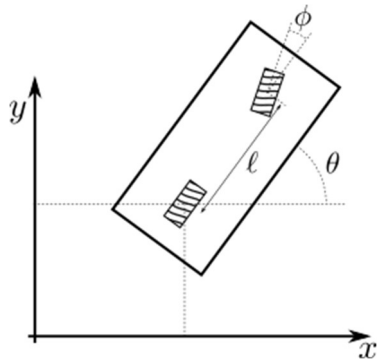
TRAJECTORY TRACKING CONTROL OF A CAR-LIKE ROBOT

Alessandro Puglisi, 10696892

PROBLEM STATEMENT

Consider the following car-like robot, simulated using its bicycle kinematic model, or (for a more reliable simulation) its single-track dynamic model (Linear or Fiala tyre model), rear-wheel drive :

KINEMATIC MODEL

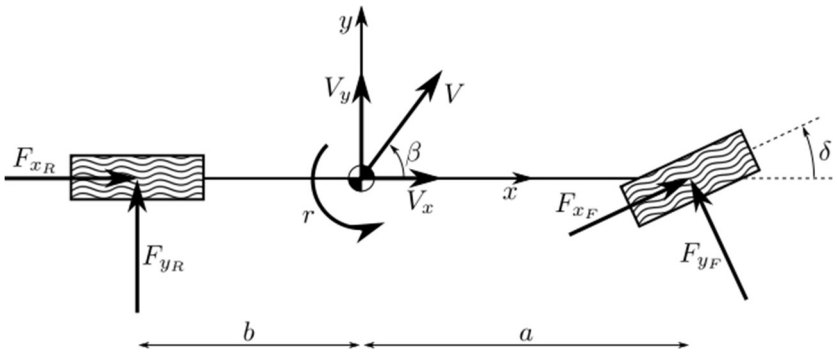


$$\begin{cases} \dot{x} = V\cos(\theta) \\ \dot{y}_V = V\sin(\theta) \\ \dot{\theta} = V\tan(\phi)/l \end{cases}$$

With V and ϕ Control input

MODEL PARAMETERS
m = 0.5 Kg
a = 0.14 m
b = 0.12 m
(l = 0.26m)
μ = 0.385
$C_{\alpha f}$ = 50 N/rad
$C_{\alpha r}$ = 120 N/rad
I_z = 0.023 Kg*m²

DYNAMIC (+ kinematic) MODEL (linearized with small angles hypothesis)



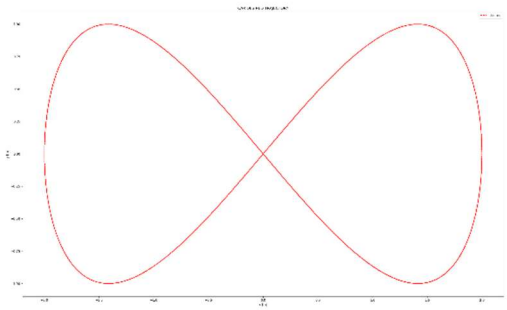
$$\begin{cases} \dot{r} = (aF_{yF} - bF_{yR})/I_z \\ \dot{\beta} = \left(F_{yF} - F_{yR}\right)\cos(\beta)/(mV_x) - r \\ \dot{x} = V_x\cos(\psi) \\ \dot{y} = V_x\sin(\psi) \\ \dot{\psi} = \mathrm{r} \end{cases}$$

Still control at velocity/steer level with control input V_x and δ (same as ϕ and V used before)

F_{yF} and F_{yR} according to the used tyre model (Linear or Fiala)

OBJECTIVE TRAJECTORY

Our goal is to design a PI trajectory tracking Controller to follow the following trajectory:

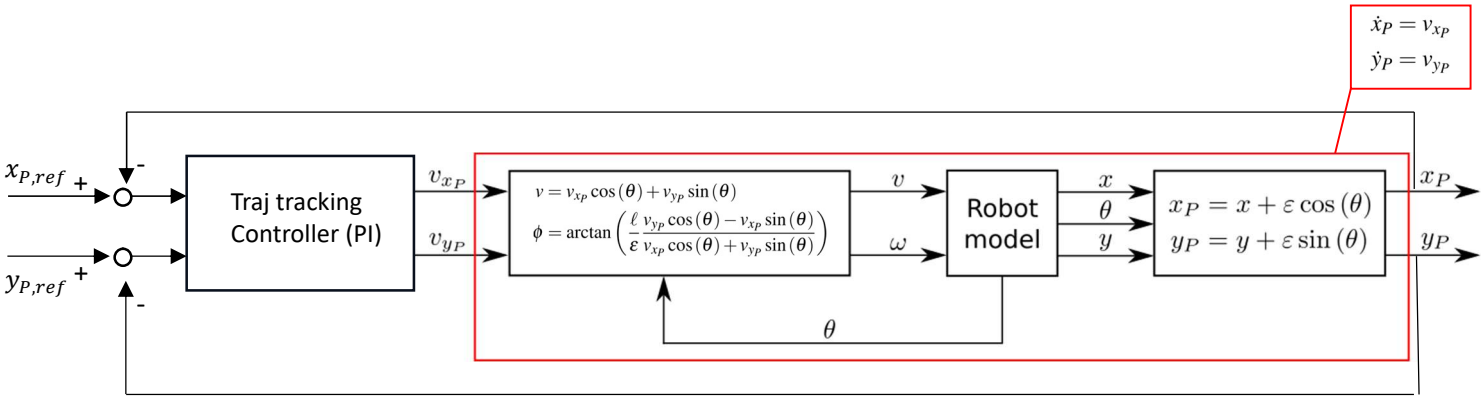


TRAJ PARAMETERS
T = 2.7 s
amplitude = 2 m

CONTROL SCHEME

For the defined models of our robot, I can implement a **feedback linearization based control**, this allow to make the kinematic model behaves like a couple of independent integrator at velocity level, with respect to a user chosen point P (laying in the velocity line).

Notice that for the dynamic model the feedback linearization law (based on bicycle kinematic model), guarantee just a partial linearization, the non linearity of the dynamic model is not completely negligible. Anyway, controlling at velocity level for the auxiliary point P, I can implement the controller as before, but taking into account the more "realistic" behavior, with adjustment on the controller parameters accordingly to simulations (iterative tuning).



So, I can approach the control of x_P and y_P independently.
(As an overall, the linearized system behaves like two decoupled integrator, there is no reason to approach the control problem with a centralized multivariable control design)
This reduce the control problem to a simple design of a controller for an integrator (with unitary gain) with control input v_{xp} and v_{yp} respectively.
Notice that, the scheme above neglect the feed-forward effect that is added to v_{xp} and v_{yp} with an analytical computation of $dx_{P,ref}$ and $dy_{P,ref}$. This does not affect stability, and the performance from now on can be analyzed without considering the effect of this term, so I will refer to "PI Control" omitting that, in reality, it is a **PI+ feed-forward controller**
(this ensure a 0 tracking error even in case of complex set points, more reactive control)

CONTROLLER TUNING

I tune the controller according to this hypothesis, with a simplified tuning procedure, in which I can exploit the PI controller and choose the control objective accordingly to some considerations in the frequency domain.
An important parameter is ω_C which is the cut-off frequency, this set the **bandwidth** of the system (this affect the "speed" of the closed loop dynamic, higher for an higher ω_C).
A PI controller in this case ensure enough degrees of freedom to properly place ω_C as desired, through a tuning procedure on the controller parameter K_P and T_i (same for x and y, no reason to do an asymmetric control of the coordinates).

Before seeing in detail the tuning procedure, based on some consideration in frequency domain, Let's define the general expression of the Controller in Laplace domain (useful for the analysis in frequency with the Bode plot)

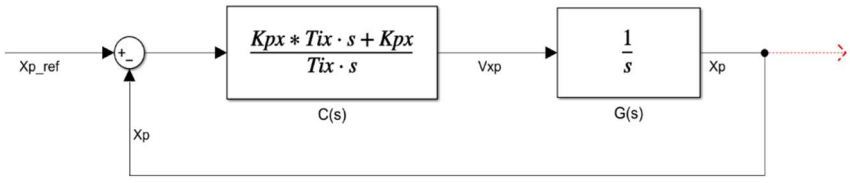
Time domain control law:

$$v_{Px} = K_{Px}\big(x_d(t) - x(t)\big) + \frac{K_{Px}}{T_{ix}}\int_0^t \big(x_d(\tau) - x(\tau)\big)d\tau$$

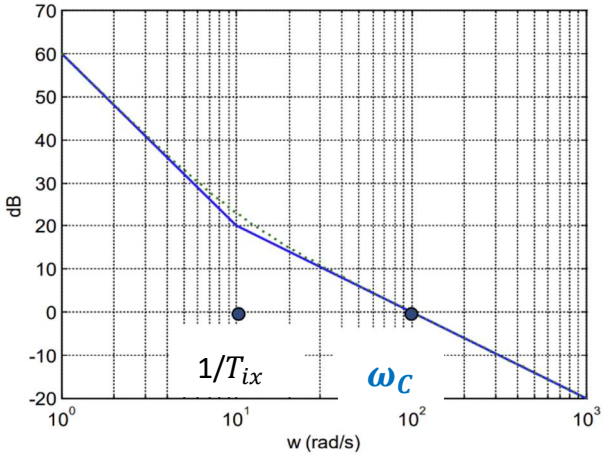
- Here feed forward term neglected
- Integral part is implemented using forward Euler discretization)
- Same formulation for yp

