International Journal of Robotics and Automation, Vol. 38, No. 10, 2023
An Open Access Paper

VISUAL SERVOING IN VIRTUALISED ENVIRONMENTS BASED ON OPTICAL FLOW LEARNING AND CONSTRAINED OPTIMISATION

Takuya Iwasaki,* Solvi Arnold,* and Kimitoshi Yamazaki*

Abstract

In this paper, we describe a visual servoing method for object picking. We propose a new architecture for generating robotic manipulator motions approaching a target object for grasping. The architecture consists of two convolutional neural networks (CNNs), one generating goal-directed motion and one collision avoidance motion. The networks' outputs are combined, along with additional constraints, such as motion ranges of the joints, by means of quadratic programming (QP). One issue with learning-based approaches is that large amounts of training data are required. We devise an operation strategy that reduces the amount of training data required using a physics simulator. This method enables visual servoing that is unaffected by texture and colour variation in real environments. We show the effectiveness of the proposed method in experiments using simple shapes as target objects.

Key Words

Visual servoing, neural network, virtual environment

1. Introduction

Bin picking is a common task in robotic manufacturing. For instance, operations, such as product assembly and packaging, begin with picking up the relevant parts. In such tasks, a robot approaches a target object with its hand from an appropriate direction, and then grasps the object. Similar actions are also common in the tasks of household robots. For example, when picking up an item from a table, a robot has to reach for it while avoiding obstacles. One major approach for accomplishing these actions is to use motion planning. In motion planning, complete motion sequences are generated before the robot starts to move. These are then sent to the robot for execution, upon which the actual reaching and grasping are performed.

Another choice is visual servoing [1]–[3]. In visual servoing, velocity command values for each joint of the manipulator are generated on basis of the current visual input and the visual input that the robot would see from the goal state. The manipulator then moves on basis of these commands. By repeating this process, it is possible to bring the hand into a pose from which the target object can be grasped. The advantages of visual servoing are that the robot can start moving immediately once the tracking target is determined, and its robustness against dynamic environments.

The purpose of this study is to construct a visual servoing system for generating reaching motions towards a target object. In traditional visual servoing, manipulator motion is determined by explicitly considering the image Jacobian. Conversely, in the present study, we propose an approach wherein manipulator movement is acquired through advance learning. One issue with learning-based approaches is that they require large amounts of training data. In this study, we reduce the data acquisition burden by collecting training data in a virtual environment. Furthermore, if the shape and pose of the target object and surrounding obstacles can be reproduced appropriately in the virtual environment, then the virtual environment can be used to perform visual servoing while the robot is performing real-world reaching tasks. Using simulation in this way, we can perform visual servoing that is unaffected by the texture variation presented by real environments.

The main contributions of this work are as follows:

- We show an approach to performing both training and actual visual servoing in a virtualised environment. This enables us to reduce the load of collecting training data by actual experiments.
- We propose a novel learning-based visual servoing architecture, which generates appropriate joint angle commands using two convolutional neural networks (CNNs) and then mediate the results by quadratic programming (QP) optimisation.

* Shinshu University, Matsumoto, Japan; e-mail: {21w4005c, s_arnold, kyamazaki}@shinshu-u.ac.jp
Corresponding author: Kimitoshi Yamazaki

Recommended by Gian Luca Foresti
(DOI: 10.2316/J.2023.206-0810)

- We confirmed the effectiveness of the proposed method in experiments. We show that suitable reaching motions can be generated for a diversity of obstacle placements, target object shapes, and target object poses.

The structure of this paper is as follows. In the next section, we introduce related work. In Section 3, we present our concept and the proposed visual servoing architecture. We explain the proposed method in Sections 4 and 5. Experimental results are reported in Section 6, and Section 7 concludes the paper.

2. Related Work

Visual servoing has long been studied in the field of robot vision [4]–[7]. Iwatani *et al.* [8] proposed a visual servoing method that is robust to occlusion, integrating visual tracking, and visual servo control into a vision-based control method with occlusion handling. Chesi and Hashimoto [9] controlled a robot such that certain feature points were always kept in the camera image. Their method switches back and forth between position-based control strategies and backward motion. Bakthavatchalam *et al.* [10] proposed photometric image moments as new visual features for image-based visual servoing. Castelli *et al.* [11] developed a visual servoing system for the automation of copper wire winding. Their framework introduced machine learning and synthesised visual servoing using a Gaussian mixture model. Vakanski *et al.* [13] proposed a framework for trajectory learning from demonstrations. They formulated the learning problem as a convex optimisation problem. Agravante *et al.* [12] solved visual servoing as a quadratic optimisation problem. By defining an acceleration-resolved form, they integrate visual servoing tasks into an existing whole-body control framework for humanoids. The method proposed in the present paper is inspired by the idea of Agravante's approach. However, we formulate the QP problem differently.

The problem of visual servoing can be divided into two parts: (a) formalising the relation between camera motion and visual input change and (b) controlling camera motion on basis of this formalisation. The former problem is usually addressed using the image Jacobian [14]. The image Jacobian can be obtained analytically if the relationship between local features between images can be identified clearly. However, this approach is difficult to use in cases where objects have few textural features. Therefore, methods automating the extraction of suitable image feature are also being proposed. One such method uses neural networks. Bateux *et al.* [15] proposed a method of mapping images to command values directly using a CNN. The present work adopts this approach for obtaining motion command candidates from image inputs. That is, image feature points are not needed in the proposed method. Tokuda *et al.* [16] presented a CNN-based visual servoing scheme. The proposed method named DEFINet makes it possible to do the positioning of an object even if there is a large displacement from the desired state.

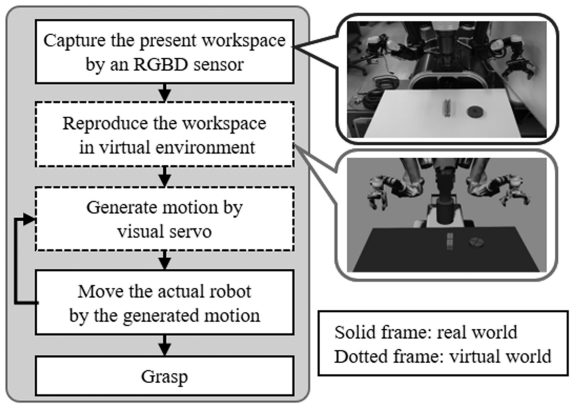

Figure 1. Concept of visual servoing [19]. In addition to training data collection, the virtual space is used when the robot physically performs a reaching task as well.

