Processing Interval Sensor Data in the Presence of Outliers, with Potential Applications to Localizing Underwater Robots

Jan Sliwka and Luc Jaulin
Bureau D214
ENSTA-Bretagne
Brest, France
{luc.jaulin,jan.sliwka}@ensta-bretagne.fr

Martine Ceberio and Vladik Kreinovich
Department of Computer Science
University of Texas at El Paso
El Paso, Texas, USA
{mceberio,vladik}@utep.edu

***Abstract*—Measurements are never absolutely accurate, the measurement result $\widetilde{x}$ is, in general, different from the actual (unknown) values x of the corresponding quantity. In many practical problems, we only know upper bounds Δ on the measurement errors $\Delta x \stackrel{\text{def}}{=} \widetilde{x} - x$. In such situations, once we know the measurement result, the only conclusion that we can make about the actual value x is that this value belongs to the interval $[\widetilde{x} - \Delta, \widetilde{x} + \Delta]$. There exist many efficient algorithms for processing such interval data. However, these algorithms usually assume that all the measurement results are valid. In reality, due to factors such as sensor malfunction, some measurement results may be way off (outliers), for which the difference between $\widetilde{x}$ and x is much larger than the upper bound Δ on the measurement error. In this paper, we overview the algorithmic problems related to processing interval sensor data in the presence of outliers. Our case study – for which we develop and analyze these algorithms – is localization of underwater robots, a problem in which a significant number of measurement results are outliers.**

***Index Terms*—interval uncertainty, outliers, sensor data, underwater robots**

I. Introduction

Processing sensor data: need for interval uncertainty. Sensors are never absolutely accurate. The value $\widetilde{x}$ measured by a sensor is, in general, different from the actual (unknown) value x of the corresponding physical quantity. When processing sensor data, it is important to properly take into account the resulting *measurement errors* $\Delta x \stackrel{\text{def}}{=} \widetilde{x} - x$.

Traditionally, in science and engineering, it is assumed that we know the probability distribution of the measurement error; in many cases, this distribution is assumed to be Gaussian. In such situations, we can use the standard statistical techniques to process this uncertainty.

However, in many real-life situations, we do not know these probabilities. In many such situations, the only information that we have is the upper bound Δ on the measurement error: $|\Delta x| \le \Delta$; see, e.g., [12]. In this case, once we know the measurement result $\widetilde{x}$, the only information that we have about the actual value x is that x belongs to the interval $\mathbf{x} = [\underline{x}, \overline{x}]$, where $\underline{x} = \widetilde{x} - \Delta$ and $\overline{x} = \widetilde{x} + \Delta$.

Need to consider outliers. These exist many efficient techniques for processing such interval data; see, e.g., [2], [10]. These techniques form an important part of *granular computing*; see, e.g., [15].

Most of these algorithms assume that for each measurement, the actual values x is guaranteed to be in the corresponding interval $[\widetilde{x} - \Delta, \widetilde{x} + \Delta]$.

In practice, however, due to a sensor malfunction (or to a human error), we sometimes get erroneous measurement results which are drastically different from the actual values; such results are called *outliers*. It is therefore necessary to take such outliers into account when processing sensor data.

Outliers are usually characterized by a *proportion* ε of measurement results that could be erroneous. For example, $\varepsilon = 0.1$ means that at most 10% of the measurement results are erroneous – and thus, at least 90% of the interval do contain the corresponding actual values.

Sometimes, we are not sure what is the proportion of outliers, so, to be on the safe side, we would like the supply the users with the results corresponding to different values α.

What is known. In some practical situations, there exist efficient algorithms for processing interval sensor data in the presence of outliers. However, in general, the presence of outliers often turns easy-to-solve interval problems into difficult-to-solve (NP-hard) ones.

This is true even in the simplest case, when we simply repeatedly measure several quantities $x_1, \ldots, x_d$. After each measurement, for each measured quantity x_i, we get an interval $[\underline{x}_i^{(j)}, \overline{x}_i^{(j)}]$. So, if this measurement is not an outlier, we can conclude that the actual values of the state $x = (x_1, \ldots, x_d)$ belongs to the d-dimensional box

$$X^{(j)} = [\underline{x}_1^{j}, \overline{x}_1^{(j)}] \times \ldots \times [\underline{x}_d^{(j)}, \overline{x}_d^{(j)}].$$

What can we then conclude about the actual value of x_i?

In the absence of outliers, the answer is simple: since the state x has to belong to all the boxes, it must belong to its intersection, and this intersection can be determined by simply taking, for each dimension i, the intersection of all the corresponding intervals $[\underline{x}_i^{(j)}, \overline{x}_i^{(j)}]$.

It turns out that in the presence of outliers, when, for some k, we know that at least $n - k$ of these boxes contain the

state x, the problem becomes NP-hard: actually, it is NP-hard even to check whether there are $n-k$ boxes with a non-empty intersection [8], [9]. For fixed d, there is a polynomial-time algorithm for solving this problem – but its running time is $O(n^d)$, and grows exponentially with the dimension d.

What we do in this paper. In this paper, we start, in Section II, with an overview of known algorithms for the the corresponding sensor data processing problems. Then we describe new – more adequate and/or more efficient – algorithms, as well as new theoretical results explaining the limitations of the related problems.

II. Case Study

We encountered the need to take outlier into account when we were developing algorithms for localizing underwater robots; see, e.g., [7], [13]. For this localization, we use distance measurements produced by sonars.

A sonar produces not only the distance to the desired object, its signal also contains the echoes from other objects (or walls) along the path. For example, when a robot tries to localize itself by measuring the distance to the nearest wall, the sensor may instead detect a reflection from the nearer surface wave or another object (e.g., a fish or a diver) – and thus, produce a clear outlier. When several sensors perform such measurements – or when we make repeated measurements with the same sensor – we get several values that include both actual measurement results and a significant number of outliers.

