# Deep Learning based end-to-end Grasping Pipeline on a lowcost 5-DOF Robotic arm

Pranay Junare, Mihir Deshmukh, Mihir Kulkarni, Prashant Bartakke
*Dept. of Electronics and Telecommunication Engineering*
College of Engineering Pune, Pune, India - 411005
Email: junarepg18.extc@coep.ac.in, deshmukhmp18.extc@coep.ac.in, kulkarnimm18.extc@coep.ac.in, ppb.extc@coep.ac.in

2022 IEEE 19th India Council International Conference (INDICON) | 978-1-6654-7350-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/INDICON56171.2022.10040180

***Abstract*—The problem of robotic grasping is still an unsolved problem with many approaches trying to generalize grasp predictions for unseen and dynamic environments. In this paper, we propose a complete end-to-end pipeline for the task of Deep Learning based robotic grasping on a lowcost 5-DOF arm. We explore Transfer learning approach and then train our grasping model from end-to-end. In the transfer learning approach we tried 2 base models, VGG-16 and ResNet-50. Our grasping model when ResNet-50 is used as base architecture provided better results with a testing accuracy of 83.3% while VGG-16 provided an accuracy of 78.2%. In order to test our model on a real robotic arm, we built a 5-DOF arm and added a custom parallel plate gripper. Complete ROS and Moveit support is added to our developed robotic arm. The processed RG-D image from the KinectV2 camera is given as an input to the model which predicts the 5-D grasp configuration. Required electronic system design and its PCB is built which controls the robotic arm. The predicted 5-D grasp configuration is then transformed to the object pose w.r.t the base link frame of the robot. Lastly, a ROS node is written that automates the task of picking objects lying in different positions & orientations and sends the joint angle values over pyserial communication to the microcontroller's PCB.**

***Index Terms*—Robotic Grasping, Deep Learning, CNN, Grasping Pipeline, Grasp detection**

## I. Introduction

We humans are capable of recognising distinct items and determining how to pick them up quite instantly. Robotic capabilities are far behind. For decades, researchers have attempted to develop cognitive robots with the same level of handiness as humans. Despite research and business interest, robotic grasping still remains an unsolved problem. This is due to inaccurate potential grasp recognition, which hinders the selection of the ideal grasp for a given object. Many recent studies have addressed this problem by recasting it as a detection problem that utilizes visual components of the image to decide where the robotic gripper should be positioned. Recent research has focused on the application of deep learning approaches to improve performance in the areas of visual perception, Autonomous vehicles, audio recognition, personalized product recommendation and natural language processing, opening up new possibilities in the field.

So the key difficulty where we may use deep learning methods is to design robots which can operate in unknown, dynamic and ever-changing environments (e.g. cluttered environments, household environments, bin-picking, healthcare facilities, etc). Deep learning-based algorithms are used in robotic grasping to perform the task automatically without the need for human intervention. These methods are categorised primarily by the kind of learning methods utilised, such as Supervised learning or Reinforcement learning as put forth in [1].

Typical pipelines for Deep learning-based robotic grasping approaches usually contain determination of the accurate object pose, selection of suitable grasp pose on the test object, path planning and lastly execution of complete grasping task. When models are trained in simulation, sim-to-real techniques are required for implement the model in real-world, so the model can perform up to par in the real world. Also, approaches can use both RGB as well as RGB-D data to estimate the grasp pose.

## II. Literature Review

In [2], the authors aim to Predict a 5-D grasp configuration using a Lightweight CNN model. They have trained on the Cornell Grasping dataset. SqueezeNet-RCM model is directly applied on the RGB-D image while using a symmetric parallel gripper. Grasp configuration for a 3 finger gripper using point cloud data is determined in [3] and Object point cloud segmentation is done using a variant of PointNet architecture. Also, a 4-dimensional grasping configuration is used in this research which is not as common as a 5-dimensional one.

