Augmented Reality Robot Operation Interface with Google Tango

Michael Gradmann, Eric M. Orendt, Edgar Schmidt, Stephan Schweizer, and Dominik Henrich
www.robotics.uni-bayreuth.de, Universität Bayreuth, Bayreuth, Germany

Abstract

In this paper we present a novel smart-device-based robot operation interface providing augmented reality (AR) and utilizing a device equipped with self-tracking functionality and a depth camera according to Google Tango reference design. The robot can be both programmed remotely by our Java application and taught by kinesthetic programming. The system also provides an object detection that is capable of recognizing objects. Additionally we designed and investigated an adaptive program execution. The ease of use is ensured by AR features such as the highlighting of detected objects and the preview of robot operations. The main contribution is a robot operation interface that can be multifariously programmed and is capable of adapting to changing surroundings based on commercially available devices.

1 Introduction

Although the fields of research covering robots like humanoids, social robots, dialog systems including robots or human robot interaction are popular among scientists all over the world, most robots by far are installed in industrial environments. In these surroundings, robots are executing tasks that are usually repeated with no or minor changes of environment. Due to textual programming, reprogramming the robot has to be done by experts as the knowledge of a programming language is compulsory.

As such an adaption results in high costs, a high number of repetitions is necessary to run an automated production plant profitably. Tasks in smaller medium-sized enterprises or households mostly show small lot sizes in contrast. Additionally, tasks can vary occasionally and might be performed in a non-static environment. Hence an automation of these tasks requires a system that can be adjusted to the tasks with minor effort and has to be able to adapt to changes of its environment automatically. Robot systems meant for household applications furthermore need to offer programming interfaces that can be operated by users with little prior knowledge or programming skills.

The number of smartphones, tablets and smart devices in general increased impressively during the last ten years, proving that many people are not only capable of administrating such a device. Beyond that, this kind of interaction makes these devices accessible for people regardless of age, social background, or proficiency in computer science. Accordingly, one can deduce that instructing robots with such a well established device makes automation accessible for a large group of people as well. However, the system requires 3D information to be aware of its surroundings that smart devices usually do not provide.

In our approach, we consequently utilized a tablet computer to provide a remote control. To avoid the use of further external sensors, we realized our Augmented Reality Robot Operation Interface (AuRROc) utilizing Google Tango reference design (formerly Google Project Tango), which provides depth information and a sophisticated self-tracking of the device. **Figure 1** shows a typical programming situation with our system. Our system design comprises basic robot commands, augmented reality information, object detection, different programming approaches, and an adaptive program execution.

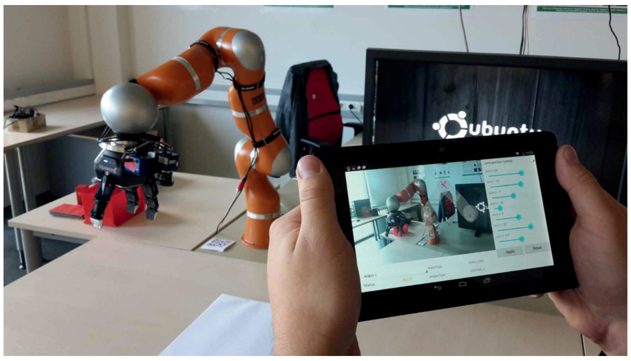


Figure 1 Robot operation interface using augmented reality support. The Google Tango equipped smart device enables non-experts to command and program a robot manipulator.

2 Related Work

A high number of researchers in the field of robotics focuses on the integration of robot platforms to new application domains like private households or small and medium-sized enterprises (SMEs). This means that an increasing number of non-experts will have to operate robots to solve their individual tasks. To allow them to do so, we identified two major requirements: First, a user interface with a high degree of usability and the opportunity to provide necessary information from a workspace. Second, an intuitive operating paradigm that allows to use almost the full capability of a general purpose robot.

From our view, interfaces with augmented reality (AR) meet the first requirement most closely. AR allows to embed and visualize additional information about real world structures. From this, AR can help to overcome a user's perceptual limitation when collaborating with robots [1]. There are several approaches for AR-based interfaces, which have shown that AR matches established guidelines for efficient Human Robot Interaction (HRI) [2]. Most of the work in this field covers different ideas of gesture-based commands or graphical user interface design [3, 4, 5, 6, 7, 8]. Other approaches, not only from a robot application domain, focus on the technical aspects of spatially correct object projection, alignment control, or image overlay [9, 10].

Another requirement, an intuitive operating and programming paradigm, is part of robotics research for some decades. One of the most popular approaches to allow the use of robots with little prior knowledge is the Programming by Demonstration (PbD) paradigm [11]. The core idea of PbD is to let a user demonstrate a certain task solution in a certain situation and to reproduce this solution later in similar situations [12]. A well-established approach from this field is kinesthetic programming, which means that a user is guiding the robot directly through a task, while the robots intrinsic force sensors record the necessary program information (e.g. a trajectory). A niche of kinesthetic PbD is so called One-Shot Programming, where only a single demonstration is shown to create a model of the task solution [13]. Hence, it offers a very intuitive method to enable robot programming for non-experts [14, 15]. However, most of the One-Shot PbD approaches utilize some kind of model (like key frames [16], oriented particle chains [17] or state machines [18]), which is filled by different sensor information regarding the task solution. In [19] and [20] a graphical user interface running on a stationary workstation visualizes the reproduction of a demonstrated task solution and allows editing as well. There also exists a framework utilizing a purchasable smart device as a user interface for robot programming [21, 22]. From our perspective, this is the most promising approach to bring intuitive, mobile interfaces to non-expert domains.

