FAST INITIAL POSE ESTIMATION OF SPACECRAFT FROM LIDAR POINT CLOUD DATA

F. Rems[1], J. A. Moreno González[1], T. Boge[1], S. Tuttas[2], and U. Stilla[2]

[1]*German Aerospace Center (DLR), 82234 Weßling, Germany, florian.rems@dlr.de, toralf.boge@dlr.de*
[2]*Technische Universität München (TUM), 80290 München, Germany, sebastian.tuttas@tum.de, stilla@tum.de*

ABSTRACT

Advanced mission concepts like On-Orbit Servicing or Active Space Debris Removal require reliable relative navigation to a possibly uncooperative target. As optical navigation sensors, 3D-LiDAR scanners provide a three-dimensional point cloud of the target. For 6-DOF pose estimation, the tracker algorithm must be provided with an initial pose. This paper presents the prototype of a pose initialization algorithm tailored for the boundary conditions of spacecraft relative navigation. Test results based on real LiDAR data are evaluated.

Key words: spacecraft relative navigation; pose estimation; lidar.

1. INTRODUCTION

In a Low Earth Orbit the far range camera of a servicing satellite spots its designated target: A science satellite carrying expensive sensors for astronomical observations. It is adrift. Although its instruments are still fully operational, a malfunction of its batteries has rendered the scientific spacecraft widely useless. As the servicing satellite draws nearer, its optical navigation sensors determine the target's relative position and thus it can follow a precise approach trajectory. Soon, the distance shrinks to tens of meters and the close range camera as well as the LiDAR sensor begin to track the target's full 6 degree of freedom pose. With that information, the servicer calculates an optimal angle for the final approach. Adhering to that angle, it succeeds in grasping its target using a robotic arm. The servicer successfully captures its client satellite and replaces the defective battery unit. The astronomical satellite may still have years of useful operational lifetime.

This is how a so-called On-Orbit Servicing (OOS) mission [EKS08, SLW+09] could look like in the future. Such missions have the potential to prolong a satellite's lifetime considerably before it has to be replaced. Financial savings could be substantial. A different concept - Active Space Debris Removal (ADR) - aims not at repairing defective satellites in space, but rather at de-orbiting them safely and in a controlled fashion [BRD13]. For example, this may become necessary if a critical malfunction prevents a satellite from de-orbiting itself as planned or if some old upper rocket stage becomes a threat for other satellites in the same orbit.

Both concepts have a thing in common: The need for reliable relative navigation with respect to a possibly inactive target object. At large distances, only the relative direction of the target is considered, at a smaller distance, its relative position is determined and finally, at close range, its full six degrees of freedom position and attitude - its pose - is calculated. Knowing about the target's instantaneous attitude and motion is vital for preparing any docking or grasping maneuver. [BBR14]

Since the target's position in orbit must be expected to be only known roughly, optical navigation sensors are required to obtain real-time data about the target's position and attitude with respect to the servicing satellite. One type of sensor well-suited for providing these data is a Light Detection and Ranging (LiDAR) based scanner. By means of the time of flight of laser pulses, it measures the relative distance to the target. 3D-scanners use a mirror system to periodically deflect the laser pulses. Thus, a three-dimensional "image" of the target is created. This image can then be used to estimate relative position and attitude. [RLB12, NTA+06]

The usual way of continuously calculating the target pose from point cloud data is to use some variant of the Iterative Closest Point (ICP) algorithm. Given two point clouds of the same object, ICP iteratively minimizes the average squared distance between the nearest neighbor points, thereby calculating the coordinate transformation with the smallest error [Rus01, SMFF07]. Here, one of these point clouds is sampled from the reference model, the other is provided by the LiDAR scanner. In the context of real-time pose estimation, ICP constitutes a tracking algorithm. The current pose is estimated based on the solution of the previous time step. Evidently, a method is required that calculates the initial pose for the succeeding tracking algorithm without any a priori knowledge of the target pose.

The importance of this pose initialization algorithm in

spacecraft relative navigation must not be underestimated. It is true that the tracking algorithm is active during the complete close range approach, and it is also true that it constitutes a major factor for overall relative navigation accuracy and performance. However, without proper initialization the tracker will not work at all. In that case, unreliable pose initialization may impede a mission considerably. While a tracking algorithm may provide a solution with bad accuracy (as long as tracking is not lost) from time to time which is attenuated by the navigation filter, a wrong initial pose can not be tolerated. It may cause the tracker to converge to a local and completely wrong solution with corresponding consequences for relative navigation. But reliability is not enough. Pose initialization also has to be fast. In case tracking is lost during close range approach, fast re-initialization may be an important safety factor. And in scenarios that involve a tumbling target object, fast pose initialization is simply necessary. The tracker cannot work with out-dated initial values.

The problem of estimating the pose of an object from a scan point cloud is equivalent with calculating the transformation between two point clouds representing the object from different perspectives. This task is referred to as range image registration. Usually, the term is associated with constructing three-dimensional models of real objects based on range image data which give a partial view of the object from different viewpoints. Following the categorization of [SMFF07], fine and coarse registration methods can be distinguished. The former calculate very accurate solutions based on initial transformations (e.g. ICP). The latter do not need any initial information and achieve a lower accuracy. The coarse type is exactly what is needed for pose initialization. Common methods include spin image, principal component analysis, genetic algorithms and RANSAC-based DARCES [SMFF07]. However, at least without any adaptation, these methods are only of limited suitability in the context of spacecraft relative navigation.

This becomes clear when looking at the boundary conditions of pose initialization.

- In this work, only uncooperative target objects are considered. There are no markers (retroreflectors) available for aid. Moreover, the target does not provide any information about its position (e.g. by GPS) or attitude.
- Satellites often incorporate symmetries in their principle shapes. Moreover, many flat and large areas (e.g. solar panels) have to be expected.
- Since the target must be tracked in real-time with a period of at most a couple of seconds, the scan point cloud will be sparse. In general, not more than a few hundred points are available. Naturally, the data will be subject to noise. And since mainly scanning LiDAR sensors are considered in this work, the point distribution will also be unstructured; there will be no matrix-like ordering of the points. Additionally, scanning the moving target takes a certain amount of time, naturally, and therefore results in a distorted point cloud. However, this effect can be assumed negligible in the context of initial pose estimation.
- The scan point cloud covers the target only partially ($<50\%$) but is completely contained in the model, i.e. in the model point cloud. There is only a single target object to be considered. Therefore, no segmentation of the scan data is required. LiDAR laser beams that do not hit the target will simply never return an echo.
- A low accuracy of the initial pose estimate can be tolerated insofar the succeeding tracking algorithm converges to the correct solution. An angular accuracy of $\pm$ 10 degrees and a translational accuracy of 15% of the target size are considered sufficient in this paper.
- Since the target may be tumbling, pose initialization should take at most a couple of seconds on space-qualified hardware. It has to be considered that computing power is strongly limited on board of a spacecraft.
- In contrast, reliability is highly important. If a solution is provided, it must be the correct solution. No solution is preferable over a wrong solution.

