Control of Differential Drive Robots

Cyber Physical Systems Project

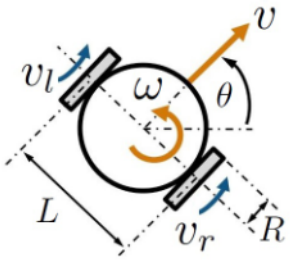
Maria Grazia Berni

Problem Statement

The aim of this project is to develop a controller for a differential drive robot, using a PID (Proportional Integral Derivative). The robot has 3 different behavior : reaching a goal, avoiding an obstacle and reaching a goal while avoiding an obstacle. The PID tuning will be performed via Reinforcement Learning and, to deal with process and observation noise, to the controller will be associated a kalman filter. The desired properties of the controller will be expressed using STL formulas and checked. Also falsification will be performed.

Model Plant

The system to control is a 2-wheeled differential drive robot moving in a $2d$ grid. The robot considered in this project has wheel radius $R = 0.0325m$ and distance between the two wheels $L = 0.1m$.



It has 3 degrees of freedom (x,y,ϕ) and the navigation can be controlled specifying the velocities of the right and left wheel (v_l, v_r). These velocities are computed by a transformation from the linear and angular velocity of the robot unicycle (v, w). The dynamics of the differential drive are defined as:

$$\frac{dx}{dt} = \frac{R}{2}(v_r + v_l)\cos\phi$$
$$\frac{dy}{dt} = \frac{R}{2}(v_r + v_l)\sin\phi$$
$$\frac{d\phi}{dt} = \frac{R}{L}(v_r - v_l)$$

The following relations hold:

$$\omega = \frac{R}{L}(v_r - v_l)$$
$$\text{v} = \frac{R}{2}(v_r + v_l)$$

In this project the linear velocities is maintened constant, so the only control signal is ω and the discrete equations of the evolution of the system are:

$$\text{x}_{k+1} = x_k + v\cos\phi_k dt$$

$$\text{y}_{k+1} = y_k + v\sin\phi_k dt$$

$$\phi_{k+1} = \phi_k + \omega dt$$

where dt is the time step of the discrete simulation.

PID controller

The control strategy of the PID is based on the simple formula:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{d}{dt}e(t)$$

where $e(t)$ is the difference between the desired value of the signal we are controlling (the angular velocity in this case) and the actual value of it .

Tuning the PID with Reinforcement Learning

The PID parameters were found using the $TD3$ advanced policy algorithm. Briefly speaking, this reinforcement learning method use a structure called actor, which is a neural network that decide which actions have to be taken based on the actual state of the system and on a value, computed by another structure called the critic, which is even itself a neural network. In order to tune the PID parameters, the state of the system is the error between the reference signal and the actual signal.

A positive reward is given every time the robot reach a goal position, and a negative reward equal to the squared norm of the error state is given at every step of the simulation : in this way the robot learn how to move in a desired direction. Once the training of the TD3 agent is done, the pid parameters are set as the weights of the actor network, which has only one

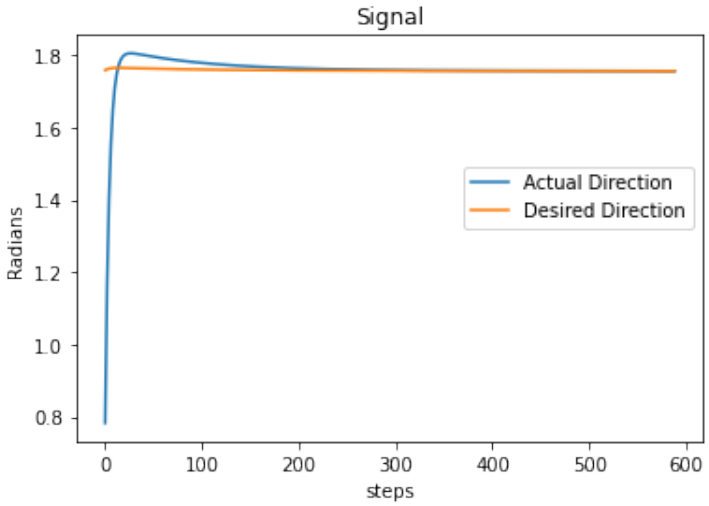
layer and 3 outputs in this special case. This procedure provided the following parameters, but it's worth pointing out that every training procedure can provide different parameters:

$k = 2.14470$

$k_i = 0.02368$

$k_p = 0.00061$

Changing the goal position in the space, the robot is always able to reach it in a minimum number of steps, so the robot is able to go in a desired direction. Also the performances in terms of overshoot, rise time, settling time and steady state error are really good, as shown in the following figures.



