International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 12, Number 11 (2019), pp. 1881-1889

Multi-object Detection for Autonomous Motion Planning based on Convolutional Neural Networks

Nouaim Mebarki
PAUSTI, Nairobi
mebarkinouaim@gmail.com

Rehema Ndeda
JKUAT, Nairobi
reheman@eng.jkuat.ac.ke

Noureddine Ouadah
CDTA, Algiers
nouadah@cdta.dz

Christopher O. Adika
Multimedia University, Nairobi
otychriss@yahoo.com

Abstract
We present a multi-object detection approach based on deep learning for motion planning. The multi-object detector is based on efficient convolutional neural network, and the used approach outperforms state of the art methods in the detection of objects leading to efficient autonomous motion planning. The multi-detector model is more efficient the reason that it: a) detects multiple objects given one input image. b) uses one neural network that makes it computationally cheaper, and faster in test time. c) makes a robot able to perform a process of goal motions given only one detection input carrying multiple objects.

1. Introduction

Object detection has become a dominant deep learning problem where the accuracy of detection, speed of processing are the essential factors to improve while working with convolutional neural networks (CNNs).
In robotic applications, using detection techniques that combine computer vision with deep learning has proven an impact in making robotic systems more autonomous and intelligent. However, it is quite important to get a significant detection in order to make these robotic systems safe, accurate, and fast.
This research introduces a Multi-object detector based on object detection that can be used in many industrial robotics applications including: robotic grasping, pick and place, painting, objects tracking...
More specifically, it uses object detection for goal assignment and motion planning. The used approach outperforms state of the art considering these following points:

1. The detector is able to detect more than one object to assign goal trajectories given only one detection image as input, rather than detecting one region of interest on one object [1, 2].
2. It uses only one neural network for the prediction instead of using more than a network that first determine many candidate regions of proposal, and then second make a classification decision for each one of those candidate proposals such as [3].
3. It presents an algorithm based on the Robot Operating System(ROS) to allow a robot to plan goal trajectories towards multiple objects given one sole detection input which has lacked in [1].

Because CNNs usually need a high computational power and space, researchers tend to think of ways to optimize the computational efficiency, that is why the approach used in this research uses quite more efficient model. It is faster and more accurate than the previous state-of-the-art object detection techniques for single shot detection as YOLO [4]. It is as accurate as methods that use explicit region proposals and pooling such as Faster R-CNN [3]. The model can be used in embedded applications which makes it flexible with systems like robots, IOT, and also does not resample pixels or features for bounding rectangles locations and it is as accurate as models that do.
Finally, this research introduces an algorithm based on ROS to test the detection model on a robot. It allows a robotic arm to plan motions towards each object detected by the camera, and one input image is enough for the motion to be executed on each object appeared in the detection.

2. Related Work

As compared to image localization, object detection uses a different paradigm, so one approach that is very common and has been used for a long time in computer vision is the method of sliding window. The idea in this approach is that it takes different crops in the input image, so given a crop the neural network will make a classification decision, and also this approach adds another category that is the background and the network can predict it in case it does not see any of the categories. And it will continue with predicting many other crops until it predicts all categories in the image. The problem in the sliding window approach is how to choose the crop, because there could be any number of objects in an image, these objects could appear at any location in the image, and could appear at any size, and at any aspect ratio, so the sliding window will make a model test tens of thousands of many different crops, and this would be completely computationally intractable and this is why

recent state of the art object detection research avoid using this approach with convolutional neural networks.

