IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 May 2022, accepted 18 June 2022, date of publication 23 June 2022, date of current version 1 July 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3185763

Estimation of Missing LiDAR Data for Accurate AGV Localization

ARPAD GELLERT, DARIUS SARBU, STEFAN-ALEXANDRU PRECUP, ALEXANDRU MATEI, (Graduate Student Member, IEEE), DRAGOS CIRCA, AND CONSTANTIN-BALA ZAMFIRESCU

Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu, 550025 Sibiu, Romania

Corresponding author: Constantin-Bala Zamfirescu (constantin.zamfirescu@ulbsibiu.ro)

This work was supported by the Hasso Plattner Excellence Research Grant financed by the Knowledge Transfer Center of the Lucian Blaga University of Sibiu under Grant LBUS-HPI-ERG-2020-03.

ABSTRACT This article evaluates several machine learning methods to substitute the missing light detection and ranging data for better spatial localization of industrial automated guided vehicles. Decision trees and ensemble of trees using bagging or boosting techniques have been considered. Also, the k-nearest neighbors algorithm was analyzed. Most of the algorithms have been evaluated based on multiple criteria and hyper parameter tuning. The analysis of the results was done in a comparative way, multiple regression evaluation metrics being considered. The experiments have shown that the extreme gradient boosting algorithm provides the best results in terms of performance, but with timing and resource allocation drawbacks. On the other hand, a simple decision tree model seems to give good results if a tradeoff between performance and prediction time must be made. The k-nearest neighbors algorithm is also performing pretty well, especially because we are experimenting in a static environment.

INDEX TERMS Automated guided vehicle, digital twin, LiDAR, point cloud estimation.

I. INTRODUCTION

The backbone of autonomous vehicles is the ability to perceive the environment, create or use an existing map to localize itself and plan a path to the destination point. This is done with the help of Simultaneous Localization and Mapping (SLAM) algorithms that use input data from different type of sensors like Light Detection and Ranging (LiDAR) or cameras. Map building and localization is a challenging research area in robotics when the employment of a global positioning system is unfeasible. While there are several studies showing that more data contribute to increased accuracy and stability, the approach poses various difficulties for achieving robust deployment in real world settings: high-resolution maps are memory and computationally unfeasible to be maintained on the hardware of Automated Guided Vehicle (AGV), while dealing with high amounts of uncertainty in sensors data presumes real-time estimation of predictive uncertainties. Consequently, to our knowledge there are no studies that explore the effects of low-cost LiDAR or cases where LiDAR data is missing and how these issues can be solved for robust deployment of AGVs in indoor locations. The problem presented in this paper is about missing LiDAR data and such we will take a closer look into missing sensor data imputation methods.

Missing data can be the effect of unreliable sensors, connection errors or physical obstructions, the last one being the case investigated in this paper. We focus on the estimation of missing LiDAR data for better AGV localization and robust deployment in factory settings. The LiDAR data consists in the distances detected from the AGV to the surrounding obstacles, which are missing in the directions of the wheels. To substitute the missing distances, we applied methods based on decision trees and ensemble trees. We included the random forest, adaptive boosting and gradient boosting algorithms, as well as the k-Nearest Neighbors (KNN) model based on a simple search over the whole dataset, and a distance measurement.

The rest of this paper has the following structure: Section II discusses the relevant related work, Section III presents the workflow of our application and the involved machine learning algorithms, Section IV provides the experimental results and, finally, Section V concludes the paper and presents further work directions.

The associate editor coordinating the review of this manuscript and approving it for publication was Rongbo Zhu.

II. RELATED WORK

Widely used state-of-the-art SLAM algorithms in the robotics field include LiDAR-based algorithms like GMapping [1], HectorSLAM [2] or Cartogapher [3] and visual-based like ORB-SLAM2 [4] that can use monocular, stereo and RGB-depth cameras. One issue is that the SLAM algorithm must be chosen and then customized for a specific robot, according to its hardware and computational capabilities, sensor constrains and finally the environment. The input data of the SLAM algorithm also plays an important role in the stability and reliability of the method. As such, combining multiple LiDAR sensors to have more input data or to cover a larger area of the environment [5]–[7] is a popular choice that has increased financial and computational costs.

Unreliable sensors, connection errors or physical obstruction are common problems that cause missing data. Missing data imputation or prediction problem can be found in many fields of research, being mostly researched in the statistics field. The missing data is classified depending on the mechanism that caused it into four categories [8], [9]: missing completely at random (MCAR), missing at random (MAR), a non-ignorable case or missing not at random (MNAR) and missing by natural design (MBND). Classical statistical methods include expectation-maximization (EM) [10]–[12], maximum likelihood, partial deletion, hot/cold deck, mean substitution [10], [13]–[16], etc., while more classical machine learning approaches include Markov Chain Monte Carlo computations [17], [18], linear regression [13], [14], [19], KNN [10], [11], [13]–[16], [20], Support Vector Machines (SVMs) [13], [14], Neural Networks (NNs) [10], [13], [21], Vector Autoregressions (VARs) [13], Decision Tree Regressors (DTRs) [13], or deep neural networks [9], [16].

In [13], the authors present an in-depth study of multiple univariate and multivariate methods for missing data imputation of sensor data in the field of maritime industry. They compare 20 models using 7 different metrics, describing advantages and disadvantages for each of them. Their conclusion is that none of the evaluated methods can be used universally, and that their performance depends on the type of data, missing mechanism, and the presence of a real-time requirement.

Authors of [10] analyze multiple sensor data imputation methods for a water quality information system on two separate datasets. The algorithms used are statistical based, model based, and neural network based, with neural network algorithms achieving the best results among the three.

In [20], several data imputation methods – Multiple Imputation by Chain Equations (MICE), Variational Autoencoder (VAE), NN with Random Weights (NNRW), KNN, Random Forest – are used to replace missing data that would improve the calibration process of IoT sensors. In their study, the VAE missing value imputation method shows the best results, followed by the NNRW.

A study on air pollution sensor calibration [11] evaluates KNN, Regression Imputation (RI), EM and Random Forest techniques individually against their proposed meta-predictor based on k-Means clustering, a variation of Best Fit Missing Value Imputation (BFMVI) method, that can choose an optimal imputation method at a given time. The proposed BFMVI algorithm partitions data into clusters using the k-means algorithm and then a best fit estimation method is chosen for each partition by evaluating each technique individually on that specific partition. In this case, BFMVI showed the best results, followed by the KNN method.

In [12], a new sequence-to-sequence imputation model (SSIM) is proposed to predict missing data from a network of wireless water quality sensors. Their method combines a deep learning network architecture with a Long Short-Term Memory Network. Their proposed method is compared with other algorithms like the AutoRegressive Integrated Moving Average (ARIMA), Seasonal ARIMA, Matrix Factorization (MF), MICE and EM.

An extensive study of mean imputation, predictive mean matching (PMM), Bayesian Principal Component Analysis (PCA), KNN, Iterative KNN and meta predictor on a group of 84 datasets from UCI Machine Learning Repository was made in [15]. The best performing algorithms, in general, were their proposed meta predictor, followed by KNN and BPCA while the worst performing was mean imputation and PMM.

Another recent benchmark study of missing data imputation was done in [16] on 69 datasets from OpenML database with different experimental settings, regarding missing data mechanism and completeness of the data in the training dataset. The tested methods are mean and mode imputation, KNN, Random Forest, discriminative Deep Learning and Generative Deep Learning. Their conclusion is that in general, KNN and random forest perform the best in most situations.

Other approaches used to impute missing sensor data include methods used in forecasting problems, since the two problems are very similar. In [22], a framework is proposed based on Seasonal and Trend decomposition using Loess (STL) [23] to impute large missing gaps of sensor data for industrial Internet of Things (IoT) manufacturing while in [24] Bayesian Maximum Entropy is used for imputing similar IoT sensors data.

LiDAR is a method for determining ranges, with application in multiple domains like agriculture, autonomous vehicles, archeology, biology, geology, military, robotics, etc. A LiDAR sensor, depending on its complexity, can output directly 2D or 3D scan of the environment, usually with the goal of creating a 3D model of the environment by combining multiple scans [25].

