Domestic Robot Grasping using Visual-Servoing and Deep Learning

João Manuel Pereira
joaomacpereira@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

November 2020

Abstract

Object grasping and manipulation are important capabilities for domestic service robots, with extensive research done in this area. This work proposes a grasping pipeline with a visual-servoing main component, used to precisely control the end-effector of a robotic arm towards the target object using visual feedback. A depth-camera, fixed to a mobile robot's head, observes both the object and the end-effector and estimates their positions using different methods: a neural network is used for object detection and combined with depth data for localization, and the end-effector position is obtained by tracking AR tags in the arm wrist. These camera-frame positions are used to calculate an error value which is independent from calibration of both the camera and the arm's joints. A proportional control law outputs the required end-effector velocity to minimize the error, and the arm is actuated using differential kinematics. The pipeline is used to successfully grasp several household objects in different positions, obtaining good results in a benchmark scenario to be used in European Robotics League competitions.

Keywords: Robotics, Manipulation, Visual-Servoing

1. Introduction

This work focuses on improving the grasping and manipulation functionalities of MBot, a domestic service robot. The goal is to develop a new system to grasp several household objects in a way that is less prone to calibration errors, with the purpose of achieving better results in benchmarks and robot competition challenges. This system should be integrated in the existing ROS ecosystem of the SocRob @Home team, such that members can easily configure and extend the system.

To improve the hardware capabilities of the MBot platform we installed a new robotic arm, designing and assembling new parts to accommodate it. The arm and its drivers were integrated with the existing software.

We propose a novel grasping system that can detect several household objects using supervised learning, and obtain their position using depth-sensor data. After moving the arm to a pregrasp pose, position-based visual-servoing is used to guide the gripper to the desired position, using visual markers on the gripper to track it. Since both the target and gripper positions are described in camera frame, the servoing error is not affected by either camera or joint calibration errors.

More precisely, the target object's position is obtained by combining the bounding-box output by a convolutional neural network for object detection, with depth information from the center of the bounding-box. The end-effector is located by tracking AR tags placed around the arm wrist.

A complete grasping pipeline is developed which obtains the scene's octomap, moves the arm to a pregrasp pose where the end-effector is visible to the camera, and activates visual servo control.

The system obtained good results in a grasping benchmark, and was shown to be tolerant to kinematic measurement errors. The solution was implemented as a ROS package, added to the SocRob team repository, and existing MBot packages were updated to use the new grasping capabilities.

2. Related Work

2.1. Robotic Grasping and Visual-Servoing

To make robotic grasping applicable to dynamic environments, a fixed point-to-point control scheme — used for decades in industrial robots [24] — is not enough. Vision-based grasping systems were developed for this purpose: using a camera to obtain the target object's pose, the manipulator can adapt to it.

Initial visual systems [17] were open-loop: the arm controller operated separately from vision. The

accuracy of this method depends directly on the accuracy and calibration quality of the camera and the manipulator [8].

The control loop is closed by using visual-feedback: in addition to the target object, the arm's end-effector pose is continuously estimated by the vision system, and corrected in real-time [22, 7]. This approach came to be known as *visual-servoing* (VS).

Visual-servoing has been widely used and developed in research, leading to the appearance of several variants. One of the most recent — Direct Visual-Servoing (DVS) [4, 2] — uses the full camera image as an input, removing the traditional requirement of placing artificial markers on the target object.

2.2. Object Grasping using Machine Learning

Standard visual-servoing approaches require application-specific engineering: manually-designed image features, *a priori* knowledge of the target object's 3D model, predefined grasp points, etc. More recently, machine learning methods have been applied to object grasping to make systems more general. Experiments have been developed using different types of learning for visual-servoing [5, 2], 3D pose estimation [25] and grasp pose detection [12, 10]. End-to-end learned systems have also been developed [13, 9], using reinforcement learning to train an agent that performs the full grasping task based only on input images.

In [5] the authors use an artificial neural network to compute the required end-effector movement to align a 3-DoF arm with a target object containing 4 visual markers. The image coordinates of the markers form the input vector for the neural network. Advances in deep-learning allow for a more recent approach [2] to use all the image pixels as input for a convolutional neural network (CNN), which calculates the required arm movement to reach a desired target image.

3. Theoretical Background

3.1. Kinematics

Frame Transformations In robotics it is common to deal with multiple *frames of reference*. In the example of a robot with a camera, it's useful to translate the spatial coordinates of a point from the camera frame to the robot frame.

We can represent a relationship between two frames by using a *homogeneous transformation matrix*. In Fig. 1, if p^c are the cartesian coordinates of a point P in the camera frame, and we wish to obtain p^r, the coordinates of P in the robot frame, we can do so by using T_c^r, the homogeneous transformation matrix from c to r:

$$p^r = T_c^r p^c \tag{1}$$

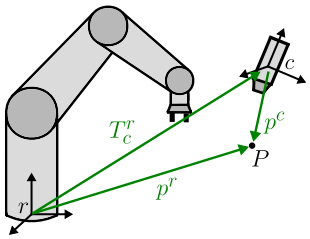

Figure 1: A point P described in different frames

This notation is used in the rest of this work.

