Combining Classification and Regression for WiFi Localization of Heterogeneous Robot Teams in Unknown Environments

Benjamin Balaguer, Gorkem Erinc, and Stefano Carpin

***Abstract*— We consider the problem of team-based robot mapping and localization using wireless signals broadcast from access points embedded in today's urban environments. We map and localize in an unknown environment, where the access points' locations are unspecified and for which training data is a priori unavailable. Our approach is based on an heterogeneous method combining robots with different sensor payloads. The algorithmic design assumes the ability of producing a map in real-time from a sensor-full robot that can quickly be shared by sensor-deprived robot team members. More specifically, we cast WiFi localization as classification and regression problems that we subsequently solve using machine learning techniques. In order to produce a robust system, we take advantage of the spatial and temporal information inherent in robot motion by running Monte Carlo Localization on top of our regression algorithm, greatly improving its effectiveness. A significant amount of experiments are performed and presented to prove the accuracy, effectiveness, and practicality of the algorithm.**

I. Introduction

As a result of the evident necessity for robots to localize and map unknown environments, a tremendous amount of research has focused on implementing these primordial abilities. Localization problems have been extensively studied and a variety of solutions have been proposed, each assuming different sensors, robotic platforms, and scenarios. The increasingly popular trend of employing low-cost multi-robot teams [14], as opposed to a single expensive robot, provides additional constraints and challenges that have received less attention. A tradeoff naturally arises, because reducing the number of sensors will effectively decrease the robots' price while making the localization problem more challenging. We anticipate that team-based robots will require WiFi technology to exchange information between each other. We also foresee robots will continue to supply rough estimations of local movements, via odometry or similar inexpensive low accuracy sensors. These team-based robots have the advantage of being very affordable. It is clear, however, that these robots would not be practical in unknown environments due to their lack of perception abilities and, as such, we embrace an heterogeneous setup pairing a lot of these simple robots with a single robot capable of mapping an environment by traditional means (e.g., SLAM using a laser range finder or other sophisticated proximity sensors). Within this scenario, our goal is to produce a map of an unknown environment in real-time using the more capable robot, so that the less sophisticated robots can localize themselves.

Given the sensory constraints imposed on the robots, we exploit wireless signals from Access Points (APs) that have become omnipresent in today's urban infrastructure. Wireless signals are notoriously difficult to work with due to non-linear fading, obstructions, and multipath effects, but a localizer based on WiFi offers its share of advantages. Indeed, WiFi APs are uniquely identifiable, can be used indoor or outdoor, and are already part of the unknown environment to be mapped. Additionally, transmitted signals are available to anyone within range, allowing robots to exploit APs without ever connecting to them. We cast the problem of WiFi localization as a machine learning problem, which we initially solve using classification and regression theory. Then, thanks to a Monte Carlo Localization (MCL) step, we improve the localizer by exploiting the spatial and temporal information inherently encoded within the robots' odometry. In this manuscript we offer the following contributions:

- we implement and contrast six algorithms, three previously published and three unpublished in the WiFi localization literature, that solve the problem cast as classification;
- we propose a novel regression algorithm that builds upon the best classification algorithm;
- we develop an end-to-end WiFi localization algorithm that adds odometry for robustness, implemented as MCL;
- we outline crucial design choices with respect to the algorithm's applicability to unknown environments and real-time performance;
- we evaluate and compare numerous algorithms across different datasets, which is, to the best of our knowledge, the first attempt to unequivocally establish the best method for WiFi localization.

The rest of the paper is organized as follows. Section II highlights previous work on WiFi localization, covering both the classification and regression approaches. In section III we describe the problem statement and introduce mathematical definitions used throughout the paper. Since each component of our localizer depends on the results of the previous one, we structure the algorithm's description somewhat unorthodoxly. More specifically, we present our first component, classification, in Section IV-A followed directly by its results in Section IV-B. In Section V-A, our findings from classification are exploited for our second component, regression, the results of which are provided in Section V-B. We finalize the algorithm in Section VI-A, presenting the results in Section VI-B. Final remarks and future work conclude the paper in Section VII.

School of Engineering, University of California, Merced, CA, USA, {bbalaguer,gerinc,scarpin}@ucmerced.edu.

II. Related Work

The utilization of wireless signals transmitted from APs or home-made sensors has enjoyed great interest from the robotics community, in particular for its applicability to the localization problem. In this area, the application of data-driven methods has prevailed thanks to two distinct but popular approaches. On one hand, the modeling approach attempts to understand, through collected data, how the signal propagates under different conditions, and the goal is to generate a signal model that can then be exploited for localization. On the other hand, the mapping approach directly uses collected data by combining spatial coordinates with wireless signal strengths to create maps from which a robot can localize. Since signals are distorted due to "typical wave phenomena like diffraction, scattering, reflection, and absorption" [3], practical implementations of signal modeling are not yet available for unknown environments since they need to be trained in similar conditions to what will be encountered (i.e., they require at least some a-priori information about the environment) [21]. Moreover, it has been shown in [11] that a signal strength map should yield better localization results than a parametric model. Consequently, the rest of this section highlights related works regarding the mapping approach, which we employ for our algorithm. Interested readers are referred to [9] for more information on signal modeling techniques.