A remaining problem is that the shown approach in [21, 22] uses an intuitive interface, but chooses a purely virtual programming paradigm based on a tablet surface. Furthermore this virtual environment may produce tasks, which can not be solved by the robot system. From our perspective, it is possible to overcome this limitation by the use of kinesthetic programming, because the non-expert user can directly find out which task can be solved by the provided robot system (e.g. regarding joint angle limits or workspace limitations) and which can not.

Our contribution is a framework using a purchaseable consumer smart device that enables a non-expert user to easily generate robot programs with an AR supported preview of virtual robot commands or directly by kinesthetic programming. Both remote programming and kinesthetic programming represent two flavors of the well known playback programming. Both approaches record instructions and will replicate those instructions regardless of any change of the environment. The idea to switch between two operating paradigms follows the findings from [23], where is shown that there is no optimal paradigm in general, since users prefer different input modes depending on the task and individual attitudes. Further, we use built-in sensors only to get the opportunity to integrate the framework easily on every upcoming smart device with a depth sensor and self tracking functionality. Thus, we merge intuitive approaches from both interfaces and operating paradigms to provide a robot system that is easy to use in a non-expert domain.

3 System Architecture

This section gives a comprehensive overview of the proposed application and the most relevant hardware components regarding the user interface and the robot platform.

3.1 Google Tango

Google Tango describes a reference design for Android devices that comprises tracking functionalities and additionally provides depth information. In **Figure 2** the Google Tango Developers Kit is illustrated exemplarily. A Tango device is equipped with three cameras:

- A standard RGB camera
- A depth camera including an active infrared device (the sensor is not specified in particular, there are Tango devices with active triangulation or time-of-flight sensors)
- A wide-angle camera for self-tracking, providing a solution to the SLAM problem (Simultaneous Localization and Mapping)

The latter cameras differentiate the Google Tango device from usual smart phones or tablets. In the wide-angle images, several key points are set automatically and searched for in the next image of the stream. Hence the movement of the device can be calculated continuously based on the image stream of the wide-angle camera. To improve the quality of the visual odometry, the software solution takes acceleration information into account as well. Both tracking and depth information are utilized in this work.

3.2 Setup

For our experimental evaluation we use two KUKA Lightweight Robots (LWR) and the Google Tango Devel-

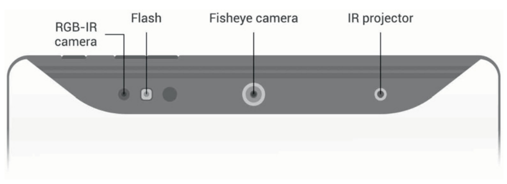


Figure 2 Google Tango reference design with three cameras (RGB and depth camera are combined in this example), exemplary Google Tango Developers Kit, edited, courtesy of [24].

opers Kit running Android 4.4. The mobile device application running on the tablet is programmed in Java. A basic gripper system is mounted to each of the robots. The LWR is a 7 DOF robot with a maximum load of 7 kg, which allows to handle most everyday objects. The used kinesthetic robot control enables a user to guide the manipulator through a task, based on so called gravity compensation [25]. As working area we use a plain desk. The robot system is connected to a computer, which processes the command and parameter information sent bidirectionally as structured string via a TCP-based network. In this design the tablet acts as server, which the robot clients are connected to. **Figure 3** provides a schematic overview of the system. The following robot commands were implemented for this work and can be called across the network:

- Apply changes to robot's joint configuration
- Reposition the tool center point and invoke motion
- Change the robot's control modes (position control, gravitation compensation)
- Access gripper controls (adaptive due to different hardware configurations)
- Perform pick-and-place task based on the commands described above

Besides, we realized further commands and client information such as disconnect, emergency stop, status information (READY, BUSY, OFF), set impedance constraints and set velocity, acceleration and jerk. These commands provide a comfortable user interface when performing experiments, but are not focused on in this work. In our laboratory there are manipulators equipped with different hardware available, e.g. one LWR is equipped with a Robotiq 3-finger gripper and another LWR is equipped with a servo electric 2-finger-parallel gripper from SCHUNK. The client passes