In [4], the authors aim to improve grasping using a self-supervised approach. The network is corrected in 2 ways: Obtaining new possible grasping points & Pruning incorrect grasping points predicted candidates. Research in [5] outperforms the current state of the art techniques both in terms of accuracy and as well as speed. The accuracies achieved by them are: 89.21% image-wise split and 88.96% object-wise split. Authors have used two approaches: a) Unimodal: With only RGB images, ResNet 50 pre-trained on ImageNet Dataset. b)Multimodal: With RGBD data, converted to two data: RGB and Depth. Both are fed parallelly to the network. In [6], Deep Neural Network architecture with ResNet-50 is proposed. Grasp orientation is determined using classification instead of Regression. ( i.e theta=180 degree is divided into total 20 labels. This solution detects and localizes grasps at around 8fps with 89.0% grasping success rate. [7] attempts to determine

*This work has been supported by Robotics & Automation Laboratory-COEP and Center of Excellence-SIP. *All Authors have contributed equally.*

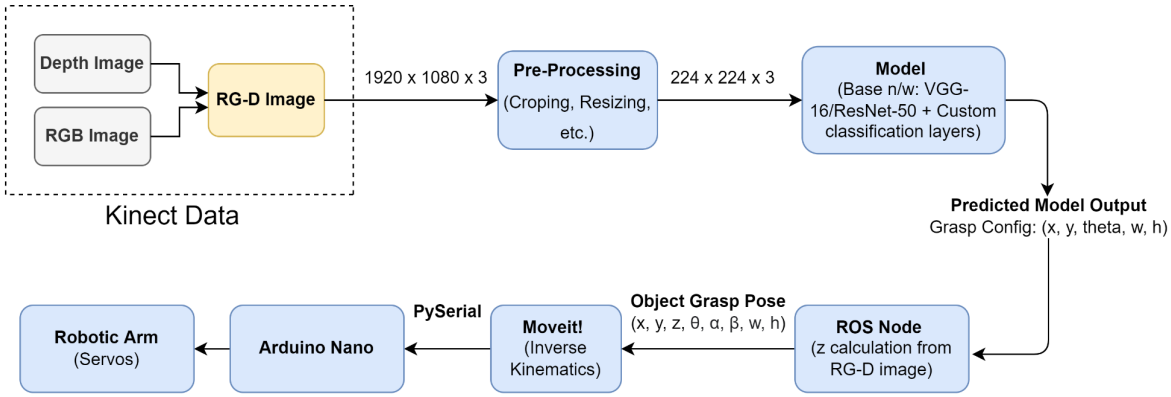



Fig. 1: Complete Block Diagram of our System.

ideal grasping parameters, in a complex environment, with densely located objects. The proposed approach, which comprises of the REG-model, probabilistically expresses the ease with which a robotic hand can grasp a target object. Moveit was used for Path planning. They achieved Grasping success rate of 82%. In [8] light-weight generative Grasp convolutional Neural Network is presented which predicts the grasp pose in non-static environments.

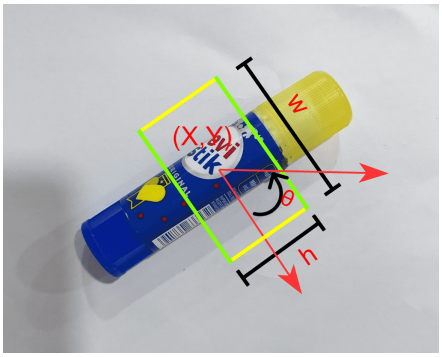



Fig. 2: Grasp configuration representation.

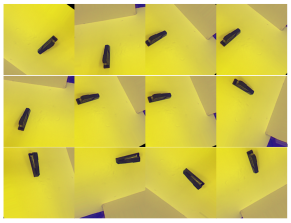
## III. Problem Statement

The problem statement consists of grasping unknown, unordered, and randomly oriented objects. Normally in Industries Robotic arms can pick objects that are initially placed in a predefined order for example an assembly line. The robot is hardcoded to pick and place known items. But what if the items are unknown or placed in a random manner or what if there are multiple objects stacked in a place and we want to separate items individually. This is where a Computer Vision based algorithm is needed. The arm should automatically orient itself in a suitable grasping position which will be different for different objects.

