Fabio Marti, Matthias Roggo, David Lehnen, Daniel Gilgen

Robocup: Cooperative estimation and prediction of player and ball movement

Group Project

Department:
IfA – Automatic Control Laboratory, ETH Zürich

Supervising Professor:
Prof. Dr. John Lygeros, ETH Zürich

Supervisor:
M.S., Ph.D. Student Sean Summer, ETH Zürich

Zürich, March 2012

Abstract

Hier kommt das Abstract ...

Contents

List of Figures

1. Introduction

2. Simulation

One main aspect of this group work among the design of a Kalman filter was the construction of a graphical environment for our work, so we decided to implement a simulation of a Nao soccer match in MATLAB. Therefore we created three independent modules which build the framework for the simulation. The three parts contain functions concerning the playing field, the robots and the ball respectively. Since these parts are constructed in a modular fashion, we can change one module without influencing the other two.

2.1. The Playing Field

All graphical features concerning the playing field are implemented by the function `plot_env(ball).m`. As you can see the graphics of the ball are also part of this function. The reason why we didn't need a seperate function for the ball is that the appearance of the ball, contrary to this of the robots, always stays the same. The function `plot_env(ball).m` itself is again subdivided in three functions, which draw the field, the ball and, as a neat add on, the scorecounter seperately. We mostly use built-in MATLAB commands for this task such as `rectangle()` or `line()`. The following short code excerpt shows for example how the center point of the playing field is drawn

```
% CenterCircle
draw_circle(0,0,Field.centerCircleRadius,'k',0);
```

where `draw_circle(x,y,r,color,filled)` is a custom-build function to draw marker circles of the field, but also circles representing the robots and the ball. All functions are designed for fast calculations since we want as many frames as possible if the simulation is running. Figure **??** shows the playing field after the execution of `plot_env(ball).m` with suitable parameters for `ball`

Figure 2.1.: Playing field after initialization.

2.2. The Robots

Maybe the most important part of the simulation is the adequate depiction and behaviour of all eight robots. The latter task is split in three components: In a first step the robots are initialized, after that, their new positions on the field, according to their motion equations, are computed and in the end we are adding measurement noise for our filtering task. The initialization of the robots is quite simple. The function `dummy_init().m` just creates eight structs with the essential informations for every robot, i.e. its horizontal and vertical position, its direction and its team affiliation. Furthermore the function defines the robot's radius and its maximum possible angular change for one timestep as global variables. In a latter stage of development we dropped the simplification of a global eye and assumed that a location of a robot's position is only possible if it is in the sight of view of at least one other robot. So in the second version `robot_init().m` of this function, we additionally defined global variables for the robot's velocity, its distance of sight as well as its angle of sight. Once initialized we use the functions `dummy_step(Robot).m` and `robot_step(Robot).m` respectively to compute the attributes of all robots for every timestep. The key issue of both functions however is the recalculation of the position, i.e. the motion of the robots

```
%————————— Motion equations for robots  —————————%

    for i=1:8
        RobotStep(i).color = Robot(i).color;
        RobotStep(i).x = velocity(i) * cos(Robot(i).dir) + Robot(i).x;
        RobotStep(i).y = velocity(i) * sin(Robot(i).dir) + Robot(i).y;
        RobotStep(i).dir = d_omega(i) + Robot(i).dir;
    end
```

Later on the non-linearity of these equations will make it necessary that we use an extended Kalman filter instead of a simple linear one. The addition of process noise and the collision detection of robots also happen in these functions. Up to this point we computed the behaviour of an ideal robot. Since our goal is to filter out the uncertainty of motion of the robot parameters, we artifically have to add some measurement noise which we can filter later on. The functions `dummy_measure(Robot).m` and `robot_measure(Robot).m` are implemented for that purpose. `dummy_measure(Robot).m` simply adds white Gaussian noise to the position and the direction of every robot. Additionally with a certain possibillity there is no measurement at all, so the corresponding parameter is dropped. What we are assuming with this model is, that there is some global eye available which measures the position and direction of every robot with some given resolution. Since this scenario is not realistic in a RoboCup soccer match, because only the robots themselves can gain visual information, we developed the function `robot_measure(Robot).m` to solve this problem. According to this philosophy, a measurement of a robot is only available if it stands in front of a characteristic point on

the field or if it is within the distance of sight of another robot. Note that a robot only locates other robots if it is aware of its own position and that we get information from only one team, which will be the case for official RoboCup matches. If more than one measurement is available for a robot, the mean of all measurements is computed. After this computational part, the robots are added to the graphical environment by using the function `plot_robot(Robot,style).m`. It provides several features such as the coloration, the shape and the label of the robots on the field. A sample output on the graphical interface after several timesteps is shown in figure 2.2 below

Figure 2.2.: Capture of a regular simulation frame.

