# Drone Path Planning for Secure Positioning and Secure Position Verification

Pericle Perazzo*, Francesco Betti Sorbelli†, Mauro Conti‡, Gianluca Dini*, Cristina M. Pinotti§
* University of Pisa, Italy. Email: {name.surname}@iet.unipi.it
† University of Florence, Italy. Email: francesco.bettisorbelli@math.unifi.it
‡ University of Padua, Italy. Email: conti@math.unipd.it
§ University of Perugia, Italy. Email: cristina.pinotti@unipg.it

**Abstract**—Many dependable systems rely on the integrity of the position of their components. In such systems, two key problems are secure localization and secure location verification of the components. Researchers proposed several solutions, which generally require expensive infrastructures of several fixed stations (*anchors*) with trusted positions. In this paper, we explore the approach of replacing all the fixed anchors with a single drone that flies through a sequence of waypoints. At each waypoint, the drone acts as an anchor and securely determines the positions. This approach completely eliminates the need for many expensive anchors. The main challenge becomes how to find a convenient path for the drone to do this for all the devices. The problem presents novel aspects, which make existing path planning algorithms unsuitable. We propose *LocalizerBee*, *VerifierBee*, and *PreciseVerifierBee*: three path planning algorithms that allow a drone to respectively measure, verify, and verify with a guaranteed precision a set of positions in a secure manner. They are able to securely localize all the positions in a generic deployment area, even in the presence of drone control errors. Moreover, they produce short path lengths and they run in a reasonable processing time.

## 1 Introduction

The dependability of many distributed systems relies on knowing the position of the component devices. Some examples are sensor networks for environmental monitoring (both for civilian or military purposes), geographic routing, autonomous vehicle coordination, and so on. In all these cases, if the system believes that a device is in a position different from the real one, then it could infer wrong information and possibly take wrong decisions. Periodically measuring the position of the devices is not enough to guarantee security. Indeed, the majority of the positioning methods are vulnerable to attacks in which an adversary falsifies the position measurement [10].

Providing secure measurement of positions has shown to be a non-trivial problem [3], [14], [20], [21], [22]. A promising approach is *verifiable multilateration* [21], which is a provably secure technique to determine a position by measuring the distances from (at least) three anchors by means of *distance bounding protocols* [2]. A distance bounding protocol is a cryptographic protocol able to measure a secure upper bound to the distance between two devices. Verifiable multilateration is an extremely versatile technique, since it can withstand both external adversaries and compromised devices. However, it requires an expensive infrastructure of many fixed anchors. The number of necessary anchors grows roughly linearly with the size of the area in which the devices are deployed [15], [21]. Another problem is that the fixed anchors must be truly "fixed", otherwise an adversary could simply move one of them to jeopardize the security of the system. This makes the infrastructure cost even higher, since the anchors cannot be attached to the ground or to the walls in a cheap and insecure manner.

In this paper, we explore the possibility of using the emerging *drone technology* to solve these issues. Drones, or Unmanned Aerial Vehicles (UAV), are aircraft with no human pilot on board. They can enjoy different levels of autonomy [11], ranging from being remotely piloted to being completely autonomous in movements and decisions. Our approach is to replace many fixed anchors with a single drone which passes through a series of waypoints. At each waypoint, it acts as an anchor by executing a distance bounding protocol with one or more ground devices. We thus completely eliminate the need for many expensive fixed anchors.

Now, the problem becomes how to determine a convenient path for the drone. We cannot use existing path planning algorithms, because they are not thought for verifiable multilateration, and a valid path for verifiable multilateration must respect additional geometric constraints. In particular, the polygon formed by the waypoints must contain the position of the device, otherwise such a position cannot be considered trusted. Furthermore, other specific issues must be addressed, like the imprecision on the control of the drone movements.

**Contribution** The contribution of this paper is as follows.

- We explore the approach of using drones to securely localize a set of devices by means of verifiable multilateration.
- We propose three path planning algorithms for secure positioning and secure position verification: LocalizerBee, VerifierBee, and PreciseVerifierBee. LocalizerBee securely determines the position of a set of devices. VerifierBee securely verifies the position of a set of devices. In contrast with LocalizerBee, VerifierBee assumes to know the untrusted position of the devices, which must be veri-

fied. PreciseVerifierBee is an extension of VerifierBee which guarantees a bound on the positioning error, at a cost of a longer path.

- We run a thorough experimental evaluation of the proposed algorithms, and we compare them with the literature. The results of our experiments show that the localization-aimed paths proposed by the literature cannot be used with verifiable multilateration, because they are not able to localize all the devices in a generic deployment area. On the other hand, our proposed algorithms securely localize all the devices even in the presence of drone control errors. Moreover, they produce short path lengths and they run in a reasonable processing time.

**Organization** The rest of the paper is organized as follows. Section 2 compares with relevant related work. Section 3 introduces the basic concepts. Section 4 introduces the assumptions and the requirements of drone-based verifiable multilateration. Sections 5, 6, and 7 describe the proposed path planning algorithms, respectively LocalizerBee, VerifierBee, and PreciseVerifierBee. Section 8 reports the results of their experimental evaluation. The paper is concluded in Section 9.

## 2 Related Work

Secure positioning aims at measuring the position of a device in the presence of an adversary that wants to falsify such a measurement. Researchers proposed many methods [10], which offer different levels of security (provable or only statistical), and defend against different kinds of adversary (external or internal). Čapkun and Hubaux [21] proposed a provably secure positioning method called *verifiable multilateration*. In this proposal, the system measures the distances from a set of trusted anchors by means of distance bounding protocols. The position is computed by trilateration, and it is considered secure if it lies inside the convex hull of the anchors. Perazzo et al. [15] improved verifiable multilateration in such a way to require sensibly less anchors to cover the same area. This is achieved by leveraging the enlargement attack resistance of wireless distance bounding protocols, analyzed by [6] and [19]. In this paper, we replace the fixed anchors with a single mobile drone, thus completely eliminating the need for an expensive anchor infrastructure.

Perazzo et al. [14] studied the security of verifiable multilateration in the presence of non-ideal distance bounding protocols, vulnerable to some extent to PHY-level attacks. In this paper, we consider the distance bounding protocol to be ideal, i.e., immune to attacks. We leave the analysis of drone-based verifiable multilateration with non-ideal distance bounding as future work.

Čapkun et al. [20] proposed a secure location verification mechanism based on mobile stations (not necessarily drones). In their system, the movements of the mobile stations are random, and the security is based on the assumption that the adversary cannot observe nor predict such movements in any way. This assumption could be too strong for some applications. Instead, our approach does not require the unobservability of the drone movements. Moreover, a random path can take a very long time to cover a given area, thus it could be unsuitable for a limited-battery drone. On the other hand, by using verifiable multilateration we can plan shorter, non-redundant and deterministic paths.

In our previously published conference paper [13], we introduced the approach of drone-based verifiable multilateration, and we proposed a first version of the VerifierBee path planning algorithm. In this paper we extend such work by proposing the LocalizerBee and the PreciseVerifierBee path planning algorithms and analyzing their performances. LocalizerBee is for secure localization, rather than secure location verification, so it does not need prior information about the positions of the ground devices. PreciseVerifierBee is an extension of VerifierBee which guarantees a provable bound on the positioning error, at a cost of a longer path.

A problem related to ours is drone-based (insecure) localization of ground devices [4], [8], [12], [17], [18]. All these works do not have security in mind, and their position measurements cannot be considered trusted in a hostile environment or in the presence of compromised devices. One of the simplest approaches is the one given by Corke et al. [4], in which a robot sweeps the entire area and periodically broadcasts its GPS position. The devices collect such positions and infer their own position by averaging them. Such a method is not secure, since an adversary could simply send fake position broadcasts, in such a way to confuse the devices. Authenticating the position broadcasts does not solve the issue, since an adversary could listen to a legitimate broadcast and replay it on different positions. Another common method (see for example Sichitiu et al. [18]) is to infer the distance between the device and the mobile anchor from the strength of the received messages, and hence derive the position by trilateration. All the methods based on the received signal strength are poorly secure, since an adversary has an easy play on falsifying this information. In this paper, we assure the trustworthiness of the measured positions by using verifiable multilateration [21], which is a provably secure method.

Koutsonikolas et al. [12], Huang et al. [8], and Rezazadeh et al. [17] proposed and compared different trajectories that a mobile anchor can follow in order to localize a set of devices. They compared such trajectories in terms of length, coverage, and localization precision. In this paper, we test some of these trajectories, and we show that they are unsuitable to be used with verifiable multilateration. This is because a valid trajectory for verifiable multilateration must respect additional geometric constraints. As a consequence, the trajectories in [8], [12], [17] fails to localize securely all the devices in a generic deployment area. On the other hand, our path planning algorithms securely localize all the devices, and at the same time they produce short path lengths and they run in a reasonable processing time.

## 3 Preliminaries

A *distance bounding protocol* [2] is a cryptographic protocol able to measure a distance between two devices, in such a way that an adversary cannot falsify the measurement to be shorter than the real distance (*reduction attack*). A distance bounding protocol determines a distance by precisely measuring the round-trip time between a challenge

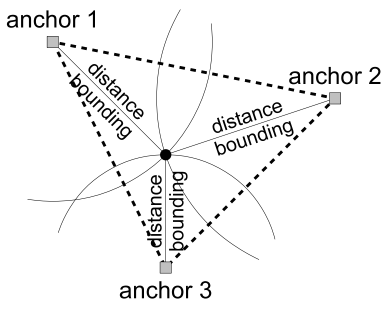


Fig. 1. Verifiable multilateration. The dashed triangle represents the verifiable triangle.

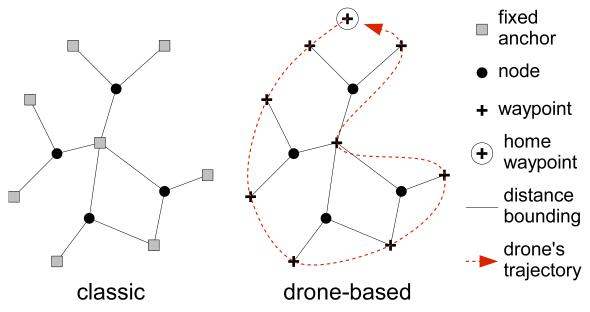


Fig. 2. Classic vs. drone-based verifiable multilateration.

and a response message. The messages convey numeric quantities which are unpredictable by an external adversary, so that she cannot reduce the round-trip time by guessing and transmitting in advance the messages. Moreover, the correct response is unpredictable by the responding device itself before having received the challenge. In this way, a compromised device cannot reduce the round-trip time by responding in advance.

