

PID: Flight Control

Personal Project

ABSTRACT

Implementing and tuning a PID-controller in an App built using MIT Appinventor. The controller enables a plane to land on a runway in different scenarios, including wind and limited manoeuvring speed. The PID-controller is compared to the BangBang algorithm, which shows that the BangBang algorithm is almost as well performing, but less realistic than the PID-controller. Additionally, there was a not fully understood problem with the PID-controller which resulted in shaky movements when heavy weights for the derivative term are used.

MARCO B. GABRIEL

April 2022

:



UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Process Control
Department of Informatics
University of Fribourg (Switzerland)

Table of Contents

1. Introduction	2
1.1. Introduction	2
1.2. State Of The Art	2
2. Flight Control	4
2.1. Specific Setting	4
2.2. Solution	5
2.2.1. Implementation	6
2.2.2. PID Tuning	9
2.3. Result	11
3. Discussion	13
3.1. Discussion	13
3.2. Improvements	13
4. Conclusion	14
A. License of the Documentation	15
References	16
Referenced Web Resources	16

List of Figures

2.1. Screenshot of the App, showing the simulation screen	5
2.2. The code blocks for the PID-controller	7
2.3. Screenshot of the settings including my additions	8
2.4. Overview of technologies used and the flow direction of information . . .	8
2.5. Look-up table for PID-controller tuning with the Ziegler-Nichols method, where T_c = oscillation period and K_c = minimal value for K_p where oscil- lation starts.	9
2.6. Plot of the error with $P=0.05$ $I=0$ $D=0$	9
2.7. Plot of the error with $P=50$ $I=0$ $D=0$	10
2.8. Plot of the error with $P=0.04$ $I=0.0001$ $D=-300$	10
2.9. Plot of the error and banking angle for $P=0.3$ $I=0.002$ $D=-3000$	11
2.10. Plot of the error and banking angle of the Fighter-Jet mode	12
2.11. Plot of the error and banking angle of the Drink mode	12
2.12. Plot of the error and banking angle of the Normal mode with instant runway relocation	12

List of Tables

2.1.	The available parameters to tune	5
2.2.	The BangBang logic	6
2.3.	The values for the different modes	11

1

Introduction

1.1. Introduction	2
1.2. State Of The Art	2

1.1. Introduction

As part of the Process Control 2022 course[7], I am tasked with creating a PID-controller which acts as the flight-controller of a simulated aircraft. The simulation is built in an App using the MIT Appinventor[5]. The task is to implement the code needed for the PID functionality and find suitable PID values for three different scenarios. These scenarios are:

- a) The plane is required to stay on course as much as possible (Fighter-Jet)
- b) The plane carries passengers holding a drink, that should not spill (Drink)
- c) The plane carries passengers that fastened their seat-belts and want to land on the runway (Normal)

Furthermore, I should compare the PID-controller with the BangBang-controller and explain how I found adequate values for the PID-controller.

1.2. State Of The Art

Controlling processes is an important field in the industry. Not only does the manufacturing of goods require clever controlling, some products itself are also in need for efficient algorithms to control various procedures. For example, in the context of the production of aluminium, the controlling algorithm must operate the heating such that there is a constant temperature despite the addition of new aluminium, which cools down the oven. Examples for products that include some form of controllers are the assistant systems of modern cars (lane-keeping) or the air conditioning (temperature). Both of these systems need to compete against uncontrollable interferences like curves in the road or the shining hot sun respectively. In such systems, the exact dynamics are unknown.

Already a hundred years ago, Nicolas Minorsky found out, that the applied actions to control such a system should be calculated by the following formula:

$$action = weight_1 * error(t) + weight_2 * \int_0^t error(t)dt + weight_3 * \frac{d}{dt}error$$

which represents the core of every PID-controller.[1] Subsequently, this formula was tuned until near perfection. Nowadays, there already exist very well tuned PID-controllers for many applications. It is no surprise that PID-controllers are still relevant today and used in various industries.[3]

2

Flight Control

2.1. Specific Setting	4
2.2. Solution	5
2.2.1. Implementation	6
2.2.2. PID Tuning	9
2.3. Result	11

2.1. Specific Setting

The bare bone of the App is already given. It consist of two screens, the simulation and the settings. In the simulation, a plane is steered by either BangBang or the PID-controller and attempts to land on a runway. The plane can only land if it stays within a specific tolerance of the center of the runway. The position of the runway can be changed by touching on the screen. Tilting the phone to one side (left or right) introduces wind, which pushes the plane off it's course to one side. More tilt equals more wind. The described situation is shown in Figure 2.1. The velocity, how fast the aircraft can correct it's course is limited by the drag parameter. Lower drag means the plane can fly faster, however it has no effect on the plane's acceleration.