As a recent trend, many studies have aimed to acquire behavioural abilities for "observe-and-grasp" scenarios using reinforcement learning. Levine *et al.* [17] proposed a method to assess the probability that an action will succeed from a pair of images and the predicted gripping action of the robot. In the context of reinforcement learning in robotics, domain randomisation has been adopted to facilitate Sim-to-real transfer [18]. In the present study too, a simulation environment is used, but here we use the simulated environment not just for training, but also during real-world motion execution. This approach makes it possible to reduce the load on training data collection.

3. Concept and Structure of Visual Servoing

3.1 Visual Servoing Concept

We assume a task setting where a robot equipped with a manipulator is positioned in front of a table, with a target object to be grasped located on the table. Additionally, there may be obstacle objects on the table as well. The shapes of the objects are arbitrary, but should be reasonably approximable with primitive shapes such as rectangular cuboids or cylinders.

Assuming the abovementioned problem setting, Kawagoshi *et al.* [19] proposed the concept shown in Fig. 1. An RGBD sensor is installed so as to face the table surface directly. The depth information obtained from the sensor is used to estimate the approximate shape, position, and orientation of the object to be grasped. The scene is then virtualised in the simulator, reproducing the robot and table setup of the actual environment. A colour camera is attached to the wrist of the virtual robot. The camera's line of sight is aimed towards the tip of the hand.

Before executing a picking task, we first prepare a goal image as follows. Using the simulation, we capture a colour image of the moment just before the target object is grasped, using the virtual colour camera on the wrist of the simulated robot. Then the grasping task starts. First, the actual robot performs sensing, and the object arrangement

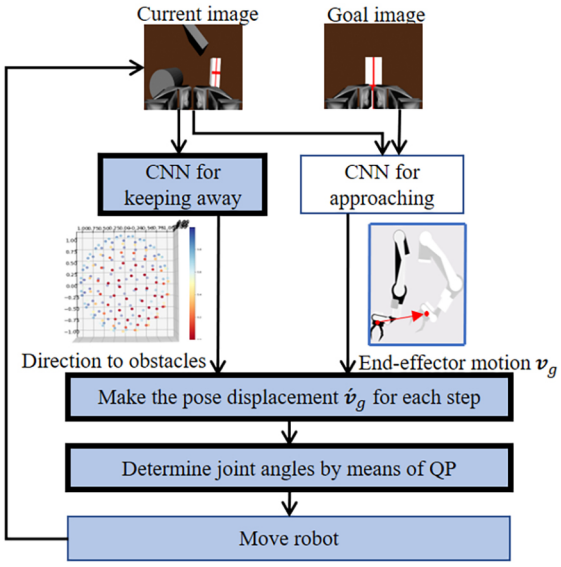

Figure 2. The overall structure of the proposed visual servoing. The red squares are the newly added parts from our previous work [19].

on the table is reproduced in the virtual space. Next, visual servoing is performed on basis of the goal image and the colour image captured by the virtual robot's wrist camera in the current state. That is, each joint of the manipulator is controlled by visual servoing so as to approach the goal image. Visual servoing is repeated until the goal image and the current image have become sufficiently similar for the robot to be able to grasp the target object by simply closing its hand.

In the procedure described above, the actual robot does not need to have a camera equipped on its wrist. By using the virtual space to perform visual servoing while the actual robot is physically performing the reaching task, it is possible to perform visual servoing unaffected by the textures of the surrounding physical environment. Moreover, this approach allows for the collection of training data to be performed in the virtual space exclusively. This eliminates the burden of data collection on the physical experimental setup.

In the method of [19], the presence of obstacles was not considered. That is, there was no ability to avoid obstacles on the trajectory of the arm that moves toward the object. However, obstacle avoidance is often required when reaching to an object placed on a table, for example. Therefore, in this study, we maintain the advantage of the previous work while extending the functionality to enable both approaching the target object and avoiding collisions with obstacles.

3.2 Visual Servoing Architecture

Figure 2 shows the global architecture of the proposed visual servoing method. There are three squares with red borders, which are new components added from our previous study. The input data is a tuple of images, consisting of the current image and the goal image to be observed by the end of the reaching motion. The current image is presented as input to a CNN tasked with generating motions away from obstacles. We refer to this CNN as $\text{CNN}_{\text{avoid}}$ below. A second CNN takes both the current image and the goal image as input and generates motions for bringing the wrist camera to its intended goal pose. We refer to this CNN as $\text{CNN}_{\text{approach}}$ below. The motion output from the two CNNs is combined to compute the actual hand displacement $\acute{\boldsymbol{v}}_g$ for the present frame.

Displacement $\acute{\boldsymbol{v}}_g$ is not guaranteed to be physically executable on the robot being used. Furthermore, if the motion causes the target object to leave the camera view even temporarily, it may cause visual servoing to break down. Therefore, we adjust $\acute{\boldsymbol{v}}_g$ on basis of two additional constraints. Specifically, we apply QP with constraints to keep the generated joint angles within the admissible range of each joint, and to keep the target object within the camera's field of view at all times. The resulting motion commands are then sent to the manipulator's joints for execution.

In the next section, we explain the two CNNs architectures, and our method for deriving displacement $\acute{\boldsymbol{v}}_g$ from the networks' outputs. In Section 5, we detail the QP optimisation process.

4. Motion Generation Using CNNs

4.1 Generating Goal-directed Motion

$\text{CNN}_{\text{approach}}$ outputs a pose change $v_g = (v_x, v_y, v_z, v_\phi, v_\theta, v_\psi)$ for the robot's hand. The network takes two colour images as input data. Beyond the input layer, the internal network structure follows that of FlowNet 2.0 [20], an architecture originally proposed for the purpose of optical flow extraction. However, diverging from the original architecture, we introduce a fully connected layer on the output side with 2,048 in- and output elements. The final network output is the six-dimensional vector v_g.

Training of $\text{CNN}_{\text{approach}}$ proceeds as follows. We place a target object on the table in the virtualised environment. We set up the camera view, and, assuming that the target object is visible from the camera, capture an image from a random hand pose. Then we randomly move the hand into another pose and capture another image. We call the former image as Image 1, and the latter as Image 2, respectively. Repeating this procedure numerous times, we collect data tuples of image pairs and hand motions. Then we train the network by setting an image pair as input and the corresponding hand motion as target output, and updating connection weights accordingly. The loss function for the training consists of the squared error of hand pose, which is represented by six-dimensional vector:

$$loss = \|\Delta\widetilde{\boldsymbol{x}} - \Delta\widehat{\boldsymbol{x}}\|^2$$

where $\Delta\boldsymbol{x}$ is the vector obtained by subtracting the hand pose when capturing Image 1 from the hand pose when capturing Image 2. $\Delta\widetilde{\boldsymbol{x}}$ is the true value and $\Delta\widehat{\boldsymbol{x}}$ is its predicted value obtained from $\text{CNN}_{\text{approach}}$.

