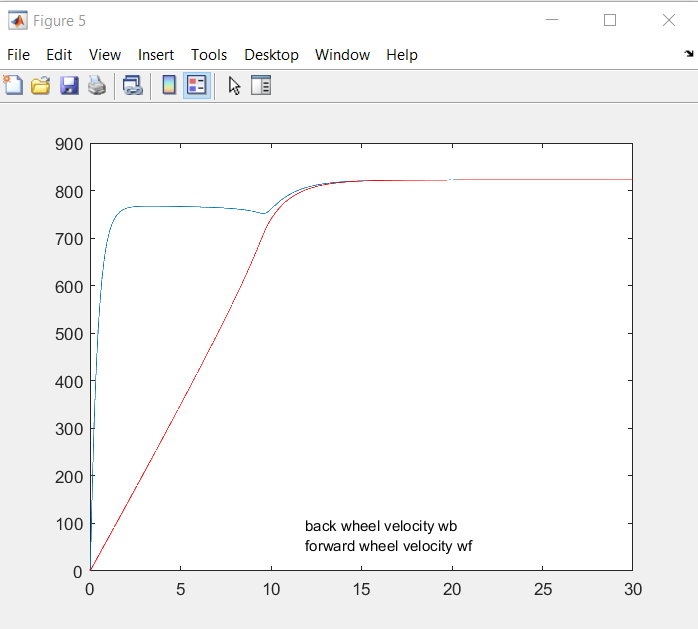
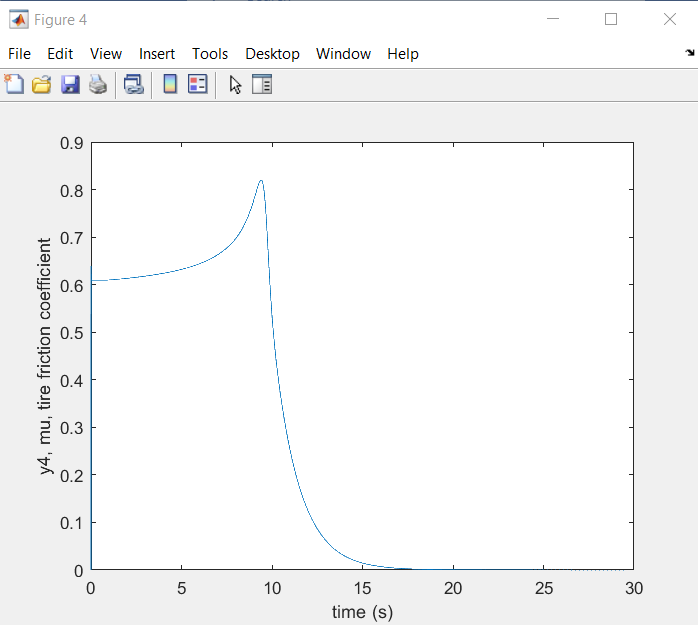
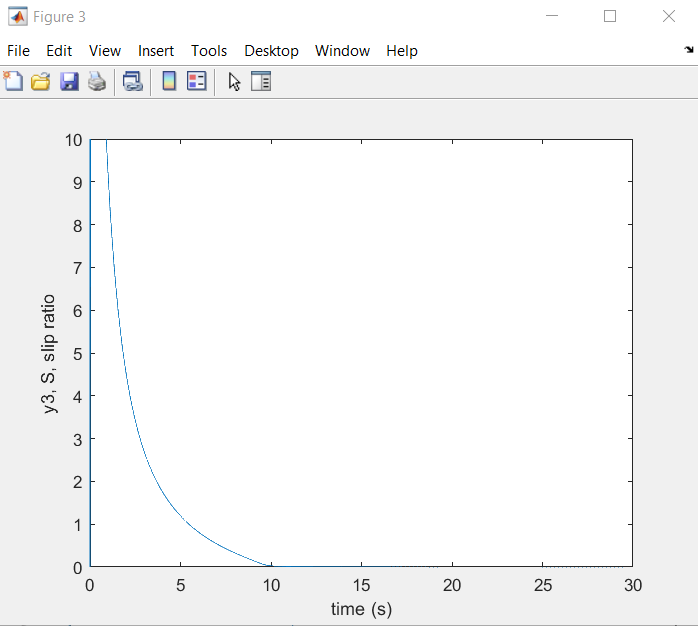
As traction control is the basic requirement of the car system and should be coupled with launch control, speed control and brake control. We first study how to maintain traction control.

# Traction control

Before controlling, the plots are like below, where slip ratio decrease to 0 when both wheels reach max speed(800rads/s). Before reaching max speed, the difference of velocity between two wheels always exist. So clearly, it is bad traction control.

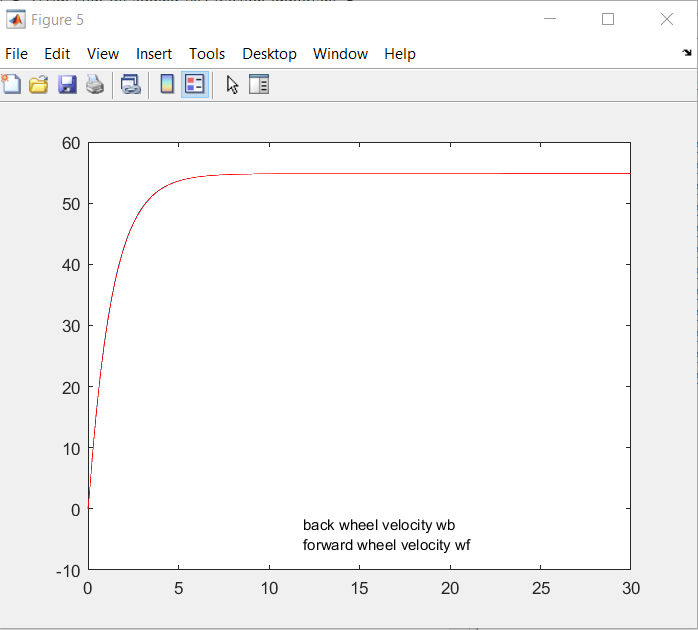


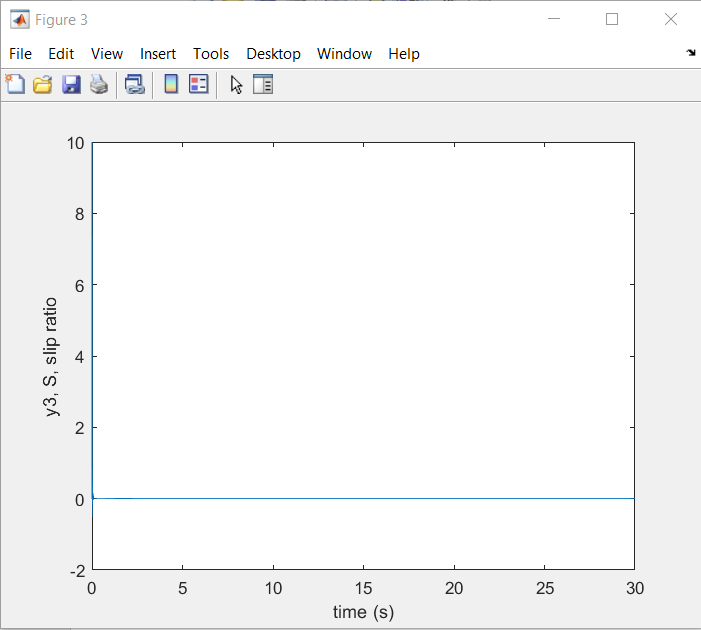


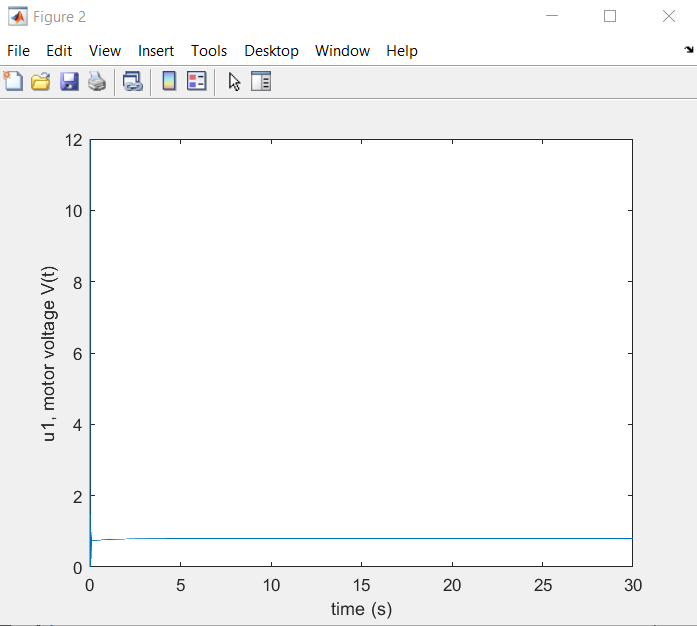


Now we set PID control variable to slipratio and try to control slipratio to 0.2.