Frequency domain controller (error to velocity TF):

C(s) = $K_{Px}(1 + \frac{1}{sT_{ix}})$ used to control G(s) = 1/s (velocity to position TF)



I can identify the Loop transfer function L(s), very useful in control theory in order to study stability and performance of the system in closed loop as L(s) = $\frac{K_{Px}}{s}\frac{1+sT_{ix}}{sT_{ix}}$, with the following Bode diagram approx:



As can be noticed, in high frequency it is well approximated by $\frac{K_{Px}}{s}$, meaning that: $\boldsymbol{\omega_C \cong K_{Px}}$

In order to enforce that, the zero placed by the choice on T_{ix} should be placed at least one decade before the cut-off frequency. So as rule of thumb I can take $\boldsymbol{1/T_{ix} \cong 0.1\omega_C}$

Notice that, the additional integrator added with the controller to the loop (even if it comes with a cost in terms of stability) it is very useful. In fact, if with the feedback linearization I don't obtain an ideal integrator, but an approximated one, in that case there is no guarantee of convergence of the error. Therefore that Integral action add robustness to The control system.

Also, the integral ensure an approximated unitary gain of the closed loop TF in steady state, so a "perfect tracking of the set point at regime", where the regime is reached approximatively in $5/\omega_C$ (settling time approximation by closed loop approx)

For the controller tuning, in a trajectory tracking problem a bandwidth approximatively of a human driver (10÷15 Hz), theoretically would guarantee good results.

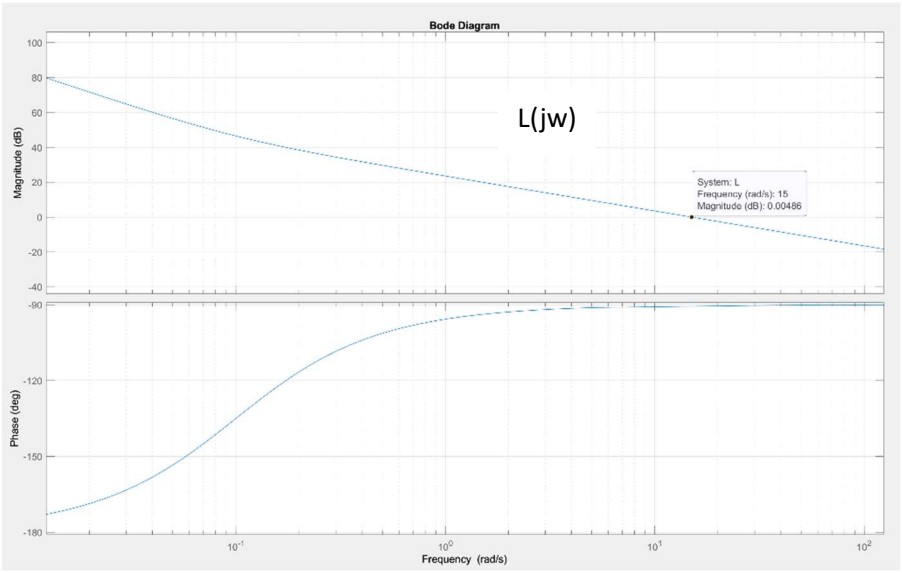
(also considering the duration of our trajectory period, the decision frequency has no need to reach hundred of Hz)

So at first approximation I try with this assumptions for the control tuning process:

$\boldsymbol{K_{Px} = K_{Py} = 15}$

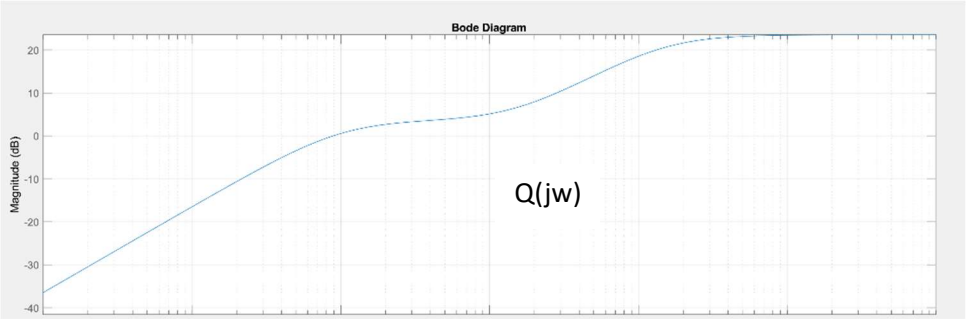
$\boldsymbol{T_{ix} = T_{iy} = 0.667}$

Let’s also check if with this controller the frequency theory assumption holds:



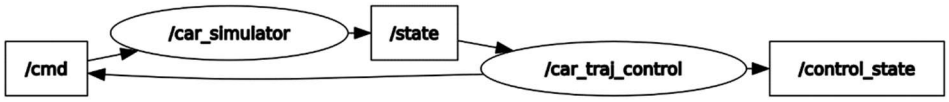
The approximation is met perfectly (this is in the hypothesis of perfect linearization of the robot model obviously), and also stability is achieved robustly for bode criterion (phase margin of 90°)

Another important aspect of the control system is the control effort, I have to take into account that the actuators have saturation limits. (limit on maximum achievable steering rate and linear velocity, that we cast into proper limit in Vxp and Vyp)
The sensitivity of control TF Q(s) = C(s)/(1+L(s)) is representative of the required control action, representing the reference to control action TF.
From its bode plot with the controller designed, I can check that it is well bounded with a maximum linear gain of 10 (reasonable)



Now It is time to test the controller on the overall system, and see if some correction has to be done on the tuning. I expect to obtain good results in the kinematic model where the linearization is perfect (and uncertainty in the simulation environment does not occurs). While some troubles can arise on the dynamic model (maybe less on the linear tyre one, but some in Fiala where the dynamic modelling is more complete). This call for some iterative tuning procedure or a correction taking somehow into account dynamic model.

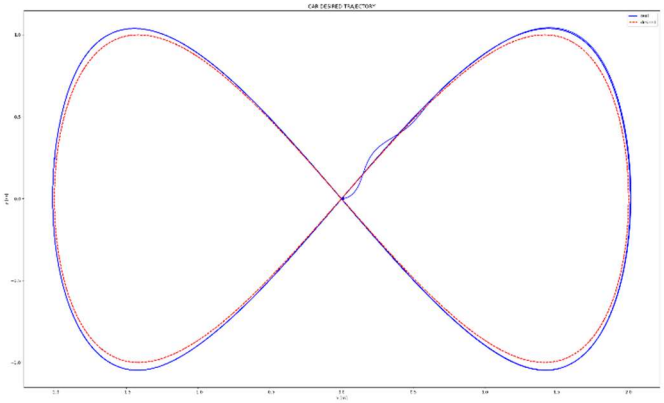
The control performance simulation will be done in ROS, where one simulation node has the rule of simulating the car-like robot (with bicycle/single track model), using the odeint library in order to integrate the kinematic/dynamic model. That node publish the information of the current robot state, and take input from a control node, which internally perform both the trajectory generation (analytical path generation), feedback linearization and control law computation (accordingly to our controller tuning).



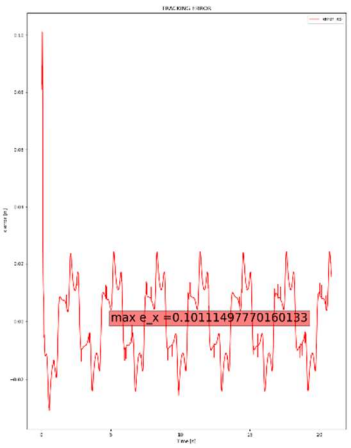
The implementation highlight another aspect of the control system, the real controller obviously is discrete time (control action computed by an algorithm running on a proper control board with its computation unit), so a parameter that affects the control performance is the sampling time **Ts.**
(In principle also the simulator itself is a discretized approximation of the real word, but with an integration frequency of 1kHz we can assume that the "sampling" is enough accurate to represent the real system dynamic)
Testing the system with fs = 100, 1000 Hz (Ts = 0.01, 0.001) I will probably notice that an higher sampling frequency of the controller ensure better performance with a finer control.
The choice will be affected by the ADC on the control board, a frequency of 1kHz is easily guaranteed.
Finally the simulation can be performed.