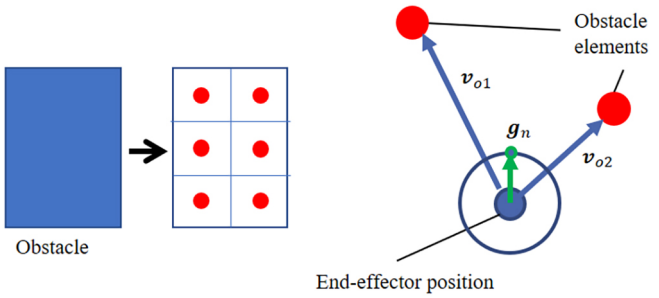

Figure 3. Left: Conversion of an obstacle to a group of obstacle elements. Right: A vector from the hand center to the n^{th} measurement point, and vectors from the hand center to obstacle elements.

4.2 Generating Collision–Avoidance Motion

$\mathrm{CNN}_{\mathrm{avoid}}$ outputs a map of values representing collision safety. Its input is just the current state image. Beyond the input layer, the network structure follows $\mathrm{CNN}_{\mathrm{approach}}$, except here the output layer is a vector of collision safety values for populating the value map.

In order to train the network to produce appropriate safety values, we need to generate ground-truth data. Here we explain how we generate this data. We again place a target object on the table in the virtualised environment. Additionally, we place randomly shaped obstacle objects at random positions on the table. Then, we approximate the obstacles' shapes as collections of small cubes. We refer to these cubes as "obstacle elements" below. Figure 3 illustrates this process with a two-dimensional example. Next, we define a unit sphere surrounding the current hand position, and place measuring points on the surface of the sphere at suitable intervals. We let $\boldsymbol{g}_n$ denote the vector from the hand position to the n^{th} measuring point. We let $\boldsymbol{v}_{\mathrm{om}}$ denote the vectors from the hand position to obstacle elements, with m indexing the obstacle elements. We then compute the evaluation value G_n for $\boldsymbol{g}_n$ as follows.

$$G_n = \max\left\{\boldsymbol{g}_n \cdot \frac{\boldsymbol{v}_{o1}}{\|\boldsymbol{v}_{o1}\|}, \cdots, \boldsymbol{g}_n \cdot \frac{\boldsymbol{v}_{om}}{\|\boldsymbol{v}_{om}\|}, 0\right\} \quad (1)$$

In other words, we calculate the cosine similarity for $\boldsymbol{g}_n$ against the vectors to all obstacle elements, select whichever is largest from this set of similarities or 0, and use the resulting value as the evaluation value. The right-hand side of Fig. 3 illustrates the idea, simplified to two dimensions. The lower the evaluation value for a given measuring point, the lower the risk of collision with an obstacle for movements toward that point. The above procedure treats the hand as a point. Consequently, when an obstacle exists particularly close to the hand, edges of the hand may collide with it even when moving in the direction of an evaluation point with a low G_n value. We avoid this as follows. We add reference points on the edges of the hand and compute additional cosine similarities over $\boldsymbol{g}_n$ and the vectors from these points to the obstacle elements. We add the resulting values to the set we max over in (1). Consequently, when motion towards a measuring point would cause an edge part of the hand to approach an obstacle, that measuring point receives a high evaluation value, just like when the motion would move the centre of the hand towards an obstacle.

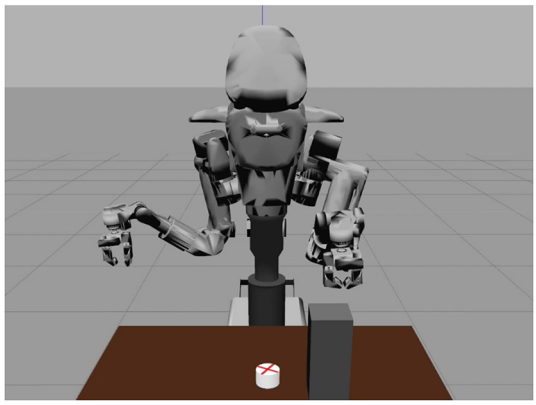

Figure 4. A snapshot of the virtualised environment.

For the training of $\mathrm{CNN}_{\mathrm{avoid}}$ that works as describes above, the following loss function is used.

$$\mathrm{loss} = \frac{1}{N}\sum_{n=1}^{N}\left\|\widetilde{G}_n - \widehat{G}_n\right\|^2,$$

where $\widetilde{G}_n$ is the true value, and $\widehat{G}_n$ is a predicted value obtained from $\mathrm{CNN}_{\mathrm{avoid}}$.

4.3 Generating Motion from CNN Outputs

The routines detailed in Sections 4.1 and 4.2 above provide us with two types of information for determining hand motion. Here we combine these to obtain a single pose change vector $\acute{\boldsymbol{v}}_g$. The procedure is as follows. First, from the set of points for which evaluation value G_n as computed in (1) falls below the threshold t_g, we select the point for which $\boldsymbol{g}_n \cdot \boldsymbol{v}_{g(xyz)}$ produces the largest value. Here $\boldsymbol{v}_{g(xyz)}$ refers to the position elements of the vector $\boldsymbol{v}_g$ given by $\mathrm{CNN}_{\mathrm{approach}}$. We call this point $\boldsymbol{g}_{\mathrm{max}}$. We then construct pose change vector $\acute{\boldsymbol{v}}_g$ as follows.

$$\acute{\boldsymbol{v}}_g = \begin{pmatrix} |\boldsymbol{v}_{g(\mathrm{xyz})}|\,\boldsymbol{g}_{\mathrm{max}} \\ \boldsymbol{v}_{g(\varphi\theta\psi)} \end{pmatrix} \quad (2)$$

Here $\boldsymbol{v}_{g(\varphi\theta\psi)}$ refers to the orientation elements of $\boldsymbol{v}_g$.

In (2), hand rotation and the length of the translational motion are retained from $\boldsymbol{v}_g$, whereas the direction of the motion is replaced by $\boldsymbol{g}_{\mathrm{max}}$. As $\boldsymbol{g}_{\mathrm{max}}$ corresponds to the collision-free direction that brings us closest to the target, defining $\acute{\boldsymbol{v}}_g$ as above lets us approach the target under strict collision avoidance conditions. This strategy was found to be effective in the authors' experimentation.