Direct Kinematics Frame transformations are also used for manipulator kinematics. In *direct kinematics* we wish to compute the pose of the end-effector as a function of the joint angles q. A coordinate frame can be attached to each robot link, and the transformation describing the pose of link n with respect to link 0 is given by the kinematic chain:

$$T_n^0(q) = A_1^0(q1)A_2^1(q2)\cdots A_n^{n-1}(qn) \tag{2}$$

where A_i^{i-1} is the homogeneous transformation matrix between two consecutive link frames, function of qi, the current angle of joint i connecting the links.

Inverse Kinematics For manipulation tasks it is necessary to transform end-effector poses and motions into joint values and velocities. This is the *inverse kinematics* problem.

Regarding the direct kinematics equation (2), given a set of joint angles there is always one solution for the end-effector pose. The inverse kinematics problem is more complex: given an end-effector pose there can be multiple solutions, infinite solutions, or no solutions for the joint values. Calculating all exact solutions is challenging, as they depend on the degrees-of-freedom, mechanical joint limits and dexterous workspace of the manipulator [21].

The direct kinematics equations can be inverted to enumerate all solution branches, but this only works when the number of constraints for the end-effector pose is equal to the degrees-of-freedom of the robot. More general numerical techniques exist that can find approximate solutions for any kind of manipulator.

Differential Kinematics Differential kinematics is used to obtain the relationship between joint velocities and the corresponding end-effector velocity. This is done by calculating the Jacobian matrix, which depends on the manipulator configuration. End-effector velocity v_e is given by

$$v_e = J(q)\dot{q} \tag{3}$$

where $\dot{q}$ is the vector of linear joint velocities, and $J(q)$ is the Jacobian matrix, dependent on joint angles q.

Determining the joint velocities required for a certain end-effector velocity is possible by inverting (3):

$$\dot{q} = J^{-1}(q)v_e \tag{4}$$

This is the *inverse differential kinematics* problem. Since J may not be square or invertible, obtaining J^{-1} is not always possible. Instead, we can calculate the *Moore-Penrose inverse* (or *pseudoinverse*) $J^{\dagger}$, which is defined and unique for all numerical matrices. It can be obtained in several ways, but the most common computational method is using *singular value decomposition*, from which results:

$$J = UDV^T, J^{\dagger} = VD^{\dagger}U^T \tag{5}$$

Computing $J^{\dagger}$ this way, and using it in place of J^{-1} in equation (4) provides a computational method to obtain the joint velocities required for a desired end-effector velocity.

This method can be extended to solve inverse kinematics: instead of the desired end-effector velocity v_e, we can use direct kinematics to compute $\Delta p = p(q_0 + \Delta q) - p(q_0)$, the change in end-effector position given the current joint angle changes Δq. Δq can be iteratively improved using the Newton-Raphson method, minimizing an error function that measures distance to the desired end-effector position ($error = ||p(q_0 + \Delta q) - p_{desired}||$) [23].

3.2. Visual-Servoing

Visual-servoing is a method of robot control which uses visual feedback to control the robot's actuators continually, in closed-loop. Over time, several visual-servoing approaches have been developed, broadly characterized based on their camera configuration and control architecture.

Camera Configuration There are two typical configurations for the camera in visual-servoing systems: end-effector mounted (*eye-in-hand*), or fixed in the workspace (*eye-to-hand*) [8]. Fig. 2 shows a comparison between the two.

In *eye-in-hand* systems there is a fixed transform between the camera and the end-effector's pose, making for a more direct estimation of the target. *Eye-to-hand* systems can obtain a more panoramic view of the workspace, independent of the arm's orientation, however additional work is required to estimate the end-effector's pose in the image.

There have also been experiments on multi-camera systems which implement hybrid *eye-in-hand / eye-to-hand* approaches, merging information from both camera configurations [15].

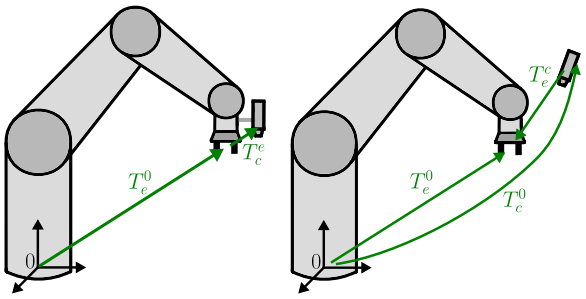

Figure 2: Camera configurations. **Left:** eye-in-hand. **Right:** eye-to-hand. On the latter, T_e^c needs to be computed by the vision system

Control Architecture Sanderson and Weiss [20] describe a taxonomy of visual-servoing systems including four categories, based on the way they answer two questions:

1. Does the system apply joint inputs (angles and/or velocities) directly, or does it use a joint controller for stabilization?

2. Is the error signal defined in terms of image features, or in 3D workspace frame using pose estimation?

The first question distinguishes between *classical* and *dynamic* visual-servoing. In the former, joint inputs are directly applied to the manipulator motors, while the latter uses joint sensor feedback to stabilize the motion. Since nowadays many manipulators include joint controllers, most recent systems use *dynamic* VS.

