Recent advancement in vision-based robotics and deep-learning techniques has enabled the use of intelligent systems in a wider range of applications requiring object manipulation. Finding a robust solution for object grasping and autonomous manipulation became the focus of many engineers and is still one of the most demanding problems in modern robotics. This paper presents a full grasping pipeline proposing a real-time data-driven deep-learning approach for robotic grasping of unknown objects using MATLAB and convolutional neural networks. The proposed approach employs RGB-D image data acquired from an eye-in-hand camera centering the object of interest in the field of view using visual servoing. Our approach aims at reducing propagation errors and eliminating the need for complex hand tracking algorithm, image segmentation, or 3D reconstruction. The proposed approach is able to efficiently generate reliable multi-view object grasps regardless of the geometric complexity and physical properties of the object in question. The proposed system architecture enables simple and effective path generation and a real-time tracking control. In addition, our system is modular, reliable, and accurate in both end effector path generation and control. We experimentally justify the efficacy and effectiveness of our overall system on the Barrett Whole Arm Manipulator.

1. Introduction

Recent advancements in robotics and autonomous systems have resulted in an increase in autonomous capabilities and the use of more intelligent machines in a wider range of applications [1–4]. Gripping various geometric objects is a key competency for robots to achieve more general-purpose functionality [5]. The perceptual abilities of such general-purpose robots might be used to visually recognize robotic end effector configurations to safely grab and lift an object without slipping [5]. Multiple techniques have been proposed to generate effective grasps for everyday robotic applications. Nevertheless, robotic grasping still provides many challenges for researchers and is still among the most demanding problems in modern robotics.

The various approaches of robotic grasping can be divided into two categories, analytical and data-driven [5]. Most of the effort of traditional analytical approaches lies in designing hand-engineered features for the detection of grasps. The performance is then evaluated according to physical models [6] such as the ability to resist external wrenches [7] or the ability to constrain the object’s motion [6, 8]. Such hand-engineered function could reach a very complex state which implies lots of generalization and simplification of the kinetic and physical problem. The work in [9] describes an automated grasping system for 2.5D objects. The approach relies on capturing the object surface curvature with a focus on concave and convex edges to achieve 2D immobilization using numerical computation and hand-engineered equations. The optimal grasp is then generated and extended to 2.5D by performing tactile probing. The work in [10] proposes a grasping algorithm for 3 finger grippers. Their grasp synthesis approach relies on object curvature by mean of a curvature function. The generated grasps are then filtered to achieve maximum stability using multiple kinematic criteria such as the force-
closure criterion and finger adaptation criterion. Analytical methods may not be the ideal approach for the exploration of previously unknown scenes since they restrict the system to situations predicted by the programmer [5]. In addition, such methods tend to heavily rely on the geometric features of a specific object, which might be corrupted with partial and noisy information from image and depth sensors [11]. On the other hand, empirical methods provide an increased cognitive and adaptive capacity to the robots, while eliminating the need to manually model a robotic solution [12]. Some techniques associate good grasps points with an offline database [11] reducing the challenge to object recognition and pose estimation during online execution. However, this approach is unsuitable for the exploration of unknown scenes and is unable to generalize to new objects.

Deep learning techniques were proven efficient in solving multiple automation problems [13–15] in a variety of fields including medicine [16–18], security [19, 20], management [21–24], and Internet of things [25]. With the proliferation of vision-based deep-learning techniques [26–29], significant advancements have been made in the generation of potential grasps. Multiple deep-learning approaches have been studied over the years; see, e.g., [5] and the references therein. Some of the methods rely on deploying static cameras for object detection and pose estimation [30], and the other class makes use of eye-in-hand cameras mounted on the robot manipulator [31]. Although static cameras can achieve efficient task performance, they require complex hand tracking algorithms, 3D reconstruction, and image segmentation for accurate estimation of object pose and orientation [30]. As for the control and decision-making problem, multiple architectures have been proposed to achieve autonomous manipulation [30–32]. The work in [30] proposes the use of a logical structuring of tasks driven by a behavior tree architecture. This allows a reduction in execution time while integrating advanced algorithms for autonomous manipulation. However, behavior trees are found to be computationally heavy due to the involved ticking procedure [33]. For complex movement tasks, the reader is referred to the work in [34]. This study proposes a covariance matrix adaptation-evolution strategies to ensure the safety and robustness of the robot’s movements in a noisy environment. The approach takes advantage of the dynamic movement primitives method to represent the robot’s reference trajectories in joint space that are used for trajectory tracking. A vision system is employed to get the coordinates of the target object. The effectiveness of the entire system is tested on a 4-DOF Barrett WAM robotic arm. The Robot Operating System (ROS) on the other hand is also widely used for both the control problem and interprocess communication [12]. ROS is an open-source robotic middleware platform for large-scale robotic system development. The work in [35] provides a structured communication layer on top of the host operating system for heterogeneous cluster computing [36]. While ROS is proven efficient in robotics applications, real-time capabilities remain a serious issue [37]. ROS does not support real-time applications and may not be appealing for heavy industrial applications [37]. Real-time control is a critical and essential feature for operation of autonomous industrial manipulators [38]. Multiple techniques have emerged to enable real-time control of autonomous systems [39, 40]. To enable flexible and easy control of industrial manipulators, suppliers like universal robot or Barrett often include external controllers or internal PCs. Those external controllers however increase the cost of robot manipulators and make it even harder for companies to adopt autonomous solution for their desired applications. Implementing easy flexible and modular real-time controllers could help engineers build real-time solutions and decrease the cost autonomous systems.