Run1: kp = 4, ki=kd=0;







backwheel =

struct with fields:

RiseTime: 2.8655

SettlingTime: 5.1271

SettlingMin: 49.3378

SettlingMax: 54.8178

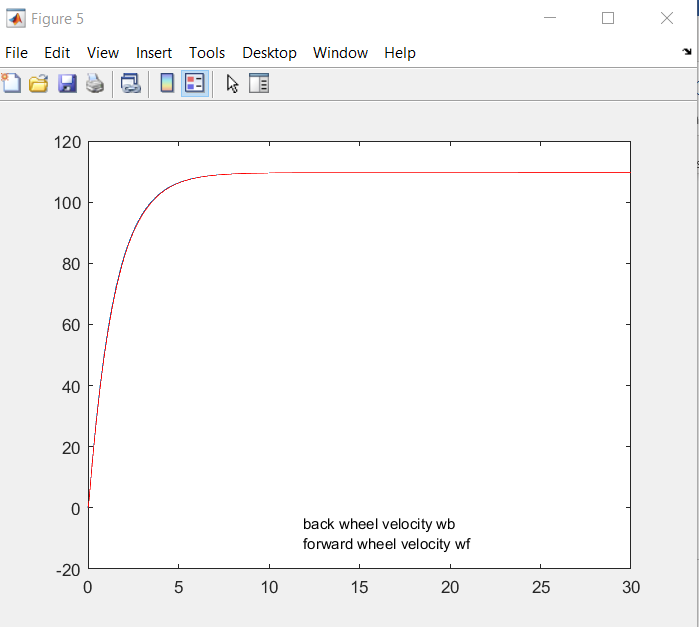
Overshoot: 0

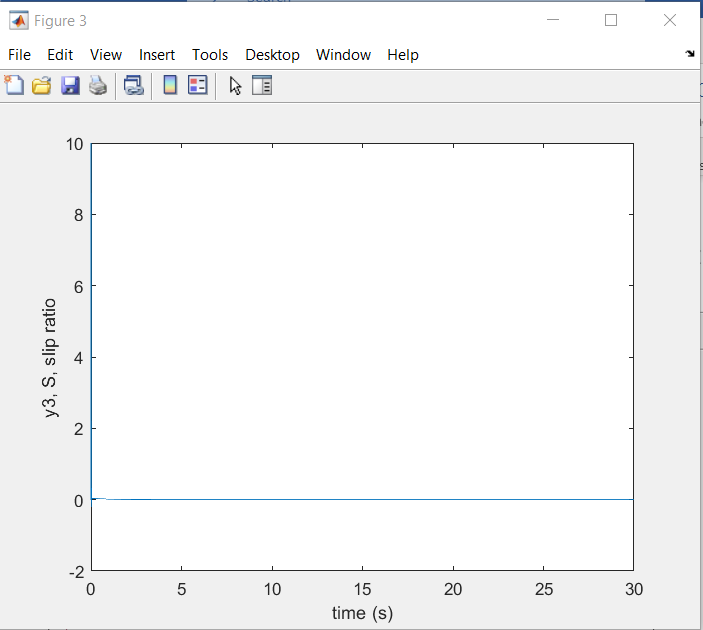
Undershoot: 0.0317

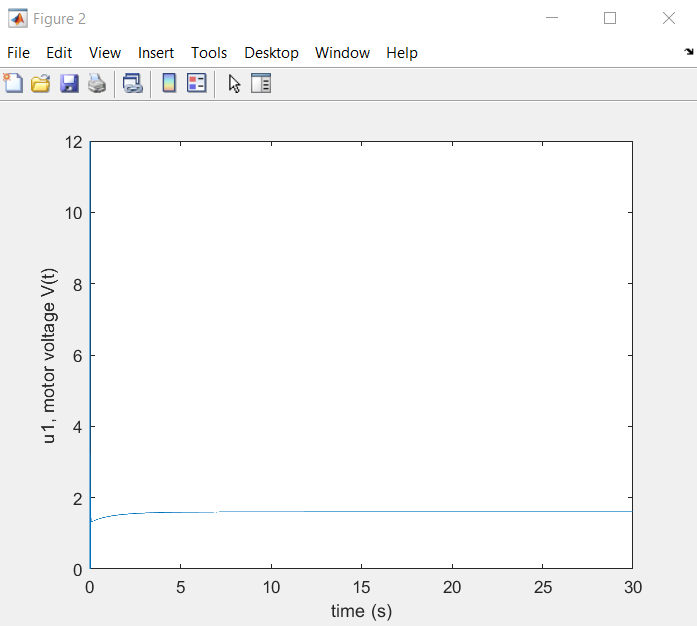
Peak: 54.8178

PeakTime: 24.7270

Run2: kp = 8, ki=kd=0;







backwheel =

struct with fields:

RiseTime: 3.1493

SettlingTime: 5.6322

SettlingMin: 98.6723

SettlingMax: 109.6356

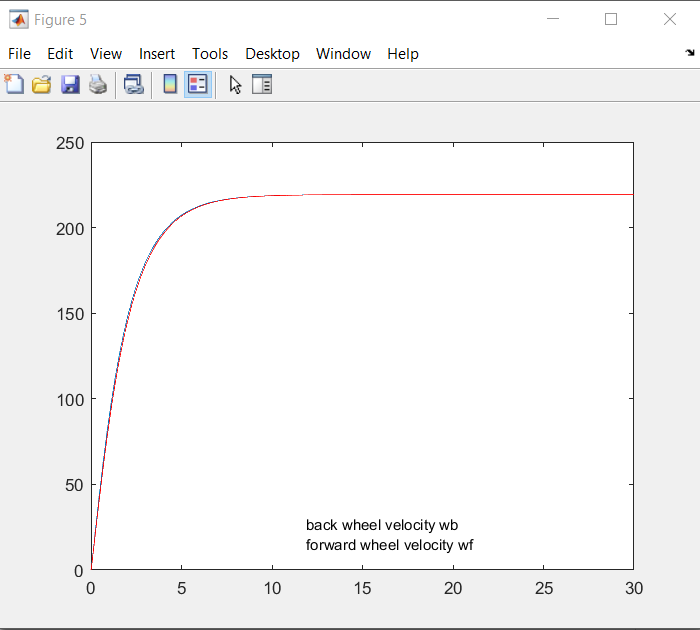
Overshoot: 0

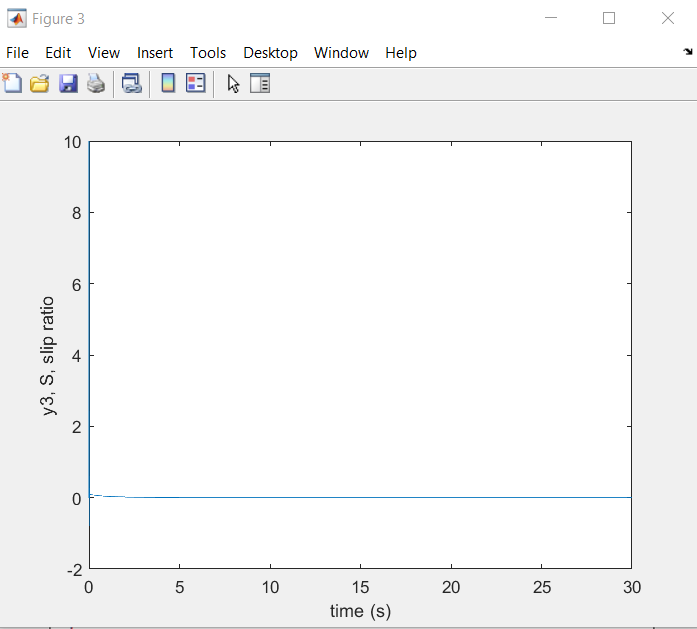
Undershoot: 0.0217

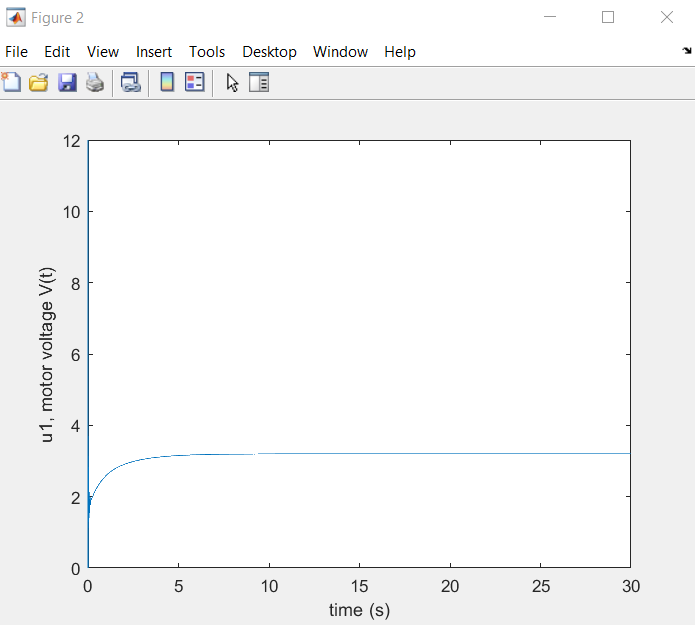
Peak: 109.6356

PeakTime: 23.8020

Run3: kp = 16, ki=kd=0;







backwheel =

struct with fields:

