Object Tracking For Robotic Assembly Dataset Creation And Tracking Evaluation

Leon Magnus, Svenja Menzenbach, Max Siebenborn, Niklas Funk, Boris Belousov, Georgia Chalvatzaki

***Abstract*— While many current object tracking approaches focus on tracking humans or traffic situations, and not include robotic models, in this work we specifically focus on the task of object tracking in a robotic assembly environment. For this, we outline the specific challenges and chances of object tracking for the assembly purpose and then create a dataset in a PyBullet simulation environment, where a robot arm picks, moves and places a block in the scene. We then evaluate the tracking performance of the AR tracking library VisionLib and conclude that it is not suitable for tracking objects in robotic assembly. Finally, we outline further research topics which are important for creating a good object tracker in robotic assembly.**

I. Introduction

Object tracking has a remarkable importance for many modern technologies. It plays a fundamental role in applications as Augmented Reality (AR) and traffic tracking. In this context, reliability in the tracking results is very important. This work addresses the problem of object tracking specifically for robotic assembly. Object tracking in general can be described as the process of estimating a target object's state over multiple frames, given an initial state (e.g. object's pose) [18]. There is a lot of literature about tracking algorithms and their challenges and problems. For example, Yilmaz et al. [20] appoint complex situations for robust tracking and classify different approaches to it.

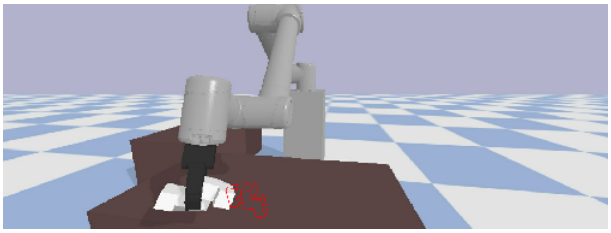
The goal of the robotic assembly setup is to make a robot arm independently move and stack parts for building architectural objects. In this work, the architectural assembly setup is simulated in a PyBullet [1] environment, consisting of an UR10 robot arm [14] with an attached ROBOTIS RH-P12-RN gripper [9]. An overview of the environment scene can be seen in figure 2 and the environment is described in detail in section III. For the assembly task, the arm moves solid SL-Blocks around the scene.

Architectural assembly relies on good object trackers. Because there are multiple cameras in the environment it is possible to use lots of data for state estimation. Especially estimating the poses of the SL-Blocks is essential for the robot to move and stack them optimally. Additionally, including kinematics and the measured poses of the robot arm's joints are very beneficial for state estimation. This highlights the need to address object tracking in robotic assembly in more detail.

It is very reasonable to consider the problem of object tracking in robotic assembly from a probabilistic perspective. The probabilistic view on robotics is mainly described by Thrun et al. [13]. They describe real-world systems as complex and only partially observable environments. In addition to that they are also increasingly unstructured and unpredictable.

(a) Tracked state.

(b) Critical state.



(c) Lost state.

Fig. 1. VisionLib's three tracking states: 1(a) tracked, 1(b) critical and 1(c) lost.

Therefore, including uncertainties and probabilities is a very promising approach because that offers the advantage that multiple hypotheses can be handled and a tradeoff between exploration and exploitation is feasible. In addition, sensors are noisy and do not display the current state correctly. Despite that, every gathered information, even if it is really noisy, helps to model the environment and make estimations better if probabilistic methods are used for modeling [3].

Probabilistic object trackers rely on good datasets. That is why as a first step towards an object tracker for robotic assembly we created a dataset in the assembly simulation environment in section IV. The dataset can be used for learning from the data and evaluating the tracking performance. This allows the object tracker to adjust to the environment and improve its performance.

Using this dataset, we evaluated the tracking performance of the AR tracking library VisionLib [15]. Some example tracking frames with their respective tracking status in

VisionLib can be seen in figure 1. VisionLib takes an .obj file and an image sequence as input, and estimates the objects pose in subsequent frames, while expressing its own confidence in the tracking results with the three states 'tracked', 'critical' and 'lost'. In section V we evaluated VisionLib's performance and applicability for robotic assembly on some example frame sequences from the assembly environment to get an overview of the main problems which occur when tracking objects for robotic assembly.

II. Related Work

Object Tracking: Common methods for object tracking are the Kalman filter [5, 3, 13, 19, 20], extended Kalman filter [3, 13, 19], unscented Kalman filter [3, 13, 19] and the particle filter [3, 13, 18, 19, 20]. Many object tracking approaches take depth images as input [4, 7, 19]. Since we also record depth images and especially focus on object tracking using a range camera, approaches that use depth information are of special interest. Depth images are also useful in the widely spread topic of 3D model-based object tracking. They make representations of 3D objects possible. To take advantage of the information given by depth images it is often useful to have shape information like 3D-meshes of the object. Worth mentioning are the papers from Radkowski [7], Isaac et al. [4] and Wuest et al. [10]. To track the object, methods like robust Gaussian filters [4] or point cloud matching [7] are proposed. These approaches are promising and should be considered for use in robotic assembly.