Looking at the aforementioned coarse range image registration methods: Spin image is computationally expensive and strongly depends on point cloud resolution. Principal component analysis requires a sufficient number of points and a large overlap. Genetic algorithms also demand high computing performance. RANSAC-based DARCES shows long processing times for large point clouds and the accuracy depends on point cloud resolution. Since in spacecraft relative navigation involving LiDAR sensors the number of points are limited and compromise concerning accuracy can be made, an approach oriented towards RANSAC-based DARCES [CH99] looks the most promising.

This paper presents the prototype of a pose initialization algorithm which is tailored for the requirements of LiDAR-based spacecraft relative navigation for OOS, ADR and alike scenarios. The algorithm shows some similarity with RANSAC-based DARCES, although it differs from this concept in some major aspects. Test results based on real LiDAR data obtained from simulations with the the European Proximity Operations Simulator (EPOS) are given. Finally, future work to improve the algorithm are indicated.

Throughout this paper, the spacecraft that actively carries out e.g. the OOS maneuver is denoted *chaser*. The client spacecraft or the piece of space debris is called the *target*. This terminology is generic and covers many mission scenarios where relative navigation is important. The combination of position and attitude is called *pose*.

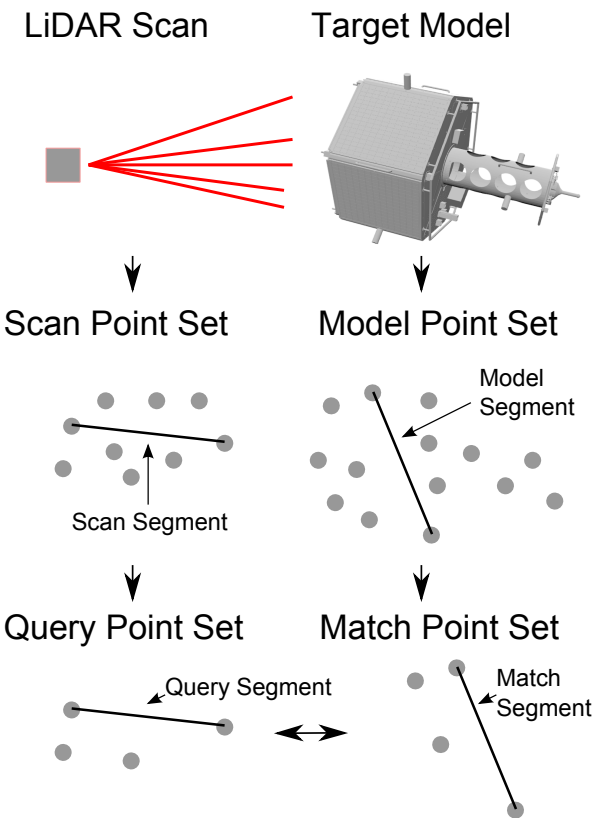

Figure 1. Illustration of expressions related to scan and model points used in this paper.

2. ALGORITHM

It is given a model of the target, usually a three-dimensional mesh derived from a CAD-model, in the target frame, as well as a scan point cloud in the LiDAR frame. The goal is to obtain the coordinate transformation between LiDAR frame and target frame. For reasons that will become clear below, the target mesh is sampled to obtain a point cloud of the model, henceforth denoted model point set or simply model set $M := \{\mathbf{m}_1, \mathbf{m}_2, \dots\}$. Similarly, the LiDAR data is called scan point set or simply scan set $S := \{\mathbf{s}_1, \mathbf{s}_2, \dots\}$. In terms of range image registration, a scan set is registered with a model set, thereby calculating the transformation that leads to the minimum error between both point sets.

The algorithm's principle strategy involves choosing a subset of scan points - a query (point) set $Q := \{\mathbf{q}_1, \mathbf{q}, \dots\}$, $Q \subseteq S$ - and identifying these very points in the model point set, giving a match (point) set $A := \{\mathbf{a}_1, \mathbf{a}, \dots\}$, $A \subseteq M$. With this correspondence given, the transformation can readily be calculated, using e.g. least-squares fitting [KSA87]. It is not necessary to derive any features using neighborhood information or curvature etc.. Figure 1 gives an overview of the terms used.

To find candidate match sets from query sets, we ask "What is the most simple property of the query set which can be found in the model set?" It is the distance between pairs of points. In this paper, these point-pairs are called segments. A segment is characterized by a length and its associated two points. For example, a scan segment $\{\mathbf{s}_i, \mathbf{s}_j\}$ has the length $L\{\mathbf{s}_i, \mathbf{s}_j\}$. A segment represents an *unordered* pair of points, i.e. $\{\mathbf{s}_i, \mathbf{s}_j\} = \{\mathbf{s}_j, \mathbf{s}_i\}$. As indicated in Figure 1, scan segments, query segments, match segments and model segments are associated with the different point sets. The query set Q can be characterized as a number of segments $\{\mathbf{q}_i, \mathbf{q}_j\}$ with $i \neq j$ and $i, j = 1 \dots |Q|$. The segment lengths $L\{\mathbf{q}_i, \mathbf{q}_j\}$ could be interpreted as a table or matrix, where columns and rows relate to the query points. This matrix can be considered characteristic for the query point set. Any set of model points, i.e. any match set, which shows point-to-point distances consistent with those of the query set, is a candidate for correctly representing the query set in the model. (Due to ambiguities because of geometry and noise, a match set fulfilling this criterion can, but must not be the correct solution.)

The statement that a model segment is consistent with some query segment, i.e. that the lengths are similar, implies some kind of tolerance. This is due to the finite resolution of the model set and to the presence of noise in the scan set. Therefore, while scan and query segments have a definitive length only, model segments additionally are assigned a minimum and a maximum length. This definition is used to keep this tolerance range generic. A query segment is consistent with a model segment, if the query segment's length lies between minimum and maximum length of the model segment.

$$L_{\min}\{\mathbf{m}_h, \mathbf{m}_i\} \leq L\{\mathbf{q}_j, \mathbf{q}_k\} \leq L_{\max}\{\mathbf{m}_h, \mathbf{m}_i\}$$

The condition is henceforth abbreviated

$$\{\mathbf{q}_j, \mathbf{q}_k\} \blacktriangleleft \{\mathbf{m}_h, \mathbf{m}_i\}$$

The width of the interval must be chosen according to the resolution, possibly specific for each segment. Here, half of the distance of a model point to its nearest neighbor model point is chosen as an approximation of the point's local resolution. For a segment, the individual local resolution of both points is subtracted from the actual length of the model segment to obtain the minimum length, and added to obtain the maximum length.

