Fast High Resolution 3D Laser Scanning by Real-Time Object Tracking and Segmentation

Jens T. Thielemann*, Asbjørn Berge, Øystein Skotheim and Trine Kirkhus

Abstract—**This paper presents a real-time contour tracking and object segmentation algorithm for 3D range images. The algorithm is used to control a novel micro-mirror based imaging laser scanner, which provides a dynamic trade-off between resolution and frame rate. The micro-mirrors are controllable, enabling us to speed up acquisition significantly by only sampling on the object that is tracked and of interest. As the hardware is under development, we benchmark our algorithms on data from a SICK LMS100-10000 laser scanner mounted on a tilting platform. We find that objects are tracked and segmented well on pixel-level; that frame rate/resolution can be increased 3-4 times through our approach compared to scanners having static scan trajectories, and that the algorithm runs in 30 ms/image on a Intel Core i7 CPU using a single core.**

I. Introduction

Sensors providing high-quality, densely sampled 3D data are enabling technology for robots that interact with objects in their surroundings. Laser scanners are popular sensors for robots due to that they provide high-quality data with a reasonable data rate, also in adverse conditions like sunlight. The data quality and robustness is largely due to that laser range scanners acquire distance measurements by sequentially illuminating and measuring individual scene points.

To obtain an image, the point measured must be scanned across the scene. Today, most laser scanners do this by using a 1D mirror providing only a single (usually horizontal) line of data. By themselves, such laser scanners are unable to provide an image of their surroundings, which strongly limits the amount of scene interpretation that can be done. To obtain 3D images, some sort of tilting platform is employed which tilts the sensor such that the image can be built up one line at a time. This means that each image will take a few seconds to capture, usually too slow for real-time interaction with moving objects. By adjusting the vertical tilting speed of the laser scanner, it is possible to trade lower image resolution for a higher frame rate.

In this paper, we present a combined hardware and software concept providing high quality, high resolution imaging laser scanner data of objects that are automatically tracked by the sensor itself.

Our approach is based on a hardware concept of controllable micro-mirrors, combined with a fast time-of-flight range measurement unit. The range measurement unit provides data at a constant rate. By controlling the vertical speed of the mirror, we can dynamically make a tradeoff between frame rate and spatial resolution. This enables *foveation*, meaning that we acquire high quality data only on the object being tracked, possibly even with a higher frame rate than for a regular scanner.

To control these mirrors in real-time, we present a real-time tracking algorithm which is capable of both track and pixel-level segment objects based on a very rough initialization of the object contour. This tracking algorithm enables us to only capture data on the object of interest, thus speeding up acquisition. The high-quality, high frame-rate data provided by this concept can open up for better object interaction and recognition in dynamic scenes.

The main contributions of this paper are a real-time computationally efficient algorithm which tracks and segments object in 3D range data, and the application of this algorithm to control a foveating 3D laser scanner.

The rest of this paper is organized as follows: Section II describes related work. Section III provides a system overview and section IV details the actual tracking algorithm. Section V provides our experimental setup, which results are presented in section VI. We conclude the paper in section VII.

II. Related work

For this paper, related work encompasses both foveating sensors, as well object tracking algorithms for range images.

A. Foveating sensor

Existing foveating sensors are mainly realized by pixel grouping on sensors or macro hardware movements.

Sensor-based approaches include early foveating sensors [1] that mimic the human eye by using a log polar mapping of the imaging sensor area. Other approaches enabled dynamic grouping of sensor pixels [2], adaptively controlling which CMOS imaging sensor areas to read with high resolution.

Hardware movement methods use a pan/tilt unit to achieve foveation. Bimbo and Pernici [3] present a system that foveates by positioning a 2D camera with an external pan/tilt device. Similarly, a combination of range sensors to get active gazing control is presented in [4]. Here data from a 2D laser range finder is used detect obstacles, which in turn is used to pan/tilt an external time-of-flight camera. In [5] a laser scanner is mounted on motorized head to enable a roundly swinging motion to increase sampling density in predefined regions of interest.

* Research supported by EC Grant 248623 under FP7.

All authors are with SINTEF ICT, PB 124 Blindern, 0314 Oslo, Norway. E-mail: {jtt,trk,asbe,osk}@sintef.no.

By contrast, the micro-mirror based approach that we are developing does not need large sensor movements. By also including real-time tracking software, we also solve the problem of *where* to gaze, which previous approaches often have left unresolved.

B. *Tracking in range data:*

Object tracking in range images tend to focus on template alignment approaches, such as the Iterative Closest Points algorithm (ICP) [6], an iterative approach for aligning a prior 3D model. The ICP algorithm is not sufficiently robust for use alone, due to bad convergence properties. Particle filters has been used in conjunction with ICP alignment, both in registration and tracking approaches [7]. Modern software libraries for point clouds follow this strategy [8], using GPUs to achieve real-time performance. Active contours using filtered range images and visual images is discussed in [9], which also uses a particle filter using depth information for target positioning. Other real-time contour based approaches for object tracking and segmentation [10] do not provide the necessary features for working on range images.

