RESEARCH

A Recursive Kinematic Random Forest and Alpha Beta Filter Classifier for 2D Radar Tracks

Lars W. Jochumsen[1,2*], Jan Østergaard[2], Søren H. Jensen[2], Carmine Clemente[3†] and Morten Ø Pedersen[1]

[*]Correspondence: lwj@terma.com
[1]Terma A/S, Hovmarken 4, Lystrup, Denmark
Full list of author information is available at the end of the article
[†]This work was partially supported by the Engineering and Physical Sciences Research Council (EPSRC) Grant number EP/K014307/1 and the MOD University Defence Research Collaboration in Signal Processing.

Abstract

In this work, we show that by using a recursive random forest together with an alpha beta filter classifier it is possible to classify radar tracks from the tracks' kinematic data. The kinematic data is from a 2D scanning radar without Doppler or height information. We use random forest as this classifier implicit handles the uncertainty in the position measurements. As stationary targets can have an apparently high speed because of the measurement uncertainty, we use an alpha beta filter classifier to classify stationary targets from moving targets. We show an overall classification rate from simulated data at 82.6 % and from real world data 79.7 %. Additional to the confusion matrix we also show recordings of real world data.

Keywords: Radar; Classification; Random forest; Alpha beta filter; Kinematic

1 Introduction

The increasing demand for protection and surveillance of the coastal areas requires modern coastal surveillance radars. These radars are designed such that small objects can be detected. Therefore, there is an increasing amount of information for the radar observer. Moreover, the number of false and unwanted objects increases as the demand for seeing small objects makes the radar more sensitive. Generally, the false objects can be avoided by using a reliable tracker. However, the tracker does not exclude unwanted objects. The difference between false and unwanted objects are that false objects do not originate from true objects but are mainly noise objects, whereas the unwanted objects originate from true objects but are unwanted in the surveillance image. These objects depend on the purpose of the radar however,

for coastal surveillance radars the unwanted objects are normally birds, wakes from large ships etc.

It has been shown in [1] that it is possible to classify tracks by using a recursive classifier where a Gaussian mixture model (GMM) is used to model the probability distribution function (PDF) of targets kinematic behavior. However the classifier does not handle the uncertainty in the measurements from the radar. In [2] the position uncertainty is used as an input to the classifier. The classifier also use a GMM to model the PDF of the kinematic behavior of the target. The problem with this is that it is very computationally expensive. To obtain an easier way to handle uncertainty, joint target tracking and classification can be used, as shown in [3, 4, 5]. The problem with joint target tracking and classification is that it is difficult to achieve a high degree of freedom in the filters to separate the classes. For example a car driving 130 km/h on highway is not likely to accelerate but more likely to decelerate. This is very hard to model with a tracking filter. A particle filter can be used but this is computationally expensive. In [6] the authors are describing a method to classify trucks and cars from GPS measurements. The classifier consists of a support vector machine (SVM) and the features are primarily acceleration and deceleration. The classifier is non-recursive, which means that the complete length of the tracks is required. The measurements from a GPS device is generally more accurate than the position measurements then a radar. In [7] a decision tree is used for a recursive classification of four different target classes. The data are from a radar with height information. The decision tree has the advantage that it in some way implicitly handles the uncertainty. That is, features that do not separate the classes will not be used as much as features, separating the classes. The disadvantage is that the classifier has a high variance of the classification results. In [8] the random forest classifier is introduced. The random forest is a bagging classifier[9] where multiple decision trees are used to reduce the variance of the classification results. For this reason random forest is selected in this work.

In this work we introduce a classifier which uses position measurements to classify radar tracks from a 2D scanning radar. The classifier consists of a alpha beta filter[10] and a random forest classifier. The alpha beta filter is classifying stationary or moving and the random forest classifies the moving targets. The classify is recursive such that the classification results is being updated for each scan of the

radar. The classifier performance is shown by using simulated track data and real world radar data.

In section 2.1 we will introduce the random forest classifier by describing the training of a decision tree and then explain how this tree is used in the random forest. In section 2.2 we will explain how we utilize the probability estimates from the random forest in a recursive framework. In section 2.3 we introduce a alpha beta filter classifier, which classifies targets as either stationary or moving. This is introduced because stationary targets can have high speeds because of they fluctuate in the position because of measurements uncertainty or the main scatter points is moving i.e. wind turbine. In section 2.4 we combine the random forest and the alpha beta filter to our proposed classifier. In 2.5 we describe, which features we use in the random forest. The simulation study is shown in section 3 and in section 4 the real world results are shown. We discuss the results in section 5 and conclude the work in section 6.

