Proceedings of the ASME 2017 Dynamic Systems and Control Conference
DSCC 2017
October 11-13, 2017, Tysons, Virginia, USA

DSCC2017-5285

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

Matthew Piper[α], Pranav A. Bhounsule[α†], and Krystel K. Castillo-Villar[β]
[α] Robotics and Motion Laboratory
[β] Manufacturing Systems and Automation Laboratory
Department of Mechanical Engineering,
The University of Texas at San Antonio. 1 UTSA Circle, San Antonio, TX 78249
Email: matthewpiper@utexas.edu, pranav.bhounsule@utsa.edu, krystel.castillo@utsa.edu

ABSTRACT

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 tubes; (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 tubes 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.

1 INTRODUCTION

Flappy Bird was a popular mobile game released in May 2013 [2]. It consists of a bird flying horizontally at a constant speed but falling continuously under gravity. The bird can be made to fly upwards 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 Fig. 1. The horizontal distance between two consequent pair 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 is over if the bird hits either pipe or the floor.

Although the game looks overly simple, it is substantially difficult game to score high number of points. Our experience has indicated scores in the range of 0 to 6 after sufficiently long practice sessions. Notwithstanding, it was the most downloaded game on iTunes by end of January 2014 and it's developer, Dong Nguyen, has said that he was earning $50,000 a day through in-app advertisements and sales [2]. Finally, in February 2014, Dong pulled the game off the iTunes store due to the addictive nature of the game. Since then other developers have recreated Flappy Bird and released them on the web. In this paper, we

*THE PHRASE "HOW TO BEAT FLAPPY BIRD" IS INSPIRED FROM A HILARIOUS YOUTUBE VIDEO [1]

†Address all correspondence to this author.

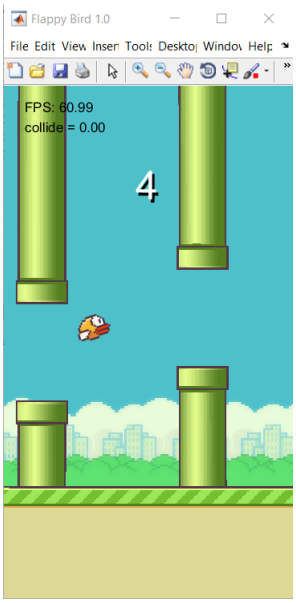

FIGURE 1: Flappy bird game

use an openly available MATLAB® version of the Flappy Bird, which preserves the features and difficulty of the original game.

The main contribution of the paper is the formulation and solution of the Flappy Bird game as a linear, mixed-integer model predictive control (MPC) problem. A secondary contribution is to compare the MPC with two controllers: a manually tuned controller and an optimization-based controller with heuristically tuned constraints.

2 BACKGROUND AND RELATED WORK

Past works on automatic control of Flappy Bird have extensively focussed on using machine learning algorithms. Shu et al. [3] used reinforcement learning (RL) and were able to achieve scores of around 1500 [4]. The states for learning in their formulation were the $x-$, $y-$ position of the bird relative to the top of the upcoming bottom pipe and the bird velocity, and the action was to flap or to not flap. RL optimizes state-action pairs and transition probabilities to maximize a function.

Ebeling-Rust et al. [5] and Chen [6] used Q-learning, a variant of RL, but were only able to score around 200. We suspect that the modest scores were because the states used for learning were limited to the $x-$, $y-$ position of the bird relative to the upcoming pipe while ignoring the bird velocity.

Shu et al. [3] also used Support Vector Machine (SVM), a supervised learning algorithm, to achieve scores of around 1200 [7]. 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 $x-$, $y-$ 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. Because of the curse of dimensionality, either the features have to be kept small or the grid size has to be coarse. 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 easily available. This motivates the use of model-based control algorithms. Takács et al. [8] used explicit model predictive control (explicit MPC) on flappy bird but have not reported their results. Explicit MPC solves the optimization problem offline and stores the solution. During implementation, the stored policy is searched and interpolated as needed to determine the necessary control actions.

