Autotuning PID using Relay Feedback Method

ETACS

Peter L. Almvig, 202006421
Adrian D. Anthony, 202205347
Simon A. Andreasen, 202207041

Group 1

5. nov. 2024

Introduction

Manual tuning of PID controllers is an involved process that can take a lot of time and iterations to get right. Auto tuning a PID controller is a method that allows a user to automatically tune a PID controller. Even if the auto tuning isn't as good as a user manually doing it, a lot of time can potentially be saved (as long as the auto tuning is good enough). It can also be helpful if the controller being tuned is controlling a system that can have some variations over time (like changing load, degradation or other things that can impact its performance).
In this case study we have looked at auto tuning using the relay method in respect to a LEGO car capable of driving forward.
We have designed two PID controllers for the LEGO car, one that we manually tuned and one that we used the relay method to tune and compared their performance driving 50 cm. The performance metrics used to measure the car was speed (time to reach 50 cm), accuracy (how close it was to reaching 50 cm) and overshoot (how much the car went past 50, before settling).

Problem Statement

In this project we seek to examine autotuning PID controllers, specifically the relay method. To this end, we wish to first look at how PID controllers are traditionally tuned, followed by a description of how PID controllers can be automatically tuned using the relay method.
We wish to create and manually tune it for the LEGO car and compare it to a PID controller that's been auto tuned using the relay method. We wish to compare both methods in an experiment of how accurately the LEGO car can move 50 cm in the shortest amount of time.

Methods and Materials

The following section will describe the manual design of our PID controller.

Defining the transfer function

The transfer function of the lego car has been modelled as a black box using a step response to estimate the model parameters of the lego car. This was done by stepping the motor signal and observing the wheel velocity. These parameters were found to be:

Stationary gain: $K_0 = \frac{0.181}{50}$

Time to rise: $\tau = 0.15$

$$\alpha_1 = \frac{1}{\tau}$$

A scaling factor: $K_2 = \frac{K_0}{\tau}$

This gives us the transfer function for the car (expressed as the velocity at the output):

$$G_v = \frac{K_2}{(s+a_1)}$$

Integrating this gives us the transfer function for the car expressed as the distance the car has traveled (in meters):

$$G_s = \frac{1}{s} G_v$$

The bode plot for this transfer function can be seen in Figure 1.

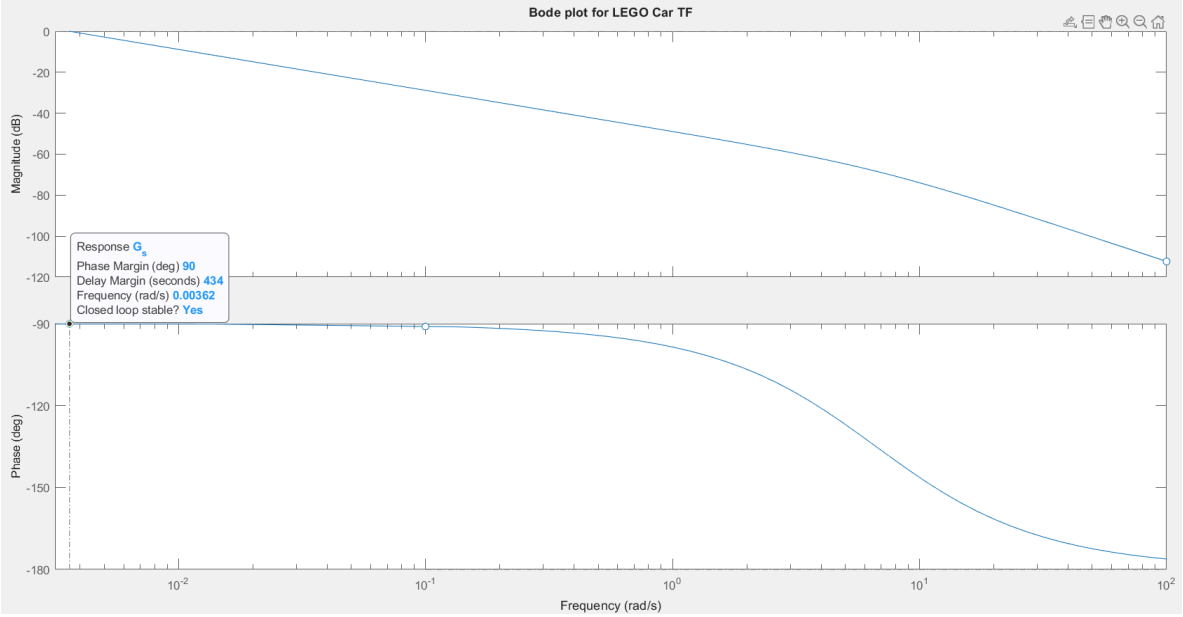

Figure 1: The transfer function of the LEGO car with no controller

The bode plot shows us a very slow crossover frequency and giving the car a step shows a rise time of 607 seconds.

Manually tuned PID

We found that changing the proportional gain was enough to make the car move 50 cm fast enough. An integrator was added to remove the stationary error. This was not necessary for the simulation but may have been needed if we added it to the physical car. Since the OS% was minimal in general, we decided that a derivative for the controller would be mostly redundant. The value for K_p was deduced using the relation between the OS%, dampening factor ζ, and the phase margin γ. We wanted a max of approx. 5% OS, which deduced a gain of 57 dB. Using the logarithmic relation between dB and magnitude, and iterative tweaking for the integrator parameter K_i we get:

$$K_p = 10^{\left(\frac{57}{20}\right)}$$

$$K_i = 2$$

$$PI = K_p + K_i \cdot \left(\frac{1}{s}\right)$$

The bode plot for the PI controller looks as follows.

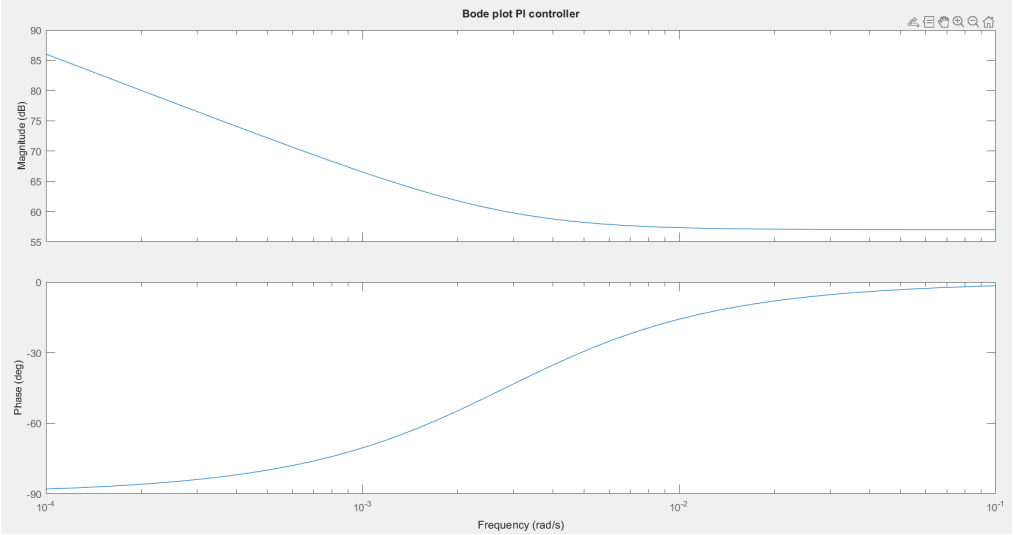

Figure 2: Bode plot for the PI controller. Increasing the crossover frequency and changing its type to remove stationary error.

The bode plot of the LEGO car shows phase margin of 70° which is expected given the overshoot and damping we were aiming for. We also see a much faster crossover frequency and the type of the system has changed.

