Control of a [2,0] Mobile Robot

Introduction

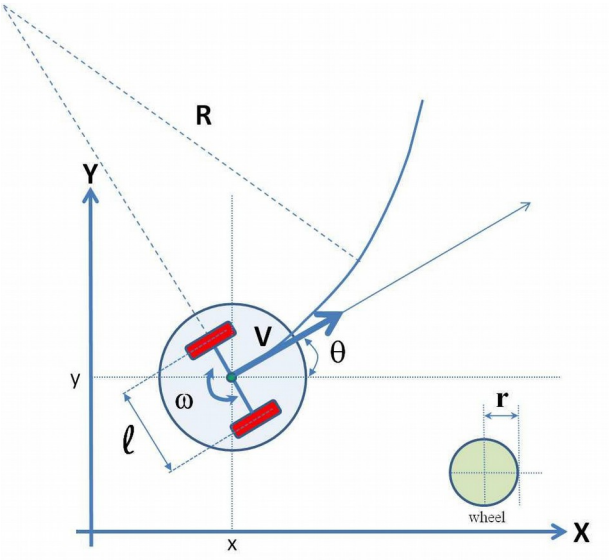
The aim of this lab is to control a simple [2,0] robot with three different type of control: static, dynamic and Lyapunov.

The control should take the robot from an initial position to along a circular trajectory with centre in the origin and of a given radius R; this trajectory start at [R,0] with pi/2 as orientation and with constant angular velocity w_d.

Note that all the three controller are settable and runnable directly through Matlab (script runStatic.m runDynamic.m and runLyapunov.m), which will also plot all the graph.

0. Theoretical model

The robot is a simple one with 2 fixed wheels. The model is represented by three states:



$$\dot{X} = V \times \cos(\theta)$$
$$\dot{Y} = V \times \sin(\theta)$$
$$\dot{\theta} = W$$

The input here are the linear velocity V and angular velocity W in the plane, but we use another kind of inputs: the wheel spins. This is because doing so the model is more realistic (in real world we can directly control the spin of the wheels and not the velocities in the plane) but also to check how much this spin is: with too much velocity of the wheel, things (path, errors…) could appear good but we would hide that the wheels are too fast for the real robot. When this happens, first it is a problem because we ask to the motor spins that it could not give. But the most important thing is that, even if the motor could give certain high spins, the robot will loose controllability.

So the input for the model are Φ_1 and Φ_2:

$$V = \frac{r}{2} \times \Phi_1 + \frac{r}{2} \times \Phi_2$$
$$W = \frac{r}{2L} \times \Phi_1 - \frac{r}{2L} \times \Phi_2$$

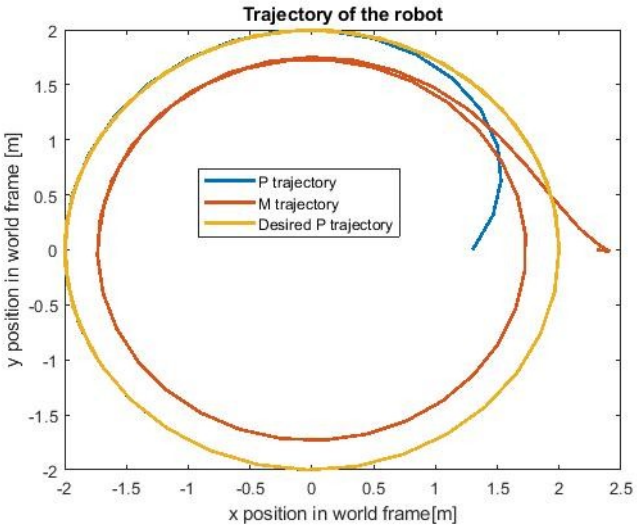
with r = radius of the wheel and L = distance of each wheel from the mobile frame ($l = 2 \cdot L$), both in meters and fixed as L=0.12m and r=0.1m .

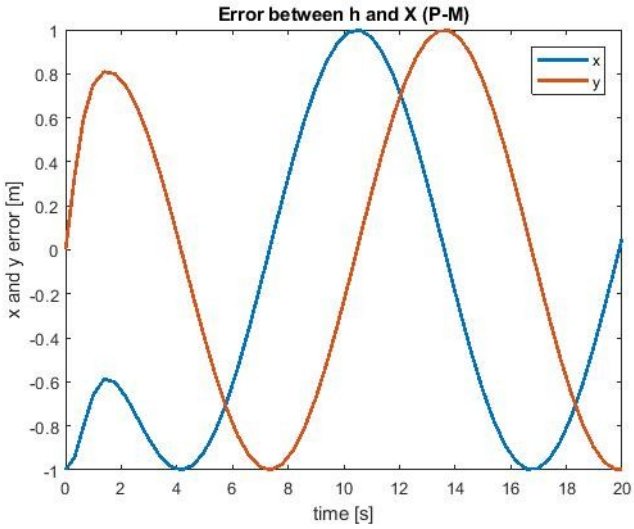
From these equations we construct our robot block, the one used in all three types of controls.
The other two blocks are the trajectory generator and the proper controller block, which are different for different method of controlling.

1. Static decoupling controller

For the static controller, we can't control directly the centre M of the robot (the middle point between the wheels), we can only control a point P. This because our control will be in form of $u = K^{-1} \times W$; the matrix K is a 2x2 with determinant $d \times \cos(\gamma)$. So it has rank 1 when P coincides with M (d=0) and/or when cos(γ) = 0 (P positioned on axis y_m of the robot), so it would not be invertible any more.
If we want control the centre M of the robot, we need another type of control (for example dynamic or Lyapunov, as in the next sections).
For this controller, we set a point P which must follow the circle. It is a point positioned in x axis of the robot, at distance d from the robot frame origin. This d is always 0.15 but otherwise specified (in the next 4 figures, for example).

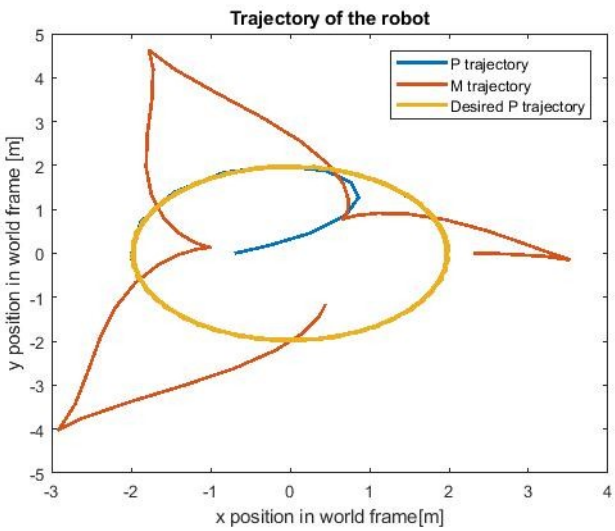
Give that the robot is a [2,0] type, the largest linearisable subsystem has dimension 2 and so we must accept to control only the x and y position. So we must remember to be care of the orientation which is not controlled.

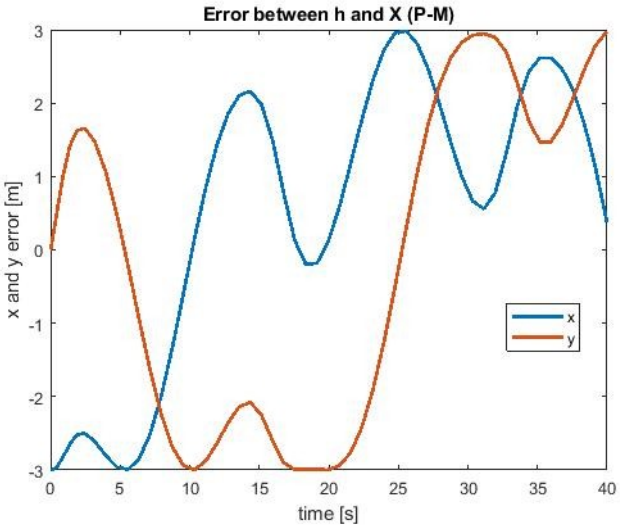




R=2 d=1 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [1 0;0 1]
Simulation time: 20 second

In the figures above we see that the trajectory of the robot (M) is different from the desired one because it is the point P which follows it. We choose d=1 so we can notice well the differences from the trajectory of M and P.
The figure on the right shows the distance on x and distance on y between point P and robot M. Note that, at steady state, when one is zero, the other is at his maximum: this is the point where the robot and point P are horizontal (error on y zero) of vertical (error on x zero).





R=2 d=3 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [1 0;0 1]
simulation time: 40 second

In these figure aboe on the left image the question is the same as before; in this case we have that the distance of point P from the robot is bigger that the radius wanted: so the unique way to make P to follow the wanted trajectory is to make the robot performs this strange “flower”.
With this resolution for the image is not noticeable but for the difficult of path followed (lot of changes of velocities and changes of direction) we note that the blue path is not so good as the one in the previous figure. Also the distance P-M is affected by these strange paths.

1.1 Simulink model

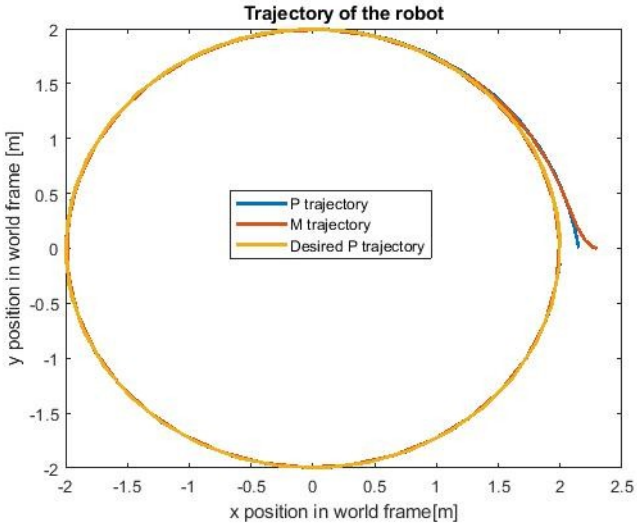
Trajectory Generator

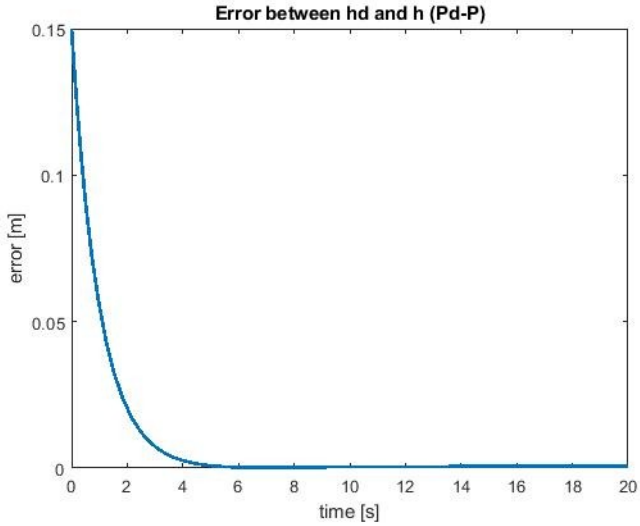
First thing is to build a "trajectory generator" block, which, with given angular desired velocity W_d and radius of the trajectory R, will compute the necessary desired state (h_d) and desired linear velocity (along x and y axis) ($\dot{h}_d$), at each instant (a clock is inside the block).
The derivative $\dot{h}_d$ is compute analytically (and not with a Simulink block from h_d) because numerical derivative will introduce unnecessary computational errors.
These "h" are then taken as reference by the controller.