Our approach works on range data alone without requiring a prior model of the object to be tracked. Instead, the tracker is initialized with a rough contour of the object, which is updated and refined on pixel-level by the tracker automatically. Furthermore, we do not need GPUs to achieve real-time performance.

III. System Overview

The foveating 3D sensor under development is built on three key technologies: controllable micro-mirrors, 3D time-of-flight hardware and a 3D tracking algorithms that is used to control the scanning trajectory.

The controllable micro-mirrors and time-of-flight hardware enable the scene to be sampled with varying spatial and temporal resolution. The tracking algorithm employed enables the control of spatial and temporal resolution to happen in an intuitive fashion. Combined, this sensor concept can provide significantly better data than existing sensors. This is partly due to the measurement principle itself, and partly due to the sensor's foveation capability.

A. *Micro-mirrors and 3D time-of-flight hardware*

The hardware is an adaptive 3D sensor that uses lightweight, robust micro-mechanical scanning elements for flexible two-dimensional beam-steering of fast single point time-of-flight distance measurements. Figure 1 outlines the hardware setup.

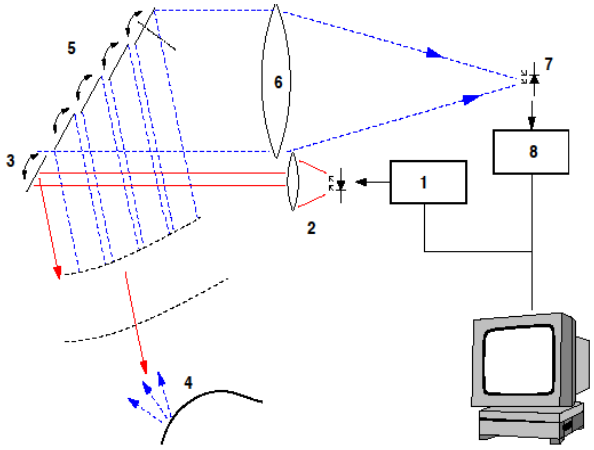


Figure 1. Hardware concept of the 3D sensor. The emitted modulated (1) laser beam (2) is scanned by mirrors (3) on the target. The light reflected from the measured surface (4) is collected by mirrors (5), reaches the single element detector (7) via collecting optics (6). The distance to the target point follows from the traveling time of received (8) with respect to the emitted signal (1) [11].

The pulsed laser beam and time-of-flight hardware will be capable of measuring up to one million 3D points per second. To build up images from these single measurements, the laser beam must be swept across the scene. This is accomplished by using controllable micro-mirrors that allow for very precise and rapid control of the beam direction.

Current laser scanners use large mirrors that move in fixed patterns (e.g., constant oscillation or rotation). This means that an extraordinary amount of power is required to enable rapid shifts of scanning patterns. The use of novel quasi-static MEMS scanning mirrors [12] enables the system to provide foveation – i.e., rapidly controlling the beam direction and thus adjusting the spatial and temporal resolution of the acquired data.

B. *3D foveation*

3D foveation enables the sensor to go beyond simply providing data with high spatial or temporal resolution – it allows the sensor to adjust the resolution according to the scene.

Representing 3D data as range images ease later analysis of the data on the robot. In principle, the use of controllable micro-mirrors allows arbitrary scanning patterns. The scanning patterns are chosen to be raster scans to facilitate the conversion into range images. Foveation by increasing resolution is achieved by adjusting the vertical mirror speed.

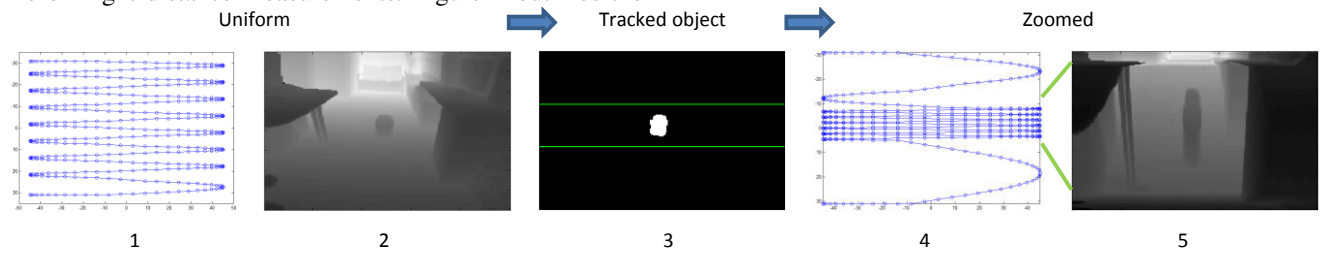


Figure 2. Time series showing how the foveation software uses saliency analysis of a uniform image from the scene to capture an image zoomed in at the region-of-interest. A sinusoidal mirror plan (1) is used to capture the uniform intensity (2) and range images (intensity image is shown to ease visual scene interpretation). The object of interest is detected in the the captured data (3). A mirror plan for zoomed data acquisition (4) is computed from the sampling density function. This results in zoomed intensity (5) and range images, where the object-of-interest appears magnified due to the increased sampling density