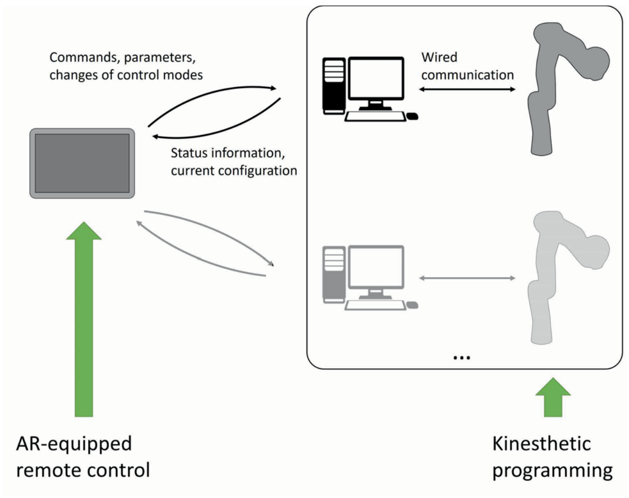

Commands, parameters, changes of control modes
Status information, current configuration
Wired communication
...
AR-equipped remote control
Kinesthetic programming

Figure 3 System setup as used in our approach. The system allows user input through both remote control via smart device and kinesthetic programming. The green arrows indicate, that the user can access the system through two ways.

the known hardware configuration through a login string to the server when the connection is established. The interface can register several robots at the same time. From this, our framework allows to switch the interface to a specific manipulator and sequentially command or program any number of robots.

4 Prototype Features

To be able to operate the adaptive program execution, the presented system has to provide several functionalities we implemented on the smart device. The most important software solutions are presented in this section.

4.1 AR Elements

Our application features an augmented reality representation of our scene. All AR elements are rendered as superposition in the viewport of the RGB camera and remain located correctly when the smart device is moved.

- The tool center point (TCP) is displayed in the camera image and can be repositioned by the mobile device application. Hence the gripper location can be adjusted in AR prior to task execution.
- A virtual robot (i.e. a semitransparent robot model) is superimposed on the camera image after registration (see Section 4.2). The accuracy of the registration can be easily checked on the display. The virtual robot is updating its position as soon as the joint configuration is changed, hence providing a preview for robot movements before they are executed on the actual hardware.
- Detected objects are highlighted by axis-aligned bounding boxes (AABB). Moreover, the color of a box indicates the object's average color.
- The limits of the working area are indicated by a wireframe cuboid in the image.

4.2 Registration

In order to be able to display robotic information located correctly on the screen or to acquire valid position information of objects using the tablet, it is necessary to detect the position and orientation of the tablet with respect to the robot coordinate system CS_R. The coordinate system CS_T is stationary with respect to the Google Tango device. As the device can be moved, the location and orientation of the coordinate system at different time steps $CS_{T,\,t_{i-1}}$ and $CS_{T,\,t_i}$ can differ from each other. **Figure 4** illustrates the mentioned coordinate systems and their relations.
In this approach, CS_R is defined as the world coordinate system. Data referring to CS_R can be easily transformed to the tool center point coordinate system CS_{TCP} according to (1), as the robot's geometry is known in Denavit-Hartenberg parameters and its joint configuration is sensor monitored. Describing the coordinate transformation from system i to j with the matrix M_i^j the forward transformation

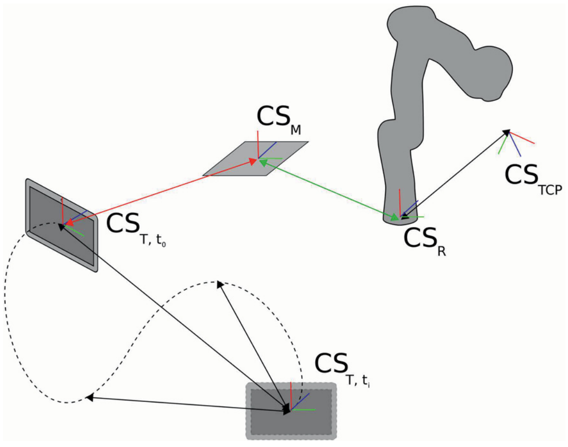


Figure 4 Different coordinate systems that can be converted into each other. Note that all poses CS_{T,t_k}, $t_0 < t_k < t_i$ can be retrieved from the Tango framework. The arrow illustrated in green indicates a transformation that are measured conventionally, black arrows are obtained by robot kinematics or Google Tango tracking and the transformation displayed in red is retrieved through registration.

can be written as

$$M_R^{TCP}(\theta) = M_{JCS_n}^{TCP}(\theta) \cdot M_{n-1}^{JCS_n}(\theta) \cdot \ldots \cdot M_R^{JCS_1}(\theta) \quad (1)$$

where n is the number of robot joints, JCS_k refers to the joint coordinate system of robot joint k and θ is a vector of length n containing the angles of each joint.

While working with the robot, the tracking of the smart device is performed via Google Tango functionality (see Section 3.1). Referring to the latest tracking information at time t_i, the transformation based on the pose known at the time t_{i-1} is calculated by the matrix $M_{T,t_{i-1}}^{T,t_i}$. Hence the transformation to a time point t_0 can be calculated analogously to (1):

$$\begin{aligned} M_{T,t_i}^{T,t_0} &= \prod_{k=1}^{i} M_{T,t_k}^{T,t_{k-1}} \\ &= M_{T,t_1}^{T,t_0} \cdot M_{T,t_2}^{T,t_1} \cdot \ldots \cdot M_{T,t_i}^{T,t_{i-1}} \\ &= M_{T,t_{i-k}}^{T,t_0} \cdot M_{T,t_i}^{T,t_{i-k}}. \end{aligned} \quad (2)$$

Note that any pose at timepoint t_k, $t_0 < t_k < t_i$ can be obtained at time t_i.

