 DOI: 10.3217/978-3-85125-528-7-14

Real-time tracking of rigid objects using depth data

Sharath Chandra Akkaladevi[1, 2], Martin Ankerl[1], Gerald Fritz[1], Andreas Pichler[1]

[1]Department of Robotics and Assistive Systems
Profactor GmbH, Im Stadtgut A2, Steyr-Gleink, 4407, Austria
{firstname.lastname}@profactor.at

[2]Institute of Networked and Embedded Systems
Alpen-Adria-Universität Klagenfurt, Austria

Abstract
In this paper, a robust, real-time object tracking approach is presented. The approach relies only on depth data to track objects in a dynamic environment and uses random-forest based learning to deal with problems like object occlusion and clutter. We show that the relation between object motion and the corresponding change in its 3D point cloud data can be learned using only 6 random forests. A framework that unites object pose estimation and object pose tracking to efficiently track objects in 3D space is presented. The approach is robust against occlusions in tracking objects and is capable of real-time performance with 1.7ms per frame. The experimental evaluations demonstrate the performance of the approach against robustness, accuracy and speed and compare the approach quantitatively with the state of the art.

1. Introduction

Object tracking has been widely researched in the vision community over the recent past and many methods are proposed in literature to track objects [6]. Until the last decade the methods mainly considered 2D image data as input and in some cases stereo vision and served applications like surveillance, military use, security and industrial automation. However, 2D image data only captures the 3D projection into two dimensions and is sensitive to illumination changes. With recent development of RGB-D devices like Kinect, researchers all over the world are exploiting depth data for object recognition and tracking [7]. Tracking can be defined as the problem of estimating the trajectory (6 DOF – 3 translations, 3 rotation parameters) of an object in the 3D image plane as it moves around a scene. Though there has been a lot of work in tracking humans using RGB-D devices [8], not much work is done in the field of tracking objects that could be used in industrial settings which often have real-time requirements.
Object tracking in general is a challenging problem. Tracking objects becomes difficult due to abrupt object motions, object to object occlusions, clutter, camera motion and noisy sensor data. When considering its application in industrial settings the problem of designing a successful tracking algorithm becomes even more difficult. This is due to the requirement of higher levels of robustness, accuracy and speed. Also, industrial objects tend to have little texture. In this paper, we describe an approach for real-time tracking of objects [12] that aims to answer these challenges. The main contribution of this paper is the extended evaluation of the work in [12] and its comparison with the state of the art approaches.

Inspired by [5] we describe a fast and accurate 3D object tracking algorithm for rigid objects. The proposed approach is model-based, uses only depth data and achieves very good accuracy utilizing a framework that combines object localization and object tracking.