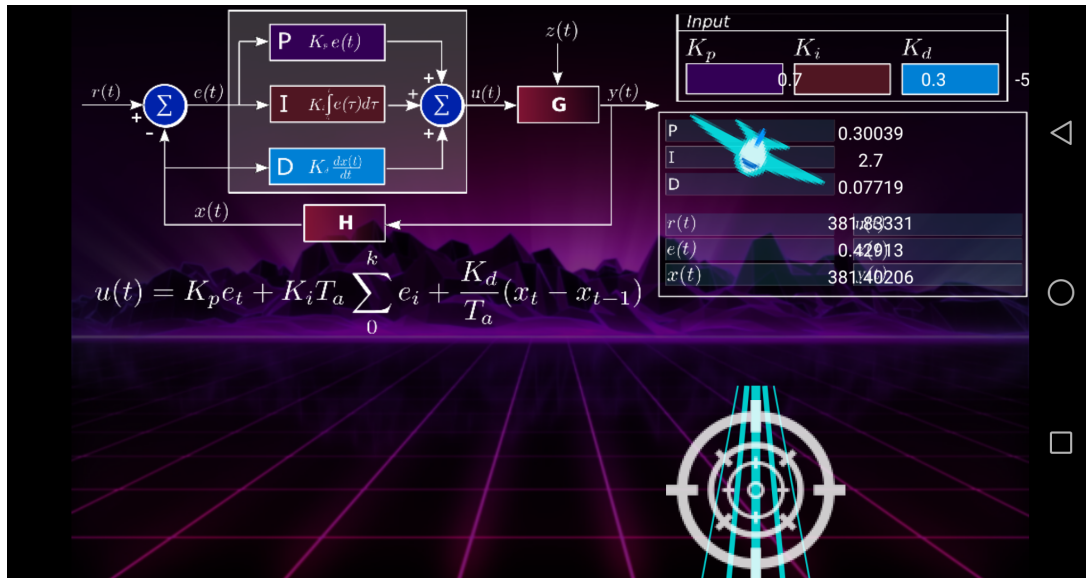


Fig. 2.1.: Screenshot of the App, showing the simulation screen

In the settings menu, it is possible to change various parameters used in the simulation. These are listed in Table 2.1. I have to find values for these parameters that do not cause jumpy corrections when the runway location changes suddenly and do not lead to a 'locking' effect when the wind is too strong for a prolonged period of time [7].

Parameter	Description
Kp	Weight of the proportional error
Ki	Weight of the integral of the error
Kd	Weight of the error's derivative
Drag	The drag the aircraft experiences, lower means higher maximum velocity
Acceptable Error	The horizontal offset the plane is allowed to have to continue landing

Tab. 2.1.: The available parameters to tune

Now only one, but three sets of these values have to be found to satisfy following three scenarios[7]:

- a) Fighter-Jet: Always stay on course (with a very agile aircraft)
- b) Drink: No sudden moves (because a passenger is holding a drink which should not spill)
- c) Normal: Normal landing (movements can be a bit rough if needed)

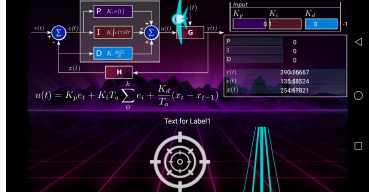
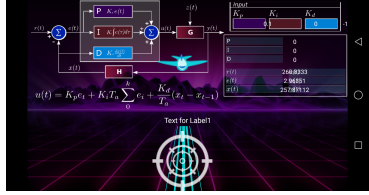
2.2. Solution

The task consists of two parts, the implementation and the tuning of parameters.

2.2.1. Implementation

BangBang

The BangBang algorithm is implemented rather simply. If the current offset (denoted as 'error') is bigger than the hysteresis (the range of values that require no correction, denoted as 'Kh') to the left or to the right, the plane steers as much as it can to the appropriate direction. Inside the acceptable range, the plane steers straight ahead. For visual examples, refer to Table 2.2.