During a training phase, signal mapping techniques tie a spatial coordinate with a set of observed signal strengths from different APs, essentially creating a signal map. Given a new set of observed signal strengths acquired at an unknown coordinate, the goal is to use the map to retrieve the correct spatial coordinate. A variety of methods have been devised to solve this problem. Two of the earliest solutions used nearest neighbor searches [1] and histograms of signal strengths for each AP [20]. Starting from the observation that histograms of signal strengths are generally normally distributed [8], various other methods have recently been proposed using Gaussian distributions to account for the inevitable variance in received signal strengths [11], [15], [2]. Specifically, for each location and each AP, a Gaussian distribution is derived from training data. An unknown location described by a new set of observed signal strengths is then determined using Bayesian filtering. These methods exploit the inherently available spatial and temporal information of a moving robot through a Hidden Markov Model (HMM) [15] or MCL [2], which, unlike the HMM, requires additional sensory feedback (e.g., odometry). Another recent approach, from the wireless network community, exploits Support Vector Machines by formulating the problem, similarly to what we propose in this manuscript, as a classification instance [19].

For completeness, we briefly mention the works of Duvallet et al. [6], Ferris et al. [7], and Huang et al. [12] that are based on Gaussian processes. Unfortunately, they require a long parameter optimization step that makes them difficult to use in our scenario, where real-time operation is required and computational power limited.

III. System Setup and Problem Definition

We consider a team of r robots, one of which is a "sensor-full" robot, the *mapper*, capable of building a map by traditional means (e.g., SLAM, GPS). The remaining $r-1$ are "low-cost robots", the *localizers*, whose only sensors are odometry and a WiFi card. The goal is to develop a system where the *mapper* creates a WiFi map that the *localizers* can successfully use to localize, strictly utilizing their limited sensors. For experimental purposes, we use a MobileRobots P3AT equipped with an LMS200 Laser Range Finder (LRF) as our *mapper* and various iRobot Create platforms as our *localizers*. The *mapper* robot uses *gmapping* [10] to solve the SLAM problem. All of the code runs on a typical consumer laptop, without relying on multiple cores, GPUs, or extensive memory. Our approach is split into a mapping phase, involving the *mapper*, and a localization phase involving the *localizers*. During the mapping phase, the *mapper* periodically collects WiFi signal strengths from all the APs in range and associates them with current Cartesian coordinates provided by *gmapping*. In terms of notation, for every sample Cartesian position $C_p = [X_p, Y_p]$, an observation $Z_p = [z_p^1, z_p^2, \dots, z_p^{|a|}]$ is acquired using a WiFi card, where $|a|$ is the total number of APs seen throughout the environment. Each signal strength z_p^a is measured in *dBm*, the most commonly provided measurement from hardware WiFi cards that typically range from -90 to -10 dBm with lower values (e.g., -90) indicating worst signal strengths [13]. We note that since the environment's APs cannot all be seen from a single location, the observation vector, Z_p, is dynamically increased as the robot moves in the environment and identifies previously unseen APs. Additionally, since some APs cannot be seen from certain locations, we set $z_p^a = -100$ for any AP a that cannot be seen from location p. In order to obtain an indication of the signal strength noise and increase the algorithm's robustness, we collect multiple observations at each location, resulting in a vector of observations for each location, $\mathbf{Z}_p = [Z_p^1, Z_p^2, \dots, Z_p^{|s|}]$, where $|s|$ is the total number of observations performed at each location p. The entire data used to build a complete WiFi map in real-time, acquired by the *mapper*, can then be represented as a matrix T of locations and observations

$$T = \begin{bmatrix} C_1 & Z_1^1 & Z_1^2 & \dots & Z_1^{|s|} \\ C_2 & Z_2^1 & Z_2^2 & \dots & Z_2^{|s|} \\ & & \dots & & \\ C_{|p|} & Z_{|p|}^1 & Z_{|p|}^2 & \dots & Z_{|p|}^{|s|} \end{bmatrix}$$

where $|p|$ is the total number of positions for which WiFi signals were acquired. Therefore, T is comprised of $|s|$ observations for each of the $|p|$ locations and each of the $|a|$ access points in our map, where each observation is labeled with its Cartesian coordinates acquired from *gmapping*, C_p. A typical sample of the data collected for the WiFi map is shown in Figure 1.

Two parameters are evidently important, the number of locations to consider for the WiFi map, $|p|$, and the number of observations performed at each of those locations, $|s|$.

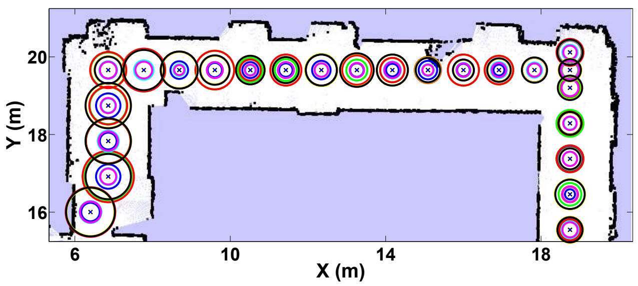

Fig. 1. Illustrative representation of the WiFi readings for 7 APs, superimposed on the map. The size of each ring represents the AP's signal strength (the bigger, the better) and each color indicates a unique AP.