2. State of the Art

With the introduction of RGB-D sensors like Kinect, various approaches for object tracking in 3D were proposed, which ranged from tracking humans [10], hand tracking [8], and tracking rigid and non-rigid objects. For a better comparison of our approach with the state of the art, the scope of this section is limited to approaches that focus on frame-to-frame tracking of rigid objects using RGB-D data. The proposed approaches can be broadly classified into two categories: a) approaches that are based on 3D models and b) approaches that do not assume pre-defined object models.
For example, an approach that does not rely on prior knowledge of the target object representation is described in [14]. The approach uses adaptive Gaussian Mixture Models (GMM) to represent multiple objects that move independently. The object model is updated incrementally at each time instant with the help of the feedback results from the robust tracking process. To correct falsely detected objects in presence of occlusions and various types of interactions among multiple objects, an approach that exploits component-level spatiotemporal association is proposed in [10]. However, the approaches of "individuation-by-feature" [14] and "individuation-by-location" [10] require high computation time to learn each object model at every time instant and would exponentially increase with the number of objects and their spatial relations. Moreover, in an industrial environment which involves human actions, situations keep changing every time instant. To achieve robustness and computational efficiency in such scenarios, applying one individuation method is not sufficient. To alleviate this problem, an approach that determines individuation strategy (by location and/or by feature) depending on the object situation is proposed in [15]. The main assumption of the approach is that falsely segmented objects can be detected and rectified using both location and position information. It also assumes that objects do not change substantially in terms of shape or position from one frame to the other. A probabilistic framework for simultaneous tracking and reconstruction of rigid 3D objects using RGB-D sensor is proposed by [7], where the probabilistic method is used to statistically determine occlusions. Intensity images are used to model appearance of an object while modeling occlusions.
With the availability of reliable, fast and simple object reconstruction solutions like ReconstructMe[1], 3D object models can be obtained in real-time. A popular approach for model-based object tracking is based on the particle filters [10][18]. For example, the authors in [4] propose a 3D model-based visual tracking approach using edge and keypoint features in a particle filtering framework. This approach does not assume the initial pose of the object. It uses given 2D-3D keypoint correspondences to calculate a set of possible pose hypothesis of the object. Once the intial pose is estimated, edge points are used to track movement of the object from frame-to-frame. This approach is extended by [5] where an RGB-D object tracking method using a particle filter on GPU is proposed.
Another popular method for 3D object tracking is the Iterative Closest Point (ICP) approach which has many variants [16]. The algorithm uses a set of initial parameters and refines them iteratively to reach a set of optimal parameters by minimizing the object function. This approach has problems in dealing with occlusions and object clutter, which result in a local-minimum. To overcome this problem, a model-based learning approach is proposed in [18]. This approach learns the relation between the parameters that induce object' motion and the change they induce on the 3D point cloud using random forests. In order to track the object in motion, the change in the 3D depth data

[1] ReconstructMe http://reconstructme.net

is used to predict the parameters of this motion. The advantage of using random forests is that it is a collection of trees that learn and predict independently, even when some input data is affected due to occlusions, other trees can still provide good predictions. In order to track objects in different views, [18] trains a random forest for multiple views of the object that leads to a high computational effort. Moreover, the approach is not suitable for tracking symmetrical objects as the multiple-pose hypotheses are averaged and this leads to erroneous tracking of symmetrical objects. An offline learning based approach with known 3D object models based on particle filters is proposed in [9]. In [20], the authors propose a learning based approach inspired by [18] with reduced computational cost and improved occlusion handling capability.
In the proposed approach, we make the following contributions: a) we argue that it is sufficient to train only 6 random forests, to learn the relation between object motion and its corresponding change in 3D point cloud data, which in turn reduces the computational complexity b) dealing with symmetrical and non-symmetrical objects and c) a framework that is capable of tracking objects in presence of partial occlusions. A quantitative comparison is also carried out in this paper that uses synthetic data (that includes ground truth) provided by [5] to compare our approach against the state of the art.

3. Method

This section illustrates the proposed approach for localizing and tracking 3D objects with high performance and accuracy. First we describe the global localization algorithm RANGO, followed by the local tracking algorithm. Then, we illustrate how both components are combined into the full tracking framework

3.1. RANGO – RANdomized Global Object localization

RANGO is an algorithm for 3D object localization. It is based on a random sampling algorithm (RANSAC) described in [1][3] with several performance and robustness improvements, allowing a very fast detection rate when compared to the registration approach proposed in [7]. Its main contribution is the replacement of K-nearest neighborhood search for inlier detection with a probabilistic grid based approach. Thus the time complexity for the evaluation of a hypothesis (acceptance function) is reduced from $O(n * log(m))$ where n is the number of model points, m denotes the number of points in the scene, to $O(n)$. Additionally, the evaluation of the number of model points that fit the hypothesis is stopped early when the probability of finding a good match is too low.
Sparse 3D Voxel Grid. Each 3D point of a scene is approximated into a sparse axis aligned 3D grid. Each voxel of this grid is defined by a (x, y, z) tuple where x, y, z are (integer) coordinates for the voxel location. In RANGO this (x, y, z) position is hashed into a single 32bit number which is used as an index in a hash table. Due to hashing collisions it is possible that two different points hash to the same voxel even though their position is unrelated, but the probability is low enough that it is not a problem for our use case. This 3D voxel grid is then used for fast verification of candidate transformations.
To evaluate a transform matrix, we iterate over a set of sample points of the model and transform them into the scene. Each sample point is hashed into the 3D voxel grid containing scene points. If the hashed voxel is filled with a point and has a similar normal vector orientation as the model point, we count that as an inlier. This verification method has a complexity of $O(m)$ where m is the number of sample points. This verification is only approximate as it is possible to miss a neighboring sampling point because we only lookup the voxel the sample point hashes to, ignoring neighboring

