Robot Joint Tracking With Mobile Depth Cameras for Augmented Reality Applications

Antonio Zea, **Michael Fennel**, and **Uwe D. Hanebeck**
Intelligent Sensor-Actuator-Systems Laboratory (ISAS)
Institute for Anthropomatics and Robotics
Karlsruhe Institute of Technology (KIT), Germany
antonio.zea@kit.edu, michael.fennel@kit.edu, uwe.hanebeck@kit.edu

***Abstract*—Augmented reality (AR) in mobile devices (such as smartphones and tablets) is becoming more popular each day, and because of this many newer devices are starting to ship with embedded depth sensors. This presents a great opportunity for the field of extended object tracking, whose algorithms are well-suited for dealing with varying measurement quality while requiring little CPU usage. In this paper, we present an application in the field of robotics, based on the idea of reconstructing the dynamic state of a robot (joint positions and velocities) simply by observing it with an AR device, and using only the robot specification (its URDF file) as prior knowledge, without requiring a connection to the robot's control system. This can allow the mobile device to identify where a robot is, or viceversa, without requiring markers such as QR codes. Additionally, this can serve as a stepping stone for more sophisticated assistance systems that can interact with the robot without requiring any access to its internals, which could otherwise make it difficult to deploy the AR app in sensitive systems. Using the iPad Pro 2020 as an example device, we examine the challenges involved in processing mobile depth images, how to develop a robust shape model and the corresponding estimator, and how the app can ask the user to help in its initialization using AR. We will also provide an evaluation with real data that shows how the proposed system can track a moving robot robustly even if measurement quality is reduced significantly.**

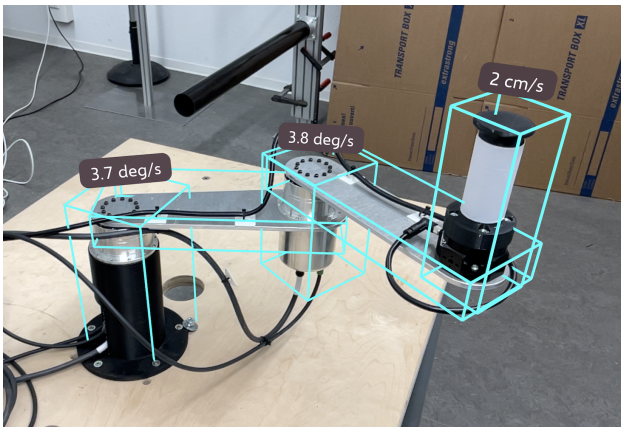
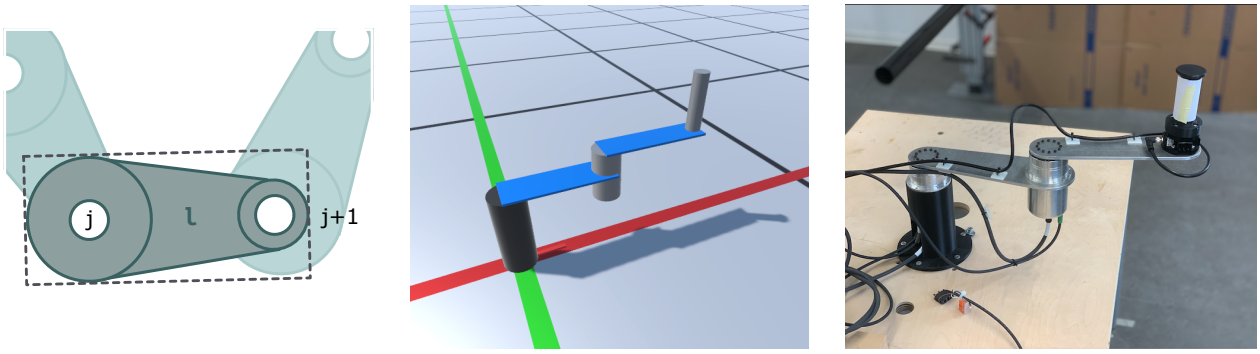


Fig. 1: Example AR visualization of a robot's velocities using an iPad Pro 2020. Using the iPad's depth images and the robot's description, an estimator can reconstruct its joint positions and velocities from the outside, without requiring a connection to the robot's internal system.

I. Introduction

Extended object tracking (EOT) [1] focuses on estimating the pose of moving targets from point measurements by exploiting information about the target's shape. Unlike traditional tracking techniques, which model the object being tracked as a point, in EOT the target is assumed to have an extent, i.e., we assume that multiple measurements from different points on its surface can be generated in a single scan. In turn, this means that the target's center cannot be observed directly, raising the challenge of how to incorporate information from unstructured, noisy measurements in an efficient way, given that both the shape and pose parameters may be unknown and have to be estimated simultaneously. There are multiple models in literature depending on how the target's shape is approximated, such as rectangles [2] or ellipses [3]. More complex star-convex shapes can be described using Fourier coefficients [4] or Gaussian processes [5]. Traditionally, EOT has been used to track relatively large targets such as airplanes, ships, or cars using LIDAR or radar measurements. As high-quality depth cameras become more affordable, for example with the Microsoft Kinect, EOT techniques have started being applied to smaller targets such as hand-held objects [6], [7]. Lately, a new generation of miniaturized mobile depth cameras embedded in smartphones and tablets has reached the market, noteworthy among them the iPads and iPhones from Apple, which promise to extend the applicability of EOT techniques in even more domains.

