Learning of a Tracker Model from Multi-Radar Data for Performance Prediction of Air Surveillance System

Marja Ruotsalainen and Juha Jylhä
Laboratory of Signal Processing
Tampere University of Technology
Tampere, Finland
Email: marja.ruotsalainen@tut.fi, juha.jylha@tut.fi

***Abstract*—A valid model of the air surveillance system performance is highly valued when making decisions related to the optimal control of the system. We formulate a model for a multi-radar tracker system by combining a radar performance model with a tracker performance model. A tracker as a complex software system is hard to model mathematically and physically. Our novel approach is to utilize machine learning to create a tracker model based on measurement data from which the input and target output for the model are calculated. The measured data comprises the time series of 3D coordinates of cooperative aircraft flights, the corresponding target detection recordings from multiple radars, and the related multi-radar track recordings. The collected data is used to calculate performance measures for the radars and the tracker at specific locations in the air space. We apply genetic programming to learning such rules from radar performance measures that explain tracker performance. The easily interpretable rules are intended to reveal the real behavior of the system providing comprehension for its control and further development. The learned rules allow predicting tracker performance level for the system control in all radar geometries, modes, and conditions at any location. In the experiments, we show the feasibility of our approach to learning a tracker model and compare our rule learner with two tree classifiers, another rule learner, a neural network, and an instance-based classifier using the real air surveillance data. The tracker model created by our rule learner outperforms the models by the other methods except for the neural network whose prediction performance is equal.**

I. Introduction

Modern air surveillance systems comprise networked remote-controllable sensors such as primary radars, secondary radars, sensors for various regions of electromagnetic spectrum, and acoustic sensors. The functioning and performance of a modern multi-radar tracker system are hard to predict accurately for every situation in prevailing conditions. The knowledge concerning the performance of the system in various situations and conditions is however vital for predicting its near-future performance and controlling it optimally. Optimal control refers to optimal parametrization and mode selection of the system when using it in varying conditions. In addition to optimal control, the acquired knowledge can be used for analyzing and developing new components for the system.

We have earlier studied performance modeling of primary radars, verification and validation of radar models based on measurement data, optimization of radar models, and calculation of measures that describe the radar and tracker performance [1], [2], [3]. This paper considers the modeling of a multi-radar tracker. Unlike primary radars, the literature does not provide a tracker, which is a complex software system, explicit modeling principles with a physical or mathematical description. A typical performance model of a multi-sensor tracking system involves Monte Carlo simulations where the tracking algorithms [4], [5] are run for simulated flight trajectories and radar detections. Simplified methods for predicting the performance of the tracker are needed [6], for example, to reduce the computational load for real-time applications and to produce the prediction in diversified conditions for which the creation of simulation system setups is not straightforward. Our solution for the problem of modeling a multi-sensor tracker is machine learning. We build a classifier with performance measures which are calculated using appropriate recordings from an air surveillance system and from the satellite navigation system of the cooperative aircraft. The created classifier provides the estimate for the tracker performance fast in all the learned conditions at any location. In addition, the performance model is easy to implement into various simulation tools due to its simple form. As far as we know, our research is the first study that proposes creating the performance model of a tracker using machine learning.

The performance measures are derived from radar target detections and tracks by comparing them with the corresponding aircraft satellite navigation data. This association is performed by our automatic association software. The association lays the foundation for the calculation of tracker and radar performance measures. The procedure reveals the probability of the tracker to track a certain type of aircraft in certain area as well as the location accuracy of the tracks, both important for describing the quality of the air picture for system operators. From radar detection recordings, we calculate corresponding performance measures—the probability of detection and location accuracy—for each radar for each specific volume in the air space. This paper proposes an approach to learning

978-1-5090-4601-0/17/$31.00

tracker's performance model that predicts the level of tracker performance from radar performance measures. Because the radar performance modeling is well-known in the literature [5], the learned classifier can be utilized without measured radar data by simply using the modeled radar performance as the input features. Thus the overall system performance can be predicted in real-time cases, such as the optimal system control, using the physical radar model together with the classifier as the tracker model. The association of aircraft trajectories and multi-radar tracks as well as the calculation of tracker and radar performance measures are not described in more detail in this paper due to their extensive and application-specific nature. Thus, in this paper, we make an assumption that the performance measures for the radars and the tracker are available.

Supervised machine learning techniques are generally used to create models that predict the categories of unseen patterns. Besides the prediction capability, compact and easy-to-interpret models provide valuable knowledge on the relationship between the performance of the radars and the performance of the trackers to support decision making concerning the air surveillance system. The prevailing challenge is to develop a method for learning tracker models that are compact and easily interpretable even by surveillance system developers and operators with no machine learning background. In addition, a possibility to parametrize the structure of the resulting model is desired. Due to these demands, machine learning methods such as neural networks [7], Bayesian classifiers [8], and instance based learning methods [9], [10] are not practical. Regression analysis, applied to predict tracker performance values instead of the tracker performance levels, neither produces easy-to-interpret models of a kind we are looking for. Tree classifiers [11], [12] and rule learner algorithms [13], [14] could be applied, but they typically provide small possibilities of parametrizing the structure of the model or steering the learning process based on application-spesific domain knowledge. Due to this deficiency, we chose genetic programming (GP) [15], [16] as a learning method. In this paper, we address the benefits of GP in general for this kind of real-world application and we compare it with the different learning methods.

Our GP-based approach to rule learning, AMANA (aerial maneuver analysis), has earlier been found useful in lifecycle management of aircraft structures for finding what kind of flying results in major fatigue damage to the structures of fighter jets [17], [18], [19]. We decided to test the same method for this, totally different, application we have at hand. The problem of predicting the level of fatigue damage was a binary classification problem whereas predicting the level of tracker performance is a multi-class problem. Thus an algorithm for combining results from multiple binary classifiers is needed. A simple algorithm for that purpose is presented in Section IV. The AMANA, presented in [17], required a few modifications to be applicable to learning a tracker performance model. In this paper, the modified version of the AMANA is referred to as GPRL (genetic programming based rule learner). The biggest modification is related to the processing of features. Differences between the AMANA and the GPRL are addressed in Section III. The main novelty of this paper is an approach to creating an easy-to-interpret tracker performance model based on measurements.