RiseTime: 3.8236

SettlingTime: 6.7746

SettlingMin: 197.3467

SettlingMax: 219.2712

Overshoot: 0

Undershoot: 0

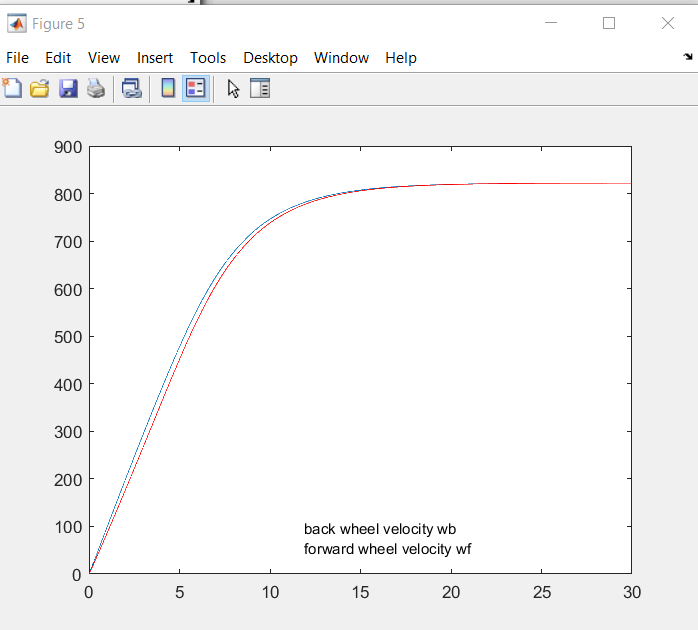
Peak: 219.2712

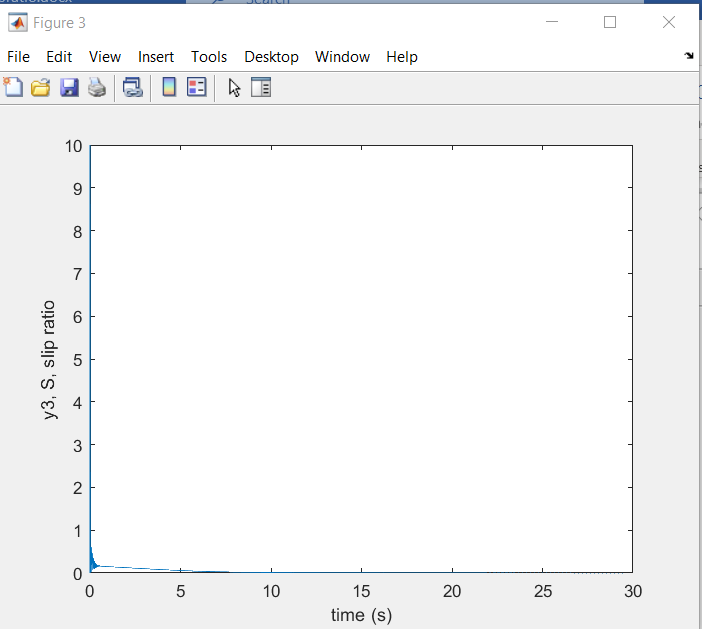
PeakTime: 28.9170

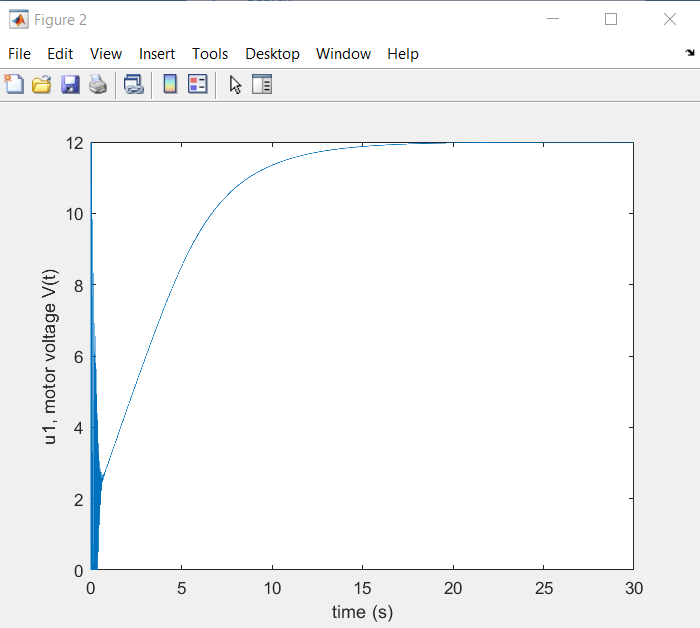
After 3 runs, we find out kp has some proportional correlation with the final velocity. So if kp=16, wb=wf= 200-220, we expect if kp=(800/220)\*16 = 60 , wb=wf will be 800rads/s.

(we did a bonus test run at kp = 48, and we find the proportional correlation still, where final wb=wf = 640rads/s)

Run4: kp = 60







backwheel =

struct with fields:

RiseTime: 9.1096

SettlingTime: 15.0013

SettlingMin: 739.9487

SettlingMax: 822.1405

Overshoot: 0

Undershoot: 0

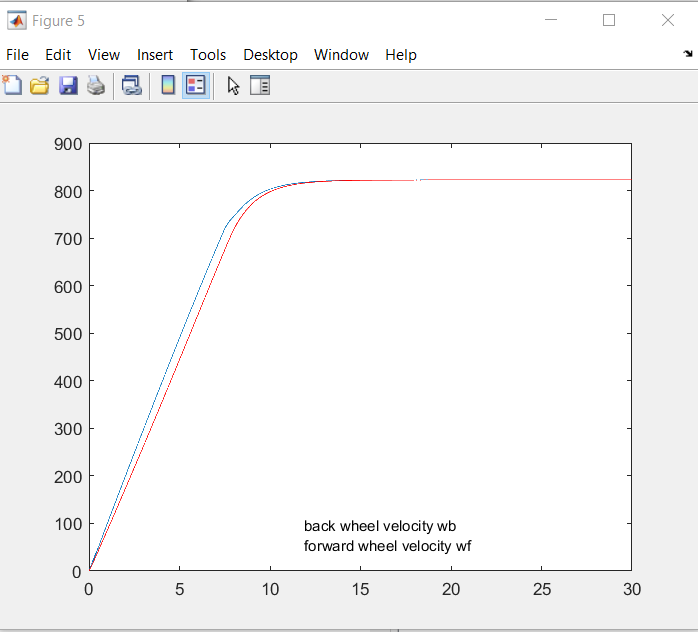
Peak: 822.1405

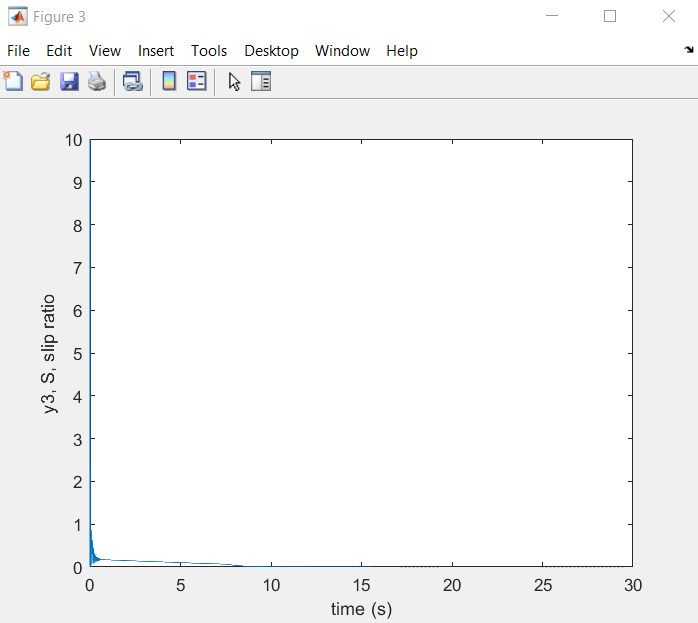
PeakTime: 29.9990

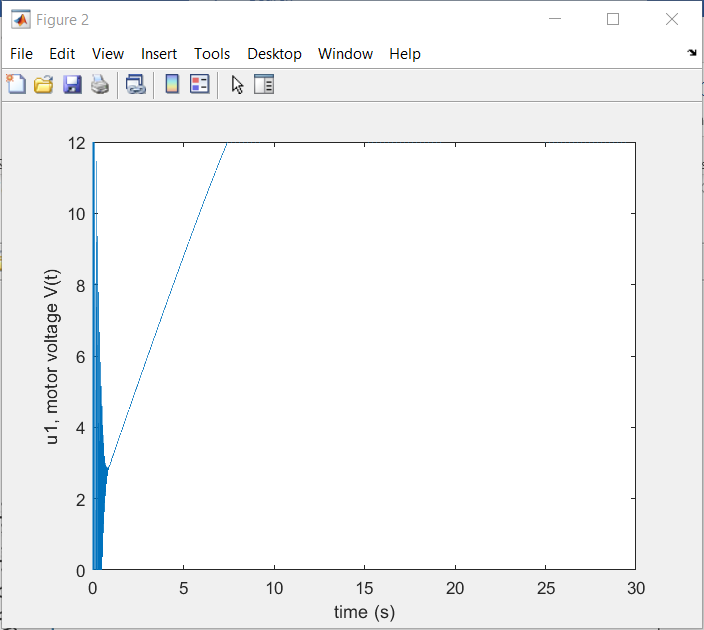
Thus, we can see from kp = 0 to 60, final speed could be controlled and maintained proportionally. Therefore, we find the “speedometer” range from kp [0 60]

Then, we continue increase kp to see the results.