The proposed robust Gaussian filter should be less susceptible to problems that occur using a standard Gaussian filter, such as fat-tailed measurement noise of depth sensors and the exorbitant computational cost due to high-dimensional measurements. The robustification method models each pixel as an independent sensor which allows parallelization. To handle the fat-tailed measurement noise they replaced the actual measurement with a virtual measurement [4].

The point cloud matching method tries to minimize the mean squared error between the given point cloud by the depth image and the point cloud of the 3D-object mesh data. Here it is important to say that not all points of the mesh model are used to make the calculation online. The several point selection methods have their own respective advantages. For further information, we refer to the paper from Radkowski [7].

To reduce problems of unmodeled objects some papers introduce an additional observation model explicitly for occlusions [4, 19]. Issac et al. [4] implemented this by a uniform distribution.

Datasets: To get an idea of what we have to consider when we create our dataset, we looked up other datasets like the YCBInEOAT dataset by Bowen Wen et al. [17]. They pointed out that existing datasets have either static objects on a table, where the camera is moving around, or the objects are manipulated by hand. This means they are not suited for robot manipulation tasks because they do not consider additional data like forward kinematics. Also, there is not much video footage where the robot manipulates the objects in the scene. Therefore, our dataset should include manipulation tasks by a robot arm and also record forward kinematics which can be used to improve the tracker.

Other datasets [8, 12] have the problem that they mostly have humans as target objects [18], which is not what we need to train an object tracker for robot assembly. In our case, we need a dataset for a specific target object, the SL-Block. That is one of the reasons which motivates us to create a new dataset in the robotic assembly environment.

It is also important to provide ground-truth poses in every frame which is not always the case in current datasets [18]. So, we provide ground-truth poses of the target objects and the cameras since this information is essential to evaluate the tracker properly.

VisionLib: VisionLib [15] is a multi-platform augmented reality tracking library by Visometry. All the following information about VisionLib is from their official documentation [15], which we refer to for more details on the library. Its Unity, C, or Objective-C API makes it possible to implement AR applications at an industrial scale, but for this work we are only interested in the computer vision tracking technologies they use. It offers

- Model Tracking and State Detection,
- Marker and Feature Tracking,
- Multi-Camera and Multi-Model-Tracking,

as well as a combination of multiple tracking techniques. Since we do not want to use markers and would like to keep it rather simple at this point, we are focusing on Model Tracking and State Detection. VisionLib's Enhanced Model Tracking should help to overcome typical problems of AR like unstable light conditions and moving elements in the real world, so there is no need for preparations. Model tracking uses 3D and CAD data to detect, localize and track objects. The data is used as a tracking reference for the physical object to derive the edges of the 3D model, and the edges in the video stream to match them. The edges form a line model of the object. The better the 3D model matches the physical object, the better is the tracking. When the object is tracked, VisionLib calculates and delivers the core information to align the coordinate systems of tracking and 3D graphics. The usage of 3D models also explains the robustness against typical problems of AR. While VisionLib does not provide any further information on the algorithms behind the tracking API, the founders published a paper [10] whose principles might be used in VisionLib. In this paper, they propose a method that is based on direct image alignment between consecutive frames over a 3D target object. In comparison to established direct methods that only rely on image intensity, they also model intensity variations using the surface normal of the object under the Lambertian assumption.

Tracking Evaluation: Wu et al. [18] present attributes for a test sequence for a better evaluation of tracking algorithms. These attributes give an overview of what difficulties of tracking (like occlusion, fast motion, illumination variation, etc.) should be addressed. Therefore, we want to cover as many of these difficulties as possible to evaluate the tracker's robustness.

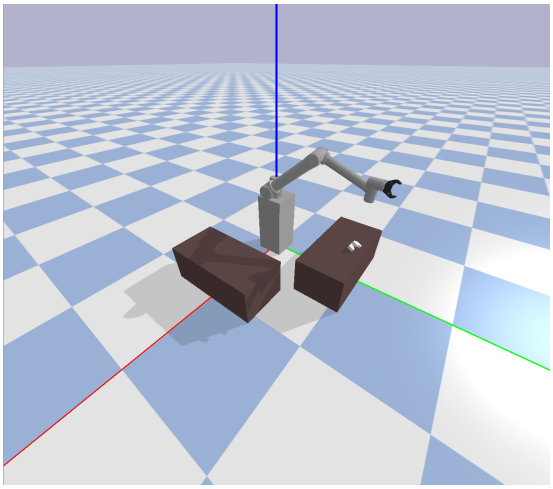


Fig. 2. Overview of the simulation environment with coordinate system. Each square of the blue and white grid measures $1\,\mathrm{m} \times 1\,\mathrm{m}$. red: x-axis green: y-axis blue: z-axis

