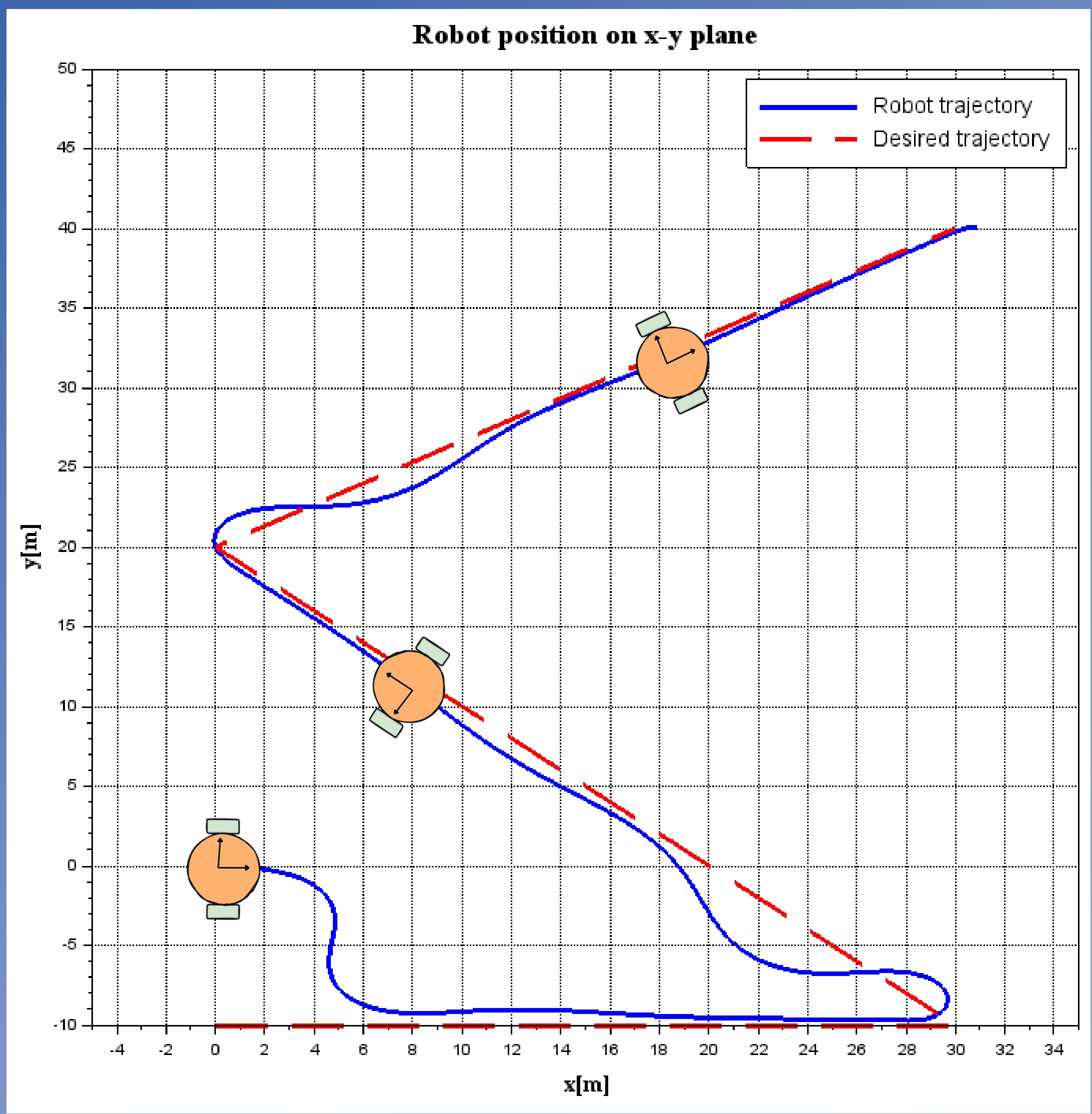
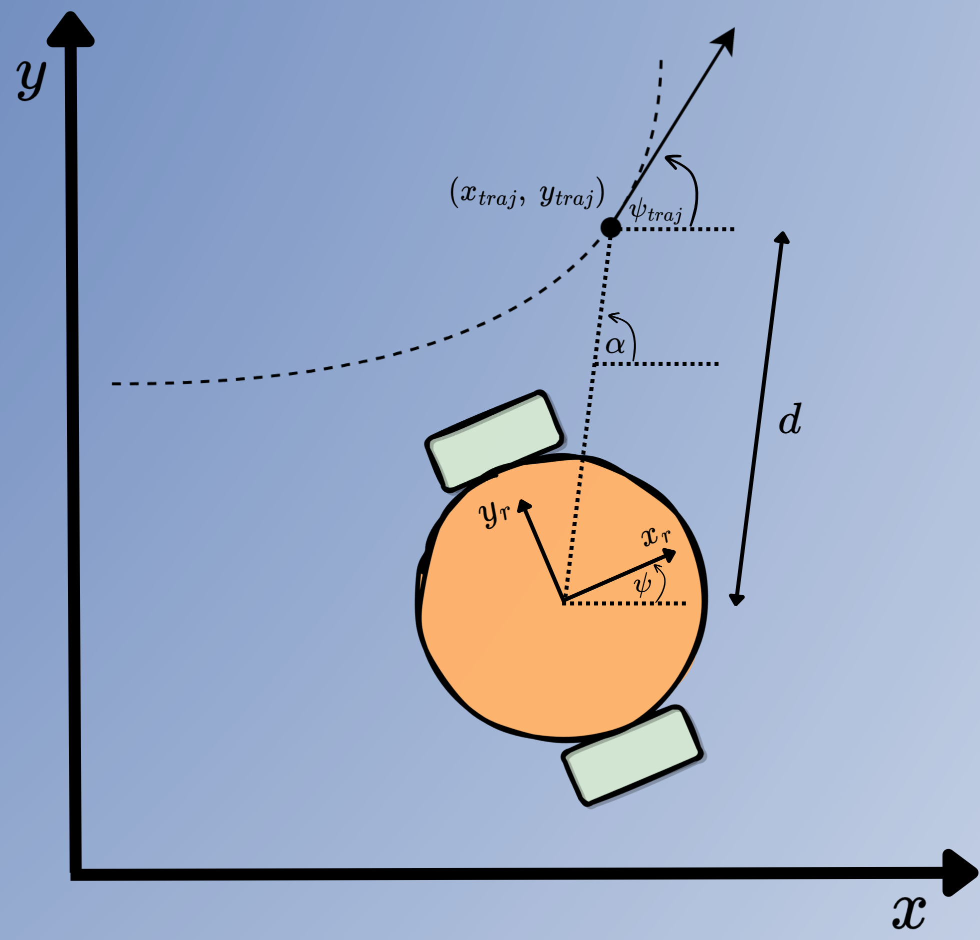