Instead, there are other approaches, one of them is Region proposals. Given an input image, a region proposal network will then give a thousand boxes where an object might be present, so it will output some set of candidate proposal regions where objects might be potentially found. This is relatively fast to run. Some common examples of region proposal methods: Selective search [5], it spits out 2000 region proposals in the input image where objects are likely to be found. So rather than applying the classification network to every possible location and scale in the image, Instead, it first applies one of the region proposal networks to get some set of proposal regions where objects are likely located, and next will apply a convolutional network for classification to each of these proposal regions and this will end up being much more computationally tractable than trying to do all possible locations and scales. All this came together in R-CNN [6]. Given an input image, in this case, there is a region proposal network to get a set of proposed regions of interest, resize them so all have the same pixel size that is expected as input to the downstream network. So after warping them to a fixed size, then it will run each of them in a convolutional network to make a classification to predict categories for each of those regions, and also it predicts a regression (a correction to the bounding box).
However, there are problems about the R-CNN framework, it is still computationally expensive, because if there are 2000 region proposals, it is running each of those proposals independently, which can be expensive. There is also the question of relying on fixed function [5] for calculating the region proposals that the network has not learned them, and in practice that ends up being slow. In the original implementation of R-CNN would dump all the features to disk, so it will take hundreds of gigabytes of disk space to store all these features which makes training slow since there are all different forward and backward passes through the image and it took 84 hours [6] in training time. At test time it is also slow, it takes roughly 1.5 minutes per image because it needs to run thousands of forward passes through the convolutional network for each of these region proposals. Fast R-CNN [7] has fixed a lot of those problems. For fast R-CNN, rather than processing each region of interest independently, the entire image is going to be run through some convolutional layers all together to generate a high resolution convolutional feature map. In fact, it is still using some region proposals from a fixed function like Selective search [5], but rather than cropping out the pixels of the image corresponding to the region proposals, instead those region proposals are projected onto the convolutional feature map, and then taking crops from the convolutional feature map corresponding to each proposal rather, and this allows not to reuse a lot of the expensive convolutional computation across the entire image. One problem in Fast R-CNN being bottlenecked by computing the region proposals. Thankfully, Faster R-CNN [3] has solved this by making the network itself predict its own region proposals. The entire input image is run through a set of convolutional layers to get a convolutional feature map representing the entire high resolution image, and now there is a separate region proposal network which works on tops of these convolutional features and predicts its own region proposals inside the network. A problem in faster R-CNN is that it has to do four things all at once, balancing out this four-way multi task loss is somehow difficult (classification and bounding-box regression losses for proposals and Classification plus bounding-box regression losses after determining the best proposals).

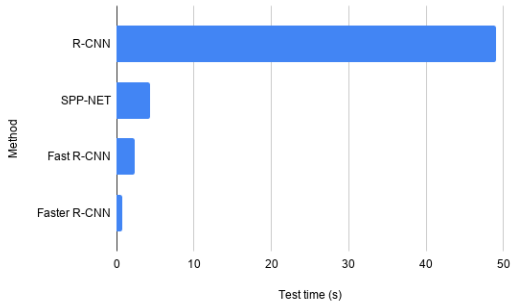

Figure 1. A comparison between the efficieny of R-CNN models

There is another family for object detection that has solved that and works without the proposal regions is YOLO [4], and SSD [8] which is relied on in this paper. The idea is that rather than doing independent processing for each of the potential regions, instead it treats this as a regression problem and make all these predictions all at once with one big convolutional neural network. Given an input image, it is divided into some coarse grid (7x7 grid) and within each of those grid cells, we imagine some set of base bounding boxes, and for each of these base bounding boxes a network predicts several things: an offset off the base bounding box, the true location of the object off this bounding box, and also predicts classification scores. So at the end, it will end up predicting from an input image a giant tensor of 7x7 x(5*B + C). So that is just where it has B bounding boxes, five numbers of each giving our offset, and the confidence for the base bounding box, and C classification scores for the C categories. Using SSD [8] for our research has outperformed Faser R-CNN [3] in the detection accuracy and test speed, and it makes a robot able to detect multiple objects which has lacked in the Multigrasp model proposed by [1], and has lacked in many robotics machine learning approaches for goal detection such as [2, 9].

3. Problem formulation

Given an input image, in addition to predicting what the category of an object is, a neural network wants also to know where is that object in the image. So it wants to draw a bounding rectangle around the region of the object in that image. The process here is classification plus localization. The distinction here between this formulation and the model of object detection is that in the localization scenario, assuming ahead of time that there is exactly one object in the image, as shown in Figure 2.
The input image is fed through a convolutional neural network, which will give a final vector summarizing the content of the image, then it will have some fully connected layer which goes from the final vector to the class scores for classification. Also,

it will have another fully connected layer which goes from a vector to four numbers: the height, width, x and y positions of that bounding box. The neural network will produce these two different outputs: one is the set of class scores, and the other is four numbers giving the coordinates of the bounding rectangle in the input image. In training, this network will have two losses, knowing that this scenario is considered a fully supervised setting, so it has each of the training images annotated with both a category label, and a ground truth bounding box for that category in the image. So it has two losses, a softmax loss that is computed using the ground truth category label and the predicted class scores, and another loss that gives some measure of dissimilarity between the predicted coordinates for the bounding box and the actual coordinates of the bounding box (Regression loss between the predicted bounding box coordinates and the ground truth bounding box coordinates).