Industrial companies such as Ocado technologies, BMW group, and Amazon are looking for finding simple and effective solutions that can lead to accurate grasp generation of unknown objects with low computation efforts [41, 42]. The latter is considered as the primary motivation behind our work. We propose a simple, effective, and complete grasp pipeline for practical applications, a pipeline that is easy to implement by novice engineers, easy to handle by average computers, and performs well in noisy industrial environment. We present a full real-time grasping pipeline with real-time MATLAB Simulink controller and data-driven deep-learning approach for robotic grasping of unknown objects using generative grasps Deep Convolutional Neural Networks (CNN).

We make the following contributions:

(i) Computer Vision: the proposed computer vision approach integrates multiple simple and effective techniques for grasp generation. In addition, relying on eye-in-hand camera and visual servoing, we are able to reduce the error propagation while eliminating the need of complex hand tracking algorithms. Detecting and centralizing the object in question helped in capturing the object features and led to more accurate grasps.

(ii) Real-Time control: this paper proposes a real-time controller using MATLAB Simulink real time for autonomous manipulation. The proposed real-time controller eliminates the need of internal PC or the need of external controllers. Our proposed approach involves a simple block diagram, highly flexible and more intuitive for the user.

(iii) Flexible grasp generation: since CNN limits the grasp generation to the normal of the plane of view, we propose a flexible multi-view grasp generation while eliminating such limitations. The generated grasp is therefore not restricted to a single plane but rather multiple angles of view capturing more features.

The structure of the paper is as follows. Section 2 presents the proposed system as well as an overview of the methods used for autonomous grasp generation and manipulation. Section 3 includes our experimental results. Finally, the paper comes to a conclusion and motivates further challenges.
2. Methodology

In this section, we describe each part of our system architecture illustrated in Figure 1. In addition, we present our proposed grasp planning system that deals with object detection, object centralization, and grasp generation algorithm; path planning that generates the desired trajectories of the robot joint positions; and the associated tracking controller.

2.1. System Architecture

2.1.1. Hardware. For autonomous grasping, the Barrett Whole Arm Manipulator (WAM) 7-DOF robotics arm is employed. The WAM arm has direct-drive capability supported by Transparent Dynamics between the motors and joints providing a robust control of contact forces. Mounted on the Barrett manipulator is the B8-series Barrett Hand, a highly flexible eight-axis gripper with three fingers. In our work, the third finger is removed to simulate a parallel plate gripper effect. For RGB-D image acquisition, the Microsoft Azure Kinect RGB-D sensor is mounted on the wrist to achieve dynamic sensing of the environment and objects. The Azure Kinect integrates a Microsoft-designed 1-Megapixel Time-of-Flight (ToF) depth camera using the image sensor presented at ISSCC 2018. The depth sensor supports the following five modes: NFOV unbinned, NFOV 2×2 binned, WFOV 2×2 binned, WFOV unbinned, and passive IR.

2.1.2. Real-Time Controller and Communication. We set the Barrett WAM control loop frequency at 500 Hz. That is, at each 2 ms interval, a signal is sent to the joint encoders (puck) for the arm to function properly; otherwise, this might lead to a heartbeat fault where the manipulator would stop operating. It is important to note that along the experiment the internal PC of the Barrett WAM was not used. In addition, for the closed-loop control, current joint angle acquisition is necessary as well as the desired end effector pose. This calls for a real-time controller that will ensure stable communication in a precise time window. For that purpose, MATLAB Simulink real-time is selected for robot manipulation. To ensure reliable communication between the WAM and the MATLAB, a UDP communication is established. In particular, a UDP-receive and UDP-send block is used. Each joint has a unique ID which helps in identifying the origin of the message sent as well as the destination of the generated control signal. A customized UDP-to-CAN bus communication converter is implemented allowing for commands send by UDP from the MATLAB controller to be converted to CAN and sent to the WAM joints.