TABLE I
CAMERA PROPERTIES

Properties	Values
Target Position	[0, 0, 0.5]
Camera up-vector	[0, 0, 1.0]
Height	1080
Width	1920
Field of View	100
Far value (range)	10
Near value (range)	0.02

III. Simulation Environment For Architectural Assembly

While recording data on the real robotic assembly setup is expensive and requires a lot of manual labor, using a simulation environment allows for easy adjustments and simple usage. Therefore in this section, we describe the PyBullet [1] simulation environment for robotic assembly we created. For further information on PyBullet, we refer to the official documentation [1].

A. Robotic Assembly Setup

The environment replicates a real robotic assembly setup. The most important part is the UR10 robot arm [14] which consists of six rotational joints. The attachment on top of it is the ROBOTIS RH-P12-RN gripper [9] which consists of four joints. The robot arm is placed on a block so that its base is up $35.5\,\mathrm{cm}$ over the plane which makes the robot's movements easier and more natural. Both are attached using the PyBullet planning library by Caelan Garrett [2]. The library allows for easy control and movement of the robot arm. The environment consists of two cuboid tables whose measures are $1\,\mathrm{m} \times 0.5\,\mathrm{m} \times 0.4\,\mathrm{m}$ for length, width, and height. The tables are modeled as simple GEOM boxes. The first table will be referred to as *pickup table* and is located at a distance of $85\,\mathrm{cm}$ along the y-axis in front of the robot arm's base, which corresponds to the direction of the y-axis of PyBullet (cf. figure 2). At the beginning of the simulation, an SL-Block is placed on top of the pickup table. The second table, which we call the *placement table*, is located at a distance of $85\,\mathrm{cm}$ along the x-axis from the arm's base and, at the beginning of the simulation, has no block on it.

TABLE II
ROS MESSAGE TYPES FOR ALL TOPICS

Topic	ROS Message	Description
depthimage	Image	Recorded depth image matrix
rgbimage	Image	Recorded RGB image matrix
block	Pose	Block pose in world coordinates
cam	Float64MultiArray	View & Projection matrix
jointname	Joint	Pose of each joint of the robot

B. Modular Camera

In addition to the described environment, we implemented a modular camera class in PyBullet. The advantages of using the modular camera are as follows:

- The camera can be placed anywhere in the environment which is very reasonable because in the real assembly setup the camera can also be put at any location. This allows for an easy alignment of the camera if the previously used position is not suitable for a certain tracking scene.
- The modular camera makes it possible to place multiple independent cameras simultaneously in the environment and is essential for multiple camera tracking. We also used this advantage for our dataset where we recorded from four different locations at the same time.

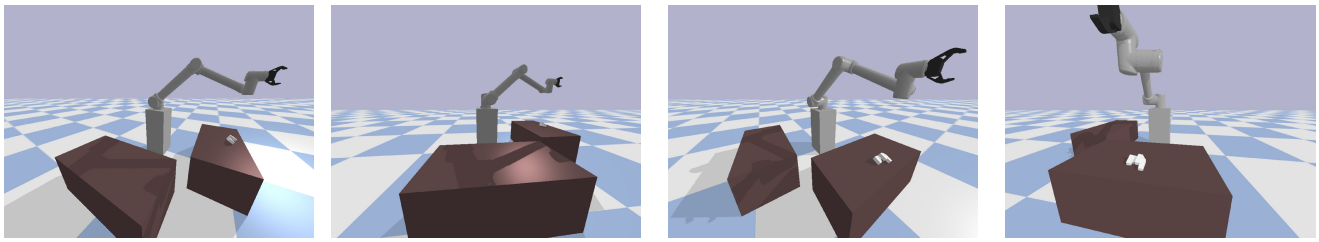
The camera records RGB-D images. Table I shows the properties we used for the modular camera. It is also possible to adjust these parameters, if needed.

C. ROS Integration

The environment is integrated with ROS [11] for easy data storage and processing. At each frame, the position of the block, the joint position of all joints of the robot, and the RGB-D images from all cameras are published. Each data type is published with its own topic. A topic for the RGB image and a topic for the depth image is used for each camera. Table II shows which ROS message types are used. Using existing ROS message types is appropriate here to make the data access comprehensible. The separation of data in different topics enables modular analysis of data and targeted access to relevant information. When using the ROSBAG tool [11], the time of data publication is also recorded which is a further advantage of the ROS integration. This makes the playback of the scene convenient and it is possible to get information about each frame correctly. The intended use of the individually recorded data is described in section IV.

