The Application of the Kalman Filter on Robot Soccer

Dylan Oelmann, Zachary Rothsching

December 10 2018
## Contents

1 Introduction 3

2 Basic Concepts of the Bayesian Filter 7
   2.1 State, Environment, and Control Data 7
      2.1.1 State 7
      2.1.2 Measurement Data 8
      2.1.3 Control Data 8
      2.1.4 Belief Distribution 8

3 The Bayesian Filter 9
   3.1 An Overview of the Bayes Filter 9
   3.2 Derivation of the Bayes Filter 9
   3.3 The Markov Assumption 12
   3.4 Necessity to Restrict to Gaussian Distributions 12

4 The Kalman Filter 14
   4.1 Explanation of the Kalman Filter Algorithm 14
   4.2 Application of the Kalman Filter on Robot Soccer 18
      4.2.1 The Variables of the Kalman Filter 19
      4.2.2 The Modelling of Robot Soccer Using Python Code 21

5 Conclusion 24
Abstract

Bayesian Filters are algorithms which allow for the recursive calculation of the current belief of a state. It allows robots to move and make decisions based on its model of the current state. Additionally, it does not limit robots to performing a simple task autonomously, rather it allows to robots to react based on what the current state is in order to accomplish a specific task. One of the most common variations of the Bayesian filter is the Kalman filter, which allows the Bayes filter to be modelled linearly. The non-linear case is seen in the extended Kalman filter, which will not be discussed in this paper. This paper discusses the Bayes filter and some of its basic properties as well as one of its derivatives, the Kalman filter. The Kalman filter is then applied to a real world scenario in the context of robot soccer. The robots playing soccer use the Kalman filter to model the position and velocity of the ball in the x and y directions. The computations used by the robots have been found to have excellent accuracy in modelling the velocity and position of the ball, and carry very small discrepancies from the true trajectory. Kalman and Bayes filter modelling are very important fields of study because they not only push forward the development of artificial intelligence, but also push forward the human understanding of the brain.

1 Introduction

Robotics has become one of the most predominant fields of study in the last few decades. The goal is to build a robot which can exist in the physical world and perform actions based on its perception of the environment through its sensors, much like a human. Over the years, many successful robotic systems have come to fruition, such as actuated arms used during surgery or in assembly lines, self-driving cars, or robotic vehicles used in planetary exploration. The opportunities robotics provides are endless, and humans are continuously pushing their understanding in order to create robots which function more fluidly and more autonomously. As such, artificial intelligence has also undergone many advancements, and robotic systems are being placed in new situations where they must operate in unpredictable and unstructured environments, that are not limited to the repetition of identical tasks. The Robot Soccer World Cup, also known as RoboCup, was created under this idea; to provide an unpredictable environment for robots to operate in, in order to push the human understanding of artificial intelligence.

The concept of robots playing soccer was first initialized in 1992 by a group of Japanese scientists who gathered together to discuss the major challenges of artificial intelligence. This gathering eventually led to the creation of the Federation of International Robot-soccer Association (FIRA) in 1997. That year, the first official RoboCup was held in Nagoya Japan, where 40 teams
participated, and over 5000 spectators attended. The creators of the RoboCup had set a goal to have fully autonomous robots capable of beating the top human players of the FIFA World Cup by the year 2050. Since its creation, the competition has expanded and now holds multiple versions of robot soccer, where different types of robots compete. Most recently, the 22nd edition of the competition took place in June of 2018, and was held in Montreal. Over 5000 robots competed through various divisions.

Robot soccer has the same underlying rules as human soccer, but with a few exceptions. Some characteristics which distinguish it from FIFA is that games last 20 minutes (2 halves of 10 minutes), the games are played on artificial turf, and the field’s dimensions are 9 m by 6 m. 2.6 m separate the goal posts and games are usually played with 1 to 4 players. The robots vary in size, being kid-sized, and adult-sized, depending on which league they are participating in. The image above shows a picture of a humanoid kid-size league, where robots measure between 40 and 90 cm tall.
The robots are prohibited to exhibit “forceful contact that significantly destabilizes a player, such that walking and/or kicking is impeded” (RoboCup Humanoid League Rulebook 2018). This includes falling into a player or walking carelessly into another player at a significant speed. Therefore, the robots are expected to be able to walk and run dynamically, stand upright, and recover from a fall and kick the ball while maintaining balance. Additionally, they must be able to localize themselves, the ball, other players, and the field through their sensors. It is prohibited for a robot to measure Earth’s magnetic field to orient themselves. As such, the robots rely on their sensors and cameras, and use the lines of on the field in order to orient themselves. These sensors are also used to identify objects in front of them, as well as measure distances, such as the distance between the robot and the ball.

