RF-Based Relative Localization for Robot Swarms

Wolfgang Hönig[1], Nitin Kamra[1]

Abstract— **We evaluate the radio signal strength indicator (RSSI) of an nRF5188 based robot swarm for relative localization. We discuss an efficient way of high-speed data collection with up to 40,000 RSSI samples per second. Semi-automatic radio parameter selection as well as robot-assisted data collection methods are presented. Furthermore, we show our collected data including the variance for selected outdoor experiments. The resulting data is used to fit a novel sigmoid based power vs. log-distance model, which in turn is the foundation of a centralized anchor-free localization algorithm. We present a simple gradient descent based localization approach which is computationally simple, easily extensible to distributed swarms, and scales efficiently to large number of robots in a swarm.**

I. INTRODUCTION

Localization remains a hard problem in robotics. Using multiple robots which are able to localize each other can help for formation control and improve the global localization of the group (for example, if one of the robots has access to GPS-based localization). The signal strength of radio communication correlates with the distance between the sender and receiver. Assuming a 2D plane, the formation can be reconstructed up to orientation and mirror uncertainties if all robot-to-robot distances are known.

The scope of the project is to evaluate whether the RF-chip of the Crazyflie 2.0 quadrotor can be used as foundation for the relative localization for a small swarm of robots in a simplified 2D-plane setting. We consider the case of static nodes (Crazyflies) on the ground and attempt to do a relative localization for each Crazyflie using only RSSI data from its neighbors.

II. RELATED WORK

Localization for swarms has mainly been discussed in the Wireless Sensor Networks (WSNs) literature. Recent surveys [1], [2] classify the research in this field either by measurement techniques (such as Time of Flight or signal strength), or localization approach (self-localization or target localization).

For our work, we will focus on related work which either uses the radio signal strength indicator (RSSI) or discusses self-localization algorithms. In general, the problem of localization with RSSI data can be divided into two phases:

1) Retrieve distance information from RSSI data.
2) Estimate positions from pairwise distance information using some optimization algorithm.

Retrieving distance information from RSSI readings has been attempted in literature using both deterministic functions and stochastic methods. The most common approach to model the RSSI vs. distance function is by using a Log-distance path loss model [3]. It assumes that measured RSSI decreases linearly with $\log_{10} d$ and models uncertainty in this process by imposing a zero-mean gaussian noise on top of the straight line. Furthermore the model has been extended with a dynamic variance (LNSM-DV) [4].

After retrieving the function (or probability distribution) relating RSSI and distance, the next step involves solving for exact 2-D or 3-D coordinates using pairwise distance data which can be done by formulating the problem as an optimization problem. A large portion of the wireless localization literature revolves around ways to solve the resulting optimization problem.

Moore *et al.* use fairly accurate ($5\,\mathrm{cm}$) sonar-based distance sensors to achieve relative localization between moving robots using a distributed algorithm [5]. They employ a simple computational approach to find quadrilaterals in a local neighborhood which are robust to noise and then compute relative transformations between local clusters, all of which can be done in $O(n^3)$ time using efficient data structures. While the algorithm shows good theoretical and practical results, it is questionable if it will work for the kind of data we obtain using RSSI, which suffers from fading and interference effects and as such has a lot of noise.

RSSI-based approaches often use so-called beacons or anchor nodes, which are nodes deployed at a fixed and known position. Zanca *et al.* compare different RSSI-based methods with varying number of anchor nodes in an indoor setting [6]. The results suggest an expected accuracy of about $2\,\mathrm{m}$ with 5 anchor nodes using a Maximum Likelihood approximation. On the contrary, we want to analyze the possibility of using RSSI measurements without anchor nodes for relative localization.

Another set of approaches to localization derive their inspiration from biological swarms. Kulkarni *et al.* [7] present a comparison of the two most popular bio-inspired optimization algorithms: Particle Swarm Optimization (PSO) and Bacterial Foraging Algorithm (BFA) in terms of the number of nodes localized, localization accuracy and computation time.

In this work we propose a smooth sigmoid function to model the variation of RSSI vs. distance and then present a simple gradient descent based optimization approach which is computationally simple, easily extensible to distributed swarms and scales efficiently to large number of robots in a swarm.

[1]Authors are with the Department of Computer Science, University of Southern California, USA {whoenig, nkamra}@usc.edu

III. TARGET PLATFORM

The Bitcraze Crazyflie 2.0 Nano Quadcopter [8] is a flying development kit that targets researchers and hobbyists. It weighs about $27\,\mathrm{g}$, is about $92\,\mathrm{mm}$ motor to motor, and fits in the palm of a human hand. The software and hardware specifications are freely available online, which makes the platform great for research.

The Crazyflie has a dual-MCU architecture using an nRF5188 (Cortex-M0, $32\,\mathrm{MHz}$, $16\,\mathrm{kB}$ SRAM, $128\,\mathrm{kB}$ flash) for power management and communication and an STM32F405 (Cortex-M4, $168\,\mathrm{MHz}$, $192\,\mathrm{kB}$ SRAM, $1\,\mathrm{MB}$ flash) for the control algorithm. Both MCUs are connected using UART. Sensors include the MPU9250 (gyroscope, accelerometer, magnetometer) and a LPS25H (pressure). The setpoint (yaw, pitch, roll, thrust) can be sent from a PC using a special USB dongle, or using BlueTooth LE.

