HOW TO BEAT FLAPPY BIRD: A MIXED-INTEGER MODEL PREDICTIVE CONTROL APPROACH

by

MATTHEW PIPER, B.S.

THESIS
Presented to the Graduate Faculty of
The University of Texas at San Antonio
In Partial Fulfillment
Of the Requirements
For the Degree of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

COMMITTEE MEMBERS:
Pranav Bhounsule, Ph.D., Chair
Krystel Castillo, Ph.D.
Ahmad Taha, Ph.D.

THE UNIVERSITY OF TEXAS AT SAN ANTONIO
College of Engineering
Department of Mechanical Engineering
May 2017

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Pranav Bhounsule, for suggesting this research topic and introducing me to the field of robotics and controls. I learned a lot at The University of Texas at San Antonio and look forward to applying my skills to innovative engineering projects in the future.

May 2017

HOW TO BEAT FLAPPY BIRD: A MIXED-INTEGER MODEL PREDICTIVE CONTROL APPROACH

Matthew Piper, M.Sc.
The University of Texas at San Antonio, 2017

Supervising Professor: Pranav Bhounsule, Ph.D.

Flappy Bird is a mobile game that involves tapping the screen to navigate a bird through a gap between pairs of vertical pipes. When the bird passes through the gap, the score increments by one and the game ends when the bird hits the floor or a pipe. Surprisingly, Flappy Bird is a very difficult game and scores in single digits are not uncommon even after extensive practice. In this paper, we create three controllers to play the game autonomously. The controllers are: (1) a manually tuned controller that flaps the bird based on a vertical set point condition; (2) an optimization-based controller that plans and executes an optimal path between consecutive pipes; (3) a model-based predictive controller (MPC). Our results showed that on average, the optimization-based controller scored highest, followed closely by the MPC, while the manually tuned controller scored the least. A key insight was that choosing a planning horizon slightly beyond consecutive pipes was critical for achieving high scores. The average computation time per iteration for the MPC was half that of optimization-based controller but the worst case time (maximum time) per iteration for the MPC was thrice that of optimization-based controller. The success of the optimization based controller was due to the intuitive tuning of the terminal position and velocity constraints while for the MPC the important parameters were the prediction and control horizon. The MPC was straightforward to tune compared to the other two controllers. Our conclusion is that MPC provides the best compromise between performance and computation speed without requiring elaborate tuning.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: INTRODUCTION

1.1 Flappy Bird

Flappy Bird was a game made for smart phones in 2013. It consists of a bird flying horizontally at a constant speed but falling continuously under gravity. The bird can fly upward by repeatedly tapping on the screen. The objective of the game is to get the bird to pass through a series of green pipes as shown in figure 1.1:

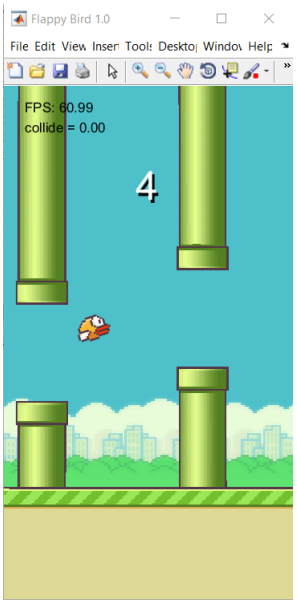


Figure 1.1: Flappy Bird game

The horizontal distance between two consecutive pairs of pipes is fixed. Although the vertical height of the gap between a pair of pipes is fixed, the gap location changes randomly. The player scores a single point every time she/he is able to successfully pass through the gap between the pair of vertical pipes. The game ends if the bird hits either the pipe or the floor. It quickly became popular, and for those who have played it, it quickly became frustrating. It got a reputation for being addicting [1] and 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. In this paper, an openly available MATLAB®version of the Flappy Bird was used, which preserves the features and difficulty of the original game [3].

1.2 Previous Work

Past works on automatic control of Flappy Bird have extensively focused on using machine learning algorithms. Shu et al. [7] used reinforcement learning (RL) and were able to achieve scores of around 1500 [8]. The states for learning in their formulation were the x and y position of the bird relative to the top of the upcoming bottom pipe and the bird velocity. The action was to flap or not flap. RL optimizes state-action pairs and transition probabilities to maximize a function.

Ebeling-Rust et al. [9] and Chen [10] used Q-learning, a variant of RL, but were only able to score around 200. The modest scores may have been a result of not using velocity as a state for the learning algorithm.

Shu et al. [7] also used Support Vector Machine (SVM), a supervised learning algorithm, to achieve scores of around 1200 [11]. One disadvantage of SVM is that it requires extensive training data generated from manually playing the game. SVM uses the training data to create a mapping from user-defined features to the two actions, flap or not flap. The authors used the x and y position of the bird relative to the pipe, bird velocity, as well as high order terms in the position and velocity, a total of 9 features.