In [26], the authors present a novel approach to improve the rendering quality of LiDAR point-cloud. Usually, large datasets resulted from LiDAR scans have incomplete information or lack the quality for the desired visual appeal. They focus on the reconstruction of missing building walls, parts of the terrain and of water surfaces. They propose a processing pipeline enhancing through data fusion the point-cloud, achieved by using vector maps, color information and by calculating point normals for the illumination for a more visually appealing outcome. Their approach provides remarkable results.

In [27], a new method for the recovery of missing LiDAR data points from vehicle trajectory is presented. The method uses microscopic traffic flow models. For short gaps in the data points, the authors state that the data can be recovered using a simple linear regression, while for bigger gaps their method is preferred. They calibrated and tested several car following models such as Gipps, Pipes, Newell or IDM. Based on their results, the Gipps model performed the best.

Machine Learning methods have been employed in a lot of fields, especially during the last years. This also applies to multi-output regression models. There are multiple domains where multi-output problems might occur, starting from life related sciences, to ecology, industry manufacturing, multi-media, and so on. Multiple methods have been proposed for dealing with multi-output prediction problems. They can be categorized in different ways. One way is to consider the context, which can be local or global. When local context is used, the multi-output problem will be split into simpler methods or single-output predictions and parts of the output will be predicted, and then the separate methods will be combined to get the overall multi-output result. If global context is considered, the prediction will be determined for the whole structure entirely, one model for predicting the multi-output vector. The global approach has the advantage of taking the dependencies between target variables into account. It can also be more efficient from a computational perspective. If the output is very large, applying a basic model for each target variable can be unfeasible. According to [28], the multi-output prediction methods can be divided into two categories: problem transformation, which corresponds to the local context outlined earlier, and algorithm adaptation, which corresponds to the global context.

Next, several related work cases will be presented briefly. According to [29], multi- output regression models were used to perform ecological modelling. They did a comparative analysis of single-output models and multi-output regression models, and they have shown the advantages of using the multi-output models. They used a decision tree and ensemble of decision trees.

A multi-output regression model was used for predicting real-time gas tank levels [30]. The model was based on the least square support vector algorithm. Another study shows that multi-output regression models (specifically kernel regression) were used to predict multiple sound variables representing the wind noise for a specific vehicle component [31]. In [32], the task of using extra predictive clustering tree ensembles to predict multi-output target has been analyzed. The experimental evaluation was considered over 41 datasets, that covered different application domains. A comparative analysis over three ensemble learning algorithms has been presented. It included extra predictive clustering trees, random forest and bagging method. Together with these, a comparative analysis between the ensemble methods and a single predictive model was presented. When it comes to the overall results, it was shown that the ensemble methods yield better results and outperform significantly a single predictive model. Comparing the ensemble methods alone, extra predictive clustering trees provided the best results over multiple datasets.

In [33], a high-level analysis of the learning models for structured outputs prediction is outlined. In this case, the problem is stated at a higher level of abstraction, and the target is defined as a generic structured output, which can be a vector, or a graph, or any structured data format, in contrast to the standard regression problems with single-value output. The authors mentioned that the multi-target prediction problem is related to multitask learning. In this paradigm, instead of having only one input source, multiple input sources can be used to predict a continuous or discrete variable, which can be a single value or multiple values. It is shown that the time complexity of a global model is lower than the complexity of local models due to a lot of repetitive sorting operations that are needed for local models (since multiple simple models will be used to predict each variable from the target output). On the other hand, the computational complexity is amplified for local models because they are bigger compared to a global model. Also, it was presented that the global methods are scalable even for a large dataset considering different dimensions, like the number of samples, the size of the input and output vectors. As noted in [33], multiple evaluation metrics have been considered for predicting multi-output continuous variables: root mean squared error, relative root mean squared error and correlation coefficient.

By looking at the analyzed state-of-the-art, there is strong experimental evidence that all these methods outperform classical statistics-based approaches for missing sensor data [10]–[16] while in other experiments, the classical statistics methods are not even considered [20]–[22] anymore. However, it remains unclear which method performs best in a wide range of AGV localization scenarios in industrial settings when the AGV is often exposed to perturbations such as light and humidity conditions. Table 1 summarizes the analyzed related work with brief description of used method, application area, missing data mechanism and the evaluation metrics used in each paper.

III. MACHINE LEARNING METHODS TO SUBSTITUTE MISSING LIDAR DATA

This article aims at solving the missing data (caused by natural design) from a LiDAR sensor, which is hosted on an AGV. The LiDAR sensor is set up on the bottom of the vehicle due to design constraints. An AGV is an autonomous vehicle that is used to accomplish specific tasks in an industrial manufacturing environment.

In Figure 1, the AGV is carrying its payload across an assembly line to the worker. The payload represents some

TABLE 1. Literature review summary.

Paper	Methods Used	Application Area	Missing Data Mechanism	Evaluation Metric
[9]	Machine Learning, Deep NN	Engineering Systems	MCAR and MAR	MSE, RMSLE (root mean squared logarithmic error), correlation coefficient, relative prediction accuracy
[10]	Mean imputation, Linear imputation, EM, KNN, Multivariate Imputations by Chained Equations (MICE), Sequence-to-Sequence Imputation Model (SSIM), Dual-SSIM, Recurrent NN	IoT – water monitoring system	MAR	RMSE, MAE
[11]	k-Means meta-predictor, KNN. RI, EM, Random Forest	IoT – Air pollution sensor data	MAR	RMSE, MAE, Coefficient of Determination
[12]	ARIMA, SARIMA, MF, MICE, EM, SSIM	IoT – Water quality sensor data	MCAR	RMSE, MAE, MAPE and SMAPE
[13]	Mean imputation, STL, exponential smoothing, autoregressive integrated moving average, linear regression, KNN, SVM, NN, VAR, Decision trees regressors, ensemble and boosting	Maritime industry sensor data	MCAR	Execution time, MSE, MSLE, RMSE, MAPE, MedAE, Max Error
[14]	linear regression, mean imputation, KNN, SVM, replace with 0	IoT - Building sensor data	MCAR	normalized RMSE, average normalized RMSE
[15]	Mean, PMM, bayesian PCA, KNN, Iterative KNN, meta predictor	Multiple datasets	MCAR, MNAR	RMSE, MAE
[16]	Mean, Mode, KNN, Random Forest, Deep Learning	Multiple datasets	MCAR, MAR, MNAR	RMSE, macro F1-score
[17]	Markov Chain Monte Carlo computations on a Bayesian Markovian hierarchical model	IoT - Smart house sensor data	MNAR	convergence of the Markov chains
[18]	Markov Chain Monte Carlo	Industrial IoT	MCAR and MAR	MSE
[19]	multiple linear regression	IoT – Wireless sensor network data	MCAR	normalized RMSE, mean relative error
[20]	VAE, Neural Networks, MICE, Random Forest, KNN	IoT – Calibration of particle and gas sensors	MAR	RMSE
[21]	Neural Network, PCA	Road traffic forecasting	MNAR	RMSE, MAPE
[22]	STL	Industrial IoT	MAR	RMSE, relative RMSE
[24]	Bayesian Maximum Entropy	IoT	MCAR	RMSE

modules required for the assembly of a modular tablet. The wheels of the AGV obturate significant regions of the LiDAR, preventing key landmark features to be observed. Our goal is to try to substitute the disruptive data, which is generated in front of the vehicle wheels. By leveraging the power of Digital Twins, we created a virtual counterpart of our physical setup using Gazebo, a robot simulation tool. By correlating real information from the AGV's sensors, especially LiDAR data, with information from its Digital Twin, we are hoping to increase the localization accuracy of our AGV. To make this possible, first we must estimate the missing data points from our physical AGV.

The approach that was taken into consideration during this work, is to try to predict the missing data using machine learning methods. Since the LiDAR sensor attached to the AGV will provide a set of numbers representing the distance, we are facing a multi-output regression problem. Using the simulator, the Digital Twin was able to provide a training dataset, but without the disruptive sensor information. In this way, the data can be split so that the sensor information, which is clean, will represent the input data for the machine learning problem, and the sensor information which will be disruptive in the real environment, will be considered as target data, the vector that must be predicted. In this context, supervised machine learning methods can be used to learn the relationship between relevant sensor information and the obstructed data, if there is such a connection.

