Learning Robots to Grasp by Demonstration

Elias De Coninck[a,*], Tim Verbelen[a], Pieter Van Molle[a], Pieter Simoens[a], Bart Dhoedt[a]

[a] *IDLab, Department of Information Technology at Ghent University - imec, Technologiepark-Zwijnaarde 126, B-9052 Ghent, Belgium*

Abstract

In recent years, we have witnessed the proliferation of so-called collaborative robots or cobots, that are designed to work safely along with human operators. These cobots typically use the "program from demonstration" paradigm to record and replay trajectories, rather than the traditional source-code based programming approach. While this requires less knowledge from the operator, the basic functionality of a cobot is limited to simply replay the sequence of actions as they were recorded.

In this paper, we present a system that mitigates this restriction and learns to grasp an arbitrary object from visual input using demonstrated examples. While other learning-based approaches for robotic grasping require collecting a large amount of examples, either manually or automatically harvested in a real or simulated world, our approach learns to grasp from a single demonstration with the ability to improve on accuracy using additional input samples.

We demonstrate grasping of various objects with the Franka Panda collaborative robot. We show that the system is able to grasp various objects from demonstration, regardless their position and rotation in less than 5 minutes of training time on a NVIDIA Titan X GPU, achieving over 90% average success rate.

Keywords: Artificial Neural Networks, Machine Learning, Collaborative

---

*Corresponding author

Email addresses: `elias.deconinck@ugent.be` (Elias De Coninck), `tim.verbelen@ugent.be` (Tim Verbelen), `pieter.vanmolle@ugent.be` (Pieter Van Molle), `pieter.simoens@ugent.be` (Pieter Simoens), `bart.dhoedt@ugent.be` (Bart Dhoedt)

Robotics, Industrial Internet of Things

1. Introduction

The term “Industry 4.0” or “Smart Industry” has become a well-known concept for the next industrial revolution, where manufacturing environments will benefit from smart systems using Industrial Internet of Things (IIoT) sensors and robotics to trigger actions and control actuators [1]. One of the key drivers of the Industry 4.0, are the so-called industrial collaborative robots or cobots [2], which have become more productive, flexible, versatile, safer and reliable over the years. The idea behind cobots is to increase the collaboration and efficiency between robots and humans by removing the requirement for safety enclosures allowing the integration of these robots into human work spaces. Examples of such cobots available on the market are the Kuka LBR series [3], the Universal Robots UR series [4] and the Franka Panda [5]. These cobots are used in a variety of applications, e.g, production line loading and unloading, product assembly, and machine tending [6].

As manufacturing moves towards high-mix low-volume production cycles controlled by factory Cyber-physical systems (CPSs), the environment and actuators should be easily programmable for changing factory floor configurations. Agile manufacturing requires the ability to quickly adapt to (new) customer requests keeping the cost and quality in check [7, 8]. The unique feature of collaborative robots is the way they can be programmed by recording demonstrations, where the operator manipulates the robot to move to certain poses, which are then replayed as a program. In manufacturing this technique is often called “learning from demonstration”, which in machine learning literature points at training a generalized policy from demonstrations [9]. Therefore, we will use the term “program from demonstration” instead, as it better reflects the record and replay feature of cobots.

The downside of “program from demonstration” is its inability to adapt to changes in the work space. The replay feature executes actions exactly as they were recorded. Hence, when grasping an object any perturbation in the object's position or orientation compared to the demonstration jeopardizes success. In a flexible assembly line it is hard and expensive to perfectly align objects relative to the robot. A solution to this problem would be to learn a generalized policy from a vision and/or depth sensor in the environment to adapt robot actions based on recognized objects with its position and orientation [10]. However, training a generalized, robust and accurate model requires a huge amount of computation time. These models are typically trained on large datasets of labeled examples, which are not always readily available. An other problem of such generalized policies, while they can grasp unseen objects, is that the grasp location can vary for the same kind of objects resulting in a much harder assembly task.

In this paper we propose a system which combines the advantages of cognitive computing, collaborative robots, IIoT and interactions with human operators. The system offers a standard "program from demonstration" workflow, but also captures all available IIoT sensor information (i.e. a wrist-mounted camera) in addition to the robot pose for each recorded action. Our system focuses specifically on pick-and-place tasks. During replay we can leverage the recorded sensor information, and use deep learning techniques to instantiate a closed-loop controller for the Reach-to-Grasp (RTG) action, that enables to pick the object from the demonstration at any location in the work space. The policy can be further optimized by showing additional demonstrations.

The remainder of this paper is structured as follows. In the next section we discuss the related work in scope of learning from demonstration and grasping objects using machine learning. In Section 3 we give an overview of the main architectural components of the system and some implementation details. Section 4 goes deeper into our approach to learn from a single demonstration using vision sensors and the approach that is evaluated in the experiments in Section 5.

2. Related work

Robot manipulation for object grasping has been extensively studied, but is still an open challenge in research [11]. Here, we discuss prior research efforts in grasp detection and learning from demonstration that are most relevant to our approach.

2.1. Grasp detection and planning

In grasp detection research the focus lies on finding the correct position and orientation to make robust and accurate grasps for a given object. In [12, 13, 14, 15] models are trained on readily available labeled datasets, with printable 3D-objects [16, 17], 2D-images [18] or real life benchmark objects [19, 20]. While this tends to create good grasping points for known objects, it is challenging and time consuming to create or to extend to new objects: each dataset sample needs to be labeled with the position, orientation and in most cases even the height and width of the grasp location.

A second approach uses Reinforcement Learning (RL), where the models are trained by trial and error [21, 22]. In a real world setup many robots are used to gather grasping attempt samples, and the setup needs to run for a long time [23, 24, 25] as RL requires huge amounts of samples to arrive at stable and robust policies. Instead of sampling on real robots, which is time consuming and expensive, a simulated physics environment can generate trials to learn a model, but in this case an extra step is required to transfer results from the simulation environment to the real world [26]. However, these RL techniques have some caveats: they are typically sample inefficient and require a good reward signal to model the desired behaviour.

Vision-based deep learning techniques for grasp detection [15, 17, 24, 27, 28] share their main idea, i.e. training a large Convolutional Neural Network (CNN) to classify and rank multiple probable grasp poses sampled from a single image or point cloud, by using a sliding window with a fixed offset to position and rotation. After calculation a robot executes the best ranked grasp candidate.

This approach only works in a static environment where the camera is precisely calibrated and the robot manipulation is planned based on this selected grasp pose candidate. These approaches require multiple forward passes through the network to detect the best candidate, which is computationally expensive making them unfeasible to use as closed-loop controllers. To improve the execution time the number of grasp candidates can be limited [27] or a discrete set can be calculated in parallel [24, 29], resulting in a trade-off between execution time and potential grasp accuracy.