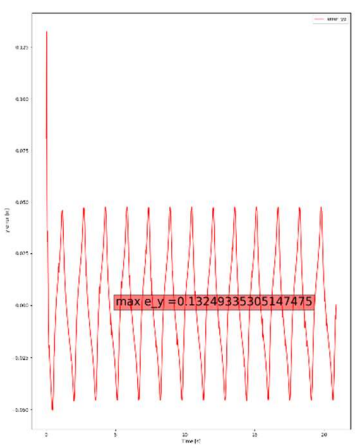
BICYCLE KINEMATIC MODEL

First of all, motivation for the **choice of the sampling time Ts** can be shown:



Ts = 0.01 s the overall tracking error is bounded, and the trajectory seems already good accordingly to our performance specification, but a more reactive control can bound more the error





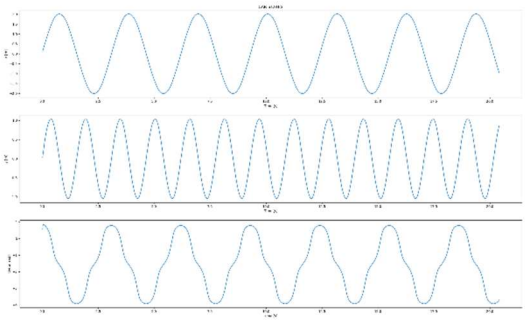
Notice also that, this is not a simple control problem with a constant reference, the fact that the reference is time varying imply that the reactiveness of C(s) guarantee a well bounded error, but we obviously cannot expect e(t)=0 from a certain time.

Ts = 0.001 s

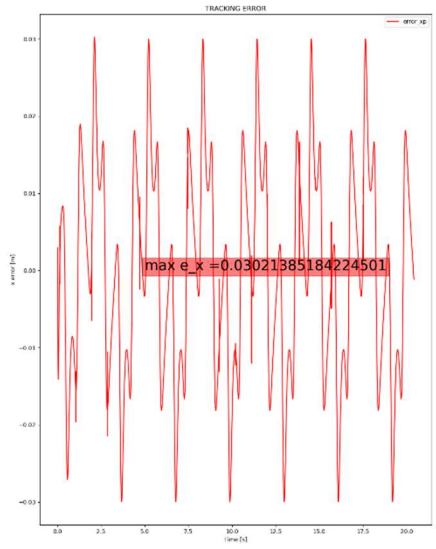
Simulations done in an interval of 20 seconds:

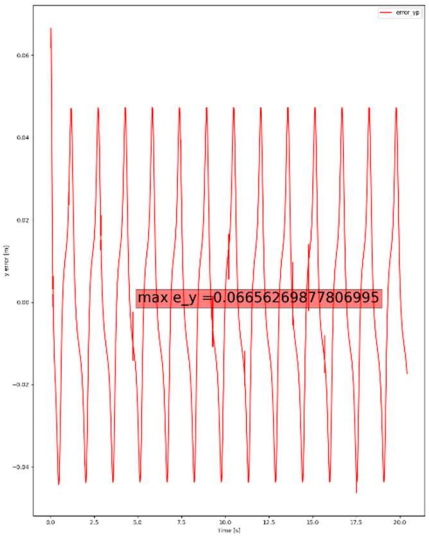
ROBOT TRAJECTORY

ROBOT STATES

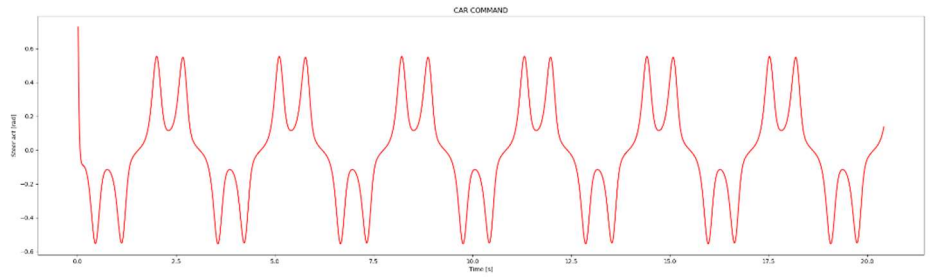


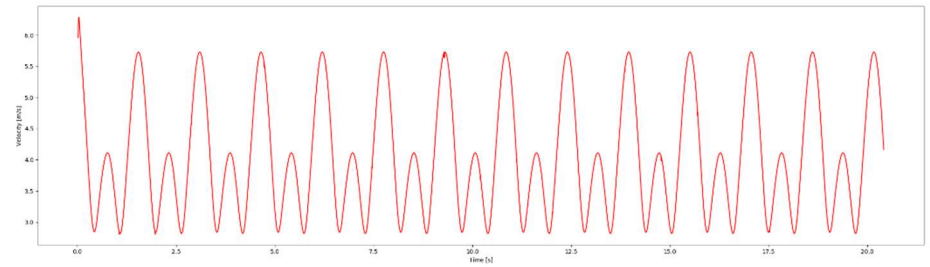
TRACKING ERRORS





The achieved performance are very good, with a maximum error of: $e_x = 0.032$, $e_y = 0.066$

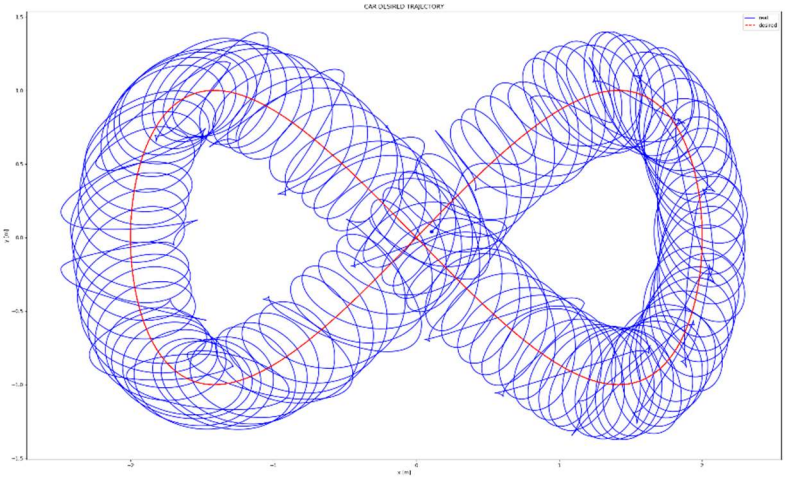


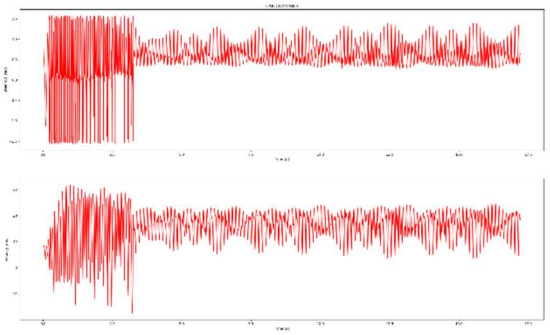


Also the command action is well bounded, with a maximum of 6 m/s of linear speed and 0.5 rad/s of steering. Reasonable actuators command in a real setting.

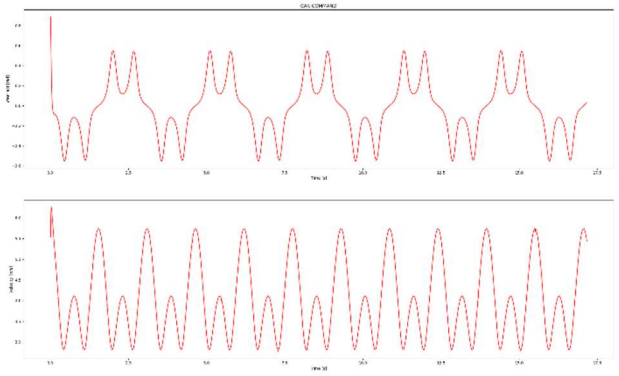
So, the assumptions done in frequency domain proves to be right in the simple kinematic model. There is no need to correct the controller by this kinematic model simulations.
Even notice that if we try to speed up to much the system, the performance are lost because the control action become too aggressive.
In principle,by an iterative tuning of the controller parameters we expect:
Fixing Kp, if Ti increse the zero get closer to the origin, with the risk of “canceling out” the integral action introduced by us, while decreasing Ti move the rigth and the approximation on the cut-off frequency becomes no more reliable, at the limit the zero could even be moved over ω_C, affecting the phase margin obtaining a less robust system.
Fixing Ti, if Kp increase we “move up” the magnitude diagram, increasing directly ω_C, this speed up the system but it can increase also the actuation cost and spikes without any need (trivially a decrease over Kp decrease the bandwidth affecting tracking performance).
Here are some examples of the above theoretical assumptions:
(using Tix = Tiy = Ti, Kpx = Kpy = Kp)