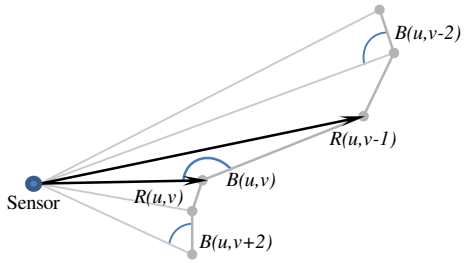


Figure 3. Illustration of bearing angle $B(u,v)$. Sensor measures neighbouring pixels $R(u,v)$ and $R(u,v-1)$. The bearing angle allows depth-invariant classification of depth transitions.

We have chosen a foveation regime as outlined in Figure 2. In this regime, every second frame is acquired as a normal range image with uniform sampling. This range image is analyzed by our tracking algorithm to determine the current silhouette of the object-of-interest. Based on this object silhouette, the alternating second frame is ”zoomed in”, i.e., acquired with increased spatial resolution on the tracked object. The number of data points is equal for both frames to maintain frame rate, and the full field-of-view is kept. This real-time feedback system is detailed further in [13].

IV. Tracking Algorithm

Our tracking approach consists of three distinct steps, each one adapted for working on range images in real-time. The object to be tracked is represented as an object silhouette. This silhouette is tracked and updated according to camera/object motion. This is done in a three step process consisting of edge detection (section A), rigid tracking (section B) and contour adaption (section C). Section D outlines how we use the object silhouette to control the sensor and provide high resolution imagery of the tracked object.

A. Edge detection

As basis for later tracking and segmentation, we rapidly calculate the depth transitions in the scene. This is done in two steps: Detection through bearing angles followed by separation and normalization. The range image $R_t(u,v)$, containing the range to each point in the pixel with coordinate (u,v) for time t, is used as input for the algorithm.

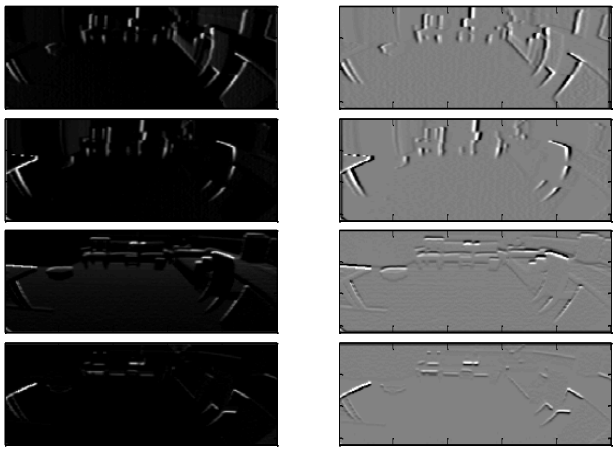
Edge detection through bearing angles: Harati [14] introduce the notion of "bearing angle" for detecting edges in range images. The "bearing angle" indicates the angle between the measurement beam and the surface (Figure 3). For real depth edges, the bearing angle will be close to 0 or 180 degrees. This measurement can be used to create a more robust method for edge detection than simply detect the depth of edge transitions, followed by a hard threshold.

Harati [14] calculate these explicitly using trigonometric formulas. To gain speed, we instead calculate the approximated bearing angle measure in each pixel position (u,v) for the u direction at the time t as

$$B_{t,u}(u,v) = \frac{\Delta_u R_t(u,v)}{R_t(u,v)}$$

where $\Delta_u R_t(u,v)$ is the detected edge strength in u direction, and $R_t(u,v)$ is the measured range. The

(a) Original range image



(b) Edge images E_t (c) Zero-crossing images Z_t

Figure 4. Result of edge detection on an example range image (a). Edge images (b) and zero-crossing images (c) shown in top-down order left, right, top and bottom edges.

calculation of $\Delta_u R_t(u,v)$ is done using a Sobel-operator. This provides a rapid near depth-invariant measure for edge strength. Approximated bearing angle in v direction, $B_{t,v}(u,v)$, is calculated using the same procedure but with the edge filter transposed. This provides two images $B_{t,u}(u,v)$ and $B_{t,v}(u,v)$ providing edge magnitude rowwise and columnwise.