Offset	Correction applied	Image of the situation
$\text{error} > 0 + K_h$	Max correction to the left	
$\text{error} < 0 - K_h$	Max correction to the right	Analogous to the previous image
$ \text{error} < K_h$	No correction	

Tab. 2.2.: The BangBang logic

PID-controller

To implement the PID-controller, I stuck closely to the pseudo code shown in class[6]. I will explain my implementation based on illustration 2.2. The error is calculated by subtracting the x-axis coordinates of the plane from the runway. The time difference 'dt' is set using SystemTime.

Afterwards, the error is integrated by multiplying it with 'dt' and adding it to the previous value called 'accumulated error' or 'acc_err'. To avoid a 'locking' effect the previous accumulated error is multiplied by 0.99 to make it slowly 'forget' the past. Also, this value has to stay within the boundaries given by 'CorrectorMin' and '-Max'. I implemented the option to link these limits to the used weight, because in my opinion that made more sense and led to a better behaviour. Otherwise the integral part alone could never make use of the whole correction range, because it would be impossible to reach the maximum, and therefore struggled to keep up against the wind.

The derivation is calculated by dividing the $\Delta x - \text{coordinate}$ by the Δtime , so to speak, the derivative term represents the velocity. At the end, the three terms are weighted, summed and the correction is applied.