Controller

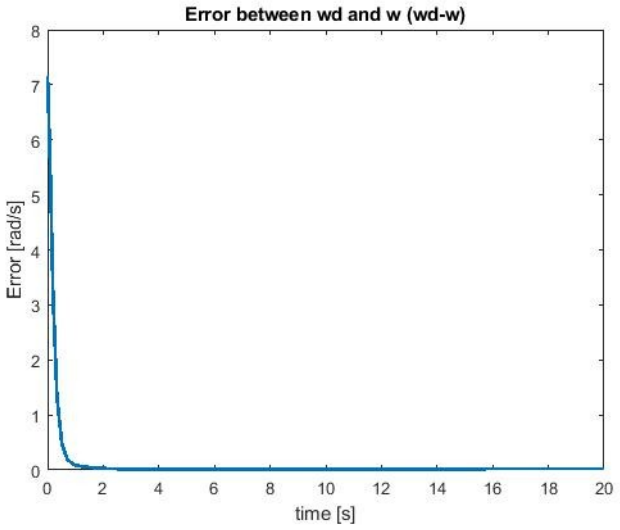
The controller is a simple static one which consider the error between desired trajectory and the real one (taking into account only the position on x and y and not the orientation theta)
The gain K_p is a 2x2 matrix because we have x and y positions.
It has no sense to set the gains differently since they are both positions with same importance, so K_p will always have the form K*eye(2).
As last thing, the controller "converts" the control from $\begin{matrix} v \\ w \end{matrix}$ to the real input spins $\begin{matrix} \phi_1 \\ \phi_2 \end{matrix}$ having the r and L of the robot (this is done also in the next controllers).

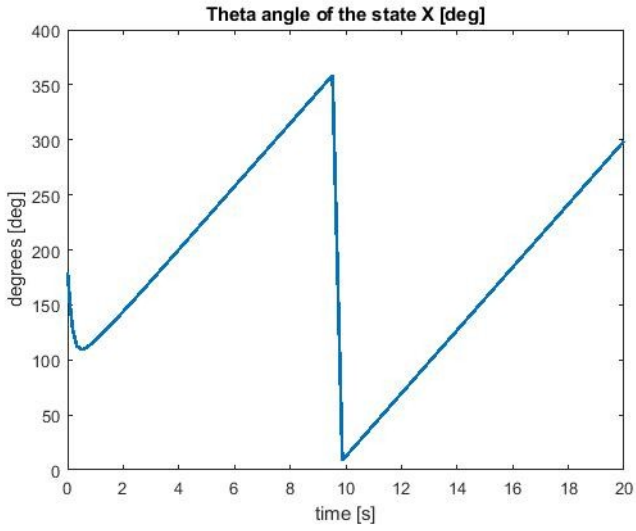
1.2 Result with ideal conditions





R=2 d=0.15 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [1 0;0 1]



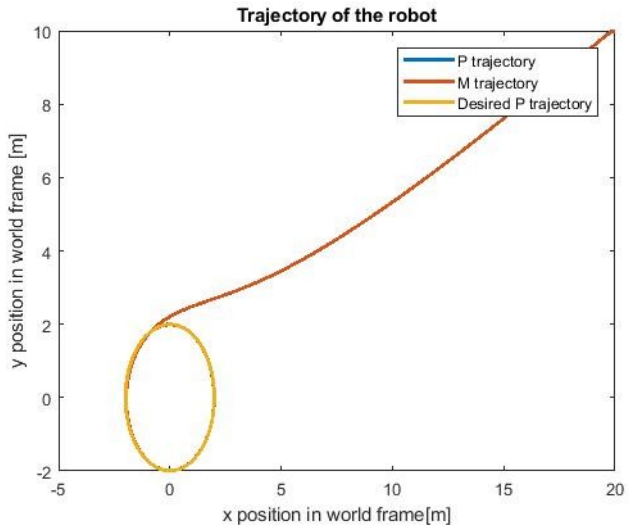


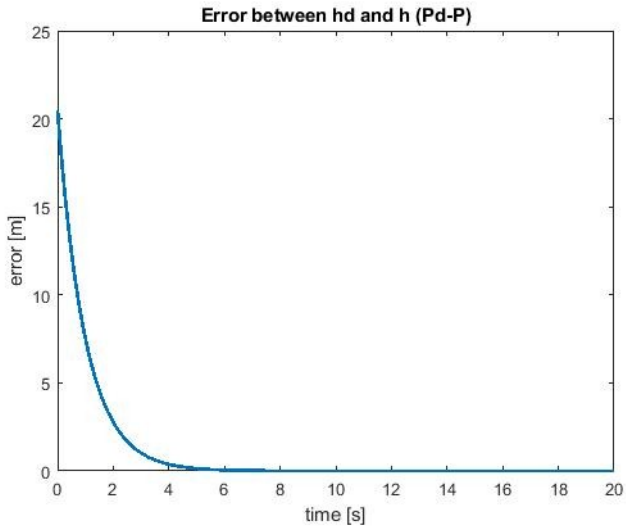
We need to keep under eye the orientation theta because it is not controlled and so it could diverge. Here his behaviour is good.

It has sense to compare the errors between desired trajectory and point P, and between desired angular velocity and real one, to see how much the controller is good.
We see that the control satisfies keeping the error $h_d - h$ bounded and bring it to zero at certain time. This is valid also for the angular velocity error.

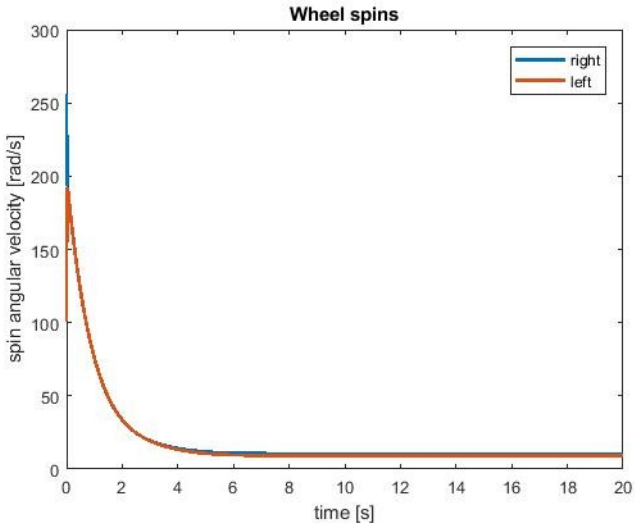
The experiment above was very simple: little initial condition errors (little distance from the ideal initial position for the circle trajectory [R, 0, pi]), good gains, no limitation on wheel spins and ideal model (no slippage, unique point of contact between wheel and ground, perfect knowledge of L and r…).
Now we try to add some different errors to make things more interesting.

1.3 With high initial errors



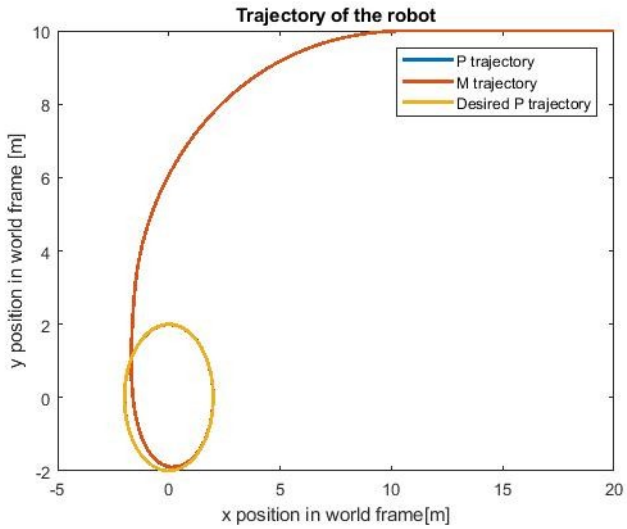


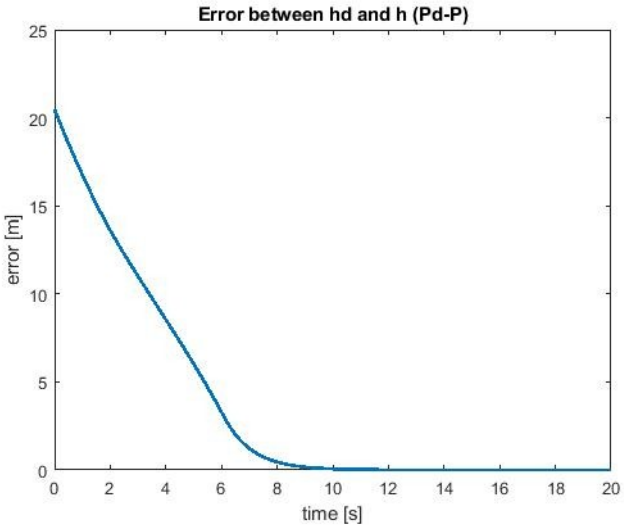
R=2 d=0.15 wd = 0.5 initialCond = [20 10 Π] gainKp = [1 0;0 1]
without saturation



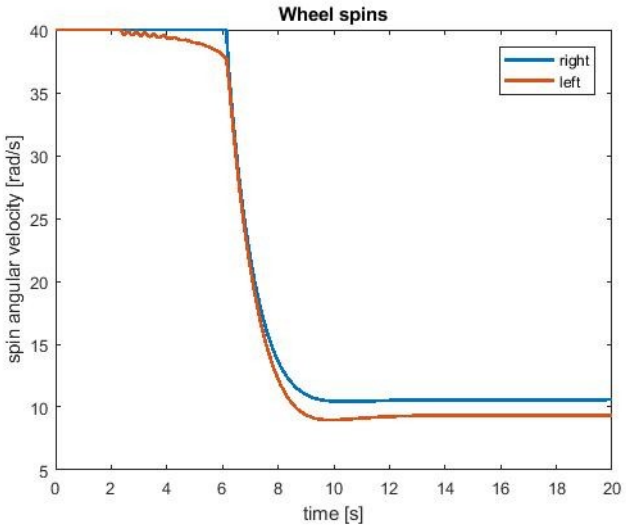
If the initial error is high, the robot tends to do a big acceleration to follow as soon as possible the desired trajectory. This means that the spin of the wheels at the beginning is very high, too high to be real.

We add in simulink a saturation block which limits the spin wheels from -40 to 40 rad/s (that is, twice the spin when the robot moves at 2 m/s in a straight line).





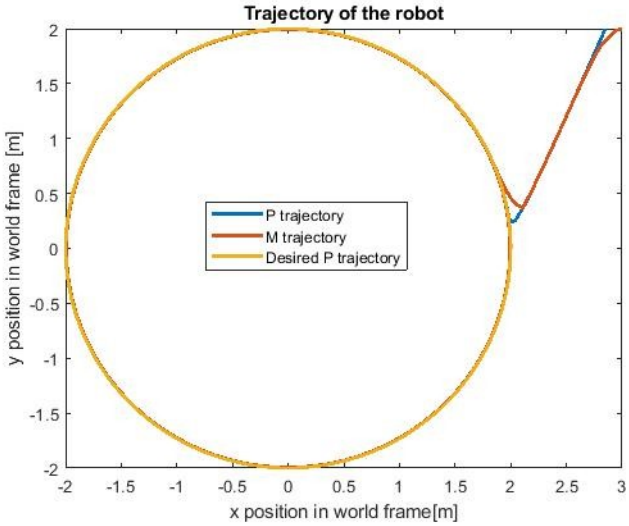
R=2 d=0.15 wd = 0.5 initialCond = [20 10 Π] gainKp = [1 0;0 1]
with saturation [-40;40]rad/s

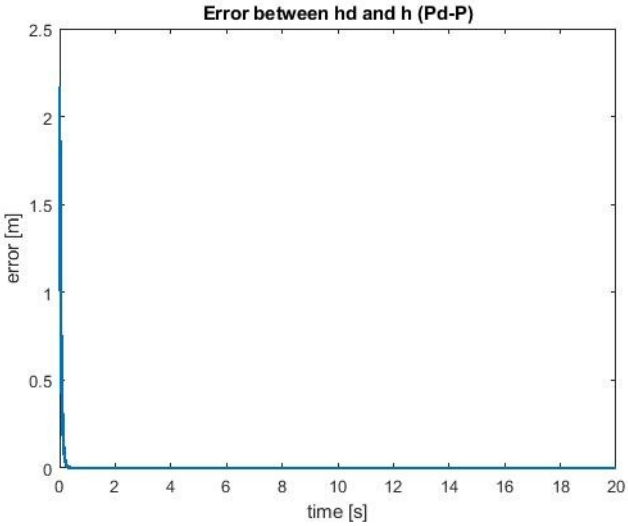


Now we can see the robot takes more time to set itself in the right trajectory, that is because wheel can't rotate anymore at unreal infinite velocity.
However, the control still converges at some point.
As said, the wheel spins are saturated: for 6-7 second they go at maximum speed possible to set as soon as possible in the right trajectory. Without saturation we simply have a spike and not a high constant speed for a while because we say that the robot can move wheel at infinite velocity.
Some chattering is noticeable for the left wheel: this is not good and could introduce bad behaviours (loose of controllability for example).

