Politecnico di Milano
Computer Engineering

Pac-Bot

Documentation

Coordinator/Tutor:
Prof. Andrea Bonarini

Students:
Mikel Vuka Matr. 759564

ACADEMIC YEAR 2012-2013

Index

1. Introduction

This project's objective is the implementation of a game of the *Highly Interactive, Competitive Robogames* ("HI-CoRG") category. These games feature the use of autonomous robots and contain elements of interaction between the human players and the robot (or robots).

This game, Pac-Bot, is a real-time game based on the classic Pac-Man videogame.

2. Guidelines

First of all, the intention was to build an interactive competitive game of a certain success. A Robogame must fulfill the following characteristics:

- Consist of at least one robot and one human player able to interact between them cooperatively or competitively
- low-cost and high-efficiency use of the involved components
- safe playing
- easy and entertaining
- the robot has to be seen as a rational agent from the user
- the game must be tested on the field
- the game must involve many senses (sight, sound, touch)

The difficulty of the game have to be suitable or adaptable to the user. The rules must be easy to understand and learn, and the actions easy to carry through. Furthermore, in order for the game to be involving, the system must react to the actions in real time.

The robot must fulfill the following characteristics:

- Receive input in a reliable way
- Aspect suitable to the game
- Behaviors that don't appear casual, but rational and thought through
- Real-time functioning

The general purpose of the robogames is to bring gaming towards the "physical", first started from the Nintendo with the Wii, and nowadays embraced by the majority of the videogame firms. Robogames also want to introduce robots as utilities in the everyday use, and diffuse the interest for robotics to a wider public.

3. Game description

The rules and principal elements of the game are discussed in this chapter. Also the requirements of the various components are explained in detail.

3.1 Objective

The objective of the Pac-Bot game is to prevent Pac-Bot from getting to all the Pac-dots in the maze. The phantom robot, controlled by the user, is to make contact with Pac-Bot before he gets all the Pac-dot, in case he isn't in Super Mode (which happens when Pac-Bot gets a special Pac-dot), or get away if Pac-Bot is in Super-Mode.

3.2 Recipients

The game is played with two robots, an autonomous one and a remotely user controlled one. The recipient range is very wide. It is suitable for children and adults, from 13 to 60 years old.

3.3 Basic requirements

Environment

The game is played inside a maze. The maze can be of any dimensions as long as it respects the layout presented here. Any environment that offers that space is adequate. However, a smooth indoors pavement is recommended in order to allow the robots to move easily inside the maze.

Robot

The robot has to feature a camera in order to recognize its opponent, distance sensors in order to move inside the maze and an RFID reader sensor in order to read the Pac-dots when he moves on them. The speed is not very important as long as it makes the game dynamic and fun to play with. The robot needs to feature a wireless connection in order to pass sensor data to the computer and receive actuator commands (in case the processor of the robot is not very powerful, a computer is needed to interpret the sensor data and take decisions).

Therefore a SpyKee robot model was used for this project.

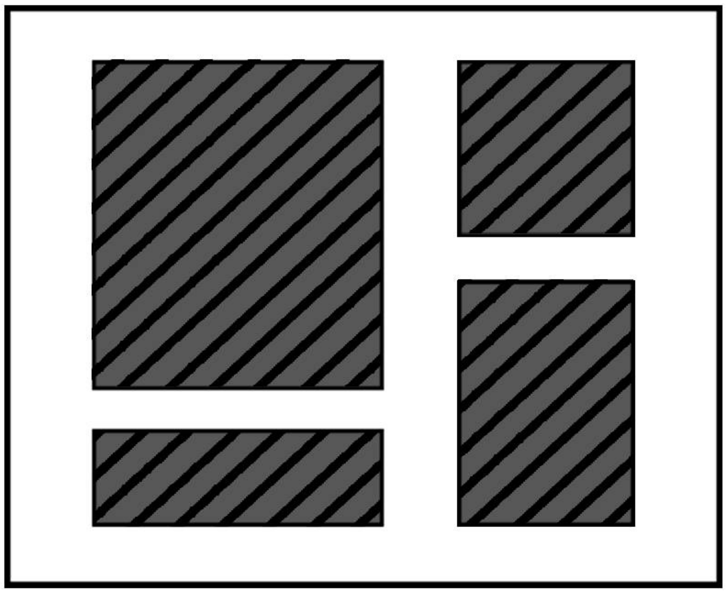
Computer

A computer is used to interpret the sensor data, take decisions. The computer also works as a referee and provides the player with information on the state of the game (for instance the Pac-Dots that Pac-Bot already ate, the Super Mode state of Pac-Bot, the winner of the match).