Note that measurements (white crosses) are not available for most robots (blue and magenta circles).

2.3. The Ball

The treatment of the ball is quite similar to that of the robots. The ball object is also represented by a struct containing its horizontal and vertical position, its direction and the velocity. These parameters are set by executing `ball_init().m`. This function also defines the ball's radius, its initial velocity and the friction towards the ground. The function `ball_step(Ball,Robot).m` does, like the equivalent function for the robots, the computations of the ball's next position and direction on the field. These parameters however do not only depend on the ball's dynamics but also whether it collides with one of the robots. The following MATLAB code shows the algorithm for this collision detection

```
%————————— Checking collision with robots —————————%

    for i=1:8
        dX = Ball.x - Robot(i).x;
        dY = Ball.y - Robot(i).y;
        if (dX.^2 + dY.^2 < (RobotParam.radius + BallParam.radius).^2)
            BallStep.dir = angle(dX + dY*j);
            BallStep.velocity = 1;
            BallStep.x = BallParam.velocity * Ball.velocity * cos(BallStep.dir) + Ball.x;
            BallStep.y = BallParam.velocity * Ball.velocity * sin(BallStep.dir) + Ball.y;
        end
    end
end
```

In our simple model we assume that if the ball collides with one of the robots, it regains its initial velocity and bounces away, perpendicular to the robot's position. Since the ball

too is a subject of measurement, we also needed a measurement function for this object. `ball_measure(Ball).m` adds measurement noise or, as the case may be, drops the measurement completely. This procedure is repeated with every timestep.

2.4. A Random Simulation

3. Kalman Filtering

3.1. Linear Kalman Filter

There exists a recursive Kalman filter algorithm for discrete time systems.[1] This ongoing Kalman filter cycle can be divided into two groups of equations

1. Time update equations

$$\begin{aligned} \hat{x}_k^- &= A\hat{x}_{k-1} + Bu_{k-1} \\ P_k^- &= AP_{k-1}A^T + Q \end{aligned} \tag{3.1}$$

2. Measurement update equations

$$\begin{aligned} K_k &= P_k^- H^T (HP_k^- H^T + R)^{-1} \\ \hat{x}_k &= \hat{x}_{k-1}^- + K_k(z_k - H\hat{x}_k^-) \\ P_k &= (I - K_k H)P_k^- \end{aligned} \tag{3.2}$$

To test this algorithm and to learn something about applying the Kalman filter on linear systems we created an example. A detailed description of the following example can be found in appendix A.1.
We consider a linear, timeinvariant model, given by the following circuit diagram in figure 3.1. First we added process noise and measurement noise to the system output.

(-3,-1)(9,5) (-2,4)A (-2,0)B (8,4)C0 (8,0)D (3,4)A2 (3,2)M (3,0)B2 (6,4)C2 (6,0)D2
[intensitylabel=i_L,labeloffset=-0.2](A)(A2)L [tensionlabel=U_1,tensionlabeloffset=-1.2,tensionoffset=-0.8,intensitylabel=i_1,dipoleconvention=generator](A2)(M)C_1
[tensionlabel=U_R,tensionoffset=0.8,labeloffset=-0.5](B2)(M)R
[tensionlabel=U_2,tensionlabeloffset=-1.2,tensionoffset=-0.8,intensitylabel=i_2,dipoleconvention=generator](C2)(D2)C_2 (B)(D) (A2)(C0)
[fillstyle=solid](A)0.075 [fillstyle=solid](B)0.075 [fillstyle=solid](C0)0.075
[fillstyle=solid](D)0.075 (A)(B)U_{in} (C0)(D)U_{out}

Figure 3.1.: Example: Linear electrical circuit.

The aim is to get an estimation of the noisy output. Therefore we applied the Kalman

filter algorithm based on the equations 3.1. In the figure 3.2, above we can see the input signal and the ideal measurement of the output signal. That ideal measurement includes process noise, which can obviously not be filtered by the Kalman filtering algorithm. Below one can see the noisy measurement on the left side and the filtered output on the right side.

Figure 3.2.: Example: Input signal, output signal, noisy output, and filtered output.

