Classical and Deep Learning based Visual Servoing Systems

A Survey on State of the Art

Machkour, Zakariae; Ortiz Arroyo, Daniel; Durdevic, Petar

Published in:
Journal of Intelligent and Robotic Systems

DOI (link to publication from Publisher):
10.1007/s10846-021-01540-w

Publication date:
2022

Document Version
Accepted author manuscript, peer reviewed version

Noname manuscript No.
(will be inserted by the editor)

Classical and Deep Learning based Visual Servoing Systems: A Survey on State of the Art

Zakariae Machkour · Daniel Ortiz-Arroyo · Petar Durdevic Løhndorf

Received: date / Accepted: date

Abstract Computer vision, together with bayesian estimation algorithms, sensors and actuators are used in robotics to solve a variety of critical tasks such as localization, obstacle avoidance, and navigation. Visual servoing uses computer vision algorithms to guide robot movements. Classical approaches in visual servoing systems relied on extracting features from images to control robot movements. Now, state of the art computer vision systems use deep neural networks for object recognition, detection, segmentation, and tracking. These networks and specialized controllers play a predominant role in the design and implementation of modern visual servoing systems due to their accuracy, flexibility, and adaptability. Recent research in direct systems for visual servoing has created robotic systems that rely only on the information extracted from images. Furthermore, end-to-end systems eliminate entirely the controller by learning the control laws during training.

This paper presents a comprehensive survey on the state of the art in visual servoing systems, discussing the latest classical methods not included in other surveys but emphasizing the new approaches based on deep neural networks and their applications within robotics.

Keywords Computer vision · Deep Neural Networks · Visual servoing · Robotics

1 Introduction

Robotic systems have an increasingly important role in areas such as manufacturing, inspection, surveillance, and health care, among others. Autonomous robots perform a given task interacting with a static or dynamic environment using their sensors and actuators [15]. An example of the earliest industrial robots is a robotic arm designed in the 1930s by Willard V. Pollard [144]. Later, in 1961, the first programmable robot called Unimate [22] was created by General Motors for the task of moving hot metal pieces and placing them in cooling liquid.

Z. Machkour · D. Ortiz-Arroyo · P. D. Løhndorf
Niels Bohrs Vej 8, 6700 Esbjerg, Denmark
E-mail: zma@et.aau.dk, doa@et.aau.dk, pdl@et.aau.dk

Recent advances in sensors, actuators, control, operating systems, and communications have given robots increased capabilities to interact with their surrounding environment. This has been facilitated by the use of standard meta operating systems such as the Robotic Operating System (ROS) that enables the integration of sensors, actuators, and a variety of algorithms for localization, path planning, and navigation. ROS has a message-passing distributed architecture that uses the publisher-subscriber paradigm and several specialized communication protocols.

The sensors used in modern robotic systems include a large variety of types, from video or infrared cameras to Inertial Measurement Units (IMU), LiDARs, ultrasonic sensors, global position systems (GPS), Real-time kinematic (RTK) positioning, and radars, among others [24, 27, 30]. However, in spite of their wide use, these sensors have limitations. For instance, GPS works for localization and navigation in outdoor environments that do not require high precision but cannot be used to navigate indoors, or on cluttered urban areas [102]. RTK is more accurate than GPS but is more expensive and requires a special setup. Ultrasonic sensors have a limited range of operation. LiDARs are ineffective in heavy rain or sun reflections but have issues detecting narrow surfaces such as wind turbine blade edges [57]. IMUs with sensor fusion provide accurate data about the orientation and heading of a robot, but suffer from drifting [23, 199].

Video cameras provide rich information about the environment and are one of the main sensors used in robotics. Cameras with high resolution are also cheap[1], have small size, and consume little power. These characteristics make them ideal for resource-constrained robots that work on a relatively narrow indoor environment or in large open spaces. However, video cameras have also limitations. For instance, they do not work well in poor lighting conditions, with motion blur, or with specular reflections that produce noisy images [89, 104]. Due to these issues, cheap video cameras are commonly used in robotics with other sensors, in addition to Kalman filters and sensor fusion techniques [70, 183] .

Regarding the software and algorithmic aspects in robotics, probabilistic and machine learning algorithms are now used extensively, due to their ability to handle uncertainty and adapt to unknown environments [66, 186]. In particular, deep learning has been one of the most successful techniques. Deep learning comprises a variety of neural network architectures designed to solve problems in computer vision, speech recognition, and time series prediction. Deep Neural Networks (DNNs) [66], an extension to Shallow Neural Networks (SNN), are designed with a hierarchical structure consisting of multiple layers with thousands of connections and weights. DNNs' design and structure emulate how the human vision system is organized [61].

The recent development of new DNN architectures, together with specialized controllers, sensors, actuators and a wide variety of algorithms available for localization, path planning, obstacle avoidance, have facilitated the creation of autonomous robots, capable of performing increasingly complex tasks. One of the main applications of DNNs in robotics is in designing its visual servoing system, a mechanism that allows a robot to move and positioning itself at a target location using the video information from a camera. Previous surveys on visual servoing systems such as ([32, 33, 103, 176]) described research work on the use of classical computer vision techniques in single or dual robotic manipulators [176]. More re-

[1] e.g. 5M High-Resolution Camera costs 65 dollars[1]

cently, other surveys on unmanned aerial vehicles (UAV) [94] and computer vision in general [8] have been published.

One of the goals of this paper is to fill the gap left by the latest surveys on classical approaches to design visual servoing systems. To accomplish this goal, this survey includes the most recent research work on a variety of robotic platforms and application domains, but emphasizes the more recent DNN-based approaches to visual servoing and its applications. Lastly, we extend the taxonomy of classical visual servoing systems to include the new approaches, specifically end-to-end systems, direct visual servoing, and object tracking visual servoing, describing how these systems are included in the general definition of visual servoing systems.

The paper is organized as follows: Section 2 provides a definition of visual servoing and describes classical systems. Section 3 introduces an extension to the classical systems taxonomy of visual servoing systems. Section 4 describes examples of recent systems that use classical visual servoing techniques grouped by their type and robotic platform, additionally to discussing some of the limitations of the classical methods. Section 5 describes neural network based visual servoing systems, including the use of DNNs and Convolutional Neural Networks (CNN), and provides a description of recent examples of the modern architectures used in CNNs. Section 6 shows how DNNs are integrated into visual servoing systems and provides examples of both End-to-End and Direct visual servoing systems. Lastly, section 7 describes the state of the art in DNN-based visual servoing grouped by system type and robotic platform, and section 8 presents a summary and discusses current developments and issues in visual servoing systems.

2 Visual Servoing

The concept of visual servoing was introduced in the control of robotic arms [84]. However, this concept can be also applied in other types of robotic systems such as ground robots, unmanned aerial and underwater vehicles that use computer vision to guide robot movements. In this section, we introduce the concept of visual servoing, its definition and the main categories of systems.

2.1 Visual Servoing Definition

Visual servoing can be defined in terms of the solution to an optimization problem (e.g., [19, 32]), where the goal is to minimize an error function expressed by the following equation:

$$\hat{r} = \arg\min_{r} e(\boldsymbol{r}, \boldsymbol{r}^*) \tag{1}$$

Where $\hat{r}$ is the pose of the camera reached at the end of the optimization process, e denotes the error function that measures the positioning error between the current pose r of the camera and the desired pose r^*, ideally at the end the visual servoing system should make $\hat{r} = r^*$.

The error function can be defined as [21] :

$$\boldsymbol{e}(r) = \|\boldsymbol{s}(\boldsymbol{r}) - \boldsymbol{s}^*\| \tag{2}$$

as the Euclidean norm of the distance between $\boldsymbol{s}$ a vector of k visual features (e.g. the image coordinates of points of interest or the image coordinates of the center of an object) extracted from the image at pose $\boldsymbol{r}$, and the set of the target features s^* extracted at pose $\boldsymbol{r}^*$.

The vector $\boldsymbol{s}^*$ is considered constant in case of a motionless target, but if the target is moving the visual servoing system is also tracking the object.

It can be also said that in visual servoing, the pose (i.e. position and orientation) of a robotic platform, relative to a target, is controlled by using the visual features extracted from images [94].

Visual servoing systems differ in the way vector $\boldsymbol{s}$ is designed and computed. In the classical approaches, the features included in $\boldsymbol{s}$ are taken directly from the image or from the 3D parameters calculated using image measurements.

2.2 Classical Visual Servoing Systems

Classical visual servoing systems differ in the type of control architecture, in the number of video cameras used (mono, stereo, or multi-camera), and also in how they are placed in the system. For instance, in eye-in-hand the camera is rigidly mounted on the robotic manipulator, but in eye-to-hand, the camera that observes the robot is mounted on the workspace [59, 84].

The classical visual servoing approaches are classified, depending on the definition of the error function adopted to regulate the system (known as task function) [58, 166], into the following categories:

1. Image-based Visual Servoing (IBVS). In IBVS, the error is computed from a set of visual features extracted from 2D image space and the vector $\boldsymbol{s}$ may be, for example, point coordinates, a set of straight lines, or the contours of an object. This method requires camera intrinsic parameters that allow converting from image measurements expressed in pixels to features and accurate feature matching.
2. Position-Based Visual Servoing (PBVS). In PBVS, the pose of the camera within a reference coordinate system is used to compute $\boldsymbol{s}$ as a set of 3D parameters. The method requires a 3D model of the scene and camera intrinsic parameters to be known a prori. The error is computed in the Cartesian task space from a set of 3D parameters.
3. Hybrid systems. In Hybrid visual servoing, the error function is a combination of Cartesian and image measurements or features. This method is based on the estimation of the camera displacement (the rotation and the scaled translation of the camera) between the current and desired views of an object. As an example of the definition of the error function $\boldsymbol{e}$, in [129] $\boldsymbol{s}$ contains the coordinates of an image point, and the logarithm of its depth.

Systems that belong to the classes described in the previous taxonomy, have been the subject of various tutorials in [32, 33, 84] and the surveys in [103] for single arm manipulators, and [176] on dual arm manipulators. The next section describes an extension to the classical visual servoing taxonomy to include the new types of visual servoing systems that have been proposed in recent years.

