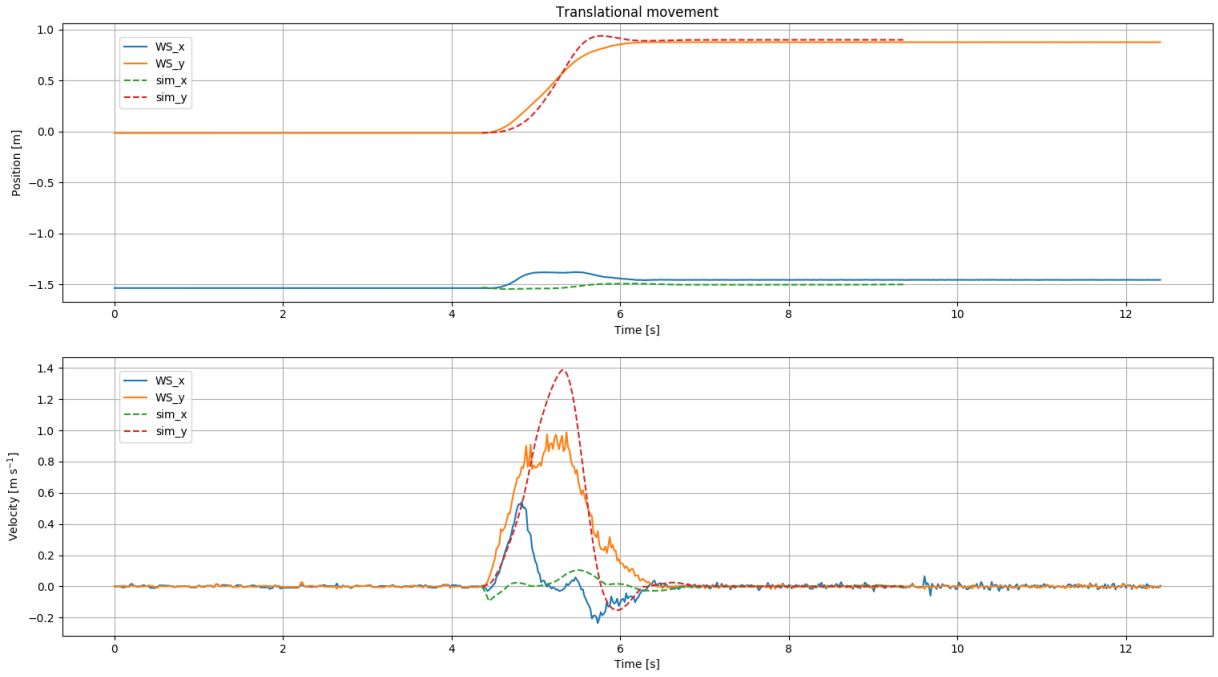
Simen Bootsma
27-09-2018

Comparison to simulation

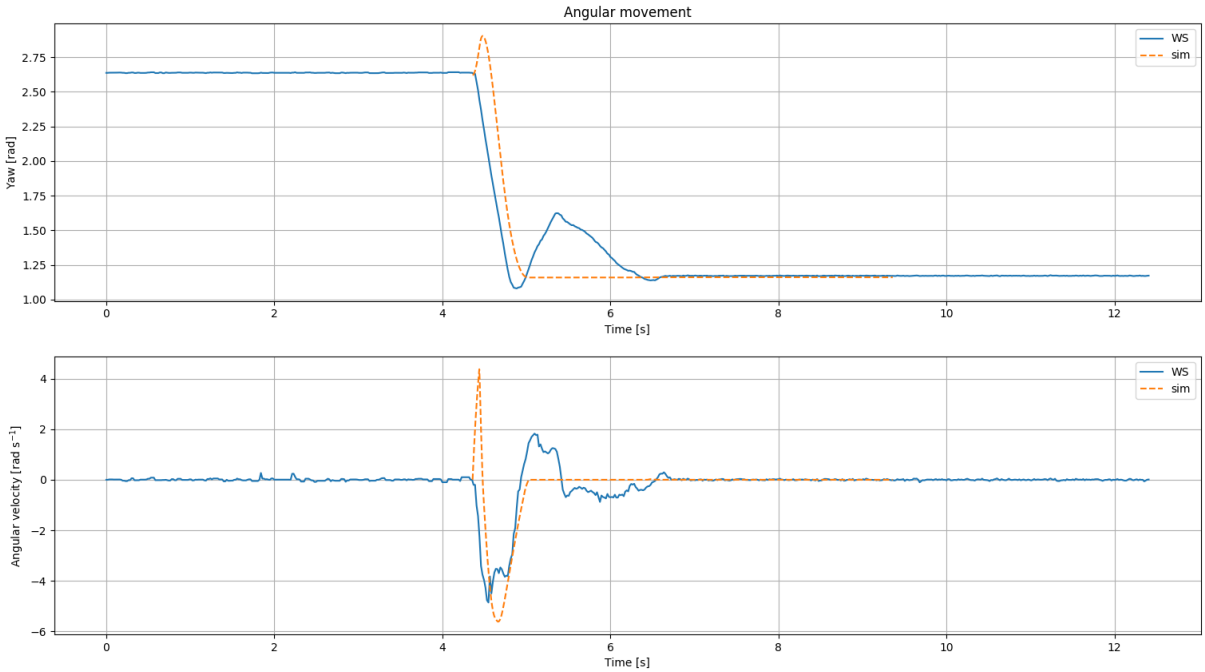
I recorded the data from the world_state topic and the XSens chip three times. After that, I let the 20-sim simulation go from the same start state to the same end state in order to see what the difference is between response when they are given the same