The organization of this paper is as follows. Section II introduces our research problem mathematically and defines the related terms. Section III presents the principles of the GPRL and describes how it has been implemented for this particular application. Section IV documents the comparison of the GPRL to other learning methods. Section V is discussion and Section VI gives the conclusions.

II. Problem definition

To describe radars and trackers by performance measures is a wide and system-specific topic [5]. Thus, in this paper, we made an assumption that the needed performance measures are available and we omit the exact description of their calculation.

Assume the airspace that is divided into N voxels v_n with the side length of a in X-direction and Y-direction, and the side length of b in Z-direction. For each voxel v_n, we have seven radar performance measures

- the probability of detection R_n^{Pd} as a maximum over all the radars,
- the Euclidian position error in meters R_n^{EEuclid} of the radar having the highest probability of detection,
- the range position error in meters R_n^{ERange} of the radar having the highest probability of detection,
- the azimuth position error in degrees R_n^{EAzimuth} of the radar having the highest probability of detection,
- the aggregated scan frequency of radars R_n^{FScan} as a sum where the scan frequency of each radar is weighted using the probability of detection,
- the weighted range position error in meters R_n^{ERangeW} where the weights are the probability-of-detection-weighted scan frequencies of radars, and
- the weighted azimuth position error in meters $R_n^{\text{EAzimuthW}}$ where the weights are the probability-of-detection-weighted scan frequencies of radars

and two tracker performance measures

- the probability of track T_n^{Ptrack}, and
- the Euclidian position error of tracks in meters T_n^{EEuclid},

based on measurement data.

A. Definition of tracker performance levels

Assume an operator has defined five tracker performance levels—`excellent`, `good`, `moderate`, `weak`, and `poor`—by determining corresponding thresholds for the tracker performance measures

- $r_{\text{excellent}}^{\text{Ptrack}} > r_{\text{good}}^{\text{Ptrack}} > r_{\text{moderate}}^{\text{Ptrack}} > r_{\text{weak}}^{\text{Ptrack}}$ and
- $r_{\text{excellent}}^{\text{EEuclid}} < r_{\text{good}}^{\text{EEuclid}} < r_{\text{moderate}}^{\text{EEuclid}} < r_{\text{weak}}^{\text{EEuclid}}$.

Voxels v_n belong to the defined five categories $C_{\text{excellent}}$, C_{good}, C_{moderate}, C_{weak}, and C_{poor} as follows

$$\begin{cases} v_n \in C_{\text{excellent}} & \text{if } (T_n^{\text{Ptrack}} > r_{\text{excellent}}^{\text{Ptrack}}) \\ & \wedge (T_n^{\text{EEuclid}} < r_{\text{excellent}}^{\text{EEuclid}}) \\ v_n \in C_{\text{good}} & \text{if } (T_n^{\text{Ptrack}} > r_{\text{good}}^{\text{Ptrack}}) \wedge (T_n^{\text{EEuclid}} < r_{\text{good}}^{\text{EEuclid}}) \\ & \wedge v_n \notin C_{\text{excellent}} \\ v_n \in C_{\text{moderate}} & \text{if } (T_n^{\text{Ptrack}} > r_{\text{moderate}}^{\text{Ptrack}}) \\ & \wedge (T_n^{\text{EEuclid}} < r_{\text{moderate}}^{\text{EEuclid}}) \\ & \wedge v_n \notin (C_{\text{excellent}} \cup C_{\text{good}}) \\ v_n \in C_{\text{weak}} & \text{if } (T_n^{\text{Ptrack}} > r_{\text{weak}}^{\text{Ptrack}}) \wedge (T_n^{\text{EEuclid}} < r_{\text{weak}}^{\text{EEuclid}}) \\ & \wedge v_n \notin (C_{\text{excellent}} \cup C_{\text{good}} \cup C_{\text{moderate}}) \\ v_n \in C_{\text{poor}} & \text{else} \end{cases} \tag{1}$$

That is, the tracker performance level for a voxel is the best of the performance levels, whose threshold for the probability of track is smaller than the probability of track in the voxel and whose threshold for the position error is larger than the position error in the voxel.

B. Definition of rules

Learning a classification model is based on features F_i that consist of radar performance measures R^{Pd}, R^{EEuclid}, R^{ERange}, R^{EAzimuth}, R^{FScan}, R^{ERangeW}, and $R^{\text{EAzimuthW}}$. The larger the value of R^{Pd} or R^{FScan} the better the radar performance. Contrary, the smaller the value of R^{EEuclid}, R^{ERange}, R^{EAzimuth}, R^{ERangeW} or $R^{\text{EAzimuthW}}$ the better the performance. Assume feature F_i has a threshold t_i. In the case of R^{Pd} or R^{FScan}, the numerical value of feature F_i for the voxel v_n is converted into boolean using formula

$$F_i^{\text{B}}(v_n) = \begin{cases} \texttt{True} & \text{if } F_i(v_n) \geq t_i \\ \texttt{False} & \text{else} \end{cases}. \tag{2}$$

Whereas in the case of R^{EEuclid}, R^{ERange}, R^{EAzimuth}, R^{ERangeW} or $R^{\text{EAzimuthW}}$, the numerical value of feature F_i for the voxel v_n is converted into boolean using formula

$$F_i^{\text{B}}(v_n) = \begin{cases} \texttt{True} & \text{if } F_i(v_n) \leq t_i \\ \texttt{False} & \text{else} \end{cases}. \tag{3}$$

The boolean features F_i^{B} are combined with logical operators `AND` and `OR` in order to form logical rules for classification. The logical rules are able to describe if the tracker performance level is explained by some radar performance measures separately (operator `OR`) or as a combination of several radar performance measures (operator `AND`).