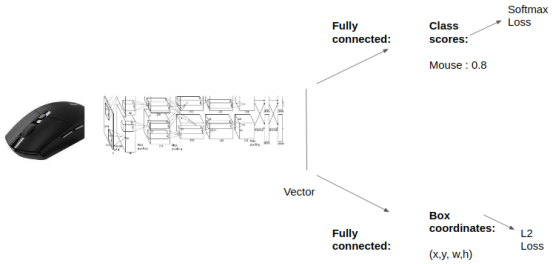

Figure 2. Localization and classification approach

In object detection, it usually starts with a fixed set of categories in the dataset, and the task is that: given an input image, every time one of those categories appears in the image, a neural network wants to draw a box around it and predict the category of that box. So this is different from localization and classification because there might be a varying number of objects for every input image. The neural network does not know ahead of time how many objects it expects to find in each image so this ends up being more challenging problem. The architecture of this problem looks like Figure 3.

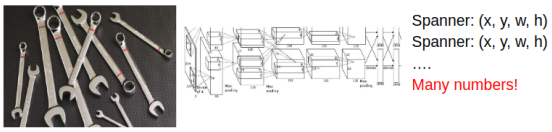

Figure 3. Object detection approach

3.1. System representation

Given an input image, a neural network must find a way to successfully predict the locations, category classes of each of the objects that appear in the image. In this context, the labeled data of the images contains bounding boxes where objects are located with their category class. This enables the neural network to predict box coordinates and classes for each object appeared in the image. The output of the neural network is instantly used by a ROS algorithm to assign goal locations and execute motions towards every object.

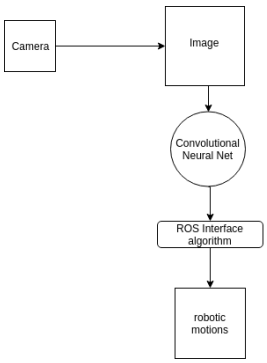

Figure 5. System process

4. Model

The object detection model can be tried in a simpler way and focus the neural network on predicting the bounding box of the object(Regression solely) without classification, but when the neural network uses transfer learning, it always finds better performance if it fine-tunes the whole system jointly. So, if an approach takes a pretrained model, trains it on a dataset, the performance would be better even if the neural network is changed from the pretrained one.

Given the network trained on the object detection dataset, the robot uses this model to make a goal motion towards every object appeared in the detection.

The model is an SSD with Mobilenet, where MobileNet is the backbone(base network) of SSD. MobileNet is placed as the feature extractor network. The original SSD was using VGG [10] as the base network, but later on other variants of VGG outperformed it as MobileNet, Resnet, and Inception. In this case, The approach used in this research uses Mobilenet since it is quite faster and accurate [11] in a reasonable way compared to other CNNs that are used for classification.

The SSD approach has a feed-forward convolutional neural network which produces a fixed-size collection of bounding rectangles and scores if object class instances in those rectangles are present, next there is a non-maximum suppression step to produce the final detections. The mobilenet network layers are used for high quality image classification (truncated before any classification layers), which represents the base network. SDD has auxiliary structure in the network to make detections in the following key features: Multi-scale feature maps for detection, Convolutional predictors for detection, Default boxes and aspect ratios [8]. The architecture of the model is shown in Figure 4.
For Transfer learning, the model used is SSD with mobilenet trained on Microsoft COCO dataset from the object detection API provided by [12], it runs on a speed of 30 ms, with mAP of 21 , and it outputs bounding boxes instead of masks.

In the implementation of the object detection model, we make it able to run 5 functions:

- preprocessing: scaling, shifting, and reshaping of the input values which is necessary prior for running the object detector on a given input image.