2.1.3. Task Planner and Computer Vision. Once our real-time controller is implemented, our manipulator becomes capable of reaching the desired end effector target. At this stage, a task planner that generates the desired end poses based on the surrounding environment and the desired graspsable object is needed. Coded in Python, we implement a task planner that uses PyTorch and OpenCV to detect and locate the object; we apply visual servoing and generate firm grasps based on RGB-D images of the object in question. More details about the planner are provided in the following section. Once the desired end effector is generated, the output is sent to the real-time controller for execution.

2.2. Grasp Planning. Generating a gripper configuration that maximizes a success metric consists of the main objective of grasp planning. Taking into consideration numerous object shapes and all possible orientation variations in the workspace, the planner aims at providing the correct grasp representation that leads to successful object gripping. Robust grasp planning considers the presence of perturbation in the object properties such as shape and poses caused by noisy and partial sensor readings as well as control imprecision. To ensure successful and accurate grasp planning, three key tasks are defined: (1) object detection, (2) object centralization, and (3) grasp generation.

2.2.1. Object Detection. Autonomous manipulation requires localization of the desired graspable objects in 3D space. This information is essential for motion planning since it sets the goal location for the manipulator end effector. For perceiving objects in the vicinity of a robot, 2D vision data such as edge information are used to localize the object. A 3×3 kernel horizontal and vertical edge enhancement is implemented. The resultant image tends to be noisy which necessitates the use of noise reduction where in our case we employ erosion kernels. A 3×3 kernel is subsequently applied to reduce noise and distortion in the image followed by a 3×3 dilation kernel to restore the previous edge scale. Following this step, bounding boxes are generated around the available objects each with a coupled ID. After specifying the desired Object ID, the generated Bounding Box is locked reducing any effect of the external environment noise and/or perturbation. Figure 2 illustrates the steps involved in our object detection approach.

2.2.2. Object Centralization. The main cause behind inaccurate grasp generation is in the camera perspective. For robust grasp generation, localizing the object is not sufficient and uniform object locations in the field of view of the camera could be beneficial. Furthermore, most depth cameras possess an optimal working range. However, when dealing with unknown objects with different heights and shapes, using a stationary camera would not be the best solution. In our system, we install the camera at the end effector making the camera dynamic. As the end effector or camera moves towards an object from its initial position, a better shot can be captured depending on the object shape and camera position. This approach assists in reducing the errors resulting from multilevel processing and camera perspective. Visual servoing, also known as vision-based robot control, is implemented to center the object in the field of view of the camera. Visual servoing [43] is a technique...
that uses feedback information extracted from a vision sensor to control the motion of a robot [44]. This technique minimizes the camera distortion effects and ensures that the object is fully contained in the field of view. Three visual servoing techniques are discussed in the literature; see, e.g., [45], in particular, position-based visual servo (PBVS) and image-based visual servo (IBVS) and hybrid visual servo. In PBVS, vision data is used to reconstruct the 3D pose of the manipulator and a kinematic error is generated and mapped to the robot actuator commands. In IBVS, the position error is generated directly from the image plan features. Furthermore, there exist two architectures for visual servo integration in the control loop. The first architecture is known as dynamic look and move, which relies on external input provided by the vision system to the joint closed-loop control of the manipulator. The second architecture is known as direct visual servo, which uses the vision system directly in the control loop. In our work, we use an image-based visual servo in a dynamic look and go configuration. The bounding box location is used as our visual feedback and the arm is kept moving until the calibrated camera center-line is aligned with the object location, and until the distance from the camera to the object’s uppermost plane is equal to an “optimal” depth value for grasp generation. Consequently, the acquired depth image is used for grasp generation.

2.2.3. Deep-Learning-Based Grasp Generation. In order to generate efficient grasps for diverse geometric shapes, the employed deep-learning approach is presented. To ensure firm grasping with an effective approach in capturing the geometric key-points of the objects, we use preprocessing techniques targeting edge enhancement, and postprocessing techniques for feature matching. In what follows, we describe the steps employed in our preprocessing and postprocessing techniques and present our CNN for grasp generation. (see Table 1)

(1) Preprocessing. Irrespective of the available depth-sensing techniques such as Stereo Reconstruction, Infrared, Time of Flight, and LiDAR, the depth information is always bounded by a minimum and maximum range, beyond which the depth information is clipped. Adding to this, light reflection, refraction, diffusion, and shadows all contribute to lowering the efficiency of the depth sensor and producing invalid spots in the depth image. To mitigate the effect of invalid data in our depth image that produces invalid grasps, prepossessing of the depth map is crucial before further data manipulation. To do so, we propose a technique that detects invalid pixels in the depth image and reconstructs the invalid data. The proposed preprocessing method is detailed in Algorithm 1 and illustrated in Figure 3.