2 Method

When using a random forest, a feature vector is needed. We define our feature vector as a set of kinematic and geographic features. The feature vector is derived from the radar position measurements. We define this set of position measurements as

$$\{Z_n\}_k = \{Z_n \cdots Z_{n-k}\}, \tag{1}$$

where $Z_n = [x_n, y_n]^T$, x and y is the position in a Cartesian coordinate system with the origin at the location of the radar, n is the measurement number index and k is the set size.

2.1 Random forest

In this section, we introduce the random forest classifier [8, 11]. Random forest is a bagging algorithm, which means that the random forest consists of a number of weak classifiers [12], which has zero bias, but high variance of the true value. The weak classifiers are decision tress [9]. We start this section by describing how to grow a decision tree and then move on to the random forest.

A decision tree consists of a number of nodes e.g. $(N_1 \cdots N_3)$ and a number of leafs e.g. $(N_4 \cdots N_7)$. This is shown in Fig. 1. A node is defined by more than one

class existing in the node data, whereas a leaf only has one class. In every node a decision must be made such that we either go left or right in the tree. The decision must always be true or false. A leaf is defined as a node where all of the data in the node only consists of one class therefore no more splits are required.

To train the tree we start with a feature vector F of size $N_s \times D$ where N_s is the number of samples and D is the number of features i.e. dimensions in the feature vector. We now want to split the data such that we make the best separation of the classes by choosing the best feature and feature value. To do this we need to find the best feature to split and the best value to split at. To explain the algorithm we assume that there are only 2 classes so it forms a binary classification problem and that the values of the feature belong to a finite sample space. This is done to make the explanation easier.

We start by assuming that a split already has been made and we want to evaluate how good the split is. For this, we use a normalized the entropy measure to do that[12]. An alternative to the normalized entropy is the more common Gini index [13] however, for this work the normalized entropy as shown better results. We define the set of samples in the parent node as s_1 and the number of samples in the set as $|s_1|$. Similarly we define the set of samples in the children as s_2 and s_3 and the number of samples as $|s_2|$ and $|s_3|$. Further we index the samples belonging to class ℓ by the superscript ℓ such as s_1^ℓ, where $\ell \in \{1, 2\}$. We can calculate the empirical entropy for the children as

$$H(s_i) = -P(s_i^1)\log_2(P(s_i^1)) - P(s_i^2)\log_2(P(s_i^2)), i = \{2, 3\}, \tag{2}$$

where $P(s_i^1) = |s_i^1|/|s_i|$ and $P(s_i^2) = |s_i^2|/|s_i|$. It follows that $P(s_i^1) = 1 - P(s_i^2)$. As the entropy does not take into account how many samples there are in each child we normalize the entropy as

$$\hat{H}(s_i) = \frac{|s_i|}{|s_1|}H(s_i), i = \{2, 3\}. \tag{3}$$

We can now calculate the information gain from the split as

$$\tilde{H} = H(s_1) - (\hat{H}(s_2) + \hat{H}(s_3)). \tag{4}$$

From (4) we now have a measure for how good a split is, and now able to optimize each split of the data such that we choose the best feature to split on and the best value of the feature. We split the data and continue to split the data until all data in a node is of the same class i.e. the node becomes a leaf. To prevent over fitting a decision tree must be prone. However, an advantage of using random forest is that it is not necessary to prune the decision trees. The random forest is a bagging classifier[9]. This means that the random forest consist of a number of trees N_t where each tree is trained with a random part of the samples and a random part of the features. That is, we draw a random subset of the training data and select a random subset of the features. We then train each tree with these random subsets and we assume that the trees are statically independent of each other. A decision tree classifies the data by following a path through each node. The path is decided by the feature and feature value that made the best split in the training. The data which must be classified follow the path until a leaf is met. The leaf has a unique class and the data is classified as this class. The classification of the data is a majority vote of the result from each of the individual decision trees. That is each tree is a unique classifier which classifier the data individual.

In general the random forest is not a probabilistic classifier but a majority vote between each of the tress. However, by counting the votes for each class and normalizing with the total number of trees an empirical probability can be achieved.

$$\hat{P}(c_i|\{Z_n\}_k) = \psi_i/N_t, \tag{5}$$

where ψ_i is the the number of votes for class i.

In the next section we explain how we (5) obtained from the random forest to achieve a recursive update of the probability for the class given all the measurements.