Run5: kp = 90, ki=kd=0;







backwheel =

struct with fields:

RiseTime: 7.3529

SettlingTime: 10.5870

SettlingMin: 740.0419

SettlingMax: 822.2668

Overshoot: 0

Undershoot: 0

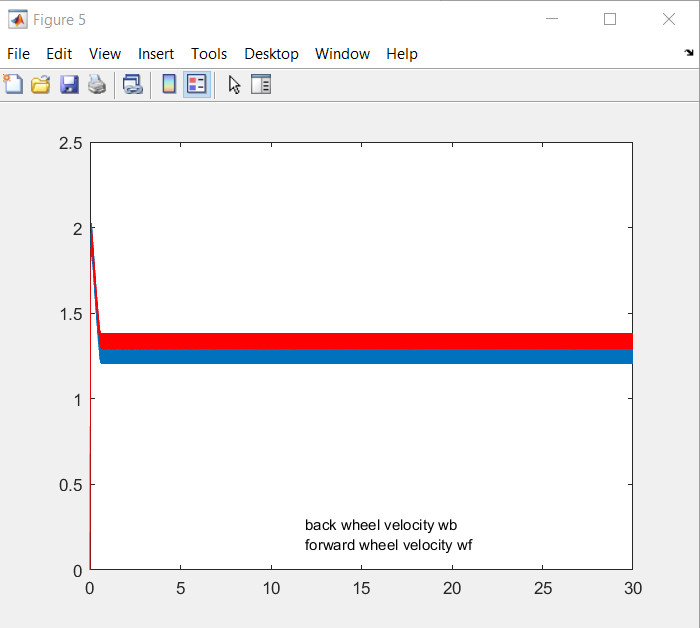
Peak: 822.2668

PeakTime: 29.3700

We find out that when kp = 60 to 90, the traction control will be slightly infected, but time response is significantly improved. Therefore, we set kp = 90.

Then we find ki and kd.

After several test runs, we find out only when ki = kd = 0, traction control can be coupled with launch control. Otherwise, the speed plot will be like plot below:



Therefore, we find out that in kp = 90,ki=kd=0, controlling slipratio to 0.2 can achieve good traction control.

Meanwhile, maintaining that PID parameter, we can stay at that speed (speed control), and the acceleration(time response of wheel velocity) is good compared to “no-PID“ performance. However, we can also separate speed control and launch control out. To achieve this goal, both speed control and launch control (and also brake control) involves one parameter: acceleration.

For speed control, acceleration should be maintained to 0;

For launch control, acceleration should be as large as possible.

For brake control, acceleration should be as small (a negative value) as possible.

# Speed control

For achieving speed control, we wrote this codes:

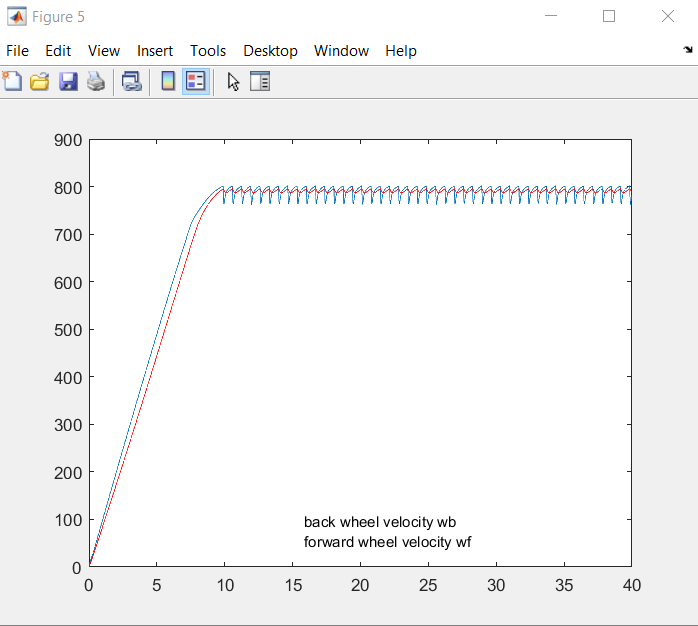
“if (y[2] < 795) pid\_traction(t, dt, slipratio, 0.2, u[1]); // launch control and traction control

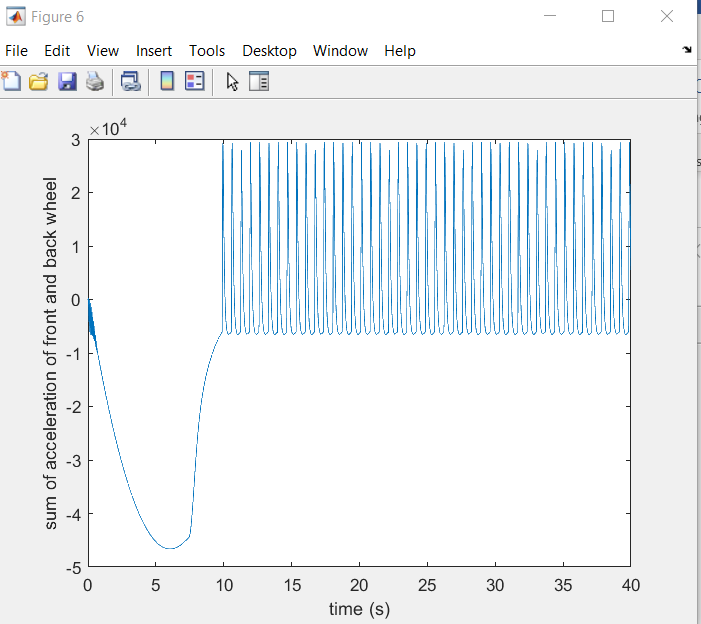
else pid\_speed (t, dt, a1+a2, 0, u[1]); // speed control”

So we know after max speed is attained, the “pid\_speed” function will take over, controlling the sum of accelerations(a1+a2) of front wheel and back wheel. (If we only control one acceleration, velocity difference may increase, that is, bad traction control)

If we set pid\_speed to kp = ki = kd = 0, this is what happened:

Run6: in pid\_speed, kp = ki = kd = 0;

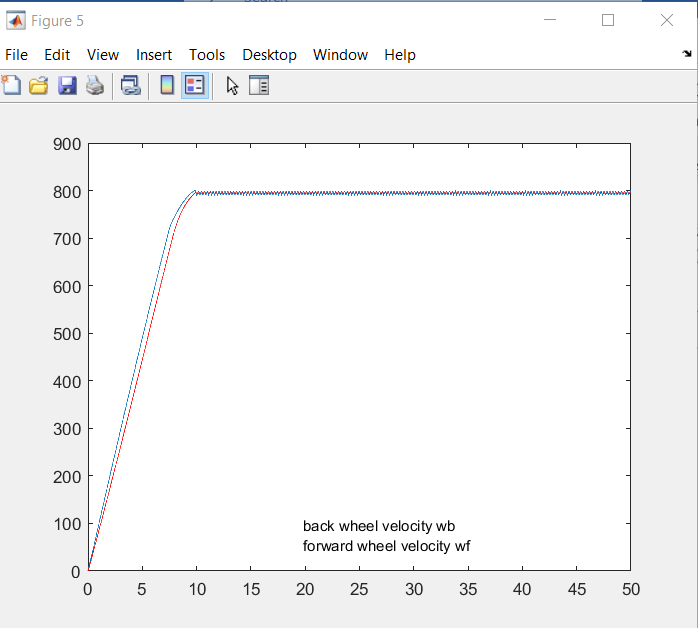
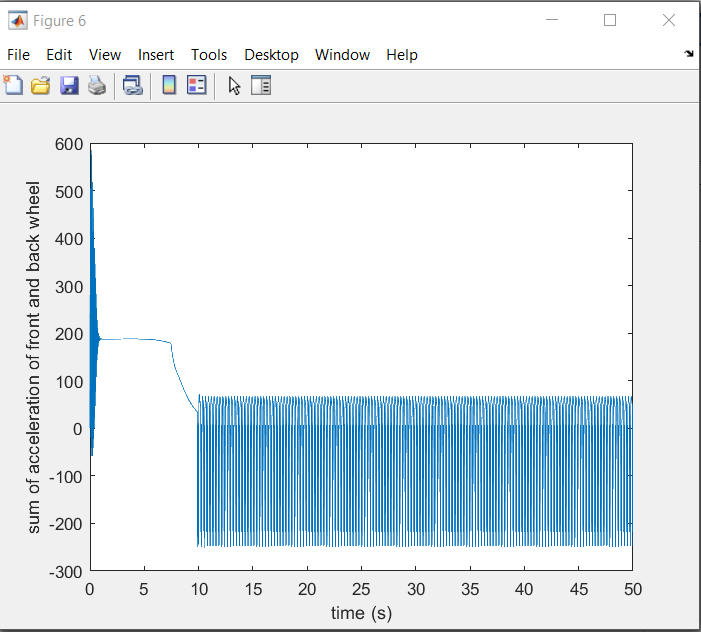