As stated, the goal of the RoboCup is to create robots capable of competing against human soccer players. To reflect this idea, the robot’s form is humanoid, and their actions are restricted to the kinematic equivalence of human motion. For example, their neck joint can only move 270° horizontally and 180° vertically. They also are allowed sensors which measure volume, frequency, touch, force, and temperature. Furthermore, they can also measure the local state of the system, such as voltages, currents, forces, movements, accelerations, and rotational speed. As there are many sensors incorporated into the robots, the likelihood of accurately predicting a state becomes more difficult, especially considering the unpredictability of the world. Even sensors and motors are subject to noise. Certain things like angle of orientation, roughness of ground, position of opposing player, and many more, all contain noise because they exist in the real world. As a result, the robot must be implemented with an appropriate probabilistic model which will be able to account for the randomness of the robot’s environment.

Regardless of how accurate a robot’s sensors are, there will always be noise in the environment, which emphasizes the importance of implementing a model
which takes into account uncertainty while computing accurate predictions of future states. As a result, modern day robotics often prefers probabilistic modelling as opposed to deterministic modelling. Deterministic modelling predicts one course of action in a given environment or situation. It does not take into account alternative courses of action or noise. The world does not favor deterministic modelling since there are almost infinitely many sources of noise. Thus, having a more flexible model which accounts for such randomness is preferred, which is why robots use probabilistic modelling. Probabilistic modelling takes into account all plausible dynamics in a distribution, as well as accounting for noise. Rather than predicting one exact value for the outcome of an event, it derives a possible set of solutions based on the initial information of the system, and plots them under a probability distribution curve. Another benefit to probabilistic modelling is that it allows for approximate but controlled calculations. As a result, the robot can compute calculations much more quickly and efficiently.

The preferred algorithm for probabilistic modelling in modern robotics is the Kalman Filter, which is a derivation of the Bayesian filter, and was originally conceptualized in the late 1950’s by Rudolph Kalman. Despite being met with skepticism in the 1950’s, it is now one of the most popular applications of probabilistic modelling. It is used in many different fields, but is widely used in control systems, navigation, signal processing, and guidance; it was even used in the navigation of the spacecraft during the Apollo missions in the 1960’s. It filters noise, estimates non-observable states and makes predictions that are not limited to one outcome. The Kalman filter uses a recursive state estimation algorithm to predict a set of variables and then estimates a joint probability distribution between the variables of the state at different time steps. The algorithm takes into account the noise at each time step and limits the number of computations by only taking into account the relationship between the previous state and the proceeding state. In other words, it determines how much the uncertainty of the previous state will influence the following state. Because it does not retain a memory of all previous states, the Kalman filter becomes very light on computations, making it very efficient for real life applications. Essentially, it uses a Bayesian approach where one state or belief is only determined by the state that immediately precedes it, which only works under the condition that the initial belief of the state is accurate. Furthermore, for the algorithm to work, it must assume that all of its components follow a Gaussian distribution, so that the dynamics of the system can take on a linear form. By doing so, continuous states can be modelled through the mean and the covariance of the distribution; the distribution is centered around the predicted state denoted by the mean $\mu$, with uncertainty denoted by the covariance $\Sigma$. The algorithm essentially functions in two part: the prediction step, and the measurement update step, which will be discussed in depth in this paper.
The robots in the RoboCup actually apply the Kalman filter as one of the ways in which they self-localize themselves. They also use the Kalman filter to track the position of the ball on the field. Because the measurement of the ball’s location will be taken in such short time fragment, its movement can be considered linear. Acceleration is set to zero because the acquisition and update of the data are very quick, which allows for the impact of acceleration to be overlooked. Theoretically, assuming there is no noise, the position of the ball will follow simple kinematic laws. However in the real world, the randomness of the environment makes it unrealistic to model the position and velocity of the ball using only the kinematic laws. By using a Kalman filter algorithm, it will be possible to predict the location of the ball based on the previous position, and the knowledge of the initial position. It will also allow the robot to make its predictions at a fast speed since it operates based on strictly the previous state, and does not store the unnecessary information of all previous states. It provides the robot with fast and easy computations in order to keep up with the position and velocity of the ball.

In this paper, basic probabilistic concepts such as state and measurement data will be explained, followed by how the Bayesian and Kalman algorithms model state, and how the robots in robot soccer apply the Kalman filter to play the game.

2 Basic Concepts of the Bayesian Filter

2.1 State, Environment, and Control Data

2.1.1 State

The state $x_t$ is a continuous random variable which accounts for various factors relative to the robot’s environment that can influence its future. It processes information relative to the surrounding environment of the robot, as well as information about the robot itself. Furthermore, states can be dynamic or static. Dynamic states are states which are in continuous motion and change given time. Contrarily, static states remain constant and do not change relative to time. Complete states attempt to model every possible aspect of a robot’s environment, which is very computationally complex and generally unrealistic given the randomness of the world. As a result, the more computationally efficient incomplete state is favored as it does not take into account unnecessary information. This unnecessary information is then categorized as noise. States which are by definition incomplete can still be considered complete given the Markov assumption, described in detail in section 3.3. The assumption allows the posterior state at time $t$ to be found as a result of Markov computations. This increases computational efficiency, as only the previous state is needed in order to determine the posterior state $x_t$. Essentially, the robot does not need to keep track of every prior state, rather it only requires preceding state $x_{t-1}$. This assumption is only valid given that the initial state $x_0$ is accurately initialized.
2.1.2 Measurement Data

Measurement data is information obtained by the robot about its environment through its sensors. The sensors provide information about a given state and allow the robot to physically alter the state through actions. Measurement data at time $t$ is denoted by the variable $z_t$. This information improves the robot’s knowledge of a given state and allows for the robot to make better decisions. Simple static observation improves a robot’s knowledge of any given state, as it is simply gathering information. Alternatively, a robot in motion will have a decreased knowledge of state because of the dynamic nature of the environment. The continuously changing state found when a robot is in motion increases the influence of noise, which makes planning out the best course of action more difficult. This is counterbalanced by incorporating measurements into the state estimation at every time step.