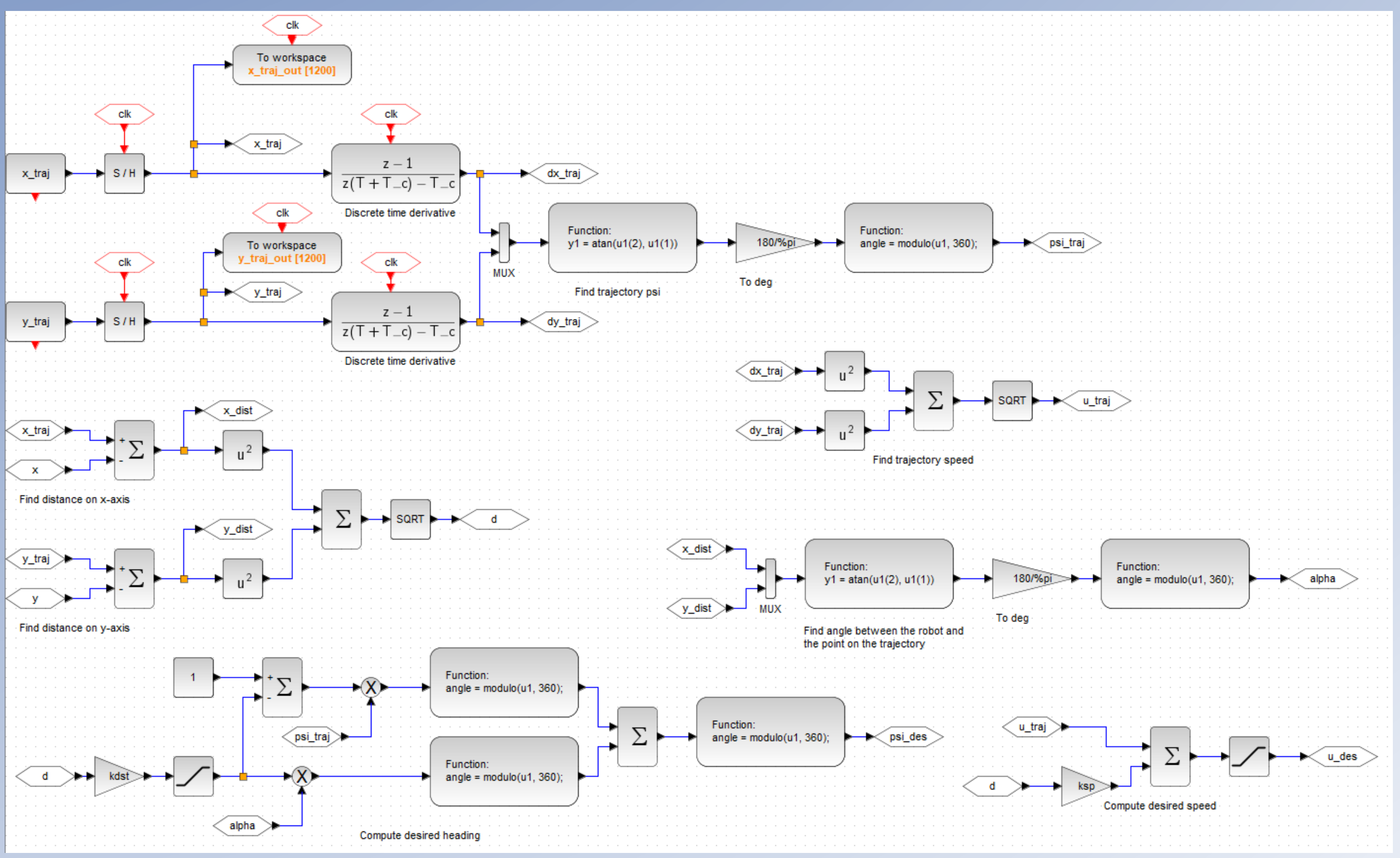
Trajectory control of a robot