III. Range of Possible Value of Each Quantity: How to Compute and Represent It Efficiently

Formulation of the problem. Let us start with the simplest situation in which we have several (n) measurements of the same quantity x. After each measurement j, we get an interval $[\underline{x}^{(j)}, \overline{x}^{(j)}]$.

We know that some of these intervals may be outliers, and we know the upper bound $\varepsilon > 0$ on the proportion of measurements which are outliers. This means that the actual (unknown) value x satisfies at least $n \cdot (1-\varepsilon)$ of n constraints $\underline{x}^{(j)} \le x \le \overline{x}^{(j)}$.

What is known about computing this set. One way to compute this set (see, e.g., [8], [9]) it so sort all the endpoints $\underline{x}^{(j)}$ and $\overline{x}^{(j)}$ into a sequence

$$x_{(1)} \le x_{(2)} \le \ldots \le x_{(2n)},$$

thus dividing the set of possible values of x into $2n-1$ zones $[x_{(1)}, x_{(2)}]$, $[x_{(2)}, x_{(3)}]$, $\ldots$, $[x_{(2n-1)}, x_{(2n)}]$. For each zone, we count the number of constraints which are satisfied for elements of this zone. If this number is greater than or equal to $n \cdot (1-\varepsilon)$, we add this zone to the desired set.

How much time does this algorithm take? Sorting takes time $O(n \cdot \log(n))$ (see, e.g., [1]). Checking all the constraints for all the zones can be done in linear time:

- checking n constraints for the first zone takes time $O(n)$;
- then, when we move from one zone to the zone, at most one constraint changes, so with $2n-1$ zones, we need a total $O(n)$ time.

Thus, totally, we need time

$$O(n \cdot \log(n)) + O(n) + O(n) = O(n \cdot \log(n)).$$

What if we are only interested in the interval of possible values. Instead of deciding which zones belong to the set and which do not, we may only want to know the interval (range) of possible values of the quantity x. If this is the case, we can perform computations faster.

To explain the difference between the set and the interval, let us give a simple example. Suppose that we have two intervals $[-2,-1]$ and $[1,2]$, and we know that at most one of them is an outlier. In this case, it is thus guaranteed that one of the intervals is *not* an outlier an so, contains the actual value x. Therefore, the actual value x is either in the interval $[-2,-1]$ or in the interval $[1,2]$. So, the set of all possible values of x is the set $[-2,-1] \cup [1,2]$.

In this situation, the smallest possible value of x is -2, and the largest possible value of x is 2. Thus, the smallest interval that contains all possible values of x is the interval $[-2,2]$.

How can we compute such an interval?

Computing interval of possible values: analysis of the problem and the resulting algorithm. Let us sort all n upper endpoints $\overline{x}^{(j)}$, $1 \le j \le n$, into an increasing sequence $u_1 \le u_2 \le \ldots \le u_n$. Then, we can guarantee that x is smaller than or equal to at least $n \cdot (1-\varepsilon)$ terms in this sequence. So, we conclude that $x_i \le u_{n \cdot \varepsilon}$.

Similarly, if we sort the lower endpoints $\underline{x}^{(j)}$, $1 \le j \le n$, into a increasing sequence $\ell_1 \le \ell_2 \le \ldots \le \ell_n$, then we conclude that $x \ge \ell_{n \cdot (1-\varepsilon)}$.

Thus, we can conclude that the desired interval of possible values x is equal to $[\ell_{n \cdot (1-\varepsilon)}, u_{n \cdot \varepsilon}]$, where

- $\ell_{n \cdot \varepsilon}$ is the $(n \cdot (1-\varepsilon))$-th element in the set of the lower endpoints $\underline{x}^{(j)}$, and
- $u_{n \cdot \varepsilon}$ is the $(n \cdot \varepsilon)$-th element in the set of the upper endpoints $\overline{x}^{(j)}$.

At first glance, it may sound as if we did not gain anything since we still sort to sort the endpoints, but actually we do not need to do all the sorting: it is sufficient to find elements of given rank, and finding such elements can be done in linear time $O(n)$ (see, e.g., [1]).

Thus, we can indeed compute the *interval* of possible values much faster than the *set* of possible values: namely, in time $O(n) \ll O(n \cdot \ln(n))$.

Application. In the underwater robot localization problem, once we know the upper bound ε on the proportions of outliers, we can thus find an interval which is guaranteed to contain the actual distance to the wall (or to another surface).

Our preliminary experiments confirm that this technique indeed correctly locates the robot [7], [13].

A natural fuzzy representation of the above information. When finding the set of possible values, for each element x, we compute the proportion $\mu(x)$ of the number of constraints which are satisfied to the total number of constraints. This

number from the interval $[0,1]$ describes to what extent this element satisfies the given constraints.

It is therefore reasonable to interpret this number as a *membership degree*, and the function that assigns this degree to each element as a *membership function* in the sense of one more important component of granular computing – fuzzy logic; see, e.g., [3], [11], [18].

If we know the upper bound ε on the proportion of malfunctioning sensors, then for the actual value, at least $(1-\varepsilon)$-th fraction of the constraints must be satisfied. Thus, the actual value x must belong to the set of all the values x for which $\mu(x) \geq 1-\varepsilon$. In fuzzy terms, this set is known as an α-cut of the original fuzzy set corresponding to $\alpha = 1-\varepsilon$. Thus, the set of all possible x is the α-cut.

Computing and storing the whole fuzzy set – as opposed to an appropriate α-cut – makes sense when we are not sure about the actual proportion of outliers, and we would like to supply the user with the results corresponding to different values α.

In particular, for the case when we are interested in the intervals, the corresponding α-cut is the interval $[\ell_{\alpha\cdot n}, u_{(1-\alpha)\cdot n}]$.

