

AUTOMATED CONTROL FOR LUNAR LANDER GAME USING FUZZY LOGIC

Siddharth Pandey, Pranshu Ranjan Singh, Eduard Anthony Chai
Nyon Yan Zheng, Tan Kok Keng

Institute of Systems Science, National University of Singapore

ABSTRACT

Lunar Lander was a single-player arcade game in the Lunar Lander subgenre developed by Atari, Inc. and released in August 1979. In the game, the player controls a spaceship as viewed from the side and attempts to land safely on the Moon. This project implements a fuzzy automated controller for a modified version of the Lunar Lander game. The fuzzy logic approach is used to build an automated player for two version of Lunar Lander, basic version and obstacle version. The inputs to the system are distance and speed variables such as distance between spaceship and landing platform, spaceship and obstacle, and speed of the spaceship and the output is the controls of the spaceship, that is, firing of the four thrusters attached to top, bottom, left and right of the spaceship. This firing of thrusters controls the position of the spaceship and navigates it to the landing platform by avoiding obstacles and collision with boundary walls.

1. INTRODUCTION

Lunar Lander is the name of a genre of video games in which the player controls a spaceship as it falls towards the surface of the Moon or other astronomical bodies, and must maneuver the ship's thrusters so as to land safely. In many games in the genre, the available fuel in spaceship is limited or the player must adjust the ship's orientation, as well as its horizontal and vertical velocities. [1] The aim of this project is to develop an automated control for a modified version of the Lunar Lander game.

For this project, two versions of Lunar Lander game are considered; basic version and obstacle version. In basic version of the game, the player needs to adjust the vertical and horizontal velocities in order to land the spaceship smoothly on a platform at some given location. In obstacle version, the basic version of the game is extended to add some obstacles and the player needs to ensure that the spaceship doesn't collide with any of the obstacles.

The Lunar Lander game code used for this project is a Java Version obtained from GitHub. [2] The code obtained implements the basic version of the game without the boundary wall collisions constraints. We implement

collision constraint for spaceship to crash when it collides with boundary walls. For the obstacle version of game, we implement the code for obstacles initialization, spaceship collision with obstacle and obtaining the coordinates of the nearest obstacle.

The universe of game is defined in a window size of dimension 1280 x 720 which acts as a boundary of walls. The coordinates system for the game follows the Java 2D API x-y Coordinate System. The x-coordinate increases to the right, and the y coordinate increases downward. The top-left corner of a window has the coordinates 0,0. All coordinates are specified using integers. [3] The game consists of a single spaceship of dimensions 64 x 64, single landing platform of dimensions 135 x 33 and multiple obstacles in form of square boxes of dimensions 32 x 32. [as shown in Fig. 1] The spaceship can have three states; normal, landed and crashed. The game begins with a normal state of spaceship. If the spaceship collides with boundary walls (including ground) or obstacles then the state of spaceship changes to crashed and the game is over. The spaceship crashes even if the landing on platform is not smooth, that is, if the y-direction speed of the spaceship during landing is greater than 5 units then the spaceship crashes. Both landing platform and obstacles have a single active state throughout the execution of the game.

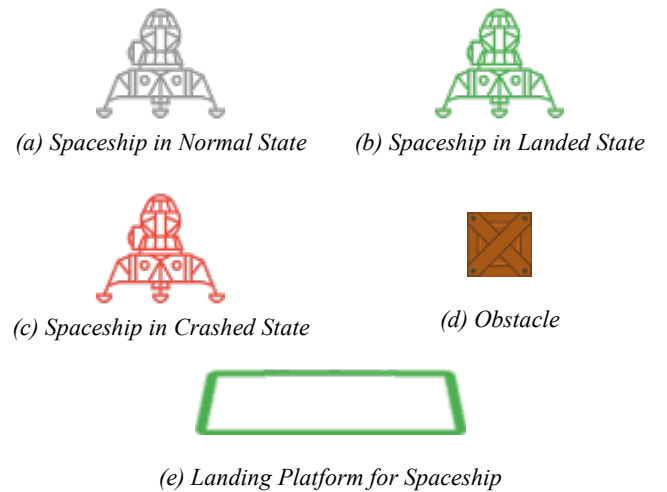


Fig. 1 Components of Lunar Lander Game [(a) – (e)]

