Abstract. In this paper we describe a system to simulate and operate Virtual and Real robots (ViRBot). The system consists of several layers that control the operation of real and virtual robots. A 3D graphic engine interface allows to test multiple virtual robots, that are a close simulation of the real ones, the virtual robots are able to execute the same commands, using a robot’s programming language, that real robots can, including behaviors, movement equations and sensors readings.

1 Introduction

In the development of algorithms to control mobile robots it is necessary to simulate them, using virtual robots, in order to test their performance before implementing them in the real ones. This is done because sometimes it is too expensive or unfeasible to execute hundreds of testing operations with a real robot. Therefore, we have developed a system, named the ViRBot, where operational algorithms for mobile robots can be tested using virtual robots with the same conditions as if they were the real ones. The main idea is to be able to control the real and virtual robots indistinctly and to test the mobile robots algorithms using first the virtual robot and the real one later. The ViRBot system allows the testing of multiple virtual robots, at the same time, of different forms and sizes, multiple viewpoints, wired frame scenes, shaded or textured polygons, interaction with the system using different interfaces such as keyboard, mouse, joy pad and internet. Every part of the ViRBot was developed by the authors of this document at the laboratory of Human Interfaces.

The virtual environment was developed using a virtual reality toolkit called WTK Sense8 that gave us a powerful tool to program 3D world applications using models [1]. Using this tool is also possible to create boxes, tables, buildings, factories, dungeons, labyrinths and everything needed to construct different environments where the virtual robots can be tested.
2 Mobile Robot Operation

There are several architectures to control mobile robots, our architecture is based on the Aura architecture developed by Ronald Arkin [2] [3]. In our approach the mobile robot's operation is formed by several layers, see Figure 1, each one having a specific function that in whole control the behavior of the robot.

Figure 1. Virtual and Real Robot (ViRBot) System.

Each of the layers are described briefly in the following sections, given more details on the virtual environment and the simulator.

2.1 Perception

One of the main objectives of the perception module is to obtain a symbolic representation of the data coming from the robot's internal and external sensors, from the Human/Robot interface, from the robot's tasks and if chosen from the simulator. The symbolic representation is generated after applying digital signal processing algorithms on the data generated by the sensors. With this symbolic representation a belief is generated, later with the world representation a situation recognition is created.
2.2 Internal Sensors

Each of the sensors is represented by a data structure that has the following elements: sensor's name, sensor's type, sensor's position in the robot and a set of the sensor's values. We use a B14 robot, developed by Real World Interfaces, that it has two wheel encoders and a battery level detector.

2.3 External Sensors

The robot has an array of 8 infrared sensors, 8 sonar sensors, 20 tactile sensors and a video camera [8].

2.4 Simulator

In the ViRBot system, it is possible to simulate each of sensors' signals according to a physical model of the sensors. Thus, when the system is tested using the virtual robot, the simulator provides each of the sensors' values. A user may include his own simulation algorithms [3]. The simulation of the sensors is explained in detail in section 3.

2.5 Human Robot Interfaces

The communication between the robot and a user is done by several means: speech recognition, text provided by keyboard and control pads.

2.6 Robot's Tasks

The robot is programmed to perform certain tasks, such as picking objects and delivering them to another place, during the day at a certain time.

2.7 World Model

The world model module uses the belief generated by the perception module and together with the information provided by the cartographer and the knowledge representation it generates a situation that needs to be solved.

2.8 Cartographer

For every room of the working environment there is a representation of how each of the rooms are interconnected between them, as well as each of the known obstacles included in them. The links between rooms are used later to form a tree that it has in the root the origin room and one or several leafs have the destination room.
The known obstacles are defined as polygons which consist of a clockwise ordered list of its vertex. Representing the obstacles as polygons makes easier the search for a solution to navigate from one point to another. They are represented as a fact in an expert system as follows:

\[(\text{polygon type obstacle' room obstacle' name } x_0 \ y_0 \ ... \ x_n \ y_n)\]

where type represents the type of obstacle, such as a wall, a desk, etc. The \(x_i\) and \(y_i\) represents the coordinates of the obstacle. From this obstacle representation forbidden areas are created, which are areas not allowed for the robot to enter, they are built by growing the polygons by a distance greater than the radius of the robot, to consider it as a point and not as a dimensioned object [7]. Figure 2 shows the forbidden areas of the experiment’s room. It is possible to create the configuration space in this way because our robot has cylindrical shape.