By analyzing the RGB-D image from the camera, a 5-dimensional grasp configuration needs to be predicted. The grasp configuration($g$) includes the location of the gripper $(x, y)$, the angle($\theta$) between the horizontal axisand gripper plate, gripper opening distance($h$) and width of the gripper($w$). The Grasp Configuration is represented as shown in the Ref. Figure 2. Thus our Learning model should outputs this grasp configuration, from where we calculate the grasp pose of the object. The grasp configuration is given by the following equation: $g = \{x, y, \theta, w, h\}^T$. The two main points of problem statement that we wish to address are:

- Grasping different types of objects having different shape and size. Ref. Figure 3a
- Grasping object which is in different pose(i.e. position and orientation). Ref. Figure 3b

(a)



(b)

Fig. 3: a) Different Test Objects from Cornell Dataset. b) Test Object having different Orientation and Position.

## IV. Proposed Solution

This research examines how deep learning algorithms can be used to train robotic arm to grasp different objects in unstructured environments. A custom Deep Learning based robotic grasping model is developed. The model is based on Transfer Learning and predicts the 5-D grasp configuration. We also make use of an external RGB-D camera(Kinect v2). Pre-processing of the real world Kinect data is also an important step. This is accomplished using a variety of image processing software libraries. The neural network is then used to recognise grasp rectangles from the Kinect captured image. Since a very few studies have described end-to-end grasping pipleline which includes testing the Model's

accuracy on a real robotic platform, we built our own 5-DOF robotic arm with the help of all the available low cost electronic components. We also developed a PCB for it and wrote its complete low level embedded code on an Arduino Nano microcontroller. Complete ROS & Moveit support is provided for the developed robotic arm. Thus, full end-to-end grasping pipeline is established right from capturing RGB-D image, prediction of rotated bounding boxes, ROS and moveit support for the robotic arm, 3D grasp pose determination from predicted grasp configuration, Transforms from 2D image to the base link and finally the trajectory planning of Robotic arm.

## V. Methodology

### A. Data Preprocessing

We have used Cornell Grasp Dataset for the training purpose. It contains 885 RGB images, Point cloud data, labelled positive rectangles and labelled negative rectangles. These images are of nearly 240 different objects such as fruit, scissor, marker, brush, tools and few other household objects. The positive and negative rectangles simply signifies good and bad graspable locations. All these RGB images in the dataset have a size of 640 pixels in width and 480 pixels in height. The depth information for each image is extracted from the Pointcloud data available in the dataset. Also the depth information ranging from 0.5 - 2.5m is then normalised to fall between 0 and 255. In depth images the white value increases as the depth increases. Some pixels lack depth information(NaN) because they are occluded; we substitute these pixel values with 0.

Now that we have grayscale depth image, the blue channel of colored RGB image is replaced by the D channel in order to create the RG-D images. We cannot directly combine RGB image and Depth image to form a 4 channel RGB-D data. Because most of the pre-trained neural network models only accept 224x224x3 channel images. While Cornell dataset is a relatively small grasping dataset, the data best suits parallel plate gripper and multiple labelled grasps are provided per image. Hence we augmented the Cornell Grasping Dataset with random translation and rotations which generates 125 augmented data for each image. In pre-processing first we center crop image to width and height of 320 x 320 pixels. Then we translate the image randomly upto 50 pixels in both x and y direction and randomly rotate it between 0 to 360 degree. Also the annotations for augmented images were generated by appropriately translating and rotating the original coordinates. Finally the image is resized to 224 × 224 pixels to fit the input of VGG-16 [9] and ResNet-50 [10] architecture.

