Odometry under Interval Uncertainty: Towards Optimal Algorithms with Potential Application to Self-Driving Cars and Mobile Robots*

Raphael Voges, Bernardo Wagner
The Real Time Systems Group (RTS)
Institute of Systems Engineering (ISE)
Leibniz University of Hannover
30167 Hannover, Germany
voges@rts.uni-hannover.de,wagner@rts.uni-hannover.de

Vladik Kreinovich†
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
vladik@utep.edu

Abstract

In many practical applications ranging from self-driving cars to industrial application of mobile robots, it is important to take interval uncertainty into account when performing odometry, i.e., when estimating how our position and orientation ("pose") changes over time. In particular, one of the important aspects of this problem is detecting mismatches (outliers). In this paper, we describe an algorithm to compute the rigid body transformation, including a provably optimal sub-algorithm for detecting mismatches.

Keywords: interval computations, interval uncertainty, odometry, incremental localization, rigid body transformation, outliers, robotics
AMS subject classifications: 65G30, 65G40, 65D19, 68T40, 70E60, 86A30

*Submitted: July xxx, 2019;Revised: ; Accepted: .
†This work was supported by the German Research Foundation (DFG) as part of the Research Training Group i.c.sens (grant RTG 2159), and by the Leibniz University of Hannover. It was also supported in part by the US National Science Foundation grants 1623190 (A Model of Change for Preparing a New Generation for Professional Practice in Computer Science) and HRD-1242122 (Cyber-ShARE Center of Excellence). This paper was written when V. Kreinovich was visiting Leibniz University of Hannover.

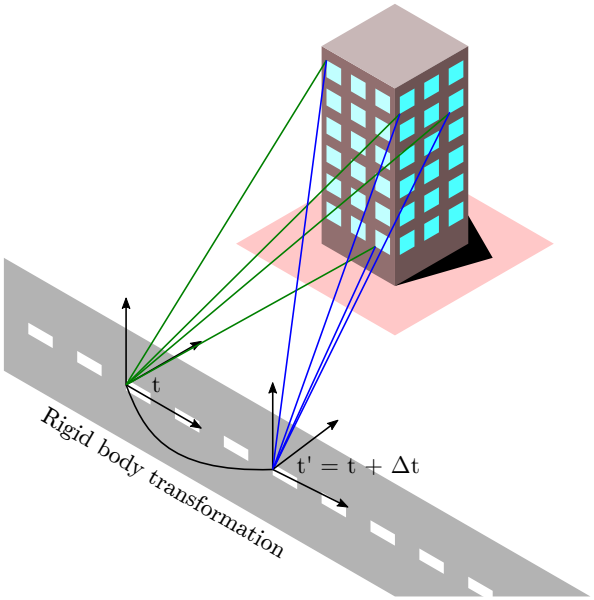

Figure 1: General odometry idea: corresponding 3D keypoints on rigid objects are employed to find the rigid body transformation between two consecutive points in time.

1 Formulation of the Problem

Need for odometry. For a self-driving car to be successful, it needs to correctly keep track of its position and orientation ("pose"). Often, the Global Positioning System (GPS) can be used to compute the pose. However, in case of GPS outages due to, for example, high buildings, the self-motion has to be computed step by step starting at the last known position. The corresponding estimations are known as *odometry*; see, e.g., [1, 4, 11, 16]. Odometry is often performed by tracking the relative position of rigid objects, i.e., objects that do not move and do not change. Such tracking can be useful also for human drivers and for mobile robots especially in harsh environments (e.g. after a natural disaster); see, e.g., [6].

How odometry is performed now. At present, odometry is based on visual images (often, stereo images) and on the use of laser scanners and similar distance-measuring devices. Based on the images obtained at each moment of time t, appropriate image-processing software locates an object and finds keypoints of this object; see, e.g., [7]. The location of these keypoints at moment t is then characterized by spatial coordinates in a special frame coordinate system associated, e.g., with the current location of the self-driving car.

