

Recognition and Detection of Objects and Their Position for Robotic Grip Decision

Pavel Koprov

Abstract

The modern world demands high variability of products with high customization. Fast paces of technical development made small computing devices and robots available for small companies, but what is left out of the scene is a new framework for small companies to utilize the robots for custom orders. This can be easily done with the unique framework that allows robots to use low-cost cameras and machine learning (ML) models to detect the objects, their position and decide how to grasp the object. This allows the user to load the pile of parts on the "table" and let the robot assemble the product. In this paper, I attempt to tackle this problem using Raspberry PI and robots available at NCSU ISE labs.

Keywords: position detection; grip decision for the robot; raspberry pi; low-cost camera; YOLO

1. Introduction

The modern world demands high variability of products with high customization [1]. Humans can produce products with significant variability, while automated lines are better suited to repetitive, monotonous work. On the other side, robots outperform human workers in speed, quality level, and labor cost. The optimal employee is a robot that can adjust to the environment and fast-changing objectives. This can be achieved through the implementation of machine vision and decision-making based on the retrieved information.

People work alongside robots, and this imposes a challenging task for robotics developers. The machines must be aware of surrounding objects, adjust to the part's non-standard orientation and know how to use them. Current technological progress rates bring the prices for computing devices really low, leading to widespread 'Smart' devices worldwide. Raspberry PI (RPI) is the corollary of this progress; more than 20 million of these low-cost computers have been sold worldwide [2]. Robotic vision systems can be built using RPIs and low-cost cameras. Many DIY materials on the internet show how to create simple object recognition and detection systems [3]–[5].

Object recognition is not a problem nowadays: different ML models outperform humans in object recognition accuracy [6], [7], and speed [8]. Nevertheless, they are either slower in the first case or not as accurate in the second case. Some commercial "Smart" cameras exist that have both advantages at the cost of a high price.

This paper aims to develop a prototype of a machine vision system that will consist of relatively cheap components and recognize the objects located in front of a camera. This system has to work quickly to timely decide how to grab the part with a mechanical grip.

2. Related work

There are different approaches to how the ML system decides which object is in front of it and its current position. [9] implemented a two-stage recognition approach. In their work, they used The Cornell Grasping Dataset [10], where the first stage was to detect the object and feed the result to the next model. The second stage defines the gripping area based on the detected object and the corresponding gripping area available for this object. Tsarouchi et al. [11] used a different approach where the object was predefined, but its position varied. They created labels for the predefined positions and used the ML model to extract a binary image and classify the position. This approach looks very robust in serial production, where the objects for the handling are not varying wildly. Besides, this approach utilizes 3D CAD model that corresponds to the position. Based on this information, the system decides how to grab the object. Kaymak et al. [12] implemented the actual performing prototype that was able to recognize objects, grab them and locate them in the desired bin. The drawback of this work is that the objects were simple, and the Z coordinate was fixed. Chatterjee et al. [13] used the modified YOLOv3 model to install it into a low-computing humanoid robot. It was capable of detecting the objects with high accuracy and speed. This work is a practical implementation of machine vision systems for simple and relatively cheap hardware. Pillai [14] developed a SLAM-supported object recognition system that improves the recognition based on time accumulated data. It outperforms the frame-based recognition in accuracy and provides the possibility to use a single camera to calculate the distance and update the information for the grip decision.

3. Methodology

In current work object detection was performed by using 2 frameworks: YOLOv4 and YOLOv4tiny. These frameworks use DarkNet deep neural network (DNN) that sacrifices the accuracy in the cost of inference time. As the task for this project is not complicated and there is no complicated objects and the number of classes is not more than 10, the YOLO is enough to perform the detection successfully. The full YOLOv4 uses 53 layers in the backbone whereas YOLOv4tiny uses only 9 [15]. This brings a much better accuracy to the bigger YOLO than the smaller but on contrary weights file is 10 times smaller for the tiny version.

3.1. Data collection









The first step in creating a custom classifier is to collect the data. For this work I used 8 different types of Mega Blocks™ provided to me by Dr. Starly. The list of blocks and corresponding labels is in the table 1. The pictures were collected via smartphone Samsung Galaxy S21 Ultra with different backgrounds. There were approximately 30 pictures per each separate block with all different positions. Additionally, 46 pictures were made where all 5 objects were presented together. All pictures have different lighting, camera position and object positions. Labeling the pictures was performed on the platform roboflow.com, using their built-in labeling tool. This platform allows to resize all the images and create formatted label file specifically for the YOLO DarkNet format. Also, roboflow.com allows to increase the dataset by adding modified source images. Modifications include Static Crop, Tile, Grayscale, Flip, Shear, Blur, Noise, etc. For this stage of the work none of the modifications were made and the dataset contained 200 images total.

The training of the models was performed using Google Colab platform and the Jupiter notebook created by Roboflow for YOLOv4 and YOLOv4 tiny: <https://colab.research.google.com/drive/1PkffxcLlp6tuAI7iW5bjDADI0RupSTvh>. Generated weights files were 24 Mb and 244 Mb for the full and tiny versions respectively. It took

approximately 1.5 hours to train YOLOv4tiny with mAP = 98% and 4 hours to train YOLOv4 with the same mAP.