commands. All data is saved in files which are also on the drive. I will discuss each of these files separately.

Straight1

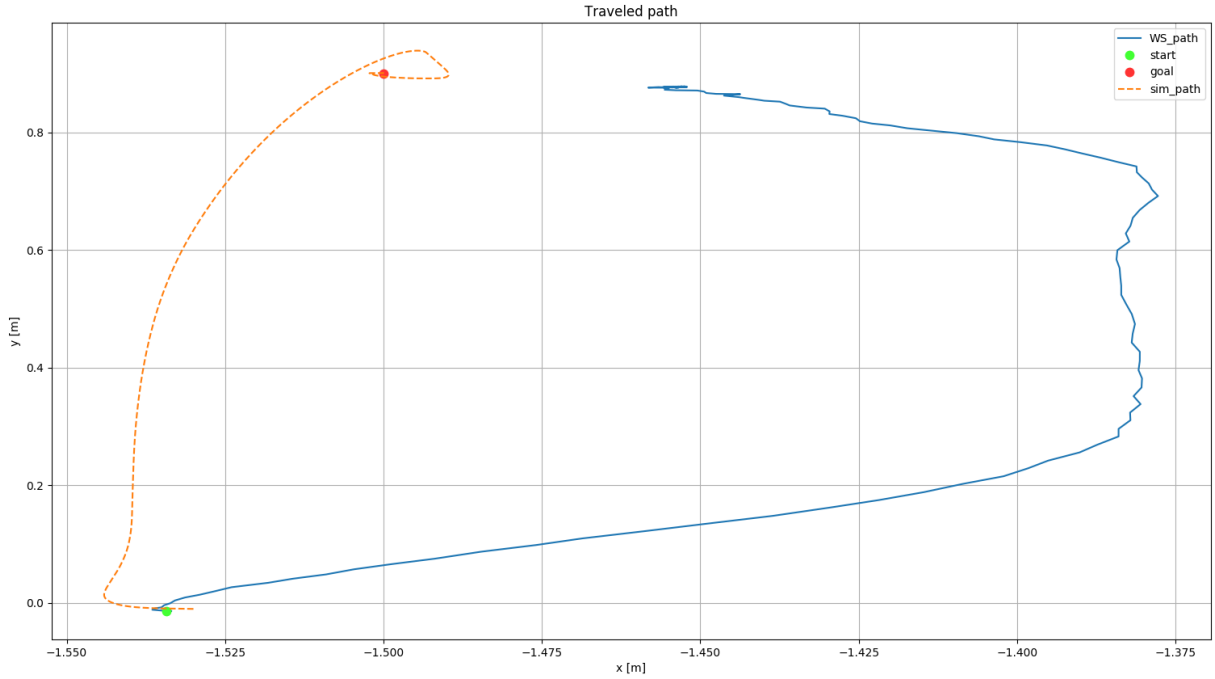
State at start: {x = -1.53; y = -0.01; yaw = 2.62}
State at end: {x = -1.50; y = 0.90; yaw = 1.16}



This all seems to work just fine. Notice that during the first 0.4 seconds the velocity increases equally in both directions.