### B. Model Architecture

We have employed the model architecture as shown in Figure 4 which is based on the VGG16 architecture with a few modifications in the classifier part or the fully connected layers. The last fully connected layer is removed and 2 fully connected layers are added for 2 different outputs i.e. Cls_score and Rect_pred. The Cls_score is nothing but the

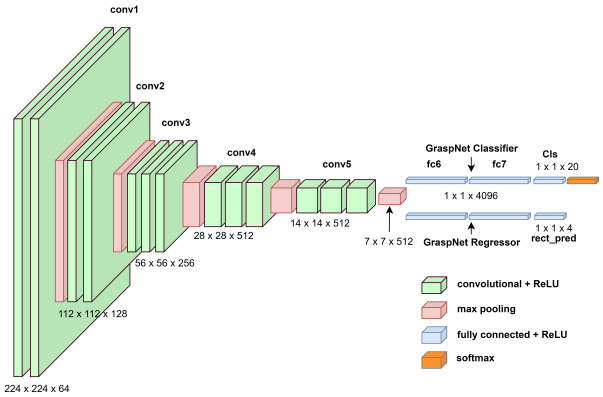



Fig. 4: Our Model Architecture based on VGG16

predicted grasping angle($\theta$). 180 degrees are divided into 20 groups or classes of 9 deg each. The second output i.e Rect_pred gives the predicted rectangular bounding boxes. The model takes an input RGD image of size 224 x 224 x 3.

Both the models with base of VGG-16 and ResNet-50 were trained on the Nvidia DGX system. Pytorch, with CUDA enabled GPU support was setup in Docker environment. We used the SGD optimiser with a momentun of 0.9 which decays the learning rate. As already mentioned the model output are two entities. One is the Cls score vector which is passed through a softmax function. As this is a classification output we use the cross entropy loss. Second output entity is the predicted rectangle. This is a regression output. Now, for calculating the total loss, we take the sum of these two losses. The model is trained for 30 epochs, with a learning rate of 0.001 for ResNet50 and 50 epochs with learning rate of 0.0001 for VGG16. The batchsize is 64.

### C. Real world Transforms

*1) Camera Transforms:* We have used Kinect v2 RGB-D camera for taking visual feedback along with the depth information of the objects which are to be picked. All the available cameras always perform some kind of transformation from 3D world coordinates to the local 2D space coordinates. In short camera takes real-world 3D scene and maps it into the 2D image plane. And in order to determine and understand the acquisition system we need to find the parameters of transformation. These parameters are classified into two types:

- Intrinsic parameters
- Extrinsic parameters

*2) Intrinsic Parameters:* Intrinsic parameters are the parameters that maps the 3D camera coordinate system into the 2D Image coordinate system. Intrinsics are specific to a given Camera. Intrinsic parameters are given by:

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}$$

Here, $(f_x, f_y)$: Scale factors in pixel along x and y direction respectively. In a true pin-hole camera focal length fx and fx

both have the same.
$(x_0, y_0)$: Camera center in pixel.
$f_x = F/p_x$
$f_y = F/p_y$
$s = f_x * \tan(\alpha)$
$(p_x, p_y)$: Size of pixel in world units.
$F$: It is the Focal length
$s$ : Skew coefficient, which is non-zero if the image axes are not perpendicular.

*3) Extrinsic Parameters::* Extrinsic Parameters are the parameters that determine the pose of the camera(ie. The orientation and location) with respect to the world coordinates. It has two components, Translation and Rotation.

$$R = \begin{pmatrix} R_1 \\ R_2 \\ R_3 \end{pmatrix} = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}; \qquad t = O_w - O_c$$

Where,
$O_c$: centre of Camera coordinate system
$O_w$: centre of World coordinate system
$R$: Rotation Matrix describes the camera's orientation w.r.t. to world coordinate
$t$: Translation vector describes the change in position of the coordinate centre $O_w$ and $O_w$.
The Rotation matrix $R$ and Translation vector $t$ inturn represent the extrinsic parameters of the camera. It can be represented in the form

$$\begin{bmatrix} R \mid \mathrm{t} \end{bmatrix} = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]$$