Comment. So far, fuzzy sets are just an interpretation, they do not add any new algorithms. However, in the following text, we show that this analogy with fuzzy sets can actually help in computations – by using known algorithms for processing fuzzy data.

IV. Taking Outliers into Account: Beyond Counting

Not all sensors are equally reliable. In the previous text, we simply counted how many constraints are satisfied. This counting kind of implies that all the constraints are equally "soft". In real life, we may have more confidence in some constraints and somewhat less confidence in other constraints.

For example, some sensors may be more reliable and some less reliable. This is often the case, in particular, for sensors used by an underwater robot.

How can we describe this different reliability in precise terms: idea. Let us assume that we have n constraints. To each constraint, we put into correspondence the set X_i of all the elements $x \in X$ that satisfy these constraints. For example, X_i may be the set of all the states which are consistent with the i-th measurement result.

Let p_i be a probability value that describes our certainty that the i-th constraint is satisfied, i.e., that $x \in X_i$. Let us assume that different constraints are independent. In this case, for every element x, we can find the probability that x is consistent with all the constraints: we simply multiply the probabilities p_i corresponding to all the constraints which are satisfied (i.e., for which $x \in X_i$) and the probabilities $1-p_j$ corresponding to all the constraints which are not satisfied (i.e., for which $x \notin X_j$). Thus, we get a value

$$p(x) = \prod_{i:x\in X_i} p_i \cdot \prod_{j:x\notin X_j} (1-p_j).$$

From the above idea to the exact description. The above formulas assigns a non-zero probability to all possible values x. What we want is to decide which values are possible and which are not.

This situation is typical in probability theory. For example, if we know that some quantity is normally distributed, with a known mean a and a known standard deviation σ, then for each possible value x, we get a non-zero probability density of this value x. In practice, we conclude, e.g., that the values outside the "three sigma" interval $[a-3\sigma, a+3\sigma]$ are impossible – because the probability of being outside this interval is very small. If we want to be on the safe side, then instead of a 3σ interval, we can consider a 6σ interval – but no matter what we do, we always set up some threshold, and dismiss all the values for which the probability is smaller than this threshold.

Similarly, in our case, we should select some threshold probability p_0, and conclude that only states x with $p(x) \geq p_0$ are possible.

Towards an effective algorithm. The above expression for $p(x)$ can be represented as

$$p(x) = \prod_{i:x\in X_i} p_i \cdot \frac{\prod\limits_{j=1}^{n}(1-p_j)}{\prod\limits_{i:i\in X_i}(1-p_i)} = \prod_{i:x\in X_i} \frac{p_i}{1-p_i} \cdot C,$$

where $C \stackrel{\text{def}}{=} \prod\limits_{j=1}^{n}(1-p_j)$ does not depend on x at all.

By taking the logarithm of both parts of this inequality, we conclude that $\sum\limits_{i:x\in X_i} w_i \geq t_0$, where $w_i \stackrel{\text{def}}{=} \ln\left(\frac{p_i}{1-p_i}\right)$ and $t_0 \stackrel{\text{def}}{=} \ln(p_0) - \ln(C)$.

Thus, the probabilistic approach is equivalent to assigning *weights* to different constraints. In the degenerate case when all the probabilities are equal $p_1 = p_2 = \ldots = p_n = p$, all the weights are equal $w_1 = \ldots = w_n = w$, and thus, the condition $\sum\limits_{i:x1X_i} w_i \geq t_0$ is equivalent to $\#\{i : x \in X_i\} \geq q \stackrel{\text{def}}{=} \frac{t_0}{w}$, to the above condition that at least q constraints should be satisfied.

The above algorithms can be easily modified to accommodate such weights.

Discussion. The addition to weights provides a more detailed picture of what is going on. For example, if we have two constraints, then the original approach based on the number of satisfied constraints separates:

- the intersection $X_1 \cap X_2$,
- the set $(X_1 - X_2) \cup (X_2 - X_1)$ of all the elements which satisfy only one constraint (here, $X_1 - X_2$ denotes set difference), and
- the elements that do not satisfy any constraint at all.

If different constraints have different weights, then, by considering different thresholds t_0, we also separate the two difference sets $X_1 - X_2$ and $X_2 - X_1$ – because for them, the sum of the weights is different.

Fuzzy interpretation. We would like to extend the above fuzzy interpretation to this more general case. In the more

general case, the degree to which each element x satisfied all the constraints is proportional to the sum of the weights w_i of all the satisfied constraints.

In general, this sum cannot be directly interpreted as the fuzzy degree since it can be larger than one. To make such an interpretation possible, we can normalize these values – by dividing by the largest possible amount $\sum\limits_{i=1}^{n} w_i$, and considering the ratio

$$\mu(x) = \frac{\sum\limits_{i:x\in X_i} w_i}{\sum\limits_{i=1}^{n} w_i}.$$

When all the weights are equal to each other, this ratio becomes equal to the ratio of the number of constraints satisfied by x to the total number of constraints – exactly as in the case when did not assign any weights.

Comment. It is worth mentioning that in this interpretation, the α-cuts coincide with the sets of all x for which $p(x)$ is larger than a certain threshold – i.e., with *confidence sets* corresponding to different confidence levels.

V. From Describing Each Quantity to Joint Processing of These Quantities: A Simple NP-Hardness Result

Formulation of the general problem. In the above text, we mainly concentrated on determining the possible values of different physical quantities $x_1, \ldots, x_d$. However, often, we are interested not only in the values of the directly measurable quantities x_i, but also in the values of other quantities y_j that are related to x_i by a known dependence.

Indeed, many quantities in which we are interested are difficult (or even impossible) to measure directly. For example:

- while it is possible to directly measure a distance between the two places on the same street – e.g., by walking or driving between them – it is not possible to directly measure a distance to a star;
- similarly, while it is possible to directly measure the amount of water in a bottle – e.g., by weighing it – it is not possible to directly measure the amount of oil in an oil well;
- it is not possible to directly measure the 3D spatial coordinates y_j of an underwater robot; however, we can reconstruct these coordinates if we measure the distances x_i from the robot to several objects whose location is known.