All the machine learning algorithms require parameter tuning, extensive learning period, and careful selection of states or features. The greatest advantage of machine learning is that it does not need knowledge of the physics of the system. However, the physics of the Flappy Bird game is simple and available. This motivates the use of model-based control algorithms. Takacs et al. [12] used explicit model predictive control (explicit MPC) on flappy bird but have not reported their results. Explicit MPC solves the optimization problem off line and stores the solution. During implementation, the stored policy is searched and interpolated as needed to determine the necessary control actions.

1.3 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]. A model predictive control is one that predicts the states and inputs of a system over a certain period of time, called the prediction horizon. Knowing the dynamics and constraints of the system, an optimization can be performed to solve for the states and inputs over the prediction horizon. These inputs are then applied to the actual system for a period of time, called the control horizon. At the end of the control horizon the initial conditions of the optimization are updated and the process begins again. In this paper, the problem is formulated as a mixed integer linear program (MILP) and the optimization software, Gurobi [6], is used to solve the optimization problem.

1.4 Controlling Flappy Bird

In this paper, multiple methods were used to create controllers for Flappy Bird. First, a heuristic approach is used for a simple solution to the problem and to provide a baseline for more complex methods. Next, an optimization was used to create a control sequence to be implemented over the entire prediction horizon. This method included heuristically tuned constraints based on prior knowledge of the system behavior. Then, a model predictive control was implemented that only applied the beginning of the calculated control sequence as described above.

Chapter 2: THE SYSTEM

2.1 Notation

Table 2.1: Notation

Y	Vertical position of the middle of the bird
V	Vertical velocity
k	Current time step (subscript)
n	Prediction horizon (subscript)
p	Control horizon
g	Gravity constant = .1356
M	Large constant for Big-M method
z	Binary variable: 1 = jump, 0 = don't jump
J	Cost Function
$init$	Initial point for optimization (subscript)
$endYlow$	Lower bound on final position
$endVel$	End constraint on velocity
$endYhigh$	Upper bound on final position
lb	Lower bound (superscript)
ub	Upper bound (superscript)
$PipePos$	Position of the top of the bottom pipe

2.2 Dynamics

The Flappy Bird game used in this paper was downloaded from the MathWorks File Exchange [3]. The way the game works is that the player presses the space bar to make the bird jump. In the code, when the space bar is pressed, the velocity, V_k, is set to a value of -2.5. Otherwise, the bird is acted upon by gravity, which is simulated by increasing the velocity a constant amount in the downward direction at every time step. The system dynamics are described by the following discrete system of equations:

$$Y_{k+1} = Y_k + V_k \tag{2.1}$$

$$V_{k+1} = \begin{cases} -2.5, & z_k = 1 \\ V_k + g, & z_k = 0. \end{cases} \tag{2.2}$$

Where, Y_k, is the position of the middle of the bird at time, k. The y axis is positive downward, and the origin is at the top left corner of the screen. V_k is the vertical velocity of the bird at time k, and z is the binary variable that decides if the bird will jump ($z_k = 1$) or not ($z_k = 0$). Y_k and V_k are in units of pixels and pixels per unit time respectively. The time step is 1 unit so that time does not appear in any equation. Also, the bird is moving in the horizontal direction with constant speed of 1 pixel per unit time. The game was modified by replacing the space bar input with the binary decision variable, z_k, at every time step. These sets of equations are represented as constraints in the optimization problem introduced in chapter 3.

Chapter 3: METHODS

3.1 Heuristic Method

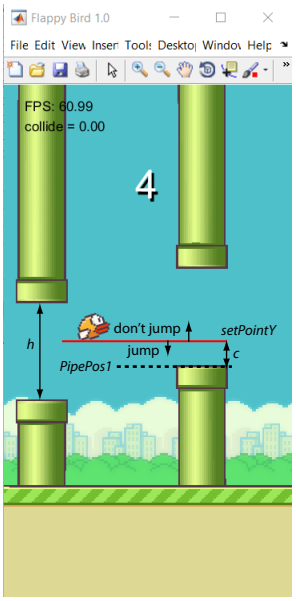


Figure 3.1: Visualization of heuristic method

The first method was developed intuitively by observing the game being played by humans. A set point was created based on the height of the leading pipe. Then an if statement was implemented so that the bird would jump if it was below that set point:

$$if \quad Y_k < setPointY, \; z_k = 1,$$
$$else, \; z_k = 0,$$

where $setPointY = PipePos1 + c$ and $PipePos1$ is the position of the top of the bottom pipe. The only tuning parameter is c which was manually tuned to 10. The controller logic can be seen in figure 3.1. The vertical distance between the pipes is $h = 48$.