voxels. A kd-tree would allow for exact nearest neighbor queries, but would have $O(m * log(n))$ complexity. The speedup achieved by the $O(m)$ verification allows us to evaluate more candidate transformations at the same time to boost accuracy.

Filtering Candidate Solutions. After the random sampling and matching process is over, the candidate solutions are filtered, since it is likely that multiple similar solutions have been found. In [3] a pose clustering approach is used. The pose clustering combines multiple similar poses to find an average position from these candidate solutions. This approach falls short for symmetric objects. E.g. a sphere where the reference frame is off center will result in many different potential poses, but each pose will have a completely different translation and rotation. In RANGO, we have replaced the clustering approach with a filtering approach. All candidate solutions are sorted by the number of inliers, highest number first. Iteratively, each solution is re-checked if it meets a given inlier threshold, and if it does, all scene voxels that were used in this inlier check are removed from the 3D voxel grid. This way only the best fitting candidate solution for a potential pose is used, while it is still possible to find multiple instances of the same object in the scene data. This approach works well for both complex and symmetrical objects.

3.2. Multi-Forest Tracking

Our multi-forest tracking approach is a variation of the multi-forest tracking algorithm described in [18]. Our modification to this algorithms retains the performance characteristic of [18] while having a significantly lower memory and training overhead, which allows the use of this algorithm on devices with limited computational power such as a tablet pc. It is noteworthy that we only use depth data for both tracking and object localization. The reason is that our main goal is to be able to robustly track industrial parts, and these usually do not carry much color information.

Single-view-Tracking. The multi-forest tracker described in [18] uses $6 * n_c * n_t$ random forests, where the number of dimensions to represent a pose is 6, n_c is the number of camera positions and n_t the number of trees in each forest. For each camera position sample points of the objects are extracted and used to train 6 random regression forests for tracking of this camera view. An algorithm switches between the camera views that are currently best visible. They parameterize this as $n_c = 42$ and $n_t = 100$ resulting in 25200 random trees. Each tree is generated from a test set of 50000 samples, resulting in a significant training effort. In our approach we have reduced this effort significantly to only $6 * n_t$ trees. After the samples have been generated, each random forest is trained with all samples for a single dimension of the pose vector. That is, random forests 1 to 3 are trained for changes in translation (*x*, *y*, and *z)*, and random forest 4 to 6 are trained on the changes in rotation (roll, pitch and yaw) parameters respectively. During tracking, the depth changes are used to predict the changes in pose vector by simply combining the predictions of each random forest.

In practice, we set $n_t = 70$, resulting in only 420 random trees which in turn leads to a 60 times faster training time and 60 times less memory requirements. It typically takes about 3 minutes on an Intel(R) Core(TM) i5-3570 CPU (the proposed approach is implemented and tested on such a workstation) to train a new object for tracking. This low memory requirement also allows the tracking to run in real time.

We initially sample a set of approximately 400 points from the surface of the object. Sampling is done by raytracing points onto the objects surface, creating a 3D grid around that object and then sampling a single point per grid. This sampling approach leads to evenly spaced sample points all over the visible surface of the object. The depth distance of these sample points to the visible depth map is then used in the training data. Since we have sampled points from all around the objects, many points will not lie on the visible surface but will be behind it. We rely on the random

regression forest to figure out what this means in term of object movement. In our experiments, the use of these single set of points has proven to lead to a highly stable tracking performance.