Similar to our approach, Morrison et al. [13] and Varley et al. [30] use a small CNN to output an energy heat map revealing how well every location in the image would act as a grasp location, giving the ability to perform closed-loop control. While there focus lies on learning a generalized policy, trained on a large dataset, to grasp unseen objects, our approach learns a more accurate robust grasp from a single sample without generalizing over multiple objects. In this work we train to grasp an object in the exact same position and orientation in order to make assembly line tasks possible. In contrast [30] learns a generalized policy from a dataset with known 3D meshes which results in varying grasps of the same object. We refer to [10, 31, 32] for a more in depth survey on data-driven grasping techniques.

2.2. Learning from demonstrations

Bootstrapping robotic grasping by giving demonstrations is an other key research area to make robust grasps possible. Tegin et al. [33] present a framework which is able to use human demonstration experience in combination with perceptional cues of objects to create a more stable approach vector to grasp objects. Learning from demonstration is often combined with other learning algorithms like RL [34] or guided policy search [35], but these algorithms require multiple demonstrations to train their policies. In [36] a large recurrent neural network is trained end-to-end from raw image input for multiple tasks simultaneously, increasing the success rate and robustness of each task while decreasing the training time for newly added tasks. Although the many recent advancements in learning from demonstration research, there is still a huge gap with cobot programming as implemented in industry [37].

This paper extends previous work [38], where we showcased how we can learn a neural network to train a closed-loop controller for perpendicular grasping of toy blocks from a single demonstration. We extend this work by also taking varying rotations into account, as well as allowing to tilt the gripper during demonstration resulting in grasps which were not possible before (Figure 1). With some objects a vertical grasp was not feasible and the ability to tilt the gripper during demonstration allows to grasp these objects from the side. Moreover, we evaluate on a more extensive set of objects. Furthermore, we integrate this algorithm in a "program from demonstration" system that closely matches

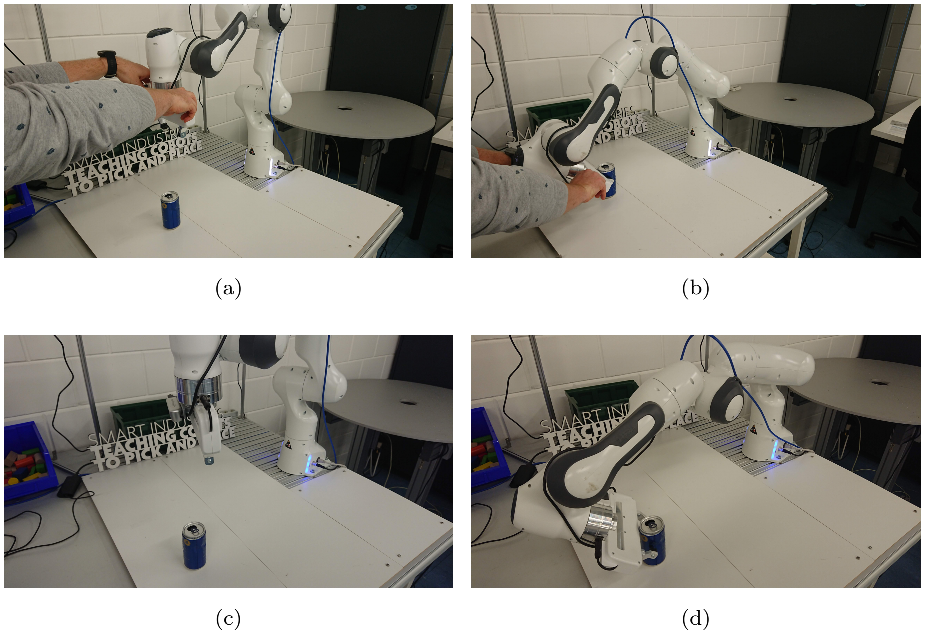


Figure 1: A RTG action demonstration consists of a hover position p_{hover} (a) and a grasp target position p_{target} (b). The demonstrated grasp does not need to be perpendicular to the workspace, as the gripper can be tilted when grasping an object. After training, the object is searched in the workspace (c) and grasped as demonstrated (d), relative to the position after searching.

the current operator workflow.

3. A flexible system to program from demonstration

Collaborative robotics can typically be programmed “from demonstration”, by simply guiding the end-effector, manually or with a teacher pendant, and recording the robot pose each step. Typically an operator would record an action in a sequence of multiple steps: initial hover pose, action location and resulting location, but the step sequence can be extended to decrease the possibility of collisions between steps. We propose a system that in addition to the robot pose, also records information from any sensors in the environment. This data can then be leveraged to train smarter control policies using machine learning techniques. For example, a wrist mounted camera can be used to identify the object to grasp, instead of merely the position to grasp. This section elaborates on the high level architecture and implementation to create a flexible system.

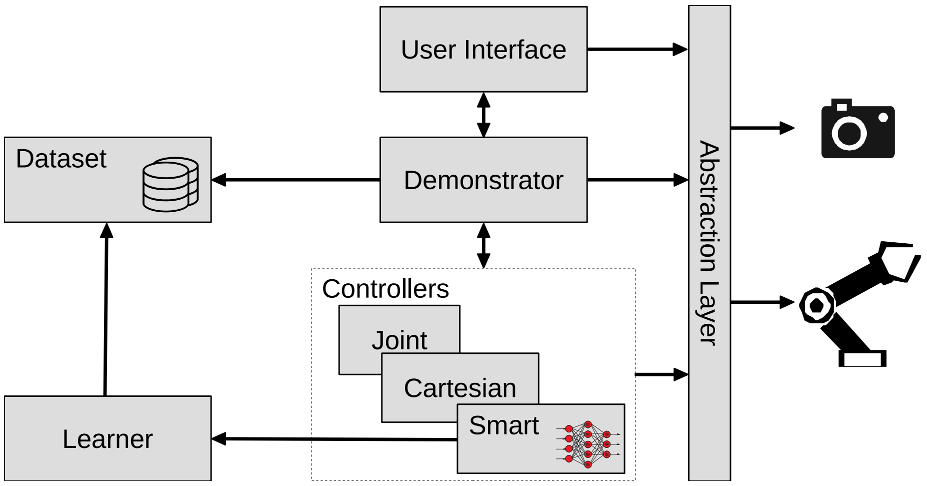


Figure 2: Architectural overview of the system. *Demonstrator* collects states of the environment and stores this information in the *Dataset*. Datasets are used by the *Learner* to learn a policy. Multiple *Controllers* are available to execute robotic movements, e.g. a *Smart Controller* using a trained policy.