The radio chip nRF51822 is a SoC which is able to use BlueTooth Smart and custom $2.4\,\mathrm{GHz}$ wireless radio communication. It is possible to communicate over 127 different channels in the $2.4\,\mathrm{GHz}$ band and it supports RSSI (Received Signal Strength Indicator) with an accuracy of $\pm 6\,\mathrm{dB}$ (resolution of $1\,\mathrm{dB}$ [9, Table 42]). Additionally, the power amplifier for the RF communication is software controlled.

IV. APPROACH TO DATA COLLECTION

In the following we will describe the various steps which were taken in order to collect RSSI data between different Crazyflies.

A. Implementing PTX mode and RSSI measurements

With the default firmware every Crazyflie acts as primary receiver (PRX), while the PC-side (using the USB-Dongle Crazyradio) operates as primary transmitter (PTX). The primary transmitter can send a data packet, which will be received by the receiver and acknowledged using up to $32\,\mathrm{Bytes}$ in the acknowledgment packet. For communicating within the swarm, it is required that each Crazyflie can operate in PTX mode as well. This mode is not implemented in the default firmware. Therefore, we used a library by the chip vendor [10] initially, however it turned out to be not a good fit for two reasons:

1) The queues for sending and receiving are completely independent, i.e. it is impossible to send the correct acknowledgment packet immediately as a response. This asynchronous protocol behavior made higher-level communication primitives much more complicated and error-prone.
2) While low-frequency RSSI sampling was supported, adding a special high-frequency sample mode was too complicated due to the complexity of the library code.

Nevertheless, working code based on this library can be found in one of our branches[1].

To overcome the above discussed limitations, we implemented our own radio driver[2]. A callback mechanism allows to acknowledge packets immediately, as long as the required computation is very small (the callback has to be called during an interrupt handler). Furthermore, we implemented high frequency RSSI sampling with up to 60,000 samples per second. This data is filtered to only take those data points into account where the radio is in receive mode, the Crazyflie device address matches the packet header, and a correct CRC was sent. In practice, this reduces the sampling to about 40,000 samples per second if the transmitting Crazyflie transmits as fast as possible.

We use a hybrid approach to stream the data out. First, the data is averaged on chip, by computing the count and sum during every RSSI interrupt. Both count and sum are sent via SEGGERs Real Time Terminal [11] using a SEGGER J-Link EDU every $10\,\mathrm{ms}$. This requires the Crazyflie Debug Adapter Kit which contains a small PCB to be soldered on the Crazyflie to obtain an SWD debug adapter port. On the PC side small python scripts are used to receive, plot, and store the data.

B. Parameter selection: Power, Datarate, Channel

The radio allows to select eight different transmit power levels ($4\,\mathrm{dBm}$, $0\,\mathrm{dBm}$, $-4\,\mathrm{dBm}$, $-8\,\mathrm{dBm}$, $-12\,\mathrm{dBm}$, $-16\,\mathrm{dBm}$, $-20\,\mathrm{dBm}$, $-30\,\mathrm{dBm}$), three datarates ($250\,\mathrm{Kbit/s}$, $1\,\mathrm{Mbit/s}$, $2\,\mathrm{Mbit/s}$), and 127 channels which correspond to a carrier frequency. It is expected that all those parameters do not alter the obtained RSSI significantly, and hence it should be possible to select a fixed set of parameters. We did three different experiments to validate this assumption.

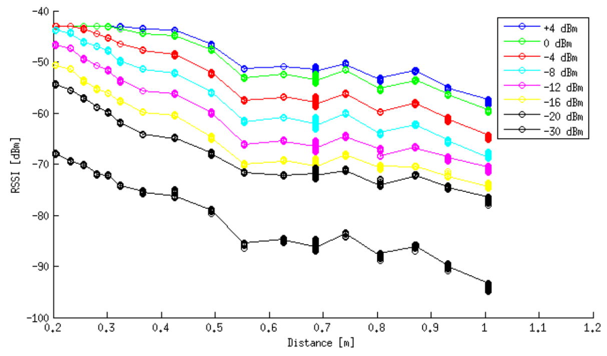
First, we implemented a special powerscan mode in the firmware, which automatically switches between all available transmit power levels and collects the RSSI values. These values were recorded at different distances indoors. The Crazyflies were placed manually in a room equipped with a motion capture system at different distances and values were recorded once the data stabilized. The results are visible in Fig. 1(a) and show that there is just a fixed offset between the power levels as expected. However, there is also a saturation visible for very short distances. Therefore, we selected $-8\,\mathrm{dBm}$ as power level for future experiments.

Second, we implemented a similar mode for the different data rates. There was no visible change in the RSSI values and we choose the slowest possible mode. This maximizes the number of RSSI samples we get during communication, because the samples are taken at a fixed frequency independent of the data rate.