(2) Generative Grasp Convolutional Neural Network. As mentioned in the introduction section, the aim of our work is to implement a full, simple, and effective grasp pipeline to enable grasping of unknown objects that can be easily adopted in the industry. Thus, one of the main factors to be considered is the speed and size of the network. Many networks have been implemented in the field to enable accurate and precise grasp generation such as the network presented in [48]. For our pipeline, we use a fully convolutional neural network for grasp generation that is proposed in [31], which is a generative grasp convolutional neural network. The main advantage behind this network relates to its small number of parameters. It consists of six convolutional layers (see Figure 4) with a total of 62,420 parameters compared to 8,806,986 parameters in the network proposed by [31]. The relatively small number of parameters yields less computation efforts that can be easily
Table 1: What our work has to offer. A comparison between our approach and the related work in the field.

<table>
<thead>
<tr>
<th>Feature</th>
<th>[46]</th>
<th>[11]</th>
<th>[30]</th>
<th>[31]</th>
<th>[47]</th>
<th>Our approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unknown objects</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Object feature matching</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Eye-in-hand camera and object centering</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Grasp generation complexity</td>
<td>DL + analytical</td>
<td>DL</td>
<td>3-stage</td>
<td>DL</td>
<td>Analytical</td>
<td>DL</td>
</tr>
<tr>
<td>Training data availability</td>
<td>No</td>
<td>No</td>
<td>NA</td>
<td>Yes</td>
<td>NA</td>
<td>Yes</td>
</tr>
<tr>
<td>Real robot experiments</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Multi-view grasp generation</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Robot control</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Real-time control without arm controller</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

(1) Capture image (with likelihood of invalid depth data)
(2) Generate 2D gradient of the depth map
(3) Threshold the gradient to produce a binary mask
(4) Flood-fill the mask. At this stage a mask that indicates all the invalid spots in the depth image is generated
(5) Dilate generated mask with a $3 \times 3$ kernel
(6) Feed the generated mask to OpenCV inpainting function that attempts to deduce and reconstructs the invalid data from the pixels’ value in its neighborhood

Algorithm 1: Preprocessing algorithm.

Figure 3: Depth image preprocessing steps.

Figure 4: Description of the neural network architecture that is implemented in our study. The generative grasp CNN takes as an input an inpainted depth image, and generates a grasp pose for every pixel, comprising the grasp quality, $Q_\theta$, grasp width, $W_\theta$, and grasp angle, $\varphi_\theta$. 
handled by our employed network compared to other more evolved CNN used for the same purpose.

There are two main advantages of the proposed GG-CNN over other state-of-the-art grasp synthesis CNNs. First, rather than sampling grasp candidates, we directly create grasp poses on a pixel-by-pixel basis, similar to improvements in object detection, where fully convolutional networks are frequently utilized to provide pixel-wise semantic segmentation rather than sliding windows or bounding boxes [49]. Second, unlike other CNNs used for grasp synthesis, our GG-CNN has orders of magnitude fewer parameters, allowing for fast closed-loop grasping. Our grasp detection pipeline can be executed in only 19 ms on a GPU-equipped desktop computer.

The neural network proposed approximates the complex function $M$: $I \rightarrow G$, by training the network. $M(I)$ can be learned by using a set of input images $I$ and corresponding output grasps $GT$. The network equation can be expressed by

$$M_\theta(I) = \{Q_\theta; \varphi_\theta; W_\theta\}. \quad (1)$$

The grasp map $G$ estimates the parameters of a set of grasps, for each Cartesian point in the 3D space corresponding to each pixel in the captured image. It constitutes a set of 3 images denoted as, $Q$, $\varphi$, and $W$. The representations are as follows:

$M_\theta(I)$ denotes the neural network with $\theta$ being the weights of the network and $I$ representing the $300 \times 300$ pixel RGB-D input image. $Q_\theta$ describes the grasp quality. Its scalar value ranges within $[0,1]$ with 0 indicating the lowest quality while one the highest grasp quality. $\varphi_\theta$ denotes the grasp angle where the angles belong to $[(-\pi/2), (\pi/2)]$. $W_\theta$ describes the grasp width of a grasp to be executed.

The network is trained over the Cornell Grasping Dataset which contains 885 RGB-D images of everyday objects, with a variable number of human-labeled positive and negative grasps. The network is trained for 32 epochs over 8 hours on an NVIDIA Quadro P100 GPU.

(3). Postprocessing. The grasp representation generated by the GG-CNN output consists of a line position coupled with a specific width and angle. The line position specifies where to look for potential grasp in the depth image, while the angle provides us with the information about the rotation of the manipulator end effector with respect to its normal. After experimenting with the neural network performance, a considerable amount of false-positive potential grasps has been observed. This can be brought back to the noise introduced by the background of the object (i.e., uneven surface of the table that translates into a small variation in the depth image and triggers the GG-CNN to detect an invalid grasp) or sensor readings. For this purpose, we implement a grasp validation condition that checks the amount of depth variation in the potential grasp in order to increase the chances of successful grasps. In particular, if depth is below a certain threshold, the grasp generated is disregarded and not accounted for. This approach would eliminate all false-positives and would only keep the valid grasps for consideration.