We have no control over the parameter $|a|$, the total number of APs, because it is dictated by the structure of the unknown environment. Evidently, $|p|$ will be proportional to the mapped environment's size and dependent on the measurements' density. We have experimentally found, as have authors of [2], [15], that recording observations approximately every meter yields accurate maps, but no restrictions are placed on the locations' alignment (i.e., they do not need to be grid-aligned or follow any particular structure). The parameter $|s|$ encompasses an important tradeoff that we analyze in Section IV-B since the higher the $|s|$, the better the noise model but the longer the mapping process. We conclude this section by mentioning two important observations from analyzing the data collected for this paper. First, we have found signal strength readings to be consistent across different robots having similar hardware, a fact corroborated by [11] that implies the possibility of sharing WiFi maps amongst robots. Second, signal strength measurements taken at different days or times were, although different, not significantly so, indicating that WiFi maps acquired at a certain time can still be effective for robots operating in the same environment at a later time.

IV. Classification

A. Description

Figure 1 gives a good qualitative indication that it is possible to differentiate between different locations by only considering signal strengths received from APs. We start developing the WiFi localizer by casting it as a machine learning classification problem. Mathematically, we produce a function $f : Z \rightarrow p$ from the training data T acquired by the *mapper*. Function f takes a new observation $Z = [z^1, \ldots, z^{|a|}]$ acquired by one of the *localizers* and returns the location p of the robot, from which we can easily look up the Cartesian coordinate, C_p. We note that casting the problem in such manner is a simplification, since a robot needs to be positioned exactly at one of the locations in the training data in order for f to return the exact coordinate C_p (i.e., classification does not perform interpolation or regression). Solving the classification problem is nevertheless an important step since the effectiveness of various algorithms can be evaluated and applied to build a better localizer, as will be shown in the next sections. Computing f from T can be achieved using different techniques and to the best of our knowledge, no comparison has been made in the literature to find the best performing algorithm. Consequently, we implement a total of six algorithms, three of which have been published in the WiFi localization literature (Gaussian Model, Support Vector Machine, and Nearest Neighbor Search) and three others that have enjoyed popularity in machine learning tasks (Decision Tree, Random Forest, and Multinomial Logit). We briefly present each algorithm in the context of our problem's definition, the description of which are kept short due to space restrictions, and provide references for interested readers looking for more details.

Decision Tree [5]: a decision tree is a binary tree constructed automatically from the training data T. Each node of the tree corresponds to a decision made on one of the input parameters, z_p^a, that divides the node's data into two new subsets, one for each of the node's sub-trees, in such a way that the same target variables, p, are in the same subsets. The process is iterated in a top-down structure, working from the root (whose data subset is T) down to the leaves, and stops when each node's data subset contains one and only one target variable or when adding new nodes becomes ineffective. Various formulas have been proposed to compute the "best" partitioning of a node's data subset, the most popular of which being the Gini coefficient, the twoing rule, and the information gain. After experimental assessments, we found the choice of partitioning criterion insignificant in terms of localization accuracy and use, arbitrarily, the Gini coefficient. In order to classify a new observation Z, the appropriate parameter z^i is compared at each node, the decision of which dictates which branch is taken - a step that is repeated until a leaf is reached. The target variable p at the leaf is the location of the robot.

Random Forest [4]: random forests are an ensemble of decision trees built to reduce over fitting behaviors often observed in single decision trees. This is achieved by creating a set of $|d|$ trees, as described in the previous paragraph, each with a different starting dataset, $\hat{T_d}$, selected randomly with replacement from the full training data T. An additional difference comes from the node splitting process, which is performed by considering a random set of $|q|$ input parameters as opposed to all of them. The partitioning criterion is still used, but only the $|q|$ randomly selected parameters are considered. In order to classify a new observation Z, it is processed by each of the $|d|$ trees, resulting in $|d|$ output variables p_d, some of which may be the same. In some sense, each decision tree in the forest votes for an output variable and the one with the most votes is chosen as the location of the robot. It is important to note that a lot of information can be extracted from this voting scheme, since the votes can trivially be converted to the robot's probability of being at a particular location, $P(p|Z) = V_p/|d|$, where V_p is the total number of votes received for location p with Cartesian coordinate C_p. The number of trees $|d|$ encompasses a tradeoff between speed and accuracy. We set $|d| = 50$ after determining that it yields the highest ratio of accuracy to the number of trees.

Gaussian Model [11], [15], [2]: the Gaussian model technique, proposed in the robotics literature, attempts to model the inherent noise of signal strength readings through a Gaussian distribution. For each location p and for each AP a the mean and standard deviation of the signal strength readings are computed, yielding μ_p^a and σ_p^a, respectively. This means that a total of $|p| \times |a|$ Gaussian distributions are calculated, where all μ_p^a and σ_p^a are computed from $|s|$ values (i.e., the total number of observations performed at each location). The location, p, of a new observation, Z, is derived using the Gaussian's probability density function:

$$\arg\max_p \left[\prod_{a=1}^{|a|} \frac{1}{\sigma_p^a \sqrt{2\pi}} \exp\left(-\frac{(z^a - \mu_p^a)^2}{2(\sigma_p^a)^2} \right) \right]$$

Support Vector Machine [19]: support vector machines work by constructing a set of hyperplanes in such a way that they perfectly divide two data classes (i.e., they perform binary classification). Generating the hyperplanes is essentially an optimization problem that maximizes the distance between the hyperplanes and the nearest training point of either class. Although initially designed as a linear classifier, the usage of kernels (e.g., polynomial, Gaussian) enables non-linear classification. Since we are looking to divide our training data T into $|p|$ classes, we use a support vector machine variant that works for multiple classes. We chose the one-versus-all approach (we empirically determined it was better than one-versus-one), which consists in building $|p|$ support vector machines that each try to separate one class p from the rest of the other classes. This essentially creates a set of $|p|$ binary classification problems, each of which is solved using the standard support vector machine algorithm with a Gaussian kernel. Once all the support vector machines are trained, we can localize given a new observation Z by evaluating its high-dimensional position with respect to the hyperplanes, for each support vector machine p. The class p is chosen by the support vector machine that classifies Z with the greatest distance from its hyperplanes.