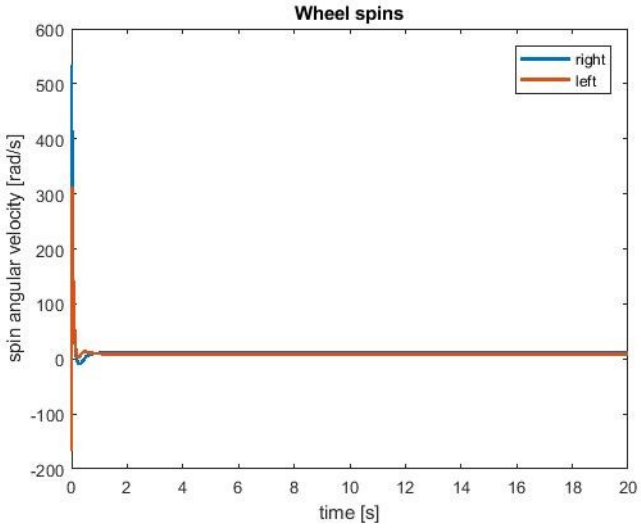
1.4 With high gain for the controller

Note: to appreciate the behaviour with high gain we set the initial position in [2;3;pi]

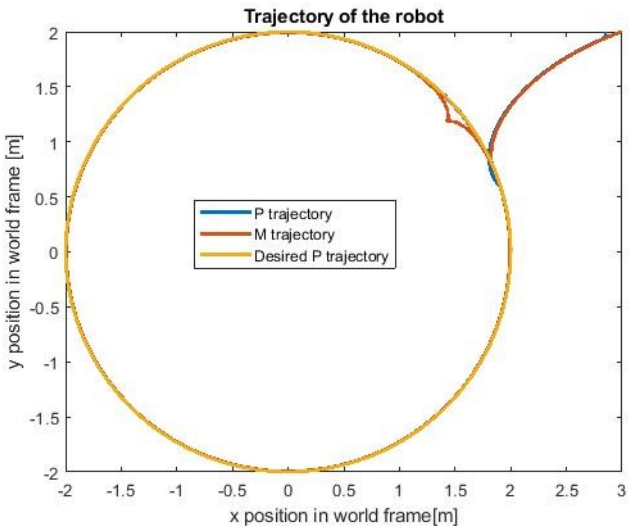


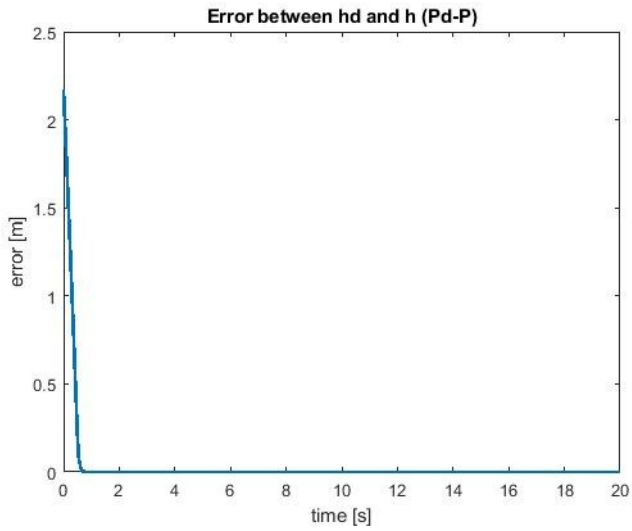


R=2 d=0.15 wd = 0.5 initialCond = [3 2 Π] gainKp = [20 0;0 20]
without saturation

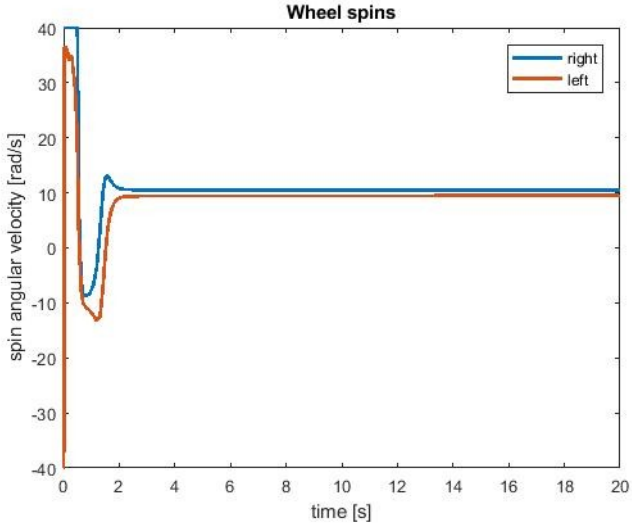


Despite having moderate initial error, thanks to this high gain (20), the robot goes around the trajectory very fast.
The problem here is the same with the high initial error: the spins are too fast to be realistic and we need a saturation block to model the fact that the spins can't rotate at infinite velocity.





R=2 d=0.15 wd = 0.5 initialCond = [3 2 Π] gainKp = [20 0;0 20]
with saturation [-40; 40] rad/s

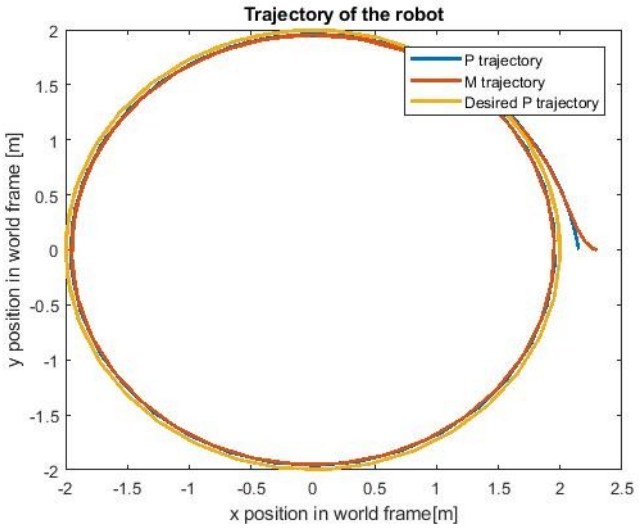


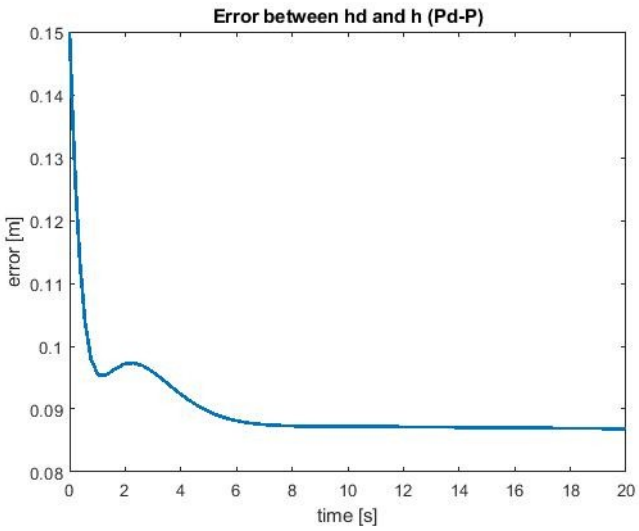
In fact, with saturation, we don't have convergence of error so fast, however it still converge in a acceptable time.

As last thing we must remember that also be too near the saturation point (as in this case) is potentially bad because we can loose controllability of the robot (it moves too fast).

1.5 Not perfect model (error on radius r and distance L)

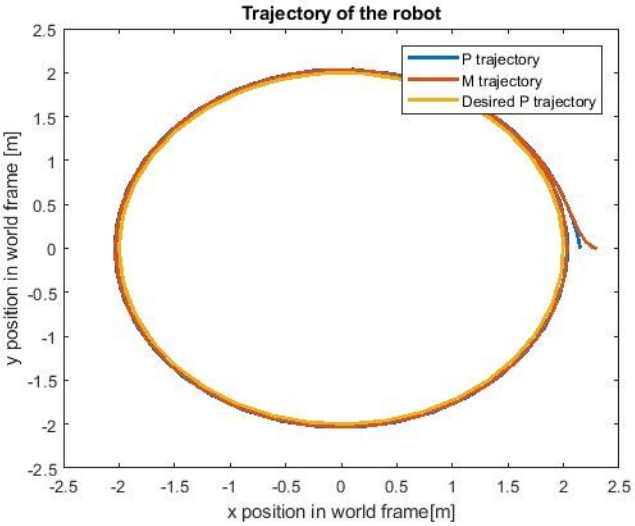
Another kind of evaluation is to add to the model that the robot is not perfect, to test the robustness of the controller. The ideal thing to do this is to implement that it has some slippage/skidding of the wheels or that they are worsen, not identical and without unique contact point. But model this would be really difficult: so we simply add an error on the knowledge of r (radius of wheel) and L (distance from wheels to centre of the robot) which will introduce similar bad behaviours.
So now the value of r and L which we pass to the controller are not identical to the real r and L of the robot block.

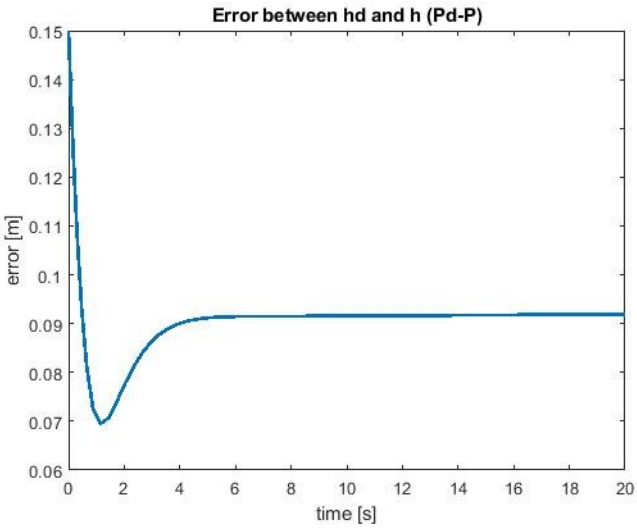




R=2 d=0.15 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [1 0;0 1]
error on r = +10% real r

If the controller thinks that the radius is bigger than the real one, wheels will need less spin to perform the w_d wanted; and less spin with the same angular velocity (the one desired is fixed) will imply a trajectory with littler radius.
The error at steady state can't go to zero: the controller does not know that radius is wrong and so the errors will tend to a constant different from zero.