With this notation, given a query set Q with size $|Q|$, a match set A with size $|A| = |Q|$ is a candidate match set if

$$\{\mathbf{q}_i, \mathbf{q}_j\} \blacktriangleleft \{\mathbf{a}_i, \mathbf{a}_j\} \tag{1}$$

for all segments, $\forall\, i, j \in 1 \dots |A|$ with $i \neq j$.

In the following way, all candidate match sets for a given query set are found progressively.

Let $Q := \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \dots\}$ be a query set of size $|Q|$. Query points $\mathbf{q}_1$ and $\mathbf{q}_2$ are considered first, i.e. the first query segment. All model-point-pairs, i.e. model segments with a length consistent with that of the first query segment, have to be found. From each such model segment, a candidate match set is created. I.e.

$$\{A | A \subseteq M, |A| = 2, \{\mathbf{q}_1, \mathbf{q}_2\} \blacktriangleleft \{\mathbf{a}_1, \mathbf{a}_2\}\}$$

But with only two points no coordinate transformation can be calculated. The next query point $\mathbf{q}_3$, is now considered. The model point set is searched for all points with distances to match points 1 and 2 consistent with the distances of query point 3 to query points 1 and 2. For every model point that fulfills this condition, a match set is created. In other words, for each former match set of size two, it can be found 0, 1 or more match sets of size three.

$$\{A|A \subseteq M, |A| = 3, \{\mathbf{q}_1, \mathbf{q}_2\} \blacktriangleleft \{\mathbf{a}_1, \mathbf{a}_2\}, \\ \{\mathbf{q}_2, \mathbf{q}_3\} \blacktriangleleft \{\mathbf{a}_2, \mathbf{a}_3\}, \\ \{\mathbf{q}_1, \mathbf{q}_3\} \blacktriangleleft \{\mathbf{a}_1, \mathbf{a}_3\}\}$$

Three points, a triangle, is the minimum to calculate a coordinate transformation. Among these candidate match sets are some that lead to a coordinate transformation with sufficient accuracy for the succeeding tracking algorithm. At this point, in principle, all these coordinate transformations could be calculated and applied to the whole scan point set to filter out the best solution by the averaged squared distance sum.

But can that effort be reduced by identifying match point sets, i.e. triangles that are likely to lead to wrong solutions? For that purpose, the remaining query points are considered. For each match set, a model point consistent with the next query point is searched. As soon as one is found, it is added to the match set and the next query point is dealt with. If none is found, the whole match set is discarded. The difference to the original first three points should be emphasized. Match sets are not created for *each* next model point that is consistent with the other match points and the query set. The question is rather, can one (or more) or none be found? Example: Given $A = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3\}$ which is consistent with the first three elements of Q, try to find a $\mathbf{m} \subset M$ that fulfills

$$\{\mathbf{q}_1, \mathbf{q}_4\} \blacktriangleleft \{\mathbf{a}_1, \mathbf{m}\} \\ \{\mathbf{q}_2, \mathbf{q}_4\} \blacktriangleleft \{\mathbf{a}_2, \mathbf{m}\} \\ \{\mathbf{q}_3, \mathbf{q}_4\} \blacktriangleleft \{\mathbf{a}_3, \mathbf{m}\} \\ \{\mathbf{q}_4, \mathbf{q}_4\} \blacktriangleleft \{\mathbf{a}_4, \mathbf{m}\}$$

If one can be found, $\mathbf{a}_4 = \mathbf{m}$ is added to the match set, obtaining $A = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4\}$. Otherwise the match set is discarded completely. This is executed until the remaining candidate match sets have the same size as the query set. The idea behind this is that a correct match set should also be consistent with a larger number of query points, for these very points are indeed present in the scan and can thus be expected to be present in the model. A wrong match set should, sooner or later, fail to pass this test.

The result is a considerably smaller number of candidate match sets of which some may represent the match set in the model with sufficient accuracy and which all have the same size as the query set, fulfilling (1). There can only be as many as or fewer than the original triangles, i.e. match sets of size 3.

This routine, `findCandidateMatchSets`, is summarized in Figure 2 as a Nassi-Shneiderman diagram.

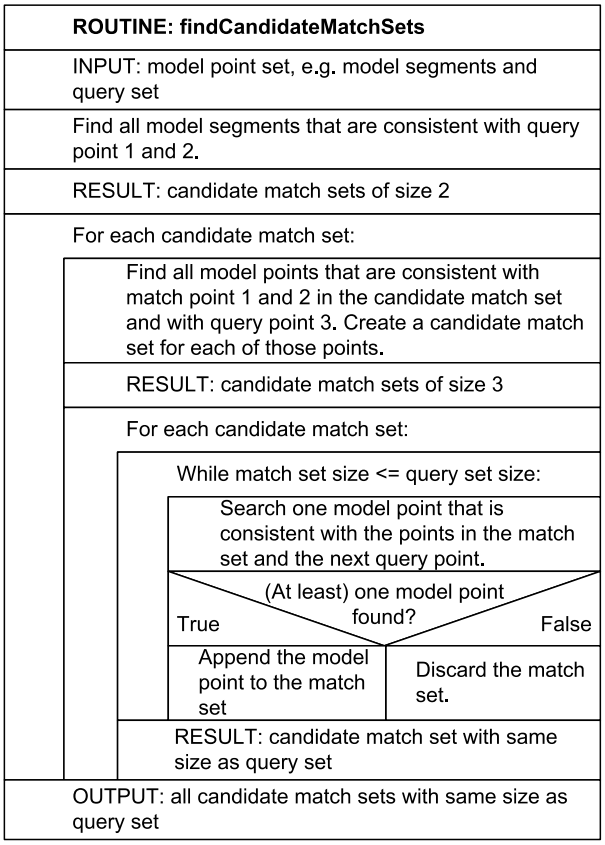

Figure 2. Nassi-Shneiderman diagram of routine `findCandidateMatchSets`.

The remaining steps are trivial and have already been indicated. All candidate match sets are used to calculate the transformation. Then these transformations are applied to the whole scan set and the averaged squared distance sums of the nearest neighbor points are calculated as error metric. The transformation with the smallest error is reported as estimated pose.

Note that the size of the query set is critical in the routine `findCandidateMatchSets`. Since the real scan data includes noise and a small fraction of outliers, a query set that is too large will have all candidate mach sets be discarded. If the query set comprises too few points, a large number of candidate match sets will be found and this will worsen performance. But how is the optimal query size determined? A small change to `findCandidateMatchSets` handles this problem.