3.1. Architecture overview

Figure 2 gives a high level overview of the system, which provides the following features:

- Record demonstrations as a sequence of steps, collecting the robot's state, as well as any other sensory input available.
- Replay demonstration steps with different control algorithms, that can be switched at runtime.
- Make all recorded information available as datasets to execute machine learning algorithms.
- Visual feedback during the demonstration and replay.

3.2. Demonstrator

The *Demonstrator* is responsible for recording demonstrations, offered by the operator, which can be replayed by controllers (see Section 3.4). Each demonstration consists of a sequence of steps, containing the robot arm's state, step action type and any extra sensor values available. The recorded state is a combination of the gripper's pose (position and orientation) and the arm's joint states. Each step also contains one of three generic action types: move, pick and place. During the demonstration the operator specifies which action is taken for the current step. During replay, all steps are executed by the controllers to move the robotic arm from one pose to the next, with pick and place actions opening or closing the gripper, respectively.

In addition, external sensors can be connected to the system to add extra information to each step, e.g. a camera. The sensors values are collected and can be used to visualize steps in a front-end or used by other components of the system, e.g. the *Learner* component.

3.3. Datasets

To learn a policy through machine learning, a *Dataset* component is required to provide data samples and their corresponding labels. The *Learner* component queries the *Dataset* samples in batches, stored in memory or on disk, to train a model for the task at hand. The *Dataset* component is responsible to generate input-label pairs, from raw demonstration data, that can be used to train a machine learning model by the *Learner*. Data augmentation can be used to crop and rotate samples, or for further augmenting images by applying random changes to brightness, contrast and saturation.

3.4. Controllers

The *Controller* components are responsible for the interaction with the robotic arm to execute each step of a demonstration. The 'move' actions are by default executed by the Joint or Cartesian *Controller*, which takes over control until the desired position and orientation is reached, to replay steps as they were recorded. These two controllers differ in the way they execute the requested pose. For the Joint controller interpolates between two steps in joint space, while the Cartesian controller moves the end effector in a straight line in Cartesian space.

In addition, the system can be extended with *Smart Controller*s. These can override the default behavior for certain steps, and can query the current robot state and sensor information and output a custom control action. This allows for more complex control policies, for example to avoid collisions with objects that are sensed in the environment, or use a neural network to learn to grasp objects based on their visual appearance, as we will describe in 4.

3.5. Learner

Given a *Dataset*, the *Learner* performs the actual pollicy training. The learning is initiated and configured by the *Demonstrator*, that provides a configuration dictionary with hyper parameters such as the learning rate, batch size, loss function, etc., and the network's structure. The training continues in the background until the stop criterion is met (i.e. number of iterations) and the neural network's final weights are stored for each step separately.

When the *Demonstrator* requests to execute a step controlled with a *Smart Controller*, the corresponding network model is loaded with its last updated weights. This enables the operator to already test the learned behaviour while the *Learner* is still training.

3.6. User interface

The system provides a user interface where the operator can create, load, edit and replay demonstrations. Each demonstration is saved to disk with the corresponding steps and sensor values.

Figure 3 shows the interface (left) in combination with the operator's performed actions (right). During the demonstration a live camera feed is streamed with a blue rectangular box in the center of the feed, which gives a visual aid

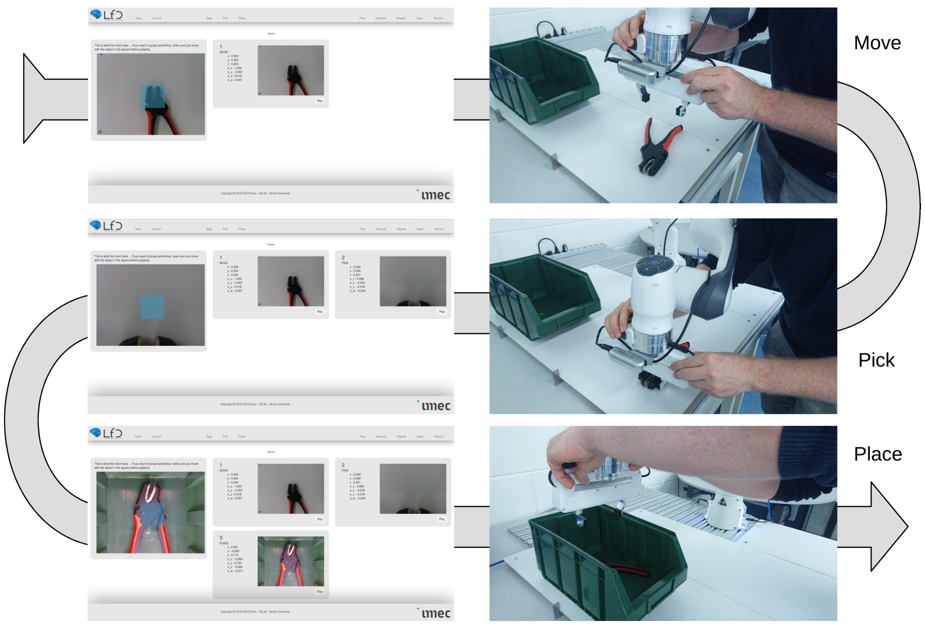


Figure 3: The user interface provides the operator a method to record, replay and review given demonstrations. Demonstrations can have any number of steps with each step being one of the supported action types: move, pick or place. The example shows the user interface in combination with the operator's actions on the robotic arm for grasping a wire-clipper and placing it in the green bin.

for the operator where to focus. This is used in our smart grasping controller presented in 4. To record a step, the operator can click on the 'move','pick' or 'place' menu items, or use one of the keyboard shortcuts.

Figure 3 shows a typical pick-and-place workflow. First, the operator records a 'move' step, guiding the end-effector to a position somewhat above the object he/she wants to grasp, resulting in the hover pose p_{hover}. Next, the operator moves the cobot's end-effector to the pose p_{target} where the object is grasped, saving this step as a 'pick' action which closes the gripper. Finally, the operator guides the cobot in one or more steps to the green container, ending with a 'place' action releasing the gripper and dropping the object.

For each recorded step, the user interface shows the recorded end-effector pose combined with any sensor information, in this case an image captured from an in-hand mounted camera. After recording, all demonstration steps can be replayed again, or one can trigger (and edit) each step individually for debugging purposes.