In all these cases, we need indirect measurements, i.e., we need to measure the values of some easier-to-measure quantities x_i that are related to the desired quantity y_j by a known dependence, and then reconstruct the value of y_j from the results of measuring x_i.

Outliers make this computational problem computationally difficult. In the previous text, we mentioned that if we have measurement uncertainty *and* outliers, then the problem becomes NP-hard. Let us show that for data processing, the problem is NP-hard even when we can ignore the measurement uncertainty, i.e., when we only have outliers. Moreover, we will show that the problem is NP-hard even in the simplest case of linear dependence.

Simplest case: linear dependence. In general, we may have complex non-linear dependence between the easier-to-measure quantities x_i and the desired quantity (or quantities) y_j. The simplest case is when we have a linear dependence, i.e., when we have constraints

$$\sum_{j=1}^{d} a_{ij} \cdot y_j = x_i, \quad i = 1, \ldots, n$$

for known coefficient a_{ij}.

Linear dependence: case of no outliers. We assume that we know the values x_i, and we want to find the values y_j. In the absence of outliers, to find y_j, we simply solve the corresponding system of linear equations – a task for which feasible algorithms are well-known.

Linear dependence: case of outliers. Let us now assume that we are given a real number $\varepsilon \in (0, 1)$ that describes the upper bound on the percentage of outliers among measurement results. Then, we face the following problem:

- given the values a_{ij}, x_i, and $\varepsilon \in (0, 1)$,
- we need to check whether out of n constraints

$$\sum_{j=1}^{d} a_{ij} \cdot y_j = x_i,$$

we can select a consistent set of $n \cdot (1 - \varepsilon)$ constraints.

We will prove that this problem is NP-hard.

Proof of NP-hardness. A standard way to prove an NP-hardness of a problem is to reduce, to this problem, one of the problems which is already known to be NP-hard; see, e.g., [1]. As such a known NP-hard problem, we take the following *subset sum* problem: given positive integers $s_1, \ldots, s_m$, and s, check whether $s = \sum\limits_{i=1}^{m} \varepsilon_i \cdot s_i = s$ for some $\varepsilon_i \in \{0, 1\}$.

We will reduce each instance of this problem to the following instance of the above problem (reconstructing y_j from x_i in the presence of outliers). In this instance, we have $n = m/\varepsilon$ constraints:

- $2m$ constraints $y_1 = 0$, $y_1 = 1$, …, $y_m = 0$, $y_m = 1$; and
- $n - 2m$ identical constraints $\sum s_i \cdot y_i = s$.

Since $0 \neq 1$, out of each pair of constraints $y_i = 1$ and $y_i = 1$, only one can be satisfied. So, at most $n - m$ constraints can be satisfied.

If the subset sum problem has a solution, then:

- all $n - 2m$ constraints $\sum s_i \cdot y_i = s$ are satisfied, and
- for each i, either $y_i = 0$ constraint or $y_i = 1$ constraint is satisfied,

to the total of $n - m = n \cdot (1 - \varepsilon)$ constraints.

Vice versa, if $n - m$ constraints are satisfied, then at most m constraints must be violated. Thus, for every i, we must

have $y_i = 0$ and $y_i = 1$ and we will also have $\sum s_i \cdot y_i = s$. So, we have a solution to the original subset sum problem. The reduction is proven, so our problem is indeed NP-hard.

VI. Joint Processing of Several Quantities: Feasible Algorithms for the Simple Case

Formulation of the problem. In the previous section, we showed that in general, the problem of joint processing in the presence of outliers is NP-hard. Crudely speaking, NP-hard means that (unless P=NP) we cannot have a single feasible algorithm that solves *all* the particular cases of this problem. Instead, we can look for algorithms that solve *some* cases of this problem.

Let us start with the simplest case, when data processing consists simply of adding two quantities $y = x_1 + x_2$. This simple problem often occurs in practice.

For example, we may have a distance that consists of two parts, we measure these parts separately, and then add the results. Alternatively, we want to find the weight of a system, we weigh each component separately, then add the results.

Two possible cases. When we had only one quantity, we had only one formulation of the problem: namely, we have the upper bound on the number (or, equivalently, on the frequency) of outliers. With two quantities, we have two possibilities.

- The first possibility is that the measurements of x_1 and x_2 are done by two *different* types of sensors.
- The second possibility is when, to measure both x_1 and x_2, we use sensors of *the same* type.

First possibility. The first possibility is that the measurements of x_1 and x_2 are done by two *different* types of sensors. For each type, we have its own upper bound on the frequency of outliers. Based on the bound corresponding to measurements of x_1, we can compute the interval $[\underline{x}_1, \overline{x}_1]$ of possible values of x_1. Similarly, Based on the bound corresponding to measurements of x_2, we can compute the interval $[\underline{x}_2, \overline{x}_2]$ of possible values of x_2. Then, the interval of possible values of y has the form

$$[\underline{y}, \overline{y}] = [\underline{x}_1 + \underline{x}_2, \overline{x}_1 + \overline{x}_2].$$

In this case, we can handle more complex cases as well, when we have several variables and/or a more complex relation $f(x_1, \ldots, x_d)$ between y and x_i. To find the corresponding range

$$[\underline{y}, \overline{y}] = \{f(x_1, \ldots, x_n) : x_1 \in [\underline{x}_1, \overline{x}_1], \ldots, x_d \in [\underline{x}_d, \overline{x}_d]\},$$

we can use techniques for estimating such ranges developed in interval computations [2], [10].

Second possibility: analysis. The second possibility is when, to measure both x_1 and x_2, we use sensors of *the same* type. In this case, the limitation on the proportion of outliers refers to *both* measurements. Let us describe this situation in more detail.

