INTEGRATING ROSSUM’S PLAY HOUSE AND FUZZYJ TOOLKIT TO CREATE
AN AUTONOMOUS MOBILE ROBOT SIMULATION

A Project
Presented to the Faculty of
California State University, Chico

Partial Fulfillment
of the Requirements for the Degree
Master of Science
In
Computer Science

by
Chakorn Vithayasai
Spring 2005
ACKNOWLEDGEMENTS

The successful completion of this master’s project is due to the support I received from several people. First, I thank my parents and brother for their encouragement, patience, and support. Second, I would like to express my gratitude toward my project committee members, Dr. Benjoe Juliano and Dr. Reneé Renner, for their guidance and direction upon this project. Finally, I acknowledge my thesis editor/formatter, Josie Smith, who dedicated so much time and effort for the write-up for this endeavor.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>Abstract</td>
<td>vii</td>
</tr>
<tr>
<td><strong>CHAPTER</strong></td>
<td></td>
</tr>
<tr>
<td>I.   Introduction</td>
<td>1</td>
</tr>
<tr>
<td>Testing</td>
<td></td>
</tr>
<tr>
<td>II.  Review of Related Literature</td>
<td>13</td>
</tr>
<tr>
<td>Webots</td>
<td></td>
</tr>
<tr>
<td>TRSoccerbots</td>
<td></td>
</tr>
<tr>
<td>Karel J Robot</td>
<td></td>
</tr>
<tr>
<td>EASY-ROB</td>
<td></td>
</tr>
<tr>
<td>Simulation Studio</td>
<td></td>
</tr>
<tr>
<td>RoboWorks</td>
<td></td>
</tr>
<tr>
<td>III. Design and Implementation</td>
<td>23</td>
</tr>
<tr>
<td>IV.  Summary and Recommendations</td>
<td>46</td>
</tr>
<tr>
<td>References</td>
<td>49</td>
</tr>
</tbody>
</table>
Appendices

A. User Manual .............................................................................................. 51
B. ClnFuzzy Source code ............................................................................... 55
C. ClnPositionEventHandler Source code .................................................. 65
D. RsBodyRangeSensor Source code .......................................................... 65
LIST OF TABLES

Table 1 Comparison Original RP1 with Fuzzy version Table
LIST OF FIGURES

Fig 1 The Webots™ interface
Fig 2 The TRSoccerbot interface
Fig 3 Karel J Robot simulation panel
Fig 4 The EASY-ROB interface
Fig 5 The Simulation Studio interface
Fig 6 The RoboWorks™ interface
Fig 7 RP1 Graphic panel showing the original RP1 environment.
Fig 8 RP1 Control panel
Fig 9 FuzzyValue sample code
Fig 10 FuzzyVariable sample code
Fig 11 FuzzyRule
Fig 12 FuzzyJ toolkit steps diagram
Fig 13 FuzzyRule sample code
Fig 14 Visual Representation of simple triangular FuzzySet
Fig 15 Visual Representation of more complex FuzzySet
Fig 16 ClnFuzzy header
Fig 17 fuzzie method format
Fig 18 Fuzzy control rules used.
Fig 19 Sample output based on 9 rules
Fig 20 Fuzzy decision surface

Fig 21 RP1 Graphic panel with new environment

Fig 22 Behavior of robot with Fuzzy control in original RP1 environment

Fig 23 Samples of robot behavior with Fuzzy control in new environment.
INTEGRATING ROSSUM’S PLAYHOUSE AND FUZZYJ TOOLKIT TO CREATE
AN AUTONOMOUS MOBILE ROBOT SIMULATION

By
Chakorn Vithayasai

Master of Science in Computer Science
California State University, Chico
Summer 2004

Robots are becoming a part of our household. Not long ago, having robots work for us was like a dream. The Roomba™\(^1\)[1] robot was a good example of a robot that uses logic to solve problems. The Roomba™ was the first commercially available autonomous vacuum available in the U.S. The Roomba™ was the first step towards creating an exciting new consumer market category called intelligent home appliances that use robotic technology to deliver significant personal time and energy savings.

This masters project outlines an autonomous mobile robot simulation using fuzzy control based on the Rossum’s Playhouse (RP1)[2] robot simulator. FuzzyJ Toolkit [3] was used to incorporate fuzzy logic into RP1’s control logic.

Fuzzy logic is a technique that facilitates solutions to complicated problems that typically involve a lot of factors. The major role of fuzzy logic in autonomous mobile

\(^1\) Roomba™ is registered trademark of iRobot® Corporation
robots is its capability to combine inputs that do not have to be crisp (or discrete) values and use these values to process the result using fuzzy rules that we can customize as we need to.

The development of techniques for autonomous navigation is preferred rather than the difficulties inherent in hard-coding all possible scenarios. In this case study, we discuss how fuzzy computation techniques have been used in the RP1 to address some of these difficult issues; how to integrate FuzzyJ Toolkit to the existing RP1; and how to control the robot from the result of the integration with FuzzyJ Toolkit.
CHAPTER I

INTRODUCTION

Developing autonomous robot navigation techniques is one of the major trends in the current research on mobile robotics. This trend is based on two motivations. First, current industrial robots lack flexibility and autonomy. Industrial robots typically use pre-programmed sequences and cannot make decisions when faced with unexpected situations. Second, there appears to be a growing market for autonomous robots; this can range from intelligent service robots in offices, factory and hospitals, to maintenance robots operating in high risk environments, or rescue robots operating in disaster field.

One of the main challenges in developing autonomous robot navigation systems is that the information about the robot’s environment is normally uncertain, approximate, incomplete, or noisy. Additionally, most real-world environments are dynamic limiting the utility of prior information. For instance, a piece of furniture in a typical office environment may change in position and figure (folding table); objects may move in unexpected directions. Another factor to consider is that sensors are affected and limited by noise, field of sight, and the inherent difficulty of the sensory interpretation process.
The autonomous mobile robot simulation presented in this manuscript can be used for educational purposes. It shows how fuzzy logic can be integrated with an existing simulation. The user can customize fuzzy rules, fuzzy parameters and the RP1 maze. The user can experience the simulation and behavior of the robot. This project provides a great tool for teaching fuzzy logic. Users can experience fuzzy logic and learn how to make modifications to their fuzzy models.

The scope of this project is limited to the integration of fuzzy logic control to the existing RP1 robot simulator. The major concern is the development of an appropriate range sensor model to be used as input for the FuzzyJ Toolkit. A limitation of this project is the absence of a mapping system that keeps track of the place that the robot has visited so it can backtrack when necessary. This feature is not built-in to the original RP1, but it is possible to integrate this feature as an extension to this project.

Currently, there seems to be no autonomous robot simulator available that integrates fuzzy logic into its control structure. Most robots that have fuzzy control are actual robot implementations. There are a lot of reasons for using RP1 to develop a navigation system for this project. Both RP1 and FuzzyJ Toolkit are implemented in Java™\textsuperscript{2} [4]. RP1 uses a client-server architecture; this give more flexibility to run

\footnote{Java™ is registered trademark of Sun Microsystems, Inc.}
simulations remotely. Since Java™ is portable, RP1 is platform-independent also. RP1 is an open source product distributed under the GNU Public license (GPL)[5].

This project provides a tool that can be used as a teaching application in an Artificial Intelligence class. This will help students to understand how robots can make decisions and students can customize the behavior of the robot. This tool can display the behavior of fuzzy logic controllers graphically and it can help the user understand the characteristics of each fuzzy rule given different inputs.
CHAPTER II

REVIEW OF RELATED LITERATURE

The goal of simulating mobile robots is to eventually operate with real robots, but it is practical to investigate with a simulation robot prior to applying potential techniques to real robots. Simulation can increase performance, is easier to setup, more convenient to use, less expensive and faster. More time is required to build a real robot, but most simulation applications take just a few hours to build and setup. Simulation can dramatically improve the speed of the testing process and the cost of operation. In virtual time we can increase the speed to get a faster result. Most simulation applications also allow changes to the robot configuration. After some experimentation with the simulation and gaining some level of the satisfaction in observed behavior and performance, you can start implementing the real robot. A real robot might be available as a model robot that needs programming and loading of the control program to the model robot. This programmable robot approach is very flexible. An example of this is the K-Team Khepera™3. The K-Team [6] Khepera™ is one of the most popular miniature mobile robots that has all functionality and similarity to large scale robots.