3.4 Involved objects

Maze

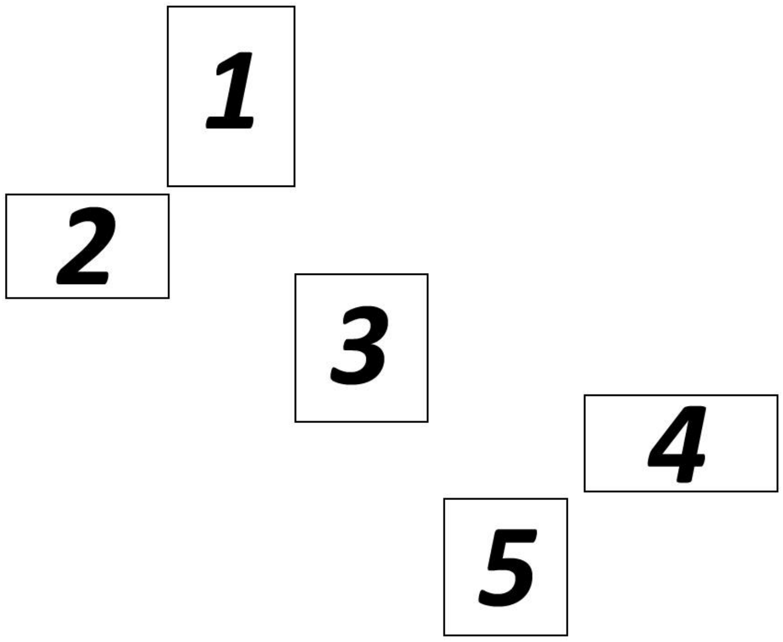
The maze is made of wooden walls. The robot knows the structure of the maze, so in order to guarantee the best performance of the match the structure shown below must be respected.



The walls must not be shorter than 20 cm in order for the robot to get the distance from them.

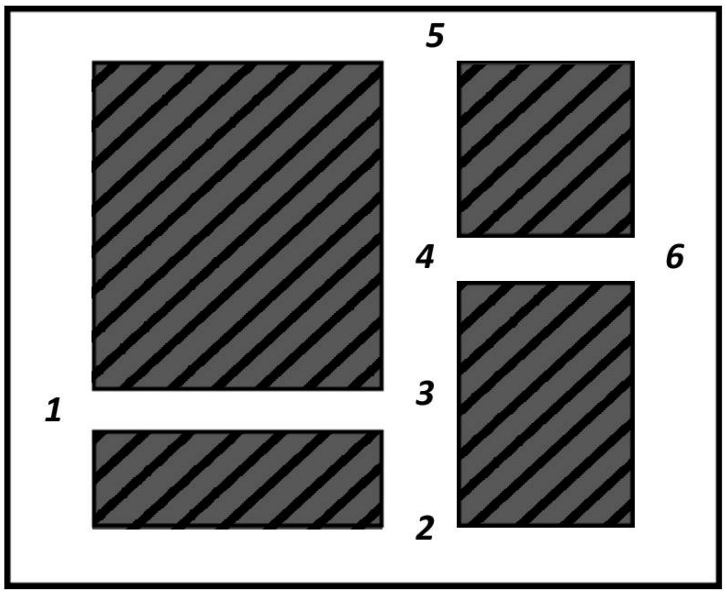
Pac-dots

RFID cards are used to represent the Pac-dots. In order to increase the probability that the robot will actually read them when he passes near them, at least 5 cards are used for each Pac-dot (the more cards, the higher the probability that the robot will read at least one of them). They should be be put in a way to maximize the area they cover, for instance like shown below:



Because the robot tends to move them when he passes on top of them, it is recommended to fix them to the ground somehow (for instance using adhesive tape).