3 A Taxonomy for Visual Servoing Systems

New approaches in visual servoing systems have been proposed in the literature in recent years. These include approaches that rely only on the information contained in images and systems that do not require a controller. Additionally, target tracking methods, in which the controller aims at minimizing the tracking errors due to the target motion, have received considerable attention. To include these new type of systems in a unified taxonomy, we added the following classes to the classical methods taxonomy:

1. Direct Visual Servoing. In direct visual servoing, the full image information is used to solve a positioning task. The method does not use geometrical features and the control laws are obtained directly by measuring the similarity between two images. Hence, this method requires a robust similarity evaluation, together with efficient optimization algorithms [18]. In this type of VS system, the image is considered as a whole, hence the vector s becomes the image itself [44]: $\boldsymbol{s}(r) = I(r)$. We substitute the value of $\boldsymbol{s}$ in (2), the optimization process then can be expressed as :

$$\hat{r} = \arg\min_{r} \|\boldsymbol{I}(\boldsymbol{r}) - \boldsymbol{I}^*\| \tag{3}$$

   where $I(r)$ and I^* are respectively the image captured at the position r and the reference image.
2. End-to-End systems. Unlike classical visual servoing systems, which may be described as having an explicit controller module, this class includes systems that do not have a controller. End-to-end systems may use supervised or unsupervised machine learning approaches based on DNNs to learn the control laws at training time [101]. The error function used in these systems may be the mean square error between the current steering angle and the predicted one as is described in [101].
3. Fixed and Moving Target Tracking Visual Servoing. In the fixed target tracking case, it is assumed that the camera is moving while tracking a fixed object. In the second case, the dynamics of the target should be estimated to keep track of the object. In this case, the vector $\boldsymbol{s}^*$ contains a dynamic set of features from the target object.
4. Single and Multiple Target Tracking Visual Servoing. In single target tracking there is only one moving target of a given class of target objects. Contrarily, in multiple target tracking, a limited number of moving targets that belong to the same class or that share the same features are tracked. In this case the vector $\boldsymbol{s}^*$ contains a dynamic set of features from the target objects. The error function may be the mean position of the tracked objects with respect to the center of the image.
5. Explicit and Implicit Controller Visual Servoing. The explicit controller class includes the classical and DNN based visual servoing systems that have a feedback control loop containing a controller module. Implicit controller systems do not have a controller, but the control laws are learned and embedded into the weights and connections of the DNN.

Direct and end-to-end systems share some similarities as they are both capable of learning control laws i.e. they both may be classified as of the implicit controller

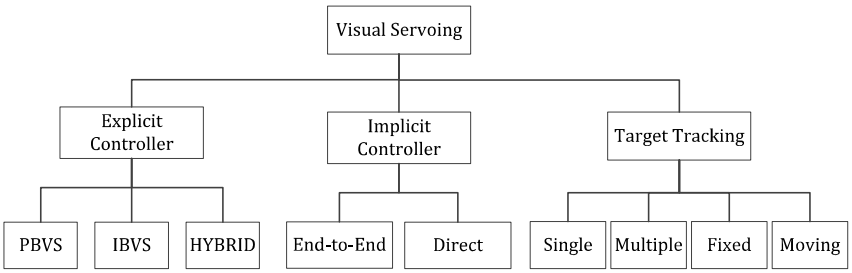

Fig. 1 Taxonomy of Visual Servoing systems, based on their type and applications.

type. However, direct visual servoing systems use similarity measures and particle filters or CNNs to learn the control laws[18]. Moreover, direct servoing is mainly used in manipulators. Contrarily, end-to-end systems rely on training a DNN to learn the control laws using supervised or unsupervised methods but are also used in a wide variety of robotic systems. In some end-to-end systems, the controller is not completely eliminated, but the control commands to determine the position of a UAV are estimated by a DNN from images taken by several cameras and sent to a controller whose only purpose is to keep a constant speed as is suggested in [169].

Distinctly, visual servoing systems may belong to one or more of the previous categories. For instance, a visual servoing system with an explicit or implicit controller may be used to navigate or keep track of a single target object that may be at a fixed position or moving.

Figure 1 shows the whole taxonomy of the different types of visual servoing systems, categorized according to the type of controller used, its purpose, and the type of target object being tracked. Table 1 includes selected examples of the types of visual servoing systems, according to its controller type (explicit-Control or implicit-control, direct or end-to-end), its system type (linear or non-linear), robotic systems type (arm, UAV, wheeled-robot or underwater vehicle), and camera type (mono or stereo-camera).

Lastly, Table 2, contains examples of visual servoing systems grouped according to the classes in the new taxonomy, i.e. STTVS (Single-Target Tracking), MTTVS (Multi-Target Tracking), FXTTVS (Fixed-Target Tracking), and MBTTVS (Mobile-Target Tracking) visual servoing systems.

Table 1 The four main classes in Explicit and Implicit Controller-based Visual Servoing

References	Explicit Controller			ICS	Platform	Camera	Controller
	PBVS	IBVS	HYVS				
[74]	✓				Robotic arm	Mono Camera CCD	Linearized Controller
[73]	✓				Robotic arm	-	Non-linear Controller
[197]	✓				Robotic arm	-	PD Controller
[120]	✓				Robotic arm	-	-
[171]	✓				Robotic arm	-	-
[87, 165]	✓				Robotic arm	Mono Camera CCD	-
[113]	✓				Robotic arm	Stereo camera	-
[10, 13, 29, 71, 160, 187, 191]	✓				-	-	-
[194]		✓			Robotic arm	Mono Camera CCD	Adaptive Controller
[195]		✓			Robotic arm	Endoscope Camera CCD	Adaptive Controller
[206]		✓			Mobile Dual-Arm	Stereo and mono Cameras	-
[119]		✓			Dual-Arm Areal Manipulator	Mono Camera	Adaptive Controller
[193]		✓			Wheeled robot	Mono Camera	Adaptive Controller
[188]		✓			Wheeled robot	Omnidirectional Camera	Switching Controller
[132]		✓			Wheeled robot	Catadioptric Camera	-
[117]		✓			Wheeled robot	Ceiling-mounted camera	Adaptive Controller
[96]		✓			Wheeled robot	Mono Camera	Model Predictive Controller
[152]		✓			Wheeled robot	Mono Camera Adaptive controller	
[133]		✓			UAV	Mono Camera CMOS	Non-linear Controller
[14]		✓			UAV	Multi Cameras	PID Controller
[178]		✓			UAV	Mono Camera (GoPro)	-
[149]		✓			UAV	Mono Camera	PD Controller
[184]		✓			UAV	Mono Camera	Geometric Visual Controller
[210]		✓			UAV	Mono Camera	Non-linear Controller
[43, 148, 153]			✓		Robotic arm	Mono	-
[5]			✓		-	Catadioptric camera	Homography-based Controller
[123]			✓		Underwater Vehicle	-	-
[91]				DIRECT	UAV	Stereo camera	-
[18, 130, 131, 145]				DIRECT	Robotic arm	-	-
[140]				ETE	Wheeled robot	Stereo	-
[26]				ETE	Wheeled robot	Multi cameras	-
[36]				ETE	Wheeled robot	-	-
[169]				ETE	UAV	Mono	-
[25]				ETE	UAV	Multi cameras	-

HYVS: Hybrid or 2 1/2 D VS
ICS: Implicit Controller Systems
ETE: End-To-End systems

Table 2 Visual Servoing Taxonomy based on Target Tacking scenarios.

References	STTVS	MTTVS	FXTTVS	MBTTVS
[73, 74, 120, 133, 149, 194]	✓			✓
[14, 134, 141]	✓		✓	
[17, 164, 205]		✓		

STTVS: Single-Target Tracking Visual Servoing
MTTVS: Multi-Target Tracking Visual Servoing
FXTTVS: Fixed-Target Tracking Visual Servoing
MBTTVS: Mobile-Target Tracking Visual Servoing

4 Classical Visual Servoing Techniques

In this section, we describe each of the classical visual servoing techniques. Then, these classical techniques are grouped by the type of the robotic system where they are applied and examples of most recent research work are described. The section concludes by discussing some of the issues and limitations of the classical techniques.

4.1 Position Based Visual Servoing

PBVS systems require a priori knowledge of the camera calibration and a geometric model of the target, to estimate the target's position with respect to the camera [129]. Examples of PVBS systems are discussed in [73, 74], where an observer-scheme based visual servoing system was introduced for application in manipulators. Another example is discussed in [10] where a camera captures videos and images with some noise, and an $\alpha - \beta - \gamma$ filter [174] is used to produce values closer to the true spatial measurements of the robot and target.

Other PVBS systems like [35] use Extended Kalman filters (EKF) for nonlinear state estimation to reduce errors in the sensor's measurements. The usage of extended Kalman filtering techniques is also used to improve the precision of the estimation for end-effectors as is described in [120, 171, 197]. Some other examples using Unscented Kalman Filter techniques to estimate the pose, velocity, and acceleration of a target object are discussed in [87, 165]. Particle-based filtering methods are applied in pose estimation, via a 3D model-based stereo-vision, in [113, 191].

Another approach in PVBS systems is to use stereo-vision as is described in Anderson [13], Allen et al. [10], Bukowski et al. [29], and Rizzi et al. [160]. In stereo vision, the images from two cameras are analyzed to note their differences and calculate depth using disparity. An example of this approach is [71], where a PBVS system based on stereo vision techniques is capable of estimating the 3D coordinates of any point observed in two views of the same scene applying a triangulation process.

PVBS systems should solve the problem of keeping the object frame origin within the field of vision. One possible solution for this is a method proposed in [187]. However, to use this method, the depth cue of the object origin was required to be known a priori, at both the initial and the desired poses of the camera.

The majority of the PBVS systems shown previously rely on the usage of different filters like Kalman, EKF, Unscented Kalman, or particle filter in order to estimate the pose of the target and to deal with the different noise sources found in real environments.

4.2 Image Based Visual Servoing

Unlike PBVS, IBVS uses only 2D image features and does not require a 3D pose estimation of the target object. As a result, the calculations that provide the robot control signals can be performed very quickly [176]. The IBVS method has been widely used for positioning manipulators [194], [83], [195], Unmanned Aerial Vehicles (UAVs) [133], and the formation control of mobile robots [193].