The game begins with a fixed position for landing platform, anywhere on the ground and fixed positions for obstacles (in obstacle version). The spaceship initializes to any position within the window size of the game. The spaceship has an initial speed of 1 unit in downward direction at the start of the game. The player can control the position of spaceship by controlling the four thrusters attached to top, bottom, left and right side of spaceship. When the top thruster fires, it increases the acceleration in downward direction by 1 unit. Similarly, the bottom, left and right thrusters increase the acceleration of spaceship in upward, right and left direction respectively. The change in acceleration in specific direction in turn changes the speed in that direction. For example, the initial speed in x-direction is 1 unit and spaceship position is 400 units and left thruster is fired, the speed increases to 2 units and position is changed to 402. But if the left thruster is fired again it the previous value of speed and then adds it to the position of spaceship which becomes 3 units and position becomes 405 and so on. The same formulation is followed for the speed in y-direction along with the presence of a small gravity component (of 0.16 unit) in downward direction throughout the universe of the game. This gravity component ensures that even if player is not controlling the thrusters of the spaceship, the spaceship moves in downward direction with this constant gravity acceleration. The goal of the game is to navigate the spaceship to the landing platform smoothly by avoiding any collision with the boundary walls, ground and any obstacles.

We implement a configuration menu to initialize values to various configurable variables before the start of the game. The configurable variable includes, the coordinates of the initial rocket position, x-coordinate of landing platform (as y-coordinate for landing platform is fixed to the ground), the coordinates of the obstacles and the fuzzy rules to automate the game control. The next two sections explain the fuzzy logic approach used to automate the controls of the Lunar Lander game for basic version and obstacle version respectively.

2. APPROACH FOR BASIC VERSION OF LUNAR LANDER

Fuzzy Logic is used to implement the automated control for the Lunar Lander game. Since, the basic code obtained for the game was written in Java, the Java implementation of FuzzyLite library (jfuzzylite 6.0) was used. jfuzzylite is a free and open-source fuzzy logic control library programmed in Java for multiple platforms (e.g., Windows, Linux, Mac, Android). [4] The fuzzy controller takes independent decision for each iteration of game loop based on the input variables provided to the controller.

2.1. Input / Output Variables and Membership functions

For the basic version of the game, the goal is to land the spaceship at the landing platform smoothly by avoiding any collision with the boundary walls. Takagi-Sugeno Fuzzy inference model is chosen to model the controls of the spaceship according to varying input parameters. [5] The input and output variables are described in Table 1 along with their membership functions.

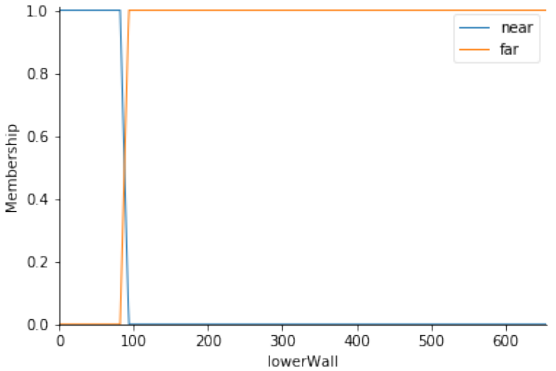
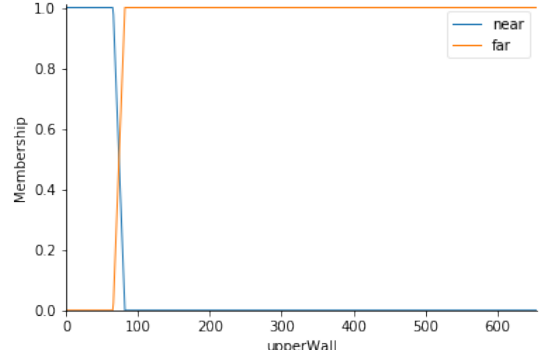
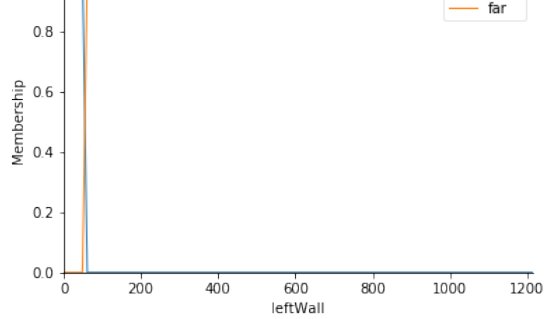
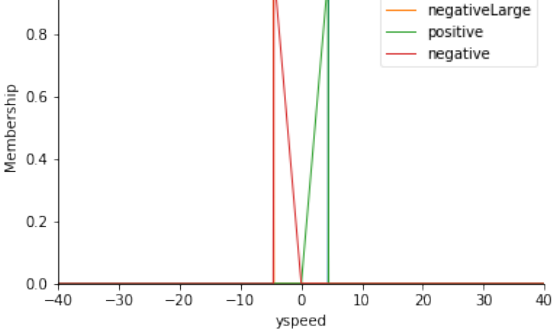
The input variables activated during the basic version of Fuzzy Lunar lander are lowerWall, upperWall, leftWall, rightWall, yspeed, xspeed, xlandingPlatform, and landingAllowed. The lowerWall, upperWall, leftWall and rightWall variables measure distance from respective wall and the spaceship. The xspeed and yspeed variable measure the speed of spaceship in x-direction and y-direction respectively. The xlandingPlatform and landingAllowed variables control the landing of the spaceship. The output variables activated after fuzzy inference are xOutputMove and yOutputMove. Both control the firing of thrusters of the spaceship to navigate to the desired position. For detailed despeciation of each variable refer to Table 1.