The second question distinguishes between *image-based* (IBVS) and *position-based* (PBVS) systems. In IBVS the error is calculated in the feature space of the 2D camera image. While simple, this method achieves only local asymptotical stability [3] as there is a loss of dimensional information. PBVS performs 3D localization of the object based on its features, and the resulting pose is used to control the manipulator in Cartesian coordinates. This operation can be expensive, but PBVS is shown to achieve global asymptotic stability [3].

Fig. 3 illustrates the internal architecture of *eye-to-hand*, *dynamic position-based* visual-servoing, the approach used in this work.

PBVS Control In eye-to-hand PBVS the kinematic error function is given by [8]:

$$E(T_e^0, h^e, g^0) = T_e^0 h^e - g^0 \tag{6}$$

T_e^0 is the transformation matrix from the end-effector frame (e) to the arm's root frame (0), which is the variable we can control by actuating the arm (the matrix is function of the joint angles). h^e are the coordinates, in end-effector frame, of a point on

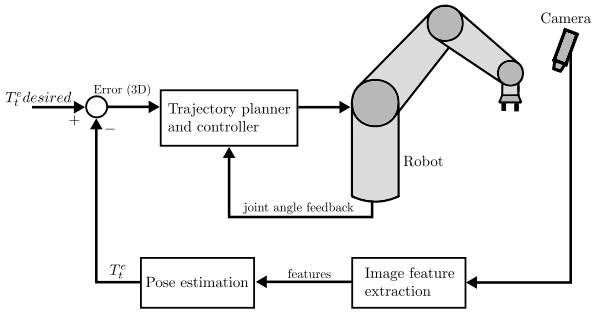

Figure 3: Dynamic position-based visual-servoing

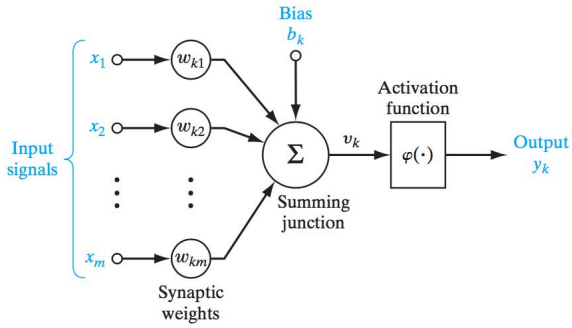

Figure 4: Architecture of a single neuron in a neural network [6]

the arm (the *tool* or *hand*) which we want to place in a fixed goal position g^0.

We wish to apply a linear velocity u^0 to the end-effector to minimize this error. Open-loop positioning can be achieved by applying the proportional control law

$$u^0 = -k\,(\hat{T}_e^0 h^e - \hat{T}_c^0 \hat{g}^c) \tag{7}$$

where $\hat{T}_e^0$ is shown as an estimate (as it is subject to errors in the arm's joint sensors), $\hat{T}_c^0$ is the estimated transformation matrix from the camera frame to the root frame (usually based on fixed transforms from manual measurements) and $\hat{g}^c$ is the estimated pose of the goal in camera frame, given by the vision system. $k > 0$ is a proportional feedback gain.

Because they are estimated, errors in $\hat{T}_e^0$, $\hat{T}_c^0$ or $\hat{g}^c$ will lead to end-effector positioning errors.

The control loop can be closed by continuously observing h^e (the hand) and estimating its position. Doing this, (7) turns into

$$u^0 = -k\,\hat{T}_c^0(\hat{h}^c - \hat{g}^c) \tag{8}$$

where $\hat{h}^c$ are the camera frame coordinates of the observed hand point, as estimated by the vision system.

In this equation u^0 doesn't depend on $\hat{T}_e^0$ anymore. Also, if $\hat{h}^c$ and $\hat{g}^c$ (dependent on the same camera calibration) are equal, then $u^0 = 0$ and equilibrium is reached, independently of errors in robot kinematics or camera calibration. This is the crucial advantage of visual-servoing algorithms.

3.3. Object Detection

To detect the object to grasp, our system employs the YOLO algorithm, which uses a convolutional neural network (CNN) for real-time object detection. This section introduces neural networks, explains CNNs, and specifies how YOLO uses them.

Neural Networks A neural network is a supervised learning technique which uses a connected network of artificial neurons (also called perceptrons) for classification. For each neuron, its input vector $\mathbf{x}$ is multiplied by a weight vector $\mathbf{w}$ and a bias value b is added. The result is fed to an activation function σ which computes the neuron's output y. Fig. 4 illustrates this process.

Given a dataset consisting of several input/target pairs $\{\mathbf{x},t\}$ the goal is for to learn a model of the relationship between $\mathbf{x}$ and t. To train the neural network is to obtain the weight vector $\mathbf{w}$ that produces a function y as close as possible to t.

For every training dataset example $\{\mathbf{x},t\}$ an error signal is calculated:

$$e = ||t - y(\mathbf{x})||^2 \tag{9}$$

with $y(\mathbf{x})$ given by the network with current weight vector $\mathbf{w}$. While this corresponds to quadratic error, other error functions can be used.

Then, the weights are adjusted using gradient descent, in a direction which minimizes the error:

$$w_{n+1} = w_n - \eta \frac{\delta e}{\delta \mathbf{w}} \tag{10}$$

η is the *learning rate* parameter, which can be set to a constant or be a decreasing function over the training time. The derivative of the error in respect to the weights $(\frac{\delta e}{\delta \mathbf{w}})$ can be calculated as a product of derivatives between each layer of the network, using the backpropagation algorithm [11].