2.2 Recursive update of the random forest probability

The empirical probabilities obtained from the random forest classifier are obtained as the fraction of the number of trees which predicts c_i divided with the total number of trees. By this definition, the resolution of the probability estimates is given by the number of trees in the random forest. To prevent that a class is assigned a zero

probability, we modify it in the following way:

$$P(c_i|\{Z_n\}_k) = \frac{\hat{P}(c_i|\{Z_n\}_k)(1-2/N_t)+1/N_t}{\gamma}, \tag{6}$$

where γ is a normalization constant such that such that $\sum_i P(c_i|\{Z_n\}_k) = 1$. By this formula the probability never reaches zero for any of the classes.

Based upon the above, we have the probability for the class given the current set of features $P(c_i|\{Z_n\}_k)$. However we want the probability given all measurements, that is $P(c_i|\{Z_n\})$, where $\{Z_n\} = \{Z_n\}_n$. We have, however, not been able to find a simple way to recursively update $P(c_i|\{Z_n\})$ based on the previous $P(c_i|\{Z_{n-1}\})$ and which works for all n. Instead we propose the following recursive function $f(c_i|\{Z_n\})$, which is everywhere non-negative and sum to one. Thus, $f(c_i|\{Z_n\})$ can be considered to be a probability mass function (PMF), which we will use as an approximation for the true $P(c_i|\{Z_n\})$. In particular, we define:

$$f(\{Z_n\}_k, c_i) \triangleq \frac{P(c_i|\{Z_n\}_k)^w}{\phi_n} f(\{Z_{n-1}\}_k, c_i), \tag{7}$$

where w is a weighting factor, $P(c_i|\{Z_n\}_k)$ is given by (6) and where ϕ_n is the normalization constant such that $\sum_{c_i} f(\{Z_n\}_k = 1$. The introduction of the weighting by w is inspired by the weighted Bayesian classifier used in [14]. In particular, we choose $w = 1/k$ since the features of the random forest is given by a set of measurements where only one out of k measurements is substituted at each update.

In the next section we describe our alpha beta tracking filter. This filter is used to classify if a target is non moving or moving. The reason for applying such a filter is to classify stationary targets, which have high apparent speed due to measurement uncertainties.

2.3 Alpha beta filter

The alpha beta filter is a simple tracking filter [15]. By using the alpha beta filter, we assume that we can describe the target movements with a first order Markov chain. We have the state vector $X_n = [\hat{x}, \hat{y}]^T$ and the measurement Z_n. The alpha beta filter is trying to predict Z_n given the speed V_{n-1} at time $n-1$ and the state X_{n-1} as

$$X_n^- = X_{n-1}^+ + \tau V_{n-1}^+, \tag{8}$$

where τ is the time between Z_{n-1} and Z_n and the superscript $-$ is the prediction before the measurement are used and the superscript $+$ is after the measurement is used. The filter Assumes the speed is constant between n and $n-1$ that is $V_n = V_{n-1}$. The error can be calculated as

$$R_n = Z_n - X_n^-, \tag{9}$$

with the residual we update the estimate of the V_n^- and X_n^- as

$$\begin{aligned} X_n^+ &= X_n^- + \alpha R_n \\ V_n^+ &= V_n^- + \frac{\beta}{\tau} R_n, \end{aligned} \tag{10}$$

where α and β are the constants in the alpha beta filter. To calculate the probability for Z_n given X_n^-, α and β we use a multivariate normal distribution

$$P_{\alpha\beta}(Z_n|X_n^-, \alpha, \beta) = \frac{1}{2\pi\sqrt{|\Sigma_n|}} \exp\left(-\frac{1}{2}(Z_n - X_n^-)^T \Sigma_n^{-1}(Z_n - X_n^-)\right) \tag{11}$$

where Σ_n is the covariance of the position, and the subscript $\alpha\beta$ is to emphasize that this is the probability for the alpha beta filter. The purpose of the alpha beta filter is to separate nonmoving targets i.e. stationary targets from moving targets. We therefore define two filters: a stationary filter with the parameters $\alpha = 0.1$ and $\beta = 0.0$, which allows the position part of the state to move slightly but force the speed to be constant at zero. The possibility for a slight movement of the state is because of the possibility for false starting measurements. As the parameters α and β is given of the class c_s we use the notation $P_{\alpha\beta}(Z_n|X_n^- c_s)$. Likewise We define the moving alpha beta filter as $P_{\alpha\beta}(Z_n|c_m, X_n^-)$ with the parameters $\alpha = 1.0$ and $\beta = 1.0$ i.e. we hold the speed constant from update to update but allow both the movement and the speed to change with the measured change. If we know $\{Z_{n-1}\}$ which is the set measurement up to $n-1$ and α and β we can calculate X_n^- we can therefore write $P_{\alpha\beta}(Z_n|c_i, \{Z_{n-1}\})$ instead of $P_{\alpha\beta}(Z_n|c_i, X_n^-)$. For this work we want the alpha beta filter to classify if the target is stationary or non-stationary, we therefore recursively update the probability of the alpha beta filter.