Pose Forecast. Tracking is performed on a frame by frame basis, by checking how depth values from sample points of the current positions vary when compared to the depth values of the current depth map. This means when an object is moving fast between two frames so that no or only few sample points of the previous position overlaps with the new position, the object cannot be tracked. In our algorithm we use the previously estimated movement prediction as a starting guess for the new movement prediction. This allows for objects to move further between two frames, and also provides a more accurate initial guess for the pose estimation. Initially when performing object localization, a movement of zero is assumed.

3.3. Combined Object Localization with Tracking

The full object tracking framework as shown in Fig. 1 combines both global object localization and tracking. The goal of this framework is to produce a continuous, low latency stream of the current object position. Whenever possible the system uses the fast tracking described above, and if tracking was not successful it switches back to the slower but global object localization. To track multiple objects simultaneously this tracking framework is run in parallel for each object.

From Object Localization to Tracking. Depending on the configuration of the object localization, it is possible to find multiple instances of the same object in a single frame. For our use case we assume that only a single instance is the correct one. To determine which of these instances the correct one is, we perform two checks. First, the instances are ranked by the number of *inliers (r)*, and only if it passes a certain threshold it is considered as a potential correct pose. For each pose the multi forest tracker is evaluated and tracking is performed. When running the tracking algorithm on a correct pose, the estimated tracking transformation should be a minimal *movement (m)*. When the tracking algorithm would estimate a large movement, this means that either this current position is wrong or cannot be tracked successfully. We use this tracking movement as an additional filtering criterion. In practice, we use both parameters to form a single sorting criterion *quality (q)* in (1):

$$q = \frac{r}{m^2} \qquad (1)$$

We rank all candidate poses by this criterion, which leads to more accurate results than the use of either one of the criterion separately.

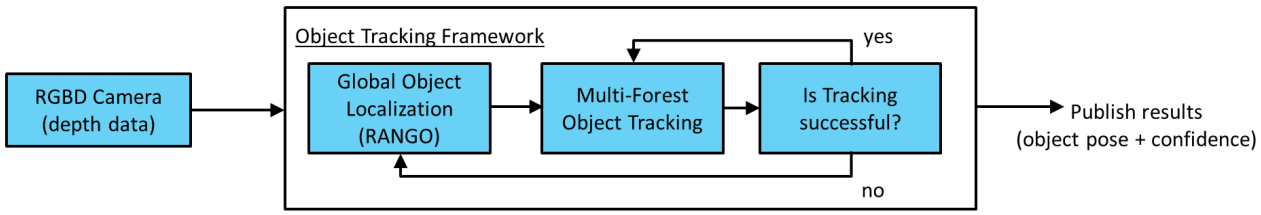

Figure 1. Object tracking framework. The framework first performs global object localization on the input depth data. After the detection results are filtered they are passed to the Object Tracking module. If tracking was successful, the framework will directly use the tracking module to track objects for the next sensor input. If not, it will revert back to global object localization for the next sensor input frame. The tracking results are published to a higher level system.

Tracking Verification. After tracking has been performed, we calculate the movement of the object with respect to the camera position. To determine if tracking was successful, we calculate if the current movement is reasonably realistic. We do this by calculating the acceleration of the object within the last 3 frames. If the acceleration is above a threshold, we consider the tracking as

not successful. This happens when the tracking has "lost" the object and consecutive tracking iteration move the object around in the sensor data. In practice we have set this velocity to 80mm per frame. With 30 fps this means a velocity of about 2.4m per second. This is enough to accurately track quickly moving objects, while being robust to detect the random movements that typically occur when tracking of the object fails.

4. Experiments and Evaluation

In order to compare the performance of our approach against the state of the art we use the synthetic dataset provided by Choi and Christensen [5]. The approach is also evaluated on real-world objects and the results are presented in [12]. Interested readers can find more information here[2]. The dataset in [5] consists of four object models and a synthetic test sequence (1000 RGB-D frames) for each object. The test sequence is obtained by placing each object in a virtual kitchen model and moving a virtual camera around the model. The object trajectories w.r.t the virtual camera coordinate frame serves as the ground truth pose (error free since it is generated via rendering) of the object. Fig. 2 shows one such frame of each object sequence. The performance of the proposed tracking approach on the synthetic data set is as shown in Figure 3.