3.7. Implementation details

Our system is implemented using OSGi [39], a modular service-oriented framework in Java. Each component is written as an OSGi bundle and exposes its services through well defined APIs. We build on our previous work on a Thing Abstraction Layer (TAL) [40], which supports a wide range of Internet of Things (IoT) sensors and actuators including our robot environment through Robot Operating System (ROS) [41]. For training, managing and building neural networks we use DIANNE [42], a modular deep learning framework with native bindings to Torch7. By building on the OSGi-based technology stack, we are able to distribute the components on the available infrastructure [43], for example deploying real-time components, e.g. the controllers, on an edge device close to the robot, while offloading the *Learner* to a remote GPU cloud server.

4. Learning a smart grasp controller

Of course the system presented in Section 3 is most valuable when one adds *Smart Controller*s. The basic functionality of programming pick-and-place operations by demonstration is merely recording the position where to grasp, not the object to grasp. When we have a camera sensor mounted on the robot end-effector, we can use the visual feedback to pick the object whenever it is in view. To learn the visual appearance of the object to grasp, we use a neural network model. Although neural networks typically require a huge amount of data to train on, we mitigate this by taking the following assumptions:

- The workspace where the robot is positioned during replay is the same as during the demonstrations. Resulting in the same robotic arm reach and visual features.
- The object to grasp during replay is visually the same as during the demonstration.
- The position of the object to grasp is in view of the camera at the demonstrated p_{hover} pose and is within reach of the gripper.
- The object to grasp has unique visual features in the cropped center image of p_{hover} pose.

The first three requirements are defined in order to train a small neural network, which is not expected to generalize over multiple objects and scenes, allowing to control the robot in real-time. The last requirement makes sure we can detect the object and train a neural network only on this one demonstration. In the remainder of this section we will first describe how we generate a dataset from our demonstration, next how we train a neural network for spotting the object to grasp, and finally how we implement a closed-loop controller for executing the grasp.

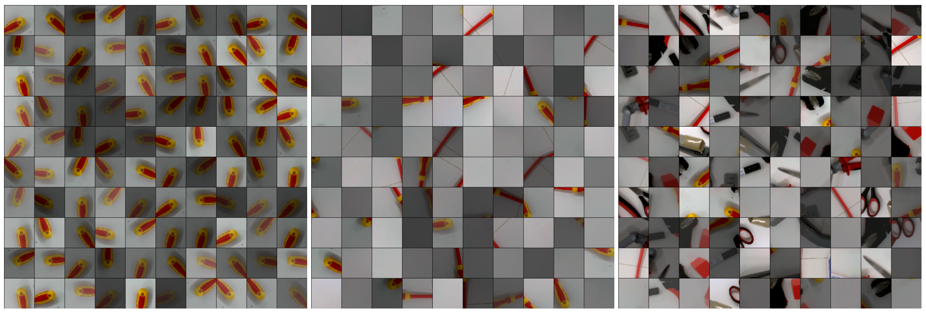

(a) (b) (c)

Figure 4: From the demonstration camera frame, we generate random positive (a) and negative (b) samples by cropping, rotating and adapting brightness and contrast. By including other objects in the same frame, additional negative samples with objects to ignore are generated (c).

4.1. Generation of grasping dataset

For each pick step in the demonstration, a dataset is generated based on the camera frame captured at p_{hover}. During the demonstration, the operator has to make sure the object to grasp is focused at the center of the camera frame, as this will contain the features the controller searches for at run-time. As shown in Section 3, the user interface provides visual feedback on what is currently in focus. From this frame we generate both “positive” and “negative” samples. A “positive” sample is a center crop of 128×128 of the image randomly rotated around its center. A “negative” sample is a randomly picked cropping outside this center region rotated around its center. Each sample is labeled with a positive or negative label with the values 1 and 0, respectively, and the applied rotation angle.

Furthermore, we augment the dataset by applying random perturbations on brightness and contrast [44]. In addition, we also add center crops from other demonstrations as negative samples for the current demonstration, which will make the neural network more robust. An example dataset is shown in Figure 4. An other approach is shown in Figure 4 (c) where multiple objects are positioned in the same scene to create negative samples for all objects outside the center crop. During the demonstration the operator needs to make sure the target object does not overlap or is to close to other objects as this would than be included as a positive sample.

4.2. Neural network architecture

We use a convolutional neural network architecture as shown in Figure 5. It consists of 4 convolutional layers followed by an average pooling layer to down-sample the feature planes and two fully connected layers. The network has three output values, each with its own activation function to limit to a valid range. The first value represents the Grasp Quality Q and has a sigmoid activation to limit the range to $[0, 1]$. This Grasp Quality output is trained using the "positive" or "negative" label of the dataset. The other two values encode the rotation angle Φ, represented by the cosine O_1 and sine O_2 components of the rotation angle on a unit circle. We clamp the original rotation angle Φ range to $[-\frac{\pi}{4}, \frac{\pi}{4}]$ and multiply this by 2 to increase the accuracy around 0. Because we limit the angle to $[-\frac{\pi}{4}, \frac{\pi}{4}]$ we can limit the range of the first output component $\cos(2\Phi)$ to $[0, 1]$ with a sigmoid function, while for the second component $\sin(2\Phi)$ the hyperbolic tangent activation is used to limit its values to $[-1, 1]$. This angle representation is chosen to increase the accuracy for smaller angles [13], and to

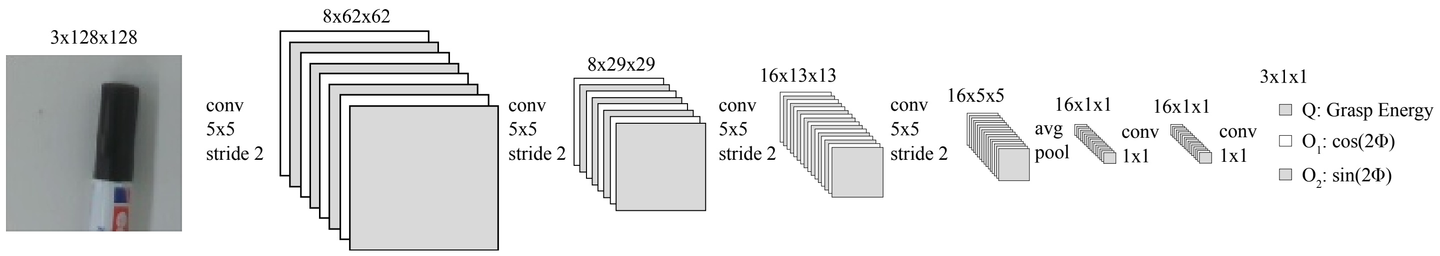


Figure 5: The GraspNet neural network model to support learning from a single demonstration with a dataset sample of a black marker. The model accepts 128×128 cropped images or larger as input and forwards it through 4 convolutional layers with 8, 8, 16 and 16 filters of size 5×5 and a stride of 2. An average pooling layer to down-sample the feature plane followed by two fully connected layers implemented as 1×1. The outputs represent a per region label revealing the estimated grasp quality and the angle, in two vector components of a unit circle.

create fluent and continuous distributions, which tends to be easier to learn for neural networks [45].