3.2 Initial Optimization

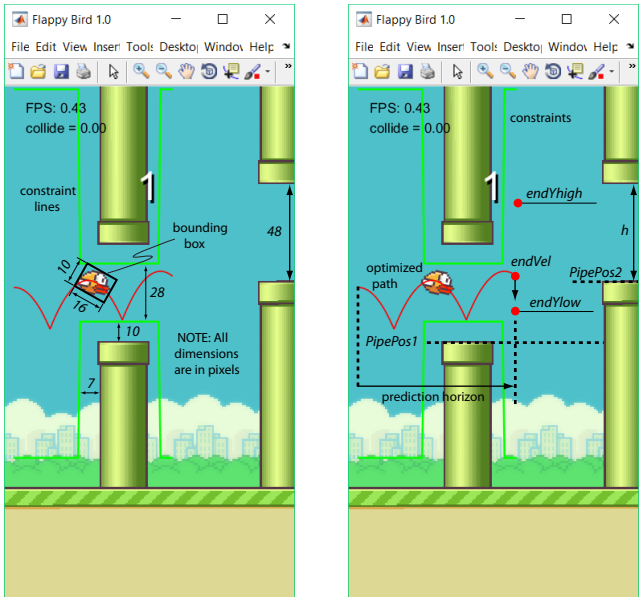


Figure 3.2: Optimization Parameters

In the next method, the problem was formulated as a mixed integer linear program to solve for the path of the bird over a prediction horizon of 80 pixels. The prediction horizon of 80 pixels is long enough for the bird to travel through one set of pipes. Given the initial conditions, Y_{init} and V_{init}, the objective of the optimization was to minimize the number of jumps over that interval subject to heuristically tuned end of path constraints. The formulation of the mixed integer linear program is given below:

$$\text{Minimize} \quad J = \sum_{k=1}^{n} z_k, \tag{3.1}$$

$$\text{Subject to} \quad Y_{k+1} - Y_k - V_k = 0, \tag{3.2}$$

$$-V_{k+1} + Mz_k \leq 2.5 + M, \tag{3.3}$$

$$V_{k+1} + Mz_k \leq -2.5 + M, \tag{3.4}$$

$$-V_{k+1} + V_k - Mz_k \leq -g, \tag{3.5}$$

$$V_{k+1} - V_k - Mz_k \leq g, \tag{3.6}$$

$$Y_k^{lb} \leq Y_k \leq Y_k^{ub}, \tag{3.7}$$

$$Y_1 = Y_{\text{init}}, \tag{3.8}$$

$$V_1 = V_{\text{init}}, \tag{3.9}$$

$$endYlow \leq Y_n \leq endYhigh, \tag{3.10}$$

$$V_n \leq endVel \quad \text{if} \quad PipePos2 > PipePos1. \tag{3.11}$$

Equations 3.3 - 3.6 were derived using the Big-M method [4] where M is a large constant and z is a binary variable that represents the jumps. The Big-M method is a technique of writing switching equations, such as equation 2.2, into multiple equations using a large number M. This is done in order to keep the constraint equations linear in the binary decision variable, z_k. By keeping the constraints linear, the efficient linear mixed integer programming software, Gurobi [6], could be utilized. Y_1 and V_1 are the initial position and velocity of the path and are set equal to the end values of the previous path. The constraints around the pipes were bigger than the pipes 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 pipe the game ends. This can be seen in figure 3.2.

The heuristically tuned constraints, represented by equations 3.10 and 3.11, are defined by the variables: $endYlow$, $endYhigh$ and $endVel$. In these equations, $PipePos1$ and $PipePos2$ represent the height of the bottom of the leading pipe and the following one respectively. These

constraints are visualized in figure 3.2. The end position of the bird is constrained by lower bound, $endYlow = 0.5(PipePos1 + PipePos2)$ and upper bound, $endYhigh = (PipePos2 + h) - 0.5|PipePos2 - PipePos1|$. The end constraint on velocity was $endVel = 1.1$ (positive velocity is downward) and was active only if the following pipe was lower than the current pipe as indicated by equation 3.11 (note that positive direction is downwards). These parameters were tuned by trial-and-error to maximize the score.

The optimization solved for the control sequence over the entire prediction horizon, z_k, where $k = 1, 2, ..., 80$. The whole control sequence was implemented, then at the end of the path, the optimization was run again using the initial conditions as described previously. This continued until the bird hit a pipe or the ground.

3.3 Model Predictive Control

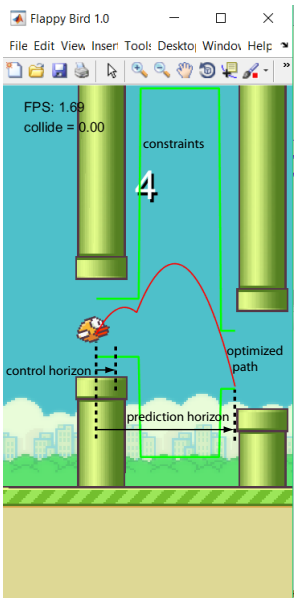


Figure 3.3: Visualization of MPC method

The third method implemented a model predictive control (MPC). The same optimization was used as the previous model except there were no need for the end constraints:

$$\text{Minimize} \quad J = \sum_{k=1}^{n} z_k, \tag{3.12}$$

$$\text{Subject to} \quad Y_{k+1} - Y_k - V_k = 0, \tag{3.13}$$

$$-V_{k+1} + Mz_k \leq 2.5 + M, \tag{3.14}$$

$$V_{k+1} + Mz_k \leq -2.5 + M, \tag{3.15}$$

$$-V_{k+1} + V_k - Mz_k \leq -g, \tag{3.16}$$

$$V_{k+1} - V_k - Mz_k \leq g, \tag{3.17}$$

$$Y_k^{lb} \leq Y_k \leq Y_k^{ub}, \tag{3.18}$$

$$Y_1 = Y_{\text{init}}, \tag{3.19}$$

$$V_1 = V_{\text{init}}, \tag{3.20}$$

The heuristic constraints from the previous method aren't needed because in this method, the optimization is done much more frequently. The result of the optimization is the control inputs for the entire prediction horizon, n. The prediction horizon and control horizon are the only parameters that need to be tuned. The control horizon, p, is the small portion of the control input that is implemented before the next optimization is done, as seen in fig. 3.3. Besides the absence of end constraints, another difference between this method and the previous one is that the MPC only uses a small part of the optimized control sequence, $p < n$, before the next path is optimized. The previous method used the entire control sequence before the next path is solved for, $p = n$.

Chapter 4: RESULTS

4.1 Comparison of Methods

4.1.1 Heuristic Method

The heuristic method is a very simple solution involving only a single if statement. Because of this, it takes negligible time and is implemented very easily. Its performance is not good though because it can only prepare for the first oncoming pipe, unable to set itself up for the next pipe.

4.1.2 Initial Optimization Method

The first method using optimization scores much higher than the previous method. Though, because it uses a mixed integer program to plan a path, it takes time to produce a solution. The time of the solution depends on the size of the mixed integer programming formulation. That includes the size of the prediction horizon and the number of different constraints. To make this method reliable, heuristically tuned constraints had to be included with the constraints that defined the dynamics of the system including end constraints on position and velocity. This method performs well, but it takes a lot of time.

4.1.3 Initial Model Predictive Control

The next method was a model predictive control. This method is listed separately than the main model predictive control because of the way it is implemented. The prediction horizon was set to be 160 and the control horizon set to 80. This is equivalent to planning a path over two pipes and implementing the solution over one pipe. Doing this ensures that the bird will make it over the first pipe and end up in a position that will produce a feasible solution for the next pipe making it unnecessary for the end constraints added in the previous method. Of course, doubling the prediction horizon dramatically increases the time for optimization. This provides the motivation for the next method.

4.1.4 Model Predictive Control

This method is the same model predictive control except that the prediction horizon and control horizon are decreased. Decreasing the prediction horizon alone may affect the reliability of the solution because it won't take into consideration the same amount of constraints. Decreasing the control horizon means that the path will be updated much more often than before so as constraints come into view, the path is updated. Otherwise, the path might not be updated in time to consider the new constraints as they appear. The size of the problem is reduced, but the optimization has to be run more often.

4.2 Results of Methods

4.2.1 Testing Methodology

In the game, the pipes are generated by a random number generator. To test the methods in this paper, the seed of the random number generator was controlled. This ensures that the same set of pipes and gaps are present for a given simulation run. Each method was tested for ten different seeds and the score for each seed was capped at 500. This created 5000 random situations that could be experienced in the game. All the computer simulations were done using a laptop circa 2016, with an Intel Core i7-6500U, 2.5 GHz CPU. A video of the three controllers in action is in reference [5].

4.2.2 Heuristic Method

Table 4.1: Results of Heuristic Method

Seed	Score
1	23
2	135
3	135
4	40
5	41
6	29
7	105
8	19
9	30
10	9
total	566

The results of the heuristic method are seen in table 4.1. This method was the simplest method to implement and its results provided a baseline of performance for other methods. With an average score of 56.6 per game, it scored better than a lot of humans can, but it scored significantly less than the other methods.

4.2.3 Initial Optimization

Table 4.2: Initial Optimization Results

Seed	Score	Avg Opt Time (s)	Score w/ I.G.	Opt time w/ I.G.
1	500	0.1165	387	0.1110
2	500	0.1143	500	0.1166
3	500	0.1086	500	0.1122
4	500	0.1024	500	0.1064
5	500	0.1059	500	0.1127
6	500	0.1040	500	0.1020
7	500	0.1073	500	0.1054
8	500	0.1001	500	0.1053
9	500	0.0999	500	0.1049
10	500	0.1026	500	0.1136
Total/avg	5000	0.1062	4887	0.109

