# Intelligent Systems

Fuzzy control system for the Flappy Bird game

2019. 06. 15.

ISEL - Instituto Superior de Engenharia de Lisboa

**Grupo 3.**

Péter Ádám Bánkuti (46684)
Pedro Vasconcelos (36789)
Ivanilda Cardoso (46618)

# Introduction

The aim of this project is to implement a fuzzy logic control to automate the Flappy Bird game, using Matlab's Fuzzy toolbox. The game is a side-scroller where the player controls a bird, attempting to fly between columns of green pipes without hitting them. Originally this game was released for mobile-phones, however a different developer implemented it in Matlab and released it on GitHub. During the development the Matlab version was modified to accommodate the control system and to visualize the information.

In brief, the game works by simulating a falling bird which moves horizontally with a constant speed, while different towers appear from the right. The player can only control the bird in one input channel, by jumping. During each jump the vertical acceleration is reset to a constant value, consequently, the jump is always of a fixed sized and the timing of the jump is essential. (For example: the vertical acceleration is -4.23, thus is falling; the jump causes it to become 2.5 and the bird goes up for a while, however the effect of the gravity brings it down eventually.)

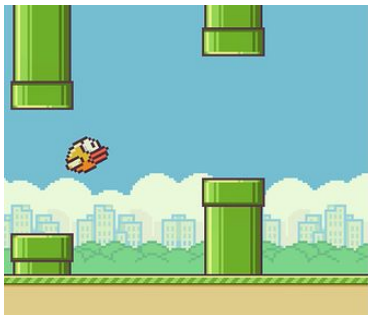


***Fig 1.:*** *Screenshot of the Flappy Bird game.*

## Development

During the development several control modes were implemented, some with severe modifications on the original control. The inputs of the Fuzzy Control were altitude, vertical distance to the next gap $y_{center}$, and its derivate. The control outputs that were tested:

I. **Delay the next jump.**

   In this setup at frame $z_n$ of the gameplay a jump is executed and the Fuzzy Control is called with the *altitude* and *vertical distance to the gap center* values, and after evaluating the values, the fuzzy system would return a temporal $t$ value signifying a delay until next jump/evaluation in frame $z_{n+t}$. By using this setup the bird reacted slowly to the changes, and besides the towers often hit the ground, too.

II. **Decide to jump or not jump at each frame.**

   Feeding (at each frame of the gameplay) the input value of *vertical distance to the gap center* to the fuzzy system, the resulting output value was considered as a boolean, meaning the need for a jump in the next frame or not. This resulted in a smooth fast control, which rarely collide with the towers and never with the ground. On the other hand the control became so simple that it could hardly be called a fuzzy system.

III. **Decide to jump at every $n^{th}$ frame**

   This was a modification of the previous control system, where the evaluation happened at every $n^{th}$ frame. This required a slightly more complex ruleset.

**IV. Regulate the jump acceleration at every $n^{th}$ frame**

In order to have a complex enough fuzzy system, the game control logic was modified. Instead of executing a fixed sized jump at any chosen time, the jump size would vary and execution would happen at every $n^{th}$ frame. This resulted in a good control mode, with few collisions with the towers and ground.

## 1. Setting up the fuzzy logic

The best implementation of the fuzzy logic was in the IV. control mode, which sought to regulate the jump sizes at fixed frame rate, and this was achieved by modifying the game logic. The fuzzy logic is evaluated at every 20th frame and the with the input values are:

- $y_{altitude}$ - altitude
- $y_{center}$ - vertical distance to the gap center;
- $dy_{center}$ - derivate of the vertical distance to the gap center.

While the output is:

- $a_{jump}$ - jump acceleration (if is not positive the bird keeps falling).