2.1.3 Control Data

Control data is responsible for modelling the change in state and environment of the robot, and is denoted by the variable $u_t$. Control data monitors how the posterior state has changed given a certain action in order to provide information on the current state. For example, control data will measure how much a robot’s motors have turned in order to provide the robot information on its current pose. It can also serve as a correction update to improve the accuracy of modelling.

2.1.4 Belief Distribution

The belief distribution is a probabilistic model representing the robot’s internal knowledge about the state of the environment. Belief state is computationally essential for the robot since the true state of the environment is unknown. As a result, the robot must rely on its internal knowledge to determine what it believes the current state is. The belief state at time $t$ is modelled by the probability distribution $\text{bel}(x_t)$. Furthermore, the belief state can be represented by a probability density function, which models the current state, given all prior knowledge on the environment and control data:

$$ \text{bel}(x_t) = p(x_t | z_{1:t}, u_{1:t}) $$

Furthermore, the robot also makes predictions of what the upcoming state will be based on all past results up until the previous result. This prediction belief is denoted by the distribution $\overline{\text{bel}}(x_t)$, and is also the result of a probabilistic equation:

$$ \overline{\text{bel}}(x_t) = p(x_t | z_{1:t-1}, u_{1:t}) $$

This takes into account all prior knowledge about the environment, as well as the control data up until time $t$, but not the most recent observation. This allows for the calculation of the probability of the state at time $t$. The prediction belief is essentially a prediction of the current state given the entire history of
the observations, and is used to calculate the belief distribution. The step of incorporating the current observation $z_t$ into the predicted belief distribution is known as the *Measurement Update*.

# 3 The Bayesian Filter

## 3.1 An Overview of the Bayes Filter

The Bayes Filter is a recursive algorithm allows the robot to calculate its belief of the current state, $\text{bel}(x_t)$, given the previous state $\text{bel}(x_{t-1})$. This calculation is done in two essential steps:

1. The prediction belief $\overline{\text{bel}}(x_t)$ is obtained by taking the integral of the belief at the previous state, $\text{bel}(x_{t-1})$, multiplied by the probability distribution of the state $x_t$ given control data $u_t$ and the previous state $x_{t-1}$:

$$\overline{\text{bel}}(x_t) = \int p(x_t|u_t, x_{t-1}) \text{bel}(x_{t-1}) \, dx$$

2. Once the prediction belief has been calculated, the Bayes filter then multiplies $\overline{\text{bel}}(x_t)$ with the probability that the observation $z_t$ is realized. This multiplication does not return a probability, thus it is normalized by the normalization constant $\eta$.

$$\text{bel}(x_t) = \eta p(z_t|x_t) \overline{\text{bel}}(x_t)$$

This is the crucial Bayesian step which integrates the observations in the robot’s beliefs about the predicted state. Thus, the Bayes filter algorithm is:

\[
\begin{align*}
\overline{\text{bel}}(x_t) &= \int p(x_t|u_t, x_{t-1}) \text{bel}(x_{t-1}) \, dx \quad (1) \\
\text{bel}(x_t) &= \eta p(z_t|x_t) \overline{\text{bel}}(x_t) \quad (2)
\end{align*}
\]

This provides the belief of the state at time $t$. One time step later, $\text{bel}(x_{t-1})$ becomes $\text{bel}(x_t)$, and gets fed back into equation 1 of the filter. As a result, the filter is recursive.

## 3.2 Derivation of the Bayes Filter

In order to derive the Bayes filter, it will be shown that it correctly calculates state probability distribution $p(x_t|z_{1:t}, u_{1:t})$ from the previous state distribution one step earlier: $p(x_{t-1}|z_{1:t-1}, u_{1:t-1})$. Furthermore, in order for the filter to be correct, the initial belief state, $\text{bel}(x_0)$ must be correct. The Bayes filter, at its core, follows Bayes’ rule. As such, the first step in deriving the Bayes filter
above is to apply Bayes’ rule:

\[
p(x_t|z_{1:t}, u_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t-1})p(x_t|z_{1:t-1}, u_{1:t-1})}{p(z_{1:t}, u_{1:t})}
\]

The equation then get simplified by rewriting the denominator \(p(z_{1:t}, u_{1:t})\) as the normalization factor \(\eta\). Thus, the equation becomes:

\[
p(x_t|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t, z_{1:t-1}, u_{1:t-1})p(x_t|z_{1:t-1}, u_{1:t-1})
\]