Kp = 15, Ti = 0.00667

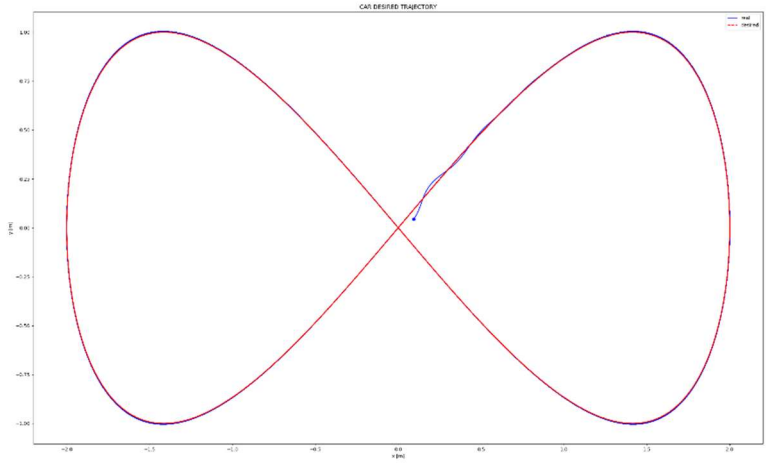




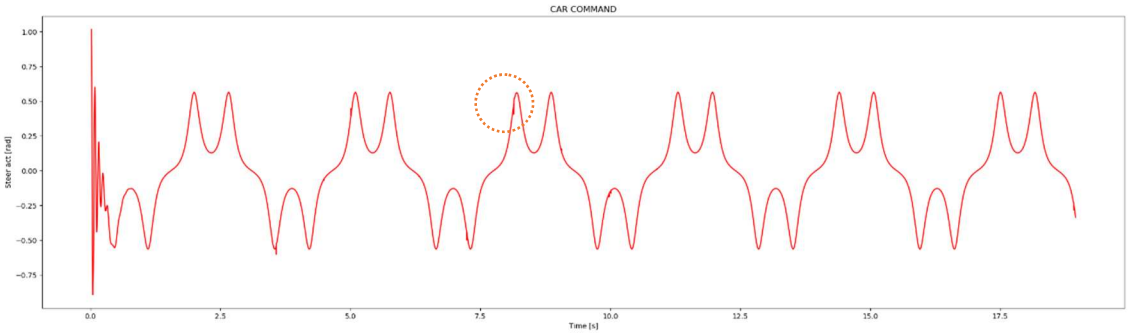
Kp = 15, Ti = 66.67



Kp = 150, Ti=0.667

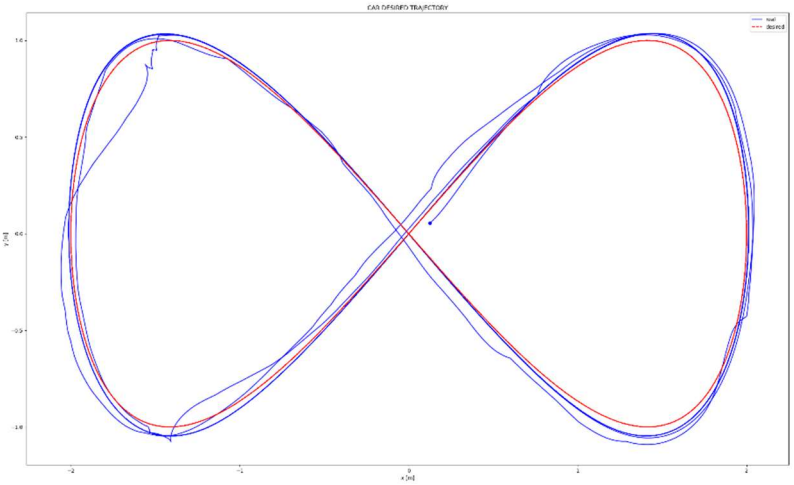


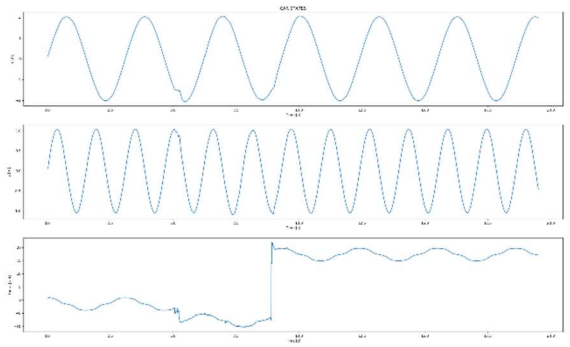
Here the tracking is almost perfect, and the control action is not even too high, but some spikes show in the actuator commands, that can be unfeasible due to the obvius rate limitation in the real control action.
Also, an increase of the bandwidth will possibly include and even amplify high frequency disturbance.

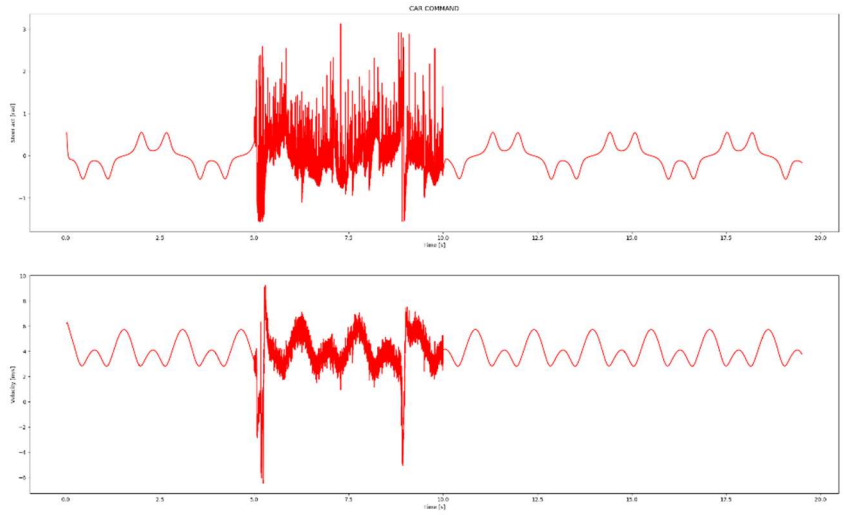


A final simulation test that can be done to prove the quality of the performances is to add a white noise to the V, phi control action and see what happen to the system:
(Here a white noise of 0 mean and standard deviation 0.5 is added (squared for the steer) in the 5÷10 sec

Kp = 15, Ti = 0.667





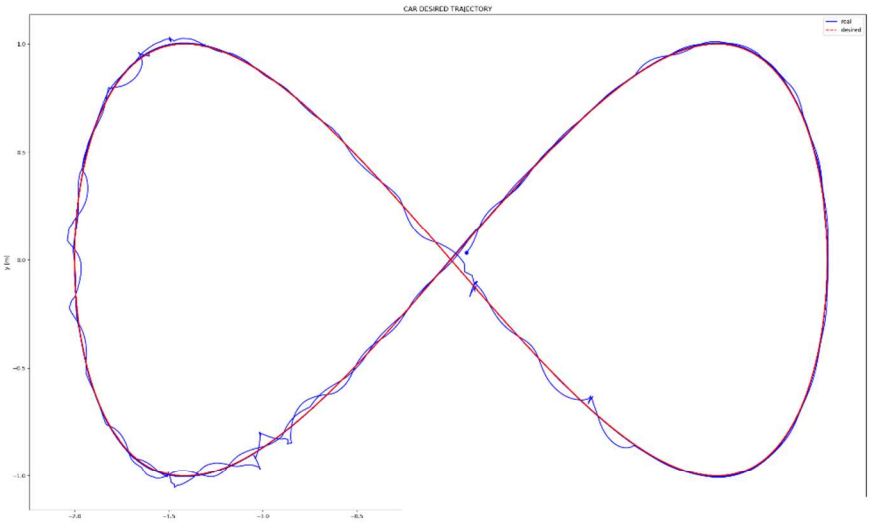


Obviously the control action in order to reject this disturbance becomes more demanding, but still somehow “reasonable”.

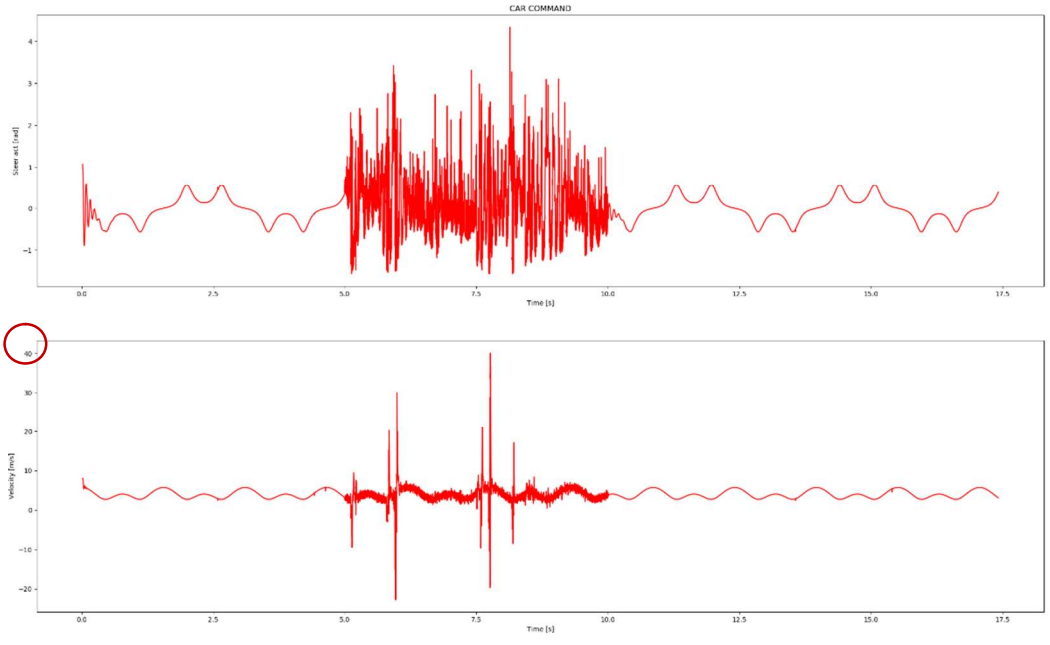
Notice also that after the disturbed interval the system is able to come back to the good tracking behavior without going too far away from its track.

