applied sciences

MDPI

Article

A YOLOv2 Convolutional Neural Network-Based Human–Machine Interface for the Control of Assistive Robotic Manipulators

Gianluca Giuffrida *, Gabriele Meoni * and Luca Fanucci

Department of Information Engineering, University of Pisa, 56122 Pisa, Italy; luca.fanucci@unipi.it
* Correspondence: gianluca.giuffrida@phd.unipi.it (G.G.); gabriele.meoni@ing.unipi.it (G.M.); Tel.: +39-050-2217625 (G.G. & G.M.)

Received: 26 April 2019; Accepted: 28 May 2019; Published: 31 May 2019

Abstract: During the last years, the mobility of people with upper limb disabilities and constrained on power wheelchairs is empowered by robotic arms. Nowadays, even though modern manipulators offer a high number of functionalities, some users cannot exploit all those potentialities due to their reduced manual skills, even if capable of driving the wheelchair by means of proper Human–Machine Interface (HMI). Owing to that, this work proposes a low-cost manipulator realizing only simple tasks and controllable by three different graphical HMI. The latter are empowered using a You Only Look Once (YOLO) v2 Convolutional Neural Network that analyzes the video stream generated by a camera placed on the robotic arm end-effector and recognizes the objects with which the user can interact. Such objects are shown to the user in the HMI surrounded by a bounding box. When the user selects one of the recognized objects, the target position information is exploited by an automatic close-feedback algorithm which leads the manipulator to automatically perform the desired task. A test procedure showed that the accuracy in reaching the desired target is 78%. The produced HMIs were appreciated by different user categories, obtaining a mean score of 8.13/10.

Keywords: YOLOv2; robotic arm; disability; human machine interface; deep learning; artificial intelligence; Kernelized Correlation Filter (KCF); Image Based Visual Servoing (IBVS)

1. Introduction

During the last years, modern electronic power wheelchairs have been equipped by manipulators to compensate the deficit in the manuals skill of users due to accidents or disabling diseases [1]. Such robotic arms are designed to perform simple operations, such as knocking on a door or pressing buttons in a lift panel, turning on the light in a room, etc. Owing to the benefits introduced by such manipulators in terms of increment of mobility and autonomy, the research has been focusing on the realization of robotic arms able to perform very complex tasks such as interaction with small objects [2]. However, for highly impaired users exploiting all the potentialities offered by such manipulators is often hard, especially if the the only device to control the robot is the power wheelchair joystick. Thus, various Human–Machine Interfaces (HMIs) have been developed for people with different disabilities. There are several typologies of HMIs, which are optimized for specific classes of disease. In the case of severe upper-link deficits, brain–computer interfaces can be used. This approach is very promising thanks to its usability by those users with no possibility of moving their arms. However, such HMI requires the presence of numerous electrodes placed on the body of the users [3] or dedicated helmets [4] resulting more invasive than other interfaces.Instead, for less severe disabilities different HMIs are convenient. Ka, Chung, Ding, James, and Cooper [5] propose a semi-autonomous robotic arm with a HMI based on 3D vision to assist users to control the robot. Even if the handling is manual, users are supported in the objects interaction by some predetermined manipulations options

that can be addressed by voice commands. Although such interface is really promising, the voice interaction with the HMIs might be really arduous for users with severe dysarthria, which is typical in the case of degenerative diseases or strokes [6]. Indeed, in the presence of severe dysarthric, a speaker dependent voice commanded interface might be necessary in order to make the recognition performances acceptable [7]; it would require much effort by the user to train the interface to increase its usability. Previous versions of this work [8–10] implement an autonomous eye-in-hand manipulator which exploits the image features extracted by means of a camera and other sensors data to realize a close-feedback loop for the control of robot kinematics. The offered HMI is based on a touch screen in which the user can visualize the camera frames and press the target area where he wants to lead the manipulator. A Computer Vision algorithm is used to perform the image features extraction and their tracking over different images. The main drawback of this approach is the lack of robustness to the errors of the users. In fact, to lead the robotic arm to a predetermined area, it is necessary that the users press exactly the correspondent area on the HMI. Such task can be hard due to spasms or in the case of limited hands control. Rabhi, Mrabet, Fnaiech [11] used a machine learning approach to adapt the HMI to the manual skills of the users. Such method makes the system user dependent and requires a preliminary HMI training phase to be adapted on the user, which might be frustrating. In this work, we propose a user independent five-Degrees of Freedom (DOFs) manipulator equipped by a monocular camera proximity and force sensors. In [8–10], the robotic arm autonomy is reached by a closed-feedback loop that exploits image features extracted by the camera images. The main advantages of this work is due to an Artificial Intelligence (AI) algorithm that recognizes and locates the objects with respect to the camera frames. In fact, thanks to the AI, objects, such as the buttons of a lift panel, are shown to the users surrounded by bounding boxes.This increases the HMI robustness, since it accepts clicks only inside the bounding box shown, avoiding starting the approach in the case of user errors. For more severe impairments, a *Manual raster scanning* and a *Time raster scanning* HMIs were also implemented. In all cases, the HMIs are pretty intuitive, requiring very short training periods (in the order of ten minutes). The system software was implemented inside Robotic Operative System (ROS) environment. ROS is an open-source and multi-platform framework that manages the multiple tasks as nodes of a network that exchange messages in form of *topics* or *services*. The entire software runs on a Raspberry PI 3 board, with the only exception of the AI which runs on a hardware accelerator. The remainder of the paper is organized as follows. Section 2 shows the conceptual architecture of the proposed prototype, illustrating its working principle and explaining the offered advantages in terms of accessibility, easiness of use, etc. Section 3.1 describes the structure of the manipulator and the used platform and sensors. Section 3.2 reports the prototype software structure and illustrates how it reflects the conceptual system architecture. The performances demonstrated by the prototype to reach the final target in simulation and during the tests are detailed in Section 4. It also describes the results of the surveys composed by the students, relative to their experience with the prototype. Finally, in Section 5, conclusions are drawn.

2. Concept

As described in Section 1, one of the main challenges is reducing the effort to control the manipulator in order to make it accessible even to severely impaired people. In the already mentioned work [5], the user is assisted in the manual control of the manipulator by several audio commands, which are associated to predefined movements of the manipulators. In this way, the capacity of controlling the manipulator to perform these movements is not required to the user. A similar idea is used in our previous works [8–10], where, limiting the manipulator functionalities to the pressure of a button or knocking on a door, the only task assigned to the user is communicating, through the HMI, where he wants to lead the manipulator end-effector. In these cases, the entire control of the manipulator is delegated to the system software, further reducing the user effort. To do that, the authors of [8–10] exploited the information contained in the images by a monocular camera to control the manipulator through a *Visual Servoing* algorithm.

