Embodied Reactive Agents

Marcos García, Borja Jiménez

Jaume I University, Spain

al375909@uji.es

al361929@uji.es

***Abstract*—. Embodied Reactive Agents are an important research field in robotics. This technology consists of the use of an agent inside a real or simulated environment with the aim to solve a given task. We decided to implement it in a simulated environment to achieve two different objectives. Our simulated test bench is split into two different areas. The first task is related to visual recognition and obstacles avoidance, and the second one could be summarized in a pick and place task, where the agent has to find the object and place it in the correct place. Once the robot was configured properly it was capable of performing both tasks.**

***Keywords*— Embodied agent, Reactive system, Pioneer, Try-a-bot, Environment.**

I. Introduction

In computerized reasoning an embodied agent, sometimes denoted as an interface agent, is an astute operator that communicates with the environment through a physical body, usually using a set of sensors that provide it with the necessary data about the surroundings [1]. This branch of artificial intelligence combines research from different fields such as psychology, sociobiology, and biology. Especially in this paper, the model has been implemented in virtual environments, therefore computer graphics, animation techniques are also present [2].

The term *reactive*, refers to the connection between the sensor that records the data, and the motors of the agent that work as the powerful force, which are capable of producing a physical reaction to the sensor input. This conversion of inputs into actions is denoted as the "agent function". It is not a specific command given as a piece of code but a concept. The conjunction of all the programs present in the code of the agent and the synchronous execution of it, produces the given function.

The conjunction of the previous features can lead to a *rational agent*. This term in artificial intelligence refers to an agent with clear preferences with the objective of maximizing the expected value, therefore executing the most efficient and reliable action. In order to obtain the level of rationality of a robot, the measurement must be adapted to the specific work that the agent is expected to get done.

The remaining sections of the paper are as follows: Section II presents the fundamentals and types of intelligent agents; Section 3 works as a short introduction to the material used to carry out the tests, that include the Pioneer and the simulator used as the environment. Section 4 describes the simulator and the programming language used to control the robot; Section 5 includes the development of the code and the utilities of the Pioneer. Finally, Section 6 is the overall conclusion of the two tests.

II. Intelligent Agents

Intelligent agents refer to entities capable of acting autonomously. They may also learn to use knowledge to achieve the desired goal by evolving from a very simple computational structure to a complex one. In all the situations the structure can be reduced to a simpler mathematical concept: the agent function, as we previously mentioned. The set of programs of the agent (P*) may lead into a physical reaction (A) as shown in Fig. 1.

$$f : P^* \rightarrow A$$

Figure 1 Mathematical representation of the Agent Function

Agents can be grouped into five different classes based on the degree of perceived intelligence and capability [3].

A. Simple Reactive Agents

Simple reflex operators act based on the present perceived, disregarding the remainder of the percept

history. The specialist work depends on the condition-activity rule: "if condition, then action".

This agent will only succeed when the environment is completely observable. Some reflex agents can sometimes contain data on their present state which permits them to ignore conditions whose actuators are already noticing.

The main weakness of the simple agents is the infinite loops, which are regularly unavoidable. Only if the activities of the agent are random, it might be able to escape from unending circles.

B. Model-Based Reactive Agents

Model-Base agents expand the utility of the simple agents. They can handle partial observable environments by storing their actual state and using it to create a representation of the surroundings that are not accessible to their sensors. This representation is denoted as a model.

The main drawback is that the perception of the agent may be wrong, then leading to a misunderstanding and a poor performance.

C. Goal-Based Agents

These agents further expand the capabilities of the previous agents based on models of the environment, these use the goal or objective information to influence the desirable execution. This allows the agent to choose among multiple options that lead to the expected goal. In order to produce all the possible paths, the program must consider the future.

It is important to consider that this model only provides binary options. It selects a possible execution either as a goal state or as a non-goal state. So the model is able to calculate quantitatively how accurate a given option can be.

D. Utility-based Agents

Taking into account the limitations of the previous model, the utility-based agent obtains a measurement of how desirable a particular state is by computing the utility function. It is created by assigning a number to each state.

E. Learning Agents

The most distinctive feature of this type of agent is its ability to operate in unknown environments and become more competent than its initial knowledge might allow. This is possible thanks to its learning feature responsible for making improvements over time. In order to know how good the current state is and consider possible improvements, the agent needs a way to compute its score-performance. It is done by using a score function adapted to the specific work that the agent is designed to accomplish.