This only works if the density \(p\) can be considered a complete model of the state of the environment. The idea that the state is complete can be exploited to further simplify the equation; This Markov assumption means that we can use the following simplification:

\[
p(z_t|x_t, z_{1:t-1}, u_{1:t}) = p(z_t|x_t)
\]

This equation essentially states that the historical control data is not needed to determine the measurement data if the state is complete. The complete state will have already taken into account everything necessary, so the control data becomes obsolete. When substituted into the equation above, the Bayes filter is further simplified to:

\[
p(x_t|z_{1:t}, u_{1:t}) = \eta p(z_t|x_t)p(x_t|z_{1:t-1}, u_{1:t-1})
\]

Furthermore, the predicted belief state:

\[
\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t})
\]

can also be substituted into the equation, resulting in equation 2 of the Bayesian Filter Algorithm:

\[
bel(x_t) = \eta p(z_t|x_t)\overline{bel}(x_t)
\]
The derivation of the first equation in the algorithm first begins with the expansion of the formula:

\[ \text{bel}(x_t) = p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) \]

This can be expanded into:

\[ \text{bel}(x_t) = \int p(x_{t-1}, z_{1:t-1}, u_{1:t})p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) \, dx_{t-1} \]

Once again, the assumption that the state is complete can allow for the implementation of the following formula to simplify the above equation:

\[ p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t}) = p(x_t|x_{t-1}, u_t) \]

\( z_{t-1} \) is not necessary since the state is assumed to be complete; the state has already taken the information into account. Control data \( u_t \) cannot be omitted, since it provides information on the change of state. This results in the following formula:

\[ \text{bel}(x_t) = \int p(x_{t-1}, z_{1:t-1}, u_{1:t})p(x_t|x_{t-1}, u_t) \, dx_{t-1} \]

Recalling from the section on belief distribution, substituting the equation:

\[ \text{bel}(x_t) = p(x_t|z_{1:t}, u_{1:t}) \]

will result in equation 1 of the Bayesian filter. Thus:

\[ \text{bel}(x_t) = \int p(x_t|u_t, x_{t-1})\text{bel}(x_{t-1}) \, dx \]
3.3 The Markov Assumption

The Bayes Filter assumes that the world is Markovian. In other words, it assumes that the state is complete. It suggests that all past and future states are independent if one knows the current state. It allows the current state to be computed as a result of a Markov chain, omitting the need for any states other than the previous one. The Markov assumption only works properly under the additional assumption that the initial state \( x_0 \) is correctly initialized. Correctly initializing the initial state serves as a starting point for the Markov chain, and if it is correct, so will all future states. However, in practice there may be variables which violate the Markov assumption. One such example is certain aspects of the environment which remain unmodeled in the state \( x_t \). Omitting background processes in the environment violates the assumption that the state is complete. Furthermore, there may be Gaussian approximation errors when approximating the predicted belief \( \hat{b}(x_t) \).

3.4 Necessity to Restrict to Gaussian Distributions

In order to make the model computationally efficient, it is necessary to restrict the Bayes filter to Gaussian distributions. The general Bayesian filter models its variables based on conditional probabilities, which in general cannot be modelled by a linear regression. As such, restricting to Gaussian improves computational efficiency, “and provides an optimal linear estimator” (Kohanbash 2014). It also expresses all necessary components of the Bayesian filter using only two variables: mean \( \mu \) and covariance \( \Sigma \). Gaussian modelling models a probability distribution curve with best state estimate \( \mu \), and with uncertainty \( \sigma \). This still allows for modelling of every possible state outcome and takes into account the randomness of the environment. As such, Kalman Filters are often advantageous in the field of robotics, due to the simplicity and computational efficiency of their algorithm in contrast to a nonlinear Bayesian filter.

To implement the Kalman filter, the posteriors of the Bayesian filter must follow the Markov assumption, and the posteriors are Gaussian under the condition that the following three properties hold.

1. The next state must be a linear function of the previous state and the control with noise.

\[
x_t = A_t x_{t-1} + B_t u_t + \epsilon_t
\]

Here \( x_t \) and \( x_{t-1} \) are state vectors, and \( u_t \) is the control vector. Both are column
vectors and take on the following form:

\[ x_t = \begin{bmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{bmatrix}, \quad u_t = \begin{bmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{n,t} \end{bmatrix} \]

Also, \( A_t \) is a square matrix of size \( n \times n \), where \( n \) is the dimension of state vector, \( x_t \). When multiplying \( A_t \) with the posterior state vector \( x_{t-1} \) a linear equation is obtained. Similarly, \( B_t \) is also a matrix, but of size \( n \times m \), where \( m \) is the dimension of the control vector. The multiplication of \( B_t \) and \( u_t \) also provides a linear dependence. Therefore, adding these multiplications allow the state transition functions to have linear dynamics.