Edge separation and normalization: We focus on tracking foreground objects, meaning that the background always will be behind the object. This means that we expect that the sign of the detected edge magnitudes to align with the edge direction on the object itself. On the left edge of the object we expect the edge magnitude to be positive, on the right side negative. Similar assumptions are made for top/down of the object.

To better preserve edges prior to smoothing, we separate $B_{t,u}(u,v)$ and $B_{t,v}(u,v)$ according to the sign of the edge magnitude such that we get a bearing angle measurement A_t that is directional

$$\begin{aligned} A_t(u,v,L) &= \min(1,\max(0,B_{t,v}(u,v)/Q)) \\ A_t(u,v,R) &= \min(1,\max(0,-B_{t,v}(u,v)/Q)) \\ A_t(u,v,U) &= \min(1,\max(0,B_{t,u}(u,v)/Q)) \\ A_t(u,v,D) &= \min(1,\max(0,-B_{t,u}(u,v)/Q)) \end{aligned}$$

where $A_t(u,v,L), A_t(u,v,R), A_t(u,v,U), A_t(u,v,D)$ are respectively Left, Right, Up and Down edge magnitude images, and Q is a chosen upper limit for normalization.

Each of these four images $A_t(u,v,\cdot)$ are separately smoothed with a 1D Gaussian filter providing us edge magnitude images $E_t(u,v,\cdot)$ as shown in Figure 4b. $C_t(u,v,L)$ and $C_t(u,v,R)$ are filtered with a horizontal 1D Gauss filter, for $C_t(u,v,U)$ and $C_t(u,v,D)$ a vertical filter is used. Due to that the edges have previously been separated according to direction, they are preserved better during smoothing.

The E_t images are then convolved with a 1D filter providing images $Z(u,v,\cdot)$ for edge detection through zero-crossings (Figure 4c). In effect, we have found that these $Z(u,v,\cdot)$ images are also well suited for detecting sudden changes in normal direction.

B. Rigid tracking

It is reasonable to assume that the object does not change shape significantly from frame to frame. A fully rigid tracking of the object from the previous frame to the next can thus be done. This is done by minimizing the following cost function with respect to a movement (m,n) in pixels from the previous frame, in the (u,v) direction:

$$\begin{aligned}&\epsilon(R_{t-1}, R_t, E_t, Z_t, m, n)\\&= \frac{1}{N_C}\sum_{c_i \in C} \epsilon_{contour}(E_t, c, m, n)\\&+ \frac{\alpha}{N_O}\sum_{o_j \in O} \epsilon_{range}(R_t, R_{t-1}, o, m, n)\end{aligned}$$

where C is the contour of the object in the previous frame R_{t-1}, and O is the pixels covered by the object in previous frame. N_C and N_O indicate the number of points in C and O. $\epsilon_{contour}$and ϵ_{range} are cost functions described later, and α is a scaling factor. The contour C may be unordered, which allows for rapid updates when maintaining C through the algorithm. Furthermore, C may belong to multiple disjoint objects that are moving at the same speed.

The cost function chosen ensures that the object silhouette rigidly snaps to object depth edges similar to the one from the previous image, while also ensuring that the underlying range data match between the previous and current frame.

Contour cost term: The contour part $\epsilon_{contour}$ of the cost function is determined by first extracting the object contour on pixel-level, and determining with four-connectivity which neighboring pixels are background. The relative placement of the contour pixel and the neighboring background pixel gives a direction d_i. Each contour pixel c_i is thus represented as its pixel coordinates (u_i, v_i) plus the direction d_i, i being index. d_i may be either L, R, U or D. This is used to choose which $E_t(u,v,\cdot)$ image to perform edge strength lookup in.

If a contour pixel has multiple background pixel neighbors, the contour pixel is repeated accordingly in C. The tracker then uses the following formula for calculating the contour alignment per contour pixel:

$$\epsilon_{contour}(E_t, c_i, m, n) = -E_t(u_i + m, v_i + n, d_i)$$

Due to the previous smoothing of E_t, the convergence basin of $\epsilon_{contour}$ is widened. The contour term is minimal if the contour precisely aligns with the object's depth edges.

Range cost term: Simultaneously with the extraction of the contour, all foreground object pixels $o_j = (u_j, v_j)$ are enumerated into O. These are used as basis for a template matching, by calculating the range cost term as

$$\begin{aligned}&\epsilon_{range}(R_t, R_{t-1}, o_j, m, n)\\&= \left\| R_{t-1}(u_j, v_j) - R_t(u_j + m, v_j + n) \right\|\end{aligned}$$

The range cost term is minimal if the objects precisely align over each other.

Actual optimization: ϵ is optimized with respect to (m,n) using the simplex direct search method (Nelder-Mead), initialized with a step size of 1 pixel. This provides an estimate of object movement in pixels from the previous frame to the current frame.