*4) Getting Real-world coordinates::* Now combining both the intrinsic and extrinsic camera calibration Matrix we get a Projection matrix P. Where

$$P = \overbrace{K}^{Intrinsic\ \text{Matrix}} \times \overbrace{\begin{bmatrix} \mathrm{R} \mid \mathrm{t} \end{bmatrix}}^{Extrinsic\ \text{Matrix}}$$

$$= \overbrace{\underbrace{\begin{pmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{pmatrix}}_{2D\ \text{Translation}} \times \underbrace{\begin{pmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{2D\ \text{Rotation}} \times \underbrace{\begin{pmatrix} 0 & s/f_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}}_{2D\ \text{Shear}}}^{Intrinsic\ \text{Matrix}} \times$$

$$\overbrace{\underbrace{(I|t)}_{3D\ \text{Translation}} \times \underbrace{\left(\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right)}_{3D\ \text{Rotation}}}^{Extrinsic\ \text{Matrix}}$$

Now the relation from 2D pixel coordinate to 3D real world coordinate is given by:

$$x_t = PX$$

Where,
$x_t$ : $[x \;\; y \;\; z]^T$
$P$ : Projection matrix consisting of intrinsic and extrinsic parameters
$X$ : 3D real world coordinates in the world frame

Now, to get the pixel coordinates by transforming the image coordinates: we simply divide $x, y$ by $z$ to get homogeneous coordinates in the image plane.
$[x, y, z]$ to $[\mathrm{u}, \mathrm{v}, 1] = [\mathrm{x/z}, \mathrm{y/z}, 1]$
Thus if we want the to find out the 3D real world coordinate from the 2D pixel coordinate then we simply have to use the following equation:

$$X = P^{-1} \cdot x_t$$

### D. Moveit-Setup Assistant

We use the ROS package 'Moveit!' to configure Inverse Kinematics for our custom written URDF of our robot. First we launch the moveit setup assistant and give it the path of the located urdf file. Further, we define a link in our robot that will act as the end effector. In our case, the link 'hand' is our end effector. We used the KDL IK solver due to its robust performance and reliability in finding Optimised solutions. Then, we defined multiple predefined poses for the robot named home, idle and pick. Finally, the setup assistant generates a package for us that we will be using for movement and path planning of the robotic arm.

### E. Communication between ROS and Arduino

In order to communicate between the ROS environment from our Laptop to the Arduino we need a communication protocol which could send the joint angles. These joint angle values need to be sent continuously along the trajectory to the Robotic arm working on Arduino. For achieving so we made use of Pyserial python library. The Python API module PySerial is essentially used for serial communication between any microcontroller and the Arduino. The data was sent at a baud rate of 9600.

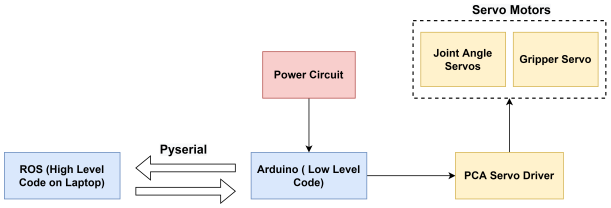


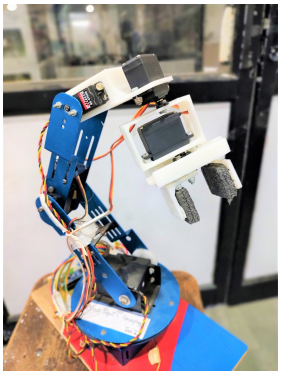

Fig. 5: Communication between ROS and Arduino

## VI. Hardware Implementation

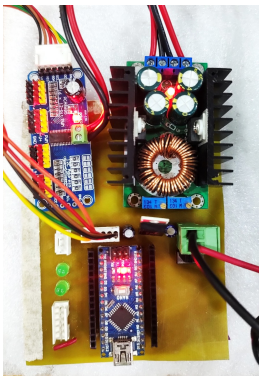
### A. Hardware - Robotic Arm Structure

The robotic arm is primarily assembled using CNC cut sheet metals made of stainless steel. 3D printed rack and pinion are used to actuate the parallel plate gripper. The weight of the entire robotic arm is 3kg. The arm originally was a 4 DoF and was unable to perform rotation in the z-axis (yaw movements) because the wrist DoF motor was absent. Also, the yaw movement is particularly important for us to achieve grasping from random positions and orientations of the object. Therefore, we added an extra degree of freedom by mounting

a servo motor on the arm using a custom 3D printed clamp as shown in the Figure 6a. We also made the complete electronic circuit design & PCB for the same.(Ref. Fig 6b)



(a)



(b)

Fig. 6: a) Developed Robotic Arm with added DOF. b) Designed & Implemented PCB board.