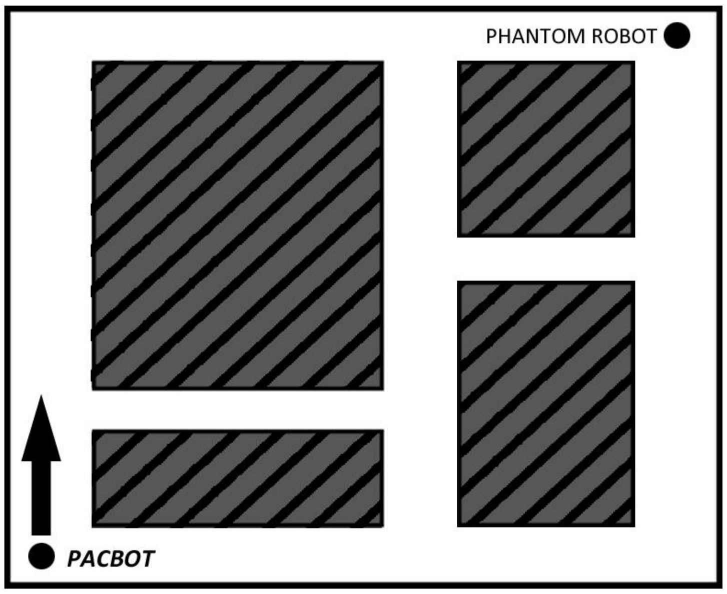
Furthermore, the robot uses the Pac-dots to find its current position in the maze and decide the direction he should follow next. Therefore the Pac-dots should be distributed in the maze in each crossroad, in the following order (RFID Cards are all numbered):



3.5 Game rules

Preparation

Initially the Pac-Bot robot should be put in the top-left corner, facing east, whereas the Phantom robot in the bottom-right corner, like shown in the picture below:



Progress of the game

During the progress of the game, the player commands the Phantom robot to follow Pac-Bot in order to make contact. In case Pac-Bot gets a special Pac-dot that makes him turn into Super Mode, the player must command the Phantom robot to stay away from Pac-Bot. In case of contact while Pac-Bot is in Super Mode, the winner is Pac-Bot. Otherwise the winner is Phantom Robot. Pac-Bot can win the match if he gets to all the Pac-Dots and no contact was made up until then.