Nearest Neighbor Search [1]: the nearest neighbor search does not require any processing of the training data, T. Instead, the distance between all the points in T and a new observation Z is computed, yielding $|p| \times |s|$ distances D_p^s. The location p of the new observation is then selected with the formula: $\arg\min_p \left(D_p^s\right)$. The only necessary decision for this algorithm involves the choice of one of the numerous proposed distance formulas (e.g., Euclidean, Mahalanobis, City Block, Chebychev, Cosine). We use Euclidean distance to rigorously follow the algorithm presented in [1].

Multinomial Logit [16]: multinomial logit is an extension of logistic regression allowing multiple classes to be considered. Logistic regression separates two classes linearly by fitting a binomial distribution to the training data (i.e., one class is set as a "success" and the other class as a "failure"). This generalized linear model technique produces a set of $|a| + 1$ regression coefficients β_i that are used to calculate the probability of a new observation being a "success". Conversely to the multi-class support vector machine, multinomial logit follows a one-against-one methodology where one class p_{ref} is trained against each of the remaining classes. Consequently, $|p| - 1$ logistic regressions are trained, yielding $|p| - 1$ sets of regression coefficients, β_i^p. Once all the regression coefficients are learned, the probability of being at location p given a new observation Z can be calculated as follows:

$$P(p|Z) = \frac{\gamma^p}{1 + \sum_{j=1}^{|p|-1} \exp(Z \cdot \beta^j)}, \; j \neq p_{ref}$$

where $\gamma^p = \exp(Z \cdot \beta^p)$ when $p \neq p_{ref}$ and $\gamma^p = 1$ when $p = p_{ref}$. The location p can then be selected as the maximum probability by utilizing the formula $\arg\max_p P(p|Z)$.

B. Results

The experimental results of the WiFi localizer cast as a classification problem greatly influenced our algorithmic design choices for the end-to-end algorithm. Therefore, we present the results before continuing with the algorithm's description. Specifically, we gathered a large indoor training dataset at our university, stopping the robot at pre-determined locations approximately 1 meter apart. The entire dataset is comprised of 156 locations ($|p| = 156$), 20 signal strength readings for each location ($|s| = 20$), and a total of 48 unique APs ($|a| = 48$). The results shown in this and subsequent sections are performed by sub-sampling the entire dataset. Specifically, $|s|$ is varied from 1 to 19, essentially representing a percentage of the dataset being used for training, and the remaining data is used for classification. This procedure effectively mimics inevitable differences between data acquired by the *mapper* and *localizers*. Moreover, the training and classification data are randomly sampled 50 different times for each experiment in order to remove any potential bias from a single sample. Presented results are averages of those 50 samples and we omit error bars in our graphs since the results' standard deviation were all similar and insignificant. We note, once again, that this process does not represent a real-world scenario since it assumes the *mapper* and *localizers* follow the exact same path. It provides, however, a very good platform for comparing the six presented classification algorithms in Section IV-A.

We start by analyzing each algorithm's accuracy as a function of the number of readings taken at each location, $|s|$, a plot of which is shown in Figure 2. Since our goal is to ultimately be able to map unknown environments in real-time, the initial data acquisition performed by the *mapper* needs to be efficient. It takes on average 411ms to get signal strength readings from all the APs in range so we want to minimize $|s|$. As expected, Figure 2 shows a tradeoff between the algorithms' localization accuracy and the number of readings used during the training phase. Since we want to limit the time it takes to gather the training data, setting the number of readings per location to 3 ($|s| = 3$) provides a good compromise between speed and accuracy, especially since the graph shows an horizontal asymptote starting at or around that point for the best algorithms. As such, the rest of the results presented in the paper will be performed using 3 readings per location.

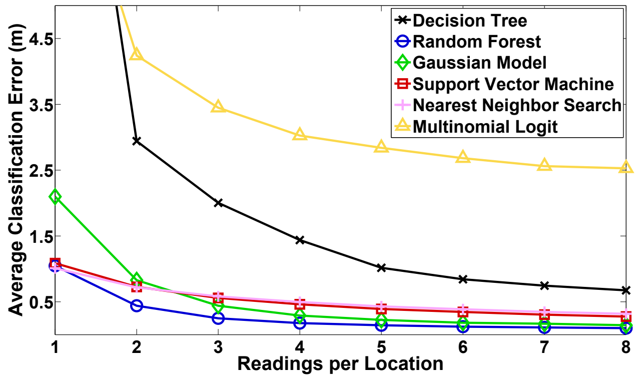

Fig. 2. Average classification error, in meters, for each classification algorithm as a function of increasing number of readings per location ($|s|$).

On average, when $|s| = 3$, the Random Forest's classification error is 43.42%, 55.47%, and 57.23% better than the previously published WiFi localization algorithms (Gaussian Model, Support Vector Machine, and Nearest Neighbor Search, respectively). Figure 3 shows the cumulative probability of classifying a location within the error margin indicated on the x-axis. This figure corroborates the findings of Figure 2, showing that the best algorithm is the Random Forest, which had never been exploited in the context of WiFi localization. Moreover, the Random Forest can localize a new observation, Z, to the exact location (i.e., zero margin of error) 88.58% of the time.