C. Definition of binary classifiers

We formulate the five-class classification problem as the creation of four binary classifiers to separate voxels in category C_{high} from voxels in category C_{low}. These categories are defined separately for each classifier using the five tracker performance levels introduced in (1). Target categories for the four classifiers are

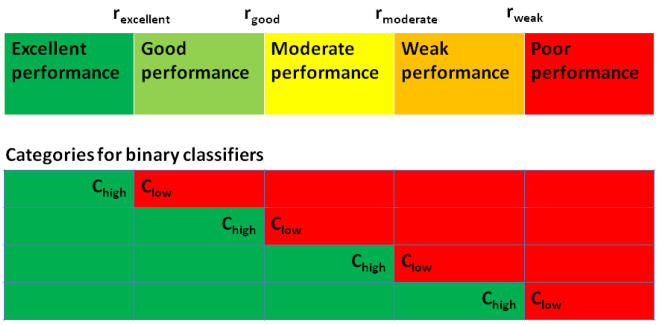

Fig. 1. Five tracker performance levels and related target categories C_{high} and C_{low} for four binary classifiers. Voxels with, for example, moderate tracker performance belong to category C_{low} in the first and second classifier and to C_{high} in the third and fourth classifier.

$$\begin{aligned} C_{\text{high}}^1 &= C_{\text{excellent}} \\ C_{\text{low}}^1 &= C_{\text{good}} \cup C_{\text{moderate}} \cup C_{\text{weak}} \cup C_{\text{poor}} \\ C_{\text{high}}^2 &= C_{\text{excellent}} \cup C_{\text{good}} \\ C_{\text{low}}^2 &= C_{\text{moderate}} \cup C_{\text{weak}} \cup C_{\text{poor}} \\ C_{\text{high}}^3 &= C_{\text{excellent}} \cup C_{\text{good}} \cup C_{\text{moderate}} \\ C_{\text{low}}^3 &= C_{\text{weak}} \cup C_{\text{poor}} \\ C_{\text{high}}^4 &= C_{\text{excellent}} \cup C_{\text{good}} \cup C_{\text{moderate}} \cup C_{\text{weak}} \\ C_{\text{low}}^4 &= C_{\text{poor}}. \end{aligned} \tag{4}$$

Fig. 1 illustrates the target categories.

Categories C_{high} and C_{low} are treated as fuzzy. The membership $\mu_{\text{high}}^{(\text{Ptrack, n})}$ of a voxel v_n in C_{high} based on probability of track T_n^{Ptrack} is defined as follows

$$\mu_{\text{high}}^{(\text{Ptrack, n})} = \begin{cases} 0 & \text{if } T_n^{\text{Ptrack}} \leq \\ & 2r^{\text{Ptrack}} - T_{\text{max}}^{\text{Ptrack}} \\ \frac{|2r^{\text{Ptrack}} - T_{\text{max}}^{\text{Ptrack}} - T_n^{\text{Ptrack}}|}{|2r^{\text{Ptrack}} - 2T_{\text{max}}^{\text{Ptrack}}|} & \text{else} \end{cases}, \tag{5}$$

where T_n^{Ptrack} is the probability of track in voxel v_n, r^{Ptrack} is a threshold for the probability of track to separate the two categories, and $T_{\text{max}}^{\text{Ptrack}}$ is the maximum of the probability of track values. Equation (5) gives, for example, the membership of a voxel v_n in C_{high}^2 (see (4)) based on the probability of track, when $r^{\text{Ptrack}} = r_{\text{good}}^{\text{Ptrack}}$. For the voxels with the highest probability of track $\mu_{\text{high}}^{(\text{Ptrack, n})} = 1$, and for the threshold r^{Ptrack}, $\mu_{\text{high}}^{(\text{Ptrack, n})} = 0.5$. Contrary to the probability of track, the smallest values of position error indicate the best tracker performance. Thus, the membership $\mu_{\text{high}}^{(\text{EEuclid, n})}$ of a voxel v_n in C_{high} based on position error T_n^{EEuclid} is

$$\mu_{\text{high}}^{(\text{EEuclid, n})} = \begin{cases} 0 & \text{if } T_n^{\text{EEuclid}} \geq \\ & 2r^{\text{EEuclid}} - T_{\text{min}}^{\text{EEuclid}} \\ 1 - \frac{T_n^{\text{EEuclid}} - T_{\text{min}}^{\text{EEuclid}}}{2r^{\text{EEuclid}} - 2T_{\text{min}}^{\text{EEuclid}}} & \text{else} \end{cases}, \tag{6}$$

where T_n^{EEuclid} is the position error of tracks in voxel v_n, r^{EEuclid} is a threshold for the position error to separate the two categories, and $T_{\text{min}}^{\text{EEuclid}}$ is the minimum of the position error values. These two memberships, $\mu_{\text{high}}^{\text{(Ptrack, n)}}$ and $\mu_{\text{high}}^{\text{(EEuclid, n)}}$, are combined into one membership value by taking their minimum as follows

$$\mu_{\text{high}}^{\text{n}} = \min(\mu_{\text{high}}^{\text{(Ptrack, n)}}, \mu_{\text{high}}^{\text{(EEuclid, n)}}). \quad (7)$$

The membership μ_{low}^n of a voxel v_n in C_{low} is defined as

$$\mu_{\text{low}}^n = 1 - \mu_{\text{high}}^n. \quad (8)$$

Our problem of finding four binary classifiers to separate categories C_{high} and C_{low} refers to learning a logical rule L, i.e. feature thresholds $t_1 \dots t_I$ and logical structure to combine features $F_1^{\text{B}} \dots F_I^{\text{B}}$. The amount of features in the rule is not defined beforehand. Each rule representing one of the binary classifiers is optimized so that weighted classification accuracy of voxels v_n to the corresponding two categories, C_{high} and C_{low}, is maximized. Weighted classification accuracy is calculated based on membership values μ_{high}^n and μ_{low}^n and it is discussed in more detail in Subsection III-C along with secondary optimization criteria.