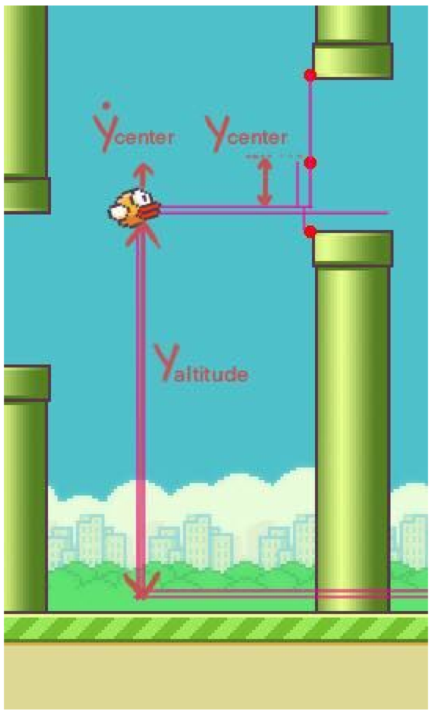


***Fig 2.:*** *Visualization of the input variables of the fuzzy logic.*

After having set up the use of  fuzzy logic within the code of the game the next step was to create the logic with the toolbox.

A Fuzzy Inference System (FIS) is a way of mapping an input space to an output space using fuzzy logic. A FIS tries to formalize the reasoning process of human language by means of fuzzy logic (that is, by building fuzzy IF-THEN rules). For instance: If the bird is flying dangerously low to the ground and is falling with high speed than jump upwards fast.

The matlab fuzzy toolbox offers two interface types: Sugeno or Mamdani. For this control mode the Mamdani interface was utilized, since the output value is not-discreet. (For the control modes discussed in the II. and III. point the Sugeno interface were utilized, where the output was discreet decision to jump or not at the evaluated frame.)

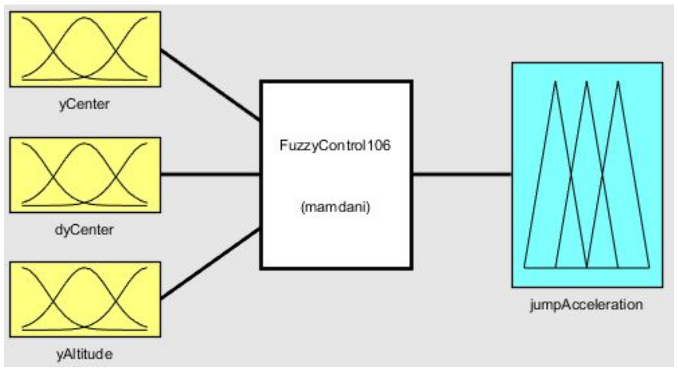


***Fig 3.:*** *Visualization of the input and output variables of the fuzzy logic.*

## 2. Defining the membership functions

The next step was to define the membership functions for the variables. These variables were not normalized, because it wouldn't have helped the empirical analysis of the game. Instead, the input variables $y_{center}$ and $y_{altitude}$ continued to represent the distance metric defined in the game. (This metric consisted of 0 as the ground and 185 as upper edge of the window.)

For example: in the case of the:

- $y_{altitude}$ variable (fig. 5) two functions were defined, ***danger*** and ***ok***, where the former was used to avoid the collision with the ground in edge cases.
- $y_{center}$ represented (fig. 6) the offset from the targeted altitude at a given stint of the game. This target altitude was the mid-point between the next tower pair (fig. 4). The variable has 5 member functions: far below, just bellow, centered, just above and far above target altitude.
- $dy_{center}$ has the information (fig. 7), whether the bird is falling fast, gliding down slowly or ascending.