However, it is necessary to initially register the device's coordinate system with respect to CS_R. In this approach a QR marker (Quick Response Marker) is utilized to determine the tablet's initial pose CS_{T,t_0}. The QR marker is located on the table in a known distance and rotation relative to the robot providing the static transformation matrix M_M^R. The information is mapped to the depth image, hence providing the translation from tablet to marker T_{T,t_0}^M. The orientation of the tablet compared to the marker R_{T,t_0}^M is calculated based on the QR marker's key points forming a right angle and their surface normal gathered from corresponding depth information. Thus, the pose of the tablet with respect to the marker is described by

$$M_{T,t_0}^M = R_{T,t_0}^M \cdot T_{T,t_0}^M. \quad (3)$$

With the results from (2) and (3) the camera coordinates can be converted into CS_R by calculating

$$M_{T,t_i}^R = M_M^R \cdot M_{T,t_0}^M \cdot M_{T,t_i}^{T,t_0} \quad (4)$$

where M_{T,t_i}^{T,t_0} is changing with the current time step t_i.

With this approach we avoid the restriction of 2D-RGB camera based marker detection, which requires the exact knowledge of the size of the marker to be able to determine orientation and distance in a camera image.

After a successful registration, the virtual robot is rendered to the viewport and a wireframe AABB indicates the working area of the system. The implementation provides a manual correction that allows to shift the registration alongside the axes of CS_R.

4.3 Object Detection

The basic concept of the provided object detection assumes that the working area consists of an empty plane. All areas that are higher than the table are interpreted as objects (also see Section 4.2). In this approach, we also consider the color information of each object.

The object detection works in three steps. In the first step, the depth image is matched with the color information from the RGB camera. Afterwards, the point cloud P is converted to robot coordinates CS_R and cropped to reduce data to object candidates. The cropped point cloud P_C consisting of points $\vec{x}$ is defined as indicated in (5).

$$P_C \leftarrow \{\vec{x} \in P \,|\, \vec{x} \text{ located in working area}, x_z > o\} \quad (5)$$

The offset o is equivalent to the minimum height of the objects. In the second step, the relation of the points in the remaining point cloud is checked based on the DBSCAN algorithm (Density-Based Spatial Clustering of Applications with Noise, see [26]). The set of clusters C can be formally described by $C \leftarrow \text{DBSCAN}(P_C, \varepsilon, minPts)$, where ε defines the maximum distance of adjacent points p_i and $minPts$ sets the minimum number of points that have to be *directly reachable* from a point p within the radius ε to be interpreted as a cluster.

In the last step the corresponding color information is obtained. We calculate the average color value of all pixels

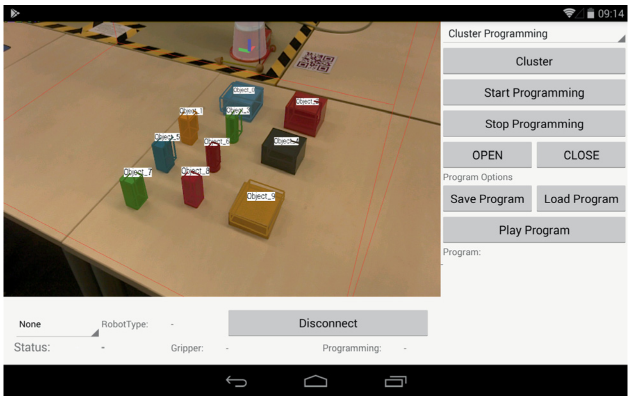


Figure 5 Example of ten objects successfully clustered with DBSCAN. The objects are highlighted with axis-aligned bounding boxes colored in object's average color.

that belong to an object. The color of the edges of the bounding box indicates the resulting average color value. **Figure 5** shows a screen image with detected objects.

In our approach we optimized the DBSCAN algorithm, whose runtime on unordered point clouds is bounded by $\mathscr{O}(n^2)$ (with n points to be clustered), leading to runtimes of several seconds in the first draft. By creating a k-d tree as an ordered data format from the point cloud (which is possible at a cost of $\mathscr{O}(n\log(n))$), the DBSCAN algorithm cost is reduced to $\mathscr{O}(n\log(n))$.

5 Intuitive Robot Programming Approaches

To facilitate the automation of tasks, our framework offers different robot programming modes. Both remote instructions and kinesthetic programming are possible and will be described in the following.