The last variable, \( \epsilon_t \) is a Gaussian random vector that models the randomness in each change of state with the same dimension as the state vectors. It has a mean of 0 and its covariance matrix will be represented as \( R_t \). To determine the state transition probability, the multivariate normal distribution is used. First the general form of the multivariate Gaussian is seen in the following equation with column vector \( V \), similar to the state vector \( x_t \).

\[
p(V) = (det 2\pi\Sigma)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (V - \mu)^T \Sigma^{-1} (V - \mu) \right\}
\]

To apply it in a Kalman filter, \( V \) will be replaced by the state at time \( t \), the mean, \( \mu \) in this equation will be the mean of the posterior, which takes the form of \( A_t x_{t-1} + B_t u_t \), and the covariance will be replaced by \( R_t \). This gives the equation:

\[
p(x_t|u_t, x_{t-1}) = (det 2\pi R_t)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (x_t - (A_t x_{t-1} + B_t u_t))^T R_t^{-1} (x_t - (A_t x_{t-1} + B_t u_t)) \right\}
\]

2. The measurement probability, \( p(z_t|x_t) \) must also be a linear function with additive Gaussian noise. To satisfy this condition, the measurement data will be determined according to the following equation:

\[ z_t = C_t x_t + \delta_t \]

Here \( C_t \) is a matrix of size \( k \times n \) where \( k \) is the dimension of the measurement vector \( z_t \) and where \( \delta_t \) is the vector that describes the observational noise. It has a multivariate gaussian distribution with mean of 0 and covariance denoted as \( Q_t \). The same process performed to arrive at the multivariate equation for state vectors (seen in the first condition) can be applied to have the measurement probability be determined by a multivariate Gaussian distribution.

\[
p(z_t|x_t) = (det 2\pi Q_t)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t) \right\}
\]
3. The last condition is that the initial belief, \( \text{bel}(x_o) \) must also be a normal distribution. For this the mean of the initial belief will be denoted by \( \mu_o \) and the covariance matrix of the initial state will be denoted by \( \Sigma_o \). This gives \( \text{bel}(x_o) \) determined by the following multivariate distribution:

\[
\text{bel}(x_o) = p(x_o) = (\det 2\pi \Sigma_o)^{-\frac{1}{2}} \exp \left\{ -\frac{1}{2} (x_o - \mu_o)^T \Sigma_o^{-1} (x_o - \mu_o) \right\}
\]

4 The Kalman Filter

4.1 Explanation of the Kalman Filter Algorithm

The Kalman filter is a filter which allows one to compute beliefs about continuous states. It is inapplicable to states which are discrete. The Kalman filter states that the belief is represented by a multivariate Gaussian distribution with mean \( \mu_t \) and covariance \( \Sigma_t \). The mean is essentially the best estimator for the state at time \( t \). The covariance is expressed as a matrix where each of its entries is the degree of covariance between the \( i \)th state variable and the \( j \)th state variable, making it a square matrix. These matrices will take on the following forms, given a random state vector \( x_t \):

\[
x_t = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mu_t = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}, \quad \Sigma_t = \begin{bmatrix} \text{Var}(x_1) & \text{Cov}(x_1, x_2) & \ldots & \text{Cov}(x_1, x_n) \\ \text{Cov}(x_2, x_1) & \text{Var}(x_2) & \ldots & \text{Cov}(x_2, x_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{Cov}(x_n, x_1) & \text{Cov}(x_n, x_2) & \ldots & \text{Var}(x_n) \end{bmatrix}
\]

If a robot’s state is being tracked based on its position and velocity, the mean and covariance can take on the following matrix forms:

\[
\begin{bmatrix} \text{position} \\ \text{velocity} \end{bmatrix}, \quad \begin{bmatrix} \Sigma_{pp} & \Sigma_{pv} \\ \Sigma_{vp} & \Sigma_{vv} \end{bmatrix}
\]

These matrices provide the necessary information on the current state of the robot. The robot must be able to predict what the next state will be given the current state. The step in which the prediction is made is implemented by the transformation matrix \( A_t \). This takes all the possible values of the initial state and transforms them using the dynamics to a new predicted state. As previously mentioned, this only works if the starting state was correctly initialized.

In the case where state is defined as the robot’s x and y positions, as well as its x and y velocities, the kinematic equations are used in order to predict the next state.
\[ p_t = p_{t-1} + \Delta t v_{t-1} + \frac{1}{2} a \Delta t^2 \]

\[ v_t = v_{t-1} + a \Delta t \]

These equations are then rewritten in matrix form, which gives the following equation:

\[ \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tilde{\mu}_t = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tilde{\mu}_{t-1} \]

\[ \tilde{\mu}_t = A_t \tilde{\mu}_{t-1} \]  \hspace{1cm} (4)

This is a prediction matrix which provides the next state from the previous state. In order to update the covariance matrix, the following property of covariance matrices is then applied:

\[ \text{Cov}(x) = \Sigma \]
\[ \text{Cov}(Ax) = A \Sigma A^T \]

Applying this to the covariance of the previous state results in the following equation, which allows the robot to update the state’s covariance matrix.

\[ \Sigma_t = A_t \Sigma_{t-1} A_t^T \]  \hspace{1cm} (5)

Additionally, there are outside factors in the world which can alter the system. This is denoted by the control vector \( u_t \). Assuming that in a scenario, we know acceleration, our \( u_t \) will be denoted by \( a \). By converting the kinematic equations into matrix form, the following is obtained:

\[ \begin{bmatrix} \Delta t^2 \\ \Delta t \end{bmatrix} a \]

\[ \begin{bmatrix} 1 \\ 0 \end{bmatrix} \]

\[ \tilde{\mu}_t = A_t \tilde{\mu}_{t-1} + B_t u_t \]  \hspace{1cm} (6)

The control variable serves as a correction factor, which allows for more complex models. In the scenario of Robot Soccer however, simpler models are favored and the control vector \( u_t \) will simply be 0.

