Extended Kalman Filter for Tracking a Two-Wheeled Robot

An Extended Kalman Filter is to be designed for tracking the position and orientation of a two-wheeled robot that is moving on a plane. A schematic drawing of the robot is shown in Fig. 1.

![Diagram of a two-wheeled robot](image)

**Figure 1:** Top view of the two-wheeled robot (left) and relevant physical quantities (right): \((x, y)\) is the robot’s position, \(r\) its orientation, and \(s_R\) and \(s_L\) denote the right and left (linear) velocity.

The robot can command its left and right wheel angular velocities, \(u_L(t)\) and \(u_R(t)\) (in rad/s), at discrete time instants \(t_0 = 0, t_1 = T, t_2 = 2T, \ldots\), where \(T\) is the sampling time in s. The commands are kept constant in between sampling instants and are assumed to be followed instantaneously. Throughout this problem description, we use \(k\) to denote the discrete-time index: for example, \(u_L[k] = u_L(t_k) = u_L(kT)\) and \(u_R[k] = u_R(t_k) = u_R(kT)\).

The translational speed \(s_t\) (in m/s) of the vehicle is

\[
s_t[k] = \frac{s_R[k] + s_L[k]}{2} 
\]

with \(s_R\) and \(s_L\) the right and left wheel (linear) velocities,

\[
s_R[k] = W_R u_R[k] \quad \text{and} \quad s_L[k] = W_L u_L[k],
\]

and \(W_R\) and \(W_L\) the right and left wheel radii (in m). The wheel radii are assumed to be known exactly. The rotational speed \(s_r\) (in rad/s) of the vehicle is

\[
s_r[k] = \frac{s_R[k] - s_L[k]}{2B},
\]

where \(B\) is the wheel base (distance in m of the wheels from the robot center), see Fig. 1. The wheel base is constant with time, but it is not known perfectly; it is modeled as a continuous random variable according to

\[B = B_0(1 + \xi),\]
with the known nominal wheel base $B_0$ (in m) and the uniformly distributed random variable $\xi \in [-\xi, \xi]$.

The kinematic equations of the robot can be written in continuous-time as follows:

\begin{align*}
\dot{x}(t) &= s_t(t) \cos(r(t)) \\
\dot{y}(t) &= s_t(t) \sin(r(t)) \\
\dot{r}(t) &= s_r(t),
\end{align*}

where $(x(t), y(t))$ is the position of the robot (in m) and $r(t)$ its orientation (in rad). Since the robot’s commands are constant over a sampling interval $[kT, (k+1)T)$, $s_t(t) = s_t[k]$ and $s_r(t) = s_r[k]$ for $t \in [kT, (k+1)T)$. The robot is assumed to start at $(x(0), y(0)) = (x_0, y_0)$ with orientation $r(0) = r_0$, where $x_0, y_0 \in [-p, p]$ and $r_0 \in [-r, r]$ are uniformly distributed random variables.

A discrete-time model of (4)–(6) with sampling time $T$ is given by

\begin{align*}
x[k+1] &= x[k] + T s_t[k] \cos(r[k]) - \frac{1}{2} T^2 s_t[k] s_r[k] \sin(r[k]) \\
y[k+1] &= y[k] + T s_t[k] \sin(r[k]) + \frac{1}{2} T^2 s_t[k] s_r[k] \cos(r[k]) \\
r[k+1] &= r[k] + T s_r[k],
\end{align*}

where $x[k] = x(kT)$, $y[k] = y(kT)$, etc.

At varying instants of time, the robot may receive measurements of its position and orientation that are corrupted by sensor noise, i.e.

\begin{align*}
z_x &= x + w_x \\
z_y &= y + w_y \\
z_r &= r + w_r,
\end{align*}

with $w_x, w_y \sim \text{Tri}(\bar{p})$ and $w_r \sim \text{Tri}(\bar{r})$, where $a \sim \text{Tri}(\bar{a})$ denotes a continuous random variable with triangular probability density function centered at zero with width $2\bar{a}$, see Fig. 2. At any sampling instant $t_k = kT$, measurements may be available from one, two, three, or none of the sensors.

\begin{figure}[h]
\centering
\includegraphics[width=0.3\textwidth]{triangle.pdf}
\caption{Triangular probability density function, $a \sim \text{Tri}(\bar{a})$.}
\end{figure}

\footnote{The derivation can be found in the Appendix.}
Estimator Design Part 1

The objective is to design an Extended Kalman Filter (EKF) to estimate the position and orientation of the two-wheeled robot. The estimator will be implemented in discrete time. At time \( t_k = kT \), the estimator has access to the time \( t_k \), the control inputs \( u_L[k] \) and \( u_R[k] \), and possibly the measurements \( z_x[k], z_y[k], \) or \( z_r[k] \). Furthermore, the constants \( W_R, W_L, B_0, \xi, \pi_r, \pi_l, \) and \( \pi \) are known to the estimator. The orientation, the position, and the wheel base are estimator states.