Convolutional Neural Networks A Convolutional Neural Network (CNN) is a type of neural network architecture, commonly used to process images. The main component of CNNs is the *convolutional layer*: it consists of a set of learnable filters that slide through the image space and activate when they detect visual features such as edges or special shapes. The initial convolutional layer detects simple shapes, while deeper layers detect more complex patterns.

Between consecutive convolutional layers, *pooling layers* are inserted. Their purpose is to downsample the image representation, reducing its spatial size and therefore the amount of parameters, making the network more efficient and less susceptible to noise.

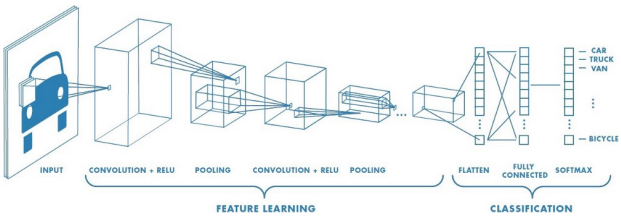

Figure 5: Full CNN architecture [16]

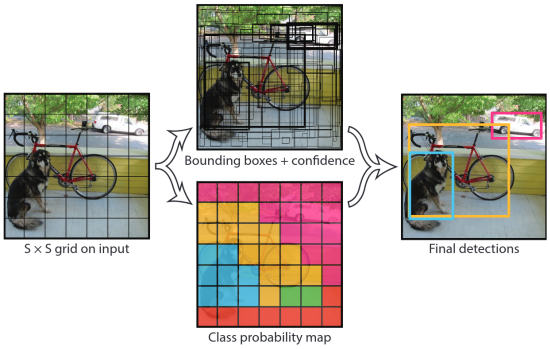

Figure 6: YOLO model predictions: bounding boxes and class probabilities are combined to obtain the final result: object labels and their bounding boxes [18]

Pooling layers don't add learnable parameters to the network, since they apply a fixed operation on the input.

Fully connected layers are commonly placed at the end of the CNN, after the highest level feature outputs. This is done so that nonlinear combinations of these high-level features can be learned. In the case of image classification, the size of the final output vector is equal to the number of detectable classes (see Fig. 5).

YOLO Object Detection Initially CNNs were mostly used for image *classification*, which finds the most relevant class (e.g. cat, flower, car, etc.) given an input image. Later, CNNs were adapted for object *detection*: identifying and locating several objects in an image. Early object detection systems were simply repurposed classifier networks, applying them at various image subregions. The location and scale of the subregions can also be learned parameters. The runtime performance of these systems is poor, since the classification process must be run many times.

YOLO (You Only Look Once) [18] is an object detection approach that uses a custom CNN to predict bounding boxes and class probabilities of objects. The network runs only once to predict bounding boxes and probabilities for all classes. This improves its runtime performance, making it able to perform real-time object detection at more than 30 frames per second.

The input image is divided into an $S \times S$ grid. Each cell predicts B bounding boxes centered on that cell. Confidence scores are assigned to the boxes, reflecting how likely it is for the box to contain an object, and how accurate the box is, measured by the intersection-over-union (IOU) metric.

Each cell also predicts object class probabilities $P(\text{Class}_i|\text{Object})$ in its vicinity.

The YOLO CNN has 24 convolutional layers, with maxpool layers in between them. 2 fully connected layers are placed at the end. The first 20 convolutional layers are pretrained on the ImageNet dataset, 4 additional layers are added to perform detection, and the model is trained based on ground-truth bounding boxes. The loss function penalizes a) classification error for cells that contain objects, and b) bounding box misalignment based on IOU.

4. Implementation

4.1. Robot Platform Integration

In order to improve MBot's grasping and manipulation capabilities we installed a new robotic arm and integrated it in the existing ROS ecosystem, allowing not only for its use in this work, but also for other future grasping pipelines.

Hardware Integration We replaced the previous arm in MBot with a Kinova Gen2 arm featuring more precise joint actuators and sensors, and ROS driver support. To aid our choice, we prototyped the arm in simulation before acquiring it.

Because of the arm's larger length we decided to mount the arm's base inside MBot's body, to make the whole robot more compact — see Fig. 7. To achieve this we modified the existing right-side plate in SolidWorks, adding a hole for the arm to pass through. Two copies of the part were machined using 2D CNC milling, and the double layer was installed in the robot replacing the previous single layer plate, adding support. We also designed a new part which attaches the arm's base to the inside of MBot's left-side plate. This part is made of acetal (also called polyoxymethylene or POM) and was machined using 3D CNC milling, according to our CAD model.

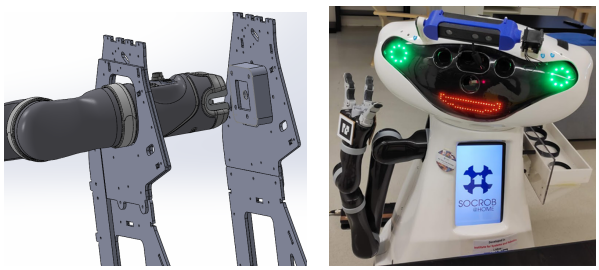

Figure 7: **Left**: Arm assembly with modified side-plate and new mounting fixture. **Right**: final result

Software Integration After installed, the arm

was integrated with MBot's ROS packages to allow for its observation and control. The first step was to change MBot's URDF (Unified Robot Description Format) file to include the arm's model, adding a fixed transform from the robot's body to the base of the arm, based on measurements. When the arm is connected, the driver package communicates with its joint sensors to update the URDF joint states, making the robot model mirror the actual robot joints.