A basic example of such a distance bounding protocol is the following:

M1: $B \longrightarrow A : \mathrm{commit}(b, open)$
M2: $A \longrightarrow B : a$
M3: $B \longrightarrow A : b \otimes a$
M4: $B \longrightarrow A : open, b, a, \mathrm{MAC}_k(b, a),$

where $A$ is the party that measures the round-trip time (for us, the drone), $B$ is the untrusted party (for us, the device), $a$ and $b$ are randomly generated numbers, $k$ is the secret shared by $A$ and $B$, and $\mathrm{MAC}_k(\cdot)$ is a function computing a Message Authentication Code. The function $\mathrm{commit}(b, open)$ implements a commitment scheme, by which $B$ commits to using a particular quantity $b$ without revealing it unless after having transmitted the corresponding quantity $open$. M2 and M3 are the challenge and the response messages, between which $A$ measures the round-trip time. Such a protocol has been adapted by Poturalski et al. [16] for the IEEE 802.15.4a Ultra-Wideband (UWB) PHY protocol [9], which allows us for a precision of few centimeters in the distance estimation. From now on, we will say “$A$ measures $B$” as a shorthand for “$A$ measures its distance from $B$ by means of a distance bounding protocol.”

*Verifiable multilateration* [21] is a provably secure technique to determine a position, which leverages distance bounding. In verifiable multilateration, the position of a *node* is determined by measuring the distances between the node and at least three *anchors* whose positions are known (Fig. 1). The distance measurements are performed by means of distance bounding protocols. The node's position is computed by trilateration, and is accepted only if it lies inside the triangle formed by the anchors (*verifiable triangle*). Otherwise, it is discarded as untrusted. Indeed, if an adversary wants to falsify a position measurement inside the verifiable triangle, then she must perform a reduction attack against at least one distance bounding protocol, which is infeasible. Note that the coverage of verifiable multilateration is only the verifiable triangle, because the outside positions are discarded. In the case in which more than three anchors are involved in the localization process, the position is accepted only if it lies in at least one of the verifiable triangles formed by a triplet of anchors. Verifiable multilateration can defend against different threat models: an external adversary or a compromised node.

## 4 Drone-Based Verifiable Multilateration

Verifiable multilateration is able to securely measure the position of a node, but it needs an expensive infrastructure of fixed anchors. In case of a set of nodes sparsely deployed on a large area and with a limited communication range, it is necessary to deploy many anchors to reach them all. In addition, the coverage is restricted to the verifiable triangle only, so it is not enough that three anchors are within the communication range: they also have to “surround” the node in order to locate it securely. In the average case, this increases the number of anchors necessary to cover a given area, as shown in [6]. Finally, the anchors must be truly “fixed”, otherwise an adversary could simply move one of them to jeopardize the security of the system. This makes the infrastructure cost even higher, since the anchors cannot be attached to the ground or to the walls in a cheap and insecure manner. The scalability challenges of verifiable multilateration have been studied by Čapkun and Hubaux [21]. In particular, they proposed to place the anchors following a grid of regular triangles, in order to minimize the anchors necessary to cover a given area. Although a regular anchor placement can sensibly improve the scalability, the number of anchors cannot scale better than linearly with the size of the area to cover.

With the increased availability of drones [11] on the market, it became affordable to replace many fixed anchors with a single drone. The drone follows a *path*, touching a sequence of *waypoints*. At each waypoint it acts as an anchor (Fig. 2), and performs one or more distance bounding protocols with the nodes on the ground. If the drone measures a node on a waypoint, by extension we say that such waypoint measures that node, or that such waypoint is a *measuring waypoint* for that node.

The mechanism just described solves the scalability issues of verifiable multilateration. The problem becomes now how to find a convenient path for the drone in order to securely measure a set of positions.

Note that the energy consumption of the UWB distance bounding protocols is negligible compared to the consumption of the drone motors. As an example, a DJI Phantom

4 Professional drone[1] mounts an $81.3\,\mathrm{Wh}$ battery, and it can flight for a maximum time of 28 minutes. This means that the motors consume about $174\,\mathrm{W}$. On the other hand, a DecaWave DWM1000 UWB transceiver[2] consumes far less: about $211\,\mathrm{mW}$ in receive mode, $102\,\mathrm{mW}$ in transmit mode, and $6.6\,\mu\mathrm{W}$ in sleep mode. Note also that the drone can save the energy consumption of transmitting the measured distances during the mission. It is sufficient that the drone stores them locally, and then transmits them in bulk once landed. Then, the position of the nodes can be determined from the measured distances with a full-resource device.

The nodes are required to be static, or at least not to move during the time period from their first distance bounding execution to the last one. If a node moves, then its estimated position will not be reliable. More precisely, if the node is in three different positions at the times of the distance bounding executions, then the estimated position will be somewhere between these three positions. Such an estimated position may not correspond to any of the positions that the node assumed in time. The problem of drone-based verifiable multilateration in the presence of mobile nodes is interesting, but it falls outside the scope of the present paper. In this paper we consider the nodes to be static, and we focus on the path planning problem.

## 4.1 System Model

We suppose that the drone shares a different secret $k$ with each node, by which the distance bounding protocols are executed. To distribute in a secure manner the secrets to the nodes, a generic key deployment method (e.g., [7]) can be used. This falls outside the scope of the present paper.

In our system, all the nodes are on the ground, while the drone flies at a non-negligible *altitude* ($h$). We imagine the waypoints to be projected onto the ground. The drone "visits" a waypoint when its position is above the waypoint. The verifiable triangle is considered to be projected onto the ground too.

A *path* is a sequence of *waypoints* $\{W_1, \ldots, W_m\}$, each of which is a point on the Cartesian plane. The drone visits the waypoints in the order specified by the sequence. We require that the path is *closed*, in the sense that the drone goes again to the first waypoint at the end. The first waypoint ($W_1$) is also called the *home waypoint*. It is a special waypoint, since it is in a predefined position, and cannot be changed by the path planning algorithm. The drone is supposed to take off from the home waypoint, perform the mission, and land at the home waypoint again.

Due to many factors, the movement of the drone is not perfectly controllable, because for example the wind strongly affects it. Moreover, the drone movement has to respect some dynamic constraints, for example it cannot change direction instantaneously. Due to this, the "true" waypoints actually visited by the drone (*actual waypoints*, $W'_i$) could be different from the planned ones. We call the distance between the planned and the actual waypoint the *waypoint control error*. We assume that the waypoint control errors are bounded, and we call such a bound the *waypoint control precision* ($\gamma_W$):

$$\|W_i - W'_i\| \leq \gamma_W. \tag{1}$$

The actual waypoints are unknown at the time of the path planning. However, the drone can measure its position during the mission, so the actual waypoints are measured and can be used to estimate the position of the node.

The altitude is also not perfectly controllable, so the "true" altitudes of the drone at each waypoint (*actual altitudes*, $h'_i$) could be different from the planned one. We call the absolute difference between the planned and the actual altitude the *altitude control error*. We assume that the altitude control errors are bounded, and we call such a bound the *altitude control precision* ($\gamma_h$):

$$|h - h'_i| \leq \gamma_h. \tag{2}$$

The actual altitudes are unknown at the time of the path planning. However, we assume that the drone measures them during the mission, so they can be used to estimate the position of the node.

When the drone performs a distance bounding protocol with a node, it measures the line-of-sight distance, which we call the *slant distance*. We suppose that the drone has a limited *communication range* ($s_{max}$). Nodes having a slant distance beyond the communication range could be impossible to reach with a distance bounding protocol. For the aim of the localization, we are interested in the projection of the slant distance onto the ground, which we call the *ground distance*. The drone at the $i$-th waypoint computes a ground distance $d$ by:

$$d = \sqrt{s^2 - {h'_i}^2}, \tag{3}$$

where $s$ is the measured slant distance. Once the drone has collected three or more ground distances from a node, it can determine the position of such a node by trilateration. Without measurement errors, the circumferences centered on the measuring waypoints and with radii the computed ground distances (*measurement circumferences*) will intersect in the position of the node. In the practical case, the measured slant distance will always be affected by some error (*slant error*), and thus the computed ground distance will be affected by an error too (*ground error*). Moreover, the drone position and altitude will be affected by measurement errors as well (*waypoint measurement error* and *altitude measurement error*). All these error sources make the measurement circumferences not to intersect in a point. We thus estimate the position $\tilde{X}$ (*measured position*) by solving the following least squares problem:

$$\underset{\tilde{X}}{\text{minimize}} \quad \sum_{i \in \mathcal{R}} \left( \|\tilde{X} - W'_i\| - d_i \right)^2,$$

where $\mathcal{R}$ are the indices of the measuring waypoints of the node, and $d_i$ is the ground distance from the $i$-th waypoint. The distance between the true and the measured position is the *positioning error*.

We distinguish two types of mission: the *localization mission* and the *verification mission*. In a localization mission, we do not have any prior knowledge of the position of the nodes. We only know that the nodes lie somewhere inside a

1. https://dl.djicdn.com/downloads/phantom_4/en/Phantom_4_User_Manual_en_v1.2_20160805.pdf.

2. http://www.decawave.com/sites/default/files/product-pdf/dwm1000-product-brief.pdf.

given *deployment area* $D \subset \mathbb{R}^2$. In this paper, we suppose the deployment area to be rectangular, of sides $D_A \times D_B$.

On the other hand, in a verification mission we assume to have prior knowledge of the position of the nodes, from which we compute the drone's path. We call them *prior positions* ($N_i$). The prior positions are not trusted, therefore we want to securely verify them by means of verifiable multilateration. If the positions determined by verifiable multilateration are not consistent with the prior ones, an attack is detected. The prior positions could be imprecise, and thus different from the *actual positions* ($N_i'$). We call the distance between the prior and the actual position the *prior position error*. We assume that the prior position errors are bounded, and we call such a bound the *prior position precision* ($\varepsilon_N$):

$$\|N_i - N_i'\| \leq \varepsilon_N. \tag{4}$$

Note that in a localization mission we need sometimes to measure a node from more than three waypoints. Indeed, we must measure it three times to get a first estimate of its position, but this position could be outside the verifiable triangle. In this case, we need to measure the node from additional waypoints to include it in a verifiable triangle. To include as many nodes as possible in the verifiable triangles during a localization mission, we assume that the drone measures all the nodes within its communication range at each waypoint. On the other hand, in a verification mission the drone measures each node from exactly three waypoints, determined by the path planning algorithm.