FIGURE 1. AGV carrying the payload.

At first, a general mathematical perspective over the multi-output regression problem will be covered, together with the main two approaches for this problem, output transformation to apply single-output models, and adapting the models to predict a vector instead of a scalar value. Secondly, the main algorithms implemented in this work, will be covered in a mathematical form. Most of them are tree-based, that meaning decision tree, and then ensemble of trees using bagging or boosting techniques. Also, the KNN algorithm was analyzed. Most of the algorithms have been analyzed based on multiple criteria and hyper parameter tuning. The analysis of the results was done in a comparative way, multiple regression evaluation metrics were taken into consideration when comparing the models.

The evaluation is an important phase when it comes to machine learning models. There are certain metrics that are usually used for multi-output regression problems. A mathematical representation will be covered for each of the evaluation metrics used in the experimental phase.

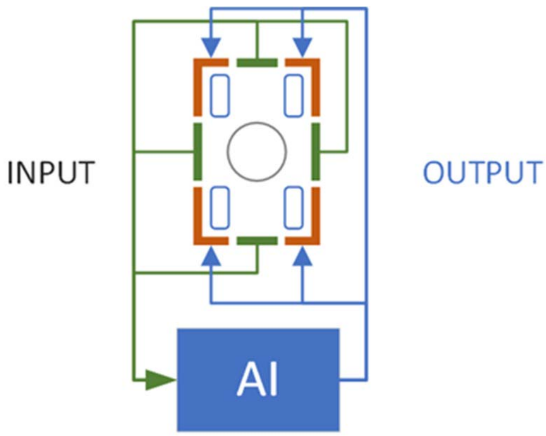

FIGURE 2. Workflow of the AI model.

Figure 2 depicts the AGV with LiDAR and its reading zones. The green zones are Li-DAR readings that are always visible, while the orange obturated zones need to be estimated. Our approach is composed of two phases. The first phase involves the use of data points acquired from the virtual environment to train our models to better estimate the missing data. The second phase represents the usage of the trained models in a real environment on the physical AGV to predict the obstructed zones.

Currently, the dataset is generated by simulation, where both green and orange areas are available for reading. To perform the simulations, we chose Gazebo as the AGV is developed on top of the Robot Operating System (ROS) platform. Gazebo allows us to simulate the real-world physics and reproduce our automated assembly system within the simulator. After replicating the real-world automated assembly system footprint in Gazebo (see Figure 3), we implemented the AGV kinematic model and a basic visual representation to help us with testing.

After the simulator setup was completed the ROS setup was required. We installed the packages for robot movement and visualization of LiDAR (RViz). The final step in the preparation was to create an automated flow for testing. First, we recorded the commands sent to the AGV by the user to achieve consistent behavior. Lastly, we prepared a script to be run after the testing was completed that resets the world and replays the recorded commands in sequence.

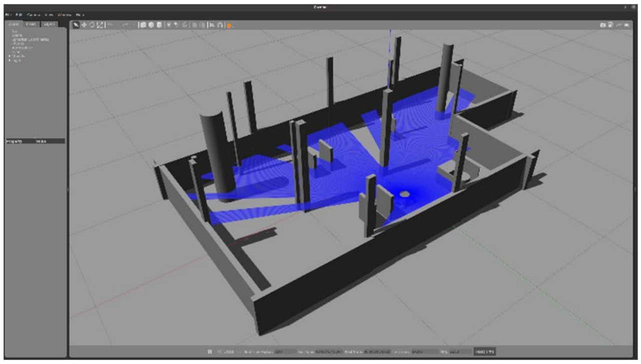

FIGURE 3. Simulated assembly system footprint in Gazebo.

Next, this section will cover several machine learning algorithms from a theoretical perspective. Most of the methods that will be covered are based on tree models. These methods were chosen because of their simplicity, ease of understanding and efficiency (good balance between complexity and running time). The analysis was started from very simple methods and continued building up models by adding one more layer of complexity on top with the goal of having a meaningful comparison between different tree-based methods by varying their tuning parameters. For each of these methods we will consider a training dataset D of N instances as $D = \{(X_1, Y_1), \ldots, (X_N, Y_N)\}$. Each predictor instance X_k consists of a set of n variables $X_k = \{x_1, \ldots, x_n\}$, and each output variable Y_k consists of a vector of m variables $Y_k = \{y_1, \ldots, y_m\}$, where m is the number of target variables with $k \in \{1, \ldots, N\}$.

A. REGRESSION TREES

Regression trees [34] are quite identical to classification trees, with the only difference being that the output variables will take a list of ordered values used to approximate real-valued functions, when classification trees output labels. The used regression tree algorithm is CART, where node impurity is calculated as the sum of squared deviations from the mean and the prediction of each node will be the sample mean from the Y variable.

The CART Algorithm:

1) Start by analyzing the top (root) node.
2) Taking into account each training instance X, determine S so that the sum of the impurities in the child nodes will be minimized, and choose the split X^* that will satisfy the minimum condition.
3) Detect if the stopping condition has been reached and exit. Otherwise, the step 2 will be applied for the child nodes.

Each class variable Y_k consists of a set of k possible values $Y_k = \{1, \ldots, d\}$ with $i \in \{1, \ldots, n\}$ and d representing the number of sets. The splitting criteria used will be the residual sum of squares (RSS), where $\hat{Y}$ is the mean of the observation in the set S_l:

$$RSS = \sum_{l=1}^{d} \sum_{i \in S_l} \left(Y_i - \hat{Y}\right)^2 \quad (1)$$

The goal is to find the sets in order to minimize the RSS. Since this is a computationally intensive operation, another method is usually used: recursive binary splitting.

The Recursive Binary Splitting Algorithm:

1) Apply a recursive greedy approach.
2) Split all observations into two branches at each level in the regression tree.
3) The best split is selected by using the greedy approach, without looking ahead.
4) Considering the splitting point s and a splitting variable k the following pair will be defined:

$$R_1(j, s) = \{X | X_j < s\} \quad (2)$$
$$R_2(j, s) = \{X | X_j \geq s\} \quad (3)$$

5) Then we find the best s and k variables that minimize the RSS for the regression tree:

$$RSS = \sum_{x_i \in R_1(j,s)} \left(Y_i - \hat{Y}_1\right)^2 + \sum_{x_i \in R_2(j,s)} \left(Y_i - \hat{Y}_2\right)^2 \quad (4)$$

where $\hat{Y}_a$ is the average of $\hat{Y}_i$ is the set S_l with $a \in \{1, 2\}$:

$$\hat{Y}_a = \frac{1}{n} \sum_{i \in S_l} Y_i \quad (5)$$

B. BAGGING

One of the main issues with machine learning methods such as regression trees is that they tend to produce a high variance. The variance, which is usually noted as σ^2, is a measure of how much each value is differing from the mean, and it is calculated in the following way: the difference between each value and the mean is determined, which will provide the distance from each value relative to the mean, then the square of each distance will be added together, and the sum will be divided to the number of values. The mathematical formula for this is the following:

$$\sigma^2 = \frac{1}{N} \sum_{i=1}^{N} \left(X_i - \hat{X}\right)^2 \quad (6)$$

where N is the number of samples in the dataset and $\hat{Y}$ is the mean of the variable X,

$$\hat{Y} = \frac{1}{N} \sum_{i=1}^{N} X_i \quad (7)$$

The standard deviation is the square root of the variance, and it is represented in a mathematical way as follows:

$$\sigma = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \left(X_i - \hat{X}\right)^2} \quad (8)$$

One of the solutions for reducing the variance issue for the regression trees is called bagging [35], or bootstrap aggregation. Let us consider a set X of independent random variables, $X = \{X_1, \ldots, X_N\}$, with N observations, and each of them with variance σ^2. If we compute the variance of X the result will be:

$$Var(X) = \frac{\sigma^2}{N} \quad (9)$$

Based on this, we can say that averaging a set of random observations will end up reducing the variance. In this way, a solution for a better prediction accuracy is to create multiple separate prediction models and then average the results of the predictions. But then, the next question comes: how do you get a set of separate training sets in order to have multiple separate models? The answer is to take repeated samples from the same training set and use them in order to have multiple separated models. This process is called bootstrapping.