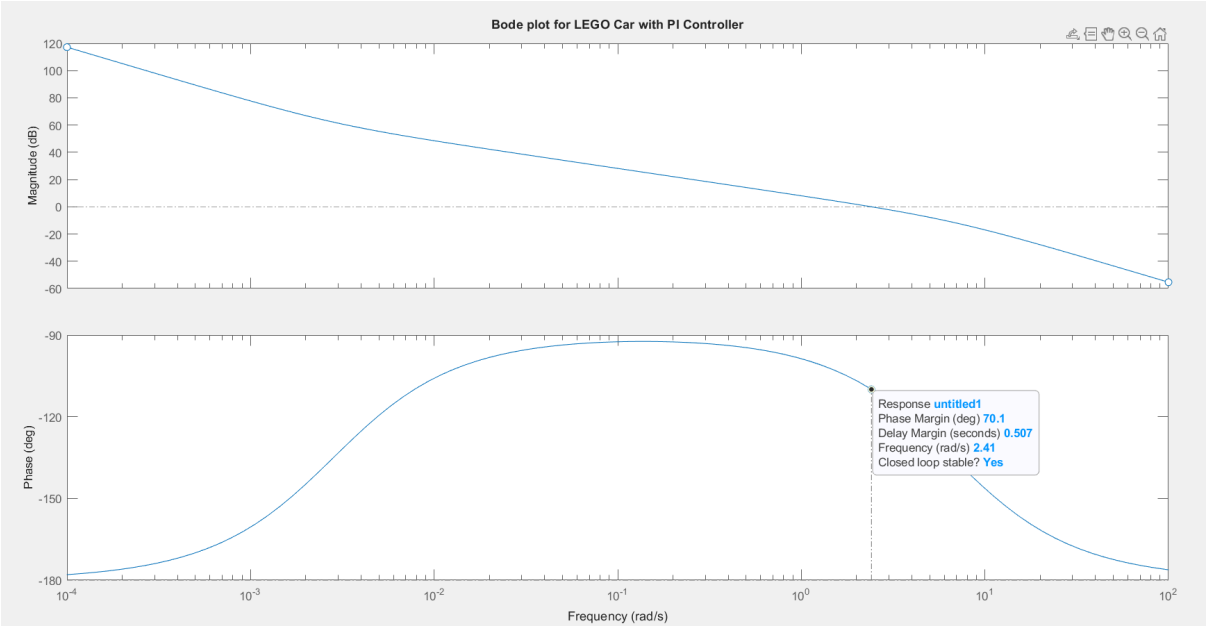

Figure 3: Combined bode plot of the LEGO car with PI controller
Showing it's much faster crossover frequency and lower stationary error.

The step response for the lego car now looks much more reasonable with a rise time of 0.602 seconds and overshoot of only 1.49%. We also see no stationary error so the car should stop after reaching 50 cm.

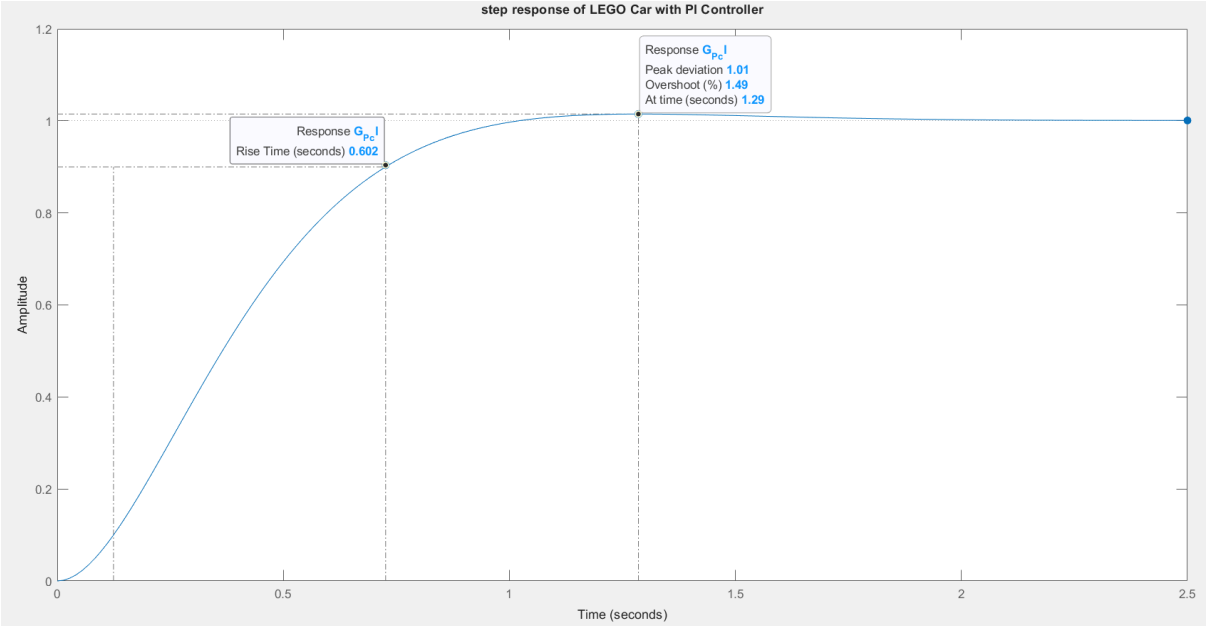

Figure 4: Step response of the LEGO car with PI controller.

When simulating it on the car we have to cap in the control signal so that it stays reasonable. The control signal is the blue plot. The yellow plot shows the LEGO car's position. Here we can see that the LEGO car moves 50 cm in 2.5 seconds without error. The reason for the settling time in the LEGO car model being slower to that of the Matlab step response is because of the saturation of the model, where control signals above 70 aren't handled properly by the model.

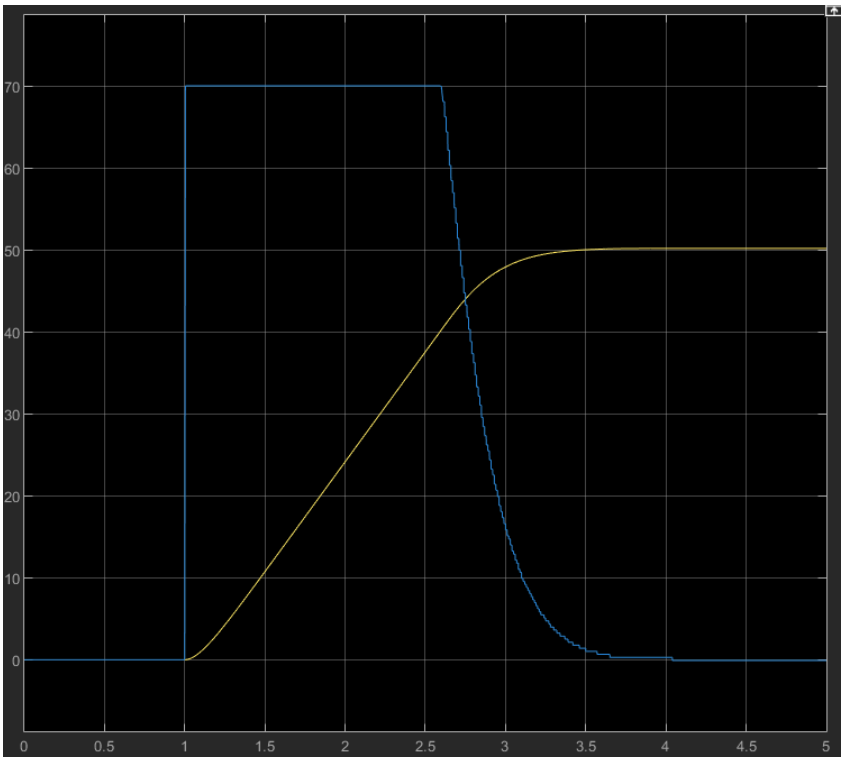

Figure 5: Plot of the LEGO car's position compared to the input signal.
The yellow plot shows the LEGO car's position in centimeters.
The blue plot shows the input signal that has been capped.