The same procedure is repeated at the next moment of time $t' = t + \Delta t$, and the differences between the coordinates of the corresponding points are used to find the rotation and shift relating the frame coordinate systems corresponding to moments t and t'. Figure 1 visualizes the general idea of odometry estimation from 3D keypoints.

Need for interval uncertainty. Measurements are never absolutely precise; in general, measurement results somewhat differ from the actual (unknown) values of the corresponding physical quantities. In particular, the spatial coordinates are known with some uncertainty. Besides, temporal parameters such as an imperfect sensor clock synchronization can also lead to such uncertainties of the spatial coordinates [12, 13, 14]. In many such situations, we do not know the exact probabilities of different values of estimation errors, all we know is the upper bounds on the corresponding estimation errors. In this case, if the estimated value of a coordinate is $\widetilde{x}$ and the upper bound on the estimation error is Δ, then the set of possible values of x is an interval $\mathbf{x} = [\underline{x}, \overline{x}] = [\widetilde{x} - \Delta, \widetilde{x} + \Delta]$. Because of this fact, this type of uncertainty is known as *interval uncertainty*.

It is desirable to take this uncertainty into account when processing our data. Because of this uncertainty, the results of data processing are also only known with uncertainty. For example, we need to present not only numerical estimates for the rotation and shift between the two frames, we also need to describe how accurate these estimates are.

Need to detect outliers. Algorithms producing keypoints are not perfect. In about 5% of the cases, they mismatch the corresponding point, placing it at a different location; see, e.g., [7, 11]. This can happen, e.g., when a moving object (e.g., a car) appears in front of the original rigid-body building. The system that looks for keypoints, e.g., as points with the largest gradient may pick up points from the moving object at the moment when the moving body is in front of the building. This temporary keypoint does not correspond to any keypoint in the original image. So, from the viewpoint of finding a proper rigid-body transformation, this keypoint is an outlier.

A similar problem occurs when the same moving body is present in both frames. In this case, the keypoints corresponding to the building are obtained from each other by an appropriate rotation and shift, but the keypoints corresponding to the moving body also move, so from the viewpoint of finding the rotation and shift, these keypoints are a mismatch.

To properly process data, we need to detect such outliers.

What is known and what we do in this paper. There exist algorithms for detecting outliers under interval uncertainty. The challenge is that this needs to be done fast, before the next imaging cycle. It is therefore important to design fastest possible algorithms for solving this problem. This is what we do in this paper.

We also provide an algorithm for finding the corresponding rigid-body transformation between the frames corresponding to two consequent moments of time.

2 Towards Fast Algorithms for Detecting Outliers

Main idea. For a rigid body, the distance between the two points does not change. Thus, for every two points i and j, the distances measured at time t and at time $t' = t + \Delta t$ must coincide. Let us denote the coordinates at time t by x_i, y_i, and z_i, the coordinates at time t' by x'_i, y'_i, and z'_i, and the corresponding distances by d_{ij} and d'_{ij}. Then, we should have

$$d_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 =$$

$$(x'_i - x'_j)^2 + (y'_i - y'_j)^2 + (z'_i - z'_j)^2 = (d'_{ij})^2.$$

In practice, we do not have the exact values of the spatial coordinates, we only have intervals of possible values. Based on these intervals, for each i and j, we can only compute intervals of possible values of d_{ij}^2 and $(d'_{ij})^2$. The actual value of the square of the distance must be in both intervals, thus, these intervals must have a non-zero intersection.

If at least one of the points is mismatched, the distances become different, and there is no longer any intersection; see, e.g., [4]. From the purely mathematical viewpoint, it is possible that two randomly selected narrow intervals happen to intersect, but the probability of such a non-empty intersection is very small. So, from the practical viewpoint, if two intervals intersect, we conclude that both points are not outliers. In other words:

- if the two intervals intersect, both points i and j are not outliers, but
- if the two intervals have an empty intersection, at least of the two points i and j is an outlier.

Let us show how this idea can be transformed into an algorithm.