5.1 Program Representation

In a first step we realized a program representation that is capable of storing commands and allows to combine several commands sequentially. An inserted program can be compartmentalized into basic robot commands as introduced in Section 3.2. Robot programs are stored as scripted sequence of robot commands, formatted to

$$\texttt{COMMAND arg1 arg2 ....} \tag{6}$$

In **Code Example 1** a short example is given representing a pick-and-place task. Once the program is loaded, these robot commands will be processed consecutively.

```
PICK -0.59004366 0.27676687 0.07024907
PLACE -0.6456661 -0.26341385 0.25003824
MOVEJOINT 0.0 0.27925253 -0.10471964
  -1.3089969 -5.56E-4 1.76422 2.148943
```

Code Example 1 Example of command sequence of a pick-and-place task

The programming approaches as described in Sections 5.2, 5.3 and 5.4 store the taught programs according to this syntax. The files are accessible and can be edited with an Android text editor.

5.2 Remote Programming

Based on the basic robot commands as introduced in Section 3.2 we implemented a remote programming functionality. Therefore, the application provides buttons for each command. Some commands allow the adaption of parameters via slide controls. The commands can be invoked directly or stored to a program file. For each step, the application provides a preview through AR elements as described in Section 4.1. The command sequence is stored on the smart device and can be loaded and executed at will.

5.3 Kinesthetic Programming

To allow kinesthetic programming, the robot has to operate in gravitation compensation mode (see Section 3.2), which can be activated likewise in the application. Subsequently, the robot is moved by the user and the robot's trajectory is recorded as soon as a certain change of the joint configuration is detected. The gathered information is mapped to basic robot commands and can be stored to a file similar to Section 5.1.

5.4 Adaptive Program Execution

Based on the given information and sensors provided by Google Tango, we implemented an automated program adaption. In the first step of this approach, the object detection is called to find objects including their average color information. In the second step, the robot is taught to grasp an object and release it at another position. The object is identified by its position during the teaching process and its corresponding color is additionally stored to the program as described in Section 5.1.

When the program is loaded from storage, the scene is clustered again before the program is started. The position information in the program is updated with respect to the new clustering results. Based on their color information, the objects are recognized, e.g. the object with the color most similar to the stored color is identified as the object to grasp. The object's position information is updated to the latest cluster results and the program is adapted. Hence the pick and place operations are adapted to the new scene and can be executed correctly considering the new positions of the objects.

6 Experiments

In order to evaluate our system and to demonstrate its capabilities, we performed experiments to show the appropriate runtime and reliability of the object detection algorithm. The corresponding experiments and results are explained in Section 6.1. In Section 6.2, we present the experiment design and results of the adaptive program execution tests (also see Section 5.4).

6.1 Experiments on Object Detection

We implemented the object detection as described in Section 4.3. The entire algorithm is processed on the Project Tango Developers Kit running Android 4.4. First we investigated the overall object detection runtime subject to the number of clustered points, as it highly influences the applicability of the algorithm.

For the measurements the tablet was mounted on a tripod facing towards the working area. The plane of the working area was found by registration as described in Section 4.2. From that scene we made 59 measurements with up to 14 objects. To provoke a worst case runtime we also reduced the assumed working plane altitude, so that the entire table was interpreted as an object and a large number of points had to be taken into account. The results are presented in **Figure 6**. The maximum DBSCAN runtime was measured at 198 ms. Besides the chart shows the expected trend as the DBSCAN algorithm runtime increases with the number of clustered points.

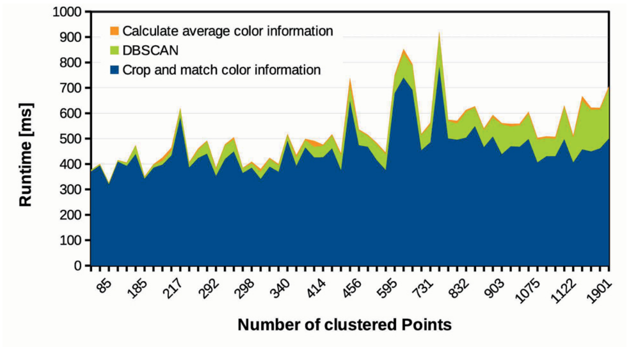
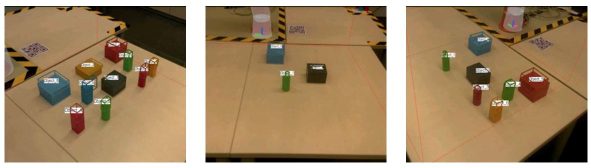


Figure 6 Object detection runtime split into crop (matching RGB camera data and crop point cloud to working area), DBSCAN (find clusters in the point cloud) and color information calculation (calculate average color information based on the identified clusters) with respect to the number of clustered points.

Whereas the calculation of the clusters' average color information hardly influences the entire runtime, the crop and color matching highly contributes to it. In this part of the approach, the color information from the RGB camera is matched to the point cloud. Then the data is transformed to robot coordinates and the points located in the working area are identified. The cost of that part of the algorithm scales with the number of points in the original point cloud, which is approximately constant. Although the runtime measurements result in a roughly constant value of about 400 ms, there is significant noise visible that almost doubles the expected runtime on rare occasions. Our data prove that there is neither a dependency of these values to the overall point cloud size nor to the number of objects to be clustered. Hence we assume that background processes are invoked randomly by the device's operating system, such as the Java garbage collection, that lead to these peaks in runtime.