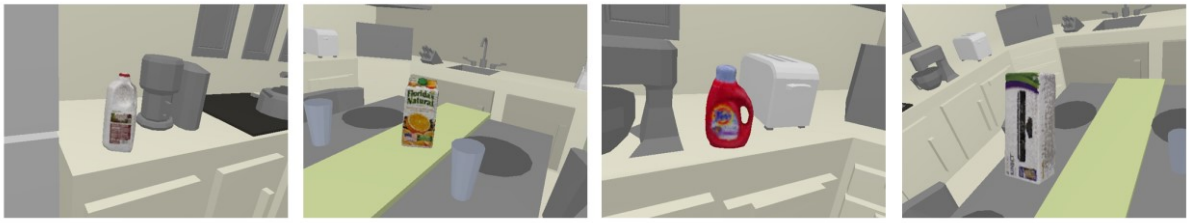

Figure 2. Example images from the synthetic test data set provided by [5], a) Milk b) Orange Juice c) Tide d) Kinect Box

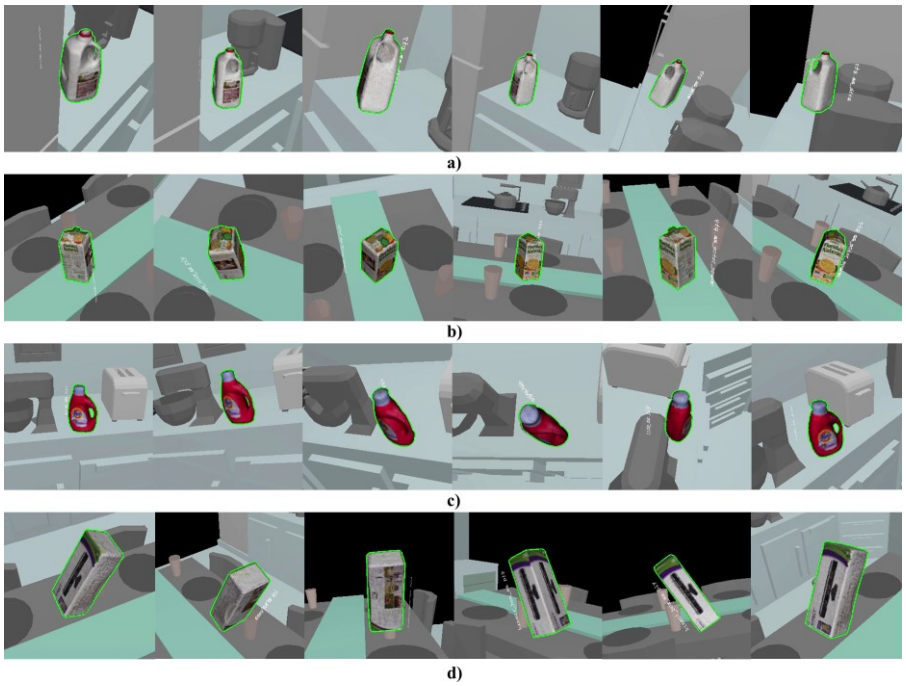

Figure 3: Results of the tracking approach on the synthetic data set a) Milk b) Orange Juice c) Tide and d) Kinect box

[2] http://tracking.profactor.at

Evaluation. The evaluation aims at comparing our approach against the particle filter based approaches [5][18][9] and online learning based approach [20] in estimating the translation (in x, y, and z axis) and rotation (roll, pitch and yaw) parameters. We compute the root mean square (RMS) errors (translation, rotation) and average time per frame. Table I shows our approach outperforms [5] and [18] over all sequences. Unlike [5] our approach only uses depth data for 3D tracking. It also performs better than [9] on average of about 0.31 mm and 1.16 deg in estimating the translation and rotation parameters respectively. Our approach requires much less computational time (1.7 ms per frame) when compared with [9] (131 ms). Though our approach performs on par with [20] in terms of run-time, it performs better in estimating the translation (by 0.31 mm) and rotation parameters (by 0.15 deg) on average.