However, depending on the placement of obstacles, the evaluation value of the point in the direction of the target pose continues to fall below the predetermined threshold value t_g, and deadlock occurs. To prevent this, t_g is updated as follows for each step.

$$t_g^{n+1} = \begin{cases} t_g^n + \varepsilon, & \text{if } \boldsymbol{v}_{g(\mathrm{xyz})} \cdot \boldsymbol{g}_{\mathrm{max}} \le 0 \\ t_g^n, & \text{if } \boldsymbol{v}_{g(\mathrm{xyz})} \cdot \boldsymbol{g}_{\mathrm{max}} \ge 0 \end{cases} \quad (3)$$

Here, ε is a small positive constant and t_g^n represents the threshold at the nth step. In this equation, when the target motion for each step is in the opposite direction from the final target pose, the constraint due to obstacles is relaxed in the next step. This increases the risk of collision with obstacles, but in our experience, obstacles can almost certainly be avoided by setting the initial threshold t_g^0 small.

5. Mediation of CNNs' Result by QP Optimisation

5.1 Formulation as QP Problem

The method described in the preceding section lets us generate pose change vectors for hand motions. However, there is no guarantee that these pose changes are physically realisable by adjusting the manipulator's joint angles. Hence, it is necessary to translate these motions into movement commands in a way that takes the robot's physical limitations into account. In the present work, we consider two constraints. The first is the range of motion of the manipulator's joints. The second is the need to keep the target object within the camera's field of view at all times. To obtain solutions that respect these restrictions, we perform optimisation by means of QP.

QP's basic form is as follows.

$$\begin{aligned} \text{minimise} \quad & f_o = \frac{1}{2}\boldsymbol{x^T}\,\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c^T}\boldsymbol{x} \\ \text{s.t.} \quad & \boldsymbol{f_c} = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} \leq \boldsymbol{0} \end{aligned} \tag{4}$$

Here $\mathbf{Q}$ is a real symmetric matrix, $\boldsymbol{A}$ is a matrix, and $\boldsymbol{b}, \boldsymbol{c}$ are vectors. The T on the right-hand side of (3) indicates transposition. We derive the value for $\boldsymbol{x}$ that best minimises the objective function.

For our proposed method, we define the objective function as follows.

$$\text{minimise} \quad f_o = |\acute{\boldsymbol{v}}_g - \boldsymbol{v}|^2 \tag{5}$$

Here $\acute{\boldsymbol{v}}_g$ is the initially obtained per-step pose change and $\boldsymbol{v}$ is the properly constrained pose change to be derived. However, what we actually need are angular displacement values for the manipulator joints, which we denote as $\boldsymbol{q}$. We now let $\boldsymbol{J}$ denote the Jacobian matrix, substitute $\boldsymbol{v} = \boldsymbol{J}\boldsymbol{q}$ in (5), and perform the following transformation.

$$\text{minimise } f_o = |\acute{\boldsymbol{v}}_g - \boldsymbol{J}\boldsymbol{q}|^{\boldsymbol{2}} = \boldsymbol{q^T}\boldsymbol{J^T}\,\boldsymbol{J}\boldsymbol{q} - 2\acute{\boldsymbol{v}}_g{}^T\boldsymbol{J}\boldsymbol{q} + \acute{\boldsymbol{v}}_g{}^T\acute{\boldsymbol{v}}_g \tag{6}$$

The minimisation problem now amounts to deriving the value $\boldsymbol{q}$ that minimises $\boldsymbol{q^T}\boldsymbol{J^T}\,\boldsymbol{J}\boldsymbol{q} - 2\acute{\boldsymbol{v}}_g{}^T\boldsymbol{J}\boldsymbol{q}$ in the rightmost expression in (6). This in turn amounts to substituting $\boldsymbol{Q} = \boldsymbol{J}^T\boldsymbol{J}$ and $\boldsymbol{c^T} = \acute{\boldsymbol{x}}_{\boldsymbol{g}}{}^{\boldsymbol{T}}\boldsymbol{J}$ in (3) and solving the resulting minimisation problem.

5.2 Constraints

To generate motions that are physically executable for a given robot, we impose a few constraints. The first concerns the range of motion for each of the robot's joints. Assuming that vectors $\overline{\boldsymbol{q}}$, give the upper and lower limits, respectively, for all of the relevant joints, we define $\boldsymbol{A}$ and $\boldsymbol{b}$ for (4) as follows.

$$\boldsymbol{A} = \begin{pmatrix} \boldsymbol{I} \\ -\boldsymbol{I} \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} \overline{\boldsymbol{q}} \\ -\underline{\boldsymbol{q}} \end{pmatrix} \tag{7}$$

Here $\boldsymbol{I}$ is the identity matrix.

The second constraint is that the object to be grasped should be kept within the camera's field of view at all times. The idea is as follows. We let $\boldsymbol{e}_t$ denote the coordinates of the target object within the image at time t. To obtain $\boldsymbol{e}_t$, we should first select coordinates in the world coordinate system corresponding to the surface or interior of the target object. For example, consider the case where the hand coordinates for the target pose are given by the 3D coordinates of a point reached by lowering the hand straight down from its current position by a given distance. If the hand camera's current pose and the camera parameters are known, we can calculate the coordinates of this point using perspective projection and coordinate transformation. Furthermore, given coordinates $\boldsymbol{e}_t$ and subsequent coordinates $\boldsymbol{e}_{t+1}$ for time $t+1$, we can calculate the displacement between the frames as follows.

$$\boldsymbol{e}_{t+1} = \boldsymbol{e}_t + \boldsymbol{L}\boldsymbol{J}\boldsymbol{q}. \tag{8}$$

Here $\boldsymbol{L}$ is the image Jacobian. To ensure that $\boldsymbol{e}_{t+1}$ stays within a given range, we use the constraint $\underline{\boldsymbol{e}} < \boldsymbol{e}_{t+1} < \overline{\boldsymbol{e}}$, where $\underline{\boldsymbol{e}}$, $\overline{\boldsymbol{e}}$ indicate the top-left and bottom-right coordinates of the rectangular area.

6. Experiments

6.1 Experimental Setup

We use the open-source robot simulator Gazebo [21] to create the virtual environment. We use HIRO (Kawada Robotics Inc.) as our virtual robot. HIRO is a humanoid robot with two 6-DOF manipulators, one axis for the waist and two axes for the neck. Figure 4 shows a snapshot of our simulation. In the present experiments, only the left arm is used. A virtual camera was set up to provide colour images at VGA resolution (*i.e.* 640 × 480 pixels). The camera is equipped on the wrist of the left arm, and installed so that it faces in the direction of the fingertips.

Target objects in our experiments are white with a red cross texture added on top, as shown in Fig. 5. Obstacles are grey. These tweaks were found to be effective for training the visual servoing system. Both training and operation are performed using the virtualised environment, so the same setup can be applied. The neural networks are implemented using tensorflow [22]. For QP optimisation, we use the cvxopt library [23]. The main specification of the machine used for training are as follows: CPU: Xeon 3.60GHz (Intel), GPU: Quadro p4000 (Nvidia), RAM: 64GB.