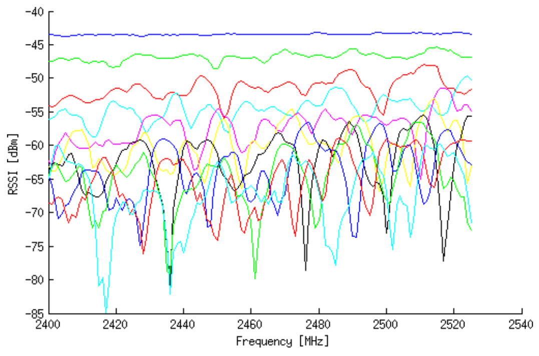
Third, we implemented a special channelscan mode which collects the RSSI values for all available channels while keeping the distance constant. Due to multi-path interference and other signals on the $2.4\,\mathrm{GHz}$ band (such as WiFi), a high value does not necessary mean a strong signal. Therefore we

[1]`https://github.com/whoenig/Crazyflie2-nrf-firmware/tree/cs599_ptx`

[2]`https://github.com/whoenig/Crazyflie2-nrf-firmware/tree/cs599_datacollection`



(a) Powerscan: RSSI at different power levels vs. distance.



(b) Channelscan: RSSI vs. Frequency. Different lines represent different distances.

Fig. 1. Data collection results for parameter selection.

used an approximate method to find a good channel: The data is collected at different, discrete distances (Fig. 1(b)). For each distance sample, the mean across all frequencies is computed. Finally, the frequency which was most often for all different distances close to the mean is selected as a good channel. Using this method, we found that channel 66 ($2466\,\mathrm{MHz}$) is a good choice for the indoor experiments.

We found a more accurate method for the channelscan later by using a receiver only which samples RSSI values constantly. For channels which are not occupied, the measured RSSI value is simply fully saturated ($-100\,\mathrm{dBm}$), however channels which are used by other signals pick up a slightly higher power level. This method was used for the outdoor experiments and suggested channel 90.

C. Indoor collection (small distance ranges)

In order to obtain a larger amount of data, we put the transmitting Crazyflie on top of a Clearpath Turtlebot, which was remotely controlled with a joystick. The joystick was directly attached to the netbook so that no WiFi was required. As before, the ground truth (distance and orientation) of both Crazyflies were collected using the Vicon motion capture. The data was streamed in real-time to a PC and oscillations were clearly visible while moving or during other external events such a person walking by. Hence, an operator could use another joystick to store data after it stabilized.

While there was a clear trend of RSSI values visible in case of a fixed orientation (Fig. 2(a)), no relationship was visible when rotating at least one of the Crazyflies (Fig. 2(b)). In a swarm, the relative angle between the antennas can not be fixed and thus the RSSI/distance model needs to hold for arbitrary rotations or the angle needs to be taken into account in the mode explicitly. In order to minimize interference effects, we decided to collect the data outdoors instead.

D. Outdoor collection with magnetometer

To obtain enough datasets to estimate the function $RSSI = f(\theta, d)$ we need a reliable way of measuring the angles of the two Crazyflies. The on-board IMU has a magnetometer, which is not used in the default firmware. Magnetometers need to be calibrated for hard-iron and soft-iron errors to achieve reasonable results. We followed a method briefly described in [12] which in turn is based on several scientific papers [13]–[15]. We collect n readings of the raw magnetometer data (3-dimensional vector m_i) along with relative rotations to earth (R_i), based on accelerometer and gyroscope sensor fusion [16]. Ideally, the following equations should hold, where b is the earth magnetic field at the fixed calibration location in the earth coordinate frame, M corrects for rotation and stretch, and B is a fixed offset:

$$m_i = M(R_i b + B) \quad \forall i \in \{1 \ldots n\} \tag{1}$$

We have $6n$ values given and 15 unknown and can pose the problem as optimization problem to minimize the least mean square error for a large enough n. We collected the data using the logging capabilities of the default firmware and implemented an iterative optimization method using Matlab.

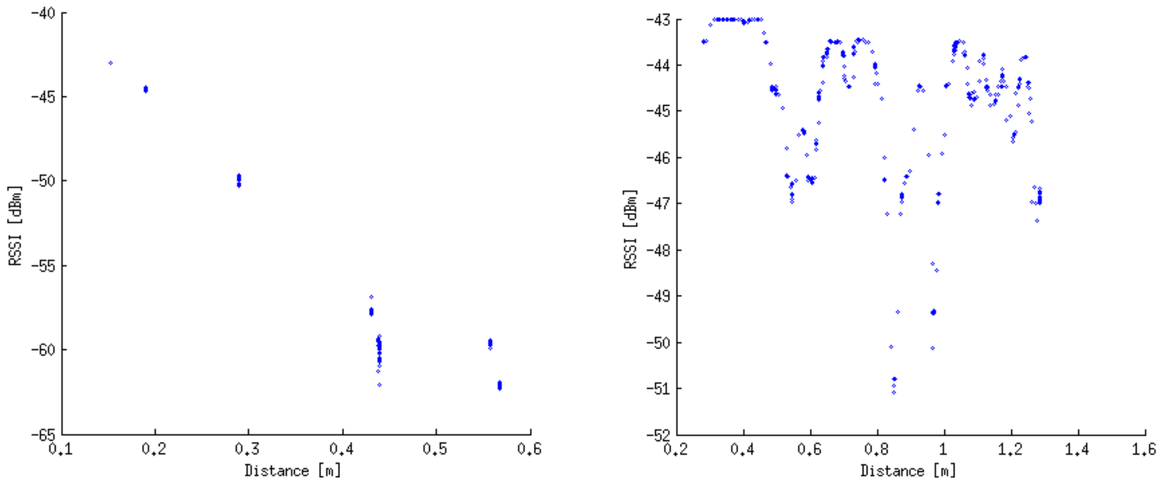
We calibrated the magnetometer for both Crazyflies and used a state-of-the-art algorithm to fuse it with the accelerometer and gyroscope [16]. The data was collected outdoors with varying distances between $0\,\mathrm{m}$ and $1.5\,\mathrm{m}$ at different rotations. As in the other experiments, an operator collected the data only after it stabilized using a real-time plotting tool and a joystick.