III. Applying GPRL to learning tracker model

This section introduces the implementation of the GPRL [17] for learning a tracker model in order to predict the level of tracker performance in a certain area. The GPRL algorithm is mainly the same as the AMANA that we presented in [17]. The main difference between the GPRL and the AMANA is the type of features. Input to the AMANA was a set of multi-sensor time series. While learning the relationship between the input and the output, the AMANA learned how to convert the multi-sensor time series into a set of features, single numerical values, that are usable in analysis. In the GPRL, features are assumed single numerical values by default, so the learning of the conversion related to time series is omitted. Other, smaller differences, are addressed throughout this section.

Fig. 2 shows the GP process in general. The GPRL contains several adjustable parameters and they are, for notational convenience, referred to using their real names and typewriter font in the text. The data used for learning includes radar performance measures R^{Pd}, R^{EEuclid}, R^{ERange}, R^{EAzimuth}, R^{FScan}, R^{ERangeW}, and $R^{\text{EAzimuthW}}$ and tracker performance measures T^{PTrack} and T^{EEuclid}. At first, target categories, C_{high} and C_{low}, are defined for a binary classifier to be learned. Then the GPRL begins with the generation of an initial population of solution candidates, which are called programs. After that, fitness values are calculated for each program in the population. Then programs are selected, crossed, and mutated. The selection employs the tournament selection algorithm [15]. The amount of programs attending to tournaments is controlled by parameter `tournamentSize`. The crossover is based on the subtree crossover algorithm [15]. In mutation, a randomly chosen node is modified. If the node is an internal node `AND`, it is changed to `OR`. If it is `OR`, it is changed to `AND`. For a leaf node, threshold t_i is chosen randomly. Mutation differs from that in the AMANA as we do not have to choose values for the feature parameters.

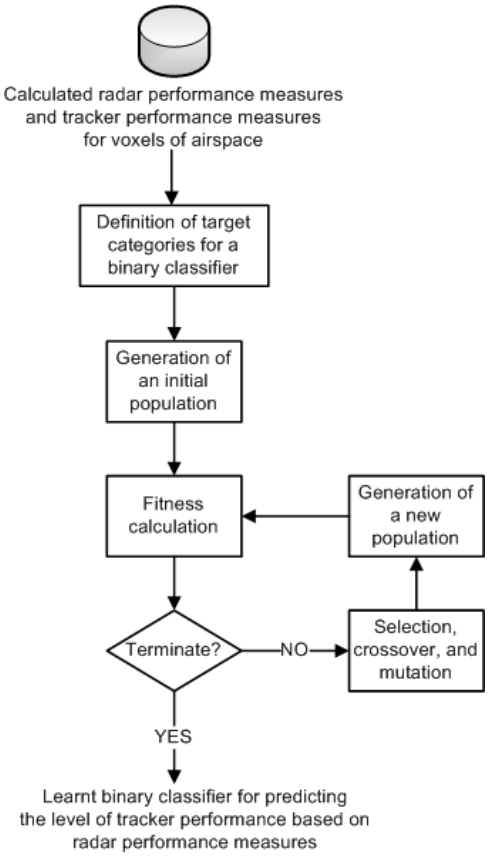

Fig. 2. The application of GP to learning binary classifiers for predicting the level of tracker performance. At first, the target categories, C_{high} and C_{low}, are defined. Then an initial population is generated. Iteration consisting of fitness calculation, selection, crossover, mutation, and generation of a new population continues until a predefined number of generations has been calculated.

New population is composed of crossed and mutated programs. Parameter `crossoverProb` defines the amount of programs in the new population that are produced by crossover. The rest of the programs are generated using mutation with node-specific mutation probability `mutationProbNode`. Elitism is in use, i.e. the fittest program from the previous population is copied into the new population. Iteration continues until a predefined number of generations has been calculated. The amount of generations can be adjusted using parameter `maxGenerations`. Finally, the learned rule, i.e. a binary classifier to separate C_{high} and C_{low}, is outputted as a result. Altogether four classifiers are created to solve our five-category problem.

Blocks (a) the generation of an initial population and (b) fitness calculation are considered in Subsections III-B and III-C, respectively. The next Subsection III-A discusses the representation of programs and Subsection III-D considers the pruning of logical rules.

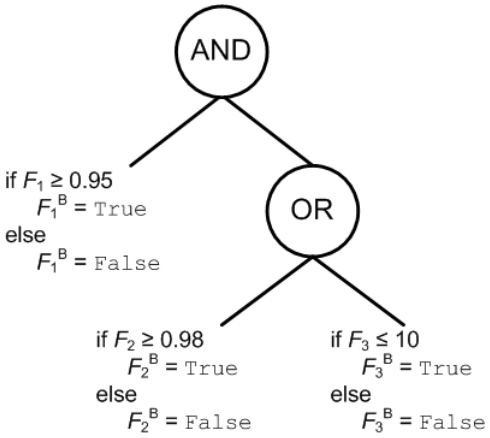

Fig. 3. An example of the tree representation of solution candidates. This tree expresses the program $\texttt{AND}(F_1^{\mathrm{B}}, \texttt{OR}(F_2^{\mathrm{B}}, F_3^{\mathrm{B}}))$. The numbers in the figure stand for thresholds t_i. The example illustrates the representation of programs but it does not represent an analysis result with real measurement data due to their confidential nature.

A. Representation of programs

We express the programs as syntax trees, which are a typical representation of solution candidates in GP [15]. The leaves of the tree are based on boolean features defined in (2) and (3). The definition of the boolean features in the GPRL differs from that in the AMANA. In the GPRL, features are numerical values and they are converted into boolean by simply thresholding. In AMANA, boolean features were achieved by calculating features from time series according to the feature parameters and by thresholding the calculated feature values. Feature parameters were not known in advance but they were learned while learning the logical rule. The internal nodes of the syntax tree are logical operations `AND` and `OR` as in the AMANA.