4.3. Real-time smart grasp controller

The real-time smart closed-loop grasp controller replaces the standard Cartesian or Joint controller for steps with the 'pick' action. When the demonstration reaches the p_{hover} pose, the system streams full resolution images (3 (RGB) $\times$ 480 ($height$) $\times$ 640 ($width$)) to the controller. As our GraspNet is trained on 128×128 images, we basically evaluate the neural network on every 128×128 crop of the camera image as a sliding window. By implementing the fully connected layers as 1×1 convolutions, we can execute this in a single forward pass, resulting in a $3 \times 23 \times 33$ output.

The first output plane represents a grasp energy heat map labelling how well each region would act as a grasp location, while the other two output planes give the corresponding estimated angle representation. We find the (x, y) position that yields the maximum energy, and calculate the corresponding angle Φ:

$$\Phi = \frac{1}{2} \arctan \frac{O_2}{O_1} \tag{1}$$

From this output we calculate a translation velocity vector v in the XY-plane. The direction of the velocity is given by the direction of highest activation. The velocity itself is proportional to the distance of this location w.r.t. the center of the image, the further the distance from the center the higher the outputted velocity, followed by additionally applying a bezier function to smoothen the approach towards the center. Similarly, we calculate an angular velocity around the Z-axis, based on the rotation angle Φ. These values are then applied to the robot's end-effector and the next camera frame is processed.

Figure 6 shows an example of the closed-loop controller capturing an image from the wrist-mounted camera, forwarding it through the GraspNet model and resulting in an grasp energy heat map and an estimated rotation angle. Darker regions in the energy heat map represent better grasp locations.

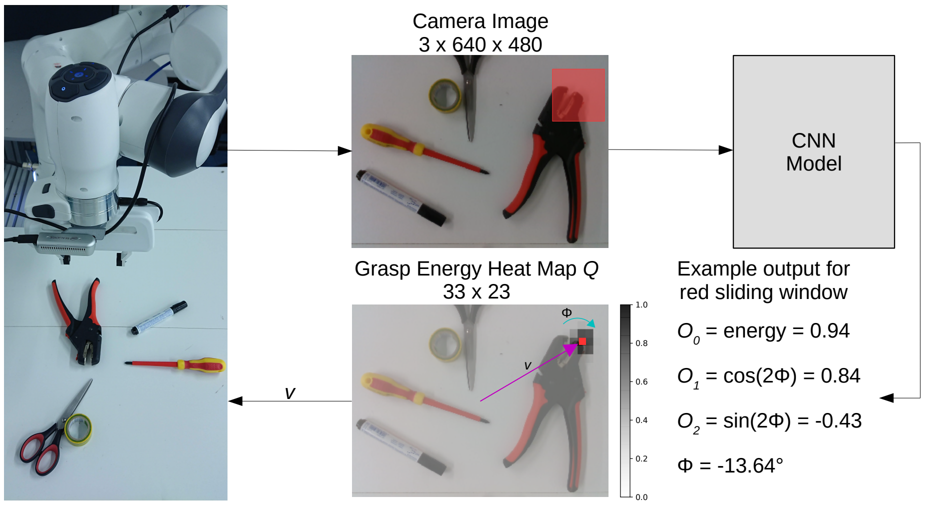


Figure 6: Example of the smart grasp controller picking a wire clipper from a bunch of objects. The trained network outputs an energy heat map, upscaled and overlayed on the input image. Darker regions show higher activation points. The red square represents one sliding window ($128x128$) resulting in the red pixel of the grasp energy heat map. The vector v represents the velocity direction the controller has to take starting from the center of the image and the vector Φ represents the angle velocity. For the region with the highest activation the cosine and sine value is used to calculate the rotation angle.

The closer the end-effector comes to the grasp object and the better the orientation angle matches, the slower the end-effector moves, until all velocity values drop below a configured threshold. Finally the controller executes the relative grasp movement as demonstrated from p_{hover} to p_{target}. This enables the controller to perform the demonstrated grasp from any position and orientation. An additional benefit of our approach is that the system does not require any camera calibration with respect to the robot arm.

5. Experiments

We evaluate our demonstration framework on pick and place experiments conducted on seven household objects: coffee mug, duck tape roll, tape dispenser, screwdriver, stapler, whiteboard marker and wire clipper. For each object we provided a single demonstration to pick up the object. We evaluate both the neural network accuracy on the dataset as well as the success rate on real-world grasps. We also evaluate the real-time performance of the grasp controller.

5.1. Hardware

Our experimental setup, shown in Figure 6 consists of the Franka Emika Panda cobot [5] with 7 degrees of freedom, with a Realsense D435 RGB-D camera mounted on the end-effector. The camera is configured at a frame-rate of $30fps$ with a resolution of 640×480. A Jetson TX2, a development board with an embedded GPU, is used to control the cobot. This includes processing the camera frames and performing inference of a trained neural network during grasping. The Jetson is able to forward and process the RGB frames in $\pm 18.38ms$ on average, which is fast enough to operate at the $30Hz$ of the camera. Training the neural networks is performed on a separate server running Ubuntu 16.04 with an Intel(R) Core(TM) i5-2400 CPU running at 3.10GHz and a NVIDIA Titan X graphics card. On our hardware it takes less than 5 minutes to train the Graspnet network. The deployment setup for our experiments is shown in Figure 7.

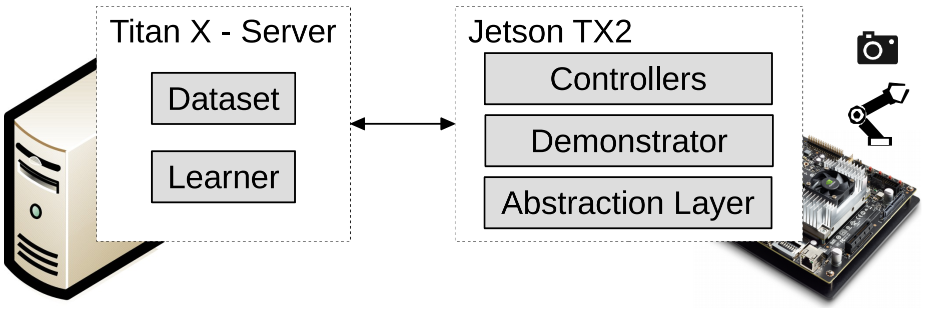


Figure 7: The experiments deployment scheme where the *Dataset* and *Learner* is deployed on server grade hardware with a dedicated GPU (Titan X), while the *Controllers*, *Demonstrator* and *Abstraction Layer* components are deployed on a embedded GPU system (Jetson TX2) to control the robotic arm directly and stream the camera through the network at inference time.