The data showed non-consistent results, i.e. the random noise was larger than the effect of the different orientations. This led us to the conclusion to try to find a solution which works on a larger scale (up to $15\,\mathrm{m}$).

E. Outdoor collection with iRobot Create2

To minimize human intervention and maximize the amount of data we collect, we put the transmitting Crazyflie on an iRobot Create2. The robot was connected to a laptop using a long USB cable for remote control. Unfortunately, the existing programming packages are either for the Create1, or do not have proper odometry support. Therefore, we implemented our own python driver which allows us to stream the optical encoder data every $15\,\mathrm{ms}$ for odometry estimates and controlling the Create2 using a Joystick.

The experiment was repeated five times with a constant slow forward motion of the Create2, while the data was collected continuously. The raw data is shown in Fig. 3(a). Please note that we increased the power level to $-4\,\mathrm{dBm}$.



(a) RSSI vs. distance using fixed orientations of the Crazyflies. (b) RSSI vs. distance using varying orientations of the Crazyflies.

Fig. 2. Indoor data collection within small distances, using channel 66, $-8\,\mathrm{dBm}$, and $250\,\mathrm{Kbit/s}$.

V. MODEL FITTING

The most common model used to fit RSSI data is the Log-distance path loss model [3]. It models the received RSSI reading (P) as a straight line function of $\log_{10}(d)$ with superimposed gaussian noise. The gaussian noise parameters can be assumed constants or varying with distance. The model is given by:

$$P = P_0 - \gamma \log_{10} d + X_g \tag{2}$$

In general, this model works well within a range of distance $d_1 \leq d \leq d_2$, but the RSSI readings get saturated outside this distance range. So we use a piecewise linear (PWL) model which follows the Log-distance path loss model between the cutoffs d_1 and d_2, but imposes a saturated reading outside the range i.e.

$$P = \begin{cases} P_0 - \gamma \log_{10} d_1 + X_g, & d \leq d_1 \\ P_0 - \gamma \log_{10} d + X_g, & d_1 \geq d \geq d_2 \\ P_0 - \gamma \log_{10} d_2 + X_g, & d_2 \leq d \end{cases} \tag{3}$$

For parameter estimation, we choose the values d_1, d_2, P_0 and γ which maximize the log-likelihood of occurrence of our data set. We iteratively search for the cutoffs d_1 and d_2, while computing P_0 and γ every time using gradient descent for each pair (d_1, d_2) and finally retain the best choices. Figure 3(a) shows our dataset and the fitted piecewise linear model.

Though the piecewise linear model is a good fit for making predictions, it is not a good model to use in any optimization since it is not smooth. We aim to develop a decentralized localization algorithm later on, and most robots are not sophisticated enough to run complicated non-linear optimization techniques. Hence it is important to have a smooth, continuous and differentiable model for RSSI vs. $\log_{10} d$ fit, so that even robots with low computation power can use the localization algorithm while employing a simple gradient descent for optimization.

For this reason, we came up with another model to describe the RSSI vs. $\log_{10} d$ data. We model our data with a sigmoid curve which takes the following form:

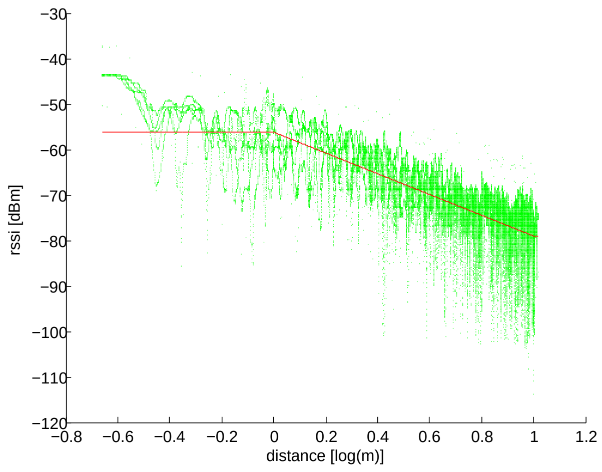
$$P = P_L + \frac{P_H - P_L}{1 + e^{-s(\log_{10} d - \mu)}} + X_g \tag{4}$$

The sigmoid function is a continuous and differentiable function and hence has nice properties with respect to differentiation. To the authors' knowledge, this model has not been used in the localization literature up till now. The sigmoid fit to our dataset has been shown in Fig. 3(b). It is clear that not only does the model fit better to the data now, but also permits the use of local gradient-based optimization methods for localization later on. Since X_g is distributed normally, we need its mean and standard deviation as functions of distance, but for most practical purposes the mean can be set to 0, and the standard deviation can be assumed constant. It can be observed from Fig. 3(c) that the best-fit straight line has an average standard deviation of about $4\,\mathrm{dBm}$, which we will use as a measure of standard deviation for our model.

