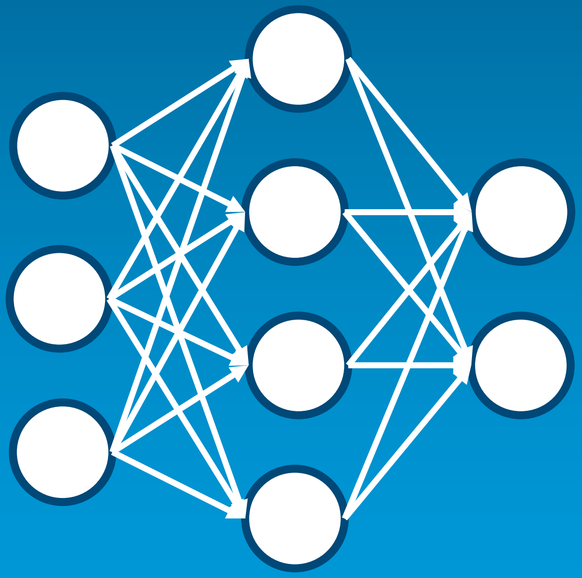
Reinforcement Learning Applied to Autonomous Vehicles



GUYOT Antonin

Content

- Context
  - QCAR presentation
  - Reinforcement Learning introduction

- Strategy and results
  - Bicycle model control
  - Simulation with MATLAB/Simulink
  - HiL integration

- Conclusion

QCAR presentation

- Developped by Quanser
- MATLAB Toolbox
- Control with WiFi
- LIDAR, 360 Vision, RGBD Camera, Accelerometers

Reinforcement Learning (RL) introduction

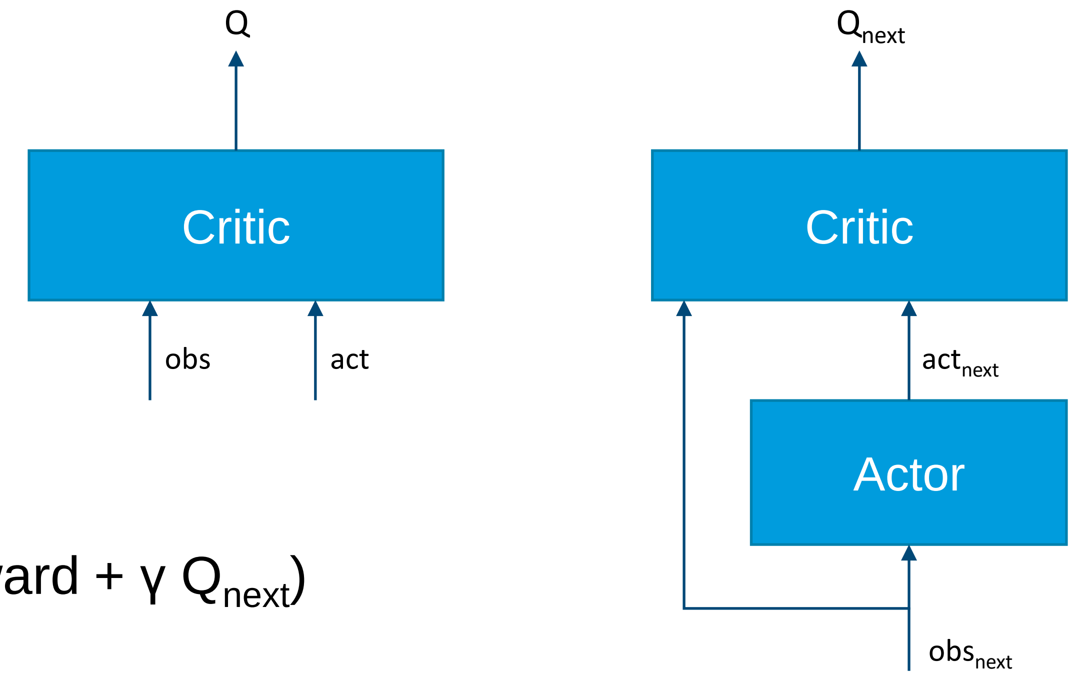
- RL => an agent tries to learn a policy by trial and error

- Setup :
  - Observations
  - Actions
  - Reward

- DDPG => Policy Iteration (actor/critic architecture)
  - Actor applies the policy
  - Critic looks at the associated reward to tune the policy

Reinforcement Learning (RL) introduction

- Actor-critic network



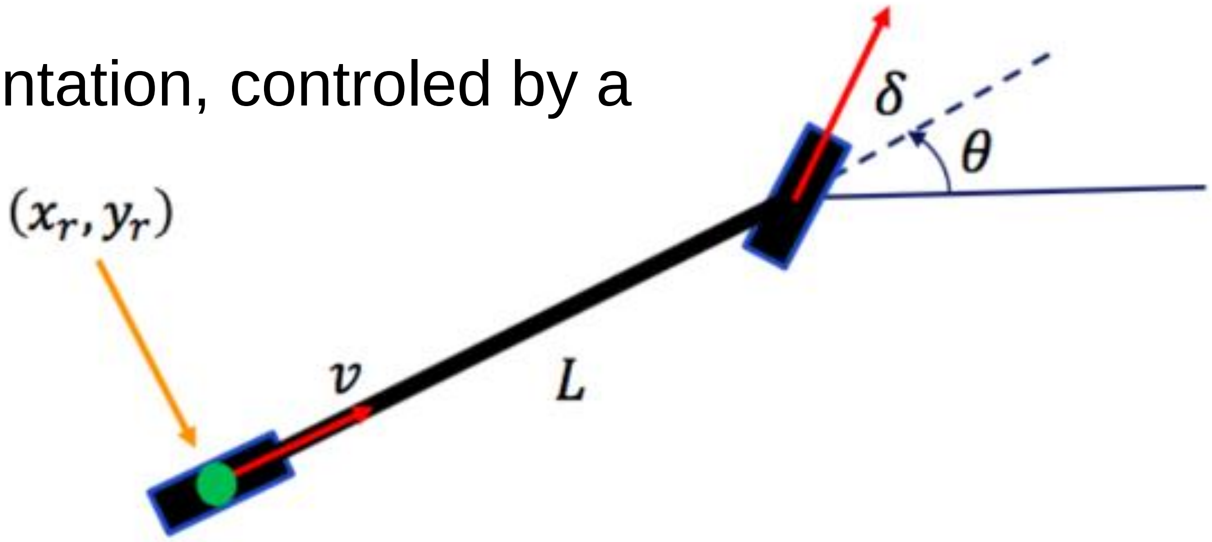
- Q-value
  - $Q = reward + \gamma \cdot Q_{next}$
  - Train actor : Maximize Q
  - Train critic : Minimize Q – (reward + γ Q_{next})

- Replay Buffer

$$[< obs, act, reward, obs_{next} >]$$

Bicycle model control

- Simplified representation of a 4 wheels vehicle
- 2 wheels, one steering wheel
- Represented by a position and an orientation, controled by a velocity and a steering angle
  - $\dot{x}_r = v \cdot \cos(\theta)$
  - $\dot{y}_r = v \cdot \sin(\theta)$
  - $\dot{\theta} = \frac{v}{L} \cdot \tan(\gamma)$