This work exploits a similar approach, as shown in Figure 1, but it empowers the system through an *Object Recognizer*. The latter has the task of recognizing the objects to interact with inside the camera frames and to provide their coordinates expressed in pixels with respect to a reference point in the image, e.g., the camera center or a corner. Even in this case, camera frames are shown to the user on a dedicated HMI, in which the recognized objects are surrounded by a bounding box. After the user chooses one of the recognized objects through the HMI, the object camera plane coordinates are processed by the *Target Stabilizer* block. It increases the robustness on the desired object position estimation against mechanical shaking, vibrations and frame losses. To do that, the *Target Stabilizer* tracks the object area independently from the *Object Recognizer*, providing a better estimation of the 2-D coordinates. The latter are sent to the *Robot Logic*. Exploiting the coordinates of the selected object and the distance from it, obtained by the *depth sensor*, the *Robot Logic* subsystem calculates the movements that each joint of the robot should carry out to approach the target. Such information is used by the *Robot Actuators* to drive the *Robotic Arm* motors. The use of *Object Recognizer* introduces numerous advantages at the expense of increasing the computational complexity. First, as explained in Section 1, it increases the safety of the HMI toward the user. In fact, an incorrect pressure outside the bounding boxes is ignored by the system. In addition, as explained, the *Object Recognizer* limits the user actions to the selection of the desired target. Such task can be easily performed through many standard HMIs, including manual raster, raster time, a keyboard, etc. Using these interfaces is much more tricky with systems such as those in [8–10], which require the user to select the area of the objects rather than the object itself. Thus, the introduction of a system for the target recognition increases the number of usable interfaces, for a higher system accessibility.

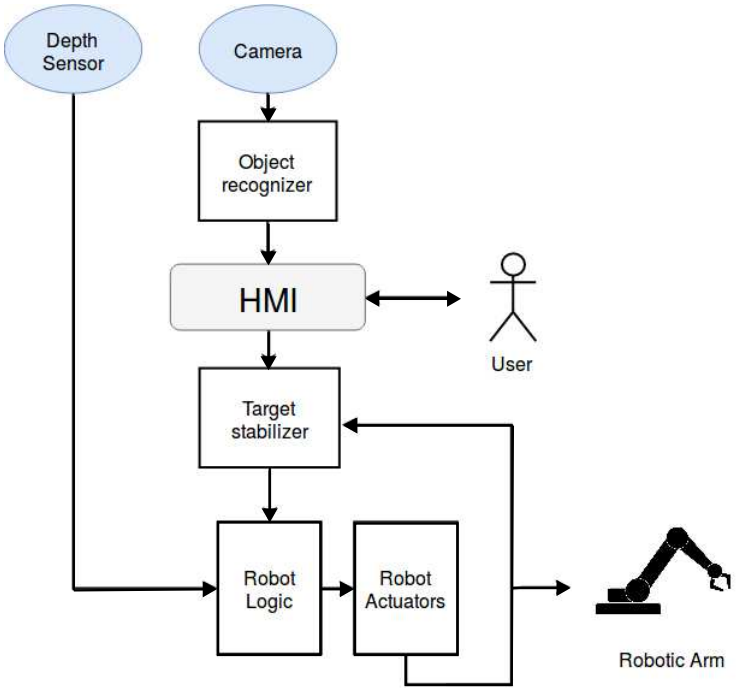

Figure 1. The Conceptual architecture. HMI, Human–Machine Interface.

3. Methods and Materials

3.1. Hardware Setup

The conceptual architecture described in Section 2 was validated through a proof of concept prototype, whose hardware setup is described in this section. Indeed, some of the subsystems shown

in Figure 1 are realized through hardware components, which are interfaced by the system software over dedicated structures of software.

As shown in Figure 2, the *Robotic Arm* was realized through a five-DOFs manipulator. The arm links are connected to each other only by means of rotary joints, whose position is established through *HS-785HB DC Servo Motors*.

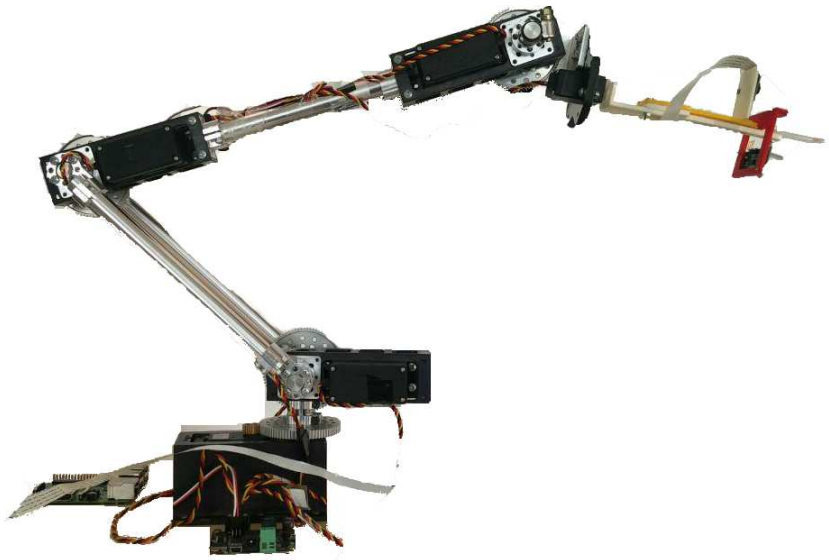

Figure 2. The prototype.

The end-effector of the arm is realized to perform only simple tasks, i.e., pressing a panel lift button or touching an object. Nevertheless, thanks to the modularity of the system software, it is possible to increase the functionalities by substituting the existent modules.

In addition to the manipulator, the other hardware components used are:

- **Raspberry PI 3B board**: This is the platform that runs the system software with the sole exception of the AI algorithm, as explained in Section 3.2. It interfaces all the other hardware components directly or through dedicated boards (e.g., the *Phydget board*, as successively explained).
- **Phydget v1.1 board**: It is a commercial motors driver board. Its use is necessary since the *Raspberry PI 3B board* is not able to provide the sufficient power to the five motors. A dedicated driver motor board allows choosing a platform for the sub-system independently by its ability to drive the motors.
- **5MP Raspberry Camera module**: The hardware part of the conceptual subsystem *Camera* is a low cost monocular camera that provides images from which some information is extracted for the control of motion in the space. The acquired images are 640×480 with 8 bits for each color (RGB).
- **HCSR04 proximity sensor**: The hardware component of *Depth Sensor* was realized through a low cost ultra-sound proximity sensor. As described in Section 2, it is necessary to estimate the distance along the axis of the camera frame between the camera and the interest object (*depth*). Due to the low cost nature of the sensor, the accuracy of the measurement is low, but it is compensated by the IBVS approach, as explained in Section 3.4.3.
- **Force sensor**: The force sensor is used to avoid breaking the end-effector or damaging the object while the manipulator grabs them.