A query size is chosen so large that no candidate match sets of the same size can be found. Let $C := \{A_1, A_2, \dots\}$ be a set of candidate match sets. In `findCandidateMatchSets`, instead of discarding a candidate match set A as soon as no more consistent model points are found, the following cases are considered. If the candidate match set in question is as large as the other candidate match sets in C, it is added to C. If A is strictly larger, all other candidate match sets are removed from C first, before A is added to C. How-

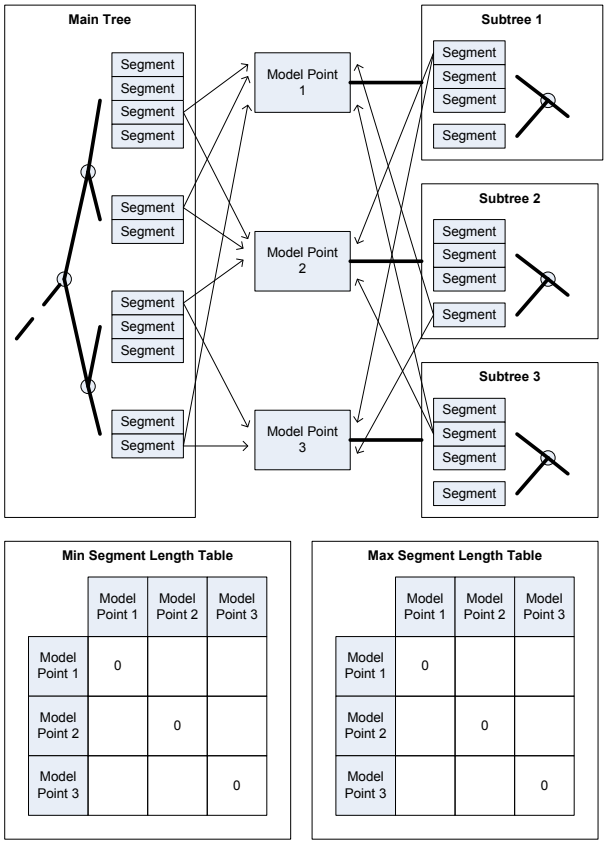

Figure 3. Overview of used data structures.

ever, if it is smaller, it is simply discarded. The result is a number of candidate match sets of equal size which represent the largest match sets consistent with the first $|A|$ query points in the query set that could be found. Checking any of these match sets for another query point would fail. In essence, as many of the original triangles as possible are discarded while keeping those with the highest probability of being accurate ones. This approach works because it is guaranteed that any point in the scan set is contained in the model set. As a consequence, the candidate match sets are as large as or smaller than the query set, $|A| \leq |Q|$.

To speed up routine `findCandidateMatchSets` a combination of pre-calculated data structures is used (see Figure 3). So-called segment trees play an important role. Note that the term segment in "segment tree" has a slightly different meaning than the segments, i.e. point-pairs as used throughout this paper. In short, a segment tree is a data structure that efficiently stores intervals for fast lookup. The tree can be used for rapidly reporting all intervals stored in the tree that contain some value [dBCvKO08]. And this is precisely what can speed up the correspondence search. Each model segment has a minimum and a maximum length, in other words, it represents an interval. Each scan/query segment has a single definitive length. To find a model segment that is consistent with some scan or query segment, such a tree can be used to report all model segments fulfilling the condition

$$\{\mathbf{q}_j, \mathbf{q}_k\} \blacktriangleleft \{\mathbf{m}_h, \mathbf{m}_i\}$$

One tree - here called main tree $\mathfrak{T}$ - stores all model segments $\{\mathbf{m}_i, \mathbf{m}_j\}$, $i \neq j$, $i, j \in 1 \ldots |M|$. Each model segment in the main tree is associated with precisely two model points. And each model point is associated with another segment tree, called a subtree in this paper. The subtree of model point 1 $\mathfrak{T}(\mathbf{m}_1)$ stores all model segments that start (or end) at point 1, i.e. $\{\mathbf{m}_1, \mathbf{m}_j\}$, $j \neq 1$, $j \in 1 \ldots |M|$. Finally, two tables are stored for minimum and maximum model segment length, These tables are basically the matrices $L_{\min}\{\mathbf{m}_i, \mathbf{m}_j\}$ and $L_{\max}\{\mathbf{m}_i, \mathbf{m}_j\}$ for all segments. Figure 3 gives an overview of the data structures. How can these data structures accelerate the search? To see this, let's consider another example. A query set $Q := \{\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_2\}$ is given, as well as a corresponding match set with the two first match points $A := \{\mathbf{a}_1, \mathbf{a}_2\}$. According to `findCandidateMatchSets`, all model points $\mathbf{m}_i$ have to be found that are consistent with the already found match points and with the query set, which is equivalent to the conditions

$$\{\mathbf{q}_1, \mathbf{q}_3\} \blacktriangleleft \{\mathbf{a}_1, \mathbf{m}_i\}$$
$$\{\mathbf{q}_2, \mathbf{q}_3\} \blacktriangleleft \{\mathbf{a}_2, \mathbf{m}_i\}$$

Instead of going through all points, $\mathfrak{T}(\mathbf{a}_1)$ is used to rapidly provide us with all model segments that have $\mathbf{a}_1$ as one of their points and fulfill the first condition above. The segments' other points are the very $\mathbf{m}_i$ that fulfill the first condition. The second condition is applied to these segments by looking up minimum and maximum segment lengths in the tables. The main tree is used for first search, when there haven't been determined any match sets yet. Considering $L\{\mathbf{q}_1, \mathbf{q}_2\}$, the main tree reports all model segments fulfilling condition $\{\mathbf{q}_1, \mathbf{q}_2\} \blacktriangleleft \{\mathbf{m}_i, \mathbf{m}_j\}$.

But there is another possibility to speed up the correspondence search. The query points can be ordered in such a way that the overall number of necessary iterations is minimized. To see this, assume a situation where `findCandidateMatchSets` returns candidate match sets with the same size as the query set. To recapitulate: The first match point corresponds to the first query point, the second to the second, etc.. If the order of query points is changed prior to the search, the routine will return candidate match sets with an accordingly changed order so that query and match points are still associated correctly. So, the order of processing the query points can be chosen freely without altering the result. The principle job of `findCandidateMatchSets` is to discard "bad" match sets. The earlier in the search (early meaning small match sets) wrong candidates can be discarded, the fewer consistency checks have to be carried out later in the search when match sets are already large. The probability of discarding a match set is higher, if there are only few possible model segments in the subtree for the query segment in question. By ordering the query points such that the query segment with the

lowest number of possible model segments occurs first, the query segment with the second lowest number second etc., the number of overall consistency checks is minimized for a given query set. The main tree can easily be used to report only the number of model segments. Thus, the reordering can be done efficiently.