3.2. Extended Kalman Filter (EKF)

In the section above we discussed the usage of a Kalman filter on linear systems. Most systems, including the motion equations of our robots, however are nonlinear. Nevertheless linearizing our system around the current estimate still makes our Kalman filter useful and leads to the concept of the extended Kalman filter [1]. The recursive equations are quite similar to those of the linear Kalman filter

1. Time update equations

$$\begin{aligned} \hat{x}_k^- &= f(\hat{x}_{k-1}, u_{k-1}, 0) \\ P_k^- &= A_k P_{k-1} A_k^T + W_k Q_{k-1} W_k^T \end{aligned} \tag{3.3}$$

2. Measurement update equations

$$\begin{aligned} K_k &= P_k^- H_k^T (H_k P_k^- H_k^T + V_k R_k V_k^T)^{-1} \\ \hat{x}_k &= \hat{x}_{k-1}^- + K_k(z_k - h(\hat{x}_k^-, 0)) \\ P_k &= (I - K_k H_k) P_k^- \end{aligned} \tag{3.4}$$

The function $f(\hat{x}_k, u_k, w_k)$ contains the system's nonlinear dynamics where w_k represents the current process noise and $h(\hat{x}_k, v_k)$ denotes the nonlinear state-to-output relationship with the measurement noise v_k. Furthermore the matrices A_k, H_k, W_k and V_k are linearizations, i.e. partial drivatives, of their respective functions at time step k

$$
\begin{aligned}
A_{i,j} &= \frac{\partial f_i}{\partial x_j}(\hat{x}_{k-1}, u_{k-1}, 0) \\
H_{i,j} &= \frac{\partial h_i}{\partial x_j}(\hat{x}_{k-1}, u_{k-1}, 0) \\
W_{i,j} &= \frac{\partial f_i}{\partial w_j}(\tilde{x}_k, 0) \\
V_{i,j} &= \frac{\partial h_i}{\partial v_j}(\tilde{x}_k, 0)
\end{aligned}
\tag{3.5}
$$

We dropped the fact that all matrices should have a subscript k and that they are allowed be different at each time step. Again, before applying the theory to our simulation, we created a generic example of a nonlinear system. Further details can be looked up in section A.2. We did essentially the same as we did for the linear system: We simulated the ideal system, then added process and measurement noise and in the last step checked whether we could get rid of our artifically added measurement noise. The results for a sample run on MATLAB are shown in figure 3.3 below

Figure 3.3.: Example: Input signal, output signal, noisy output, and filtered output.

The two pictures above show the sinusoidal input on the left and the ideal output, i.e. without process and measurement noise, on the right. Thereunder we can see the plots of the noisy signal and the signal after it has been filtered by the extended Kalman filter. We can conclude that the extended Kalman filter too provides an acceptable performance if our measurement noise is not too big.

4. Estimation

4.1. Estimate of the Ball

4.2. Estimate of the Robots

A. Examples

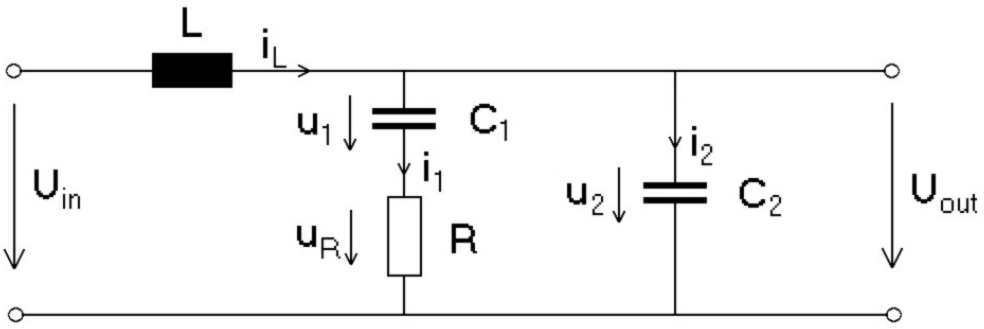
A.1. Kalman Filtering of Linear System

Group project - Linear model

Daniel Gilgen, David Lehnen, Matthias Roggo, Fabio Marti

March 12, 2012

We are considering a linear, timeinvariant model, given by the following circuit diagram