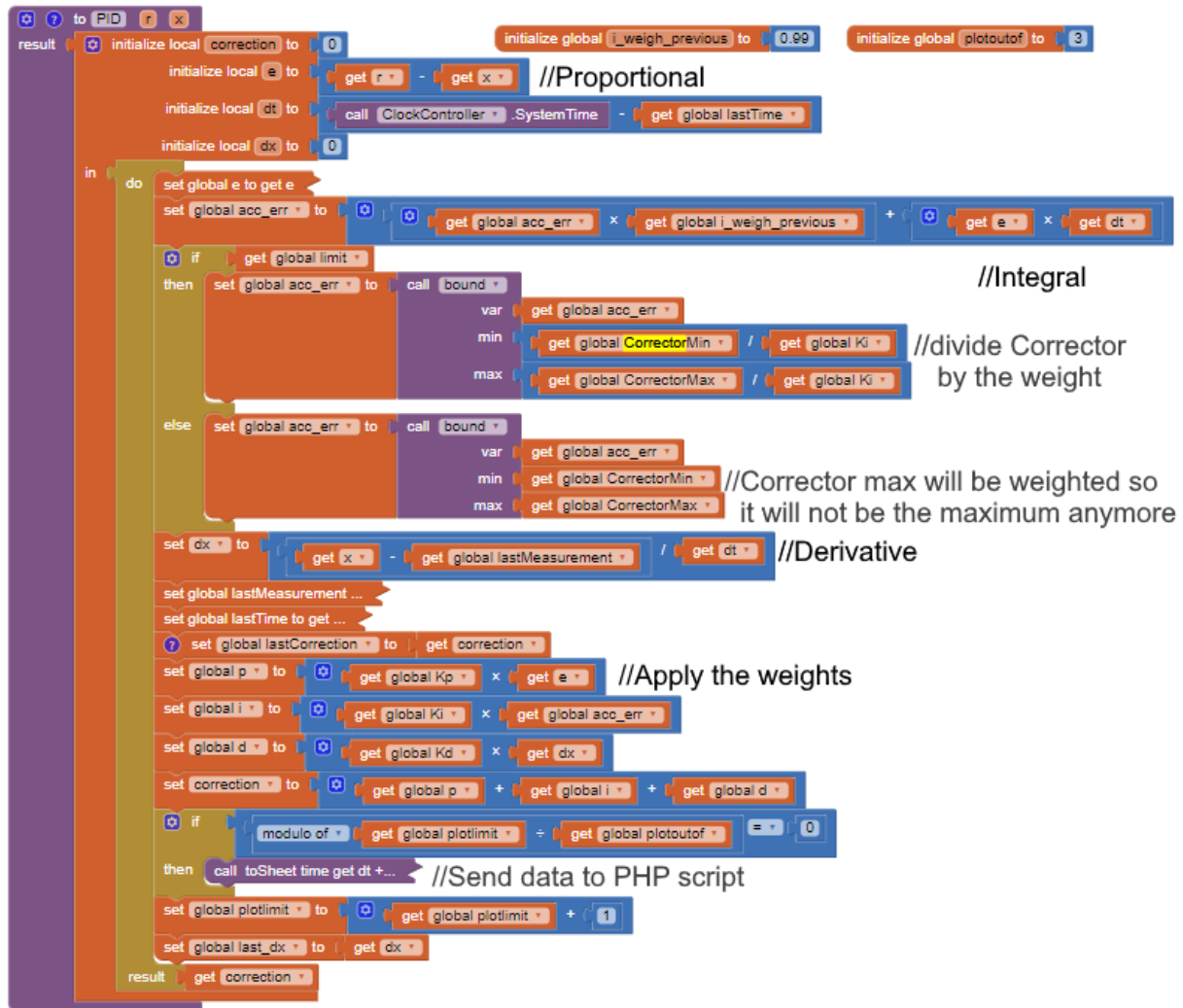


Fig. 2.2.: The code blocks for the PID-controller

Improved Settings

I added a selection to choose between the three different PID-modes in the settings screen, to allow for an efficient way of testing the different scenarios.

Additionally, I added a switch for the previously mentioned weighing of integral limits, as can be seen in Figure 2.3.

Furthermore, I added a switch to turn on smoothing of the aircraft's movements. This is done by limiting the angular difference the plane's banking can have compared to the last iteration. Without this smoothing, the aircraft's movements get shaky for big K_d values, I suspect that inaccuracies in the timing used for the derivation is the cause of the problem. This measure simultaneously solves the problem of having jumpy corrections when the runway location changes suddenly. Of course, one can argue that with this modification the PID-controller is not a pure PID-controller anymore.

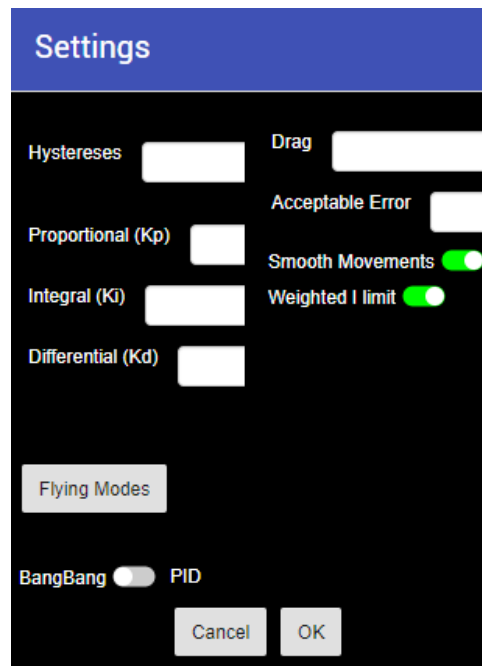


Fig. 2.3.: Screenshot of the settings including my additions

Logging Via Web

To access the simulation's raw data easily, I had to find a way to transfer the data from the App to my computer, where I can analyse and visualise it better. We normally used MQTT to achieve communication between multiple devices. However, since I need to upload many data points in real time, without losing any data, I decided against using MQTT. I reasoned, that MQTT is not well suited for queuing log-like messages because it is organised using topics, where only the latest event is stored and not the entire history. Of course, we could scan for new messages faster than we publish them, but I wanted to avoid such a race-condition. Or I might just not realise the MQTT-solution to my problem due to limited experience in that technology.

Therefore I settled on using a simple PHP script. An overview over the whole process is given in Figure 2.4. The App sends the timestamp and the current error encoded as HTTP-GET parameters to a PHP script run on my server. That script stores the received data in a CSV-file and makes it accessible to GET requests from Clients. On the client, I can use Excel to analyse and visualise the data. Additionally, I wrapped the complete communication in a VPN, so I do not need to worry about security issues.

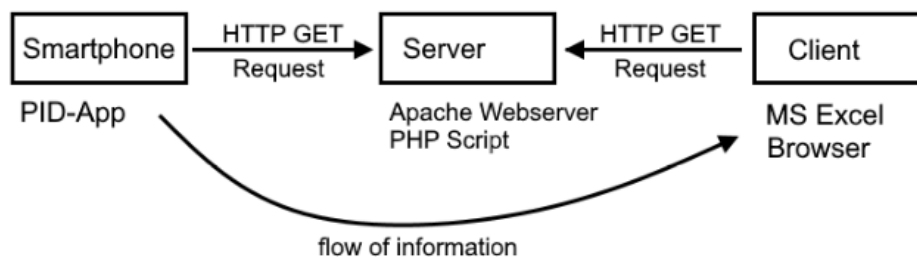


Fig. 2.4.: Overview of technologies used and the flow direction of information

2.2.2. PID Tuning

First, I tried to use the Ziegler-Nichols method for tuning PID-controllers. I used the formula mentioned in Figure 2.5[2].