C. Adaption of object's contour

The third part of the tracking adapts the current contour of the object to any changes in the object contour due to change of viewpoint or object shape.

First, the object silhouette is moved according to the (m,n) established in the previous rigid tracking, and the contour C of the moved silhouette is extracted.

The goal of the algorithm is to adapt the contour C such that $Z_t(\cdot,\cdot,d) > 0$ on the contour, and $Z_t(\cdot,\cdot,d) < 0$ on the immediate outside of the contour, where d chosen according to the contour's local direction.

Before running the actual contour update, we establish $d_{min}$and d_{max}as the minimum and maximum range of accepted new object pixels. These are calculated by estimating minimum and maximum range of the object in the previous range image R_{t-1} plus a pre-defined tolerance.

To perform fast pixel-level contour updates, a state-image S is maintained with equal resolution to the original range image. Each pixel contains the state of each pixel, which is defined as follows to enable 8-bit state pixel storage:

$$S(u,v) = \begin{cases} 0 \dots \mathrm{V} & \text{if background} \\ 254 & \text{if contour} \\ 255 & \text{if foreground object} \end{cases}$$

The image $S(u,v)$ is initialized based on the object silhouette and the definition above, setting background pixels to 0. Furthermore, a variable V is set to 0, indicating the current pixel value in S of the background. V is incremented with each iteration over all C. Pixels $S(u,v) \leq V$ are considered to be background.

After these initial steps, each contour pixel $c_i = (u_i, v_i)$ in C is processed according to the algorithm described in Figure 5, and illustrated in Figure 6.

For each contour pixel, we check whether the four-connected neighbours are background pixels. The position of

- For each contour pixel $c_i \in C$ do:
  - For each pixel $P = (u_p, v_p)$ of the four-connected neighbours of $c_i = (u_i, v_i)$, check whether P is background by checking that $S(u_p, v_p) \leq V$. If so:
    - Calculate the direction d that relates c_i and P as $d = P - c_i$, and classify it as c_d into one of L, R, U, D.
    - If $Z_t(u_i, v_i, c_d) < 0$ shrink the object by one pixel by removing (u_i, v_i) from the object:
      - Set $S(u_i, v_i) = V + 1$, making it background. By not setting it to V (or 0), later processed contour pixels in this iteration will not consider this pixel background, ensuring that the order each contour pixel is processed makes no difference.
      - Set $c_i' = c_i - c_d = (u_i', v_i')$, effectively moving the contour pixel one pixel inwards
      - If $S(u_i', v_i') = 255$, then convert this foreground pixel into a contour pixel:
        - Set $S(u_i', v_i') = 254$
        - Replace the entry c_i in C with c_i'
      - Else:
        - Remove c_i from the C, as c_i' already either is contour or background
      - For each of the four-connected neighbours $N = (u_n, v_n)$ of c_i':
        - If $S(u_n, v_n) = 255$, set $S(u_n, v_n) = 254$ and insert N into C. This ensures that newly exposed foreground pixels becomes contour pixels.
    - If $Z_t(u_p, v_p, c_d) > 0$, expand the object by one pixel:
      - Check that $d_{min} \leq R_t(p_i, p_j) \leq d_{max}$. Abort expansion if this is not fulfilled.
      - Insert P into C
      - Set $S(p_i, p_j) = 254$ to indicate that it is contour
  - If c_i has no background pixels as neighbours, it is removed from C, and $S(u_i, v_i)$ is set to 255.
- Increase V by 1.

Figure 5. One iteration of the algorithm for adjusting contour according to object boundaries.

the found background pixels determines which $Z_t(u, v, \cdot)$ are used to update the contour in each direction.

The update will convert some contour pixels into background pixels. However, to ensure that C can be processed in an unordered fashion, it is important that these newly created "background" pixels are not considered to be background pixel in the current iteration over C. This is done by setting such "background" pixels in $S(u, v)$ to $V + 1$.

In the subsequent iteration, V is increased by one. As pixels $S(u, v) \leq V$ are considered background, fresh background pixels from the previous iteration will now be