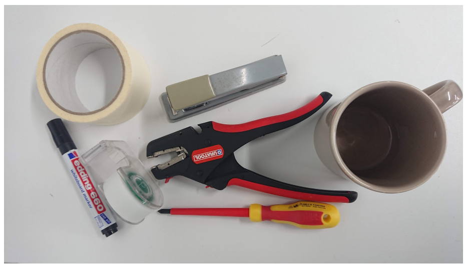


Figure 8: Test objects for the experiments consisting of some household items.

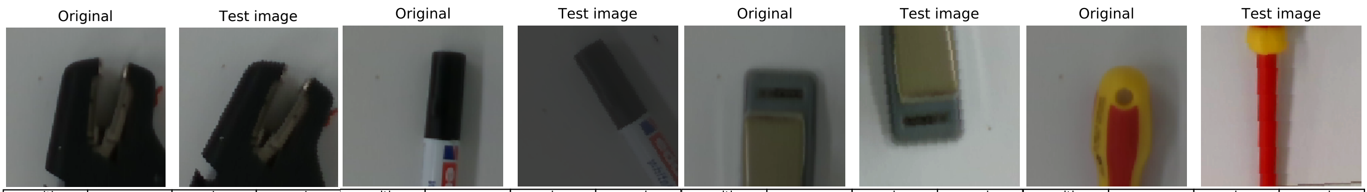
5.2. Test objects

In the experiments we use common household objects, e.g. screwdriver, duct tape, coffee mug (Figure 8). These objects vary in size, shape and difficulty to grasp, while still being feasible for the parallel gripper to grasp. The gripper is able to lift objects with a maximum width of $80mm$ and up to $3kg$ in weight. Our objects are a subset of the ACRV Picking Benchmark (APB) [20] and the Yale-CMU-Berkeley (YCB) Object Set [19]. In the experiments the objects are grasped without tilting the end-effector, however this would not alter the accuracy of the system as the object is searched in the p_{hover} position.

5.3. Demonstration flow and learning

For each object we record a single demonstration following the workflow presented in Section 3.6. We select a different hover height pose based on the object to grasp, in order to have enough distinct features in the center crop of the image. In general, for larger objects we applied a higher hover pose, giving a larger field of view of the camera.

For each demonstration, we generate a dataset and train a neural network as defined in Section 4. Each neural network is trained for 1000 iterations by minimizing the mean squared error loss using the Adam optimizer [46] with a learning rate of 0.001 and a batch size of 128. For each mini-batch we generate new samples from the demonstration's *Dataset*.



	position	cos	sin	angle
ground truth	1.0	0.96	-0.29	-17.04
predicted	0.92	0.93	-0.27	-21.89

(a)

	position	cos	sin	angle
ground truth	1.0	0.8	0.6	37.06
predicted	0.96	0.77	0.62	39.33

(b)

	position	cos	sin	angle
ground truth	1.0	0.71	-0.71	-45.0
predicted	1.0	0.74	-0.82	-42.43

(c)

	position	cos	sin	angle
ground truth	0.0	1.0	0.0	0.0
predicted	0.0	0.02	-0.47	-88.78

(d)

Figure 9: Four samples of the dataset's original image accompanied by a test image randomly sampled from a generated test set. For each sample we show the ground truth labels, predicted labels and the calculated angle Φ from Eq. (1) in degrees. Samples (a), (b) and (c) are positive and (d) is a negative sample. For sample (d) only the position is correctly predicted because the loss is not backpropagated for the cosine and sine errors as these values are irrelevant.

5.4. Results

We first evaluate the estimated grasp success rate and rotation accuracy based on a separately held-out generated test set with samples of size 128×128. The estimated grasp success accuracy is calculated by comparing the grasp label with the predicted value rounded to the nearest integer, while the rotation accuracy is compared by rounding to a single digit (tenth of a radian). On average, we get a grasp accuracy of 99% and a rotation accuracy of 80% with

Table 1: Grasp success rate of seven household objects.

household objects	success rate (%)
coffee mug	100
duct tape roll	100
screwdriver	90
stapler	90
tape dispenser	95
whiteboard marker	95
wire clipper	70
average	91.34

an average error of only 0.09 *radians*. Figure 9 shows the results for four samples from the original dataset accompanied by a test set sample comparing the ground truth to the predicted values.

Next we evaluate the grasp performance on the real cobot. For each object we performed 20 grasps by randomly placing each object within view of the camera and executing the recorded demonstration. In order to keep within the limits of the physical robot we limited the object rotation between ±160 degrees. The grasp is successful from the moment it was placed on the recorded location of the demonstration. The grasp success rate for each object can be found in Table 1. For all objects combined we achieved an average success rate of 91.34% and it took on average 5 seconds to search for the object and execute a grasp.

Figure 10 shows the velocity outputs and the highest and center heat map energy value of the closed-loop controller over time while moving towards the object. We place the object in view of the camera at timestep 10. The closer the object is to the center of the camera frame the slower the end-effector moves until it eventually reaches zero and performs the grasp. The angular velocity can alter between zero and a small velocity because the movement has not stopped in the XY-plane.

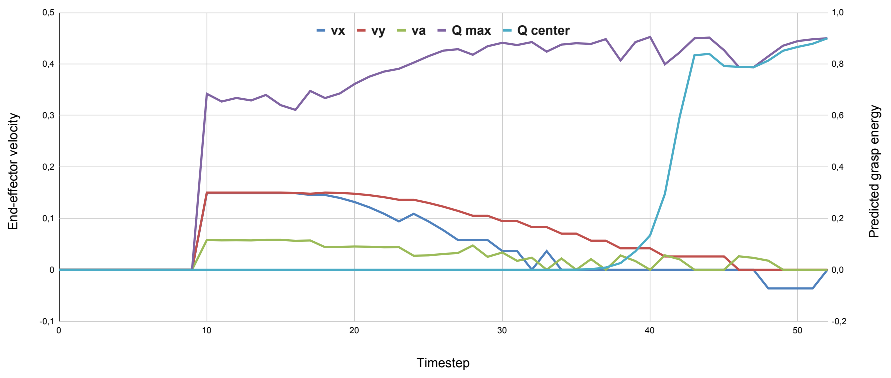


Figure 10: The translational and angular velocity over time, as well as the highest and center grasp energy in view. We place the object in view of the camera at timestep 10. The closer the end-effector gets to the object to grasp the slower it moves.