To account for this semantically: From a query set $Q := \{\mathbf{q}_1, \mathbf{q}_2, \dots\}$ a query sequence $\tilde{Q} := (\tilde{\mathbf{q}}_1, \tilde{\mathbf{q}}_2, \dots)$ is created by reordering. Respectively, the correspondence search takes a query sequence and returns a number of candidate match sequences $\tilde{A} := (\tilde{\mathbf{a}}_1, \tilde{\mathbf{a}}_2, \dots)$.

The improvements and changes made to `findCandidateMatchSets` above, are summarized in the Nassi-Shneiderman diagram depicted in Figure 4. The improved routine is called `findCandidateMatchSequences`. Note that the ordering of query points is done as a pre-step for the correspondance search and is therefore not depicted in the diagram.

There is one final piece missing. How is a query set chosen from the scan set? In [CH99], the authors suggest for their RANSAC-based DARCES approach to choose points that form a triangle as small as possible above some edge length threshold. Thereby, they reduce the volume of possible candidate points in the model. But, by using the pre-calculated data structures, our algorithm works differently and is not subject to such restriction. Moreover, in spacecraft relative navigation, the target will always be some kind of satellite. As already indicated in the introduction, such satellites have a relatively simple major shape, usually some primitive like a cube or prism. This implies large and flat surface areas. If the first three initial query points are chosen to form a small triangle, there is an enormous number of possibilities to place this triangle on the aforementioned flat faces and thus an enormous amount of possible match sequences at this stage of the correspondence search. For these reasons, a different approach is followed here. The idea is to cover the major shape of the target uniformly, implying query points that cover ideally the whole volume of the scan set. A sampling method well-suited is farthest point sampling [ELPZ97]. For the algorithm in this paper, the sampling is started with a random initial scan point to allow multiple major iterations of the algorithm without using the same query set over and over again. The next sample is chosen to be the scan point with the largest distance to the initial point. The following sample has the largest distance to both already sampled points and so forth.

Now all parts necessary to assemble the complete algorithm are available. It is summarized in the Nassi-Shneiderman diagram depicted in Figure 5 as routine `estimatePose`. The input is formed by a scan point set and a model point set. First, the query set is obtained from the scan set by farthest point sampling with a random initial point. This query set is then reordered into a query sequence for optimal performance of the following routine `findCandidateMatchSequences`. This routine returns a number of candidate match sequences as large as or smaller than the query sequence. The transformations between query sequence and each candidate match sequence is calculated by least-squares fitting [KSA87]. For each of these transformations, the RMS value of the averaged squared distance sum between scan point set and model point set is calculated as an error metric. The very transformation with the smallest error is chosen as the estimated pose, provided that the error is below some threshold. If this is not the case, the whole process is repeated with a different query set.

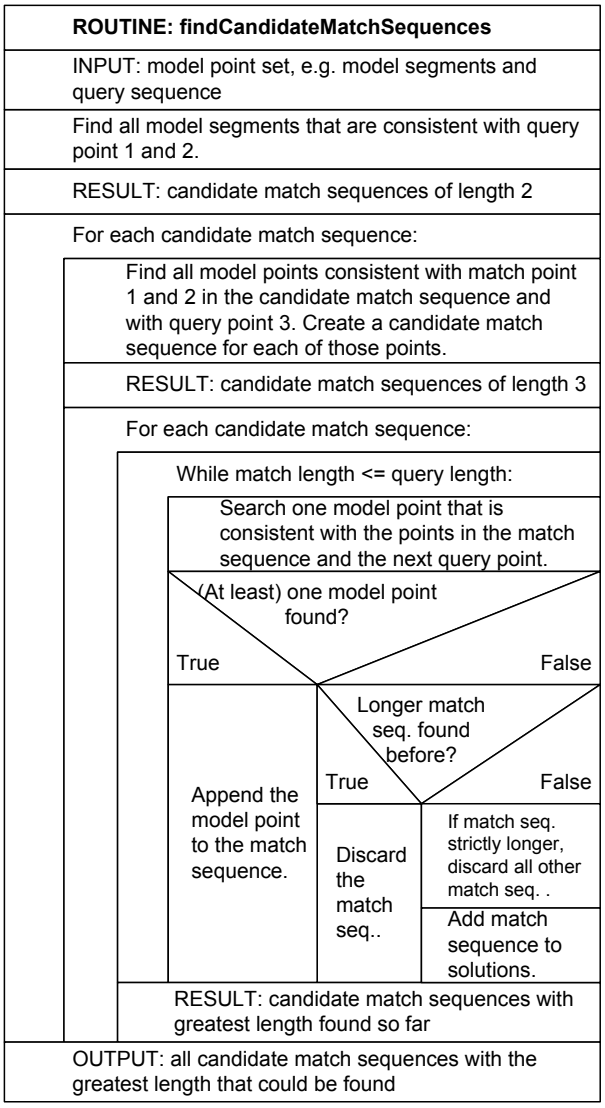

Figure 4. Nassi-Shneiderman diagram of routine `findCandidateMatchSequences`.

3. ALGORITHM VERIFICATION

In this section, the algorithm is tested with point cloud data obtained by real LiDAR sensor hardware. The data represent one test trajectory from a larger test campaign conducted by the German Aerospace Center (GSOC) in

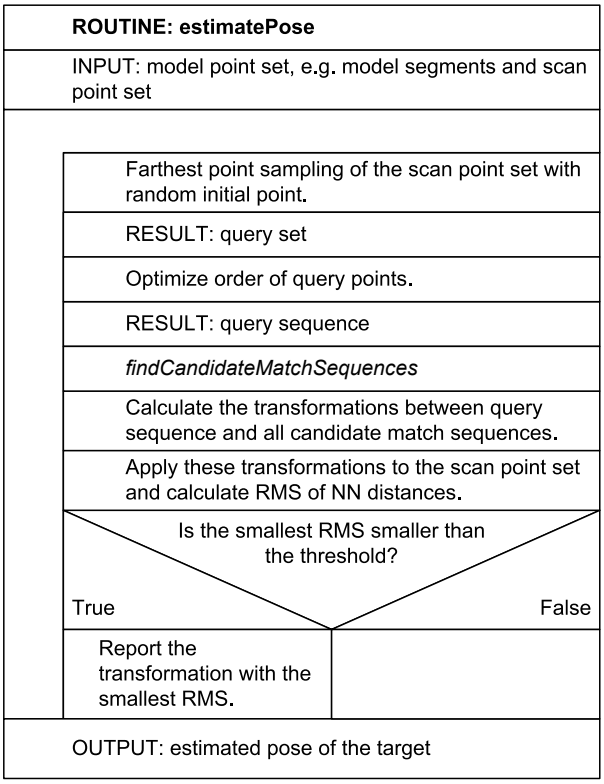

Figure 5. Nassi-Shneiderman diagram of routine `estimatePose`.