$$P_{\alpha\beta}(c_i|\{Z_n\}) = \frac{P_{\alpha\beta}(Z_n|c_i, \{Z_{n-1}\}) P_{\alpha\beta}(c_i|\{Z_{n-1}\})}{P_{\alpha\beta}(Z_n|\{Z_{n-1}\})}. \tag{12}$$

To reduce the computational complexity we assume that the positions are controlled by a first order Markov chain i.e. $Z_n \leftrightarrow Z_{n-1} \leftrightarrow \{Z_{n-2}\}, \forall n$.[1]

$$P_{\alpha\beta}(c_i|\{Z_n\}) = \frac{P_{\alpha\beta}(Z_n|c_i, Z_{n-1})P_{\alpha\beta}(c_i|\{Z_{n-1}\})}{P_{\alpha\beta}(Z_n|Z_{n-1})}, \tag{13}$$

In the next section we describe how we combine the random forest classifiers and the alpha beta filter classifier such that a classifier, which is a combination of the two classifier are created.

2.4 Combining the alpha beta filter with random forest

In our work we let the alpha beta filter classify if the target is stationary or non-stationary i.e. the alpha beta filter has two classes. The random forest has a stationary class and multiple non-stationary classes. We define for the random forest c_0 to be the stationary class and $c_{1\cdots}n_C$ to be the moving classes, where n_C is the total number of classes. For the alpha beta filter we have the two classes as c_s and c_m for stationary and non-stationary classes respectively. We want the alpha beta filter classifier to have a larger weight on the classification result of stationary vs. moving then the random forest. We therefore use the recursive updated probability from (13). We do this as described in(14), (15).

$$\hat{P}(c_0|\{Z_n\}) = f(\{Z_n\}_k, c_0)P_{\alpha\beta}(c_s|\{Z_n\}), \tag{14}$$

$$\hat{P}(c_i|\{Z_n\}) = f(\{Z_n\}_k, c_i)P_{\alpha\beta}(c_m|\{Z_n\}), i = 1\cdots n_C \tag{15}$$

We then normalize $\hat{P}(c_i|\{Z_n\})$ as

$$P_c(c_i|\{Z_n\}) = \frac{\hat{P}(c_i|\{Z_n\})}{\hat{\omega}}, \tag{16}$$

where $\hat{\omega}$, is a constant such that $\sum_i P_c(c_i|\{Z_n\}) = 1$. By including the alpha beta filter in this manner, we ensure that the alpha beta filter, classifies if a target is stationary while the alpha beta filter classifier do not have influence on the different moving classes.

[1] We denote the Markov chain by $a \leftrightarrow b \leftrightarrow c$, such that a is statistically independent of c if we know b.

In the next section we will describe the features we use for the random forest feature vector, we will also describe how these are derived from the position. We only utilize position dependent features such as speed, acceleration etc.

2.5 Features

For the feature vector, we draw inspiration from [16] for some of the features. In this work, we set the number of position measurements k in (1) to 10. The number of measurements used in the feature vector is a compromise between the time it takes to get the number of measurements required for a full feature vector and the amount of information contained in the feature vector. Larger k requires more measurements i.e. more time before a classification results is made whereas for smaller k the first classification result comes earlier albeit with a greeter uncertainty due to the smaller amount of available information. The features and their descriptions can be seen in Table 1. Remembering we defined $\{Z_n\}_k$ to be $\{Z_n \cdots Z_{n-k}\}$. To make the notation easier we index each measurement in $\{Z_n\}_k$ by i such that i represent the i'th element in the set of measurements $\{Z_n\}_k$, that is $0 \leq i < k$. Likewise we define the set of time stamps of the measurements as $\{t_n\}_k$ with the individual measurement being observed at time t_i. We start by calculating the vectorial distance between the measurements as:

$$\delta_i = Z_i - Z_{i-1}, \tag{17}$$

with the scalar distance given by

$$\Delta_i = |Z_i - Z_{i-1}|, \tag{18}$$

and the time difference between the measurements as

$$\tau_i = t_i - t_{i-1}. \tag{19}$$

The 2-point velocity estimate is

$$v_i = \frac{\Delta_i}{\tau_i}, \tag{20}$$

for $1 \leq i < k$ and the 3-point acceleration estimate is

$$a_i = \frac{2(v_{i+1} - v_i)}{\tau_{i+1} + \tau_{i-1}}, \tag{21}$$

for $1 \leq i < (k-1)$. The normal acceleration $a_i^{\perp}$ is given by the product of the speed and angular velocity

$$a_i^{\perp} = \left(\frac{v_{i+1} + v_i}{2}\right)\left(\frac{2}{t_{i+1} - t_{i-1}}\right)\cos^{-1}\left(\frac{\delta_{i+1} \cdot \delta_i}{\Delta_{i+1}\Delta_i}\right). \tag{22}$$

We also use land/sea as information These can be extracted from the SWBD database from [17]. The database is a set of polygons describing the coastline. Because of errors in the database a hard threshold cannot be used for land and sea. We therefore proposed to use the distance to the coastline d_i for each measurement as a feature. By using these polygons it is possible to calculate the distance from a measurement to the coastline. However it is getting more and more computational expensive to calculate the distance as the distance to the nearest coastline increases. We therefore assign a maximum distance ξ to the coastline from the target. If the target is farther away then ξ we assign ξ to the distance. the sign of the distance decide if it is over land or sea. We set $\xi = 700$ meters to accommodate for errors in the SWBD database.

In the next section we will show some simulation results of the classifier. We will also show some real world results of the classifier.

3 Simulation study

We start by showing the performance of the algorithm versus the number of measurements k which the extracted features is from. The size of the feature vector change by k and the table shown in Table 1 for $k = 10$. The data we use are simulated data from a controlled random walk. The controlled random walk consist of a three state transition matrix which has a deceleration, steady state and acceleration state. Parameters for maximum and minimum speed are incorporated which changes the probability in the transition matrix if the speed is not within the boundary of the permitted speed range. The data for different targets are generated such that they have nearly the same support in speed and the main difference is the acceleration support. The random walk creates position p_m^x and p_m^y which are

extrapolated from some smooths speeds $\hat{v}_m^x$ and $\hat{v}_m^y$ described later.

$$p_m^x = p_{m-1}^x + \Delta t \hat{v}_m^x + \Sigma_m^x \tag{23}$$

$$p_m^y = p_{m-1}^y + \Delta t \hat{v}_m^y + \Sigma_m^y, \tag{24}$$

where Δt is the time between the updates for m and $m-1$ and Σ_m^x and Σ_m^y are position uncertainty drawn from a distribution.

$$\begin{bmatrix} \Sigma_m^x \\ \Sigma_m^y \end{bmatrix} \sim \mathcal{N}(0, \Sigma_e), \tag{25}$$

where Σ_e is the position covariance and $\mathcal{N}$ denotes the normal distribution. The smooth speeds are speeds v_m^x and v_m^y which are convolved with a 25 tap moving average filter h. This is done to avoid to quick changes in the speed.

$$\hat{v}_m^x = h * v_m^x \tag{26}$$

$$\hat{v}_m^y = h * v_m^y. \tag{27}$$

The speeds (27) are extrapolated from accelerations $a_j^x(m)$ and $a_j^y(m)$, where j denotes the depending upon the state j described in (32). The speeds are given as

$$\begin{bmatrix} v_m^x \\ v_m^y \end{bmatrix} = \begin{bmatrix} v_{m-1}^x + \Delta t o_j^x(m) \\ v_{m-1}^y + \Delta t o_j^y(m), \end{bmatrix} \tag{28}$$

where $o_j^x(m)$ and $o_j^y(m)$ are accelerations which is drawn from two normal distributions given by

$$o_j^x(m) \sim \mathcal{N}(\mu_{x,j}, \sigma_{x,j}^2) \tag{29}$$

$$o_j^y(m) \sim \mathcal{N}(\mu_{y,j}, \sigma_{y,j}^y) \tag{30}$$

The parameters for the normal distribution $\mu_{x,j}$,$\mu_{y,j}$,$\sigma_{x,j}^2$ and $\sigma_{y,j}^2$ are given from the function $\phi_j(v(m-1), \Gamma)$. This is done because we want to control the maximum and minimum allow speed. We define this function as:

$$\phi_j(v(m-1), \Gamma) = \begin{cases} \psi_j(1) & \text{if } v_{m-1} > \zeta^{max}, \\ \psi_j(2) & \text{if } \zeta^{min} \leq v_{m-1} \leq \zeta^{max}, \\ \psi_j(3) & \text{else,} \end{cases} \tag{31}$$

where $\psi_j(1)$, $\psi_j(2)$ and $\psi_j(3)$ is the set of parameters $\{\mu_{y,j}\mu_{x,j}, \sigma^2_{y,j}, \sigma^2_{x,j}\}$ used in (30) and $\Gamma = \{\zeta^{min}, \zeta^{max}\}$. The state machine consists of three states: deceleration (d), constant (c) and acceleration (a) states, see Fig. 2. Further the state machine is also controlled by the speed. We define the state transition probabilities as:

$$P_{\hat{j},j}(v(m-1),\Gamma) = \begin{cases} \Psi_{\hat{j},j}(1) & \text{if } v_{m-1} > \zeta^{max}, \\ \Psi_{\hat{j},j}(2) & \text{if } \zeta^{min} \leq v_{m-1} \leq \zeta^{max}\ , \\ \Psi_{\hat{j},j}(3) & \text{else,} \end{cases} \tag{32}$$

where $\hat{j}$ is the previous state and $\Psi_{\hat{j},j}$ is the transition probability. An example of a track can be seen in Fig. 3 The speed PDFs can be seen in Fig. 4 and the accelerations PDF can be seen in Fig. 5. The performance of the classifier versus the number of measurement k can be seen in Fig. 6. Further we show the performance of the classifier vs. the number of trees N_t used in the random forest, see Fig. 7. The confusion matrix of the classification results for the four classes can be seen in Table 2, where we have used $k = 10$ and $N_t = 100$.

4 Real world results

The data used for this work consist of Automatic Identification System (AIS), which is a broadcast system used for large ships, Automatic Dependent Surveillance-Broadcast (ADS-B) which is a broadcast system used for commercial aircrafts, GPS logs and real world radar data. The classes for this work is typically classes for coastal surveillance e.g. large ships, birds, small boats etc.

We show a confusion matrix for real world data in Table 3. As a confusion matrix does not take into account how the probability develops over time we also show some real world scenarios. For these scenarios extra classes are used. The scenarios are images showing all tracks within a specific time period. The scenarios have both known and unknown targets. It is therefore not possible to make a confusion matrix of the scenario however, it is possible to have a good estimate of the performance of the classifier in real world situations. The scenarios are recorded with different radars and antennas, further the sampling rate can be different for the different scenarios. We show two scenarios from coastal surveillance applications. The first coastal surveillance scenario is recorded in Denmark where a rigid inflatable boat (RIB) is sailing from west to east and zigzagging back. Towards the north of the

RIB there are two unknown vessels, further there are some sea buoys present both to the north of the RIB but also to the far south. The rest of the tracks are believed to be bird. See the scenario at Fig. 8. The second scenario is also from Denmark and shows two wind turbines farms. A commercial plane is flying in from the west to the east and a small personal aircraft is circling over first the wind farm to the north then the second wind farm and finally leaving towards the east. Three vessels is present one to the east of the wind farm in the north (above the other wind farm) the second vessel is sailing through the wind farm in the south. The last vessel is sailing from west to east under the south wind farm. The rest of the tracks are believed to be birds, see Fig. 9. As the majority of previously published results are based on a joint tracking and classification approach, mostly on simulated data, it is not directly possible to compare the obtained classification accuracy.

In the next section we will discus the results of the classifier.

5 Discussion

In Fig. 6 the performance of the classification results for the simulated data set is shown, where we vary the number of measurements k, in (1), used to extract the features. The performance is calculated as the mean of the diagonal in the confusion matrix. It is clear the more measurement (longer feature vector) used the better the classification results. This is clear as more information to the classifier gives better estimation of the class and therefore it is more likely to classify correct. The downside of increasing the number of measurements is that it takes longer time from a track is seen until the first probability of the target is shown. For our results, the sampling rate varies between 0.333 to 1 Hz. For 10 measurements this gives, a maximum waiting time of 30 seconds, which we believe for the application in hand, is acceptable. In Fig. 7 the performance can seen when varying the number of trees used in the random forest. The plot is made with $k = 10$. It can be seen that the performance does not get better after around 170 trees. The increase the number of trees take longer time to train the random forest and is more computational expansive and memory requiring when using the classifier for testing i.e. the purpose of the classifier is to run in real time. The performance of $k = 10$ and $n_t = 100$ can be seen in Table 2. It is clear that type 2 and type 3 has the most confusion between them. This is also natural if we look at the speed PDF's and the acceleration PDF's

in Fig. 4 and 5 respectively as these is very similar. In general the of diagonal numbers in the confusion matrix is at the left side. This is due to the fact the large allowed acceleration still contains smaller acceleration which therefore will be classified as a lower class type.