- **Raspberry 7" touch screen display**: As described in Section 2, the *HMI* is necessary to show the user the selectable objects to interact with. In addition, it might also be exploited by users with a limited level of manual skills. Thus, a touch screen display has been used.

It is clear by knowing the used hardware the low cost nature of the manipulator. This prototype, in fact, costs less than €1500 to demonstrate that it is also possible to build a low-cost robotic arm for assistive technologies.

3.2. Software Architecture

The system software is the core of the entire system, as shown in Figure 3. In fact, it allows interfacing the user and driving the manipulator according to the conceptual architecture shown in Figure 1 by using the information provided by the sensors described in Section 3.1.

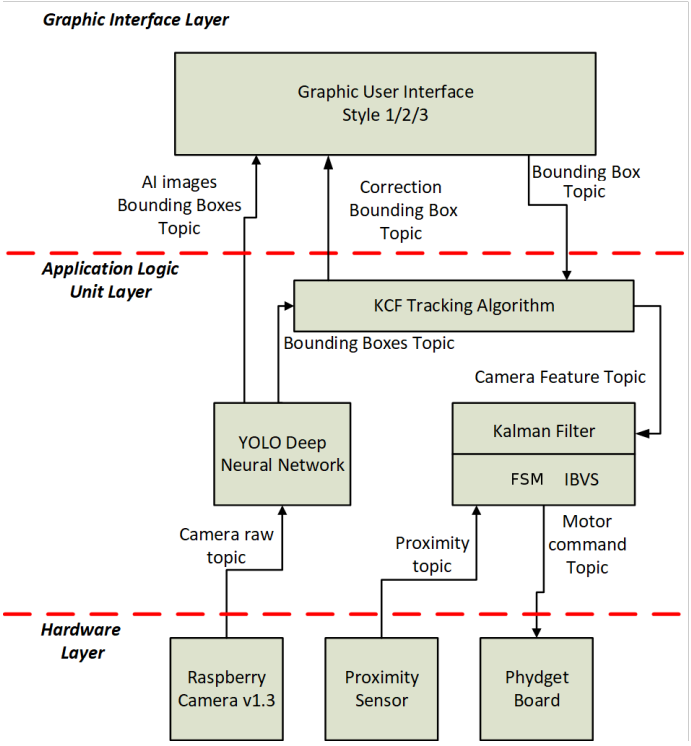

Figure 3. Layered structure of the system software.

To increase the modularity of the system, the software architecture has been divided into three different levels of abstraction:

- **Hardware layer**
- **Application Logic Unit Layer**
- **Human–Machine Interface Layer**

The different sections of the code have been implemented in order to be modular and independent from the others. In particular, each section can be substituted without changing the remaining code. Each layer communicates only with its neighbor; the *HMI Layer* is the only one that interacts with the user.

To maintain a good scalability and efficiency, ROS [12] is used. It is a collection of software to develop robotic systems. The main features of ROS are the independence from platform, the support of several programming languages and the possibility of exchanging data independently from the

physical nodes distribution. The standard topology of a ROS system is star-topology where the central node is the ROSCORE and all other nodes are the leaves attached to it. ROS offers two different communication paradigms: *topics* and *services*. *Topics* are a continuous stream of data that are received by the ROSCORE and exposed to all the interested nodes. Instead, *services* are offered by a node, and they are on-demand functions, even if the connection is handled by the ROSCORE.

To keep the desired modularity, it is necessary that every layer produces its outputs as requested by the adjacent layer, as shown in Figure 4. Details on each layer are provided in Sections 3.3–3.5 and shown in Figure 3.

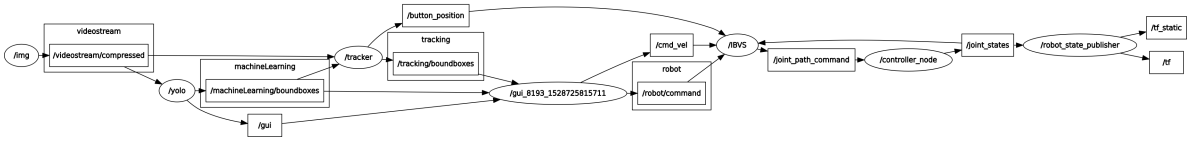

Figure 4. The topics (rectangles) and the nodes (ellipses) of the Robotic Operating System (ROS).

3.3. Hardware Layer

The *Hardware layer* is responsible for the sensors/board communications and to drive the actuators (described in Section 3.1). Each sensor is interfaced by a ROS node that collects the raw data and elaborates them to transmit more meaningful data to the upper layer. The nodes inside the *Hardware layer* section are:

- **Camera node**: The camera node handles all the frames taken by the Raspberry Camera Module and it represents the software part of the *Camera* conceptual block. In addition to the data acquisition, the *Camera Node* modifies the resolution and the brightness of the images to provide better frame quality. To do that, a topic is exposed. To reduce the latency and increase the fluidity of the stream of frame, the camera node performs an image compression by reducing the quality of the original image through an 8-bit truncation of the colors, i.e. the images stream are converted to JPEG and encapsulated inside a ROS standard message to be exposed in a topic.
- **Proximity node**: The *Proximity node* collects the data provided by the proximity sensor and by data processing to evaluate the distance from the desired object. Since the data provided represent the time to go and return of the ultrasound signal, the measured distance can be calculated simply multiplying then times the sound speed divided by 2. The distance measurements are then transmitted as a message on a dedicated exposed topic. The *Proximity node* together with the hardware proximity sensor realize the *Depth Sensor* conceptual subsystem.
- **Controller Motor Node**: The *Raspberry* board is not able to provide the power for five motors; thus, the *Phydget* board is used. However, the latter cannot interpret the data provided by *The Application Processor*, which sends the position of joints as angles. In this layer, every position is converted into the format required by the *Phidget API* without changing the upper layer output data. The *Controller Motor Node*, together with the *Phydget* board, implement the *Robotic Actuators* conceptual subsystem.

3.4. Application Logic Unit Layer

The *Application Logic Unit Layer* controls the whole system by interfacing the user through the *Human–Machine Interface Layer* and by using the data provided by the *Hardware layer*. Indeed, it exploits the image data collected by the *Camera Node*, contained in the *Hardware layer*, to feed an Artificial Intelligence (AI) algorithm based on *You Only Look Once (YOLO) v2* convolutional neural network. The latter realizes the conceptual *Object Recognizer*, as better explained in Section 3.4.1. In fact, the *YOLOv2* network recognizes multiple different objects in the image and estimates their positions over different frames.

The acquired images and the relevant bounding boxes are shown to the user through the *Human–Machine Interface Layer*, which also collects the user commands. When one of the recognized objects is selected, the *Human–Machine Interface Layer* sends the correspondent bounding box corner coordinates to the *Target Stabilizer*. It is realized through the *Kernelized Correlation Filter (KCF)* algorithm, described in Section 3.4.2. The KCF algorithm is used to improve the object position estimation over consecutive frames. The low quality camera produces a lot of noise which is bad interpreted by the network or the *KCF*. To keep the cost of the entire system low, a combination of the two algorithms is used. In fact, when the user selects a target, the two algorithms start communicating each other to improve the location of the target during the movement of the manipulator.