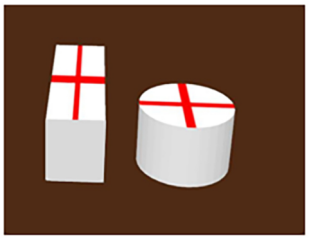

Figure 5. Virtually defined target objects. The cross texture is beneficial for learning translational and rotational camera motions.

6.2 Data Collection and Training for $\text{CNN}_{\text{approach}}$

We generate training data with two data collection strategies. The first addresses variability of target object positions, and proceeds as follows. We place the cuboid seen on the left in Fig. 5 on the table. Then we aim the camera such that the object is visible, and capture an image. Then we randomly move the hand some distance, and capture another image. Again, the target object should be visible. We store the pair of images along with the camera motion to the dataset. By repeating this procedure, we collect 20,000 examples. For camera pose selection, we define the ranges listed below.

- Range 1. Position range: x, y, z: $\pm$100 mm, relative to a reference point 150 mm above the target object's centre. Orientation range: φ, θ: $\pm5°$, ψ: $\pm30°$.
- Range 2a. Position range: x, y: $\pm$50 mm, z: $\pm$30 mm, relative to a reference point 80 mm above the target object's centre. Orientation range: φ, θ: $\pm5°$, ψ: $\pm10°$
- Range 2b. Position range: x, y, z: $\pm$10 mm, relative to a reference point 80 mm above the target object's centre. Orientation range: φ, θ, ψ: $\pm5°$

Here φ, θ, and ψ denote rotation angles around the x-, y-, and z-axes, respectively. The x-axis extends forward from the robot, and the z-axis points upwards. In data collection, we (a) select both the initial and destination pose from Range 1, moving the hand position from the former to the latter, or (b) move the hand from a pose in Range 2a to a pose in Range 2b. We refer to the data collected with this method as Dataset A.

The reason for using different ranges is that the quality of motion required for visual servoing changes as the reaching action progresses. Sampling from Range 1 yields data with large differences between the images. This data allows the system to learn bold motions for when the current image is significantly different from the target image. Meanwhile, sampling motions from Range 2a to Range 2b allows us to improve final positioning accuracy when the current image closely resembles the target image.

The second data collection strategy addresses shape variability and proceeds as follows. We prepare primitive shapes, such as the cuboid and cylinder, shown in Fig. 5, and randomly vary their sizes within the following ranges: short side: 30–60 mm, long side: 40–150 mm, height: 20–60 mm. Again we place the objects on the table at random positions. Then we capture images while performing hand motions between poses with coordinates x, y drawn from the range $\pm$30 mm and z from the range $\pm$20 mm, relative to a reference point placed 80 mm above the object centre. Angles φ, θ are drawn from $\pm$ 5°, and ψ from $\pm10°$. Then, we capture examples of small motions by sampling positions from Range 2b. We collect a total of 20,000 examples with this collection strategy and refer to the resulting data as Dataset B. We also perform the same collection procedure with obstacle objects placed on the table surface, collecting a set of 12,000 examples that we refer to as Dataset C.

6.3 Data Collection and Training for $\text{CNN}_{\text{avoid}}$

Data collection for $\text{CNN}_{\text{avoid}}$ randomises target object shapes within the ranges given above in Section 6.2. Obstacles shapes are set in the same manner as well. We place zero to three obstacle objects on the table. Obstacle objects are placed such that their centre coordinates are at a horizontal distance of 70 to 120 mm from the hand position. For each generated example, we calculate the evaluation map introduced in Section 4.2. First, we place 180 measuring points on the sphere surface, using GSS Generator [24] to obtain near-uniform spacing between points. Hence, the n of Section 4 is 180 here. We convert each obstacle object into a set of cubes (obstacle elements) with a side length of 10 mm each. Then we calculate the evaluation value for each point.

Using the collected images as input and the corresponding evaluation maps as ground truth, we train $\text{CNN}_{\text{avoid}}$. Training logic is the same as for $\text{CNN}_{\text{approach}}$, except here we train on a single dataset for 500 epochs. The final loss here was 0.15.

6.4 Simulation-based Experimental Results

We let $\text{CNN}_{\text{avoid}}$ estimate obstacle presence around the hand. Figure 6 shows example results. Images on the left are given to the network as input, and the graphs on the right show the corresponding network outputs. In the graphs on the right, differences in color intensity imply differences in the ease of moving the hand. In the top example, we see that it is easy to move in the direction of pulling toward the front, while in the bottom example, we see that the presence of the obstacle on the left side of the image is detected correctly.

Figure 7 shows an example of successful visual servoing in a scene with an obstacle object on the table. The top four images show snapshots of the robot's motions, proceeding from panel (1) to (4) in order. We see that the hand approached the target without colliding with the obstacle. The bottom three images show, from left to right, the initial image from the hand camera, the goal image, and the image actually obtained by the end of visual servoing. We see that even from an initial state where the target object is only partially visible, the system was able to generate motions that resulted in the capture of an outcome image closely resembling the goal image. Figure 8 shows the pose transition over time for this example. The pose stabilised in about 40 iterations. No oscillation was observed at any time over the course of the motion.