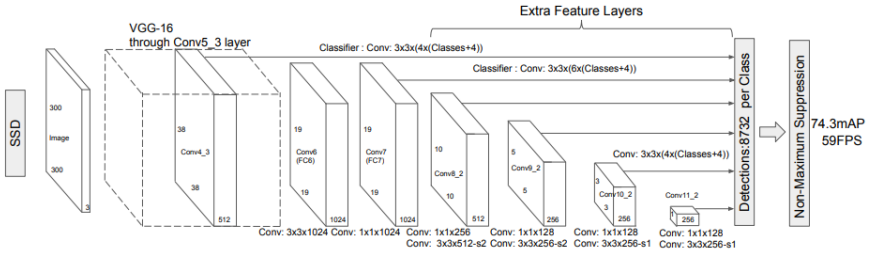

Figure 4. CNN architecture based on SSD [8]

- predicting: Producing raw prediction tensors that can be passed to loss or postprocess functions.
- postprocessing: Converting predicted output tensors to final detections.
- loss: Computing scalar loss tensors with regard to the provided groundtruth labeled data.
- restoring: Loading a checkpoint into the Tensorflow graph that will be used next for the grasping detection.

5. Training approach

5.1. Data preparation process

Since the model is fully supervised, the training data consists of images, each image has all the objects categories marked with bounding boxes for each instance of that category. The data collection process has been made according to these following steps:

- Collect training and testing images. 500 images for training set, and the number of images for the testing set must be 30% of the number of training set(The more images there are the more accurate the model can be, however due to hardware computational limitations, the dataset used is not very large).
- Resize images to 960 width, 540 height.
- Annotate the images: We used labelimg [13] to annotate the images. This is an Opensource handle tool where the created annotations will be in the Pascal VOC format which will help later on importing the labeled data to the model. Essentially the goal is identifying the location of the bounding rectangle (xmin, ymin, xmax, ymax). Figure 6 shows an example of an annotated image.

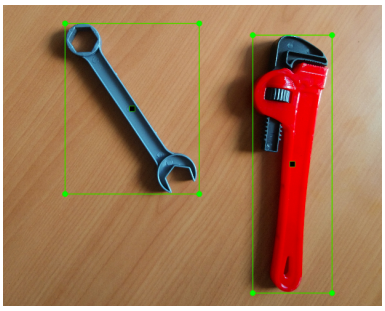

Figure 6. Annotation example

- Tensorflow API needs the dataset to be in TF-Record format. We have made a special script to transform the labels in an xml file to a specific format corresponding to the Tensorflow ground Truth file.
- A label.pbtxt file must then be created to convert label name to a numeric id.
- Create train.record and test.record files corresponding to Tensorflow framework.

5.2. Loss functions

There is a multi-task loss which is having many losses in one neural network. Whenever we take derivative of a scalar with respect to the network parameters and use that derivative to take gradient steps. It has got two scalars to both minimize. In practice, the goal is to have some additional hyper parameter that gives some weighting between these two losses, so it will take a weighted sum of these two different loss functions to give the final scalar loss, and then it will take gradients with respect to this weighted sum of the two losses. The losses in mathematical functions are described as:

5.2.1 Regression Loss

The regression loss will evaluate the bounding boxes that the neural network wants to predict. Known as a localization loss in the model.
It is a Smooth L1 loss [7] that calculates the difference between the predicted rectangle (l) and the actual ground truth rectangle (g) parameters. Similar to Faster R-CNN [3], the aim is to regress to offsets for the center (cx, cy) of the default bounding rectangle (d) and for its width (w) and height (h).

$$L_{loc}(x,l,g) = \sum_{i\in Pos}^{N} \sum_{m\in\{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$
$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$
$$\hat{g}_j^{w} = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^h}{d_i^h}\right) \tag{1}$$

5.2.2 Classification Loss

It is also known as the confidence loss, described as the softmax loss over multiple classes (c).

$$L_{conf}(x,c) = -\sum_{i\in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i\in Neg} log(\hat{c}_i^0) \quad (2)$$
$$where \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

$x_{ij}^p = \{1,0\}$ is an indicator for matching the i-th default rectangle to the j-th ground truth rectangle of the category p. In the matching strategy, it is represented as $\sum_i x_{ij}^p \geq 1$.

5.2.3 Total Loss

The total objective loss function is a weighted sum of both localization loss and classification loss:

$$L(x,c,l,g) = \frac{1}{N}(L_{conf}(x,c) + \alpha L_{loc}(x,l,g)) \quad (3)$$

N is the number of matched default rectangles. If $N = 0$, the loss will be set to 0.
The weight term α is set to 1 by cross validation.