4. Hardware Architecture

4.1. Elaboration Unit

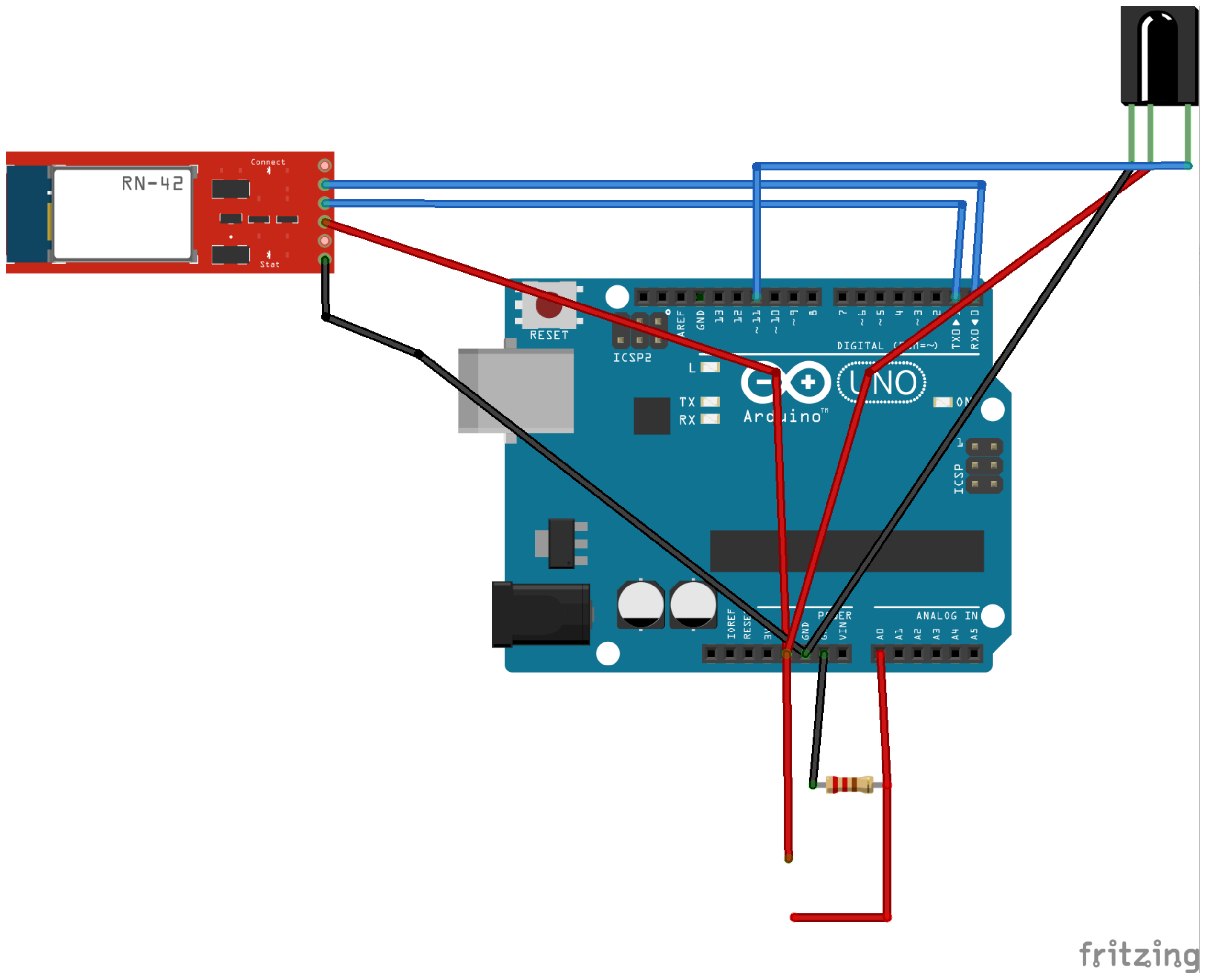
Elaboration of the sensor data is done in an external computer. The robots send the data coming from the sensors towards the computer (SpyKee uses XBee radio communication, PhantomRobot uses bluetooth), and the latter uses this information to take decisions and send them back to the robots in order to act.