On the other hand, in most cases, the width of the line estimated by the neural network does not match the exact width of the object’s feature. A discrepancy between the real feature width and the manipulator opening width can lead to undesirable results and failure of the grasping attempt. To mitigate the effect of this mismatch, we propose a post-processing method presented in Algorithm 2 and illustrated in Figure 5.

Adding to the above, the neural network can detect various potential grasps for a unique object. To eliminate further chances of errors and confusion, ten trials are specified. Similar grasps are aggregated based on the output of the trials. After querying the trial, the manipulator considers the most recurring grasp as the final potential grasp that is ready for application. Once the ideal grasp is generated, there is no need to keep the object in the center of view. The next challenge lies in applying the grasp by accurately controlling the robotic manipulator.

2.3. Path Planning

2.3.1. Inverse Kinematics. The motion planning component of the system is responsible for computing trajectories for the arm that will place the gripper fingers at or close to the generated grasp configuration. To complete this task, a mapping between the world and joint coordinates is required. We approach this problem by using inverse kinematics. Inverse kinematics (IK) is a technique that maps the end effector position to the corresponding joint angles. Given a desired end effector pose, the IK solver generates the joint angles needed to move the arm to the desired pose avoiding collisions with itself. To compute the correct joint positions that leads to a certain six-dimensional Cartesian pose, detailed knowledge of the geometry and configuration of the robot is essential. The SolidWorks Computer Aided Design (CAD) model provided by Barrett offers detailed information about the joint configuration of the robot. To transfer this information into our controller, a Unified Robot Description (URDF) model is generated. The URDF model takes advantage of the geometric data provided by SolidWorks and generates an XML file that contains detailed information associated with each link such as the joints limits and link length. The latter parameters are listed in Table 2.

To apply inverse kinematics (IK), the Inverse Kinematics block provided by Simulink is used. The IK solver provided by MATLAB allows for accurate joint angle computation without the need for in-depth knowledge of the robot geometry and joint configuration. It also eliminates the need of setting Denavit–Hartenberg (DH) frames transformation necessary for generation of the robot inverse kinematics model. The IK block requires a stored configuration file and three main inputs. The configuration file allows for specific equation generation associated with the described robot geometry. The generated URDF model...
(1) Plot the depth versus image pixels from the starting point of the line to its end.

(2) Apply a low pass filter to the data. In this paper, a 5th-order Butterworth filter is selected due to its flat profile across the bandwidth of interest.

(3) Take the gradient of the low pass filtered depth data.

(4) Find all the local minima and maxima above a certain threshold. This is essential due to the fact that some small irregularities in object’s surface can induce a local minima or maxima.

(5) Select the first minimum and last maximum and compute the grasp width from the distance between the generated points.

**Algorithm 2: Postprocessing algorithm.**

### Table 2: Barrett WAM joint limits and link lengths.

<table>
<thead>
<tr>
<th>Joint</th>
<th>Lower limit (rad)</th>
<th>Upper limit (rad)</th>
<th>Link</th>
<th>Length (mm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>J1</td>
<td>−2.6</td>
<td>2.6</td>
<td>Base link</td>
<td>186</td>
</tr>
<tr>
<td>J2</td>
<td>−2.0</td>
<td>2.0</td>
<td>L1</td>
<td>550</td>
</tr>
<tr>
<td>J3</td>
<td>−2.8</td>
<td>2.8</td>
<td>L2</td>
<td>300</td>
</tr>
<tr>
<td>J4</td>
<td>−0.9</td>
<td>3.1</td>
<td>L3</td>
<td>60</td>
</tr>
<tr>
<td>J5</td>
<td>−4.76</td>
<td>1.24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J6</td>
<td>−1.6</td>
<td>1.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>J7</td>
<td>−3.0</td>
<td>3.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 5:** Depth image postprocessing: matching the generated grasp width to the object feature. The blue line represents the grasp generated by the GG-CNN. The purple line represents the processed grasp generated matching the grasp line to the object width.
is used as a configuration file. The proposed method makes the approach highly flexible. Changing the entire manipulator geometry or even using another manipulator can be simply done by replacing the configuration file stored in the IK block by the corresponding arm model, eliminating the need for link parameter adjustment or axis reconfiguration.