Oberpfaffenhofen, Germany, for EADS Astrium (now Airbus Defense and Space) which provided the LiDAR sensor, the target mockup and simulated approach trajectories. The test campaign involved the European Proximity Operations Simulator (EPOS) located at GSOC.

3.1. Setup

Figure 6 gives an overview of the EPOS laboratory. The main elements of the facility are two standard industrial robots, one of them mounted on a linear real. Each robot has six Degrees of Freedom (DOF). Along with one DOF of the linear rail, 13 DOF are available for simulating the relative position and orientation of two spacecraft in Rendezvous and Docking (RvD) scenarios. A real-time control system allows conducting Hardware-in-the-Loop (HiL) tests with real sensors and true-to-scale satellite mockups. A 12 kW daylight spotlight provides realistic illumination conditions concerning power density and sun-resembling spectrum. [BWMT10, TB11, TB12]

For the test case trajectory, a mockup of the target satellite was mounted to the flange of robot 2 and the LiDAR sensor breadboard to the flange of robot 1. Figure 7 gives an impression of this configuration. During the test, the robots are controlled according to a simulated approach trajectory such that the *relative* motion between sensor and target is identical to the simulated motion, just like it is expected to occur during an OOS mission in space.

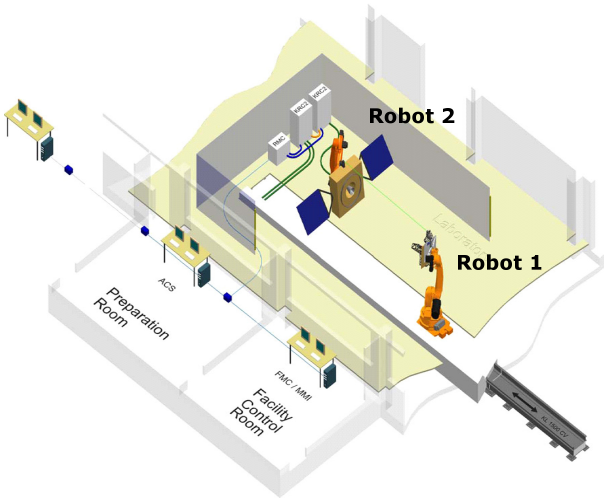

Figure 6. Layout of EPOS facility.

The EPOS simulator shows a translational accuracy of $< 2\,\text{mm}$ (3σ) and a rotational accuracy of $< 0.2\,\text{deg}$ (3σ) [TB12].

The topmost photo in Figure 8 allows a closer look at the true-to-scale target mockup. It consists essentially of two parts: The main body is formed by a prism with a hexagonal base. And a cylindrical docking boom is mounted along the central axis of the main body. The overall length along this central axis is approximately $2.2\,\text{m}$, the main body measures roughly $1.5\,\text{m}$ from solar panel to opposite solar panel and $0.8\,\text{m}$ from hexagon to hexagon. The mockup's surface consists of materials with realistic optical properties compared to a real satellite. Note the high symmetry of the target. Rotating it around the central axis through multiples of $60\,\text{deg}$ results in the same silhouette, save for some small details.

Figure 8 also shows the rendered mesh of the target mockup and the derived model point set.

Figure 9 depicts the scanning LiDAR sensor breadboard as it is mounted to the flange of robot 1. The $4\,\text{ns}$ wide

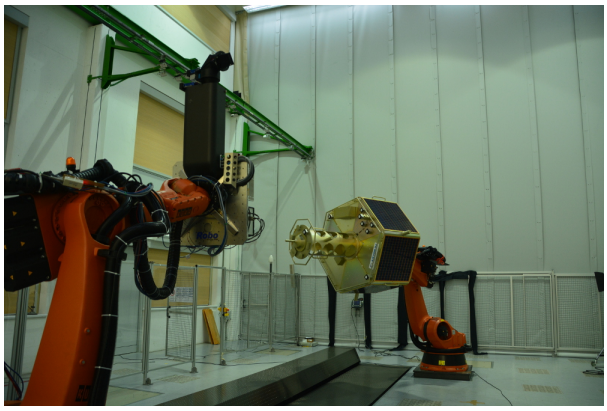

Figure 7. Look into the EPOS laboratory hall.

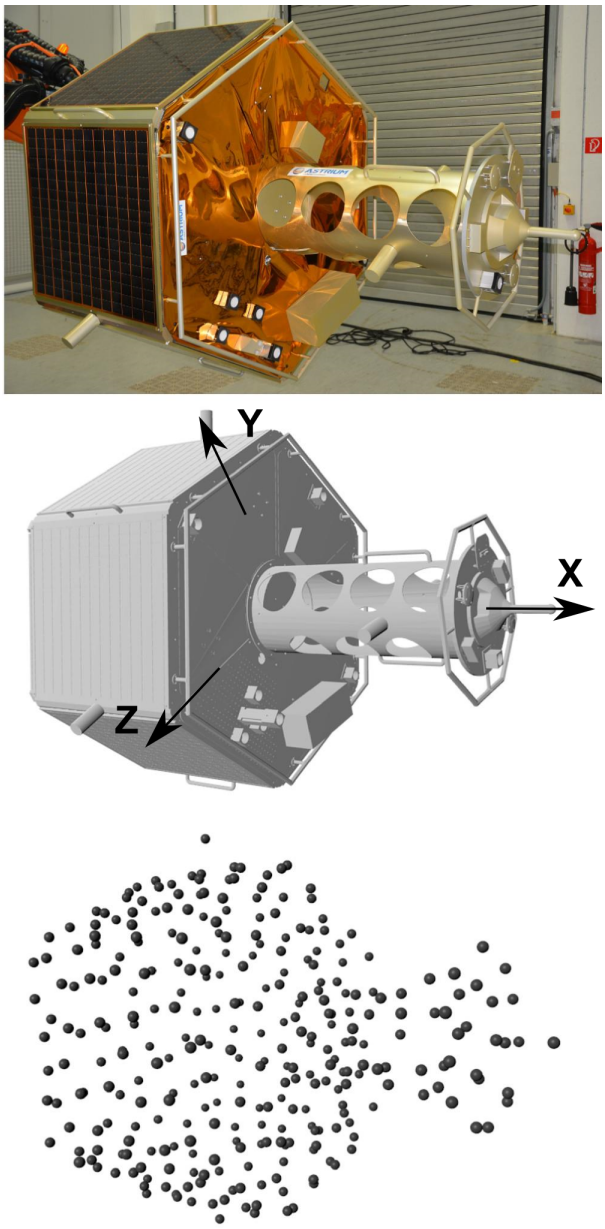

Figure 8. Photo, mesh and model point set of target mockup.

laser pulses are reflected by a single gimbal-mounted mirror, which oscillates in both axis with different frequencies to generate a scanning pattern of more or less parallel lines on the target. The wavelength of the laser is 1550 nm and the pulse repetition rate 30 kHz. For the trajectory considered in this paper, the mirror frequencies were chosen such that the sensor field of view was scanned completely about three times per second.