4.2. The Pac-Bot Robot

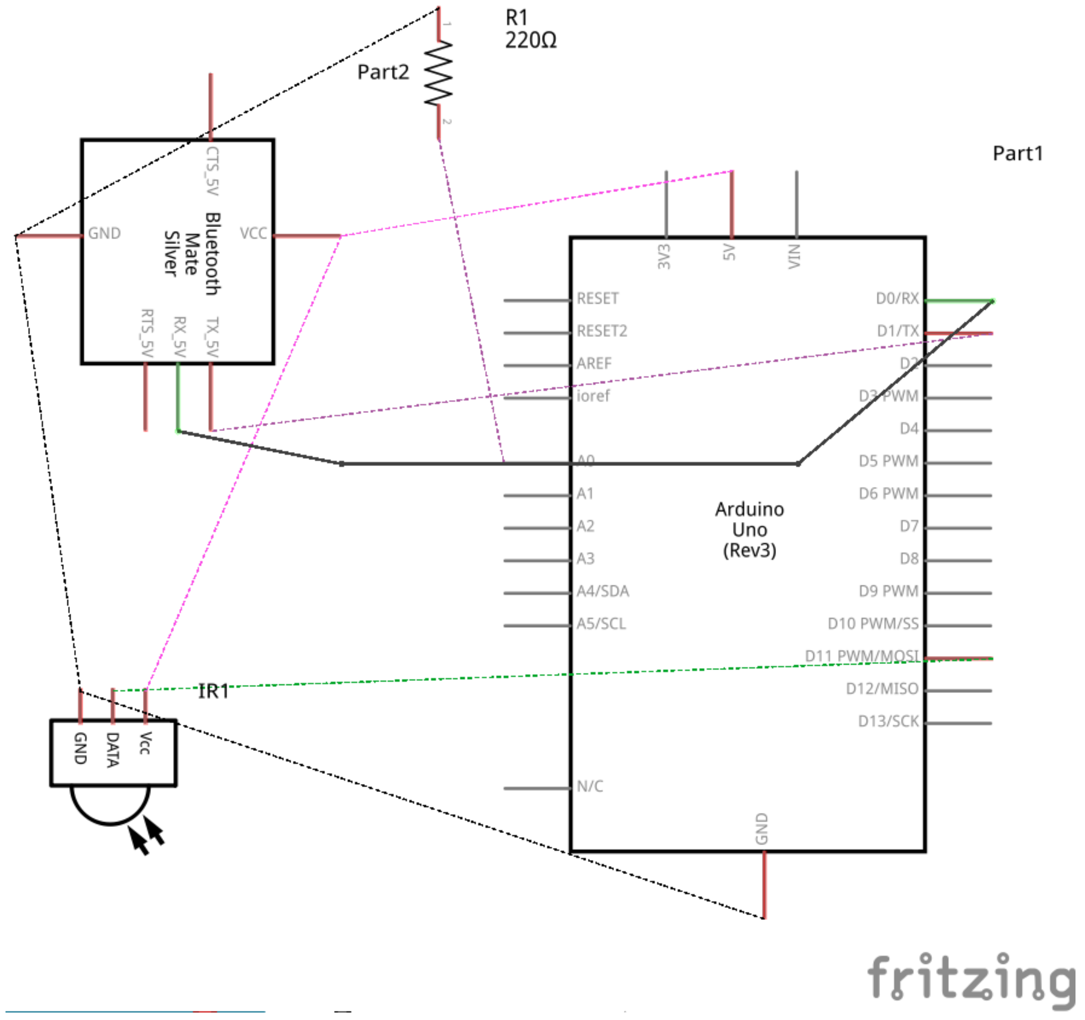
The principal robot in this game is Pac-Bot, represented by a SpyKee robot produced by Meccano. This robot was used previously in different Robogames projects of the Artificial Intelligence and Robotics Laboratory (AIRLab), most recently in the project “RoboTower”. Most of the modifications and hardware implementations were featured in the latter, for further information refer to the bibliography.

The only feature added for the purpose of the Pac-Bot project was the IR proximity sensors, two in front of the robot and two to the sides.

4.3. The PhantomRobot



The PhantomRobot was assembled and programmed to represent the ghost (or phantom) that competes with Pac-Bot. It features an Arduino UNO microcontroller board with an ATMega328 chip, a Keyes L298 motor driver, an Infrared receiver, and an JY-MCU bluetooth module. In order to detect contact with the other robot, an ad-hoc circuit was implemented as shown in the figure.



4.4. Pac-dots

To represent Pac-dots, RFID tag cards were used, more specifically the EM4001 64-bit RFID which is compatible with the RFID reading sensor mounted on SpyKee (ID-12).

5. Software Architecture

The software source code of the game can be found on the github repository (https://github.com/mikelv92/PacBot). For the Echoes, SpyKee and Vision packages, refer to the "RoboTower" project, as they were imported and not modified.

5.1. PhantomRobot

This node is responsible for reading the bluetooth stream coming from the PhantomRobot. This stream notifies the AI node whether a contact between the robots have been observed or not.

5.2. AI

The AI node is responsible for the artificial intelligence of PacBot. In this node sensor data coming from Echoes is processed, elaborated, and based on that decisions are taken for the future actions to undertake. AI features in particular a fuzzy logic inference engine (MrBrian) which processes the IR sensor distance data and based on the fuzzy rules implemented on it takes decisions to send to the actuators of the robot (i.e. speed and direction). MrBrian infers decisions taking note also of the presence or absence of the enemy (PhantomRobot). In fact, the size of the blob received from the Vision node is passed to MrBrian, and the fuzzy rules take more informed decisions. In order to follow a specific path inside the maze, the robot uses the RFID cards to get oriented and infer the position and direction it is facing.