One of the problems of IBVS is that it requires the use of visual marks on the observed object to identify its geometric features. To avoid this problem, in [45], a motion-based technique is used as an input to the control system, where motion in the image can be estimated without any a priori knowledge of the observed scene. Another issue of IBVS is the estimation of the feature depth. To tackle this issue, an extended 2D visual servoing scheme for depth estimation, was developed in [48, 170].

A different approach to design IBVS systems is discussed in [38], where a robot is moved from its initial location to the goal location, using a path planning algorithm. An optimal, collision-free path is chosen to avoid obstacles and to maintain the safety of the robot, using constraints such as visibility, workspace, and joint constraints. At the same time, the cost function represented by the trajectory length is minimized. In this regard, a recent survey on 3D path planning algorithms that can be used for this type of IBVS systems is presented in [202].

All previous IBVS systems were designed for a single robotic arm, but IBVS for dual-arm manipulators have been studied in [119, 206]. Dual-arm manipulators have the additional problem of requiring algorithms for coordinating their movements.

4.3 Hybrid Visual Servoing

The hybrid visual servoing approach, known also as 2D 1/2 visual servoing, was first introduced in [129]. This approach consists of using information available directly at the image level jointly with the information reconstructed from two views of a rigid object (i.e., the displacement of the camera), decoupling rotations and translations. In an example of hybrid servoing, the work in [43] introduces an approach where the servoing task is split into main and secondary tasks. The main task maintains the image centered to keep the features of interest within the field of view. The secondary task scans other positions while keeping the object centered in the image to bring the camera to the desired pose. This technique needs a depth estimate obtained from an offline procedure. The paper addresses two examples by which depth estimates are extracted from the robot's odometry, assuming that all features are on a plane. The secondary task uses the notion of parallax, where the features that are tracked (typically points) are chosen by an initialization procedure applied on the first frame.

The work in [123] presents another example of hybrid visual servoing, where a method for station-keeping an underwater vehicle is described, using a single onboard video camera. The proposed method takes into account the restrictive controllability of the vehicle due to its thruster's configuration.

Using a hardware oriented approach, [5] describes how to use an omnidirectional camera in a 2.5D visual servoing scheme. This camera solves the problem of keeping the target object within the field of view.

In [153], a hybrid visual servoing system estimates the error function in the image space as in IBVS. This is combined with the depth information of the image features obtained from the point cloud using Kinect sensor and given directly as input to the interaction matrix or image Jacobian that relates image feature velocities with the camera velocity. The method was implemented in a Gazebo simulator with the 6 DOF UR5 mounted Kinect sensor at the end-effector link.

Another recent example of a hybrid visual servoing control in the medical field is described in [148], where a robotic system called deTattoo consisting of a robotic arm equipped with an RGB-D camera and a laser pointer is used for tattoo removal. To perform this operation, a hybrid visual control was defined. The combination of both the 3D reconstruction of the selected tattoo image provided by the RGB-D camera, along with the updated position of each point representing the tattoo, was used in the proposed hybrid visual servoing control.

4.4 Target Tracking Systems

The main idea behind target tracking systems is to keep the target objects at a certain position or distance from the camera and within its field of view.

In single target tracking, the camera pose is controlled so that the estimated center of the object appears at the center of the image. In the case of tracking fixed objects, the camera is moved to keep it at a certain fixed distance from the target object. In multi-target tracking systems, the camera pose is controlled so that a limited number of moving target objects is tracked. Some of the criteria used to control the pose of the camera in multi tracking systems is to keep the mean position of all tracked objects in the image center the whole time or to keep all tracked objects within the field of view of the camera.

In the rest of this section, we describe the different types of target tracking visual servoing systems for a variety of applications.

4.4.1 Single and Moving Target Tracking Systems

An example of a system that is both single target tracking visual servoing systems (STTVS) and mobile target tracking visual servoing (MBTTVS) system is [73], where a 6-DOF hand-eye configuration is guided using an IBVS approach to follow a circular quasi-planar object which moves in front of a homogeneous background. The authors use an observer for estimating the object velocity making the strong assumption that the acceleration of the target object is zero, and propose a nonlinear controller with an observer. Based on the object motion model, the observer is formulated as a nonlinear adaptive identification problem where unknown parameters such as direction, position, center of circle, velocity are estimated.

Since the computational workload of the controller was too large, it was not suitable for robots that have more than three degrees-of-freedom. Later, Hashimoto et al. [74] proposed a linearized version of the observer by neglecting the nonlinear dynamics of the robot.

Another example of STTVS and MBTTS system with eye-in-hand camera and a nonlinear observer approach was designed in [194]. The goal was to lock the projection of the moving target object in 3-D space, on the image plane to a particular position (e.g., the center), by controlling robot motion. The proposed nonlinear observer estimates the 3-D motion of the object online, and it has been implemented and tested on a 3-DOF robot manipulator.

A similar work reported in [120] presents an algorithm for the visual motion estimation of the pose of a moving object of known geometry. Based on the prediction capacity of the Extended Kalman Filter (EKF), this adaptive approach can automatically tune the statistics of the state noise and the observation noise to realize a real-time dynamic image feature selection of the moving objects that is used for pose estimation.

Another example of a system that is both STTVS and MBTTVS system is [133], where a nonlinear controller that stabilizes UAVs in GPS-denied environments with respect to visual targets is proposed. In this work, the visual feedback information used for the control law was expressed in the spherical coordinate system of only 4 points that characterizes the target object. These spherical visual features were used to estimate the translational velocity and the autonomous positioning of the quadrotor, without the use of an IMU. The system has also the capacity to track moving targets whose speed can exceed 2 m/s.

Lastly, the work in [149] performs object tracking and following. The system was applied on a flying robot capable of following a variety of static and moving targets at varying distances from 1-2 m to 10-15 m. The work uses a C++ open source implementation of the OpenTLD tracker (TLD for tracking-learning-detection [93]). This tracker does not require any previous knowledge of the tracked object, and it can robustly track objects on the drone's video stream by outputting a bounding box (location, height, and width) around the tracked object along with a confidence ratio.

4.4.2 Single and Fixed Target Tracking Systems

The work in [14] is an example of a Single and Fixed-Target Tracking Visual Servoing (STTVS-FXTTVS) system. In this paper, the authors present two visual servoing approaches for power line inspection with UAVs. In both approaches, the UAV is kept within a close and safe distance to the power lines while the inspection is performed. The first approach combines IBVS with a Linear Quadratic Servo (LQS) technique to improve the control design of the UAV. The LQS approach takes into account the output errors to add an integral action to the controller in order to stabilize the quadrotor. While in the second approach, the control problem was solved by implementing the 2-1/2-D hybrid visual servoing.

Another example of the STTVS-FXTTVS system is the work described in [134] where a vision-based feature tracker was designed and implemented for an autonomous helicopter. A visual control loop estimates both the position and velocity of a set of features with respect to the helicopter. In real-time, the helicopter is autonomously guided to track these features in the fixed target (in this case,

windows in an urban environment). The system tracks only 4 points of the target, using a visual processing rate of 20 fps.

Lastly, a real-time pose tracking system for Autonomous Underwater Vehicle (AUV) is described in [141]. The proposed PBVS method employs a 3D marker as a fixed object to be tracked and used for the pose estimation of the underwater vehicle equipped with a dual-eyes camera. In this system, a 1-step Genetic Algorithm (GA) is used to estimate in real-time the relative pose between the designed 3D marker and the vehicle. To test the algorithm, a Remotely Operated Vehicle (ROV) was used that can recognize the target, estimate the relative pose of the vehicle with respect to the fixed target and control the vehicle to bring it in the desired pose.

4.4.3 Multi-Target Tracking Systems

The authors [205] controlled the pose of a camera based on the color of the tracked objects as well as the mean position of the targets. An IBVS scheme was used to perform multiple target tracking and control, based on the color of each target. The results show that the mean position of targets was maintained in the image center during the whole time of the UAV mission, and at the same time, two tracked objects were maintained within the camera's FOV (Field Of View). Through the majority of the time, the mean error was not greater than 10 pixels.

The work reported in [164] is another example of an MTTVS system developed for real-time detection and tracking of multiple objects in uncontrolled environments. The multi-object tracking model is designed with a multithreading architecture, where every thread corresponds to a detected motion area and integrates motion and color tracking. For motion detection, the algorithm uses techniques based on adaptive thresholds and on color detection using hue segmentation (HSV color space) to solve the problem of overlapping detected motion areas.

Another example of MTTVS system is described in [17]. The work presents a method that is able to estimate the 3-D position of a time-varying number of people and simultaneously perform visual servoing. The visual servoing module requires estimating an interaction matrix that maps observed image features on to robot velocities, which in turn requires 3-D information. This is done by combining a calibrated camera pair mounted onto the robot head with a person detector. The resulting estimation of the motor velocities are then taken into account by the person tracker, which is formulated as a variational Bayesian filtering [16], that has the advantages of (i) handling a varying number of persons over time, and (ii) efficiently dealing with disappearing/reappearing persons.

4.5 Classical Visual Servoing Issues

One of the issues that classical PBVS and IBVS have is that they may suffer from convergence and stability problems, as reported in [31, 129]. Another limitation is that PBVS requires geometric information about the target object location and to keep the object's features within the field of view of the camera. Furthermore, in eye-in-hand robotic systems with 6 or more degrees of freedom, where objects are placed at static positions, IBVS systems can get trapped into local minima. This occurs due to the existence of unrealizable image motions [37, 172].

Moreover, in IBVS methods, the interaction matrix or image Jacobian, which relates the time variation of the features to the camera velocity, depends on a priori knowledge of the intrinsic camera parameters [168].

Hybrid systems improve PBVS and IBVS systems by not requiring camera calibration or a 3D target model. Additionally, they have better stability properties [129]. However, they are susceptible to image noise and require a minimum number of point correspondences (4 to 8 or more) to estimate the projecting homography matrix.