The improved object coordinates are exploited by the *Robot Movement Control System*, which implements the conceptual *Robot Logic* unit. The *Robot Movement Control* algorithm uses the information obtained by the *KCF* algorithm as target point for the arm end-effector. To do that, it implements a *Finite State Machine (FSM)*, described in Section 3.4.3, which drives the manipulator to the target position by performing an established sequence of movements. To guarantee the arrival to the right position at every step, the *Robot Movement Control* block exploits a close feedback loop algorithm called *Image Based Visual Servoing (IBVS)*, which permits to drive a manipulator exploiting a 2D algorithm. It is better described in Section 3.4.2.

The following sections report detailed descriptions of the *YOLO Deep Neural Network* (Section 3.4.1), the *Kernelized Correlation Filter* (Section 3.4.2) and the *Robot Movement Control System* (Section 3.4.3).

3.4.1. YOLO Deep Neural Network

In this paper, we use the *YOLOv2* convolutional neural network to realize the conceptual *Object Recognizer*, which permits identifying multiple objects and tracking them over different frames. *YOLOv2* belongs to a broader class of algorithms note in literature as *Artificial Intelligence* algorithms. The latter are used to extract particular features from images to trace an area of interest on different frames [13].

Generally, the most common AI approach for image recognition are based on the use of *Convolutional Neural Networks* (CNNs) followed by a *Fully-Connected Feed-forward Neural Network* (FCFNN).

CNNs are networks used to extract features from images by performing the convolution among some trainable filter kernels, with fixed geometric sizes, and the given image. The outputs of a CNN layer are K features maps, where K is the number of the filters in that layer. The dimension of the output depends on the image size, the stride value and the size of the filter.

The FCFNN is another topology of neural network composed by several layers, containing neurons. Each neuron of a layer is connected with all those that belong to the next one. A FCFNN is mainly used to take the final decisions or to assign a specific label to the output. Furthermore, the neurons execute an activation function over the sum of the linear combination of the inputs. For multi-class classification problems, the activation function of FCFNN layers is generally a *ReLU* [14]. The *Softmax* layer is generally used to obtain the probability for each class of the output of the network. This probability is estimated as described by Equation (1):

$$p(j) = \frac{e^{o(j)}}{\sum_{i=0}^{N-1} e^{o(i)}} \quad (1)$$

where $p(j)$ is the probability for the input to contain the class j and $o(j)$ is the jth element of the output layer.

Each layer needs a training phase to tune its weights in order to classify the input features. This is made by a supervised approach, which uses a set of ordinate input/output couples, called *training set*. The network training is performed by executing some algorithms as *Descendent Gradient* [15] or

Back Propagation [16]. Such algorithms compare the training set outputs with the one generated by the network with the correspondent inputs. The comparison produces an error calculated by using a particular *loss function*. The error value is used to update the weights at each step.

For our aims, the AI algorithm shall provide the coordinates of the recognized objects in the camera frame. To do that, different approaches exist. *R-CNNs* [17] offer the best accuracy on the positioning and the recognition of the objects. The major problem of *R-CNN* is the necessary depth of the network. In fact, to reach high accuracy, deep networks are necessary with reduction of the frame rate [18]. To overcome the problem of the low number of frame rates, there are some solutions: the *fast R-CNNs* [19] and *faster R-CNNs* [20]. Despite the improvement offered by the networks, the performances provided are not sufficient for our purposes, because the neural network shall act as objects recognizer and tracker. Indeed, the *faster R-CNNs*, which reach 17 Frames Per Second (FPS), also do not allow controlling the manipulator in a smooth way, introducing some non-predictable movements that affect the functionality of the *IBVS* and the *Kalman* filter. We also considered the possibility of a machine learning approach, as described in our previous papers [8,9]. Nevertheless, the execution time required by algorithms such as *SIFT* [21] on *Raspberry PI 3* is higher than a CNN approach. For the same reason, some other approaches based on CNNs, such as *key-points* network [22], are not feasible. Indeed, they require more computational power owing to the complexity of the network. Furthermore, the difficulties to reconstruct some objects shape, such as a lift button, starting from their key-points are higher than using a simpler bounding box approach.

In view of that, we decide to choose the YOLOv2 instead to introduce some latency in the movements that would annoy the users. *YOLOv2* network [23] is a CNN made up by 24 convolutional layers, followed by two fully connected layers.

This network is inspired by *GoogLeNet* but has some differences that make it lighter and more efficient. The architecture of *YOLOv2* is shown in Figure 5 [24].

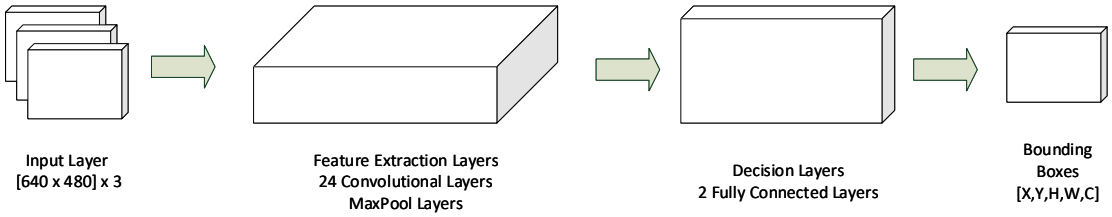

Figure 5. The YOLO network architecture.

The first convolutional layer performs a downsampling of the input images; the other convolutional layers are used to perform features extraction. The fully connected layers, instead, make the real decision about the bounding boxes and the found objects.

Each frame is divided into a grid of $S_x \cdot S_y$; in each grid cell, a maximum number of B bounding boxes are searched; for every bounding box, a certain confidence score is assigned. The confidence score is the probability that inside a bounding box an object is contained and how accurate the decision is. This confidence is expressed as $p(Object)$.

Thus, each bounding box consists in five values: X, Y, W, H and, $Confidence$. Furthermore, for each grid cell, the probability of the C conditional class is computed. In this way, it is possible to obtain for the entire image N bounding boxes, divided into grids. This leads to:

$$p(Class_i|Object) \cdot p(Object) \cdot IOU_{pred}^{truth} = p(Class_i) \cdot IOU_{pred}^{truth} \quad (2)$$

for a total of $S_x \cdot S_y \cdot (B \cdot 5 + C)$ numbers of tensors for iteration, where:

- $p(Class_i|Object)$: Given a grid cell with at least one object, it represents the conditional probability for one of them to belong to the ith class.

- $p(Object)\cdot IOU_{pred}^{truth}$: It represents the confidence of each bounding box inside a cell grid.