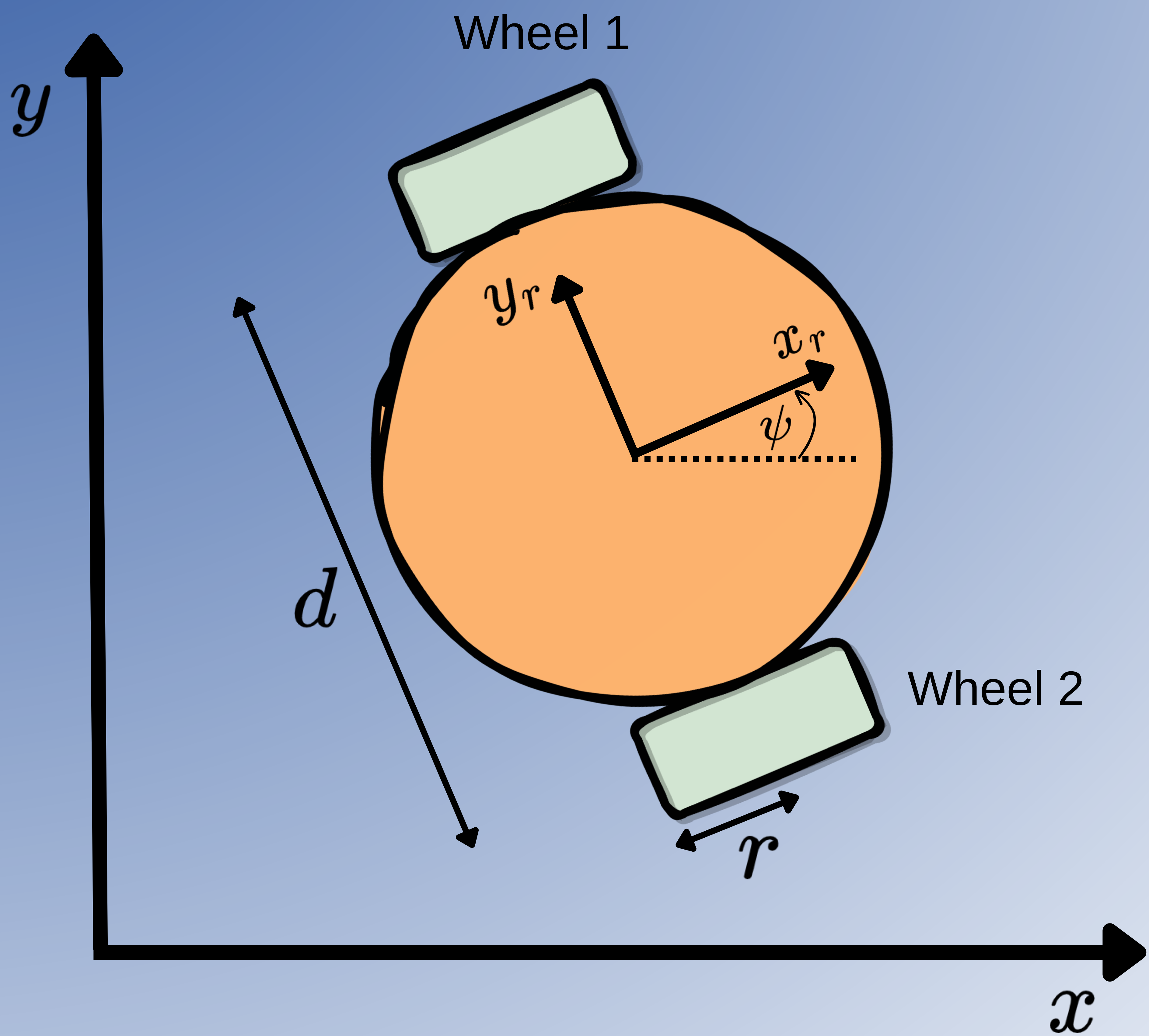




Simone Bertoni

Model
https://github.com/simorxb/trajectory-control

Robot description



Simone Bertoni

Differential equations of motion

Assuming a kinematic model where the two wheels can only move along x_r when they are spinning (i.e. they don't slip) and calling ω_1 and ω_2 the angular speed and u_1 and u_2 the linear speed of respectively wheel 1 and wheel 2, we have:

$$u_1 = \omega_1 r$$
$$u_2 = \omega_2 r$$

Let u and v be the linear speed of the centre of mass of the robot along x_r and y_r, then:

$$u = \omega_1 \frac{r}{2} + \omega_2 \frac{r}{2}$$
$$v = 0$$

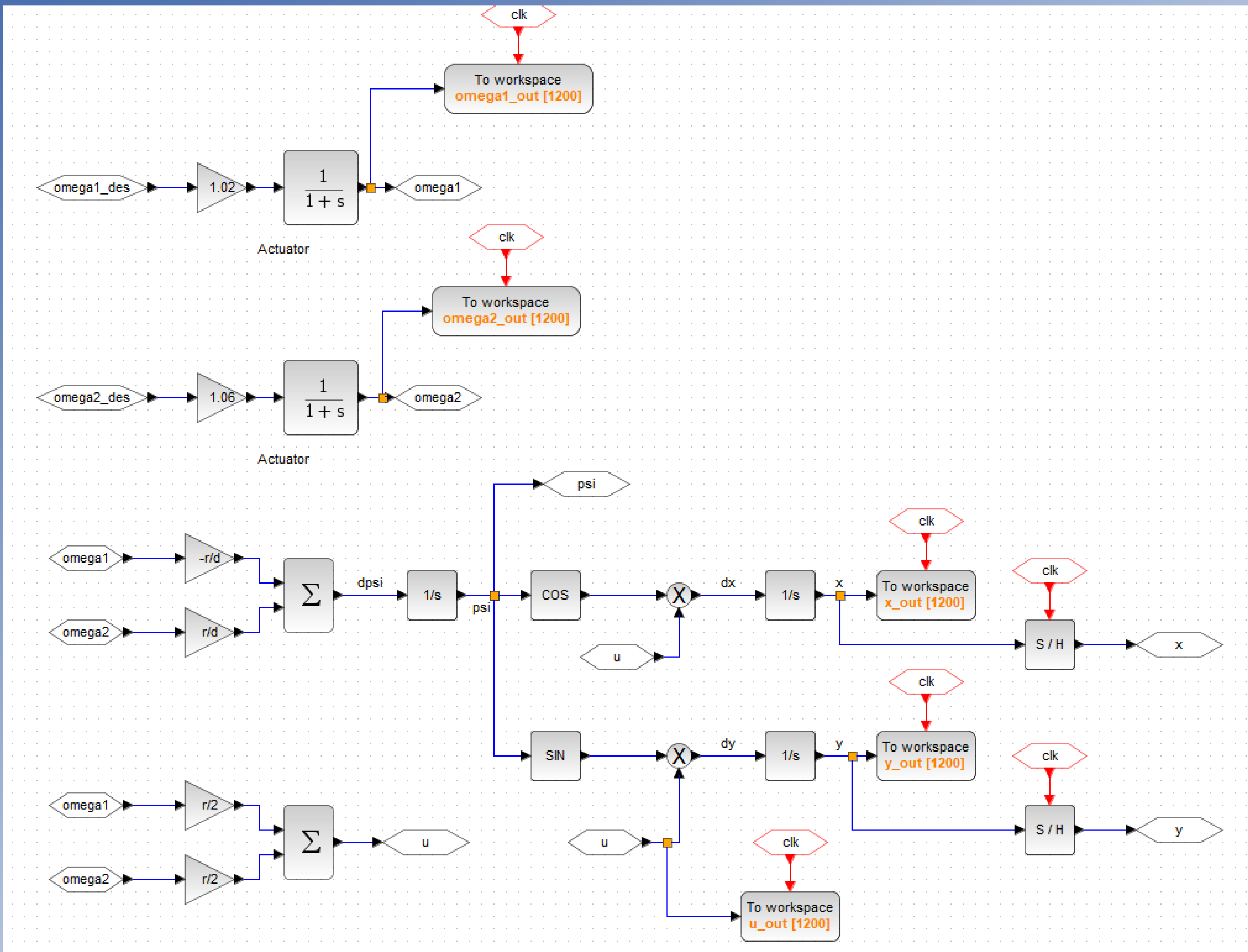
And finally the differential equation of motion, where the state variables are $[x, y, \psi]$:

$$\dot{x} = u \cos(\psi)$$
$$\dot{y} = u \sin(\psi)$$
$$\dot{\psi} = \omega_2 \frac{r}{d} - \omega_1 \frac{r}{d}$$

To make the model more realistic we assume that each wheel's speed controller responds as a first order with transfer function $\frac{1}{s+1}$ and has an error factor respectively of 1.02 and 1.06.