In this paper, we use online model predictive control (MPC) with a tunable prediction and control horizon. The problem is formulated as a mixed-integer linear programing problem and solved using the optimization software, Gurobi [9]. We do parametric studies to understand how the prediction horizon, the control horizon, and the cost function affects the computation speed and performance of the algorithm. We also compare the MPC with two other controllers, a manual controller and an optimization-based controller with heuristically tuned optimization constraints.

3 FLAPPY BIRD MODEL

A MATLAB® version of the game was obtained from Mathworks file exchange [10]. The game is played manually by using the spacebar key. Pressing the spacebar sets the velocity to a fixed value in the upward direction. Otherwise, the bird is acted upon by gravity in the downward direction that increases the velocity by a fixed amount every step in the downward direction. The dynamics of the bird can be written as follows:

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

$$V_{k+1} = \begin{cases} -2.5, & z_k = 1 \\ V_k + g, & z_k = 0. \end{cases} \tag{2}$$

where k is the time index, Y_k is the height of the middle of the bird at time k (the y-axis points downward), V_k is the bird's velocity at time k, gravity is $g = 0.1356$, z_k is a binary decision variable, where $z_k = 1$ indicates bird flaps to move up and $z_k = 0$ indicates that the bird does nothing. Note that Y_k and V_k is in unit 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. We modified the downloaded game by replacing the spacebar input with the binary decision variable, z_k.

4 METHODS

4.1 Manual controller with heuristic tuning

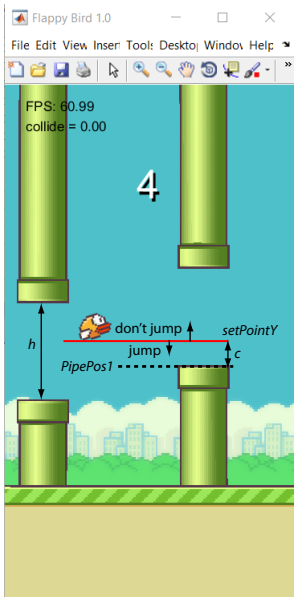

FIGURE 2: Visualization of the manual controller.

We defined a controller structure by observing our own technique while playing the game manually. Our controller structure is simple: flap the bird if it's altitude is below a set-point (a fixed distance above the top of the upcoming bottom pipe). The controller logic is shown in Fig. 2 and defined by the following pseudo-code:

$$\begin{aligned} if \quad & Y_k < setPointY, \;\; z_k = 1, \\ & else, \;\; z_k = 0, \end{aligned}$$

where the $setPointY = PipePos1 + c$ where $PipePos1$ is the position of the top of the bottom pipe and the only tuning parameter is c. We manually tuned c to 10. Note that the distance between the pipes is $h = 48$. That is, the optimal c is biased to be closer to the bottom pipe. The reason for this was that the physics of the game allows the bird to jump 2.5 pixels up but can descend only $V_k + 0.1356$ in a single integration step (see Eqn. 2).

4.2 Optimization-based control with heuristic tuning of terminal constraints

In our next method, we formulate and solve the flappy bird problem using mixed-integer linear optimization program. The problem is defined as follows: Given the initial state Y_{init} and V_{init} and the model of the bird motion, the optimization problem is to

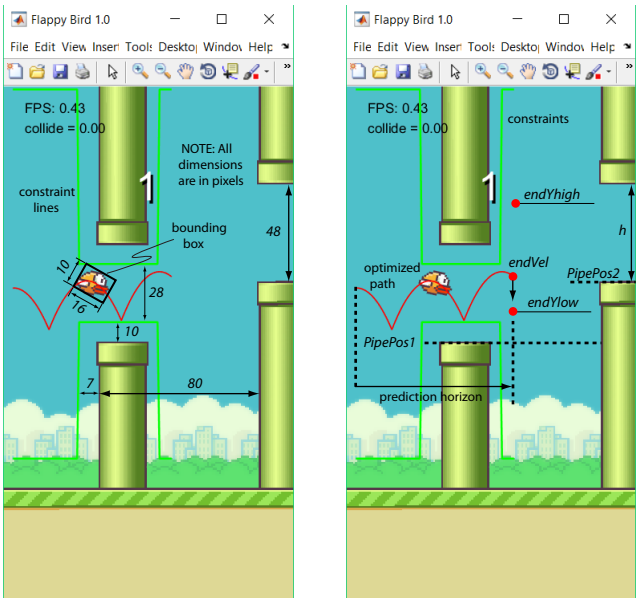

FIGURE 3: Constraints for the optimization

minimize the number of flaps over $k = 1, 2, \ldots, n$ time steps, subject to path constraints (pipe locations) and heuristically tuned end of path constraints, Y_n and V_n. The mathematical details of the optimization problem are given below:

$$\text{Minimize} \quad J = \sum_{k=1}^{n} z_k, \tag{3}$$

$$\begin{aligned}
\text{Subject to} \quad & Y_{k+1} - Y_k - V_k = 0, && (4)\\
& -V_{k+1} + Mz_k \leq 2.5 + M, && (5)\\
& V_{k+1} + Mz_k \leq -2.5 + M, && (6)\\
& -V_{k+1} + V_k - Mz_k \leq -g, && (7)\\
& V_{k+1} - V_k - Mz_k \leq g, && (8)\\
& Y_k^{lb} \leq Y_k \leq Y_k^{ub}, && (9)\\
& Y_1 = Y_{\text{init}}, && (10)\\
& V_1 = V_{\text{init}}, && (11)\\
& endYlow \leq Y_n \leq endYhigh, && (12)\\
& V_n \leq endVel \quad \text{if} \quad PipePos2 > PipePos1. && (13)
\end{aligned}$$

Equations 5 - 8 are a reformulation of Eqn. 2 using the Big-M method [11]. The Big-M method is a technique of writing switching equations, such as Eqn. 2, into multiple equations using a large number M. The value of M is large enough (we used $M = 500$) that the results of the optimization do not depend on its specific value. That is, different M's near our chosen value give the same optimization results. Big-M method is used in order to keep the constraint equations linear in the binary deci-

sion variable (in our case z_k). By keeping the constraints linear, we are able to use computationally efficient linear mixed-integer programming software such as MATLAB® *intlinprog* and Gurobi [9]. The constraints posed by the game window (bird should not go out of the screen) and pipe edges are enforced at each time step using lower bound Y_k^{lb} and upper bound Y_k^{ub}. These bounds were set after taking into account bird dimensions (see Fig. 3). This is because, although the trajectory is specified by the mid-point of the bird, collision checking is done by using the bounding box as shown in the figure.

The heuristically tuned variables are described next. The time span for optimization n was tuned to 80. This window corresponds to the horizontal distance between consecutive pipes. The position variable $endYlow = 0.5(PipePos1 + PipePos2)$, constrains the terminal position of the bird to be halfway between the top edge of consecutive bottom pipes (see Fig. 3). The position variable $endYhigh = (PipePos2 + h) - 0.5|PipePos2 - PipePos1|$, provides an upper bound on the end position. The velocity variable $endVel = 1.1$ (positive velocity is downward) and is active only if the following pipe is lower than the current pipe as indicated by Eqn. 13 (note that positive direction is downwards). These parameters were tuned by trial-and-error to maximize the score.

The execution of the algorithm is as follows. The optimization plans the control action to flap or not flap for time indices, $k = 1, 2, ..., n$ (where n is fixed ahead of time). Then the bird executes the entire control sequence followed by the optimization and execution for the next n time steps. The process repeats until the bird hits the floor or a tube.

4.3 Model predictive control (MPC)

The third method used model predictive control (MPC). In MPC, an optimization problem is formulated and solved over a prediction horizon (say n). Only a part of the controller policy is implemented (say $p < n$) and the rest is discarded. The process of prediction and control is repeated until a termination criteria is achieved (e.g., end of game).

The prediction horizon was n and the control horizon was fixed at $p = 1$, unless noted otherwise (see Fig. 4). The optimal control problem formulation over the horizon n is the same as that in Sec. 4.2 but without the heuristic constraints given by Eqns. 11 and 12. We do not need these constraints because the controller planning is done frequently. The only parameter in the MPC is the prediction horizon, n. A small value of n allows quick planning, but has higher chance of failing through constraint violation (e.g., the bird does not have enough time to avoid the pipe). A big value of n produces better results but requires more computations. We tune n by simply increasing its value until the controller is able to achieve a high score. Except for the prediction horizon n, an easy parameter to tune, there are no parameters that need to be heuristically chosen.