In each cell, only the classes with the highest $Confidence$ are taken.
For the training phase, the network exploits a multi part loss function (see Equation (3)).

$$\begin{aligned} Loss = \lambda_{coord}\cdot & \sum_{i=0}^{S_x\cdot S_y}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(x_i-\hat{x}_i)^2+(y_i-\hat{y}_i)^2 \\ +\lambda_{coord}\cdot & \sum_{i=0}^{S_x\cdot S_y}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(\sqrt{w_i}-\sqrt{\hat{w}_i})^2+(\sqrt{h_i}-\sqrt{\hat{h}_i})^2 \\ + & \sum_{i=0}^{S_x\cdot S_y}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2+\lambda_{noobj}\sum_{i=0}^{S_x\cdot S_y}\sum_{j=0}^{B}\mathbb{1}_{ij}^{obj}(C_i-\hat{C}_i)^2 \\ & + \sum_{i=0}^{S_x\cdot S_y}\mathbb{1}_{i}^{obj}\sum_{c\in classes}(p_i(c)-\hat{p}_i(c))^2 \end{aligned} \tag{3}$$

where $\mathbb{1}_{ij}^{obj}$ denotes the jth bounding box responsible of the prediction of the object in the ith cell, $\mathbb{1}_{i}^{obj}$ indicates if an object appears in the ith cell, λ_{coord} is a constant used for the cell with some objects in it, and λ_{noobj} is a constant used for the cell without objects in it.

For our project, the network was retrained to allow the recognition of buttons of the lifts panels. To reduce the training time, we started with the weights of the *Darknet* and we executed a *fine-tuning* on the *YOLOv2*, by changing the weights of the last two layers. This procedure allows maintaining the same feature extraction layers, retraining only the decision part. In addition, the time required for a fine-tuning is less than the one for a full training. The *training-set* was composed by 23,000 images; the training was represented by 70% of the images, while the validation and test were represented by 15% each. The dataset was created using, respectively, 57% and 43% of doors and lift buttons images. Such pictures were taken from the *ImageNet* dataset [25]. Owing to the low number of lift buttons present on the dataset, some additional images were used, which were labeled by using the *COCO* annotation [26]. In view of the reduced number of training data and of the noise introduced by the low cost camera, the *mAP* was 45% during the working conditions.

The *YOLOv2* network used in this project is written in *Keras* [27], a wrapper of *TensorFlow*. The latter one, which acts as a back-end, is an open source software framework for high performance numerical computation. The combination of *Keras* and *TensorFlow* permits an easy implementation and a simple management of complex neural network architectures. Thus, *Keras* is a good abstraction from the back-end, though using the native *TensorFlow* functions is possible.

3.4.2. Kernelized Correlation Filter Algorithm

KCF is one of the faster tracking algorithm present in the literature [28]. Its power derives by the combination of the *motion model* with the *appearance model*. This combination permits identifying the object among different frames without losing the high performance given by the *YOLOv2* neural network. Indeed, in this work, a combination of the two algorithms is used r to obtain a single tracking algorithm that exploits the information taken from the *YOLOv2* and *KCF*. The *YOLOv2* recognizes objects, as already explained in Section 3.4.1, and sends the position of the selected bounding box corner coordinates to *KCF*. This information is exploited to calculate the (X,Y) coordinates in the camera plane with respect to object center of gravity. After that, a $10px \times 10px$ square is created around this point, and its area is tracked by the *KCF*.

To this aim, it extracts data about the shape and color of the object, and it reuses them to track the object in the next frames.

Nevertheless, *KCF* suffers from the *occlusion* problem: if a part of the tracked area is covered during the motion, *KCF* loses it, and it is not able to recognize it even after it becomes visible again. This leads to a total loss of the interested area after several frames.

Thus, the AI re-sends the position of the object after a predefined quantity of frames, in order to calibrate the *KCF* algorithm. This technique permits to increase the frames rate and to improve the target tracking to avoid its loss during the movement.

KCF was implemented by using the same approach used in the version of *OpenCV*.

3.4.3. Robot Movement Control System

The *Robot Movement Control System* is responsible for controlling the movement of the robotic arm end-effector to reach the desired object position. The information on the latter is inherited by the *KCF*, which tracks it frame by frame.

The arm movement is managed through the *FSM* shown in Figure 6. The *FSM* states are:

- **IDLE**: Before receiving a user command, the system remains in the *IDLE* state. It corresponds to the initial position of the robotic arm. When a new command is received—i.e., a new target position—the system transits to the *XY Kalman* state.
- **XY Kalman**: In this state, the robotic arm is forced to align the target position (X,Y) with respect a predefined point (K_x, K_y), called *Camera Features*. A square of side of 25×25 pixels is defined around the target position. When the difference between the (X,Y) coordinates of the object and (K_x, K_y) coordinates of the fixed point is enclosed in the square, the end-effector is considered aligned. Thus, the system transits to the *APPROACHING Kalman* state.
- **APPROACHING Kalman**: In this state, the robotic arm approaches the object by moving along a direction orthogonal to the camera plane axis. If the movement leads to a significant misalignment on the (X,Y) plane (greater than 35 pixels), the systems transits back to the *XY Kalman* state. The information provided by the *Proximity Node* is used to monitor the distance between the object and the end-effector. When the end-effector reaches a fixed distance, the system transits to the *STOP* state.
- **STOP**: The *STOP* state is used by the system to perform simple movements that depend on the task required. For instance, in the case of a pressure of a key, the manipulator touches the button. After the completion of the task, the system comes back to the *IDLE* state.

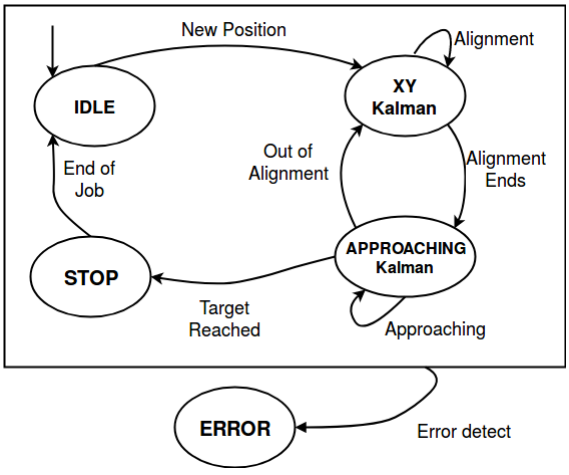

Figure 6. Finite State Machine for the control of the robotic arm end-effector movement.

In the states *APPROACHING Kalman* and *XY Kalman*, a *Kalman filter* is used to make the object's position estimation more robust by disturbance such as vibrations induced by the arm movements.