We can see how if we set Kp=150, as expected in order to stabilize the system as desired the control action becomes much more demanding

Kp = 150, Ti = 0.667



Careful on the requested control action,
for sure no sense for our small robot.



Final Choice: In light of the simulations done and on the previous observations, we stay with the initial control tune of Kp = 15, Ti = 0.667 both in xp,yp controller

Note on the kinematic simulations:

car_traj_control_kin.launch (car_ws/src/car_traj_control/launch/car_traj_control_kin.launch) in order to run the simulation with the chosen Kp,Ti (defined in car_traj_control_kin.yaml), the white noise test has been commented in the car_traj_control.cpp file to not affect simulations..

plot_result.py (car_ws/src/car_traj_control/script/plot_result.py) for the plots, called in the terminal with the name of the bag file where ros messages of simulation have been stored

But, all these observations in case of the simple kinematic model proves to be overkill. Let's see what happen when we include the dynamical model.

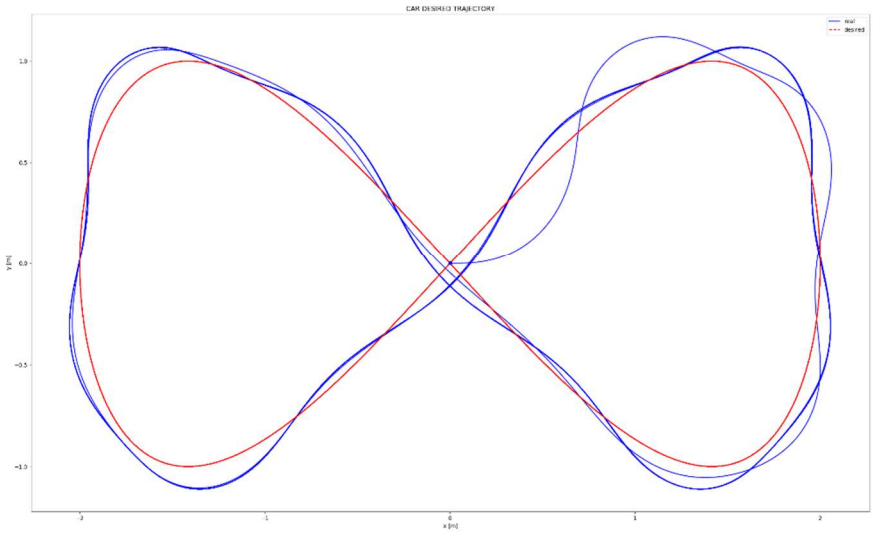
SINGLE-TRACK DYNAMIC MODEL, LINEAR TYRE MODEL

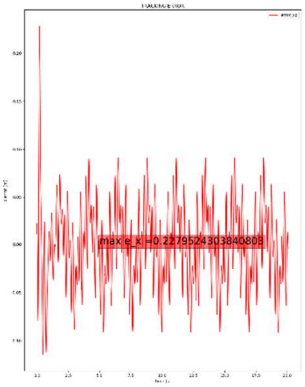
Linear tyre model uses the linear approximation on the tyre forces:

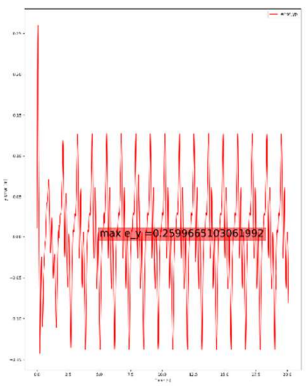
$$F_{y_F} = -C_{\alpha_F}\alpha_F \qquad \alpha_F = \beta + \frac{ar}{V_x} - \delta$$

$$F_{y_R} = -C_{\alpha_R}\alpha_R \qquad \alpha_R = \beta - \frac{br}{V_x}$$

Kp = 15, Ti=0.667

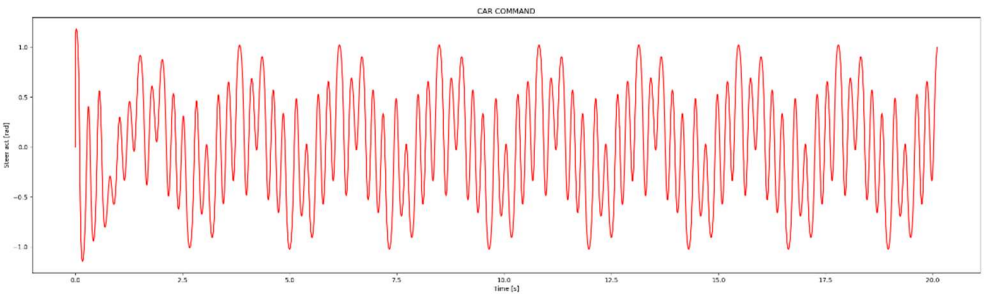


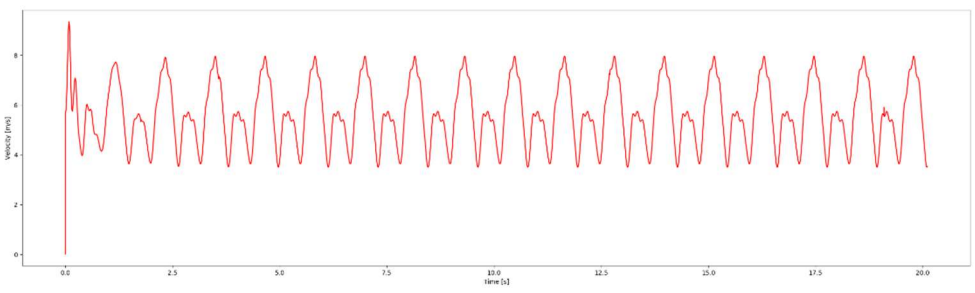




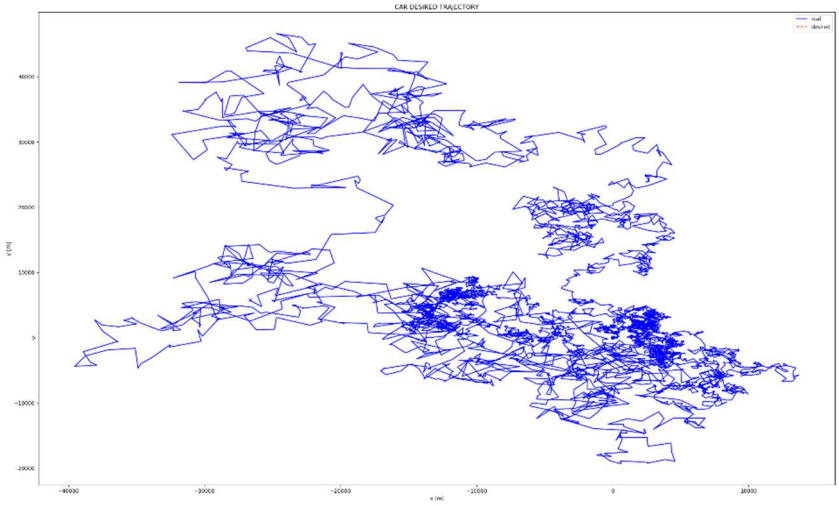
The controller used for the kinematic model now is not enough, the trajectory is a little bit too wavy, and the error is not well bounded.

With the same sensitivity comments done before about Kp and Ti (even if the dynamic model is more complex than before and the linearization is not full, in approximation it still holds), let's try with an iterative procedure to find a better tuning for the trajectory tracking controller