The MoveIt! ROS framework includes several grasping and manipulation utilities including inverse kinematics, motion planning, obstacle avoidance based on 3D perception, visualization, interactive planning, etc. Its modular design allows for different algorithms to be used interchangeably. To make use of the MoveIt! pipeline, not only in this work but for other manipulation tasks on the same platform, a MoveIt! configuration package is necessary.

To create this package we use the MoveIt! Setup Assistant tool to load the URDF configured as above, select the movable joints and end-effector, and add pre-defined poses such as the folded resting pose.

Finally, we added several manipulation-specific functions to MBot Class — a python package aggregating the main functionalities used by the SocRob team in robot tasks. We expose several convenient arm functions such as moving to a certain absolute or relative pose, getting the current pose of the end-effector, and opening or closing the gripper.

4.2. Grasping System Implementation

Architecture A traditional visual-servoing control scheme was chosen, using supervised learning for the visual feedback, specifically for obtaining the target object's position. To achieve this, the YOLO CNN-based object detection algorithm is used to output a 2D bounding box over the part of the image containing the target object. By sampling image depth, the 3D position of the object's center-point is estimated.

Our system takes advantage of MBot's depth-camera, fixed to its head. This makes it an eye-to-hand system, requiring the end-effector's pose to be estimated by the vision system. To do this, three AR markers were added to the arm's wrist, and the ALVAR package is used to track their 3D position.

The error is given by the difference between the target object and end-effector positions, in camera frame. A proportional control law is used to obtain the end-effector velocity required to minimize the error. Using singular value decomposition to compute the Jacobian pseudo-inverse, the joint velocities are calculated and given as inputs to the arm's controller. Fig. 8 shows the diagram of the system.

As opposed to using a more end-to-end approach, our solution uses supervised learning as a "sensor" that provides the object position as an input for a classical position-based visual-servoing control algorithm (PBVS). This creates a separation of concerns between perception and control, facilitating debugging and error analysis. Assuming the target is correctly localized, the system takes advantage of the well-studied stability guarantees of PBVS [8, 3], whereas end-to-end systems must learn the control function through examples, requiring a large dataset of experimental data.

Target Position Estimation To localize the target object, a YOLO v3 CNN receives RGB images from the head-camera and outputs 2D bounding boxes around the recognized objects. We used a pre-trained network capable of identifying 80 common objects from the COCO dataset [14]. The network consists of 75 convolutional layers with leaky ReLU activation. Instead of pooling layers, downsampling is done by convolutional layers with a stride of 2 (sliding 2 pixels at a time), preventing the loss of low-level features. Bounding box detections are performed at three different image scales [19].

Since our system assumes knowledge about the type of object to grasp, only the bounding boxes corresponding to that class are considered. A region of points in the center of the bounding box is sampled, and the depth values of those points are obtained from the captured depth image. The 25^{th} percentile depth value is chosen as representative of the surface's depth d.

A pinhole camera model, created from the camera calibration parameters, is used to obtain a rectified image without lens distortion. The center pixel of the bounding box is chosen and a ray is projected from the lens, intersecting the rectified pixel. This ray is represented by the unit vector $\mathbf{r} = [x\ y\ z]^T$ which points in the direction of the object's center-point. The depth value is multiplied by this vector to obtain the 3D position of the goal point in camera frame: $\hat{g}^c = d \cdot \mathbf{r}$.

End-Effector Pose Estimation To localize the end-effector, the ALVAR ROS package was used. ALVAR is an open source library for tracking and localizing AR tags based on the size and angle of the observed visual features, and *a priori* knowledge of the marker's size and the camera's intrinsic parameters,

Only one visible marker is needed for accurate 3D pose estimation, but to improve camera visibility of the wrist independently of its rotation, we placed three different markers (IDs 0-2) around the wrist, as seen in Fig. 10. A bundle file is created which stores the relative position of two markers (IDs 1

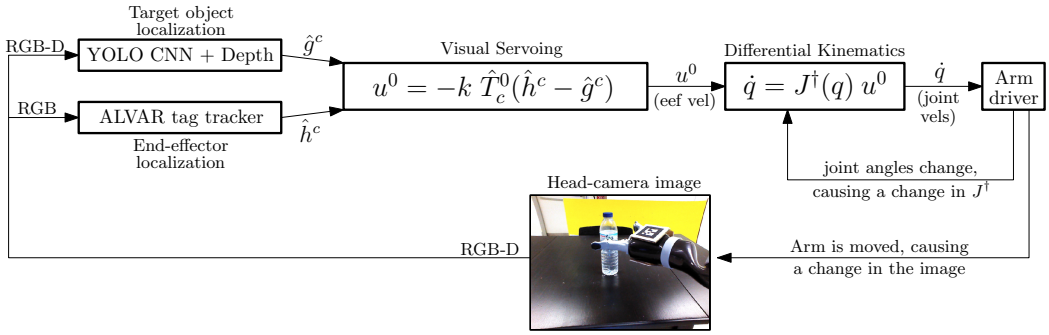

Figure 8: Diagram of the visual-servoing architecture.