Fig. 3 gives an example of the tree representation of a solution candidate. It expresses the program $\texttt{AND}(F_1^{\mathrm{B}}, \texttt{OR}(F_2^{\mathrm{B}}, F_3^{\mathrm{B}}))$. Assume feature F_1 represents a radar performance measure, whose value in a voxel in question is 0.96. Assume feature F_2 represents the same radar performance measure so its value is 0.96 too. F_3 represents another radar performance measure and its value is 8. Fig. 3 shows three thresholds $t_1 = 0.95$, $t_2 = 0.98$, and $t_3 = 10$. Because F_1 is larger than 0.95, F_1^{B} is `True`. Feature F_2 is smaller than 0.98, so F_2^{B} is `False`. Because F_3 is smaller than 10, F_3^{B} is `True`. That is, $\texttt{OR}(F_2^{\mathrm{B}}, F_3^{\mathrm{B}})$ is `True` and the whole rule $\texttt{AND}(F_1^{\mathrm{B}}, \texttt{OR}(F_2^{\mathrm{B}}, F_3^{\mathrm{B}}))$ is `True` for the voxel in question. Thus the rule suggests that the voxel belongs to the category C_{high}.

B. Generation of an initial population

Generation of an initial population of programs is controlled by parameters `populationSize`, `maxLevels`, `minOperands`, and `maxOperands`. The amount of programs in the initial population is defined by `populationSize`. The maximum number of levels of the initial programs is restricted by parameter `maxLevels`. Parameters `minOperands` and `maxOperands` define the minimum and maximum amount of operands for each `AND` and `OR` operator in a tree.

Initial programs are formed using so called grow method [15], where the nodes to the tree are selected randomly from the sets of internal nodes and leaf nodes. The set of leaf nodes is infinite as the threshold of a feature can be whatever decimal digit within the range of the selected performance measure. Therefore we, departing from the original grow method, give 50% change to the node to be either an internal node or a leaf node. An exception is level `maxLevels`, where all nodes are forced to be leaf nodes. If the node is an internal node, the operator is chosen randomly from set $\{\texttt{AND}, \texttt{OR}\}$. Whether the node is a leaf node, the radar performance measure is chosen randomly from set $\{R^{\text{Pd}}, R^{\text{EEuclid}}, R^{\text{ERange}}, R^{\text{EAzimuth}}, R^{\text{FScan}}, R^{\text{ERangeW}}, R^{\text{EAzimuthW}}\}$. Ranges of the radar performance measures are calculated from the training data. Threshold t_i for each leaf node is selected randomly from the calculated range of the selected radar performance measure. In the AMANA, the values of the feature parameters were also selected randomly. In the GPRL, features do not have parameters so that selection step is omitted.

C. Fitness calculation

Fitness of a program is evaluated using three separate criteria: weighted classification accuracy, the size of the rule, and unweighted classification accuracy. In the calculation of weighted classification accuracy, fuzzy target categories are used. Contribution of a voxel v_n to the weighted classification accuracy of a rule L_j is defined as

$$c^n = \begin{cases} \mu^n_{\text{high}} & \text{if } v_n \in C_{\text{true}} \\ 1 - \mu^n_{\text{high}} & \text{if } v_n \in C_{\text{false}} \end{cases}, \tag{9}$$

where C_{true} is a set of voxels v_n for which the rule L_j is `True` and C_{false} is a set of voxels for which L_j is `False`. At optimum, all the voxels are classified correctly and the contribution of a voxel is

$$c^n_{\text{opt}} = \begin{cases} \mu^n_{\text{high}} & \text{if } v_n \in C_{\text{high}} \\ 1 - \mu^n_{\text{high}} & \text{if } v_n \in C_{\text{low}} \end{cases}, \tag{10}$$

where C_{high} and C_{low} are as defined in (4). Weighted classification accuracy of a rule L_j is then

$$\texttt{Acc}_{\text{w}} = \frac{\sum_{n=1}^{N} c^n}{\sum_{n=1}^{N} c^n_{\text{opt}}}, \tag{11}$$

where N is the amount of voxels. Weighted accuracy $\texttt{Acc}_{\text{w}}$ is in range $[0, 1]$ being 1, when all the voxels are categorized correctly. The voxels with large memberships in C_{high} or C_{low} weight the most. Those voxels are roughly speaking either (a) voxels with the high probability of track and low position error (from C_{high}) or (b) voxels with the low probability of track or high position error (from C_{low}).

The second optimization criterion, the size of the rule, is defined as the number of internal and leaf nodes together.

The third criteria, unweighted classification accuracy (the probability of correct classification), for a logical rule L_j is

$$\texttt{Acc} = \frac{\text{num}(C_{\text{true}} \cap C_{\text{high}}) + \text{num}(C_{\text{false}} \cap C_{\text{low}})}{\text{num}(C_{\text{high}} \cup C_{\text{low}})}, \quad (12)$$

where C_{true} is a set of voxels for which the rule is true and C_{false} is a set of voxels for which the rule is false. C_{high} and C_{low} are as defined in (4). Function `num` represents the number of voxels in the set.

Weighted classification accuracy, the size of the rule, and unweighted classification accuracy are not combined into one figure, but they are used separately. Weighted classification accuracy is maximized, the size of the rule is minimized, and unweighted classification accuracy is maximized. In GP, it is enough to be able to compare and find the fittest from a set of programs. When comparing programs, weighted classification accuracy is considered first. If two or more programs have the same weighted classification accuracy, the sizes of the rules are considered. If two or more programs are of the same size, then the unweighted classification accuracy is considered. So, the results are mainly based on the most important criterion, weighted classification accuracy. The criterion concerning the size of the rule performs a kind of pruning to the programs without any loss in weighted classification accuracy. (Pruning algorithm presented in Subsection III-D is not lossless in that sense with all `minContribution` values.) The least important criterion is there to make sure that if we sometimes get multiple programs with the same weighted classification accuracy and the size of the rule, then we choose the one with the highest unweighted classification accuracy. The order in which the criteria are applied in the GPRL differs from the AMANA, where the order was: unweighted classification accuracy, the size of the rule, and weighted classification accuracy.