How to compute the intervals, given i and j. The intervals for distances can be easily (and fast) computed. Namely, the expression for d_{ij}^2 is a single-use expression (see, e.g., [2, 3, 5, 8, 9, 15]) and thus, this interval can be computed exactly by using straightforward interval computations, i.e., by replacing each operation with numbers with the corresponding operation of interval arithmetic:

$$\left[\underline{d_{ij}^2}, \overline{d_{ij}^2}\right] = (\mathbf{x}_i - \mathbf{x}_j)^2 + (\mathbf{y}_i - \mathbf{y}_j)^2 + (\mathbf{z}_i - \mathbf{z}_j)^2,$$

where, as usual,

$$[\underline{a}, \overline{a}] - [\underline{b}, \overline{b}] = [\underline{a} - \overline{b}, \overline{a} - \underline{b}],$$

$$[\underline{a}, \overline{a}]^2 = \left[0, \max\left(\underline{a}^2, (\overline{a})^2\right)\right] \text{ if } \underline{a} < 0 < \overline{a},$$

$$[\underline{a}, \overline{a}]^2 = \left[\min\left(\underline{a}^2, (\overline{a})^2\right), \max\left(\underline{a}^2, (\overline{a})^2\right)\right] \text{ otherwise,}$$

and

$$[\underline{a}, \overline{a}] + [\underline{b}, \overline{b}] = [\underline{a} + \underline{b}, \overline{a} + \overline{b}].$$

Similarly,

$$\left[\underline{d'_{ij}}^2, \overline{d'_{ij}}^2\right] = (\mathbf{x}'_i - \mathbf{x}'_j)^2 + (\mathbf{y}'_i - \mathbf{y}'_j)^2 + (\mathbf{z}'_i - \mathbf{z}'_j)^2.$$

How to check whether the two given intervals intersect. This part is easy: the intervals $[\underline{a}, \overline{a}]$ and $[\underline{b}, \overline{b}]$ intersect if and only if the following two inequalities are satisfied: $\underline{a} \le \overline{b}$ and $\underline{b} \le \overline{a}$.

How to check whether all points are matched correctly (and there are no outliers). Following our idea, the only way to check for outliers is to check, for some i and j, whether the corresponding intervals have a non-empty intersection. Thus, the corresponding checking algorithm should start with checking this property for some pair, then maybe generating another pair, checking this property for another pair, etc.

To minimize the overall time, we thus need to minimize the number of such intersection-checks. For a situation when there are no mismatches but we need to check it, we can have the optimal algorithm, with the smallest possible number of checks.

Proposition 2.1 For n points:

- every algorithm for testing whether there are mismatches takes at least $\lceil n/2 \rceil$ intersection-checks; and
- there exists an algorithm that tests whether there are mismatches by using $\lceil n/2 \rceil$ intersection-checks.

Proof: Let us first prove the first part of the proposition. After each successful intersection-check, we know that two points are correct. Thus, after k intersection-checks, we learn correctness of at most $2k$ points. To be able to conclude that all points are normal, we thus need to have $2k \geq n$, i.e., $k \geq n/2$. Since k is an integer, we must have $k \geq \lceil n/2 \rceil$. The first part of the proposition is proven.

To provide the second part, let us divide n elements into non-intersecting pairs. If n is even, we can do this; if n is odd, we have one element unattached – we combine this element with one of the already paired elements into one more pair. In both cases, we have $k \geq \lceil n/2 \rceil$ pairs. If all points are correct, then, by applying the intersection-check to all these pairs, we will confirm that all the points are normal. The proposition is proven. □

What if there are mismatches. If there is at least one pair (i, j) for which the distance intervals do not have a non-empty intersection, this means that at least one of the points is a mismatch. Based on only this information, we do not know which of the two points i and j is a mismatch. To find out, we need to perform the intersection-check between one of these points i and a point i_0 about which we know that it is correct.

Such know-to-be-correct points exist: if we know that the proportion of mismatched points is at most m, then there are no more than $m \cdot n$ mismatched points, and thus, no more than $m \cdot n$ mismatched pairs. These pairs contain $2m \cdot n$ points. So, if $m < 0.5$ (and usually, $m \leq 0.05$), then at least in some pairs, both points from the pair are correct, and thus, the above scheme will detect both points from the corresponding pair as correct.