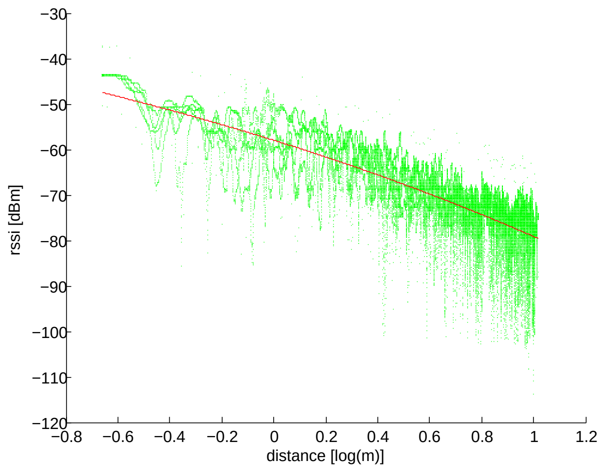
VI. CENTRALIZED LOCALIZATION

We used our model to conduct experiments for relative localization with a swarm of six Crazyflies. Though we later intend to have a full decentralized localization algorithm, we currently solved the localization problem in a centralized way.

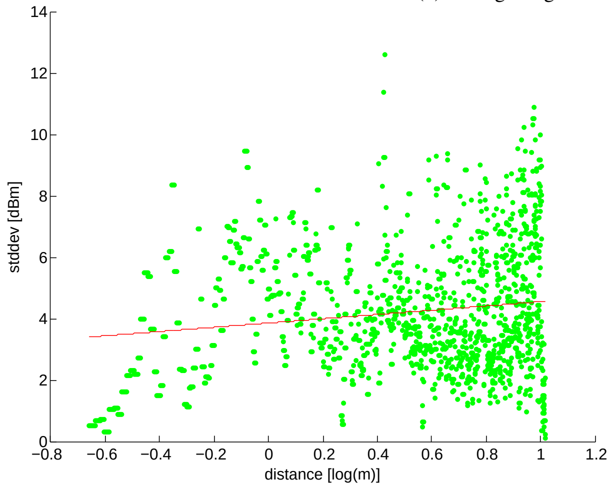
The swarm of Crazyflies was kept in a static configuration indoors and we allotted each of them a unique ID. The idea was to let them exchange RSSI data amongst themselves, which would then be streamed out to a PC and a localization algorithm run on it.



(a) Fitting a piecewise linear model to RSSI dataset.



(b) Fitting a sigmoid model to RSSI dataset.



(c) Std. deviation as a function of distance about the fitted sigmoid.

Fig. 3. Model fitting to RSSI dataset.

A. Approach

Each Crazyflie was pre-programmed to transmit at $-4\,\mathrm{dBm}$, $250\,\mathrm{KBit/s}$ and $2490\,\mathrm{MHz}$, and ran a state machine which initialized all of them in a *wait to synchronize* mode. On receiving the *synchronize* signal from a PC, all of them would start their clocks together and go into a *signaling* mode. In this mode, the Crazyflies took turns transmitting messages (containing their ID) one-by-one in the order of their unique IDs while all other Crazyflies would listen and measure RSSI signal strength. This way each Crazyflie would end up having signal strength data from each of their neighbors. This stage was followed by a *data sharing* stage, where each Crazyflie would transmit all its signal strength data to a PC, where it would be fed to a localization algorithm.

B. Problem Formulation

Using our sigmoid model for RSSI data, we now formulate the problem of localization as a maximum likelihood problem. Table I shows the notation used in this paper.

Given a set of readings for n Crazyflies in the format: $[TxID, RxID, P_{ij}]$, the problem of relative localization is of determining the euclidean coordinates (x_i, y_i) of each Crazyflie from the above data. We formulate this problem as a maximum likelihood problem as follows:

n	Number of Crazyflies
$TxID$	Transmitter Crazyflie ID
$RxID$	Receiver Crazyflie ID
P_{ij}	RV representing RSSI value with $TxID = i$ and $RxID = j$
p_{ij}	RSSI value with $TxID = i$ and $RxID = j$
X_i	RV representing x-coordinate of i^{th} Crazyflie
x_i	x-coordinate of i^{th} Crazyflie
Y_i	RV representing y-coordinate of i^{th} Crazyflie
y_i	y-coordinate of i^{th} Crazyflie
D_{ij}	RV representing distance between i^{th} and j^{th} Crazyflies
d_{ij}	Distance between i^{th} and j^{th} Crazyflies
A_{ij}	1 if p_{ij} value is available, 0 otherwise
$\mathcal{N}(i)$	Set of neighbors of Crazyflie with ID = i