Autotuned PID

In this case study we have tasked ourselves with implementing relay feedback auto tuning. This method makes use of a relay feedback:

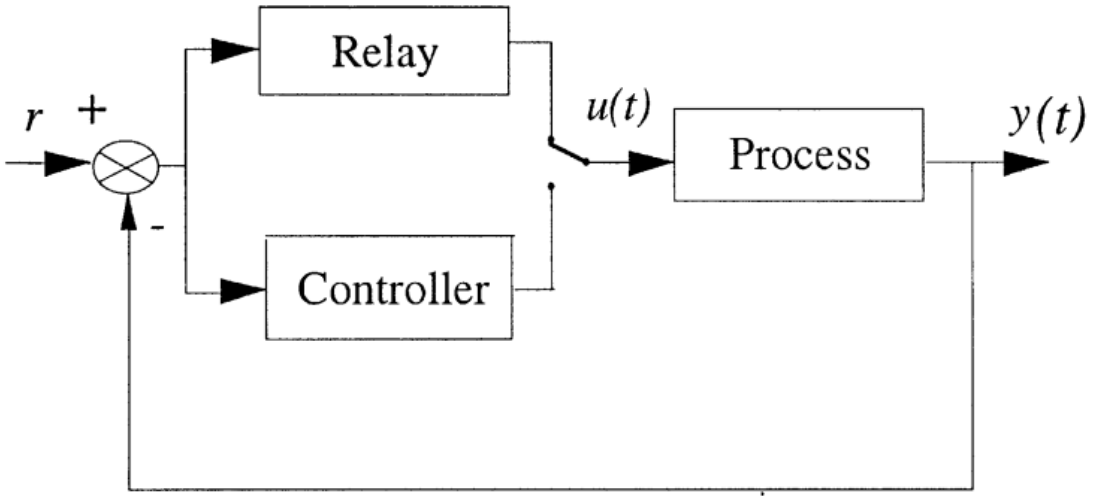

Figure 6: Block diagram of the relay feedback system.[2]

Here, the relay switches between two actuation signals which have the same absolute value but opposite signs this absolute value is d. This creates a pulse signal, causing the plant to go into oscillation. This oscillating signal is very useful for creating a controller. The universal gain Ku and period Pu, can then be approximated from the signal.

If a plant uses a specific step it can be smart to use a reference r which is equal to that step. Since the plant may have disturbances that affect the relay, the reference would work as a form of bias, making the parameters that we get from the relay feedback test more accurate.

Where the gain is given by this equation:
$k_u = 4d/\pi a$ where d is the amplitude of relay, and a is the amplitude of the oscillating signal of the plant.

T_u is the period of the plant signal.
After these values have been found, tuning methods that make use of the Ku and Tu may be used. A classic example of this is the Ziegler-Nichols (hereafter referred to as ZN) method, which we choose to use. The parameters can be seen in the table below

Ziegler–Nichols method[1]

Control Type	K_p	T_i	T_d	K_i	K_d
P	$0.5K_u$	–	–	–	–
PI	$0.45K_u$	$0.8\bar{3}T_u$	–	$0.54K_u/T_u$	–
PD	$0.8K_u$	–	$0.125T_u$	–	$0.10K_uT_u$
classic PID[2]	$0.6K_u$	$0.5T_u$	$0.125T_u$	$1.2K_u/T_u$	$0.075K_uT_u$
Pessen Integral Rule[2]	$0.7K_u$	$0.4T_u$	$0.15T_u$	$1.75K_u/T_u$	$0.105K_uT_u$
some overshoot[2]	$0.3\bar{3}K_u$	$0.50T_u$	$0.3\bar{3}T_u$	$0.6\bar{6}K_u/T_u$	$0.1\bar{1}K_uT_u$
no overshoot[2]	$0.20K_u$	$0.50T_u$	$0.3\bar{3}T_u$	$0.40K_u/T_u$	$0.06\bar{6}K_uT_u$

Figure 7: Different values controllers in the Ziegler-Nichols method [5]

After obtaining the K_u and T_u parameters from the relay feedback test we tried implementing some of the ZN method controllers. Where we used a saturation block in the simulink model to simulate the actuator constraints.

We found that the actuation signal was too high and the controller had windup issues. The control signal before and after saturation can be seen on Figure 8.

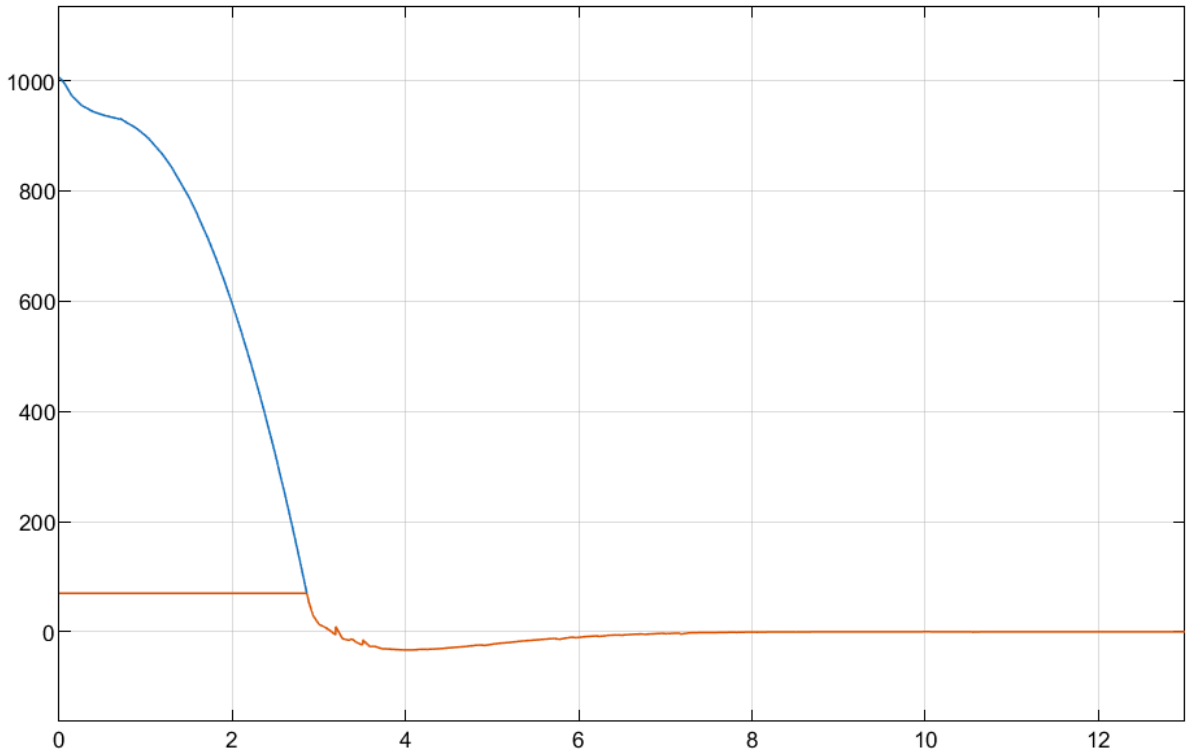

Figure 8: control signal (blue) which the classic ZN PID creates and simulated actuator output (orange).

This led us to implementing clamping in the PID controller. This implementation can be seen on Figure 9.

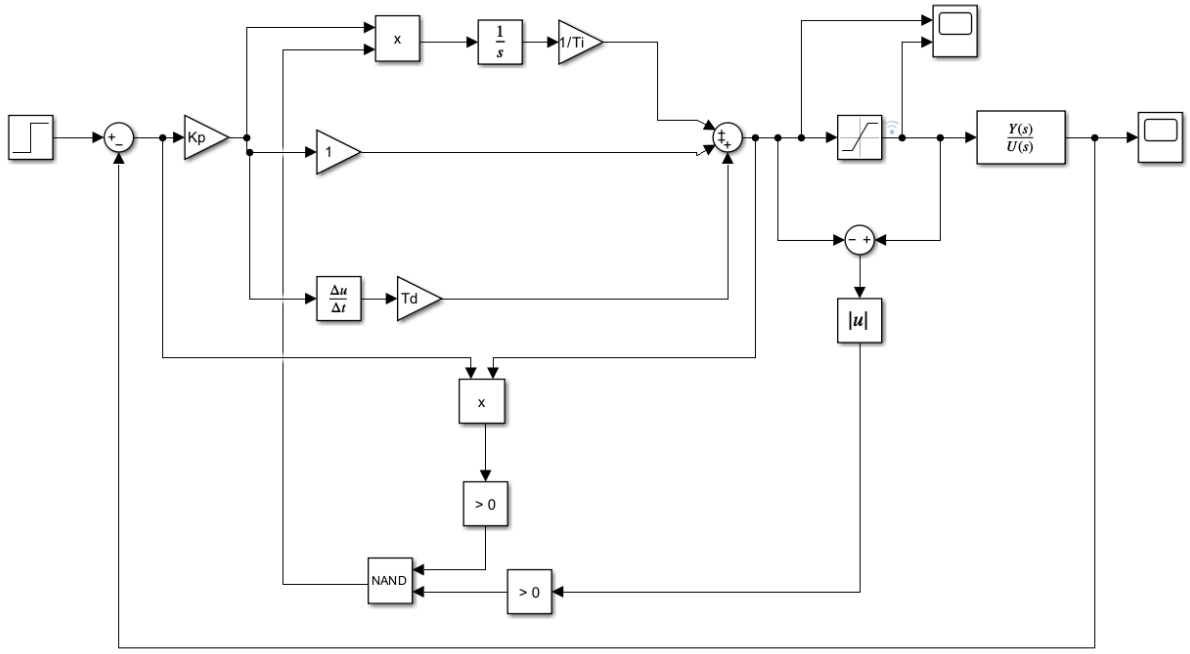

Figure 9: Simulink model of Clamped PID controller.