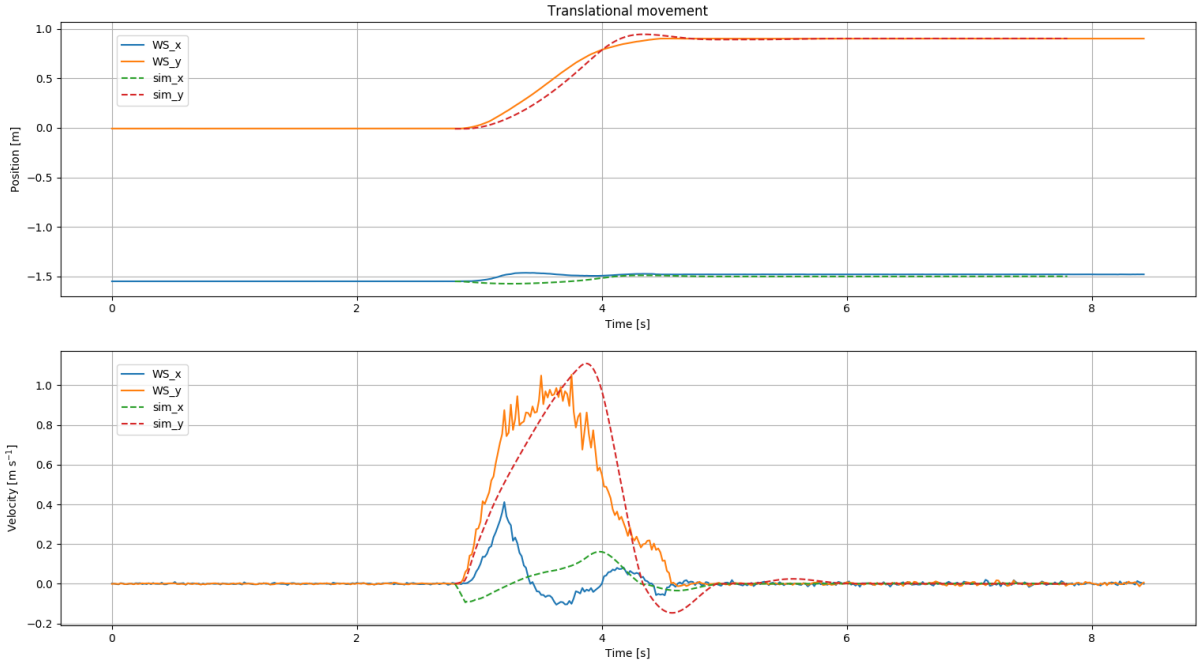
The simulation seems to rotate in the wrong direction prior to rotating to the desired yaw very fast and very accurately. The real robot seems to do well at first, but then suddenly rotates in the other way. The moment this happens, the robot also stops overshooting in the x-direction.