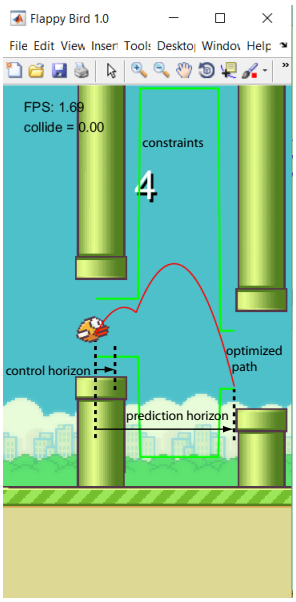

FIGURE 4: Visualization of MPC method

Note that the main difference between the MPC and optimization-based controller is that the MPC plans n control steps but executes only the first p steps while in optimization-based control the planning and execution steps are identical, that is $n = p$.

5 RESULTS

In order to compare all methods objectively we simulated 10 games, each game was initialized with a random number generator. This ensures that the same set of pipes and gaps are present for the given simulation run. All the computer simulations were done using a laptop circa 2016, Intel Core i7-6500U, 2.5 GHz CPU. A video of the three controllers in action is in reference [12].

Table 1 shows results for the manual controller with heuristic tuning of control parameters (Sec. 4.1). Since the controller decision to flap or not flap is based on simple limit checking, the computations for this controller are negligible. However, it is only able to score an average score of 56.6 points per game. While this is significantly higher than what we were able to do by playing the game manually (roughly $0-6$ points per game), the next two methods show a substantial improvement.

Table 2 shows results for optimization with heuristic tuning of constraints (Sec. 4.2). The optimization window was the distance between two consecutive pairs of pipes or $n = 80$ and we had to heuristically set the terminal constraints of position and velocity to ensure that the optimization worked well. In all the runs, the bird went past the 500th pipe (this was our termination criteria). Thus the average score was 500 (maximum possible

TABLE 1: Scores for manual controller. Average score was 56.6. The time for computing the control command was negligible.

Run no.	Score
1	23
2	135
3	135
4	40
5	41
6	29
7	105
8	19
9	30
10	9

TABLE 2: Results for optimization with heuristic tuning of constraints. The average score was 500 (indicating no failure). The average optimization time is per iteration. The maximum optimization time is the longest time needed for a single optimization across all iterations and runs.

Run no.	Score	Avg. opt. time	Max. opt. time
1	500	0.1165	0.78766
2	500	0.1143	0.6863
3	500	0.1086	0.8101
4	500	0.1024	0.5053
5	500	0.1059	0.6243
6	500	0.1040	0.5273
7	500	0.1073	1.2596
8	500	0.1001	0.5508
9	500	0.0999	0.4946
10	500	0.1026	0.5173

score as per our termination criteria), the average optimization time per iteration was 0.1062 s and the maximum optimization time ever needed for an iteration was 1.2596 s, about 12 times the average optimization time. This was obtained by using the MATLAB® command *tic* and *toc*. Figure 5 shows the optimization time for a single run as a function of optimization index, k.

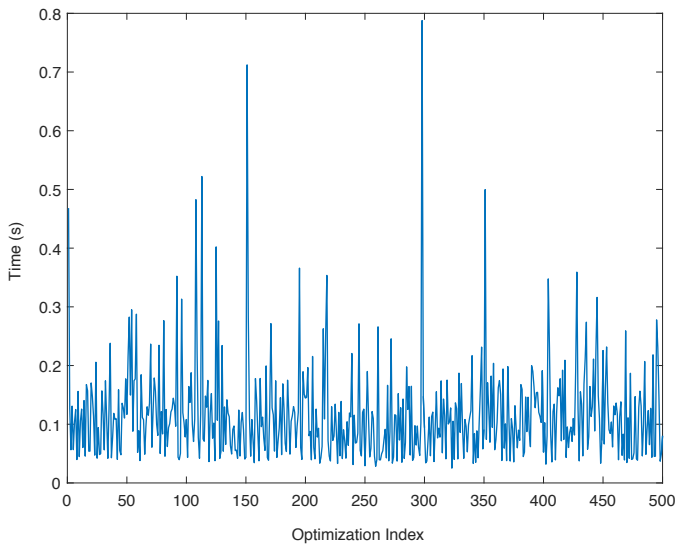

FIGURE 5: Optimization time per iteration versus optimization index per pipe for a single simulation run (that is 500 pipes). For each pipe, the optimization computes optimal solution for over time index $k = 1, 2, ..., n$.