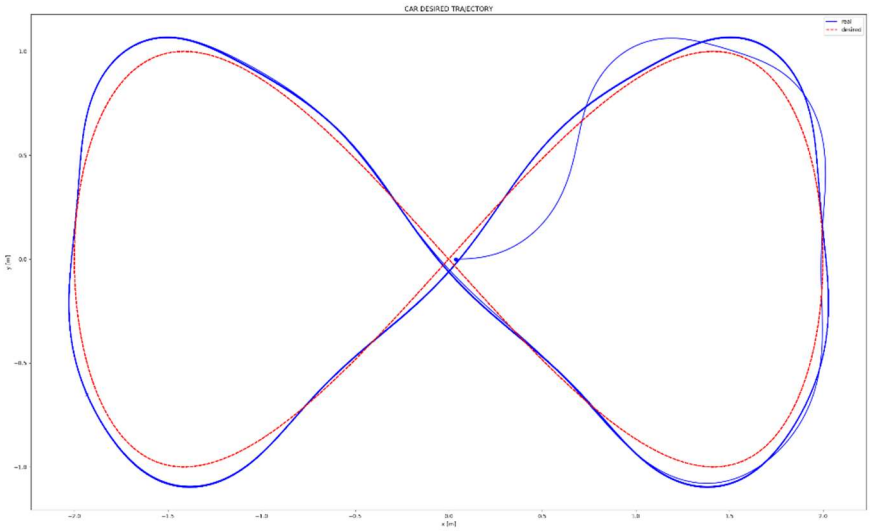
Kp = 30, Ti=0.5

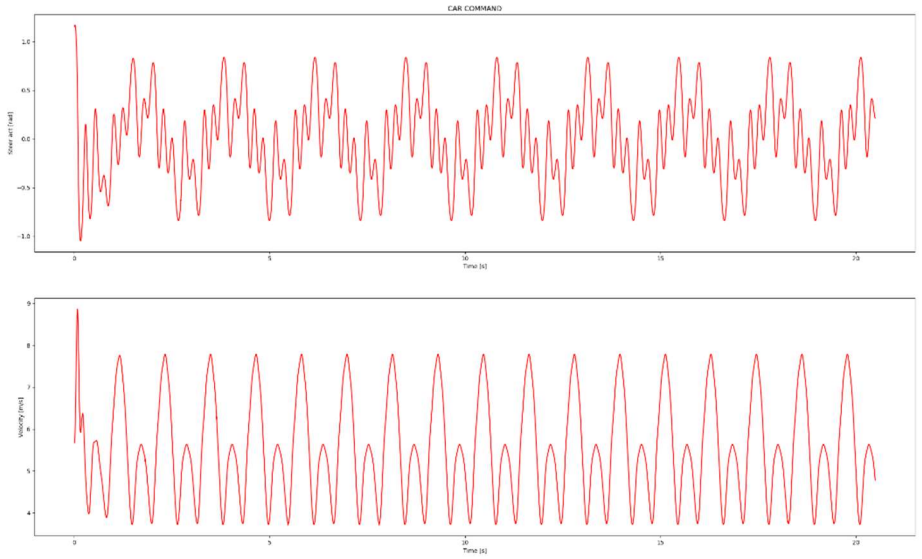


Working in higher frequency the model degenerate and our control theory in frequency domain does not holds anymore. This show us that the proportional gain should kept limited, in general avoid to exploit higher frequency with high proportional gain.

We can work decreasing Kp and increasing Ti in order to increase the integral action (higher action with respect to the cumulative error with respect to the current error)
First look at how a stronger integral action works in our case, taking care of reducing also the proportional one to not enhance high frequency too much

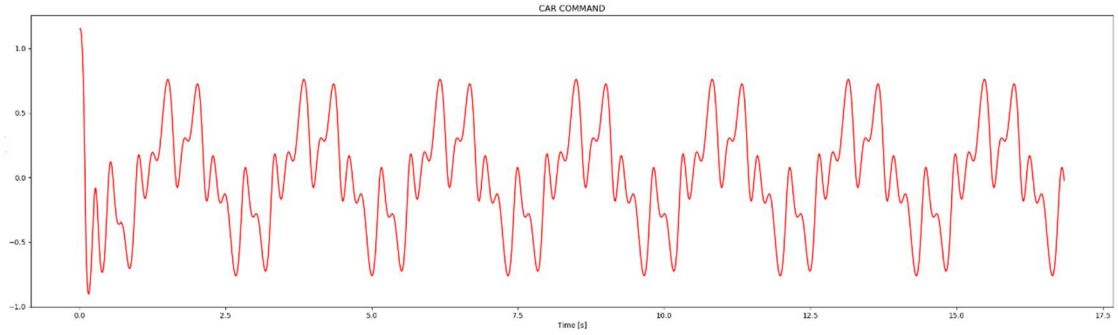
Kp = 12, Ti=10

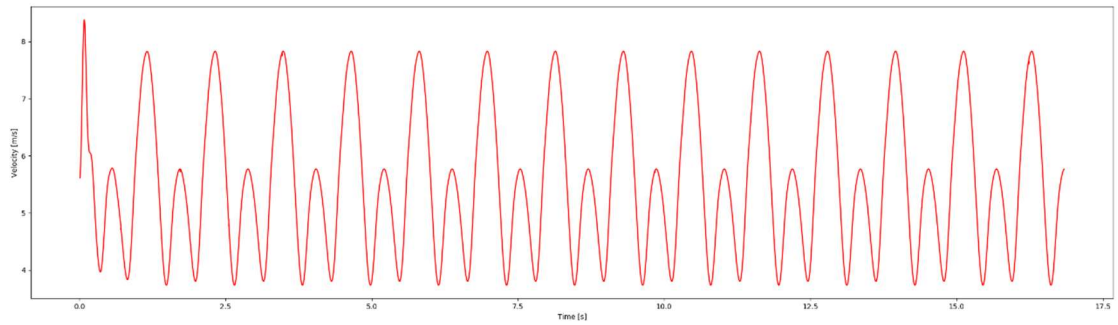




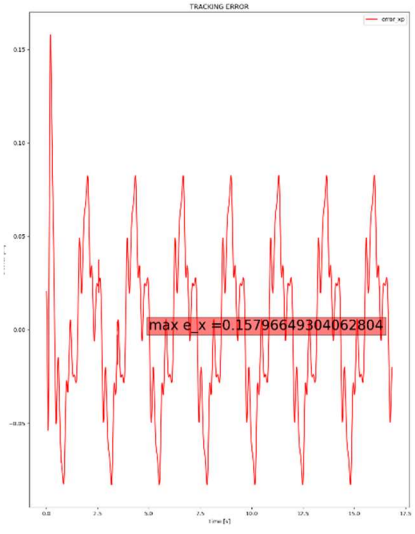
Final choice: Kp = 8, Ti= 15

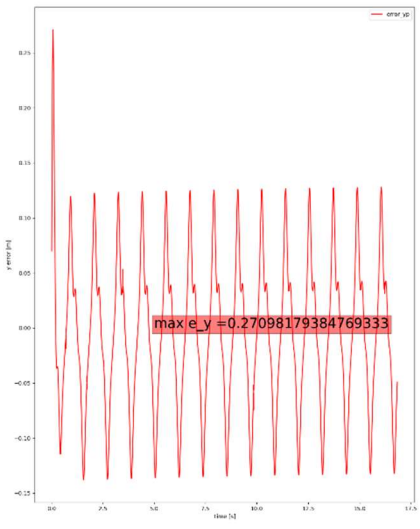
This solution is enough satisfactory for our goal, even if it has a small shift with respect to the desired one, the control effort is well bounded and the trajectory achieved is satisfactory as we can also see from the error plot.



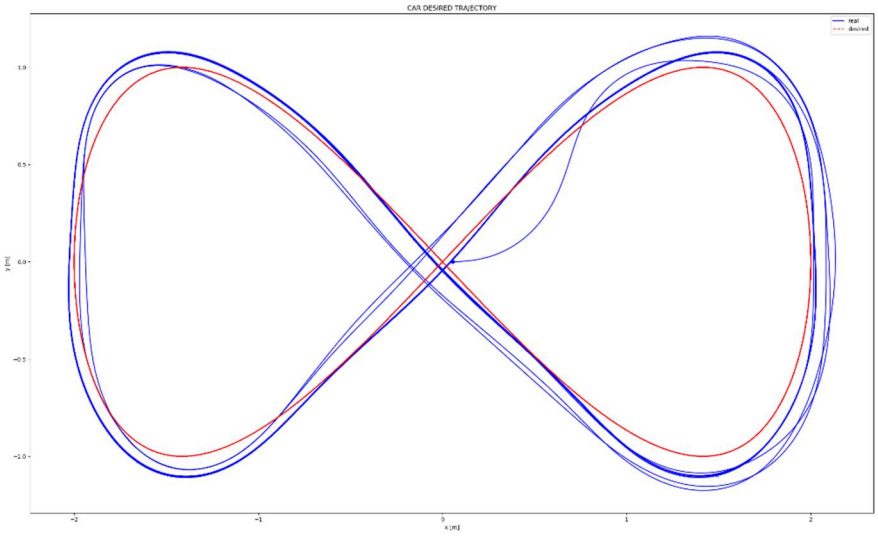


The additional integral action somehow affect the behaviour in presence of errors in the control action? (maybe caused by a change in the terrain that affect the speed/steer with some gaussian noise)
I can simulate the behaviour in presence of the same noise as before..

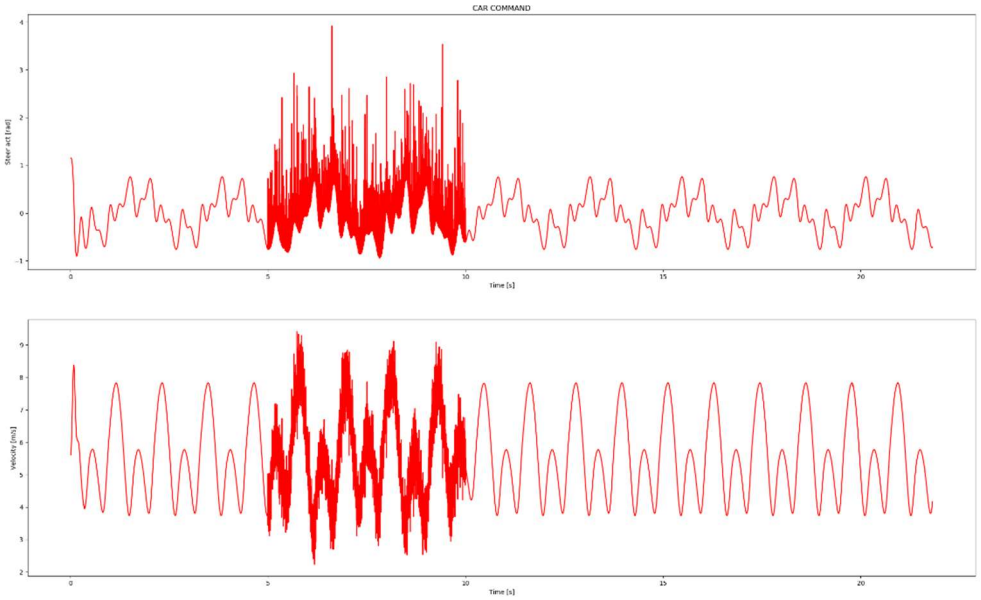




With control additive gaussian noise (as done for kinematic simulation)



Obviousle for the 5 seconds in which noise acts on the actual control action the trajectory deviate, but than comes back to the nominal performance, without a diverging trajectory.