### 4.2 General Requirements Of The Path

For both localization and verification missions, the path must respect the following requirements.

- **Communication range requirement.** Each waypoint must measure nodes within the communication range. Farther nodes could be impossible to reach with a distance bounding protocol.
- **Verifiable triangle requirement.** Each node has to be measured from three or more distinct waypoints, and at least a verifiable triangle formed by them must contain the node. This is required for the localization to be secure.
- **Control error tolerance requirement.** The above two requirements have to be tolerant to control errors. The drone should not miss to measure a node or fail to include it in the verifiable triangle due to control errors.
- **Path length requirement.** The path length should be as short as possible. This is preferable for saving time and drone's battery life.

In the following, we will present three path planning algorithms that fulfill such requirements: LocalizerBee, VerifierBee, and PreciseVerifierBee. LocalizerBee produces paths for localization missions, while VerifierBee and PreciseVerifierBee for verification missions. PreciseVerifierBee is an extension of VerifierBee which offers a provable bound on the positioning error, at a cost of a longer path.

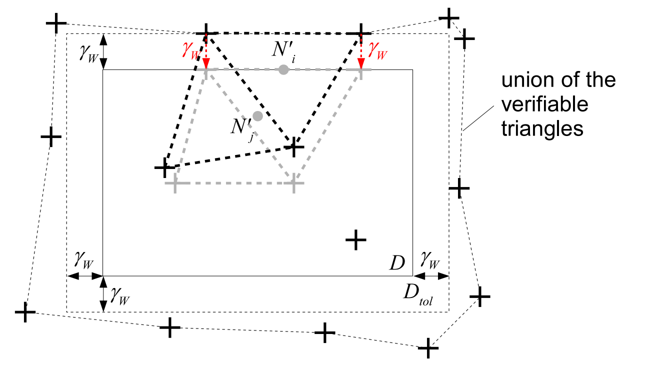


Fig. 3. Worst-case control error for verifiable triangle requirement in a localization mission. The black crosses are the planned waypoints, while the gray crosses are the actual ones. The gray dashed triangle is the actual verifiable triangle. The gray dots are the positions of the nodes.

## 5 LocalizerBee

This section is structured as follows. In Section 5.1 we formalize the problem of determining a path for localization missions which fulfills the requirements of Section 4.2. In Section 5.2 we describe LocalizerBee, an algorithm which solves such a problem.

### 5.1 Problem Formalization

In a localization mission, we do not have any prior knowledge of the position of the nodes. We only know that the nodes lie somewhere inside the deployment area. To respect the verifiable triangle requirement, given a set of planned waypoints forming a set of verifiable triangles, we have to be sure that each node is contained in at least one of them. We can reach this by imposing that the verifiable triangles are contiguous, and that their union includes the whole deployment area. In this way, we do not leave any part of the deployment area uncovered. However, this is not enough, since we have to be tolerant also to control errors. In the presence of control errors, the verifiable triangle actually drawn by the drone (*actual verifiable triangle*) could be different from the planned one. As a consequence, the node could lie outside the verifiable triangle again. To avoid this, it is sufficient that the union of the planned verifiable triangles is a superset of the deployment area plus four $\gamma_W$-wide borders. We call such an area the *tolerance area* ($D_{tol}$). We can prove this by considering the worst case, shown in Fig. 3. The verifiable triangle has a side parallel to the boundary of the deployment area, where the node $N_i'$ is. The actual waypoints are shifted (with respect to the planned ones) of $\gamma_W$ the direction orthogonal to the boundary. The actual verifiable triangle must contain the node even in this case. By geometrical evidence, we obtain this if the union of the planned verifiable triangles is a superset of the tolerance area. Note that a node which is not at the extreme border of the deployment area is not influential for the worst case. For example the node $N_j'$ in Fig. 3 is always contained either in the upper-right or in the lower-left verifiable triangle, whatever the waypoint control error is.

We also have to fulfill the communication range requirement, so we have to impose that the node is reachable by all the waypoints forming the verifiable triangle. To do this even in the presence of control errors, it is sufficient that the sides of the verifiable triangles are smaller than or equal to a *maximal triangle side* ($L_{max}$), defined as:

$$L_{max} \triangleq \sqrt{s_{max}^2 - (h+\gamma_h)^2} - 2\gamma_W. \tag{5}$$

Such a formula considers the worst-case control error, shown in Fig. 4.

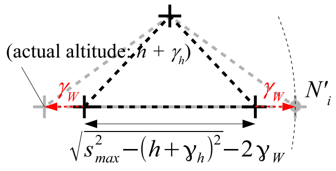

(actual altitude: $h + \gamma_h$)
$\gamma_W$
$\gamma_W$
$N'_i$
$\sqrt{s_{max}^2 - (h+\gamma_h)^2} - 2\gamma_W$

Fig. 4. Worst-case control error for communication range requirement in a localization mission.

The actual waypoint on the right is shifted of $\gamma_W$ toward right with respect to the planned waypoint. So the actual verifiable triangle includes the position $N'_i$. At the same time, the actual waypoint on the left is shifted of $\gamma_W$ toward left. The position $N'_i$ must be within the communication range of the left waypoint also in this case.

### 5.2 Proposed Algorithm

We present LocalizerBee, an algorithm which solves the problem formalized in Section 5.1. LocalizerBee uses a Traveller Salesman Problem (TSP) solver algorithm as a building block. The TSP solver is not required to be optimal, but rather to find an approximate shortest path that visits a set of points, and then returns to the first point (closed path). Of course the performances of the TSP solver will affect those of LocalizerBee both in terms of path length and processing time.

LocalizerBee operates in two phases: (i) waypoint grid construction; (ii) waypoint ordering. In the waypoint grid construction phase, it builds a set of waypoints forming a grid of isosceles triangles which covers the whole tolerance area. The triangles have the same base $T_A$ and the same height $T_B$, and they are placed as shown in Fig. 5.

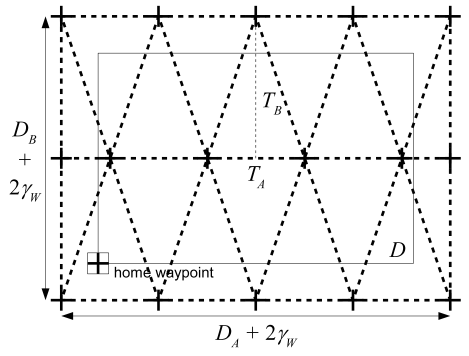


Fig. 5. Waypoint grid construction phase of LocalizerBee.

Note that we added two halved triangles on the left and the right sides of each triangle row, in such a way that the union of the verifiable triangles is a rectangle. The home waypoint is finally added to such a set. We use a grid of isosceles triangles and not, for example, regular triangles.

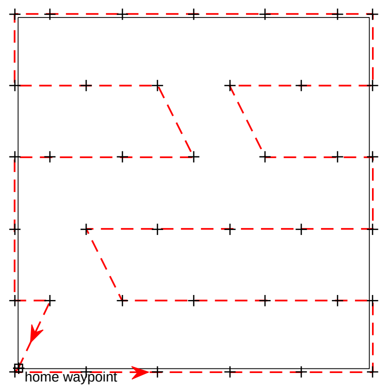


Fig. 6. Example of LocalizerBee path. The black crosses are the waypoints, the red dashed line is the path.

Indeed, isosceles triangles are more flexible, because they can be "resized" along both dimensions. This allows us to cover the whole tolerance area as tightly as possible, without wasting path length for covering external areas uselessly. The base and the height of the triangles must be as long as possible, without violating the maximal triangle side. To this aim, given a base $T_A$, the longest feasible height $T_{Bmax}$ is:

$$T_{Bmax} = \sqrt{L_{max}^2 - (T_A/2)^2}. \tag{6}$$

LocalizerBee computes the base and the height of the triangles as follows.

$$T_A = (D_A + 2\gamma_W) / \left\lceil \frac{D_A + 2\gamma_W}{L_{max}} \right\rceil \tag{7}$$

$$T_B = (D_B + 2\gamma_W) / \left\lceil \frac{D_B + 2\gamma_W}{T_{Bmax}} \right\rceil . \tag{8}$$

In this way, we fill tightly the tolerance area with an integer number of triangle rows and an integer number of triangles on each row. More precisely, there are $\left\lceil \frac{D_B+2\gamma_W}{T_{Bmax}} \right\rceil$ triangle rows, each containing $2 \cdot \left\lceil \frac{D_A+2\gamma_W}{L_{max}} \right\rceil - 1$ unhalved triangles plus 2 halved ones.

In the waypoint ordering phase, LocalizerBee executes the TSP solver to connect all the waypoints. Fig. 6 shows an example of LocalizerBee path for a deployment area $D = 1000\,\mathrm{m} \times 1000\,\mathrm{m}$, with $s_{max} = 300\,\mathrm{m}$ (as claimed by DecaWave for their IEEE 802.15.4a UWB transceivers [5]), $h = 150\,\mathrm{m}$, $\gamma_W = 10\,\mathrm{m}$, and $\gamma_h = 10\,\mathrm{m}$. We fixed the home waypoint to be in the south-west corner of the deployment area. As TSP solver, we employed the Chained Lin-Kernighan heuristic implemented by the University of Waterloo[3], which provides for a state-of-the-art trade-off between efficiency and optimality [1].

## 6 VerifierBee

This section is structured as follows. In Section 6.1 we formalize the problem of determining a path for verification missions which fulfills the requirements of Section 4.2. In Section 6.2 we describe VerifierBee, an algorithm which solves such a problem.

3. http://www.math.uwaterloo.ca/tsp/concorde/downloads/downloads.htm

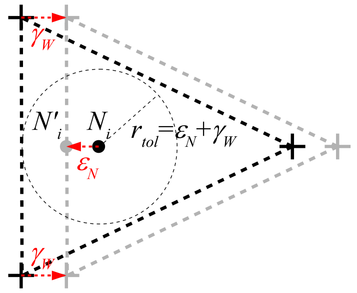


Fig. 7. Worst-case control error for verifiable triangle requirement in a verification mission. The black dot is the prior position, while the gray dot is the actual one.