TABLE I. Notation used in this paper (RV = Random Variable).

$$\max_{x,y} \mathcal{P}(P = p | X = x \text{ and } Y = y) \tag{5}$$

Since all readings are independent, the above can be written as a product of probabilities of occurring of individual readings.

$$\max_{x,y} \prod_{\substack{i,j \\ j\in\mathcal{N}(i)}} \mathcal{P}(P_{ij} = p_{ij} | X_i = x_i, Y_i = y_i, X_j = x_j, Y_j = y_j) \tag{6}$$

Using our sigmoid model with gaussian noise, this can be written as:

$$\max_{x,y} \prod_{\substack{i,j \\ j\in\mathcal{N}(i)}} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}\left(p_{ij} - P_L - \frac{P_H - P_L}{1+e^{-s(\log_{10} d_{ij} - \mu)}}\right)^2} \tag{7}$$

where

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \tag{8}$$

Since σ has been assumed independent of distance (and hence of x and y), we can ignore it and further transform the problem to a minimization problem by taking negative-log of the objective function.

$$\min_{x,y} \frac{1}{2} \sum_i \sum_{j\in\mathcal{N}(i)} \left(p_{ij} - P_L - \frac{P_H - P_L}{1+e^{-s(\log_{10} d_{ij} - \mu)}}\right)^2 \tag{9}$$

Now considering the sigmoid function as:

$$g(z) = P_L + \frac{P_H - P_L}{1+e^{-s(z-\mu)}} \tag{10}$$

and letting A_{ij} denote the availability of the value p_{ij}, we can write the above minimization problem as follows:

$$\begin{aligned} &\min_{x,y} F(x,y) \\ &\text{where} \\ &F(x,y) = \frac{1}{2}\sum_i \sum_j A_{ij}\left(p_{ij} - g(\log_{10} d_{ij})\right)^2 \end{aligned} \tag{11}$$

C. Optimizing with Gradient Descent

To optimize our objective function (11), we use a simple gradient descent algorithm. But to compute the gradient of $F(x,y)$, we first need the following derivatives:

$$\frac{\partial d_{ij}}{\partial x_i} = -\frac{\partial d_{ij}}{\partial x_j} = \frac{x_i - x_j}{d_{ij}} \tag{12}$$

$$\frac{\partial d_{ij}}{\partial y_i} = -\frac{\partial d_{ij}}{\partial y_j} = \frac{y_i - y_j}{d_{ij}} \tag{13}$$

We have the derivative of sigmoid function as:

$$\frac{d(g(z))}{dz} = \frac{s(P_H - P_L)e^{-s(z-\mu)}}{(1+e^{-s(z-\mu)})^2} \tag{14}$$

which can be re-written as follows:

$$\frac{d(g(z))}{dz} = \frac{s(g(z) - P_L)(P_H - g(z))}{P_H - P_L} \tag{15}$$

With the above derivatives in place, we can now calculate the gradient of $F(x,y)$ as follows:

$$\begin{aligned} \frac{\partial F(x,y)}{\partial x_k} = &-\sum_j \Bigg\{ A_{kj}\left(p_{kj} - g(\log_{10} d_{kj})\right) \\ &\frac{s(g(\log_{10} d_{kj}) - P_L)(P_H - g(\log_{10} d_{kj}))}{P_H - P_L} \frac{(x_k - x_j)}{d_{kj}^2 \log 10} \Bigg\} \\ &-\sum_i \Bigg\{ A_{ik}\left(p_{ik} - g(\log_{10} d_{ik})\right) \\ &\frac{s(g(\log_{10} d_{ik}) - P_L)(P_H - g(\log_{10} d_{ik}))}{P_H - P_L} \frac{(x_k - x_i)}{d_{ik}^2 \log 10} \Bigg\} \end{aligned} \tag{16}$$

$$\begin{aligned} \frac{\partial F(x,y)}{\partial y_k} = &-\sum_j \Bigg\{ A_{kj}\left(p_{kj} - g(\log_{10} d_{kj})\right) \\ &\frac{s(g(\log_{10} d_{kj}) - P_L)(P_H - g(\log_{10} d_{kj}))}{P_H - P_L} \frac{(y_k - y_j)}{d_{kj}^2 \log 10} \Bigg\} \\ &-\sum_i \Bigg\{ A_{ik}\left(p_{ik} - g(\log_{10} d_{ik})\right) \\ &\frac{s(g(\log_{10} d_{ik}) - P_L)(P_H - g(\log_{10} d_{ik}))}{P_H - P_L} \frac{(y_k - y_i)}{d_{ik}^2 \log 10} \Bigg\} \end{aligned} \tag{17}$$

The above gradient equations are correct assuming that $A_{kk} = 0\,\forall k$.