3 Khepera™ is register trademark of K-Team S.A.
The mobile robot simulation applications presented in this chapter support more than one platform. Some of the simulations are commercial applications and some of them are created for specific industrial needs.

**Webots™**

Webots™ is a commercially available Mobile Robot Simulation software from Cyberbotics [7] that provides easy prototyping simulation of mobile robots and environments. Webots™ includes robot libraries that facilitate transfer of control programs to a variety of commercial mobile robots. Mobile robots that come with Webots include Lego® Mindstorms®, K-Team Khepera®, K-Team Koala® and Sony[9] Aibo®.

One can also easily change the configuration (sensors and actuators) of model robots in Webots™. Webots™ can simulate mobile robots simultaneously in a shared environment. Another feature is that the program allows you to use your favorite development environment, simulate the robot and transfer the resulting programs onto your real robots.

---

4 Webots™ is registered trademark of Cyberbotics Ltd.
Webots is available for Windows™⁵, Linux and Mac™ OS X™⁶ platforms. You can program the simulated robots in Webots in C, C++ and Java™, or from other applications through TCP/IP.

---

⁵ Windows™ and Microsoft™ is registered trademarks of Microsoft Corporation.
⁶ Mac™ and OS X™ are registered trademarks of Apple Computer, Inc.
**TR Soccerbots**

TR Soccerbots[10] is a Soccer Robot Simulation Software. TR Soccerbots is written in Visual C#™ .NET™ and it is a free educational program to share ideas on how to create a robot simulator. This simulator uses Teleo Reactive Programs as control logic. Teleo from the Greek *teleo* means “result”. Teleo Reactive Program is an artificial intelligence application that is designed to establish rules of behavior for changing environments. Teleo Reactive Programs was developed by Nils Nilsson and a research team in the AI (Artificial Intelligence) Lab at Stanford University. Nilsson is among one of the people who first developed AI.

![Fig 2 The TRSoccerbot interface](image)

---

Visual C#™ and .NET™ are registered trademarks of Microsoft Corporation.
TRSoccerbots is divided into two components: The first is TREditor, a graphical user interface that allows the user to program the TRSoccerbots simulator by using the mouse and keyboard. The second is TRConfig, which provides an interface for the user to set up soccer matches for the simulator. Each simulated robot in an application process is run parallel, allowing users to see the robot move independently from each other.

TRSoccerbots is only available for the windows platform. Because this simulator is designed specifically for soccer simulation and the developer provides only the application itself, users can not alter the program to create a new robot for applications other than soccer.

**Karel J Robot**

Karel J Robot[11] is an open source 2 dimensional robot simulation. This simulation is developed by Joseph Bergin, Mark Stehlik, Jim Roberts and Richard Pattis. This simulation looks similar to RP1, but this application provides a graphical interface to create the maze, and allows one to save the environment file for future use. Karel J Robot was implemented in Java™ so it has the flexibility to run on any platform.
The Karel J Robot simulation uses a grid system limiting the turning angles to
only 90 degrees left or right.

**EASY-ROB™**

EASY-ROB™[12] is a 3D simulation tool for industrial robotic applications and
educational applications. The program uses OpenGL™ and is a 32-bit Microsoft™[13]
Windows™ application. EASY-ROB™ was implemented in C and C++. The
documentation in for this application claims it is well designed for feasibility studies,
virtual testing, visualization of mechanical system, layout planning, research and
development. EASY-ROB™ also has a virtual robot library available as an option. This

---

8 EASY-ROB is registered trademark of EASY-ROB.
9 OpenGL™ is registered trademark of Microsoft Corporation.
application provides its own interface and programming language to set up simulations.

EASY-ROB™ is a commercial application that available only on Windows™ platform.

![Fig 4 The EASY-ROB™ interface](image)

**eyeWyre™ Simulation Studio**

Simulation Studio[14] is a commercially available 3D simulation that allows interaction with the simulation in real time. This application can simulate some popular real robots like Parallax’s Boe-Bot. Simulation Studio uses an integrated BASIC Stamp 2

---

\[10\] eyeWyre™ is registered trademark of eyeWyre Corporation.
series simulation. This component allows you to program in a syntax colored editor just like some popular programming editors. Simulation Studio provides a real physics simulation environment. Every component in a simulation will interact with the simulation in a realistic way.

Fig 5 The eye Wyre Simulation Studio interface.

Simulation Studio is available for the Windows™ platform only. Because this application has its own interface and programming language, it does not have the ability to use other robot simulations imported from outside data.

RoboWorks™

\[11\] RoboWorks™ is registered trademark of Newtonium.
RoboWorks[15] is a simple and easy to use 3D simulation. The user can create interactive animations via keyboard or import data files from other sources. RoboWorks is based on the OpenGL Graphic engine. The application is small and it is easy to create a graphical model of a robot or mechanical hardware. RoboWorks does not have any control logic that will handle real time simulations, but you can link the control logic from other applications by using RoboTalk™[15]. RoboWorks alone does not provide much when it comes to mobile robot simulation projects. It just provides a graphical model of the robot and allows one to set up movements for each component.

Fig 6 The RoboWorks™ interface.
RoboWorks supports the Windows platform only, but it is compatible with RoboTalk™. RoboTalk is an open source application. It is a programming interface application that allows the user to run programs on any platform through TCP/IP.
CHAPTER III

DESIGN AND IMPLEMENTATION

Rossum's Playhouse (RP1)

RP1 is a Java™-based two-dimensional robot simulator. It is designed to be a tool for developers working on robot navigation and control logic. Specifically, RP1 was designed to simulate a fire fighting robot. RP1 has three modes available to a user. The autopilot mode will randomly choose a target and then follow an optimal path to find it. In the curve mode, the robot will attempt to follow a curved path. In line-path mode, the robot will always follow a straight line path in the general direction of a mouse click. One of the most interesting points of RP1 is the logic control used in autopilot mode to navigate the robot to the target.

RP1 implements client-server architecture. The server provides a virtual landscape where one or more robots may interact with their environment. All robot control functions come from RP1 clients. Clients, which run as separate executables, connect to the server and provide the robot with instructions for moving within the simulated environment. This project uses RP1 version 0.49 which is the last version of RP1 implemented in Java™. According to RP1’s author, G. W. Lucas, RP1 from version 0.50 on will be implemented in C.
Fig 7 RP1 Graphic panel showing the original RP1 environment.

Fig 8 RP1 Control panel
FuzzyJ Toolkit

FuzzyJ Toolkit is a useful application for exploring the ideas of fuzzy logic and fuzzy reasoning in a Java™ setting. The National Research Council [16](NRC) of Canada's Institute for Information Technology FuzzyJ Toolkit can be used to create Java™ programs that encode fuzzy operations and fuzzy reasoning. There is great flexibility with the FuzzyJ Toolkit, in that you can setup fuzzy sets, fuzzy variables and fuzzy values. The process of creating fuzzy rules is summarize in Fig 12.

Fuzzy Value

A FuzzyValue object is used to create a specific fuzzy concept for a given FuzzyVariable. For example, we can represent some concepts like “distance is very far”, assuming that we have a FuzzyVariable for a distance called “far”. We can start to create the FuzzyValue by specifying the distance FuzzyVariable and linguistic expression (in this case “very far”). The expression is phrased and a FuzzySet that mathematically defines the shape of this concept, is created and stored with the FuzzyValue. FuzzyValue is an association of a FuzzyVariable and a linguistic expression to describe a fuzzy concept. FuzzyValue can be created by using a FuzzyVariable and a FuzzySet, but it will have no English-like expression associated with it. In this case FuzzyJ Toolkit will associate the expression “???” instead.
The user can add a linguistic expression string to the FuzzyValue with the setLinguisticExpression method.