## 6.1 Problem Formalization

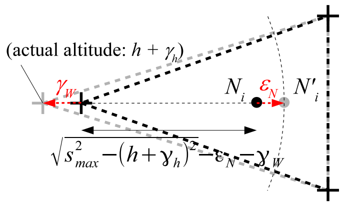
In a verification mission, we have prior knowledge of the position of the nodes. By leveraging this information, we can design a path planning algorithm which produces shorter paths than LocalizerBee. The prior positions are affected by an error bounded by $\varepsilon_N$. To be sure that a verifiable triangle contains the $i$-th node, the verifiable triangle must include the whole circle centered in $N_i$ and with radius $\varepsilon_N$. However, this is not enough, since we have to be tolerant also to control errors. To assure this, it is sufficient that the verifiable triangle contains the whole circle centered in $N_i$ and with radius $\varepsilon_N + \gamma_W$. We can prove this by considering the worst case, shown in Fig. 7. The actual waypoints are shifted (with respect to the planned ones) of $\gamma_W$ on the direction orthogonal to the side of the verifiable triangle. The real node's position is shifted (with respect to the prior one) of $\varepsilon_N$ on the opposite direction. The actual verifiable triangle must contain the node even in this case. By geometrical evidence, we obtain this if the planned verifiable triangle contains the circle with center $N_i$ and radius $\varepsilon_N + \gamma_W$. We call such a radius the *tolerance radius* ($r_{tol}$):

$$r_{tol} \triangleq \varepsilon_N + \gamma_W. \tag{9}$$

We also have to fulfill the communication range requirement, so we have to impose that the node is reachable by all the waypoints forming the verifiable triangle that contains it. To do this even in the presence of control errors, it is sufficient that the ground distances between the planned waypoints and the prior position are less than or equal to a *maximal ground distance* ($d_{max}$), defined as:

$$d_{max} \triangleq \sqrt{s_{max}^2 - (h + \gamma_h)^2} - \varepsilon_N - \gamma_W. \tag{10}$$

Such a formula considers the worst-case control error, shown in Fig. 8.



(actual altitude: $h+\gamma_h$)
$\gamma_W$
$N_i$
$\varepsilon_N$
$N'_i$
$\sqrt{s_{max}^2-(h+\gamma_h)^2}-\varepsilon_N-\gamma_W$

Fig. 8. Worst-case control error for communication range requirement in a verification mission.

The actual waypoint on the left is shifted of $\gamma_W$ toward left with respect to the planned waypoint. At the same time, the actual position $N_i'$ is shifted toward right with respect to the planned position $N_i$. The position $N_i'$ must be within the communication range of the left waypoint also in this case.

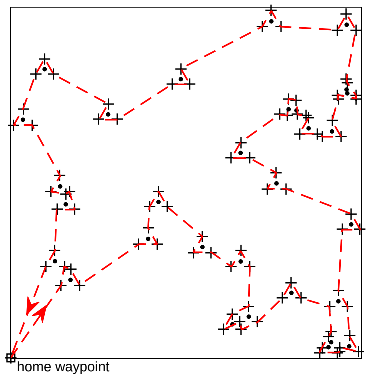


Fig. 9. Example of initial path of VerifierBee with 30 nodes. The black dots are the nodes.

## 6.2 Proposed Algorithm

We present VerifierBee, an algorithm which solves the problem formalized in Section 6.1. VerifierBee uses a TSP solver as a building block to find a first feasible solution. Then, such a solution is iteratively improved, following a local search heuristic. Of course the performances of the TSP solver will affect those of VerifierBee both in terms of path length and processing time.

VerifierBee operates in three phases: (i) initial path construction; (ii) iterative improvement; (iii) waypoint reordering. In the initial path construction phase, it builds a set of waypoints: the home waypoint plus three waypoints for each node, placed at fixed positions to form a *minimal verifiable triangle*. The minimal verifiable triangle is a regular triangle centered on $N_i$ and with the vertices at $2r_{tol}$ from the center. By geometry, this is the smallest distance with which the regular verifiable triangle respects the $r_{tol}$ constraint. VerifierBee orients all the minimal verifiable triangles with a vertex toward north, but the successive iterative improvement phase may rotate and distort the triangles in order to find shorter paths. After having built the set of waypoints, we run the TSP solver on them to find an approximate optimal path that touches them all. The initial path is thus complete, and it is formed by $3n + 1$ waypoints, where $n$ is the number of nodes. Fig. 9 shows an example of initial path for 30 nodes randomly distributed in a deployment area $D = 1000\,\text{m} \times 1000\,\text{m}$, with $\gamma_W = 10\,\text{m}$ and $\varepsilon_N = 5\,\text{m}$. Note that the drone passes very close to each node. This makes the initial path quite sub-optimal in length, since the drone does not use its full communication range.

After having built the initial path, VerifierBee changes it iteratively, following a local search heuristic. At each step, VerifierBee analyzes the possible changes (e.g., moving a waypoint in another position) and applies the most convenient one, that is the one that decreases more the total path length. The iterative improvement phase terminates when

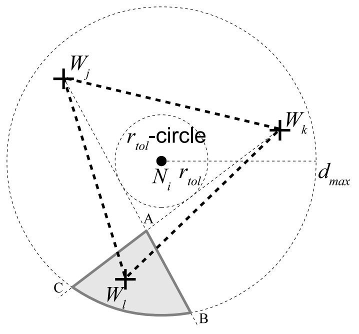


Fig. 10. Freedom space of $W_l$ (the gray area) in VerifierBee.

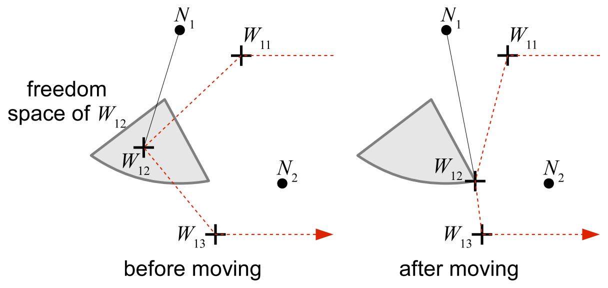


Fig. 11. Example of waypoint moving in VerifierBee.

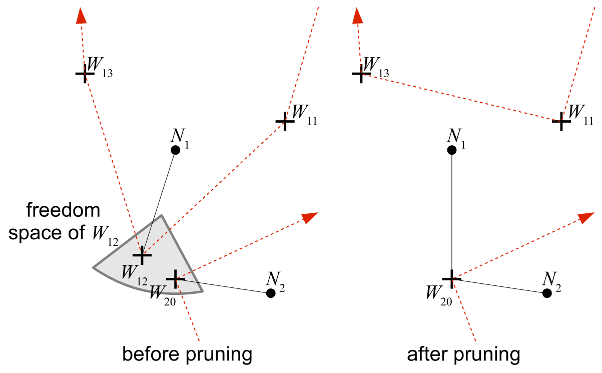


Fig. 12. Example of waypoint pruning in VerifierBee.

**Algorithm 1:** VerifierBee

```
Require: $\{N_i\}$, $W_1$, $r_{tol}$, $d_{max}$
 1: $Path$ ← a set of waypoints, one on $W_1$,
      and the others on the minimal verifiable triangles.
 2: $Path \leftarrow \mathrm{SolveTSP}(Path)$ {initial path}
 3: loop
 4:   $Path \leftarrow \mathrm{IterativelyImprove}(Path, r_{tol}, d_{max})$
 5:   if $Path$ has not been improved then
 6:     exit loop
 7:   end if
 8:   $Path \leftarrow \mathrm{SolveTSP}(Path)$
 9:   if $Path$ has not been improved then
10:     exit loop
11:   end if
12: end loop
13: return $Path$
```

no change is possible or convenient anymore, meaning that we found a local minimum.

The changes are of two kinds: waypoint moving and waypoint pruning. *Waypoint moving* changes the position of a waypoint, while *waypoint pruning* removes a waypoint and "substitutes" it with another existing one. Note that the home waypoint cannot be moved nor pruned. Both moving and pruning make use of the concept of *freedom space*. The freedom space of a waypoint is the area where the waypoint can be moved without violating any constraint of the problem (all the other waypoints remaining fixed). It can be computed geometrically, as illustrated in Fig. 10. The curved border of the freedom space ($\overset{\frown}{BC}$) is the limit of the $d_{max}$ constraint. The straight borders ($\overline{AB}$ and $\overline{AC}$) are the limits of the $r_{tol}$ constraint.

*Waypoint moving* changes the position of a waypoint, in such a way to shorten the global path. Fig. 11 shows an example, in which $W_{12}$ is moved so that the drone shortens the path going from $W_{11}$ to $W_{13}$. The best position where to move a waypoint is always: (i) somewhere on the border of the freedom space (like in Fig. 11), or (ii) coincident with another waypoint in the interior of the freedom space. In the latter case, we do not apply waypoint moving but rather waypoint pruning (see below), that is we eliminate the moved waypoint and substitute it with the coincident one.

*Waypoint pruning* removes a waypoint (*pruned waypoint*) and substitutes it with another existing one (*substitute waypoint*). The drone will not visit anymore the pruned waypoint. As a consequence, it will miss to execute a distance bounding protocol. The missing distance bounding is executed when the drone passes through the substitute waypoint. Waypoint pruning reduces the total number of waypoints in the path. Fig. 12 shows an example of waypoint pruning, in which $W_{12}$ is pruned and substituted by $W_{20}$. When the drone visits $W_{20}$, it runs two distance bounding protocols: one with $N_1$ and one with $N_2$. It is possible to prune a waypoint only when its freedom space contains the substitute waypoint. Note that, after pruning, the substitute waypoint has to measure two nodes instead of one. Consequently its freedom space will narrow, because it has to take into account the constraints relative to both nodes. The resulting freedom space is the intersection of the freedom spaces relative to the single nodes. In some cases, the freedom space of the waypoint to prune contains more than one waypoint. All these waypoints are suitable candidates to be the substitute waypoint. Which one to choose is indifferent in terms of path length. VerifierBee chooses the one which narrows less its freedom space, in such a way to leave more "freedom" to the next steps of the iterative improvement.

The iterative improvement phase may change the position and the number of the waypoints, but it does not change their order, which remains the same of the initial path. As a consequence, sometimes it is convenient to reorder the waypoints by running the TSP solver again. This is done in the waypoint reordering phase. The iterative improvement and the waypoint reordering phases are repeated, until the path length stops decreasing.

Algorithm 1 shows a pseudo-code description of VerifierBee. The function $\mathrm{SolveTSP}(\cdot)$ is our TSP solver. The function takes a path, reorders the waypoints to form an (approximate) optimal path, and then returns such a new

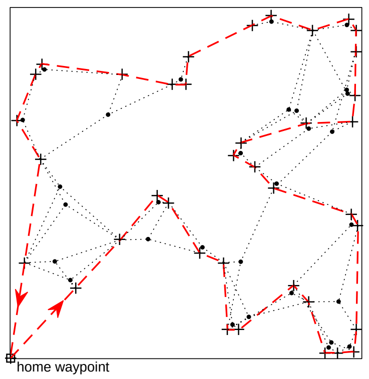


Fig. 13. Example of VerifierBee path. The black dotted lines are the distance bounding protocols.