D. Randomization

In the environment, it is possible to randomize the colors of the SL-Block, tables, and background. This randomization



(a) View from cam0 (b) View from cam1 (c) View from cam2 (d) View from cam3

Fig. 3. The different camera views which are used for the dataset.

enables the creation of an extensive dataset, and to train and evaluate object trackers under different circumstances.

IV. Dataset Creation

Object tracking is a very complex topic and faces several specific problems in the robotic assembly setup. A reasonable approach for handling these issues is making the algorithms learn from data. In this section, we outline the main difficulties of object tracking in the assembly setup and describe how we created a dataset for robotic assembly which is a first and fundamental step towards learning object tracking algorithms. The dataset can be accessed via [6].

A. Difficulties And Chances Of Object Tracking In Robotic Assembly

As stated in section II, object tracking has a lot of difficulties that good tracking algorithms must address and deal with. For the robotic assembly setup as mainly described in section III, the most important difficulties are as follows:

- Complex object motion.
- Handling noise.
- Handling occlusions.
- Most cameras are static. This is a problem because the cameras might lose track of the objects when they are too far away. The problem of static cameras is related to the problem of occlusions.

Considering these problems, a very reasonable approach is to make object tracking algorithms learn from data to have more robustness. Creating a dataset has multiple advantages:

- *Adapt to environment and specific parts.* As described in section III, mainly SL-Blocks are moved in the environment. For real-world assembly applications, it is also probable that there are only some specific parts that are moved. Therefore, our dataset allows us to specifically deal with SL-Blocks and adapt to their structure and dynamic properties.
- *Include kinematics for better training and evaluation.* Recording the kinematics (joint positions) makes it possible to use this data for the process model. This is an important advantage of robotic environments in comparison to e.g. human tracking, where reliable kinematics models are hard to obtain and rarely even exist.
- *Include uncertainties.* The many difficulties of object tracking are inevitable and good object algorithms especially have to address the problems of occlusions and noise. The most reasonable way to do this is using probabilistic methods as proposed in [13, 19, 3]. These methods usually require good measurements and evaluation data, which the dataset provides.

Because of these advantages, learning from data is a fundamental idea for this work. The basis of any good learning strategy for object tracking is a good and appropriate dataset. For this, we created a dataset using the simulation environment from section III. The dataset mainly focuses on the difficulties of complex object motion and static cameras.

TABLE III

Position of each camera in the environment

Camera Name	Position
cam0	[1.30, 0.70, 1.00]
cam1	[1.50,-0.15, 0.80]
cam2	[0.80, 1.30, 1.00]
cam3	[-0.25, 1.50, 0.80]

B. Simulation Setup And Dataset Recording

The dataset consists of one trajectory which is recorded by four cameras (cf. figure 3), under different randomizations (see section III-D) and is based on the simulation environment described in section III. For our evaluation purposes using one trajectory with different randomizations is sufficient. Despite that, the simulation environment allows to record other trajectories and extend the dataset, if that becomes necessary in the further process of developing an object tracker. In the following, we describe the concrete scenery which is recorded.

1) Camera Alignment: The scenery consists of four modular cameras (cf. figure 3) as described in subsection III-B which should be used to track the movements. The alignments of the cameras with their respective positions can be seen in table III. Here, the camera names correspond to the ROS topic names of each camera where the projection matrix and view matrix are published. This is shown in table II.

Using multiple cameras is necessary because some static cameras will always be too far away from the tracking object or the tracking object will be occluded when watching the scenery from a specific view. Therefore, it is necessary to include multiple camera tracking for object tracking in the

TABLE IV
INTERMEDIATE POSITIONS OF THE END EFFECTOR IN XYZ-COORDINATES

Step	End Effector Position
1	[0.00, 1.00, 1.00]
2	[-0.10, 0.96, 0.64]
3	[0.00, 1.06, 0.80]
4	[0.48, 0.95, 0.80]
5	[0.85, 0.63, 0.80]
6	[1.05, 0.18, 0.80]
7	[0.85, 0.40, 0.67]
8	[0.80, 0.00, 1.00]

robotic assembly setup. The dataset allows to either track the object for each camera independently or to combine the results to get the best position estimate.

2) Movement Trajectory Of The Robot Arm: The intermediate positions of the robot arm's end effector can be seen in table IV. Here the XYZ-Coordinates correspond to the coordinate system of the PyBullet environment, which is shown in figure 2.

- Step 1: The robot starts at the specified position.
- Step 2: The robot moves to the specified position and grasps the part by setting a constraint between its gripper and the block.
- Step 3-6: The robot performs a quarter-circular movement around the z-axis until it is above the placement table.
- Step 7: The robot drops the block.
- Step 8: The robot stops at the specified position.