Bicycle model control

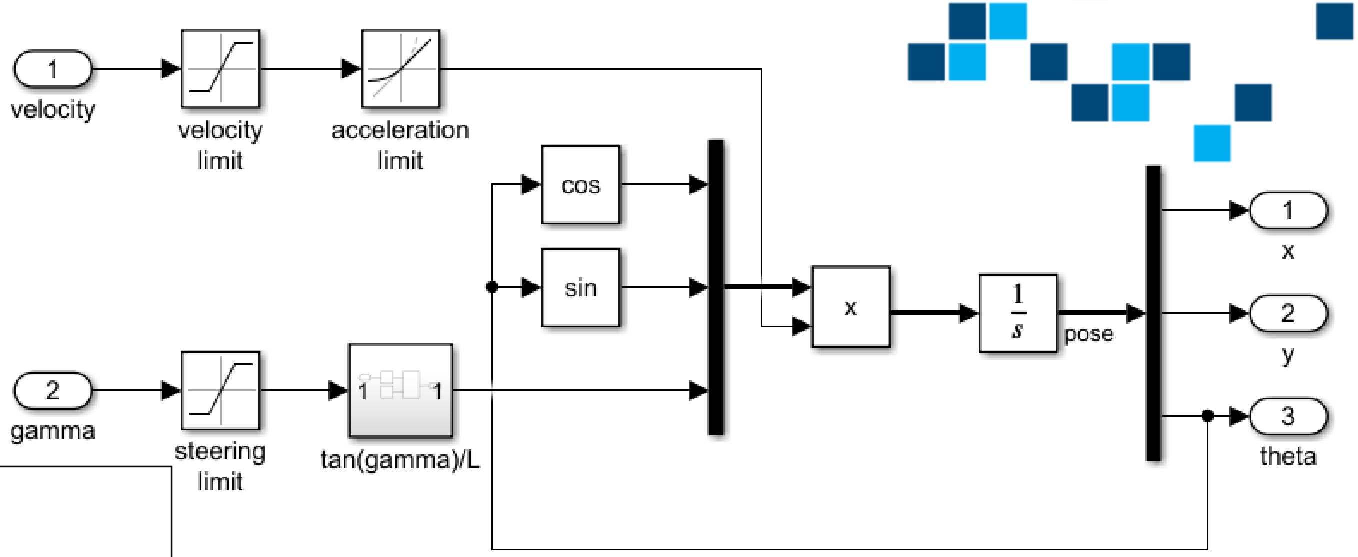
- To get started => work of Peter Corke

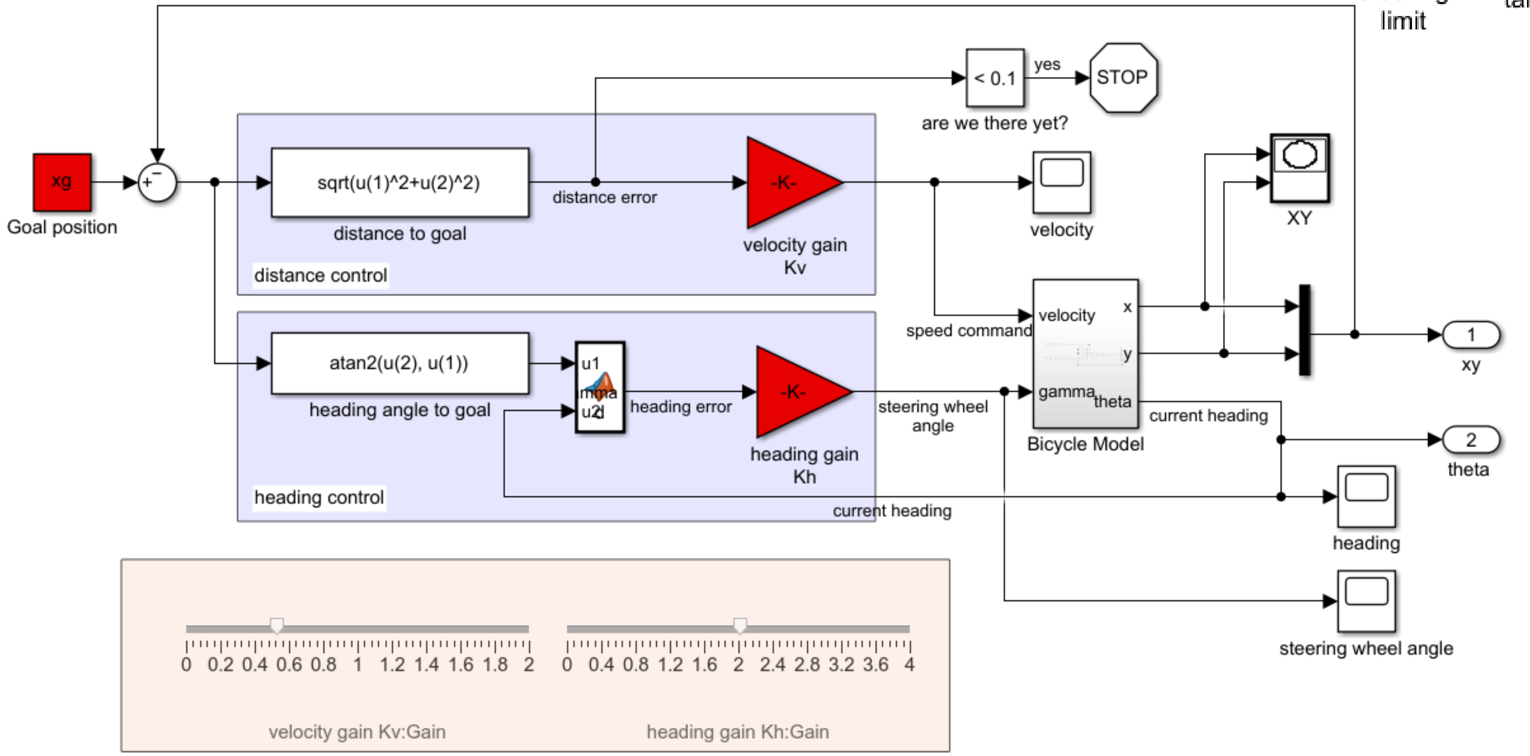
- Point to point control
  - Goal of coordinates (x_g, y_g)

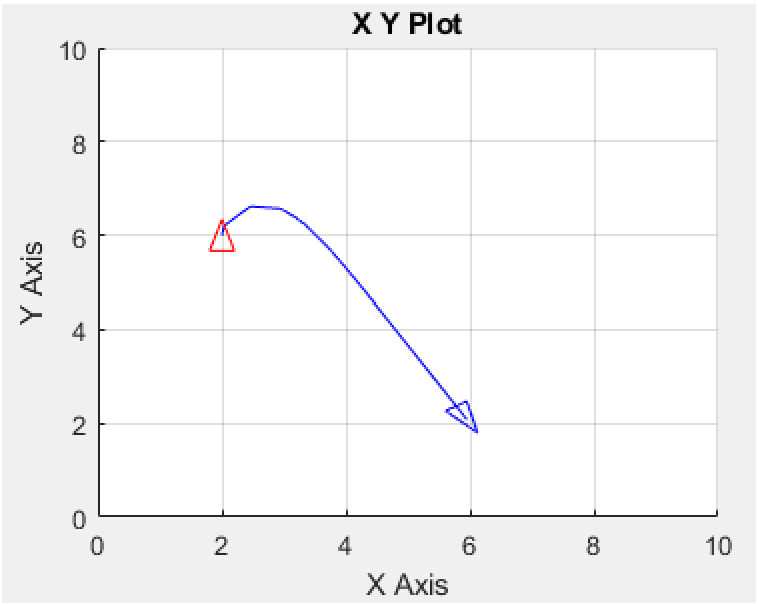
  - $v = K_v \cdot \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2}$
  - $\gamma = K_h \cdot (tan^{-1}\left(\frac{y_g - y_r}{x_g - x_r}\right) - \theta)$