For the real world scenarios we use $k = 10$ and $n_t = 170$. As it can be seen the confusion matrix in Table 3 shows relative good performance. Nearly all of the stationary sea targets and commercial aircrafts are classified correct. The helicopters are confused with birds. This can be because of the helicopters can move as slow as birds. The are some confusion between large ships, birds and RIBs. All of these classes has kinematics which are close to each other.

In Fig. 8 one of the real coastal surveillance scenario is shown. The scenario shows a RIB sailing out from a marina and zigzagging back again. The RIB is classified as a small fast boat. The reason that it is not classified as a jetski/RIB is that it sails more like a fast boat whereas jetski/RIB often makes turns, accelerate and decelerate. The two slow moving vessels to the north of the RIB is classified correctly. Some of the sea buoys are classified correct as stationary targets. Only a few birds are classified correctly. In Fig. 9 two wind farms can be seen and nearly all of the wind turbines is classified as stationary, while a few are misclassified as small slow moving boats. The commercial aircraft is between commercial aircraft and small aircraft, however the target is primary classified as commercial aircraft. The small aircraft circling the two wind farms is classified correctly even though the aircraft is flying below stall speed. This can be due to the strong winds, and therefore the real airspeed is much larger. The one sea vessel that is sailing between the wind turbines is misclassified as a bird, while the other sea vessels are classified as small slow boats, small fast boats and helicopters. Unfortunately, nearly all the birds are misclassified as either unknown or as helicopter. We believe this is because that the training data do not contain any birds at that distance and speeds (because of the wind). Further the radar used to record this scenario is different from the radars used for the training data.

6 Conclusion

We have shown that it is possible to use a recursive approach to classify radar tracks from kinematic data. We have also showed that it is possible to use an alpha beta

filter together with the random forest such that stationary targets are classified as stationary. The study both use simulated data, which is simulated to behave as real targets and real world data. We have shown both scenario and confusion matrix to get an overview of the performance.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Terma A/S, Hovmarken 4, Lystrup, Denmark. [2]Aalborg University, Department of Electronic Systems, Fredrik Bajers Vej 7b, Aalborg, Denmark. [3]University of Strathclyde, Technology & Innovation Centre, 99 George St, Glasgow, Great Britain.

References

1. Jochumsen, L.W., Pedersen, M.., Hansen, K., Jensen, S.H., Østergaard, J.: Recursive bayesian classification of surveillance radar tracks based on kinematic with temporal dynamics and static features. In: Radar Conference (Radar), 2014 International, pp. 1–6 (2014)
2. Jochumsen, L.W., Nielsen, E., Østergaard, J., Jensen, S.H., Pedersen, M..: Using position uncertainty in recursive automatic target classification of radar tracks. In: Radar Conference (RadarCon), 2015 IEEE, pp. 0168–0173 (2015)
3. Nguyen, D.H., Kay, J.H., Orchard, B.J., Whiting, R.H.: Classification and tracking of moving ground vehicles. Lincoln Laboratory Journal **13**(2), 275–308 (2002)
4. Challa, S., Pulford, G.W.: Joint target tracking and classification using radar and esm sensors. IEEE Transactions on Aerospace and Electronic Systems **37**(3), 1039–1055 (2001)
5. Angelova, D., Mihaylova, L.: Joint target tracking and classification with particle filtering and mixture kalman filtering using kinematic radar information. Digital Signal Processing **16**(2), 180–204 (2006)
6. Sun, Z., Ban, X.J.: Vehicle classification using gps data. Transportation Research Part C: Emerging Technologies **37**, 102–117 (2013)
7. Garg, M., Singh, U.: C & r tree based air target classification using kinematics. In: National Conference on Research Trends in Computer Science and Technology (NCRTCST), IJCCT_Vol3Iss1/IJCCT_Paper_3 (2012)
8. Breiman, L.: Random forests. Machine learning **45**(1), 5–32 (2001)
9. Theodoridis, S., Koutroumbas, K.: Pattern Recognition, Fourth Edition, 4th edn. Academic Press, ??? (2008)
10. Brookner, E.: Tracking and Kalman Filtering Made Easy. Wiley-Interscience, ??? (1998)
11. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning vol. 1. Springer, ??? (2001)
12. Bishop, C.M.: Pattern Recognition and Machine Learning vol. 1. springer New York, ??? (2006)
13. James, G., Witten, D., Hastie, T.: An Introduction to Statistical Learning: With Applications in R. Taylor & Francis, ??? (2014)
14. Chen, L., Wang, S.: Automated feature weighting in naive bayes for high-dimensional data classification. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, pp. 1243–1252 (2012). ACM
15. Lawton, J.A., Jesionowski, R.J., Zarchan, P.: Comparison of four filtering options for a radar tracking problem. Journal of guidance, control, and dynamics **21**(4), 618–623 (1998)
16. Mohajerin, N., Histon, J., Dizaji, R., Waslander, S.L.: Feature extraction and radar track classification for detecting uavs in civillian airspace. In: Radar Conference, 2014 IEEE, pp. 0674–0679 (2014)
17. SWBD: [Accessed 4 December 2014]. http://dds.cr.usgs.gov/srtm/version2_1/SWBD/