Linguistic expressions can be created by combining terms with the operator “and” and “or”. It also can create this by using a set of system or user supplied modifiers such as “very”, “not” and “slightly”. These English-like expressions can describe the required fuzzy concepts for the variable.

```java
static FuzzyVariable left_sensor;
static FuzzyValue Lfar = null;
static FuzzyValue Lmed = null;
static FuzzyValue Lclose = null;
FuzzyValue Lfar = new FuzzyValue(left_sensor, "very far");
FuzzyValue Lmed = new FuzzyValue(left_sensor, "medium");
FuzzyValue Lclose = new FuzzyValue(left_sensor, "slightly close");
```

Fig 9 FuzzyValue sample code

Fuzzy Variable

A FuzzyVariable object is used to create an instance of a fuzzy variable. This variable can specify a name (for example distance, weight, speed and volume), the units of the variable if required (for example meters, lbs, mile per hours and liters), the universe of discourse for the variable (for the example a range from 0.0 to 0.12), and a set of primary fuzzy terms (like far, close), that will be used to represent the specific fuzzy concepts associated with the fuzzy variable.
/**Sample of creating fuzzy variable by FuzzyJ Toolkits*/
FuzzyVariable sensor = new FuzzyVariable("distance", 0, 0.12, "m");
/**Add fuzzy terms*/
double xFar[] = {0.08, 0.12};
double yFar[] = {0, 1};
double xClose[] = {0.0, 0.04};
double yClose[] = {1, 0};
sensor.addTerm("far", xFar, yFar, 2);
sensor.addTerm("close", xClose, yClose, 2);
sensor.addTerm("medium", "not far and not close");

Fig.10 FuzzyVariable sample code

Fuzzy Rules

A FuzzyRule object consists of three sets of FuzzyValues that represent antecedents, conclusions and input values of a fuzzy rule. A rule can be written as:
The antecedents are the premises of the rule. All antecedents that correspond to a rule must be true before the conclusions can be asserted by matching a fuzzy set of inputs with a fuzzy set of antecedents that corresponded. The FuzzyJ Toolkit can determine a set of conclusions by using FuzzyRuleExecutor.

FuzzyRuleExecutor is implemented as Mamdani[17] min inference operator with the Max-Min composition operator. This algorithm is the most common algorithm for fuzzy inferencing.
Mamdani proposed a fuzzy implication rule for fuzzy control in 1977[17]. This is a simplified version of L.A. Zadeh’s implication operator [18]. The Mamdani fuzzy logic operator is given in equation 1.

\[ \phi [\mu_A (x), \mu_B (y)] \equiv \mu_A (x) \land \mu_B (y) \]  

(1)

Example

In order to create a rule that can be executed (fired), we need to create an instance of a FuzzyRule and then add the FuzzyValues that make up the antecedent, the conclusions and the inputs. In order to create the appropriate FuzzyValues for the rule we need to create the FuzzyVariables for the concepts we are using (distance and degree)

Fig 12 FuzzyJ toolkit steps diagram
//Create FuzzyVariables
left_sensor = new FuzzyVariable("distance", 0, 0.12, "m");
left_sensor.addTerm("far", xFar, yFar, 2);
left_sensor.addTerm("close", xClose, yClose, 2);
left_sensor.addTerm("medium", "not far and not close");

angTurn = new FuzzyVariable("degree", -90.0, 90.0 , "deg");
angTurn.addTerm("highleft", xHighLeftTurn, yHighLeftTurn, 2);
angTurn.addTerm("lowleft", xLowLeftTurn, yLowLeftTurn, 3);
angTurn.addTerm("higright", xHighRightTurn, yHighRightTurn, 2);
angTurn.addTerm("lowright", xLowRightTurn, yLowRightTurn, 3);

FuzzyValue antFval_Lfar    = new FuzzyValue(left_sensor , "far");
FuzzyValue antFval_Lclose  = new FuzzyValue(left_sensor , "close");
FuzzyValue antFval_Rfar    = new FuzzyValue(right_sensor, "far");
FuzzyValue antFval_Rclose  = new FuzzyValue(right_sensor, "close");

//left far , right far
rule0.addAntecedent(antFval_Lfar);
rule0.addAntecedent(antFval_Rfar);
rule0.addConclusion(concFval_NoTurn);

// Create FuzzyValue from Crisp Value
input_Left  =  new FuzzyValue(left_sensor, "far");
input_Right =  new FuzzyValue(right_sensor, "far");
rule1.removeAllInputs();
rule1.addInput(input_Left);
rule1.addInput(input_Right);
FuzzyValueVector fvv = rule1.execute();
System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("X", -90 , 90));
FuzzyValue rule1FVresultAt0 = fvv.fuzzyValueAt(0);
System.out.println("AFTER" + rule1FVresultAt0.plotFuzzyValue("G", -90 , 90));

Fig 13 FuzzyRule sample code

Fuzzy Sets

A fuzzy set is a mapping of a set of real numbers onto a membership value in the range from 0 to 1. A fuzzy set can be written in a set of pairs. When $\chi$ is a real number and $u$ is the membership value then we can represent the fuzzy set with $\{ \mu_1/\chi_1, \mu_2/\chi_2, ... \}$.
\( \mu_{x} = \chi \) \}. The real number \( x \) in fuzzy set is sorted by value (\( x_1 \leq x_2 \leq \ldots \leq x_n \)). This relationship is generally represented in graph with line that connect each point together.

For example, a simple triangular fuzzy set and a slightly more complex fuzzy set are shown next.

**Simple Triangular Fuzzy Set**

This FuzzySet given in Fig. 14 is represented by the three points, { 0.0/0.2, 1.0/0.5, 0.0/0.8 }....It is a convex FuzzySet.

![Fig. 14 Visual Representation of simple triangular FuzzySet](image)

**Sample Code**

```java
FuzzySet fSet = new triangleFuzzySet( 0.2, 0.5, 0.8 );
```
Slightly more complex FuzzySet

The FuzzySet given in Fig. 15 is represented by the five points: \{ 0.0/0.0, 1.0/0.2, 0.5/0.5, 1.0/0.8, 0.0/1.0 \}. It is a non-convex FuzzySet.

Fig 15 Visual Representation of more complex FuzzySet

Sample Code

double yValues[] = {0, 1, 0.5, 1, 0};
double xValues[] = {0.0, 0.2, 0.5, 0.8, 1.0};
FuzzySet fSet = new FuzzySet( xValues, yValues, 5 );
FuzzySet in FuzzyJ Toolkit has a hierarchy of subclasses. FuzzySet can be built by this set of subclasses to represent a fairly complete set of common shapes. This gives the user more flexibility to generate further subclasses to meet specific needs.
Integration

The original simulation, RP1, uses hard-coded paths to guide the robot. It simply performs a left first search (Depth First Search). When the target sensor detects the target it just goes straight to the target and stops when it hits the target. After the robot extinguishes the target, the robot turns 180 degrees to get back to the hard-coded path.

From the original demo it is clear that the simulated robot just looks for the target within the sight of the target sensor by following a predefined path. The robot will detect the target if the target falls within the sight of the range sensor. What if the target is not in the line of sight of the robot? The robot will not be able to extinguish the target, since the robot does nothing when its range sensor detects walls within its range.

The original RP1 does not facilitate simulation of an autonomous robot. The design of the RP1 is not easy to make modifications to, especially for the Autopilot mode. The simulation begins by reading wall, target, node and link information from a FloorPlan file. Node and Link are used for making paths. The Autopilot navigation system uses ClnNavNet to generate routes which are stored in an array.

After trying to get the range from the range sensor, we can do something about this data to control the robot by using the FuzzyJ Toolkit. The most difficult part is to find out how the existing robot works. Understanding the simulation allows us to identify the range sensor object. The range sensor object is supposed to return range distance in
every cycle so we can use that measure to make decisions for our robot. The next step is
to add FuzzyJ toolkit code for use in this simulation. The FuzzyJ toolkit folder must be in
the same directory as our simulation. FuzzyJ toolkit consists of 3 folders: META-INF,
java_cup, and nrc. FuzzyJ toolkit can now be accessed by using the import Java™
keyword in the appropriate class.

Integration of the FuzzyJ toolkit with the RP1 simulation was accomplished
through the following steps:

1. Add FuzzyJ Toolkit to RP1

   FuzzyJ Toolkit can be downloaded from
   http://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit2.html

   Installation is really quite simple; just extract the files from the
   compressed file (normally a zip file) that was downloaded. Place the jar file (e.g.
   fuzzyJ15.jar) in an appropriate place and make sure it is included in the
   CLASSPATH for your application.