Let us assume that we have n_1 measurements of x_1 and n_2 measurements to x_2. As a result of processing all the x_1-measurements, we find, for every $\alpha \in (0, 1)$, the α-cut $\mathbf{x}_1(\alpha) = [\underline{x}_1(\alpha), \overline{x}_1(\alpha)]$, i.e., the set of all the values of x_1 which are possible if at least α-th part of the sensors work well. Similarly, as a result of processing all the x_2-measurements, we find, for every $\alpha \in (0, 1)$, the α-cut $\mathbf{x}_2(\alpha) = [\underline{x}_2(\alpha), \overline{x}_2(\alpha)]$, i.e., the set of all the values of x_2 which are possible if at least α-th part of the sensors work well.

We do not know which portion of x_1-sensors works well and which portion of the x_2-sensors work well, all we know is that overall, the number of sensors that work well is at least $\alpha = 1 - \varepsilon$, for some given ε. So, if α_1 is the proportion of x_1-sensors that work well and α_2 is the proportion of x_2-sensors that work well, we have

$$\alpha_1 \cdot \frac{n_1}{n_1 + n_2} + \alpha_2 \cdot \frac{n_2}{n_1 + n_2} = \alpha.$$

For each such pair α_1 and α_2, we have the lower bound $\underline{x}_1(\alpha_1)$ for x_1 and the bound $\underline{x}_2(\alpha_2)$ for x_2; in this case, the sum $y = x_1 + x_2$ is bounded from below by the sum $\underline{x}_1(\alpha_1) + \underline{x}_2(\alpha_2)$. We do not know the values α_i, we only know that y is larger than one of these sums – the sum corresponding to the actual (unknown) value α_1. Thus, we can conclude that y is larger than the smallest of such sums:

$$\underline{y}(\alpha) = \min_{\alpha_1 \cdot \frac{n_1}{n_1+n_2} + \alpha_2 \cdot \frac{n_2}{n_1+n_2} = \alpha} (\underline{x}_1(\alpha_1) + \underline{x}_2(\alpha_2)).$$

Similarly, we can conclude that y is smaller than or equal to the largest of the corresponding sums of upper bounds:

$$\overline{y}(\alpha) = \max_{\alpha_1 \cdot \frac{n_1}{n_1+n_2} + \alpha_2 \cdot \frac{n_2}{n_1+n_2} = \alpha} (\overline{x}_1(\alpha_1) + \overline{x}_2(\alpha_2)).$$

How can we compute these sums?

Simplifications. An upper bound for $y = x_1 + x_2$ is minus the lower bound for $-y = (-x_1) + (-x_2)$. So, without losing generality, we can concentrate on the lower bounds.

The above formula for the lower bound can be somewhat simplified if we introduce two auxiliary variables

$$t_1 \stackrel{\text{def}}{=} \alpha_1 \cdot \frac{n_1}{n_1 + n_2} \text{ and } t_2 \stackrel{\text{def}}{=} \alpha_2 \cdot \frac{n_2}{n_1 + n_2}.$$

In terms of these variables, the above formula has the form

$$\underline{y}(\alpha) = \min_{t_1, t_2 : t_1 + t_2 = \alpha} (f_1(t_1) + f_2(t_2)), \tag{1}$$

where we denoted $f_i(t_i) \stackrel{\text{def}}{=} \underline{x}_i\left(t_i \cdot \frac{n_1 + n_2}{n_i}\right)$.

What we do in this section. Following [17], we prove that the existing algorithms for processing fuzzy numbers can be used to compute this expression.

Zadeh's extension principle: reminder. Specifically, we will use fast algorithms for computing the result of processing fuzzy data. This result is described by the following Zadeh's extension principle: once we know the membership functions $\mu_1(t_1), \ldots, \mu_n(t_n)$ corresponding to n variables $t_1, \ldots, t_n$, the membership function $\mu(t)$ corresponding to $t = f(t_1, \ldots, t_n)$ takes the form

$$\mu(t) = \max_{t_1, \ldots, t_n :\ f(t_1, \ldots, t_n) = t} f_{\&}(\mu_1(t_1), \ldots, \mu_n(t_n)). \tag{2}$$

The most most widely used fuzzy "and"-operations are the minimum $f_{\&}(a,b) = \min(a,b)$ and the algebraic product $f_{\&}(a,b) = a \cdot b$. Thus, we arrive at the following formulas:

$$\mu(t) = \max_{t_1,\ldots,t_n:\ f(t_1,\ldots,t_n)=t} \min(\mu_1(t_1),\ldots,\mu_n(t_n)), \quad (3)$$

which is the most widely used form of Zadeh's extension principle, and

$$\mu(t) = \max_{t_1,\ldots,t_n:\ f(t_1,\ldots,t_n)=t} \mu_1(t_1)\cdot\ldots\cdot\mu_n(t_n). \quad (4)$$

In particular, for the simplest case of the addition function $f(t_1+t_2)=t_1+t_2$, the above formula takes the form

$$\mu(t) = \max_{t_1,t_2:\ t_1+t_2=t} \mu_1(t_1)\cdot\mu_2(t_2). \quad (5)$$

Straightforward computation of the expression (5). In reality, we can only know the values of $\mu_1(x)$ and $\mu_2(x)$ for finitely many values x. Let us denote the total number of such values by n. In this case, it is reasonable to compute only n values of $\mu(x)$. For each of these n values, according to the formula (5), we must find the largest of n products. Computing each product takes 1 elementary computational step, computing the largest of n numbers requires that we do $n-1$ comparisons. So, the total number of computation steps that needs to be done to compute one value of $\mu(x)$ is $2n-1=O(n)$. Thus, to compute *all* n values of the desired membership function $\mu(x)$, we need $n\cdot O(n) = O(n^2)$ computational steps.

For large n, this number if large, so it is desirable to have faster algorithms for computing this expression.