TABLE I. COMPARISION OF OUR APPROACH WITH THE STATE OF ART AGAINST THE RMS ERRORS IN TRANSLATION (IN MM), ROTATION (DEGREES) AND THE RUNTIME (MS)

			PCL [18][1]	Choi [5][2]	Krull [9][3]	Tan [20][4]	Ours[5]
a) Milk	***RMS Error***	*Transl. (x)*	13.38	0.93	**0.51**	1.23	0.63
		Transl. (y)	31.45	1.94	1.27	**0.74**	1.19
		Transl. (z)	26.09	1.09	0.62	**0.24**	0.48
		Roll	59.37	3.83	2.19	0.50	**0.19**
		Pitch	19.58	1.41	1.44	**0.28**	**0.28**
		Yaw	75.03	3.26	1.90	0.46	**0.27**
		Time	2205	134	135	**1.5**	1.7
b) Orange juice	***RMS Error***	*Transl. (x)*	2.53	0.96	0.52	1.10	**0.39**
		Transl. (y)	2.20	1.44	0.74	0.94	**0.37**
		Transl. (z)	1.91	1.17	0.63	**0.18**	0.37
		Roll	85.81	1.32	1.28	0.35	**0.12**
		Pitch	42.12	0.75	1.08	0.24	**0.17**
		Yaw	46.37	1.39	1.20	0.37	**0.15**
		Time	1637	117	129	**1.5**	1.69
c) Tide	***RMS Error***	*Transl. (x)*	1.46	0.83	0.69	0.73	**0.42**
		Transl. (y)	2.25	1.37	0.81	0.56	**0.51**
		Transl. (z)	0.92	1.20	0.81	**0.24**	0.64
		Roll	5.15	1.78	2.10	0.31	**0.22**
		Pitch	2.13	1.09	1.38	**0.25**	0.29
		Yaw	2.98	1.13	1.27	0.34	**0.30**
		Time	2762	111	116	**1.5**	1.7
d) Kinect Box	***RMS Error***	*Transl. (x)*	43.99	1.84	0.83	1.54	**0.30**
		Transl. (y)	42.51	2.23	1.67	1.90	**0.49**
		Transl. (z)	55.89	1.36	0.79	0.34	**0.31**
		Roll	7.62	6.41	1.11	0.42	**0.21**
		Pitch	1.87	0.76	0.55	**0.22**	0.27
		Yaw	8.31	6.32	1.04	0.68	**0.23**
		Time	4539	166	143	**1.5**	1.71
Mean		***Transl.***	18.72	1.36	0.82	0.81	**0.50**
		Rot.	29.70	2.45	1.38	0.37	**0.22**
		Time	2786	132	131	**1.5**	1.7

1,2 Intel Core2 Quad CPU Q9300, 8G RAM with Nvidia GTX 590 GPU; **3** Intel(R) Core(TM) i7 CPU with a Nvidia GTX 550 TI GPU; **4** Intel(R) Core(TM) i7 CPU; **5** Intel(R) Core(TM) i5 CPU

6. Conclusion

We have presented a framework for combining object tracking and object localization to provide robust tracking performance in a challenging scenario. A quantitative analysis of the evaluation on popular test data set is also presented. The evaluation shows that our approach performs better than the state of art in terms of estimating the translation and rotation parameters. The approach is

capable of real-time computation at 1.7 ms per frame on average. The next steps would be to combine the real-time object tracking approach with human tracking and extend the framework towards human activity recognition in industrial settings.

7. Acknowledgment

This research is funded by the projects KoMoProd (Austrian Bundesministerium für Verkehr, Innovation und Technologie), SIAM (FFG, 849971) and CompleteMe (FFG, 849441).

8. References

[1] Armando Pesenti Gritti *et al.*, "Kinect-based people detection and tracking from small-footprint ground robots", *in Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, 2014.