Simone Bertoni

Robot model (Xcos)



Simone Bertoni

Control system

The control system assumes 2 setpoints (that could be from a user or from a path planner):

$\psi_{deg_{des}}$: yaw angle in degrees
u_{des}: linear speed in m/s

To control the linear speed we assume that we have no access to the actual speed measurement, therefore we simply use the knowledge of the system:

$$\omega_{avg_{des}} = \frac{u_{des}}{r}$$

To control the yaw angle we use a PI controller that outputs $\Delta\omega_{des}$.

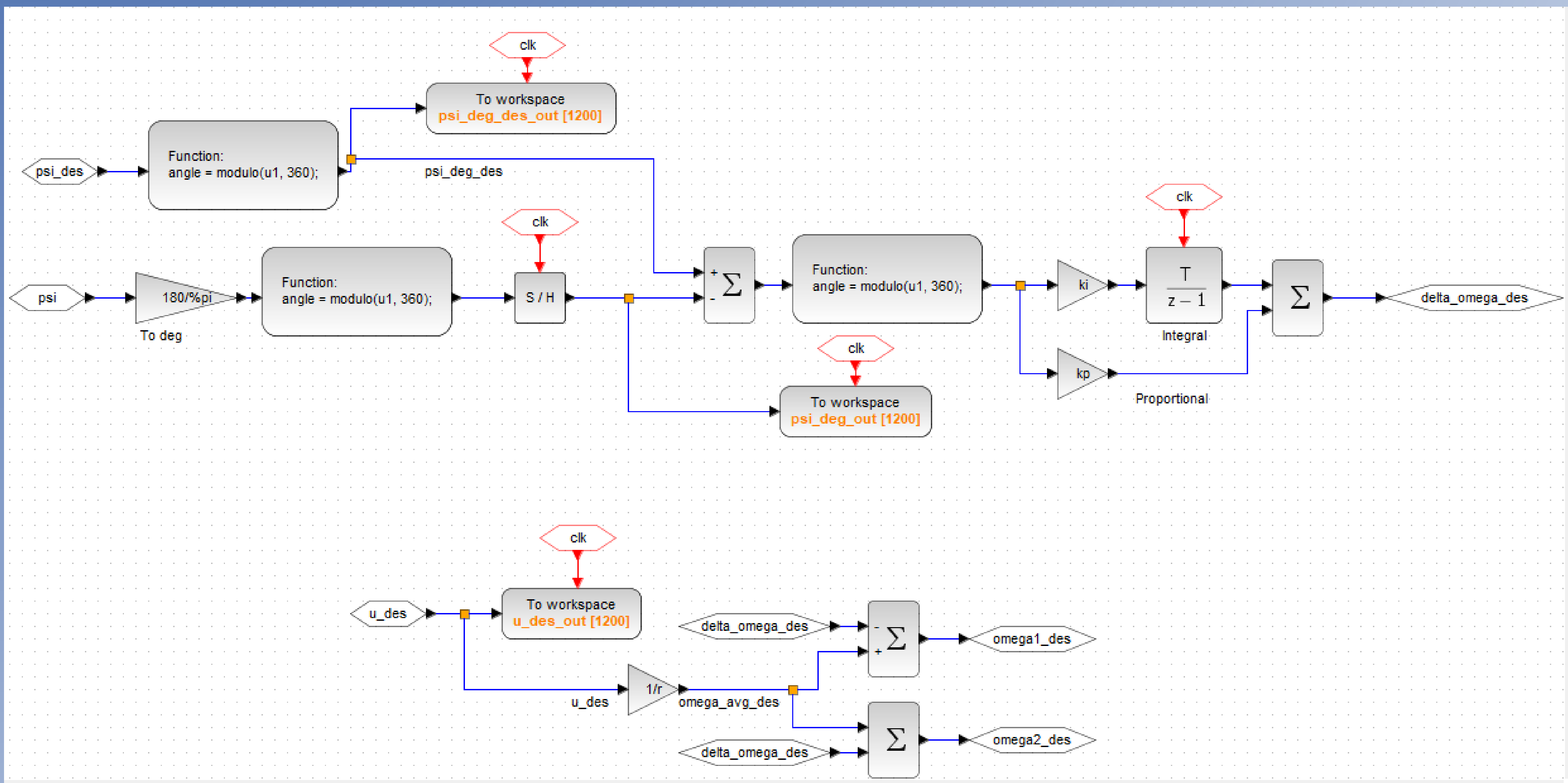
Then we have:

$$\omega_{1_{des}} = \omega_{avg_{des}} - \Delta\omega_{des}$$

$$\omega_{2_{des}} = \omega_{avg_{des}} + \Delta\omega_{des}$$

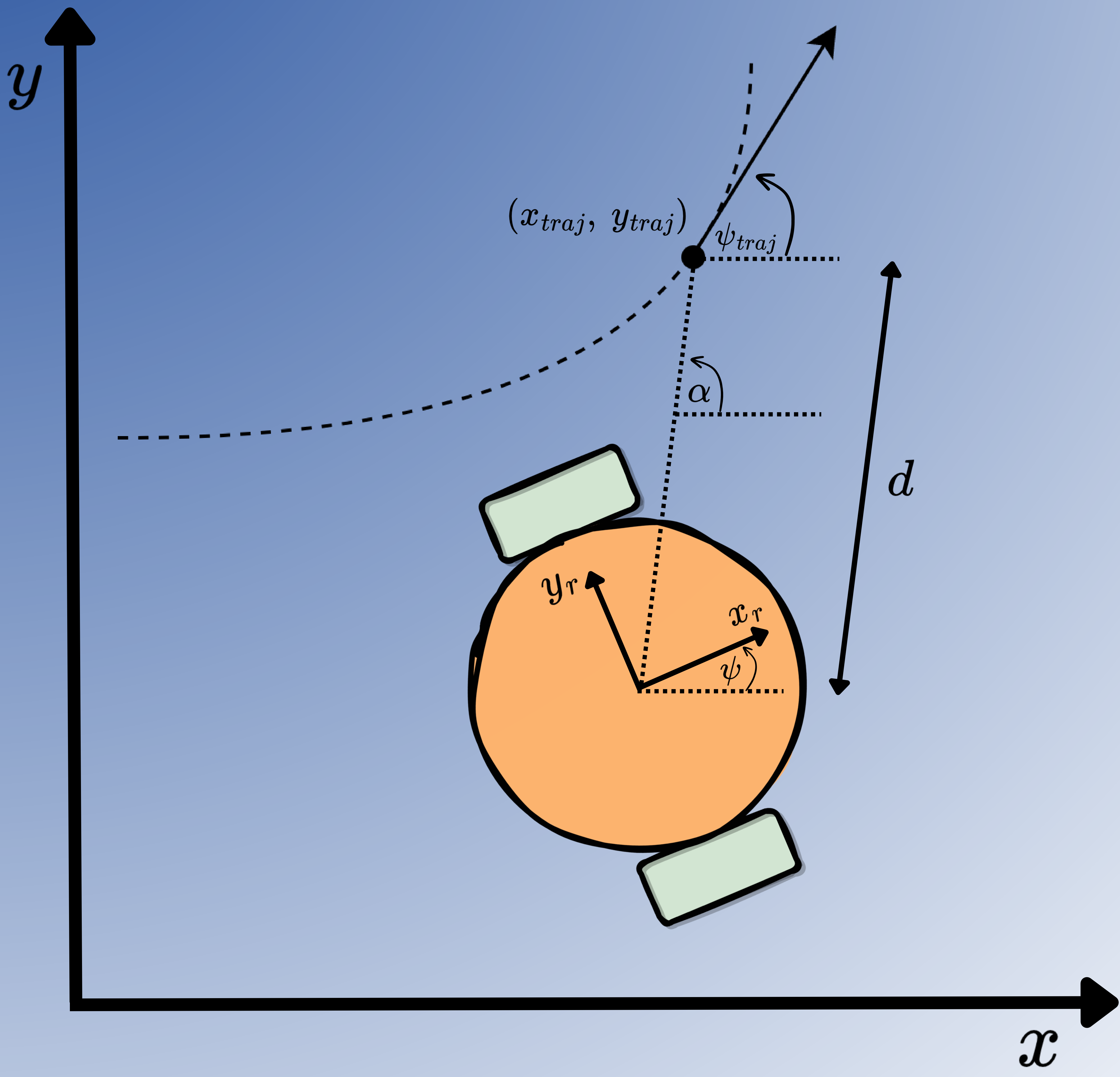
Simone Bertoni

Control system (Xcos)



Simone Bertoni

Trajectory control



Simone Bertoni

Trajectory control – Algorithm – 1

The desired trajectory is defined by the coordinates

$$x_{traj}(t),\ y_{traj}(t)$$

at any point in time. The objective is to keep the robot as close as possible to such coordinates.

Let's find first the heading of the trajectory:

$$\psi_{traj} = \mathrm{atan}(\dot{y}_{traj}, \dot{x}_{traj})\frac{180}{\pi}$$

and its speed:

$$u_{traj} = \sqrt{\dot{y}_{traj}^2 + \dot{x}_{traj}^2}$$

Now let's find the distance between the robot and the point on the trajectory:

$$d = \sqrt{(y_{traj}(t) - y(t))^2 + (x_{traj}(t) - x(t))^2}$$

and the angle between the position of the robot and the position of the point on the trajectory:

$$\alpha = \mathrm{atan}(y_{traj}(t) - y(t), x_{traj}(t) - x(t))\frac{180}{\pi}$$

Trajectory control – Algorithm – 2

Now we need to generate the two setpoints for the control system:

$\psi_{deg_{des}}$: yaw angle in degrees
u_{des}: linear speed in m/s

and we need to do it so that the robot will follow the point on the trajectory.

Regarding $\psi_{deg_{des}}$, the idea is:

- If the robot is far from the point, we aim to the point and request $\psi_{deg_{des}} = \alpha$.
- If the robot is on the point, we request the trajectory heading: $\psi_{deg_{des}} = \psi_{traj}$
- If the robot is in between, we interpolate between α and ψ_{traj}

Mathematically, we achieve this with the following algorithm:

$$\psi_{deg_{des}} = (1 - a)\psi_{traj} + a\alpha$$

where $a = sat(k_{dst}d, 0, 1), \;\; k_{dst} = 0.2$

$k_{dst} = 0.2$ means that for $d > 5\ m\ a = 1$ (if the distance is greater than 5 meter we aim at the point on the trajectory).