In spite of the progress made in the last decades on the classical approaches to visual servoing, the computer vision methods used by these systems suffer from a lack of adaptation to changes in the environment. Moreover, they are unable to generalize the patterns or features that are extracted from images to similar objects that may have different textures and be placed in different conditions of illumination. DNNs solve these problems by learning the patterns extracted from large training datasets of labeled images and generalizing these patterns using efficient optimization methods and loss functions. Lastly, DNNs are also capable of learning the control laws, opening the possibility for eliminating the controller in a visual servoing system. In the following sections, we discuss how visual servoing systems are applied to a variety of robotic platforms, from ground robots to UAVs and underwater vehicles.

4.6 Classical Visual Servoing by Robotic System Type

The classical visual servoing techniques can be grouped by the type of robotic system where they have been applied, regardless of its classification type. In this section, we discuss the application of classical visual servoing systems in a variety of robotic systems.

4.6.1 Ground Robots

The work in [188] is an example of the application of visual servoing and navigation systems in ground mobile robots. The paper describes a method based on the use of an omnidirectional vision sensor to stabilize a tractor capable of navigating autonomously to a target position by measuring the deviations between the vehicle's current view and a target view of landmarks within a workspace.

Another example is [132] that presents an IBVS strategy that employs the epipolar geometry defined by the current and desired camera views to drive a nonholonomic mobile robot toward a desired configuration, without any knowledge of the 3-D scene geometry.

Using a different approach, [117] proposed an adaptive image-based visual servoing approach to visually guide wheeled mobile robots with a ceiling-mounted camera. This configuration does not require that the image plane of the camera be parallel to the motion plane of the mobile robot. In a similar type of application, the work in [152] presents a visual servo tracking scheme that exploits the homography-based algorithms designed for wheeled mobile robots to track a given trajectory defined by a sequence of images previously recorded.

In a more control-centered approach, in [96] an IBVS strategy is combined with model predictive control (MPC). This is used to stabilize a physically constrained

mobile robot where the system's kinematics are transformed into a symmetric form using linear control theory. Then, an explicit exponential decaying term is used to avoid uncontrollability, and the MPC strategy is transformed iteratively into a constrained quadratic programming (QP) problem that is solved using a primal-dual neural network.

4.6.2 UAVs

UAVs have multiple applications in domains such as fire fighting, traffic surveillance, or communications [50, 185]. Another recent application of UAVs is in inspection tasks. For instance, UAVs are used in the inspection of wind turbines in [178]. This system estimates the relative position and distance between the UAV and the wind turbine, as well as the position of its blades. Then the Hough transform algorithm is used for detection of the wind turbine tower, hub, and blades, and a Kalman filter is used to keep the target within UAV's field of vision.

The work cited in [149] presents a visual-based object tracking and following system. The system was applied on a flying robot capable of following a variety of static and moving targets at varying distances from 1-2 m to 10-15 m.

In a very different type of system, a bird-inspired IBVS scheme was proposed in [184], for micro aerial vehicles capable of performing high-speed aerial grasping tasks.

In [210], a quadrotor that has an on-board monocular camera and an inertial measurement unit sensor is used to propose a new IBVS control law. Image rotations and translations dynamics are used together with quadrotor dynamics to derive a non-linear controller for the UAV.

4.7 Underwater Vehicles

In [95] a position-based visual servo control for an Autonomous Underwater Vehicle (AUV) is presented. The pose of the AUV with respect to the target position is obtained using a laser system consisting of two lasers that project its beams in the image plane, while computer vision algorithms keep track of the target position.

Visual servoing is also used in [11] to implement an underwater pipeline following AUV. AUV's servoing system uses binormalized Plücker coordinates of the pipeline borders that are detected in the image plane, as feedback information. The control scheme proposed in this paper considered the full system dynamics to improve stability.

5 Neural Network Based Visual Servoing Systems

This section discusses the use of neural networks and in particular of DNNs in visual servoing systems.

In classical computer vision systems, feature descriptor methods were often combined with other traditional machine learning algorithms [138] like Support Vector Machines, and K-Nearest Neighbors for the problem of object recognition

and detection [112]. Other classical methods like Adaboost and a cascade of classifiers have also been used to select a critical set of features in the problem of face detection [192] with high success. Histograms were used for human detection in [47] and Shallow Neural Networks (SNN) in [189] for face localization.

SNN trained with the backpropagation algorithm [110] have been used in visual servoing as described in [150] and [151]. In these applications, SNNs were trained to solve the problem of road following with an autonomous land vehicle. The SNNs consisted of three layers, an input layer of size 30 x 32 containing the pixels of a road image and an 8 x 32 range finder image connected to one hidden layer of 29 units. This layer was fully connected to a 45 neurons output layer to indicate the 45 directions that the autonomous vehicle can take. In a subsequent work, the architecture of the SNN [151] was changed to contain only 5 neurons in the hidden layer and 30 neurons in the output layer.

SNNs were successfully used in the task of real-time road detection. However, when more complex image features should be detected, SNNs did not work well. To address this problem, recent approaches in computer vision use deep neural networks (DNN) [12]. Contrary to SNNs, DNNs consist of a deep stack of multiple neuron layers, where each layer is capable of learning a partial representation of the features of an object. Applied in a visual servoing system, DNNs can, for example, learn the complex implicit relationship between the pose displacements of a robot and the observed variations in the global descriptors of the image, such as points, lines, or geometric moments [196].

DNNs include a wide variety of networks such as Convolutional Neural Networks (CNNs) for computer vision, Long Short Term Memory (LSTM) networks for applications in speech recognition, Recurrent Neural Networks (RNN) or Gated Recurrent Units (GRU) [12] for time series forecasting, among others. CNNs [179] are DNNs that were originally designed to process images. LetNet [111], the first implementation of a CNN, was designed for handwritten and machine-printed character recognition, but now CNNs are used in the following computer vision tasks:

1. Image classification: The goal is to classify or predict the class of an object in an input image [42, 105].
2. Object detection and localization: The input is an image, and the output are bounding boxes surrounding the detected object(s)[181].
3. Object segmentation: The goal is to group pixels in a digital image into multiple regions or segments and assign each group to a specific class [66].
4. Target tracking: The goal is to estimate the state of a target object present in the scene from its previous state information [149, 152].

Several surveys, covering the recent advances in DNN's architectures and its applications in computer vision tasks, have been published in [12, 98, 116, 137, 179, 207]. In the rest of this section, we present a brief overview of CNNs. Lastly, we describe some examples of the most recent work on CNNs for computer vision tasks.

5.1 General Architecture of CNNs

The main purpose of this section is to introduce CNNs and their use, a more detailed description of DNNs and CNNs can be found in [66].

CNNs are first trained with a large dataset of images. Once they are trained, the networks are used to perform inferences on images that have not been seen before. The CNNs learn to extract object's features with a deep pipeline of layers. Figure 2 shows the architecture and layers of a generic CNN for image classification.

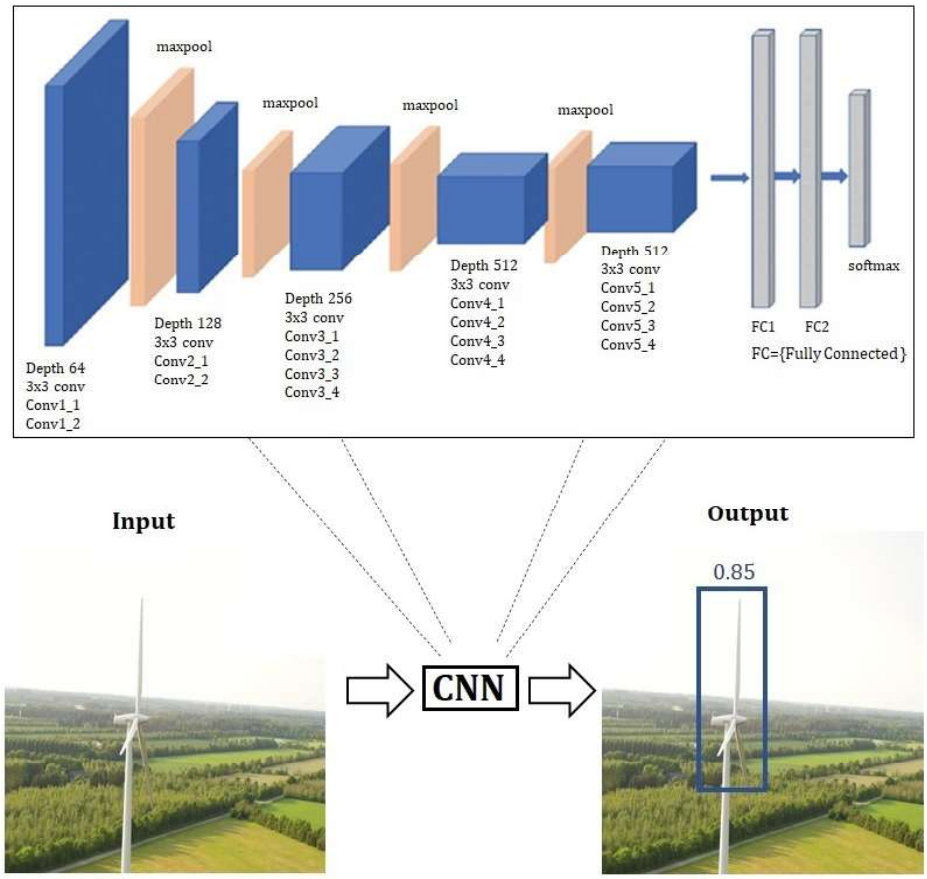

Fig. 2 An example of CNN architecture for object detection application.

Some of the most common layers found in modern CNN architectures are:

1. Convolution: The convolution layer applies the convolution operation to the input image, represented as a vector, using filters (kernels). The use of kernels helps to keep the 2D spatial information. The image is scanned with these kernels producing as output, a feature or activation map.
2. Pooling: The pooling layer is usually placed after the convolution layers. Its goal is to perform dimensionality reduction on the number of weights used in the layer. This is done to reduce model's complexity and avoid overfitting. Some of the common pooling operations are average and maximum.

3. Fully Connected: The Fully Connected Layer (FC) is also called Dense Layer because it connects all the neuron's outputs of a layer to the inputs of the next layer).