The first input specifies the current joint angle configuration of the robot. The second input allows us to control the error tolerance of the end effector position. That is, if the end effector is within the specified margin, the pose is considered as attained. This input is presented with a list of six variables: \( \delta x, \delta y, \delta z \) represent the error tolerance of the Cartesian translation components of the end effector, and \( \delta \psi, \delta \theta, \delta \phi \) represent the error tolerance of the Euler rotation components of the end effector. The Euler angle system is a method to describe the coordinate transformations. It consists of three independent variables, \( \psi, \theta, \phi \) defined as successive planar rotation angles around the \( x, y, \) and \( z \) axes, respectively. The latter values describe the end effector pose. High tolerance in the translational \( x \)-axis means that a relatively high error in the \( x \) translation component of the gripper is acceptable. Similarly, a relatively high tolerance in the rotational \( z \)-axis means that a high error in the \( z \) orientation component is acceptable. This helps the IK solver in defining the most critical position components and plan the path accordingly.

The third and final input deals with the initial guess. Inverse kinematics algorithms do not provide one unique joint configuration. Multiple joint angles, as well as multiple paths, can lead to the same end-effector pose in the 3D space. Some IK solvers fail to converge sometimes with the presence of strict time limits and others require heavy computation to plan one path. The initial guess helps the IK solver in finding the best path by providing an ultimate starting point for the algorithm. This approach reduces computation, makes the planner faster, and leads to a more accurate path. The current joint angles are fed as an initial guess to inform the solver that joint angles closer to the actual robot state are favored over others. However, since the base joint of the Barrett is limited to \( -2.6 \) and \( 2.6 \) radiant as shown in Table 2, choosing the closest joint configuration as a starting point can result in longer path leading to non-optimal path generation. To resolve this issue, we propose the use of the two inverse kinematics modules. One uses the current joints as initial guess and the other has no initial guess. The two outputs are fed into an error function. Based on the output of this function, the configuration with smaller error is used.

The proposed approach is simple and comes with low computational efforts. The efficacy of our proposed solution is justified experimentally. It also generates the joint angles within five milliseconds. This approach eliminates the need for generating adequate initial guess or the use of optimization techniques. It is also important to note that the generated arm motion is not limited to a planar motion and can follow any reachable path in the 3D space.

2.3.2. Coordinate System. For robotic manipulation, we define two coordinate systems depicted in Figure 6, namely, A and B. We set A, the frame corresponding to the base link of the Barrett WAM, to be the base frame. We define B to be the camera frame placed at arm end effector. To obtain the camera frame, the forward kinematics plugin provided by MATLAB is used. The camera fixation part is designed in SolidWorks and added to the urdf model of the manipulator as a virtual link. The latter is then used to generate the camera frame using forward kinematics and end effector pose. Following this coordinate system, all end effector goal poses are defined with respect to the manipulator base. Similarly, all detected poses and vision output are transformed back from the camera frame to the base frame. The joint coordinates are defined with respect to the manipulator home pose, which implies a fully extended arm in the positive \( z \) direction. Therefore, a pose in the joint space is based on seven joint angles describing the difference angle between the joint goal and home pose. Mapping from a 6D pose defined in the global base frame to seven joint values defined in the joint space is the main task of the inverse kinematics block.
2.3.3. Safe Motion. Once the joint positions are calculated by the IK block and before feeding it to the position controller, certain conditions have to be applied to the velocity to secure safe control. In Barrett safety system, $VL_2$ is defined as the safety system’s motor velocity fault limit while $TL_2$ represents the safety system’s motor torque fault level. Velocities higher than $VL_2$ cause a velocity fault and stop the entire arm motion in an E-stop effect. Similarly, if any joint torque exceeds $TL_2$, a torque fault is raised and the arm stops. To avoid any velocity or torque fault, a velocity limiter block is added. The specified block can slow down the velocity while keeping the control operation safe. This step fortifies a safe operation but can introduce jerky movements. To ensure a smoothed path, a spline function is used. This function smooths out the generated path while securing zero initial velocity and acceleration.
2.4. Controller. The nonlinear dynamics model describing the continuous-time dynamics of a rigid $n$-link serial robot manipulator, with all actuated revolute joints described in joint coordinates, can be represented by the Euler-Lagrange model as follows

$$u = M(q)\ddot{q} + C(q, \dot{q})\dot{q} + b(\dot{q}) + g(q),$$  

(2)

where $q \in \mathbb{R}^n$ represents joint angles, $\dot{q} \in \mathbb{R}^n$ represents joint velocities, $M(q) \in \mathbb{R}^{n \times n}$ is a symmetric positive definite inertia matrix, $C(q, \dot{q}) \in \mathbb{R}^{n \times n}$ is the Coriolis and centripetal matrix, $b(\dot{q}) \in \mathbb{R}^n$ represents the joint friction, $g(q) \in \mathbb{R}^n$ is the gravity vector, and $u \in \mathbb{R}^n$ stands for the torque input.