Finally we can implement the gradient descent in an iterative loop as shown in algorithm 1. α is the step-size factor and can be fine-tuned to speed up or slow down the convergence of the algorithm.

```
1: Initialize $x$, $y$ randomly
2: **repeat**
3:     $x_k := x_k - \alpha \frac{\partial F(x,y)}{\partial x_k}\ \forall k$
4:     $y_k := y_k - \alpha \frac{\partial F(x,y)}{\partial y_k}\ \forall k$
5: **until** convergence
```

Algorithm 1. Gradient Descent Algorithm

VII. RESULTS AND DISCUSSION

Preliminary tests with randomly generated RSSI values (including noise) showed that our centralized localization algorithm can find the correct solution. One of the nodes was always held fixed at the origin to avoid convergence issues due to absence of a global reference frame. This avoided a global translation ambiguity in the solution, however the solution does suffer from global reflection and rotation ambiguities. Seeding the initial values with a noisy guess of the actual solution often derived a very close match if the standard deviation of the super-imposed noise was not very large.

Results for a run of the centralized localization algorithm can be seen in Fig. 4. The data for this experiment was created by placing six nodes randomly and generating their

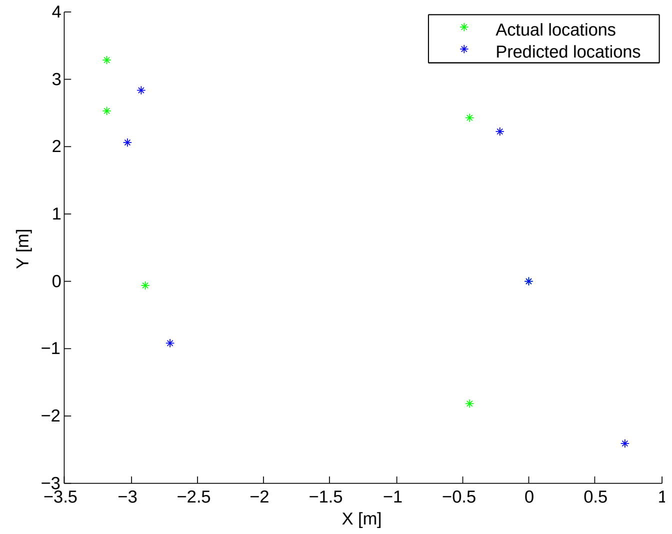


Fig. 4. Results of Centralized localization with 6 nodes and randomly allotted locations. The data was created by adding zero-mean gaussian noise with $4\,\mathrm{dBm}$ std. deviation to the RSSI readings generated by the sigmoid model. A noisy estimate of the actual locations was provided to seed the optimization algorithm. The algorithm does not give a very accurate estimate finally because of the added noise in the RSSI readings.

pairwise RSSI readings according to our sigmoid model. A zero-mean gaussian noise of std. deviation $4\,\mathrm{dBm}$ was added to the RSSI readings before giving them to the optimization algorithm. The algorithm was seeded with a close (but random) initial estimate of the nodes' locations. As can be observed from the figure, the algorithm does produce estimates close to the actual positions. It could not converge to the actual positions because of the noise in RSSI readings.

Further tests with actual data did not work out that well and the algorithm did not find a close match. There are many possible causes for this behavior:

- The algorithm should behave better with a large number of nodes but we were limited by the number of available robots to six nodes.
- The Crazyflies have slightly different RSSI vs. distance curves based on their actual antenna placement, chip variations etc. Nevertheless, the model had been created using a specific single pair of Crazyflies and not for all of them. So it would be useful to create a model which takes the individual transmission and reception power offsets of each Crazyflie into account, by accepting them as input parameters.
- Finally, our approach assumes a constant variance, while Fig. 3(c) suggests that the variance is actually increasing with distance and the variation of RSSI with distance should be included in the model as well.

VIII. Future Work

Future work on this project would proceed along the lines of obtaining better models and filtering the acquired data.

More specifically we wish to do a histogram-based data collection [17] which rejects the lower and upper 25% RSSI samples at each distance before computing an average. This leads to better noise rejection for the collected samples.

Another way to improve the model-fitting could be to do outlier rejection on the dataset first. It is clear from figures 3(a) and 3(b) that there are many outliers at every distance and rejecting them would give us a better model representing the data.

Moreover we currently include only those readings in our optimization objective where p_{ij} is available. But if p_{ij} is not available, it signifies that Crazyflies i and j are probably far apart and this information can also be considered in the optimization objective to make the localization more accurate.

Lastly, since each Crazyflie is slightly different, we would like to model the individual transmission and reception offsets of the wireless nodes (Crazyflies) as pointed out in section VII.

References

[1] G. Mao, B. Fidan, and B. D. Anderson, "Wireless sensor network localization techniques," *Computer Networks*, vol. 51, no. 10, pp. 2529 – 2553, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1389128606003227

[2] G. Han, H. Xu, T. Duong, J. Jiang, and T. Hara, "Localization algorithms of wireless sensor networks: a survey," *Telecommunication Systems*, vol. 52, no. 4, pp. 2419–2436, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11235-011-9564-7

[3] T. S. Rappaport, *Wireless Communications: Principles and Practice*, 2nd ed. Prentice Hall, 2002.