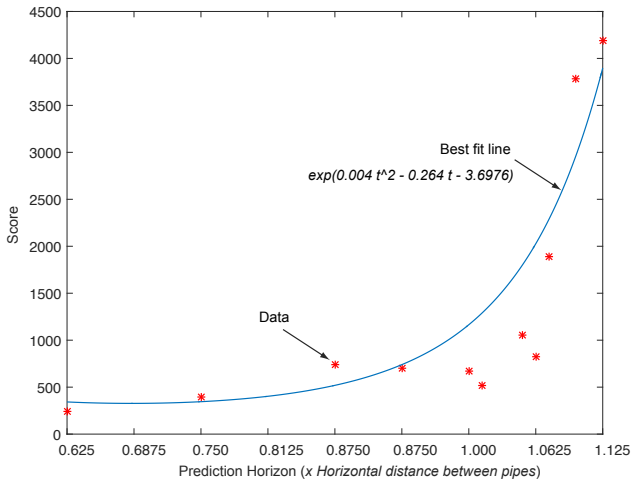

FIGURE 6: Total score versus prediction horizon for the MPC. The prediction horizon is expressed in terms of a factor of the horizontal distance between consecutive pipes

Note that the maximum time peaks to about 0.5 at least 4 times in the optimization.

Next, we show results for the MPC. First, we needed to get the optimum prediction horizon, n. We ran simulations for different prediction horizons, n, and plotted the scores and average optimization times, each as a function of n as shown in Figs. 6

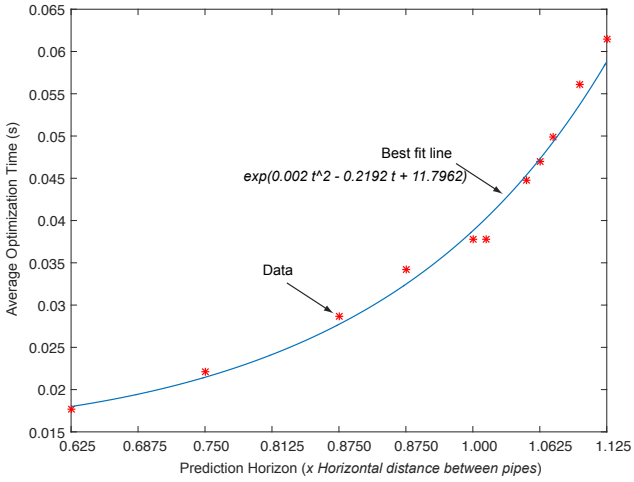

FIGURE 7: Average optimization time versus prediction horizon for the MPC.

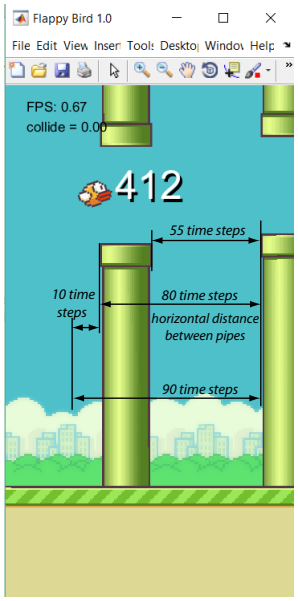

FIGURE 8: Visualization of optimal prediction horizon of 90 ($\sim$ 1.125 horizontal distance between consecutive pipes) that gives high scores with moderate optimization time.

and 7. It can be seen that as the prediction horizon increases, the score as well as average optimization time increases. We found that a prediction horizon of 90 or 1.125 times the horizontal distance between consecutive pipes gives high scores and we used this value to test the MPC. Since the distance between pipes is 80, the optimum of 90 indicates that the bird needs to plan the control policy slightly beyond 1 pipe to be successful. Thus,

TABLE 3: Results for the MPC with prediction horizon of 90 and control horizon of 1. The average score was 418.6 (maximum was capped at 500). The average optimization time below is per iteration. The maximum optimization time is the longest time needed for the MPC to compute a control policy at a given iteration observed across all iterations and runs.