III. Material and Methods

Once the fundamentals of intelligent agents have been exposed, we are going to introduce the simulator and the Pioneer robot with which we carried out the task the professor asked to solve in the laboratory section of the Intelligent Systems course. They consist of two main tests.

The first challenge is a circuit surrounded by walls, as Fig. 2 shows, where the goal of the agent is to reach the end. It should follow the purple line using the shortest path and avoiding any obstacle that could cause a collision.

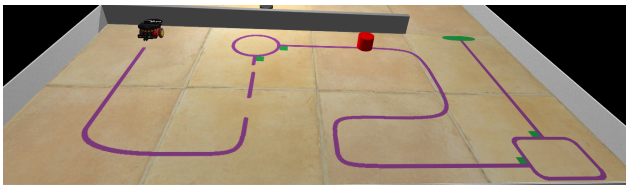

Figure 2. Fist challenge environment consisting of a circuit and an obstacle

We have divided the challenge into two main parts. First, the agent has to be able to detect and follow properly the line and always choose the shorter option. These situations refer to the two roundabouts where the correct turn is indicated with green squares on the floor. Second, the robot has to be aware of the path and avoid the red obstacle present at the mid-circuit.

The last experiment we realized with the Pioneer is a bit more complex than the fist one. In this case, we want to use the robot to find a set of five blue balls and place them in a red area one by one. The continuation of the last environment can be seen in Fig 3.

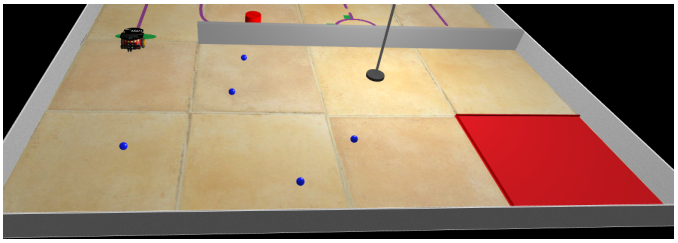

Figure 3. Second challenge environment with the five blue balls and the red base where they have to be put.

For this new challenge, we have to make use of another tool: the motorized tweezers. As we explained in Section V-A the tool allows the robot to manipulate a physical object, in our case, the blue balls. This skill is totally dependent on the vision provided by the Kinect. Otherwise, the robot will not be able to find the balls and orientate in the space just like in the first challenge.

The execution of this test can be split into two different sections: finding and grabbing one ball at a time and then, driving it to the base and placing it at the border.

IV. Try-a-robot Simulator

The simulator used to carry out the experiments consists of a web interface where the virtual robot is running in a 3D environment, as Fig.4 shows. The web interface uses Jupyter Notebook for program development and sending commands to the virtualized robot. As a programming language, we used Python and the Pioneer library that includes all the commands and functions needed to control the robot. At the same time, the simulator also allows to change the environment or world, so different tests can be done [4].

Figure 4. Try-a-Bot VM running on Google Chrome window with the Pioneer robot.

V. Pioneer

The Pioneer, as shown in Fig 5, is a robotic wheeled base designed for autonomous navigation. It comes with SONAR, battery, wheel encoders, retractable tweezers for handling objects and other features and expansion accessories that we did not use in our tests.

Figure 5. Pioneer in the virtual simulation with the equipment used in the tests

We have splitted the Pioneer´s abilities used in the tests in four main sections:

A. Motion

The Pioneer has two motored wheels and a freewheel at the back to give stability to the vehicle. In conjunction, they form the motion system.

There are four possible states (Fig. 6). If both wheels spin in the same direction and velocity it will move in a straight line. In case the wheels spin in different velocities, the robot will turn to the direction of the slowest wheel. Finally, to turn around, the spin of the two wheels must be opposite. Therefore, the different motions of the robot are directly related to the velocity and direction of the wheels spin.

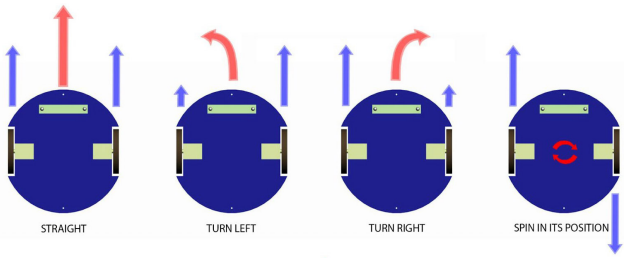

Figure 6. The different motion types.