Experiments

The following section shows our experiments for our autotuned PID.

Autotuned PID

We first created a simulink model to get the parameters relevant to Ku and Tu for the model of the car which we got from the step test. See Figure 10 for reference.

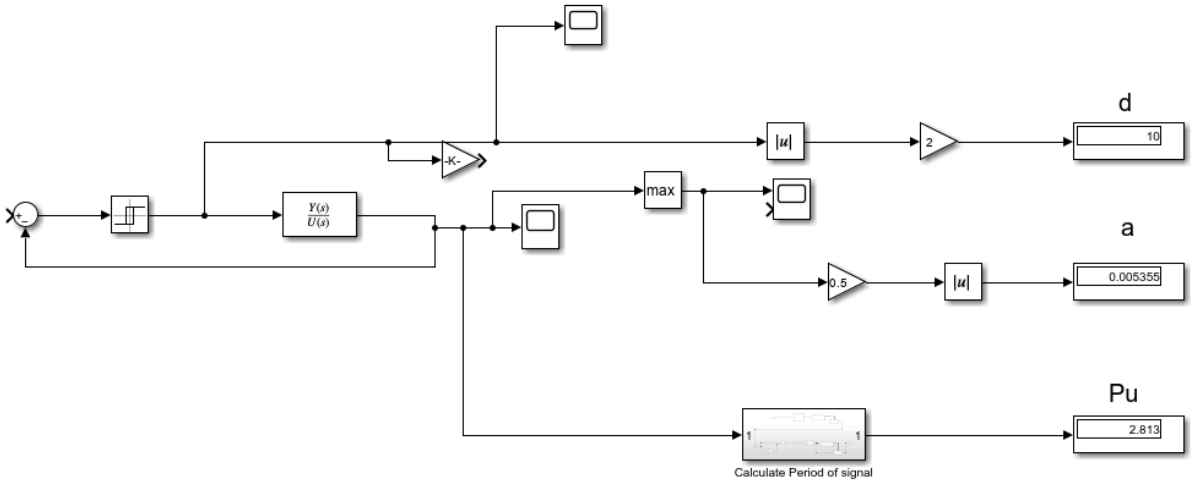

Figure 10: Simulink model of the relay system, which also calculates parameters d,a and P_u (where $P_u = T_u$).

The actuating signal goes between +5 and -5 in the relay, and changes when the feedback crosses zero, creating a pulse signal. Half of the maximum value of the plant is used to get the oscillation amplitude, and the amplitude of the pulse signal is easily found by taking the absolute value at any time of the actuation signal.

We opted not to use a reference signal meaning that the oscillations will be around 0. This makes getting the period easier, but introduces a bias for the test at 0 (since the reference signal is effectively 0).

To calculate the period of the signal some signal processing was necessary. Therefore, a subsystem was created to handle this (calculate period of signal block in simulink). See Figure 11 for implementation.

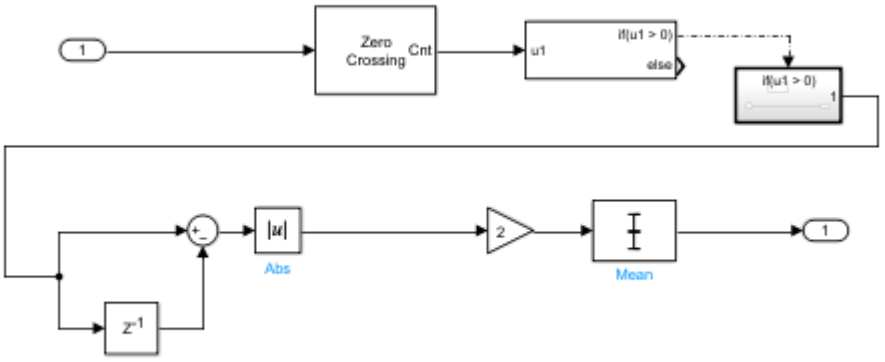

Figure 11: Subsystem used to calculate the period of the signal.

A zero crossing block was used, which outputs one when the signal crosses zero. Since we used a bias of zero when stepping, this block was quite useful.

We then used the difference of a delayed signal and the current signal to calculate the approximate half period. A gain of 2 was then applied to get the full period and the mean of the signal over time was calculated using the mean block.
We then implemented this in the LEGO car simulation, which can be seen on Figure 12.

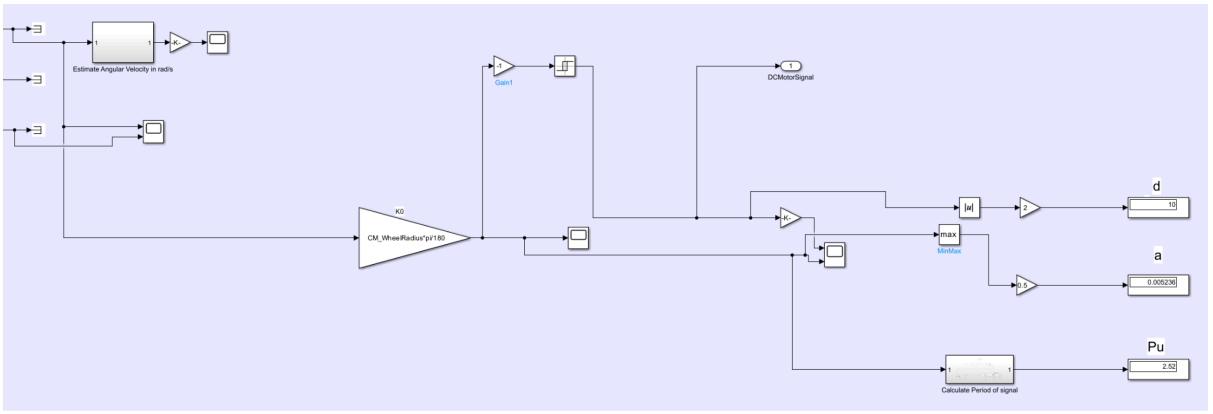

Figure 12: Relay system in the LEGO car model.

For this step, some changes were necessary because of the more complex behaviour of the model and the fact that this controller is in discrete time. The period subsystem stopped functioning because the output of the average was often 0. To fix this, a if-else block was added to check if the output was greater than 0. See Figure 13 for fix implementation.

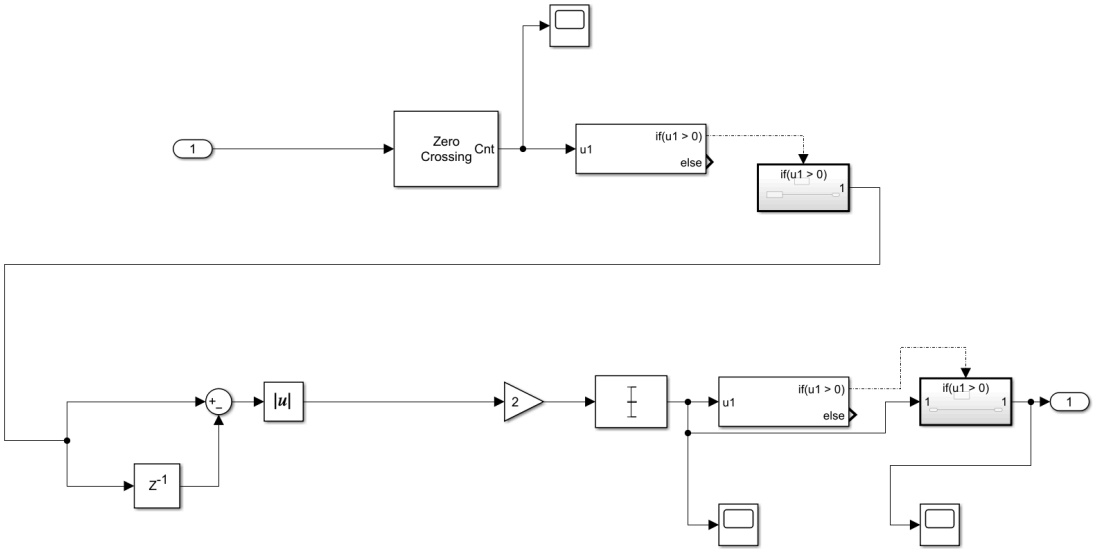

Figure 13: Subsystem used to calculate the period of the signal, slightly altered on the LEGO car.