A faster algorithm for computing the expression (5): main idea. Such faster algorithms are known. For example, an algorithm described in [5], [6] is based on the well-known fact that for non-negative numbers $\mu_1,\ldots,\mu_n$, we have

$$\max(\mu_1,\ldots,\mu_n) = \lim_{p\to\infty} (|\mu_1|^p + \ldots + |\mu_n|^p)^{1/p}$$

(see, e.g., [4]). Therefore, for sufficiently large p, we have

$$\max(\mu_1,\ldots,\mu_n) \approx (|\mu_1|^p + \ldots + |\mu_n|^p)^{1/p};$$

the larger p, the better the quality of this approximation.

Applying this approximate formula to the values

$$\mu_1(t_1)\cdot\mu_2(t-t_1)$$

maximized in the formula (5), we come up with an approximate formula $\mu(t) \approx M(t)^{1/p}$, where we denoted

$$M(t) = \sum_{t_1} (\mu_1(t_1)\cdot\mu_2(t-t_1))^p.$$

The formula for $M(y)$ can be rewritten as:

$$M(t) = \sum_{t_1} (\mu_1(t_1))^p \cdot \mu_2(t-t_1))^p.$$

In the natural assumption that the values x_1 are equally spaced, with step h, this sum becomes a *convolution* of two functions: $M_1(x) = (\mu_1(x))^p$ and $M_2(x) = (\mu_2(x))^p$. Now, we can use the following two ideas to compute $M(x)$ fast:

- It is known that the Fourier transform of the convolution $M_1 * M_2$ of two functions M_1 and M_2 is equal to the product of their Fourier transforms.
- Fourier transform can be computed in time $O(n\log(n))$ [14], [16]; the corresponding algorithms are called *Fast Fourier Transform* (FFT, for short).

In view of these two facts, we can use the following algorithm to compute the membership function that expresses the sum of two given fuzzy numbers:

Given: the values $\mu_1(t_1)$ and $\mu_2(t_2)$ for n equally spaced values t_1 and t_2.

Algorithm: First, we pick a large number p (the larger p, the better the results of our computations). Then, we do the following:

1) For each of n values x_1, we compute the values $M_1(x) = (\mu_1(x))^p$ and $M_2(x) = (\mu_2(x))^p$.
2) We apply FFT to the functions $M_1(x)$ and $M_2(x)$ and get their Fourier transforms $\hat{M}_1(\omega)$ and $\hat{M}_2(\omega)$ (for n different values ω).
3) We multiply $\hat{M}_1(\omega)$ and $\hat{M}_2(\omega)$; let us denote the corresponding product by $\hat{M}(\omega)$.
4) We apply inverse Fast Fourier transform to the product $\hat{M}(\omega)$ (computed on the previous step). As a result, we get a function $M(t)$.
5) Finally, we reconstruct $\mu(t)$ as $(M(t))^{1/p}$.

Number of computational steps. Let us estimate the number of computational steps of this algorithm. Stages 1, 3, and 5 require linear time ($O(n)$ steps each, so, $O(n)$ total). Stages 2 and 4 involve FFT and therefore, require the time $O(n\log(n))$. Therefore, the total number of computational steps is equal to $O(n) + O(n\log(n)) = O(n\log(n))$, which is much smaller than the $O(n^2)$ time that is needed for straightforward computations.

Comment. A similar algorithm can be applied for computing the sum of more than two fuzzy numbers. Alternatively, we can first use the above algorithm to add the first two of these fuzzy numbers, then add the third one to the result, etc.

What we want. We know that for the problem of computing expression (5), there is an efficient algorithm which is faster than a straightforward $O(n^2)$ algorithm. We would like to use to use this algorithm to come up with a similar faster algorithm for computing the desired expression (1).

Analysis of the problem. The main difference between the desired formula (1) and the formula (5) that describes Zadeh's extension principle is that:

- the desired formula (1) uses addition, while
- the formula (5) corresponding to Zadeh's extension principle use multiplication.

Another difference is that:

- the desired formula (1) uses minimum, while

- the formula (5) corresponding to Zadeh's extension principle use maximum.

Thus, to reduce our problem to the problem of computing Zadeh's extension principle, we must reduce addition to multiplication, and minimum to maximum.

How to reduce addition to multiplication: reminder. It is well known how to reduce addition to multiplication: use an exponential function $\exp(k \cdot x)$ since

$$\exp(k \cdot (a+b)) = \exp(k \cdot a) \cdot \exp(k \cdot b).$$

We want the resulting value $\exp(k \cdot x)$ to be from the interval $[0,1]$ for all $x > 0$. Thus, we must select $k < 0$ – otherwise, we will get values $\exp(k \cdot x) > 1$. The simplest such value is $k = -1$.

The function $\exp(-x)$ is decreasing, so it automatically reduced minimum to maximum.

Resulting reduction: idea. To compute the value (1), we consider the functions $\mu_1(t_1) = \exp(-f_1(t_1))$, $\mu_2(t_2) = \exp(-f_2(t_2))$, and $\mu(t) = \exp(-\underline{y}(t))$.

By definition (1), $\underline{y}(t)$ is the smallest of possible values $f_1(t_1) + f_2(t - t_1)$ corresponding to all possible t_1. Since the function $\exp(-x)$ is decreasing, its values at the smallest of the arguments is the largest, i.e.,

$$\mu(t) = \exp(-\underline{y}(t)) = \max_{t_1} \exp(-(f_1(t_1) + f_2(t - t_1))).$$

Here,

$$\exp(-(f_1(t_1) + f_2(t - t_1))) =$$

$$\exp(-(f_1(t_1)) \cdot \exp(-f_2(t - t_1)) = \mu_1(t_1) \cdot \mu_2(t - t_1),$$

hence

$$\mu(t) = \exp(-\underline{y}(t)) = \max_{t_1} \mu_1(t_1) \cdot \mu_2(t - t_1).$$

This is exactly the formula (5).