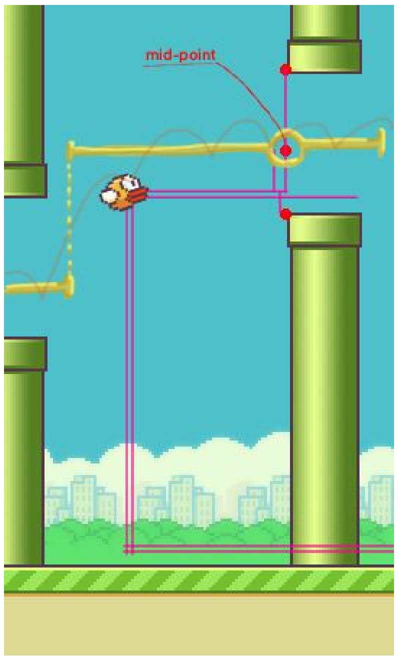


***Fig 4.:** Visualization of the reference level for the control system..*

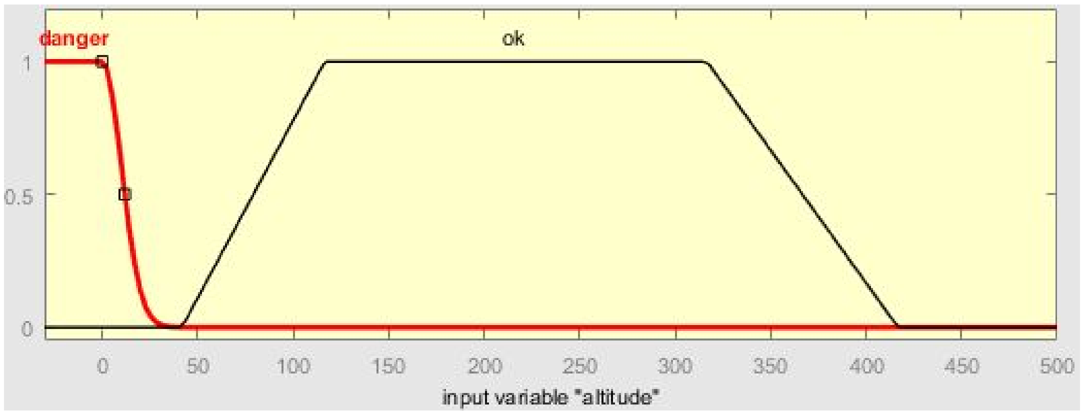


***Fig 5.:** Membership functions for the altitude variable.*

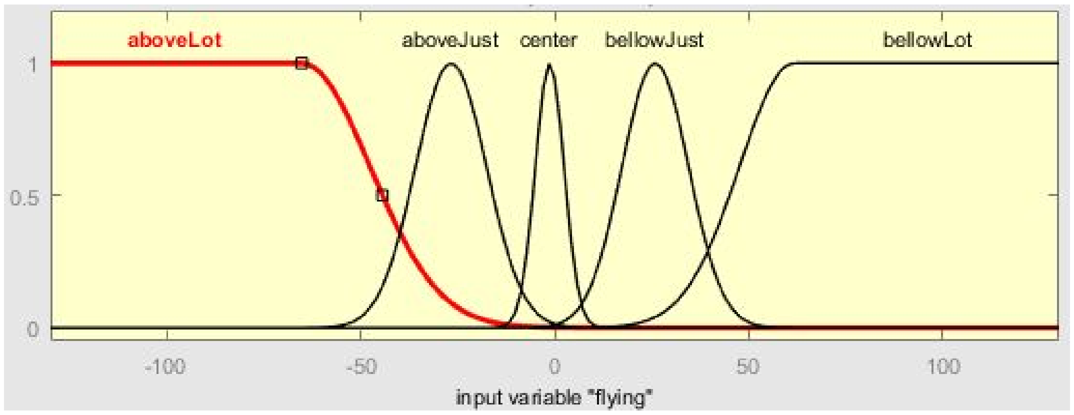


***Fig 6.:*** *Membership functions for the flying (y) variable.*

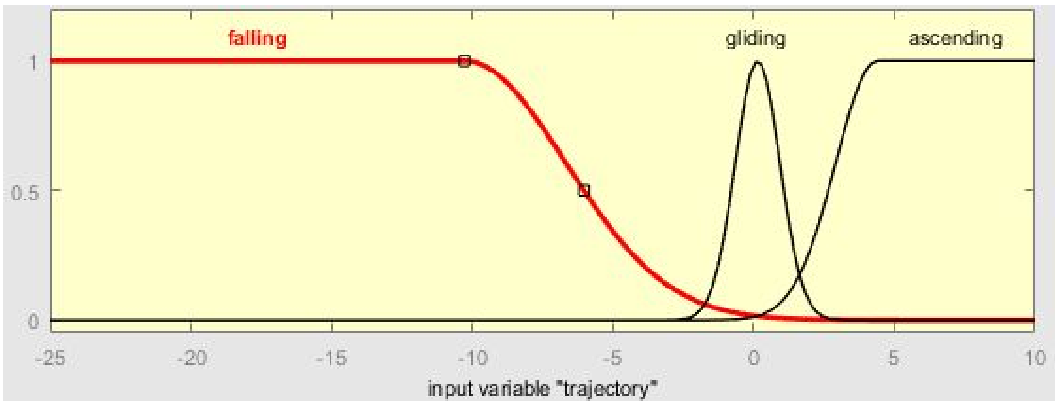


***Fig 7.:*** *Membership functions for the trajectory (dy) variable.*

The output variable (jump acceleration *overwrite) has 5 membership functions (fig. 8), where each signifying an increasing acceleration level. The function “0” has negative values, which in the code of the game is interpreted as “do nothing, keep falling”. After evaluation of the FIS the output value (if is positive) overwrites the one present in the code. (The value is overwritten and added because the original game logic did it this way, and to include less modification in the game logic it was kept the same way.)

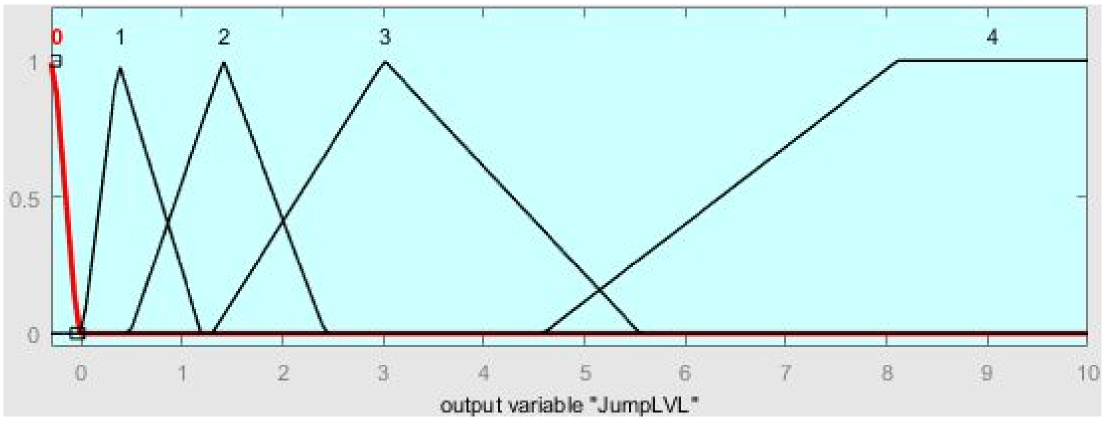


***Fig 8.:*** *Membership functions for the output (jump acceleration *overwrite) variable.*

## 3. Defining the rule set

The aim of the fuzzy logic is to reduce the collisions with the towers and ground. By checking the behaviour of the game with the manual jump inputs a general sense for the rules and membership functions can be derived. After defining the initial ruleset further adjustments were made by observing the effectiveness of the obstacle avoidance.

| | $y_{center}$ | $\mathrm{d}y_{center}$ | $y_{altitude}$ | Level of $a_{jump}$ |
|---|---|---|---|---|
| 1 | bellow a lot | falling | - | 4 |
| 2 | bellow a lot | not falling | - | 3 |
| 3 | just bellow | falling | - | 2 |
| 4 | just bellow | not falling | - | 1 |
| 5 | center | ascending | - | keep falling |
| 6 | center | not ascending | - | 2 |
| 7 | just above | falling | - | 1 |
| 8 | just above | gliding | - | 0 |
| 9 | just above | ascending | - | 1 |
| 10 | above a lot | - | - | keep falling |
| 11 | - | - | danger | 3 |