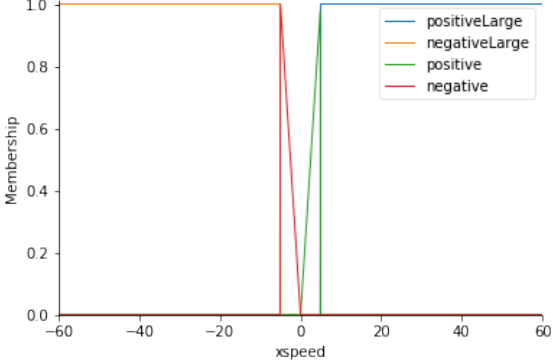
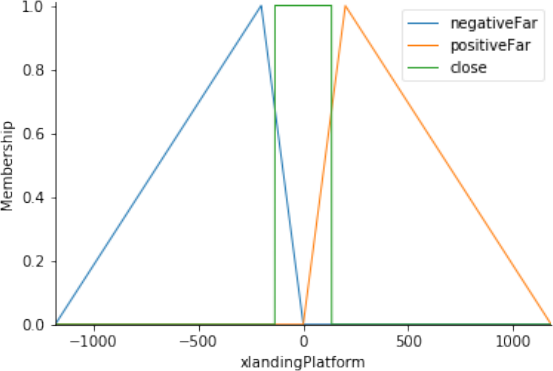
2.2. Fuzzy Rules

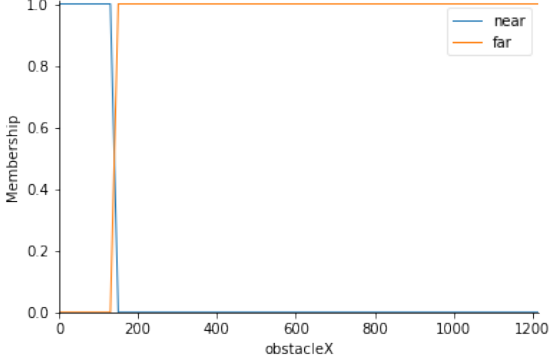
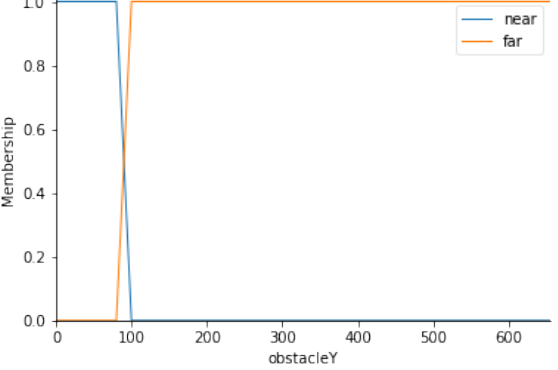
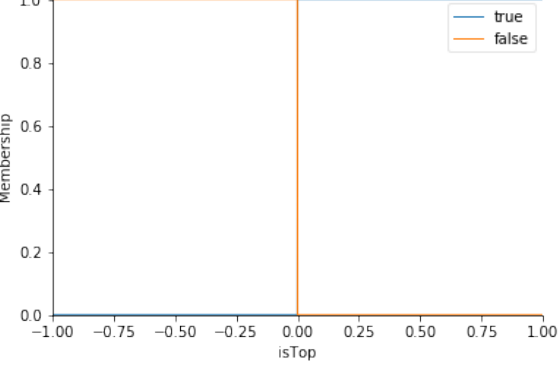
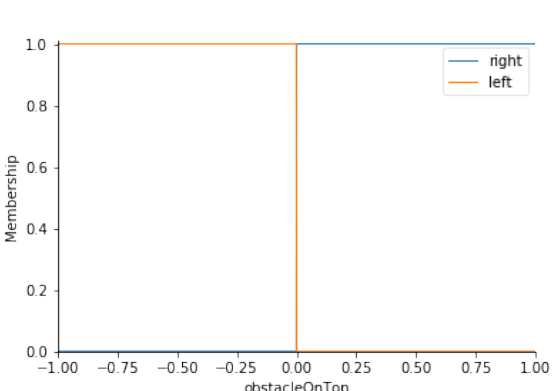
The fuzzy rules that control the operations of output variables yOutputMove and xOutputMove and are described in Table 2. The rules from (1) to (4) [rule numbers as specified on Table 1], specify the boundary walls constraints and ensure that the spaceship doesn't collide with the boundary walls. For rule (1), when the distance from lower wall (ground) is near and the spaceship is not above the landing platform, the bottom thruster of the spaceship fires and the spaceship moves up in order to avoid collision with the ground. But when the spaceship is above the landing platform, this rule is not fired so as to land the spaceship on the platform.

