**ME 4773/5493 Fundamental of Robotics**

**Fall 2016**

**San Antonio, TX, USA**

Playing flappy bird using a model predictive control

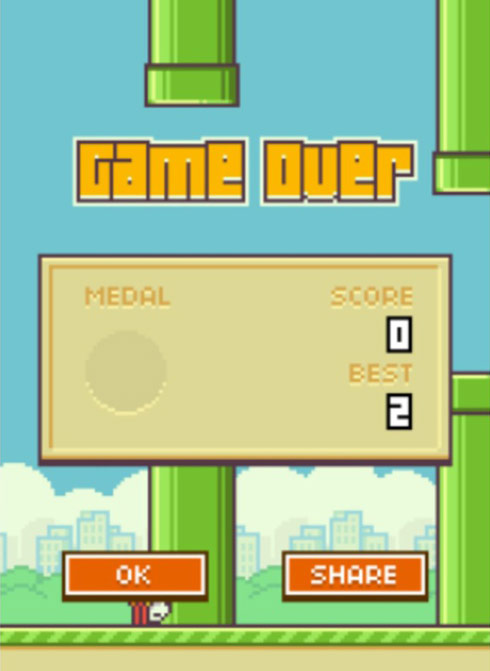
|  |
| --- |
| Matthew Piper  Dept. of Mechanical Engineering  San Antonio, TX, USA 78249  MatthewPiper@utexas.edu |

Abstract

If it’s hard for humans to do, write a program to do it. When Flappy Bird was introduced to the world it proved to be a very difficult game. Using this platform, two methods of path planning were used to beat the game. One method was a heuristic and the other a model predictive control that used mixed integer linear programming. The heuristic method didn’t require calculations, so it was fast and was able to be used online with no problems. It only achieved a score of 23 though. The model predictive control scored 344, but the optimization took time and wasn’t feasible to be used online, yet. The goal for the future is to improve the model predictive control so that it can be used online smoothly.

1. InTRODUCTION

Flappy Bird was a game made for smart phones in 2013. It quickly became popular and for those who have played it, it quickly became frustrating. It got a reputation for being addicting [1] and being very hard to play.



The fact that it is very difficult to play as a human being makes it a good problem to solve with controls.

The game design is simple: sets of tubes move across the screen and the bird jumps when the player touches the screen. The idea is to make it through as many tubes as possible. If the player hits a tube, Game Over.

A couple of methods were used to create a controller to play the game autonomously. One was a heuristic method and the other was a model predictive control. Model predictive control (MPC) has become more popular in industry as technology surrounding it has improved. As computers get faster and algorithms more efficient, MPC is being utilized in a wider range of applications [2]. Beating video games may not be one of those applications, but solving a straight forward relatively simple problem such as the Flappy Bird problem could provide a general solution that can be used in a wide range of problems.

1. Nomenclature

Variables:

|  |  |
| --- | --- |
|  | Vertical position of bird |
|  | Vertical velocity of bird |
|  | Gravity constant |
|  | Adjustable set point |
|  | Height of bottom tube |
|  | Constant to adjust set point |
|  | Large constant for Big-M method |
|  | Integer variable (0 or 1) |
|  | End constraint of path |
|  | Current x position |

1. METHODS

The Flappy Bird game used in the project was downloaded from the MathWorks File Exchange[3], so the first thing that had to be done was to understand how the code worked. The most important part was figuring out the dynamics of the bird. The equations that describe the motion of the bird are:

|  |  |
| --- | --- |
|  | (1) |
|  | (2) |

Where k is the current iteration, Y is the bird’s vertical position, V is the bird’s vertical speed and g is a constant that represents gravity. Figuring out the purpose of the variables in the game was important for collecting data and passing information to the function I wrote for the controller. The position and height of the tubes, as well as the dynamics of the bird were used to create a controller for the game.

The way the game works is that the player will press the space bar to make the bird jump. In the code, when the space bar is pressed, the velocity, , is set to a value of -2.5. The y axis increases downward, so -2.5 is an upward velocity.