2. Create Fuzzy Class for this purpose

   a) We start by simplifying the input from two front range sensors only. The
   result will be the degree that the robot is supposed to turn. In this case I
   created a class named CInFuzzy.java in the clientzero package. To
   make FuzzyJ Toolkit available for this class we put the FuzzyJ Toolkits
package in the same folder of our simulation and use the \texttt{import} statement.

```java
package clientzero;
import nrc.fuzzy.*;  //To use FuzzyJ Toolkits
public class ClnFuzzy {
}
```

Fig 16 ClnFuzzy header

b) Create a method called \texttt{fuzzie()} that requires 2 parameters: left distance and right distance, and returns the turn angle (left turn will be a negative value). These 2 parameters are crisp values, so we need to convert these values to \texttt{TriangleFuzzySet} before we can use them.

```java
public double fuzzie(double leftDist,
        double rightDist){
}
```

Fig 17 fuzzie method format

c) Set the variables \texttt{FuzzyValue}, \texttt{FuzzyVariable} and \texttt{FuzzyRule} appropriately.

3. Get the range from \texttt{RsBodyRangeSensor}
a) In the package rossum there is a file named
   \texttt{RsBodyRangeSensor.java}

Write the code that keeps each range sensor status locally in
\texttt{RsBodyRangeSensor}. Check sensor ID in \texttt{computeAndSetState}()
(Sensor ID: Front left, Front right, Rear left, Rear right), get the current range and keep the value in the corresponding parameter. Basically this range sensor checks the sight distance from each range sensor to each wall in the wall array and picks the minimum distance. After it checks all the walls, it will return the closest distance from that range sensor.

b) Create methods that return range. This might not be the best way to get the current range but this is the current implementation in this robot simulation. The problem in keeping the range value locally in \texttt{RsBodyRangeSensor} is that the range might not be the current value at the time the range value is asked for.

4. RPI controls the robot by getting range values to be the input in \texttt{ClnPositionEventHandler} in the clientzero package.

   The existing \texttt{ClnPositionEventHandler} can control the robot by mouse clicks. This is the part that can be modified to use fuzzy control for the robot.
a) Create an instance of \texttt{ClnFuzzy} in the \texttt{ClnPositionEventHandler}. This instance will be used in method \texttt{process()}. 

b) Get range from \texttt{RsBodyRangeSensor}. In this case we use two front range sensors left and right. 

c) Call \texttt{fuzzie()} by using front left and front right distance to get the result turn angle. Result turn angle will be negative if the decision is a left turn, likewise positive for right turn. 

d) Within \texttt{process()} a multiply factor was added to modify the result turn angle that makes the robot turn more than the original turn result. This was done because I noticed that when the simulation starts, the robot makes relatively small turns when detecting obstacles.
Fuzzy Rule in English format

From the concept of FuzzyValue, an antecedent has three possible fuzzy values for the left sensor and three possible fuzzy values for the right sensor resulting in nine possible antecedents. We decide to make nine rules for this simulation.

| Rule 1: Left Far, Right Far; No Turn   |
| Rule 2: Left Far, Right Medium; Low Left Turn |
| Rule 3: Left Far, Right Close; Medium Left Turn |
| Rule 4: Left Medium, Right Far; Low Right Turn  |
| Rule 5: Left Medium, Right Medium; No Turn   |
| Rule 6: Left Medium, Right Close; Low Left Turn |
| Rule 7: Left Close, Right Far; Medium Right Turn |
| Rule 8: Left Close, Right Medium; Low Right Turn |
| Rule 9: Left Close, Right Close; High Right Turn |

Fig 18 Fuzzy control rules used.

Rule 1: IF distance from Left sensor is far and distance from right sensor is far

THEN make no turn.

Rule 2: IF distance from Left sensor is far and distance from right sensor is medium

THEN make low left turn.

Rule 3: IF distance from Left sensor is far and distance from right sensor is close

THEN make medium left turn.

Rule 4: IF distance from Left sensor is medium and distance from right sensor is far

THEN make low right turn.

Rule 5: IF distance from Left sensor is medium and distance from right sensor is medium
THEN make no turn.

Rule 6: IF distance from Left sensor is medium and distance from right sensor is close
    THEN make low left turn.

Rule 7: IF distance from Left sensor is close and distance from right sensor is far
    THEN make medium right turn.

Rule 8: IF distance from Left sensor is close and distance from right sensor is medium
    THEN make low right turn.

Rule 9: IF distance from Left sensor is close and distance from right sensor is close
    THEN make high right turn.

Fig 19 Sample output based on 9 rules when leftsensor = 0.085,
rightsensor = 0.09, then turnangle = 3.41

Fig 20 Fuzzy decision surface

The fuzzy decision surface shown in Fig 20 illustrates what the conclusion will be when the robot gets distance readings from both range sensors. The antecedent for Rule 9 is left close and right close but the conclusion is high right turn. This rule will get executed when the robot faces the wall directly. If the controller decided to make no turn when the distance reading from the right sensor and the left sensor is equal, the robot will hit the wall. To avoid this situation we have to decide to turn one way or the other. In this case, we arbitrarily pick a right turn to get the robot out of the obstacle more efficiently. Because of this decision, the robot will prefer to turn right when it moves directly towards a wall.
Fig 21 RP1 Graphic panel with new environment

Fig 22 Behavior of robot with original RP1 controller in original RP1 environment
Fig 23 Behavior of robot with fuzzy controller in original RP1 environment

The behavior of the fuzzy robot in the original environment from RP1 (see Fig 24) with the fuzzy control is random because of the inconsistency at initial time slot. During this time slot the robot begins to evaluate obstacles at different positions with each movement. In the original environment, the robot successfully reaches the target about 65 percent of the time. However, in the new environment the robot succeeded around 75 percent of the time. The number of iterations in Original RP1 controller counts when the robot moves from one node to another node, while the Fuzzy control counts the total number of executions. In other words, the number of iterations for the Fuzzy control is larger than Original RP1 control due to its real time executions and its smaller
movements, compared to the original RP1 control which uses a predefined path and one iteration of movement that is not limited in length.

Table 1 Comparison of Original RP1 with Fuzzy version Table

<table>
<thead>
<tr>
<th>Controller</th>
<th>Success rate</th>
<th>Average number of iterations for success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original RP1 Control</td>
<td>65%</td>
<td>10</td>
</tr>
<tr>
<td>Fuzzy Control</td>
<td>75%</td>
<td>2300</td>
</tr>
</tbody>
</table>

Fig 24 Samples of robot behavior with Fuzzy control in new environment.

<table>
<thead>
<tr>
<th>Controller</th>
<th>Original RP1 Controller Success rate</th>
<th>Fuzzy Controller Success rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original RP1 Environment</td>
<td>65%</td>
<td>75%</td>
</tr>
<tr>
<td>New</td>
<td>Depends on predefined path</td>
<td>75%</td>
</tr>
</tbody>
</table>
Table 2 Comparison result of Original RP1 and Fuzzy version
CHAPTER IV
SUMMARY AND CONCLUSION

In this project, The FuzzyJ Toolkit was used to successfully embed a simple fuzzy logic control algorithm into the Rossom’s Playhouse (RP1) simulation program. Unfortunately, the robot is not guaranteed to reach the target because we do not have the logic that keeps track of the path that has been visited. This controller depends on the two front distance sensors only. The existing RP1 in Autopilot mode requires a predefined, hard-coded path for the robot. This path played a major role for the robot to get through all the rooms. If the path leads to the wrong spot the robot will not detect the target. The path in this simulation has to be provided before running the simulation. The range sensors in the existing RP1 do nothing at all to be useful in deciding the next move. For a fire-fighter robot, it does not make any sense that the robot just moves to all the rooms that were pre-programmed and does not do anything when its detects obstacles, or even stop when it hits something. This project demonstrates that we can use the robot to make decisions in real-time using fuzzy logic. Every time the robot receives the distance from its left and right sensors, that data will be used to calculate the next movement. As you can see, The robot can turn before it hits the wall. The FuzzyJ Toolkit is a very powerful tool that provides a lot of flexibility to customize fuzzy sets. The FuzzyJ Toolkit allows the user to display input and output fuzzy sets graphically to help us understand the characteristics of each rule.
This project has laid down the groundwork for additional studies in autonomous robot simulation. Future work and/or possible extensions may include the following

1. Adding more sensors by defining appropriate objects. This will be an interesting option that allows a user to customize the location, type and number of sensors on the robot. The addition of this option allows the developer to place a sensor at every location that is supposed to have a sensor and when user runs the simulation, they can enable each sensor individually. When a sensor is disabled, rules that relate to this sensor should not be executed.