Control Type	K_p	K_i	K_D
P	$0.50K_c$		
PI	$0.45K_c$	$1.2K_p/T_c$	
PID	$0.60K_c$	$2K_p/T_c$	$K_p T_c/8$

Fig. 2.5.: Look-up table for PID-controller tuning with the Ziegler-Nichols method, where T_c = oscillation period and K_c = minimal value for K_p where oscillation starts.

However, this approach did not seem to work well. When searching for K_c , Very small values for K_d already induced oscillation. I then added the appropriate K_i and K_d values to the controller. Nevertheless, the oscillations where still present and the controller was too weak (e.g. too small parameters) to counteract even little wind disturbances, as seen in Figure 2.6.

Out of curiosity, I tried higher values for K_p and was surprised, that they stayed better at the target value than the values proposed by the Ziegler-Nichols method, although they still oscillated, shown in Figure 2.7. I suppose, the Ziegler-Nichols method does not work in this context because there is inertia and drag associated with the planes movement.

Consequently, I sought to tune the PID-controller manually. I inspected the code to determine which parameter is responsible for what behaviour. The proportional is straight forward just a correction to where the runway is currently. The integral sums up all errors over time, therefore corrects long lasting offsets caused by disturbances. The derivative resembles the speed, so if we weigh it negatively, we can avoid fast movements and therefore hopefully mitigate oscillations and overshoot.