Run no.	Score	Avg. opt. time	Max. opt time
1	500	0.0671	3.591
2	500	0.0641	3.059
3	500	0.0640	2.8842
4	500	0.0596	2.9635
5	80	0.0550	0.9668
6	500	0.0613	2.9596
7	500	0.0573	3.0536
8	500	0.0680	3.8055
9	500	0.0567	1.9173
10	106	0.0616	1.7036

when the bird is 10 time indices away from the upcoming pipe, it is able to see and plan for two pipes as shown in Fig. 8. A planning horizon less than 80 means that the bird cannot 'see' the beginning of the following pipe before it reaches the current pipe. This is an issue if the gap between next pair of pipes is too high or too low, which will not leave enough time for the bird to change its altitude. Since the distance between the trailing edge of the current pipe and the leading edge of the next pipe is 55, a planning window less than this distance will lead to poor results as seen in Fig. 7.

Table 3 shows the optimization results for 10 simulations runs with prediction horizon of 90 and control horizon of 1. The average score of 418.6 was better than the manual controller but was slightly worse than the score of 500 (perfect score) obtained using optimization with heuristically tuned constraints. The average time for the MPC was 0.0623 but the maximum time at a particular iteration was 3.8055, about 60 times the average time.

Table 4 compares different control horizons, p, for the same prediction horizon, $n = 90$. It can be seen that while the average time remains almost the same the average score increases with a decrease in p.

Figure 9 shows how solutions change when the inactive constraints (ceiling and floor) are changed by a few pixels. Small changes in inactive constraints lead to different solutions with the same cost. This indicates that there are multiple solutions to the optimization problem.

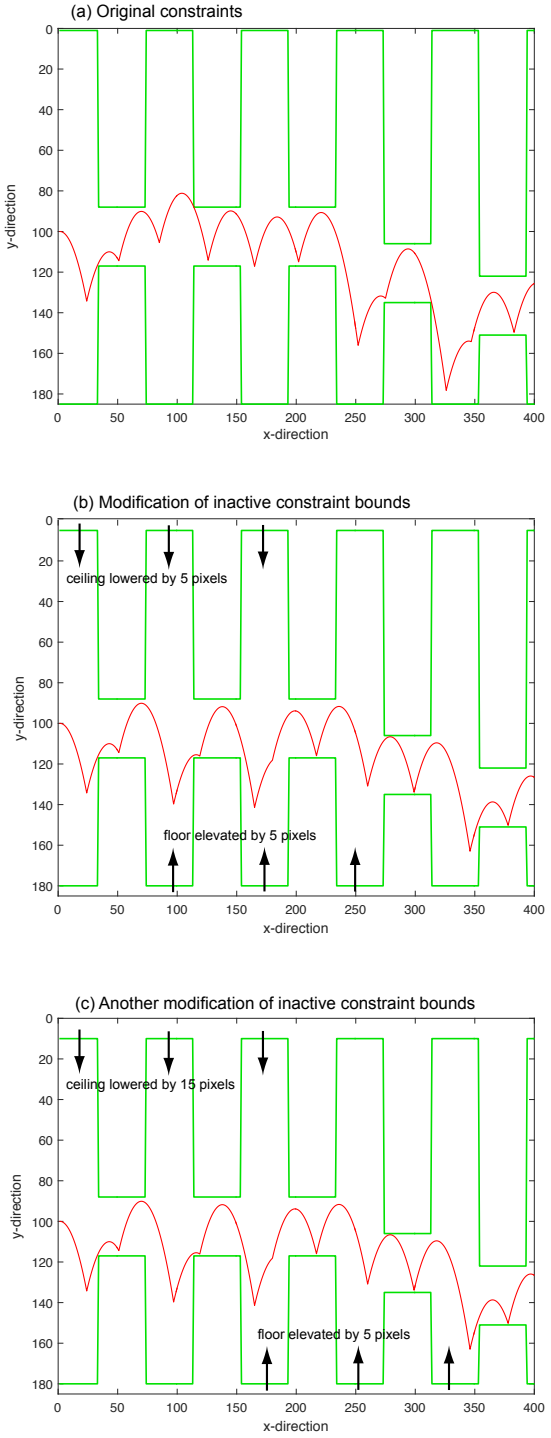

FIGURE 9: the MPC produces different solutions but with the same cost for slight changes in inactive constraints.