D. Pruning of programs

To avoid bloating, programs resulting from crossover are pruned if their size exceeds the size defined by parameter `maxNodes`. AMANA did not have parameter `maxNodes` and it pruned programs regardless of their node amount. The idea in pruning is to remove all such nodes that do not positively contribute to the classification result. Pruning utilizes contribution, which we define as

$$\texttt{Contribution} = TP_{\text{node}} - FP_{\text{node}}, \quad (13)$$

where $TP_{\text{node}} = TP_{\text{program}} - TP_{\text{program}-\text{node}}$ and $FP_{\text{node}} = FP_{\text{program}} - FP_{\text{program}-\text{node}}$. TP_{program} is the amount of the true positive classifications of the unpruned program and $TP_{\text{program-node}}$ is the amount of the true positive classifications of the pruned program, where one leaf node has been removed. Thus, TP_{node} describes how many true positive classifications are lost if a particular node is removed. Similarly, FP_{program} is the amount of the false positive classifications of the unpruned program and $FP_{\text{program-node}}$ is the amount of the false positive classifications of the pruned program. FP_{node} describes how many false positive classifications are lost if a particular node is removed. FP_{node} is often negative meaning that keeping the node in the tree reduces the amount of false positive classifications. A node with contribution zero does not influence on classification accuracy. Parameter `minContribution` is used to determine the minimum contribution that is required from the nodes remaining after pruning. Our pruning algorithm is presented in more detail in [17].

IV. Experiments

This section presents the comparison of the GPRL to (a) the C4.5 tree classifier, (b) the random forest tree classifier, (c) the PART rule learner [14], (d) the multilayer perceptron, and (e) the K-nearest neighbors classifier with the real air surveillance data from five primary radars. Data contained the seven radar performance measures and the two tracker performance measures, introduced in Section II, for 173 voxels. There were plenty of recorded flight trajectories in each voxel, so each performance measure was produced as a voxel-specific average.

The values of tracker performance measures were used to define target categories `excellent`, `good`, `moderate`, `weak`, and `poor`. In addition, they were used to calculate the membership of voxels (see (7) and (8)) in categories defined by (4) for the GPRL. Voxels in the one half of the studied air space were used as training data and voxels in the remaining other half of the air space were used as test data to evaluate the generalization capability of the classifiers.

Waikato Environment for Knowledge Analysis [20] software was used to run the C4.5, the random forest, the PART, the multilayer perceptron, and the K-nearest neighbors classifiers. The C4.5 tree classifier was run using its default parameters, where the minimum number of instances per leaf was two and pruning was in use. In the case of the random forest classifier, the number of trees was reduced from default value to five in order to get a more compact set of models—this had no relevant influence on its classification accuracy. The PART rule learner was run using its default values, where pruning was in use. The multilayer perceptron was run with its default settings. For K-nearest neighbors classifier, the number of neighbors to use was raised from its default of one to three in order to get more reliable classification results.

The parameter values of the GPRL were not optimized for learning a tracker performance model. Default parameter values [17] were used except for parameter `maxGenerations`, which was reduced from 100 to 20 because resulting models seemed to converge faster than in our earlier problem concerning aircraft structural health monitoring. In addition, `maxNodes` is a new parameter. Parameter values used in these experiments were

- `maxGenerations` = 20 (see the beginning of Section III),
- `populationSize` = 10 000 (see III-B),
- `maxLevels` = 1 (see III-B),
- `minOperands` = 2 (see III-B),
- `maxOperands` = 5 (see III-B),

```
1  L1: classifier 1
2  L2: classifier 2
3  L3: classifier 3
4  L4: classifier 4
5  v: a voxel
6  C: classification result
7
8  if L1(v) == TRUE
9      C := EXCELLENT;
10 elseif L2(v) == TRUE
11     C := GOOD;
12 elseif L3(v) == TRUE
13     C := MODERATE;
14 elseif L4(v) == TRUE
15     C := WEAK;
16 else
17     C := POOR;
18 end
```

Fig. 4. Algorithm for combining results from the learned four binary classifiers. The algorithm selects the best of the tracker performance levels for which the learned rule is fulfilled by the voxel in question.

- `maxNodes` = 5 (see III-D)
- `tournamentSize` = 2 (see the beginning of Section III),
- `minContribution` = 1 (see III-D),
- `crossoverProb` = 0.8 (see the beginning of Section III), and
- `mutationProbNode` = 0.4 (see the beginning of Section III).

The C4.5, the random forest, the PART, the multilayer perceptron, and the K-nearest neighbors are multi-class classifiers, so they used multi-class labels `excellent`, `good`, `moderate`, `weak`, and `poor` in learning. For the GPRL, the multi-class labels were converted into labels `high` and `low` as in (4). Four different definitions for `high` and `low` were used in order to train four binary classifiers, which together can separate the five classes from each other. The first binary classifier was trained to separate tracker performance level `excellent` from the other levels. The second classifier was trained to separate tracker performance levels `good` and `excellent` from the other levels. The third classifier was trained to separate the levels that are at least `moderate` from the other levels. And the fourth classifier was trained to separate the levels that are at least `weak` from `poor`. Classification results from binary classifiers were combined using the algorithm presented in Fig. 4. According to the algorithm, a voxel is assigned to the best tracker performance level of those suggested by binary classifiers.