Bicycle model control

- Simulink Model

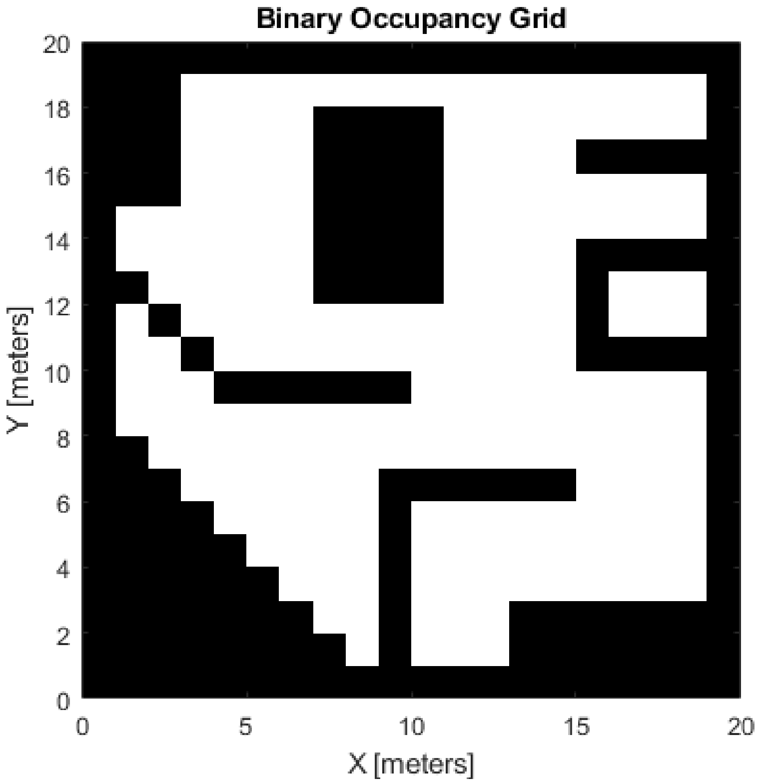


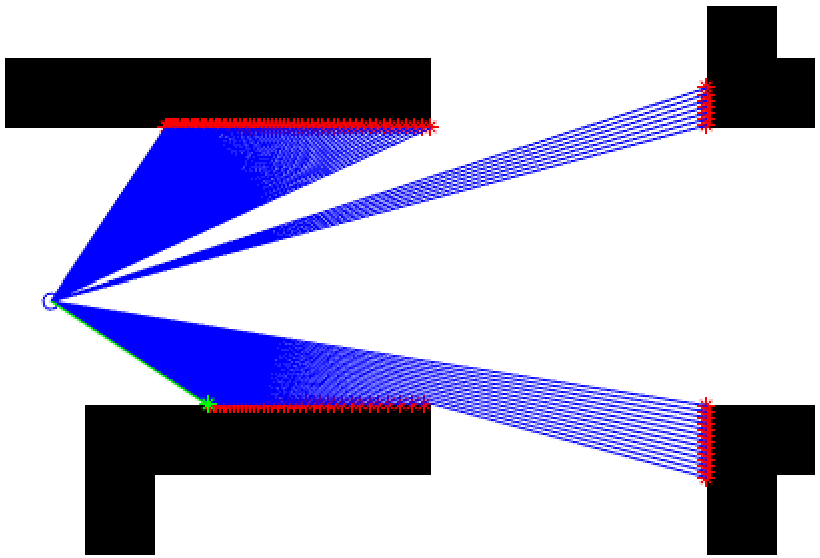




Simulation with MATLAB Simulink

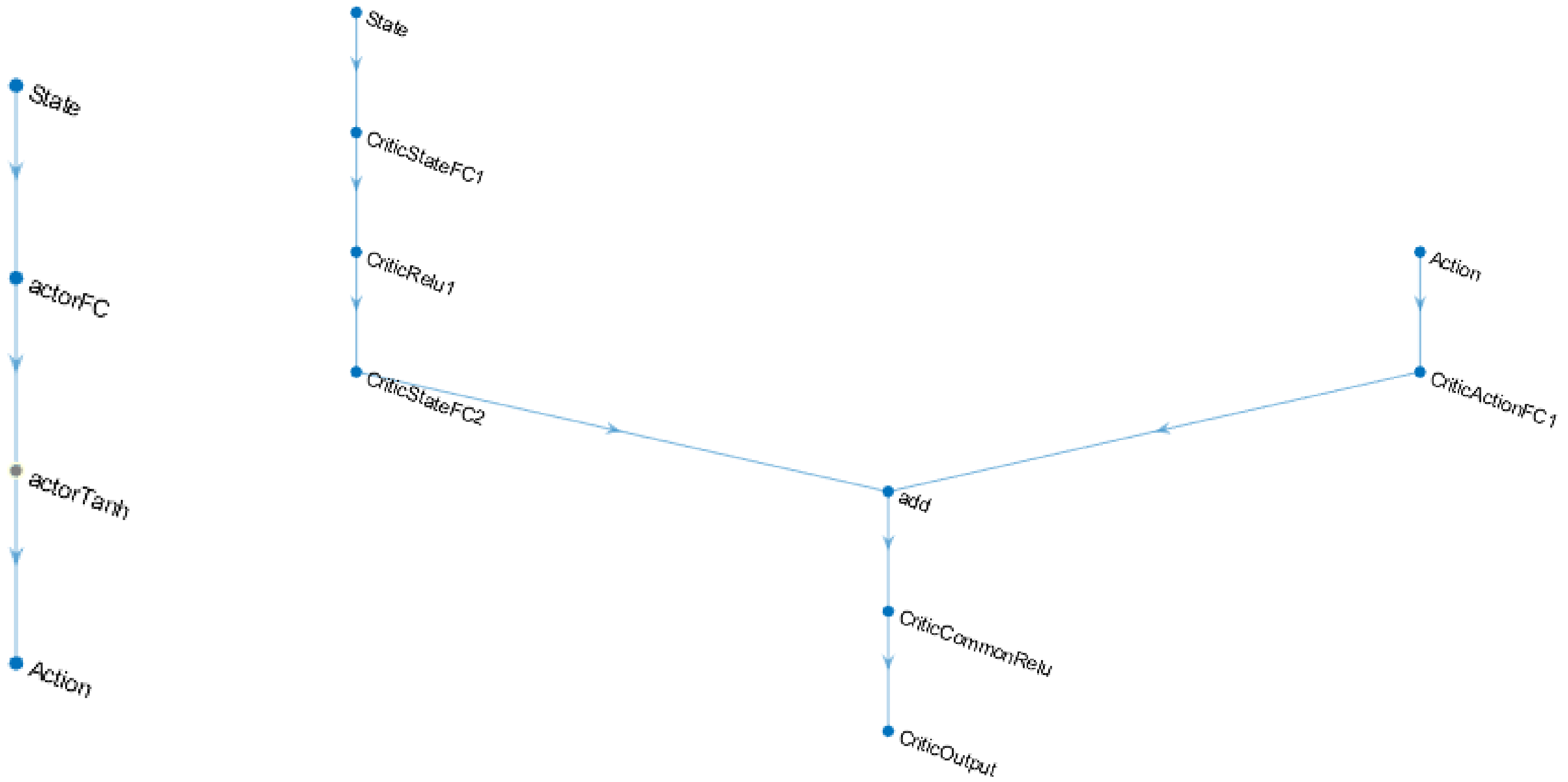
- Modelisation of the components
  - Environment
  - LIDAR