## VII. Experimental Setup

### A. Robotic Arm workbench setup

The trained grasping model's robotic gripping capacity was demonstrated using our modified 5-DOF arm. Our robotic arm has a maximum range of 320 mm, a maximum payload capacity of 100g (excluding the weight of the end-effector).

| Parameters | Value |
|---|---|
| Arm Maximum Reach | 32 cm |
| Payload Capacity | 100g |
| Kinect Height | 105cm |
| Gripper Width | 10-42 mm |
| Power Requirement | 5V-4Amps |
| DOF | 5 |
| Motors Used | Servo HS-475HB |

TABLE I: Experimental Setup Parameters

The ROS infrastructure is used to control the entire robotic system. As the arm is custom, Moveit is used for inverse kinematic solver, which is used to direct the arm to pick up things once their grip locations have been identified. Color and depth images i.e RGB-D; of the work area are captured using a Microsoft Kinect v2 camera. To interface kinect with ROS libfreenect2 [11] and iai_kinect2 [12] packages were used. We have used the Arduino Nano for controlling the arm movement and for communication with the laptop in order to receive the joint angles. The Kinect is mounted on top for getting the grasping pose. It is located on top of the work space so basically it is a hand-to-eye system. Additional physical parameters can be found in the Table I. The experimental setup can be seen in Figure 7.

### B. Pick and Place Taskflow

Now, that we have the setup ready, the following steps define the taskflow which is going to be followed:

Fig. 7: Experimental Setup

- **HOME:** We keep the arm initially at the defined home position.
- **GET-POSE:** First, we obtain the dynamic grasp configuration of test object from the model. Then we pass it to the ROS node that commands the arm to move.
- **PRE-PICK:** Before moving the arm, we ensure that the gripper is completely open. The arm moves to a position that is 5 cm above the object.
- **PICK:** Then the arm moves downwards(Z axis) to pick the object. Once the end effector reaches the desired pose, the gripper closes to grasp the object.
- **POST-PICK:** Before moving to the drop position the arm moves upwards in the Z axis to verify the grasp.
- **DROP:** Then the arm moves to the pre-defined drop location that is kept same for all objects and opens the gripper to drop the object.
- **HOME AGAIN:** Finally, the arm returns to the initial home position and is ready to take the next grasp configuration.

## VIII. Results

All the grasp configurations that are predicted by our grasping model shall be evaluated in order to determine whether it is an acceptable grasp or not. In most of the object detection cases we can fairly use rectangular metric for evaluation. But in our case since the predictions are rotated bounding boxes we need to take grasp angle detection in account too. As a result, our standard Jaccard Index metric is somewhat adjusted, and the addition of grasp angle detection is also taken into account. Thus, the new metric defines a successful grasp under the two conditions listed below.

1. The difference in grasp angles should be less than $30°$.

2. Between the two grasps, the jacquard index should be greater than 25% The following equation generates the Jacquard index (J):

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

The Ref. Table II displays the IOU scores and accuracy's. For our Grasping Architecture based on VGG16, testing accuracy of 78.2% was achieved. Whereas for our another architecture based on Resnet50, testing accuracy of 83.3% was achieved.

| Base Model | Training IOU | Training Accuracy | Testing IOU | Testing Accuracy |
|---|---|---|---|---|
| VGG16 Regression+ Classification | 0.48 | 84.4% | 0.45 | 78.2% |
| Resnet Regression+ Classification | 0.62 | 92% | 0.52 | 83.3% |

TABLE II: Results.