In conclusion, with respect to the single-track dynamic model with linear tyre model, a PI+feedforward with **Kp=8.0 and Ti = 15.0** (both in xp,yp) is satisfactory.

Notice that probably, with a complete feedback linearization, and so linearizing also the dynamic model we would be able to achieve better result. In particular, if the required speed increases (and so the trajectory period decreases) the kinematic would not be enough representative of the system behaviour, with performance getting worse proportionally to how much we decrease trajectory period T (and a request of higher bandwidth in closed loop in order to maintain good performance). While the control with the dynamical model, tuning the controller in a better way according to the dynamic model, can exploit better the real system dynamic.

Note on the single track dynamic with linear tyre simulations:
car_traj_control_dynlin.launch (car_ws/src/car_traj_control/launch/car_traj_control_dynlin.launch)
in order to run the simulation with the chosen Kp,Ti (defined in car_traj_control_lin.yaml), the white noise test has been commented in the car_traj_control.cpp file to not affect simulations..
plot_result.py (car_ws/src/car_traj_control/script/plot_result.py)
for the plots, called in the terminal with the name of the bag file where ros messages of simulation have been stored

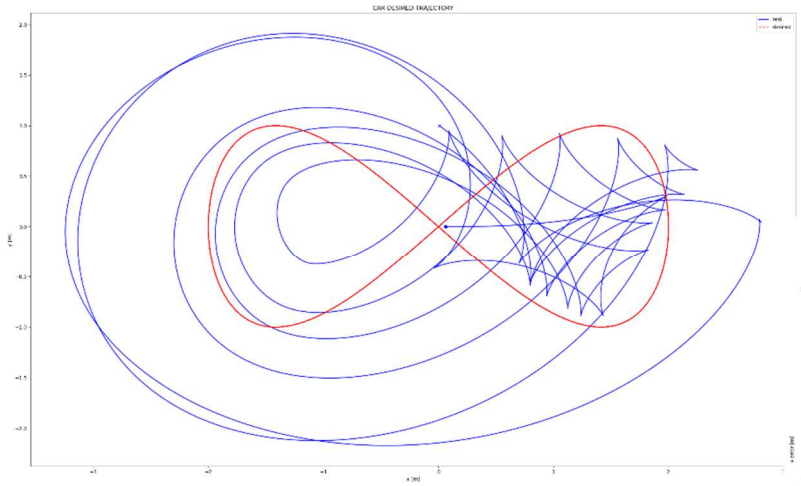
In general the linear tyre model can be not enough to fully capture the real behavior (it is a simplification of the Pasejka magic formula around small slip angles), that's why for control purposes we can use a still managable model but more complete: the Fiala tyre model (here used with the saturation taken into account)

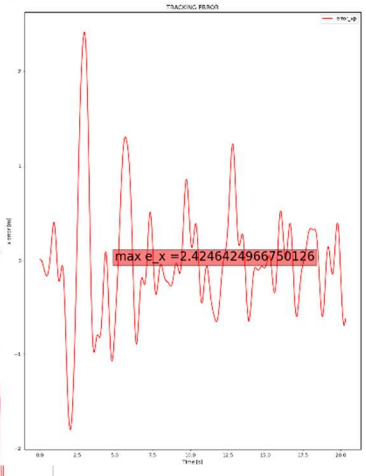
SINGLE-TRACK DYNAMIC MODEL, FIALA WITH SATURATION TYRE MODEL

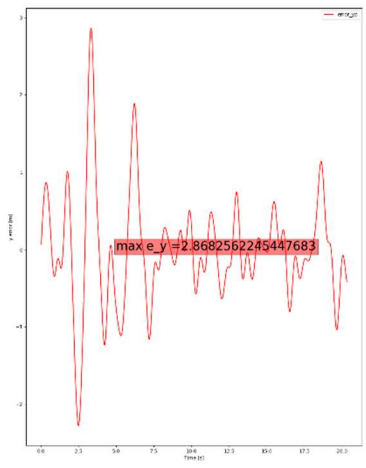
$$F_y = \begin{cases} C_\alpha z\left(-1+\frac{|z|}{z_{sl}}-\frac{z^2}{3z_{sl}^2}\right) & |z| < z_{sl} \\ -\mu F_z \text{sign}(\alpha) & |z| \geq z_{sl} \end{cases} \qquad (z = \tan\alpha \qquad z_{sl} = 3\mu F_z / C_\alpha)$$

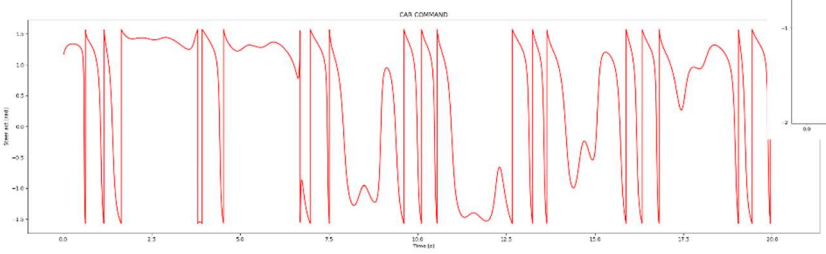
Both front and rear

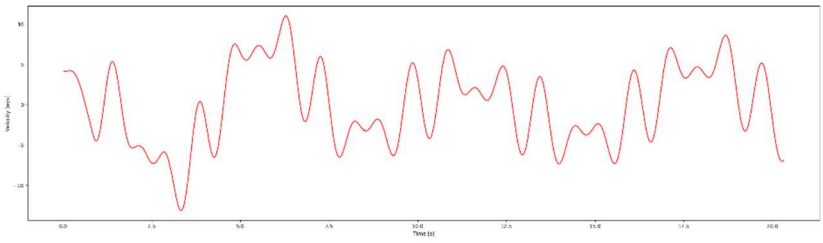
(Notice also that in this case the dynamic of the model is enhanced by the fact that the period of the trajectory T is reduced of the 25%)
Starting the simulation testing the same controller used for the linear model:
Kp = 8, Ti=15