We will consider M different training sets from the original training dataset, D_i^*, with $i \in \{1, \ldots, M\}$. Let us define f^i as being the function representing the prediction for the training set D_i^*. So, in order to apply the bagging procedure, we have to proceed with the following steps:

1) Calculate f^i for the separate train sets.

$$f^i(x), \ldots, f^M(x) \quad (10)$$

2) Apply the bagging step to determine the average result and improve the high variance problem.

$$f^* = \frac{1}{M} \sum_{i=1}^{M} f^i(x) \quad (11)$$

C. RANDOM FOREST

Random forest methods [36] are very similar to the bagging methods, but with a small difference in the way the trees are selected. Similar to the bagging approach, the bootstrapping technique will be used to select the trees, but instead of using all the variables from the input vector, only a random subset will be used when the split is performed.

The split, in the random forest case, will use only a subset of X_k instance, which will be selected randomly, with p number of variables, $q \in \{1, \ldots, p\}$:

$$X'_k = \{x_1, \ldots, x_q, \ldots, x_p\} \quad (12)$$

where, $X'_k \subseteq X_k$, $p \approx \sqrt{n}$ and $p < n$.

When building the random forest model, p must be strictly lower than n, and in most of the cases it will consider a minority of all the available variables in the input set. This is the main idea of the random forest model, compared to bagging. Since it will select all the time a random sample of variables, it will not be affected so hard by very strong predictors, which will decrease the number of highly correlated trees and the reduction in variance will be higher.

D. BOOSTING

Boosting is also a method based on decision tree which tries to improve the overall predictions. Boosting is a technique that can be used in multiple machine learning scenarios, either for classification or for regression problems. Boosting models work in a very similar way to bagging models, but instead of building separate decision trees, it will grow them sequentially. Instead of creating multiple trees from the same dataset, each tree will be created based on the information from previous decision trees. The model is fitted on a modified state of the initial dataset, instead of creating separate subsets, like in the bootstrapping procedure. We choose two boosting algorithms, adaptive boosting and gradient boosting, as described next.

The adaptive boosting (AdaBoost) algorithm variation as described in [37]:

1) Assign weight to each training set instance, $w_i = 1$, $i \in \{1, \ldots, N\}$.
2) Compute probability of training instance i being in the training set, $p_i = \frac{w_i}{\sum_{j=1}^{N} w_i}$. Pick a training set by selecting a number in the interval $\left[0, \sum_{j=1}^{N} w_i\right]$. The selected number corresponding to i will determine the training set.
3) Create regression tree from the selected training set. The model will define a hypothesis: $h_t : X \rightarrow y$.
4) Fit the model through the training set and get predictions, $y_i^p(X)$ with $i \in \{1, \ldots, N\}$.
5) Compute loss: $L_i = 1 - \exp\left[\frac{-|y_i^p(X_i)-y_i|}{D^2}\right]$ where $D = \sup\left|y_i^p(X_i) - y_i\right|$ with $i \in \{1, \ldots, N\}$.
6) Compute average loss: $\bar{L} = \sum_{i=1}^{N} L_i p_i$.
7) Compute measure of confidence: $\beta = \frac{\bar{L}}{1-\bar{L}}$.
8) Update weights: $w_i \rightarrow w_i \beta^{1-L_i}$;
9) At each step, the decision tree will make a prediction for input X, h_t with $t \in \{1, \ldots, B\}$ and B being the number of trees. The cumulative prediction based on B is $h_f = inf\left\{y \in Y : \sum_{t:h_t \leq y} \log\left(\frac{1}{\beta_t}\right) \geq \frac{1}{2}\sum_t \log\left(\frac{1}{\beta_t}\right)\right\}$. Each tree t will have a prediction and a measure of confidence associated to it. The predictions will be relabeled in the following way $y_{1i} < \cdots < y_{ti} < \cdots < y_{Bi}$, with $t \in \{1, \ldots, B\}$. Then, $\log\left(\frac{1}{\beta_t}\right)$ will be summed up until the inequality will be satisfied for the smallest t and the prediction for t will the considered as the ensemble prediction.
10) If the stopping criterion is not reached, repeat from the step 2, otherwise exit. The stopping criterion is based on the average loss, which should be less than a threshold.

The gradient boosting algorithm based on Friedman's [38] proposal is:

1) Set $\widehat{f_0}$ to a constant value, where $\widehat{f}$ is the estimate function and $\widehat{f_0}$ is the initial estimation.
2) For $k \in \{1, \ldots, B\}$:
   2.1) Calculate negative gradient $g_k(x)$, where $g_k(x) = E_y\left[\frac{\partial \mathcal{L}(y,f(X))}{\partial f(X)}\middle| X\right]_{f(X)=\widehat{f_{k-1}}(x)}$ with E_y referring to the expectation over the whole dataset and $\mathcal{L}(y, f)$ being the generic loss function.
   2.2) Train the new learner $h(X, \theta_k)$.
   2.3) Determine the best step size for the iteration:
   $$\rho_k = arg\min_{\rho} \sum_{i=1}^{N} cL\left[y_i, \widehat{f_{k-1}}(X_i) + \rho h(X_i, \theta_k)\right]$$
   2.4) Update the learner estimate function $\hat{f_k} \leftarrow \widehat{f_{k-1}} + \rho_k h(X, \theta_k)$.
   2.5) If the stopping criterion is not reached, repeat from the step 3, otherwise exit.

E. KNN

The KNN algorithm is one of the basic machine learning methods. A detailed mathematical approach to KNN is described in [39]. The main idea is to find the nearest data point, in the training dataset, for the new data point that must be classified or determined. KNN can be applied both for regression and classification problems. If it is a classification problem, the most frequent value in the k-nearest data points can be used as the prediction value, and if it is a regression problem, then the mean on the k-nearest values can be used. In order to find the k-nearest data point, the algorithm will calculate the distance between the two data points.

The KNN algorithm does not actually involve a training stage, the distance between the current data point and the other data points from the dataset will be calculated at the prediction time. So, the y target variable will not be used in the training part.

If we consider the current data point $X^* = \left(x_1^*, \ldots, x_k^*, \ldots, x_n^*\right)$, for each data point X_i, the distance $\mathcal{D}(X^*, X_i)$ will be calculated, with $i \in \{1, \ldots, N\}$. There are multiple methods for calculating the distance:

- Euclidian Distance,
$$\mathcal{D}\left(X^*, X_i\right) = \sqrt{\sum_{k=1}^{n} \left(x_k^* - x_{ij}\right)^2} \tag{13}$$
- Manhattan Distance,
$$\mathcal{D}\left(X^*, X_i\right) = \sum_{k=1}^{n} \left|x_k^* - x_{ij}\right| \tag{14}$$
- Chebyshev Distance,
$$\mathcal{D}\left(X^*, X_i\right) = max_{k=1}^{n} \left|x_k^* - x_{ij}\right| \tag{15}$$
- Minkowski Distance,
$$\mathcal{D}\left(X^*, X_i\right) = \left(\sum_{k=1}^{n} \left|x_k^* - x_{ij}\right|^p\right)^{\frac{1}{p}} \tag{16}$$
where p is a parameter that can be defined in the algorithm.

There is not a procedural algorithmic approach to define the value of the k parameter. It can be chosen based on some research analysis for a particular problem. Also, the over-fitting issue must be taken into consideration for KNN. One valid solution is to use cross-validation, based on a separate validation training set. Usually, the default value is set to 5, as in the sklearn Python module.

IV. EXPERIMENTAL METHODOLOGY AND RESULTS

This section presents how we obtained the dataset, the structure of the dataset, the evaluation metrics and an analysis of the performance for the proposed machine learning models and their parameter tunning.

A. METHODOLOGY

This subsection explains how the data was gathered and presents the metrics used in our models' evaluation.

The dataset used for training and evaluation was generated based on Gazebo [40], a virtual environment simulator. It is a robot simulation tool which offers accurate and efficient simulations that is using the same Robotic Operating System (ROS) [41] platform as the physical AGV prototype. The training and evaluation data was generated in a static environment. The vehicle was moving along, generating about 30 thousand frames of data. Each frame corresponds to a vector of 720 entries representing the distance from the vehicle to the closest object or wall in the static environment. The angle in which the distance was measured is defined by the order in the vector, so the first entry is the distance corresponding to zero angle degrees, the second entry is the distance corresponding to 0.5 angle degrees, and so on until the last element in the vector corresponding to 359.5 angle degrees.