After this fix, the behaviour was very similar to that of the simpler simulink model as can be seen in the results section.

We then implemented the PID controller in the more complex LEGO car model as well, using the parameters which we got from the relay feedback test:

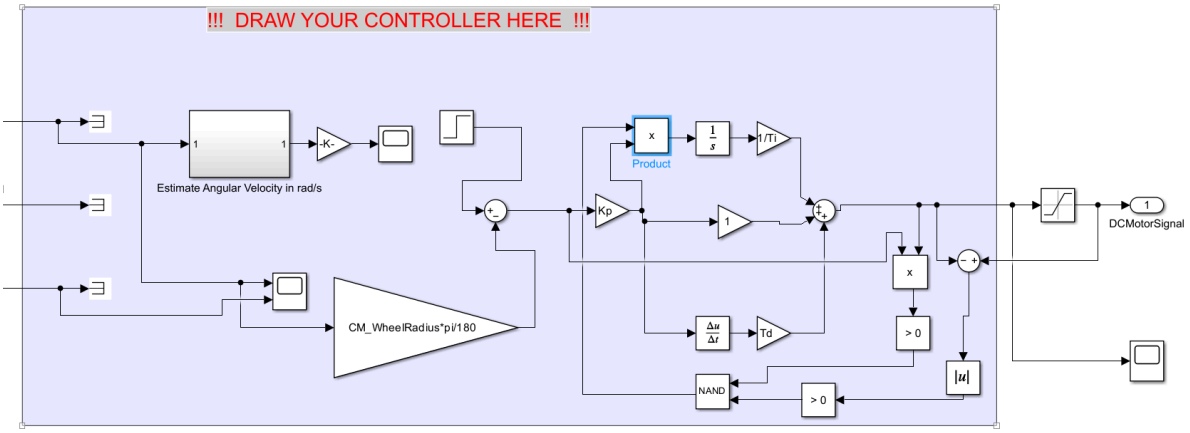

Figure 14: PID controller with clamping in the LEGO car controller.

Results

The following section discusses the relay and PID controller for the simple and complex simulink model.

Relay in simple simulink model

The experiment with the simple simulink model resulted the actuation signal in figure 15

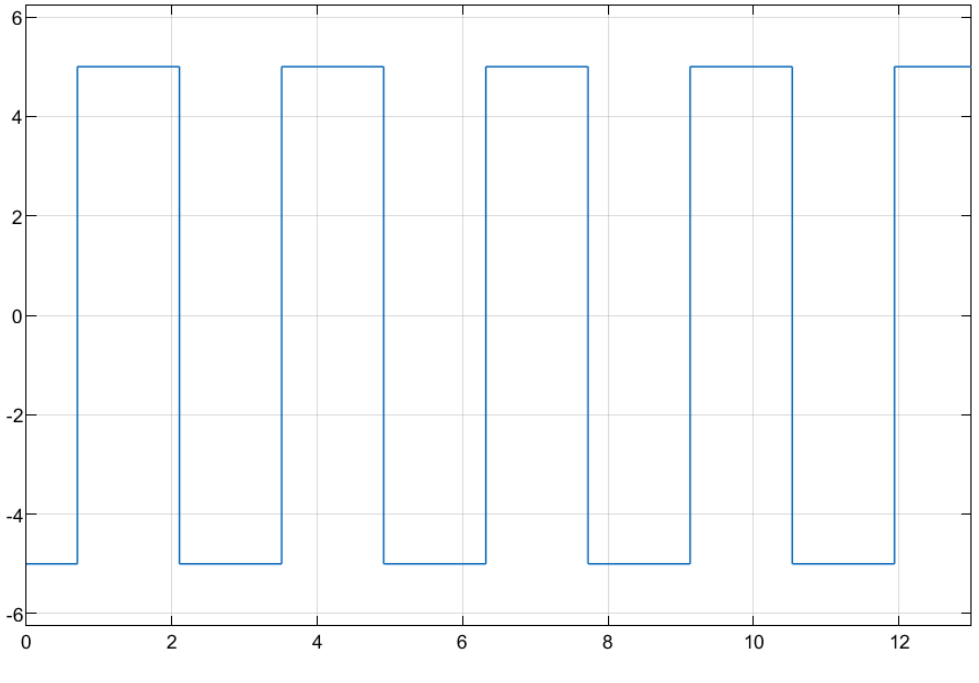

Figure 15: Actuation signal of the simulink relay system.

Which resulted in this oscillation from the plant seen in figure 16.

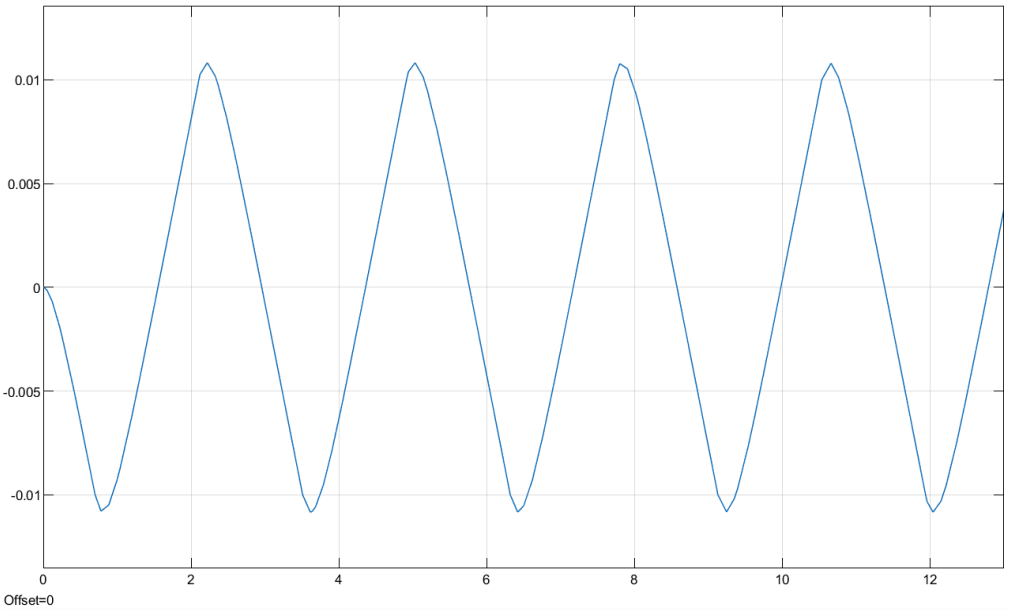

Figure 16: Car position in simple simulink model.
This makes the car oscillate from -1 cm to 1 cm from the starting position.

Relay in complex simulink model

The relay with some small changes was then implemented in the more complex simulink model giving the actuation signal seen in figure 17, and plant oscillation seen in figure 18.