3) Simulation Properties: PyBullet simulates the environment using simulation steps, where each step simulates 1/240 seconds. We recorded the dataset with 30 frames per seconds, which concludes that we recorded camera images every eighth PyBullet simulation step.

V. BRIEF EVALUATION OF VISIONLIB

To get a first idea of how current tracking libraries work with our dataset and how they handle object tracking in general, we used the object tracking library VisionLib [15] to evaluate its performance with our data. In addition to the recorded dataset for robotic assembly, we used simplified versions of the setup, mainly leaving out the robot arm and focusing on simple tracking of SL-Blocks. For a basic description of VisionLib and its functionality look into section II, for further details we refer to the official VisionLib documentation [15].

A. Simplified Environment For VisionLib Evaluation

To focus on specific difficulties which object tracking algorithms face and need to address, we created simplified versions of the environment. We aimed to record data from environments (cf. figure 4) with

- One moving block, two static blocks, and a static camera,
- A static block on a table with a camera moving around the block,
- A robot grasping the block from a table and putting it down on another table and a static camera,
- The same trajectory and setup but with an invisible robot.

In the first setup the SL-Block should fly around. We disabled gravity for that and applied force on the block. The applied force is proportional to the block's distance to a given target position. If the distance falls under a threshold, we do not apply force anymore. So, we chose four target positions that are reached sequentially. Following the trajectory, the block collides with one of the static blocks which causes rotation. This is of special interest regarding the evaluation.

In the second setup we removed one table and the robot, so there is only one block on a table. The camera orbits the block with a radius of 1.5 m.

The third setup is our dataset environment from section IV without simplifications. To make the robot invisible for our last scene, we changed the alpha value of the RGBA color in the URDF file of the robot and gripper from 1.0 to 0.0.

B. Image Sequences

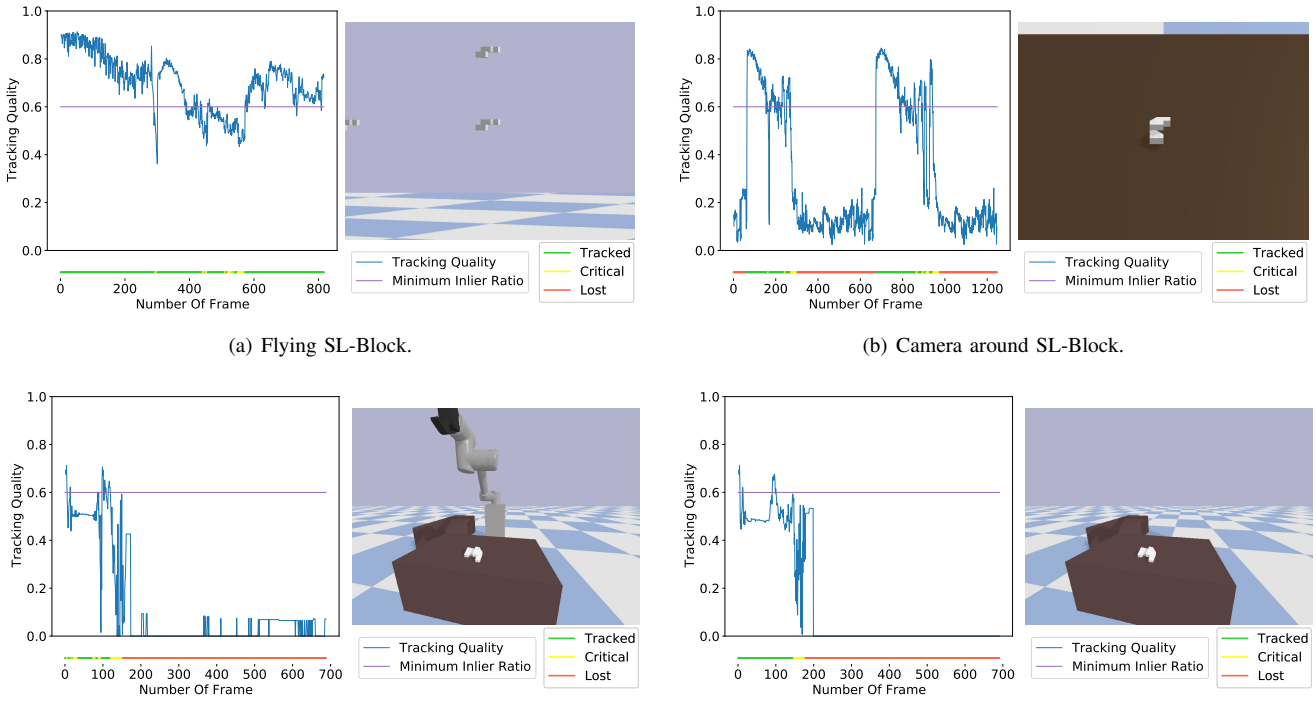
Instead of using a webcam or a mobile device as an input video stream, VisionLib allows using an image sequence in form of a folder of JPG or PNG files. So we recorded a JPG sequence of our simplified environments we introduced in subsection V-A.