To summarize we achieved an object detection algorithm with a maximum measured runtime of 920 ms. Even in worst-case scenarios we did not measure runtimes larger than one second, meaning that the entire implementation is appropriate for our experiments. None of the participants working with the mobile device application experienced the achieved runtime as limiting or uncomfortable.

To prove the capability of the implementation, we evaluated the object detection reliability as well. In the setup the Google Tango device was also mounted on a tripod. We



(a) Ten objects, P1 (b) Three objects, P2 (c) Seven objects, P3

Figure 7 Different successfully clustered scenes from positons 1 to 3 (P1 - P3) as conducted for cluster reliability measurement.

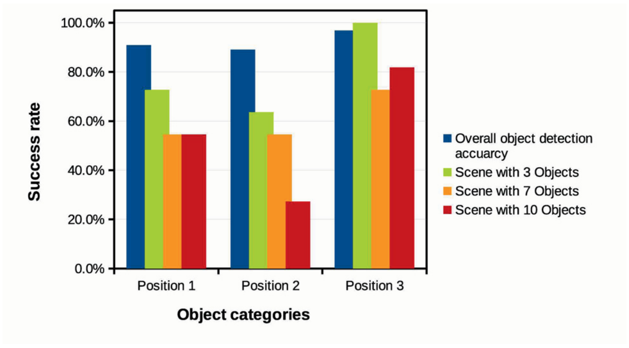


Figure 8 Object detection accuracy and scene accuracy measured with three, seven and ten objects from three different perspectives.

clustered scenes with three, seven and ten objects eleven times from three different perspectives. **Figure 7** gives an insight on the scene from selected perspectives.

We evaluated the measurements by calculating the frequency of detected objects based on two different indicators. To find a general reliability, the number of detected objects is divided by the overall number of objects of all depth images. The overall object detection accuracy hence calculates to $\frac{609}{660} = 92.3\,\%$. **Figure 8** documents furthermore, that the object detection accuracy is not significantly influenced by the position of the objects.

Moreover, we evaluated the scene accuracy, which is compulsory for the adaptive program execution. A scene is clustered accurately, when *all* objects in the scene are clustered correctly in size and color. As this criterion is more restrictive than the previously presented object detection accuracy rate, naturally the numbers degrade. Interestingly though, a position dependency can be observed in this evaluation. Figure 8 shows that Position 2 gave poor results, especially with ten objects positioned in the scene. In this setting, measurements were taken straight from the front, so that the risk of overlapping clusters increases due to geometry. As the increasing number of objects in a scene raises that risk as well, the scene accuracy does not exceed 27 % for Position 2. Position 1 and Position 3 give significantly better results. Taking all perspectives into account, the overall scene accuracy calculates to 65 %. As Position 3 delivered the best results, the experiments on the adaptive program execution were mostly performed from that perspective.

6.2 Experiments on Adaptive Program Execution

The outstanding feature of the presented approach is the adaptive program execution. It concatenates the registration, object detection and programming approaches. To evaluate its applicability, we performed pick-and-place experiments within a scene of three objects, one cylinder and two cuboids of different color. The cylinder has to be picked up by the robot and has to be stacked on one of the remaining cuboids (**Figure 10(a)**). After the task is programmed through kinesthetic programming, the scene is clustered again and the resulting program is executed.

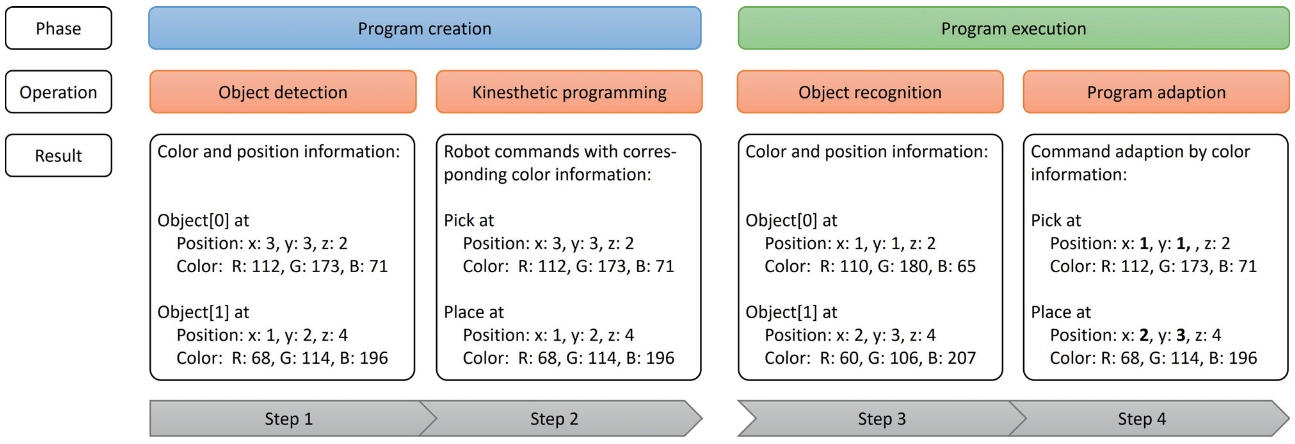
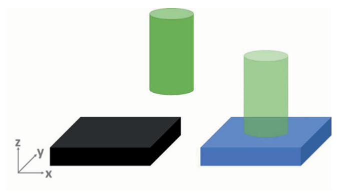
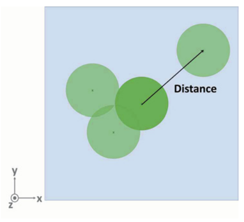

