Simple Robot Simulator 2010 (SRS10)
Written by Walter O. Krawec

Introduction

Simple Robot Simulator 2010 (SRS10) is a basic robotic simulator which operates over a network allowing multiple clients (potentially on different computers) to connect and run multiple robots (each client may run multiple robots; there is a current maximum of 10 clients). One may program a client in C++ by including the srs10Client.h file (which allows access to the simulator's primary commands). Alternatively, one may write programs in the included wok::Interpreter language. More information on both is included in this document.

The Source Code

The source code is divided into three components:

1) SRS10: the actual simulator; located in SRS10Release/
2) SRS10Module: a dynamic CommandLibrary module for wok::Interpreter; located in SRS10Module/ (Windows only)
3) SRS10Interpreter: a wok::Interpreter build containing the SRS10Module statically linked

Items 1 and 3 should be cross platform with some work. Item 2 is only needed if you don't want to use item 3 but instead with to use a generic wok::Interpreter build. However if you are using Windows, a Visual C++ 2010 solution file is included (you can download the free Visual C++ 2010 Express from Microsoft's website). Also the bin/ folder contains Windows executable for 1 and 2 along with the DLL file for item 2.

Note that, at the moment we've only tested item 1 on Windows; however the client code works on Windows and Mac.

The Simulator

The simulator source code is in the SRS10Release/ folder. Some source code also included in the wok/ and include/ folders. On Windows, the simulator can be run by executing the file bin/srs10.exe. Once run, you should see an arena load. Hold the left mouse button and drag to move the screen; use the mouse wheel to zoom in/out. Other commands include:

1) Press 0-9 then SHIFT-LEFT-CLICK to move robot 0-9 (zero being the first robot created, 9 the tenth; while the simulator can handle many more robots at once, you can only manually relocate the first 10).
2) Right click to create a simulated "clap" event.
3) Press F1 and F12 together to clear all robots.

The simulator, when run, will create a TCP host on port 50010 (you may change this port in SRS10Release/main.cpp in the WinMain() or main() procedure). Whenever a client connects, that client may add multiple robots each of which must be given a unique name. Note however that the name need only be unique to the client. So, there may be two clients each of whom creates a robot named "robot1"; but there cannot be one client who creates two robots named "robot1". Each created robot must also be specified a "type" between 0 and 255. The type specifies which data file to load (see bin/Data/Robot/Robot*.dat where the * is the type; included are 0 and 1 both of which look the same). This type is also used by the RobotProximitySensor described below.

After creating a robot (a client may create multiple robots), next sensors must be added. Supported sensors are:

1) IR: binary proximity sensor
2) IR360: binary like IR but may be rotated to any position or "auto-scan" (i.e. sweep through 180 degrees)
3) LightSensor: binary also; reports true if robot is near a light source
4) RobotProximitySensor: a sensor that looks for other robots and also "food" entities. Value returned from this sensor is: 0x04*(FOUND_FOOD) + 0x02*(FOUND_DIFFERENT_TYPE_ROBOT) + 0x01*(FOUND_SAME_TYPE_ROBOT). This sensor only covers 90 degrees of the robot. Can be either quadrant 0-3 or forward, left, etc.

After this setup stage one may poll the sensors and retrieve their values. Each client maintains a local robot state which is only updated if a poll_sensor command is issued. Also, every robot is equipped with a bump/collision detector which triggers true if the robot physically crashes into a wall or another robot (actually, one may distinguish between the two cases if so desired).

There are also events that may be processed. These include:

1) Picked up food
2) Heard a noise

Events include such information as the current state of the robot (sensors, outputs, position, etc.). One should periodically poll_events and process these every iteration. More information is contained in the next sections.

In the Data/Track/ directory you will see several .txt files specifying different tracks. On startup, SRS10 will load the file Track1.txt. If you wish to use a different track, rename it to Track1.txt. See these text files for more info on creating new tracks.

The SRS10 Interpreter

This is a programming language (with syntax similar to Lisp/Scheme) allowing you to access SRS10 and write robot programs. It is also modular (see SRS10Module for an example) meaning you can write a new DLL CommandLibrary extending the language to support other features (more complicated AI programs for example). When another module is loaded, all other loaded modules (including the built-in SRS10Module) may access each other.

SRS10 Interpreter is built on top of wok::Interpreter (included in the wok/ directory). See Interpreter/readme.pdf for more information, here however we'll include some of the basics. The syntax of this language is of the form COMMAND {ARGUMENTS}^*. In fact, any program is of this form. The "value of" a program COMMAND ARG1 ARG2 … ARG_n is simply the evaluation of COMMAND given the value of each ARG. When first run (see bin/SRS10Interpreter.exe) you should be given a SRS10> prompt. Typing:

+ 1 2

Should return an Integer type equal to "3" (1+2). Alternatively:

begin (new integer i) (set i (-3 2)) (print (get i))

Should return an integer "1". The above program uses a "begin" command which simply instructs the interpreter to run each argument as its own command (notice each argument is enclosed in parentheses). The value of a "begin arg1 arg2 … arg_n" command is simply the value of arg_n (the last argument). The "new" command creates a new variable (integer type in this case and calls it "i"). The set and get commands work as one would expect (here we set i to be the result of the command (- 3 2) which is 3-2 = 1). The "print" command simply prints to the screen whatever it's argument value is (note if we wrote here "(print i)" instead of "(print (get i))" the program would output the string "i" and not the integer value). This is because the print ARG command prints the value of ARG; the value of "i" is the string "i"; the value of (get i) is the actual value stored in variable "i". See Interpreter/readme.pdf for a list of all standard commands (and much more info on wok::Interpreter). Special SRS10 specific commands are listed now:

Command: srs10::connect (IPAddress <STRING>) (port number <INTEGER>)
Description: Will connect to the SRS10 host on given IP address and port number.
Value: 1 if successful, 0 otherwise.