Pioneer provides a value denoted as Encoder that is used to determine both the distance travelled in a straight line and rotating. To use the Encoder we

need to know the length of the wheel axis (L) and the radius of the wheels (R).

To communicate with the robot and send the motion commands, the Pioneer library provides three functions: *move(ls, rs),* that receives the left wheel speed as the first attribute and the right wheel speed as the second attribute, both in rad/s; *stop(),* that will shut down all the motion in action; and *sleep(t),* where *t* is the amount of time the robot will execute the movement.

B. *Ultrasonic Sensing*

The robot has 8 ultrasonic sensors in the front, numbered from 0 to 7 starting with the left sensor. These sensors work estimating the arrival time of a high-frequency sound wave transmitted by the sensor. As the speed of the sound is known, we can calculate the distance between the robot and the obstacle.

A threshold is used to determine if the robot is close to the obstacle or not. The value returned by the ultrasonic sensor is compared with the threshold value and if the first is lower, then the obstacle is close, so actions should be taken to avoid any collisions.

To obtain the values from the sensors, the Pioneer library gives a function called *distance[n]* where *n* is the number of the sensor that we want to check.

C. *Vision*

For our tests, the robot was equipped with a Kinect camera. This accessory provides both coloured images and depth map, although we only used the colour feature to detect objects.

The Pioneer library has a set of functions that allows us to work with the Kinect. Physically the camera can be tilted within a certain range, then improving the vision range as needed. Using the *tilt()* function with a value between 0.47 and -0.47 as input, the orientation of the camera can be changed. Once the camera is set, we can take a shot by using the *image()* function.

As we mentioned at the beginning of this section, the main goal of implementing a vision to the agent is to distinguish objects in the environment. For this purpose, certain tweaks in an image need to be done. First, we have to set the minimum and maximum range of the colour of the object we want to detect. Using those values as a colour filter, we can obtain a picture where only the object we are looking for is present. In some cases, it is useful to introduce a mask that restricts the view in the image to avoid any possible confusion and to be more precise in the selection of the object. This is done by setting to 0 certain range of pixels.

Once the picture has been modified we can obtain the exact position of the object as coordinate in the picture. This point is denoted as a centroid. It is important to mention that if several objects are detected, the centroid will represent the mean distance between each other, instead of the actual position.

D. *Manipulation*

The robot has a gripper tool used to grasp objects just below the ultrasonic sensors. For this purpose, the fingers of the grasp close horizontally. This piece is able to move vertically in order to grasp objects from the floor, but also it can move horizontally to help the grasp of the object.

To control the gripper the Pioneer library uses the function *p3dx.gripper(lift, fingers)*. The first value is the height of the gripper. The interval for this value is -0.05 for putting the gripper up and 0.05 for putting the gripper down. The second value represents the amplitude of the fingers, the interval for this value goes from 0.0 when the fingers are closed to 0.1 when the fingers are open.

VI. Results

Once we have explained the abilities of the Pioneer individually we can behind the main two experiments where all the four skills work together to achieve the goal of the tests.

A. *Challenge* 1

As we said, we have divided the challenge into two main parts. For the first task, we have to make use of the motion system to drive the robot, and the vision to follow the line. Both of those systems are controlled from a single function *follow_line()*. This

function will work until an obstacle is detected through the ultrasonic sensors, or the end of the path is reached. In order to detect only the nearest part of the line, the vision system titles the camera down and looks for a purple object below the 80 horizontal levels implementing a mask that sets to 0 all the pixels above this threshold. Once the line is detected, the Pioneer will adjust the trajectory to its shelf in the middle of the path by computing the centroid. At the same time, the vision system is also looking for green objects, equivalent to the green squares, that indicate the turn at the roundabouts. So, when one is detected the pioneer will follow it causing enough movement to result in the correct orientation and after that, the robot will keep tracking the purple line.

Finally, when an obstacle is detected by the ultrasonic sensors the *is_obstacle_detected()* function is triggered. This only happens when the object is within a determined range. In our experiment, it is 0.22m and 0.26m. Next, the *get_wall()* function takes over. It places the robot perpendicular to the object and then the function *follow_wall()* will start to drive the Pioneer around the object until the line is detected again. So in our experiment, this process produces a semi-circular trajectory around the red obstacle. Once the line is again in the field of vision of the Kinect, the function *get_line()* is executed. It turns the robot until the line that is again in the middle of the image, and therefore, the robot is aligned with the path. This is done using again the centroid of the line as reference. After the alignment, the robot keeps executing the *follow_line()* function until a new event interrupts it.