The first method used to create a controller was a heuristic one. A set point was created based on the height of the leading tube. Then an if statement was implemented so that the bird would jump if it was below that set point:

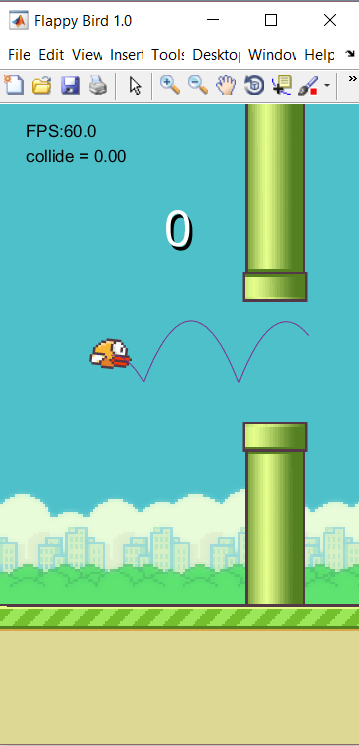
where,

and setPointY is the vertical position of the top of the bottom tube and c is a constant. The constant, c, raised or lowered the set point that the bird had to stay above, and was chosen to be 10 to position the bird with some leeway between the tubes.

The next method used was a model predictive control. A separate function was created for the controller that used mixed integer linear programming to optimize a path for the bird. The length of the path that was optimized was 80 pixels which is long enough for the bird to travel through one set of tubes. The cost function used was the sum of the jumps over that interval. MATLAB’s intlinprog() function [4] was used to do the optimization. The problem was set up in the following way in order to use linear programing to solve it:

|  |  |
| --- | --- |
|  | (3) |
|  | (4) |
|  | (5) |
|  | (6) |
|  | (7) |
|  | (8) |
|  | (9) |
|  | (10) |

Equations 4-7 were derived using the Big-M method [5] where M is a large constant and z is an integer variable. and are the initial position and velocity of the path and are set equal to the end values of the previous path. The variable, , in equation 10 constrains the end position of the path to be halfway between the height of the leading bottom tube and the height of the next bottom tube. An optimized path can be seen in the game below:



The constraints set were bigger than the tubes themselves to ensure the bird did not collide with them because the path tracks the middle of the bird, but if the edge of the bird hits the tube the game ends.

1. RESULTS

The tubes in the game are generated using a random number generator, but to compare the two methods the seed of the generator was set to 1. This way the constraints would be the same to test both methods. The heuristic method was simple and scored more points than a person new to the game might. The model predictive control didn’t work as smoothly as an online controller, but it scored much higher than the heuristic method.

|  |  |
| --- | --- |
| Method | Score |
| Heuristic | 23 |
| MPC | 344 |

1. DISCUSSION

The model predictive control was able to travel further because it took in more information than the other method. The optimization took in the height of the leading two tubes and optimized a path over the first one with respect to the next one. The optimization, however, took time, and the game paused at the end of each path to calculate the next one.

The heuristic method was only able to “see” the leading tube. This lead to situations where the bird wasn’t positioned in a feasible way after passing through one set of tubes to make it through the next. There were no calculations though, so it worked seamlessly in the game.

1. Conclusion and FUTURE work

It can be seen that a model predictive control can be used to solve a path planning problem. A limitation of this method is that it takes time to plan a path. Also, for a model predictive control to work properly, the system needs to be well defined because it plans based on the dynamics of the assumed system. In this project the system is known perfectly and there are no disturbances so the bird moves along the path without any deviation.

The main goal of the project is to find a way to improve the model predictive control so it is faster and can be used online smoothly. Future work on this project will include finding a way to introduce better initial conditions, defining the constraints in ways as to minimize calculations, and learning more about the algorithms used so they can be modified to better fit a specific problem.

Acknowledgments

I would like to acknowledge my advisor, Dr. Pranav Bhounsule, for introducing me to model predictive controls and suggesting the Flappy Bird problem.

References

[1] Nguyen, Lan Anh. “Exclusive: Flappy Bird Creator Dong Nguyen Says App 'Gone Forever' Because It Was 'An Addictive Product'” Forbes, 11 February 2014. Web. 1 December 2016.

[2] Qin S. J. & Badgwell T. A. (2003) A survey of industrial model predictive control technology. *Control Engineering Practice, 11*(7), pp. 733-764. http://www.forbes.com/sites/lananhnguyen/2014/02/11/exclusive-flappy-bird-creator-dong-nguyen-says-app-gone-forever-because-it-was-an-addictive-product/#1923d6af9f67

[3] www.mathworks.com/matlabcentral/fileexchange/45795-roteaugen-flappybird-for-matlab

[4] https://www.mathworks.com/help/optim/ug/intlinprog.html

[5] How, J., Richards, A. Mixed-integer Programming for Controls [Powerpoint slides]. Retrieved from http://acl.mit.edu/milp/MILP\_for\_Control.pdf