Command: srs10::disconnect
Description: Will disconnect from the SRS10 simulator.
Value: 1 if successful, 0 otherwise.

Command: srs10::add_robot (Robot Name <STRING>) (Robot Type <INTEGER>)
Description: Will add a new robot with given name and type.
Value: 1 if successful, 0 otherwise.

Command: srs10::add_sensor (Robot Name <STRING>) (Sensor Type <STRING>) (Sensor Name <STRING>) [ARGS]
Description: Will add a sensor of type "Sensor Type" to the specified robot. Sensor will be called "Sensor Name". Types supported (case sensitive):

1. IR
    a. ARGS = (range <DOUBLE>) (rotation <DOUBLE>)
2. IR360
    a. ARGS = (range <DOUBLE>) (rotation <DOUBLE>) (auto scan <integer>)
3. PROXIMITY
    a. ARGS = (range <DOUBLE>) (quad <INTEGER>)
4. LIGHT

Value: 1 if successful, 0 otherwise.

Command: srs10::poll_sensors (Robot Name <STRING>)
Description: Will poll all sensors on specified robot (must call this before get_sensor)
Value: 1 if successful, 0 otherwise.

Command: srs10::poll_events (Robot Name <STRING>)
Description: Will poll all events on specified robot (must call this before get_event)
Value: 1 if successful, 0 otherwise.

Command: srs10::create_event (Robot Name <STRING>) (Volume <DOUBLE>) (Creator <INTEGER>) (Reward Value <INTEGER>) (red color <DOUBLE>) (green color <DOUBLE>)(blue color <DOUBLE>)
Description: Will create a new event heard by robots within a radius of Volume. The event will be represented as a circle of the specified color emitting from the specified robot (a "chirp"). Creator and reward value are user specific (these values are saved by robots who hear this event allowing the user to specify different events based on these two values). Though reward value is not currently used or accessible.
Value: 1 if successful, 0 otherwise.

Command: srs10::event::get (event <SRS10::EVENT>) (Field Name <STRING>)
Description: Will return the specified field from the given event. Fields are:

- "x": returns x position of robot in simulator when the event occurred <DOUBLE>
- "y": returns y position of robot when the event occurred <DOUBLE>

- “type”: returns the type of event <INTEGER>
- “creator”: returns the creator of the event <INTEGER>
- “direction”: returns the direction of the event relative to the robot <INTEGER>
- “direction_str”: returns the direction of the event as a string “Forward”, “Right”, “Back”, or “Left” <STRING>
- “lastMoveCommand”: returns the move_command the robot was following when the event occurred.

Value: Depends on Field Name.

Command: srs10::get_next_event (Robot Name <STRING>)
Description: Will return the next event (call srs10::poll_events first!); returned value is a SRS10::Event type; use srs10::event::get to retrieve particular fields from this event. Event is removed from the list after this call.
Value: SRS10::EVENT

Command: srs10::get_sensor (Robot Name <STRING>) (Sensor Name <STRING>)
Description: Will return the value of the specified sensor (call poll_sensors first)
Value: An INTEGER result from the sensor

Command: srs10::move_command (Robot Name <STRING>) (command <INTEGER>)
Description: Sends a move command to the robot. Command may be one of:

- 0 = Stop
- 1 = Forward
- 2 = Backward
- 3 = Left
- 4 = Right

Value: 1 if successful, 0 otherwise.

Examples

Three example programs are included in the bin/scripts folder:

1) Example1.txt a simple obstacle avoiding robot
2) Example2.txt a remote control robot (use ‘w’ ‘a’ ‘s’ and ‘d’ to steer)
3) Example3.txt a robot that avoids walls and other robots while looking for food (green dots)
4) Example4.txt a robot that moves towards a “clap” sound (use the right mouse button in the simulator to create a clap event)

To run Example1.txt for instance, run the simulator SRS10 and also SRS10Interpreter. At the SRS10> prompt type:

file scripts/example1.txt

Type any key (at SRS10Interpreter) to quit the program.

Using C++

Instead of (or in addition to) using wok::Interpreter to program a robot, you may embed the simulator commands into a C++ program and access them directly. To do so, simply include the "include/srs10Client.h" file (which depends on "wok/Sockets.cpp" and "wok/Sockets.h" files; for windows you will also need to link with ws2_32.lib).

In your program, create a new instance of the SRS10::Client class and call the "connectToHost(char* IPAddress, int portNumber)" command to connect to SRS10 (default port number is 50010). Commands are somewhat similar to the wok::Interpreter versions. For instance, call "addRobot(const std::string& name)" to add a robot with the specified name. Call "pollSensors(name)" and then "getSensor(robotName, sensorName)" to retrieve sensor information. See the srs10Client.h file for a list of all commands.