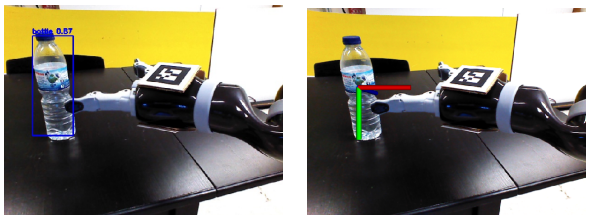

Figure 9: Target position estimation: YOLO bounding box and 3D position estimation based on the depth of the box's central region.

Figure 10: ALVAR markers used for end-effector pose estimation

and 2) relative to the main marker (ID 0), as well as the size of the markers. When any marker is visible, the ALVAR package outputs $\hat{M}_m^c$ — the pose of the main wrist marker m in camera frame. However, for visual-servoing it is useful to locate the end-effector tool (or hand) which we want to bring closer to the target object. To do this, a fixed transform t_h^m is defined from the hand to the marker frame, based on measurements. A simple frame transformation gives the hand pose: $\hat{h}_h^c = \hat{M}_m^c t_h^m$.

Pregrasp Pose Selection The purpose of this work was more on using visual-servoing to achieve a given grasp point, and less focus was put on selecting an appropriate grasp pose. However, since our system is designed to be modular and extensible, a simple grasp selection algorithm was developed to showcase the capability.

Our selection criteria is based on how high the object sits on the table. The estimated height of the object is subtracted from the table's height and compared to a threshold. If it's taller, the wrist is placed parallel to the table, as if to pick up a water bottle or coke can. Otherwise if the object is short, the robot's wrist is lifted in order to avoid colliding with the table.

These serve as *pregrasp* poses. Since the pregrasp phase cannot benefit from visual feedback (the AR tags are still not visible to the camera), the arm is placed at a small distance from the object, close enough to make the AR tags visible to the camera. For the final approach (*grasp* phase) visual-servoing starts and controls only the linear velocity of the end-effector: its orientation stays the same as in the chosen pose.

Visual Servo Control Having obtained the required variables to calculate the positioning error as the difference between the target and end-effector estimated positions, PBVS control can start: we use the proportional control law given in (8) to obtain the linear velocity required for the end-effector tool to approach the target object at every timestep:

$$u^0 = -k\,\hat{T}_c^0(\hat{h}^c - \hat{g}^c) \tag{11}$$

Controlling the arm using this law naturally makes the error $(\hat{h}^c - \hat{g}^c)$ diminish through time. For the control loop to stop, we set a *stopping distance* parameter s: the acceptable error value at which visual-servoing stops, so that the object is grasped by the gripper. s is easily configurable, and we use $s = 0.01$ (1cm) in experiments.

Although the arm's driver supports Cartesian velocity control of the end-effector (sending u^0 directly to the controller), this mode of operation has some flaws, e.g. not being able to activate the mode for certain arm positions. For this reason we decided to calculate the required joint velocities $\dot{q}$ and send those to the driver's joint velocity control mode, which works well. Doing this means solving the *inverse differential kinematics* problem described in section 3.1. The Jacobian matrix $J(q)$ is calculated at every timestep, based on the arm's kinematic structure and current joint angles q published by the arm driver. Singular value decompo-

sition is performed to obtain the Jacobian pseudo-inverse $J^{\dagger}(q)$:

$$J = UDV^T, J^{\dagger} = VD^{\dagger}U^T \tag{12}$$

The joint velocities can now be obtained, replacing $J^{-1}(q)$ with $J^{\dagger}(q)$ in (4):

$$\dot{q} = J^{\dagger}(q)\ u^0 \tag{13}$$

We make use of the Eigen C++ library to perform numerical operations efficiently.

Complete Pipeline The developed visual-servoing control scheme is capable of making the end-effector approach the target object, but by itself does not constitute a grasping system. A complete pipeline was developed making use of the ROS framework, primarily the MoveIt! package. The pipeline is implemented as a state machine:

Function *GraspObject(object_type)*:

```
start_localizer(object_type)
sweep_head()
turn_head_to_object(object_tf)
pose ← select_pregrasp(object_tf)
pregrasp(pose)
visual_servo()
close_gripper()
lift_and_lower_eef()
open_gripper()
go_to_pose('mbot_resting')
```

Algorithm 1: Object grasping pipeline

The system is started by running a ROS launch file with configuration parameters, and a grasp can be triggered by calling a ROS service, passing the YOLO class of the object as the input.

When the service is called, the object localizer node is started, which loads and runs the YOLO network (using the robot's GPU for acceleration) and the depth estimator for the given object type. This node publishes the object's position ($object\_tf$) as a ROS tf transform.