### A. Experimental Results on our Robotic arm Setup

There are just a handful studies that have used genuine robotic hardware to do grasping. Although the evaluation metrics indicate that the grasp detection is adequate, the results from actual robotic grasping will affirm the grasp detection results. So we tested our Grasping model and the entire grasping pipeline on our 5-DOF Robotic arm Setup. Test was done on nearly 5 different objects and for each object 10 trails were performed.(Ref. Table III) In Ref. Figure 8 we can see the real-world prediction result on the test object marker placed on our workbench area. The figure also shows our 5-DOF Robotic arm which is places within the FOV of Kinect camera.

| Objects | Trials | Picking Accuracy(%) |
|---|---|---|
| Screwdriver | 10 | 90 |
| Marker | 10 | 100 |
| Tape | 10 | 70 |
| Bottle cap | 10 | 80 |
| Glue-stick | 10 | 90 |

TABLE III: Experimental Results on our Robotic arm Setup

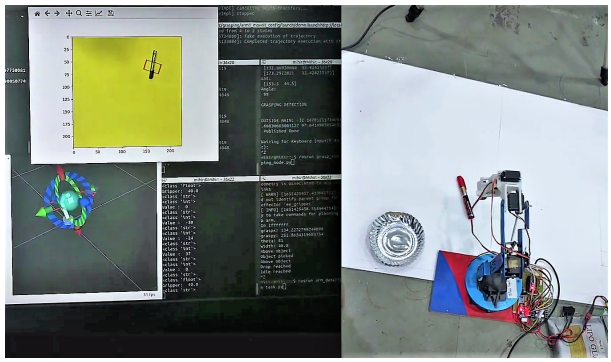


Fig. 8: Realworld Prediction Result.

## IX. Conclusion

Thus in this work we presented a Deep Learning based robotic grasping model and also developed a full-fledged task flow for robotic grasping on an actual 5-DOF robotic arm. Modifications were made to the existing 4-DOF robotic arm and an additional DOF was added to it. Required electronic design system and PCB was also made. Embedded low level code was written on an arduino-nano micro-controller in order to control the robotic arm. Also ROS and Moveit support was provided for the custom low cost robotic arm. The implementation of the deep learning model on the real robotic platform had promising results. We see our work as a stepping stone in our journey to explore the unexplored world of learning based robotic grasping. Robotics is an evolutionary field and number of robotic arms in manufacturing units and warehouse automation has been exponentially increasing. Thus in coming future we see that deep learning based approaches will be widely used by robotic arms so as to perform pick & place operation in unstructured environments. That day won't be far when robotic arms on a personal robot having cognitive abilities will be used in our homes too.

## References


[1] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, "A survey on learning-based robotic grasping," 2020.

[2] Y. Jiang, X. Li, M. Yu, and Z. Bai, "Robotic grasp detection using light-weight cnn model," *2020 Chinese Control And Decision Conference (CCDC)*, pp. 1034–1038, 2020.

[3] H.-I. Lin and M. N. Cong, "Inference of 6-dof robot grasps using point cloud data," in *2019 19th International Conference on Control, Automation and Systems (ICCAS)*, 2019, pp. 944–948.

[4] S. Kitagawa, K. Wada, S. Hasegawa, K. Okada, and M. Inaba, "Multi-stage learning of selective dual-arm grasping based on obtaining and pruning grasping points through the robot experience in the real world," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7123–7130.

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

[6] F.-J. Chu, R. Xu, and P. A. Vela, "Real-world multiobject, multigrasp detection," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3355–3362, 2018.

[7] K. Kozai and M. Hashimoto, "Determining robot grasping-parameters by estimating "picking risk"," *2018 International Workshop on Advanced Image Technology (IWAIT)*, pp. 1–4, 2018.

[8] D. Morrison, P. Corke, and J. Leitner, "Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach," in *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[9] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[10] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.

[11] D. for Kinect for Windows v2 (K4W2) devices (release and developer preview)., "Available online: https://github.com/openkinect/libfreenect2."

[12] T. for using the Kinect One (Kinect v2) in ROS., "Available online: https://github.com/code-iai/iai$_k inect2$."