For rules (2) to (4), the closeness to the respective wall is checked along with whether the direction of spaceship is in that direction. This enables the automated controller to not fire the rule (2) when the spaceship is near to upper wall but is going in downward direction (measured by yspeed is positive). In absence of this check (yspeed is positive), the rule would be fired even when the spaceship is moving down which would increase the speed in downward direction and may cause collision with any obstacle (for obstacle version).

The rules from (5) to (8) control the speed of the spaceship and ensure that it never attains high speed in both x-direction and y-direction. The speed is controlled in order to avoid collision with obstacle or walls and land smoothly (yspeed less than 5 units) on the landing platform. For rules (5) and (6), if the yspeed exceeds 4.5 units in downward or upward

Input / Output Variable and Description	Membership Function
<p>lowerWall (Input Variable)</p> <p>It specifies the distance between the ground and the bottom tip of the spaceship.</p> <p>$h = 656$ (maximum allowed distance between the ground and the bottom tip of spaceship)</p> <p>The range of values in the universe is $(0, h)$. It has two terms, “near” and “far”, both have trapezoid membership function with the configuration $[0, 0, h/8, h/7]$ and $[h/8, h/7, h, h]$ respectively.</p>	
<p>upperWall (Input Variable)</p> <p>It specifies the distance between the upper boundary wall and the top tip of the spaceship.</p> <p>$h = 656$ (maximum allowed distance between the upper boundary wall and the top tip of the spaceship)</p> <p>The range of values in the universe is $(0, h)$. It has two terms, “near” and “far”, both have trapezoid membership function with the configuration $[0, 0, h/10, h/8]$ and $[h/10, h/8, h, h]$ respectively.</p>	
<p>leftWall (Input Variable)</p> <p>It specifies the distance between the left boundary wall and the left tip of the spaceship.</p> <p>$w = 1216$ (maximum allowed distance between the left boundary wall and the left tip of the spaceship)</p> <p>The range of values in the universe is $(0, w)$. It has two terms, “near” and “far”, both have trapezoid membership function with the configuration $[0, 0, w/25, w/20]$ and $[w/25, w/20, w, w]$ respectively.</p>	
<p>yspeed (Input Variable)</p> <p>It specifies the speed of the spaceship in y-direction. Positive value indicates that the spaceship is moving down, negative value indicates that the spaceship is moving up and zero indicates that the spaceship is still in the y-direction.</p> <p>The range of values in the universe is $(-40, 40)$. The value 40 is obtained by computing the maximum possible speed in the game. It has four terms, “positiveLarge”, “negativeLarge”, “positive” and</p>	