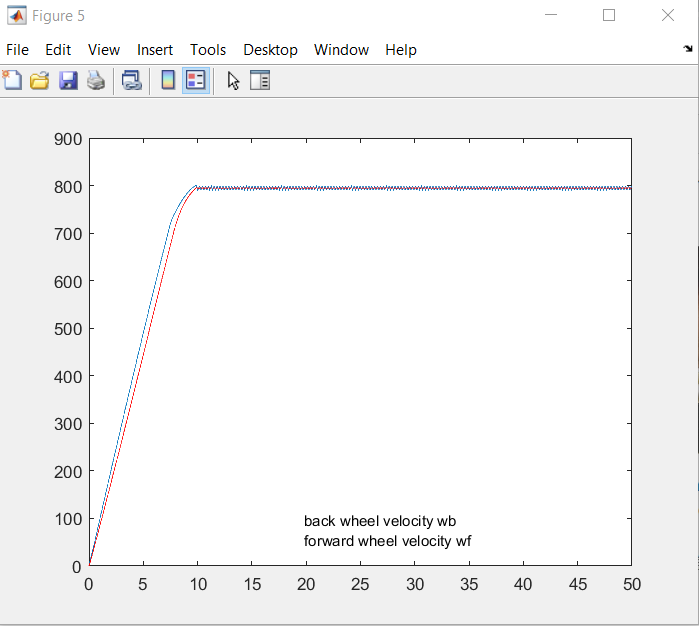
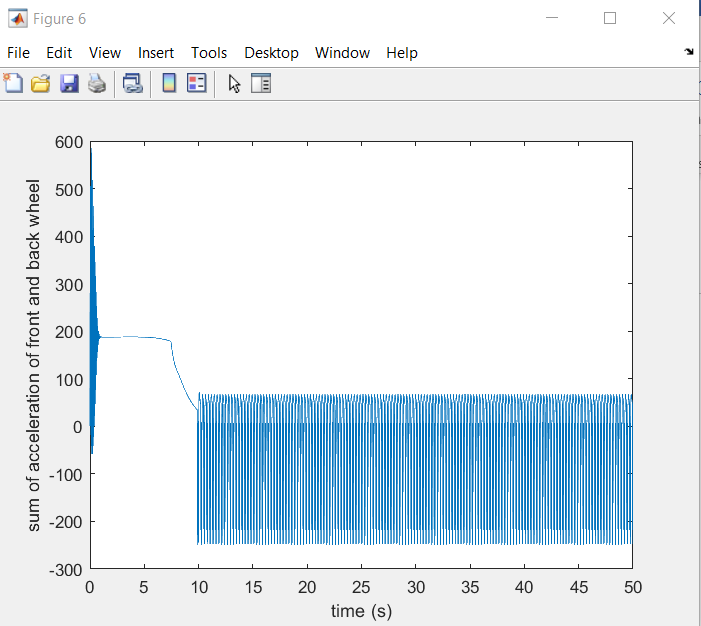
Obviously, speed control is not good. So we now tune the kp first. However, with different kp ranging from 0 to 300, we find that, the sum of a1+a2 is quite unstable.

Run7 to Run 9 : in pid\_speed, kp = 20, 50, 300

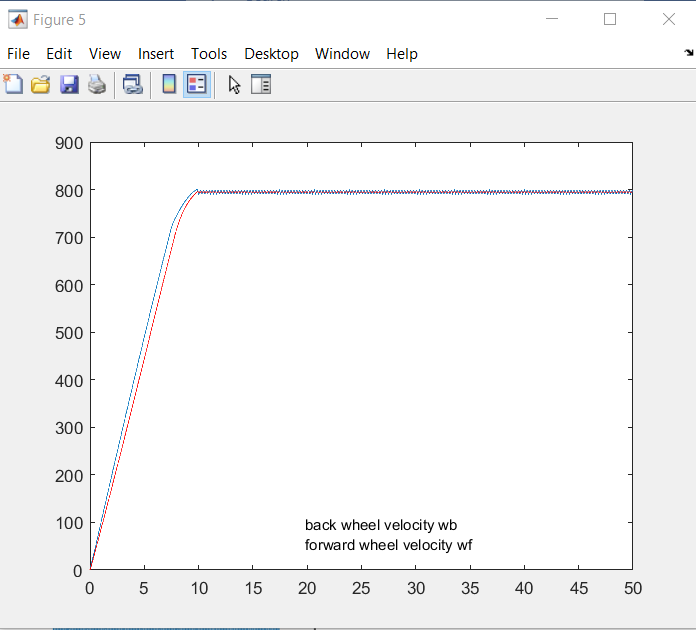
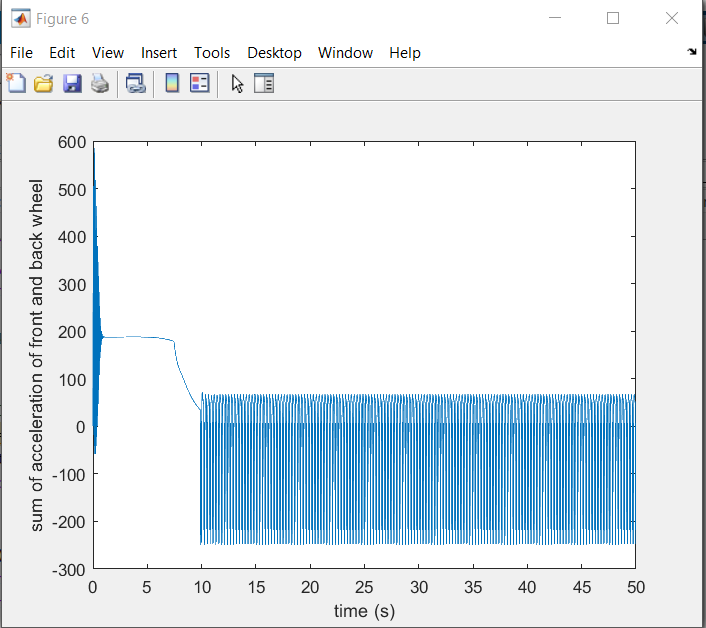
Kp= 20:



Kp = 50:



Kp = 300:

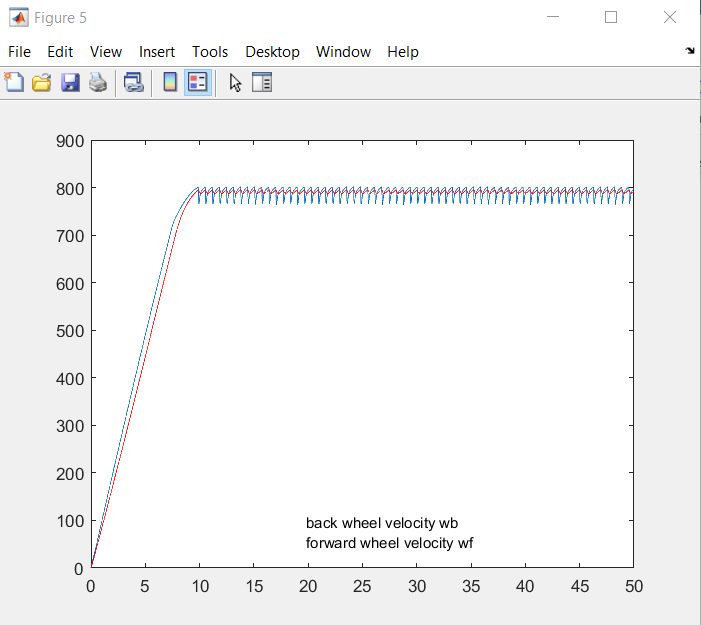
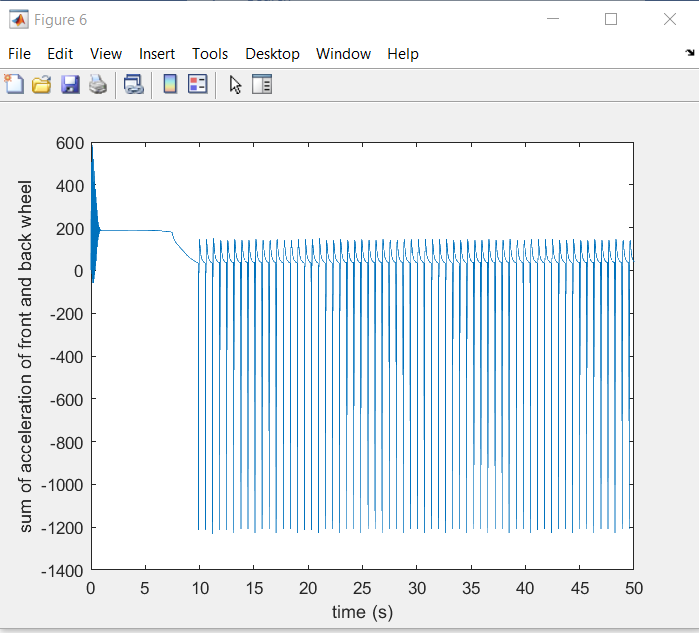


Then we realize that if we increase ki, the speed control will not be improved much. So we decided to check other settings, and we find out that the servo voltage is oscillating from 0v to 12v. So we added these codes:

” if (u - up > 1) u = up + 1;

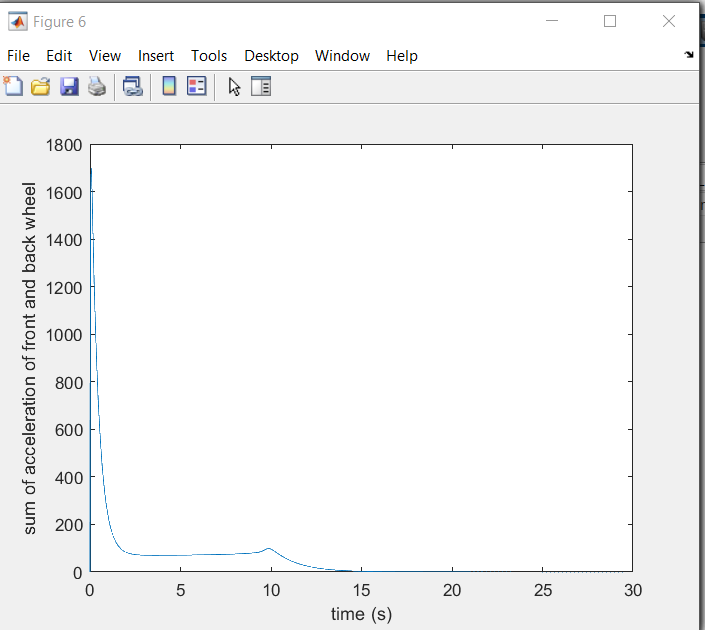
if (u - up < -1) u = up - 1;”

so that u will not change so much within dt. However, the result is even worse:



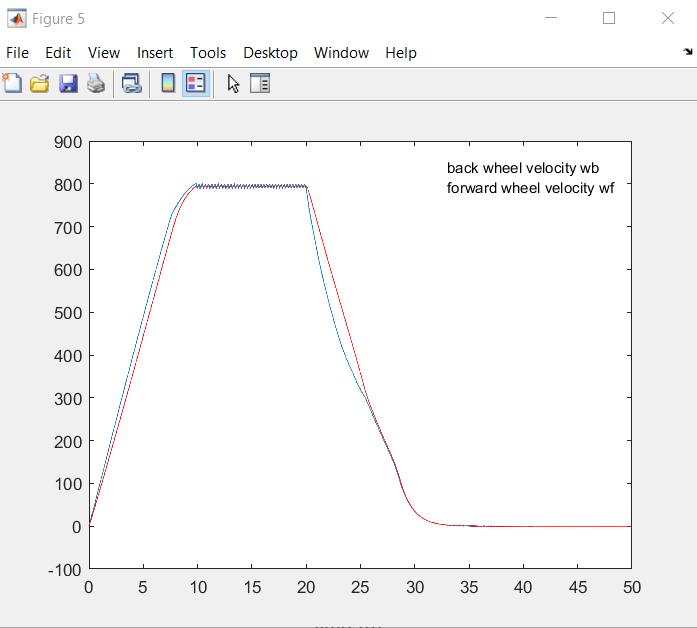
Therefore, we chose the less evil and set kp = 20, ki = kd = 0.

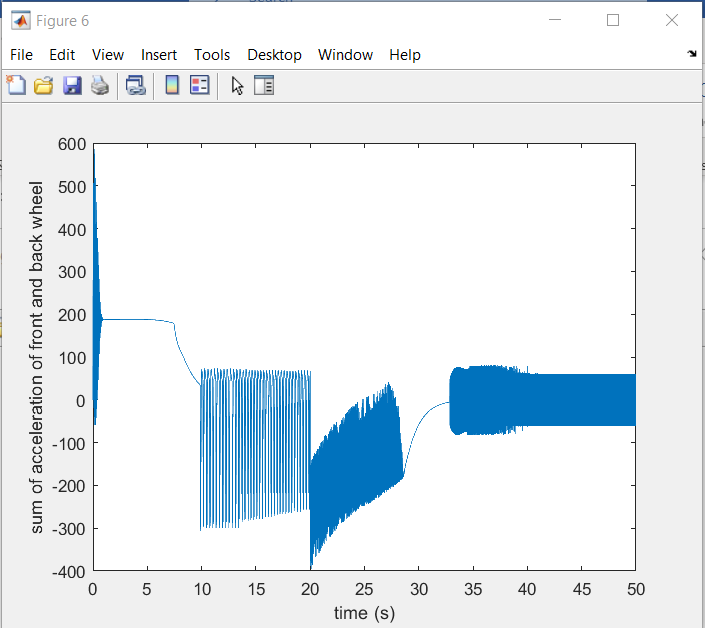
# Brake control

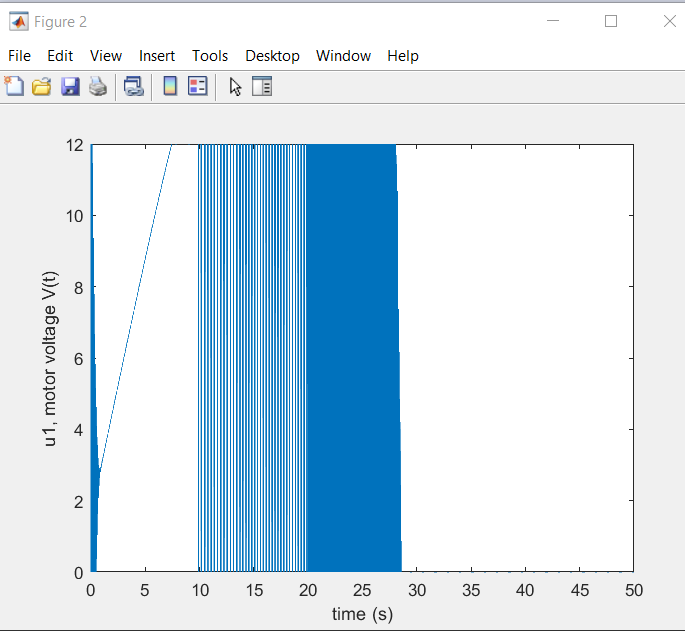
Without any pid control , the max sum of accelerations is about 1800 rads/ s2.

Hence we know the acceleration we are looking for is probably less than 1800 rads/s2. Through trail and error method, we tried different reference with kp = 1, then find out approximately when r = -180 rads/s, time response will be the best status while the traction control can also be obtained.

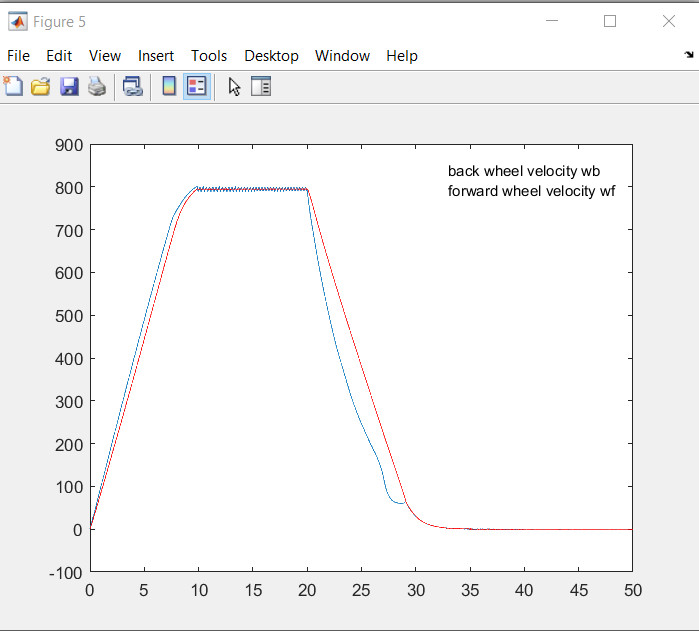
Run10: kp = 1,ki=kd = 0, reference = -180





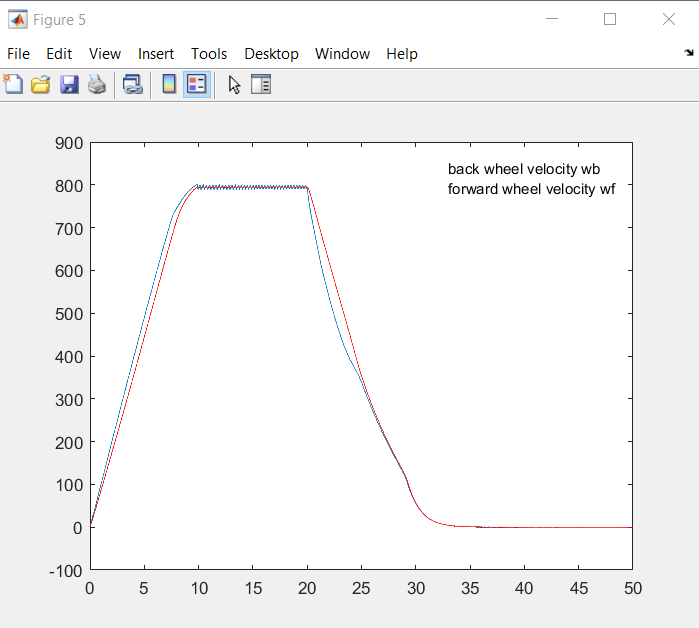


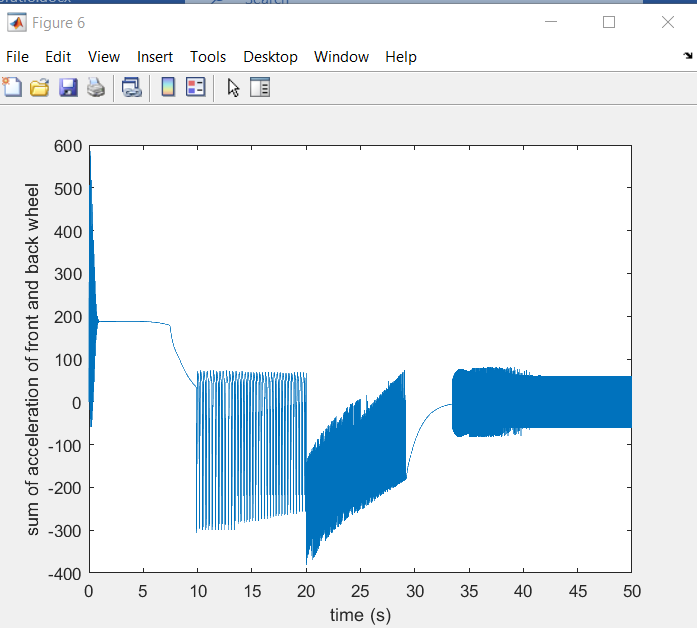
If reference is smaller than -180, traction control will be bad as the screenshot below:



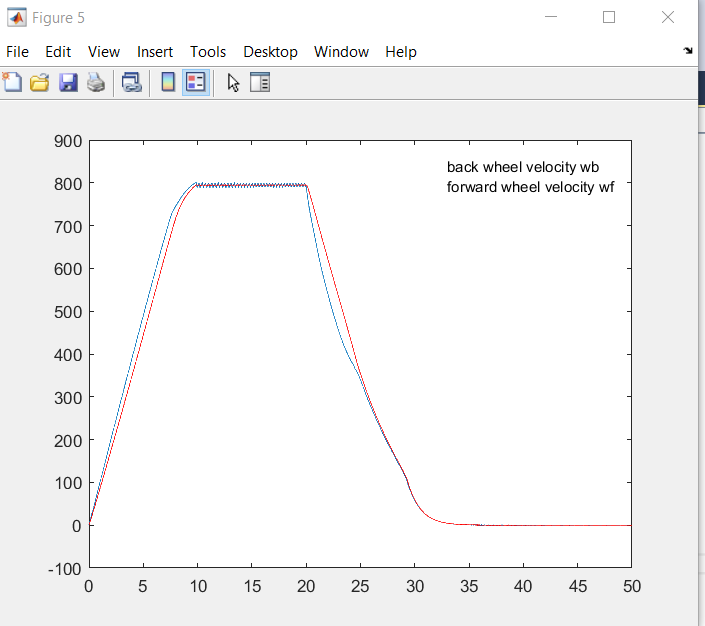
Therefore we set reference to -180. And try to find optimal kp first.