2. Adding a mapping/path tracing system. This option will be a big challenge for the developer. The mapping system can be implemented in so many ways. The logic that keeps track of the places that have been visited and making progress in searching the target is the main problem.

3. Creating a GUI that will allow the user to interactively modify the fuzzy control system. The existing RP1 already has a GUI panel from which the user can select the mode to run the simulation. It might not be so hard to add some fuzzy control to the GUI panel.

4. Interactive modification of features of the robot and its environment (maze) including simulation parameters. Creating more mazes to test the simulation is the best way to see how the robot behaves in each environment. The original
RP1 uses a FloorPlan file to specify the maze. If the developer can create mazes interactively by using a GUI, this will reduce the amount of work needed to create a maze.

In conclusion, this project successfully integrates the FuzzyJ Toolkit and RP1 together. The entire feature set that the FuzzyJ Toolkit provides can be used within RP1. This gives the user more flexibility to customize their version of the control logic. The main contribution of this project is to detail and demonstrate how to change the control in RP1 from following a pre-defined path to autonomous navigation control. This fuzzy control can be extended by adding more sensors and more sensor types.
REFERENCES

   http://rossum.sourceforge.net/sim.html
   http://ai.iit.nrc.ca/IR_public/fuzzy/fuzzyJToolkit.html
   http://www.gnu.org
   http://csis.pace.edu/~bergin/KarelJava2ed/karelexperimental.html

   http://www.newtonium.com

   "FuzzyJ Toolkit" Oct 2004;

   (1), pp. 1-13, 1975


   http://www.ecst.csuchico.edu/~juliano/Fuzzy

   Applications. Prentice Hall PTR, Upper Saddle River, NJ.


[22] Institut de Recherches Interdisciplinaires et de Développements en Intelligence
APPENDIX A
User Manual

▪ How to begin running simulation
  1. Install Java™: Java™ comes in two choices. The Java™ Development Kit (JDK) includes a compiler, debugger, and other development tools. The Java Runtime Environment (JRE) includes only the minimal installation you need to run a Java™ application (program). If you don't plan on developing code in Java™, then you will probably be content with the JRE. Certainly the JRE will save you download time.
  2. Unpack RP1 version Fuzzy Logic
  3. Run RP1 Look in the main folder for files called FireFighther.bat double click on these scripts will launch the application.

▪ How to run with RP1 Fuzzy Logic version
  o After you launch RP1 you will see 3 Panels. The first one is a terminal that displays all the status of the robot. The second one is a graphic simulation of the RP1. The last one is a control panel. You can start the RP1 Fuzzy version by right clicking anywhere in the Graphic panel. The Robot will run automatically.

▪ How to turn on/off log file
  o If you look in the main folder you will see a file name FireFighter.ini. Within this file you can set initial parameters for RP1. The parameter that can enable/disable log file is logToFile. When you want to turn on log file set logToFile = true, otherwise set logToFile =false.
# FireFighter.ini

# initializations for RP1 server, FireFighter demo

logFileName=RP1.log
logToFile=true
logToSystemOut=true
logVerbose=false
dlcEnabled=true
dlcName=demozero.DemoMain
dlcSetIO=true
dlcSetLog=true

How to control speed of simulation

- From FireFighter.ini above there is a parameter called simulationSpeed whose default value is 1.0. You can make this simulation run faster by increasing this value.

- How to customize your own maze.
  - In the main folder you will see sub directory name FloorPlans
    - The original file from RP1 name trinity2001.txt: with in this file you can create nodes, links, walls, targets and home. You can specify scale to inches and meters within this file. This original file use inches scale as a default.
    - The modification version of FloorPlans name trinity2001m.txt. This new file was changed scale to meters. This change will make everything in this simulation all in meters scale.

- The following information explains the syntax and format for the FloorPlan file
  - Set scale meters or inches
    units: meters; // (inches/meters)
Set title in Graphic panel
caption:"Fire-Fighting Robot by Fuzzy Logic control";

Create wall
Syntax
wall name { geometry: beginx, beginy, endx, endy
constraint:orthogonal;}
Sample
wall a {geometry: 0.0,0.0,2.4892, 0.0, 0.01905;
constraint:orthogonal;}

Create Target
Syntax
target target name {
label: "target label";
geometry: coordinateX, coordinateY, diameter;
color: target color;
line Width: lineWidth;
}
Sample
target F1 {
label: "F1"; // F is for fire
gleometry: 0.381, 0.9906, 0.1524;
color: red;
lineWidth: 1;
}

Create Home (Robot position)
Syntax
placement home name {
label: "homet label";
geometry: coordinateX, coordinateY, sight angle of the robot ,diameter;
color: home color;
line Width: lineWidth;
}
Sample
placement home {
Create Node
Syntax
node node name { geometry: coordinateX, coordinateY; label: "label name"; }
Sample
node n0 { geometry: 0.3302, 0.2794; label: "n0"; }

Create Link
Syntax
link link name { nodes: begin node, end node; }
Sample
link p0 { nodes: n0, n1; }

How to set Fuzzy rule
All the Fuzzy Logic setup is in clientzero package. If you look in the clientzero package, you will see ClnFuzzy.java.

Setup definition
double xHighLeftTurn[] = {-90.0,-60.0}; //angle of turn
double yHighLeftTurn[] = {1, 0};
angTurn.addTerm("highleft", xHighLeftTurn, yHighLeftTurn, 2);

Setup rule
rule0.addAntecedent(antFval_Lfar);
rule0.addAntecedent(antFval_Rfar);
rule0.addConclusion(concFval_NoTurn);

Antecedent is the premises of the rule that must be true before the conclusions of the rule can be asserted. In this simulation we use 9 rules, each rule has 2 antecedents. When both Antecedents are true, when this rule get executing, the conclusion will be as we setup here.

This portion of the ClnFuzzy will control the character of the robot when the robot get the distance form both front sensor.
package clientzero;
import nrc.fuzzy.*; //To use FuzzyJ Toolkits

public class ClnFuzzy {
   /** Creates new ClnFuzzy */
   public ClnFuzzy() {
   }

   public double fuzzie(double leftDist, double rightDist, boolean printG, boolean printR){
      try{
         //SENSOR TERMS
         left_sensor = new FuzzyVariable("distance", 0, 0.12, "m");
         left_sensor.addTerm("far", xFar, yFar, 2);
         left_sensor.addTerm("close", xClose, yClose, 2);
         left_sensor.addTerm("medium", "not far and not close");

         right_sensor = new FuzzyVariable("distance", 0, 0.12, "m");
         right_sensor.addTerm("far", xFar, yFar, 2);
         right_sensor.addTerm("close", xClose, yClose, 2);
         right_sensor.addTerm("medium", "not far and not close");

         //TURN ANGLE  left(-)  right(+)
         angTurn = new FuzzyVariable("degree", -90.0, 90.0 , "deg");
         //Left
         angTurn.addTerm("highleft",xHighLeftTurn, yHighLeftTurn, 2);
         angTurn.addTerm("lowleft", xLowLeftTurn, yLowLeftTurn, 3);
         angTurn.addTerm("mediumleft", xMediumLeftTurn, yMediumLeftTurn, 4);
         //Right
         angTurn.addTerm("highright",xHighRightTurn,yHighRightTurn, 2);
         angTurn.addTerm("lowright", xLowRightTurn, yLowRightTurn, 3);
         angTurn.addTerm("mediumright", xMediumRightTurn, yMediumRightTurn, 4);

         angTurn.addTerm("center", xCenter, yCenter, 3);
         angTurn.addTerm("empty", xEmpty, yEmpty, 3);