An exemplary scan set is shown in Figure 10. Only the points are included that represent the target. All other points like the floor or the walls of the laboratory have been removed. This was done for the complete test trajectory and accounts for the actual situation in space. The

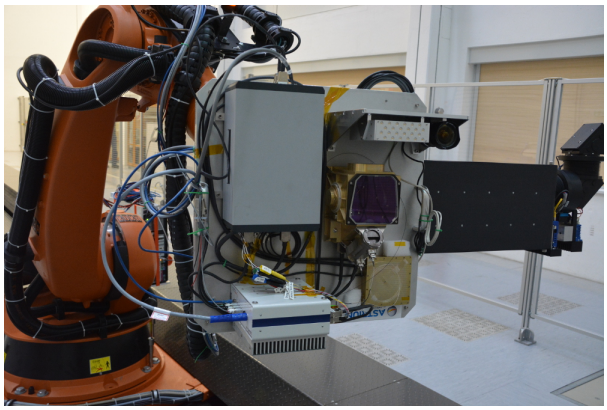

Figure 9. LiDAR breadboard mounted to robot flange.

number of scan points varies between about 600 and 900.

The 400 scan sets belong to a 130 s long trajectory. The distance between chaser and target varies only little around 8 m. Figure 11 shows the target's reference position over time with respect to the sensor. The target is tumbling slowly combined with a spin about the principle axis. Figure 12 shows the target's reference attitude in euler angles over time. Taking into account the rotation order ZYX for the diagram, the spin about the target's x-axis and the tumbling motion mainly present in the other axes is clearly seen.

The pose estimation results in the subsequent subsection have been created with a desktop computer (Intel Core i5 780 @2.8 GHz, 8 GiByte RAM, Windows 7 Professional) using the following parameters. Poisson-disk sampling was used to obtain a model set of 301 points representing the target mesh. This is precisely the model set depicted in Figure 8. The size of the query set was 15. As a minimum value for the averaged squared distance sum $0.015\,\mathrm{m}^2$ was chosen.

3.2. Results

In Figure 13, the distance error between estimated position and reference position is given as a histogram, along

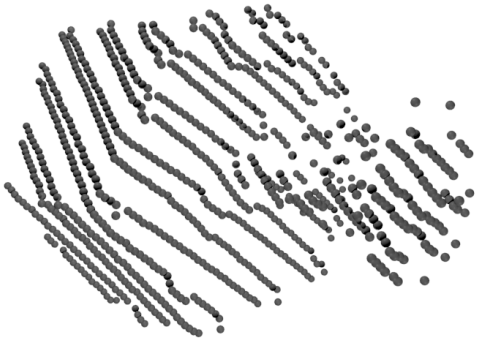

Figure 10. Example of a scan set.

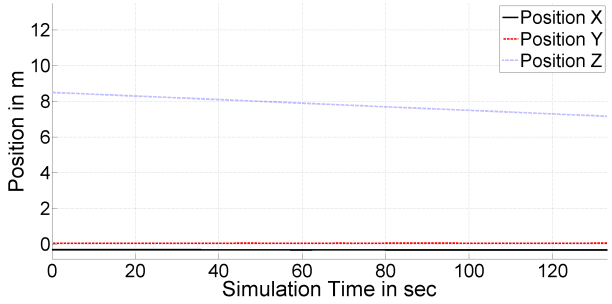

Figure 11. Reference position with respect to LiDAR.

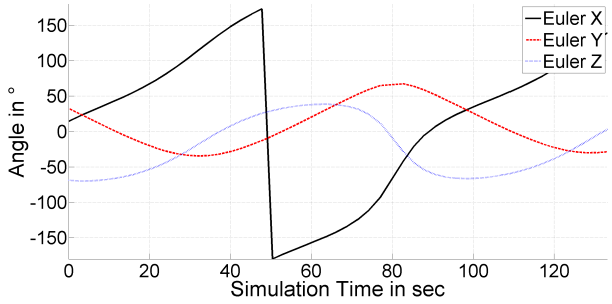

Figure 12. Reference attitude with respect to LiDAR. (Rotation order ZYX)

with mean, maximum and minimum values.

Respectively, Figure 14 shows the distribution of the deviation between estimated attitude and reference attitude. The figure shows the absolute value of the Euler angles, again including mean, maximum and minimum values. The target's symmetry is removed from Euler angle X. Multiples of 60 deg are subtracted. Thus, the Euler angle X error always lies between 0 deg and 30 deg.

The processing time is given in Figure 15. This time includes all steps of the algorithm presented in this paper. However, it does not include any pre-processing steps required for calculating the segments, the tree data structures and the segment tables. In a real mission, these calculations would be carried out beforehand.

The algorithm requires about 27 MByte of memory, including the data structures.

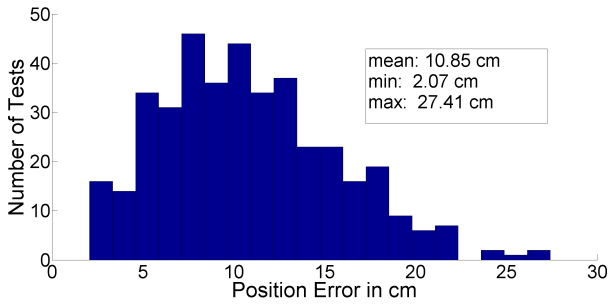

Figure 13. Position error distribution.

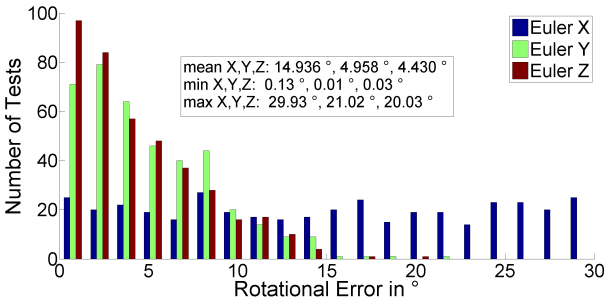

Figure 14. Rotation error distribution.