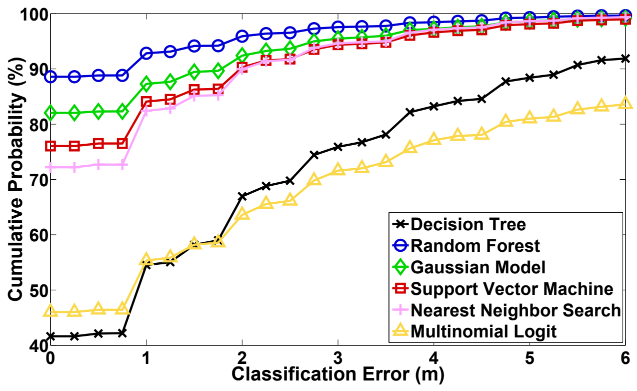

Fig. 3. Cumulative probability of correctly localizing as a function of increasing error margin, for each classification algorithm.

In addition to our dataset, we use a publicly available dataset provided by Ladd et al. [15], with data covering 3 floors of the computer science building at Rice University. We process the Rice dataset in the exact same manner as ours, dividing it in 50 random samples of training and classification data for $|s| = 3$. The average error for each algorithm and each dataset is shown in Figure 4. There are a couple of interesting observations that can be made from these results. Firstly, the trend is the same for both datasets. In other words, the list of the algorithms from best to worst average accuracy (i.e., Random Forest, Gaussian Model, Support Vector Machine, Nearest Neighbor Search, Decision Tree, and Multinomial Logit) is the same regardless of which dataset is used. Secondly, the Rice dataset uses signal to interference ratios for their observations as opposed to signal strengths. Although both measures are loosely related, this indicates that the classification algorithms are both general and robust. These important findings indicate that the methods proposed are data- and environment-independent and that we are not over fitting our particular dataset. Please note that the overall higher average classification error observed in the Rice dataset comes from sparser location samples: 3.33 meters on average as opposed to 0.91 meters for our dataset.

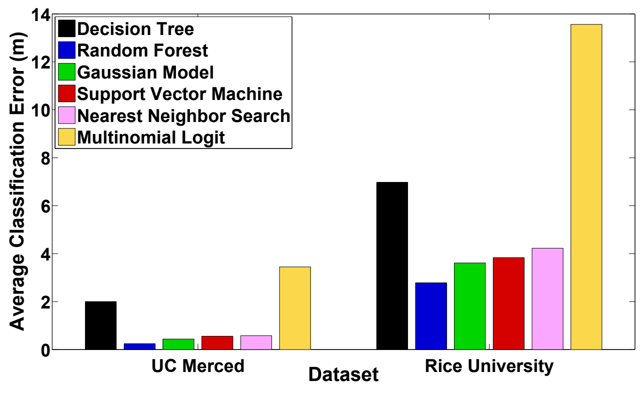

Fig. 4. Average classification error, in meters, for each classification algorithm executed on the UC Merced and Rice University datasets.

We conclude this section by mentioning that, when recording 3 readings per location, all of the algorithms can be trained in less than 30 seconds, apart from the Support Vector Machine that takes 90 seconds. In addition, the training process can be parallelized for all the algorithms, adding a potential decrease in the computation of the WiFi map. Put differently, once the *mapper* has acquired its training data, the map can be built in less than 30 seconds, which is very fast relative to the time it takes to explore the environment. Once the WiFi map is created and given to the *localizers*, they can localize in less than 200ms. Needless to say that the presented algorithms are very effective, from a computational standpoint, even in unknown environments where map building needs to be performed in real-time.

V. Regression

A. Description

Although the results of the classification algorithms are very encouraging, they do not depict a real world scenario, where locations explored by the *mapper*, upon which the WiFi map will be created, will surely be different than those explored by the *localizers*. Consequently, we re-cast the WiFi localizer as a regression problem, where some inference is performed to generate Cartesian coordinates for locations that are not encompassed in the training data. Although typical off-the-shelf regression algorithms (e.g., Neural Network, Radial Basis Functions, Support Vector Regression) might seem like a good choice initially, they get corrupted by the nature of our training data, the majority of which depicts unseen APs (i.e., $z_p^a = -100$). In addition, it would be wise to exploit the good results exhibited by the classification algorithms. We consequently design our own regression algorithm that builds upon the best classification algorithm: the Random Forest. In addition to providing the best classification results, the Random Forest is appealing due to its voting scheme, which can be interpreted as $P(p|Z)$, for each $p \in T$.

Our regression algorithm is based on a Gaussian Mixture Model (GMM) [17] described in Algorithm 1. The GMM is introduced to non-linearly propagate, in the two-dimensional Cartesian space, results acquired from the Random Forest. More specifically, we build a GMM comprised of $|p|$ mixture components (line 1). Each mixture component is constructed from a Gaussian distribution with a mean $\mu(p)$ (line 2), corresponding to the Cartesian coordinates in the training data, covariance $\Sigma(p)$ (line 3), and mixture weights $\phi(p)$ that are acquired directly from the Random Forest's voting scheme (line 4). Line 4 highlights the key difference between the classification and regression methodologies, which arises from the fact that taking the mode of the Random Forest's results, as is done in classification, discards valuable information that is instead exploited in our regression algorithm. In some sense, the mixture weights are proportional to the Random Forest's belief of being at location p given the observation Z (i.e., $\phi(p) = P(p|Z)$).