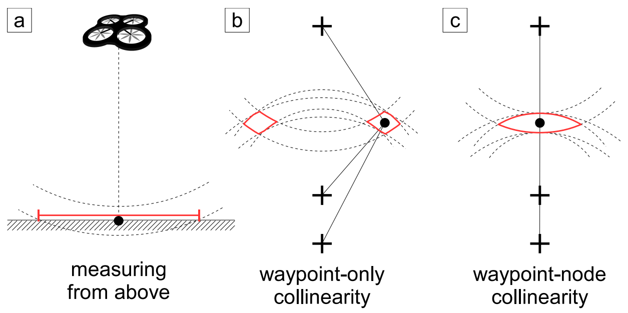


Fig. 14. Three possible bad geometry layouts.

path. The function $\text{IterativelyImprove}(\cdot)$ takes a path, performs the iterative improvement, and returns the resulting path.

Fig. 13 shows an example of VerifierBee path for the same 30 nodes of Fig. 9, with $s_{max} = 300\,\text{m}$, $h = 150\,\text{m}$, $\gamma_W = 10\,\text{m}$, $\gamma_h = 10\,\text{m}$, and $\varepsilon_N = 5\,\text{m}$. This path is much shorter than the initial path of Fig. 9. Note that many waypoints have been pruned and many others have been moved in more convenient positions.

## 7 PreciseVerifierBee

This section is structured as follows. In Section 7.1 we formalize the problem of determining a path for verification missions which fulfills the requirements of Section 4.2, and additionally offers a provable bound on the positioning error. In Section 7.2 we describe PreciseVerifierBee, an algorithm which solves such a problem.

### 7.1 Problem Formalization

VerifierBee builds a path able to securely verify a set of nodes. However, it does not take into consideration the positioning precision at all. In the process of finding an approximate optimal path, VerifierBee could place the waypoints in such a way to form "bad" geometry layouts, for example three waypoints collinear with the measured node. This in turn can cause a big error in the position estimation. Fig. 14 shows the three possible bad geometry layouts. If a waypoint is in plumb-line above the measured node (Fig. 14a), then the computed ground distance will be affected by

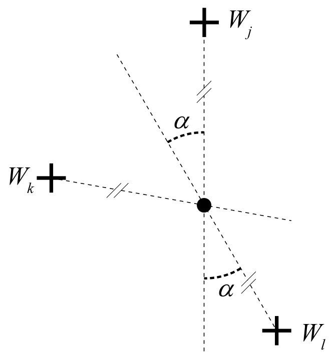


Fig. 15. Angular aperture.

a big error. Even if the ground distances are precise, if the waypoints are highly collinear with themselves (*waypoint-only collinearity*, Fig. 14b) or with the node (*waypoint-node collinearity*, Fig. 14c), then the computed position will be affected by a big error. In VerifierBee, the presence of the $r_{tol}$ constraint avoids the possibility of waypoint-only collinearity. On the other hand, the possibility of measuring from above or of having node-waypoint collinearity are avoided only if $r_{tol}$ is sufficiently large. The greater $r_{tol}$ is, the more they are mitigated. However, the $r_{tol}$ constraint is thought for fulfilling the verifiable triangle requirement, and should not be used for other purposes. To avoid bad geometry layouts, it is preferable to introduce specialized constraints. We thus impose two additional constraints to the path: the *minimal ground distance* and the *minimal angular aperture*. We also show that, by imposing these new constraints, we can guarantee a *provable* bound on the positioning error.

The minimal ground distance ($d_{min}$) forces the waypoints not to be in plumb-line above the measured node, in such a way to avoid the bad geometry layout of Fig. 14a. More precisely, we impose that the ground distances of the planned waypoints from the prior position are greater than or equal to the minimal ground distance. The minimal angular aperture ($\alpha_{min}$) forces the waypoints not to be too collinear with the prior position of the measured node, in such a way to avoid the bad geometry layout of Fig. 14c. We define the *angular aperture* ($\alpha$) as the smallest angle among those formed by the lines passing by the prior position and the planned waypoints (Fig. 15). Note that the angular aperture is small also in the case of only two of the three waypoints collinear or almost collinear with the node, which is also a bad geometry layout. The angular aperture ranges from $0°$ to $60°$. When $\alpha = 0°$ we have the worst layout, with at least two waypoints perfectly collinear with the node. When $\alpha = 60°$ we have the best layout, with the waypoints forming equal angles with the node. We impose that the angular aperture is greater than or equal to the minimal angular aperture.

We show now that, if we assume zero prior position error, zero error in the control and the measurement of the drone position, and bounded slant error, then by imposing a minimal ground distance and a minimal angular aperture we can guarantee a *provable* bound on the positioning error. We suppose that the slant error is bounded and we call such a bound the *slant precision* ($\varepsilon_s$). In the IEEE 802.15.4a UWB

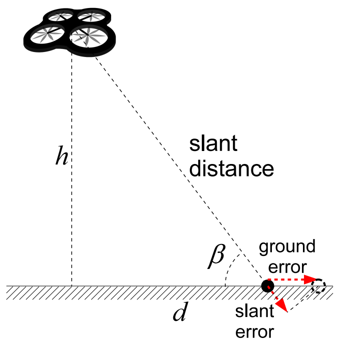


Fig. 16. Relationship between the slant and the ground error.

standard [9], the slant precision is usually of the order of centimeters. For example, the transceivers commercialized by DecaWave, which are compliant to such a standard, have a precision of 10 centimeters [5]. We define the *ground precision* ($\varepsilon_d$) and the *positioning precision* ($\varepsilon_{\tilde{X}}$) as the bounds on the ground error and the positioning error respectively. We state the following:

**Theorem 1.** *Supposing zero prior position error ($\varepsilon_N = 0\,\mathrm{m}$), zero control error ($\gamma_W = 0\,\mathrm{m}$, $\gamma_h = 0\,\mathrm{m}$), and zero waypoint and altitude measurement errors, if a node is measured with a slant precision $\varepsilon_s$ by a drone at an altitude $h$ on three waypoints respecting a $d_{min} \gg \varepsilon_s$ and an $\alpha_{min} > 0°$, then the ground precision will be:*

$$\varepsilon_d = \varepsilon_s \cdot \sqrt{1 + (h/d_{min})^2}, \tag{11}$$

*and the positioning precision will be:*

$$\varepsilon_{\tilde{X}} = \varepsilon_s \cdot \frac{\sqrt{1 + (h/d_{min})^2}}{\sin(\alpha_{min}/2)}. \tag{12}$$

*Proof.* Since we supposed that the waypoint measurement error and the altitude measurement error are zero, then the positioning error will depend only on the slant errors. Also, since we supposed that the prior position error and the control error are zero, then $d_{min}$ and $\alpha_{min}$ will be respected also by the actual waypoints and the actual node position. In this case, if the ground distance is sufficiently large compared to the slant precision (this is implied by $d_{min} \gg \varepsilon_s$), then a linear relationship will hold between the slant and the ground error. Fig. 16 shows an evidence of this. Such a relationship is given by:

$$\text{(ground error)} = \frac{\text{(slant error)}}{\cos(\beta)} = \text{(slant error)} \cdot \sqrt{1 + (h/d)^2}, \tag{13}$$

where $\beta$ is the angle of incidence of the slant distance to the ground, and $d$ is the ground distance (see Fig. 16). If $d$ is never smaller than $d_{min}$, then this becomes a relationship between the slant and the ground precision:

$$\varepsilon_d = \varepsilon_s \cdot \sqrt{1 + (h/d_{min})^2}. \tag{14}$$

This proves the first statement of the theorem (Eq. 11). Now, note that if all the ground distances are sufficiently large compared to the ground precision, then the measurement circumferences of the trilateration are linearizable at the node position, without the problem solution changing significantly. In this way, the measurement circumferences are approximated with *measurement lines*, which are perpendicular to the waypoint-node direction (Fig. 17).

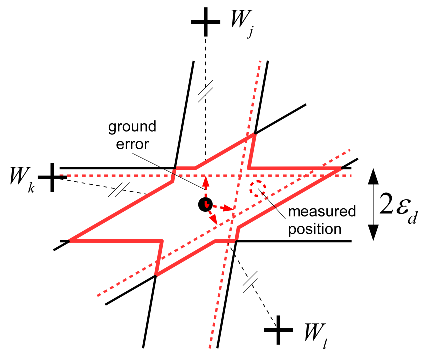


Fig. 17. Measurement lines forming a "star" shape. The dashed red lines are the measurement lines. The black solid lines are the "extreme" measurement lines. The red polygon is the star shape.

Moreover, if the waypoints are not perfectly collinear with the node (this is implied by $\alpha_{min} > 0°$), then the measured position will lie inside the "star" shape shown in Fig. 17. Such a shape is formed by three pairs of parallel "extreme" measurement lines, each relative to one of the three waypoints. The extreme measurement lines correspond to the three ground distances affected by the two extreme ground errors: $-\varepsilon_d$ and $+\varepsilon_d$. The ground error will be somewhere within $[-\varepsilon_d, +\varepsilon_d]$, thus each measurement line will be somewhere within the pair of extreme measurement lines. Note that the measured position will always be inside the triangle formed by the three measurement lines (see Fig. 17). This is due to the properties of least squares optimization. Since such a triangle is always fully contained inside the star shape, then the measured position will be inside the star shape too. The positioning error is thus bounded by the distance between the center of the star (which is the actual position of the node) and its farthest vertex (Fig. 18). Such a distance depends on the ground precision and the angular aperture:

$$\text{(positioning error)} \leq \varepsilon_d \cdot \frac{1}{\sin(\alpha/2)}. \tag{15}$$

If $\alpha$ is never smaller than $\alpha_{min}$, then this becomes a relationship between the ground and the positioning precision:

$$\varepsilon_{\tilde{X}} = \varepsilon_d \cdot \frac{1}{\sin(\alpha_{min}/2)}. \tag{16}$$

From (14) and (16) we prove the second statement of the theorem (Eq . 12). □

With Theorem 1, we can give a guarantee on the maximum positioning error by fulfilling two simple geometric constraints: $d_{min}$ and $\alpha_{min}$. Such a guarantee holds only if the prior position error is zero, and the measurement and the control errors on the drone position are zero, which is not true in the practical case. However, in Section 8.1 we will show that this approximation is reasonable, and the guarantee of Theorem 1 is fulfilled even in the presence of some errors on the prior positions and on the control and the measurement of the drone position.