Mobile devices (in particular smartphones and tablets) have also attracted attention lately for another reason: augmented reality (AR). In contrast to virtual reality, which replaces the real world with the digital, in AR the objective is to enhance how users perceive and interact with their real surroundings. Ideally, the tasks of an effective AR application include semantic recognition of objects in the scene, visualizing context-aware information around them, reacting to changes in the environment, and enabling new ways to interact with the world. However, even if some requirements such as self-localization are already mature, overall AR can be still considered to be in its infancy. This can be attributed, among others, to the fact that mobile devices do not yet have the processing capability to understand general environments quickly and robustly enough. Nonetheless, there are already multiple notable works in literature that deal with understanding the environment using mobile



(a) Example joint and link. (b) Virtual robot constructed from description. (c) Real robot.

Fig. 2: Sketch of how a robot is constructed from its specification. In Fig. 2a we observe a link l with parent joint j and child joint $j+1$. The link consists of a 'visual' property (its geometric shape, grey) and a simplified 'collider' (rectangle in dotted lines). In Fig. 2b we see the simplified collider models (cylinders and cuboids) of the robot in Fig. 2c. Originally intended to detect collisions, colliders are meant to be fast, not to represent the robot's geometry with accuracy.

depth cameras, especially iPads [8], [9], with applications ranging from veterinary [10], [11] and forestry [12], [13] to reconstruction of buildings [14] and heritage documentation [15]. This serves as a motivation for new techniques that can bridge EOT and mobile AR, as EOT algorithms are already designed to deal with noisy measurements in environments with low computational resources.

A promising and rather underexplored application of AR is in the field of robotics, with some notable exceptions [16], [17]. AR can be used to better understand robotic algorithms by showing important information related to a place on top of that place itself, rather than a 2D monitor several meters away. Ideally, a paradigm of interaction with the robot would be to simply tap on it in order to launch an assistance system, which in turn gathers all relevant information about the robot's state and its pose simply by observing it — preferably, without being connected to the robot at all. To achieve this, we can use the fact that the entire specification of the robot is contained in its URDF description, including its joints, its links, and its geometry (Fig. 2). Then, using the depth images of an AR device, we can reconstruct its dynamic state, i.e., its pose and current joint positions and velocities, with minimal latency. This approach can be beneficial in the following contexts. First, it can be used by an AR device to localize the robot – or for the robot to find the device – quickly and without requiring markers such as QR codes. Second, it can be used for validation, to ensure that the internal robot state corresponds to the real joint position and link poses. And third, it may be that the robot encoders are not sensitive enough to know the exact joint positions, especially in large and heavy robots, and thus, our system would allow for more precise controllers without changing the hardware. Implementing this, however, requires an examination of the challenges involved in dealing with mobile depth cameras, and how they can be addressed using AR interactions.

In this paper, we propose a simple EOT approach to track the dynamic state of a robot using only its URDF specification, usually stored in a public repository, and the depth images of a mobile device. The goal of this framework is to serve as a stepping stone for more complex AR-based HRI interactions in the near future. As the target application in mobile AR, we will focus on Apple's iPad Pro 2020 and its sensors as a representative example. Our contribution is divided into the following sections. We start with the problem formulation in Sec. II, followed by a description of our algorithms in Sec. III. Then, we present an evaluation using real data in Sec. IV, and conclude it in Sec. V.

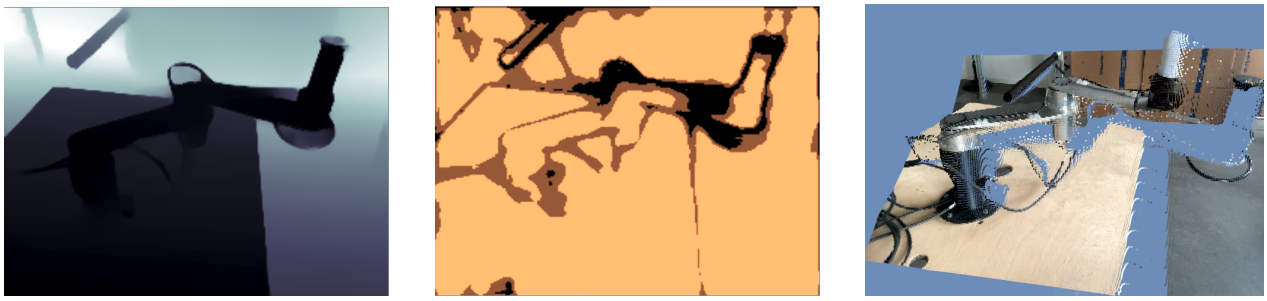
II. Problem Formulation

The objective of this work is to estimate the state $\underline{x}_k$ of a robot using noisy point measurements captured from its surface. The state

$$\underline{x}_k = \begin{bmatrix} \underline{x}_k^p & \underline{x}_k^r & \underline{x}_k^j \end{bmatrix}^T \in \mathbb{R}^n \tag{1}$$

at time step k consists of a position $\underline{x}_k^p$, an orientation vector $\underline{x}_k^r$, and a list of joint positions $\underline{x}_k^j$. Additional parameters such as velocities and accelerations can be included if needed. Structurally, the robot consists of a series of interconnected links $1 \leq l \leq n_l$ (see Fig. 2a) that form a graph (or more commonly, a tree). The shape of each link l, modeled as the set $\mathcal{S}_{k,l}$, is known a priori. These shapes do not change in time, but they are subject to rigid transformations whose parameters are determined by $\underline{x}_k^j$. Thus, the pose $[\underline{x}_k^p, \underline{x}_k^r]^T$ refers to the first link (the 'base link'), and the pose of the remaining links can be constructed recursively using standard forward kinematics. Fig. 2b shows a virtual robot constructed this way, based on the description of the real robot in Fig. 2c.

At each time step k, the robot is observed by a depth sensor, yielding a depth image $\mathcal{I}_k$ from which a series of measurements $\underline{y}_{k,1}, \cdots, \underline{y}_{k,m}$ can be obtained. Each measurement $\underline{y}_{k,i} \in \mathbb{R}^3$ is assumed to stem from a source $\underline{z}_{k,i}$ drawn from



(a) Example depth image of a robot. (b) Corresponding confidence image. (c) Reconstructed point cloud.