A. Classification performance

The classification models were learned from training data and tested using test data. Tables I–IV show four classification performance measures for the six tested classifiers. The random forest, the multilayer perceptron, and the GPRL exploit randomness. Their results in the tables are averages over ten runs and standard deviations as well as 95 % confidence intervals are provided. The assumption of gaussian distribution has been made in the calculation of the confidence intervals. The used four classification performance measures include classification accuracy, unweighted Cohen's kappa, linear weighted Cohen's kappa, and quadratic weighted Cohen's kappa [21]. Weighted Cohen's kappa takes into account the degree of disagreement between the classification result and the ground truth. The degree of disagreement refers to the fact that classifying, for example, a voxel with `moderate` tracker performance into class `good` or `weak` is more correct than classifying it into `excellent` or `poor`. The best classification results, considering accuracy and unweighted kappa, are achieved using the GPRL. In linear and quadratic weighted kappa, the prediction performance of the GPRL and the multilayer perceptron can be considered equal.

TABLE I
ACCURACY FOR SIX TESTED CLASSIFICATION METHODS. RANDOM FOREST, MULTILAYER PERCEPTRON, AND GPRL EXPLOIT RANDOMNESS AND THEIR ACCURACIES ARE AVERAGES OVER TEN RUNS. STANDARD DEVIATIONS AND 95 % CONFIDENCE INTERVALS ARE PROVIDED. THE STUDIED CLASSIFICATION PROBLEM INVOLVED FIVE CATEGORIES.

	Accuracy	Standard deviation	95 % confidence interval
C4.5 tree classifier	0.4000	-	-
Random forest	0.4506	0.0549	0.4166 - 0.4846
PART rule learner	0.4235	-	-
Multilayer perceptron	0.4706	0.0242	0.4556 - 0.4856
K-nearest neighbors	0.4118	-	-
GPRL	0.4906	0.0208	0.4777 - 0.5035

TABLE II
UNWEIGHTED KAPPA FOR SIX TESTED CLASSIFICATION METHODS.

	Unweighted kappa	Standard deviation	95 % confidence interval
C4.5 tree classifier	0.2217	-	-
Random forest	0.2778	0.0736	0.2322 - 0.3235
PART rule learner	0.2650	-	-
Multilayer perceptron	0.3179	0.0312	0.2986 - 0.3372
K-nearest neighbors	0.2494	-	-
GPRL	0.3391	0.0294	0.3209 - 0.3573

B. Interpretability of the learned model

The resulting classifier represents a tracker performance model that has been learned from the real measurement data of the air surveillance system. In order to produce valuable knowledge on the relationship between the performance of radars and the performance of the tracker, the learned model must be compact and easily interpretable even by people

TABLE III
LINEAR WEIGHTED KAPPA FOR SIX TESTED CLASSIFICATION METHODS.

	Linear weighted kappa	Standard deviation	95 % confidence interval
C4.5 tree classifier	0.3470	-	-
Random forest	0.4161	0.0732	0.3708 - 0.4615
PART rule learner	0.4312	-	-
Multilayer perceptron	0.4761	0.0331	0.4556 - 0.4966
K-nearest neighbors	0.4447	-	-
GPRL	0.4935	0.0258	0.4775 - 0.5095

TABLE IV
QUADRATIC WEIGHTED KAPPA FOR SIX TESTED CLASSIFICATION METHODS.

	Quadratic weighted kappa	Standard deviation	95 % confidence interval
C4.5 tree classifier	0.4816	-	-
Random forest	0.5493	0.0814	0.4989 - 0.5998
PART rule learner	0.5703	-	-
Multilayer perceptron	0.6197	0.0358	0.5976 - 0.6419
K-nearest neighbors	0.6067	-	-
GPRL	0.6317	0.0334	0.6110 - 0.6524

with no machine learning background. Neither the multilayer perceptron nor the K-nearest neighbors classifier fulfill this demand. In these experiments, they have been tested to set the baseline for the achievable performance of classification. The interpretability of the models by the C4.5, the random forest, the PART, and the GPRL can be compared by calculating the size of the resulting tree or rule. As the representation of the conventional decision tree and the rule learned by the PART or the GPRL is different, we converted the learned rules into the decision tree format. The conversion has been illustrated in [17]. Table V shows the number of trees in a model and the number of nodes per tree for the C4.5, the random forest, the PART, and the GPRL. Random forest and GPRL models were learned ten times due to the tree creation procedure that involves randomness. The random forest, PART, and GPRL models consist of multiple trees, so the node amounts in Table V are averages of all the created trees for these three classifier types. Standard deviation for the number of nodes per tree is provided.

A random forest model contains five trees of size 43 on average and a PART model contains nine trees of size six on average. Compared with a C4.5 model or a GPRL model, the random forest and PART models are complex. The complexity of the model created by the C4.5 and the GPRL is quite equal—the C4.5 model consists of one tree of size 23 and the GPRL model consists of four trees of size 6 on average. The GPRL model of four binary classifiers, however, provides well-defined knowledge on how the requirements for the performance of radars tighten as the level of tracker performance increases. Overall, the classifier learned by the GPRL has the best classification performance (see Tables I-IV) with the compact easy-to-interpret model and thus it is the most appropriate for this application.

TABLE V
NUMBER OF NODES IN THE LEARNED MODELS FOR C4.5, RANDOM FOREST, PART, AND GPRL. FOR RANDOM FOREST, PART, AND GPRL, THE PRESENTED NODE AMOUNT FOR A SINGLE TREE IS AN AVERAGE OF ALL THE CREATED TREES. STANDARD DEVIATIONS ARE PROVIDED FOR ASSESSING THE VARIATION IN THE NUMBER OF NODES PER TREE.

	Total number of nodes	Number of trees	Number of nodes per tree (avg)	Number of nodes per tree (std)
C4.5 tree classifier	23	1	23	-
Random forest	215	5	43	3.7
PART rule learner	54	9	6	2.4
GPRL	24	4	6	0.9

V. DISCUSSION

This work originated from a need to model the performance of a multi-radar tracker based on measurement data from an air surveillance system. We predict the level of tracker performance using radar performance measures. It is clear that the radar performance measures have a key role in learning. The performance of classification (see Tables I–IV) showed that used radar performance measures are not fully able to account for the level of tracker performance. A tight relationship between the radar and tracker performance measures cannot be learned by any classifier if that relationship is missing from the training data. Our experiments, however, showed that GP is able to produce easy-to-interpret models while getting the best classification performance set by the tested well-known machine learning methods. Next, we will develop the calculation of radar performance measures further. The current measures do not, for example, take into account the direction from which a radar sees an aircraft. The direction, however, is closely linked to the probability of detection. Generally speaking, the probability of detection is much higher if a radar sees, for example, a side of an aircraft instead of its nose. Taking the direction into account calls for calculating the radar performance values separately for each direction in each voxel.