Algorithm 1 Construct-GMM(Z)

```
1: for $p \leftarrow 1$ to $|p|$ do
2:     $\mu(p) \leftarrow C_p$
3:     $\Sigma(p) \leftarrow \sigma^2 I$
4:     $\phi(p) \leftarrow P(p|Z) \leftarrow$Random-Forest-Predict($Z$)
5: end for
6: return $gmm \leftarrow$Build-GMM($\mu$, $\Sigma$, $\phi$)
```

Algorithm 2 provides pseudo-code for the rest of the regression algorithm. Once the GMM is built (line 1), a couple of approaches are available, the most popular of which consists in taking the weighted mean of the model or drawing samples from the GMM with probability $\phi(p)$ and computing the samples' weighted mean. Instead of sampling from the GMM, our regression algorithm uses a k Nearest Neighbor Search (line 2) to provide k samples that are not only dependent on the observation Z, but also come from a different model than the Random Forest. This is a crucial step that adds robustness to the algorithm by combining two of the presented classification algorithms. In other words, where and when one algorithm might fail, the other might succeed. The choice of the Nearest Neighbor Search for this step (as opposed to the Gaussian Model or the Support Vector Machine) comes from the fact that $|s|$ times more samples can be drawn from it (a total of $|s| \times |p|$, as opposed to $|p|$). The returned Cartesian coordinate (line 3) is finally calculated as the weighted mean of the k Nearest Neighbors, where the weight of each Nearest Neighbor is set from the Probability Distribution Function (PDF) of the GMM.

The entire regression algorithm requires two parameters to be set, σ (Algorithm 1) and k (Algorithm 2). σ dictates how much the Gaussian components influence each other and should be approximately set to the distance between WiFi readings in the training data. k should be as high as possible in order to provide a lot of samples, yet low enough not to incorporate "neighbors" that are too far away. We have found setting k to 25% of all the observations in T (i.e., $k = 0.25 \times |s| \times |p|$) to be a good solution for this tradeoff.

Algorithm 2 Regression(T, Z)

```
1: $gmm \leftarrow$ Construct-GMM($Z$) // See Algorithm 1
2: $nn \leftarrow$k-NN($T$, $Z$, $k$)
3: $\hat{C} \leftarrow \frac{\sum_{i=1}^{k} nn_i \times \text{PDF}(gmm, nn_i)}{\sum_{i=1}^{k} \text{PDF}(gmm, nn_i)}$
4: return $\hat{C}$
```

Given a new observation Z, the regression essentially maps a three-dimensional surface to the X-Y Cartesian space of the environment, where the higher the surface's Z-value the more probable the X-Y location. A representative snapshot of the process is shown in Figure 5, highlighting an important behavior of the presented regression algorithm. Indeed, the algorithm is not only capable of generating Cartesian coordinates that were not part of the initial training data, but also takes into consideration neighboring votes from the Random Forest classification. In the figure, the highest Random Forest vote (represented by '*') is somewhat isolated and, consequently, its neighbors do not contribute to the overall surface. The region close to the actual robot's location (denoted by '+'), however, is comprised of many local neighbors whose combined votes outweigh those of the Random Forest classification, resulting in a better regression estimation (symbolized by 'x').

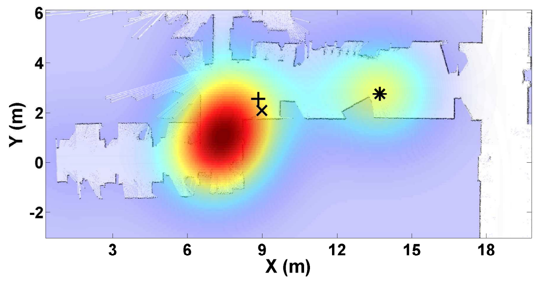

Fig. 5. Regression example, showing the PDF of the GMM overlaid on top of a section the environment's map. The markers +, x, and * represent the ground truth, regression, and random forest classification locations, respectively.

B. Results

In order to evaluate the accuracy of the regression algorithm, it is necessary to gather new data that will mimic a *localizer* exploring the environment and localizing it with the regression algorithm trained on T. Consequently, we perform 10 new runs (N_i), assuming no prior knowledge of the locations covered by T. In other words, while the *mapper* data (T) and the *localizer* data (N) approximately cover the same environment, they are not acquired at the same Cartesian coordinates. As such, this experiment provides a direct measure of the regression algorithm's strength. For evaluation purposes, we manually record the robot's ground truth position at discrete locations so that we can quantitatively evaluate the regression algorithm. Figure 6 shows the average accuracy (from the 10 runs and 50 random samples used for training) of the 4 best classification algorithms compared against the regression algorithm. Not surprisingly, the regression algorithm works very well by outperforming the Random Forest, Gaussian Model, Support Vector Machine, and Nearest Neighbor Search classification algorithms by 29.77%, 37.95%, 34.07%, and 28.23%, respectively. We conclude this section by mentioning that the regression algorithm only takes 131ms to localize.

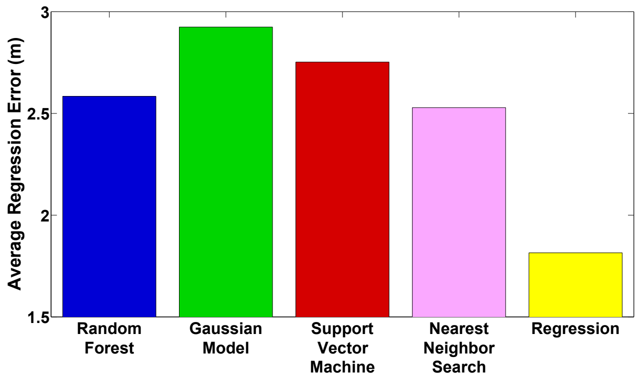

Fig. 6. Average regression error, in meters, comparing the 4 best classification algorithms (left) and the regression algorithm (right).