More specifically, the fuzzy rules are organized in three levels. In the first level, the paradigm is to try and keep the robot in the center of the path, keeping west and east distances more or less equal. The paradigm shifts to the second level in case a 90 degree turn is in order. This happens either when in front of the robot there's an obstacle or when the robot reads an RFID card that instructs to take a turn. The robot has to follow a certain path inside the maze, therefore it keeps in memory the last RFID read and comparing to the current one it is able to find the horizon direction of movement.

These two paradigms are to be ignored in case the ghost is spotted in front and the blob is large enough to represent the risk of losing the game (in case PacBot is not in super mode). Should this occur at any time, the rules at level three are executed.

The configuration files are:

- behaviour.txt - contains the list, organized in levels, of the defined paradigms. Each paradigm, consisting in a set of fuzzy rules, is contained inside a file .rul
- ctof.txt - associates to every crisp data in input a fuzzy set, used for fuzzyfication. The sets are defined in shape_ctof.txt
- s_ftoc.txt defines the output values of Brian, associated to the fuzzy sets used for defuzzyfication. The sets are defined in s_shape.txt
- Predicate.ini - defines the fuzzy predicates based on the fuzzy variables and/or other predicates
- Cando.ini defines the conditions of activation for a determined paradigm (the necessary conditions to execute that paradigm)
- Want.ini defines the conditions for which it is appropriate to activate a determined paradigm

The paradigms implemented to the robot are:
Level1:
- KeepStraight - try and keep the robot in the center of the path

Level2:
- MakeTurn - rotate 90 degrees in case of an obstacle in front
- RFIDTurn - rotate accordingly when a RFID card is read

Level 3:
- GhostSeen - Decide whether the ghost is presenting any risk
- CriticalMakeTurn - Make turn in case the robot has seen the ghost and needs to get away as soon as possible

6. Conclusions

Originally sonar distance sensors were used from this project which were inherited from the RoboTower project. From the tests made in the lab conditions it was observed that they were not appropriate for this kind of project. Because of the maze's layout the sonar signals would interfere with each other, therefore it was decided to switch to IR sensors.

The camera is very sensible and it is able to distinguish blobs in a very satisfying way.

From the tests made the game results playable. The lack of accuracy in sensorial data is compensated with the low speed of the PhantomRobot, therefore the game seems fair to both ends (user and computer). Because of the low range of the IR receiver sensor on the PhantomRobot, the user is forced to move around the maze in order to command the robot, resulting also in a slight physical movement.

Special thanks

I would like to thank prof. Andrea Bonarini, Davide Tateo, Martino Migliavacca and Marcello Pogliani for the essential help in this project.

A. Installation and startup

The software was developed to work under a Linux system, more specifically referred to the Ubuntu distribution.

The appropriate ROS middleware version is Groovy, although other older versions work too.

NOTE: ROS should compile the packages using rosbuild, the newer versions that use catkin are not compatible.

Dependencies

In order to compile the software, the following packages must be installed beforewards:

- `flex` and `bison` - used to generate the parsers of Brian
- OpenCV libraries (`libopencv-dev`) - used to process the images in Vision and LittleEndian
- `ann` and `opencv2` - to install these two packages refer to the RoboTower documentation, in the appendix section

Launch of the game

First of all the computer should connect to the XBee and the wireless network created by SpyKee. The SSID is SPYKEE followed by an ID code of the robot. It is not required to insert manually the parameters for the connection (IP address, …), because the robot runs on a DHCP server. If the Echoes node is not able to access to the serial port, it might be possible that the permissions of the user are not adequate to open the device: to resolve this problem it is sufficient to add the user to the group `dialout` or set the permissions through `udev` rules.

Afterwards the bluetooth connection must be set up. Use the Blueman package to do the pairing between the computer and the PhantomRobot. The name of the device is ‘`linvor`’ and the password is ‘`1234`’. After the pairing completes the game is ready to be launched using:

```
$ roslaunch spykee.launch
```

Bibliography

[1] Progetto RoboTower, http://airwiki.ws.dei.polimi.it/index.php/RoboTower
[2] ROS Wiki, www.ros.org
[3] MRT: Modular Robotics Toolkit, http://airlab.elet.polimi.it/index.php/MRT
[4] Arduino, http://www.arduino.cc/