Once the desired path is generated, the required torques needed to move the end effector to the desired pose must be automatically generated. Many different discrete-time multivariable controllers have been proposed that can be applied to our problem such as for the deterministic setting [51, 52] and the references therein. However, the inevitable noise associated with the measurement of the joint angles yields poor performance and possible vibrations in the arm. There are also multivariable iterative-type controllers with the capability of suppressing measurement noise such as the ones proposed in [29, 53, 54]; however, accurate tracking may require several iterations, and the learning would be based on a specific trajectory. A common class of closed-loop controllers employs slide-mode approach; see, e.g., [55, 56] and the references therein. There are also several optimal stochastic simultaneous multivariable controllers such a PD-type [57], and ones with gravity compensation [58], PID-type [59–61], and even PDD [62]. Although the works in [55–57, 62] include experimental justification on the Barrett WAM, they do not deal with controlling its gripper. All of the aforementioned strategies and many others come with a number of unique features; however, they require deep comprehension of control theory and/or some knowledge of the statistical models. In this paper, we use simple single-input single-output PID controller that can be designed by any novice engineer. In particular, we tackle the torque control problem by introducing closed-loop PID control with gravity compensation. Gravity compensation is a well-known technique in robot design, which compensates for the force generated by the gravity. This technique requires the knowledge of the gravity model. We use the corresponding model parameters provided by Barrett. To account for gravity forces, we set $B(q)$ to be the inverse of the inertia matrix, which is a symmetric and nonsingular. That is, $B(q) = M^{-1}(q)$. Thus, the gravity compensation term $A(q)$ is the product of the inverse inertia matrix and the gravity force vector, $A(q) = -M^{-1}(q)g(q) = -B(q)g(q)$. This term is added to the control vector. The gravity compensation is implemented in a MATLAB Simulink block and added to the controller. For closed-loop control, seven independent PID controllers are implemented and manually tuned for each joint. To tune the PID, a sine wave is used as a fictitious reference trajectory of the joints, and the controller parameters are adjusted accordingly. We also deploy a first-order filter at the PID inputs to reduce the measurement noise.

Before sending the torques to the arm, appropriate transformations are applied to transform the joint torques to motor torques and then to motor current. For that, we use the following:

$$M_i = VJ_i,$$

(3)

where $i$ is the joint number and $V$ is the joint to motor torque transformation matrix provided by Barrett and defined as

$$
\begin{bmatrix}
- \frac{1}{N_1} & 0 & 0 & 0 & 0 & 0 \\
0 & \frac{1}{2N_2} & -\frac{n_3}{2N_2} & 0 & 0 & 0 \\
0 & -\frac{1}{2N_2} & -\frac{n_3}{2N_2} & 0 & 0 & 0 \\
0 & 0 & 0 & - \frac{1}{N_4} & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2N_5} & -\frac{n_6}{2N_5} \\
0 & 0 & 0 & 0 & \frac{1}{2N_5} & -\frac{n_6}{2N_5} \\
0 & 0 & 0 & 0 & 0 & - \frac{1}{N_7}
\end{bmatrix},
$$

(4)

and $N_i$ and $n_i$ are constants provided by Barrett.

Following this step, we multiply the output by the motor-torque-to-motor-current ratio provided by Barrett and finally send the signal to the arm pucks for motion. The tracking performance of the seven joints and corresponding torques are shown in Figure 7. It is important to note that although the Barrett WAM comes with a force-torque sensor at the wrist, we do not use the sensor in our controller.

2.4.1. Gripper Control. For the control of the Barrett WAM gripper, no closed-loop control is needed to be implemented and no parameters are needed to be set or tuned. The gripper simply accepts finger commands in a specific CAN format provided by Barrett that allows to move the gripper fingers to the desired pose as well as open and close commands for grasping and disposing objects. Since the closing profile of the gripper is curved rather than being a straight line as shown in Figure 8, modeling of the gripper closing path needs to be done in order to link the opening width to the...
finger height and avoid any collision with the table. Following this step, we use the postprocessed width generated in Subsection B as a pre-grasping finger opening. Then, only when the gripper reaches the ideal grasping pose, is the closing signal sent. Since the Barrett gripper is equipped with a force-torque sensor, determining the sufficient force to hold the object in question is not needed. The gripper fingers stop automatically whenever enough force is applied to stabilize the object in question.

3. Experimental Results

To assess the performance of our proposed system, a series of experiments are conducted. Multiple unknown objects with different geometric representations are used to test the system performance. In our evaluation, we focus on 2 criteria: the grasp generation and the control of the manipulator. In the first series of experiments, we evaluate the performance of our approach for grasp generation. In the second series, the grasp generation is integrated into the complete grasping pipeline to assess the torque control and path planning.

3.1. Grasp Generation. The efficiency of the approach followed for grasp generation is assessed using various previously unknown objects with complex geometric representations. The objects vary between multiple mechanical tools, transparent and nontransparent bottles, boxes, tape, tubes, and other objects. The proposed approach is shown to be efficient with high-success rates as documented in Table 3 even when the objects are shiny and associated with a defective depth map. The used technique helps in the depth map reconstruction leading to accurate grasp generations. Figure 9 shows the neural network output grasp before and after applying the pre- and postprocessing techniques.