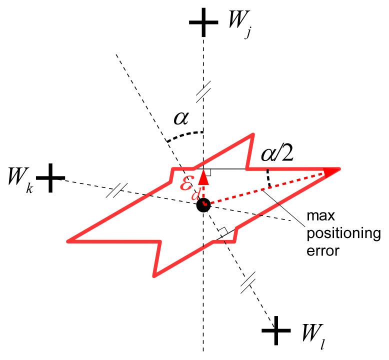


Fig. 18. Maximum positioning error with a given ground precision and a given angular aperture.

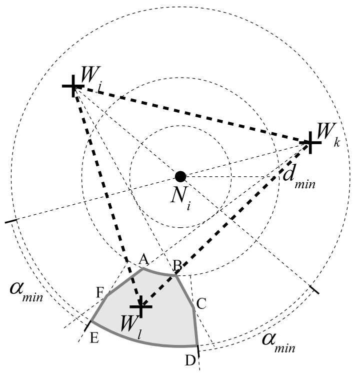


Fig. 19. Freedom space of $W_l$ (the gray area) in PreciseVerifierBee.

To guarantee a given $\varepsilon_{\tilde{X}}$, we have to choose a couple of values for $d_{min}$ and $\alpha_{min}$. However, this choice is not univocal. We can choose a larger $d_{min}$ and a smaller $\alpha_{min}$, or vice versa. In general, with different $d_{min}$-$\alpha_{min}$ couples a path planning algorithm will produce different paths. The optimal couple is the one with which the algorithm produces the shortest path. For PreciseVerifierBee, we will simulatively determine such an optimal couple in Section 8.1.

### 7.2 Proposed Algorithm

We present PreciseVerifierBee, an algorithm which solves the problem formalized in Section 7.1. PreciseVerifierBee is an extension of VerifierBee, which respects the additional $d_{min}$ and $\alpha_{min}$ constraints. In this way, it guarantees a bound on the positioning error, at a cost of a longer path. The general structure of the VerifierBee algorithm (Algorithm 1) remains the same, except that the initial path construction and the iterative improvement phases have to take into account the two additional constraints. In the initial path construction phase, the vertices of the minimal verifiable triangle must be at $\max\{2r_{tol}, d_{min}\}$ from the center. By geometry, this is the smallest distance with which the regular verifiable triangle respects both the $r_{tol}$ and the $d_{min}$ constraints. Note that, being the minimal verifiable triangle a *regular* triangle, it enjoys the best possible angular aperture ($\alpha = 60°$), and thus the $\alpha_{min}$ constraint is always fulfilled. In the iterative improvement phase, the freedom space of the waypoints has to be narrowed to take into account the additional constraints. Fig. 19 shows an example of freedom space in PreciseVerifierBee. The curved borders of the freedom space are the limits of the $d_{min}$ constraint ($\overline{\text{AB}}$) and $d_{max}$ constraint ($\overline{\text{DE}}$). The straight borders are the limits of the $r_{tol}$ constraint ($\overline{\text{BC}}$ and $\overline{\text{AF}}$) and $\alpha_{min}$ constraint ($\overline{\text{CD}}$ and $\overline{\text{EF}}$).

Fig. 20 shows an example of PreciseVerifierBee path for the same 30 nodes of Fig. 13, with $s_{max} = 300\,\text{m}$, $h = 150\,\text{m}$, $\gamma_W = 10\,\text{m}$, $\gamma_h = 10\,\text{m}$, $\varepsilon_N = 5\,\text{m}$, $d_{min} = 0.721h = 108.15\,\text{m}$, and $\alpha_{min} = 40°$. The values of $d_{min}$ and $\alpha_{min}$ have been chosen to guarantee a positioning precision of $\varepsilon_{\tilde{X}} = 5.0\varepsilon_s$, and at the same time to heuristically minimize the path length (see Section 8.1). Note that this path is longer than the path in Fig. 13 produced by VerifierBee, because PreciseVerifierBee must fulfill additional constraints.

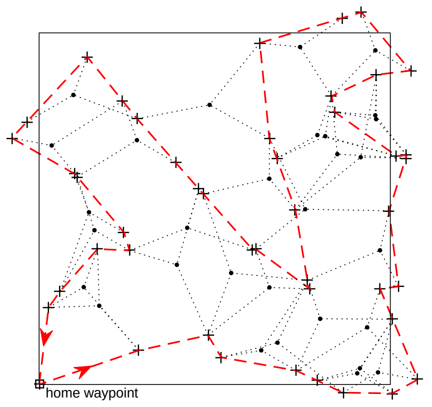


Fig. 20. Example of PreciseVerifierBee path.

## 8 Experimental Evaluation

We implemented LocalizerBee, VerifierBee and PreciseVerifierBee with the Matlab programming language and we tested their performance under different conditions. This section is organized as follows. Section 8.1 gives a heuristic parametrization of PreciseVerifierBee, and validates simulatively its precision guarantee in the presence of various error sources. Section 8.2 evaluates LocalizerBee, VerifierBee, and PreciseVerifierBee in terms of path length and processing time. Section 8.3 compares the three algorithms with the literature in terms of coverage, path length, and positioning error.

### 8.1 PreciseVerifierBee Parametrization And Validation

As anticipated in Section 7.1, in order to guarantee a given $\varepsilon_{\tilde{X}}$, we have to choose a couple of values for $d_{min}$ and $\alpha_{min}$. However, this choice is not univocal. We can choose a larger $d_{min}$ and a smaller $\alpha_{min}$, or vice versa. In general, with different $d_{min}$-$\alpha_{min}$ couples PreciseVerifierBee will produce different paths. The optimal couple is the one with which PreciseVerifierBee produces the shortest path, and it depends on the prior positions and the other constraints $r_{tol}$ and $d_{max}$. To find this optimal couple, the user should run PreciseVerifierBee several times with different couples, and then pick the one producing the shortest path. This is quite a burdensome operation. To avoid it, in Table 1 we give a set of heuristically optimal $d_{min}$-$\alpha_{min}$ couples, one for each value of positioning precision.

TABLE 1
Heuristically optimal $d_{min}$-$\alpha_{min}$ couples for PreciseVerifierBee

| $\varepsilon_{\tilde{X}}$ | $d_{min}$ | $\alpha_{min}$ |
|---|---|---|
| $3.0\varepsilon_s$ | $1.043h$ | $55°$ |
| $3.5\varepsilon_s$ | $0.918h$ | $50°$ |
| $4.0\varepsilon_s$ | $0.863h$ | $45°$ |

| $\varepsilon_{\tilde{X}}$ | $d_{min}$ | $\alpha_{min}$ |
|---|---|---|
| $5.0\varepsilon_s$ | $0.721h$ | $40°$ |
| $7.0\varepsilon_s$ | $0.540h$ | $35°$ |
| $10.0\varepsilon_s$ | $0.419h$ | $30°$ |

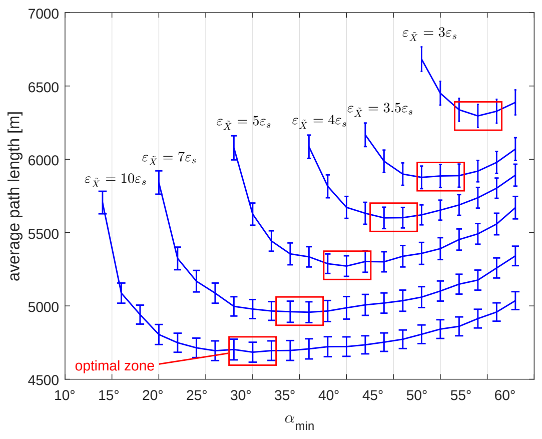


Fig. 21. Average path length of PreciseVerifierBee wrt $\alpha_{min}$, with different values of the precision guarantee. 95%-confidence intervals are displayed in error bars.

To obtain such heuristic values, we conducted a simulation campaign as follows. We run PreciseVerifierBee on a set of 20 prior positions randomly distributed on a deployment area $D = 1000\,\text{m} \times 1000\,\text{m}$. We fixed the home waypoint to be in the south-west corner of the deployment area. As TSP solver, we employed the Chained Lin-Kernighan heuristic implemented by the University of Waterloo. We supposed that the drone flies at an altitude of $h = 150\,\text{m}$, and has a communication range of $s_{max} = 300\,\text{m}$ (as claimed by DecaWave for their IEEE 802.15.4a UWB transceivers [5]). We fixed a prior position precision of $\varepsilon_N = 5\,\text{m}$, a waypoint control precision of $\gamma_W = 10\,\text{m}$, and an altitude control precision of $\gamma_h = 10\,\text{m}$. The constraint resulting from these parameters are $r_{tol} = 15\,\text{m}$ and $d_{max} = 238.77\,\text{m}$. We tested different positioning precisions, namely $\varepsilon_{\tilde{X}} \in \{3\varepsilon_s, 3.5\varepsilon_s, 4\varepsilon_s, 5\varepsilon_s, 7\varepsilon_s, 10\varepsilon_s\}$. For each positioning precision, we varied the $\alpha_{min}$ constraint within $\{2°, 4°, \ldots, 60°\}$, and we computed the corresponding $d_{min}$ constraint by inverting Equation 12:

$$d_{min} = \frac{h}{\sqrt{(\varepsilon_{\tilde{X}}/\varepsilon_s)^2 \cdot \sin^2(\alpha_{min}/2) - 1}}, \tag{17}$$

in such a way to obtain that positioning precision. Finally, we repeated the simulation for 100 different random sets of prior positions.

Fig. 21 shows the average length of the planned path with respect to $\alpha_{min}$. Note that, depending on the positioning precision, we cannot choose a too small value for $\alpha_{min}$, because the corresponding $d_{min}$ would be beyond $d_{max}$, causing the problem not to admit solutions. Note also that the strongest positioning precision we impose, the longer the produced paths will be, as expected. We can see that, for each positioning precision, there is an "optimal zone" for $\alpha_{min}$, for which the average path length reaches the minimum. To obtain the heuristic values of Table 1, we chose $\alpha_{min}$ as the approximate center of each optimal zone, and the corresponding $d_{min}$ computed by inverting Equation 12 in such a way to obtain that positioning precision.