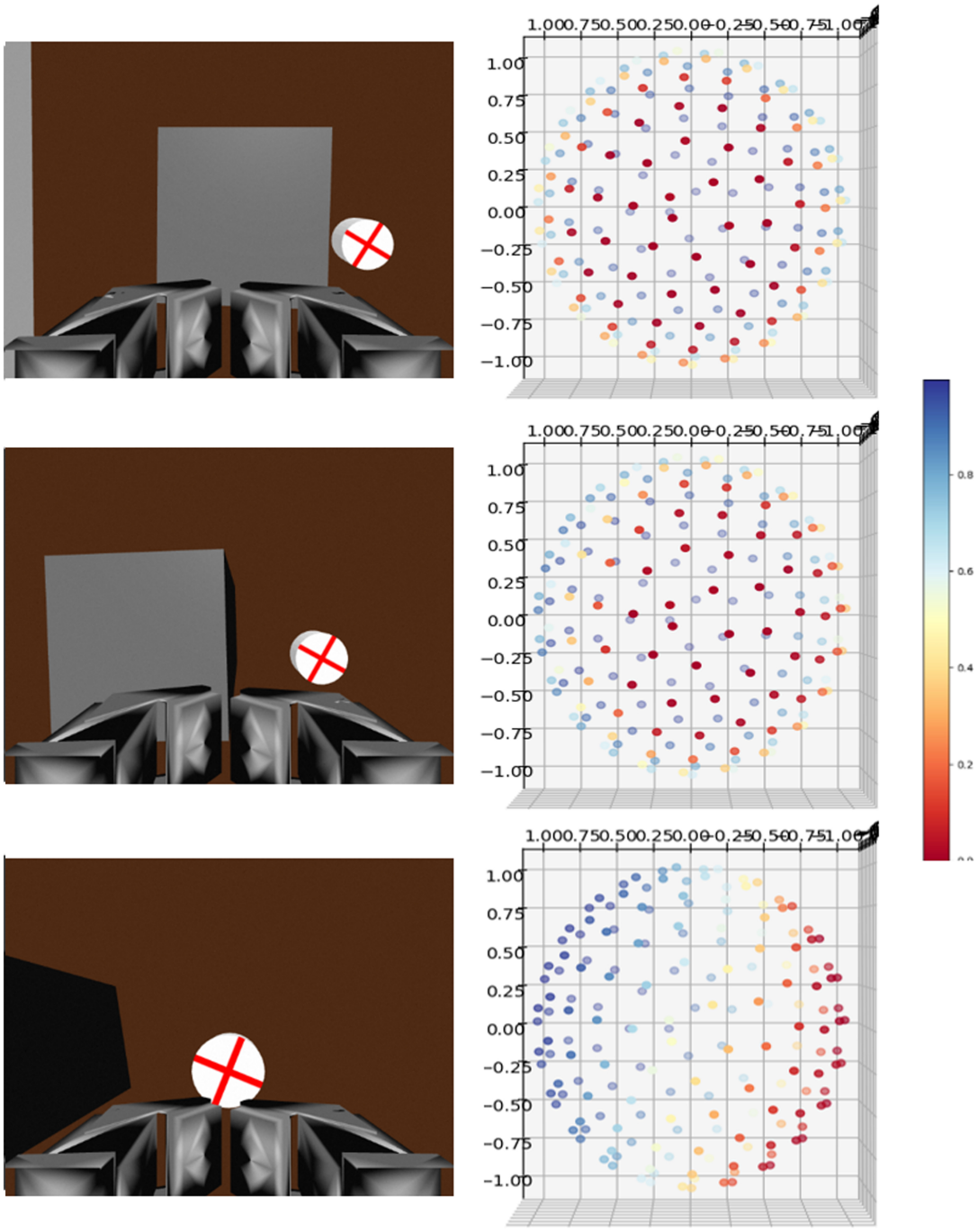

Figure 6. Verification examples of obstacle detection. The value of each directional point changes according to the risk of collision.

Figure 9 shows examples of visual servoing outcomes for a variety of configurations of the target and obstacle objects. The top row shows the final poses from an onlooker's perspective, while the bottom row shows corresponding images captured from the hand camera. The robot nonetheless managed to approach the target while avoiding obstacles, and finally assume a pose nearly identical to the pose from which the goal image was captured. We performed a total of 40 visual servoing sessions. Table 1 shows the mean and standard deviation of the error for each pose variable, calculated over the final poses obtained in these sessions. The time cost for calculating hand motions was approximately 1.2 s per iteration.

For comparison with the proposed method, we implemented DEFINet proposed by Tokuda *et al.* [16] and trained it on our dataset. DEFINet is a CNN-based visual servoing scheme that makes it possible to do the positioning of an object even if a large displacement from the desired state, like the present study. GlobalPooling incorporated in DEFINet learning is a process that greatly reduces the number of variables in the NN, and is said to have the ability to suppress overlearning. The authors of DEFINet also claim good positioning accuracy. The results show that

Table 1
Quantitative Result of Visual Servoing

	Δx[mm]	Δx[mm]	Δz[mm]	$\Delta\varphi$[deg]	$\Delta\theta$[deg]	$\Delta\psi$[deg]
Average	1.72	3.21	2.61	1.42	0.72	0.33
Std. dev.	1.35	4.62	2.25	2.14	0.55	0.50
Norm	$\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2} = 4.5$			$\sqrt{\Delta\phi^2 + \Delta\theta^2 + \Delta\psi^2} = 1.6$		

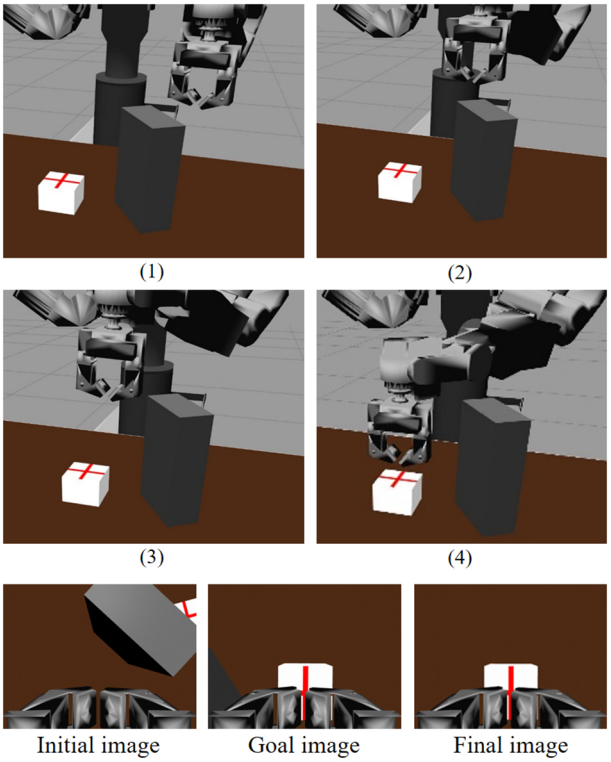

Figure 7. An example of visual servoing.

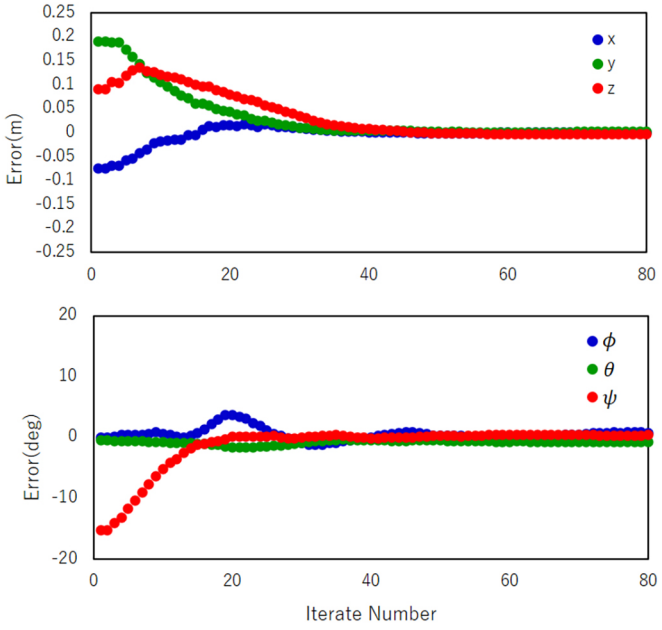

Figure 8. Transition of the end-effector pose in the experiment is shown in Fig. 7. Both position and orientation approached the goal without vibration.