C. Tracking Configuration File

The input has to be set in the tracking configuration files (.vl files) which are JSON files with a particular structure. The configuration files enable to control basic tracking behavior. Mandatory initial parameters of the configuration file are modelURI, metric, and initPose. The modelURI is the URI to the 3D file which is used as a tracking reference (in our case SL_Block.obj). The metric sets the corresponding unit size of the model in metric scales. This parameter has an immense influence on tracking quality. The initPose describes the pose of the object from which the tracking should start.

To create this file we used VisLab [16] which is Visometry's Tracking Configurator for VisionLib. We imported the object model and the image sequence into VisLab and aligned the model with the SL-Block in the first frame of the images (cam around block scene at the 64th frame).

There are optional tracking parameters that allow modifications and refine the line model and the image processing during tracking. There is the Laplace- or Contour Edge Threshold which influences the outer contour edges of the object, and the Normal- or Crease Edge Threshold which influences the crease or curvature edges of the model. We kept these values at their default value because the SL-Block is very simple and the line model fits well. Detection- and Tracking Thresholds like the minInlierRatioInit, which influences the detection sensitivity, and the minInlierRatioTracking, which influences the tracking sensitivity, describe the minimum ratio between parts of edges that are found and not found to detect and track the object. Therefore they have values between 0.0 and 1.0. There are advanced tracking parameters for a contrast threshold, detection radius, tracking radius, and a



(a) Flying SL-Block.

(b) Camera around SL-Block.

(c) Trajectory with robot arm.

(d) Trajectory with invisible robot arm.

Fig. 4. The four plots show the tracking quality of VisionLib's tracker at each frame for each evaluated tracking sequence respectively. The purple line describes the minimum inlier ratio which serves as a threshold to determine when the object is considered to be tracked or not, while the blue line shows the tracking quality. The bar under the plot displays the tracking state at each frame. On the right of each plot is a picture of the corresponding environment. 4(a) shows the scene with the flying SL-Block, 4(b) shows the scene where the camera circles around the SL-Block, 4(c) shows the scene where the robot arm grasps the SL-Block from the table and lays it down on the other table, and 4(d) shows the same scene as 4(c) but with an invisible robot arm.

keyframe distance (keyframes let the line model update and are used as recovery points beyond the initial pose when tracking is lost). We kept these at their default values, too.

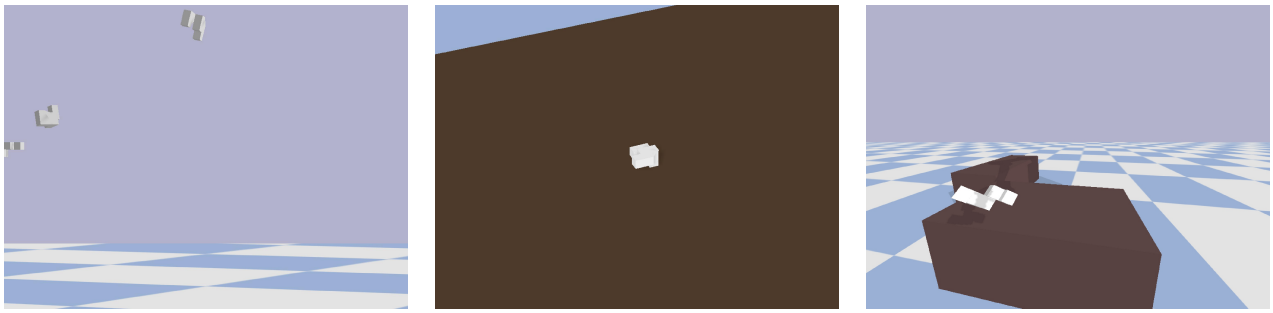
D. Unity And Evaluation

To evaluate VisionLib we used the vlUnitySDK API together with Unity 2018.4. In Unity, we first had to import the VisionLib library. We then took the example for simple model tracking and adapted it accordingly. For that, we replaced the existing object with the SL-Block and imported the tracking configuration file we created with VisLab.

To get a metric for evaluation, we wrote a C#-Script which writes the image frame number, the timestamp, the tracking state, and the quality of the tracking in a text file. For that, we used the VLTrackingState.TrackingObject class which stores the necessary information. The tracking state has three values: tracked, critical, and lost. A critical state means that the tracking is unstable and the object might be lost soon. The quality takes values from 0.0 (worst value) to 1.0 (best value) and represents VisionLib's confidence in its tracking results. The quality is connected to the minInlierRatio parameters and needs to be above these values to classify an object in a specific frame as 'tracked'.