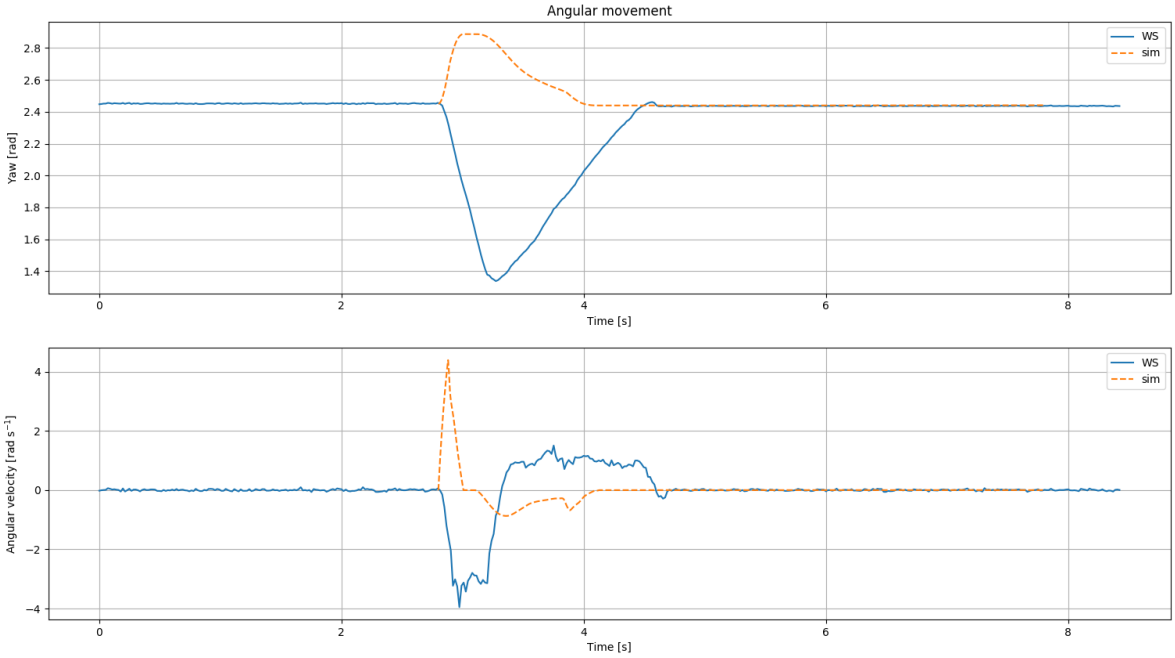
The simulation seems to do well, but the real robot overshoots a lot in the x-direction. Also, it only approaches the goal at a distance of 10 centimetres, which I find a lot.

Straight2

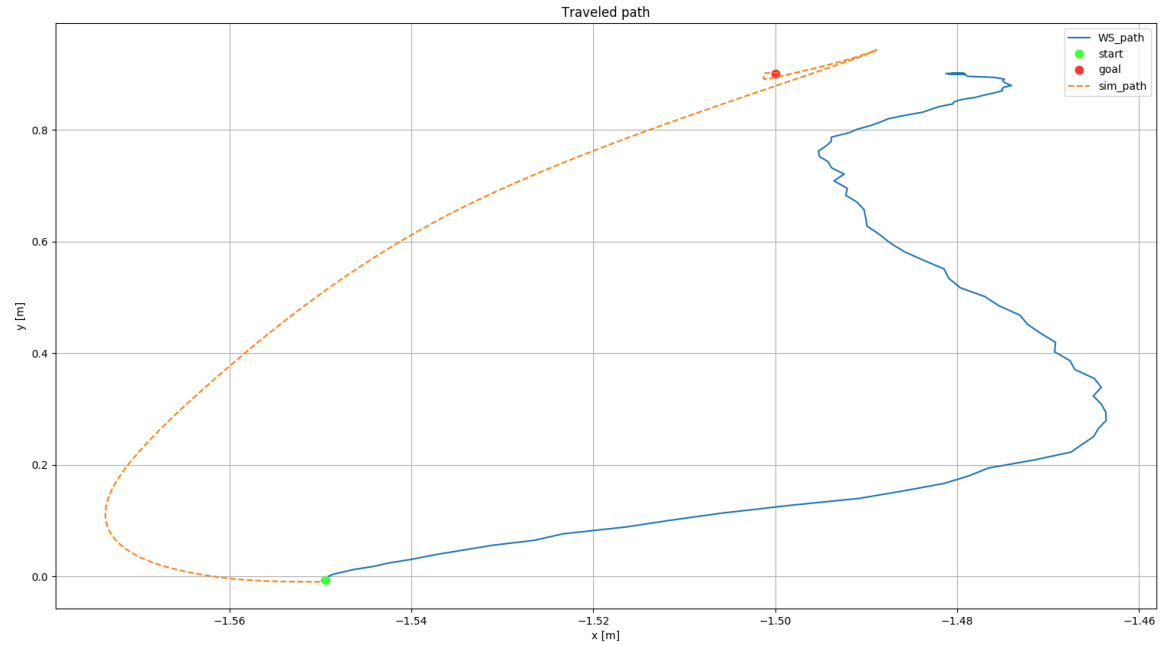
State at start: {x = -1.55; y = -0.01; yaw = 2.45}
State at end: {x = -1.50; y = 0.90; yaw = 2.44}



The response time is really good here. The real data almost looks better than the simulation data. One thing that stand out is that the acceleration in the x- and y-direction are a lot like each other. After 0.32 seconds, the x-velocity suddenly drops dramatically.