Simone Bertoni

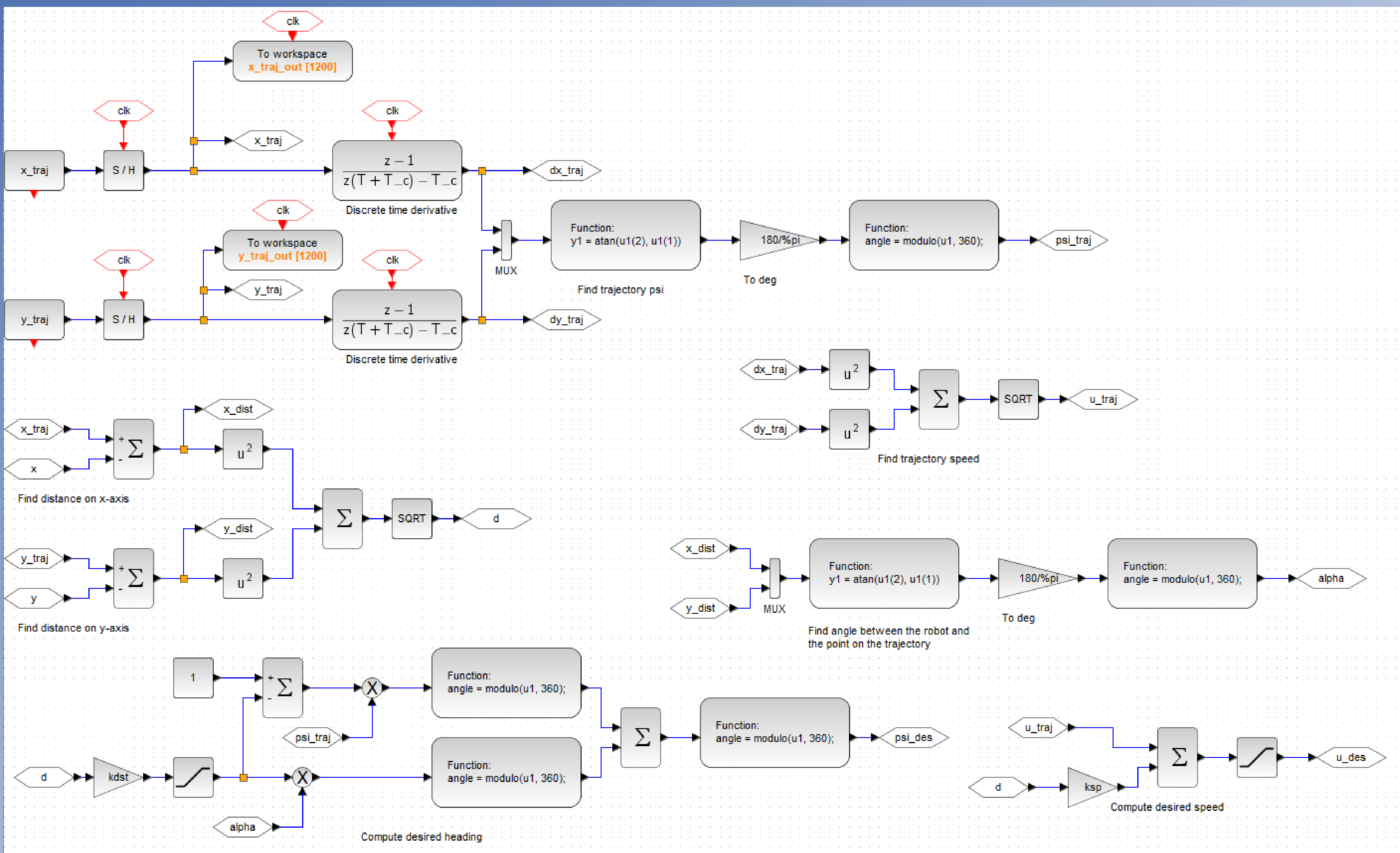
Trajectory control – Algorithm – 3

Regarding the speed demand, we use:

$$u_{des} = sat(k_{sp}d + u_{traj}, 0, u_{max}), \quad u_{max} = 3\ \frac{m}{s}$$

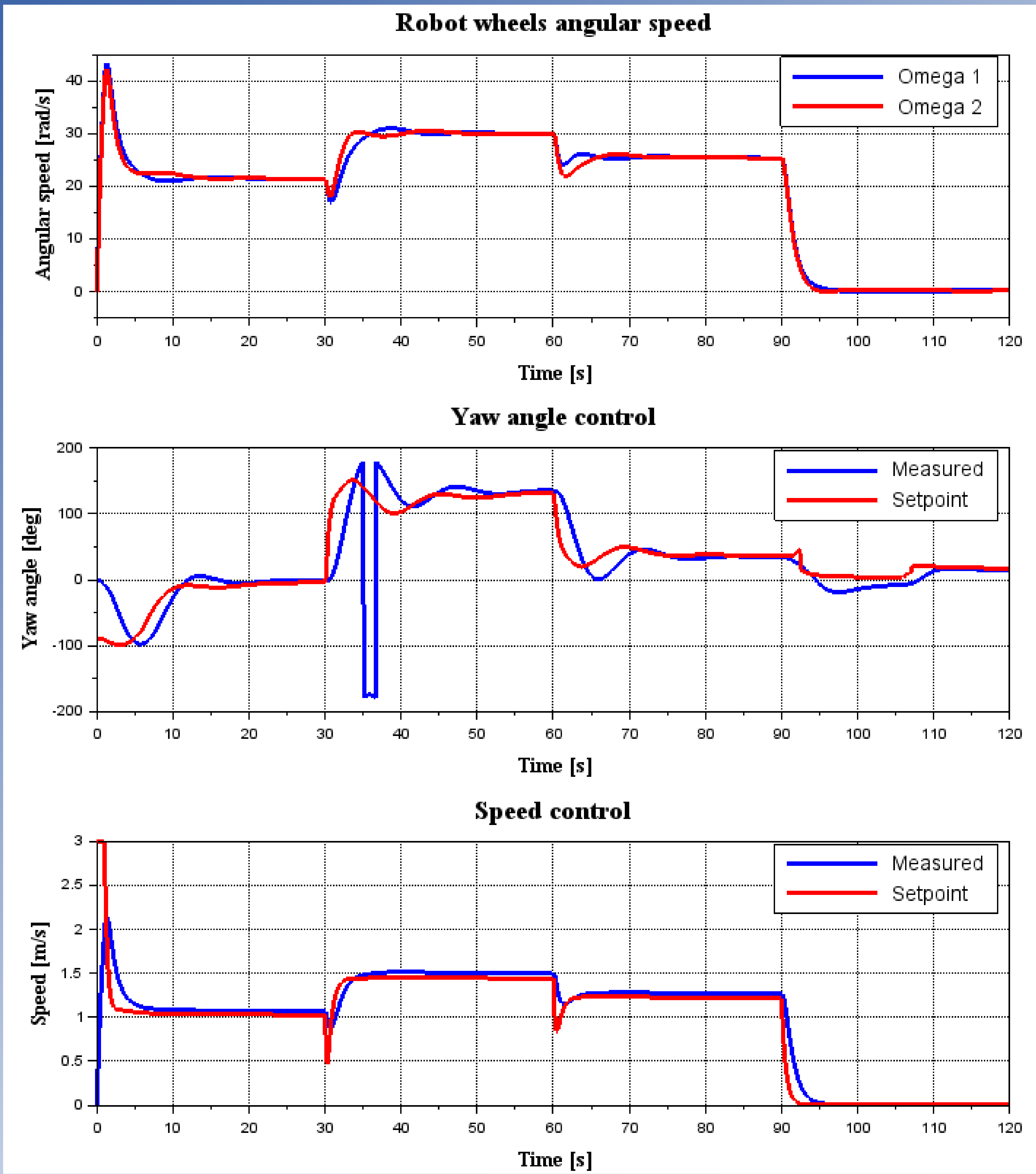
with this approach we "catch-up" with the trajectory by going faster when we are far from the point.