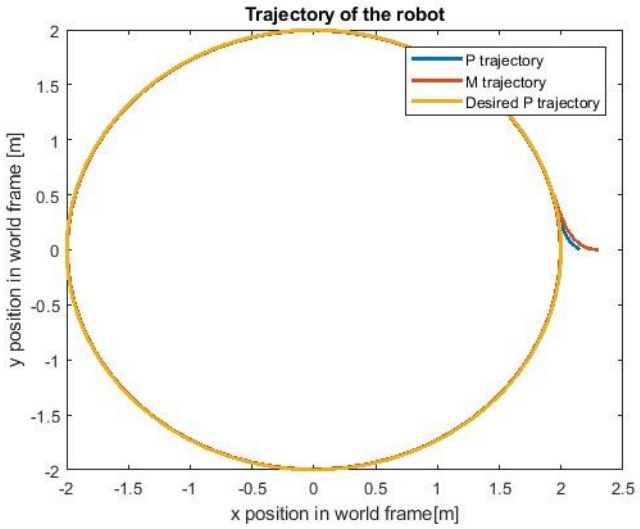


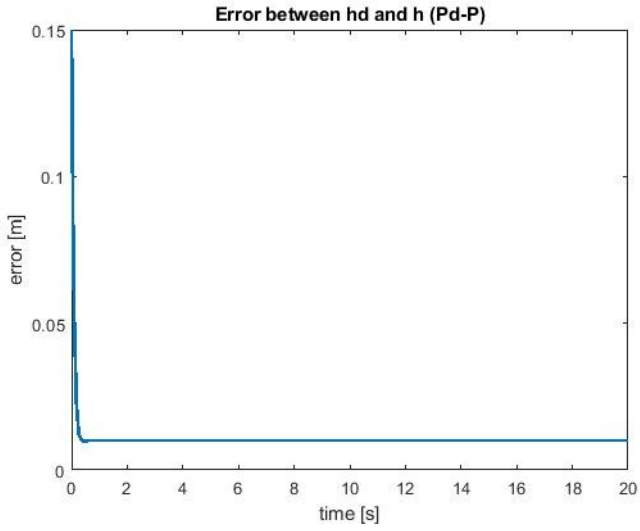


R=2 d=0.15 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [1 0;0 1]
error on r = -10% real r

If we underestimate the radius, the real trajectory will be bigger, because spin wheel will be bigger.

If we want to compensate these error, we need to increase the gain:

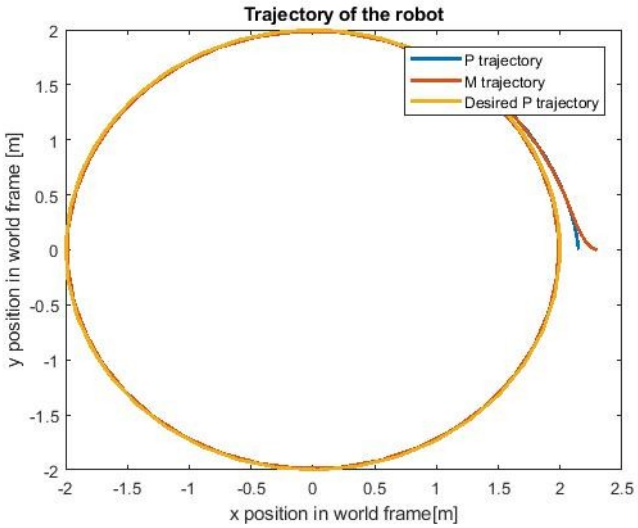


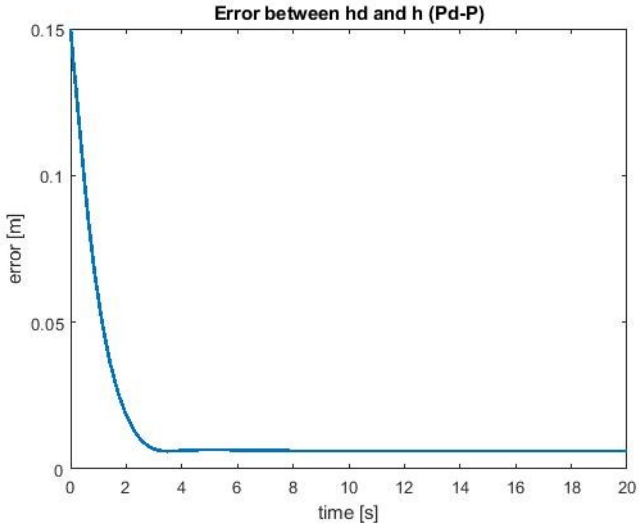


R=2 d=0.15 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [10 0;0 10]
error on r = -10% real r

Now the error at steady state is nearer to zero (but not exactly zero, which is impossible if there is some error on r).

With error on L the results are not so bad even with the usual gain = 1 : the distance of the wheels is not important to be so precise.
The only important thing is that the estimated L is not too much near zero otherwise the controller will have a lot of difficults to make the robot turn.

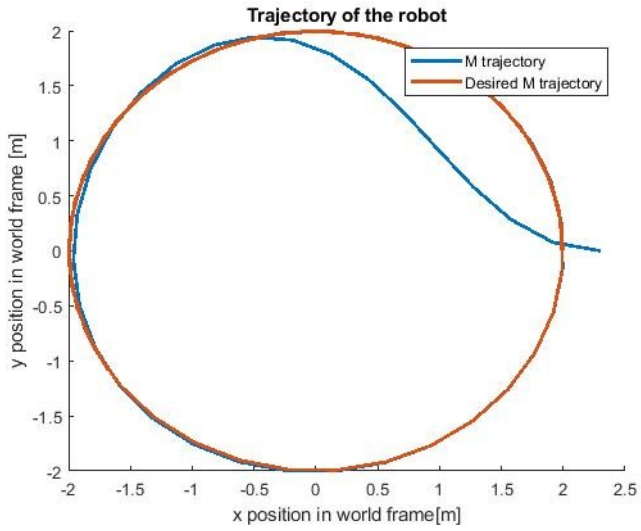


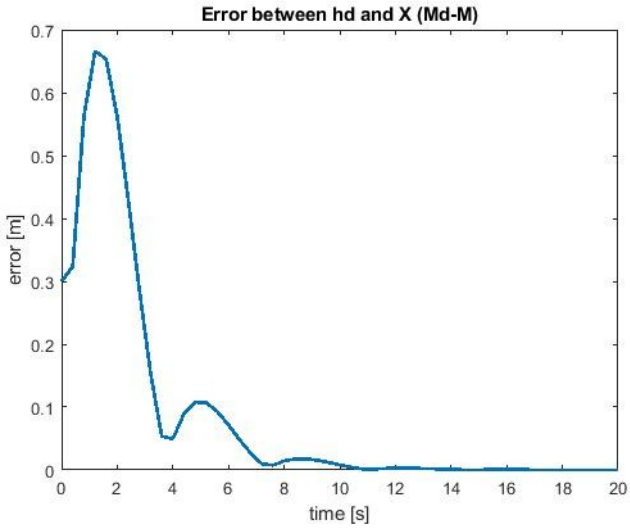


R=2 d=0.15 wd = 0.5 initialCond = [2.3 0 Π] gainKp = [1 0;0 1]
error on L = +10% real L

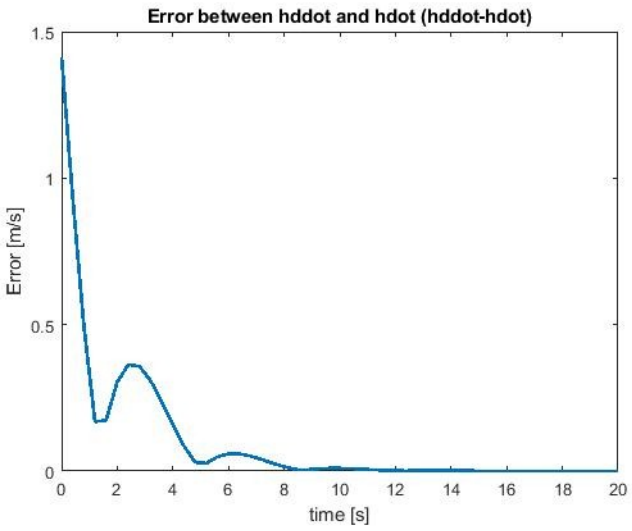
2. Dynamic decoupling controller

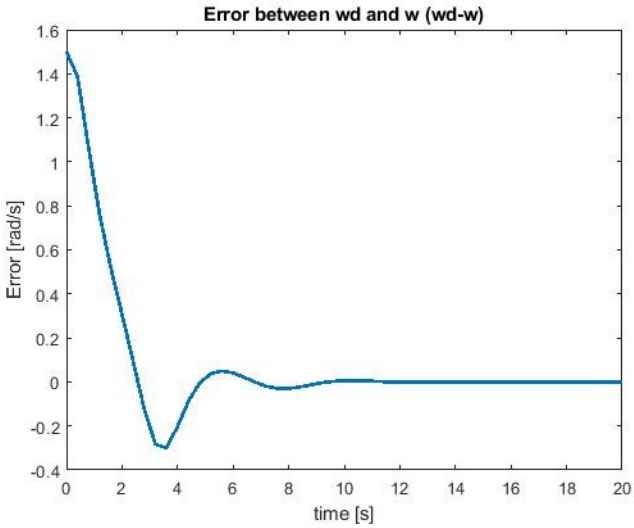
In a (2,0) robot type if the controlled point is coincident with the middle of the wheels (or in any point of y_m axis), we can't use a static feedback, because there will be a matrix with singularities, as said in the previous section. So here we use a dynamic controller.

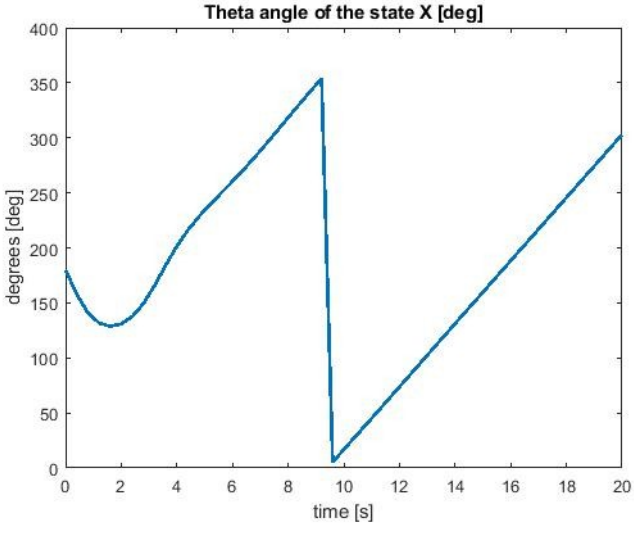




R=2 w_d = 0.5 initialCond = [2.3 0 Π] gainK_p = gainK_v = [1 0;0 1]







In the images above, we can see that after a transient phase, the trajectory is well followed, and the errors on trajectory converge. Also, as in the static controller, we must check theta because it can't be controlled.

We see that the control satisfies keeping the error $h_d - h$ bounded and bring it to zero at a certain time.

2.1 Simulink model

Trajectory Generator

The feedback u is now dynamic: it takes into account also the acceleration. So now the trajectory generator computes also the desired linear acceleration (along x and y) $\ddot{h}_d$.
As for the static one, the derivative is computed analytically to not introduce numerical computational errors.

Controller

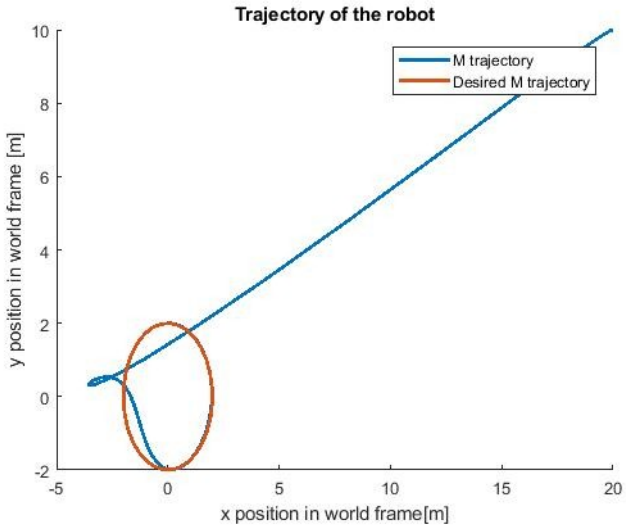
The controller takes into consideration the error on position (with gain K_p) (like the static one) and the error on velocities (with gain K_v : 2x2 matrix because velocity is divided into x component and y component). So now we have two gains to tune.
Also now the core of the controller gives linear acceleration and angular velocity (in the static one it gives linear and angular velocities).