Fig. 3: Example depth captures from the iPad Pro 2020, taken from the scene in Fig. 2c. In Fig. 3a, we see the depth image in grayscale. Note the interpolations in regions where the cable loops. Fig. 3b shows the confidence image, where the colors represent: Yellow 2 (high), brown 1 (mid), and black 0 (low). Fig. 3c presents the reconstructed point cloud of the robot.

one of the links' surfaces, which is then distorted with an additive noise term $\underline{v}_{k,i}$ as modeled in

$$\underline{y}_{k,i} = \underline{z}_{k,i} + \underline{v}_{k,i} \tag{2}$$

$$= \underline{h}_{k,i}(\underline{x}_k) + \underline{v}_{k,i} \ , \tag{3}$$

where $\underline{v}_{k,i}$ is assumed to be zero-mean with covariance matrix $\mathbf{C}^v_{k,i}$, and $\underline{h}_{k,i}(\cdot)$ is the measurement function which associates the measurement $\underline{y}_{k,i}$ to the state $\underline{x}_k$. Furthermore, we assume that the state evolves in time according to

$$\underline{x}_{k+1} = \underline{a}_k(\underline{x}_k) + \underline{w}_k, \tag{4}$$

where $\underline{w}_k$ is a system noise term assumed to be zero-mean with covariance matrix $\mathbf{C}^w_k$. The term $\underline{a}_k(\cdot)$ is the system function.

III. Processing Pipeline

In this section, we will sketch out the implementation of our AR-based estimator using mobile depth images. First, we will analyze the depth images typically produced by a mobile depth sensor, and propose mechanisms to extract useful measurements from them while avoiding outliers and other artifacts. Second, we will describe an approach to obtain the link shapes from the robot description, which we will need in order to associate measurements with them. Third, we propose a small AR-based assistant system, which simplifies the segmentation task based on user input. Finally, we derive a nonlinear estimator to estimate the pose, joint positions, and other dynamic parameters of the robot efficiently.

A. The iPad Depth Sensor

In this section, we will briefly describe the characteristics of the depth images from the iPad Pro 2020 (Fig. 3), which we will use later for the evaluation, and how to obtain meaningful measurements from them. As other mobile depth systems for AR such as the Samsung Galaxy 20 or the Microsoft Hololens 2 have similar advantages and shortcomings, we believe the iPad can serve as a representative example of the considered sensor range.

The iPad RGBD system consists of two devices: an RGB camera and a time-of-flight depth sensor, both on the back of the tablet. Unfortunately, Apple's platform ARKit does not provide direct access to the depth measurements. Instead, it fuses the LIDAR data with color information from the RGB camera producing two synthetic data streams: a processed depth image and a confidence image, both at a rate of 30 frames per second. The depth image (Fig. 3a) has a size of 256×192 pixels, and each pixel's depth is described as a 32-bit floating point number representing the measured distance in meters. According to some studies [18], this depth image is extrapolated from only 576 LIDAR pulses, which serves to explain why measured surfaces appear smooth and deformed, lack sharp corners, and have a high amount of 'flying pixels' around discontinuities. Furthermore, unlike more mainstream depth sensors like the Microsoft Kinect 2, depth pixels within a single frame are highly correlated with their neighbors, likely because of the RGB fusion procedure.

Another difference w.r.t. the Kinect is that the validity of each depth pixel is not encoded in whether its value is different from 0. Instead, the reliability of each measurement is described separately in the confidence image (Fig. 3b), also of size 256×192 pixels. Here, however, each measurement is associated with an 8-bit value that can be either 0 (low confidence), 1 (mid confidence), or 2 (high confidence). Thus, if a pixel in the confidence image has a value of 0, we know that the corresponding position in the depth image carries little to no information.

The resulting depth cloud can be seen in Fig. 3c. While the robot arm appears recognizable, slightly changing the perspective to a view from above, shown in Fig. 4, makes the relatively low quality of the depth measurements evident, especially given how close the sensor is to the scene (about 1 meter). To alleviate this issue *without* modifying the measured depths, we propose the following preprocessing steps:

1) first, we invalidate all pixels in the depth image whose corresponding confidence is not 2 (high), and
2) second, we invalidate all 'flying pixels', defined as any pixel that has at least one neighbor with a relative distance larger than 0.02 m.

Note that these thresholds are not definitive, and can be relaxed if the number of acceptable measurements is too low.

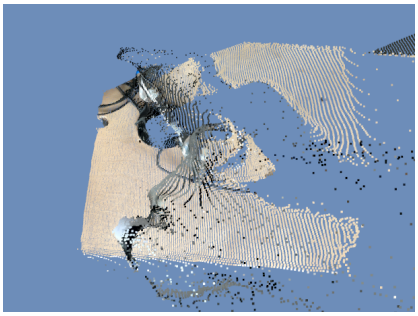


Fig. 4: View of the scene in Fig. 3 from above, highlighting the relatively low quality of the available measurements, and the strong presence of outliers and deformations.