The convolutional layers in a CNN extract object's features in a hierarchical way, from the simple ones like lines in the first layers to more complex features like forms in later layers. The final layer in object classification applies a softmax function to produce as output the probability that an object belongs to one of the training categories.

CNNs designed for object detection add other layers that produce feature maps and apply convolution filters to detect objects with bounding boxes. The feature maps are associated with a set of predefined bounding boxes of different dimensions and aspect ratios. The network generates a large number of boxes that are pruned using a technique called non-maximum suppression that retains only the most likely ones. The output of the network is a bounding box containing a detected object and an associated confidence value about the detection, as is shown at the bottom of Figure 2.

5.2 Training and Inference in DNNs

The goal of training a DNN is to find the optimal set of weights and biases that minimize a cost function. Training is done using the backpropagation algorithm [162] and an optimization method, such as stochastic gradient descent or the Adam's algorithm [124]. Training requires a large labeled dataset and is usually performed on one or more GPUs. During training, data is passed through the neurons in a DNN that contain nonlinear activation functions, producing outputs that are feed to the next layer. In modern DNNs, the rectified linear unit (ReLU) or one of its variants [64, 88, 142], is commonly used as the activation function.

To reduce simultaneously the number of images required for training and the training time, transfer learning techniques are commonly used in DNNs. In transfer learning, a model trained for a specific task is modified to make predictions for a different task [66]. To implement transfer learning, the pretrained convolution and pooling layers that are part of the first layers of a CNN are retrained with images belonging to a new task but at a very low learning rate. At the same time, the last layers of a CNN are substituted and trained at a faster rate [125]. The goal is to avoid that the first layers forget the simple features learned on the original training dataset since they will be also used in the new task.

After training, the DNN model attached to a camera performs inferences on images that the DNN has not seen before. During inferencing, pixels from an image are fed to the CNN and processed by the neurons of each layer, producing a bounding box prediction around a detected object and its associated confidence value. Modern CNN architectures are optimized for real-time execution and/or to achieve maximum accuracy.

DNN's have hyperparameters such as learning rate, batch size, and momentum that affect model's performance [66]. These hyperparameters are tuned to enhance the model's ability to perform correct inferences. Lastly, in visual servoing applications that require real-time response, the feedback signal from the DNN should have ideally low latency. A wide variety of DNN architectures have been proposed

in the literature to address these issues. The next section discusses some examples of these new architectures.

5.3 Modern CNNs

AlexNet [105] was the first CNN that won, by a large margin, the Image Classification Challenge (ILSVRC) in 2012 [105]. Other popular CNNs used for object classification are VGG16 [175] and GoogleNet [180]. These networks were designed to achieve the highest accuracy in object classification, without regard to the computing power needed to perform inferences in real time.

After the success of AlexNet, the next breakthrough work in DNNs was ResNet [75]. ResNet solved the vanishing gradient problem by allowing the data generated during training to skip one or more layers without compromising the accuracy achieved in object recognition and detection. This feature allowed training networks composed of hundreds of layers without sacrificing performance. However, one of the issues with RestNet is that it requires long training times.

The CNNs designed for computer vision systems in robotics should perform inferences in real-time with high accuracy and at the same time, have a small size to fit into resource constrained devices. Examples of CNNs that were designed with these goals are Yolo [157], SqueezeNet [85] and MobileNets [80, 81, 167] but also ShuffleNet [208] and SSD [121].

Yolo [157], a popular CNN for object detection, is capable of performing inferences in real time. Yolo achieved real-time performance but at the cost of sacrificing accuracy. More recent versions of Yolo like Yolov3 [159] or a similar network called Single Shot Detector (SSD) [121] have improved the original architecture to achieve higher accuracy without sacrificing real-time performance.

SqueezeNet [85] was one of the first small-sized models that performed well on the ImageNet dataset. Compared to AlexNet, SqueezeNet has similar accuracy but uses 50 times fewer parameters. While AlexNet uses five convolution layers with large kernels, followed by two fully-connected layers, SqueezeNet has smaller convolutional layers with 1×1 and 3×3 kernels.

More recently, other networks such as Squeeze and Excitation Networks (SENets) were introduced in [82]. These networks include a computational unit called Squeeze-and-Excitation (SE) block that can be integrated into other architectures such as ResNet. SE-blocks allow weighting each channel of a convolutional block adaptively to indicate how relevant a feature map is. The use of SE-blocks has been shown to improve training time and accuracy. Given the success of this idea, SE-blocks have been introduced in the latest architectures such as ShuffleNet [208]. However, authors in [208] report that when SE blocks are integrated into ShuffleNet, the network is significantly slower than raw ShuffleNet on mobile devices.

MobileNet is another example of an optimized network that can be used for mobile and embedded applications. MobileNetV1 [81] replaces expensive convolution layers by a cheaper depthwise separable convolution 3×3 layer followed by a 1×1 convolutional layer. The main result of this combination is reducing the parameters, while to reducing latency, MobileNet decreased the number of layers used to 28. MobileNetV2 [167] uses depthwise convolutions but rearranges them in a block called Bottleneck Residual block that consists of three convolutional layers. The first layer, called the expansion layer, is a 1x1 convolution that expands

the number of channels in the data before going into the depthwise convolution. The last two layers are a depthwise convolution that filters the inputs, followed by a 1×1 pointwise convolution layer. However, the pointwise convolution in V1, either kept the number of channels the same or doubled them, while in V2, it does the opposite, making the number of channels smaller. The name projection layer comes from the idea of projecting data with a large number of dimensions into a structure with a lower number of dimensions.

The architecture of MobileNet version 3 [80] was partially obtained via a technique called automated neural architecture search (NAS). The network uses MnasNet-A1 [182] as an initial point and then applies the NetAdapt algorithm [203] on top of it, which works to optimizing and shrinking the pre-trained MnasNet-A1 until it reaches a given latency, while maintaining accuracy high. One of the novel ideas in MobileNetV3 is the incorporation of the squeeze-and-excitation modules (SE) described previously. Another novelty is the optimization of the architecture by redesigning some of the expensive layers. For example, MobileNetV2 begins with a 3x3 convolution layer that has 32 filters, but experimentation with the network showed that this is a relatively slow layer, since only 16 filters are sufficient. In comparison with the previous versions of MobileNet that used ReLU6 as the activation function, v3 uses a version of Swish [154] called hard swish ($h\text{-}swish(x) = x\frac{RELU6(x+3)}{6}$), which is computationally less expensive.

The CNNs described above were used as a baseline for other networks that have a different number of parameters, operations, and accuracy (see Table 3). In the table, "MACs" means multiply-accumulate operations. This measures how many calculations are required to perform inference on a single RGB image.

Table 3 Complexity and accuracy of known convolutional neural networks (CNNs).

Model	# Parameters (Millions)	# MACs (Millions)	Top-5 (%)
AlexNet	1.2	860	83
VGG16	4,24	569	89.9
GoogleNet	60	650	83
ResNet-101	55	5650	94.2
ResNet-152	65	6850	94.2
ResNet-200	100	10500	95.5
SqueezeNet	2.9	66	87.7
SENet-154	138	7800	89,6
ShuffleNet	5.3	260	90
MobileNet-v1	5	750	93.3
MobileNet-v2	5,4	217	92.2
MobileNet-v3 (small)	3,47	300	91
MobileNet-v3 (large)	40	3800	93.3

A recent application of CNNs is in creating Generative Adversarial Networks (GANs) [67]. GANs are a modeling approach for learning deep representations that does not require an extensively annotated training data set. GANs can be used in semi-supervised, unsupervised, and reinforcement learning tasks. The main idea of GANs relies on training a pair of networks (Generator and Discriminator) in

competition with each other. The generator is trained to produce new examples, while the discriminator is trained to classify these examples as either fake (generated) or real (from the domain). The networks are placed to play a zero-sum game where the generator tries to fool the discriminator into classifying fake data as real. As a result, GANs can generate dynamic pictures from static ones, predicting several seconds of a movie, in data augmentation, and to generate text that describes images. Due to its predictive capabilities, GANs can be used for online object tracking systems.

6 DNN-based Visual Servoing Systems

State-of-the-art visual servoing systems employ DNNs in their feedback control loop. DNNs achieve high accuracy in object detection, recognition, and segmentation by adapting to changes in the environment, i.e., they are invariant to changes in scale, position, illumination, occlusion, background, and intraclass variations. This is due, in large part, to the usage of large training datasets, effective optimization methods, overfitting avoidance techniques, and by fine-tuning its performance with hyperparameters [66]. Furthermore, with the use of new DNN-based techniques in end-to-end and direct servoing systems, the controller could be eliminated.

DNNs for supervised learning have also some disadvantages, for instance, they require large training labeled datasets and long training times. DNNs are also generally large in size and require high processing power. Additionally, they produce delays in the feedback loop of a visual servoing system that may cause instability. Lastly, DNNs may suffer from adversarial attacks. In these attacks, a group of pixels in an image is changed to confuse the network, making it classify images incorrectly [201]. However, in the last years, a new generation of networks has been specifically designed to tackle some of these problems. The architecture of DNNs has been optimized without degrading its accuracy and increasing latency [81, 85, 208]. New hardware, such as Google Coral, Intel Movidius NCS, or Nvidia Jetson Nano, has been designed to improve DNN's inference time. Furthermore, the use of transfer learning techniques reduces the amount of images needed for training and protection mechanisms in the networks have been proposed to make them resist potential adversarial attacks [201].

In this section, we describe how DNNs are integrated in the feedback control loop of a visual servoing system. We conclude the section describing End-to-End and Direct Visual Servoing systems.

6.1 Visual Servoing Feedback Loop

Figure 3 shows a simplified block diagram of the feedback control loop of a DNN-based visual servoing system. The camera takes images of the surrounding space and feeds them to a DNN as an input matrix $I(t)$. The DNN will extract features from the image and produce inferences to recognize, detect, or track objects. For instance, in the case of a DNN designed for object detection, the output of the DNN will be a vector $s(t)$ containing the location on the image of the bounding box where the object of interest has been detected, together with a probability

representing the confidence about the detection. An error signal $e(t)$ is then calculated as the difference between the location of the target position $s^*(t)$ and the current position and given as input to the controller. The controller will produce a command vector $u(t)$ of signals that is feed to the plant, representing the robot's actuators. These commands will be processed by the plant to move the robot to a target position $y(t)$.