3.4.4. Image Base Visual Servoing

To control the end-effector movements, the *IBVS* algorithm is used. The latter is a technique to realize a closed-loop feedback control by comparing the center of gravity of the chosen object with a point (*Camera features*) in the camera frame [29], whose coordinates are called as Equation (4).

$$K_x, K_y \tag{4}$$

IBVS approaches may use more reference points to avoid rotation around the axis passing trough the camera plane. Depending on the number of camera features used, an *Interaction Matrix* (J_{IM}) is defined. For a single camera features, it is made as shown in Equation (5):

$$J_{IM} = \begin{bmatrix} \frac{-f_x}{depth} & 0 & \frac{u}{depth} & \frac{u \cdot v}{f_x} & -(f_x + \frac{u \cdot u}{f_x}) & v \\ 0 & \frac{-f_y}{depth} & \frac{v}{depth} & (f_y + \frac{v \cdot v}{f_y}) & \frac{-u \cdot v}{f_y} & -u \end{bmatrix} \tag{5}$$

where f_x is the focal length for x axis of the camera; f_y is the focal length for y axis of the camera; u is the coordinate of the center of gravity of the object on x axis; v is the coordinate of the center of gravity of the object on y axis; and $depth$ is the distance taken from the depth sensor.

Thus, the coordinates expressed as pixels and the distance from the target are taken by the *IBVS* to calculate the difference between them and the camera features position, computing the error estimation ($\epsilon_{x,y}$). The last one is multiplied by a pseudo-inverse of the *Interaction Matrix* (J_{IM}^{-1}) to calculate the effective translational and rotational velocity. It is worth notiing how the algorithm also takes in to account the depth between the end-effector and the target to compute the difference of prospective, which permits to control a robot using a 2D camera.

The procedure to compute such error estimation, the pseudo-inverse of the *Interaction Matrix* and the end-effector velocity are described in Equation (6).

$$\begin{aligned} \epsilon_{x,y} &= \begin{bmatrix} u \\ v \end{bmatrix} - \begin{bmatrix} K_x \\ K_y \end{bmatrix} \\ J_{IM}^{-1} &= J_{IM}^T \cdot (J_{IM} \cdot J_{IM}^T)^{-1} \\ \vec{V} &= J_{IM}^{-1} \cdot \epsilon_{x,y} \end{aligned} \tag{6}$$

Thanks to them, it is possible to use the standard kinematic equation to compute the next step as:

$$\dot{q} = J^T \cdot (J \cdot J^T)^{-1} \tag{7}$$

$\dot{q}$ is proportional to the increment that the rotative joints shall perform to reach the new position. Thus, once $\dot{q}$ is calculated, it is used for deriving the value of the new joint position $q[n+1]$, as expressed in Equation (8). $q[n+1]$ is provided to the *Phydget*, included in the *Hardware Layer*, which directly drives the motors.

$$q[n+1] = q[n] + \dot{q} \cdot \Delta T \tag{8}$$

ΔT is the time necessary to the *Application Logic Unit Layer* to process the relevant image frame from its acquisition to the end of the loop. This time is used to perform all the derivation needed to compute the path.

The approaching phase is handled by the *IBVS* by moving the end-effector toward the target. This phase is guided by the proximity sensor that avoid the collision of the end-effector with the

target. Using the depth sensor within the above algorithm allows you to drive a real manipulator that performs movements in three dimensions, using the algorithms KCF and AI working on two dimensions. Obviously, it is necessary that the field of action of the robot is free of obstacles, otherwise the robot enters in error mode, and waits until a hard restart.

3.5. Human–Machine Interface Layer

To provide an efficient support to people with disabilities, it is important to implement a *HMI* that minimizes the effort required for its control. This is carried out through an essential *HMI* that offers only the fundamental information and the main functionalities to reduce the possibility to make mistakes.

In our work, three *HMIs* styles are implemented. Each of them is conceived to help people with different severity of disability and so diverse manual skills:

- Touch Screen
- Manual Raster Scanning (Two Sensors)
- Time Raster Scanning (One Sensor)

The *Touching* interface, shown in Figure 7, is widely used in smart-phones and touch screen computers. This HMI is addressed to the people that have good manuals skills.

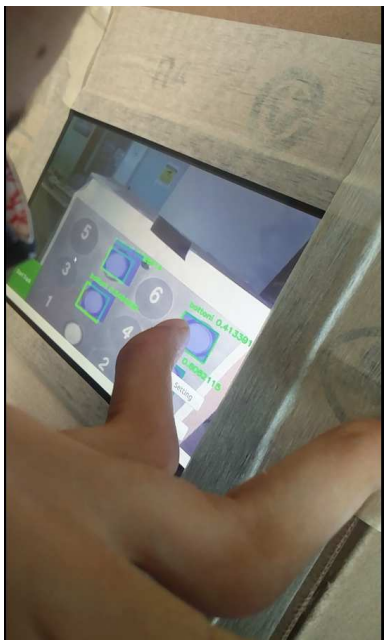

Figure 7. Touching interface tested by a user.

Our idea is to use the bounding boxes offered by the AI to surround the objects, which are possible targets for the end-user. In this way, the possibility of errors is limited to bounding boxes. Even in the case of hand tremors, this approach improves the success rate.

The *Scanning* interface is targeted to people who cannot use the touch screen but are able to control two sensors of any kind among the many available on the market for the different severity of disability. Our system provides a 3.5 mm mini jack, which is compatible with most of the sensors available on the market (buttons, grasp, pedal, etc.). The basic idea is to use two buttons, which are used to select and confirm the target object, surrounded by a bounding box. The person chooses the desired object by pressing the selection button in a scan mode, highlighting the next bounding box. When the object has been selected, the user clicks on the confirmation button for sending the target position to lower layer.

The *Time Clustering* is the most minimal interface, because it is aimed at people with severely disabilities who are able to control a sensor. The selection of the object, in this case, is performed by an automatic process that scans all the bounding boxes after a programmable time. When the desired bounding box is highlighted, the subject pushes the confirmation button to select the target.

4. Results

The prototype was characterized both for its accuracy in interacting with the required object (i.e., reaching the desired panel lift button and pressing it) and for the ability of the HMIs to provide good user experiences. Section 4.1 provide details about the test procedure and the obtained results for the accuracy investigation; Section 4.2 describes the outcome of the preliminary tests with the end-users.

4.1. System Accuracy Tests

The accuracy of the system was tested by calculating the ability of the manipulator to press the lift button chosen by the user. At the same time, to verify the goodness of the whole algorithm to reach the target regardless of the non-idealities introduced by the low cost nature of the prototype, the entire system was simulated using *RViz* simulator, as shown in Figure 8.