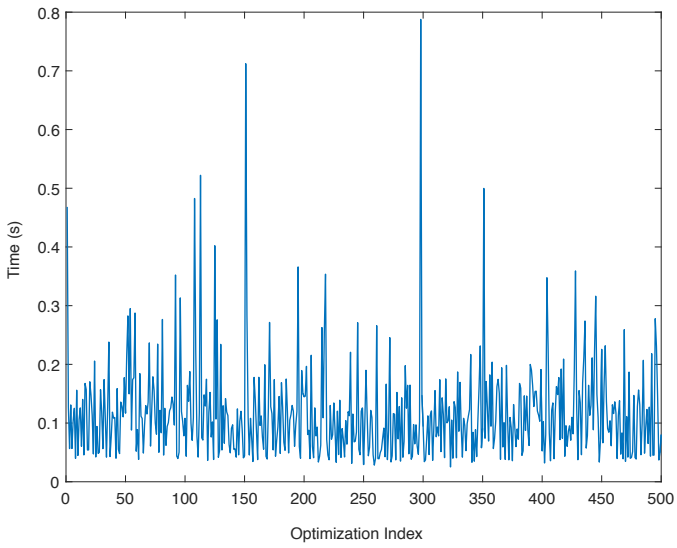


Figure 4.1: Optimization Times

The first optimization method was tested with and without user supplied initial guesses (I.G.). These results are shown in table 4.2. The supplied initial guesses were the results of the previous optimization. Since the bird travels the entire path before it updates its path, the previous result is not related to the current solution, therefore, this did not improve the time for the optimization. The solution without initial conditions made it through all 5000 situations. The times for each optimization were recorded using the *tic* and *toc* commands in MATLAB and are shown for a single run in figure 4.1. While the average optimization time was .1062 seconds, there were a number of optimizations that took significantly longer, spiking up to about 7.5 times the average.

MATLAB Optimization with Zero Cost Function

The cost function was set to zero to see the effects on the optimization time. Using inputs from the first pipe in the game, the previous solution was compared with the zero cost function solution.