Classification results coming from four binary classifiers created by the GPRL were combined to get a multi-class result. The algorithm for combining the results was simple. It just returned the best tracker performance level of those suggested by the binary classifiers. This is logical if binary classifiers have learned the ground truth correctly. Ideally, a voxel that fulfills the rule of the first classifier, that is separating class

`excellent` from the other classes, should also fulfill the rules of the three other classifiers, because their requirements for the radar performance should be looser. The usage of multiple binary classifiers provides a clear insight into how the performance of radars evolves as the level of tracker performance increases. This is a desirable feature. However, learning binary classifiers parallel in such a way that the four learning processes interact with each other could be beneficial. This way we could achieve binary classifiers that are more coherent in their structure and compose an overall tracker performance model more appropriate.

Typical performance models of a multi-sensor tracking system involve Monte Carlo simulations where the tracking algorithms are run for simulated flight trajectories and radar detections. Approaches of that kind are computationally intensive and arduous to set up as they require manual work of a simulation expert. Learning a performance model for a tracker using the GPRL provides a very computationally efficient way to predict the tracker performance in various situations and conditions. To prove its more extensive applicability, we will continue our work and apply the GPRL for learning a tracker model for another configuration of the air surveillance system. Another future challenge is taking varying conditions such as weather parameters into account. As the performance of the air surveillance system can be predicted accurately in reasonable time for various near-future scenarios, the operators can be provided with the needed knowledge and tools for decision making related to the control of the system.

VI. Conclusion

This paper considered a novel study of learning a performance model of a tracker from measured data. The learned model is intended to reveal the relationship between the performance of radars and the performance level of the tracker. We implemented our GP-based classifier, called GPRL, to the problem, because the learned model had to be easily interpretable. In addition, incorporating domain knowledge to define the structure of models, to calculate fitness, or to steer learning is relatively easy with the GPRL. In the experiments, the model created by the GPRL outperformed the models by the other classifiers in accuracy and kappa except for the multilayer perceptron whose prediction performance was equal. The achieved results from two separate research fields—lifecycle management of aircraft structures [18] and control of air surveillance system—suggest that GP is a valuable method for learning easy-to-interpret models related to complex real-world problems for which measurement data and usable domain knowledge is available.

Acknowledgment

The authors would like to thank (Finnish) Scientific Advisory Board for Defence for funding.

References

[1] M. Väilä, J. Jylhä, V. Väisänen, H. Perälä, A. Visa, M. Harju, and K. Virtanen, "A RCS model of complex targets for radar performance prediction," in *Proceedings of the 2017 IEEE Radar Conference*. To be published, 2017.

[2] M. Väilä, J. Jylhä, T. Sailaranta, H. Perälä, V. Väisänen, and A. Visa, "Incorporating a stochastic model of the target orientation into a momentary RCS distribution," in *Proceedings of the 2015 IEEE Radar Conference*. IEEE, 2015, pp. 969–973.

[3] M. Väilä, J. Jylhä, H. Perälä, and A. Visa, "Performance evaluation of radar NCTR using the target aspect and signature," in *Proceedings of the 2014 IEEE Radar Conference*. IEEE, 2014, pp. 623–628.

[4] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with Applications to Tracking and Navigation*. John Wiley and Sons, 2001.

[5] M. A. Richards, J. A. Scheer, and W. A. Holm, *Principles of Modern Radar*. SciTech Pub., 2010.

[6] W. Blair and P. Miceli, "Performance prediction of multisensor tracking systems for single maneuvering targets," *Journal of Advances in Information Fusion*, vol. 7, no. 1, pp. 28–45, June 2012.

[7] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999.

[8] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*. John Wiley & Sons, 2001.

[9] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, January 1967.

[10] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine Learning*, vol. 6, no. 1, pp. 37–66, January 1991.

[11] J. R. Quinlan, "Induction of decision trees," *Machine Learning*, vol. 1, no. 1, pp. 81–106, March 1986.

[12] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Chapman & Hall/CRC, 1984.

[13] B. R. Gaines and P. Compton, "Induction of ripple-down rules applied to modeling large databases," *Journal of Intelligent Information Systems*, vol. 5, no. 3, pp. 211–228, November 1995.

[14] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," in *Proceedings of the Fifteenth International Conference on Machine Learning*, 1998, pp. 144–151.

[15] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*, 2008, freely available at www.gp-field-guide.org.uk.

[16] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. A Bradford Book, 1992.

[17] M. Ruotsalainen, J. Jylhä, and A. Visa, "Reasoning logical rules from multi-sensor data for lifecycle management of aircraft structures," in *Proceedings of the IEEE Congress on Evolutionary Computation*. IEEE, 2013, pp. 3221–3227.

[18] M. Ruotsalainen, J. Jylhä, and A. Visa, "Minimizing fatigue damage in aircraft structures," *IEEE Intelligent Systems*, vol. 31, no. 4, pp. 22–29, July-August 2016.

[19] M. Ruotsalainen, J. Jylhä, T. Salonen, H. Janhunen, and A. Visa, "Reasoning flight parameter based rules: Why do nominally similar flight maneuvers cause diverse fatigue damage?" in *Proceedings of the 27th Symposium of the International Committee on Aeronautical Fatigue and Structural Integrity (ICAF 2013)*, vol. 1, 2013, pp. 439–450.

[20] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *ACM SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, June 2009.

[21] J. Cohen, "Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit," *Psychological Bulletin*, vol. 70, no. 4, pp. 213–220, October 1968.