TABLE 4: the MPC with different control horizon but same prediction horizon of 90. The scores are averaged over 10 simulation runs.

Control horizon	Avg. score	Avg. opt. time
10	174.8	0.0623
4	341.3	0.0664
1	418.6	0.0615

Our original cost function minimized of the number of hops (see J in Eqn. 3). We modified the cost to be the sum of the number of hops (z_k) and the vertical height of the bird (Y_k) to see if it has any effect on scores and optimization time. Since Y_k is increasing downwards, the new cost biases the solution to choose a path that is closer to the lower constraint, Y_k^{lb}. The rationale in choosing the lower bound is due to the way in which decision variable z_k affects the velocity term in Eq. 2; a flapping action ($z_k = 1$) causes the bird to move upwards with velocity of 2.5, while no flapping ($z_k = 0$) will increase the velocity towards the lower tube by relatively small amount given by $g = 0.1356$. The new cost can be written as follows

$$\text{Minimize} \quad J_{\text{new}} = \sum_{k=1}^{n} (z_k - wY_k) \tag{14}$$

where w was a suitable normalizing weight that accounts for different scales of z_k and Y_k. We chose $w = 10^{-4}$.

Table 5 compares our original cost J with new cost J_{new} for a non-optimal prediction horizon of 70. It can be seen that the average score for the new cost is about 3 times higher than the original cost. However, the increase in score comes at the cost of increasing the maximum optimization times by three orders of magnitude on average. The increase in computation time is because there are more combinations of variables due to the new cost function, thus the branch and bound method in the mixed-integer algorithm needs to evaluate solutions on more branches.

6 DISCUSSION

We compared three control algorithms for the popular Flappy Bird game: (1) A manual controller that turns flapping ON if the bird is below a user tuned threshold height which scored an average of 56.6 points; (2) an optimization-based controller that automatically plans between consecutive pipes, but is heuristically tuned to have appropriate terminal conditions (position and velocity) which scored a perfect 500 (10 trials, each capped at 500 pipes); (3) an MPC controller with an optimized

TABLE 5: Comparing original cost (number of hops) to new cost (sum of hops and deviation from reference trajectory) for a prediction horizon of 70. The average score for J and J_{new} is 74.2 and 225.8 respectively.

Seed	Score J	Max opt. time	Score J_{new}	Max opt. time
1	106	0.51535	250	46.2670
2	89	1.1611	500	56.3510
3	16	0.13513	16	1.9357
4	7	0.46548	43	29.7770
5	44	0.48417	494	24.2550
6	213	0.49736	500	48.6140
7	19	0.13135	108	15.5830
8	22	0.49177	105	15.3490
9	13	0.12185	230	18.6690
10	213	1.0791	12	2.3694

prediction horizon which an average score of 418.6 (10 trials, each capped at 500 pipes). The maximum scores ever recorded with these three controllers, with game initialized using the same random number, in the order they are described are 23, 6473, and 3961 (see video [12]). Average computation time per iteration for the optimization-based controller and the MPC were comparable, but the maximum computation ever observed in the MPC was about 3 times that of the optimization-based controller.

These results indicate that the optimization-based controller with heuristic constraint tuning outperforms the MPC with respect to highest score achieved, average score achieved for same initial conditions and terrain, and in terms of computation times. However, this was only possible because of heuristic tuning of the terminal constraints (see Eqns. 12 and 13). We found that a slight change in these constraints led to significant degradation of performance. These constraints were arrived upon by human intuition and by trial-and-error. This makes it hard to generalize this approach. On the other hand, the MPC has only two free parameters, the prediction and control horizon, and are relatively easy to tune (see Fig. 7). Thus, the MPC is straight-forward to tune and easy to generalize.