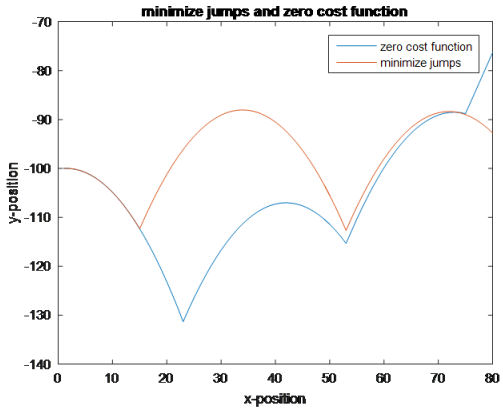


Figure 4.2: Result of one iteration of using a zero cost function

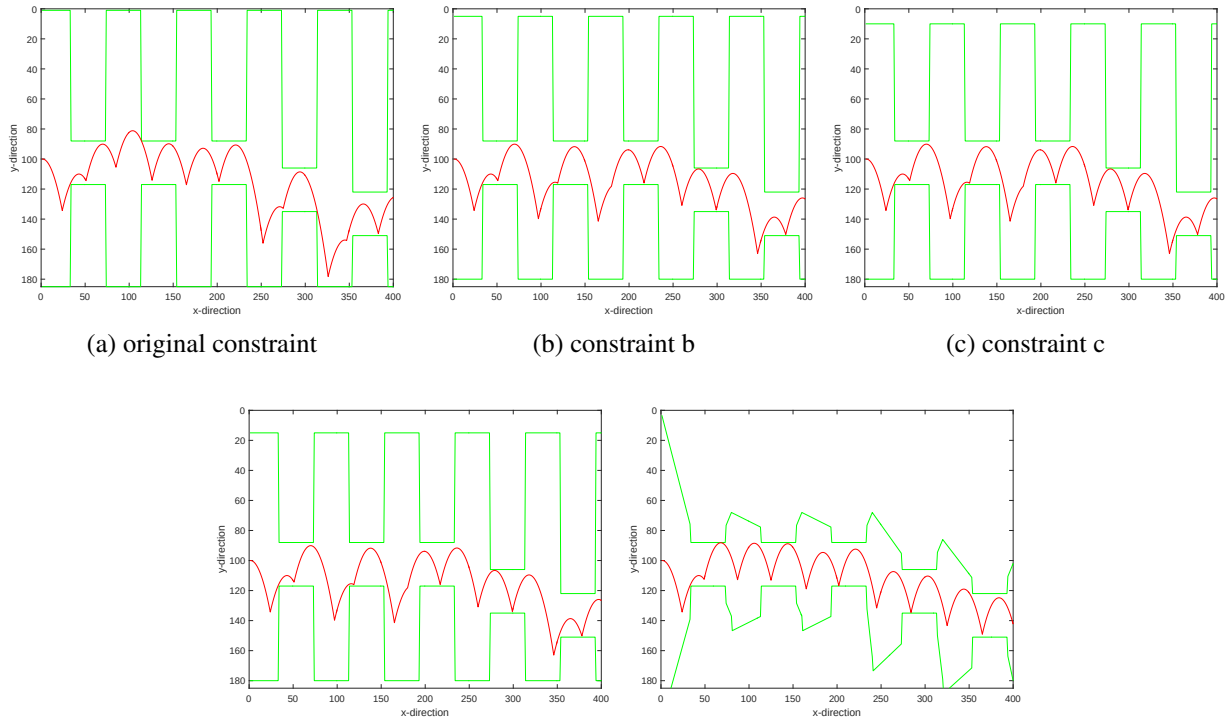
Table 4.3: Comparing cost functions

Method	Time (s)
Minimize jumps	.69
Zero cost	11.47

It can be seen in fig. 4.2 that the solution with the cost function that minimized jumps used less jumps. Though the zero cost solution took a lot more time, table 4.3, both results are feasible solutions. Using a cost function of zero could have caused the branch and bound technique used in the solver [13] to evaluate more branches than was necessary with the previous cost function. When minimizing the number of jumps, the updated bounds on the optimal solution in the branch and bound technique are tighter than when using a cost function of zero.

Tightening Constraints

Figure 4.3: Changing Constraints



(a) original constraint (b) constraint b (c) constraint c

(d) constraint d (e) constraint e

The bounds were tightened on the position of the bird. There were areas where the bird could not feasibly reach, but were included in the bounds of the optimization. This method was an attempt to remove those areas to reduce the optimization time. The changes to the bounds were as follows:

a The first change to the original constraint in figure 4.4b was lowering the ceiling by 5 pixels and raising the floor by 5 pixels.

b The next change in figure 4.4c was to lower the ceiling by 10 pixels and increase the floor by 5 pixels.

c The next change in figure 4.4d was to lower the ceiling by 15 pixels and increase the floor by 5 pixels.

d The next method in figure 4.4e reduced the constraints even more.

Table 4.4: Score with Different Constraints

Method	Score
Original (a)	500+
b	118
c	273
d	144
e	205

Table 4.5: Optimization Times of Different Constraints

Method	Score	Average Time (s)
Original (a)	118	.945
b	118	1.244
c	118	.8825
d	118	.7278
e	118	.9823

Tightening the constraints in this way was not a reliable way of increasing speed. There isn't a clear trend in the results in tables 4.4 and 4.5. By first tightening the constraints just by a little bit, the performance went down a lot. Then by tightening them a little more, the performance increased. By significantly tightening the constraints, the performance was in between the previous methods.

Having the largest bounds is obviously the safest choice, but by tightening the constraints to exclude solutions that are known to not be reasonable, it was thought that the optimization time should decrease. Instead there was no clear trend in the data that would confirm this hypothesis.

One thing that this test did make clear is that there exists multiple solutions that have the same optimal value of the cost function. Changing the parameters of the problem influenced which solution the optimization chose. This is addressed later in section 4.3.

4.2.4 Initial Model Predictive Control

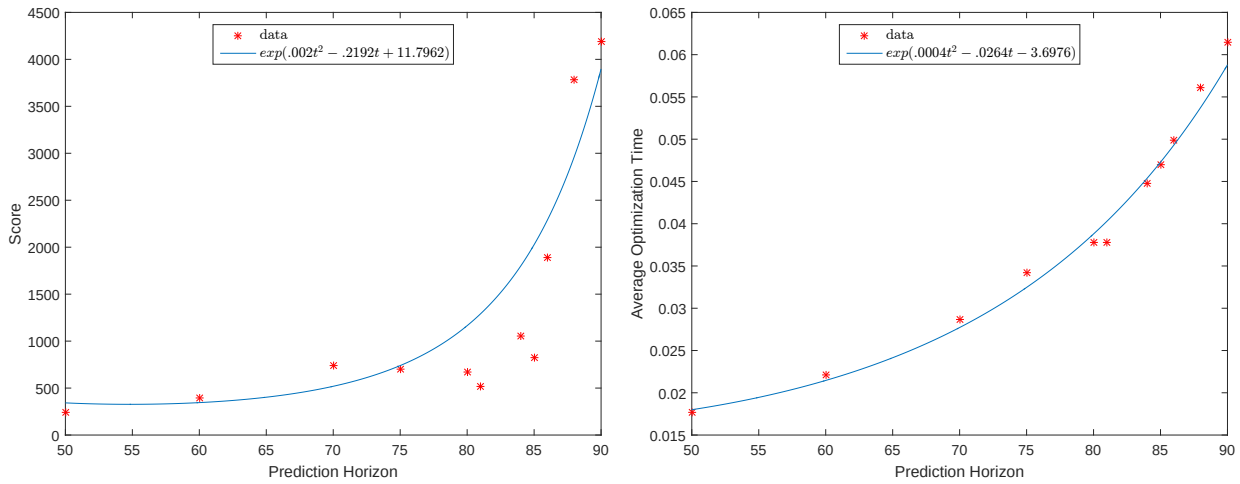
Table 4.6: Initial MPC Results

Seed	Score	Avg Opt Time (s)
1	500	1.7629
2	500	1.7514
3	500	1.7492
4	500	1.7507
5	500	1.7522
6	500	1.7465
7	500	1.7452
8	500	1.7459
9	500	1.7506
10	500	1.7467
Total/avg	5000	1.7501

The original MPC method used a prediction horizon of 160 and a control horizon of 80. This produced reliable results, but the optimization times were much slower than the previous method, as seen in table 4.6.