Like most physical systems, the plant is perturbed with disturbances d such as aerodynamic forces, affect the closed-loop system's performance but can be handled actively. The noise in the measurements n, produces a new system output defined as the signal $\hat{y}(t)$.

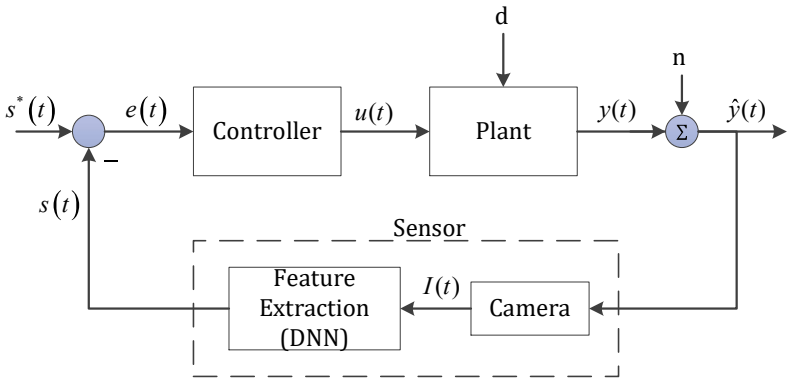

Fig. 3 Block diagram of the feedback control loop of a DNN-based visual servoing system.

6.2 End to End Systems

End-to-end visual servoing systems learn the control laws directly during training. Hence, there is no real controller module, but the control laws are implicitly embedded into the weights and connections of a deep neural network. The idea of these systems is to view the entire problem of mapping input images to a set of control commands as a single indivisible task to be learned from end-to-end. For instance, in [101] a DNN for a self-driving car is trained by feeding images of a road and the steering commands performed by a human while driving the car. Another example is described in [140], where a CNN takes stereo images as inputs and produces possible steering angles for an off-road obstacle avoidance vehicle.

Other examples in self-driving cars are introduced in [26, 36] where an end-to-end learning system is applied to obtain the proper steering angle to keep the car on the lane.

In a different type of application, an end-to-end approach is presented in[169], where the visuomotor representations (i.e., where sensor data is directly trans-

formed into motor actions) of an outdoor UAV have been learned by a CNN that computes the desired pose of a drone given a set of images. Similarly, in [25] a CNN is trained in an end-to-end fashion to predict the heading angle of an aircraft by using image inputs. The visual servoing system is used for autonomous landing of the UAV in case of sensor failure.

James et al. [86] trained an end-to-end CNN and Long Short Term Memory (LSTM) networks to execute a multi-step task for picking up a cube and dropping it to a basket on a table. The network architecture consists of 8 convolutional layers, each with a kernel size of 3x3, except the last one with a size of 2x2. Dimensionality reduction is conducted at each convolutional layer by using a stride of 2. The output of the convolutional layers is concatenated with the joint angles and then fed into an LSTM module. Finally, the data passes through a fully connected layer of 128 neurons before heading to the output layer. The network generates 6 motor velocities, 3 gripper actions, and 2 auxiliary that are cube position and the gripper position.

Other end-to-end systems employ reinforcement learning instead of supervised learning. In reinforcement learning, the network is self-trained by a process of trial and error, and a reward signal is generated when an action is performed correctly. An example is [155] where a Deep Q-learning network is used in conjunction with a Deep Actor-Critic Network to process rewards and generate discrete and continuous actions to control the movement of a ground robot.

Another example of the use of reinforcement learning in end-to-end systems is described in [108] where a robotic arm with an eye-in-hand camera is designed to solving the reaching and grasping tasks where uncertainty is involved. The system is trained by trial and error and does not require any previous information. The system uses two cameras, splitting the problem into long range and short range tasks, where each task is handled by a different controller.

In [163] a robot trained in simulation using reinforcement learning is capable of finding target objects in real cluttered environments by learning collision-free goal reaching policies and semantic labeling using two fully convolutional neural networks.

6.3 Direct Visual Servoing Systems

Direct servoing is a technique that relies on the use of similarity measures to compare the images taken at a current pose of the robot with the target image. Direct servoing aim is to replace classical visual servoing techniques, avoiding the use of any geometrical information about a scene.

Different approaches to creating direct visual servoing systems are described in [18]. For instance, in [18] the author proposed to use histograms calculated from images as similarity functions. Particle filters are then used, where the particles have virtual camera images associated, and the best particle that minimizes the error function is selected. A CNN is trained to learn the control laws that are used to calculate the optimal pose of a robotic arm with 6-degrees of freedom.

Direct servoing may be also implemented without using DNNs, as is described in [130]. In this method, two control laws are derived based on a principal analysis decomposition of the main components of the image. Another example is described in [91], where a quadcopter included an indoor navigation method based on an

optical flow sensor for obstacle-dense environments. To navigate through a sequence of obstacles, images from a stereo camera were processed and a center point–matching calculated to find the depth information that allowed the drone to pass through the center of multiple gates.

In an earliest work, a photometric visual servoing method (e.g., [44]) was proposed, where only pure photometric information (i.e., image intensity) was used in the control law to minimizing the error between the current and desired images. One of the main drawbacks of this approach is that it is sensitive to occlusions and illumination variations, which can lead to less precise positioning results [19].

To solve this problem, in [49], the desired image is adapted to the illumination conditions of the current image taken by the camera. The authors then calculate the Sum of Conditional Variance (SCV) between the reference and the adapted current image to achieve direct visual servoing.

In [145] a vector of visual features is used as a metric. The work proposed using the coefficients of a wavelet transform as the control signal inputs. A wavelet transform is a representation of a signal by an orthonormal series of functions called wavelets [68]. The proposed controller uses the multiple resolution coefficients representing the wavelet transform of an image in the spatial domain.

Lastly, Marchand in [131] proposed a new direct visual servoing technique which does not consider the image itself as a whole in the spatial domain but its transformation in the frequency domain. The Discrete Cosine Transform (DCT) [7] is used to represent the image in the frequency domain as a sum of cosine functions at different frequencies. As a result, the coefficients of the DCT are used as the visual features in the visual servoing control law.

7 State of the Art in DNN-based Visual Servoing Systems

In this section, we present a comprehensive review of recent research work on the application of DNN based visual servoing systems, classified by platform type, i.e., UAVs, ground robots, manipulators, and underwater and surface vehicles.

7.1 UAVs

A UAV designed for wind turbine inspection is reported in [54]. The work describes the implementation of a stereo camera system with 2 DNNs attached that detects wind turbines in images. Stereo triangulation is used to estimate the distance from a wind turbine to the drone. The DNN Yolov2 [158] (with ResNet50 in the feature extraction layers) was used to localize the wind turbine's position in the 3D world coordinate system, using epipolar-plane image analysis.

To navigate towards the center of the windmill, the bounding box parameters of the detected windmill generated from Yolo were used to calculate the center of the bounding box. This was used to control drone's navigation. Camera calibration was performed to compute its extrinsic and intrinsic parameters. The extrinsic parameters were used to detect the camera location relative to the world reference frame. Then the intrinsic parameters were used to transform the camera frame coordinates to pixel coordinates.

However, the experiments in [54] showed that the network was not able to detect a wind turbine at a relatively long distance. This is due in part to the noise and low image resolution (224×224) used for training. As a potential solution to this problem, GANs could be used to improve small object detection through narrowing the representation difference of small objects from the large ones, as described in [115].

Other applications of DNNs in wind turbine inspection are also discussed in [56] and [55]. In [56], WindMillNet, a DNN created by using transfer learning from AlexNet, takes an image from the camera and outputs the probability that a wind turbine appears in the image. When this probability reaches a threshold, the state of the controller in the drone changes from the "scanning state" while searching for a wind turbine to a "found state" where the drone navigates autonomously forward towards the wind turbine and it stops when it reaches a safe distance. WindMillNet architecture consisted of 22 layers including ConvNets, pooling, ReLUs, dropout, and fully connected layers. In this work, the last 3 layers of AlexNet were repurposed to perform the wind turbine classification task.

In a later work [55], a DNN called WindTurbineNet was created using the same structure as WindMillNet, but the final layers were changed to allow the DNN recognizing four classes of objects [curtain; net; wall; wind turbine], which were part of the lab's experimental setup. Contrarily to [56], the images coming from the drone's camera were split into four equally sized segments, where each segment was attached to a DNN that calculated the probability that it contained a wind turbine. A servoing system made the drone navigate left or right according to the probability value obtained by the upper two probabilities calculated by the DNNs.

The forward velocity of the drone was made proportional to the sum of the 4 probabilities corresponding to the 4 image segments, based on the hypothesis that the closer the drone is to the object the higher is the sum of these 4 probabilities. Additionally, the steering angle was made proportional to the difference value between the probability of the left and right segments.

Another example of an outdoor application of UAVs was discussed in [63] for the problem of visual perception of forest trails. The DNN used takes a 101x101 monocular image as an input and produces three values, which are the probabilities of three classes of movements [Turn left (TL), Go straight (GS), Turn right (TR)]. The DNN was trained using a dataset that was collected via three head-mounted cameras. Data augmentation, particularly right/left mirroring, and mild affine distortion were applied to the images to produce more data. To test the network, a reactive controller was implemented to translate the network outputs to control signals, where the forward speed is proportional to the probability of GS P(GS), while the yaw or steering angle is proportional to the value P(TR) - P(TL), in case it is positive the robot is steered to the right, and if it is negative the robot is steered to the left. The network did not generalize well on images with lower quality compared to the GoPro images.

A similar work, but for indoor corridor environments, is reported in [146]. In this case, the flying commands are learned using a CNN called DenseNet-161, which takes an 180x320 RGB image as input and classifies corridor images as [left, center, right, stop]. The UAV was controlled in an open-loop fashion using 4 flight commands: Roll-Right, Pitch-Forward, Roll-Left, Stop, corresponding to the 4 classes [left, center, right, stop], respectively. The navigation method achieved

a 77.3% success rate without collision. This work is based on imitation learning, which aims to learn a control policy and map states to actions by observing expert behavior. One important aspect of this approach is that it can be used in end-to-end deep learning. However, imitation learning is also challenging to scale as it is difficult to get expert examples to imitate every potential scenario that an agent may encounter.