Given an image $\mathcal{I}_k$ preprocessed with the above steps, we can extract the point measurements $\underline{y}_{k,i}$ as follows. First, we read the pre-calibrated intrinsic matrix $\mathbf{K}$ from the AR platform, which for iPads look something like

$$\mathbf{K} = \begin{pmatrix} 212.4 & 0 & 127.0 \\ 0 & 212.4 & 96.3 \\ 0 & 0 & 1 \end{pmatrix} \quad , \tag{5}$$

representing a horizontal field of view of approximately 60 degrees. Then, for each valid pixel at position $[\mathsf{x}_{k,i}, \mathsf{y}_{k,i}]$ with depth $\mathsf{z}_{k,i}$ we introduce the pseudo-measurement in screen coordinates

$$\underline{y}^s_{k,i} := \begin{bmatrix} \mathsf{x}_{k,i} & \mathsf{y}_{k,i} & 1 \end{bmatrix}^T \quad . \tag{6}$$

We also assume that there exists a sensor noise term modeled as zero-mean with covariance matrix

$$\mathbf{C}^{v,s}_{k,i} \approx \operatorname{diag}\left(\frac{1}{3}, \frac{1}{3}, 0\right) \quad , \tag{7}$$

where the variances correspond to a distribution of $\mathcal{U}(-1, 1)$, i.e., the true source position was equally likely to be between the previous, the current, and the next pixels. We also need an uncertainty for the depth $\mathsf{z}_{k,i}$, which is difficult to obtain given that it depends on multiple factors such as distance, material, and the unknown internals of the RGB fusion system. For this work, we will use the conservative approximation of

$$\sigma^2_{\mathsf{z}} \approx 10^{-5}\ \mathrm{m}^2 \quad , \tag{8}$$

calculated from empirical observations.

Based on these conditions, we obtain the measurement in camera coordinates using the standard pinhole unprojection formula

$$\underline{y}^c_{k,i} = \mathbf{K}^{-1} \cdot \underline{y}^s_{k,i} \cdot \mathsf{z}_{k,i} \quad , \tag{9}$$

with the corresponding covariance matrix following from the random variable product rule as

$$\mathbf{C}^{v,c}_{k,i} = \mathbf{K}^{-1} \left(\left(\mathbf{C}^{v,s}_{k,i} + \underline{y}^s_{k,i} [\underline{y}^s_{k,i}]^T \right) \sigma^2_{\mathsf{z}} + \mathbf{C}^{v,s}_{k,i}\ \mathsf{z}^2_{k,i} \right) (\mathbf{K}^{-1})^T \, . \tag{10}$$

As a final step, we must transform these values to world coordinates. We assume that the camera pose $[\mathbf{R}^c_k,\ \underline{t}^c_k]$ is known and provided by the AR platform. This yields the desired measurement

$$\underline{y}_{k,i} = \mathbf{R}^c_k \cdot \underline{y}^c_{k,i} + \underline{t}^c_k \quad , \tag{11}$$

$$\mathbf{C}^v_{k,i} = \mathbf{R}^c_k \cdot \mathbf{C}^{v,c}_{k,i} \cdot (\mathbf{R}^c_k)^T \quad . \tag{12}$$

This formula keeps the correlations intact. However, for reasons of speed it is often preferable to use an isotropic noise term. In this case, a reasonable approximation for points close to the image center is

$$\mathbf{C}^v_{k,i} \approx \sigma^2_{\mathsf{z}} \cdot \mathbf{I}_3 \quad . \tag{13}$$

While the proposed approach is simplified by using the camera pose from the AR platform, it should be noted that this value is also uncertain. The inter-frame noise is very low compared to the measurement noise, but it has a bias that accumulates over time, roughly reaching about 1 cm after a couple of minutes. This means that robot pose, together with the entire scene, will drift slightly over time.

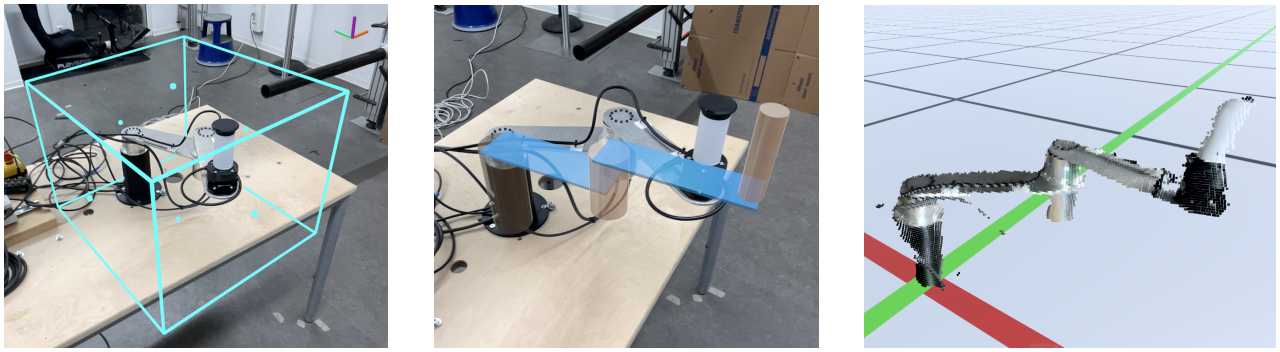
B. Constructing a Shape Model for the Robot

As mentioned before, the robot description contains the necessary information to determine the robot geometry in function of a state vector. The most commonly used format is the XML-based URDF format, generally included as a file in the robot drivers or written in the 'robot_description' parameter in a ROS system. It contains all the important properties of the robot, in particular

- a list of links (the robot 'segments'), including their shapes and 'colliders',
- and a list of joints, which connect links to each other and determine how they can move, for example by rotating (revolute) or translating (prismatic) along an axis.

Of interest in each link is the 'collider' property, which gives us a rough (usually convex) simplification of the link geometry. This is often more useful than the 'visuals', which generally consist of detailed CAD models better suited for visualization. Another important characteristic of colliders is that they are rarely covered by NDAs, as they are too coarse to contain important proprietary information. For this reason, URDFs and their colliders are often available from public repositories, which can in turn be easily accessed from our mobile app.