If the intersection-check works form the pair (i, i_0), this means that the point i is correct and thus, the point j is a mismatch. If for the pair (i, i_0), the intersection-check does not work, this means that the point i is a mismatch. But we still do not know anything about the point j. To check whether j is a mismatch, we need to apply the intersection-check once again, this time, to the pair (i_0, j).

Resulting algorithm: description. First, we divide all points into non-intersecting pairs (possible with one exception, see above) and apply the intersection-check to each pair. If the intersection-check was successful for a pair, this means that both points from this pair are correct. At the end, we select one of the correct points i_0.

For each pair (i, j) that did not pass the intersection-check, we apply the same check to the pair (i, i_0). If the check succeeds, we conclude that i is correct and j is a mismatch. If the check fails, we conclude that i is a mismatch, and check the pair (i_0, j). If this check succeeds, then j is a correct point; if it fails, then j is also a mismatch.

Resulting algorithm: worst-case complexity. In the worst case, we need two extra checks for each mismatched point. Since we have no more than $m \cdot n$ mismatched points, we thus need $\leq \lceil n/1 \rceil + 2m \cdot n$ intersection-checks.

In particular, for $m = 0.05$, we need $\leq 0.6n + 1$ intersection-checks – slightly more that $\leq 0.5n + 1$ checks needed for the case when all points are correct and there are no mismatches.

Resulting algorithm: average-case complexity. Since the proportion of mismatched points is m, the probability that a randomly selected point is a mismatch is equal to m. In a randomly selected pair, we select both points independently, so the probability that both points are mismatches is thus equal to m^2 (we ignore the possibility of selecting the same point twice, since for large n it is very small, and we usually have up to $n = 50$ keypoints). There are $m^2 \cdot n$ such pairs. For each such pair, we need two extra intersection-checks, so the total amount of intersection-checks corresponding to such pairs is $2m^2 \cdot n$.

The remaining $m - m^2$ proportion of mismatched points belong to pairs in which only one point is a mismatch. Thus, there are $(m - m^2) \cdot n$ such pairs. When we perform an additional intersection-check for each such pair, then:

- with probability 1/2, we select this point in the additional intersection-check, and thus we need 2 additional checks;
- with probability 1/2, we select the correct point in the additional intersection-check and thus, as we mentioned earlier, we need only one extra intersection-check.

Thus, on average, for each such pair, we need $(1/2) \cdot 2 + (1/2) \cdot 1 = 3/2$ additional intersection-checks, so to the total average amount of extra intersection-checks corresponding to such pairs is $1.5 \cdot (m - m^2) \cdot n$.

So, the overall average computation time is equal to

$$\lceil n/2 \rceil + 2m^2 \cdot n + 1.5 \cdot (m - m^2) \cdot n \approx (0.5 + 1.5 \cdot m + 0.5 \cdot m^2) \cdot n.$$

In particular, for $m = 0.05$, we need, on average, $0.5775n$ intersection-checks – slightly more that $0.5n$ checks needed for the case when all points are correct and there are no mismatches and slightly less than $0.6n$ intersection-checks needed in the worst case.

How to parallelize this algorithm. If we have many processors working in parallel, then, on the first stage, when we apply the intersection-check to each pair, all these applications can be done in parallel. On the second stage, for each pair (i, j) that did not pass the original test, we can apply the intersection-check to both pairs (i_0, i) and (i_0, j) at the same time – and to all such pairs in parallel. Thus, on a highly parallel computer system, we can solve this problem in two steps.

Also, for each intersection-check, we can perform all subtractions in parallel, then all squarings in parallel, then all additions in two steps, and both comparisons ($\underline{a} \leq \overline{b}$ and $\underline{b} \leq \overline{a}$) also in parallel.

3 How to Actually Find the Angles and Shifts Corresponding to Rotation and Shift Between Frames

Discussion. In the 2-D case, this problem has been solved in [10] by using forward-backward contractors. We want to extend this solution to the 3-D case, but use linear programming instead.