The tracking state is also indicated by its color (green $\hat{=}$ tracked, yellow $\hat{=}$ critical, red $\hat{=}$ lost). Figure 1 shows how each state is displayed in Unity using the scene with the robot.

Figure 4 shows the tracking results using VisionLib. The tracking quality appears to be relatively high when the tracker starts to track an object, but then decreases with the number of frame. The tracker can also handle movements, but rotations result in more critical states (cf. figure 4(a)) or even losing the object (cf. figure 4(b)). When the flying SL-Block collides with the other block at frame 284 in figure 4(a), the tracking quality reaches a minimum and the state becomes critical, but it re-detects the block fast. In the following frames, the target object is rotating which results in a lower tracking quality (cf. figure 4(a)). In comparison to the trajectory with the invisible robot, the tracking with the robot is switching in the critical state more often and the plot (cf. figure 4(c)) does not look as stable as the invisible robot plot (cf. figure 4(d)). In addition, the state becomes critical when the robot grasps the object in frame 89 which can be seen in figure 1(b). You might conclude that occlusion is a problem. Despite that, the trajectory with the invisible robot shows a very similar plot as the trajectory with the robot where we have occlusion. Therefore, we conclude that occlusion does not seem to be the main problem here as we thought at first, because the tracker even loses track of the object before the occlusions appear. We assume that the fast movements cause the bad tracking here, too. Another difficulty occurs when the SL-Block does not reveal a lot of its 3D structure and orientation as in the frames in figure 5 and in the first frame of the camera-around-block sequence which can be seen in figure 6(a). We also



(a) Critical pose in flying block video. (b) Critical pose in camera around block video. (c) Critical pose in invisible robot video.

Fig. 5. Positions where VisionLib's tracking state becomes critical

(a) First frame. (b) Initial pose frame.

Fig. 6. 6(a) shows the first frame from the camera-around-block sequence. 6(b) is the frame we used to set the initial pose.

had to change the initial pose from the camera-around-block setup from the pose at the first frame (cf. figure 6(a)) to a pose which reveals more of the block's 3D structure (cf. figure 6(b)). That is the reason why the tracking quality is low at the beginning of the plot in figure 4(b). Another point that can be observed in figure 4(b) is that once the tracker loses the object it does not start to track again unless the initial pose of the 3D model matches the object in the video again. This can be seen in the video with the camera moving around the block, and also in figures 4(c) and 4(d).

Looking at the results, especially at the scene with the robot, VisionLib does not seem to be suitable for robot assembly. That is because there are a lot of movements that lead to losing the object. When the object randomly gets into its initial pose again, the tracker is able to align the model and the object is tracked again. Since that is unlikely, the tracker stays in the lost state.

VI. Summary Of Work And Outlook

In this work, we dealt with the problem of object tracking for robotic assembly. Object tracking plays a fundamental role in the assembly setup because we want to use and incorporate the visual data from the cameras as well as poses and kinematics from the robot arm for optimal movements (e.g. around obstacles), optimal part grasping and part stacking. While there is a lot of prior work on the topics of object tracking in general and uncertainties in robotics, problem-specific object tracking algorithms for robotic purposes and related datasets are rare. Therefore we created a PyBullet [1] simulation environment for the robotic assembly setup where we can simulate the robotic movement and behavior with precise ground-truth parameters. We used this environment for creating a dataset to have object tracking algorithms learn from data and adjust to the specific environment. The dataset includes one trajectory where an SL-Block is picked up and moved to another table. We used four cameras at different positions and included randomizations of the colors of the background and the parts in the environment. As a first approach to track an SL-Block in our environment we used the AR tracking library VisionLib [15]. To get an idea how good VisionLib works for our purpose we evaluated it with simplified environments including the SL-Block. The results show that it is not suited as an object tracker for robot assembly because it loses track too often and re-detection is only possible if the object strikes its initial pose again. We also had to manually choose the initial pose for every tracking situation. The main problems the tracker seems to have are fast movements and views of the SL-Block which do not show its structure well. VisionLib cannot handle the depth information we record either. So, our next step is to look at other approaches to track the SL-Blocks.

Starting from our results, there are several interesting future research topics for this work. The first idea would be to create a dataset on the real robotic setup. This would make it possible to have more realistic learning data and to consider the problem of Sim2Real transfer. The topic of Sim2Real is important because our dataset only works with ground-truth parameters which have near-zero noise and uncertainties, while on the real setup you cannot assume that to be true. That is why it is very reasonable to include probabilistic filtering methods, as mentioned in sections I and II, and we will put a focus on such methods.