VI. Monte Carlo Localization

A. Description

Although the regression algorithm clearly improves the WiFi localizer's accuracy for real-world scenarios, further improvements can be obtained by taking into account the spatial and temporal constraints implicitly imposed by robot motion. In other words, the classification and regression algorithms discussed so far solve the localization problem without taking into consideration the robot's previous location, or, more precisely the probability distribution of the robot's previous location. Specifically, we use an MCL algorithm, built from the "standard" implementation presented in [18], with a couple of modifications. The motion model, which exploits the translational and rotational velocities of the robot, is exactly the same as in [18]. The measurement model is, evidently, different since it needs to take into account the aforementioned WiFi localizer regression. The pseudo-code is presented in Algorithm 3 and shows the effectiveness of our GMM implementation (line 1), which can seamlessly transition from regression to MCL. Indeed, the measurement model needs to assign each particle (line 2) the probability of being at the particle's state given the sensor measurement Z. Thanks to the GMM, the particle's weight is easily retrieved by using the PDF (line 3).

Algorithm 3 Measurement-Model(Z)

```
1: $gmm \leftarrow$ Construct-GMM($Z$) // See Algorithm 1
2: for $m \leftarrow 1$ to $|m|$ do
3:     $m$.weight $\leftarrow$ PDF($gmm$, $m$.state($X$,$Y$))
4: end for
```

In addition to the measurement model, we modify the particles' initialization procedure. Instead of randomly or uniformly sampling the particles' state and giving each particle an equal weight, we force the robot to perform a measurement reading, Z, and initialize the particles using that measurement, as shown in Algorithm 4. We start by constructing the GMM (line 1) and, for each particle in our filter (line 2), sample a data point from the GMM that will serve as the particle's X,Y state (line 3). Since WiFi signal strengths cannot infer the robot's rotation, we randomly sample the particle's θ state (line 4). The particles' weight is calculated from the GMM's PDF (line 5). In order to add robustness to the previously mentioned problem with robots' rotations, we pick the $|v|$ best particles (line 7) and add y new particles that are the same as v but have y different θ states each randomly sampled between 0 and 2π (line 8). Since we have augmented the total number of particles, we finalize our particle initialization by keeping the best $|m|$ particles (line 9).

Algorithm 4 Initialize-Particles(Z)

```
1: $gmm \leftarrow$ Construct-GMM($Z$) // See Algorithm 1
2: for $m \leftarrow 1$ to $|m|$ do
3:     $m$.state($X$,$Y$) $\leftarrow$ sample($gmm$)
4:     $m$.state($\theta$) $\leftarrow$ rand(0 to $2\pi$)
5:     $m$.weight $\leftarrow$ PDF($gmm$, $m$.state($X$,$Y$))
6: end for
7: for all $m$ with the top $|v|$ weights do
8:     Add $y$ particles $n^y$ where
           $n^y \leftarrow m$, and
           $n^y$.state($\theta$) $\leftarrow$ rand$^y$(0 to $2\pi$)
9: end for
10: Select the $|m|$ particles with highest weights
```

B. Results

In order to evaluate the MCL, we take advantage of the same experimental data used for the regression results. Namely, the algorithm is trained on T and we localize the robot for each of the 10 N_i additional robot runs. The MCL updates 5000 particles (i.e., $|m| = 5000$), where the computation time takes, on average, 2.1ms and 145.6ms for the motion model sampling and measurement model, respectively. In order for the MCL to output a Cartesian coordinate, we take the weighted average of all particles, the result of which can be checked against ground truth. Due to the random nature of the MCL, each of the 10 runs are localized 100 different times. The average mean for the 100 trials of all 10 runs is 0.61 meters, with a low average standard deviation of 0.0049m. Comparing these results to an offline LRF SLAM algorithm [10], which produces an average error of 0.27m, we observe a compelling tradeoff between accuracy and sensor payload (or price) when using the proposed WiFi localizer. An illustrative example of a partial run is shown in Figure 7. We encourage readers to watch the accompanied video, which visually shows the outcome of the MCL, running on a robot, for every iteration. Interested readers should also visit our website[1], where they will have access to the raw data, trained classifiers, and code presented in this paper.