The head is sweeped left and right to build an octomap of the scene for obstacle avoidance. The head is then pointed to the object, and the pregrasp pose is selected based on the object's height. A motion plan to the selected pose is obtained through the MoveIt! planner and executed by without visual feedback. With the end-effector in frame, visual-servoing starts for the final approach, until the end-effector is at the stopping distance, at which point the gripper is closed, and the object is lifted. Finally the arm returns to the rest position. Fig. 11 shows the pipeline's steps, and a video showing several grasps can be seen at `https://youtu.be/CZaLNTZ_ITU`.

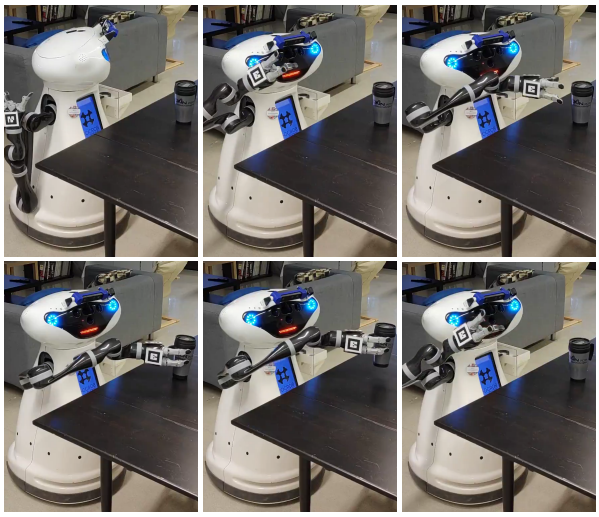

Figure 11: Steps of the grasping pipeline

5. Results

5.1. Grasping Benchmark

Setup To test our system we use the "Object Grasping and Manipulation" functionality benchmark [1] from European Robotics League. Designed to be used in ERL Consumer competitions for domestic robots, this benchmark focuses solely on evaluating grasping and manipulation capabilities. By using ERL's referee, scoring and benchmarking box (RSBB), tests can be done in a semi-autonomous way and result logs with performance information are automatically saved.

To automatically score the benchmark, the RSBB communicates with a motion capture (MoCap) system: retroreflective markers are attached to the objects so that MoCap cameras can detect their pose. We use the "successful grasp" evaluation criteria, which is positive if the object is lifted more than 2cm.

A 25x20cm rectangular region of a dining table was divided into 9 positions (3x3). Five household items were chosen and placed at each position, for a total of 45 test grasps. The total percentage of successful grasps was measured, as well as the success rate per-object and per-position. We also captured internal data from the robot to better analyze and debug the system. During each grasp we recorded the pose of the end-effector and target object, and during visual-servoing we recorded its error signal.

Benchmark Results The system achieved an overall grasping success of 82.2% — 37 successful grasps out of 45. Tables 1 and 2 show the grasp success per-position and per-object, respectively. Failures close to the robot happened due to the robot not being able to see the end-effector marker after pregrasping, while failures far from the robot were caused due to the arm reaching a kinematic limit

y(m) / x(m)	-0.125	0.00	0.125
0.20	**60%**	**60%**	**60%**
0.10	**100%**	**100%**	**100%**
0.00	**80%**	**80%**	**100%**

Table 1: Grasp performance by table position. Aggregate of all 5 objects.

Bottle	Tall mug	Large coffee	Espresso	Orange
88.8%	**88.8%**	**100%**	**66.6%**	**66.6%**

Table 2: Grasp performance by object

during visual-servoing.

The error plots (Fig. 12) show the expected system behavior: A smooth motion is applied to the arm while pregrasping, and when visual-servoing starts the error follows an asymptotic function converging to 0, stopping at the desired distance of 1cm.

5.2. Error Tolerance Tests

To test our system's ability to handle calibration errors, we added an erroneous translation to the arm in relation to MBot's body, moving it 8cm to the right in the URDF measurements. We then tested two versions of the pipeline: a modified version which does not use visual feedback, making the final approach to the object only based on joint sensors and measurements, and our version which uses visual-servoing for the final approach. Three objects were placed in the same table position for a fair comparison between the versions.

As expected, the version with no visual feedback

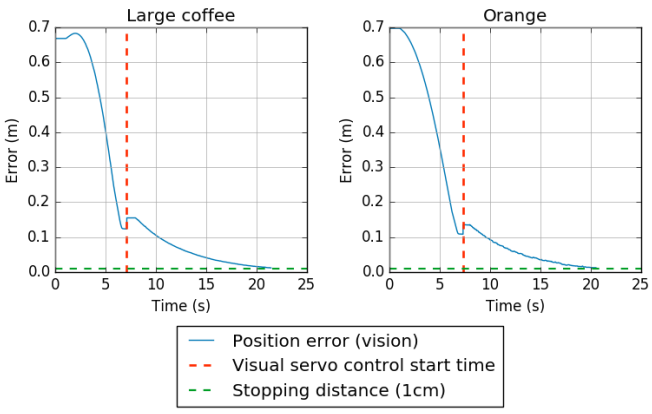

Figure 12: Error plots for two grasps

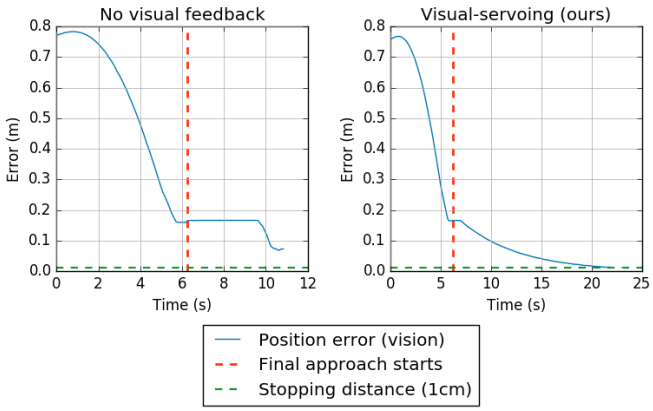

Figure 13: Comparison of grasping with and without visual feedback, with a wrong arm measurement