We repeated the same simulation campaign with different numbers of nodes (namely: 10, 20, 50, 100), different $r_{tol}$ values (namely: $10\,\text{m}$, $12.5\,\text{m}$, $15\,\text{m}$, $17.5\,\text{m}$, $20\,\text{m}$), and different $d_{max}$ values (namely: $105.00\,\text{m}$, $177.09\,\text{m}$, $238.77\,\text{m}$, $296.29\,\text{m}$, $351.61\,\text{m}$). The position of the optimal zones does not change sensibly with the varying of these parameters. We conclude that the heuristic values of Table 1 have quite a broad validity.

As we said in Section 7.1, the positioning precision guarantee of PreciseVerifierBee holds only if the prior position error and the control error are zero, which is not true in the practical case. To check if this approximation is reasonable, we conducted a simulation campaign as follows. We run PreciseVerifierBee on a set of 20 prior positions randomly distributed on a deployment area $D = 1000\,\text{m} \times 1000\,\text{m}$. We fixed $\varepsilon_N = 5\,\text{m}$ and $\gamma_h = 10\,\text{m}$, and we tested different waypoint control precisions. We fixed a positioning precision of $\varepsilon_{\tilde{X}} = 5.0\varepsilon_s$, and the corresponding heuristic values of $d_{min} = 0.721h$ and $\alpha_{min} = 40°$ given by Table 1. Then, we simulated a verification mission which follows the produced path. We assumed that the waypoint measurement errors and the altitude measurement errors are bounded. We call such bounds respectively the *waypoint measurement precision* ($\varepsilon_W$) and the *altitude measurement precision* ($\varepsilon_h$). We fixed them to $\varepsilon_W = 10\,\text{cm}$ and $\varepsilon_h = 10\,\text{cm}$, which should be feasible with a good Differential GPS (DGPS) implementation mounted on the drone. To simulate the errors on the prior positions and on the control and the measurement of the drone position, we proceeded in the following way. For each node, we set its actual position at a random point within $\varepsilon_N$ from the prior one. For each waypoint, we set the actual waypoint at a random point within $\gamma_W$ from the planned one, and the actual altitude at a random value within $[-\gamma_h, +\gamma_h]$ from the planned one. Similarly, we set the waypoint measured by the drone at a random point within $\varepsilon_W$ from the actual waypoint, and the altitude measured by the drone at a random value within $[-\varepsilon_h, +\varepsilon_h]$ from the actual one. We fixed a slant precision of $\varepsilon_s = 10\,\text{cm}$ (claimed by DecaWave for their IEEE 802.15.4a UWB transceivers [5]). For each distance bounding, we simulated a random slant error within $[-\varepsilon_s, +\varepsilon_s]$. We computed the measured position and the corresponding positioning error as shown in Section 4.1. Finally, we repeated the whole simulation for 250 different random sets of prior positions.

Fig. 22 shows the distribution of the mission's worst positioning error with respect to the waypoint control precision. Note that the worst positioning error stays under the positioning precision stated by Theorem 1 also with non-zero prior position precision and non-zero control precision. We conclude that the precision guarantee of PreciseVerifierBee holds even in the presence of a reasonable amount of error in the prior positions and the measurement and the control of the drone position.

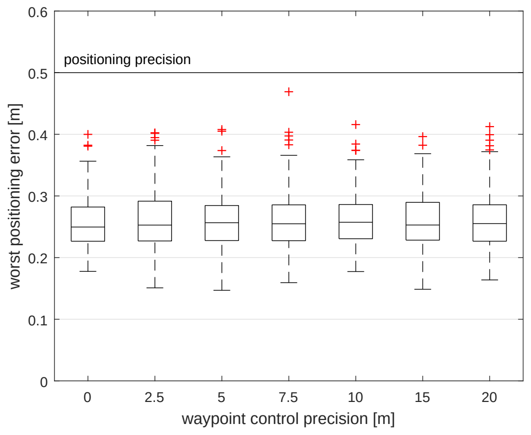


Fig. 22. Distribution of the mission's worst positioning error of PreciseVerifierBee wrt waypoint control precision. Each distribution is represented as a box-and-whisker plot, in which the box represents the median and the first and third quartiles, the whiskers' lengths are 1.5 times the interquartile range, and the crosses represent the outliers.

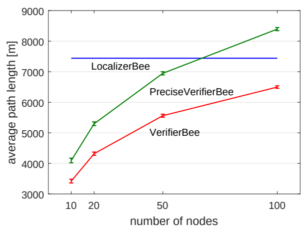


Fig. 23. Average path length wrt number of nodes. For VerifierBee and PreciseVerifierBee, 95%-confidence intervals are displayed in error bars.

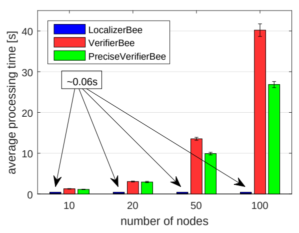


Fig. 24. Average processing time wrt number of nodes. 95%-confidence intervals are displayed in error bars.

## 8.2 Optimality And Efficiency

To evaluate the optimality and the efficiency of the three proposed path planning algorithms, we conducted a simulation campaign as follows. We run VerifierBee and PreciseVerifierBee on a number of prior positions randomly distributed on a deployment area $D = 1000\,\mathrm{m} \times 1000\,\mathrm{m}$. For PreciseVerifierBee, we fixed a positioning precision of $\varepsilon_{\tilde{X}} = 5.0\varepsilon_s$, and the corresponding heuristic values of $d_{min} = 0.721h$ and $\alpha_{min} = 40°$ given by Table 1. We repeated such a simulation for 100 different random sets of prior positions.

Fig. 23 shows the average length of the planned paths with respect to the number of nodes, compared with the length of the path produced by LocalizerBee with the same deployment area. Note that the length of the paths produced by VerifierBee and PreciseVerifierBee grow logarithmically with the number of nodes. VerifierBee produces always the shortest paths, because it enjoys the prior knowledge of the positions and at the same time it does not have to fulfill any precision guarantee. As the number of nodes grows, PreciseVerifierBee becomes significantly worse than VerifierBee, and with 100 nodes it becomes worse even than LocalizerBee. This means that guaranteeing a positioning precision is quite an expensive requirement when the nodes are many. With 100 nodes or more, it could be convenient

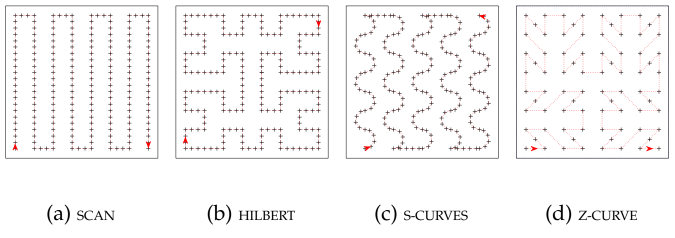


Fig. 25. State-of-the-art paths aimed at localization.

to relax the precision guarantee (i.e., $\varepsilon_{\tilde{X}} > 5.0\varepsilon_s$). The paths produced by the proposed algorithms are feasible by commercial drones with a single battery charge. For example, the Phantom 4 Professional drone is capable of running about $13\,\mathrm{Km}$ in practical tests[4]. Our path planning algorithms produce shorter paths (max $8.4\,\mathrm{Km}$ with PreciseVerifierBee and 100 nodes). This allows us to use cheaper drones or to perform additional tasks in the same flight.

Fig. 24 shows the average processing time of VerifierBee and PreciseVerifierBee, running on a 3.70GHz Intel Core i3-4170 processor. The processing time of LocalizerBee is negligible with respect to the other two algorithms. The slowest algorithm (VerifierBee) takes roughly 40 seconds to run with 100 nodes. This should be fully acceptable for an off-line computation. Implementing the algorithms in the C language (instead of the Matlab one) should improve the processing time even more. VerifierBee takes longer to run with respect to PreciseLocalizerBee. This is because both algorithms follow a local search heuristic, but VerifierBee has to respect less constraints, and thus it can do more improvement steps before stopping on a local minimum. On the other hand, PreciseVerifierBee finds a local minimum earlier, and then stops.

## 8.3 Comparison With The State Of The Art

We compared the paths produced by the proposed algorithms with other state-of-the-art paths aimed at localization with a mobile anchor. In particular, we compared with the SCAN path [12], the HILBERT path [12], the S-CURVES path [8], and the Z-CURVE path [17]. For the SCAN, the HILBERT, and the S-CURVES paths, we re-created the same paths of the original authors (see Figs. 2a and 2c in [12], and Fig. 2d in [8]), scaled for our deployment area size ($1000\,\mathrm{m} \times 1000\,\mathrm{m}$ instead of $480\,\mathrm{m} \times 480\,\mathrm{m}$). We did this by fixing the resolution to $R = (\text{deployment area width})/8 = 125\,\mathrm{m}$ and the waypoint interval to $R/3 = 41.67\,\mathrm{m}$. For the Z-CURVE path, we fixed the resolution to $R = (\text{deployment area width})/8 = 125\,\mathrm{m}$, as well. Fig. 25 shows the resulting paths.

We compared with these paths in terms of coverage, path length, and positioning error. We conducted a simulation campaign as follows. We run VerifierBee and PreciseVerifierBee on a set of 20 prior positions randomly distributed on a deployment area $D = 1000\,\mathrm{m} \times 1000\,\mathrm{m}$. We also run LocalizeBee on the same deployment area. We fixed the home waypoint to be in the south-west corner of

4. http://myfirstdrone.com/phantom-4/dji-phantom-4-real-world-range-test/

the deployment area. For VerifierBee and PreciseVeriferBee we fixed a prior position precision of $\varepsilon_N = 5\,\mathrm{m}$. For PreciseVerifierBee, we fixed a positioning precision of $\varepsilon_{\tilde{X}} = 5.0\varepsilon_s$. The examined state-of-the-art paths do not take into consideration tolerance to control errors, nor drone position measurement errors. Thus, to make a valid comparison, we considered the control precision to be perfect ($\gamma_W = 0\,\mathrm{m}$, $\gamma_h = 0\,\mathrm{m}$), and the measurement precision on the drone position to be perfect ($\varepsilon_W = 0\,\mathrm{m}$, $\varepsilon_h = 0\,\mathrm{m}$). Then, we simulated the verification/localization missions. To include as many nodes as possible in the verifiable triangles, for LocalizerBee and the state-of-the-art paths we assume that the drone measures all the nodes within its communication range at each waypoint. On the other hand, for VerifierBee and PreciseVerifierBee the drone measures each node from exactly three waypoints, determined by the path planning algorithm. We repeated such a simulation for 100 different random sets of prior positions.