In this extension, we can use the fact that the frames corresponding to the next moment of time are very close; thus, the angles describing a rotation between the two frames are very small, so we can safely ignore terms which are quadratic or higher order in terms of the angles and only keep linear terms in the resulting expansion. This is what we will do in this paper. (If we want to be precise, this can be easily done: we can bound the remainder and add this bound on the estimate Δ' of the corresponding uncertainty.)

General rotation formula and its linearized form. A general transformation of points in a rigid body can be described as a rotation followed by a shift (translation): $X = (x, y, z) \to X' = RX + S$, where $S = (s_x, s_y, s_z)$ is a shift, and a general rotation R can be represented as a composition of rotations around all three coordinate planes:

$$R = \begin{pmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 \\ 0 & \cos\varphi & \sin\varphi \\ 0 & \sin\varphi & \cos\varphi \end{pmatrix} =$$

$$\begin{pmatrix} \cos\theta\cos\psi & \sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi & \cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi \\ \cos\theta\sin\psi & \sin\varphi\sin\theta\sin\psi + \cos\varphi\cos\psi & \cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi \\ -\sin\theta & \sin\varphi\cos\theta & \cos\varphi\cos\theta \end{pmatrix}.$$

If we only keep terms which are linear in terms of small angles, then for each small angle α and β, we get $\sin\alpha \approx \alpha$, $\cos\alpha \approx 1$, and $\sin\alpha\sin\beta \approx 0$. Thus, in this linear approximation, the relation between the coordinates in the two frames takes the following form – taking into account that we can also have shifts s_x, s_y, and s_z:

$$x' = x - \psi\cdot y + \theta\cdot z + s_x;$$

$$y' = \psi\cdot x + y - \varphi\cdot z + s_y;$$

$$z' = -\theta\cdot x + \varphi\cdot y + z + s_z.$$

Here, $x = \widetilde{x} + \Delta x$, etc., where the differences Δx etc. are usually also small. Thus, e.g., in term $\psi\cdot y = \psi\cdot(\widetilde{y} + \Delta y)$ we can safely ignore the quadratic term $\psi\cdot\Delta y$ and thus get $\psi\cdot y \approx \psi\cdot\widetilde{y}$. As a result, the above formulas take the following form:

$$\widetilde{x}' + \Delta x' = \widetilde{x} + \Delta x - \psi\cdot\widetilde{y} + \theta\cdot\widetilde{z} + s_x;$$

$$\widetilde{y}' + \Delta y' = \psi\cdot\widetilde{x} + \widetilde{y} - \varphi\cdot\widetilde{z} + s_y;$$

$$\widetilde{z}' + \Delta z' = -\theta\cdot\widetilde{x} + \varphi\cdot\widetilde{y} + \widetilde{z} + \Delta z + s_z.$$

The first equation can be rewritten as

$$\widetilde{x} - \widetilde{x}' - \psi\cdot\widetilde{y} + \theta\cdot\widetilde{z} + s_x = \Delta x' - \Delta x.$$

The values Δx and $\Delta x'$ occur only in the first equation – and not in equations corresponding to other matched points in two frames. All we know about these values are the bounds $|\Delta x| \le \Delta_x$ and $|\Delta x'| \le \Delta'_x$. Thus, the set of possible value of the difference $\Delta x' - \Delta x$ is the interval $[-(\Delta_x + \Delta'_x), \Delta_x + \Delta'_x]$. So, the existence of such values is equivalent to the following inequalities:

$$-(\Delta_x + \Delta'_x) \le \widetilde{x} - \widetilde{x}' - \psi\cdot\widetilde{y} + \theta\cdot\widetilde{z} + s_x \le \Delta_x + \Delta'_x,$$

or, equivalently, that

$$\widetilde{x}' - \widetilde{x} - \Delta_x - \Delta'_x \le -\psi\cdot\widetilde{y} + \theta\cdot\widetilde{z} + s_x \le \widetilde{x}' - \widetilde{x} + \Delta_x + \Delta'_x. \tag{1}$$