<p>“negative”. Both “positiveLarge” and “negativeLarge” have rectangle membership function with the configuration [4.5, 40] and [-40, -4.5] respectively. Both “positive” and “negative” have triangle membership function with the configuration [0, 4.5, 4.5] and [-4.5, -4.5, 0] respectively.</p>	
<p>xspeed (Input Variable)</p> <p>It specifies the speed of the spaceship in x-direction. Positive value indicates that the spaceship is moving right, negative value indicates that the spaceship is moving left and zero indicates that the spaceship is still in the x-direction.</p> <p>The range of values in the universe is (-60, 60). The value 40 is obtained by computing the maximum possible speed in the game. It has four terms, “positiveLarge”, “negativeLarge”, “positive” and “negative”. Both “positiveLarge” and “negativeLarge” have rectangle membership function with the configuration [5, 60] and [-60, -5] respectively. Both “positive” and “negative” have triangle membership function with the configuration [0, 5, 5] and [-5, -5, 0] respectively.</p>	
<p>xlendingPlatform (Input Variable)</p> <p>It specifies the distance between the x-coordinate mid-point of landing platform and the x-coordinate mid-point of the rocket.</p> <p>landingPlatformMid = 200 (This value gets updated for each game and defines the x-coordinate mid-point for the landing platform)</p> <p>landingSpaceWidth = 135 (This is fixed value specifying the width of landing platform)</p> <p>wLandingPlatform = 1180.5 (Maximum possible value for xlendingPlatform)</p> <p>The range of values in the universe is (-wLandingPlatform, wLandingPlatform). It has three terms, “negativeFar”, “positiveFar” and “close”. Both “negativeFar” and “positiveFar” have triangle membership function with the configuration [-wLandingPlatform, -landingPlatformMid, 0] and [0, landingPlatformMid, wLandingPlatform] respectively. “close” has rectangle membership function with the configuration [-landingSpaceWidth, landingSpaceWidth].</p>	

<p>obstacleX (Input Variable)</p> <p>obstacleX specifies the distance between the x-coordinate mid-point of the nearest obstacle and the x-coordinate mid-point of the spaceship.</p> <p>$w = 1216$ (maximum possible obstacleX value)</p> <p>The range of values in the universe is $(0, w)$. It has two terms, “near” and “far”. Both “near” and “far” have trapezoid membership function with the configuration $[0, 0, 130, 150]$ and $[130, 150, w, w]$ respectively.</p>	
<p>obstacleY (Input Variable)</p> <p>obstacleY specifies the distance between the y-coordinate mid-point of the nearest obstacle and the y-coordinate mid-point of the spaceship.</p> <p>$h = 656$ (maximum possible obstacleY value)</p> <p>The range of values in the universe is $(0, h)$. It has two terms, “near” and “far”. Both “near” and “far” have trapezoid membership function with the configuration $[0, 0, 80, 100]$ and $[80, 100, h, h]$ respectively.</p>	
<p>isTop, isBottom, isLeft, isRight (Input Variables)</p> <p>isTop specifies whether the spaceship is on top of nearest obstacle.</p> <p>Similarity, isBottom, isLeft and isRight specify whether the spaceship is bottom, left and right with respect to the nearest obstacle.</p> <p>All four variables have values in the range $(-1, 1)$. All four have two terms, “true” and “false”. Both “true” and “false” have rectangle membership function with the configuration $[0, 1]$ and $[-1, 0]$ respectively.</p>	
<p>landingAllowed, obstacleOnTop (Input Variables)</p> <p>landingAllowed specifies whether the spaceship can be landed on the platform or not, that is, the x-coordinate mid-point of the spaceship is within the range of the landing platform width.</p> <p>obstacleOnTop specifies whether the x-coordinate mid-point of the spaceship is towards the right or left with respect to the x-coordinate mid-point of nearest obstacle.</p> <p>Both the variables have values in the range $(-1, 1)$. Both have two terms, “true” and “false”. Both “true” and “false” have rectangle membership function with</p>	

the configuration [0, 1] and [-1, 0] respectively.	
<p>yOutputMove (Output Variable)</p> <p>It specifies the control for the top and bottom thrusters of the spaceship. This is later controlled by the key events on the keyboard down arrow and up arrow keys respectively.</p> <p>The range of values in this universe is (-1, 525). As there are 525 keyboard key press events possible. The numbers 38 and 40 corresponds to the key press of up arrow and down arrow keys respectively. [6] The value -1 is used to indicate the event of pressing no key on the keyboard.</p>	
<p>xOutputMove (Output Variable)</p> <p>It specifies the control for the left and right thrusters of the spaceship. This is later controlled by the key events on the keyboard right arrow and left arrow keys respectively.</p> <p>The range of values in this universe is (-1, 525). As there are 525 keyboard key press events possible. The numbers 37 and 39 corresponds to the key press of left arrow and right arrow keys respectively. [6] The value -1 is used to indicate the event of pressing no key on the keyboard.</p>	