Trajectory control (Xcos)



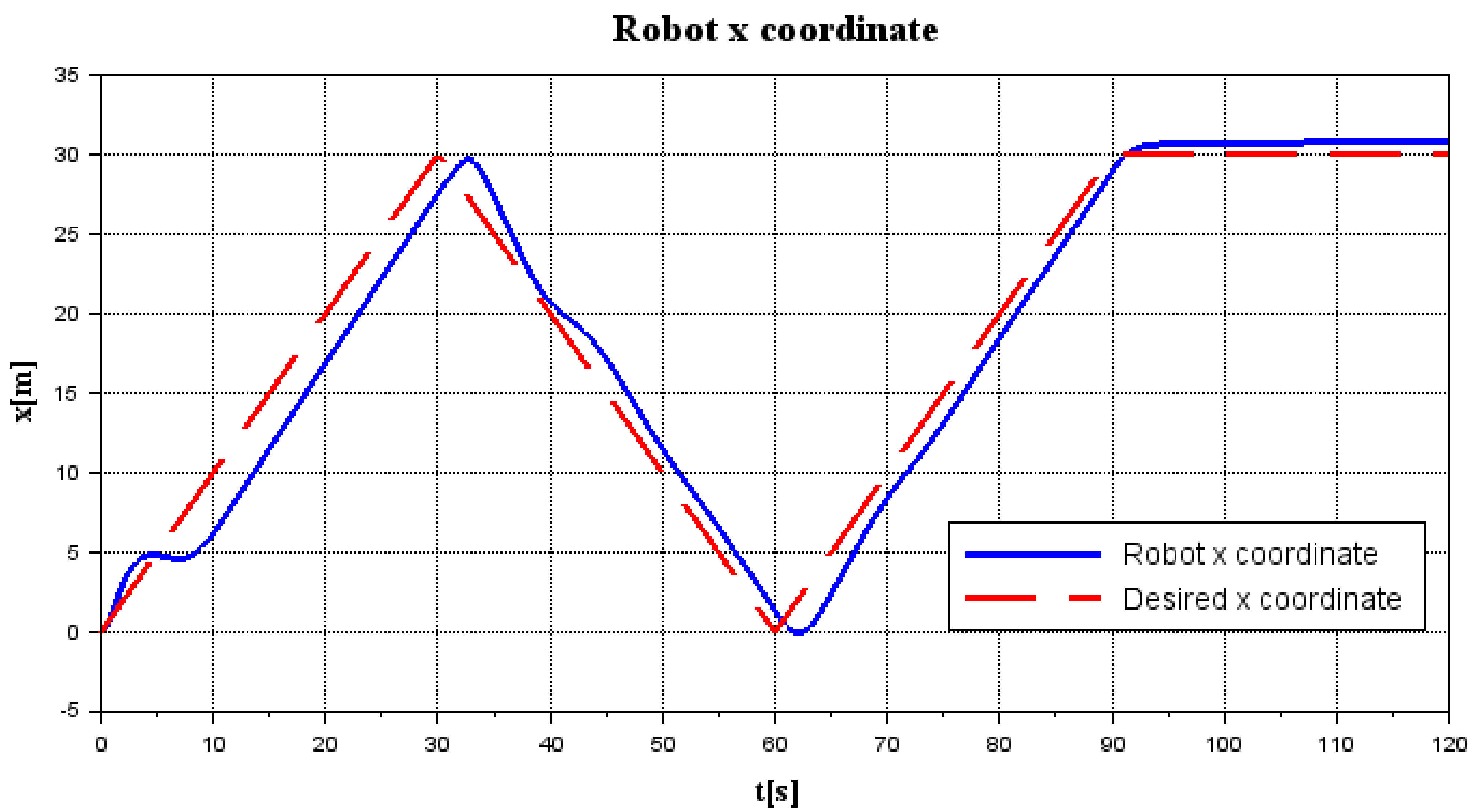
Simone Bertoni

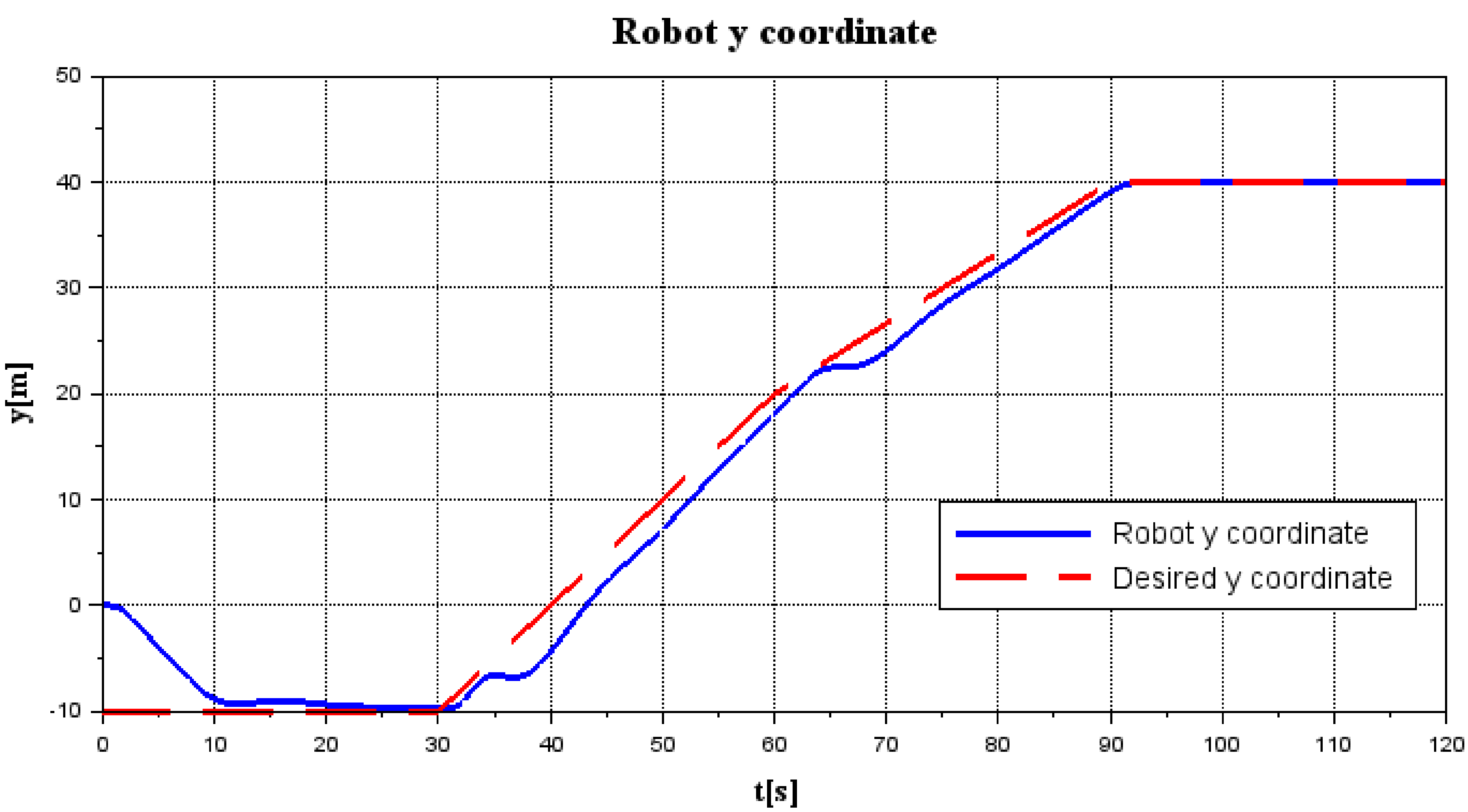
Simulation – Control system