This is obviously a system of order 3 with the states $x_1 = i_L$, $x_2 = u_1$ and $x_3 = u_2$, the input $u = U_{\text{in}}$ and the output $y = U_{\text{out}}$. The system's equations are given by

$$L \cdot \frac{di_L}{dt} = u_L = U_{\text{in}} - u_2$$

$$\Longrightarrow \qquad \frac{di_L}{dt} = \frac{U_{\text{in}}}{L} - \frac{u_2}{L}$$

$$C_1 \cdot \frac{du_1}{dt} = i_1 = u_R \cdot R = (u_2 - u_1) \cdot R$$

$$\Longrightarrow \qquad \frac{du_1}{dt} = \frac{R}{C_1} \cdot u_2 - \frac{R}{C_1} \cdot u_1$$

$$C_2 \cdot \frac{du_2}{dt} = i_2 = i_L - i_1 = i_L - (u_2 - u_1) \cdot R$$

$$\Longrightarrow \qquad \frac{du_2}{dt} = \frac{i_L}{C_2} - \frac{R}{C_2} \cdot u_2 + \frac{R}{C_2} \cdot u_1$$

This leads to the following state space representation

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 0 & -\frac{1}{L} \\ 0 & -\frac{R}{C_1} & \frac{R}{C_1} \\ \frac{1}{C_2} & \frac{R}{C_2} & -\frac{R}{C_2} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \\ 0 \end{bmatrix} \cdot u$$

$$y = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

What we have now is a continuous time state space model of the form

$$\dot{x}(t) = \bar{A}x(t) + \bar{B}u(t)$$
$$y(t) = \bar{C}x(t)$$

Now we can assume, that all electrical devices are not ideal or that there are external disturbances like fluctuations in temperature or air moisture, such that their behaviour is not ideal. These factors will lead to process noise $w(t)$. Furthermore we will measure the output voltage U_{out} with a voltmeter, which will not measure the values exactly or which has an inappropriate resolution. This will add some measurement noise $v(t)$ to our model. Our new state space representation will be

$$\dot{x}(t) = \bar{A}x(t) + \bar{B}u(t) + w(t)$$
$$y(t) = \bar{C}x(t) + v(t)$$

with

$$E[w(t)w(\tau)] = Q_e\delta(t-\tau)$$
$$E[v(t)v(\tau)] = R_e\delta(t-\tau)$$

so $w(t)$ and $v(t)$ are white Gaussian noise processes. Since we assume that our model is quite reliable and there are only few external disturbances, the process noise will be much smaller that the measurement noise. Reasonable choices for Q_e and R_e could be

$$Q_e = \begin{bmatrix} 10^{-4} & 0 & 0 \\ 0 & 10^{-4} & 0 \\ 0 & 0 & 10^{-4} \end{bmatrix}, \qquad R_e = 10^{-1}$$

The last step is to convert this system into a discrete time linear system. The state space representation for such a system is given by

$$x_{k+1} = Ax_k + Bu_k + w_k$$
$$y_k = Cx_k + v_k$$

The matrices Q_e and R_e will stay the same. All other matrices are given by

$$A = e^{\bar{A}T}, \qquad B = \int_0^T e^{\bar{A}(T-\tau)}\bar{B}\, d\tau, \qquad C = \bar{C}$$

where T represents the sampling rate of our discrete time model. By choosing appropriate values for L, C_1, C_2 and R, we finally have enough information to build a Kalman filter for this linear system.

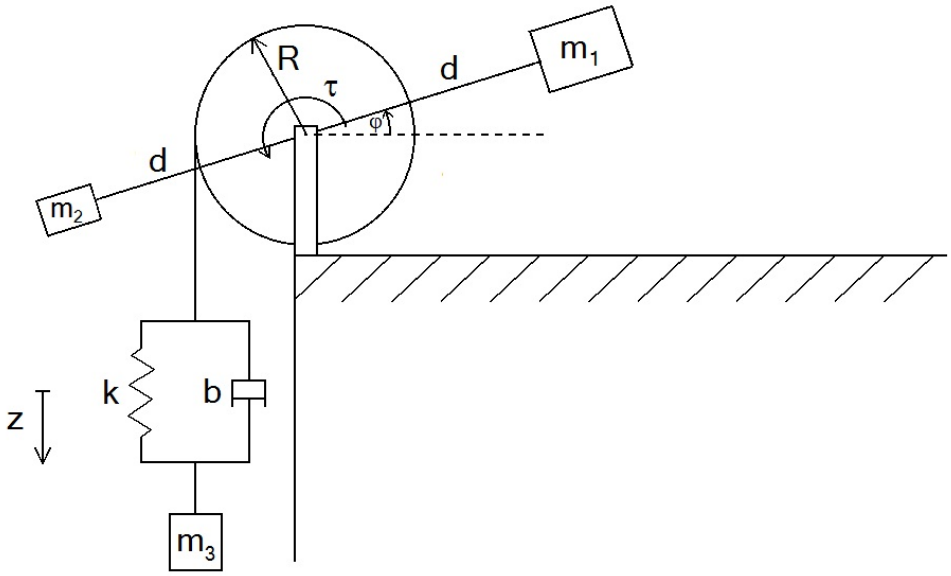
A.2. Kalman Filtering of Nonlinear System