In summary, the dataset can be represented as a data table, with 30200 entries, and 720 columns, where the columns represent the angle (in degrees) in which the distance was measured.

All the training and testing operations have been executed on a Linux machine with the following configuration: 16GB of memory and Intel(R) Core(TM) i5¬6500 CPU with 4 cores clocked at 3.20GHz.

Let's consider the test dataset D_t with N number of instances, $D_t = \left\{\left(X_1^{(t)}, Y_1^{(t)}\right), \ldots, X_N^{(t)}, \left(Y_N^{(t)}\right)\right\}$. Each instance $X_i^{(t)}$ consists of a vector on n variables $X_i^{(t)} = (x_1, \ldots, x_k, \ldots, x_n)$, and each target instance $Y_i^{(t)}$ consists of a vector of m variables $Y_i^{(t)} = (y_1, \ldots, y_l, \ldots, y_m)$ with $k \in \{1, \ldots, n\}$, $l \in \{1, \ldots, m\}$ and $i \in \{1, \ldots, N\}$. We will consider $\hat{Y}_i$ to be the predicted vector for the input instance $X_i^{(t)}$.

According to recent surveys on missing data in machine learning [28], [42], the most popular evaluation methods are Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Square Error (RMSE) and Area Under the Curve (AUC). To evaluate our methods, we choose two of them: MAE and RMSE. The evaluation metrics will be used to compare the actual vector $Y_i^{(t)}$ and the predicted vector $\hat{Y}_i$:

- MAE is the mean of the absolute difference between the actual and predicted value for all instances in the dataset [43].

$$\mathrm{MAE}\left(Y^{(t)}, \hat{Y}\right) = \frac{1}{N}\sum\nolimits_{i=1}^{N}\left|Y_i^{(t)} - \hat{Y}_i\right| \qquad (17)$$

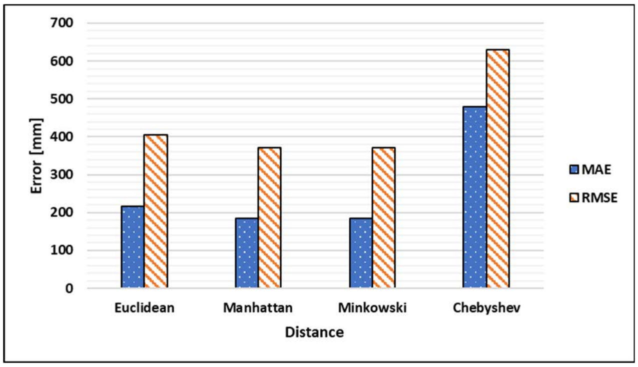

FIGURE 4. Distances used by KNN with 3 neighbors and the kd_tree algorithm.

- RMSE returns the square root of the mean squared error. As the residual error increases, RMSE will penalize more compared to MAE.

$$\mathrm{RMSE}\left(Y^{(t)}, \hat{Y}\right) = \sqrt{\frac{1}{N}\sum\nolimits_{i=1}^{N}\left(Y_i^{(t)} - \hat{Y}_i\right)^2} \qquad (18)$$

B. EXPERIMENTAL RESULTS

For the experimental results we are using the dataset presented in the previous subsection. First, we will present the fine-tunning of the hyperparameters for the proposed methods and afterwards, a comparison of these methods in terms of performance will be provided.

The first algorithm we used is the KNN implementation using KNeighborsRegressor of the scikit-learn Python module. This implementation has several parameters that can be fine-tuned: number of neighbors, metric (distance) and the algorithm to compute the nearest neighbors. The library provides three choices in terms of algorithm selection to compute the nearest neighbors: BallTree (ball_tree), KDTree (kd_tree) and the brute algorithm. We evaluated these algorithms with the default Minkowski distance and 3 neighbors. The evaluations show that the used algorithm does not have any meaningful impact on our results. However, if we are to compare the time required for training and prediction, the kd_tree algorithm is the best choice, with only 61 seconds necessary in the evaluation phase.

The next parameter is the distance formula used by the algorithm to compute the distance between data points. In Figure 4 it can be observed that the Minkowski distance has the best results, with the Manhattan distance performing similarly. The Chebyshev distance generates the worst results, with significantly higher errors compared to any other distance formula.

From Figure 5 it can be observed that the optimal number of neighbors for the KNN algorithm is 3 if both MAE and RMSE are taken into consideration. However, if we consider only the MAE, then a KNN with only 1 neighbor seems to be the best performing algorithm.

The Decision Trees are also implemented using the same module. Compared to all the other algorithms presented, these have a higher number of parameters that can be tuned.

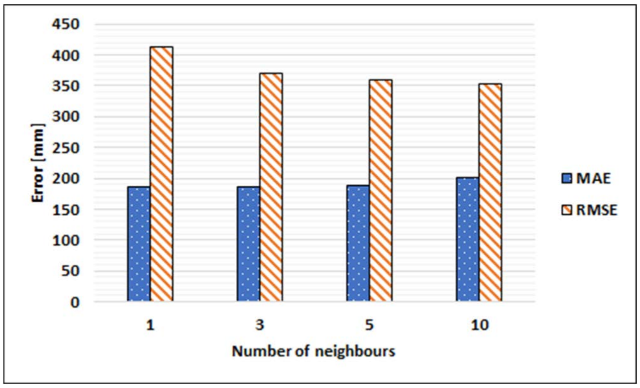

FIGURE 5. Number of neighbors for KNN with Minkowski distance and kd_tree algorithm.

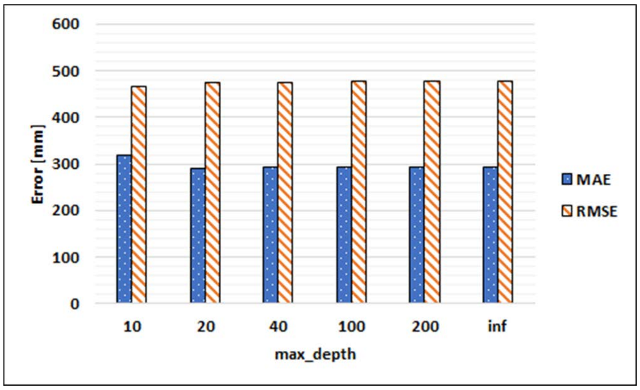

FIGURE 6. Max_depth for the decision tree regressor using min_samples_split = 8.

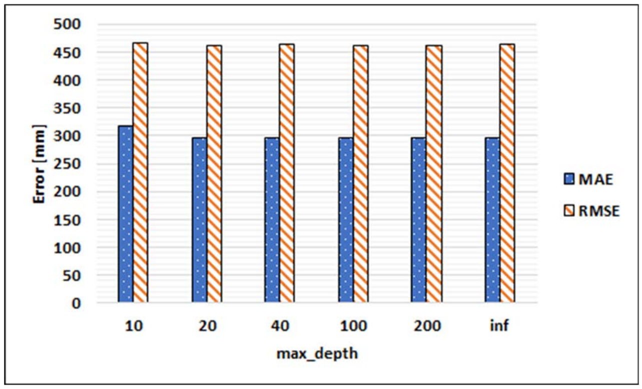

FIGURE 7. Max_depth for the decision tree regressor using min_samples_split = 16.

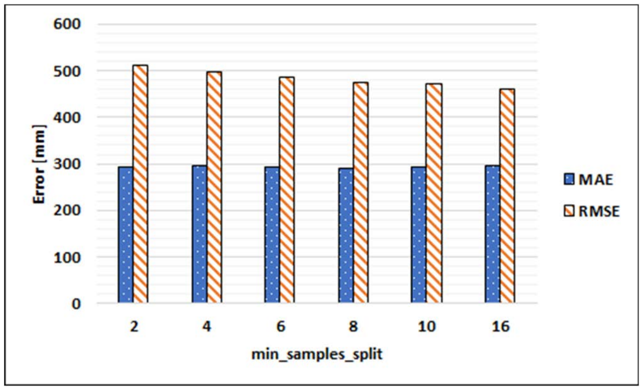

FIGURE 8. Varying min_samples_split for the decision tree regressor using max_depth = 20.