Similarly, the y- and z-equations take the following forms:

$$\widetilde{y}' - \widetilde{y} - \Delta_y - \Delta'_y \leq \psi \cdot \widetilde{x} - \varphi \cdot \widetilde{z} + s_y \leq \widetilde{y}' - \widetilde{y} + \Delta_y + \Delta'_y; \tag{2}$$

$$\widetilde{z}' - \widetilde{z} - \Delta_z - \Delta'_z \leq -\theta \cdot \widetilde{x} + \varphi \cdot \widetilde{y} + s_z \leq \widetilde{z}' - \widetilde{z} + \Delta_z + \Delta'_z. \tag{3}$$

Resulting algorithm. For each matched point, we have similar three inequalities. All these inequalities have the same six unknowns: three angles and three shifts. To find the smallest and the largest value of each of these unknowns u, we use linear programming to find the smallest (or largest) value of this parameter u under all the conditions (1)–(3) corresponding to all the keypoints.

4 Conclusions

Given a set of corresponding 3D points between distinct moments in time, this paper introduces an algorithm to estimate the so-called rigid body transformation under interval uncertainty. The computation of this rigid body transformation is, for example, important in the context of mobile robotics where it is often used to describe the motion of a mobile robot. However, outliers can occur during the generation of the aforementioned 3D point correspondences, thus requiring an efficient outlier detection algorithm, which we also introduce in this work.

In the future, we aim to extend our algorithm to a dynamical system by also taking velocities or accelerations measured by the mobile robot into account. This requires us to additionally incorporate differential equations into the estimation of the robot's pose.

References

[1] J. Graeter, A. Wilcyznski, and M. Lauer, "LIMO: Lidar-Monocular Visual Odometry", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems IROS'2018*, Madrid, Spain, October 1–5, 2018.

[2] E. Hansen, "Sharpness in interval computations", *Reliable Computing*, 1997, Vol. 3, pp. 7–29.

[3] E. Hansen and G. W. Walster, *Global Optimization Using Interval Analysis*, Marcel Decker, 2004.

[4] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems IROS'08*, Nice, France, September 22–26, 2008.

[5] 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.

[6] S. P. Kleinschmidt and B. Wagner, "Visual Multimodal Odometry: Robust Visual Odometry in Harsh Environments", *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics SSRR'18*, Philadelphia, Pennsylvania, August 6–8, 2018.

[7] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision*, 2004, Vol. 60, No. 2, pp. 91–110.

[8] R. E. Moore, *Interval Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1966.

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

[10] M. Mustafa, A. Stancu, S. Pacheco Guteirrez, and E. Alexandru Codres, and L. Jaulin, "Rigid Transformation Using Interval Analysis for Robot Motion Estimation", *Proceedings of the 20th International Conference on Control Systems and Computer Science*, Bucharest, Romania, May 27–29, 2015.

[11] S. M. Prakhya, L. Bingbing, L. Weisi, and U. Qayyum, "Sparse Depth Odometry: 3D keypoint based pose estimation from dense depth data", *Proceedings of the 2015 IEEE International Conference on Robotics and Automation ICRA'15*, Seattle, Washington, May 26–30, 2015, pp. 4216–4223.

[12] R. Voges, C.S. Wieghardt, and B. Wagner, "Finding Timestamp Offsets for a Multi-Sensor System Using Sensor Observations", *Photogrammetric Engineering & Remote Sensing*, 2018, Vol. 84, No. 6, pp. 357–366.

[13] R. Voges, and B. Wagner, "Timestamp Offset Calibration for an IMU-Camera System Under Interval Uncertainty", *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Madrid, Spain, October 1–5, 2018.

[14] R. Voges, B. Wagner, and V. Kreinovich, "Efficient Algorithms for Synchronizing Localization Sensors Under Interval Uncertainty", *Reliable Computing (Interval Computations)*, 2020, accepted.

[15] G. W. Walster, "Moore's Single-Use-Expression Theorem on extended real intervals", *Abstracts of the 2002 SIAM Workshop on Validated Computing*, Toronto, Canada, May 23–25, 2002.

[16] J. Zhang and S. Singh, "Laser-visual-inertial odometry and mapping with high robustness and low drift", *Journal of Field Robotics*, 2018, Vol. 35, No. 8, pp. 1242–1264.