[4] J. Xu, W. Liu, F. Lang, Y. Zhang, and C. Wang, "Distance measurement model based on rssi in wsn," *Wireless Sensor Network*, vol. 2, no. 08, p. 606, 2010.

[5] D. Moore, J. Leonard, D. Rus, and S. Teller, "Robust distributed network localization with noisy range measurements," in *Proceedings of the 2Nd International Conference on Embedded Networked Sensor Systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 50–61. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031502

[6] G. Zanca, F. Zorzi, A. Zanella, and M. Zorzi, "Experimental comparison of rssi-based localization algorithms for indoor wireless sensor networks," in *Proceedings of the Workshop on Real-world Wireless Sensor Networks*, ser. REALWSN '08. New York, NY, USA: ACM, 2008, pp. 1–5. [Online]. Available: http://doi.acm.org/10.1145/1435473.1435475

[7] R. Kulkarni, G. Venayagamoorthy, and M. Cheng, "Bio-inspired node localization in wireless sensor networks," in *Systems, Man and Cybernetics, 2009. SMC 2009. IEEE International Conference on*, Oct 2009, pp. 205–210.

[8] Crazyflie nano quadcopter. [Online]. Available: http://www.bitcraze.se

[9] "nRF51822 product specification v3.1," Nordic Semiconductor, Oslo, Norway.

[10] Stripped down enhanced shockburst library for the nrf51 series. [Online]. Available: https://github.com/NordicSemiconductor/nrf51-micro-esb

[11] Real time terminal. [Online]. Available: https://www.segger.com/jlink-real-time-terminal.html

[12] Calibration part 5: Magnetometers (continued). [Online]. Available: https://myahrs.wordpress.com/2012/04/19/calibration-part-5/

[13] M. Kok, J. Hol, T. Schon, F. Gustafsson, and H. Luinge, "Calibration of a magnetometer in combination with inertial sensors," in *Information Fusion (FUSION), 2012 15th International Conference on*, July 2012, pp. 787–793.

[14] V. Renaudin, M. H. Afzal, and G. Lachapelle, "Complete Triaxis Magnetometer Calibration in the Magnetic Domain," *Journal of Sensors*, vol. 2010, pp. 1–10, 2010. [Online]. Available: http://dx.doi.org/10.1155/2010/967245

[15] J. Vasconcelos, G. Elkaim, C. Silvestre, P. Oliveira, and B. Cardeira, "Geometric approach to strapdown magnetometer calibration in sensor frame," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 47, no. 2, pp. 1293–1306, April 2011.

[16] S. Madgwick, A. Harrison, and R. Vaidyanathan, "Estimation of imu and marg orientation using a gradient descent algorithm," in *Rehabilitation Robotics (ICORR), 2011 IEEE International Conference on*, June 2011, pp. 1–7.

[17] A. Awad, T. Frunzke, and F. Dressler, "Adaptive distance estimation and localization in wsn using rssi measures," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, Aug 2007, pp. 471–478.

APPENDIX A: SOURCE CODE

Crazyflie firmware changes

The code can be found online (including compilation and usage instructions):

NRF51 Firmware: PTX Mode based on Micro-ESB: This firmware uses [10] to implement both PTX and PRX mode.
`https://github.com/whoenig/Crazyflie2-nrf-firmware/tree/cs599_ptx`

NRF51 Firmware: Datacollection: This firmware uses a custom driver for the radio. It supports power, datarate, and channel scanning as well as a streaming mode for continuous data collection for fixed power level, datarate, and channel. In combination with the STM32 compass firmware, it is possible to collect the orientation based on the magnetometer as well.
`https://github.com/whoenig/Crazyflie2-nrf-firmware/tree/cs599_datacollection`

NRF51 Firmware: Centralized Localization: This firmware listens to a signal from the PC in order to synchronize clocks, collect pairwise RSSI signal strength in the whole swarm, and sends the results back to the PC for centralized localization. Hence, it can be used for pure data collection as well without the need of a special debug adapter.
`https://github.com/whoenig/Crazyflie2-nrf-firmware/tree/cs599_master`

NRF51 Firmware: (Improved) Channelscan: This firmware contains an improved channelscan which does not require a special transmitter. Instead, it continuously measures signal strength (without filtering) for different channels and sends the data to the PC using SEGGERs Real Time Terminal.
`https://github.com/whoenig/Crazyflie2-nrf-firmware/tree/channelscan`

STM32 Firmware: Compass Support: This firmware (for the STM32) allows to read the raw magnetometer data, uses a given calibration model to correct the values, and sends them to the NRF51 using UART for data collection purposes.
`https://github.com/whoenig/Crazyflie-firmware/tree/cs599_master`

Scripts

The following scripts are part of a private github repository.

- iRobot Create2:
  `https://github.com/whoenig/cs599project/blob/master/scripts/Create2.py`
- Model Fitting:
  `https://github.com/whoenig/cs599project/tree/master/scripts/Model%20learning`
- Centralized Localization:
  `https://github.com/whoenig/cs599project/tree/master/scripts/Centralized%20Optimization`
- Magnetometer Calibration:
  `https://github.com/whoenig/cs599project/tree/master/scripts/magnetometer`