4.2.5 Model Predictive Control

Figure 4.4: Model Predictive Control Trends



(a) Score vs Prediction Horizon

(b) Optimization Times vs Prediction Horizon

A common control scheme for model predictive controls is to calculate the control inputs for the entire prediction horizon and then only use the first control input before making another calculation. The MPC method used in this paper was tested for multiple prediction horizons while keeping the control horizon constant at 1 time step. Figures 4.5a and 4.5b show that as the prediction horizon increases, the score increases exponentially and the optimization times increase exponentially.

It's easy to see that this method is safer with a longer prediction horizon, but how long is long enough? It is not practical to have a really long prediction horizon because the points close to the end of the path may not affect the performance of the system in its immediate surroundings. It is also impractical to have a really short horizon because the system may encounter a constraint too late to plan a path to avoid it. We have already determined that in this case, a prediction horizon of 160, encompassing two pipes in their entirety, is safe. After testing many different horizon lengths, a horizon of 90 almost matched that performance, and below that, the performance drops dramatically. To see two pipes at all times, a horizon of 135 is required. Since two pipes was determined to be safe, an important quantity to consider is how much space the bird has in front of the first pipe when it has two pipes in sight. This is illustrated in figure 4.5:

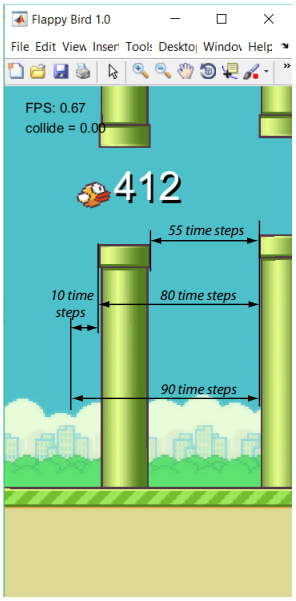


Figure 4.5: Visualization of prediction horizon

The point of seeing the second pipe is to set up the bird to exit the first pipe in a position where it can easily plan a path through the next pipe. The purpose of the extra space in front of the first pipe is to give the bird some time to recover from the previous path which was planned without the second pipe in sight.

Table 4.7: Varying Control Horizon

Control Horizon	Total Score	Avg Opt Time (s)
10	1748	0.0623
4	3413	0.0664
1	4186	0.0615

The MPC was also tested at different control horizons for a prediction horizon of 90. The lower control horizon performed much better, as seen in table 4.7.

In a common MPC the optimization would be run during the implementation of the control horizon to create a smooth controller. One that doesn't pause to make calculations. As long as the sensors of the system can sense one control horizon beyond the prediction horizon, then the optimization for the next prediction horizon can be computed during the current control horizon. One time step in the game is 1/60s. None of the prediction horizons tested have an optimization time less than or equal to this. Setting the control horizon to 4 time steps means the control is implemented for 4/60s, which matches the average optimization time of a prediction horizon of 90. This would satisfy the conditions to implement a smooth controller considering only the average optimization time. As seen in section 4.2.3, optimization times can be much larger than the average, making this very hard to implement.

Table 4.8: MPC with and without Initial Guesses

Seed	Score	Avg Opt Time (s)	Score w/ I.G.	Opt time w/ I.G.
1	500	0.0671	196	0.0603
2	500	0.0641	356	0.0579
3	500	0.0640	500	0.0503
4	500	0.0596	24	0.0641
5	80	0.0550	500	0.0544
6	500	0.0613	500	0.0533
7	500	0.0573	91	0.0541
8	500	0.0680	500	0.0546
9	500	0.0567	395	0.0569
10	106	0.0616	106	0.0545
Total/avg	4186	0.0615	3168	0.056

Initial guesses did not decrease the optimization times of the previous method, but they were expected to decrease the optimization time of the MPC. Because the path is updated much more frequently than the change in constraints, the path estimated from time step to time step does not change very much. If the constraints don't change, then it's easy to imagine that the path at the next time step is the same as the path at the current time step, but starting one time step later. The initial guess supplied to the optimization was the previous solution starting at the second time step and repeating the last time step to keep the length of the vector the same. The results are shown in table 4.8. The average optimization time decreased by 9%.