When modelling the real world, everything behaves much more randomly than it would in a theoretical model. As a result, a variable for noise must be included in the equations in order to model uncertainty in any given state. Each state exists within a region of uncertainty, with covariance \( R_t \). In other words, the covariance matrix of the noise will be denoted by the variable \( R_t \).

\[ \tilde{\mu}_t = A_t \tilde{\mu}_{t-1} + B_t u_t + z_t \]  \hspace{1cm} (7)

\[ \Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t \]
One way to improve state estimate is by gathering information about a given environment through measurements. Measurements are taken by a robot’s sensors, for example. Thus, sensors are modelled with the matrix $C_t$. Since there is also noise in the measurements, it will be taken into account in the equation that determines the measurement and will be denoted by $\delta_t$.

\[
\begin{align*}
  z_{expected} &= C_t \mu + \delta_t & (8) \\
  \Sigma_{expected} &= C_t \Sigma_t C_t^T & (9)
\end{align*}
\]

Furthermore, sensors have noise. The covariance of the uncertainty (the sensor noise) will be modelled with the variable $Q_t$ and the value of the observed information will be denoted by $z_t$.

Based on these observations, the state predictions must be reinterpreted with the added knowledge of the measurement data. This provides two probabilistic sources of information, one being observed, and one being based on the robot’s prediction of the state. The best possible prediction is one where both the observation and believed state are in agreement, but that will rarely be the case in practice. In Kalman filter modelling, the probabilistic region where the two overlap is the best estimate of where the true state is. In order to model this best estimate, the two probabilities are fused together in the Kalman filter algorithm.

Specifically, the two sources of information which are being fused together are represented by Gaussian distributions. To illustrate the idea, consider a simple system with a one dimensional state. A single Gaussian distribution has the form:

\[
N(x, \mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}
\]

When multiplying two Gaussian curves together, the resulting curve lies in the region where the two overlap:

![Graph showing the overlap of two Gaussian distributions](image)
\[ N(x_o, \mu_o, \sigma_o) \ast N(x_1, \mu_1, \sigma_1) = N(x', \mu', \sigma') \]  

(10)

After several tedious algebraic expansions and manipulations, the equations for the fused Gaussian distributions take on the following form:

\[ \mu' = \mu_o + \frac{\sigma_o^2 (\mu_1 - \mu_o)}{\sigma_o^2 + \sigma_1^2} \]  

(11)

\[ \sigma'^2 = \sigma_o^2 - \frac{\sigma_o^4}{\sigma_o^2 + \sigma_1^2} \]  

(12)

They can both be further simplified by factoring out a variable denoted by \( K \), which is also known as the Kalman gain:

\[ K = \frac{\sigma_o^2}{\sigma_o^2 + \sigma_1^2} \]  

(13)

Thus, the sources of information about the state (the dynamics and the observations), are fused optimally by a weighed sum; the optimal weights being provided by the Kalman gain. The simplified equations take on the form:

\[ \mu' = \mu_o + K(\mu_1 - \mu_o) \]  

(14)

\[ \sigma'^2 = \sigma^2 - K\sigma^2 \]  

(15)

In the case where the state is a vector, the Kalman Gain again “specifies the degree to which the update measurement is incorporated into the new state estimate” (Thrun 2005). In other words, it states the importance of the observed measurement, and will use it to alter the prediction from the dynamics according to their relative importance. For example, a large Kalman gain will increase the importance of the observed value, as shown below. The fusion of these equations in matrix form are as follows:

\[ K = \Sigma(\Sigma + Q_t)^{-1} \]  

(16)

\[ \mu_t = \bar{\mu}_t + K(z_t - z_{\text{expected}}) \]  

(17)

\[ \Sigma_t = \Sigma_t - K\Sigma_{\text{expected}} \]  

(18)

Now that all the necessary components to the Kalman filter have been derived, they can all be put together into one cohesive algorithm. By using the equations for the predicted measurement \( (z_{\text{expected}} = C_t\bar{\mu}_t, \Sigma_{\text{expected}} = C_t\Sigma_tC_t^T) \), and applying them to the equations above, the following equation is derived which models the covariance and mean at time \( t \):

\[ C_t\mu_t = C_t\bar{\mu}_t + K(z_t - C_t\bar{\mu}_t) \]  