![Figure 2. Symbolic representation of the experiment's room.](image)

## 2.9 Knowledge Representation

An expert system is used to represent the robot’s knowledge. In an expert system, the knowledge is represented by rules, each one contains the encoded knowledge of an expert, that is, the actions that the robot would do if certain conditions were met.

The environment was defined as facts in an expert system, we used the expert system shell called CLIPS, developed by NASA [4], in which we add a TK graphics environment and sockets communication that allows to send data to it and to the robot through Internet.
2.10  Goal Activations

Given a recognized situation, a set of goals is activated in order to solve them.

2.11  Options

Set of hardwired procedures that solve, partially, specific problems.

2.12  Planner

The basic problem of planning the robot’s movements is reduced to: Given the initial position and heading of the robot $A$ in space $W$, a path $\tau$ specifying a continuous sequence of positions and headings for the robot $A$ must be found in order to avoid collisions with the obstacles $B_i$’s, beginning in the initial position and heading and finishing in the goal position and heading. If there is not such path, the impossibility to solve the problem is reported.

Planning is defined as a procedure or guide for accomplishing an objective or task. This requires searching a space of configurations to find a path that corresponds to a set of the operations that will solve the problem. For example, in our system we may want the robot to move from one place to another and to take a picture. This problem will be solved by finding a sequence of operations that leads from an initial state to a goal state. Then the objective of planning is to find a sequence of physical operations in order to achieve the desired goal.

The robot’s planning organization consists of several hierarchical layers. The top layer is the planner, which takes as input an initial and a final state and an environment description, and produces an overall plan that will lead the robot from the initial to the final state. If the command was to move the robot from one room to another, the planner would find the best sequence of movements between rooms until the robot reaches its destination. Inside each room it also finds the best path considering the known obstacles, that represent some of the objects in the room. Each of the objects in the environment is represented by a set of polygons whose are used by the planner, in order to find the best path avoiding those that interfere with the goal.

2.13  Navigator

The navigator takes the overall plan and finds a set of trajectories that consist of distances and angles that the robot needs to execute in order to reach its destination. Given a set $(x_1, y_1, x_2, y_2...x_n, y_n)$ of points obtained by the planner and a time $t$ to visit them, the navigator finds the angle of rotation $\theta_i$, a distance $d_i$ and a speed $v_i$ to reach all of them.
2.14 Pilot

The pilot takes the trajectories generated by the navigator and executes them. Basically, it has to move the robot a distance $D_i$ and turn $\theta_i$ radians in each step $i$. It also search for unknown obstacles to the planner and tries to avoid them using behavioral control. look for

2.15 Controller

It controls the robot's motors and reads incoming data from the motors and sensors. The mobile robot that was used to test the ViRBot system is a cylindrical type mobile robot equipped with a wheeled base that allows two types of motion: translation (movement parallel to the robot head alignment) and rotation (movement perpendicular to the robot head alignment), see figure 3. The robot has motion controllers that control its movements, it also has odometers to measure distances.

![Figure 3. The Robot B14.](image)

2.16 Learning

The system can learn how to solve new problems using genetic algorithms, probabilistic methods, Hidden Markov models, etc.
3 Sensors’ Simulation

The simulation module allows the creation of virtual robots with multiple sensors of different types such as contact, reflective, sonar and infrared sensors.

a) Contact sensors. In order to simulate contact sensors we developed a simple polygon cross algorithm between a contact prism (sensor) and all the world (obstacles), so if the prism cross an obstacle the sensor’s status change to on, indicating a collision with an obstacle.

b) Reflective sensors. These types of sensors are useful to create line followers robots. The virtual robot gets the track/floor values returned by the sensors in the robot’s sensor position. To simulate them an average of the texels are calculated in the position where the sensors are. Texel is a texture element, the base unit of a texture over a 3D polygon [1].

c) Sonar sensors. Sonar sensors are simulated by getting the position where the sonar beam strikes an object.

d) Infrared sensors. As in the sonar sensors to simulate the infrared ones it is necessary to get the position where the infrared beam strikes an object.