         //SETUP RULES HERE
         FuzzyValue fVresultAt0 = new FuzzyValue(angTurn, "empty");
         FuzzyValue antFval_Lfar       = new FuzzyValue(left_sensor , "far");
         FuzzyValue antFval_Lmed       = new FuzzyValue(left_sensor , "medium");
         FuzzyValue antFval_Lclose     = new FuzzyValue(left_sensor , "close");
         FuzzyValue antFval_Rfar       = new FuzzyValue(right_sensor, "far");
         FuzzyValue antFval_Rmed       = new FuzzyValue(right_sensor, "medium");
         FuzzyValue antFval_Rclose     = new FuzzyValue(right_sensor, "close");
         FuzzyValue concFval_LowRight  = new FuzzyValue(angTurn, "lowright");
         FuzzyValue concFval_MedRight  = new FuzzyValue(angTurn, "mediumright");
         FuzzyValue concFval_HighRight = new FuzzyValue(angTurn, "highright");
         FuzzyValue concFval_LowLeft   = new FuzzyValue(angTurn, "lowleft");
         FuzzyValue concFval_MedLeft   = new FuzzyValue(angTurn, "mediumleft");
         FuzzyValue concFval_HighLeft  = new FuzzyValue(angTurn, "highleft");
         FuzzyValue concFval_NoTurn    = new FuzzyValue(angTurn, "center");
      }
      rule0.removeAllAntecedents();
      rule1.removeAllAntecedents();
   }
rule2.removeAllAntecedents();
rule3.removeAllAntecedents();
rule4.removeAllAntecedents();
rule5.removeAllAntecedents();
rule6.removeAllAntecedents();
rule7.removeAllAntecedents();
rule8.removeAllAntecedents();
rule0.removeAllConclusions();
rule1.removeAllConclusions();
rule2.removeAllConclusions();
rule3.removeAllConclusions();
rule4.removeAllConclusions();
rule5.removeAllConclusions();
rule6.removeAllConclusions();
rule7.removeAllConclusions();
rule8.removeAllConclusions();
//left far , right far
rule0.addAntecedent(antFval_Lfar);
rule0.addAntecedent(antFval_Rfar);
rule0.addConclusion(concFval_NoTurn);
//left far , right medium
rule1.addAntecedent(antFval_Lfar);
rule1.addAntecedent(antFval_Rmed);
rule1.addConclusion(concFval_LowLeft);
//left far , right close
rule2.addAntecedent(antFval_Lfar);
rule2.addAntecedent(antFval_Rclose);
rule2.addConclusion(concFval_MedLeft);
//left medium , right far
rule3.addAntecedent(antFval_Lmed);
rule3.addAntecedent(antFval_Rfar);
rule3.addConclusion(concFval_LowRight);
//left medium , right medium
rule4.addAntecedent(antFval_Lmed);
rule4.addAntecedent(antFval_Rmed);
rule4.addConclusion(concFval_NoTurn);
//left medium , right close
rule5.addAntecedent(antFval_Lmed);
rule5.addAntecedent(antFval_Rclose);
rule5.addConclusion(concFval_LowLeft);
//left close , right far
rule6.addAntecedent(antFval_Lclose);
rule6.addAntecedent(antFval_Rfar);
rule6.addConclusion(concFval_MedRight);
//left close , right medium
rule7.addAntecedent(antFval_Lclose);
rule7.addAntecedent(antFval_Rmed);
rule7.addConclusion(concFval_LowRight);
//left close , right close
rule8.addAntecedent(antFval_Lclose);
rule8.addAntecedent(antFval_Rclose);
rule8.addConclusion(concFval_NoTurn);

/// OVERWRITE CHECKING FOR RANGE when hit sensor or out of range
/// Becase of the way we setup defuzzification from crisp value then we can not reach the limit
if (leftDist>=0.12)
    leftDist = 0.12 - CRISP_DEGREE;
if (leftDist<=0)
    leftDist = CRISP_DEGREE;
if (rightDist>=0.12)
    rightDist = 0.12 - CRISP_DEGREE;
if (rightDist<=0)
    rightDist = CRISP_DEGREE;

if(printR==true)
    System.out.println("Left dist = " + leftDist + "    Right dist = " + rightDist);

// Create FuzzyValue from Crisp Value
input_Left = new FuzzyValue(left_sensor, new TriangleFuzzySet(leftDist-CRISP_DEGREE, leftDist, leftDist+CRISP_DEGREE));
input_Right = new FuzzyValue(right_sensor, new TriangleFuzzySet(rightDist-CRISP_DEGREE, rightDist, rightDist+CRISP_DEGREE));

if (printG == true)
    System.out.println(fVresultAt0.plotFuzzyValue("R", -90, 90));
count = 0;
if (antFval_Lfar.fuzzyMatch(input_Left) 
    && antFval_Rfar.fuzzyMatch(input_Right) )
    count++;
if (printR == true)
    System.out.println("rule 1 applied");
rule0.removeAllInputs();
rule0.addInput(input_Left);
rule0.addInput(input_Right);
FuzzyValueVector fvv = rule0.execute();

if (printG == true)
    System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("X", -90, 90));
FuzzyValue rule1FVresultAt0 = fvv.fuzzyValueAt(0);
fVresultAt0 = fVresultAt0.fuzzyUnion(rule1FVresultAt0);
if (printG == true)
    System.out.println(" AFTER " + fVresultAt0.plotFuzzyValue("G", -90, 90));
else{
if (printR == true)
    System.out.println("rule 1 not applied");
if (antFval_Lfar.fuzzyMatch(input_Left) 
    && antFval_Rmed.fuzzyMatch(input_Right) )
    count++;
if (printR == true)
    System.out.println("rule 2 applied");
rule1.removeAllInputs();
rule1.addInput(input_Left);
rule1.addInput(input_Right);
FuzzyValueVector fvv = rule1.execute();

if (printG == true)
    System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("X", -90, 90));
FuzzyValue rule2FVresultAt0 = fvv.fuzzyValueAt(0);
fVresultAt0 = fVresultAt0.fuzzyUnion(rule2FVresultAt0);
if (printG == true)
    System.out.println(" AFTER " + fVresultAt0.plotFuzzyValue("G", -90, 90));
else{
if (printR == true)
    System.out.println("rule 2 not applied");
}
if (antFval_Lfar.fuzzyMatch(input_Left) && antFval_Rclose.fuzzyMatch(input_Right))
{
    count++;
    if (printR == true)
        System.out.println("rule 3 applied");
    rule2.removeAllInputs();
    rule2.addInput(input_Left);
    rule2.addInput(input_Right);
    FuzzyValueVector fvv = rule2.execute();
    if (printG == true)
        System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
    if (printG == true)
        System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("G", -90, 90));
    else{
        System.out.println("AFTER " + fVresultAt0.plotFuzzyValue("G", -90, 90));
    }else{
        System.out.println("rule 3 not applied");}
if (antFval_Lmed.fuzzyMatch(input_Left) && antFval_Rfar.fuzzyMatch(input_Right))
{
    count++;
    if (printR == true)
        System.out.println("rule 4 applied");
    rule3.removeAllInputs();
    rule3.addInput(input_Left);
    rule3.addInput(input_Right);
    FuzzyValueVector fvv = rule3.execute();
    if (printG == true)
        System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
    if (printG == true)
        System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("G", -90, 90));
    else{
        System.out.println("AFTER " + fVresultAt0.plotFuzzyValue("G", -90, 90));
    }else{
        System.out.println("rule 4 not applied");}
if (antFval_Lmed.fuzzyMatch(input_Left) && antFval_Rmed.fuzzyMatch(input_Right))
{
    count++;
    if (printR == true)
        System.out.println("rule 5 applied");
    rule4.removeAllInputs();
    rule4.addInput(input_Left);
    rule4.addInput(input_Right);
    FuzzyValueVector fvv = rule4.execute();
    if (printG == true)
        System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
    if (printG == true)
        System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("G", -90, 90));
    else{
        System.out.println("AFTER " + fVresultAt0.plotFuzzyValue("G", -90, 90));
    }else{
        System.out.println("rule 5 not applied");}
if (antFval_Lmed.fuzzyMatch(input_Left) && antFval_Rclose.fuzzyMatch(input_Right))
{
count++;
if (printR == true)
    System.out.println("rule 6 applied");
rule6.removeAllInputs();
rule6.addInput(input_Left);
rule6.addInput(input_Right);
FuzzyValueVector fvv = rule6.execute();
if (printG == true)
    System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
FuzzyValue rule6FVresultAt0 = fvv.fuzzyValueAt(0);
FVresultAt0 = FVresultAt0.fuzzyUnion(rule6FVresultAt0);
if (printG == true)
    System.out.println(" AFTER " + FVresultAt0.plotFuzzyValue("G", -90, 90));
else{
    if (printR == true)
        System.out.println("rule 6 not applied");
}
if (antFval_Lclose.fuzzyMatch(input_Left) && antFval_Rfar.fuzzyMatch(input_Right))
    count++;
if (printR == true)
    System.out.println("rule 7 applied");
rule6.removeAllInputs();
rule6.addInput(input_Left);
rule6.addInput(input_Right);
FuzzyValueVector fvv = rule6.execute();
if (printG == true)
    System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
FuzzyValue rule7FVresultAt0 = fvv.fuzzyValueAt(0);
FVresultAt0 = FVresultAt0.fuzzyUnion(rule7FVresultAt0);
if (printG == true)
    System.out.println(" AFTER " + FVresultAt0.plotFuzzyValue("G", -90, 90));
else{
    if (printR == true)
        System.out.println("rule 7 not applied");
}
if (antFval_Lclose.fuzzyMatch(input_Left) && antFval_Rmed.fuzzyMatch(input_Right))
    count++;
if (printR == true)
    System.out.println("rule 8 applied");
rule7.removeAllInputs();
rule7.addInput(input_Left);
rule7.addInput(input_Right);
FuzzyValueVector fvv = rule7.execute();
if (printG == true)
    System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
FuzzyValue rule8FVresultAt0 = fvv.fuzzyValueAt(0);
FVresultAt0 = FVresultAt0.fuzzyUnion(rule8FVresultAt0);
if (printG == true)
    System.out.println(" AFTER " + FVresultAt0.plotFuzzyValue("G", -90, 90));
else{
    if (printR == true)
        System.out.println("rule 8 not applied");
}
if (antFval_Lclose.fuzzyMatch(input_Left) && antFval_Rclose.fuzzyMatch(input_Right))
    count++;
if (printR == true)
    System.out.println("rule 9 applied");
rule8.removeAllInputs();
rule8.addInput(input_Left);
rule8.addInput(input_Right);
FuzzyValueVector fvv = rule8.execute();
if (printG == true)
    System.out.println(fvv.fuzzyValueAt(0).plotFuzzyValue("*", -90, 90));
FuzzyValue rule9FVresultAt0 = fvv.fuzzyValueAt(0);
fvresultAt0 = fvresultAt0.fuzzyUnion(rule9FVresultAt0);
if (printG == true)
    System.out.println(" AFTER " + fvresultAt0.plotFuzzyValue("G", -90, 90));
} else {
    if (printR == true)
        System.out.println("rule 9 not applied");
}