[2] A. Yilmaz *et. al.*, "Object tracking: A survey," *ACM Computer Surveys (CSUR)*, vol. 38, no. 4, pp. 1–45, 2006.

[3] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3d object recognition", *in Proc. Of IEEE Int. Conf. on Computer Vision and Pattern Recognition,* 2010.

[4] C. Choi *et al*., "Robust 3D visual tracking using particle filtering on the special Euclidean group: A combined approach of keypoint and edge features", *Int. Journal of Robotics Research,* vol. 31, no.4, pp. 498–519, 2012.

[5] C.Choi and HI. Christensen, "RGB-D Object Tracking: A Particle Filter Approach on GPU", *in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1084-1091, 2013.

[6] C. Papazov and D. Burschka, "An efficient RANSAC for 3-D object recognition in noisy and occluded scenes", *in Proc. 10th Asian Conf. Computer Vision (ACCV),* 2010, pp. 135–148.

[7] C. Ren, V. Prisacariu, D. Murray, and I. Reid, "Star3d: Simultaneous tracking and reconstruction of 3d objects using rgb-d data," *in Proc. Int. Conf. on Computer Vision,* 2013.

[8] J. Suarez and R.R. Murphy, "Hand gesture recognition with depth images: A review," *in IEEE Int. Symposium on Robot and Human Interactive Communication*, pp. 411–417, 2012.

[9] Krull Alexander *et al.,* "6-dof model based tracking via object coordinate regression", *Computer Vision—ACCV, 2014*, Springer International Publishing, pp 384-399, 2015.

[10] M. Isard and A. Blake, "Condensation - Conditional density propagation for visual tracking," *Int. Journal of Computer Vision*, vol. 29, no. 1, 1998.

[11] Mao Ye *et al.*, "A survey on human motion analysis from depth data", *in Timeof- Flight and Depth Imaging. Sensors, Algorithms, and Applications*, pp. 149–187, Springer, 2013.

[12] S. Akkaladevi, M. Ankerl *et al.,* "Tracking multiple rigid symmetric and non-symmetric objects in real-time using depth data," [to appear] *in Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2016.

[13] S. Koo, D. Lee, and D. Kwon, "Multiple object tracking using an rgb-d camera by hierarchical spatiotemporal data association", *in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 1113–1118, 2013.

[14] S. Koo, D. Lee, and D. Kwon, "Incremental object learning and robust tracking of multiple objects from rgb-d point set data," *Journal of Visual Communication and Image Representation*, vol. 25, no. 1, pp. 108–121, 2014.

[15] S. Koo, D. Lee, and D. Kwon, "Unsupervised object individuation from rgb-d image sequences", *in Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pp. 4450– 4457, 2014.

[16] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm", *in Proc. IEEE Int. Conf. on 3-D Digital Imaging and Modeling*, pp. 145–152, 2001.

[17] S. Winkelbach, S. Molkenstruck, F. M. Wahl, "Low-Cost Laser Range Scanner and Fast Surface Registration Approach", *Pattern Recognition (DAGM 2006) LNCS 4174*, pp. 718-728, Springer 2006.

[18] Rusu, R. B. Rusu and S. Cousins. "3d is here: Point cloud library (pcl)", *in Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1- 4, 2011.

[19] Tan David Joseph, and Slobodan Ilic, "Multi-forest Tracker: A Chameleon in Tracking", *in Proc. Of IEEE Int. Conf. on Computer Vision and Pattern Recognition*, 2014.

[20] Tan David Joseph *et al.*, "A Versatile Learning-based 3D Temporal Tracker: Scalable, Robust, Online" *in Proc. IEEE Int. Conf. on Computer Vision,* vol. 1, No. 4, 2015.

[21] X. Li, W. Hu, C. Shen, Z. Zhang, A. Dick, A. Hengel, "A survey of appearance models in visual object tracking", *ACM Transactions on Intelligent Systems and Technology*, vol. 4, no. 4, pp. 1-48, 2013.