The splitter parameter represents the way in which a split is performed in each node. There are two split options: random or best. Using random splits yields mediocre results compared to using the best split, which has significantly higher results. The second and third parameters that we can vary are max_depth and min_samples_split, which, as presented in the following figures, are highly correlated. As the name suggests, the max_depth parameter represents the maximum depth of the tree and, if the default value – *None* – is used, it will extend the tree until its leaves are pure. The min_samples_split parameter represents the minimum number of samples required before a split operation on a node is performed.

In Figures 6 and 7, different values for depth have been analyzed for both min_samples_split being 8 and 16, respectively. In both cases, there is a substantial improvement by increasing the depth from 10 to 20, and almost no difference for higher depth values. A maximum depth of 20 nets us the best results, with an MAE of 291.5 when using a min_samples_split of 8 and 295.42 when min_samples_split is 16.

Using a max_depth of 20, we tried to identify the optimum value for the min_samples_split variable. As it can be observed in Figure 8, there is little to no improvement of MAE using a different min_samples_split value, a value of 8 representing the lowest error of 291.5. However, if we look at RMSE, we can observe a significant improvement, obtaining an error of 461.52 if we use a min_samples_split of 16.

Thus, the optimal Decision Tree configuration consists of a max_depth of 20 and a min_samples_split of 16.

The next analyzed algorithm is the Bagging Regressor implementation of the scikit-learns module. One parameter was varied for this algorithm which is the number of estimators that should be used in the ensemble process. As it can be observed in Figure 9, the more estimators are used, the lower the errors will be. However, with each additional estimator used, the training time will increase exponentially.

The Random Forest model is based on the bagging method, in which a subset of the input vector will be used when choosing a training sample from the dataset. Like all the other presented methods, it is implemented using the scikit-learn package.

For this algorithm we varied the number of estimators, representing the number of trees in our algorithm, and the maximum depth of the trees. As it is depicted in Figure 10, better results are obtained with more trees and a depth of 100 is the best choice since there is no significant increase in the overall results if we are using a higher depth than 100.

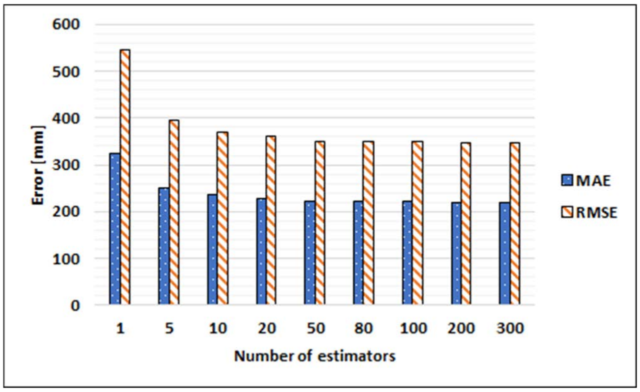

FIGURE 9. Number of estimators used for bagging regressor.

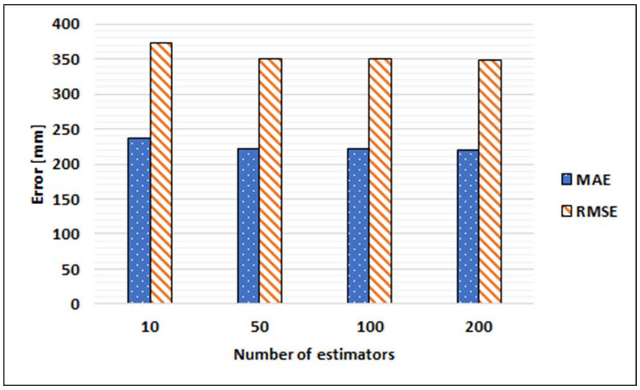

FIGURE 10. The number of estimators for random forest regressor with max_depth = 200.

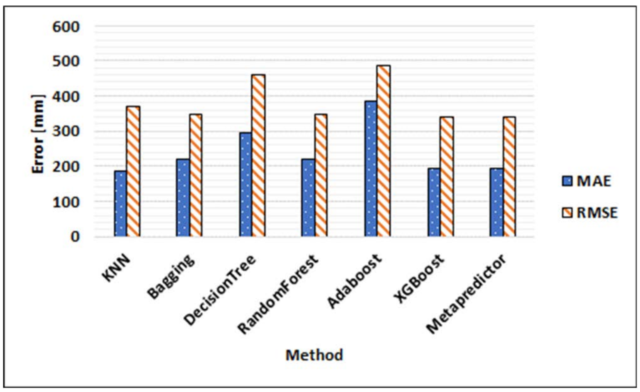

FIGURE 11. Results of all machine learning models used.

Increasing the number of the trees will lead to increasingly high training times.

Now that we presented all the algorithms and their optimal parameters, let us evaluate how they perform comparatively. Additionally, we tested the AdaBoost and XGBoost implementations of Adaptive Boosting and Gradient Boosting from the skicit-learns module.

From Figure 11 it can be observed that the best performing algorithm in terms of MAE is the KNN model. If RMSE is to be considered, which is more sensitive to outliers compared to MAE, it can be seen a shift in the best performing models, with the XGBoost model being the top performing algorithm,

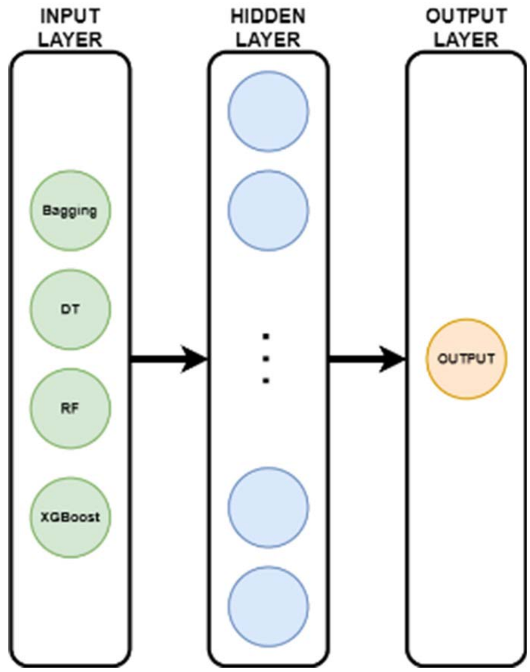

FIGURE 12. The structure of the neural network.

TABLE 2. Running times.

Model	Running time [ms]	Resource allocation [MB]
DecisionTree	12	4.4
Bagging	19783	5336
Random Forest	1391	3559
XGBoost	17071	210
AdaBoost	56996	3.9
KNN	67802	54.3
Metapredictor	19833	9109

followed by Random Forest and the Bagging models. KNN has slightly worse performance compared to Random Forest and Bagging. AdaBoost is the weakest performing model.

We also tried to aggregate the best performing models into a metapredictor. We did not consider the AdaBoost model due to its high error and running time and the KNN model due to its high running time (see Table 2). Thus, the predictor consisted of a neural network composed of 1 input layer (4 neurons), 1 hidden layer (12 neurons by default), and one output neuron, as depicted in Figure 12.

All these neurons are fully connected, and the activation function used is ReLU since it is able to provide output in the $[0, \infty)$ interval. The input values represent the prediction of each algorithm for a specific cloud point and the output represents the estimated distance. The network was trained for 850 epochs. Figure 13 presents the effect of variating the number of hidden layer neurons. As we can observe, with 12 neurons we obtained the best results, but without significant differences. The metapredictor, failed to improve the overall results of the models, being outperformed by XGBoost.

Table 2 presents the total running time and the resource allocation on the whole testing dataset for each considered

TABLE 3. Wilcoxon test.