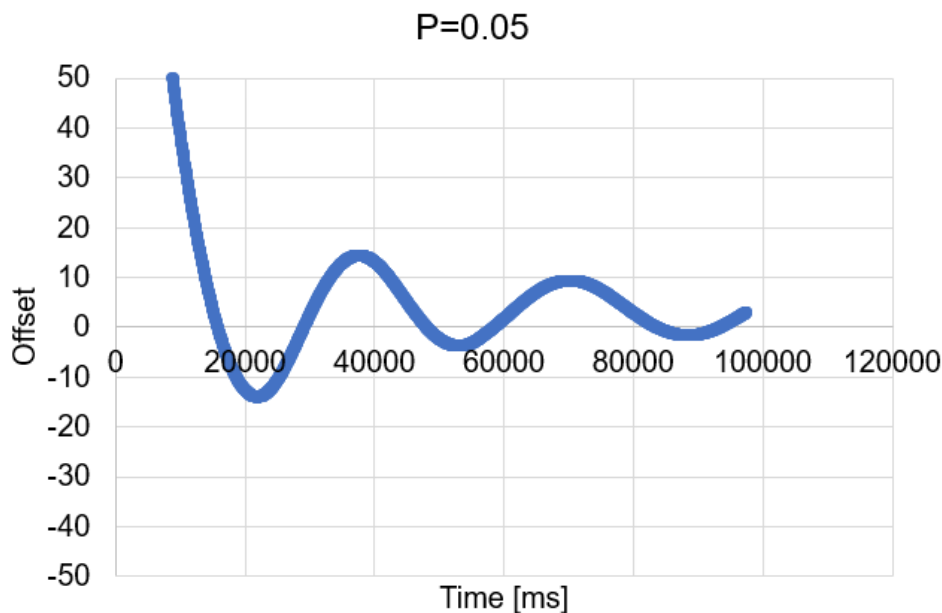


Fig. 2.6.: Plot of the error with $P=0.05$ $I=0$ $D=0$

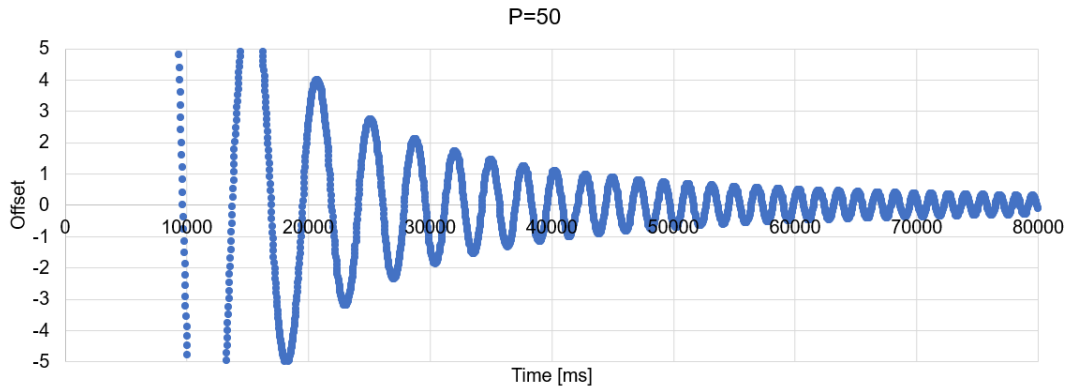


Fig. 2.7.: Plot of the error with $P=50$ $I=0$ $D=0$

I started only adjusting the proportional and derivative and keeping the integral at 0 while not introducing any wind, to rule out any oscillations before trying to correct against disturbances using the integral.

After some time of trial and error, I found a triple of values which allowed the plane to land within a reasonable amount of time, but only if there was close to no wind. Subsequently, I found out that to compensate the wind, the integral and proportional weights must be big enough, but this leads to enormous overshoot and oscillation if they are not accompanied by an even heavier weight for the derivative.

Finally, for each scenario, I tried to find compromises in the three values to enable the plane to resist some wind, without too much shaking due to a too high derivative weight. But without shaking, there are just no good solutions. Therefore I disregard the shaking as a constraint for the 'Normal' and 'Drink' mode and just focus on the plane's behavior near the runway without the shaking. I used 1 as the acceptable Error. I soon managed to get an acceptable result shown in Figure 2.9, and a video[8] is also available. The first 25 seconds are clearly dominated by this ominous shaking, but once the aircraft is close to the runway, it's banking angle is stable and moves smoothly. It even manages to continue the landing approach despite mediocre wind (notice that the wind changed direction between 33 and 40 seconds).

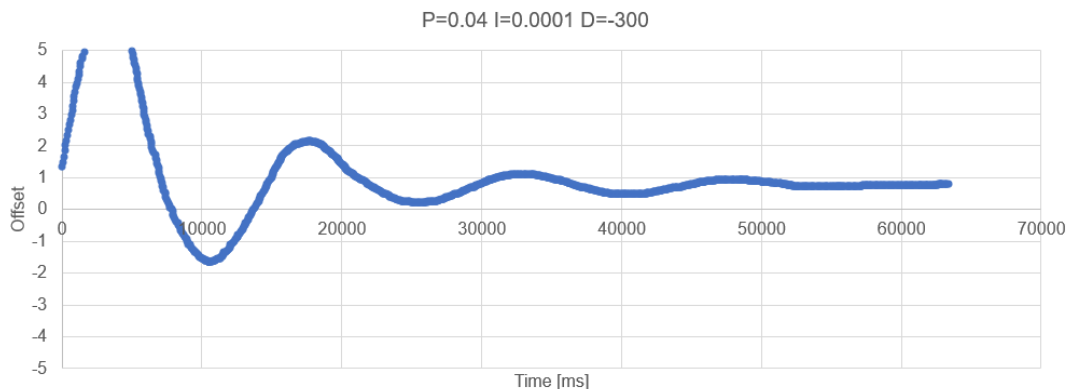


Fig. 2.8.: Plot of the error with $P=0.04$ $I=0.0001$ $D=-300$

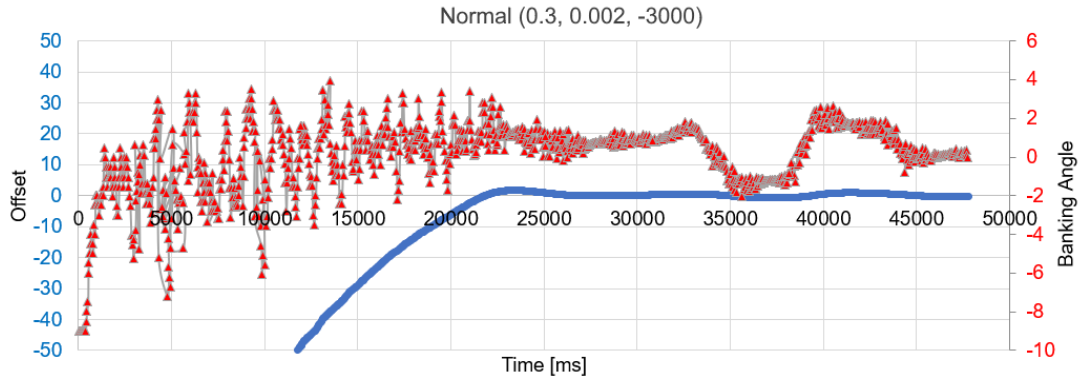


Fig. 2.9.: Plot of the error and banking angle for $P=0.3$ $I=0.002$ $D=-3000$

2.3. Result

My final results are shown in Table 2.3. Except for the uncontrollable shaking I am very satisfied with my values for the Fighter-Jet mode. In Figure 2.10 one can clearly see that the jet reacts quickly (red line) to the wind and stays perfectly (blue line) above the runway. Only the back and forth between seconds 10 and 15 is unwanted.