In order to derive our shape model, we need two things. First, we need a way to obtain the robot geometry for any given state $\underline{x}_k$. To achieve this, we calculate the pose of each link using standard forward kinematics libraries such as [19]. The robot geometry is then constructed by applying the resulting transformations onto the link shapes $\mathcal{S}_{k,l}$. Second, we need a mechanism to associate an arbitrary measurement $\underline{y}_k$ to the link shapes produced by $\underline{x}_k$. As modeled in (3), this is achieved by finding the source $\underline{z}_k$ on the robot's surface that generated $\underline{y}_k$. Unfortunately, as a consequence of the sensor noise $\underline{v}_k$, it is usually impossible to find the exact source with complete certainty. Instead, as proposed in [20], we can approximate it as the point most likely to have generated it, according to the distribution of $\underline{v}_k$. In case that the measurement noise is



(a) Determining the workplace volume. (b) Starting pose of the robot. (c) Representative point cloud after filtering.

Fig. 5: Sketch of the AR assistant system to simplify the estimation process, allowing the user to set a workplace volume for gating (Fig. 5a) and a custom starting pose (Fig. 5b). The resulting measurement point cloud (Fig. 5c) is much informative, and can be compared to the unfiltered source image in Fig. 4.

approximately isotropic, this corresponds to the point on the robot with the smallest Euclidean distance to $\underline{y}_k$, i.e.,

$$\underline{z}_k = \underset{\substack{\underline{z}_k^* \in \mathcal{S}_{l,k} \\ 1 \le l \le n_l}}{\operatorname{minarg}} \ \|\underline{z}_k^* - \underline{y}_k\|^2 \ , \tag{14}$$

which can be found in closed-form for shapes such as spheres, cylinders, cuboids, or triangle meshes.

C. Initializing with Augmented Reality

While the preprocessing steps from Sec. III-A are effective at removing spurious measurements, we still have the problem that measurements from the background, or from nearby objects, can confuse the estimator and lead to incorrect results. A way to alleviate this issue is to implement a gating mechanism that rejects measurements that are too far from the estimate, and thus, have a high likelihood of either being outliers or belonging to another object. However, this requires an initial estimate that is reasonably close to the real robot. In the worst case, if either the gating threshold or the starting pose are far from appropriate, the estimator may lose track of the robot or end up stuck in a local minimum.

To address this issue, we propose a simple additional step that ensures a robust initialization in any scenario, in the form of an AR assistant system that lets a user provide both starting parameters (Fig. 5). First, the user determines the extent of the workplace using a resizable cuboid (Fig. 5a, the dots in each face are the handles). All measurements from outside this geometry are rejected. Then, the user is asked to set the starting position and orientation of the base link (Fig. 5b) using a small on-screen joystick. Finally, with a simple tap, the application starts the estimation process. Fig. 5c shows the resulting point cloud after the preprocessing and gating, which reproduces the real robot with improved accuracy. Of course, the initialization does not require or expect a perfect user input, and as shown later in Sec. IV-A, the estimator can deal with initial offsets of 10 to 20 cm.

D. Deriving an Estimator

We will now put all the pieces in this section together to derive an estimator. Given the relatively low measurement quality, the high amount of outliers, the coarseness of the collider approximations, and the low resource availability coupled with the 'real-time' demands of the tracking application (30-60 frames per second), we propose the use of recursive nonlinear Kalman filters such as the UKF or the S^{2}KF [21]. In these filters, the state and its uncertainty are represented by the mean $\hat{\underline{x}}_k$ and the covariance matrix $\mathbf{C}_k^x$.

The initialization can be implemented as

$$\underline{x}_0 = \begin{bmatrix} \underline{x}_0^p & \underline{x}_0^r & \underline{x}_0^j \end{bmatrix}^T \ , \tag{15}$$

where $[\underline{x}_0^p, \underline{x}_0^r]^T$ is the user input described in Sec. III-C, and $\underline{x}_0^j$ is set to zero. The corresponding covariance matrix is

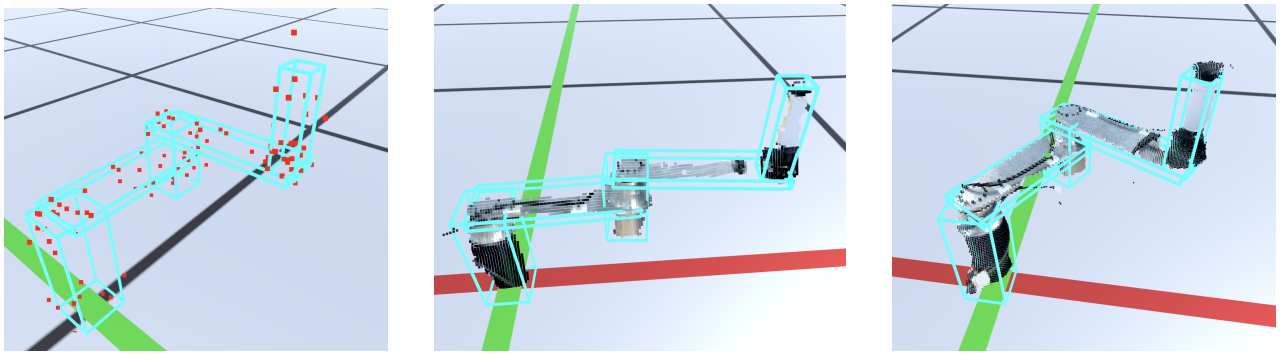
$$\mathbf{C}_0^x = \sigma_{x,0}^2 \cdot \mathbf{I}_n \ , \tag{16}$$

where $\sigma_{x,0}^2$ is suitably larger than the measurement noise, for example $\sigma_{x,0}^2 \approx 10^{-2}$.

At each time step, a new image $\mathcal{I}_k$ will be captured by the depth sensor. The measurements $\underline{y}_{k,i} \in \mathbb{R}^3$ are obtained as described in Sec. III-A, using the measurement equation (3) and as measurement function the closest source derived in Sec. III-B. In order to process multiple measurements at the same time, they can be stacked vertically into a single vector. However, as the number of valid measurements m after gating (Sec. III-C) can still be anywhere from 100 to $10,000$, it is impracticable to process them all in a single update step, as this would require a cubic operation on a large matrix of size 3 m $\times$ 3 m. A more efficient approach is to shuffle the measurements and update the state recursively using only 8 at a given time (i.e., a vector size of 24), in order to maximize SIMD usage.