Data	Real Data	Decision Tree	Bagging	Random Forest	XGBoost	AdaBoost	KNN
Real Data	1	0	0	0	0.44	0	0
Decision Tree	0	1	0	0	0	0	0
Bagging	0	0	1	0.83	0	0	0
Random Forest	0	0	0.83	1	0	0	0
XGBoost	0.44	0	0	0	1	0	0
AdaBoost	0	0	0	0	0	1	0
KNN	0	0	0	0	0	0	1

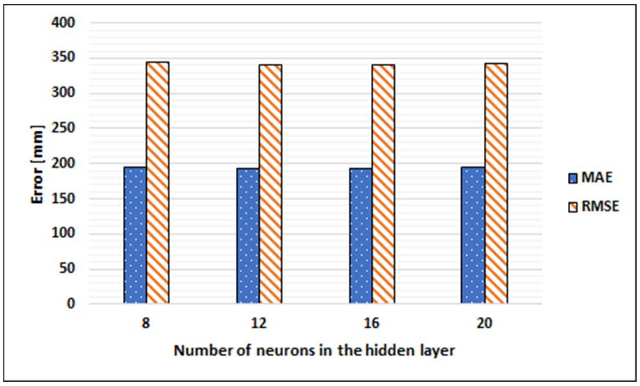

FIGURE 13. Variating the number of hidden layer neurons.

model. The Bagging, Random Forest and the metapredictor were the most inefficient in terms of resource allocation.

Table 3 presents the p-value obtained with the Wilcoxon test. The results show that the differences in performance between the methods are significant. Unsignificant difference was observed only between the Bagging and Random Forest methods. According to the Wilcoxon test, the results of the XGBoost method are the closest to the real data.

V. CONCLUSION AND FURTHER WORK

This work has provided a research analysis over multiple machine learning methods for a multi-output regression problem. It presented a theoretical perspective on machine learning algorithms focusing on the substitution of missing LiDAR data for better AGV localization. Then, the MAE and RMSE evaluation metrics were provided. Most of the algorithms that were used are based on decision trees and ensemble trees, including bagging methods and boosting techniques. More exactly, the random forest, adaptive boosting and gradient boosting algorithms were considered. Another model that was analyzed is KNN, which is based on a simple search over the whole dataset, and a distance measurement.

The evaluation was performed on a dataset obtained through simulation using Gazebo and ROS. By analyzing the results, the XGBoost algorithm has provided the best performance. On the other hand, a simple decision tree model seems to give good results if a tradeoff between performance and prediction time has to be made. The KNN algorithm is also performing pretty well, especially because we are experimenting in a static environment.

As further research, we plan to evaluate deep neural networks considering the temporal nature of our dataset. Another direction is to try to predict the next frame and check if it is a more optimal solution instead of the prediction of the missing data in the current frame. As our dataset was generated using a static environment, we also plan to generate a new dataset, which not only will be larger but will also incorporate the dynamics of a real environment. The sensor fusion approach is also analyzed.

ACKNOWLEDGMENT

The authors express their thanks to Nicolae Daniel Pop and Radu Trimbitas for their help provided in the statistical evaluation.

REFERENCES

[1] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, Feb. 2007, doi: 10.1109/TRO.2006.889486.
[2] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *Proc. IEEE Int. Symp. Saf., Secur., Rescue Robot.*, Nov. 2011, pp. 155–160, doi: 10.1109/SSRR.2011.6106777.
[3] W. Hess, D. Kohler, H. Rapp, and D. Andor, "Real-time loop closure in 2D LiDAR SLAM," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2016, pp. 1271–1278, doi: 10.1109/ICRA.2016.7487258.
[4] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017, doi: 10.1109/TRO.2017.2705103.
[5] C. Debeunne and D. Vivet, "A review of visual-LiDAR fusion based simultaneous localization and mapping," *Sensors*, vol. 20, no. 7, p. 2068, 2020, doi: 10.3390/s20072068.
[6] T.-H. Kim and T.-H. Park, "Placement optimization of multiple LiDAR sensors for autonomous vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 2139–2145, May 2020, doi: 10.1109/TITS.2019.2915087.
[7] M. Sualeh and G.-W. Kim, "Dynamic multi-LiDAR based multiple object detection and tracking," *Sensors*, vol. 19, no. 6, p. 1474, 2019, doi: 10.3390/s19061474.
[8] D. B. Rubin, "Inference and missing data," *Biometrika*, vol. 63, no. 3, pp. 581–592, 1976, doi: 10.2307/2335739.
[9] C. A. Leke and T. Marwala, *Deep Learning and Missing Data in Engineering Systems*. Cham, Switzerland: Springer, 2019, doi: 10.1007/978-3-030-01180-2.
[10] Y. Zhang and P. J. Thorburn, "Handling missing data in near real-time environmental monitoring: A system and a review of selected methods," *Future Gener. Comput. Syst.*, vol. 128, pp. 63–72, Mar. 2022, doi: 10.1016/j.future.2021.09.033.