Group project - Nonlinear model

Daniel Gilgen, David Lehnen, Matthias Roggo, Fabio Marti

April 22, 2012

Consider the following mechanical system



We assume now that the wheel doesn't have a mass, that m_1 has a moment of inertia of $J_1 = c_1 m_1$ and that m_2 has $J_2 = c_2 m_2$ with respect to the wheel's hub. Both masses are modelled as dots with distance d to the hub. Furthermore we are assuming that the force of the damper is given by $F_b = b \cdot \Delta z$. Using Newton's laws we obtain the equations

$$m_3\ddot{z} = m_3 g + k(\varphi R - z) + b(\dot{\varphi}R - \dot{z})$$

$$(J_1 + J_2)\ddot{\varphi} = \tau + gd\cos(\varphi)(m_2 - m_1) + k(z - \varphi R) + b(\dot{z} - \dot{\varphi}R)$$

By definig the states $x_1 = z$, $x_2 = \dot{z}$, $x_3 = \varphi$ and $x_4 = \dot{\varphi}$, the input $u = \tau$ and the output $y = \dot{\varphi}$ we find the following differential equations

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = g + \frac{k}{m_3}(x_3 R - x_1) + \frac{b}{m_3}(x_4 R - x_2)$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \frac{1}{c_1 m_1 + c_2 m_2}\left(u + gd\cos(x_3)(m_2 - m_1) + k(x_1 - x_3 R) + b(x_2 - x_4 R)\right)$$

$$y = x_4$$

In a next step we transform our continuous time system into a discrete time system. We use a simple Euler step for this task

$$\dot{x} \approx \frac{x_{k+1} - x_k}{T}$$

where T is the sampling period. The discrete system's equations now look as follows

$$\begin{aligned}
x_{1,k+1} &= x_{1,k} + T \cdot x_{2,k} \\
x_{2,k+1} &= x_{2,k} + T \cdot \left(g + \frac{k}{m_3}(x_{3,k}R - x_{1,k}) + \frac{b}{m_3}(x_{4,k}R - x_{2,k}) \right) \\
x_{3,k+1} &= x_{3,k} + T \cdot x_{4,k} \\
x_{4,k+1} &= x_{4,k} + T \cdot \left(\frac{1}{c_1 m_1 + c_2 m_2} \left(u_k + gd\cos(x_{3,k})(m_2 - m_1) + k(x_{1,k} - x_{3,k}R) + b(x_{2,k} - x_{4,k}R) \right) \right) \\
y_k &= x_{4,k}
\end{aligned}$$

Since we are dealing with nonideal effects, caused by model uncertainties or measurement errors, we add process noise w_k and measurement noise v_k to our model. We will get equations of the form

$$\begin{aligned}
x_{k+1} &= f(x_k, u_k, w_k) \\
y_k &= h(x_k, v_k)
\end{aligned}$$

Note that x_{k+1} as well as w_k and v_k are column vectors. In our nonlinear model, w_k and v_k are realizations of white Gaussian noise processes

$$\begin{aligned}
E[w_k w_n] &= W_k Q_k W_k^T \delta[k-n] \\
E[v_k v_n] &= V_k R_k V_k^T \delta[k-n]
\end{aligned}$$

where $W_k = \frac{\partial f}{\partial w_k}(x_{k-1}, u_{k-1}, 0)$ and $V_k = \frac{\partial h}{\partial v_k}(x_k, 0)$. Note that Q_k and R_k could change every timestep k. Possible values for them are

$$Q_k = \begin{bmatrix} 10^{-6} & 0 & 0 & 0 \\ 0 & 10^{-6} & 0 & 0 \\ 0 & 0 & 10^{-6} & 0 \\ 0 & 0 & 0 & 10^{-6} \end{bmatrix}, \qquad R_k = 10^{-2}$$

We have a full description of our nonlinear system and are now able to implement an extended Kalman filter by choosing reasonable values for the system's parameters.

Bibliography

[1] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," UNC-Chapel Hill, 2006.

[2] RoboCup Technical Committee, "RoboCup Standard Platform League (Nao) Rule Book," RoboCup, 2011.