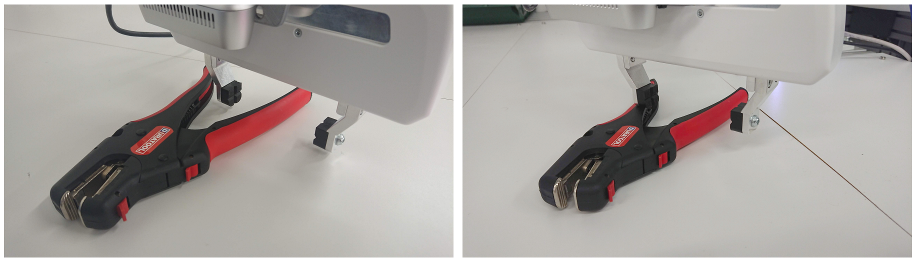


Figure 11: The end-effector's correct grip position on the left and failed attempt on the right for graspping the wire clipper.

The failure cases are mostly due to the limited spatial resolution of the activation map (currently 33×23). This resulted in the robot touching the object with one of its fingers instead of positioning the object between its fingers. Especially in the case of the wire clipper, which we pick on one of its handles, the gripper often hits the other handle as shown on Figure 11. We could potentially improve the performance, e.g. by increasing the camera resolution or removing the strides in the CNN architecture, resulting in a higher activation map resolution, at the cost of more compute power.

6. Conclusion and future work

In this paper, we proposed a flexible system which extends the default "program from demonstration" feature of collaborative robots with deep learning techniques to adapt to environments with moving objects. In particular, in order to facilitate pick-and-place tasks, we extend the system with a smart grasp controller that uses a CNN that outputs an energy heat map where each region represents how well the region would act as a grasp location and estimates the rotation angle, from a single frame captured by a wrist-mounted camera. This makes the system robust to grasping the object at any location and orientation within view of the camera, and achieves near 100% success rate for various household objects. To accommodate for the physical limitations of the robotic arm we limited the rotation of objects in the range $[-160, 160]$ degrees, an other approach would forward a mirrored image through the network and select the rotation within its limits followed by a mirrored grasp when applicable.

We believe that combining advances in artificial intelligence with cobot programming and control is crucial to achieve a truly collaborative environment where cobots and humans can work together. This work is a first step in that direction, focused on learning to pick objects, while keeping an intuitive and easy-to-use operator interface. To further improve our grasping method we could use our method as a baseline policy to fine tune using reinforcement learning techniques, i.e. detecting grasp success and using this as a reward signal. Also, we could query the operator for additional demonstrations in case of failed grasps, following an active learning approach.

To extend the current approach to allow for more complex and precise grasps we need to extend the demonstrator to be able to record high DOF grippers instead of the simple two finger style gripper. This would require additional steps during the demonstration. In order to fine tune for more precise grasps we would replace multiple steps of the demonstration sequence with multiple trained networks updating its gripper's XY-plane position and orientation in each step accordingly.

In Section 4 we made an assumption that the workspace or scene is kept the same during demonstration and execution. This limitation is pretty strict but can be removed by including images from other scenes as negative samples. An other possibility is to further augment the images to adapt textures and/or colors of objects and the scene but this requires further research. For the case study in Section 4 we selected RGB images as input for the network and left out the depth channel. This decision was made because initial experiments pointed out that augmenting the depth channel to create realistic samples from a single image is challenging and it resulted in worse grasp success rate and lower performance. Further research is required to design a better depth augmentation step in order to deal with occlusions in the depth image.

As future work we would also like to extend the system with smart controllers, i.e. for workspace monitoring and collision detection and prevention.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of GPUs used for this research. This research received funding from the Flemish Government under the "Onderzoeksprogramma Artificiële Intelligentie (AI) Vlaanderen" programme.

References

References

[1] H. Kagermann, J. Helbig, A. Hellinger, W. Wahlster, Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0: Securing the Future of German Manufacturing Industry; Final Report of the Industrie 4.0 Working Group, Forschungsunion, 2013.

[2] S. Haddadin, A. Albu-Schaffer, A. De Luca, G. Hirzinger, Collision detection and reaction: A contribution to safe physical human-robot interaction, in: 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2008, pp. 3356–3363. `doi:10.1109/IROS.2008.4650764`.

[3] Kuka AG, accessed February 19, 2019. [link].
URL `http://www.kuka.com/`

[4] Universal robots, accessed February 19, 2019. [link].
URL `http://www.universal-robots.com/`

[5] Franka EMIKA, accessed February 19, 2019. [link].
URL `http://www.franka.de/`

[6] R. Bloss, Collaborative robots are rapidly providing major improvements in productivity, safety, programing ease, portability and cost while addressing many new applications, Industrial Robot: the international journal of robotics research and application 43 (5) (2016) 463–468. `arXiv:https://doi.org/10.1108/IR-05-2016-0148`, `doi:10.1108/IR-05-2016-0148`.

[7] M. Hermann, T. Pentek, B. Otto, Design principles for industrie 4.0 scenarios: a literature review, Technische Universität Dortmund, Dortmund (2015).

[8] H. Sharifi, Z. Zhang, Agile manufacturing in practice-application of a methodology, International Journal of Operations & Production Management 21 (5/6) (2001) 772–794.

[9] S. Schaal, Learning from demonstration, in: Advances in neural information processing systems, 1997, pp. 1040–1046.

[10] J. Bohg, A. Morales, T. Asfour, D. Kragic, Data-driven grasp synthesis–a survey, IEEE Transactions on Robotics 30 (2) (2014) 289–309. `doi:10.1109/TRO.2013.2289018`.

[11] V. . Nguyen, Constructing force-closure grasps, in: Proceedings. 1986 IEEE International Conference on Robotics and Automation, Vol. 3, 1986, pp. 1368–1373. `doi:10.1109/ROBOT.1986.1087483`.

[12] J. Redmon, A. Angelova, Real-time grasp detection using convolutional neural networks, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 1316–1322. `doi:10.1109/ICRA.2015.7139361`.

[13] D. Morrison, P. Corke, J. Leitner, Closing the loop for robotic grasping: A real-time, generative grasp synthesis approach, CoRR abs/1804.05172 (2018). `arXiv:1804.05172`.
URL `http://arxiv.org/abs/1804.05172`

[14] D. Morrison, P. Corke, J. Leitner, Multi-view picking: Next-best-view reaching for improved grasping in clutter, CoRR abs/1809.08564 (2018). `arXiv:1809.08564`.
URL `http://arxiv.org/abs/1809.08564`

[15] U. Asif, J. Tang, S. Harrer, Graspnet: An efficient convolutional neural network for real-time grasp detection for low-powered devices, in: Proceedings

of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 4875–4882. doi:10.24963/ijcai.2018/677.
URL https://doi.org/10.24963/ijcai.2018/677

[16] C. Goldfeder, M. Ciocarlie, , P. K. Allen, The columbia grasp database, in: 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 1710–1716. doi:10.1109/ROBOT.2009.5152709.