Past approaches have used machine learning to play the game. One choice has been reinforcement learning which works well only after large number of learning trials. Another methods has been support vector machine, a supervisory learning method, which relies on training data from humans to enable learning. One issue with the latter approach is that good training data is hard to acquire because humans are not able to play this game that well. Learning is normally used when models are not available. However, for Flappy Bird, the model is simply that of an object falling under gravity but with a velocity reset when the bird is made to flap. When models are available, model-based control can provide quick and superior performance as we have shown here.

What are the key features/terrain information the controller should use to maximize scores? The manually tuned controller based its control on the vertical distance from the bottom pipe but it scored a measly 56 on average. The learning-based controllers in [5, 6] used the x- and y-position relative to the upcoming pipe leading to scores of around 200 points (we are assuming their game had similar difficulty as ours). Besides the x- and y-position, [3] considered velocity of the bird which increased their scores to 1600. The curse of dimensionality prevented these studies from looking beyond a single pipe. In both of our optimization-based controllers, we found that planning slightly beyond the upcoming pipe increases the scores further (our highest score was 6473). The reason is that planning slightly beyond the upcoming pipe allows the bird to set up good initial conditions to successfully navigate the next gap. This is usually important when there is a dramatic drop or elevation in the height of the gap between consecutive pipes.

The biggest disadvantage of the two optimization methods is the long computation time. For any online optimization method to succeed, the worst case time should be less than the control bandwidth. Our best computationally efficient controller is the optimization with heuristic constraints and had a worst case time of 1.29 sec which is about 77 times the control bandwidth of $1/60$ sec or 0.017 sec. If we had strictly enforced the $1/60$ sec time limit in our optimizations then we would have gotten far inferior performance. This problem can be alleviated by improving the optimization algorithm, supplying better initial conditions, and/or using a faster computer for the online optimization.

7 CONCLUSION AND FUTURE WORK

Although Flappy Bird is a difficult game to beat (implies score well) when played manually, it is certainly possible to beat it 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. Furthermore, the MPC provides an easy to tune and highly generalizable method.

The Flappy Bird game is a great platform to benchmark and test new control and learning algorithms. Some suggested future work is to use heuristic algorithms like genetic algorithm and simulated annealing. Also, including the following features 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.

MULTIMEDIA

1. A video of the three controllers is available on YouTube [12].
2. MATLAB code is available on github [13].

REFERENCES

[1] How to beat flappy bird (best method). https://youtu.be/gD-nzHy2DdU. Accessed: April 12, 2017.

[2] Flappy bird. https://en.wikipedia.org/wiki/Flappy_Bird. Accessed: April 10, 2017.

[3] Shu, Y., Sun, L., Yan, M., and Zhu, Z., 2014. Obstacles avoidance with machine learning control methods in flappy birds setting. Department of Mechanical Engineering, Stanford University.

[4] Reinforcement learning controlled flappy bird. https://youtu.be/UwfnUNhkcCg. Accessed: April 22, 2017.

[5] Ebeling-Rump, M., Kao, M., and Hervieux-Moore, Z. Applying q-learning to flappy bird.

[6] Chen, K., 2015. Deep reinforcement learning for flappy bird.

[7] Svm controlled flappy bird. https://youtu.be/cYFeI9eFaBY. Accessed: April 22, 2017.

[8] Takács, B., Holaza, J., Števek, J., and Kvasnica, M., 2015. "Export of explicit model predictive control to python". In Process Control (PC), 2015 20th International Conference on, IEEE, pp. 78–83.

[9] Gurobi optimization. http://www.gurobi.com/. Accessed: April 10, 2017.

[10] Zhang, M., Accessed: April 7, 2017. Roteaugen/flappybird-for-matlab. http://www.mathworks.com/matlabcentral/fileexchange/45795-roteaugen-flappybird-for-matlab.

[11] How, R. J., Accessed: April 10, 2017. A mixed-integer programming for controls [powerpoint slides]. http://acl.mit.edu/milp/MILP_for_Control.pdf.

[12] How to beat flappy bird: A mixed-integer model predictive control approach. https://youtu.be/P5YftCPE4rw. Accessed: April 22, 2017.

[13] Automatic control of flappy bird, matlab code. https://github.com/pab47/FlappyBirdController. Accessed: June 20, 2017.