An important aspect of the setup which is not considered that much yet is the stacking of SL-Blocks. For the architectural assembly it is necessary that when parts get stacked over each other, they have to be treated as one new independent part. You can expect that if this stacking is not incorporated into the tracking algorithm, there will be issues with reliable tracking because of the heavy occlusions which occur in these situations. Therefore, it is a good idea to include part stacking when we record the dataset on the real setup, to make the algorithms also learn this aspect of assembly.

Generally, occlusions are not covered in big detail yet,

while they are a big problem in object tracking and specifically in the assembly gym. We have not considered that yet because as shown in section IV and V, even without big occlusions there are many tracking problems, which mainly occur because of complex movements and static cameras. Therefore, currently only one part is moved and the gripper does not actually grasp but rather set a constraint on the block. To have reliable tracking it is necessary to also consider occlusions in greater detail.

Another considerable aspect is multiple camera tracking. As shown in section III, one static camera may not be enough to track the whole environment. Therefore it is reasonable to include multiple cameras and combine their results. For this, probabilistic methods like particle filters can help a lot.

As main goal for the future we want to contribute a good and reliable object tracker for the robotic assembly project. Especially the results from section V show that problem-specific algorithms are desirable to have stable tracking results. Our dataset provides a good basis for learning object algorithms on the assembly setup. Starting from that, we will evaluate several general tracking methods as described in section II and use that to develop an object tracker for the setup.

REFERENCES

[1] Erwin Coumans and Yunfei Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org. 2016–2020.

[2] Caelan Reed Garrett. *PyBullet Planning*. https://pypi.org/project/pybullet-planning/. 2018.

[3] *GitHub - rlabbe/Kalman-and-Bayesian-Filters-in-Python: Kalman Filter book using Jupyter Notebook. Focuses on building intuition and experience, not formal proofs. Includes Kalman filters,extended Kalman filters, unscented Kalman filters, particle filters, and more. All exercises include solutions.* https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python. (Accessed on 02/07/2021). Oct. 2020.

[4] Jan Isaac. *Depth-Based Object Tracking Using a Robust Gaussian Filter*. IEEE International Conference on Robotics and Automation (ICRA), 2016.

[5] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[6] Max Siebenborn Leon Magnus Svenja Menzenbach. *Tracking Dataset For Robotic Assembly*. https://git.ias.informatik.tu-darmstadt.de/tactile_robotic_assembly/object_tracking/ip_object_tracking. Mar. 2021.

[7] Rafael Radkowski. "Object Tracking With a Range Camera for Augmented Reality Assembly Assistance". In: *Journal of Computing and Information Science in Engineering* 16.1 (Jan. 2016). 011004. ISSN: 1530-9827. DOI: 10.1115/1.4031981. URL: https://doi.org/10.1115/1.4031981.

[8] Seng Keat Teh Robert Collins Xuhui Zhou. "An open source tracking testbed and evaluation web site". In: (2005). eprint: https://www.ri.cmu.edu/pub_files/pub4/collins_robert_2005_1/collins_robert_2005_1.pdf.

[9] *ROBOTIS Hand RH-P12-RN*. https://www.robotis.us/robotis-hand-rh-p12-rn/. (Accessed on 03/09/2021).

[10] Byung-Kuk Seo and Harald Wuest. "A Direct Method for Robust Model-Based 3D Object Tracking from a Monocular RGB Image". In: Oct. 2016. ISBN: 978-3-319-49408-1. DOI: 10.1007/978-3-319-49409-8_48.

[11] Stanford Artificial Intelligence Laboratory et al. *Robotic Operating System*. Version ROS Melodic Morenia. May 23, 2018. URL: https://www.ros.org.

[12] *TB-100*. http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html. (Accessed on 03/09/2021).

[13] Sebastian Thrun. *Probabilistic robotics*. Communications of the ACM, 2002.

[14] *Universal Robot UR10e*. https://www.universal-robots.com/products/ur10-robot/. (Accessed on 03/09/2021).

[15] Visometry. *VisionLib*. https://docs.visionlib.com/v20.11.1/. (Accessed on 03/08/2021).

[16] Visometry. *VisLab*. https://visionlib.com/products/vislab/. (Accessed on 03/08/2021).

[17] B. Wen et al. "se(3)-TrackNet: Data-driven 6D Pose Tracking by Calibrating Image Residuals in Synthetic Domains". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, Oct. 2020. URL: http://arxiv.org/abs/2007.13866.

[18] Y. Wu, J. Lim, and M. Yang. "Object Tracking Benchmark". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37.9 (2015), pp. 1834–1848. DOI: 10.1109/TPAMI.2014.2388226.

[19] M. Wüthrich et al. "Probabilistic Object Tracking Using a Range Camera". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Nov. 2013, pp. 3195–3202. DOI: 10.1109/IROS.2013.6696810.

[20] Alper Yilmaz. *Object Tracking: A survey*. ACM Computing Surveys, 2006.