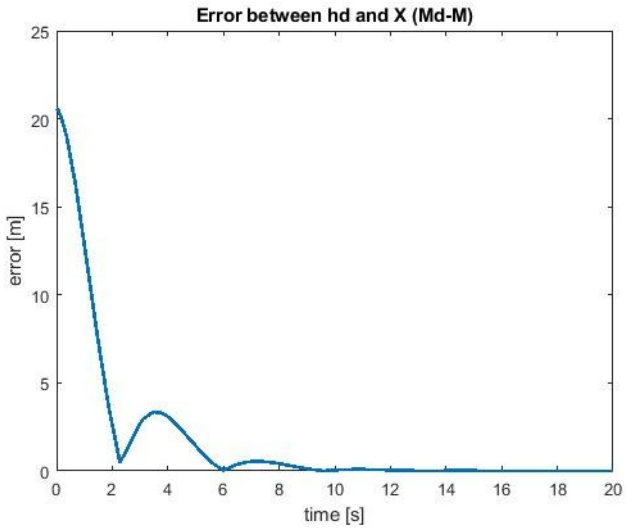
The control has this form: $u = F^{-1} \times W$ with F a 2x2 matrix which have rank 1 when the velocity is zero (determinant of F is v).
So we must pay attention to the initial velocity given to the controller: if it is zero we have at the beginning a singularity and we can't invert the F matrix.
The initial velocity is simply $w_d \times R$
This value is given as initial condition to the integrator which integrates the acceleration to have the velocity to give to the robot (after converts it into spins).

We also notice that the velocity is doing an algebraic loop: it must be known by the controller to compute the control to give to the robot, but the robot must know it to move at the beginning. So it is either an input for the robot and a feedback for the controller, without passing through any dynamic system. This cause a warning in Simulink and sometimes a error (with high gains and saturation model goes in singularity). To prevent this it is usually used a memory or input delay block, but these give us even more problems (divergences and strange chattering) and we discard the idea.

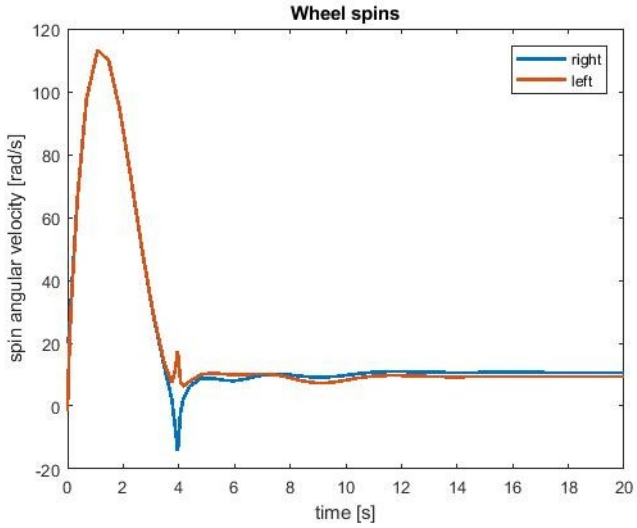
As before, we test the quality of the controller introducing various errors.

2.2 With high initial errors





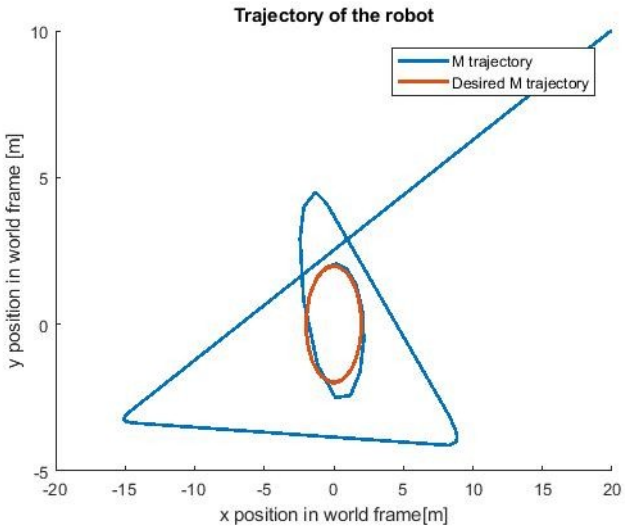
R=2 wd = 0.5 initialCond = [20 10 Π] ***gainK$_p$ = gainK$_v$ = [1 0;0 1]***
without saturation simulation time 20 second

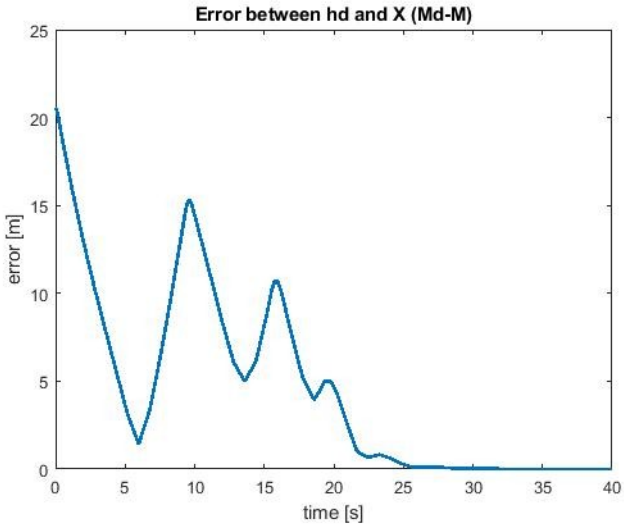


The behaviour here is good, but as for the static, this is not a real behaviour because in real world we can't ask to the wheels to spin so fast.

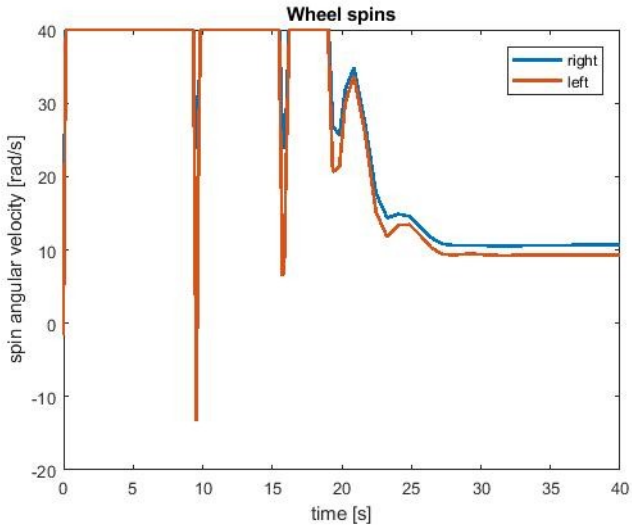
So, we will use again a saturation block:

Note that now simulation is up to 40 second to wait the robot to follow the trajectory (without saturation version was 20, it was faster).





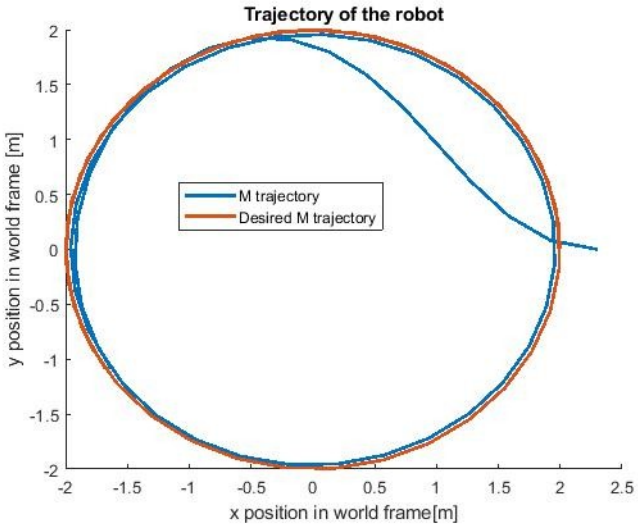
R=2 wd = 0.5 initialCond = [20 10 Π] $gainK_p = gainK_v = [1\ 0;0\ 1]$
with saturation simulation time 40 second

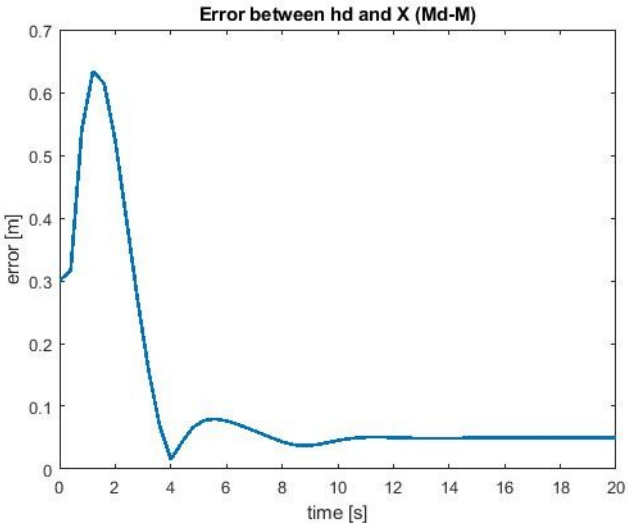


With this “more real” model, with saturation, we can see how the robot has more difficult to set in the right trajectory.
However, it still manages to converge at some point, obviously with more time respect of version without saturation. Even if we must keep in mind of that bad spikes of the spins that can damage mechanisms and that a real robot couldn’t follow these fast changes of spins.

2.3 Not perfect model (error on radius r and distance L)

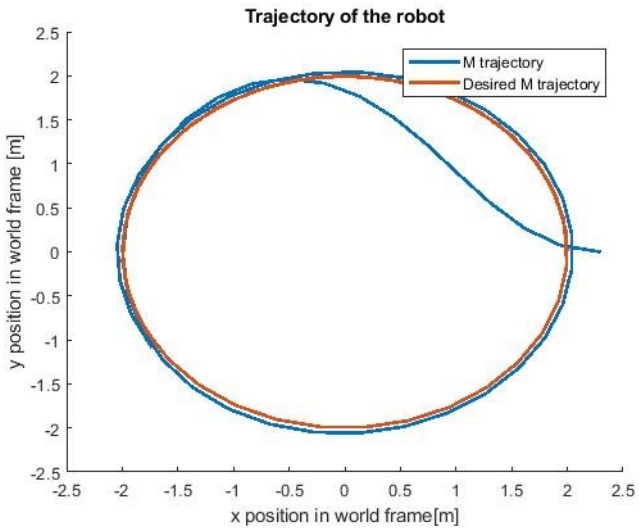
The last test is to give information not so precise about the robot to the controller: errors on measures of radius r of the wheel and distance between wheel and centre of the robot.