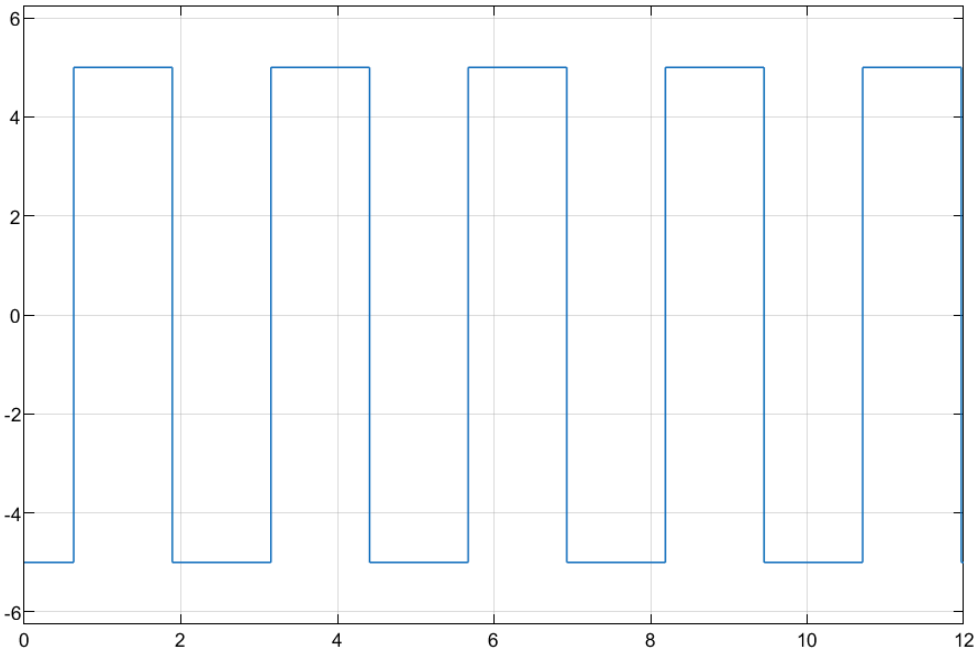

Figure 17: Actuation signal of relay feedback test in the LEGO car model.

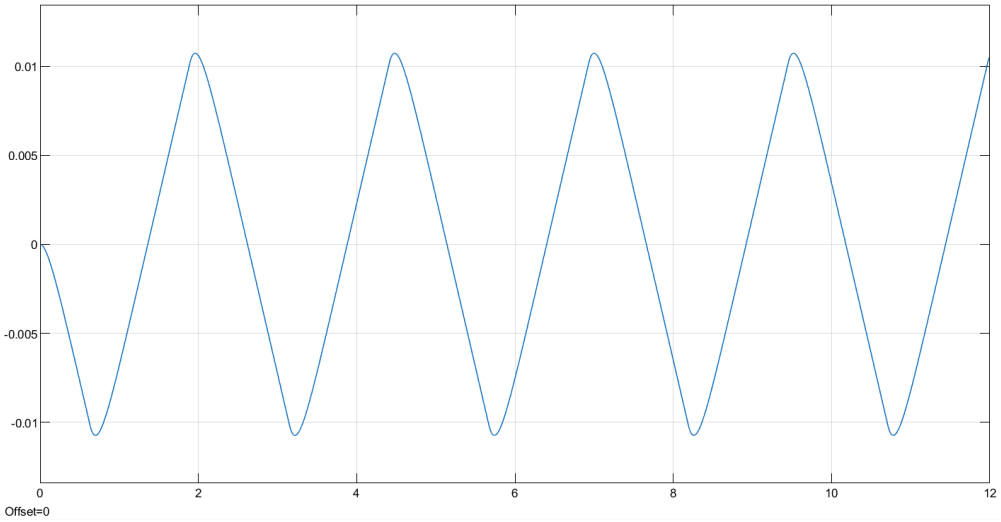

Figure 18: Position in the relay feedback test in LEGO car model.

The behaviour was very similar in both models, where the pulse signal which we used created an oscillation in the car position.

PID in simple simulink model

The PID controller which was implemented from the simple model had the following bode plot

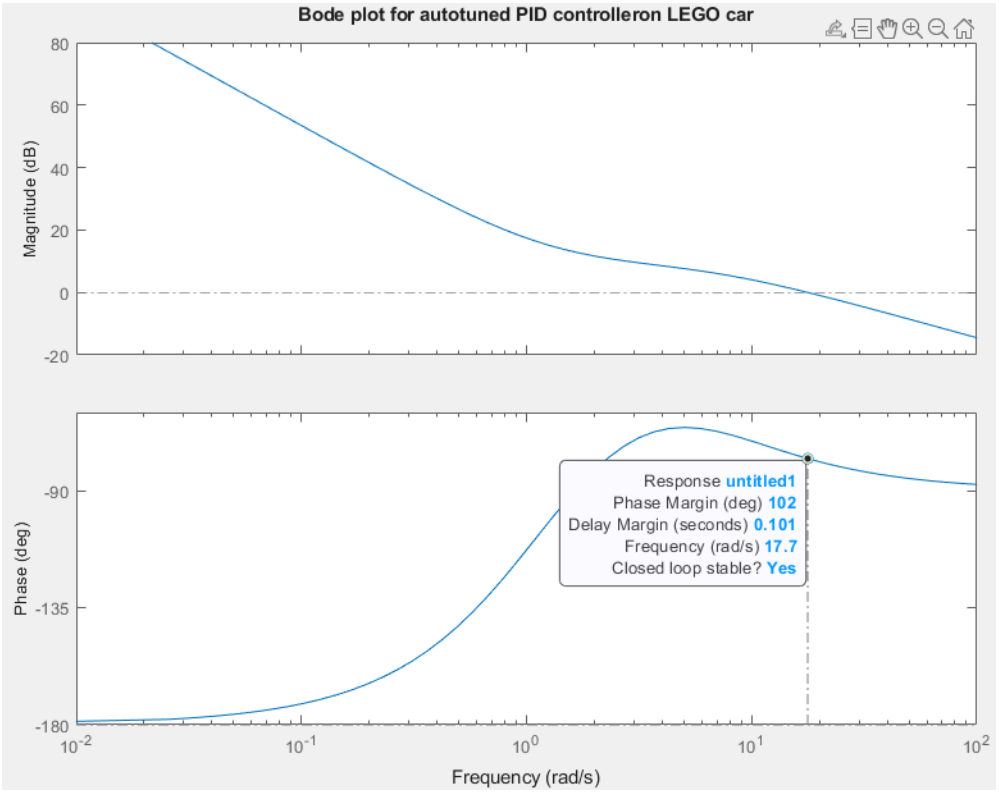

Figure 19: Bode plot for auto tuned PID controller.

This bode plot has a phase margin of 102 which should give an overshot of approximately 0, yet because of the higher order of the system, this is only an estimate.

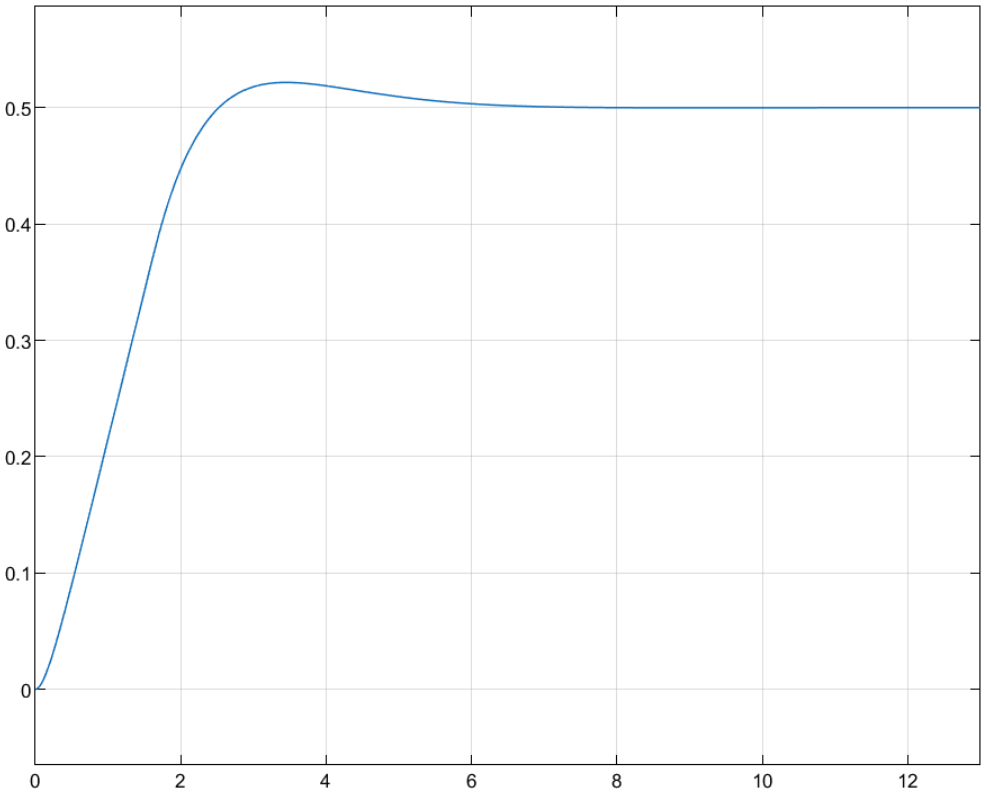

Figure 20: Step response of system with auto tuned PID and clamping.