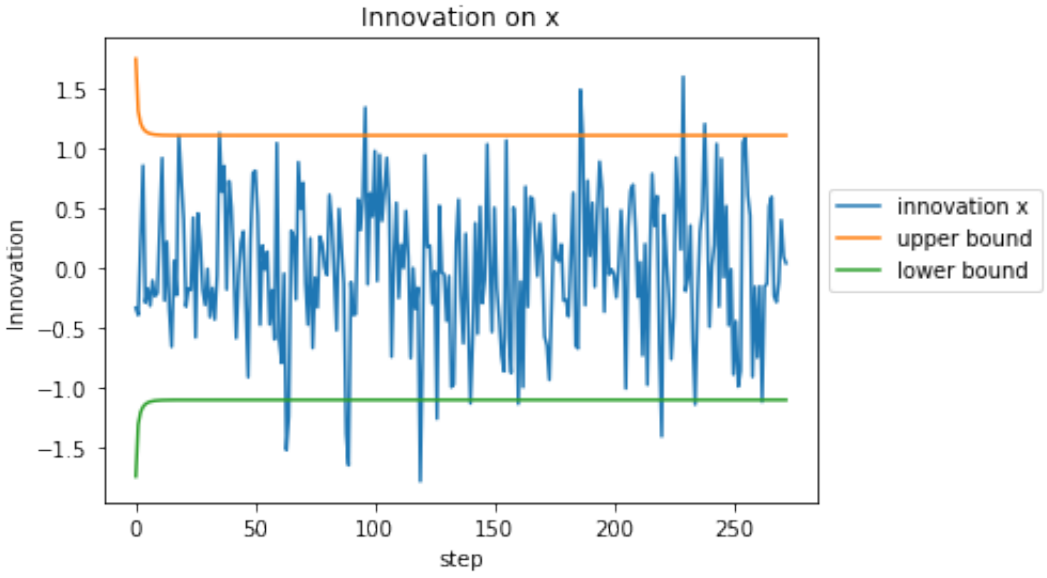
overshoot (rad)	0.0408
rise time (s)	1.4000
settling time (s)	16.4
steady state error (rad)	0.0001

Given the simplicity of the problem, the tuning strategy used was based only on reaching the goal on the minimum steps, without taking in consideration other properties in the reward shaping. When dealing with more complex problem or with strict requirement, it can be useful to include also

other performances, like overshoot or the steady state error etc, in the rewards.

Extended Kalman Filter

To simulate a real sensor the observations of position where combined with random zero mean observation noise, and also to simulate a real process random zero process noise were added to the simulations. To deal with noise the system use an Extended Kalman Filter. Being the system evolution highly non linear, the equations have been linearized at each time step. The filter is consistent if the innovation has zero mean, it's uncorrelated and it's consistent with its covariance, so 95% of the measures will be bounded by $+2\sqrt{S_k}$ and $-2\sqrt{S_k}$. Visually inspecting the behavior of innovation it can be seen that these requirements are satisfied. Here is shown the trend of the component x of the innovation. The other compontents have a similar behavior.



Requirements

In this section we formalize some properties that our system should satisfy in terms of Signal Temporal Logic. In this scenario we will consider the following situation:

- The robot must reach a goal position.

- The robot must reach a goal position avoiding an obstacle

Requirements:

- The quadratic distance of the robot from the obstacle must never be less or equal to 0.3 during all the simulation, while the robot is trying to reach the goal. Here is not important if the robot finally reach the goal. The simulation lasts 1200 steps, so $120s$. Given the structure of the problem, I decided to formalize the property in terms of the coordinate of the robot and of the obstacle, instead of giving a scalar signal such as the distance, so that the monitor will have a complete state representation of the system. In STL:

  $$\Phi_1 = G_{[0,120]}(((x_r - x_o)^2 + (y_r - y_o)^2) > 0.3)$$

  with (x_o, y_o) coordinates of the obstacle and (x_r, y_r) coordinates of the robot.

- In absence of the obstacle, the robot should reach the goal (so their distance will be less then 0.3)within the first $80s$ of simulation, and remain there for the rest of the simulation:

  $$\Phi_2 = G_{[80,120]}(((x_r - x_g)^2 + (y_r - y_g)^2) < 0.3)$$

  with (x_g, y_g) coordinates of the goal and (x_r, y_r) coordinates of the robot.

- In the presence of an obstacle the robot should reach the goal in the first $100s$ of the simulation, and remain there for the rest of the simulation.

  $$\Phi_3 = G_{[100,120]}(((x_r - x_g)^2 + (y_r - y_g)^2) < 0.3)$$

These properties have been verified for a model where noise is present but there are no initiatives to deal with it (I'v called this model "only noise"), and for a model with a kalman filter, considering random position of the robot, the goal and the obstacle. The position of the obstacle is not completely random, in fact I make sure it is in the line joining the robot with the goal, but I add to this position some random number to make sure that all possible configuration will be considered.