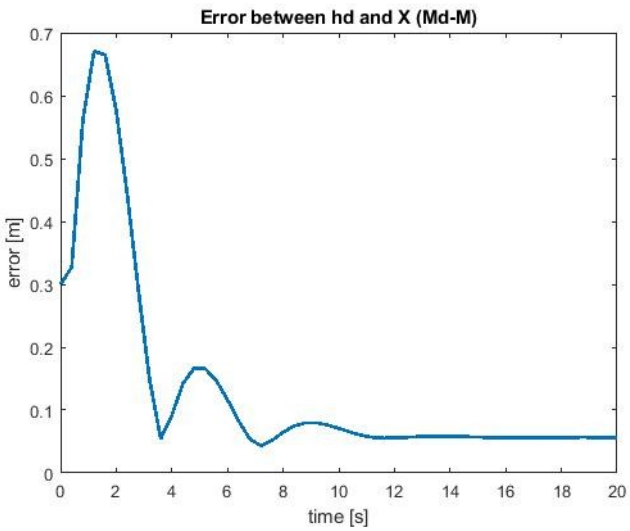




R=2 wd = 0.5 initialCond = [2.3 0 Π] $gainK_p = gainK_v = [1\ 0;0\ 1]$
error on L = +10% real L

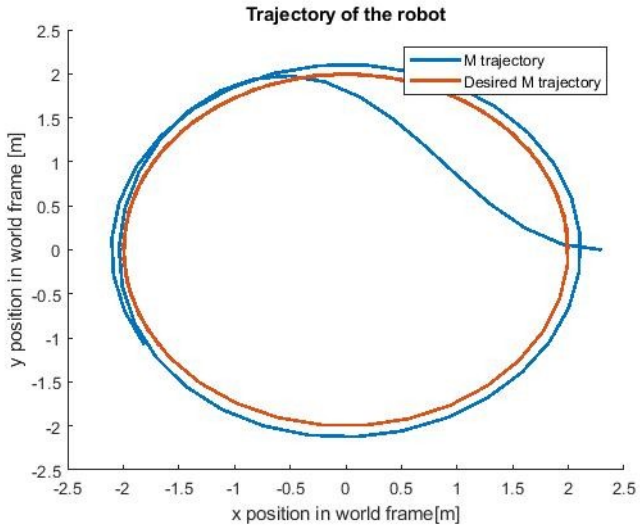
Differently from the static one, now the L error influence visibly the robot (it has similar weight as the radius wheel)

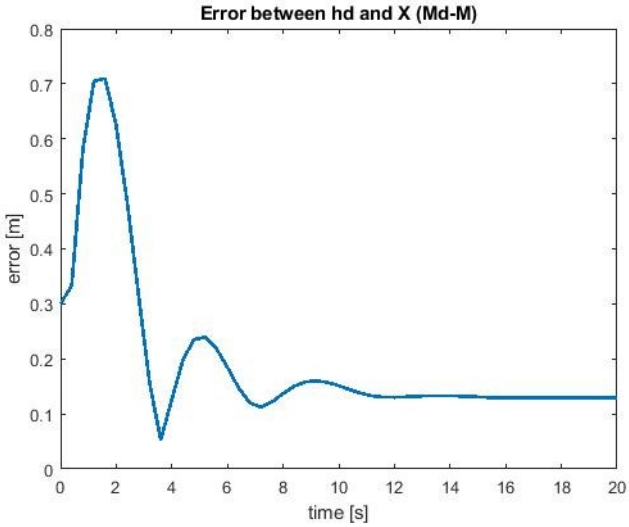




R=2 wd = 0.5 initialCond = [2.3 0 Π] $gainK_p = gainK_v = [1\ 0;0\ 1]$
error on r = +10% real r

They are, in some way, complementary (L+10% decreases the radius of the trajectory, r+10% increases it) so +10% error on both will give almost no error, and this could be confusing if we don't divide the two errors in two different experiment.

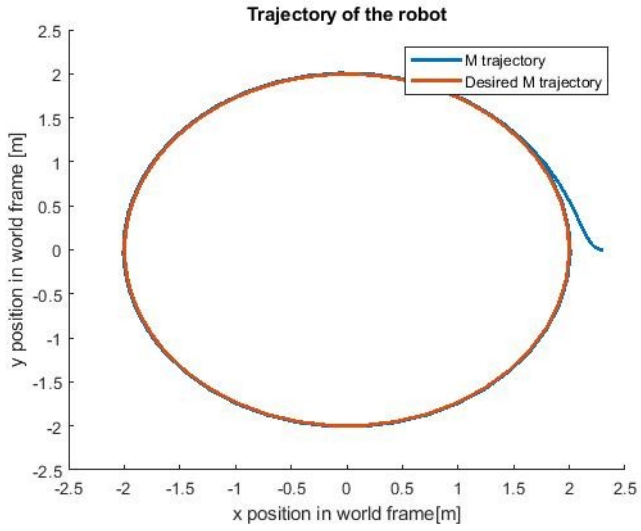


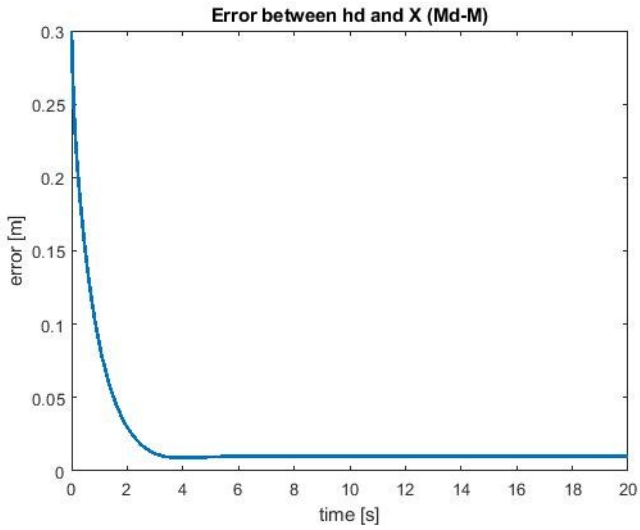


R=2 wd = 0.5 initialCond = [2.3 0 Π] $gainK_p = gainK_v = [1\ 0;0\ 1]$
error on r = +10% real r and on L = -10% real L

As for the static, the trajectory error can't converge at zero.

If we increase the gain, the trajectory error converge to a value near to zero (but, still, not exactly to zero):

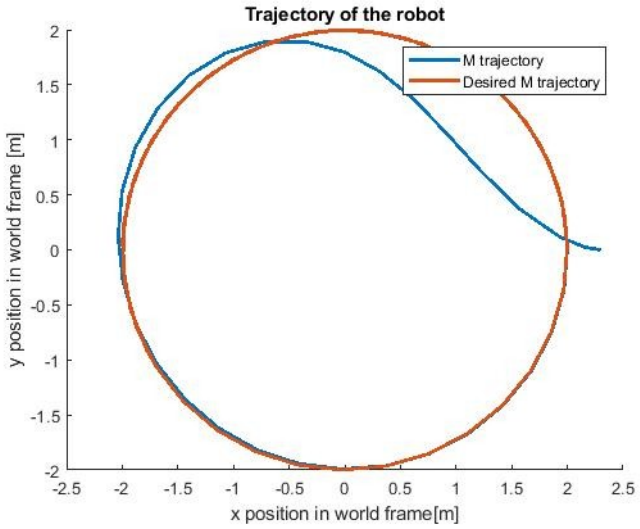


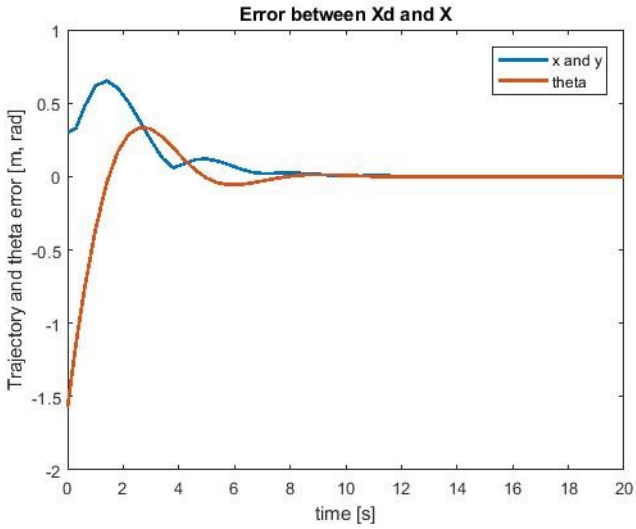


R=2 wd = 0.5 initialCond = [2.3 0 Π] $gainK_p = gainK_v = [10\ 0;0\ 10]$
error on r = +10% real r and on L = -10% real L

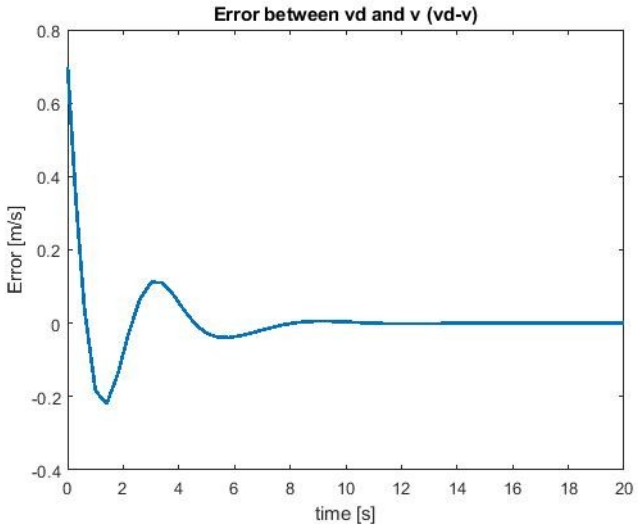
Mobile Robot 2017/18 Lab 2
Davide Torielli Fabio Fusaro

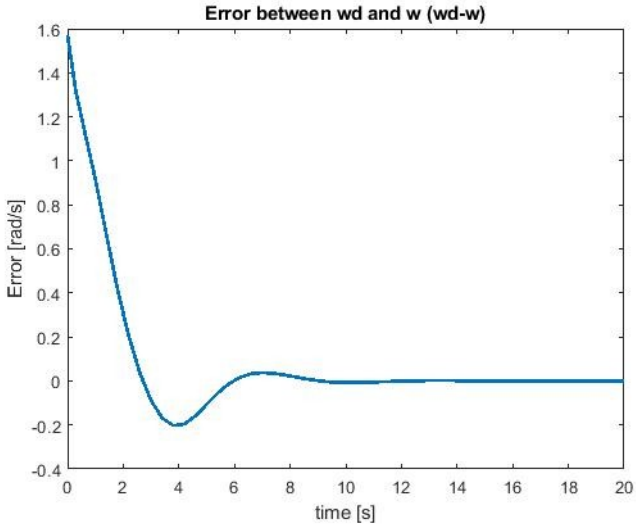
Lyapunov controller





R=2 w_d = 0.5 initialCond = [2.3 0 Π] $K_x = K_y = K_\Phi = 1$





The controller is good: path is well followed after a short time (but longer than the previous two controllers), all the errors are bounded and go to zero. Note that in the plot Xd - X now we have the error on position in the plane (as in the previous controllers) and the error on orientation (red line in second figure, not present in the other controllers).
In fact with this controller we also control the orientation theta.

3.1 Simulink model

Trajectory Generator

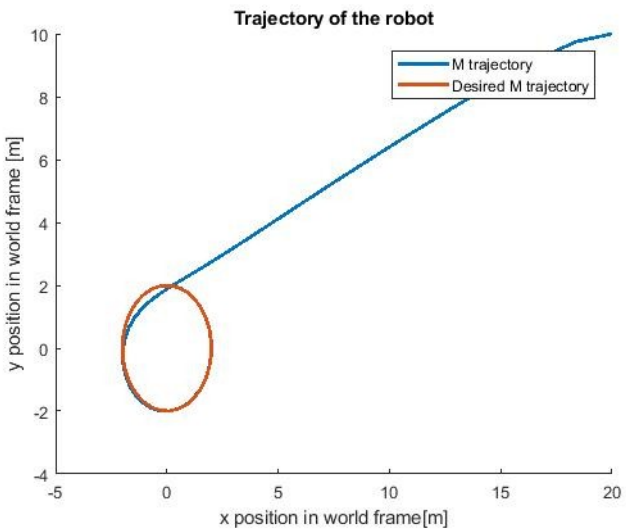
The Lyapunov controller uses the desired state $\begin{matrix} X_d \\ Y_d \\ \theta_d \end{matrix}$, the desired linear velocity v_d and desired angular velocity w_d, so the trajectory generator is modified according to this.
The w_d given to the controller is simply the one provided as reference by the user.
v_d is calculated from w_d as $R \times w_d$ (R radius wanted).
X_d and Y_d are the same of the other two cases, and θ_d is obviously the integration (in time) of w_d.
We must pay attention to the initial theta given to this integrator: it must be exactly the one wanted at initial time (pi/2 for the chosen trajectory); otherwise the controller will fail.