Once we know $\mu(t) = \exp(-\underline{y}(t))$, we can reconstruct $\underline{y}(t)$ as $\underline{y}(t) = -\ln(\mu(t))$.

Thus, we arrive at the following algorithm.

New algorithm for computing the expression (1). Once we know the values $\underline{x}_1(\alpha)$ and $\underline{x}_2(\alpha)$, to compute a similar characteristic for $y = x_1 + x_2$, we do the following:

- form functions $f_1(t_1) = \underline{x}_1\left(t_1 \cdot \dfrac{n_1 + n_2}{n_1}\right)$ and $f_2(t_2) = \underline{x}_2\left(t_2 \cdot \dfrac{n_1 + n_2}{n_2}\right)$;
- form functions $\mu_1(t_1) = \exp(-f_1(t_1))$ and $\mu_2(t_2) = \exp(-f_2(t_2))$;
- apply a fast algorithm for computing the fuzzy expression (5) to these functions $\mu_1(t_1)$ and $\mu_2(t_2)$, and thus compute a new function $\mu(t)$;
- compute $\underline{y}(t) = -\ln(\mu(t))$.

VII. From Addition to a More General Case

Formulation of the problem. In the above text, we only considered the simplest case of data processing, when we have only two inputs x_1 and x_2 and we compute $y = x_1 + x_2$. In the general case, we may have several inputs $x_1, \ldots, x_d$, and we compute a more general expression $y = f(x_1, \ldots, x_d)$.

Let us check what will happen if all these measurements are performed by sensors of the same type, with an overall bound $\varepsilon = 1 - \alpha$ on the proportion of outliers. We will assume that we made n_1 measurements of the quantity x_1, n_2 measurements of the quantity x_2, …, and n_d measurements of the quantity x_d, to the total of $n = n_1 + \ldots + n_d$ measurements. Then, if for every i, we denote by α_i the proposition of sensors that functioned well when measuring x_i, we conclude that $t_1 + \ldots + t_d = \alpha$, where $t_i \stackrel{\text{def}}{=} \alpha_i \cdot \dfrac{n_i}{n}$.

Analysis of the problem. Measurements are reasonably accurate. So, for every i, we can take some "mean" value $\widetilde{x}_i$ as the measurement result. When x_i is in the interval $[\underline{x}_i(\alpha_i), \overline{x}_i(\alpha_i)]$, the corresponding measurement error $\Delta x_i \stackrel{\text{def}}{=} \widetilde{x}_i - x_i$ is in the interval $[\Delta_i^-(\alpha_i), \Delta_i^+(\alpha_i)]$, where $\Delta_i^-(\alpha_i) \stackrel{\text{def}}{=} \widetilde{x}_i - \overline{x}_i(\alpha_i)$ and $\Delta_i^+(\alpha_i) \stackrel{\text{def}}{=} \widetilde{x}_i - \overline{x}_i(\alpha_i)$.

Once we apply the algorithm f to the measurement results $\widetilde{x}_i$, we get an estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_d)$ for the desired quantity y.

The corresponding estimation error is equal to the difference

$$\Delta y = \widetilde{y} - y = f(\widetilde{x}_1, \ldots, \widetilde{x}_d) - f(x_1, \ldots, x_d)$$

between this approximation and the actual (desired) value y. Here, by definition of the measurement errors, we have $x_i = \widetilde{x}_i - \Delta x_i$, so

$$\Delta y = f(\widetilde{x}_1, \ldots, \widetilde{x}_d) - f(\widetilde{x}_1 - \Delta x_1, \ldots, \widetilde{x}_d - \Delta x_d).$$

Since the measurements are reasonably accurate, we can expand this expression in Taylor series in terms of Δx_i and safely ignore terms which are quadratic and of higher order in terms of Δx_i. As a result, we get the following expression:

$$\Delta y = \sum_{i=1}^{d} c_i \cdot \Delta x_i,$$

where

$$c_i \stackrel{\text{def}}{=} \frac{\partial f}{\partial x_i}(\widetilde{x}_1, \ldots, \widetilde{x}_d).$$

Let $s_i \in \{-, +\}$ denote the sign of the derivative c_i. For each combination of values α_i, the smallest possible value of $y = \widetilde{y} - \Delta y$ is attained when the value of the sum Δy is the largest, i.e., when:

- for those i for which $s_i = +$, when the value Δx_i is the largest possible, i.e., equal to $\Delta x_i = \Delta_i^+(\alpha_i)$;
- for those i for which $s_i = -$, when the value Δx_i is the smallest possible, i.e., equal to $\Delta x_i = \Delta_i^-(\alpha_i)$.

Thus, the smallest possible value of y is equal to $\widetilde{y} + \Delta^-$, where

$$\Delta^- = \sum_{i=1}^{d} |c_i| \cdot (-\Delta_i^{s_i}(\alpha_i)).$$

So, in general, the smallest possible value y is attained when Δ^- takes the smallest possible value. Hence, we arrive at the following expression:

$$\underline{y}(\alpha) = \widetilde{y} + \min_{t_1,\ldots,t_d: t_1+\ldots+t_d=\alpha} \sum_{i=1}^{d} |c_i| \cdot (-\Delta_i^{s_i}(\alpha_i)).$$

This expression can be rewritten as $\underline{y}(\alpha) = \widetilde{y} + \Delta^-(\alpha)$, where

$$\Delta^-(\alpha) = \min_{t_1,\ldots,t_d: t_1+\ldots+t_d=\alpha} \sum_{i=1}^{d} f_i(t_i),$$

and we denoted

$$f_i(t_i) \stackrel{\text{def}}{=} -|c_i| \cdot \Delta_i^{s_i}\left(t_i \cdot \frac{n}{n_i}\right).$$

Reduction to fuzzy computations: idea. The above formula can be similarly reduced to computing the fuzzy expression

$$\mu(t) = \max_{t_1,\ldots,t_m:\ t_1+\ldots+t_m=t} \ \prod_{i=1}^{m} \mu_i(t_i), \tag{6}$$

if we take $\mu(t) = \exp(-\Delta^-(t))$ and

$$\mu_i(t_i) = \exp(-f_i(t_i)).$$

Thus, we arrive at the following algorithm.