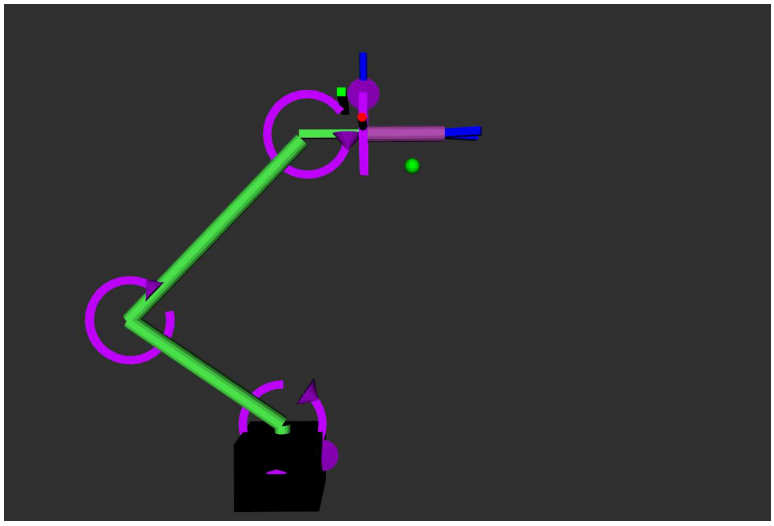

Figure 8. The manipulator structure shown in RViz simulation environment.

The simulation ran on a personal computer concurrently with the real prototype tests, by using the same algorithms. In the simulation environment, a *virtualized camera* was exploited: when an object was recognized by the AI algorithm in the frames captured by the real camera prototype, the coordinates of the found object were used in the simulation. Owing to that, only the camera non-idealities were considered in the RViz simulation, which, instead, did not consider the mechanical shaking effects. Moreover, the tests and the simulation were performed with and without the implementation of the *Target stabilizer* to test its effectiveness.

Table 1 sums up the results of this investigation. The accuracy was calculated as the success rate over 50 iterations of the experiment.

Simulations showed the manipulator can reach the target with an average distance between the object center of gravity and the arrival point of 4.5 mm and an average time of 25 s. Such time was also considered as the maximum time available in the real tests for the manipulator to reach the target for that attempt to be considered successful. Fixing a limit was necessary because in some real cases the manipulator was able to reach the desired button only after a huge quantity of time from the user request, being stuck in the proximity of the target.

Table 1. Percentage accuracy of the system over 50 experiment iterations.

Tracking Algorithms	RViz sim. Accuracy	Prototype Accuracy
Only *YOLOv2*	94%	43%
YOLOv2 + KCF	100%	78%

The results in Table 1 show a gap between the simulation and prototype tests result. It is due to the presence of mechanical non-idealities that were not considered in the simulation, which reached 100% in accuracy when the *KCF* algorithm was used. The advantages offered by the latter were proved by the sensible accuracy drop, which was verified in the prototype tests when *KCF* was not used, increasing the manipulator sensitivity to mechanical vibrations and other effects.

4.2. Preliminary Tests with the End-Users

The prototype was tested by 20 students of the University of Pisa with motor skills disabilities, who might be the target end-users of this system. The aim of the investigation was to determine the effectiveness of the HMIs explained in Section 3.5. The tests were done by dividing the students into three main groups. Each group was characterized by different motors skills and, consequently, by a different HMI. The groups were:

- Touch (High motor-skills) (12 people)
- Double sensors (Medium motor-skills) (6 people)
- Single sensor (Low motor-skills) (2 people)

Each group had the same task to complete (pressing a predefined elevator button) in the same amount of time. In this way, it was possible to estimate the performance of each HMI with respect to the level of disability. The test was divided into training and validation phases. The first phase was to explain how to control the robotic arm, showing them how the interface works and executing some tests with the users. The second phase was the pressure of the elevator button from a lift panel of nine buttons. This test showed how the users were able to control the entire system in only few minutes, reducing the training phase and their frustration. Furthermore, the time for all the categories was respected even if the disability level and manual skills were different. At the end of the tests, each student completed a survey relative to the interface used. The survey was made by eight questions. For each question, a value from 0 to 10 was given. The questions were on the usability (two questions), prototype reaction/performance (two questions), propensity to use of the system (one question), HMI usability (two questions), and satisfaction with the functions offered (one question).

The results were optimistic obtaining a mean of 8.13/10. Because of these good performances, there will be the real possibility of embedding the prototype on a power-wheelchair. Furthermore, the users suggested some nice to have functionalities, such as:

- Grabbing objects, such as bottles or glasses
- Picking up objects fallen on floor

5. Conclusions

This work proposes a low cost autonomous manipulator realizing simple tasks, such as pressing a button lift, and implementing a low effort HMIs. The latter were empowered thanks to the use of an AI algorithm based on *YOLOv2* convolutional neural network, which recognizes and extracts coordinates of a possible target objects. A custom closed feedback loop algorithm exploits the information on the object coordinates to automatize the execution of the task and to control the robot. Such algorithm uses the *KCF* as additional tracking algorithm in combination with the *YOLOv2* network. This approach minimizes the probability of losing the objects tracks and increases the immunity of the robot system

to the non-idealities of the low cost camera and of the mechanics. A *FSM* establishes a procedure composed by fixed steps of movements, each of them is realized through the *IBVS* algorithm that allows exploiting 2D algorithm to drive a real manipulator. The usage of a *Kalman filter* further make the system more robust against the mechanical vibrations.

Such approach, formalized through a conceptual architecture in Section 2, was used to implement a low cost prototype featuring three versions of HMIs for people with different grade of disability. The prototype hardware and software architectures are described in Sections 3.1 and 3.2.

Finally, a test procedure was realized aiming to characterize the prototype ability in performing the required tasks. The evaluation was performed by measuring the success rate in touching the correct panel lift button. The system was also simulated by excluding the mechanical non-idealities inside the RViz environment. The accuracy on 50 iterations was 78% with the prototype and 100% in simulation.

The HMIs evaluation was performed through preliminary tests with the end-users, who tried to control the robot by using the HMI dedicated to them. All interfaces were appreciated by the users, who were able to execute the task in the given time, regardless their manual-skills. By limiting the number of tasks executable, the proposed solution offers an example of automatic manipulator, which succeeds in reducing the user effort through the complete automatism in the object interaction. Thanks to the use of an AI algorithm, which tracks the objects in the image frames and limits the user task to the only selection of the target. The system results to be less prone to the user error, increasing the safety of the HMI itself. In addition, it also improves the modularity of the system, allowing the use of the best HMI for the given user ability.

Author Contributions: Conceptualization, methodology, resources, data curation and software, G.G.; Validation, writing—original draft preparation, writing—review and editing, G.G. and G.M.; Formal analysis, G.M.; Visualization, G.M., G.G. and L.F.; Supervision, L.F. and G.M.; Project administration, G.G., G.M. and L.F.; and Funding acquisition, L.F.

Funding: This work was realized in the contest of the RIMEDIO (Il braccio Robotico Intelligente per Migliorare l'autonomia delle pErsone con DIsabilità mOtoria") project, supported by Fondazione Cassa di Risparmio di Lucca.