Finally, for cases when the robot joints are usually moving, it makes sense to implement a motion model. In this case, the state can be extended to include a joint velocity vector $\underline{x}_k^\alpha$, leading to a linear system equation

$$\underline{x}_{k+1} = \mathbf{F}_k \cdot \underline{x}_k + \underline{w}_k \ , \tag{17}$$



(a) Estimate with simulated data (red). (b) Real data estimate, 'straight' configuration. (c) Real data estimate, 'L' configuration.

Fig. 6: Setup of experiments for the evaluation, with simulated data (Fig. 6a) and with real data, both in a 'straight' configuration (Fig. 6b) and in an 'L' position (Fig. 6c). The cyan lines represent the estimate that corresponds to the measurements.

where the transition matrix $\mathbf{F}_k$ advances the joint positions according to their velocities and the time elapsed. The covariance matrix for the noise term $\underline{w}_k$ can be modeled as

$$\mathbf{C}_k^w = \sigma_{w,k}^2 \cdot \mathbf{I}_n \ , \tag{18}$$

where $\sigma_{w,k}^2$ is suitably smaller than the measurement noise, for example $\sigma_{w,k}^2 \approx 10^{-7}$. Note that the system noise of the robot pose is non-zero to compensate for AR camera pose drifting.

IV. Evaluation

We will present an evaluation of the proposed approach using a robotic manipulator with 2 joints, shown in Fig. 2c and throughout the paper. Two sets of experiments were done, based on synthetic and real data. Fig. 6 shows a sketch of the experiment setup. For the synthetic simulations, we used the colliders from the description shown in Fig. 2b. For the real captures, we used depth images from an iPad Pro 2020 taken of the robot in Fig. 2c. Fig. 6a shows an example of the simulated point measurements (red), while Fig. 6b and Fig. 6c present example point clouds reconstructed from the depth images. The lines in cyan show example estimates.

The state consisted of

$$\underline{x}_k = \begin{bmatrix} x^p_{k,(0)} & x^p_{k,(1)} & x^j_{k,(0)} & x^j_{k,(1)} \end{bmatrix}^T \in \mathbb{R}^4 \ , \tag{19}$$

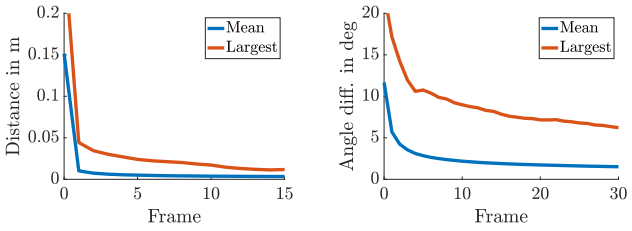
with $[x^p_{k,(0)}, x^p_{k,(1)}]^T$ representing the horizontal and vertical positions of the robot's base link on the table, while $[x^j_{k,(0)}, x^j_{k,(1)}]^T$ were two joint positions in radians. For reference, the base link is on left side in Fig. 6b (the black cylinder). The first joint $x^j_{k,(0)}$ is on top of the base link, while the second joint $x^j_{k,(1)}$ is in the center, shown bent in Fig. 6c. As the z-position of the robot does not change (it is anchored to the table plane), its value was taken directly from the position given to the AR assistant. The initialization was implemented as proposed in Sec. III-D, and for the real data the starting pose was set in a similar way as in Fig. 5b. For estimation, a S^2KF with 5 samples per dimension was used. The visualization and data collection were done using the mobile app iviz [22].

A. Synthetic Data

The experiments with synthetic data were aimed at establishing a baseline for the results that can be expected for a generic mobile sensor. Furthermore, they helped to establish a threshold when dealing with incorrect but close initializations. As is it difficult to reproduce artifacts such as outliers and deformations, we used a reduced amount of measurements per time step and increased the measurement noise compared to (8). The setup was as follows. We launched 1000 runs, each with a different initialization, with the starting positions moved a distance of $\mathcal{U}(-0.2, 0.2)$ meters from the ground truth, and both joints moved similarly with an offset of $\mathcal{U}(-\pi/8, \pi/8)$ radians. For reference, the length of the stretched arm is 0.5 meters. At each time step, we generated 100 sources from the surface of the robot colliders, and disturbed them with a measurement noise of $\mathbf{C}^v_{k,i} = 5 \cdot 10^{-4} \cdot \mathbf{I}_3$. As proposed in Sec. III-D, the update step processed measurements sequentially in batches of 8. After each update step, a prediction step was applied with an identity system function and process noise of $\mathbf{C}^w_k = 10^{-7} \cdot \mathbf{I}$. Two configurations were used: 'flat' with extended joints (Fig. 6b) and 'L' with the second joint set to 90 degrees (Fig. 6c).

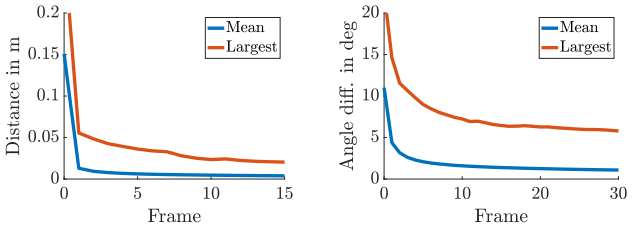
Fig. 7 and Fig. 8 show the results, with the mean value in blue and the worst case in red. In general, we see that the convergence of the position is rather fast, with the mean getting closer to 1 centimeter after the first frame. This is not surprising, given that all measurements contribute indirectly to the position. And while the convergence is slower for the L configuration (Fig. 8a) than for the straight (Fig. 7a), we see that even in the worst case we are still very close by the first second (frame 30). Of interest here is the second joint, which is the most susceptible to noise. Here, we see that we never really converge to 0, and there is a bias of about 3 degrees. The presence of bias is not surprising, as studied in [20], given the rather unrefined source assumption of taking simply the closest point in the closest link. While more sophisticated probabilistic techniques are possible, they also come at the expense of increased processing time. Furthermore, having a limit of around 7 degrees in the worst case is an acceptable

upper bound. It is also worth noting that the noise in most real sensors is much lower than modeled here.