Another recent paper about autonomous navigation in an unstructured forest trail environment is [135]. The work presents an optimized CNN architecture trained and tested using a patch of the unstructured environment dataset IDSIA available at [3]. The network takes an RGB image of size 101x101 as input, and outputs a triple vector holding the classification probability values that are translated into the path direction, whether to the left, to the right, or forward. This network was inspired by the work done in [63], applying data augmentation during training to improve model's performances. However, the method fails to generalize on data with low resolution, and the control flow was not discussed in the paper.

A similar system called Dronet is presented in [122]. Dronet is a CNN that can safely drive a drone through the streets of a city. This network was trained using data collected by bicycles and cars, which are integrated into the urban environment. The network takes an input image 200x200 frame in gray-scale, and its architecture is based on ResNet-8 (eight-layers residual network), while its last two fully connected layers can produce separately a steering angle prediction and collision probability that allows the drone to recognize dangerous situations and quickly react to them while navigating.

The predicted collision probability is used in the control law to modulate the forward velocity, so that the drone could fly at the maximum speed if the probability of collision is null and stop if the probability is 1. The predicted steering angle is mapped to the yaw angle from the range [-1,1] to $[-\frac{\pi}{2}, \frac{\pi}{2}]$.

7.2 Ground robots

DNNs have been also used extensively in stationary and mobile ground robots. The work reported in [62] uses a MobileNet implementation of Single Shot Multibox Detector (SSD) architecture to keep track of a person using a ground robot. As the target is moving, a proportional-integral (PI) control law is applied to maintain the target in the field of vision of the robot's camera. When a person is detected, the position of the bounding box generated by MobileNet is compared to a set-point bounding box position placed at the center of the image. The robot moves to keep the camera's forward speed and yaw rate at a specified safe distance from the moving target object.

Using a different approach, [92] proposed a reinforcement learning method called BADGR (Berkeley Autonomous Driving Ground Robot) that learns its physical environment by experiencing it. This self-supervised end-to-end approach to autonomous navigation uses image data from real-world environments to train a neural network without needing any human supervision or simulation. BADGR image-based neural network model is capable of predicting which sequences of actions will lead to avoid bumpy roads and collisions. The network processes the input images via convolutional and fully connected layers to form the initial hidden state of a recurrent Long Short Term Memory (LSTM) unit [79]. This recurrent

unit takes as input each of the future actions in a sequential fashion and outputs the corresponding predicted future events. Based on these events, a planner selects the actions corresponding to desirable outcomes, such as reaching a goal position or avoiding undesirable events, such as bumpy terrain and collisions.

The training dataset of a predictive model consisted of observations, actions and event labels recorded iteratively and in an online fashion while BADGR robot navigates. BADGR deployment requires the definition of a reward function that defines what the user wants the agent to accomplish in terms of the model's predicted future events, for example by penalizing collisions and positively rewarding driving towards the goal. Given the reward function, the learned predictive model, and the current observation, BADGR can plan and execute a set of actions in a way that the reward function is maximized until the task is completed. BADGR was evaluated against the use of a 2D LIDAR and the use of naïve based policies, where in the latter case, the robot navigates straight towards the goal. BADGR registered a 92% success rate (without collision) against 60% for LIDAR and naïve methods.

Robotics soccer is another example of computer vision applications that require real-time vision algorithms that run with very limited computational resources. Examples of recent robotics soccer applications include ball detection [177], soccer player detection [9, 46], and visual navigation [114].

The ball detection in [177] uses an architecture based on three convolutional layers and two fully connected layers corresponding to x and y coordinates of the ball's center. The network uses a normal distribution around the x and y values, which can help to quantify the uncertainty of the detection. Even though this network performed relatively well (81% x peak and 75% y peak), it could only process a few images per second operating on low resolution images and suffers from over-fitting.

The soccer player detection described in [46] uses two different detectors, one based on SqueezeNet and the other on XNORNet architectures. A XNORNet [156] is an approximation to standard CNNs where both the filters and input of the convolutional layers are binary, which can result in 58x faster convolutional operations and 32x memory savings.

In the case of the SqueezeNet-based detector and in order to reduce the size of the network without reducing the network accuracy, the authors follow an iterative procedure in which the RELU activation function is replaced by a PReLU activation function in the early layers. In case this approach does not increase the network accuracy, the fire modules (Squeeze and Expand layers) in SqueezeNet are replaced by extended fire modules where a 5x5 filter is added to the expand layer in the fire module. Both XNORNet and SqueezeNet based detectors were tested on a NAO robot and achieved very similar performance of around 97% detection rate and 1ms inference time.

Similarly, the work in [114] can detect players, balls, and the orientation of the robot inside the playing field. The classifiers were trained via an active learning-based algorithm that automatically selects and pseudo-annotates unlabeled data. Without using any color information, the system can work in real-time in NAO robots.

To perform robot detection, first a grayscale image is scanned using windows of 16x16 pixels, to find the high contrast regions, which may contain a robot. Then, the robot detection CNN model (RobotNet) takes these greyscale regions

as proposals and classify them into robots or non-robots. The robot's orientation detection module takes as input the bounding boxes coming from the robot's detection module, to detect the lower silhouette of the robot corresponding to its feet, by using a vertical scan lines method. The main idea behind the robot's orientation pipeline is to find the lines around the robot's feet and legs that represent the most likely direction of the robot. For this purpose, a CNN model based on SqueezeNet architecture classify the regions around the lines as side, front or back regions. The ball detection pipeline takes both the high contrast regions and the previously detected robots, in order to generate ball proposals in the grayscale image, which then fed to a cascade of two CNNs: BoostBallNet that limits the ball proposals to a maximum of five, and BallNet to classify the filtered proposals. When the robot is static, the success rates of the robot detection, ball detection, and robot's orientation detection are of 94.90%, 97.10%, and 99.88% respectively. When the robot is moving the robot's orientation detection rate is of 95.52%.

7.3 Robotic arms and Manipulators

In [6], a robotic arm manipulator with a single eye-in-hand camera grabs and holds objects such as leaves. The leaf detector is a CNN based on AlexNet architecture. The proposed CNN consists of five convolutional layers and two fully-connected layers, that classify two types of object leaves and their background. To locate and grasp a leaf from a plant, the CNN was combined with Monoscopic Depth Analysis (MDA), a visual servoing method that uses two images and triangulation to deduce the position of some feature points in the image in the Cartesian space relative to the camera.

However, this work only focused on how to bring the end-effector close to the leaf, and although the MDA approach accurately gives the leaf position, the system suffers from a limitation in which a leaf had to be tracked throughout all the steps of the control algorithm. This implies that if the identification step of a leaf fails, between camera movements, the information about the leaf position is lost, and the procedure has to start from the beginning.

Another work described in [106] presents a robotic grasp detection system. The robot is capable of predicting the best grasping pose of a robotic gripper using an RGB-D (RGB data plus depth) image of the scene. The method uses two parallel ResNet-50 CNNs, one for RGB and another for depth, to extract features from RGB-D images to produce grasp configurations for the objects in the images, using the Cornell grasp dataset [2]. In the grasp predictor, the last fully connected layer of ResNet-50 is substituted by two fully connected layers with rectified linear unit (ReLU) as activation function, and to reduce over-fitting, a dropout layer is appended to the first fully connected layer. This system does not use a physical robot to perform the experiments, and since the network architecture relies on two ResNet50, this may cause difficulties when deploying on real-world grasping [41].

An example of a visual servoing system that employs transfer learning to retrain AlexNet in performing relative camera pose estimation for a robot with 6 degrees of freedom are presented in [20] and [21]. The network is used to move the robot from an arbitrary initial pose to a desired pose with respect to the observed scene.

The authors replace the classical direct visual servoing system with a CNN-based system where the network learns the relative pose between the current and

reference images, and outputs a transformation matrix that represent the current camera frame with respect to the reference frame. To reach a pose related to a desired image, the CNN estimates the relative poses between the desired and reference images, and between the current and reference images. Then from these two transformations we can get the relative pose between the current and the desired images. This information becomes the cost function in an optimization equation. In this work, AlexNet was fine-tuned by training the upper layers and keeping the lower layers responsible for image feature extraction, since these layers perform the same task in the relative pose estimation. The last layers produced the 6DOF pose. The training data used in [20] consisted of a synthetic set of images generated from a single image by simulation.

One of the issues with this approach is that for each new reference pose, the network has to be trained again, and this makes it unpractical for actual industrial settings [204].

In [128], a CNN estimates the probability of success when grasping objects in depth images. Grasps are defined as the planar position, angle, and depth of a gripper relative to an RGB-D sensor. The model architecture consisted of four convolutional layers in pairs of two separated by the ReLU activation function, followed by three fully connected layers and a separate input layer for the distance z of the gripper from the camera.

The network takes a candidate grasp and a depth image as input, and estimates the grasp robustness or probability of success, which is used to rank grasp candidates. The performance of the grasp planning method was assessed on both known objects and novel objects with a 93% success rate and 94% precision, and 80% success rate and 100% precision, respectively. The grasp failure cases were mainly due to poor depth sensing in measuring thin parts of the object geometry, and due to collisions with the object.

Another recent work where a generative approach to grasping is presented by Morrison et al. [139]. The authors propose a Generative Grasping Convolutional Neural Network (GG-CNN) as a visual grasp detection algorithm, which is a fully convolutional network that maps an input depth image to a prediction of grasp quality and pose at every pixel in real-time. The proposed network takes a gray scale depth image (I) and generates a grasp pose for every pixel (G), consisting of the grasp quality (Q), grasp width (W), and grasp angle (F). The output of this network is used to compute the best grasp point to reach. The network was trained using the Cornell Grasping Dataset and evaluated on the Jacquard dataset [4].