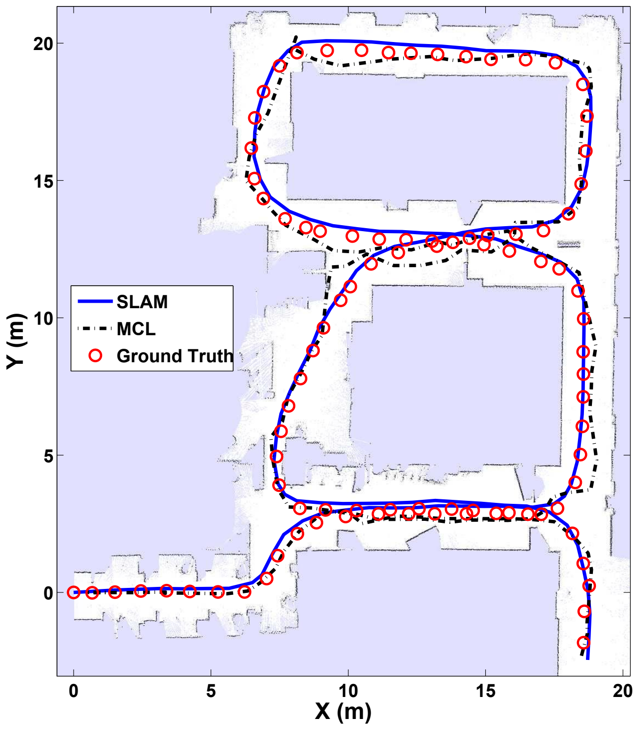

Fig. 7. Localized run showing the output of LRF SLAM (straight line), the WiFi localizer (dotted line), and Ground Truth (circles).

VII. Conclusions

We have presented an hybrid algorithm, mixing classification and regression methods, capable of localizing, with sub-meter accuracy, robots with a minimal sensor payload consisting of a WiFi card and odometry. In fact, the end-to-end WiFi localizer not only favorably compares to previously published algorithms, but its performance is also competitive with the full LFR SLAM solution. The evaluation and comparison of our WiFi localizer against both published and new algorithms, along with experiments performed on different datasets, provide compelling evidence regarding the robustness of our approach. Moreover, the algorithm is fast enough to ensure real-time mapping and localization.

There are a few of interesting future directions that are to be considered. First, we would like to modify our mapping algorithm so that it can be built incrementally, allowing the *mapper* to provide the *localizers* with partial maps or sections of the environment. Random Forests would still be a viable option in that scenario, although they would require some modifications to increase the number of trees when new data is available and to prune older trees that, over time, will not encompass enough information. Second, we believe that WiFi localization, thanks to its low sensor requirement commonly found in robots, would be a great middle-layer to merge heterogeneous maps together. Indeed, WiFi localization could solve, for example, the difficult problem of merging a Cartesian grid map with a topological or visual map. Finally, if needed, the algorithms presented can be made faster by parallelizing the training process.

[1]https://robotics.ucmerced.edu/Robotics/IROS2012/

References

[1] P. Bahl and V. Padmanabhan. RADAR: An in-building RF-based user location and tracking system. In *IEEE Int. Conference on Computer Communications*, pages 775–784, 2000.

[2] J. Biswas and M. Veloso. WiFi localization and navigation for autonomous indoor mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 4379–4384, 2010.

[3] W. Braun and U. Dersch. A physical mobile radio channel model. *IEEE Transactions on Vehicular Technology*, 40(2):472–482, 1991.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. CRC Press, Boca Raton, FL, 1984.

[6] F. Duvallet and A. Tews. WiFi position estimation in industrial environments using Gaussian processes. In *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, pages 2216–2221, 2008.

[7] B. Ferris, D. Fox, and N. Lawrence. WiFi-SLAM using Gaussian process latent variable models. In *Int. Joint Conferences on Artificial Intelligence*, pages 2480–2485, 2007.

[8] J. Fink, N. Michael, A. Kushleyev, and V. Kumar. Experimental characterization of radio signal propagation in indoor environments with application to estimation and control. In *IEEE/RSJ Int. Conference on Intelligent Robots and Systems*, pages 2834–2839, 2009.

[9] A. Goldsmith. *Wireless Communications*. Cambridge Press, 2005.

[10] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2006.

[11] A. Howard, S. Siddiqi, and G. Sukhatme. An experimental study of localization using wireless ethernet. In *Int. Conference on Field and Service Robotics*, 2003.

[12] J. Huang, D. Millman, M. Quigley, D. Stavens, S. Thrun, and A. Aggarwal. Efficient, generalized indoor WiFi GraphSLAM. In *IEEE Int. Conference on Robotics and Automation*, pages 1038–1043, 2011.

[13] Wild Packets Inc. Converting signal strength percentage to dBm values. Technical Report 20021217-M-WP007, 2002.

[14] D. Koutsonikolas, S. Das, Y. Hu, Y. Lu, and C. Lee. CoCoA: Coordinated cooperative localization for mobile multi-robot ad hoc networks. *Ad Hoc & Sensor Wireless Networks*, 3(4):331–352, 2007.

[15] A. Ladd, K. Bekris, A. Rudys, D. Wallach, and L. Kavraki. On the feasibility of using wireless ethernet for indoor localization. *IEEE Transactions on Robotics and Automation*, 20(3):555–559, 2004.

[16] P. McCullagh and J. Nelder. *Generalized Linear Models*. Chapman & Hall, New York, NY, 1990.

[17] G. McLachlan and D. Peel. *Finite Mixture Models*. John Wiley & Sons, Inc., Hoboken, NJ, 2000.

[18] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*, chapter 5,8. The MIT Press, Cambridge, Massachusetts, 2005.

[19] D. Tran and T. Nguyen. Localization in wireless sensor networks based on support vector machines. *IEEE Transactions on Parallel and Distributed Systems*, 19(7):981–994, 2008.

[20] M. Youssef, A. Agrawala, A. Shankar, and S. Noh. A probabilistic clustering-based indoor location determination system. Technical Report CS-TR-4350, University of Maryland, College Park, 2002.

[21] S. Zickler and M. Veloso. RSS-based relative localization and tethering for moving robots in unknown environments. In *IEEE Int. Conference on Robotics and Automation*, pages 5466–5471, 2010.