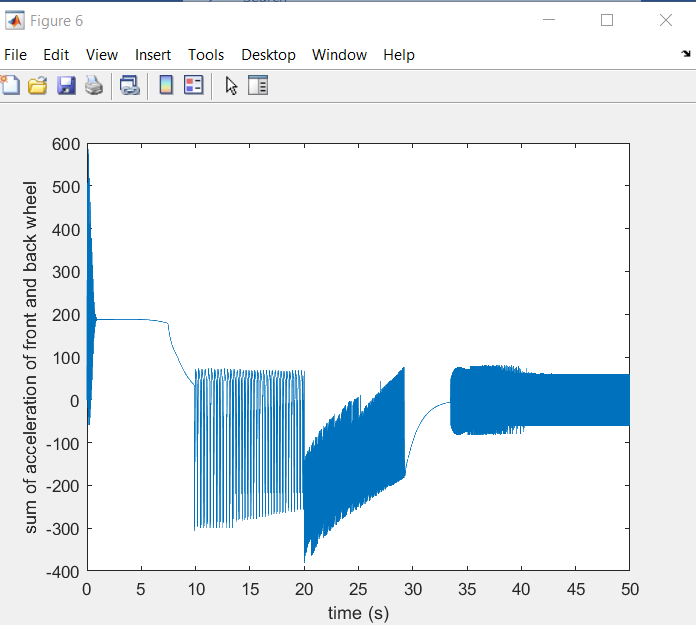
Run 11, kp = 20, ki = kd = 0:



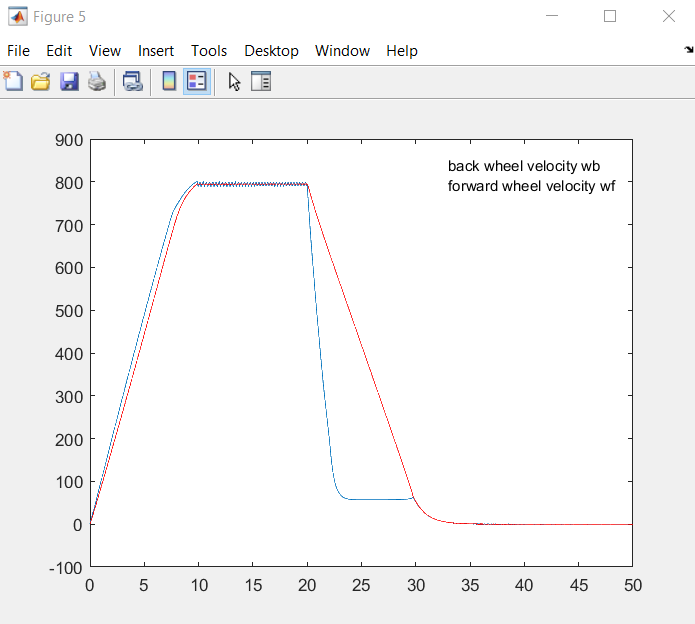


Run12 kp = 80, ki = kd = 0;





From kp=80 to 800, we find out that kp doesn’t effect time response anymore. And if we set ki or kd to a nonzero value, the traction control will be worse as shown:



So what if we use reference smaller than -180 (will result in bad traction control) but write an if statement? If velocity difference is larger than a certain value, we turn on traction control, otherwise we use brake control. In this inspiration, we wrote these codes:

if (t > 20)

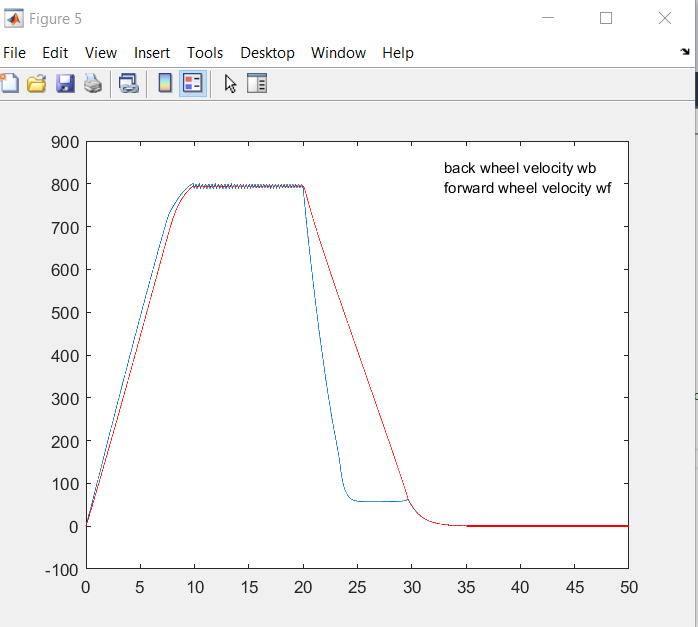
{

if(slipratio < 0.2) pid\_brake(t, dt, a1 + a2, -300, u[1]);

else pid\_traction(t, dt, slipratio, 0.2, u[1]);

}

But the result is also not good:



So we stick to the original plan: set reference to -180.

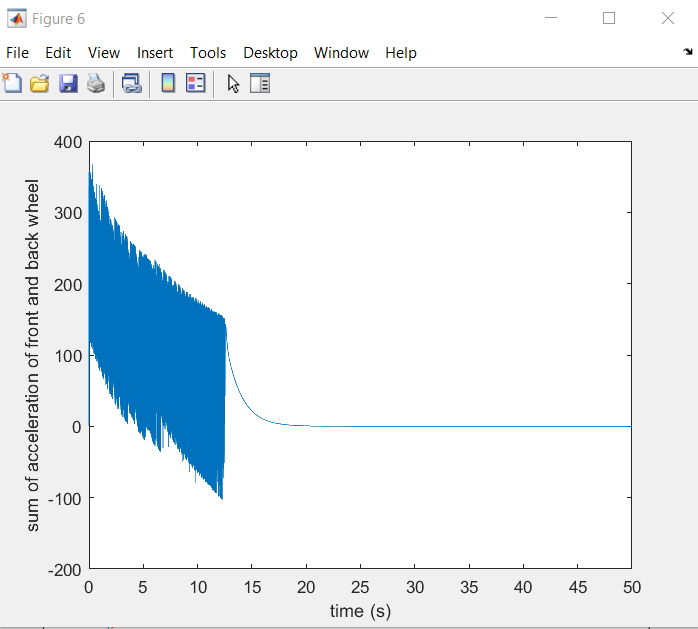
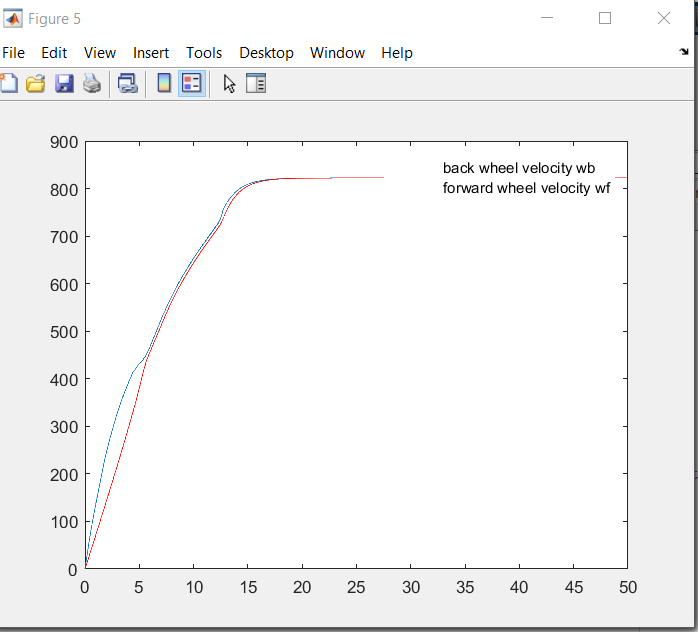
(However, if reference is -180, then the codes above can work---of course because when r= -180 traction control is good enough not to trigger the pid\_traction!)

Therefore we find out that kp = 20, ki=kd=0 can work if reference is set right.