[11] B. Agbo, H. Al-Aqrabi, R. Hill, and T. Alsboui, "Missing data imputation in the Internet of Things sensor networks," *Future Internet*, vol. 14, no. 5, p. 143, May 2022, doi: 10.3390/fi14050143.
[12] Y.-F. Zhang, P. J. Thorburn, W. Xiang, and P. Fitch, "SSIM—A deep learning approach for recovering missing time series sensor data," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6618–6628, Aug. 2019, doi: 10.1109/JIOT.2019.2909038.
[13] C. Velasco-Gallego and I. Lazakis, "Real-time data-driven missing data imputation for short-term sensor data of marine systems. A comparative study," *Ocean Eng.*, vol. 218, Dec. 2020, Art. no. 108261, doi: 10.1016/j.oceaneng.2020.108261.
[14] A. Chong, K.-P. Lam, W. Xu, O.-T. Karaguzel, and Y. Mo, "Imputation of missing values in building sensor data," in *Proc. ASHRA IBPSA-USA SimBuild Building Perform. Modeling Conf.*, 2016, vol. 6, no. 1, pp. 407–414. [Online]. Available: https://ideaslab.io/publication/chong-2016-missing/
[15] D. Bertsimas, C. Pawlowski, and Y. D. Zhuo, "From predictive methods to missing data imputation: An optimization approach," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 7133–7171, 2017. [Online]. Available: https://dl.acm.org/doi/abs/10.5555/3122009.3242053
[16] S. Jäger, A. Allhorn, and F. Bießmann, "A benchmark for data imputation methods," *Frontiers Big Data*, vol. 4, p. 48, Jul. 2021, doi: 10.3389/fdata.2021.693674.
[17] I.-H. Dinwoodie, "Missing sensor network data," *Commun. Statist., Case Stud., Data Anal. Appl.*, vol. 5, no. 2, pp. 146–152, 2019, doi: 10.1080/23737484.2018.1563513.
[18] L. Ehrlinger, T. Grubinger, B. Varga, M. Pichler, T. Natschlager, and J. Zeindl, "Treating missing data in industrial data analytics," in *Proc. 13th Int. Conf. Digit. Inf. Manage. (ICDIM)*, Sep. 2018, pp. 148–155, doi: 10.1109/ICDIM.2018.8846984.
[19] X. Yan, H. Xie, and W. Tong, "A multiple linear regression data predicting method using correlation analysis for wireless sensor networks," in *Proc. Cross Strait Quad-Regional Radio Sci. Wireless Technol. Conf.*, Jul. 2011, pp. 960–963, doi: 10.1109/CSQRWC.2011.6037116.
[20] N. U. Okafor and D. T. Delaney, "Missing data imputation on IoT sensor networks: Implications for on-site sensor calibration," *IEEE Sensors J.*, vol. 21, no. 20, pp. 22833–22845, Oct. 2021, doi: 10.1109/JSEN.2021.3105442.
[21] G. Boquet, A. Morell, J. Serrano, and J. L. Vicario, "A variational autoencoder solution for road traffic forecasting systems: Missing data imputation, dimension reduction, model selection and anomaly detection," *Transp. Res. C, Emerg. Technol.*, vol. 115, Jun. 2020, Art. no. 102622, doi: 10.1016/j.trc.2020.102622.
[22] Y. Liu, T. Dillon, W. Yu, W. Rahayu, and F. Mostafa, "Missing value imputation for industrial IoT sensor data with large gaps," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 6855–6867, Aug. 2020, doi: 10.1109/JIOT.2020.2970467.
[23] C. Robert, C. William, and T. Irma, "STL: A seasonal-trend decomposition procedure based on loess," *J. Off. Statist.*, vol. 6, no. 1, pp. 3–73, 1990.
[24] A. Gonzalez-Vidal, P. Rathore, A. S. Rao, J. Mendoza-Bernal, M. Palaniswami, and A. F. Skarmeta-Gomez, "Missing data imputation with Bayesian maximum entropy for Internet of Things applications," *IEEE Internet Things J.*, vol. 8, no. 21, pp. 16108–16120, Nov. 2021, doi: 10.1109/JIOT.2020.2987979.
[25] S. Royo and M. Ballesta-Garcia, "An overview of LiDAR imaging systems for autonomous vehicles," *Appl. Sci.*, vol. 9, no. 19, p. 4093, Sep. 2019, doi: 10.3390/app9194093.
[26] C. Bohak, M. Slemenik, J. Kordež, and M. Marolt, "Aerial LiDAR data augmentation for direct point-cloud visualisation," *Sensors*, vol. 20, no. 7, p. 2089, Apr. 2020, doi: 10.3390/s20072089.
[27] C. Sazara, R. V. Nezafat, and M. Cetin, "Offline reconstruction of missing vehicle trajectory data from 3D LiDAR," in *Proc. IEEE Intell. Vehicles Symp. (IV)*, Jun. 2017, pp. 792–797, doi: 10.1109/IVS.2017.7995813.
[28] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression," *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 5, no. 5, pp. 216–233, Sep. 2015, doi: 10.1002/widm.1157.
[29] D. Kocev, S. Džeroski, M. D. White, G. R. Newell, and P. Griffioen, "Using single- and multi-target regression trees and ensembles to model a compound index of vegetation condition," *Ecol. Model.*, vol. 220, no. 8, pp. 1159–1168, Apr. 2009, doi: 10.1016/j.ecolmodel.2009.01.037.
[30] Z. Han, Y. Liu, J. Zhao, and W. Wang, "Real time prediction for converter gas tank levels based on multi-output least square support vector regressor," *Control Eng. Pract.*, vol. 20, no. 12, pp. 1400–1409, Dec. 2012, doi: 10.1016/j.conengprac.2012.08.006.
[31] D. Kuznar, M. Martin, and B. Ivan, "Curve prediction with kernel regression," in *Proc. 1st Workshop Learn. Multi-Label Data*, 2009, pp. 61–68.
[32] D. Kocev, M. Ceci, and T. Stepišnik, "Ensembles of extremely randomized predictive clustering trees for predicting structured outputs," *Mach. Learn.*, vol. 109, no. 11, pp. 2213–2241, Nov. 2020, doi: 10.1007/s10994-020-05894-4.
[33] D. Kocev, C. Vens, J. Struyf, and S. Džeroski, "Tree ensembles for predicting structured outputs," *Pattern Recognit.*, vol. 46, no. 3, pp. 817–833, Mar. 2013, doi: 10.1016/j.patcog.2012.09.023.
[34] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*, 1st ed. Evanston, IL, USA: Routledge, doi: 10.1201/9781315139470.
[35] L. Breiman, "Bagging predictors," *Mach. Learn.*, vol. 24, no. 2, pp. 123–140, 1996, doi: 10.1007/BF00058655.
[36] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001, doi: 10.1023/A:1010933404324.
[37] H. Drucker, "Improving regressors using boosting techniques," in *Proc. ICML*, vol. 97, 1997, pp. 107–115.
[38] J. H. Friedman, "Greedy function approximation: A gradient boosting machine," *Ann. Statist.*, vol. 29, no. 5, pp. 1189–1232, Oct. 2001, doi: 10.1214/aos/1013203451.
[39] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 1, pp. 21–27, Jan. 1967, doi: 10.1109/TIT.1967.1053964.
[40] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, vol. 3, Oct. 2004, pp. 2149–2154, doi: 10.1109/IROS.2004.1389727.
[41] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source robot operating system," in *Proc. ICRA Workshop Open Source Softw.*, 2009, vol. 3, no. 3, p. 5.
[42] T. Emmanuel, T. Maupong, D. Mpoeleng, T. Semong, B. Mphago, and O. Tabona, "A survey on missing data in machine learning," *J. Big Data*, vol. 8, no. 1, p. 140, Dec. 2021, doi: 10.1186/s40537-021-00516-9.
[43] C. Sammut and G. I. Webb, "Mean absolute error," in *Encyclopedia of Machine Learning*. Boston, MA, USA: Springer, 2010, doi: 10.1007/978-0-387-30164-8_525.

ARPAD GELLERT received the M.Sc. and Ph.D. degrees in computer science from the Lucian Blaga University of Sibiu, in 2003 and 2008, respectively. He worked as a Visiting Researcher in Barcelona and Milano. Previously, he was a Java Developer at Multimedia Capital Romania. He is currently working as an Associate Professor with the Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu. He developed as a Project Manager a research grant supported by the Romanian National Council of Academic Research and four internal grants. He published five books and over 50 scientific papers in some prestigious journals and international top conferences and acquired more than 300 citations. His research interests include computer architecture, smart buildings and factories, web mining, and image processing. He was a member of several research grants. He is also a member of an ongoing Hasso Plattner Excellence Research Grant. He received in 2010 the "Ad Augusta Per Angusta" prize awarded by the Lucian Blaga University of Sibiu for Excellence in Scientific Research. More information can be found at: http://webspace.ulbsibiu.ro/arpad.gellert.

DARIUS SARBU received the B.Sc. and M.Sc. degrees in computer science from the Lucian Blaga University of Sibiu, in 2019 and 2021, respectively. He is currently working as a Software Developer at Elrond Network. Previously, he worked as a Machine Learning Engineer at Visma. His research interests include artificial intelligence, blockchain technologies, and operating systems.

STEFAN-ALEXANDRU PRECUP received the B.Sc. degree in computer science, in 2020. He is currently pursuing the M.Sc. degree in computer science with the Lucian Blaga University of Sibiu.

Since 2019, he has been working as a Research Assistant at the Lucian Blaga University of Sibiu. Previously, he worked for two years as a Full-Stack Software Developer at Transylvania Labs. His research interests include blockchain technologies, artificial intelligence, and collaborative systems.

ALEXANDRU MATEI (Graduate Student Member, IEEE) received the B.Sc. and M.Sc. degrees in computer science from the Lucian Blaga University of Sibiu, in 2017 and 2019, respectively, where he is currently the Ph.D. degree in computer science.

Since 2018, he has been working as a full-time Research Assistant at the Lucian Blaga University of Sibiu, being involved as a member of several research grants. He contributed to multiple conference and journal papers. He also participated to various summer schools and contests. Previously, he worked for two years as an iOS Mobile Software Developer at KeepCalling, Sibiu. His research interests include digital twins, robotics, artificial intelligence, and human–computer interaction. He is a member of the INCON Research Center.

DRAGOS CIRCA is currently pursuing the M.Sc. degree in advanced computer science with the Lucian Blaga University of Sibiu. He worked with the Same University at the DiFiCIL Project—''Development of Cyber-Physical Social Systems Based on the Internet of Things for the Factory of the Future'' and also working on another research project within a team of researchers both as a Developer and as a Research Assistant. He received multiple awards in various computer science related competitions and published a paper regarding human interaction with HoloLens–AR head-mounted device (HMD). He participated at BigDat 2018, a winter school in Cambridge, U.K., regarding usage and collection of large amounts of data. He also participated in a one-month internship program in Heilbronn, Germany, to further explore VR technologies and applications in medical and industrial fields and to a winter school in Barcelona to study robotics.

CONSTANTIN-BALA ZAMFIRESCU received the Ph.D. degree in automation and control from the ''Politehnica'' University Bucharest, in 2007. Since 1998, he has been with the Computer Science and Electrical Engineering Department, Lucian Blaga University of Sibiu, where he is currently leading as a Full Professor with the INCON Research Center. He also worked as an Invited Researcher in Austria, Spain, Belgium, and Germany, participating as a principal investigator in many international projects. He is trained in foresight methods by UNIDO. His current research interests include multi-agent systems, cyber physical social systems, and decision support systems. He is a member of IFAC TC 5.4 ''Large Scale and Complex Systems'' and IEEE TC on Computational Collective Intelligence.