Figure 1 An example of a decision tree where N_1 to N_3 is nodes where a decision must be made. An example could be is the ball blue (True or False)

Figure 2 The state machine used for the data generation of the simulated data. The state machine has three states, an accelerating a, decelerating d and a constant speed c state. The probability for jumping between the states is controlled with $P_{j,j}$ which is change depending on the speed.

Figure 3 An example of a simulated track

(a) Speed PDF of the the first class

(b) Speed PDF of the the second class

(c) Speed PDF of the the third class

(d) Speed PDF of the the fourth class

Figure 4 The speed PDFs of the four different classes

(a) Acceleration PDF for the first class

(b) Acceleration PDF for the second class

(c) Acceleration PDF for the third class

(d) Acceleration PDF for the fourth class

Figure 5 The acceleration PDF of the four classes

Figure 6 The overall performance of the algorithm given the number of measurement used in the feature vector.

Figure 7 The performance of the classifier for the synthetic generated data vs. the number of trees used in the random forest.

Figure 8 The scenario where a RIB is salling out and zigzagging back again, a big amount of birds is present

Figure 9 Hornsrev

Table 1 The feature vector used. The number of measurement has been chosen to be $k = 10$

Feature	Feature description
$\mathrm{std}(\Delta_\mathrm{i})$	Empirical standard deviation of sample-to-sample distances
v_1 $\vdots$ v_k	2-point speed estimate
$\mathrm{mean}(\mathrm{v_i})$	Empirical mean of the speed
$\mathrm{std}(\mathrm{v_i})$	Empiricial standard deviation of the speed
a_1 $\vdots$ a_{k-1}	2-point acceleration estimate
$\mathrm{mean}(\mathrm{a_i})$	Empirical mean of the acceleration
$\mathrm{std}(\mathrm{a_i})$	Empirical standard deviation of the acceleration
$\mathrm{mean}(\mathrm{a_i^\perp})$	Empirical mean of the normal acceleration
$\mathrm{std}(\mathrm{a_i^\perp})$	Empirical standard deviation of the normal acceleration
$\lvert z_k - z_0 \rvert$	Total distance moved
d_0 $\vdots$ d_k	Distance to coastline
$\mathrm{mean}(\mathrm{d_i})$	Empirical mean of the distance to coast line

Table 2 The confusion matrix of the simulated data

	Predicted:			
Actual:	type 1	type 2	type 3	type 4
type 1	95.2	4.8	0.0	0.0
type 2	16.7	72.1	11.2	0.0
type 3	1.0	35.6	63.3	0.0
type 4	0.0	0.0	0.0	99.9
Overall performance	82.6			

Table 3 The confusion matrix for real world data.

	Predicted:					
Actual:	Birds	RIBs	Stationary sea targets	Large ships	Helicopters	Commercial aircrafts
Birds	67.9	9.2	0.0	21.0	1.9	0.0
RIBs	6.4	62.4	0.0	31.2	0.0	0.0
Stationary sea targets	0.5	0.0	99.5	0.0	0.0	0.0
Large ships	21.4	5.1	0.3	61.5	11.6	0.0
Helicopters	12.2	0.0	0.0	0.0	87.8	0.0
Commercial aircrafts	0.8	0.0	0.0	0.0	0.0	99.2
Overall performance	79.7					