***Table 1.:*** *Rules of the fuzzy logic*

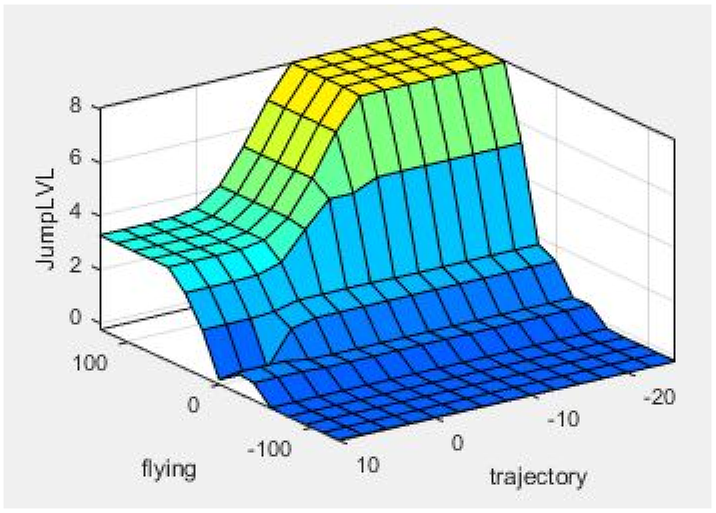


***Fig 9.:*** *Surface view of the flying (y) - trajectory (dy) - jump variables*

## Results

The control system achieved by the control-mode IV. resulted in a fairly effective control mode. In the usual setup of the game, where the towers are separated by a fix-sized gap (90 game-units) the game didn't fail.

The effectiveness of a control system was measured with the score variable (number of passed towers) and collision variable (number of frames during which the bird would have died by colliding; eg: by passing through the tower this variable would increase ~15, while touching a corner would increment only 1~3 ) . The main influencer of this score was the distance between the towers, because in the case where the passage-gaps between the two adjacent towers were far apart, and distance between those towers were short, than the bird started to hit the corners of the towers.

$$\textit{system error rate} = \textit{collision} \div \textit{score}$$

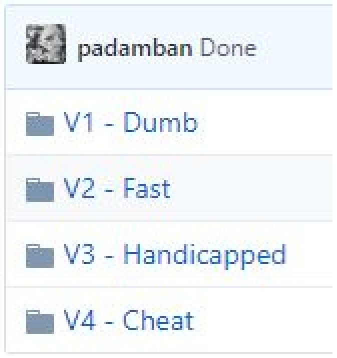
| Tower distance [gu] | 80 | 90 (default) | 90 (default) |
|---|---|---|---|
| Jumps | 391 | 435 | 2152 |
| Collision | 19 | 0 | 1 |
| Score | 100 | 100 | 500 |
| System error rate (0 is the best) | 0,19 | 0 | 0,002 |

***Table 2.:*** *Measuring the effectiveness of the control system*

## Executing the code

The code can be found in the https://github.com/padamban/FuzzyBird repository. Each folder is the implementation of a different control-modes, where "V4 - Cheat" folder contains the IV. mode, described in detail in this project documentation. To run the program open the folder in matlab and execute the flappybird.m script, and the game can be controlled by the following keys:

- Space: manual jump (functional in mode I. II. and III.)
- Space: reset after gameover; start the game
- Up-arrow: turn on/off fuzzy control; start


padamban Done

V1 - Dumb

V2 - Fast

V3 - Handicapped

V4 - Cheat

***Fig 9.:*** *Folders of the different control implementations.*

## References

1. Mingjing Zhang, Implementation of the Flappy Bird game  in Matlab, 2016 https://github.com/mingjingz/flappybird-for-matlab
2. Web based reference game, 2014 -  http://flappybird.io/
3. Source code,, 2019 -  https://github.com/padamban/FuzzyBird