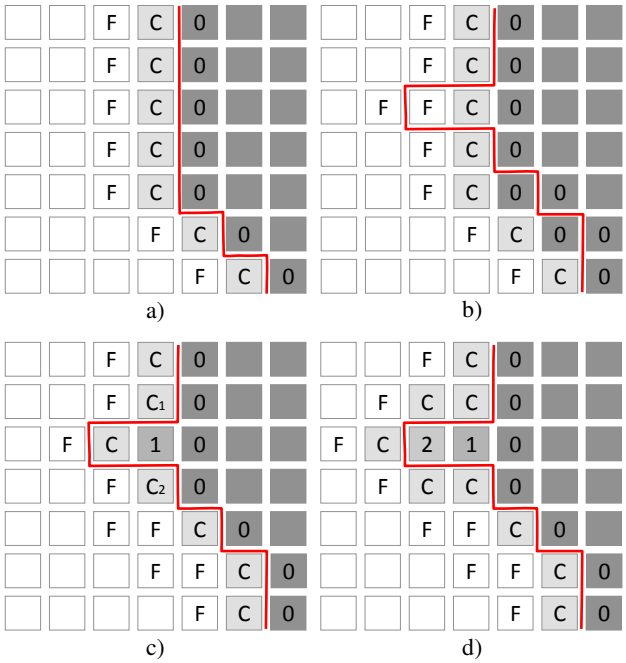


Figure 6. Illustration of the adaption of the object contour. Red lines indicate the contour to be fitted, and colors show the value of pixels in $S(u, v)$. F indicates foreground, C contour, and 0-2 is background. a) Initial contour and previous image. b)-d) adaption to the changed contour in current image through iterations, b) before iteration (c) first iteration ($V = 0$) d) second iteration ($V = 1$). Lower right of contour is grown in a straight-forward manner. Top left concavity requires shrinking of contour. In c) the contour is moved one step in, creating a new background pixel which is initialized to $V + 1 = 1$. Regardless of processing order, C_1 and C_2 will not consider the new background pixel to be background, as the criteria for background pixels is $S(u, v) \leq V$ and $V = 0$. d) shows final adaption.

considered background. Incrementing V is the element in the algorithm that allows C to be unordered and belonging to multiple disjoint objects.

The algorithm is repeated a suitable number of iterations (typically 5). This allows each contour pixel to move up to 5 pixels. The final object silhouette is extracted as $B_s = S(u, v) \geq 254$ which we subsequently normalize by performing morphological opening. The algorithm has been implemented in a mixture of Matlab and C.

D. *Using the Tracking Result for Foveation*

The binary silhouette generated by the tracker can directly be employed for making the scanner foveate. A vertical region is picked out which surrounds the detected object. This region is sent to the laser scanner for acquisition of the next frame. To be able to capture the object even in the case where unexpected motion happens, a somewhat wider region than the object silhouette is chosen. This enables us to cope with the inherent latency induced by using a previously acquired image to predict where the next region-of-interest is located. In addition, mirror inertia places constraints on the minimum size of regions-of-interest.

V. Experiments

We tested the tracking algorithm and foveation mechanism on real data of moving objects, where the sensor itself is either static or moving.

Figure 7. Tracking results for robot moving relative to the sensor with different initial contour. User has clicked on a range pixel on the robot, and all pixels closer than r meters is initial contour, as shown in frame 1. Top: $r = 0.1$. Bottom: $r = 0.3$. White line shows tracked contour. Both camera and robot is moving.

As the actual hardware is not yet available, we have used a line-scanning SICK LMS100-10000 laser scanner placed on a tilt unit to capture test data. The tilt unit enabled building a 2D raster image of range measurements. The data acquisition took roughly 30 seconds per image, so stop-motion animation techniques were used to acquire data sequences of moving objects. The foveation was done using a hardware simulator. The data are summarized in Table I.

Table I: DESCRIPTION OF SIMULATED DATA SETS

Data set	Scene description	Camera motion
0	Robot moving from left to right	Static
1	Robot moving towards the 3D sensor	Static
2	Robot moving from left to right	Moving
3	Robot moving towards the 3D sensor.	Moving
4	Several objects moving in different ways (small robot, rotating box, …)	Moving

Based on these data, we benchmark the tracker's performance; quantify the resulting performance on 3D data quality enhancement; and the tracker's real-time performance.

A. *Assessing tracker performance*

We have used the tracker algorithm to track objects in the pre-recorded sequences. The tracker has been initialized by the user clicking on a single range pixel in the first range image of the sequence, and range pixels being less than $r = 0.3$ meters from the selected pixel have been selected as the initial silhouette of the image. The manual initialization could easily be replaced by an automatic initialization from a robot at a later stage. This silhouette is subsequently updated and refined.

The tracker is benchmarked by comparing the silhouette produced by the algorithm to a manual ground truth for the recorded sequences. The ground truth is created by indicating the moving object as using a binary mask per sequence frame. To provide quantification of the tracking and segmentation, this is viewed as a segmentation problem. We estimate how many ground truth object pixels that are classified as object pixels (*object segmentation),* and how many ground truth background pixels that are misclassified as object pixels (*background segmentation)*. We normalize these numbers respectively by the number of object and background pixels and report them in percent.

To measure the effect of the separation of edge directions before smoothing as outlined in section IV.D we have performed tracking of objects with and without separation and compared results.