Estimator Design Part 2

So far, it has been assumed that the wheel angular velocity commands transform directly to translational and rotational velocity of the robot as given by equations (1)–(3). In this modified estimator design problem, non-idealities in the actuation mechanism (for example, wheel slip or imperfect command tracking) are considered: they are modeled as multiplicative noise on the actuation commands. That is, equation (2) is replaced by

\[
 s_R[k] = W_R u_R[k] \left( 1 + v_R[k] \right) \quad \text{and} \quad s_L[k] = W_L u_L[k] \left( 1 + v_L[k] \right),
\]

where \( v_R[k], v_L[k] \sim \text{Tri}(\pi) \). A second EKF design shall take the actuation non-idealities into account by including appropriate process noise. Additionally to the information of the EKF in design part 1 above, this EKF has access to the constant \( \pi \).

For both estimator designs, it can be assumed that all random variables \( \xi, r_0, x_0, y_0, \{ w_x[k] \}, \{ w_y[k] \}, \{ w_r[k] \}, \{ v_L[k] \}, \) and \( \{ v_R[k] \} \) are mutually independent and independent over time.

Provided Matlab Files

A set of Matlab files is provided on the class website. Please use them for solving the exercise.

- **run.m**: Matlab function that is used to simulate the truth system, run the estimator, and display the results. The single input argument designPart is used to specify the estimator design part (1 or 2).
- **Estimator.m**: Matlab function template to be used for your implementations of the EKFs. Use this file for both estimator design parts. You can use the input argument designPart if you need to distinguish between the two.
- **KnownConstants.m**: Constants known to the estimator.
- **UnknownConstants.m**: Constants not known to the estimator.
- **CalculateInputs.m**: Auxiliary function used to calculate the input wheel speeds.
- **Uniform.m**, **UniformMinMax.m**, **Triangular.m**: Random number generators.

Task

Implement your solutions for the estimator design part 1 and part 2 in the file **Estimator.m**. Your code has to run with the Matlab function **run.m** (for both parts, that is, **run(1)** and **run(2)**). Please use exactly the function definition as given in the template **Estimator.m** for the implementation of your estimators.

For evaluating your solution, we will test it on the given problem data. Moreover, we will do suitable modifications of the parameters in **KnownConstants.m** and **UnknownConstants.m** and also test your estimator on those.
For judging your own solution, a typical performance of an EKF implementation for part 1 and the problem data provided in KnownConstants.m and UnknownConstants.m is shown in Fig. 3 and 4.

**Deliverables**

Please hand in by e-mail your implementation of the estimator design part 1 and part 2 in Estimator.m. It has to be a single file with exactly the same function definition as in the provided template. Include the file into a zip-file, which you name RE11Ex1_Names.zip, where Names is a list of the pre- and surnames of all students who have worked on the solution (for example RE11Ex1_AngelaSchoellig_SebastianTrimpe.zip). Up to three students are allowed to work together on the programming exercise. They will all receive the same grade.

Send your file to Sebastian (strimpe@ethz.ch) until the due date indicated above. We will send a confirmation e-mail upon receiving your solution. You are ultimately responsible that we receive your solution in time.

**Plagiarism**

When handing in your solution to the programming exercise, you (that is, in case of a group work, each individual member of the group) confirm that the work is original, has been done by you independently and that you read and understood the ETH Citation etiquette (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf).

Each submitted solution may be tested for plagiarism.

**Appendix**

The discrete time model (7)–(9) can be derived from (4)–(6) as follows. Since $s_t(t)$ and $s_r(t)$ are constant over the sampling interval $[t_k, t_{k+1})$, that is, $s_t(t) = s_t[k]$ and $s_r(t) = s_r[k]$ for $t \in [t_k, t_{k+1})$, we have from (6) for $t \in [t_k, t_{k+1})$

$$
\dot{r}(t) = s_r[k] \Rightarrow r(t) = r[k] + (t-t_k)s_r[k] \Rightarrow r[k+1] = r[k] + (t_{k+1}-t_k)s_r[k],
$$

by continuity of $r(t)$ at $t_{k+1}$. By substituting the equation for $r(t)$ into (4), we obtain for $t \in [t_k, t_{k+1})$

$$
\dot{x}(t) = s_l[k] \cos(r[k] + (t-t_k)s_r[k]) = s_l[k] \cos((t-t_k)s_r[k]) - s_l[k] \sin(r[k]) \sin((t-t_k)s_r[k]) \approx s_l[k] \cos(r[k]) - (t-t_k)s_l[k]s_r[k] \sin(r[k]), \quad \text{for small } (t-t_k)
$$

$$
\Rightarrow x[k+1] = x[k] + \int_{t_k}^{t_{k+1}} \dot{x}(t) \, dt \approx x[k] + (t_{k+1}-t_k)s_l[k] \cos(r[k]) - \frac{1}{2} (t_{k+1}-t_k)^2 s_l[k]s_r[k] \sin(r[k]),
$$

by continuity of $x(t)$ at $t_{k+1}$. Similarly, one can derive

$$
y[k+1] = y[k] + (t_{k+1}-t_k)s_l[k] \sin(r[k]) + \frac{1}{2} (t_{k+1}-t_k)^2 s_l[k]s_r[k] \cos(r[k]).
$$

This discrete time model is used in run.m for numerical simulation of the robot. Clearly, one may also employ other discretization methods such as forward Euler (less accurate) or use higher order methods for numerical simulation of (4)–(6) (e.g. in the Matlab ODE suite; usually slower).
Figure 3: Typical tracking performance of an EKF implementation for design part 1 and the provided problem data. The shown position tracking error is the root mean square of the error in $x$ and $y$ position.

Figure 4: Estimation errors (blue) with +/- one standard deviation (red) for the estimator data of Fig. 3.