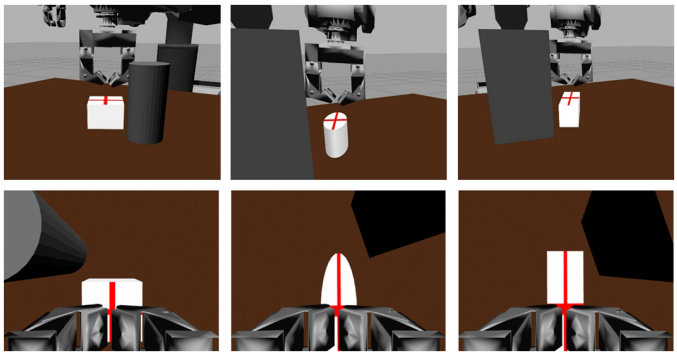

Figure 9. Examples of visual servoing. Upper panels show the final end-effector pose and bottom panels show the images captured at the final pose.

the norm of the position error is 12.4 mm and that of the azimuth error is 3.8°. Comparing these values to those in the bottom row of Table 1, it can be seen that the proposed method has more than twice the positioning accuracy.

We discuss this result. In the present study, data is collected in a virtual environment, which provides a sufficient amount of data with a small amount of effort without concern for overlearning. When the amount of data is sufficient, the positioning accuracy of our visual servoing structure was higher than that of DEFINet. In other words, the proposed combination of the appropriate use of the virtual environment and the new visual servoing structure resulted in a system that achieves high performance.

Next, the relationship between the number of training data and positioning accuracy was examined. In the aforementioned experiment, 52,000 (= Dataset A: 20,000 + Dataset B: 20,000 + Dataset C: 12,000) data were used. While maintaining this ratio, the number of training data was reduced and a neural network was trained. The accuracy of the resulting hand positioning was then evaluated. Table 2 summarises the results of calculating the norm of the error for position and orientation, respectively.

When the number of data was 6,500, the positioning error became so large that grasping was difficult. Therefore, it was found that at least 10,000 training data were necessary for the grasping task. Note that the training data used in this experiment was collected while changing the shape and position of the grasping target and obstacles to

Table 2
No. of Data versus Positioning Accuracy

No. of data	52,000	26,000	13,000	6,500
Position error [mm]	4.5	19.8	22.3	39.3
Orientation error [deg]	1.6	6.5	8.8	15.5

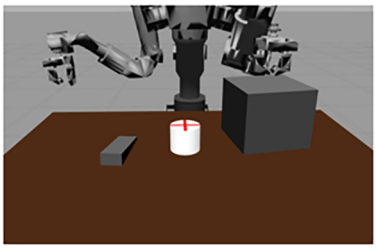

Figure 10. Reproduced virtual environment.

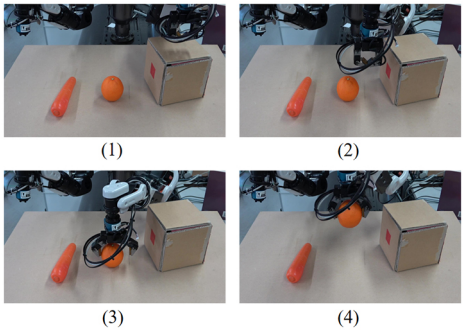

Figure 11. An example of the experimental results.

accommodate a variety of desk environments. Therefore, collecting such training data in a real environment will be very costly. Our framework can virtualise this part of the process, which greatly reduces the user's effort.

6.5 Experiment Using an Actual Robot

A target object and some obstacle objects were placed on a table, and an actual HIRO robot was placed on the front of the table. In this experiment, the given task is to pick up one known object placed on the table. Before making robot motions, the actual environment should be reproduced in the virtual environment. For this, instance, segmentation method, YOLACT [25], was used. First, where and what object exists in the image was estimated, then the 3D point cloud of a target object was obtained by referring the depth value captured from an RGBD camera. Next, a template point cloud that approximates the object to a rectangular parallelepiped or sphere was used as a reference, iterative closest point (ICP) algorithm was applied to find approximated pose of the target object. For other items placed on the desk, the size and orientation were roughly estimated by principal component analysis. They appeared in the virtual environment as a rectangular parallelepiped. Figure 10 shows an example environment reproduced. Figure 11 shows an experimental

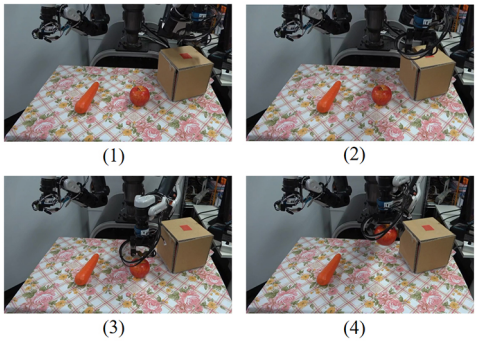

Figure 12. One case of the background with rich texture.

result. We conducted a total of 20 experiments, 4 times each, for 5 types of objects. Grasping was successful in all experiments. Figure 12 shows an experimental example when a tablecloth with a lot of textured is laid. The proposed method is not affected by such textures, so reaching and grasping were achieved without problems.

7. Conclusion

In this paper, we proposed a novel visual servoing method for generating reaching motions towards objects to be grasped. Manipulator movement is acquired through learning. The proposed method uses two CNNs to obtain the motion for goal-oriented and collision avoidance, respectively, and then mediate the motions by QP optimisation. We demonstrated the effectiveness of our method in experiments using various primitive shapes. We showed effective learning procedure which uses three types of data. We showed that with about 52,000 training data, hand positioning can be achieved with less than 5-mm accuracy in position, and less than 2° accuracy in orientation. By combining image segmentation, shape primitive fitting, and the proposed visual servoing, a real robot successfully grasped an object on a table. The robot's motion did not fail or vibrate, even if there was rich texture on the background. These results show that a certain degree of stability and reliability was confirmed.

Future directions include speeding up the visual servoing process. We also confirm that the proposed method works even in a more complicated environment. Furthermore, this paper presented comparative experiments between the proposed method and a method with a similar structure, but since end-to-end visual servoing methods have been proposed in recent years [25], [26], it is also important to compare them and summarise key points.