In addition, we measure the average time per frame for the tracking algorithm when run on a 2.0 GHz Intel Core i7.

B. *Assessing foveation performance*

In our foveation regime, the benefit of foveation is to provide high spatial resolution in detected regions-of-interest, while maintaining the overall frame rate. A non-foveating system that uses the same underlying hardware would by comparison have to provide the same high spatial resolution in the whole scene. Because of the constant sampling rate, this would result in a net lower frame rate for the system. We therefore quantify foveation as the increase in sensor frame rate compared to a non-foveating system.

The chosen foveation regime keeps a constant number of data points per range image. Therefore, the maximum achievable frame rate increase is limited by the field-of-view covered by the object-of-interest. As a result, the foveated system becomes equal to a non-foveated system if the object covers the full field-of-view. We report on the average frame rate increase enabled by the foveation system both in absolute numbers compared to the sensor without foveation, and relative to the maximum achievable frame rate increase.

It is furthermore crucial for a foveating system that it actually foveates on the object of interest. We therefore also measure how much of the ground truth object silhouette that is covered by the selected foveated region.

VI. RESULTS AND DISCUSSION

Figure 7 show some example results of applying the tracker to a sequence with moving camera and tracking a non-static object. We see that the object is tracked and segmented well from the surroundings even with a poor starting silhouette. After a few frames, the contour converges to the same contour regardless of starting contour.

Table II provides the numerical results for the tracker applied to different sequences. We see that the tracker is able to well keep track of the object, with little spillover to the background. The time consumption is approximately 30 ms for all images, utilizing a single core on an Intel Core i7 CPU, which is meets the system's realtime demands.

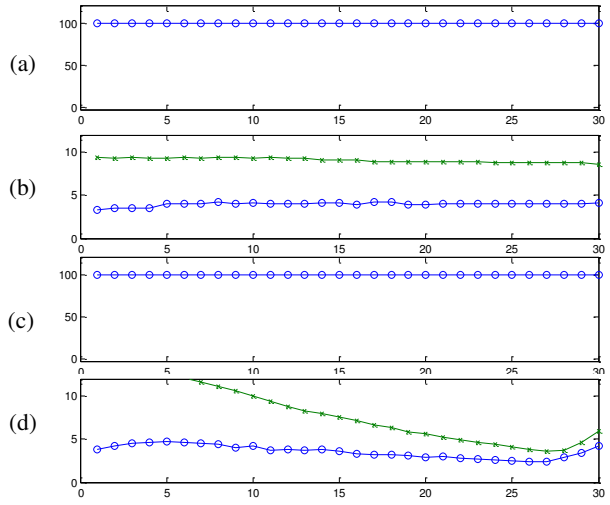


Figure 8. Performance indicators per frame for sequence 0 (a,b) and 3 (c,d). X-axis all graphs: Frame number. (a,c): Tracker's object coverage in %. (b,d): Frame rate increase; crosses: max possible, circle: achieved.

TABLE II. RESULTS ON TRACKING AND REAL-TIME PERFORMANCE

Data set	Object segmentation (%)		Background segmentation (%)		Time consumption (ms)	
	mean	*Std*	*mean*	*std*	*mean*	*std*
0	78.6	3.3	0.0	0.0	28.9	1.6
1	90.3	3.8	0.1	0.4	29.1	1.9
2	89.3	3.6	0.0	0.0	29.2	1.1
3	92.4	3.7	0.0	0.0	28.9	1.0
4 (rotating box)	79.7	9.0	0.0	0.0	29.2	1.9
4 (robot)	95.3	1.6	0.0	0.0	28.9	0.8

Table III shows the foveation performance. We achieve a three- to fourfold increase of frame rate/resolution, and are unable to reach the maximum possible frame rate increase. This is due the fact to that the sensor foveates on a somewhat larger scene region than the object itself, as indicated in section IV.D.

The average numbers in Table III are broken up per frame in Figure 8 for some selected sequences. We see that the tracker robustly track and provide foveated data on the object of interest. For the sequence where the camera and object gets gradually closer to each other, we can see that the maximum achievable frame rate increase of foveation degrades as the object gets closer.

For objects that have less defined edges (or more internal structure), we found that the edge separation improved results. Figure 9 shows an example of an object (a plant) where the tracking result is improved due to edge separation.

VII. CONCLUSION

This paper proposes a real-time tracking and object segmentation algorithm for 3D range images, and applies it to for 3D foveation. The algorithm is computationally

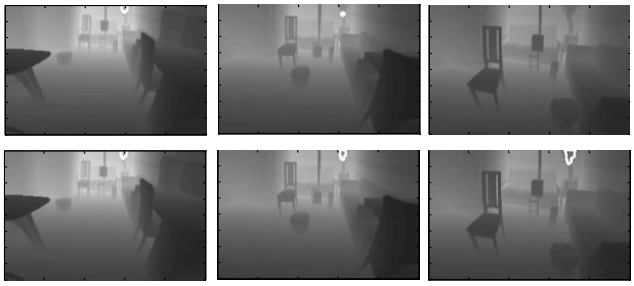


Figure 9. Tracking based on edge images where edges have been separated (bottom) and not separated (top) in sequence 4. White contour indicates object contour (pointed to by arrows). Same initial silhouette used in both cases. Without edge separation, the tracker looses the object after approximately 45 frames.