    // EVALUATE RESULT
    ///////////////////////////////////////////////////////
    turnAngle = fvresultAt0.momentDefuzzify();
    if(printR ==true){
        System.out.println(" Number of rules that applied :::: " + count);
        System.out.println(" Left dist = " + leftDist + " Right dist = "+ rightDist);
        System.out.println(" Final turn angle result :::: " + turnAngle);
    }
    } //end try
    catch (FuzzyException fe){
        System.out.println(fe);
    } //end catch

    ///////////////////////////////////////////////////////////////////////////////////////////////////////////
    // END SETTING UP FUZZY SET HERE
    ///////////////////////////////////////////////////////////////////////////////////////////////////////////
    if (Math.abs(turnAngle) > 0.0001)
        return turnAngle;
    else
        return 0;
}

    ///////////////////////////////////////////////////////////////////////////////////////////////////////////
    // START FUZZY DECLARATION
    ///////////////////////////////////////////////////////////////////////////////////////////////////////////
    static final double MAX_HIGH_TURN_ANG = 90;
    static final double MIN_HIGH_TURN_ANG = 70;
    static final double MAX_LOW_TURN_ANG = 30;
    static final double MIN_LOW_TURN_ANG = 0;
    static final double CRISP_DEGREE = 0.005;
    static final double CLOSE_MIN = 0.0;
    static final double CLOSE_MAX = 0.04;
    static final double FAR_MIN = 0.08;
    static final double FAR_MAX = 0.12;
    double xFar[] = {FAR_MIN, FAR_MAX};
    double yFar[] = {0, 1};
    double xClose[] = {CLOSE_MIN,CLOSE_MAX};
    double yClose[] = {1, 0};
    double xHighLeftTurn[] = {-90.0,-60.0};   //angle of turn
    double yHighLeftTurn[] = {1, 0};
    double xLowLeftTurn[] = {-30.0, -10.0 , 0};
    double yLowLeftTurn[] = {0, 1, 0};
    double xMediumLeftTurn[] = {-70, -60, -30, -20};
    double yMediumLeftTurn[] = {0, 1, 1, 0};
double xHighRightTurn[]  = {60.0, 90.0};   //angle of turn
double yHighRightTurn[]  = {0 , 1 };  
double xLowRightTurn[]   = {0 , 10.0, 30.0};
double yLowRightTurn[]   = {0, 1, 0};
double xMediumRightTurn[] = {20, 30, 60, 70};
double yMediumRightTurn[] = {0, 1, 1, 0};
double xCenter[]         = {-10.0 , 0, 10.0};
double yCenter[]         = {0, 1, 0};
double xEmpty[]          = {0.0,0.0,0.0};
double yEmpty[]          = {0,0,0};
static double    leftDist  = 0.0;
static double    rightDist = 0.0;
static double    turnAngle = 0.0;
static double    count     = 0.0;
static FuzzyValue    fVresultAt0         = null;
static FuzzyValue antFval_Lfar        = null;
static FuzzyValue antFval_Lmed        = null;
static FuzzyValue antFval_Lclose      = null;
static FuzzyValue antFval_Rfar        = null;
static FuzzyValue antFval_Rmed        = null;
static FuzzyValue antFval_Rclose      = null;
static FuzzyValue concFval_HighLeft   = null;
static FuzzyValue concFval_MedLeft    = null;
static FuzzyValue concFval_LowLeft    = null;
static FuzzyValue concFval_HighRight  = null;
static FuzzyValue concFval_MedRight   = null;
static FuzzyValue concFval_LowRight   = null;
static FuzzyValue concFval_NoTurn     = null;
static FuzzyValue input_Left          = null;
static FuzzyVariable left_sensor;
static FuzzyVariable right_sensor;
static FuzzyVariable angTurn;  
static FuzzyRule  rule0   = new FuzzyRule();
static FuzzyRule rule1   = new FuzzyRule();
static FuzzyRule rule2   = new FuzzyRule();
static FuzzyRule rule3   = new FuzzyRule();
static FuzzyRule rule4   = new FuzzyRule();
static FuzzyRule rule5   = new FuzzyRule();
static FuzzyRule rule6   = new FuzzyRule();
static FuzzyRule rule7   = new FuzzyRule();
static FuzzyRule rule8   = new FuzzyRule();
package clientzero;
import rossum.*;
import java.lang.Math;

public class ClnPositionEventHandler implements RsPositionEventHandler {
    private ClnMain client;
    private ClnFuzzy clientFuzzy;
    public double finalOrientation;

    public ClnPositionEventHandler(ClnMain client) {
        this.client = client;
        clientFuzzy = new ClnFuzzy();
        finalOrientation = 0;
    }

    public void processTransaction(RsTransaction t) {
        process((RsPositionEvent)t);
    }

    public void process(RsPositionEvent event) {
        if (!client.autoPilot) {
            if (client.latestMouseClick == null)
                return;
            client.body.setMotion(
                    new RsMotionNull(
                            event.simTime,
                            event.orientation,
                            event.x,
                            event.y));
            double dx, dy, x, y;
            double sinTheta, cosTheta;
            dx = client.latestMouseClick.x - event.x;
            dy = client.latestMouseClick.y - event.y;
            cosTheta = Math.cos(event.orientation);
            sinTheta = Math.sin(event.orientation);