At the end of the circuit, the Pioneer has to stop at the big green base. This is done by using the same function that detects the green squares. In this case, it has an *if-statement* that can determine whether the green object is a turning square or the final base. To classify the object the area is computed as an attribute, it is obtained while processing the image like the position of the centroid. After some tests, we determined that this area is 150000.

B. Challenge 2

First, the function *center_ball(tilt)* is executed where *tilt* is used to change the orientation of the camera. In the beginning we want to find one ball, so the tilt is set to the minimum providing a more wide field of view. *center_ball(tilt)* starts rotating the robot counterclockwise until a sub-function indicates the presence of a blue ball. This function is *close_ball()*. The robot takes a picture of the environment, sets a mask on both sides and begins from the bottom to the top of the image to look for a blue object. This technique is needed to avoid situations where several balls are in the picture, otherwise when the centroid is computed it won't indicate the position of one target but the average position of all of them.

Once the *center_ball(tilt)* has orientated the robot to a ball, it starts approaching the target. By using *get_ball_close()* the camera will tilt to the maximum, restricting the view of far objects. It will start driving the agent forward until a blue object is detected and its centroid if below 50 on the horizontal axis. At this time, the robot is stopped in front of the blue ball but it is still not close enough to catch it. We decided to stop at this position because the original orientation could contain some errors that could cause the robot not be perfectly aligned with the ball.

After the initial approach, the *grab_ball()* function is executed. It gets the pioneer close to the ball using a similar technique than *get_ball_close()* but this time using 90 as a horizontal threshold. Once it is close enough to the ball, the function moves the grippers down, closes them and moves them up again. To check whether the ball is grabbed or not we use *is_ball_grabed()* that works using the area of a blue object. After some tests, we determine that if the area of a ball is equal to 86 the pioneer has grabbed the ball. In case the process to grab the ball has failed, the robot will drive backwards and execute *get_ball_clase()* again until it produces a successful attempt.

Now that we know that the first part of the challenge is done we can continue with the last part: put the ball inside the red area and place the robot properly to begin with the search of new balls.

For this part, we begin by executing the function *center_base()* that works in a similar fashion to the *center_ball()*. In this case, we seek to orientate the Pioneer to the red base instead to a blue ball, but we don't have to worry about finding multiple objects due to there is only one red object in the scene. So the function just turns the robot until a red object is detected and its centroid is between 70 and 90 in the vertical axis.

Once the alignment is completed we can approach the robot to the base. This is done using the *get_base_close()* function. Like in the case of the balls, the camera is tilted down and the robot drives forward until the function detects a red object. In our test, the threshold for the centroid was 35 in the horizontal axis. This results in the robot leaving the ball just when it is inside the platform but the rest of the body of the robot is still outside.

Finally, the robot can drop the ball using the *drop_ball()* function that simply opens the grippers and the ball falls. This function also works as a reset: it will drive the robot backward for a given time, in or test 7 seconds was enough, and then it will turn the robot until the red base is not detected anymore. Using this technique we tried to avoid any possible confusion where the robot could detect a blue ball that is already in the red base.

VII. Conclusion

The reactive agents have to be adapted to the environment in which they will develop, both in software and hardware. It is,, if the robot only needs to follow a line painted on the floor and we are sure that there will not be obstacles in the way, we will only need the Kinect camera and we can dispense of the ultrasonic sensor. Moreover, it is important to divide the work that we want the robot to do into small actions and be clear what it will do in each of them. In this way, is how the agent develops more complex actions, through a succession of simple actions.

As we have seen, there are different types of reactive agents, therefore it is necessary to make a previous analysis of the environment to decide which type we choose. Thus, we can conclude that reactive agents have great potential. This can be clearly seen through the challenges we have performed, where with only a few sensors, the robot has accomplished all of them successfully.

References

[1] https://wiki.org/Embodied_agent

[2] Handbook of Virtual Environments: Design, Implementation, and Applications.

[3] Russell, Stuart J.; Norvig, Peter (2003). Artificial Intelligence: A Modern Approach (2nd ed.).

[4] https://sites.google.com/uji.es/try-a-bot/

[5] https://www.inf.ufrgs.br/Manual_pioneer3.pdf