On the other hand, when looking at the Drink-mode in Figure 2.11, we see that this mode reacts slowly and in sinus wave-like ways to changing winds. This should prevent the passenger from spilling her drink. Although, this mode does need to abort the landing if the wind gets any stronger than a mild breeze, as one can see at the end of the graph. As expected, the shaky noise at the start is also calmer than with the fighter-jet, because we use a smaller K_d value.

Even the edge-case of instant runway relocation is covered, as can be seen in Figure 2.12. Jumpy behaviour would look like vertical red lines in the graph, but these are clearly diagonal and therefore resemble smooth movements of the plane.

Mode	K_p	K_i	K_d
Fighter-Jet	2	0.006	-9000
Normal	0.3	0.002	-3000
Drink	0.03	0.001	-1500

Tab. 2.3.: The values for the different modes

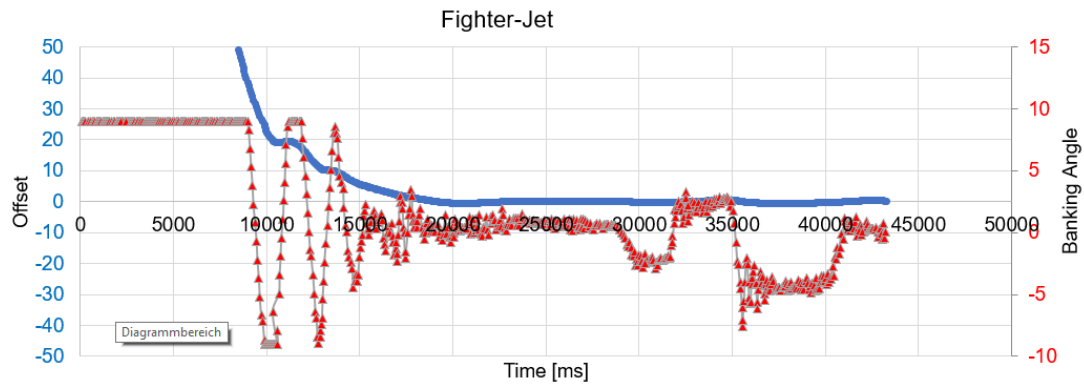


Fig. 2.10.: Plot of the error and banking angle of the Fighter-Jet mode

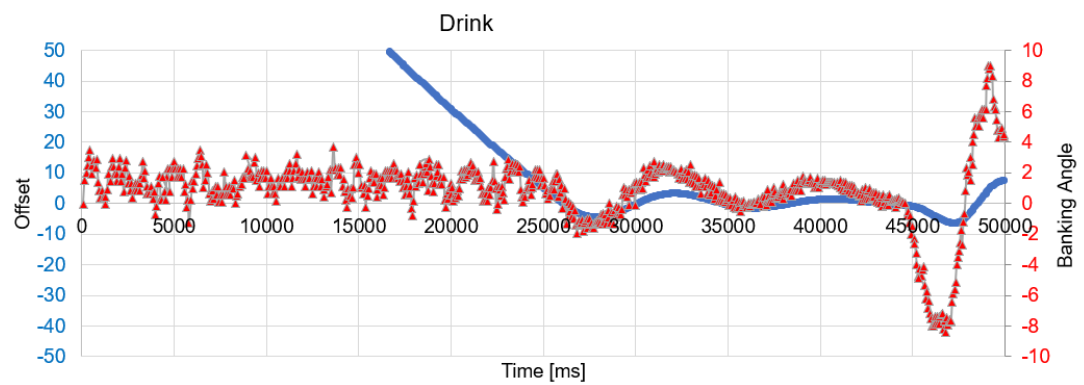


Fig. 2.11.: Plot of the error and banking angle of the Drink mode

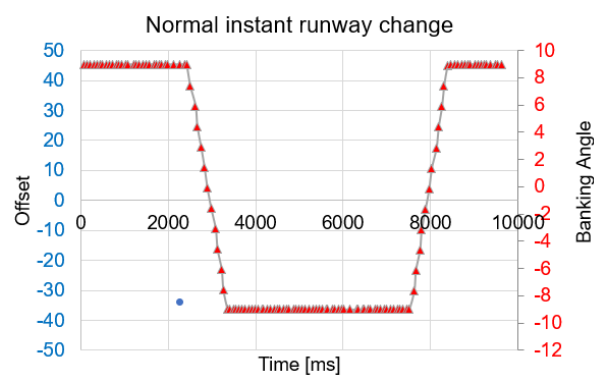


Fig. 2.12.: Plot of the error and banking angle of the Normal mode with instant runway relocation

3

Discussion

3.1. Discussion	13
3.2. Improvements	13

3.1. Discussion

Before ranking the BangBang algorithm against the PID-controller we have to define the target use case. The BangBang algorithm is simple and effective in landing the plane in mild wind, but the aircraft's movements are very jumpy and therefore impossible in reality.

Contrarily, the PID-controller leads to more realistic movements but requires tedious tuning and is in this case prone to some weird shaky behaviour when the runway is further away. All in all though, the PID-controller does handle the aircraft fascinatingly well once above the runway. Even mediocre gusts of wind can not push the plane to the side.

3.2. Improvements

The most important improvement would be to somehow smooth the plane's movement even more when using high values for K_d . Because of this phenomenon, it is factually impossible to find suitable values for the scenario with a passenger holding a drink, except for when we purposely exclude this part from our scope. So there are two possible solutions: Fix the reason for this shaky behaviour, or include the smoothing-parameter when changing modes, so we can limit the plane's movement with this variable instead of the PID-weights.