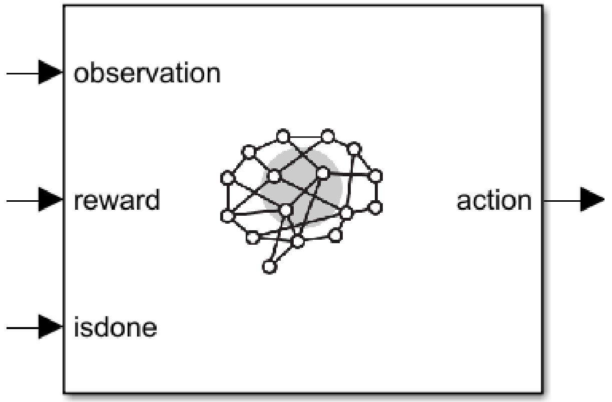




Simulation with MATLAB Simulink

- RL Toolbox

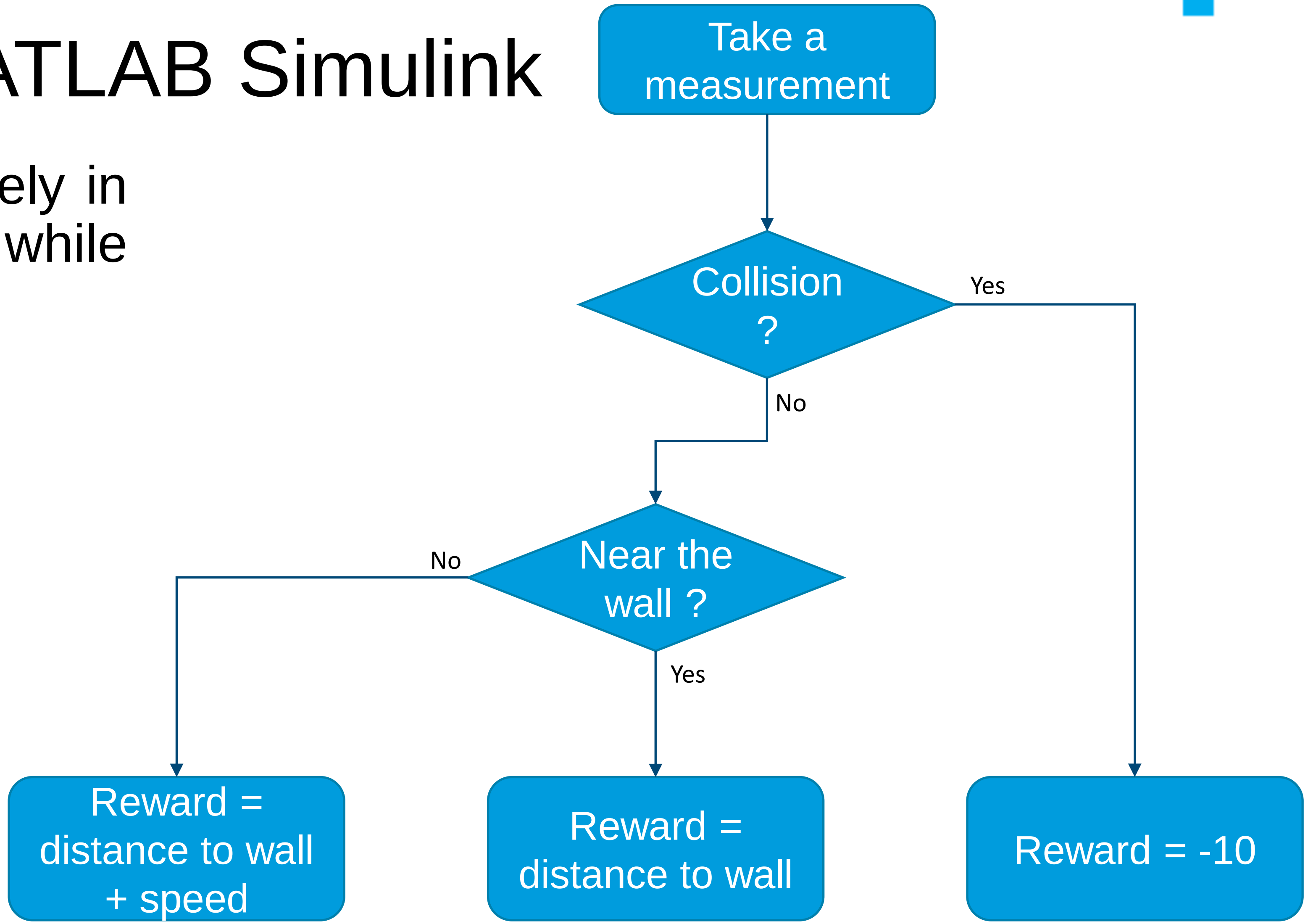




Simulation with MATLAB Simulink

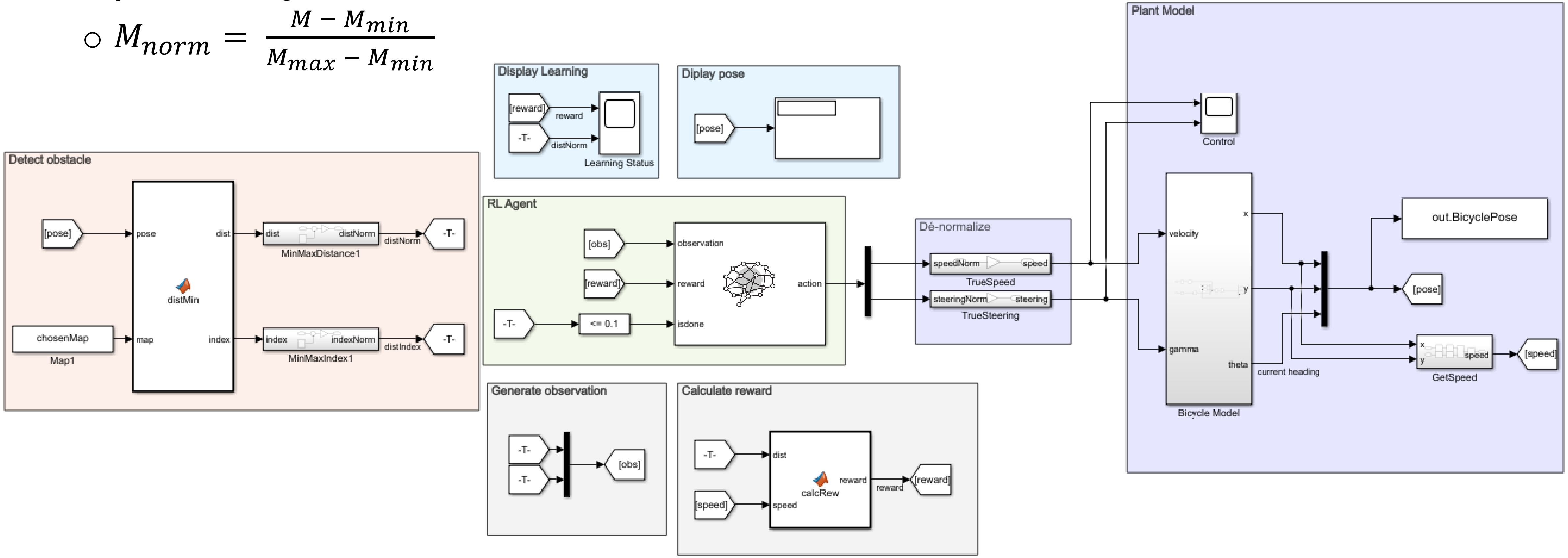
First experiment : move freely in a space with obstacles while avoiding them

- Observations
- Actions
    - Velocity control
    - Steering control
- Reward



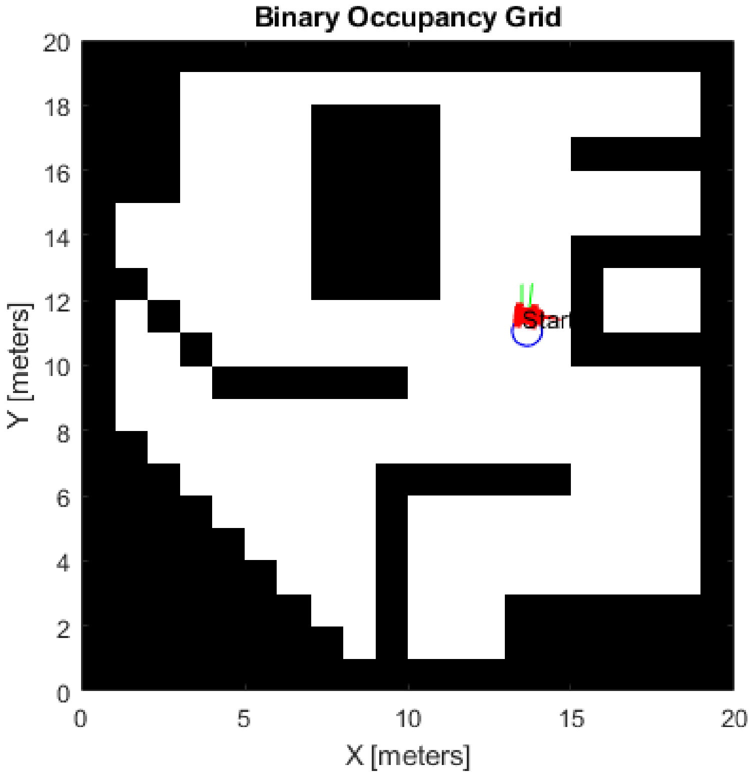
Simulation with MATLAB Simulink – Obstacle