Phase
Program creation
Program execution
Operation
Object detection
Kinesthetic programming
Object recognition
Program adaption
Result
Color and position information: Object[0] at Position: x: 3, y: 3, z: 2 Color: R: 112, G: 173, B: 71 Object[1] at Position: x: 1, y: 2, z: 4 Color: R: 68, G: 114, B: 196
Robot commands with corresponding color information: Pick at Position: x: 3, y: 3, z: 2 Color: R: 112, G: 173, B: 71 Place at Position: x: 1, y: 2, z: 4 Color: R: 68, G: 114, B: 196
Color and position information: Object[0] at Position: x: 1, y: 1, z: 2 Color: R: 110, G: 180, B: 65 Object[1] at Position: x: 2, y: 3, z: 4 Color: R: 60, G: 106, B: 207
Command adaption by color information: Pick at Position: x: 1, y: 1, , z: 2 Color: R: 112, G: 173, B: 71 Place at Position: x: 2, y: 3, z: 4 Color: R: 68, G: 114, B: 196
Step 1
Step 2
Step 3
Step 4

Figure 9 Adaptive program execution design. In Step 1 and 2 the program is created, the object's color information is stored additionally. Before the program created in Step 2 is executed in Step 4, the robot positions are updated according to the RGB color information of the clustered objects. Note that the new color information does not necessarily have to equal the stored values.

Figure 9 presents the complete process schematically.
To investigate the robustness of the system, four participants programmed at least nine pick-and-place tasks. Each of these programs was performed three times with a variety of object positions, leading to 111 program executions of which 82.0 % were performed correctly. The system had to adapt to new object positions every time. The final position of the cylinder was measured relative to the cuboid's position it was placed on. As the system places the grasped cylinder in the center of the base cuboid, we compute the distance as a measure for accuracy as indicated in **Figure 10(b)**.
We plotted the results as the distance from the base object center as a function of quantiles in **Figure 11**. A quantile is the fraction of measurements that qualifies to the corresponding value plotted on the y-axis. Taking the results of all participants (i.e. the curve *Total*) into account, the plot illustrates that 20 % of the correctly performed tasks resulted in a maximal distance of 5 mm. 50 % showed a distance of 10 mm and 80 % of successful program executions led to values of 15 mm or better. Furthermore, the graphs show that the approach is robust against individual mannerisms. For most quantiles, the curves are scattered by a distance of 5 mm or less.



(a) Experiment design: The cylinder is stacked on one of the cuboids



(b) Top view of two stacked objects to illustrate the distance measure and possible variations

Figure 10 In the experiments one object had to be stacked on one of the other ones. Subsequently, the distance to the center position was measured.

7 Conclusions & Future Work

In this work we presented an approach that allows to command a robot with basic instructions. It provides different intuitive programming approaches as well as a fast and reliable object detection. The adaptive program execution by a mobile device utilizes all features of this approach and features a one-shot programming functionality performed without additional external sensors. We show that the system is sufficient to stack objects on top of each other and that programmed tasks can be repeated precisely with varying object positions.
We found that the object detection runtime is largely influenced by the data processing of the point clouds. In future, we may optimize this fragment of the object detection algorithm in order to reach a highly applicable performance. Furthermore, the registration procedure can be replaced by a robot detection capability. Taking a CAD model of the robot, the current robot configuration can be easily com-

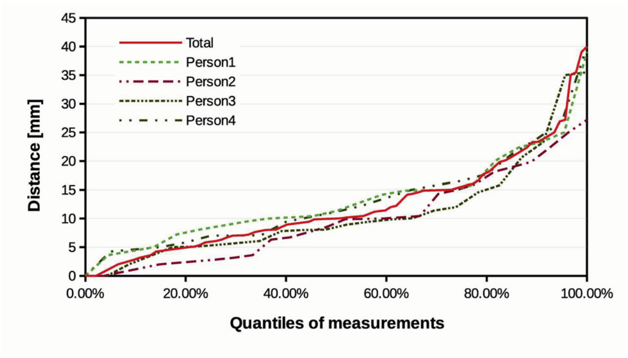


Figure 11 Adaptive program execution precision illustrated as the distances of the base object center to the actual release position related to the quantiles of measurements.

pared to data from depth images. With this approach a frequent registration would be possible, leading to increasing scene detection success rates and an even more precise adaptive program execution.

Acknowledgement

This work has partly been supported by the Deutsche Forschungsgeschmeinschaft (DFG, German Research Foundation) under grant agreement He2696/15 INTROP (INTuitive RObot Programming) and by the Technologie Allianz Oberfranken (TAO).