(19)
\[ C_t \Sigma_t C_t^T = C_t \Sigma_t C_t^T - K_t \Sigma_t C_t^T \]  
(20)

Furthermore, substituting them into equation 16 provides a formula for the Kalman gain:

\[ K = C_t \Sigma_t C_t^T (C_t \Sigma_t C_t^T + Q_t)^{-1} \]  
(21)

From the equations 19 and 20 which were just derived, a \( C_t \) and a \( C_t^T \) can be removed to simplify the equations. This results in the final equations necessary for the Kalman Filter Algorithm:

\[ \mu_t = \overline{\mu}_t + K (z_t - C_t \overline{\mu}_t) \]  
(22)

\[ \Sigma_t = \Sigma_t - K_t \Sigma_t C_t \Rightarrow \Sigma_t = \Sigma_t (I - K C_t) \]  
(23)

\[ K_t = \Sigma_t C_t^T (C_t \Sigma_t C_t^T + Q_t)^{-1} \]  
(24)

Thus, the Kalman Filter Algorithm is:

1. \( \overline{\mu}_t = A_t \mu_{t-1} + B_t u_t \)
2. \( \Sigma_t = A_t \Sigma_{t-1} A_t^T + R_t \)
3. \( K_t = \Sigma_t C_t^T (C_t \Sigma_t C_t^T + Q_t)^{-1} \)
4. \( \mu_t = \overline{\mu}_t + K (z_t - C_t \overline{\mu}_t) \)
5. \( \Sigma_t = \Sigma_t (I - K C_t) \)

This is a very computationally efficient version of the general Bayesian algorithm based on the Gaussian assumptions of the distributions of state and observations.

4.2 Application of the Kalman Filter on Robot Soccer

The robots in RoboCup utilize the Kalman filter exactly as it was described above. However, the variables and matrices described above have specific values which are needed in order for the robot to model the state of the ball. In practice, a robot has to model the state of the ball, as well as all other robots and its own state. For simplicity, only the state of the ball will be modelled in this paper. If there were no noise, the ball’s motion would follow Newtonian laws, where the predictions made by the following kinematic equations are respected.

\[ x(t) = x_o + v_x^r t, \quad v_x^r(t) = v_x^r(t) \]
Instead, the true dynamics are computed with the equations above including noise sampled from the multivariate Gaussian. The simulations use a Kalman filter to compare its predicted distributions to the true state of the position and velocity of the ball. To do so, the initial state is assumed to be correct with a starting x,y position of (100, 80) and a starting x,y velocity of (-2,-1). This gives the following initial state vector:

\[
\begin{bmatrix}
100 \\
80 \\
-2 \\
-1
\end{bmatrix}
\]

4.2.1 The Variables of the Kalman Filter

The variable matrices of the Kalman filter algorithm will now be defined in relation to robot soccer. The robot models the prediction state \( \mu_t \) with a 4x1 matrix, which takes into account the x and y coordinates of the ball on the field, as well as the x and y directional velocities of the ball at time t:

\[
\mu_t = \begin{bmatrix} x_t \\ y_t \\ v_y^t \\ v_x^t \end{bmatrix}
\]

The robot also models the prediction step matrix \( A_t \) in order to model the state of the ball at the next time step:

\[
A_t = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]

The predicted mean is then updated in discrete time steps, dt, using the prediction matrix

\[
\begin{bmatrix} x_{t+1} \\ y_{t+1} \\ v_y^{t+1} \\ v_x^{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 & dt & 0 \\ 0 & 1 & 0 & dt \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ v_y^t \\ v_x^t \end{bmatrix} + \epsilon_t
\]

Similarly, a covariance matrix of the state is necessary. The initial covariance matrix is:

\[
\Sigma_0 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}
\]
The covariance matrix is essentially the 4x4 identity matrix $I$. It is also constantly being updated in time steps, $dt$, with $A_t$:

$$
\Sigma_{t+1} = \begin{bmatrix}
1 & 0 & dt & 0 \\
0 & 1 & 0 & dt \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix}
+ R_t
$$

(29)

If the robots were also modeling acceleration, control data would need to be taken into account. The transformation matrix which updates the control data is denoted by $B_t$, a 4 x 4 matrix; however due to the level of modeling found in this paper the control matrix $B_t$ is not needed. This is because, as mentioned before, computations need to be very quick in order to keep track of the position and velocity of the ball. The robots do not need to take into account acceleration because of the low ball speeds they are working with, and the necessity to keep the algorithm as simple as possible.