As for the multiple view grasp generation, Figure 9 illustrates the results of the aforementioned techniques in filtering disturbances from the external environment. The grasp generation is successful even on transparent or light reflecting objects. This technique enables capturing the full object feature in a simple and effective way without the need for 3D reconstruction or point cloud generation which could be challenging in noisy industrial environments.

3.2. Overall Performance. Coupled with a reliable control of the Barrett manipulator, successful grasp generation led to successful grasps of multiple objects regardless of their geometric shapes or orientation. Visual servoing performance is proven efficient in limiting grasp application error resulting from camera perspective. The implemented algorithm also centers the object of interest in the field of view within the desired time frame. Our controller yields smooth, relatively fast path execution driving the end effector to the desired pose with millimeters accuracy, and matching of the end effector contact point with the generated grasp, which

<table>
<thead>
<tr>
<th>Item</th>
<th>Box</th>
<th>Tape</th>
<th>Piller</th>
<th>Tube</th>
<th>Eraser</th>
<th>Open box</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of trials</td>
<td>20</td>
<td>25</td>
<td>15</td>
<td>10</td>
<td>20</td>
<td>5</td>
</tr>
<tr>
<td>Success rate (%)</td>
<td>90</td>
<td>92</td>
<td>93</td>
<td>80</td>
<td>95</td>
<td>100</td>
</tr>
</tbody>
</table>

Table 3: Grasp performance.
made grasping of complex shapes possible. Table 3 shows the success rate of our proposed automated grasping approach.

A video describing the overall project performance can be displayed by visiting https://www.youtube.com/watch?v=gl5oefp_QKo.

4. Conclusion and Future Work

We presented a data-driven deep-learning approach to robotic grasping of unknown objects with eye-in-hand camera and real-time controller. Our paper has tackled the autonomous grasping problem from start to finish covering WAM real-time control, data manipulation, grasp generation, and execution. Our proposed approach is suitable for the exploration of unknown scenes and generation of effective grasps for various geometric shapes and unknown objects without the need for complex hand tracking algorithm, image segmentation, or 3D reconstruction. Thanks to the unique well-defined decision-making tree and the integration of different techniques, we were able to eliminate the error propagation resulting from multilevel processing, camera perspective, and non-ideal depth range. Starting with visual servoing, we centered the object in question in the field of view reducing perceptual error and eliminating error propagation. Following this step, we implemented high level preprocessing computer vision-based techniques coupled with edge enhancements and depth data reconstruction method to account for shiny surfaces and light variation making the object localization, grasp generation, and execution more robust. For the grasp generation problem, a generative grasp convolutional neural network was implemented providing efficient grasp configuration for parallel grippers. We topped this by a postprocessing layer that matched the generated width with the object feature gradients. To the best of our knowledge, the latter is a functional attribute that has not been considered in the literature.

From the robotics and control point of view, we tackled the control of the arm in a simple and effective approach. In particular, we proposed a simple real-time control implementation of the Barrett WAM manipulator using MATLAB Simulink real-time module. Unlike the majority of the work implemented on MATLAB for robotic manipulators, MATLAB was not only used for simulation or as a tool that communicates with the arm controller such as the work in [63, 64]. In fact, we used MATLAB Simulink for the full arm control whereas the arm internal PC was left undeployed. By applying our approach to a class of robotic manipulators, one can eliminate the need of the controller box that takes additional space and increases the cost of the arm. This could be substantially beneficial whenever a company desires to adapt a robotic solution in limited spaces with lower cost. Furthermore, the proposed approach is not only cost effective and leads to reduction in space; it is also simple and modular. In our work, the deployed gripper is equipped with a force torque sensor that ensured firm grasps. For future work, a force optimization layer as proposed in [65] can be added to our system to make the approach more modular and functional with different gripper types. In addition, since convolutional neural network limits the grasp to the normal of the plane of view, having an eye-in-hand camera allowed for multi-view grasp generation. This technique enables the generation of grasps in multiple plans covering multiple features and leading to a better grasp generation for complex shapes.

The multi-view grasp generation was implemented from the vision and grasp generation perspective; however, our solution has not employed kinematics. Techniques using kinematics would allow approaching the object from various angles, which could significantly increase the grasping accuracy and the object immobility. Furthermore, dynamic collision avoidance implementation could be of great benefit. Implementing such techniques might result in a more robust system suitable for human-robot application by increasing the safety level and flexibility of the system.

Data Availability

The justification of this work is experimental and no specific data are available. However, we have a link to a video in the manuscript that illustrates the overall performance that can be displayed by visiting https://www.youtube.com/watch?v=gl5oefp_QKo.

Conflicts of Interest

No potential conflicts of interest are reported by the authors.

References