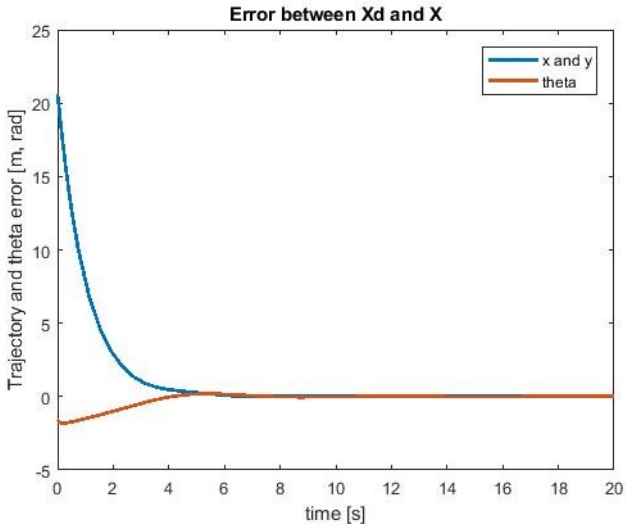
Controller

The controller has now three gains K_x K_y K_Φ, one per state. The control u will be constructed from all these information. Note that now we can control also the orientation θ, so we will plot the error also for this state.

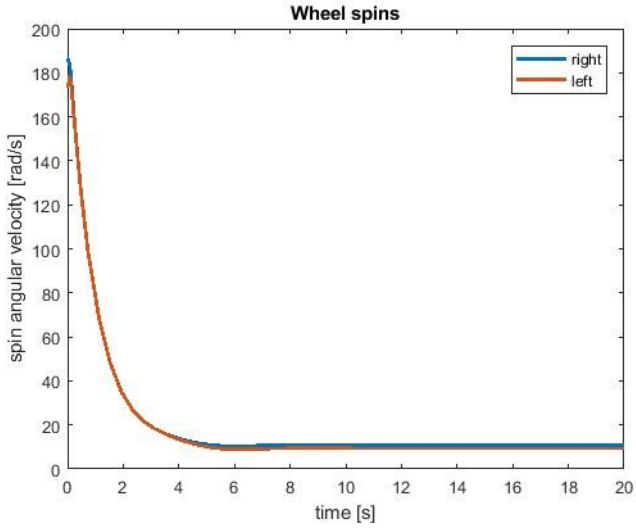
We now test the controller introducing different errors as in the previous section.

3.2 With high initial errors

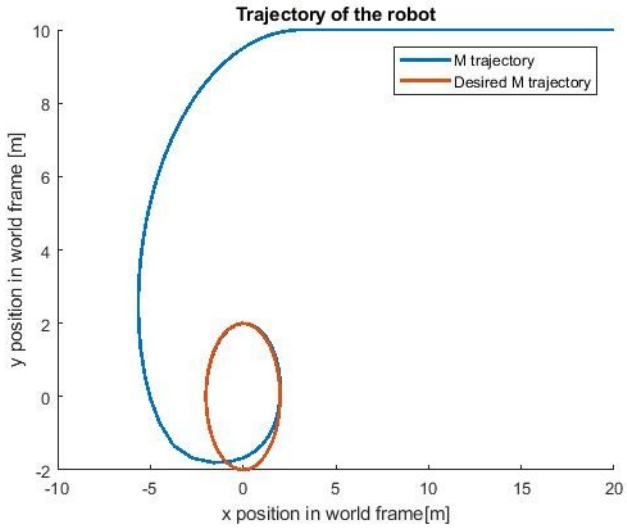


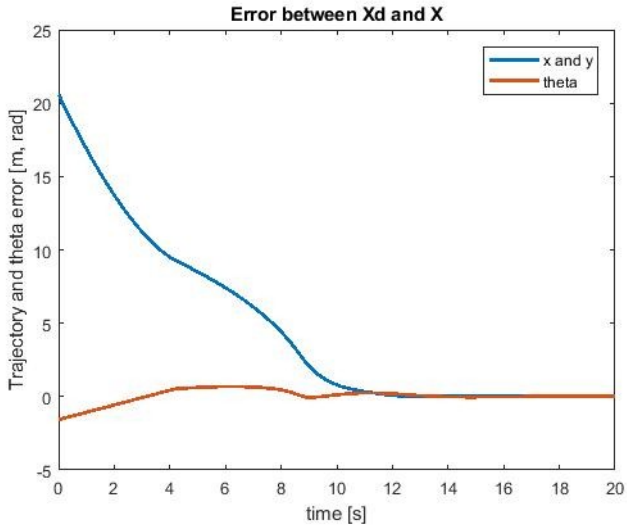


R=2 w_d = 0.5 initialCond = [20 10 Π] $K_x = K_y = K_\Phi = 1$
without saturation

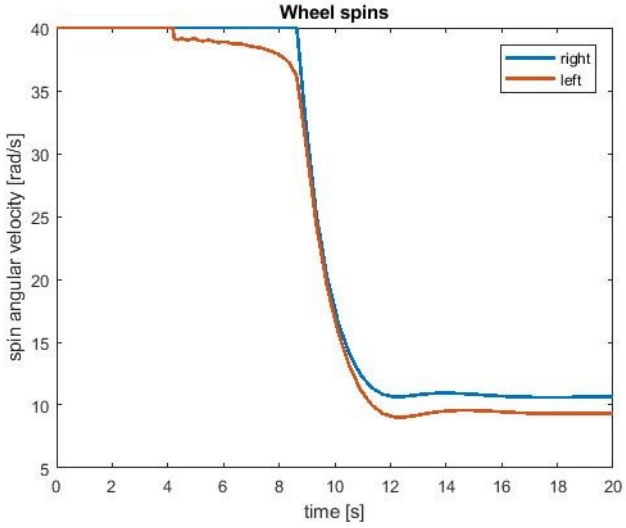


Without saturation, the controller makes the robot converge nicely in a short time.





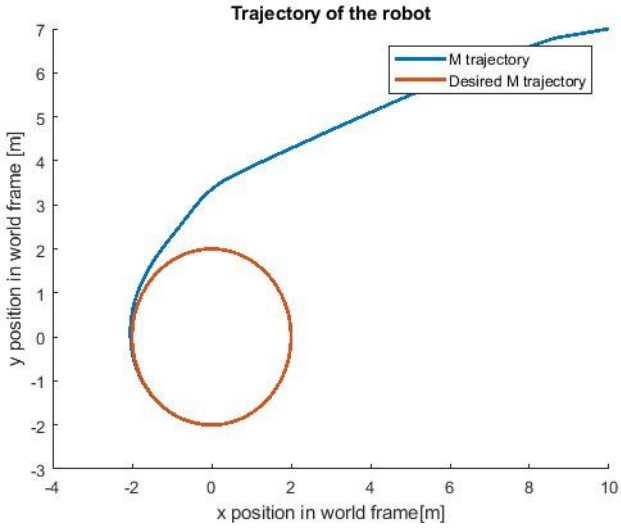
R=2 $w_d = 0.5$ initialCond = [20 10 Π] $K_x = K_y = K_\Phi = 1$
with saturation

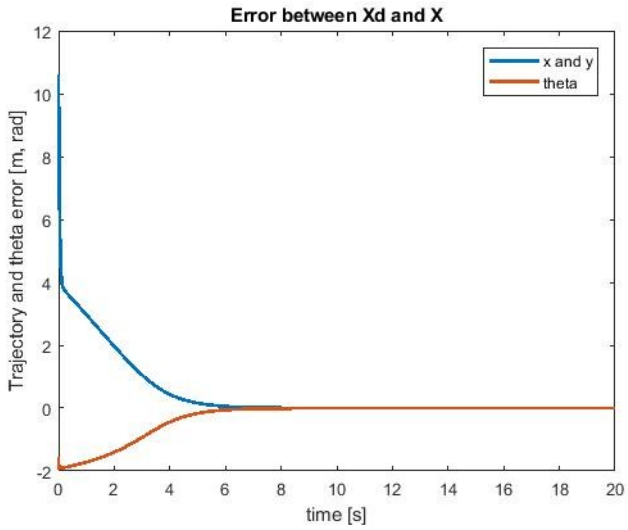


As always, introducing “real” saturation to the robot, we avoid the wheels to spin too much, so the converge of error lasts more time.

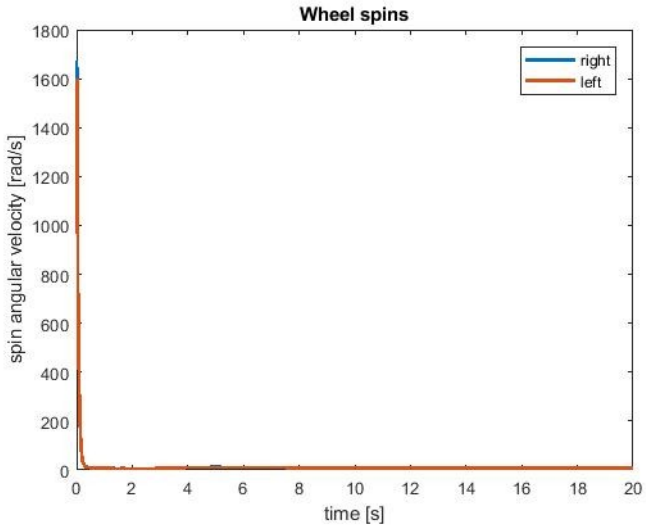
3.3 With high gain for the controller

Note: to appreciate the behaviour with high gain we set the initial position in [10;7;pi]

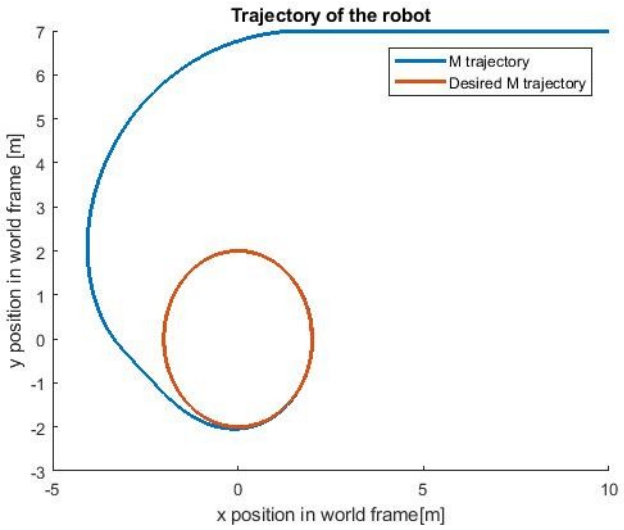


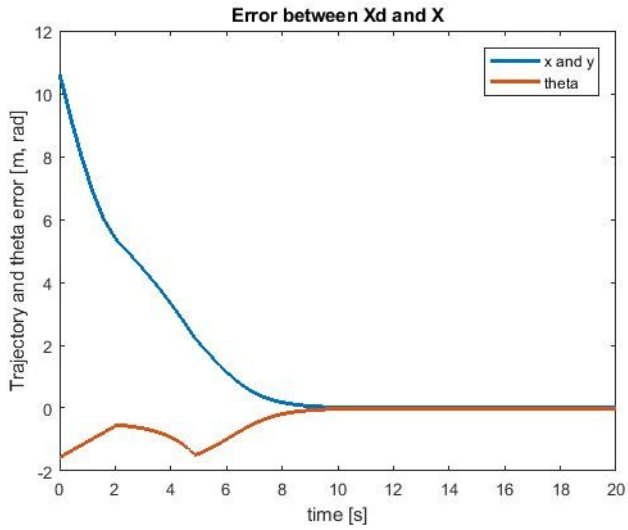


R=2 w_d = 0.5 initialCond = [10 7 Π] $K_x = K_y = K_\Phi = 20$
without saturation