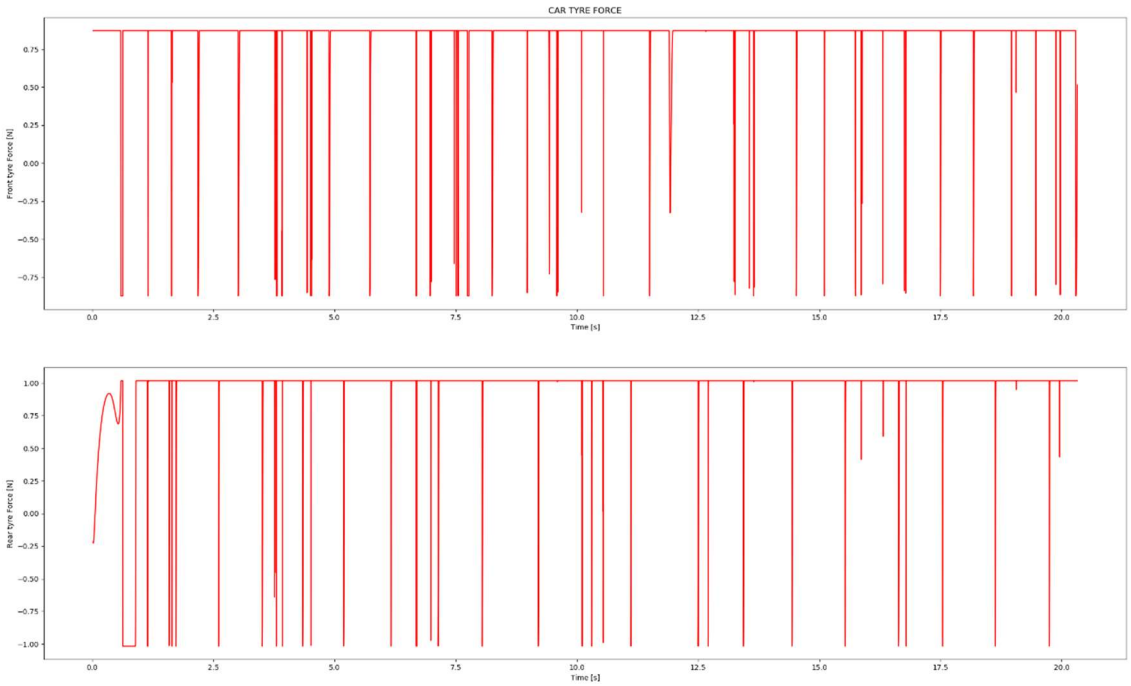






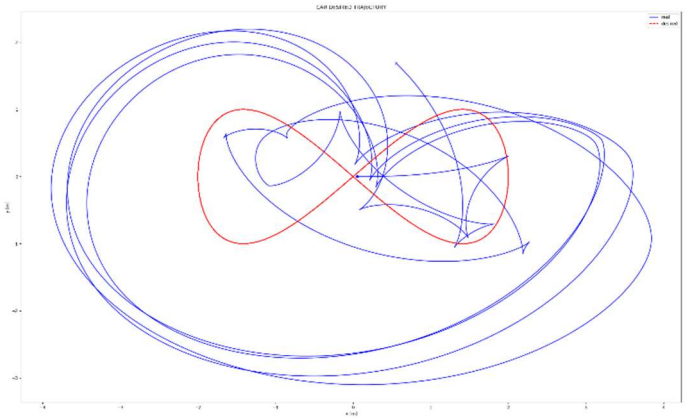




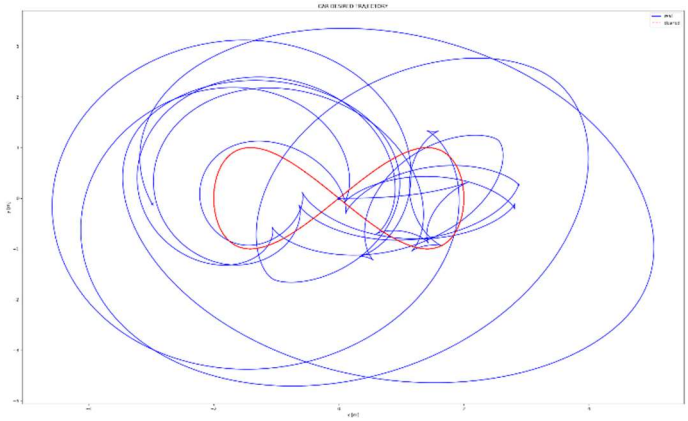


We can clearly see how saturation (which is related by the friction limit, so it has to be included for a correct model) continue to act on the system, in which the trajectory achieved is clearly not satisfactory.
But we can see that a simple iterative tuning done as before, does not provide satisfactory solution:

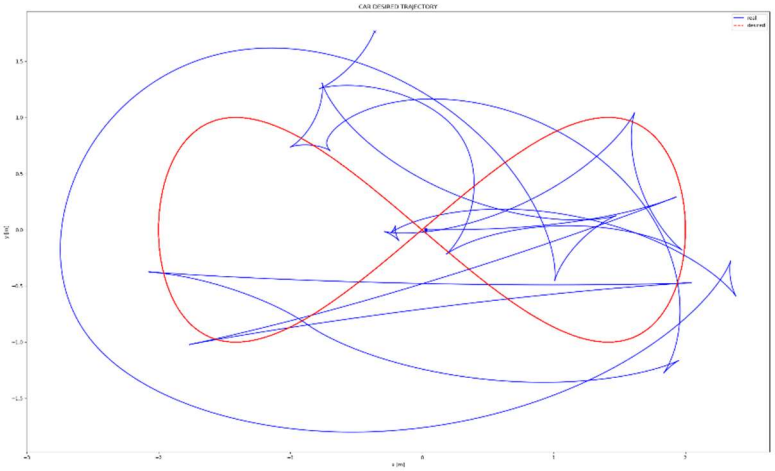
Kp = 10, Ti=20



Kp= 20, Ti=3

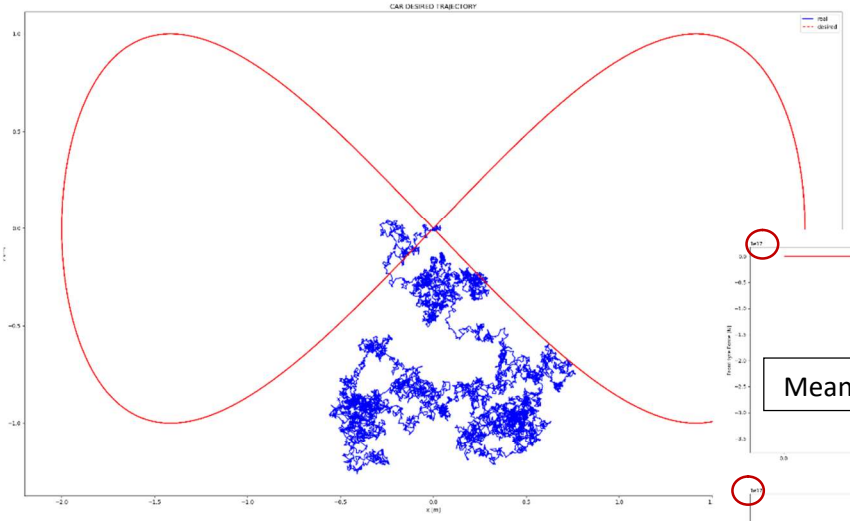


Kp = 4, Ti = 10

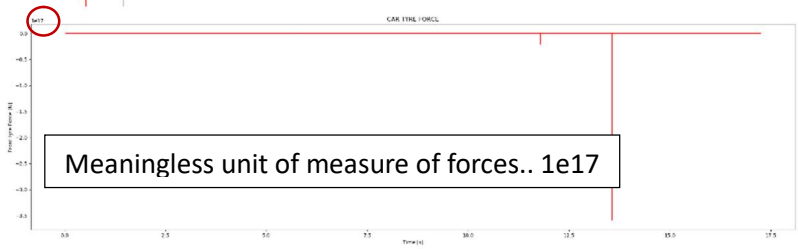


The full dynamic model (+ kin) controlled with our kinematic feedback linearizer is not a good control approach, as a sensitivity analysis show us.
Even more in this case where the required speed is higher (as we can also see from the car command plot above), the dynamic become more dominating.

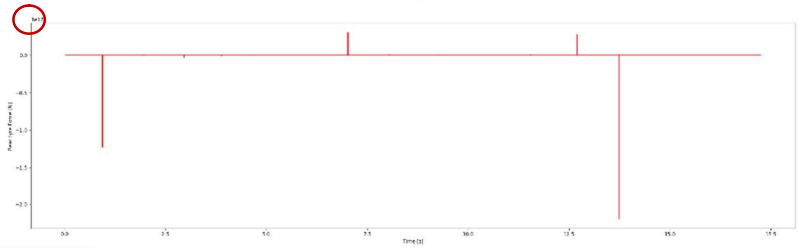
The simulation without the saturation on the tyres is even worst:

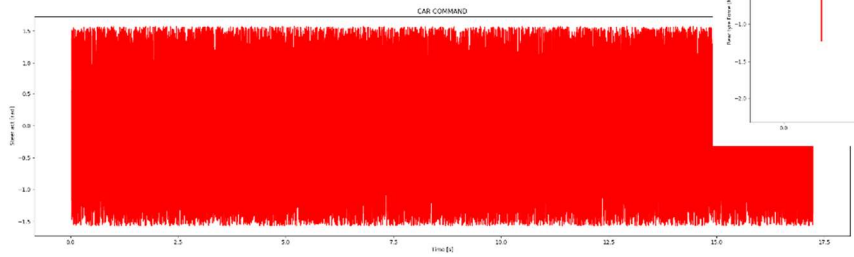


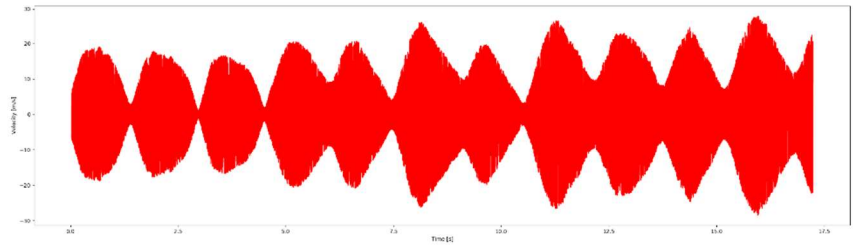
Kp = 8, Ti = 15
Fiala without saturation



Meaningless unit of measure of forces.. 1e17







Note on the single track dynamic with linear tyre simulations:
car_traj_control_dynlin.launch (car_ws/src/car_traj_control/launch/car_traj_control_dynlin.launch)
in order to run the simulation with the chosen Kp,Ti (defined in car_traj_control_lin.yaml), the white noise test has been commented in the car_traj_control.cpp file to not affect simulations..
plot_result.py (car_ws/src/car_traj_control/script/plot_result.py)
for the plots, called in the terminal with the name of the bag file where ros messages of simulation have been stored

Final note on the uploaded file:
-The overall Ros workspace has been uploaded as indicated, the single packages code is inside /car_ws/src as usual
-yaml file of configuration has been duplicated with different names just for a fast testing, no need of repeating this file