New algorithm for computing $\underline{y}(\alpha)$. We know, for every input $i = 1, \ldots, d$, the values $\Delta_i^{\pm}(\alpha_i)$ corresponding to different α_i. We want to compute, for each α, the range $[\underline{y}(\alpha), \overline{y}(\alpha)]$. For this computation, we do the following:

- for each i, we select a "mean" value $\widetilde{x}_i$;
- compute the estimate $\widetilde{y} = f(\widetilde{x}_1, \ldots, \widetilde{x}_d)$;
- compute values $c_i = \dfrac{\partial f}{\partial x_i}(\widetilde{x}_1, \ldots, \widetilde{x}_d)$ and their signs s_i;
- form functions

$$f_i(t_i) \stackrel{\text{def}}{=} -|c_i| \cdot \Delta_i^{s_i}\left(t_i \cdot \frac{n}{n_i}\right)$$

  and $\mu_i(t_i) = \exp(-f_i(t_i))$;
- apply a fast $O(n \cdot log(n))$ algorithm for computing the fuzzy expression (6) to these functions $\mu_i(t_i)$, and thus compute a new function $\mu(t)$;
- compute $\Delta^-(t) = -\ln(\mu(t))$ and $\underline{y}(\alpha) = \widetilde{y} + \Delta^-(\alpha)$.

Comment. A similar algorithm can be used to compute the upper bounds $\overline{y}(\alpha)$ corresponding to different values α.

Results and future plans. Our computational experiments on a simulated robot localization problem show that the resulting algorithms indeed lead to a faster robot localization.

After further checking, we plan to test these algorithms on the real robots.

ACKNOWLEDGMENT

This work was partly supported by a MRIS (Office of Advanced Research and Innovation) grant from the Diréction Generale de l'Armement (DGA), France, by the US National Science Foundation grants HRD-0734825 and DUE-0926721, and by Grant 1 T36 GM078000-01 from the US National Institutes of Health. Jan Sliwka was also supported by Brest Métropole Océane (BMO).

The authors are very thankful to the anonymous referees for valuable suggestions.

REFERENCES

[1] C. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, MIT Press, Boston, Massachusetts, 2009.

[2] L. Jaulin, M. Kieffer, O. Didrit, and E. Walter, *Applied Interval Analysis, with Examples in Parameter and State Estimation, Robust Control and Robotics*, Springer-Verlag, London, 2001.

[3] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*, Prentice-Hall, Upper Saddle River, New Jersey, 1995.

[4] A. N. Kolmogorov and S. V. Fomin, *Introductory Real Analysis*, Dover, N.Y., 1975.

[5] O. Kosheleva, S. D. Cabrera, G. A. Gibson, and M. Koshelev, Fast Implementations of Fuzzy Arithmetic Operations Using Fast Fourier Transform (FFT), *Proceedings of the 1996 IEEE International Conference on Fuzzy Systems*, New Orleans, September 8–11, 1996, Vol. 3, pp. 1958–1964.

[6] O. Kosheleva, S. D. Cabrera, G. A. Gibson, and M. Koshelev, "Fast Implementations of Fuzzy Arithmetic Operations Using Fast Fourier Transform (FFT)", *Fuzzy Sets and Systems*, 1997, Vol. 91, No. 2, pp. 269–277.

[7] F. Le Bars, A. Bertholom, J. Sliwka, and L. Jaulin, "Interval slam for underswater robots – a new experiment", *Proceedings of the 8th IFAC Symposium on Nonlinear Control Systems NOLCOS'2010*, Bologna, Italy, September 1–3, 2010.

[8] L. Longpré and C. Servin, "Quantum computations techniques for gauging reliability of interval and fuzzy data", *Proceedings of the 27th International Conference of the North American Fuzzy Information Processing Society NAFIPS'2008*, New York, New York, May 19–22, 2008.

[9] L. Longpré, C. Servin, and V. Kreinovich, "Quantum Computation techniques for gauging reliability of interval and fuzzy data", *International Journal of General Systems*, 2011, Vol. 40, No. 1, pp. 99–109.

[10] R. E. Moore, R. B. Kearfott, and M. J. Cloud, *Introduction to Interval Analysis*, SIAM Press, Philadelphia, Pennsylvania, 2009.

[11] H. T. Nguyen and E. A. Walker, *A First Course in Fuzzy Logic*, Boca Raton, Florida: Chapman & Hall/CRC, 2006.

[12] S. Rabinovich, *Measurement Errors and Uncertainties: Theory and Practice*, American Institute of Physics, New York, 2005.

[13] J. Sliwka, F. Le Bars, O. Reynet, and L. Jaulin, "Using interval methods in the context of robust localization of underwater robots", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'11*, El Paso, Texas, March 18–20, 2011.

[14] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice Hall, Englewood Cliffs, New Jersey, 2009.

[15] W. Pedrycz, A. Skowron, and V. Kreinovich (eds.), *Handbook on Granular Computing*, Wiley, Chichester, UK, 2008.

[16] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, SIAM, Philadelphia, 1992.

[17] K. Villaverde, O. Kosheleva, and M. Ceberio, "Computations under time constraints: algorithms developed for fuzzy computations can help", *Proceedings of the 30th Annual Conference of the North American Fuzzy Information Processing Society NAFIPS'2011*, El Paso, Texas, March 18–20, 2011.

[18] L. A. Zadeh, "Fuzzy sets", *Information and control*, vol. 8, pp. 338–353, 1965.