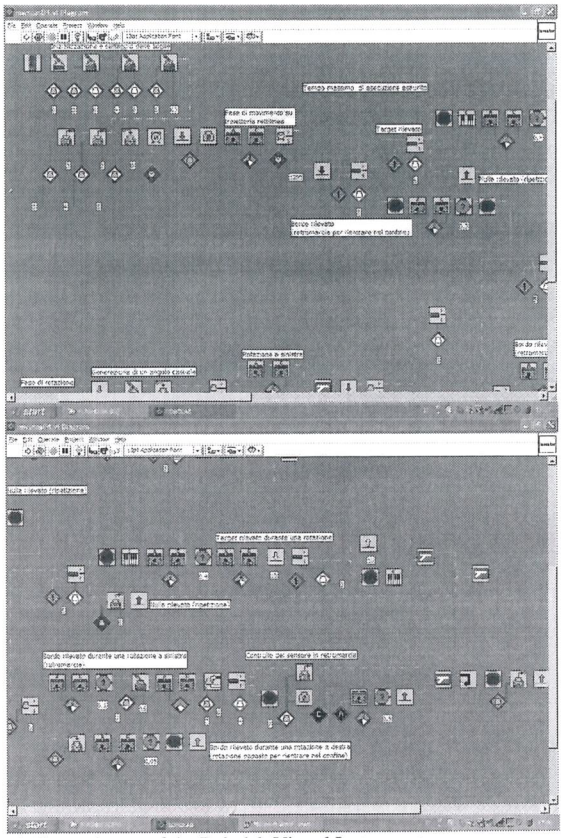

Fig. 4. Screendumps of the Robolab Virtual Instrument

VII. Preliminary Conclusions and Future Work

This work is currently in progress, thus only few preliminary results are commented. First, it is possible to find the target without the map building. Again, heuristic rules to keep some kind of memory of the search may improve the search performances and result in performances similar to less randomized search.

Acknowledgment

Antonino Serri wish thank all the students of its course "Circuiti.e Algoritmi per la Meccatronica" involved in the projects related to some topics outlined in the present work, among all of them: Pierluigi Marini and Simone Congiu (RCX programming) and Omar Porcu (Matlab simulation).

References

[1] B. Yamauchi, "A frontier-based approach for autonomous exploration", IEEE Int. Conf. On Rob. & Aut.1997, pp. 146-151.

[2] A. Serri, "Laboratory Course of Mechatronics Unsing Lego Mindstorms", Int Conf. Mechatronics, Istanbul, Turkey, Oct. 2001.

[3] Oriolo, Vendittelli, Freda, Troso, "The SRT method, randomized strategies for exploration", IEEE Int. Conf. On Rob&Aut., 2004.

[4] A. Serri, "A novel website of mechatronics for remote learning", Int. Journal of Engineering Education, 2003, vol. 19, pp. 420-426.

[5] Robolab tutorial: http://www.ceeo.tufts.edu/graphics/fll/

[6] LeJOS tutorial: http://lejos.sourceforge.net/tutorial/

TABLE II
LeJOS Code

```
/** * robust random exploration of the LEGO robot */
import josx.platform.rcx.*; import josx.util.*; import java.lang.*;
public class tesina_ruotino
  implements SensorConstants {
  private static final boolean CW = true;
  private static final boolean CCW = false;
  private static final int MAX_TIME = 120000;  // tempo max
  private static boolean timeOut;
  private static boolean rotationDone;  private static Timer
myTimer;
  private static int beginTime;  private static int elapsedTime;
  private static int floorValue;  private static int angle;
  public static void main(String[] args)
   throws InterruptedException {
    TimerListener tListener = new TimerListener() {
      public void timedOut() {
        timeOut = true;        }     };
    myTimer = new Timer(2000, tListener); // 2000 di default
    Sensor.S1.setTypeAndMode(
     SensorConstants.SENSOR_TYPE_LIGHT,
     SensorConstants.SENSOR_MODE_PCT);
    Sensor.S1.activate();      Thread.sleep(400); // necessary
    BeginTime = (int)System.currentTimeMillis();
    Sound.twoBeeps();
    floorValue = Sensor.S1.readValue();
    for (;;) {       Thread.sleep(400);
     goForward(7);
     if (!isObject()) {
      elapsedTime = (int)System.currentTimeMillis();
      elapsedTime = elapsedTime - beginTime;
      if (elapsedTime > MAX_TIME) {
       TextLCD.print("tout");
       Sound.buzz();
       Thread.sleep(1000);
       Sound.beep();
       TextLCD.print("   S");
       elapsedTime = (int)(elapsedTime/1000);
       LCD.showNumber(elapsedTime);          break;          }
       for (;;) {
       goBackward(5,200);                          // set up rif. (1)
       angle = newAngle();          Thread.sleep(400);
       rotationDone = rotate(true,CW,angle);
       if (rotationDone) {          break;          }
       else {  if (isObject()) { break; }
        else {          Thread.sleep(400);
         rotate(false,CCW,0);
         angle = newAngle();          Thread.sleep(400);
         rotationDone = rotate(true,CCW,angle);
         if (rotationDone) {          break;          }
         else {
          if (isObject()) {          break;          }
          else {          Thread.sleep(400);
           rotate(false,CW,0);          Thread.sleep(400);
           }          }          }          }
      } // end of while OR for       }
   if (isObject()) {
    Sound.systemSound(false,3);       Thread.sleep(500);
    Motor.A.setPower(4); Motor.C.setPower(4);
    Motor.A.forward(); Motor.C.backward();
    Thread.sleep(340);                          // set up rif. (4)
    Motor.A.stop(); Motor.C.stop();       Thread.sleep(400);
    Motor.A.setPower(7); Motor.C.setPower(7);
    Motor.A.forward(); Motor.C.forward();
    while (!isBorder()) ;
    Motor.A.stop(); Motor.C.stop();
    Sound.beepSequence();
    break;       }      } // end of for
  Button.RUN.waitForPressAndRelease();
  Sensor.S1.passivate();
  System.exit(0);
} // end of main
```