In the grasp detection pipeline, the GG-CNN takes a 300x300 depth map image and produces a grasp map. The grasp quality was refined using a Gaussian filter. The best grasp pose is determined from the maximum pixels in the filtered grasp quality image and the rotation and width are calculated from (F) and (W) images, respectively. The velocity signal of the end-effector was calculated by a PBVS controller, based on the poses of the grasp and the gripper fingers. The tests were performed with isolated objects and cluttered objects with a grasp success rate of 100% and 87%, respectively.

In a related work, the authors in [147] introduce a novel technique for transferring a deep reinforcement learning (DRL) grasping agent for gripper pose estimation from simulation to a real robot, in a reverse real-to-sim fashion without fine-tuning in the real world. The approach involves a CycleGAN [211] by effectively "tricking" the agent into believing it is still in the simulator. The CycleGAN

consists of two GANs, where the architecture for both generator and discriminator networks follows U-nets, with seven layers in the encoder and the decoder and the output is the semantic segmentation mask. In addition, a visual servoing (VS) grasping task is included to adjust inaccurate agent gripper pose estimations derived from deep learning. The approach used the CycleGAN to adapt a real image from the camera for its use in simulation. The training dataset consists of RGB-D images manually collected from a real robot, as well as the same amount of simulation images automatically generated using data augmentation. The pose-based VS system was integrated to refine the grasp pose estimation previously inferred by the DRL system, based on the segmentation mask obtained from CycleGAN for tasks like feature extraction, object tracking, and control law encoding. The object's centroid and orientation are the features used in the visual servoing system. The experiments showed that using the CycleGAN alone was not sufficient for transferring the pose estimation policy. However, the system succeeded in positioning the gripper correctly in the x and y directions, as well as orienting the gripper correctly, but failed in the z direction. This is because the depth image is very noisy and lacks a clear definition of the simulated images, making the CycleGAN less effective in adapting the depth images. However, using DRL + CycleGAN worked well for finding an approximate pose that can be sent to the visual servoing module.

7.4 Underwater and Surface Vehicles

The work in [118] proposed an improved recurrent neural network for unmanned underwater vehicle (UUV) online obstacle avoidance. The proposed network is a Convolutional Recurrent Neural Network (CRNN) that uses a convolutions to replace the full connection between adjacent layers of the recurrent neural network (RNN). The obstacle avoidance planning network takes as input an 81-dimensional vector consisting of a direction vector plus 80-dimensional distance-vector obtained from the sonar. The output of the CRNN is a two-dimensional vector from the obstacle, which consists of the velocity and the yaw of the UUV.

During the training step, the use of a CRNN was compared to a regular RNN and gated recurrent units (GRUs) models, in order to assess the performance of the obstacle avoidance planner. The three models have the same architecture: an input layer of 81 neurons corresponding to the input vector, the hidden layer consists of one of three network structures, the middle layer has 24 neurons, and the output layer consists of two neurons (yaw and velocity of the UUV). To avoid overfitting, a dropout with 60% retaining was performed on the three networks resulting in RNN1, RNN2, GRU1, GRU2, CRNN1, and CRNN2. The experiments showed that CRNN performs better than the RNN and GRU models in terms of both MSE and convergence, with a short training time, a simpler network structure, and better generalization performance. In the simulation tests, the planning success rate was 88% and 96% for the RNN algorithms (RNN1 and RNN2) while 98% and 99% for the proposed CRNN algorithms (CRNN1 and CRNN2).

Additionally, the proposed CRNN-based obstacle avoidance planner needs less computing time, generates shorter paths, consumes less energy through the UUV actuators, and less sensitive to noise, compared to the ACO (Ant Colony Optimization) algorithm.

A more recent work in underwater exploration by AUV is presented in [109], where a DNN called PSPNet based on ResNet does image segmentation, enabling the AUV to compute its position relative to the segmented elements. The DNN was trained using synthetic images with their corresponding labels generated from the Unreal a game engine. The DNN was implemented on an FPGA, however, the control block of the AUV was not discussed by the authors.

Another state-of-the-art method has been introduced by [51] to solve the underwater localization problem. The proposed approach estimates a cross-view and cross-domain image matching in underwater scenarios with partially submerged structures. A vehicle equipped with a Forward-Looking Sonar (FLS) obtains acoustic images in a place where there is an aerial image available. The aerial image can be obtained by drones or satellites. Then, this aerial image is preloaded into the vehicle or previously captured in the case of a hybrid vehicle that can navigate in both air and underwater and can acquire both high-resolution aerial images and underwater acoustic images when diving [53, 136]. The approach consists of three phases: Aerial image processing, Underwater Acoustic image processing, and matching process. In the first phase, segmentation and binarization of the aerial images are performed with the help of a CNN. The scene is segmented, but only the stationary structures remain in the binary image. In the second phase, a threshold is applied to acoustic images to remove low-intensity acoustic return and reduce noise. The last phase of image comparison uses a CNN capable of producing a matching score to identify similar places. The first CNN responsible for the semantic segmentation of the areal images is based on the U-Net architecture [161] with a 256x256x3 color aerial image as input and a 256x256x3 segmented image as output. The network architecture consists of five encoder layers and four decoder layers, with the original padding being preserved to avoid cropping and finally a softmax layer.

DeepLab [34] is another type of network which is widely used in semantic segmentation. DeepLab achieves high performance on semantic segmentation by combining DCNN and probability map models to improve object boundary detection segmentation extracting dense features using atrous convolution (or convolution with upsampled filters). However, the major challenge to achieve reasonable accuracy through these deep CNN models was the high computational cost [90]. To overcome this issue, Xception [39] used more efficient model parameters and gained robust performance without increasing capacity. MobileNets [34] introduced an efficient convolution neural network architecture to boost the performance of semantic segmentation by building low latency models for mobile vision applications, and it was used as a backbone for DeepLab ("Mobile DeepLabv3") in Camille et al. [100], where the issue of multi-class indoor scene segmentation was addressed. The network in [51] used in matching problems was inspired by the Siamese network [40]. The network takes two 256x128x1 images as input and produces a matching score.

Classic Image-based matching techniques proposed in the literature include block matching [173], and Semi-Global Block Matching (SGBM) [78]. Nevertheless, these methods suffer from different limitations [28] (e.g., match ambiguity, inadequacy of geometric model, and disparity sampling), and they employ cost functions that are handcrafted, or where only a linear combination of features is learned from data [126]. Contrarily, the Siamese network can learn the similarity between a template image that holds a particular object of interest and a search

image where a similar-looking object is to be found [65]. To achieve this goal, two identical CNNs are trained with their respective template and search images to represent random objects in an embedding and used to perform an efficient comparison. Cross-correlation is used to create a similarity score map, from which the maximum value is chosen as the predicted landmark location [65].

Unmanned Surface Vehicles (USVs), or Unmanned Surface Vessels, have recently attracted significant attention for their potential applications in performing time-consuming and/or dangerous missions such as surveillance and reconnaissance, patrol, environmental monitoring, and inspection of marine structures [72].

The authors [200] propose to use a deep CNN to detect the obstacles and maneuver safely by learning how to steer the vessel through sample data. The network was trained to recognize dangerous collisions and change course in accordance with the Convention of the International Regulations for Preventing Collisions at Sea (COLREGS) which describes operations and manoeuvres to perform in situations like crossing, head-on and overtaking. The CNN model was trained using USV maneuvering vision data that was created from the European Ship Simulator game. The algorithm showed good results, however, the authors did not describe the details of the CNN architecture proposed.

8 Discussion

In this survey, we presented a comprehensive review of state of the art research work on visual servoing systems. The systems we have described employ classical methods that have not been included in other surveys, together with DNN-based approaches. One of the goals of this survey was to review state-of-the-art on DNN-based visual servoing systems, since this technology has become the predominant one in computer vision, robotics and other artificial intelligence related fields.

Additionally, this paper extended the taxonomy used in classical visual servoing systems to include the newest developments in this field, e.g., direct servoing, end-to-end, and target tracking systems. In particular, end-to-end systems have changed the traditional concept of visual servoing systems as a feedback control loop system. Now, DNNs are capable of learning the control laws directly during training, eliminating the controller and transforming these systems in open-loop systems. An additional benefit of end-to-end systems is that they reduce system's complexity, at the cost of relying on the use of exhaustive training techniques. At the same time, well know analytical methods like Lyapunov analysis, commonly used in control systems to determine system's stability can not be used in these systems. Moreover, recent direct servoing systems eliminate the need in the classical approaches for having a priori geometrical information about the target object space or camera calibration parameters.

The surveyed systems show that DNNs will continue playing a key role, as new and improved DNN architectures will emerge. For instance, in [77] the concept of capsules was proposed. These structures are aimed at better modeling hierarchical relationships that are invariant to the transformations commonly found in images. Capsules will ease training and produce more accurate results in object recognition and detection. However, to be used in visual servoing systems, these new architectures need to be optimized in size and latency, as latency introduces a delay that may affect the control system's stability. Additionally, transformer

networks, originally proposed in 2017 for natural language processing [190], are now widely used in computer vision [52, 99, 198] and may even replace CNNs in the near future. Transformer networks will improve the computer vision systems used in visual servoing for robotic applications.

More importantly, depth estimation can be now performed accurately with DNNs [209]. These networks fused with object detection models makes it possible to create robotic systems that use only a monocular video camera to perform visual servoing.

In spite of the current improvements in DNN architectures, there are issues that must be addressed to implement DNN-based visual servoing systems in robotic systems. For instance, in object detection the bounding box data about the position of a detected object has statistical variations due to UAV movement. This must be modeled [54] to be properly filtered. Additionally, the uncertainty produced by the inference processing should be estimated reliably. This is a challenging problem since there are no ground truth estimations available. However, several approaches in uncertainty estimation have been proposed to solve this problem, using for instance calibration techniques [69, 76], Bayesian deep learning [127, 143] ensemble methods [107], or dropout sampling [60, 97].

Lastly, transfer learning techniques commonly used in DNNs to reduce the number of training images needed in object recognition and object detection, require labeled training datasets with bounding boxes. This preprocessing step may be partially automated by using simulations and synthetic augmentation techniques.

9 Declaration

9.1 Funding

The authors did not receive support from any organization for the submitted work.

9.2 Conflicts of interest

The authors declare that they have no conflict of interest.

9.3 Availability of data and material

Not applicable

9.4 Code availability

Not applicable