6. Training results

6.1. mAP metric

We compared training results of SSD, fast [7] and faster R-CNN [3] on the same set of RGB-D images of objects taken by a phone camera(Those images can be obtained from a dataset like COCO, but we have used our own images just to make the detection more accurate in one specific environment to reduce errors in the robotic test). Figure 7 shows that SDD has got better accuracy of detection than the R-CNN family.

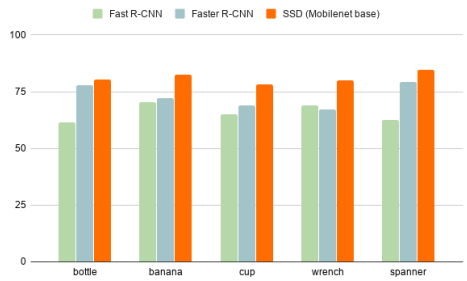

Figure 7. mAP detection results for SSD, Fast, and Faster R-CNN

The average result of the mAP metric for SSD has got the best performance than other detection methods. Results are shown in Figure 8

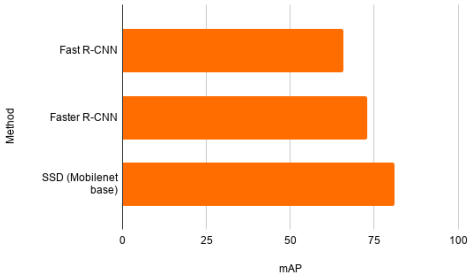

Figure 8. mAP average result for detection of previous objects

6.2. Losses

6.2.1 Classification loss

Figure 9 shows the softmax loss function. It is computed using the ground truth category labels and the predicted class scores. The values of the classification loss converge to nearly 0.43 after more than 5K training steps, and which enabled the detector to accurately predict the classes of the objects labeled in the dataset. This minimization in the loss functions corresponds to minimizing the errors in back propagation and improving the learning weights.

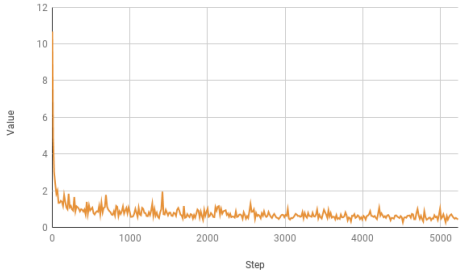

Figure 9. Classification loss

6.2.2 Regression loss

The loss function converges approximately to 0.04 after more than 5K training steps of 2 seconds learning rate, and that enabled the model to have a reasonable detection for the region of interest of each object that the model has been trained on.

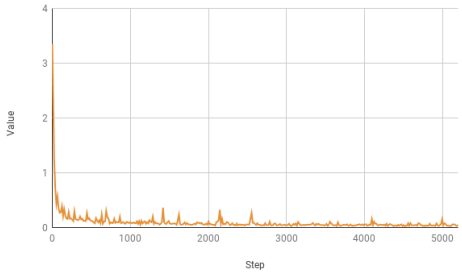

Figure 10. Localization loss

6.2.3 Total Loss

The total loss is the average of two losses. It converged to a value of 0.80 which gave a reasonable detection for our model

according to the test experiment.

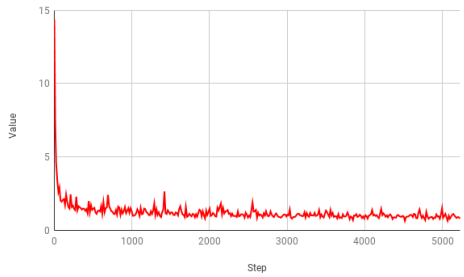

Figure 11. Total loss

7. Robotic experiment

The principal aspect of using ROS with a model of object detection is to integrate the detection model with the motion robotic model, so both of these models communicate with each other. This architecture has been made in two ROS nodes that are described in the following subsections.

7.1. Object detection node

The first goal in this node is interfacing ROS and OpenCV by converting ROS images into OpenCV and back again using cv_bridge. Then we create some set of topics to publish the detection results from the object detection model algorithm into a ROS package to another second node for motion planning.

In developing this node algorithm, there is first a class __init__ function that generates an instance of cv_bridge. The next step is to register the ROS topics and messages that the node will either publish or subscribe to. The necessary ROS topics for this node have been made as follows:

1. The first publisher: object_detection/updated_image topic to publish the updated image.

2. The second publisher: object_detection/result topic contains a list of the names of the classes, bounding rectangles around each detected object, and the number of the detections.