References

[1] L.E. Weiss, A.C. Sanderson, and C.P. Neuman, Dynamic sensor-based control of robots with visual feedback, *IEEE Journal of Robotics and Automation*, *RA-3*(5), 1987, 404–417.

[2] B. Espiau; F. Chaumette, and P. Rives, A new approach to visual servoing in robotics, *IEEE Transactions on Robotics and Automation*, *8*(3), 1992, 313–326.

[3] D. Kragic and H. Christensen, Robust visual servoing, *The International Journal of Robotics Research 22*(10–11), 2003, 923–939. doi:10.1177/027836490302210009

[4] F. Chaumette, and S. Hutchinson, Visual servo control. Part I. Basic approaches, *IEEE Robotics & Automation Magazine*, *13*(4), 2006, 82–90.
[5] J.P. Bandera, J.A. Rodríguez, L. Molina-Tanco, and A. Bandera, A survey of vision-based architectures for robot learning by imitation, *International Journal of Humanoid Robotics*, *9*(1), 2012, 1250006.
[6] S. Benhimane, and E. Malism, Homography-based 2D visual tracking and servoing, *The International Journal of Robotics Research*, *26*(7), 2007, 661–676.
[7] G. Silveira, and E. Malis, Direct visual servoing: Vision-based estimation and control using only nonmetric information, *IEEE Transactions on Robotics*, *28*(4), 2012, 974–980.
[8] Y. Iwatani, K. Watanabe, and K. Hashimoto: Visual tracking with occlusion handling for visual servo control, *Proc. of IEEE Int'l Conf. on Robotics and Automation*, Pasadena, CA, 2008, 101–106.
[9] G. Chesi, K. Hashimoto, D. Prattichizzo, and A. Vicino, Keeping features in the field of view in eye-in-hand visual servoing: A switching approach, *IEEE Trans. on Robotics*, *20*(5), 2004, 908–913.
[10] M. Bakthavatchalam, F. Chaumette, and E. Marchand, Photometric moments: New promising candidates for visual servoing, *Proc. IEEE Int. Conf. on Robotics and Automation*, Karlsruhe, 2013, 5241–5246, doi: 10.1109/ICRA.2013.6631326.
[11] F. Castelli, S. Michieletto, S. Ghidoni, and E. Pagello. A machine learning-based visual servoing approach for fast robot control in industrial setting, *International Journal of Advanced Robotic Systems*, *14*(6), 2017. doi:10.1177/1729881417738884
[12] D.J. Agravante, G. Claudio, F. Spindler, and F. Chaumette, Visual servoing in an optimization framework for the whole-body control of humanoid robots, *IEEE Robotics and Automation Letters*, *2*(2), 2017, 608–615, doi: 10.1109/LRA.2016.2645512.
[13] A. Vakanski, F. Janabi-Sharifi, and I. Mantegh, An image-based trajectory planning approach for robust robot programming by demonstration, *Robotics and Autonomous Systems*, *98*, 2017, 241–257.
[14] F. Chaumette, and S. Hutchinson, Visual servo control, Part I: Basic approaches, *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, Dec. 2006.
[15] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke, Training deep neural networks for visual servoing, *Proc. of IEEE Int. Conf. on Robotics and Automation*, Brisbane, QLD, 2018, pp. 3307–3314.
[16] F. Tokuda, S. Arai, and K. Kosuge: Convolutional neural network-based visual servoing for eye-to-hand manipulator, *IEEE Access*, *9*, 2021, 91820–91835, doi: 10.1109/ACCESS.2021.3091737.
[17] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection, *The International Journal of Robotics Research*, *37*(4–5), 2018, 421–436.
[18] S. Iqbal, J. Tremblay, T. To, J. Cheng, E. Leitch, A. Campbell, K. Leung, D. McKay, and S. Birchfield, *Toward sim-to-real directional semantic grasping*, 2019, arXiv preprint arXiv:1909.02075.
[19] T. Kawagoshi, S. Arnold, and K. Yamazaki, Visual servoing using virtual space for both learning and task execution, *Proc. of the 2021 IEEE/SICE International Symposium on System Integration*, Fukushima, 2021, 292–297.
[20] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazırbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox, Flownet: Learning optical flow with convolutional networks, *Proc. IEEE International Conf. on Computer Vision*, Santiago, 2015, 2758–2766.
[21] Gazebo: http://gazebosim.org/Point Cloud Library, https://pointclouds.org (accessed Nov. 14, 2022).
[22] Tensorflow, https://www.tensorflow.org/ (accessed Nov. 14, 2022).
[23] CVXOPT, https://cvxopt.org/ (accessed Nov. 14, 2022).
[24] A. Yamaji, GSS generator: A software to distribute many points with equal intervals on an unit sphere, *Geoinformatics*, *12*(1), 2001, 3–12.
[25] D. Bolya, C. Zhou, F. Xiao, and Y.J. Lee: YOLACT: Real-Time Instance Segmentation, *IEEE/CVF International Conference on Computer Vision*, 2019, 9156–9165.
[26] P. Katara, Y.V.S Harish, H. Pandya, A. Gupta, A. Mehdi Sanchawala, G. Kumar, B. Bhowmick, and K. Madhava Krishna, DeepMPCVS: Deep model predictive control for visual servoing, *Proc. 4th Annual Conf. on Robot Learning*, Cambridge, MA,, 2020, 1–10.
[27] E. Godinho Ribeiro, R. de Queiroz Mendes, and V. Grassi, Real-time deep learning approach to visual servo control and grasp detection for autonomous robotic manipulation, *Robotics and Autonomous Systems*, *139*, 2021, 103757.

Biographies

Takuya Iwasaki graduated from the Department of Mechanical Systems Engineering, Faculty of Engineering, Shinshu University in 2021. He is currently a member of the Interdisciplinary Graduate School of Science and Technology, Department of Engineering, Mechanical Systems Engineering Division, Shinshu University.

Solvi Arnold obtained the Ph.D. degree from the University of Nagoya in 2015. She was a JSPS Research Fellow from 2013 to 2015. In 2015, she joined Shinshu University, where she is currently a Project Associate Professor. Her research interests include artificial intelligence, intelligent robotics, and representation learning. She is a member of the Robotics Society of Japan.

Kimitoshi Yamazaki received the B.E., M.E., and Ph.D. degrees from the University of Tsukuba in 2002, 2004, and 2007, respectively. From 2006 to 2007, he was a JSPS Research Fellow. From 2007 to 2012, he was a Project Assistant Professor with the University of Tokyo. From 2010 to 2014, he was also a Researcher of PRESTO, JST. He joined Shinshu University in 2012, where he is currently a Professor. His research interests are in robot vision and motion planning. He is a member of RSJ, SICE, JSME, and IEEE.