And looking at the step response this is very much the case.

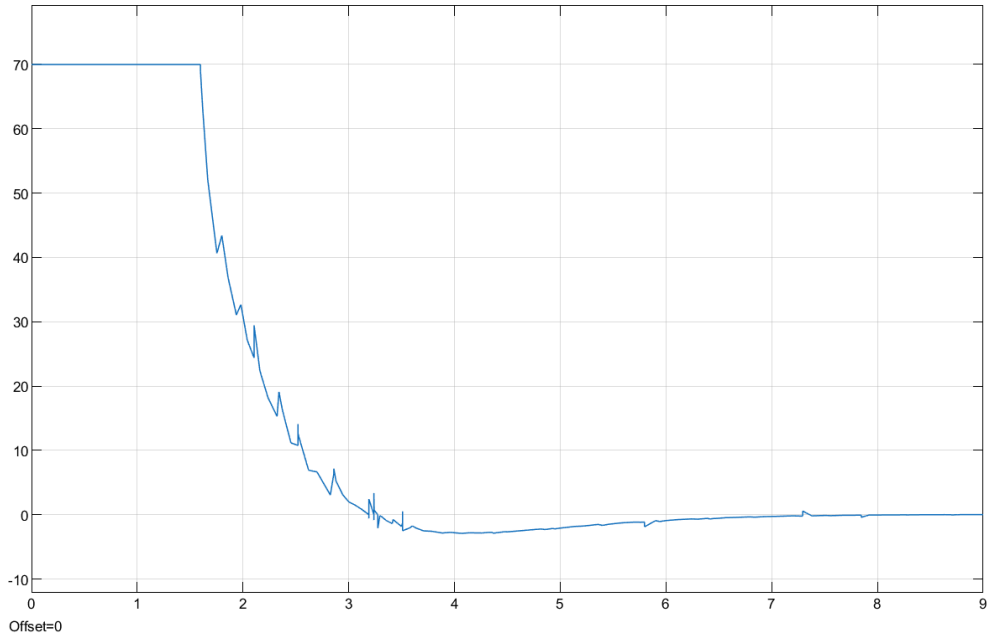

Figure 21: Actuation signal of the simple simulink PID controller

Here the actuation signal is quite choppy which is likely due to the nature of clamping. See the discussion section for more on this topic.

PID in complex simulink model

Using the relay system on the complex simulink model we got the parameters of the PID controller and deployed it, which gave the following step response

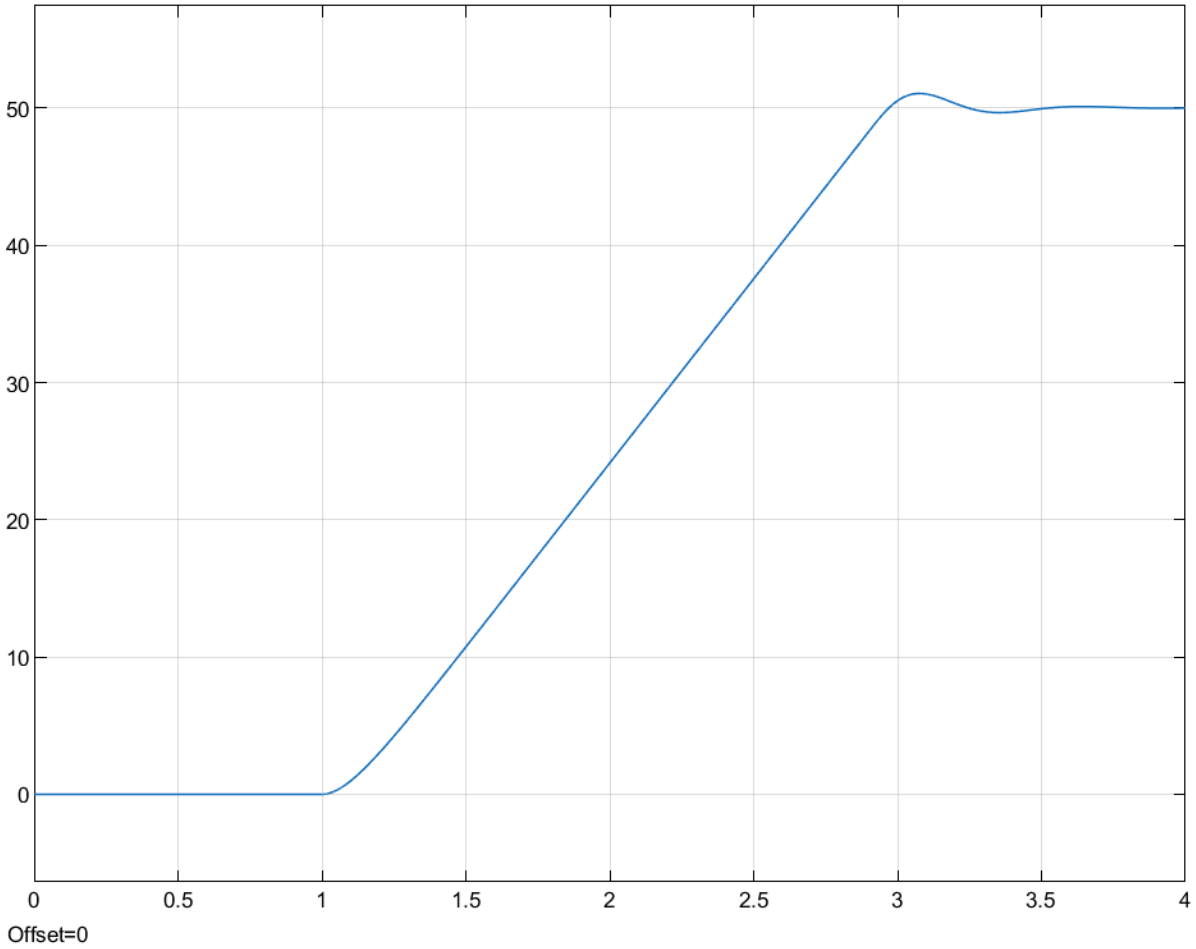

Figure 22: Step response of LEGO car simulation with auto tuned PID and clamping.

And this actuation signal, where the y-axis is in cm.

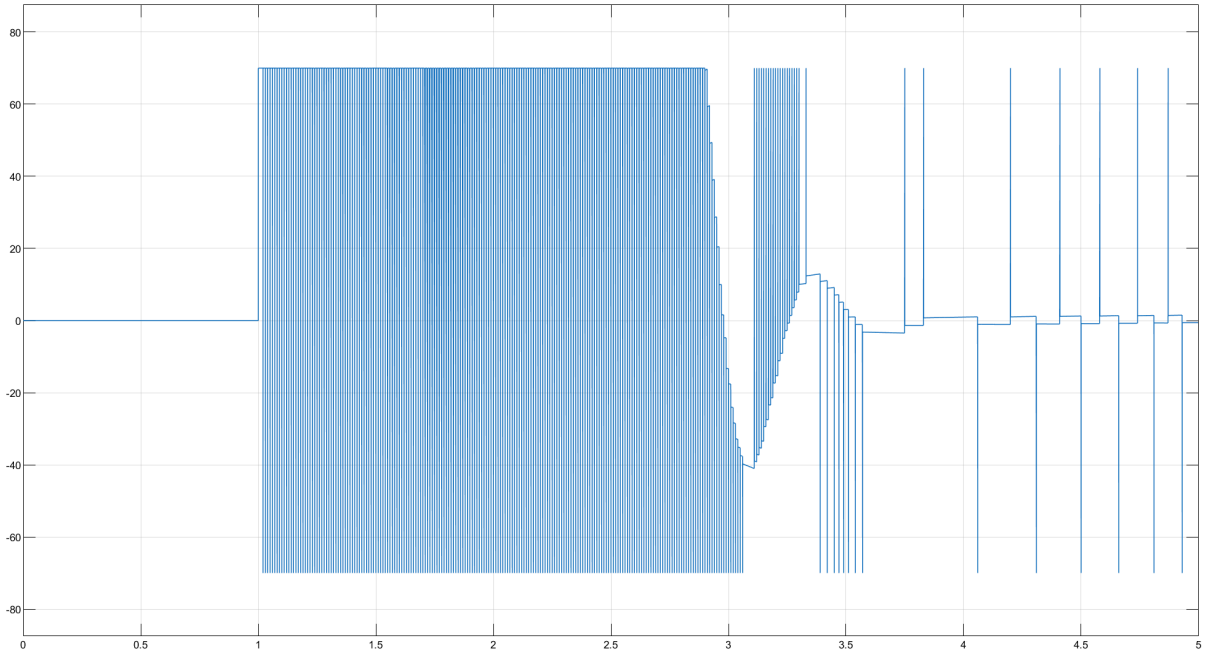

Figure 23: Control signal of LEGO car simulation with auto tuned PID and clamping.