Fig. 26 shows the average percentage of covered nodes with respect to the communication range. A node is considered covered if the requirements of Section 4.2 are fulfilled, i.e., if (i) it is within the communication range of at least three waypoints, and (ii) its measured position lies within a verifiable triangle formed by these waypoints. We can see that the state-of-the-art paths do not reach 100% coverage, and this makes them unsuitable for drone-based verifiable multilateration.

Fig. 27 shows the average path length with respect to the communication range. The examined state-of-the-art paths do not depend on the communication range, so their lengths do not vary. We can see that VerifierBee and PreciseVerifierBee produce the shortest paths. This is because they enjoy the prior knowledge of the positions. LocalizerBee produces paths shorter than the state-of-the-art paths for $s_{max} \geq 300m$. This confirms its good optimality.

Fig. 28 shows the average positioning error with respect to the communication range. Note that all the paths reach a good (sub-meter) average positioning error, but this is ascribable not to the paths themselves, but rather to the intrinsic precision of the underlying UWB technology. As expected, VerifierBee is the least precise algorithm, because it produces bad geometry layouts. Its precision is significantly improved by PreciseVerifierBee, with a cost in terms of path length. The proposed algorithms are not as precise as the examined state-of-the-art paths, because they are focused on localizing securely rather than precisely. The superior precision of LocalizerBee and the state-of-the-art paths is mainly because they measure each node from more than three waypoints in the average case. This makes the positioning more precise compared to VerifierBee and PreciseVerifierBee, which measure each node from exactly three waypoints.

## 9 Conclusions

In this paper, we explored the approach of using drones to securely localize a set of devices by means of verifiable multilateration. We proposed three path planning algorithms for secure positioning and secure position verification: LocalizerBee, VerifierBee, and PreciseVerifierBee. We run a thorough experimental evaluation of the proposed algorithms, and we compared them with the literature. The results of our experiments showed that the localization-aimed paths proposed by the literature cannot be used with verifiable multilateration, because they are not able to localize all the devices in a generic deployment area. On the other hand, our proposed algorithms securely localize all the devices even in the presence of drone control errors. Moreover, they produce short path lengths and they run in a reasonable processing time.

## References

[1] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for large traveling salesman problems. *INFORMS J. on Computing*, 15(1):82–92, 2003.

[2] S. Brands and D. Chaum. Distance bounding protocols. In *EUROCRYPT'93*, pages 344–359. Springer, 1993.

[3] A. Compagno, M. Conti, A. A. D'Amico, G. Dini, P. Perazzo, and L. Taponecco. Modeling enlargement attacks against UWB distance bounding protocols. *IEEE Trans. on Information Forensics and Security*, 11(7):1565–1577, Jul 2016.

[4] P. Corke, R. Peterson, and D. Rus. Coordinating aerial robots and sensor networks for localization and navigation. In *Distributed Autonomous Robotic Systems 6*, pages 295–304. Springer, 2007.

[5] DecaWave. ScenSor SWM1000 Module. http://www.decawave.com/products/dwm1000-module.

[6] G. Dini, F. Giurlanda, and P. Perazzo. SecDEv: Secure distance evaluation in wireless networks. In *ICNS'03*, pages 207–212, 2013.

[7] W. Du, J. Deng, Y. S. Han, S. Chen, and P. K. Varshney. A key management scheme for wireless sensor networks using deployment knowledge. In *INFOCOM'04*, volume 1. IEEE, 2004.

[8] R. Huang and G. V. Zaruba. Static path planning for mobile beacons to localize sensor networks. In *PerCom Workshops '07*, pages 323–330, Mar 2007.

[9] IEEE Computer Society. IEEE Std 802.15.4a-2007 (Amendment 1: Add Alternate PHYs), 2007.

[10] J. Jiang, G. Han, C. Zhu, Y. Dong, and N. Zhang. Secure localization in wireless sensor networks: a survey. *J. of Communications*, 6(6):460–470, 2011.

[11] F. Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *J. of Field Robotics*, 29(2):315–378, 2012.

[12] D. Koutsonikolas, S. M. Das, and Y. C. Hu. Path planning of mobile landmarks for localization in wireless sensor networks. *Computer Communications*, 30(13):2577 – 2592, 2007.

[13] P. Perazzo, K. Ariyapala, M. Conti, and G. Dini. The Verifier Bee: A path planner for drone-based secure location verification. In *WoWMoM'15*, pages 1–9, Jun 2015.

[14] P. Perazzo and G. Dini. Secure positioning with non-ideal distance bounding protocols. In *ISCC'15*, pages 907–912, July 2015.

[15] P. Perazzo, L. Taponecco, A. A. D'Amico, and G. Dini. Secure positioning in wireless sensor networks through enlargement miscontrol detection. *ACM Trans. on Sensor Networks (to appear)*, 2016.

[16] M. Poturalski, M. Flury, P. Papadimitrios, J.-P. Hubaux, and J.-Y. Le Boudec. Distance bounding with IEEE 802.15.4a: Attacks and countermeasures. *IEEE Trans. on Wireless Communications*, 2011.

[17] J. Rezazadeh, M. Moradi, A. S. Ismail, and E. Dutkiewicz. Superior path planning mechanism for mobile beacon-assisted localization in wireless sensor networks. *IEEE Sensors J.*, 14(9):3052–3064, 2014.

[18] M. L. Sichitiu and V. Ramadurai. Localization of wireless sensor networks with a mobile beacon. In *MASS'04*, pages 174–183, Oct 2004.

[19] L. Taponecco, P. Perazzo, A. A. D'Amico, and G. Dini. On the feasibility of overshadow enlargement attack on IEEE 802.15.4a distance bounding. *IEEE Communications Letters*, 18(2):257–260, Feb 2014.

[20] S. Čapkun, K. Bonne Rasmussen, M. Cagalj, and M. Srivastava. Secure location verification with hidden and mobile base stations. *IEEE Trans. on Mobile Computing*, 7(4):470–483, 2008.

[21] S. Čapkun and J.-P. Hubaux. Secure positioning in wireless networks. *IEEE J. on Selected Areas in Communications*, 24(2):221–232, 2006.

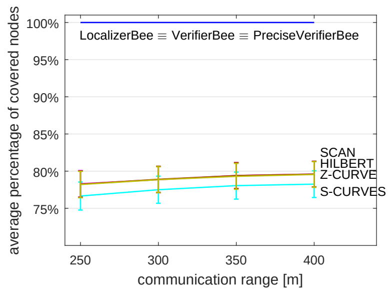


Fig. 26. Average percentage of covered nodes wrt communication range. For SCAN, S-CURVES, HILBERT, and Z-CURVE, 95%-confidence intervals are displayed in error bars.

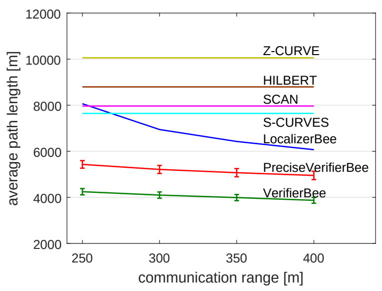


Fig. 27. Average path length wrt communication range. For VerifierBee and PreciseVerifierBee, 95%-confidence intervals are displayed in error bars.

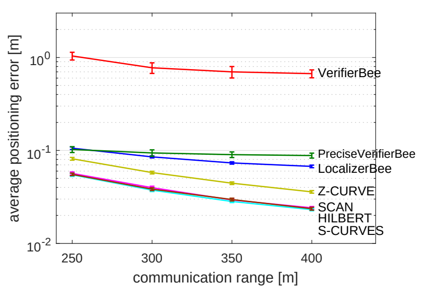


Fig. 28. Average positioning error wrt communication range. 95%-confidence intervals are displayed in error bars.

[22] Y. Zhang, W. Liu, Y. Fang, and D. Wu. Secure localization and authentication in ultra-wideband sensor networks. *IEEE J. on Selected Areas in Communications*, 24(4):829–835, 2006.

**Pericle Perazzo** received the Master degree cum laude in Computer Engineering in 2010 and the Ph.D. degree in Information Engineering in 2014, both from the University of Pisa, Italy. During his Ph.D. studies, he has been Visiting Researcher in the Institute for Parallel and Distributed Systems (IPVS) of Stuttgart, Germany. Since 2014, he has been Research Fellow at the Department of Information Engineering at the University of Pisa. His research interests include the area of security and privacy, with special emphasis on location privacy and secure localization.

**Francesco Betti Sorbelli** is currently pursuing the Ph.D. degree at the University of Florence, Italy. His research interests include wireless sensor networks, communication networks made of directional antennas and the study of aerial devices to support humans in dangerous and difficult environment.

**Mauro Conti** is an Associate Professor at the University of Padua, Italy. He obtained his Ph.D. from Sapienza University of Rome, Italy, in 2009. After his Ph.D., he was a Post-Doc Researcher at Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor the University of Padua, where he became Associate Professor in 2015. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a Fellowship by the German DAAD (2013). His main research interest is in the area of security and privacy. In this area, he published more than 150 papers in topmost international peer-reviewed journals and conferences. He is Associate Editor for several journals, including IEEE Communications Surveys & Tutorials and IEEE Transactions on Information Forensics and Security. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, and General Chair for SecureComm 2012 and ACM SACMAT 2013. He is Senior Member of the IEEE.

**Gianluca Dini** received his Master degree in Electronics Engineering from the University of Pisa, Italy, in 1990, and his Ph.D. in Computer Engineering from Scuola Superiore S. Anna, Pisa, Italy, in 1995. From 1993 to 1994 he was Research Fellow at the Department of Computer Science of the University of Twente, The Netherlands. From 1993 he is with the Department of Information Engineering of the University of Pisa, where he is now Full Professor. His research interests are in the field of distributed computing systems, with particular reference to security. Currently he is working on security in the Internet of Things. He has published more than 100 papers in international conferences, books and journals and has participated, even with coordination roles, to many projects funded by the Commission of the European Community, the Italian Government and private companies.

**Cristina M. Pinotti** received the Master degree cum laude in Computer Science from the University of Pisa, Italy, in 1986. In 1987-1999, she was Researcher with the National Council of Research in Pisa. In 2000-2003, she was Associate Professor at the University of Trento. Since 2004, she is Full Professor at the University of Perugia. Her current research interests include design and analysis of algorithms for wireless sensor networks and communication networks. She has published more than 100 refereed papers on international journals, international conferences, workshops, and book chapters.