3.3. Discussion and Possible Improvements

Considering the size of the target mockup, the position error (Figure 13) is of a reasonable magnitude. Even the few outliers with roughly 28 cm can still be tolerated.

The angular errors (Figure 14) differ distinctly for the different Euler angles. Looking at the angles Y and Z, a mean error of 5 deg is absolutely acceptable for pose initialization. But there are also few but distinct outliers at about 20 deg. Most likely, these are cases where the initial sampling of the scan set leads to a query set with many inaccurate scan points. Generally, such outliers could be reduced by decreasing the threshold for the averaged squared distance sum, thereby increasing the number of major iterations. But this would also lead to an increased processing time. Looking at the distribution, there are only very few outliers with such large angular errors, which relativizes the problem. The angular error around the X axis is a different matter. It seems that the angle around this axis cannot be determined at all. The error is distributed uniformly between 0 deg and 30 deg. The reason is evident: The target shows a certain symmetry around its x-axis. Due to the hexagonal shape, there are ambiguous orientations. Apparently, the algorithm is hampered by these ambiguities and doesn't even find one of the ambiguous orientations correctly. Although symmetries are a principle problem all range image registration algorithms show to some degree [SMFF07], here this is definitely a point for improvement. However, we are confident that the succeeding tracking algorithm easily converges to one of the ambiguous orientations about the x-axis.

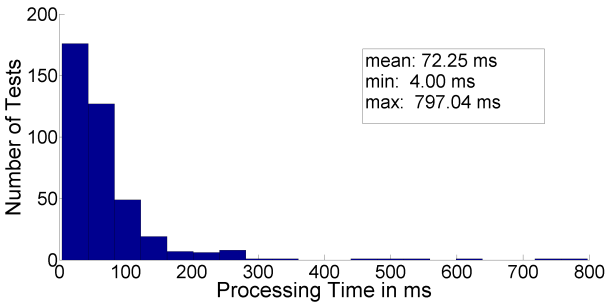

Figure 15. Processing time distribution.

Figure 15 shows the processing time distribution. Note that in the overall majority of tests, processing times lie somewhere below 200 ms. However, there are very few outliers with very large processing times up to 800 ms. The possibility of especially noisy scan data cannot be excluded. Considering the prototype status of our algorithm, these values are satisfactory.

In general, the next steps should involve increasing the algorithm's performance further. This has two aspects: The initial pose can be calculated faster or, by setting a smaller threshold for the average squared distance error, the accuracy can be increased while keeping processing time constant.

We see potential in improving the implementation itself. The algorithm involves many layers of hierarchical loops where small optimization can have large effects. Execution could also be accelerated by decreasing the number of model segments without reducing the resolution of the model set. This could be achieved by carrying out a visibility analysis. What point-pairs cannot be scanned together at all? For example, the segments of points on totally opposite sides of the target could be excluded.

4. CONCLUSION

Reliably and rapidly calculating the initial pose of a target in spacecraft relative navigation is an important step in any OOS, ADR or similar mission. The paper has presented the prototype of an algorithm that calculates this initial pose estimate based on 3D-LiDAR point cloud data. The algorithm determines scan-point/model point correspondences by comparing point-pair distances. This is accelerated by sophisticated processing of tailored data structures. Tested with real LiDAR hardware at the EPOS facility, the prototype algorithm shows promising results. Average angular errors of 5 deg and position errors of 10 cm are achieved in a mean processing time of about 70 ms.

ACKNOWLEDGMENTS

We thank Josef Sommer and Ingo Ahrns from Airbus Defense and Space for the good cooperation during the test campaign and for providing as with the LiDAR data.

REFERENCES

[BBR14] H. Benninghoff, T. Boge, and F. Rems. Autonomous navigation for on-orbit servicing. *KI-Kuenstliche Intelligenz*, 28(2):77–83, 2014.

[BRD13] C. Bonnal, J.-M. Ruault, and M.-C. Desjean. Active debris removal: Recent progress and current trends. *Acta Astronautica*, 85:51–60, 2013.

[BWMT10] T. Boge., T. Wimmer, O. Ma, and T. Tzschichholz. Epos - using robotics for rvd simulation of on-orbit servicing missions. In *AIAA Guidance, Navigation , and Control Conference*, Toronto, Ontario Canada, 2-5 August 2010.

[CH99] C.-S. Chen and Y.-P. Hung. Ransac-based darces: A new approach to fast automatic registration of partially overlapping range images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1229–1234, November 1999.

[dBCvKO08] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry - Algorithms and Applications*. Springer-Verlag Berlin Heidelberg, 3 edition, 2008.

[EKS08] A. Ellery, J. Kreisel, and B. Sommer. The case for robotic on-orbit servicing of spacecraft: Spacecraft reliability is a myth. *Acta Astronautica*, 63:632–648, 2008.

[ELPZ97] Y. Eldar, M. Lindenbaum, M. Porat, and Y. Y. Zeevi. The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6(9):1305–1315, 1997.

[KSA87] S. D. Blostein K. S. Arun, T. S. Huang. Least-squares fitting of two 3-d point sets. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9:698–700, 1987.

[NTA+06] M. Nimelman, J. Tripp, A. Allen, D. M. Hiemstra, and S. A. McDonald. Spaceborne scanning lidar system (ssls) upgrade path. In *Proc. SPIE*, volume 6201, 2006.

[RLB12] S. Ruel, T. Luu, and A. Berube. Space shuttle testing of the tridar 3d rendezvous and docking sensor. *Journal of Field Robotics*, 29(4):535–553, 2012. cited By (since 1996)2.

[Rus01] S. Rusinkiewicz. Efficient variants of the icp algorithm. In *Proc. of Third International Conference on 3D Digital Imaging and Modeling*, pages 145–152, Quebec City, Quebec, 28 May - 01 Jun 2001. IEEE.

[SLW+09] E. Stoll, J. Letschnik, U. Walter, J. Artigas, P. Kremer, C. Preusche, and G. Hirzinger. On-orbit servicing. *IEEE Robotics & Automation Magazine*, 16(4):29–33, December 2009.

[SMFF07] J. Salvi, C. Matabosch, D. Fofi, and J. Forest. A review of recent range image registration methods with accuracy evaluation. *Image Vision Comput.*, 25(5):578–596, May 2007.

[TB11] O. Ma T. Boge. Using advanced industrial robotics for spacecraft rendezvous and docking simulation. In *Proceedings - IEEE International Conference on Robotics and Automation*, Shanghai, China, 9-13 May 2011.

[TB12] M. Zebenay F. Rems T. Boge, H. Benninghoff. Using robotics for advanced rendezvous and docking simulation. In *Simulation and EGSE Facilities for Space Programmes*, Nordwijk, The Netherlands, 2012.