Simone Bertoni

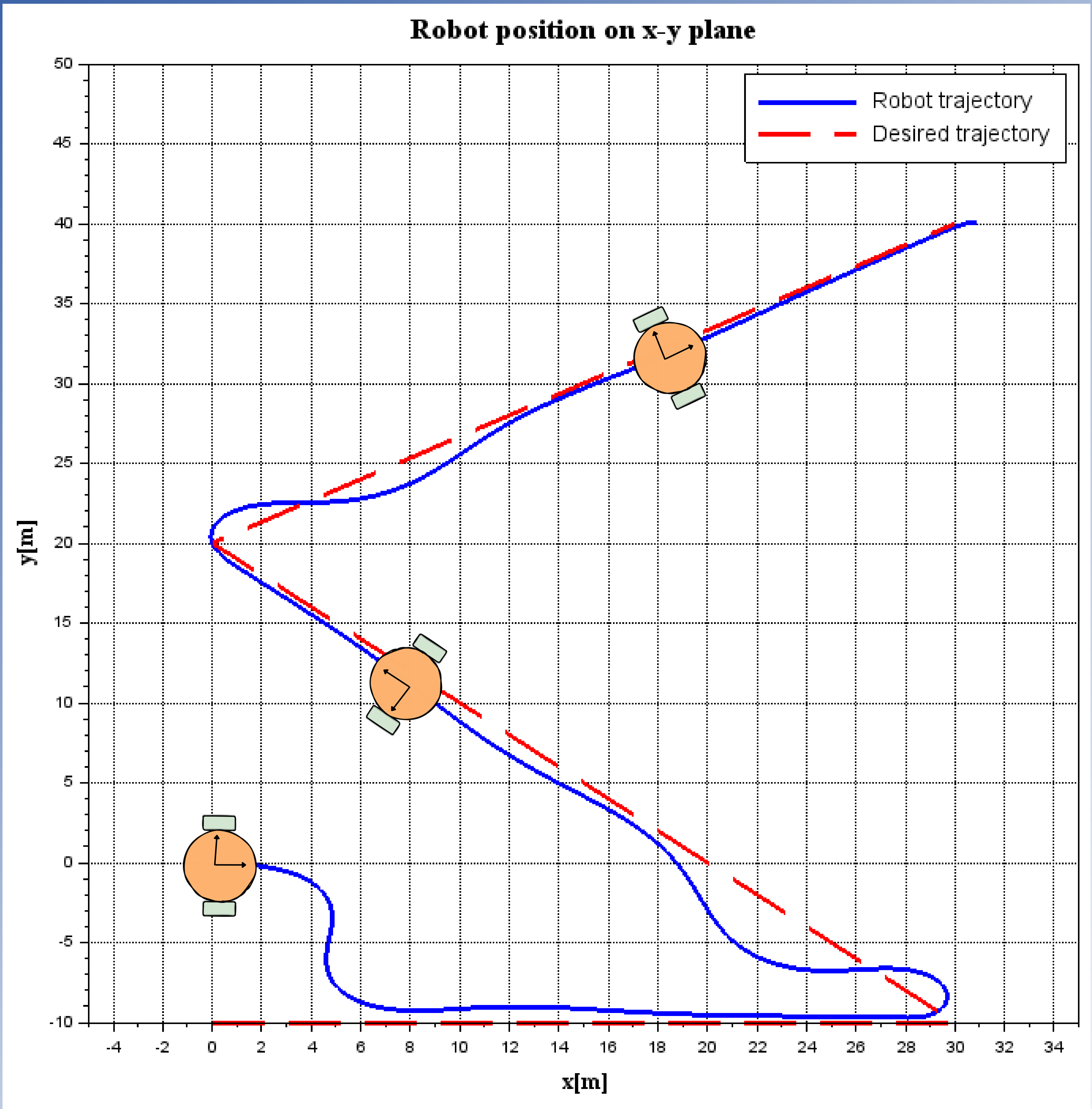
Simulation – Coordinates





Simone Bertoni

Simulation – Trajectory control



Simone Bertoni