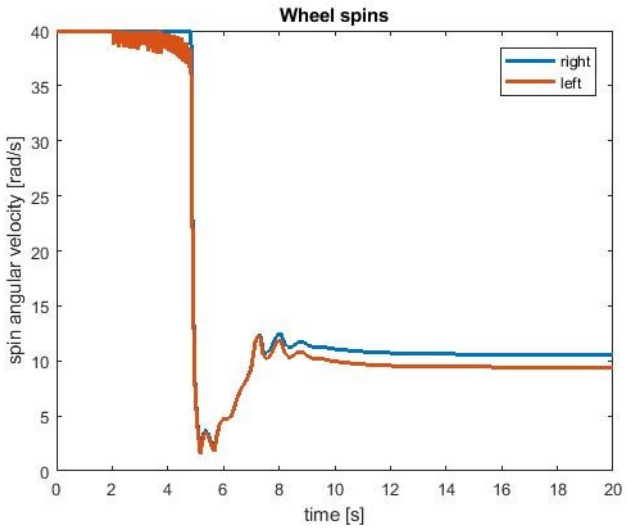


Obviously, increasing the gains the convergence is faster, but we still have to introduce the usual saturation:



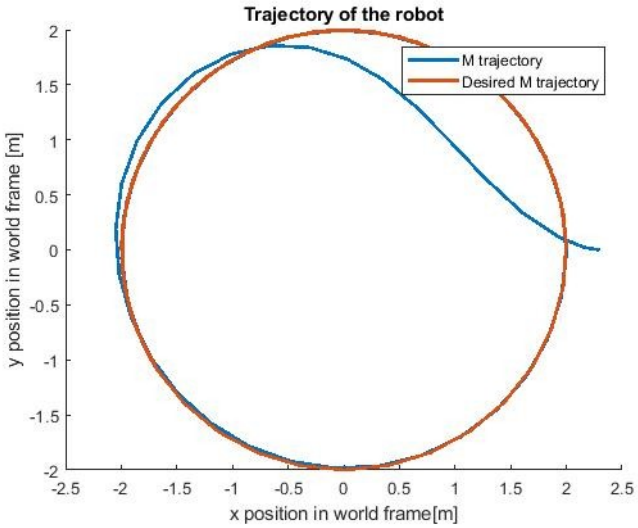


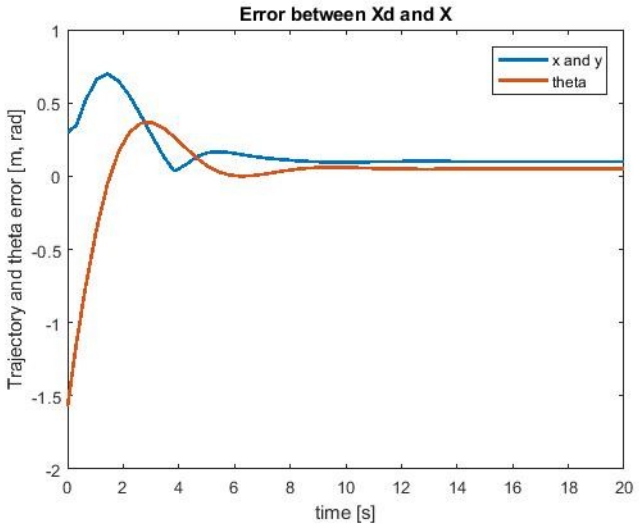
R=2 w_d = 0.5 initialCond = [10 7 Π] $K_x = K_y = K_\Phi = 20$
with saturation



With saturation, the spins stay near maximum for a bit; we see some chattering for the left wheel: this can be not good for the mechanisms, so the gain may be too much.
We can see the reason of this chattering in the trajectory path: while the right wheel spin at his maximum, the left must change his velocity very fast to correct the trajectory while the robot is approaching to the red desired trajectory.
A real robot probably can't follow this frequent changes of spin and the behaviour would be different.

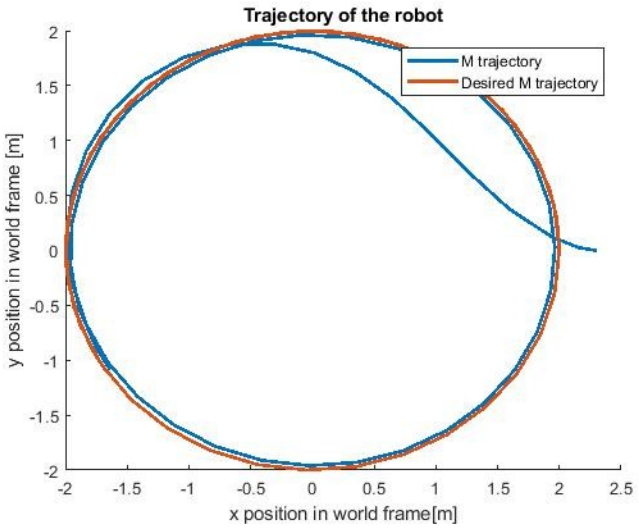
3.4 Not perfect model (error on radius r and distance L)

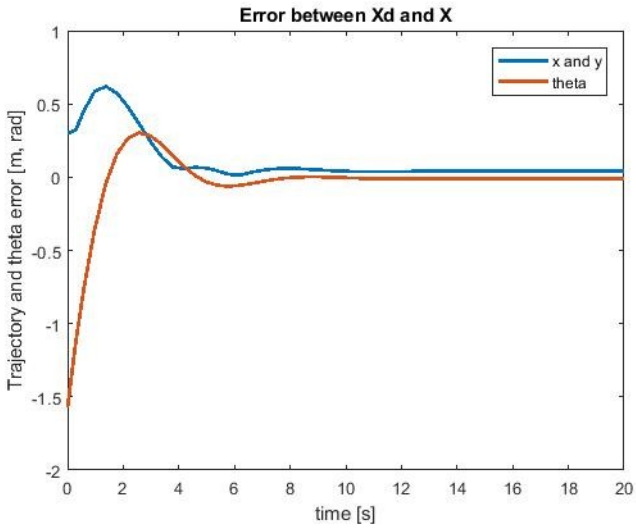




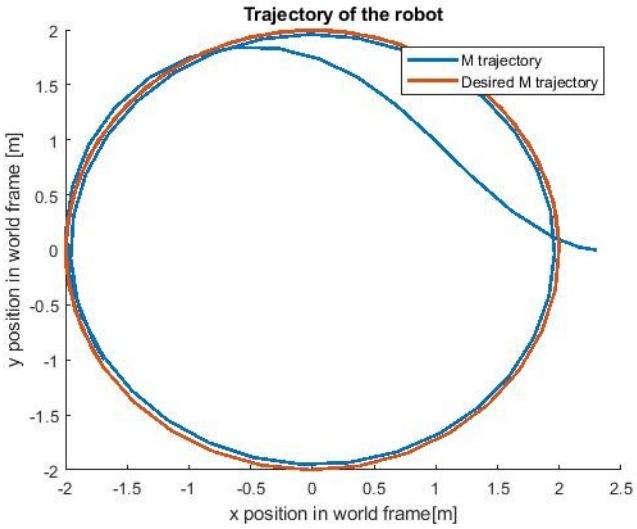
R=2 w_d = 0.5 initialCond = [2.3 0 Π] $K_x = K_y = K_\Phi = 1$
error on r = +10% real r

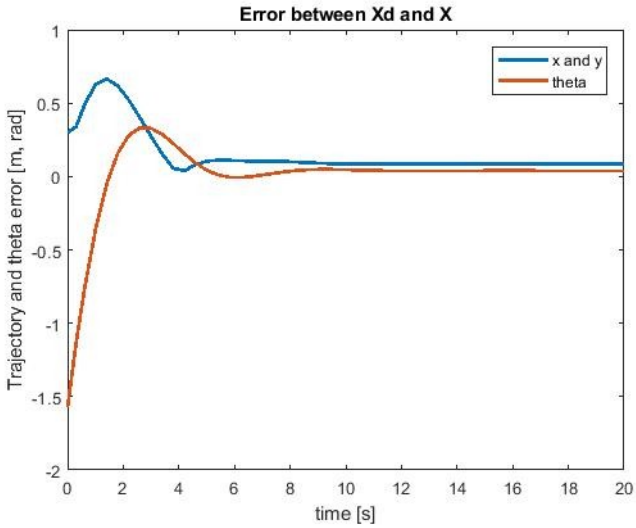
Differently from previous controllers, errors on the model are not so influential: we can see that error on trajectory converges to almost zero.
This is true for error on r (above) and on L (below).





R=2 w_d = 0.5 initialCond = [2.3 0 Π] $K_x = K_y = K_\Phi = 1$
error on L = +10% real L

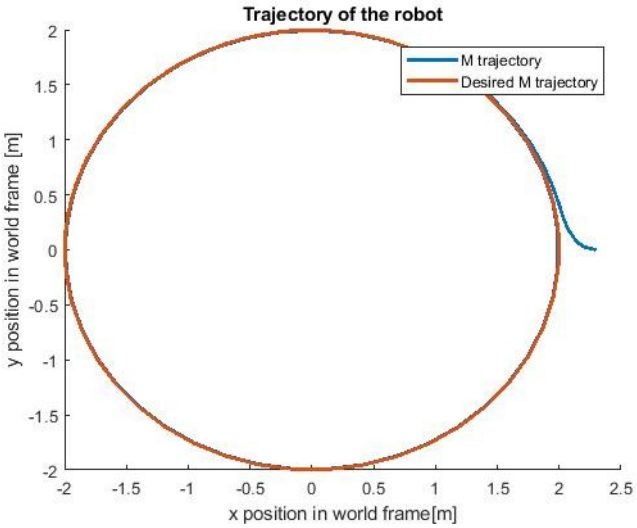


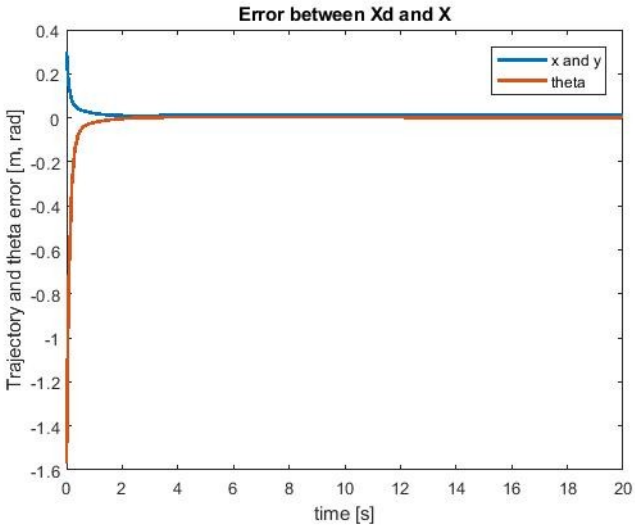


R=2 w_d = 0.5 initialCond = [2.3 0 Π] $K_x = K_y = K_\Phi = 1$
error on r = +10% real r and L = +10% real L

Above we have put both errors on r and L and we still have good final trajectory error.

However, if we can't bear even this little errors, we can always increase the gain (below):





R=2 w_d = 0.5 initialCond = [2.3 0 Π] $K_x = K_y = K_\Phi = 10$
error on r = +10% real r and L = +10% real L