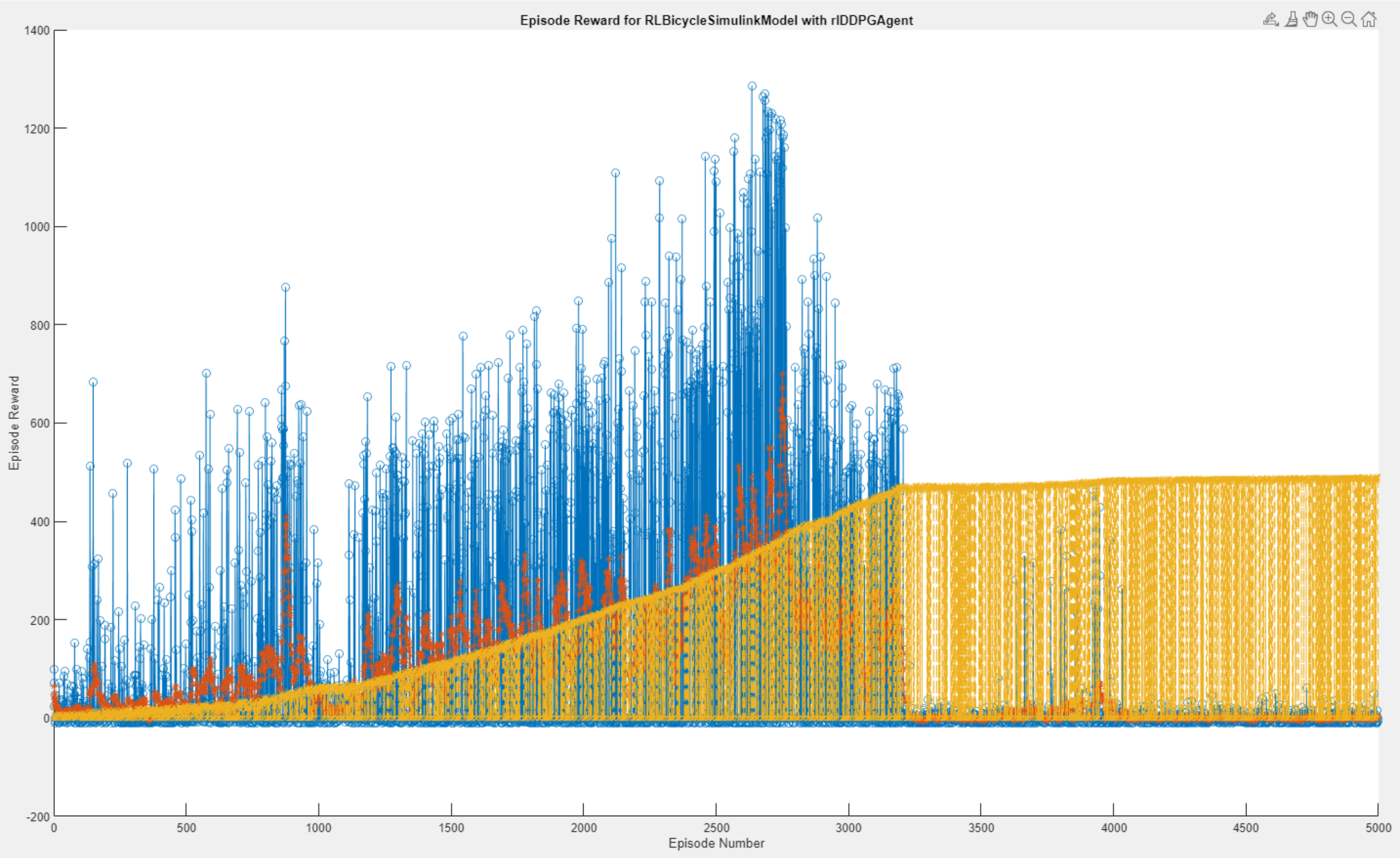
- Help convergence : MinMax normalization
  - $M_{norm} = \dfrac{M - M_{min}}{M_{max} - M_{min}}$