[17] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, K. Y. Goldberg, Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics, CoRR abs/1703.09312 (2017).

[18] A. Saxena, J. Driemeyer, A. Y. Ng, Robotic grasping of novel objects using vision, The International Journal of Robotics Research 27 (2) (2008) 157–173. arXiv:https://doi.org/10.1177/0278364907087172, doi:10.1177/0278364907087172.
URL https://doi.org/10.1177/0278364907087172

[19] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, A. M. Dollar, Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set, IEEE Robotics Automation Magazine 22 (3) (2015) 36–52. doi:10.1109/MRA.2015.2448951.

[20] J. Leitner, A. W. Tow, N. Sünderhauf, J. E. Dean, J. W. Durham, M. Cooper, M. Eich, C. Lehnert, R. Mangels, C. McCool, P. T. Kujala, L. Nicholson, T. Pham, J. Sergeant, L. Wu, F. Zhang, B. Upcroft, P. Corke, The acrv picking benchmark: A robotic shelf picking benchmark to foster reproducible research, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 4705–4712. doi:10.1109/ICRA.2017.7989545.

[21] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: 2017 IEEE

International Conference on Robotics and Automation (ICRA), 2017, pp. 3389–3396. doi:10.1109/ICRA.2017.7989385.

[22] J. Peters, S. Schaal, Reinforcement learning of motor skills with policy gradients, Neural Networks 21 (4) (2008) 682 – 697, robotics and Neuroscience. doi:https://doi.org/10.1016/j.neunet.2008.02.003. URL http://www.sciencedirect.com/science/article/pii/S0893608008000701

[23] S. Levine, P. Pastor, A. Krizhevsky, D. Quillen, Learning hand-eye coordination for robotic grasping with large-scale data collection, in: ISER, 2016.

[24] L. Pinto, A. Gupta, Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours, 2016 IEEE International Conference on Robotics and Automation (ICRA) (2016) 3406–3413.

[25] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, S. Levine, Scalable deep reinforcement learning for vision-based robotic manipulation, in: CoRL, 2018.

[26] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, P. Abbeel, Domain randomization for transferring deep neural networks from simulation to the real world, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2017) 23–30.

[27] I. Lenz, H. Lee, A. Saxena, Deep learning for detecting robotic grasps, in: Robotics: Science and Systems, 2013.

[28] S. Kumra, C. Kanan, Robotic grasp detection using deep convolutional neural networks, 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2017) 769–776.

[29] E. Johns, S. Leutenegger, A. J. Davison, Deep learning a grasp function for grasping under gripper pose uncertainty, 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2016) 4461–4468.

[30] J. Varley, J. Weisz, J. Weiss, P. Allen, Generating multi-fingered robotic grasps via deep learning, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2015, pp. 4415–4420. doi:10.1109/IROS.2015.7354004.

[31] A. Sahbani, S. El-Khoury, P. Bidaud, An overview of 3d object grasp synthesis algorithms, Robotics and Autonomous Systems 60 (2012) 326–336.

[32] S. D. Roy, S. Chaudhury, S. Banerjee, Active recognition through next view planning: a survey, Pattern Recognition 37 (2004) 429–446.

[33] J. Tegin, S. Ekvall, D. Kragic, J. Wikander, B. Iliev, Demonstration-based learning and control for automatic grasping, Intelligent Service Robotics 2 (2009) 23–30.

[34] P. Pastor, H. Hoffmann, T. Asfour, S. Schaal, Learning and generalization of motor skills by learning from demonstration, in: 2009 IEEE International Conference on Robotics and Automation, 2009, pp. 763–768. doi:10.1109/ROBOT.2009.5152385.

[35] S. Levine, C. Finn, T. Darrell, P. Abbeel, End-to-end training of deep visuomotor policies, J. Mach. Learn. Res. 17 (1) (2016) 1334–1373.
URL http://dl.acm.org/citation.cfm?id=2946645.2946684

[36] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, S. Levine, Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration, in: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2018, pp. 3758–3765.

[37] S. El Zaatari, M. Marei, W. Li, Z. Usman, Cobot programming for collaborative industrial tasks: An overview, Robotics and Autonomous Systems (2019).

[38] P. V. Molle, T. Verbelen, E. D. Coninck, C. D. Boom, P. Simoens, B. Dhoedt, Learning to grasp from a single demonstration, CoRR abs/1806.03486 (2018). arXiv:1806.03486.
URL http://arxiv.org/abs/1806.03486

[39] OSGi Alliance, Osgi service platform, release 3, IOS Press, Inc., 2003.

[40] E. D. Coninck, S. Bohez, S. Leroux, T. Verbelen, B. Vankeirsbilck, B. Dhoedt, P. Simoens, Middleware platform for distributed applications incorporating robots, sensors and the cloud, in: 2016 5th IEEE International Conference on Cloud Networking (Cloudnet), 2016, pp. 218–223. doi:10.1109/CloudNet.2016.23.

[41] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, ROS: an open-source Robot Operating System, in: ICRA Workshop on Open Source Software, 2009.

[42] E. D. Coninck, S. Bohez, S. Leroux, T. Verbelen, B. Vankeirsbilck, P. Simoens, B. Dhoedt, Dianne: a modular framework for designing, training and deploying deep neural networks on heterogeneous distributed infrastructure, Journal of Systems and Software 141 (2018) 52 – 65. doi:https://doi.org/10.1016/j.jss.2018.03.032.
URL http://www.sciencedirect.com/science/article/pii/S0164121218300487

[43] T. Verbelen, P. Simoens, F. D. Turck, B. Dhoedt, Aiolos: Middleware for improving mobile application performance through cyber foraging, Journal of Systems and Software 85 (11) (2012) 2629 – 2639. doi:10.1016/j.jss.2012.06.011.

[44] K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman, Return of the devil in the details: Delving deep into convolutional nets, in: Proceedings of the British Machine Vision Conference, BMVA Press, 2014. doi:http://dx.doi.org/10.5244/C.28.6.

[45] K. Hara, R. Vemulapalli, R. Chellappa, Designing deep convolutional neural networks for continuous object orientation estimation, CoRR abs/1702.01499 (2017). `arXiv:1702.01499`.
URL `http://arxiv.org/abs/1702.01499`

[46] D. Kingma, J. Ba, Adam: A method for stochastic optimization, International Conference on Learning Representations (12 2014).