TABLE III: RESULTS ON FOVEATION

Data set	Object coverage (%)		Frame rate increase		Max frame rate increase		Achieved frame rate increase vs. max possible
	mean	*std*	*mean*	*std*	*mean*	*Std*	*mean*
0	100.0	0.0	4.0	0.2	9.1	0.3	44.4
1	100.0	0.0	3.4	0.6	7.1	2.5	51.0
2	99.5	2.2	3.2	0.5	5.7	1.8	59.7
3	100.0	0.0	3.5	0.8	7.6	3.1	50.8
4 (rotating box)	100.0	0.0	3.7	0.4	7.1	1.0	52.2
4 (robot)	100.0	0.0	3.0	0.4	5.9	1.8	53.4

inexpensive, real-time capable and requires no other initialization than a rough initial object silhouette.

The algorithm leverages several important properties of 3D laser scan data to achieve real-time performance, primarily that edges and normal transitions define well object boundaries for foreground objects.

We have shown that this enables robust object tracking and segmentation for such objects. Furthermore, by employing the tracking algorithm to control the laser scanner, we are able to increase the scanning resolution 3-4 times for the object-of-interest without using more time per frame; 45-60% of the maximum achievable gain. The gains from improved tracking and hence higher resolution in regions of interest in range data can enable precise later interaction by robots.

Future work will focus on demonstrating the laser scanner (and the internal tracking algorithm) capabilities for visual servoing and grasping in more complex scenarios.

ACKNOWLEDGMENT

The authors thank Peter Einramhof and Robert Schwarz (Technische Universität Wien, Austria) for recording the data used for experiments, and both them and the EU TACO project consortium for fruitful discussions.

VIII. REFERENCES

[1] J. Boluda, F. Pardo, T. Kayser, J. Pérez and J. Pelechano, "A new foveated space-variant camera for robotic applications," in *Electronics, Circuits, and Systems, 1996. ICECS'96., Proceedings of the Third IEEE International Conference on*, 1996.

[2] D. Bailey and C. Bouganis, "Reconfigurable foveated active vision system," in *Sensing Technology, 2008. ICST 2008. 3rd International Conference on*, 2008.

[3] A. D. Bimbo and F. Pernici, "Towards on-line saccade planning for high-resolution image sensing," *Pattern Recogn. Lett.,* vol. 27, pp. 1826-1834, 2006.

[4] D. Droeschel, D. Holz, J. Stückler and S. Behnke, "Using time-of-flight cameras with active gaze control for 3D collision avoidance," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010.

[5] T. Yoshida, K. Irie, E. Koyanagi and M. Tomono, "3D laser scanner with gazing ability," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011.

[6] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*, 2001.

[7] M. Ben Ghorbel, M. Baklouti and S. Couvet, "3D head pose estimation and tracking using particle filtering and ICP algorithm," *Articulated Motion and Deformable Objects,* pp. 224--237, 2010.

[8] R. Ueda, *Tracking 3D objects with Point Cloud Library,* 2012.

[9] J. Lee, S. Lankton and A. Tannenbaum, "Object Tracking and Target Reacquisition Based on 3-D Range Data for Moving Vehicles," *IEEE Trans. on Image Proc.,* vol. 20, no. 10, pp. 2912-2924, 2011.

[10] Y. Shi and W. Karl, "Real-time tracking using level sets," in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, 2005.

[11] T. Sandner, T. Grasshoff and M. &. S. H. Wildenhain, "Synchronized micro scanner array for large aperature receiver optics of LIDAR systems," in *SPIE 7594*, 2010.

[12] D. Jung, D. Kallweit, T. Sandner, H. Conrad, H. Schenk and H. Lakner, "Fabrication of 3D comb drive microscanners by mechanically induced permanent displacement," in *SPIE 7208*, 2009.

[13] G. Breivik, J. Thielemann, A. Berge and O. Skotheim, "A motion based real-time foveation control loop for rapid and relevant 3D laser scanning," in *Computer Vision and Pattern Recognition Workshops (CVPRW), 2011 IEEE Computer Society Conference on*, 2011.

[14] A. Harati, S. Gächter and R. Siegwart, "Fast range image segmentation for indoor 3D-SLAM," in *6th IFAC Symp. on Intelligent Autonomous Vehicles*, 2006.