Falsification

The property Φ_1 is the most important from a safety point of view, so falsification on it has been performed considering both the model with the kalman filter and the "only model" noise. To perform falsification 40 different configurations " robot obstacle goal" has been considered. First, falsification

has been performed only respect to observation noise. Process noise was kept with standard deviation of 0.05 while the standard deviation of the observation noise was varied from 0.1 to 4, adding 0.1 at every epoch (each epoch is made of 40 simulations). The minimum value and the mean value of the robustness during each epoch have been saved. The minimum value is the only important in this scenario. Then falsification was performed varying both the standard deviations of process and observation noises, varying them from 0.1 to 4 for the observation noise, and from 0.01 to 0.4 for the process noise. Results of both validation and falsification are reported in the next section

0.1 Results

Falsification of the first property proved that the "only noise" model , with random process noise of 0.05, is safe when dealing with observation noise with standard deviation less then 0.7, while the property for the model with the kalman filter was falsified at 2.4, so the kalman filter is really robust respect to observation noise. When varying both process and observation noise, the property has been falsified for the model without kalman filter when the process and observation noise reached the standard deviation of 0.07 and 0.7, while the model with the kalman filter has been falsified with standard deviations of 0.12 and 1.2. The robustness of all other property has been calculated considering a process noise of 0.05 and observation noise of 0.5.

Falsification of property 1 : robusteness values for the model with the kalman filter

noise	mean value	min value
0.1	3.815	2.05
0.2	3.839	2.537
0.3	4.078	2.782
0.4	3.911	2.749
0.5	3.855	1.859
0.6	3.912	2.061
0.7	3.978	2.036
0.8	4.054	1.928
0.9	3.7	1.262
1.0	4.136	1.772
1.1	3.845	1.605
1.2	3.822	2.445
1.3	3.447	1.579
1.4	3.718	1.387
1.5	3.72	1.286
1.6	3.869	1.45
1.7	4.03	1.775
1.8	4.038	1.781
1.9	4.295	2.154
2.0	3.966	1.403
2.1	3.814	1.505
2.2	4.031	1.131
2.3	4.039	1.4
2.4	3.723	-0.026
2.5	3.869	-0.178
2.6	4.243	1.427

Falsification of property 1 : robusteness values for the model without the kalman filter

noise	mean value	min value
0.1	4.029	2.423
0.2	3.71	2.232
0.3	3.729	2.59
0.4	3.278	1.665
0.5	3.204	1.834
0.6	3.087	0.594
0.7	2.782	-0.007
0.8	2.723	-0.035
0.9	2.229	0.293
1.0	2.309	0.039
1.1	2.214	0.249
1.2	2.055	-0.022
1.3	1.876	-0.071
1.4	1.636	-0.049
1.5	1.471	-0.055
1.6	1.267	-0.123
1.7	1.322	-0.113
1.8	1.198	-0.153
1.9	1.356	-0.102
2.0	1.024	-0.107
2.1	1.178	-0.142
2.2	0.681	-0.14
2.3	0.868	-0.094
2.4	0.693	-0.119
2.5	0.64	-0.138
2.6	0.662	-0.063

Falsification of property 1 :
robusteness values for the model
with the kalman filter

ob_noise	pr_noise	mean value	min_value
0.1	0.01	4.302	3.632
0.2	0.02	4.139	2.968
0.3	0.03	3.763	2.423
0.4	0.04	3.453	1.952
0.5	0.05	3.218	1.433
0.6	0.06	3.035	1.55
0.7	0.07	2.56	0.253
0.8	0.08	2.565	0.349
0.9	0.09	2.517	-0.032
1.0	0.1	1.796	-0.094
1.1	0.11	1.652	-0.075
1.2	0.12	1.936	-0.166

Falsification of property 1 :
robusteness values for the model
without the kalman filter

ob_noise	pr_noise	mean value	min_value
0.1	0.01	4.302	3.632
0.2	0.02	4.139	2.968
0.3	0.03	3.763	2.423
0.4	0.04	3.453	1.952
0.5	0.05	3.218	1.433
0.6	0.06	3.035	1.55
0.7	0.07	2.56	0.253
0.8	0.08	2.565	0.349
0.9	0.09	2.517	-0.032
1.0	0.1	1.796	-0.094
1.1	0.11	1.652	-0.075
1.2	0.12	1.936	-0.166

The robustness values of property 2 and 3 are reported here :

Robustness values for property 2 and 3
Observation noise = 0.5
Process noise = 0.05
Simulations using kalman filter

Property 2	Property 3
[0.025,	[0.032,
0.043,	0.018,
0.003,	0.042,
0.11,	0.003,
0.066,	0.015,
0.085,	0.034,
0.098,	0.011,
0.004,	0.071,
0.065,	0.119,
0.152,	0.041,
0.081,	0.032,
0.103,	0.006,
0.141,	0.056,
0.078,	0.038,
0.124,	0.014,
0.054,	0.097,
0.029,	0.102,
0.038,	0.008,
0.005,	0.008,
0.056,	0.006,
0.037,	0.067,
0.051,	0.029,
0.05,	0.051,
0.035,	0.134,
0.1,	0.054,
0.027,	0.049,
0.137,	0.062,
-1.138,	0.003,
0.035,	0.058,
0.007,	0.095,
0.096,	0.077,
0.01,	0.066,
0.127,	0.004,
0.093,	0.061,
0.059,	0.025,
0.047,	0.063,
0.038,	0.044,
0.005,	0.045,
0.038,	0.054,
0.015]	0.087]