            RsMotionRequest r;
            int turnFactor = 2;
            double unitMovement = 0.003;
            double tNumber;
            double leftDist = ((RsBodyRangeSensor)client.body.getPartByID(4)).getFrontLeft();
            double rightDist = ((RsBodyRangeSensor)client.body.getPartByID(7)).getFrontRight();
            System.out.println(" Range name " +
                    client.body.getPartByID(4).getName() + " range (m) " + leftDist);
            System.out.println(" Range name " +
                    client.body.getPartByID(7).getName() + " range (m) " + rightDist);
    }
tNumber = clientFuzzy.fuzzie(leftDist, rightDist, false, false);
System.out.println(" Result Fuzzy tNumber (degree) " + tNumber );
// convert tnumber to PI --> (tnumber*PI/180 )
finalOrientation = event.orientation - (turnFactor*tNumber*Math.PI/180 ) ;
System.out.println(" Next orientation (radius) " + finalOrientation );
System.out.println(" Current orientation (radius) " + event.orientation );
System.out.println(" Current position x " + event.x + " y " + event.y );
System.out.println(" Search index " + client.searchIndex );

dx = Math.cos(finalOrientation)*unitMovement;
dy = Math.sin(finalOrientation)*unitMovement;
x = dx*cosTheta + dy*sinTheta;
y = dy*cosTheta - dx*sinTheta;

if(client.latestMouseClick.button==1){
  //turn
  r = client.body.wheelSystem.getMotionRequestForPivot(false, x, y, 0.2);
  if(r!=null)
    client.sendMotionRequest(r);
  //go
  r = client.body.wheelSystem.getMotionRequest(false, x, 0, 0.2);
  if(r!=null)
    client.sendMotionRequest(r);
}
else{// autoPilot is true
if(client.searchIndex==ClnMain.COMPUTE_APPROACH){
  ClnMove [] move = new ClnMove[2];
  move[0] = new ClnMove();
  move[1] = new ClnMove();
  client.motionPath = new ClnPath();
  int n = client.navNet.calcMove(
    event.x, event.y, event.orientation,
    client.xTarget, client.yTarget, move);
  if(n==2)
    client.motionPath.add(move[0]);
  if(n>0){
    ClnMove m = move[n-1];
    // by the contest rules, we need to approach to within
    // 12 inches (0.3 meters) of target..
    if(m.sLinear>0.3){
      m.sLinear-=0.3;
      client.motionPath.add(m);
    }
  }
  client.log(" Approach initiated with
    nMove="+client.motionPath.nMove);
  if(client.motionPath.nMove>0){
    client.searchIndex=ClnMain.APPROACH_TARGET;
    client.motionPathIndex=0;
    client.advanceMotionPath(event.x, event.y, event.orientation);
  }else{
    // we are already in position to extinguish the target,
    // there is no need for further motion.
    client.searchRelatedTimeoutIndex = client.sendTimeoutRequest
     (3000);
    client.searchIndex=ClnMain.EXTINGUISH_TARGET;
} else if (client.searchIndex==ClnMain.COMPUTE_TRIP_HOME) {
    RsNavNode startNode = client.navNet.getNearestCheckpoint(event.x, event.y);
    client.motionPath = client.homePath;
    client.navNet.findPath(
        event.x, event.y, event.orientation,
        startNode, client.navNet.getHomeNode(),
        client.linearSpeed, client.angularSpeed,
        client.motionPath);
    client.log("Trip home computed with
        nMove="+client.motionPath.nMove);
    client.searchIndex = ClnMain.RETURN_HOME;
    client.motionPathIndex = 0;
    System.out.println(" advanceMotionPath get called 2");
    client.advanceMotionPath(event.x, event.y, event.orientation);}}
}
package rossum;
import java.awt.Color;
import java.awt.Graphics;
import java.lang.Math;
import java.util.*;
import nrc.fuzzy.*; //To use FuzzyJ Toolkits

public class RsBodyRangeSensor extends RsBodySensor {

    public RsBodyRangeSensor(
            double []point,
            int nPoint,
            double _xDetector, double _yDetector,
            double _sightAngle,
            double _maxRange,
            int    _nRangeBin
    ){
        super(point, nPoint);
        name = "Unnamed Range Sensor";
        hot = false;
        hotFillColor = Color.orange;
        hotLineColor = Color.red;
        if(_maxRange>1.0e+6)
            maxRange=1.0e+6;
        else
            maxRange = _maxRange;
        if(_nRangeBin<1)
            nRangeBin=1;
        else
            nRangeBin = _nRangeBin;
        xDetector = _xDetector;
        yDetector = _yDetector;
        sightAngle = _sightAngle;
        segSect = new RsSegSect();
        segment = new RsSegment();
        rBin=-1;  // an impossible value
        counter = 0;
    }

    public boolean computeAndSetState(long simTime, RsPlan plan, RsTransform transform){
        RsObject [] objectArray;
        RsObject rsObject;
        RsWall wall;
        int iObject;
        boolean oldState;
        int oldrBin;
        double vx, vy, minT;
        oldState = hot;
        oldrBin = rBin;
        timeStateComputed = simTime;
        stateChange=false;
        hot = false;
        
    }
range = 0;
rBin = 0;
double tAngle = transform.getTheta()+sightAngle;
mappedPos = transform.map(xDetector, yDetector);
vx = Math.cos(tAngle);
vy = Math.sin(tAngle);
objectArray = plan.getObjectArray();
if(objectArray==null){
    hot = false;
    stateChange=oldState; // if oldState was true there was a change
    return stateChange;
}
segment.x = mappedPos.x;
segment.y = mappedPos.y;
segment.v.x = vx*maxRange;
segment.v.y = vy*maxRange;
segment.m = maxRange;
minT = 2.0;  // max possible value should be one.
for(iObject=0; iObject<objectArray.length; iObject++){
    rsObject = (RsObject)objectArray[iObject];
    //System.out.println(" object name "+rsObject.getName());
    if(rsObject instanceof RsWall){
        wall = (RsWall)rsObject;
        if(segSect.process(segment, wall.segmentArray[0])){
            if(segSect.t1<minT){
                minT=segSect.t1;
            }
        }
        if(segSect.process(segment, wall.segmentArray[1])){
            if(segSect.t1<minT){
                minT=segSect.t1;
            }
        }
        if(segSect.process(segment, wall.segmentArray[2])){
            if(segSect.t1<minT){
                minT=segSect.t1;
            }
        }
        if(segSect.process(segment, wall.segmentArray[3])){
            if(segSect.t1<minT){
                minT=segSect.t1;
            }
        }
    }
} /* loop check 4 times for 4 sensors */
if (counter < 3)
    counter++;
else
    counter = 0;
if(minT<1.0){
    /* a detection is within range */
    hot = true;
    range=minT*maxRange;
rBin=(int)Math.floor(nRangeBin*range/maxRange);
if(rBin>=nRangeBin)
rBin=nRangeBin-1;
if (counter == 0) //sensor 3
    frontRight = range;
else if (counter ==1)
    frontLeft = range;
else if (counter ==2)
    rearLeft = range;
else if (counter ==3)
    rearRight = range;
else
    System.out.println(" Error in RsBodyRangeSensor ");
}
else{
    range = 0.1145;
    if (counter == 0) //sensor 3
        frontRight = range;
    else if (counter ==1)
        frontLeft = range;
    else if (counter ==2)
        rearLeft = range;
    else if (counter ==3)
        rearRight = range;
    else
        System.out.println(" Error in RsBodyRangeSensor ");
}

double test1=0.0;
double test2=0.0;
if (hot){
    stateChange = (!oldState || oldBin!=rBin);
}else{
    stateChange = oldState;
}

lastestResultSimtime = simTime;
return stateChange;
}
public double currentResult(){
    return resultTest;
}
public long getResultSimtime(){
    return lastestResultSimtime;
}
public double getRange(){
    return range;
}
public double getFrontLeft(){
    return frontLeft;
}
public double getFrontRight(){
    return frontRight;
}
public RsSensorEvent getSensorEvent(long _simTime){
    return new RsRangeSensorEvent(_simTime,
        getID(),
        mappedPos.x,
public boolean isRangeSensor() {
    return true;
}

// elements which describe the sensor
protected double xDetector;
protected double yDetector;
protected double sightAngle;
protected double maxRange;
protected int nRangeBin;

// objects used as "scratch space" for performing various
// calculations (established in the constructor)
private RsSegSect segSect;
private RsSegment segment;

// elements which describe the current state or detection
// note that boolean "hot" is defined in a super class
// set by computeAndSetState() method.
protected double vx, vy;
static double range;
protected int rBin;
static double frontLeft;
static double frontRight;
static double rearLeft;
static double rearRight;
public static int counter;

static int testVal = 0;
static double leftDist = 0.0;
static double rightDist = 0.0;
static double turnAngle = 0.0;
static long lastestResultSimtime = 0;
double[] turnAngleResult = new double[RULES_SIZE];
static double count = 0;
static double sum = 0;
static double avr = 0;
double myTest = 0.0;
double resultTest = 0.0;
static FuzzyVariable outTemp;