places the hand incorrectly and fails to grasp the objects. When using our pipeline, while the pregrasp pose also places the end-effector to the left, visual-servoing then activates and starts correcting the error, guiding the arm to a successful grasp on all three objects. Fig. 13 shows error plots for both versions.

6. Conclusions

The proposed system shows success in grasping several household items in different positions. Using visual-servoing makes it tolerate calibration errors in the camera and the arm kinematics, allowing for a precise final grasp position.

Our target localization system using supervised learning proves to be useful for SocRob's purposes: many household items are detectable by pretrained models, and a specific detector can be trained to make the system able to grasp other objects. Additionally our grasping pipeline is easily usable as a part of more involved domestic and assistive robot tasks.

As future work, the grasping pipeline can be extended by adding navigation: after identifying the target object's position, the robot can move to an advantageous position for grasping. Grasp feasibility checks could be added to prevent visual-servoing from reaching kinematic limits. This work could also be augmented by adding a more robust grasp pose selection algorithm, using available data from the depth sensor, or even several viewpoints of the object, obtained by moving MBot's base, and using 3D stitching for reconstruction

References

[1] M. Basiri, E. Piazza, M. Matteucci, and P. Lima. Benchmarking functionalities of domestic service robots through scientific competitions. *KI - Künstliche Intelligenz*, 33, 09 2019.

[2] Q. Bateux, E. Marchand, J. Leitner, F. Chaumette, and P. Corke. Training deep neural networks for visual servoing. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3307–3314, 2018.

[3] F. Chaumette and S. Hutchinson. Visual servo control. I. Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, 2006.

[4] C. Collewet and E. Marchand. Photometric visual servoing. *Robotics, IEEE Transactions on*, 27:828 – 834, 09 2011.

[5] H. Hashimoto, T. Kubota, Wai-chau Lo, and F. Harashima. A control scheme of visual servo control of robotic manipulators using artificial neural network. In *Proceedings. ICCON IEEE International Conference on Control and Applications*, pages 68–69, 1989.

[6] S. Haykin. *Neural Networks and Learning Machines*. Pearson Education, third edition, 2008.

[7] J. Hill. Real time control of a robot with a mobile camera. 1979.

[8] S. Hutchinson, G. D. Hager, and P. I. Corke. A tutorial on visual servo control. *IEEE Transactions on Robotics and Automation*, 12(5):651–670, 1996.

[9] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. volume 87 of *Proceedings of Machine Learning Research*, pages 651–673. PMLR, 29–31 Oct 2018.

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

[11] Y. Lecun. A theoretical framework for backpropagation. In D. Touretzky, G. Hinton, and T. Sejnowski, editors, *Proceedings of the 1988 Connectionist Models Summer School, CMU, Pittsburg, PA*, pages 21–28. Morgan Kaufmann, 1988.

[12] I. Lenz, H. Lee, and A. Saxena. Deep learning for detecting robotic grasps. *1st International Conference on Learning Representations, ICLR 2013 - Workshop Track Proceedings*, 2013.

[13] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17:1–40, 2016.

[14] T.-Y. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. *ArXiv*, abs/1405.0312, 2014.

[15] V. Lippiello, B. Siciliano, and L. Villani. Eye-in-hand/eye-to-hand multi-camera visual servoing. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 5354–5359, 2005.

[16] MathWorks. Deep learning - convolutional neural network, 2020.

[17] J. McCarthy, L. Earnest, D. R. Reddy, and P. J. Vicens. A computer with hands, eyes, and ears. In *Proceedings of the December 9-11, 1968, fall joint computer conference, part I*, pages 329–338, 1968.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[19] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *ArXiv*, abs/1804.02767, 2018.

[20] A. C. Sanderson and L. E. Weiss. *Adaptive Visual Servo Control of Robots*, pages 107–116. Springer Berlin Heidelberg, Berlin, Heidelberg, 1983.

[21] S. Schaal. Jacobian methods for inverse kinematics and planning, 2014.

[22] Y. Shirai and H. Inoue. Guiding a robot by visual feedback in assembling tasks. *Pattern Recognition*, 5(2):99 – 108, 1973.

[23] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Robotics: Modelling, Planning and Control*. Springer Publishing Company, Incorporated, 1st edition, 2008.

[24] B. Singh, N. Sellappan, and P. Kumaradhas. Evolution of industrial robots and their applications. 2013.

[25] M. Zhu, K. Derpanis, Y. Yang, S. Brahmbhatt, M. Zhang, C. J. Phillips, M. Lecce, and K. Daniilidis. Single image 3d object detection and pose estimation for grasping. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3936–3943, 2014.