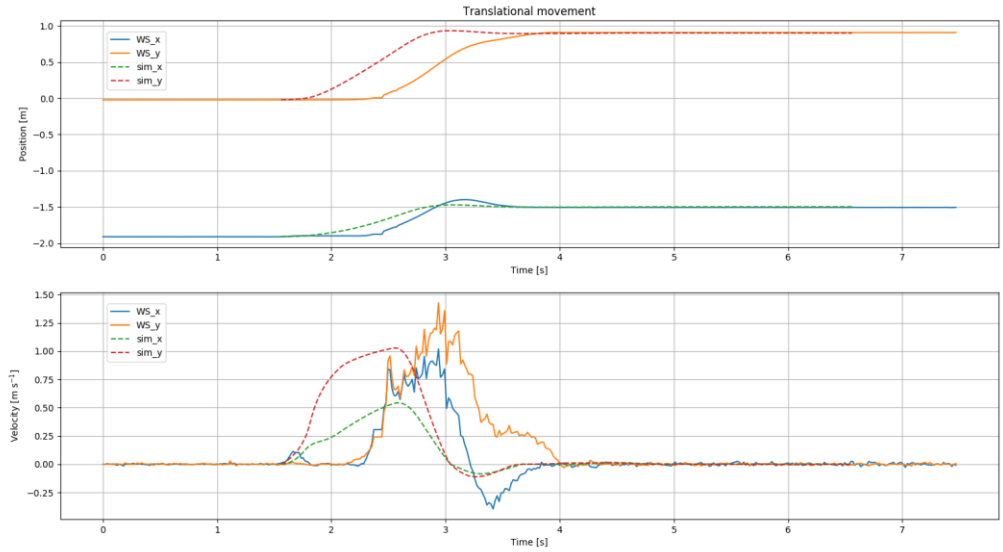
I have no idea what is going on here…. Only remark I can make is that the difference between the maximum simulation yaw and the minimum world state yaw is pi/2.



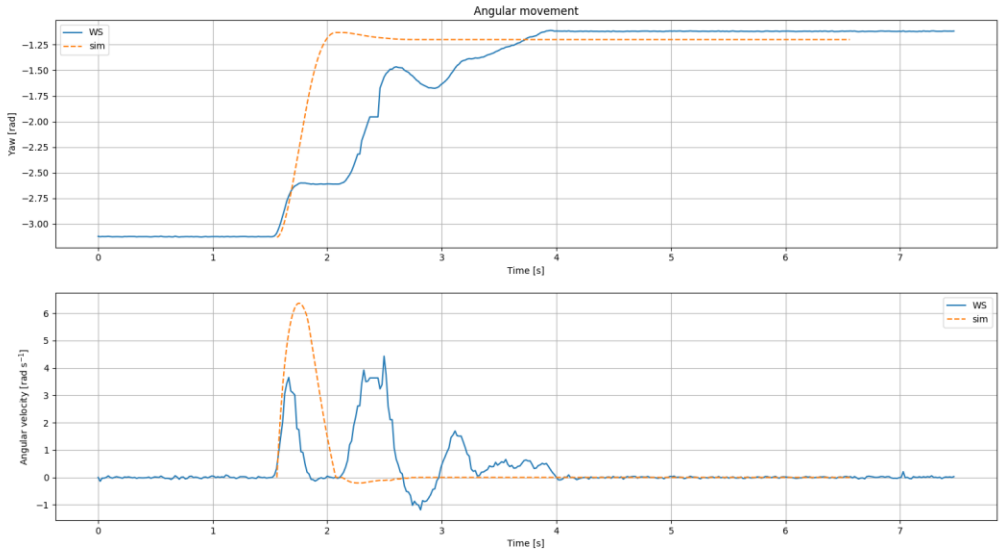
Although the deviation in the x-direction is a few centimetres, it doesn't seem like the most efficient way to travel for both the simulation and the real robot.