8 Literature

[1] Green, S. A., Billinghurst, M., Chen, X., Chase, J. G.: "Human-Robot Collaboration: A Literature Review and Augmented Reality Approach in Design", International Journal of Advanced Robotic Systems, 2008

[2] Goodrich, M. A., Schultz, A. C.: "Human-Robot Interaction: A Survey", Foundations and Trends in Human-Computer Interaction, 2007

[3] Lambrecht, J., Kruger, J.: "Spatial programming for industrial robots based on gestures and Augmented Reality", IEEE International Conference on Intelligent Robots and Systems, 2012

[4] Lambrecht, J., Walzel, H.: "Robust Finger Gesture Recognition on Handheld Devices for Spatial Programming of Industrial Robots", IEEE International Symposium on Robot and Human Interactive Communication, 2013

[5] Mateo, C., Brunete, A., Gambao, E., Hernando, M.: "Hammer : An Android Based Application for End-User Industrial Robot Programming", IEEE/ASME International Conference on Mechatronic and Embedded Systems and Applications, 2014

[6] Birkenkampf, P., Leidner, D., Lii, N. Y.: "Ubiquitous user interface design for space robotic operation", 14th Symposium on Advanced Space Technologies in Robotics and Automation (ASTRA), 2017

[7] Huang, J., Cakmak, M.: "Code3 : A System for End-to-End Programming of Mobile Manipulator Robots for Novices and Experts", ACM/IEEE International Conference on Human-Robot Interaction, 2017

[8] Kain, K., Stadler, S., Giuliani, M., Mirnig, N., Stollnberger, G., Tscheligi, M.: "Tablet-Based Augmented Reality in the Factory: Influence of Knowledge in Computer Programming on Robot Teaching Tasks", Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction, 2017

[9] Benko, H., Jota, R., Wilson, A.: "MirageTable: Freehand Interaction on a Projected Augmented Reality Tabletop", Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12, 2012

[10] Billinghurst, M., Clark, A., Lee, G.: "A Survey of Augmented Reality", Foundations and Trends in Human–Computer Interaction, 2015

[11] Argall, B. D., Chernova, S., Veloso, M., Browning, B.: "A survey of robot learning from demonstration", Robotics and Autonomous Systems, 2009

[12] Billard, A., Calinon, S.: "Robot Programming by Demonstration", Handbook of Robotics, 2007

[13] Wu, Y., Demiris Y.: "Towards one shot learning by imitation for humanoid robots," in Proceedings of IEEE ICRA, 2010

[14] Zöllner, R., Pardowitz, M., Knoop, S., Dillmann, R.: "Towards cognitive robots: Building hierarchical task representations of manipulations from human demonstration", Proceedings of IEEE ICRA, 2005

[15] Orendt, E. M., Fichtner, M., Henrich, D.: "Robot Programming by Non-Experts: Intuitiveness and Robustness of One-Shot Robot Programming", Proceedings of IEEE RO-MAN, 2016

[16] Akgun, B., Cakmak, M., Jiang, K., Thomaz, A. L.: "Keyframe-based Learning from Demonstration: Method and Evaluation", Proceedings of Social Robotics, 2012

[17] Groth, C., Henrich, D.: "One-Shot Robot Programming by Demonstration using an Online Oriented Particles Simulation", Proceedings of IEEE ROBIO, 2014

[18] Orendt, E. M., Riedl, M., Henrich, D.: "Robust One-Shot Robot Programming by Demonstration using Entity-based Resources", Advances in Service and Industrial Robotics, 2017

[19] Alexandrova, S., Cakmak, M., Hsiao, K., Takayama, L.: "Robot programming by demonstration with interactive action visualizations", Robotics: science and systems, 2014

[20] Riedl, M., Orendt, E. M., Henrich, D.: "Sensor-based loops and branches for playback-programmed robot systems", Advances in Service and Industrial Robotics, 2017

[21] Frank, J. A., Moorhead, M., Kapila, V.: "Realizing Mixed-Reality Environments with Tablets for Intuitive Human-Robot Collaboration for Object Manipulation Tasks", Proceedings of IEEE RO-MAN, 2016

[22] Frank, J. A., Moorhead, M., Kapila, V.: "Mobile Mixed-reality interfaces That enhance human robot interaction in shared spaces", Frontiers in Robotics and AI, 2017

[23] Herr, S., Gross, T., Gradmann, M., Henrich, D.: "Exploring Interaction Modalities and Task Allocation for Household Robotic Arms", CHI Extended Abstracts on Human Factors in Computing Systems, 2016

[24] Google, Tango tablet development kit user guide, Available: https://developers.google.com/tango/hardware/tablet, 2017

[25] Albu-Schäffer, A., Haddadin, S., Ott, C., Stemmer, A., Wimböck, T., Hirzinger, G.: "The DLR lightweight robot: design and control concepts for robots in human environments", Industrial Robot: An International Journal, 2007

[26] Ester, M., Kriegel, H.-P., Xu, X., Miinchen, D.: "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise", International Conference on Knowledge Discovery and Data Mining, 1996