3. The first subscriber: object_detection/start, that when received, it will make a call to a StartCallback function that starts the object detection on the following received image.

4. The second subscriber: camera/color/image_raw, which will contain the image that comes from the camera and result into a call to the Image_callback function.

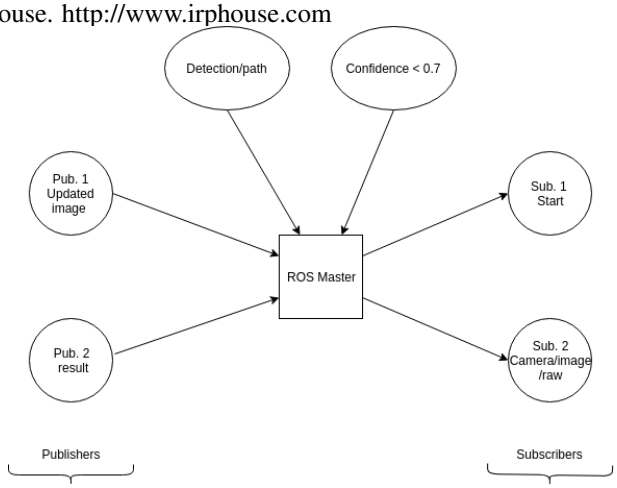

Figure 12. Object detection node

7.2. Motion planning node

The purpose of this node is to link the inverse kinematics model of the robotic arm(inverse kinematics was already done by uArm swift pro developers) with the vision model, consequently testing the detection model would only rely on sending the detection locations x, y, z corresponding to the joints that move the base of robot, link1, and link2. So, when this node receives the detection location coming from the first node, all is needed is some set of topics which subscribe to the published results from the first node, and others that subscribe to Moveit topics to execute final motions for every object goal corresponding to every location, class that were sent by the detection node. The needed topic for this node is described as follows:

1. A subscriber: object_detection/result topic subscribes to the list of the names of the classes of each detected object, bounding rectangles coordinates, and the number of the detections.

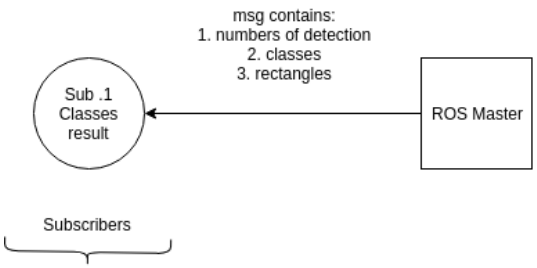

Figure 13. Motion planning node subscribers

7.2.1 Goal assignment algorithm

knowing that "rectangles" is a matrix of(n x 4). n is the number of detections, and 4 are four columns representing the rectangles coordinates for each object. "names" is a vector representing the names of each category class in the detection.

7.3. Testing

The used robot in the experiment is uArm Swift Pro in order to test the detection model as shown in Figure 14. This

Algorithm 1 Multi-goal assignment

function GO_TO_POSE_GOAL($self, data$) ▷ self to use other functions, and data to get publisher information
  $move\_group \leftarrow self.move\_group$
  pose_goal = geometry_msgs.msg.Pose()
  for $detection \leftarrow 1$ to data.number_of_detections **do** ▷ Number depends on the range of camera
    $pose\_goal.position.x \leftarrow (data.rectangles[detection][0] + data.rectangles[detection][2])/2$
    $pose\_goal.position.y \leftarrow (data.rectangles[detection][3] + data.rectangles[detection][1])/2$
    if data.names[detection]=['class1'] **then**
      $pose\_goal.position.z \leftarrow class1\_height$
    else if $data.names[detection] = ['class2']$ **then**
      $pose\_goal.position.z \leftarrow class2\_height$ ▷ Number of classes depends on the number of labled category classes
    else
      print('Nothing detected')
    end if
    move_group.set_pose_target(pose_goal)
    plan= move_group.go(wait= True) ▷ Trajectory planning
    return $all\_close(pose\_goal, motion\_speed)$ ▷ all_close to finish Moveit motion planning related functions
  end for
end function

robotic arm has 4 degrees of freedom, with two links. Table1 represents the specifications of the joint motors that are relied on in our experiment for motion planning.