Table 1 Input / Output Variables and their Membership Functions

Sr. No.	Rules for basic version
1	if lowerWall is near and landingAllowed is false then yOutputMove is up
2	if upperWall is near and yspeed is negative then yOutputMove is down
3	if leftWall is near and xspeed is negative then xOutputMove is right
4	if rightWall is near and xspeed is positive then xOutputMove is left
5	if yspeed is positiveLarge then yOutputMove is up
6	if yspeed is negativeLarge then yOutputMove is down
7	if xspeed is positiveLarge and isTop is false then xOutputMove is left
8	if xspeed is negativeLarge and isTop is false then xOutputMove is right
9	if xlandingPlatform is negativeFar then xOutputMove is left
10	if xlandingPlatform is positiveFar then xOutputMove is right
	Added Rules for obstacle version
11	if isTop is true and obstacleY is near then yOutputMove is up
12	if isBottom is true and obstacleY is near then yOutputMove is down
13	if isLeft is true and obstacleX is near then xOutputMove is left
14	if isRight is true and obstacleX is near then xOutputMove is right
15	if obstacleY is near and isTop is true and obstacleOnTop is right then xOutputMove is right
16	if obstacleY is near and isTop is true and obstacleOnTop is left then xOutputMove is left

Table 2 Fuzzy Rules

direction, then the rules are fired to activate opposite thrusters. For rules (7) and (8), the same concept is used as that for yspeed, but for both cases the x-direction speed is allowed to increase if there is any obstacle below the spaceship (for obstacle version). Both these rules ensure that the spaceship is not stationary in x-direction for the case when obstacle and landing platform are inline in y-direction.

The rules (9) and (10) control the movement of spaceship towards the target, that is, the landing platform. Both rules capture the idea that when the spaceship is far from landing platform, fire thrusters to move it over the landing platform. The output variables for all the rules is either xOutputMove or yOutputMove. For this implementation, the spaceship can move in both x-direction and y-direction simultaneously upon firing of multiple fuzzy rules.

2.3. Fuzzy Inference

For both output variables, xOutputMove and yOutputMove, there are two terms. xOutputMove has “up” and “down”, whereas yOutputMove has “left” and “right”. But both the values for these variables are opposite of each other, since at a time the spaceship can go up or down in a given frame inference. To avoid this contradiction, the value corresponding to maximum degree of membership among the two is chosen and returned to the fuzzy controller for execution. The algorithm is described below in detail.

```
MaxActivation()
{
    maxDegree = -Infinity
    maxTermValue = Not Assigned

    for (Activated activated : fuzzyOutput.getTerms())
    {
        currentDegree = activated.getDegree()
        currentTermValue = activated.getTerm().membership(currentDegree)
        if maxDegree < currentDegree
        {
            maxDegree = currentDegree
            maxTermValue = currentTermValue
        }
    }

    return maxTermValue
}
```

Fig. 2 Algorithm of MaxActivation

3. APPROACH FOR OBSTACLE VERSION OF LUNAR LANDER

For the obstacle version, obstacles are added to the universe of the Lunar Lander game and the objective still remains the same. The only difference is now the spaceship needs to avoid collision with obstacles too. We added the obstacles in the basic version and defined the functions for obstacle initialization, to check for collision of spaceship with the obstacle and obtain nearest obstacle to the spaceship at a given instant. For collision check between the spaceship and obstacle, the spaceship is divided into two rectangles of

dimensions 64 x 32, one lower rectangle and another upper rectangle. This division is done as both the lower and upper regions of spaceship are quite different in size as shown in Fig 1(a). Both the rectangles are checked for collision with the obstacles separately and if any one collides, then the entire spaceship collides. For obtaining the nearest obstacle to the spaceship, the obstacle having the least Euclidean distance is returned.