Apart from this, it might be possible to find PID-values which give similar results, that do not require modifications to the PID-controller, but run on a pure PID-formula.

4

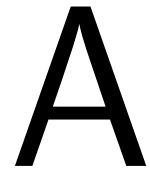
Conclusion

Both algorithms, BangBang and the PID-controller are capable of landing a plane despite some wind. The BangBang algorithm leads to movements that are physically not possible in the real world, whereas the PID-controller creates movements that look convincingly real. But I encountered a problem with the PID-controller, that led to the aircraft shaking back and forth when it was far away from the runway. The probable reason for this unappealing behaviour is some inaccuracies with the timing and big weights for the derivative term.

The source code to this App can be downloaded: [aia](#)

The APK to this App can be downloaded: [APK](#)

The raw data and graphs for this paper can be downloaded: [Excel File](#)



License of the Documentation

Copyright (c) 2022 Marco B. Gabriel.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

The GNU Free Documentation Licence can be read from [4].

References

- [1] S. Bennett. Development of the pid controller. *IEEE Control Systems Magazine*, 13(6):58–62, 1993. <https://ieeexplore.ieee.org/document/248006> (accessed April 16, 2022). 3
- [2] Christos Tsironis and K Giannopoulos. *In: Glow Discharges and Tokamaks ISBN: 978-1-61668-352-8 Editor: Sean A. Murphy*, pages 1–80. 10 2010. 9
- [3] Ramon VilanovaAntonio Visioli. Pid control in the third millennium. *Advances in Industrial Control*, page ix, 2012. https://folk.ntnu.no/skoge/puublications_others/books/vilanova-visioli-PID%20control%20in%20the%20Third%20Millennium%20-%20Springer%202012.pdf (accessed April 16, 2022). 3

Referenced Web Resources

- [4] Free Documentation Licence (GNU FDL). <http://www.gnu.org/licenses/fdl.txt> (accessed July 30, 2005).
- [5] MIT Appinventor. <http://ai2.appinventor.mit.edu> (accessed April 16, 2022). 2
- [6] PID Algorithm. https://moodle.unifr.ch/pluginfile.php/1393106/mod_resource/content/1/SteuerungRegelung.sozi.html#frame2086 (accessed April 23, 2022). 6
- [7] Moodle Page of the Process Control Course 2022. <https://moodle.unifr.ch/course/view.php?id=266201> (accessed April 16, 2022). 2, 5
- [8] Video of the Fighter-Jet mode. <https://www.youtube.com/watch?v=10-RV7210ec> (accessed April 24, 2022). 10