Simulation with MATLAB Simulink – Obstacle

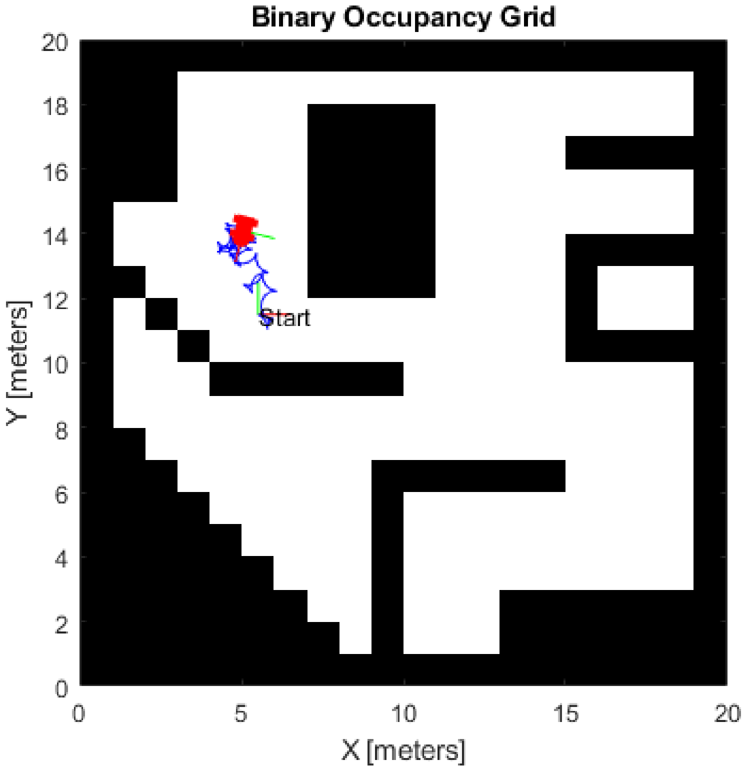
- First result : no good policy

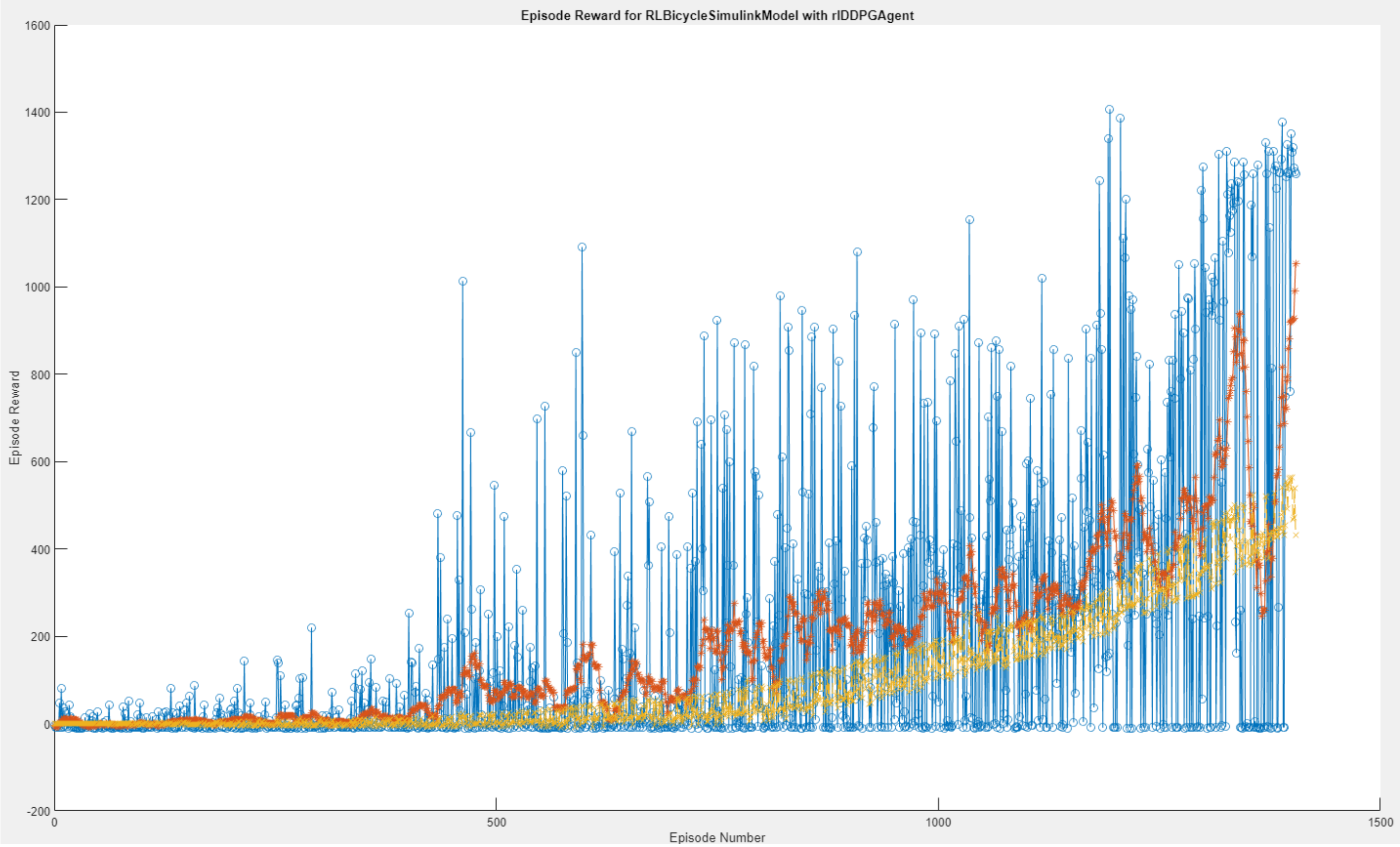




Simulation with MATLAB Simulink – Obstacle

- Second result : convergence





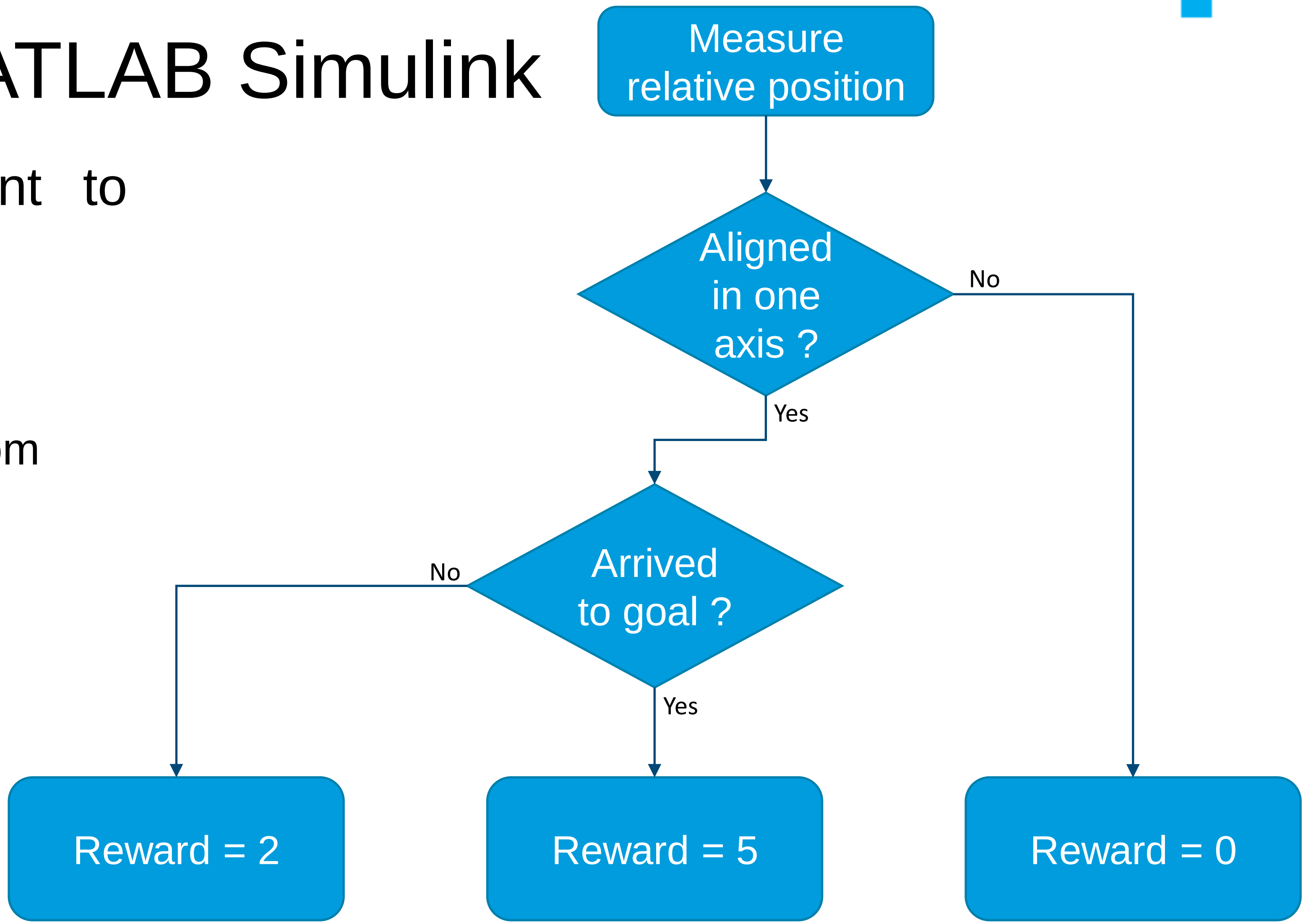
Simulation with MATLAB Simulink – Obstacle

- Improvements
  - Let it train for longer
  - Improve reward function
  - Better control of velocity (backtracking)

Simulation with MATLAB Simulink

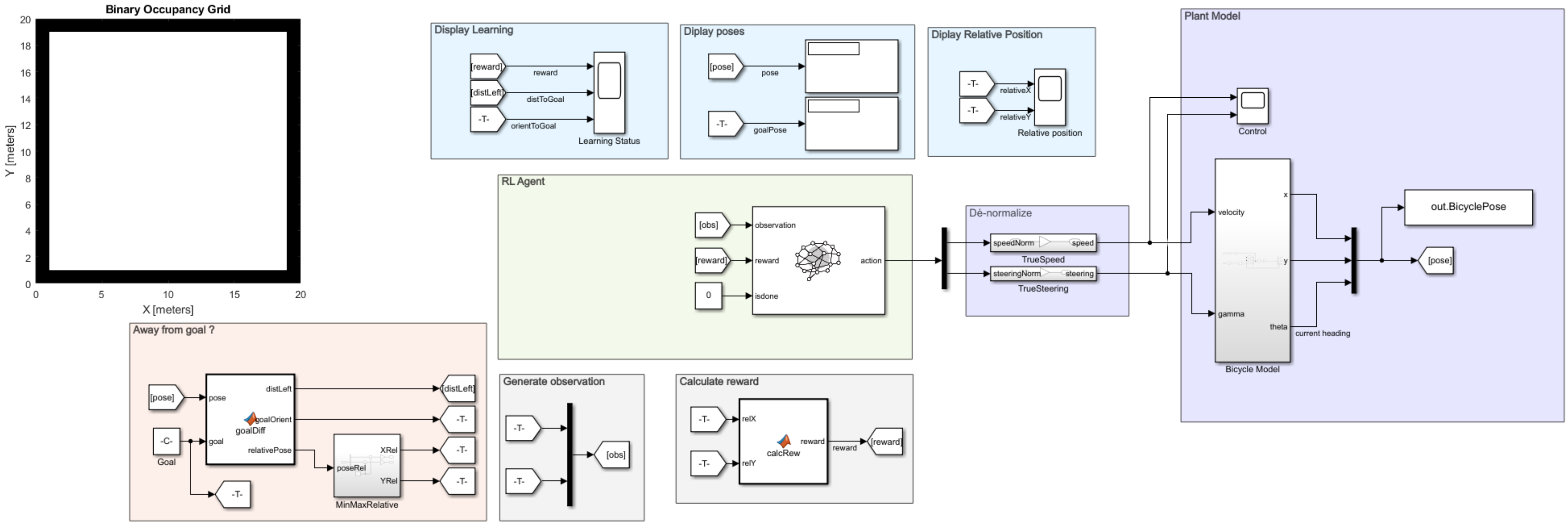
Second experiment : Point to point drive

- Observations
  - Distance and orientation from goal
  - Relative position from goal
- Actions
- Reward