Joint	Angle	Speed	Lifetime	Torque
Base Motor	0° 200°	40°/s	>3000h	12kg x cm
left Motor	0° 135°	40°/s	>3000h	12kg x cm
Right Motor	0° 100°	40°/s	>3000h	12kg x cm

Table 1. Joint motors specifications of uArm swift pro

The camera is Intel RealSense Depth Camera D435 which is compatible with ROS.
After successfully publishing the topics described by the detection node 7.1, the motion node uses the subscribed topics in 7.2 to get all the detection information, and Moveit generic subscribers for motion planning to execute joint angles based on x, y, z that the detection node ensures. To minimize errors, the object class is used to determine the value of the joint angle adjusting the height of the end-effector.

Figure 14. Motion planning experiment using uArm swift pro

We made sure in the ROS program that when the model runs as well as the name of the objects detected in an image, it gives a confidence level(class score) for the detected object. If the level is below 0.7, the object will automatically be ignored.
In test time, we tested SSD, fast R-CNN [7], faster R-CNN [3] all on a CPU core i7 and the results showed that SSD

outperformed them.

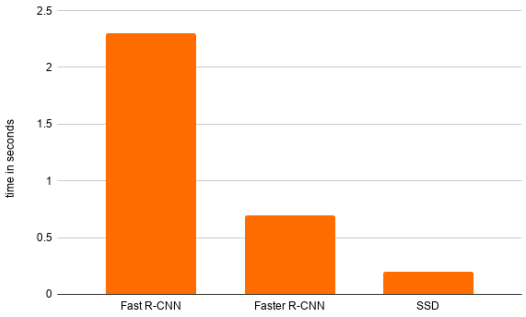

Figure 15. SSD vs. other detectors in Test time

With 100 motion trials for each method, SSD has performed more accurately in the detection than Fast, Faster R-CNN as shown in Figure 16

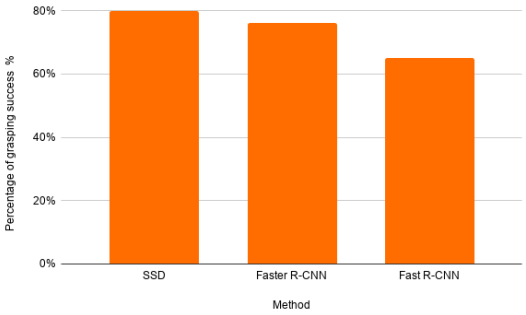

Figure 16. Percentage of goal detection success for SSD vs. other detectors

8. Conclusion

This research has presented faster and more efficient object detection model in RGB-D images based on CNNs that can be used in many robotic applications for goal assignment, and motion planning. The model is able to detect multiple objects given one sole image carrying many objects leading to execute a process of motions based on that detection, and which has lacked in previous methods. We developed the robotic algorithm using ROS which makes this project easy to run on any ROS robotic arm that is developed on the Moveit ROS software. This work is limited in predicting the orientation of the end-effector, and though the provided motion planning algorithm based on the given detection is efficient, it still lacks a feedback determining the success of the motion and improves it. For these reasons, we will extend our project by working on training robots on the motion besides the detection applying new efficient learning approaches so the ratio of a successful motion planning gets higher.

Acknowledgments

The authors are supported in part by Pan African University institute for basic Sciences, Technology and Innovation(PAUSTI). We thank, Dr. Shohei Aoki(JICA Expert at Japan International Cooperation Agency - JICA) for many insightful discussions and guidance through this research. We also thank, Dr. Lerrel Pinto, Research assistant at Carnegie Mellon University for answering many of our questions.

References

[1] J. Redmon and A. Angelova, "Real-time grasp detection using convolutional neural networks," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1316–1322, IEEE, 2015.

[2] I. Lenz, H. Lee, and A. Saxena, "Deep learning for detecting robotic grasps," *The International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 705–724, 2015.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, pp. 91–99, 2015.

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[5] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[6] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[7] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[9] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 3406–3413, IEEE, 2016.

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

[11] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[12] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, *et al.*, "Object detection api."

https://github.com/tensorflow/models/tree/master/research/object_detection, 2017.

[13] Tzutalin, "Labelimg." https://github.com/tzutalin/labelImg, 2015.