3.1. Input / Output Variables and Membership functions

The input and output variables and their corresponding membership functions are described in detail in Table 1. For obstacle version, some new input variables are activated such as obstacleX, obstacleY, isTop, isBottom, isLeft, isRight and obstacleOnTop. The obstacleX and obstacleY measure the distance between the nearest obstacle and the spaceship. The variable obstacleOnTop checks whether the x-direction mid-point of the spaceship is towards the right or left with respect to the x-direction mid-point of obstacle.

The variables isTop, isBottom, isLeft and isRight are used to return true when the spaceship is top, bottom, left and right with respect to the nearest obstacle. To obtain whether the spaceship is on top of the nearest obstacle, three conditions needs to be satisfied. These conditions are described below.

```
spaceship.y + spaceship.height + thresholdY <= nearestObstacle.y
```

```
spaceship.x + thresholdX >= nearestObstacle.x - rocket.width
```

```
spaceship.x - thresholdX <= nearestObstacle.x + nearestObstacle.width
```

Similarly, the conditions for isBottom, isLeft and isRight are derived to obtain the current situation of spaceship with respect to the nearest obstacle.

3.2. Fuzzy Rules

The fuzzy rules for the control of the spaceship are described in Table 2. The initial rules from (1) to (10) are also part of the basic version of the game. For the obstacle version, rules from (11) to (16) are added. The rules from (11) to (14) control the spaceship for four scenarios in which a spaceship can be with respect to an obstacle. For rule (11), when the obstacle is below the spaceship and the y-direction distance between them is near, then the bottom thrusters are fired to avoid collision with the obstacle. Similarly, the rules are written for other three scenarios.

The rules (15) and (16) control the movement of spaceship around an obstacle. For rule (15), when the spaceship is above and close to the obstacle and the spaceship is towards the right with respect to obstacle, then the `xOutputMove` is right. This provides the speed in right direction to avoid the obstacle.

4. EXPERIMENTAL RESULTS

The automated control system developed using the fuzzy logic concepts was tested over the Lunar Lander game. For the basic version, the accuracy is 100%, that is, the automated fuzzy player is able to land the spaceship smoothly and safely without crashing it with any of the boundary walls. For the obstacle version, for most of the configurations of obstacles the automated fuzzy player is able to land the spaceship without crashing it anywhere. But there are some scenarios in which the automated fuzzy layer collides with the obstacles and gets stuck in some deadlock of moves.

The configurations that doesn't work well are when the landing platform and obstacle are inline. When such a situation happens, half of the cases the system goes into a deadlock of moves and doesn't recover. But for other half of the cases, the rules (15) and (16) are fired and is able to recover from deadlock and land the spaceship. The other scenario in which the system fails is when the spaceship collides with the obstacle diagonally.

5. CONCLUSIONS AND DISCUSSIONS

The fuzzy logic implementation for the basic version of Lunar Lander is able to capture all the constraints posed by the game and successfully pass all of them. For the obstacle version, the challenge increases when the number of obstacles added to the game is large. For such scenarios, there are cases when the automated fuzzy player is not able to reach the landing platform. Such configuration of obstacles requires more finer adjustment for the membership function values and cut-points (in case of triangle and trapezoid membership functions). This finer adjustment for the membership function values and cut-points can be obtained by GA search over the universe of input variables. A hybrid GA-Fuzzy system may help to improve over the current only Fuzzy system.

REFERENCES

[1] en.wikipedia.org. (2018). *Lunar Lander (video game genre)*. [online] Available at: [https://en.wikipedia.org/wiki/Lunar_Lander_\(video_game_genre\)](https://en.wikipedia.org/wiki/Lunar_Lander_(video_game_genre)).

[2] GitHub. (2018). *ReyKoxha/LunarLander*. [online] Available at: <https://github.com/ReyKoxha/LunarLander>.

[3] docs.oracle.com. (2018). *Coordinates (The Java™ Tutorials > 2D Graphics > Overview of the Java 2D API Concepts)*. [online] Available at: <https://docs.oracle.com/javase/tutorial/2d/overview/coordinate.html>.

[4] Juan Rada-Vilela. *fuzzylite: a fuzzy logic control library*, 2017. URL <http://fuzzylite.com/>.

[5] Takagi, Tomohiro, and Michio Sugeno. "Fuzzy identification of systems and its applications to modeling and control." *IEEE transactions on systems, man, and cybernetics* 1 (1985): 116-132.

[6] docs.oracle.com (2018). Constant Field Values. [online] Available at: <https://docs.oracle.com/javase/7/docs/api/constant-values.html>.