Straight3

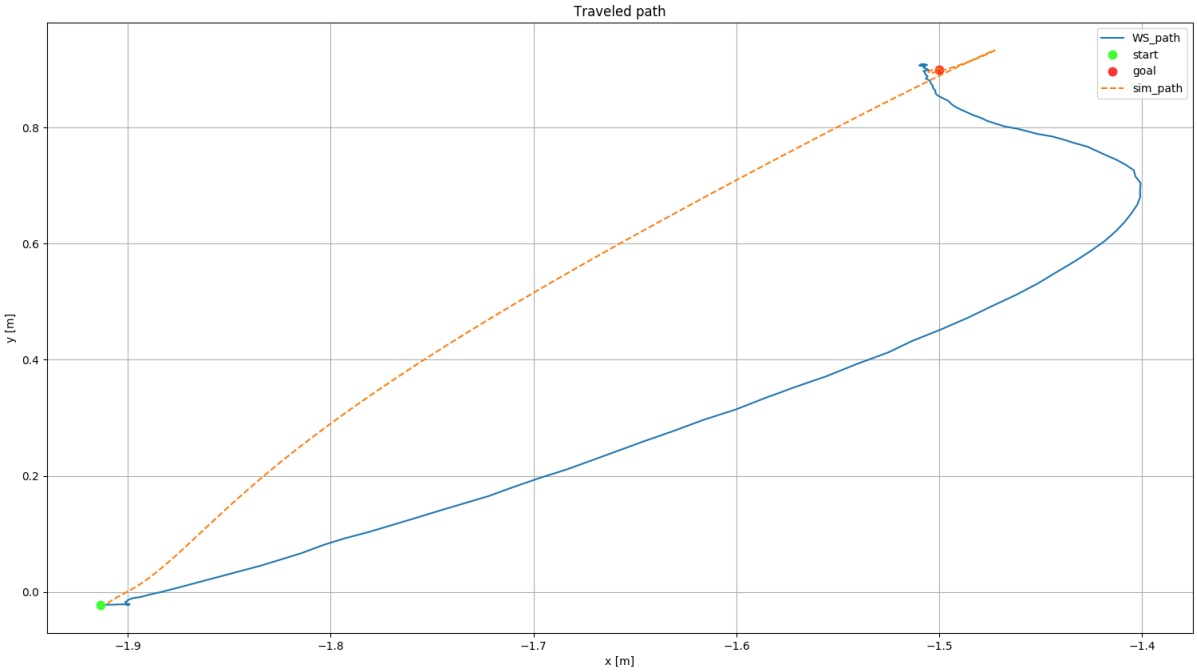
State at start: {x = -1.91; y = -0.02; yaw = -3.13}
State at end: {x = -1.50; y = 0.90; yaw = -1.20}



The position and velocity in both the x- and y-direction over time. You can see that the magnitude of the velocity seems good, but it takes ages for the robot to start moving. There is a little bump in the x-velocity, maybe the robot tried to move here but did not succeed? Might have something to do with rotating. Also, the robot starts accelerating exactly the same in both directions. Coincidence? I think not.



The yaw and the angular velocity over time. The robot starts rotating exactly the same as the simulation. However, after about 0.25 seconds, the robot suddenly stops rotating. For about half a second, it just doesn't do anything. Then when the robot starts to move it also starts rotating again. Comparing this to the translational movement graph, it seems like the robot tries to rotate and move simultaneously, but then ends up doing nothing for half a second.



This graph shows the path from the initial to the desired position as traveled by the robot as well as the simulation. The simulation seems to take a fairly efficient path. The robot however doesn't even go in the direction of the goal.

Summary

- The robot starts accelerating equally in both the x- and y-direction during the first 0.4 seconds. Cas says this might have something to do with the orientation of the robot. In a certain orientation, it might not have any choice but to move in both directions equally. It is useful to look at the movement in several directions with the same orientation. Cas also says it is possible to log the PWM that is sent to the wheels.

- Rotating and moving simultaneously seems to give problems. Cas checked and says that the encoders of the left back wheel are connected as 'BA', while the other wheels are 'AB'. To fix this, we need new PCB's. Question is whether this is worth it. Otherwise, we just need to deal with it.