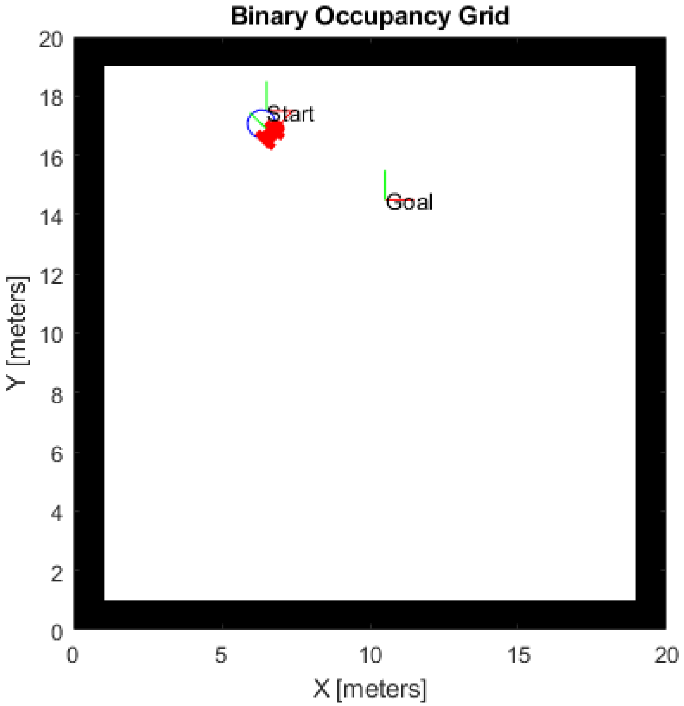
Simulation with MATLAB Simulink – Point to Point

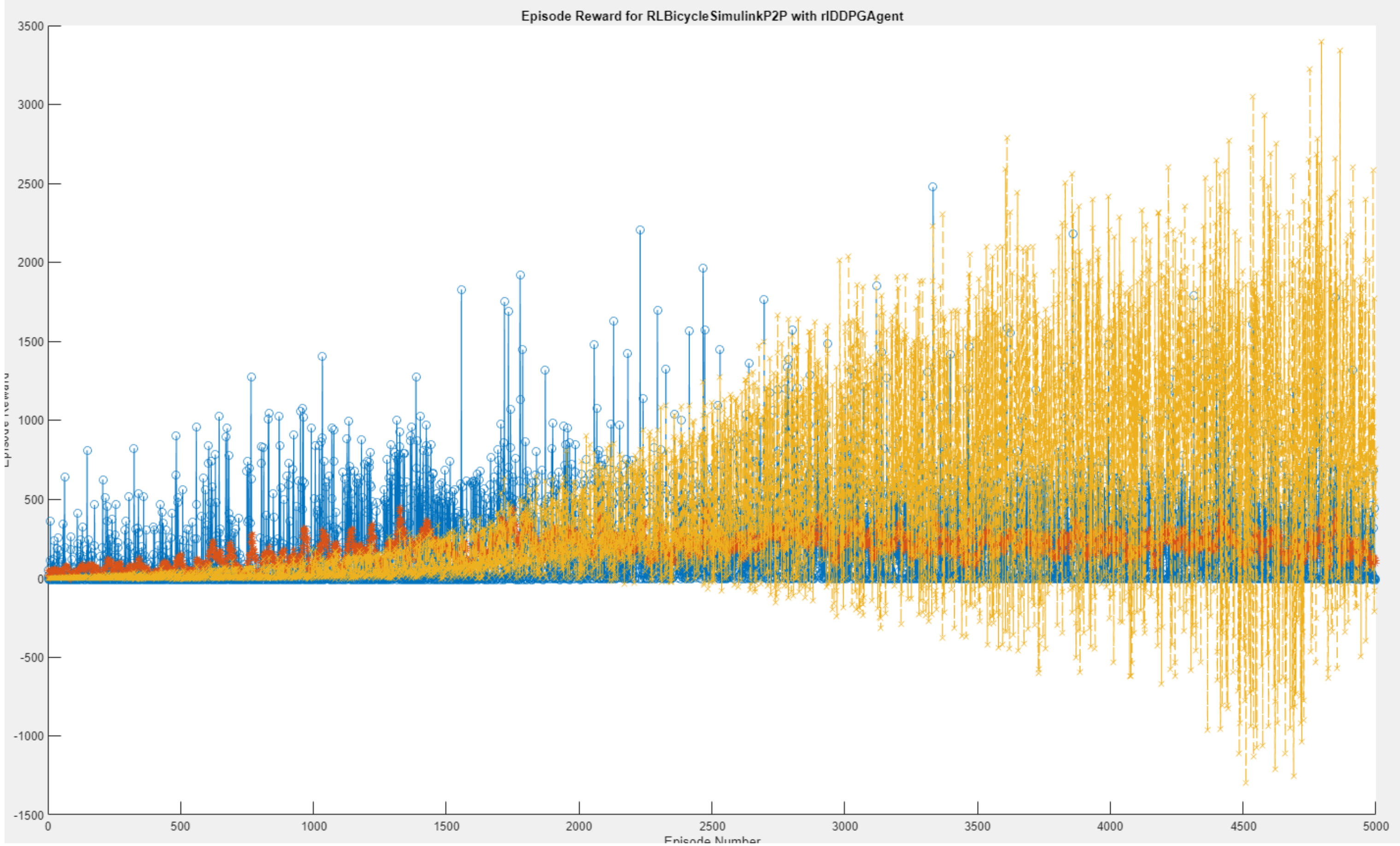
- Simplified environment



Simulation with MATLAB Simulink – Point to Point

- Results : no good policy

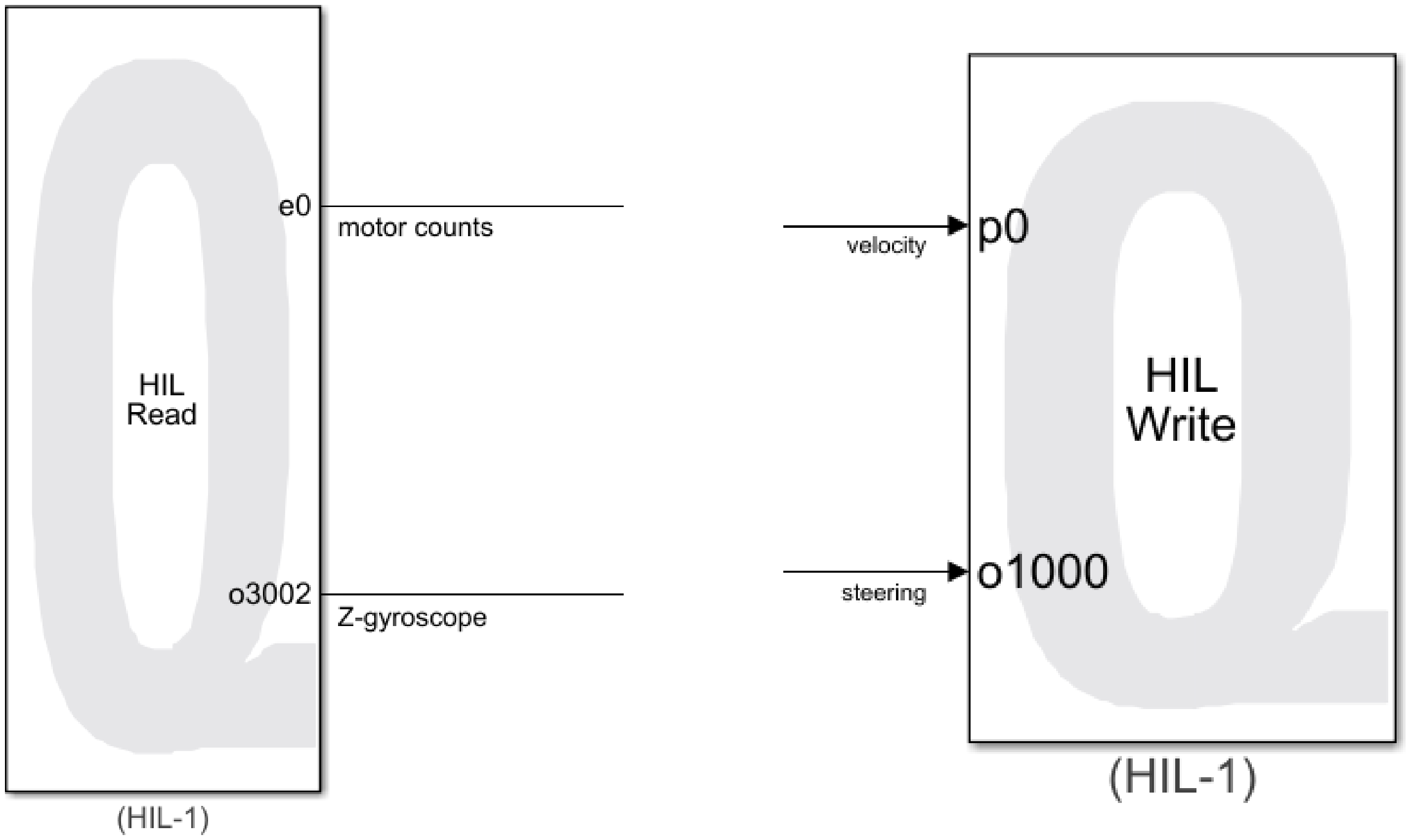




HiL integration

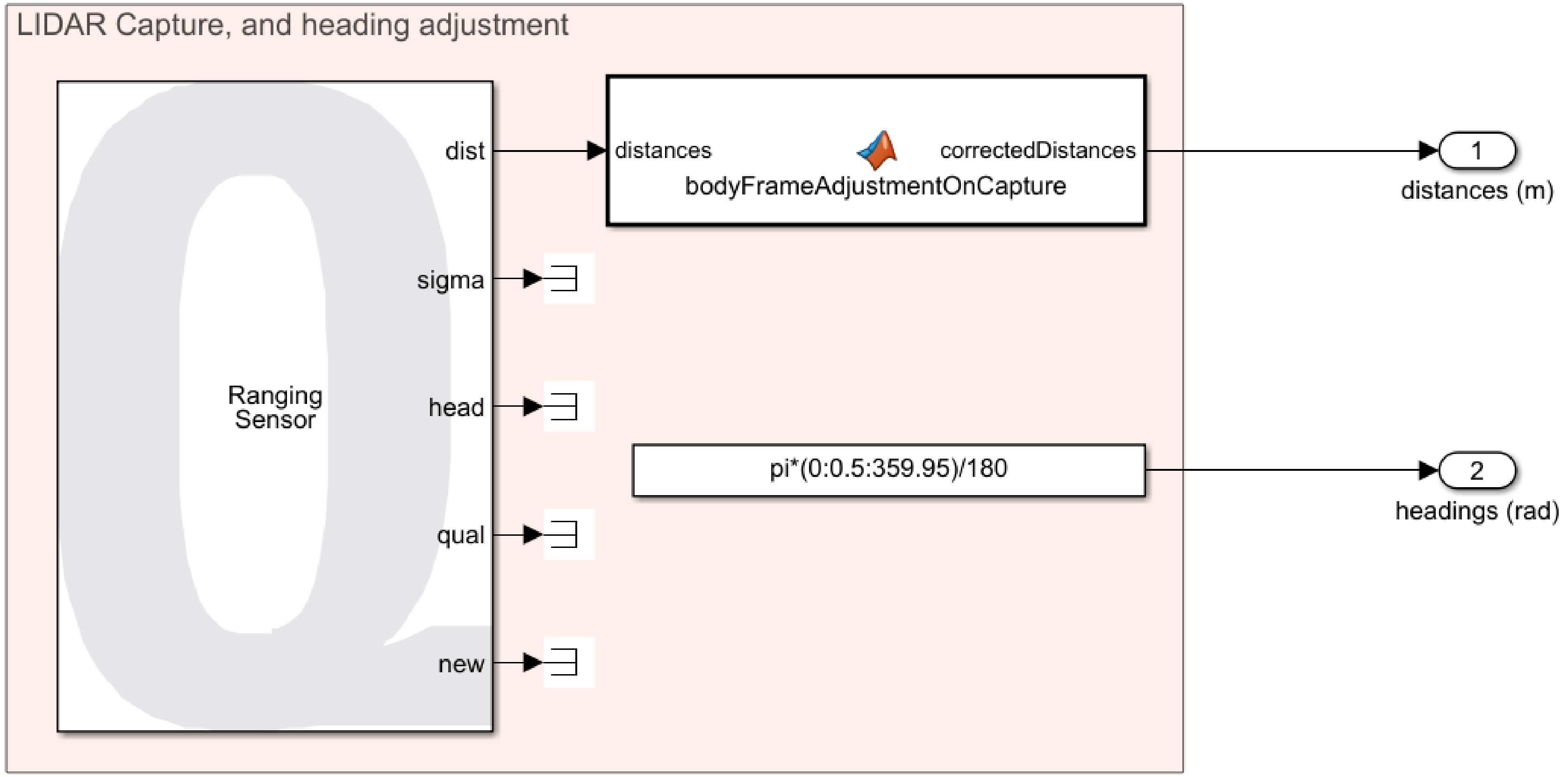
- Use of Quanser Toolbox

- Velocity and steering control



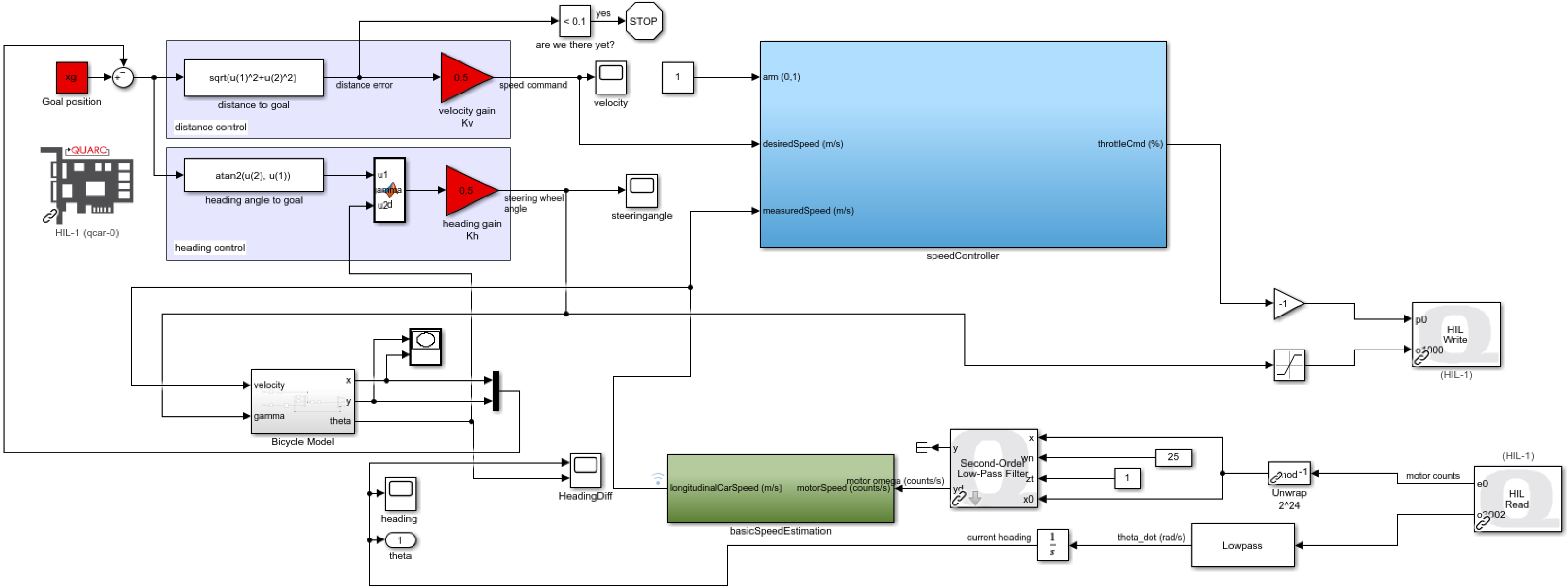
HiL integration

- LIDAR informations acquisition



HiL integration

- Calculations made on bicycle model and control injected to the QCAR



Conclusion

- Discovery of RL algorithms
  - Correlation between simulation and reality
  - Results analysis
  - Difficulty => DDPG complete understanding (black box)
  - Next step => fully integrate HiL and try more difficult tasks

- Appropriation of a simplified but complex model

- Hang of the QCAR and embedded system (and problems associated)

Thank you for listening !
Any questions ?