Discussion

This section discusses our success in implementing an auto tuned PID controller and improvements we could do in the future given more time.

Control signal of the auto tuned PID.

The control signal for the auto tuned PID controller on the LEGO car simulation, which can be seen on Figure 23, looks rather odd at first glance. It contains a lot of jittering in the range of -70 to 70, yet the step response is quite good and very similar to the step in the simple simulink model. When taking a closer look at the signal, it becomes apparent that matlab exaggerates the dimensions of the jittering. A zoomed in figure can be seen on Figure 24. Since the signal remains at 70 the vast majority of the time, where it also should be, and because the step response is reasonable we have decided that this behaviour is acceptable. We suspect that this strange behavior originates from the clamping mechanism on the car, where it shuts off periodically because of the sampling rate.

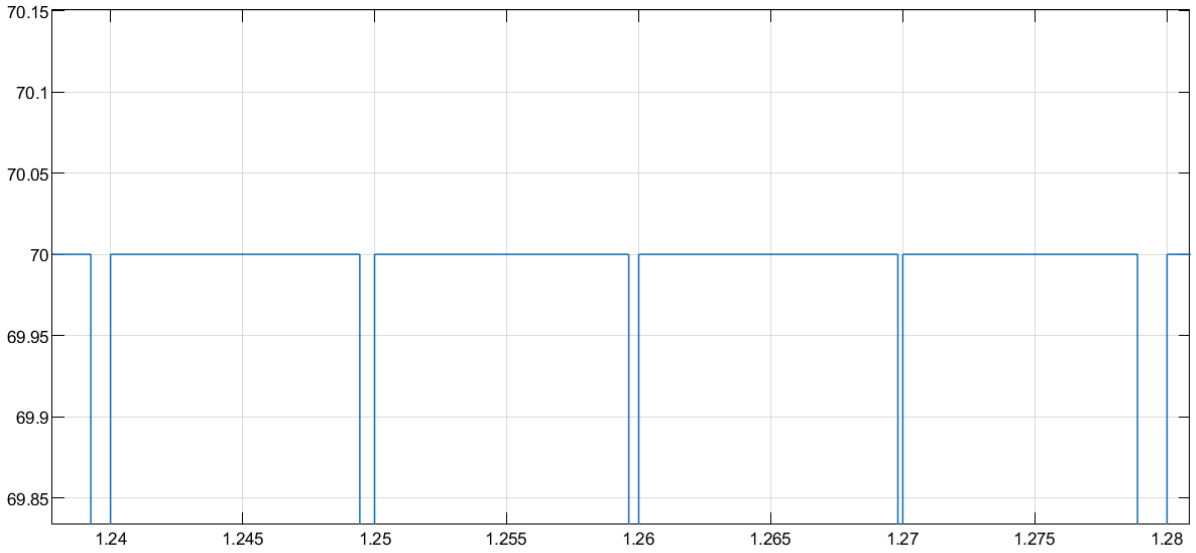

Figure 24: Zoomed in control signal of LEGO car simulation with auto tuned PID and clamping.

Autotuning in a changing environment

If assuming a constant non-changing transfer function, the implementation of auto tuning, such as relay tuning, has a lot of benefits. In settings of altering transfer function depending on differing noise from the environment, the auto tuned controller would become unreliable. In the context of the LEGO car, problems would arise if the external conditions aren't the same for the calibration process and the actual operating process of the car. External noise in the environment in this case could be rocks on the road, or anything affecting the expected behaviour of the car.

In the tutorial review[1], many improvements on the original relay autotuning method are discussed, which take into account different disturbances which disturb the autotuning process. One proposition (among others) that was discussed in the tutorial review was using fourier analysis, instead of the describing function method (the method we have used in this case study). This would make external noise less of an issue in the autotuning process.

These considerations should be taken into account depending on how one wants to use the process. For more considerations on the topic, see the Further Work section.

Comparison of manually- and auto tuned PID controllers.

Comparing the two PID controllers we see the following metrics when deployed on the complex simulink model:

	Manually tuned PI controller	Auto tuned PID controller using relay feedback test. Implemented using the classic PID controller from the ZN method.
Settling time	about 2 seconds	about 3 seconds
Steady state error	close to 0%	close to 0%
Overshoot	0%	2.1%

table 1: Metrics of the two controllers deployed in the complex simulink model.

In all of our metrics in table 1. The manually tuned PI controller was better than the auto tuned PID controller. The difference of 1 second is noticeable, but considering that the process of making the autotuned controller was running 1 test and using 3 parameters ($a, d \; and \; T_u$) to instantly create the controller this is a quite positive result.

One could imagine that if a plants behavior changes over time (for example the gears of a system loosening over time)
It could be an advantage to have an auto tuner in place, since the controller can be calibrated more frequently, and the parameters will be relevant to the current behaviour of the plant.
Meaning over a longer time span the autotuned controller may be better since the controller will change with the plant over time.

Further work

For further work on the case study, the implementation of a system to automatically apply the parameters calculated in the relay process, to an implemented PID controller in the same model. This would divide the controller into two parts, a calibration part (running the relay feedback test), and the PID controller using the parameters from the most recent calibration. Automating this process would make the implementation process of the autotuned PID, more seamless.

This could also have potential in the context of adaptive tuning, where a system might have varying behaviour depending on the external environment. As mentioned in the general discussion section, this auto-tuning method may not be preferable if the behaviour of the plant varies. The implementation of a detection algorithm for controller efficiency, combined with a callable calibration process could have great potential.

Conclusion

In this case study we tasked ourselves with creating a PID auto-tuner using a relay feedback system. First implemented this on a simple simulink model of the car and then on the more complex simulink model. In addition to this we also created a manually tuned PID controller for comparison. Both controllers were quite good.
The manually tuned controller had no overshoot and a settling time of about 2s
The autotuned controller had a settling time of about 3s and an overshoot of about 2%.

Meaning the manually tuned controller is objectively better, this however comes at the cost of time, since it was tuned manually.

The big advantage of auto tuning is that no man hours are needed, and if the system changes over time the controller can easily be autotuned again. Also all autotuners aren't made equal, we implemented the most basic relay feedback autotuner, and still got a quite good result, a more sophisticated autotuner would likely perform about as good or better than the one we tuned manually.

References

[1] Relay Feedback Auto-tuning of Process Controllers – A Tutorial Review. / Hang, C.C.; Åström, Karl Johan; Wang, Q. G.
In: Journal of Process Control, Vol. 12, No. 1, 2002, p. 143-162.
https://slunik.slu.se/kursfiler/TE0010/10095.1213/Reg1TuneReview.pdf

[2] Autotuning of a PID-Controller / Anderson C, Lindberg M.
http://lup.lub.lu.se/student-papers/record/8848024
https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8848024&fileOId=8859436

[3] PID Controllers: Theory, Design, and Tuning 2nd edition. Åström K, Hägglund T
https://aiecp.wordpress.com/wp-content/uploads/2012/07/1-0-1-k-j-astrom-pid-controllers-theory-design-and-tuning-2ed.pdf

[4] Reguleringsteknik *Ole Jannerup og Paul Haase Sørensen, Polyteknisk Forlag, 5. udgave, ISBN: 978-87-502-1105-1*

[5] Ziegler, J.G & Nichols, N. B. (1942). "Optimum settings for automatic controllers"

[6] Relay Methods on Auto-Tuning Automation Solutions. Marco Gonçalo de Sousa Neves.
https://fenix.tecnico.ulisboa.pt/downloadFile/395139427476/Resumo%20Alargado%20-%20Auto-tuning%20de%20Controladores%20PID%20pelo%20m%C3%A9todo%20Relay.pdf