# Launch control

Similarly, we try to find the max sum of acceleration to “separate” launch function apart. After several trial and error, we find out that r = 150 may be the max value:

Run 13: kp =10, ki=kd=0, in pid\_launch function, reference = 150



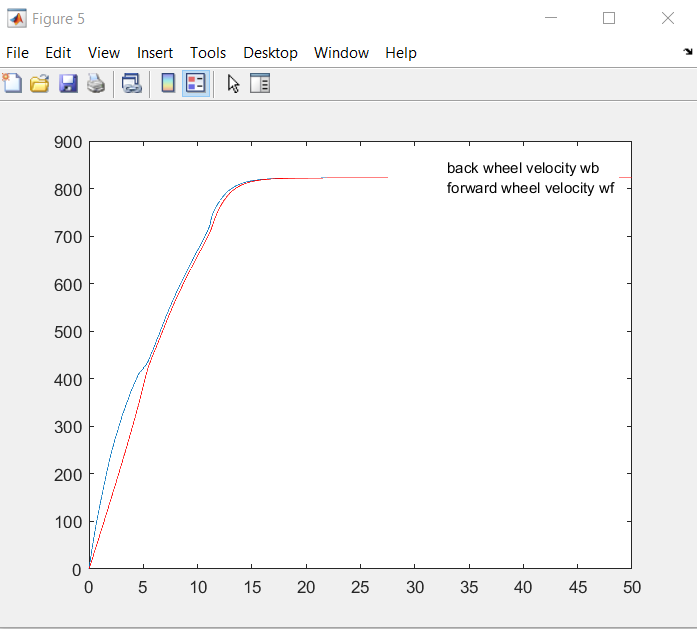
Although, using “if statement” like the example above can somehow improve time response further:

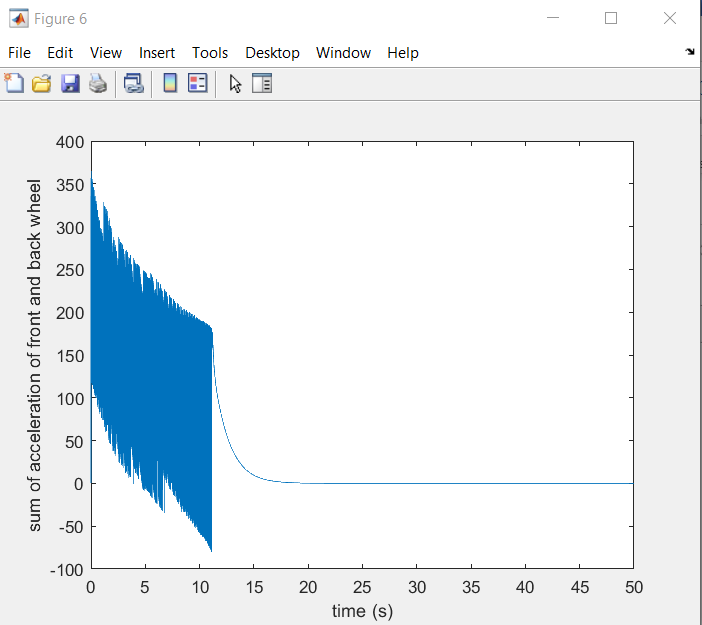
“if (slipratio < 0.2) pid\_launch(t, dt, a1 + a2, 400, u[1]);

else pid\_traction(t, dt, slipratio, 0.2, u[1]);”

However, as tested, the performance is not good enough compared to the method we used at the beginning of this document: only use pid\_traction:

Run14: kp = 80, ki = 10, kd=0:





Under this consideration, we abandon the use of pid\_launch and simply use pid\_traction when accelerating.

# Summary

Finally, we can use

” if (y[2] < 795) pid\_traction(t, dt, slipratio, 0.2, u[1]); // launch control and traction control

else pid\_speed (t, dt, a1+a2, 0, u[1]); // speed control

if (t > 20) pid\_brake(t, dt, a1 + a2, -180, u[1]); // brake control”

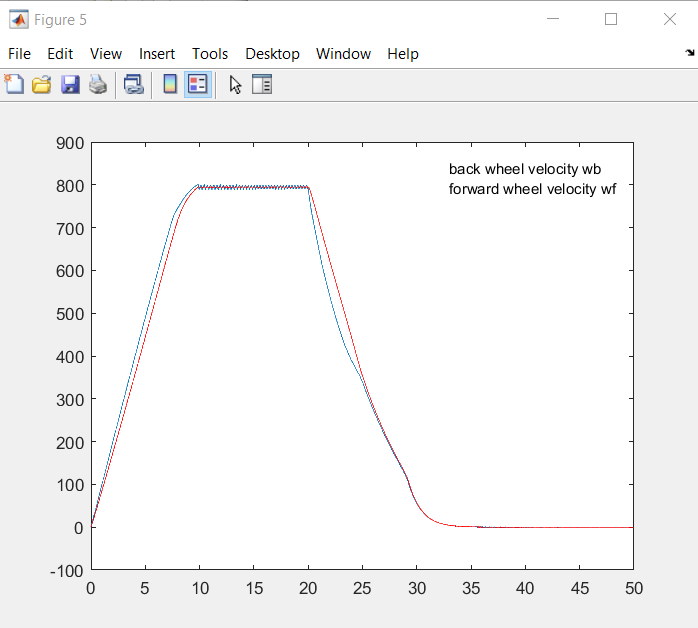
And the parameter is shown below:

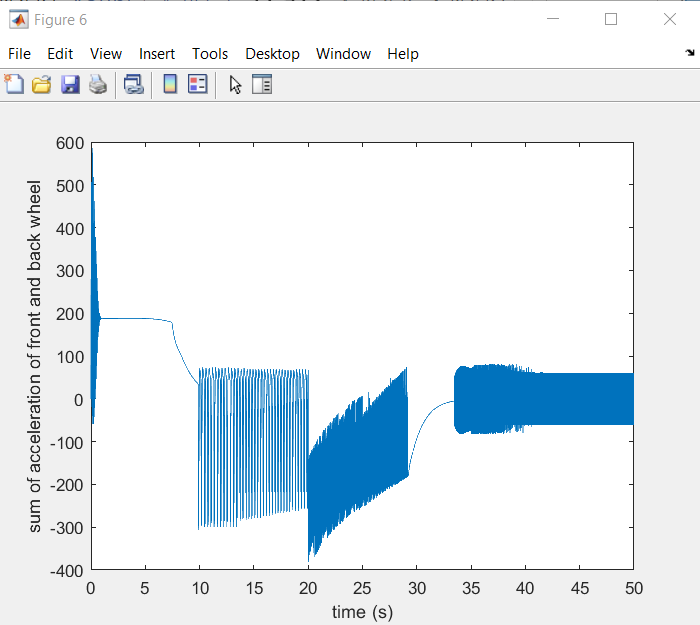
For pid\_traction, kp = 90, ki=kd=0;

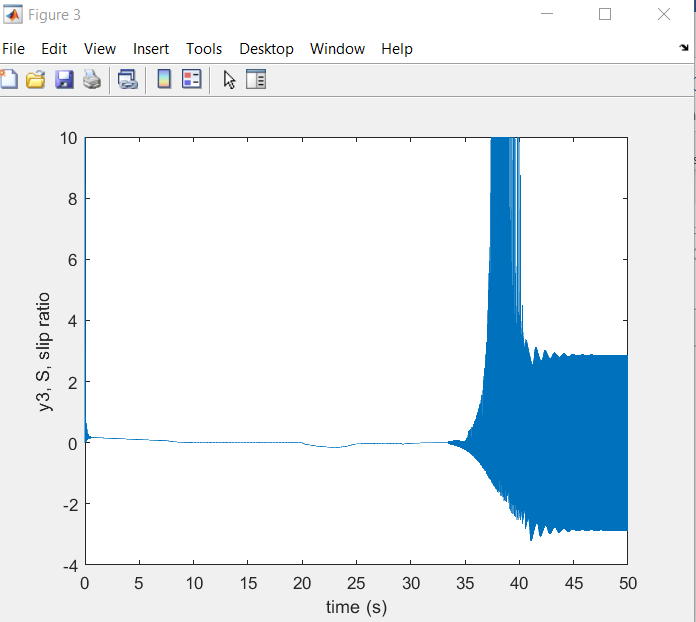
For pid\_speed, kp = 50,ki = 10, kd = 0;

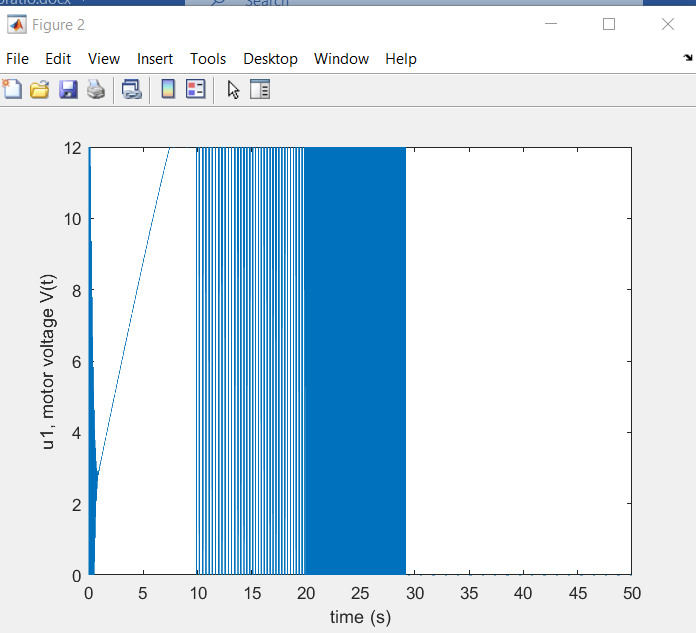
For pid\_brake, kp = 20 ,ki=kd=0;

And the results are:









# HIL simulation:

Due to some unexpected factor, the HIL simulation can only realize the launch and traction control, which is shown below:

