Exercises for Lecture Course on Modeling and System Identification (MSI)
Albert-Ludwigs-Universität Freiburg – Winter Term 2017

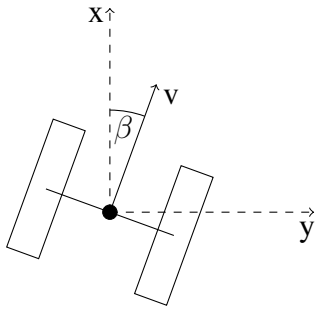
Exercise 8: Recursive Least Squares

Prof. Dr. Moritz Diehl, Robin Verschueren, Alexander Resch

In this exercise you will implement a Recursive Least Squares (RLS) estimator and a forward simulation of a robot.
For the the MATLAB exercises, create a MATLAB script called `main.m` with your code, possibly calling other functions/scripts. From running this script, all the necessary results and plots should be clearly visible.

Computer Exercise

We will apply the Recursive Least Squares (RLS) algorithm to position data of a 2-DOF moving in the $X - Y$ plane, measured with a sampling time of $0.0159\,\mathrm{s}$. The movement of the robot depends on the angular velocities of the left and the right wheel ω_L and ω_R, as well as on their radii R_L and R_R. Differing radii influence the behaviour of the robot.



The system can be described by a state space model with three internal states. The state vector $\mathbf{x} = [x\ y\ \beta]^\top$ contains the position of the robot in the $X-Y$ plane and the deviation β from its initial orientation. The system can be controlled by the angular velocities of the wheels: $\mathbf{u} = [\omega_L\ \omega_R]^\top$. The output of the system is the position of the robot: $\mathbf{y} = [x\ y]^\top$. The model follows as

$$\dot{\mathbf{x}} = \begin{pmatrix} v \cdot \cos\beta \\ v \cdot \sin\beta \\ \frac{\omega_L R_L - \omega_R R_R}{L} \end{pmatrix} \qquad \mathbf{y} = \begin{pmatrix} x \\ y \end{pmatrix} \tag{1}$$

with L being the length of the axis between the two wheels and the velocity v being

$$v = \frac{\omega_\mathrm{L} \cdot R_\mathrm{L} + \omega_\mathrm{R} \cdot R_\mathrm{R}}{2}.$$

1. **Recursive Least Squares applied to position data** (3 points)

It is your task to implement the RLS algorithm in MATLAB and to tune it with the appropriate "forgetting factors". As you can see, the model for the position of the robot is nonlinear. To keep it simple, in task (1) we approximate the position data by a fourth order polynomial. The position data for this exercise can be downloaded from the course website. You can assume that the noise on the X and Y measurements is independent. The experiment starts at $t = 0\,\mathrm{s}$.

(a) Fit a 4-th order polynomial through the data using ordinary Least Squares *Hint: you need one estimator for each coordinate*. Plot the data and the fit both in the $X - Y$ plane and separately. Does the fit seem reasonable? Why do you think that is? (0.5 points)

(b) Implement the RLS algorithm as described in the script to estimate 4-th order polynomials to fit the data. Do not use forgetting factors yet. Plot the result against the data. Compare the LS estimator from `a)` with the RLS estimator after processing N measurements. Please give an explanation for your observations. (1 points)

(c) Add forgetting factors to your algorithm. Tune them to obtain 1) a smoother curve and 2) a less smooth curve. Did you arrive to the same results for both forgetting factors? Plot the results on the same plot as the previous question. (0.5 point)

(d) Plot the "one step ahead prediction" at each point (i.e. extrapolate your polynomial fit to the next time step), along with the $1 - \sigma$ confidence ellipsoid around this point, and the data. First, think about how to compute the covariance Σ_p on the position, if you know the covariance of the estimator Σ_θ. Do the confidence ellipsoids grow bigger or smaller as you take more measurements? *Hint: use the fact that for a random variable $\gamma = A\theta$, where A is a matrix,* $\mathrm{cov}(\gamma) = A\mathrm{cov}(\theta)A^T$. (1 points)

2. **Forward simulation of a robot's position** (2 points)

In this task you will simulate the position of the two-wheel-robot using the state space model.

(a) Given the state space model and the system state $x = [x\ y\ \beta]^\top$, implement a function `[xdot] = robot_ode(t,x,u,param)` which evaluates the right-hand side of the ODE $\dot{x} = f(x, u, param)$, with $param = [R_L, R_R, L]$. Use the following parameters: R_L = 0.2 m, R_R = 0.2 m and L = 0.6 m. (0.5 point)

(b) Implement a function `[x_next] = euler_step(deltaT,x0,u,@ode,param)` which performs one integration step for a general ODE $\dot{x} = f(x, u, param)$ starting at x_0, with input u, parameters $param$ and an integration interval ΔT. (0.5 point)

(c) Use the implemented function `[x_next] = euler_step(deltaT,x0,u,@ode,param)` to build a function `[x_sim] = sim_euler(t,x0,u,param)` which simulates the robot's behaviour given a set of inputs $\mathbf{u}$. Starting at $x_0 = [0\ 0\ 0]^\top$, and using $param = [0.2\ \ 0.2\ \ 0.6]^\top$ and u, simulate the system and plot the simulated path of the robot. (0.5 point)

(d) Repeat step (b) and (c) for a Runge-Kutta integrator of order 4 instead of Euler: Implement a function `[x_next] = rk4_step(deltaT,x0,u,@ode,param)` which performs one integration step for a general ODE and a function `[x_sim] = sim_rk4(t,x0,u,param)` which simulates the robot's position. Have a look at the script on page 58 to find more information on the Runge-Kutta integrator. Plot the simulated path of the robot into the same plot as in (c). Do you see any difference? (0.5 points)