In order to model the observed results, $z_t$, another matrix is required. This observation matrix $C_t$ is as follows:

$$
C_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}
$$

(30)

This implies that the x- and y-coordinates are observed. In order to get the measurement data $z_t$, the prediction state matrix $\tilde{\mu}_t$ is multiplied by the observation matrix $C_t$, with added noise $\delta$:

$$
\begin{bmatrix}
z^x_t \\
z^y_t
\end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}
\begin{bmatrix} x_t \\
y_t \\
v^x_t \\
v^y_t
\end{bmatrix} + \begin{bmatrix} \delta_1 \\
\delta_2
\end{bmatrix}
$$

(31)

The covariance matrix of the measurement noise was created using arbitrary numbers that are believed to be realistic.

$$
Q_t = \begin{bmatrix} 0.1 & 0.001 \\ 0.001 & 0.1 \end{bmatrix}
$$

Lastly, the model also required a covariance matrix of the state noise, which was generated using intuitively correct values, just as it was done for $Q_t$.

$$
R_t = \begin{bmatrix} 0.0003 & 0.0001 & 0.00002 & 0 \\ 0.0001 & 0.0002 & 0 & 0.00002 \\ 0.00002 & 0 & 0.0003 & 0.0001 \\ 0 & 0.00002 & 0.0001 & 0.0002
\end{bmatrix}
$$

In this experiment, there were 200 timesteps, each being recorded 0.1 seconds apart ($dt = 0.1$).
4.2.2 The Modelling of Robot Soccer Using Python Code

By running the simulation in Python, the actual trajectory was determined by sampling the noise according to the model. The simulation generated the following four graphs which model the actual trajectory of the ball with noise.

The top left graph models x-coordinate, the top right graph models y-coordinate, the bottom left models x-velocity, and bottom right models y-velocity of the ball given time. On the coordinate graphs, the red dots are the actual observations of the robots. The velocity graphs are zoomed in to a very large scale, and show a very significant change in velocity, however there is little variation in velocity in the actual model. The velocity can actually be considered constant. As shown by the graphs, the velocity starts at -2.00 m/s in the x direction, and 20 seconds later has not changed significantly, being only -2.10 m/s. Similarly to the y-velocity, which starts at -1.00 m/s and is at approximately -0.95 m/s 20 seconds later.

Next, the actual x- and y-coordinates were plotted as a function of time, as well as the predictions computed by the Kalman Filter of these two variables, on the same graph.
The belief distance is shown in red and is compared to the true distance of the ball in blue. The graph on the left models the x-coordinate of the ball given time, and the graph on the right models the y-coordinate of the ball given time. The belief distance has very little deviation from the true distance in both graphs, thus demonstrating the strength and accuracy of the Kalman filter algorithm.

The following two graphs model the belief velocity of the ball using the Kalman filter algorithm that is being applied by the robot. The graphs show the belief velocity (red) versus the true velocity of the ball (blue).

The graph on the right models the y-velocity given time, whereas the left models x-velocity given time, and both show the accuracy of the filter. The belief velocity is essentially the true velocity, shifted to the right by approximately 1.5 seconds. This is because the velocities are not observed directly, but rather estimated from the dynamics and the observed positions. The observed estimation delay mimics the reaction time observed in humans, which is 0.1 seconds.

Next, the true trajectory of the ball was plotted in the x-y plane, as well as the Kalman Filter model’s belief trajectory of the ball.
This graph compares the belief trajectory (in terms of x and y position) determined by the Kalman filter (in red) to the true trajectory of the ball (in blue). As mentioned earlier, the graphs are practically identical as it is difficult to distinguish a difference between the two. The initial belief of the state along with the arbitrarily chosen covariance matrices for the noise appear to be modelled with realistic values as the graphs model the distance and the velocity of the ball very accurately.

Finally, the true x- and y-coordinates were plotted as functions of time versus the straight lines predicted by the Newtonian model.

These two graphs model the theoretical distance based on the kinematic equations (in green) vs the true distance (in blue). By assuming the acceleration is 0, the theoretical distance represented in the graphs by the green lines will be perfectly straight. This model does not take into account the randomness of the environment. In reality though, the ball’s motion cannot continuously respect the kinematic laws of motion because there is noise in the environment. From the graph it becomes noticeable that the line representing the actual trajectory of the ball (blue) deviates from the straight line that represents the theoretical trajectory. This is due to the fact that the blue line also takes into account the imperfections of the ground, which consequently causes the ball to travel...
less distance. Theoretically in a perfect world where the ground is perfectly flat and without friction, the ball would travel without any deviation in its pass, thus allowing it to maintain a constant speed. In reality, the noise from the environment leads to significant deviations from the straight line behaviour.

5 Conclusion

The Kalman filter is a mathematical algorithm which has become increasingly useful in the modern day and age. It has endless applications and is an essential component of modern robotics. It allows robots to model real world events with incredible accuracy. While the Kalman filter was only described in the linear case in this paper, it can also be extended to the non-linear case, which is the Extended Kalman Filter. Human understanding of artificial intelligence not only leads to the creation of more efficient and practical appliances, but it also pushes forward the human understanding of the brain. It is widely believed that the human brain follows a Bayesian algorithm, as people make decisions based on what they believe is the current situation, on what they observe, and based on past events similar to the current one. Further research in artificial intelligence can not only allow for practical creations, such as self-driving cars, but also pushes forward the field of cognitive science and the understanding of the human brain.
Works Cited


Schulz, Hannes. Line Structure-Based Localization for Soccer Robots. 7 Dec. 2009, pdfs.semanticscholar.org/1c5b/55bc412e91b725ff06c6b6134c9908dfeeb0.pdf.