(a) Position error, base link. (b) Angle error, second joint.

Fig. 7: Synthetic data, straight configuration.



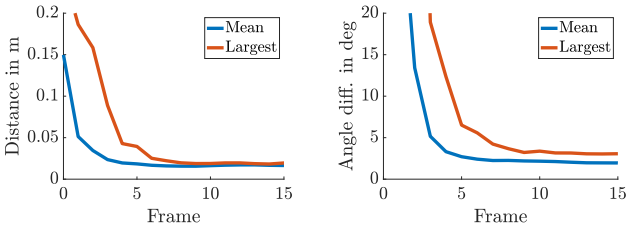
(a) Position error, base link. (b) Angle error, second joint.

Fig. 8: Synthetic data, L configuration.

B. Real Data, Static

The setup for the real data was similar to the synthetic data. The main difference was that 100 runs were used, and the measurements originated from an iPad observing the real robot. Each run was executed one after another, and each captured 30 frames. The initializations were random with the same parameters as the synthetic data. However, unlike the simulations, the real experiments processed as many measurements as they could while staying below 15 milliseconds per frame. This yielded on average 2300 measurements per frame.

For the real experiments we will focus on the L configuration only. An example estimate is shown in Fig. 6c. The results of the position errors and seconds joint errors are shown in Fig. 9. An interesting observation here is that, even though there are much more measurements, and nominally the measurement noise is lower, convergence is actually a bit slower, and the variance of estimates is higher. This can be explained by the high amount of outliers, and the fact that the colliders are not exact representations of the robot. Even worse, the cables of the robot produce several measurements that confuse the estimator. Still, after 7 or so frames, the position error (Fig. 9a) reaches its minimum. The remaining position bias is likely caused by a drift in the AR camera pose. The second joint in Fig. 9b is also slightly biased, likely due to artifacts that push the manipulator away from the camera.



(a) Position error, base link. (b) Angle error, second joint.

Fig. 9: Real data, L configuration.

C. Real Data, Moving Pendulum

For this last scenario, we used a different setup. We programmed the robot to move both joints like sinusoidal pendulums, with amplitude 30 degrees. The first joint completed a cycle every 5 seconds and the second joint every 2.5 seconds. A total of 20 runs were launched one after another, like in the previous scenario, each until they completed 750 frames. For the estimator, the state was extended to incorporate a constant velocity model for the two joints, i.e., the estimator was not directly aware of the sinusoidal motion.

Of note in this scenario is the appreciable reduction in measurement quality for the iPad sensor when capturing moving targets. On average 800 measurements were processed each time step, not because of an artificial constraint, but because that was the total of measurements available after preprocessing. As shown in Fig. 10, the iPad had significant difficulties observing the manipulator (last link) when it moved, and quite often, entire parts of the robot became invisible. This can be a consequence of the RGB fusion system, or due to motion blur in the depth sensor. The number of artifacts also increased significantly.

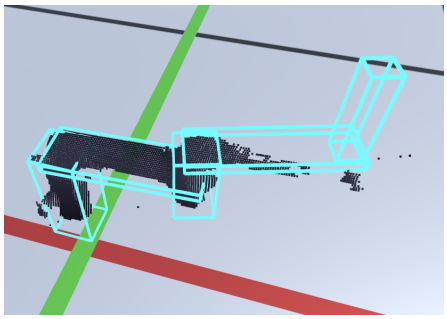
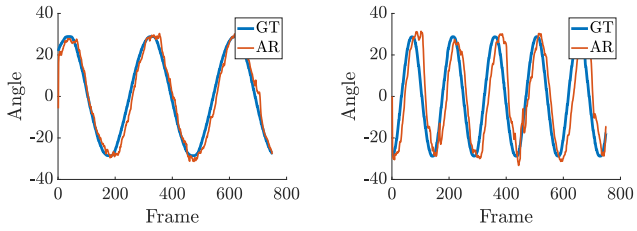


Fig. 10: Example of low measurement quality for the moving robot. Note the lack of measurements for the last links.

Fig. 11 shows the mean positions for the two joints ('AR') compared to the ground truth ('GT'). We observe that the results of the first joint are quite acceptable, as the first links do not move too much and thus do not become distorted. However, the second joint, which moves much faster, is clearly delayed, and its results are much noisier. Interestingly, the frequency is the same, and none of the runs lost its track,

showing that the estimator can keep up with the robot even with extremely few measurements.



(a) First joint position. (b) Second joint position.

Fig. 11: Pendulum scenario, ground truth and AR app.

V. Conclusion

As augmented reality (AR) becomes more popular, the number of mobile devices with depth cameras is starting to increase dramatically. This presents new opportunities for the field of extended object tracking (EOT), which is well-suited for dealing with noisy measurements while using low CPU resources. In particular, EOT can be used to fill a gap in modern AR applications: understanding the environment.

In this paper, we presented an AR application that used EOT to estimate the pose of a robot and its joint positions, using only the robot's specification as prior information. To achieve this, we used the depth camera of the AR device to obtain point measurements from the robot's surface. Then, we derived a measurement equation that can be plugged into an estimator, such as a nonlinear Kalman filter, to track the robot's movements efficiently in real time. To ensure robustness during initialization, we proposed a simple AR assistant system that lets the user determine a gating threshold and a starting pose.