Acknowledgments: We also want to thank all the students of the University of Pisa who helped us to test and to evaluate the manipulator.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AI	Artificial Intelligence
API	Application Programming Interface
CNN	Convolutional Neural Network
DoF	Degrees of Freedom
FCFNN	Fully Connected Feed-forward Neural Network
FSM	Finite State Machine
FPS	Frame Per Second
HMI	Human–Machine Interface
IBVS	Image Based Visual Servoing
KCF	Kernelized Correlation Filter
mAP	mean Average Precision
ML	Machine Learning
R-CNN	Region of Interest Convolutional Neural Network
ReLU	Rectified Linear Units
ROS	Robotic Operating System
YOLO	You Only Look Once

References

1. Campeau-Lecours, A.; Lamontagne, H.; Latour, S.; Fauteux, P.; Maheu, V.; Boucher, F.; Deguire, C.; Caron L'Ecuyer, L.J.C. Kinova Modular Robot Arms for Service Robotics Applications. *Int. J. Rob. Appl. Technol.* **2017**, *5*, 49–71. doi:10.4018/IJRAT.2017070104. [CrossRef]
2. Dynamics, E. iARM. 2018. Available online: http://www.exactdynamics.nl/site/?page=iarm (accessed on 30 May 2019).
3. Tang, J.; Zhou, Z.; Yu, Y. A Hybrid Computer Interface for Robot Arm Control. In Proceedings of the 2016 8th International Conference on Information Technology in Medicine and Education (ITME), Fuzhou, China, 23–25 December 2016; pp. 365–369.
4. Arrichiello, F.; Di Lillo, P.; Di Vito, D.; Antonelli, G.; Chiaverini, S. Assistive robot operated via P300-based brain computer interface. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 6032–6037.
5. Ka, H.W.; Chung, C.S.; Ding, D.; James, K.; Cooper, R. Performance evaluation of 3D vision-based semi-autonomous control method for assistive robotic manipulator. *Disabil. Rehabil. Assist. Technol.* **2018**, *13*, 140–145. [CrossRef] [PubMed]
6. Tolba, H.; El_Torgoman, A.S. Towards the improvement of automatic recognition of dysarthric speech. In Proceedings of the 2009 2nd IEEE International Conference on Computer Science and Information Technology, Beijing, China, 8–11 August 2009; pp. 277–281.
7. Raghavendra, P.; Rosengren, E.; Hunnicutt, S. An investigation of different degrees of dysarthric speech as input to speaker-adaptive and speaker-dependent recognition systems. *Augmentative Altern. Commun.* **2001**, *17*, 265–275. [CrossRef]
8. Palla, A.; Frigerio, A.; Meoni, G.; Fanucci, L. Object Detection and Spatial Coordinates Extraction Using a Monocular Camera for a Wheelchair Mounted Robotic Arm. In Proceedings of the International Conference on Smart Objects and Technologies for Social Good, Venice, Italy, 30 November–1 December, 2016; pp. 224–232.
9. Palla, A.; Meoni, G.; Fanucci, L.; Frigerio, A. Position Based Visual Servoing control of a Wheelchair Mounter Robotic Arm using Parallel Tracking and Mapping of task objects. *EAI Endorsed Trans. Ambient Syst.* **2017**, *17*, e1. doi:10.4108/eai.17-5-2017.152545. [CrossRef]
10. Palla, A.; Sarti, L.; Frigerio, A.; Fanucci, L. Embedded implementation of an eye-in-hand visual servoing control for a wheelchair mounted robotic arm. In Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC), Messina, Italy, 27–30 June 2016; pp. 274–277.
11. Rabhi, Y.; Mrabet, M.; Fnaiech, F. Intelligent control wheelchair using a new visual joystick. *J. Healthcare Eng.* **2018**, *2018*. [CrossRef] [PubMed]
12. Quigley, M.; Conley, K.; Gerkey, B.P.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A.Y. ROS: An open-source Robot Operating System. ICRA Workshop on Open Source Software. 2009. Available online: https://www.ros.org/ (accessed on 30 May 2019).
13. Lowe, D.G. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision* **2004**, *60*, 91–110. [CrossRef]
14. Dahl, G.E.; Sainath, T.N.; Hinton, G.E. Improving deep neural networks for LVCSR using rectified linear units and dropout. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 8609–8613. [CrossRef]
15. Hager, W.W.; Zhang, H. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM J. Optim.* **2005**, *16*, 170–192. [CrossRef]
16. Sathyanarayana, S. A gentle introduction to backpropagation. *ACM Digital Library* **2014**, *7*, 1–15.
17. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich feature hierarchies for accurate object detection and semantic segmentation. In Proceedings of the the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 24–27 June 2014; pp. 580–587.
18. Goodfellow, I.; Bengio, Y.; Courville, A. Deep Learning. 2016. Available online: http://www.deeplearningbook.org (accessed on 30 May 2019).
19. Girshick, R. Fast r-cnn. In Proceedings of the The IEEE International Conference on Computer Vision (ICCV), Santiago, Chile, 13–16 December 2015; pp. 1440–1448.

20. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In Proceedings of the Advances in Neural Information Processing Systems 28 (NIPS 2015), Montreal, QC, Canada, 7–12 December 2015; pp. 91–99.
21. Lowe, D.G. Object recognition from local scale-invariant features. In Proceedings of the ICCV 1999, Kerkyra, Corfu, Greece, 20–25 September 1999; pp. 1150–1157.
22. Peng, X.; Feris, R.S.; Wang, X.; Metaxas, D.N. A recurrent encoder-decoder network for sequential face alignment. In Proceedings of the European Conference on Computer Vision, Amsterdam, The Netherlands, 11–14 October 2016; pp. 38–56.
23. Redmon, J.; Farhadi, A. YOLO9000: Better, faster, stronger. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 7263–7271.
24. Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. You only look once: Unified, real-time object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 779–788.
25. Deng, J.; Dong, W.; Socher, R.; Li, L.J.; Li, K.; Fei-Fei, L. ImageNet: A Large-Scale Hierarchical Image Database. In Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition, Miami Beach, FL, USA, 22–24 June 2009; pp. 248–255.
26. Lin, T.Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C.L. Microsoft coco: Common objects in context. In Proceedings of the European Conference on Computer Vision, Zurich, Switzerland, 6–12 September 2014; pp. 740–755.
27. Chollet, F. Keras: The Python Deep Learning library. Available online: https://keras.io (accessed on 30 May 2019).
28. Henriques, J.F.; Caseiro, R.; Martins, P.; Batista, J. High-speed tracking with kernelized correlation filters. *IEEE Trans. Pattern Anal. Mach. Intell.* **2014**, *37*, 583–596 [CrossRef] [PubMed]
29. Siciliano, B.; Sciavicco, L.; Villani, L.; Oriolo, G. *Robotics: Modelling, Planning and Control*; Springer Science & Business Media: Berlin, Germany, 2010.