Collisions with obstacles are checked all the time, the purpose of this is to simulate the robot’s physical interaction with its environment. Thus the virtual robot can not pass through any simulated object, if it strikes an obstacle then it needs to execute additional behaviors related to the strike. The collision algorithm is based in bounding box and polygon crossing.

An user of this system can incorporate its own simulation of a particular sensor, thus a sonar specialist may test the operation of a particular sonar sensor whose physical behavior has been modeled and include it in the system using standard C/C++ routines.

Random noise can be added to the readings of the sensors, the user can select either if noise is added or not, as well as the mean and the variance of the Gaussian random variables that are added.

4 Robot’s movements

The simplest way to define the movement of a robot is through a sequence of straight lines [5], see figure 4. Basically to follow a straight line the robot requires a transition from a straight line to another, and this is obtain by coordinate transformation.
The robot’s position is described, in time $i$, by $(x_i, y_i, \theta_i)$ where $\theta_i$ represents the angle of the robot with respect to the $x$ axis. The system of coordinates $x', y'$ represents the new axis of coordinates after a rotation and displacement of the robot. Figure 5 shows these systems of coordinates from which following equations are obtained.

\[
x_i = \overline{OA} = x_{i-1} + r\cos(\theta + \theta’) = x_{i-1} + r\cos(\theta)\cos(\theta) - r\sin(\theta)\sin(\theta).
\]  
\[
y_i = \overline{AP} = y_{i-1} + r\sin(\theta + \theta’) = y_{i-1} + r\sin(\theta)\cos(\theta) + r\cos(\theta)\sin(\theta’).
\]  
\[
x_i’ = \overline{OA'} = r\cos(\theta’).
\]  
\[
y_i’ = \overline{AP} = r\sin(\theta’).
\]
\[ x_i = x_i' \cos(\theta) - y_i' \sin(\theta) + x_{i-1}. \]  
\[ y_i = x_i' \sin(\theta) + y_i' \cos(\theta) + y_{i-1}. \]  
\[ \theta_i = \theta_i' + \theta_{i-1}. \]  
\[ x_i' \text{ and } y_i' \text{ represents the displacement of the robot and } \theta' \text{ the angle of rotation in the new axis. If } y_i' = 0, \text{ then if the robot rotates } \theta_i \text{ radians and advance a distance } x_i = d_i, \text{ then the equations for the new position of the robot are:} \]
\[ x_i = d_i \cos(\theta_i) + x_{i-1}. \]  
\[ y_i = d_i \sin(\theta_i) + y_{i-1}. \]  

An important part of the simulation are the movement equations, that have to be solved in any step of the simulation engine. These equations take in consideration the speed and acceleration that the robot needs to follow, the next equations are used:

Position: \[ x(t) = a_0 + a_1 t + a_2 t^2 + a_3 t^3. \]  
Speed: \[ \frac{dx(t)}{dt} = a_1 + 2a_2 t + 3a_3 t^2. \]  
Acceleration: \[ \frac{d^2x(t)}{dt^2} = 2a_2 + 6a_3 t. \]  

Given the initial and final position, and the initial and final speed the \( a_i \) s constants are found.

An user of this system can incorporate its own simulation of the movement of the robot, thus a researcher wishing to test a control algorithm may include it in the system, using standard C/C++ routines.

To make more realistic the movements of the simulated robots, random noise can be added to the final movements, the user can select either if noise is added or not, as well as the mean and the variance of the Gaussian random variables added.

## 5 Simulation process

The simulation process [6] is described as follows, see figure 6:

Create and initialize the 3D world. The world is created adding all the objects that will exist in it, including rooms, models, lights and textures.

Create the robot. Here, using simple objects or 3D models, all the robots that will be used are composed, loaded and initialized to work in the system.