We evaluated our algorithm by tracking a serial manipulator robot using depth images from an iPad Pro 2020, selected as a representative AR device. We showed that the estimator provided very accurate results when the robot was not moving, even when shape models did not reflect the real robot very closely. And while measurement quality degraded significantly once the joints started moving as a consequence of the limitations of the mobile sensor, the estimator was still able to exploit this sparse information to follow the robot with remarkable accuracy.

References

[1] K. Granstrom, M. Baum, and S. Reuter, "Extended object tracking: Introduction, overview and applications," *arXiv preprint arXiv:1604.00970*, 2016.

[2] S. Lee and J. McBride, "Extended object tracking via positive and negative information fusion," *IEEE Transactions on Signal Processing*, vol. 67, no. 7, pp. 1812–1823, 2019.

[3] S. Yang and M. Baum, "Tracking the orientation and axes lengths of an elliptical extended object," *IEEE Transactions on Signal Processing*, vol. 67, no. 18, pp. 4720–4729, 2019.

[4] M. Baum and U. D. Hanebeck, "Shape tracking of extended objects and group targets with star-convex rhms," in *14th International Conference on Information Fusion*. IEEE, 2011, pp. 1–8.

[5] E. Özkan, N. Wahlström, and S. J. Godsill, "Rao-blackwellised particle filter for star-convex extended target tracking models," in *2016 19th International Conference on Information Fusion (FUSION)*, 2016, pp. 1193–1199.

[6] F. Faion, A. Zea, M. Baum, and U. D. Hanebeck, "Bayesian estimation of line segments," in *Proceedings of the IEEE ISIF Workshop on Sensor Data Fusion: Trends, Solutions, Applications (SDF 2014)*, Bonn, Germany, October 2014.

[7] A. Zea, F. Faion, and U. D. Hanebeck, "Tracking elongated extended objects using splines," in *Proceedings of the 19th International Conference on Information Fusion (Fusion 2016)*, Heidelberg, Germany, July 2016.

[8] M. Vogt, A. Rips, and C. Emmelmann, "Comparison of iPad Pro®'s LiDAR and TrueDepth Capabilities with an Industrial 3D Scanning Solution," *Technologies*, vol. 9, no. 2, 2021. [Online]. Available: https://www.mdpi.com/2227-7080/9/2/25

[9] A. Breitbarth, T. Schardt, C. Kind, J. Brinkmann, P.-G. Dittrich, and G. Notni, "Measurement accuracy and dependence on external influences of the iPhone X TrueDepth sensor," in *Photonics and Education in Measurement Science 2019*, vol. 11144. International Society for Optics and Photonics, 2019, p. 1114407.

[10] S. J. Valberg, A. K. Borer Matsui, A. M. Firshman, L. Bookbinder, S. A. Katzman, and C. J. Finno, "3 dimensional photonic scans for measuring body volume and muscle mass in the standing horse," *PloS one*, vol. 15, no. 2, p. e0229656, 2020.

[11] A. Matsuura, M. Dan, A. Hirano, Y. Kiku, S. Torii, and S. Morita, "Body measurement of riding horses with a versatile tablet-type 3d scanning device," *Journal of equine science*, vol. 32, no. 3, pp. 73–80, 2021.

[12] C. Gollob, T. Ritter, R. Kraßnitzer, A. Tockner, and A. Nothdurft, "Measurement of forest inventory parameters with apple ipad pro and integrated lidar technology," *Remote Sensing*, vol. 13, no. 16, p. 3129, 2021.

[13] X. Wang, A. Singh, Y. Pervysheva, K. Lamatungga, V. Murtinová, M. Mukarram, Q. Zhu, K. Song, P. Surovỳ, and M. Mokroš, "Evaluation of ipad pro 2020 lidar for estimating tree diameters in urban forest," *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 8, pp. 105–110, 2021.

[14] A. Spreafico, F. Chiabrando, L. Teppati Losè, and F. Giulio Tonolo, "The ipad pro built-in lidar sensor: 3d rapid mapping tests and quality assessment," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 43, pp. 63–69, 2021.

[15] A. Murtiyoso, P. Grussenmeyer, T. Landes, and H. Macher, "First assessments into the use of commercial-grade solid state lidar for low cost heritage documentation," *ISPRS-International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 43, pp. 599–604, 2021.

[16] D. Lee and Y. S. Park, "Implementation of augmented teleoperation system based on robot operating system (ros)," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 5497–5502.

[17] M. E. Walker, H. Hedayati, and D. Szafir, "Robot teleoperation with augmented reality virtual surrogates," in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*. IEEE, 2019, pp. 202–210.

[18] System Plus Consulting, "Apple iPad Pro LiDAR Module," https://www.systemplus.fr/reverse-costing-reports/apple-ipad-pro-11s-lidar-module/, 2021, [Online; accessed 26-Oct-2021].

[19] Orocos.org, "Orocos Kinematics and Dynamics," https://www.orocos.org/kdl.html, 2021, [Online; accessed 13-Mar-2022].

[20] F. Faion, A. Zea, M. Baum, and U. D. Hanebeck, "Partial likelihood for unbiased extended object tracking," in *Proceedings of the 18th International Conference on Information Fusion (Fusion 2015)*, Washington D.C., USA, July 2015.

[21] J. Steinbring and U. D. Hanebeck, "Lrkf revisited: The smart sampling kalman filter (s2kf)," *Journal of Advances in Information Fusion*, vol. 9, no. 2, pp. 106–123, December 2014.

[22] A. Zea and U. D. Hanebeck, "iviz: A ROS visualization app for mobile devices," *Software Impacts*, vol. 8, p. 100057, May 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2665963821000051