4.3 Changing Objective Function

In section 4.2.3, the observation was made that there existed multiple solutions to the optimization that had the same optimal value. The solution chosen by the optimization was influenced by changing the parameters of the problem. By studying these results a generalization was made that the best solution for the bird to take was the one where the vertical position of the bird was the lowest. The objective here was to find a way to change the parameters of the problem in a strategic way to influence which of these multiple solutions the optimization chose. To do this, the objective function was changed to include the minimization of the sum of the vertical position of the bird at

each point in the path eq. 4.1:

$$\text{Minimize} \quad J_{\text{new}} = \sum_{k=1}^{n} (z_k - wY_k) \tag{4.1}$$

Where the constant w was chosen to be 10^{-4} because the values of Y_k were much larger than z_k. This significantly increased the score of the previous methods with short prediction horizons. The optimization times for this method increased dramatically though. The results for a prediction horizon of 70, which scored poorly, are shown in table 4.9:

Table 4.9: Comparing New Cost Function to Old Cost Function: Prediction Horizon = 70

Seed	score	Avg. Opt. Time (s)	Score new J	Avg. Opt. Time (s)
1	106	0.0304	250	0.2556
2	89	0.0284	500	0.2306
3	16	0.0275	16	0.1353
4	7	0.0293	43	0.2385
5	44	0.0271	494	0.2193
6	213	0.0285	500	0.2570
7	19	0.0293	108	0.2017
8	22	0.0317	105	0.2152
9	13	0.0264	230	0.2057
10	213	0.0285	12	0.1655
Total/avg	742	0.0287	2258	0.2124

The original cost function only minimized jumps which produced optimal values of 2 or 3. The new cost function produced values much larger and the optimal solutions had a wider range of values. This was the reason the optimization times for the new cost function were much higher. The increase in the number of variables in the cost function introduced a higher number of possible combinations of variables in the solution. The branch and bound technique, used by Gurobi, needed to explore more branches, or make more evaluations, to find an optimal solution.

Chapter 5: FUTURE DIRECTIONS

Although Flappy Bird is a difficult game to score well when played manually, it is certainly possible to beat using tools in control theory. Our main conclusion is that a controller that plans slightly beyond the upcoming pipe performs best on the Flappy Bird game. Further, MPC provides an easy to tune and highly generalizable method model-based control.

The Flappy Bird game is great platform to benchmark and test new control and learning algorithms. Some suggested future work is to use heuristic algorithms like the genetic algorithm and simulated annealing. Also, including the following features for machine learning are vital to have high performance: position of bird relative to the pipe, relative location of consecutive pipes and velocity of the bird. To increase the difficulty of the game, the gap between the vertical pipes and the distance between subsequent pipes can be made to vary spatially as well as temporally as the game proceeds.

BIBLIOGRAPHY

[1] Nguyen, Lan Anh. "Exclusive: Flappy Bird Creator Dong Nguyen Says App 'Gone Forever' Because It Was 'An Addictive Product". Forbes.Com, 2014, 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.

[2] Qin, S.Joe, and Thomas A. Badgwell. "A Survey Of Industrial Model Predictive Control Technology". Control Engineering Practice, vol 11, no. 7, 2003, pp. 733-764. Elsevier BV, doi:10.1016/s0967-0661(02)00186-7.

[3] Zhang, Mingjing. "Roteaugen/Flappybird-For-Matlab - File Exchange - MATLAB Central". Mathworks.Com, 2017, http://www.mathworks.com/matlabcentral/fileexchange/45795-roteaugen-flappybird-for-matlab.

[4] Richards, Arthur, and Jonathan How. "Mixed-Integer Programming For Control". 2005,. Retrieved from http://acl.mit.edu/milp/MILP_for_Control.pdf

[5] Bhounsule, Pranav, and Matthew Piper. "How To Beat Flappy Bird: A Mixed-Integer Model Predictive Control Approach". 2017, https://youtu.be/P5YftCPE4rw.

[6] "Gurobi Optimization - The Best Mathematical Programming Solver". Gurobi.Com, 2017, http://www.gurobi.com/.

[7] Shu, Yi et al. "Obstacles Avoidance With Machine Learning Control Methods In Flappy Birds Setting". Department Of Mechanical Engineering, Stanford Univerisity, 2014,.

[8] Sun, Ludong. "Reinforcement Learning Controlled Flappy Bird". 2014, https://youtu.be/UwfnUNhkcCg.

[9] Ebeling-Rump, Moritz et al. "Applying Q-Learning To Flappy Bird". Department Of Mathematics And Statistics, Queen's University,.

[10] Chen, Kevin. "Deep Reinforcement Learning For Flappy Bird". Stanford University,.

[11] Sun, Ludong. "SVM Controlled Flappy Bird". 2014, https://youtu.be/cYFeI9eFaBY.

[12] Takacs, Balint et al. "Export Of Explicit Model Predictive Control To Python". IEEE, 2015 International Conference On Process Control (PC), 2015, pp. 78-83.

[13] "Mixed-Integer Programming (MIP) Basics | Gurobi". Gurobi.Com, 2017, http://www.gurobi.com/resources/getting-started/mip-basics.

VITA

Matthew Piper is from Dallas, Texas. He attended The University of Texas at Austin and received his Bachelor's degree in Aerospace Engineering then moved to San Antonio to pursue his Master's degree in Mechanical Engineering at The University of Texas at San Antonio. His future plan is to find a job in the aerospace industry.