Assign default behaviors to the robot. Default values and tasks needed to make the robots work are attached.
Initialize the communication. Here all the communication devices are initialized. If it is required, the communication with the real robot via Internet sockets starts.

Wait for an order. Constantly the data coming from an internet socket, joy pad and keyboard buffers are sensed, and the information received is analyzed.

![Figure 6. Blocks diagram of the simulation process.](image)

Execute an order. When a valid command is received it is necessary activate certain tasks needed to execute the command.

Finish 3D world. Kill all processes attached to world; send finish communication signals and close all windows related.

6 Robel

To improve the way to program and test robots, we have developed a robot’s programming language, robot behavior language or Robel, as an easy way to program and assign basic tasks (commands) to the robots. In order to use Robel, it is necessary that in the robot has been installed a Robel Virtual Machine (RVM), that execute
commands written in that language. Basically any virtual robot in the simulation is loaded with the RVM. We developed the RVM for a Real World Interface robot, for a custom robot based on the 6811 micro controller and for the virtual robots. The Robel’s instruction set is composed by 16 basic instructions which can be classified in movements, status, sensors’ readings, internet interfacing and conditionals:

Movements. These type of instructions are used to move the robot in its environment. Instructions as moving, rotating and moving after rotating are examples of them. For example the command mv $d_i$, $\theta_i$, $t_i$ rotates the robot $\theta_i$ radians, then the robot advances a $d_i$ distance, in decimeters, and in $t_i$ seconds the action is executed.

Status. As in real world, it is necessary that the robot knows where in the world it is located. Instruction as getting and setting position are examples of it.

Sensors. Maybe the most important commands are the sensor related commands, here it is possible to get the values of all the defined sensors that the robot has.

Conditionals. Instructions that control the flow of the program.

User’s functions. An user can include his own subroutines written in C/C++ allowing him to include complex behaviors to the virtual robots. Figure 7 shows the simulation of the robots on the virtual environment. The upper left window shows a close view of the 3D environment; the lower left shows a faraway view of one of the robots; the upper right window shows the point of view of one the robots; and the lower right window shows a console where the commands are typed.

![Figure 7: Simulation of mobile robots.](image)
6.1 Robel program

The following code is an example of Robel programming language. This algorithm was generated using genetic algorithms in order to evade dynamic obstacles [9].

```c
#define MAX_VALUE 5.0
#define MIN_VALUE 4.5
int state
float s1, s2
state <- 0  @1
while 1 < 2  @2
READ:  @3
    shs sonar 2  @4
    s1 <- RegF1  @5
    shs sonar 4  @6
    s2 <- RegF1  @7
    if state = 0  @8
        if s1 < MIN_VALUE  @9
            state <- 1  @10
            goto ETI1  @11
        endif  @12
        if s2 < MIN_VALUE  @13
            state <- 2  @14
            goto ETI1  @15
        endif  @16
    endif  @17
    if state = 1  @18
        if s2 < MIN_VALUE  @19
            state <- 2  @20
            goto ETI1  @21
        endif  @22
        if s1 > MAX_VALUE  @23
            state <- 0  @24
            goto ETI1  @25
        endif  @26
    endif  @27
    if state = 2  @28
        if s1 < MIN_VALUE  @29
            state <- 1  @30
            goto ETI1  @31
        endif  @32
        if s2 > MAX_VALUE  @33
            state <- 0  @34
            goto ETI1  @35
        endif  @36
    endif  @37
ETI1:  @38
    if state = 0  @39
        mv 3 0  @40
        goto READ  @41
    endif  @42
    if state = 1  @43
        mv 0 0.5236  @44
        goto READ  @45
```

7 Conclusions

Using the ViRBot system to control virtual and real mobile robots is an easy and quick way to develop complex behaviors in the robots. The same algorithms tested on the virtual robots have been tested, with slight variations, in two different platforms: a custom robot with a 6811 micro controller, and a real world interface mobile robot B14, with Linux operating system. The advantage of using simulated robots instead of the real ones is that we can test extensively robots’ algorithms in the simulated ones before the final versions are executed in the real robots.

References