The Logitech web camera was attached to the UR5 robot and connected to the Raspberry PI (fig.1). The feed from the web camera was used to capture the images of the objects that were located under the robotic gripper. The image quality of the webcam was much worse than from the cell-phone and I had to retrain the model, adding more images made by the webcam on the working table. This time the dataset was augmented by 85 images and contained more than 600 instances with additional Mega Blocks™. This dataset also contained instances augmented by roboflow.com via blur, noise and light.

Table 1. List of objects.

Picture	Label	Description
	1x1_T_BL	1x1 Tall block of blue color
	1x2_L_G	1x2 Low block of green color
	2x2_L_Y	2x2 Low block of yellow color
	1x3_T_R	1x3 Tall block of red color
	1x4_T_LB	1x4 Tall block of light blue color
	1x4_L_LB	1x4 Low block of blue color
	1x4_L_G	1x4 Low block of green
	2x2_L_R	2x2 Low block of red color

3.2. Inference

The inference of the detector was performed using the OpenCV package in Python. The code allows to feed the image file, video file or the webcam stream to the detector and add receive the output file. The detector code was run on a desktop computer. The output file is video or image with labeled bounding boxes and confidence intervals. When inference was performed using the camera from Galaxy S21 Ultra the performance was pretty high: only few times detector could not recognize some objects if they were all together. When the webcam was used the miss rate was much greater.

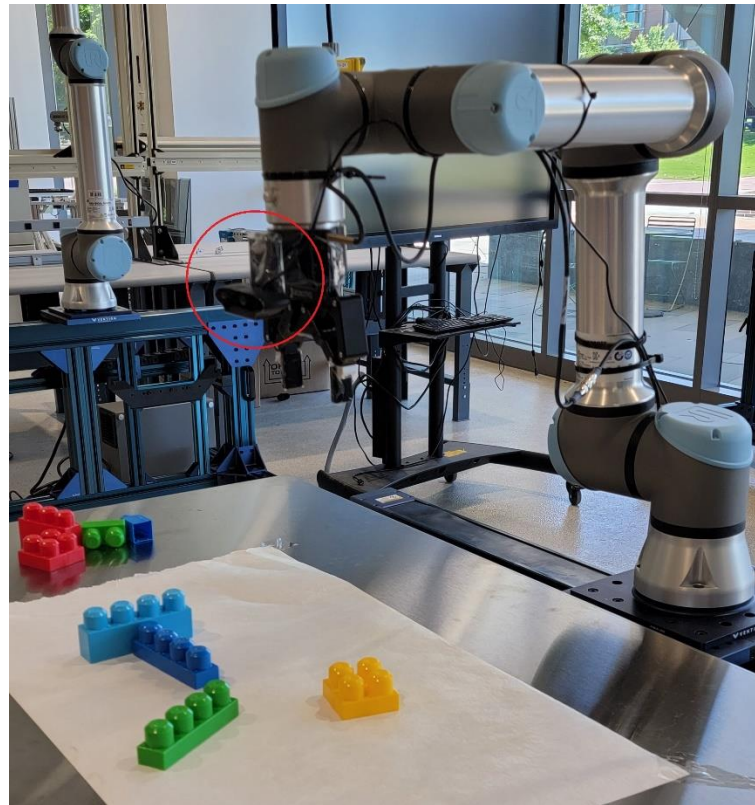


Fig. 1. UR5 robot with attached Logitech webcam (circumscribed in red)

After the objects were detected, the coordinates and sizes of the bounding boxes were used to convert them into X and Y coordinates for the robot input. In order to rotate the gripper, the angle between camera vertical line and the edge of the block needs to be calculated. Edge detection was performed by applying Canny edge detector from OpenCV and specific threshold levels. Contour finding with approximate vertices was used to identify objects with 4 or more vertices. These vertices were used to calculate the angle between the edge and the vertical line.

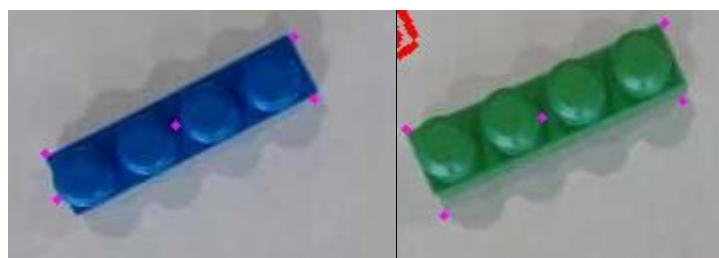


Fig. 2 Examples of detecting the center point and the vertices of the block using canny edge detection, and contour finding in OpenCV.

After the coordinates of the vertices are defined it is possible to calculate the angle via the formula (1):

$$\alpha = \cos^{-1} \frac{\bar{a} * \bar{b}}{|\bar{a}| * |\bar{b}|} \quad (1)$$

where \bar{a} and \bar{b} are the vectors of the two adjacent vertices.

After the coordinates of central point and the angle are calculated this data is send to the robot to perform the move and grasp operation.

3.3. Robotic control

UR5 robots can be controlled from the PC and from pendant. The initial home position of the robot was adjusted by the pendant (fig. 1) and the coordinates were saved in a HOME variable. Universal Robots is using their own language URScript to control the joints. Python library Socket is used to send the text command to the robot. This text contains URScript code. Coordinate variable is a vector of six values each of them is an angle in radians corresponding to the joint. The coordinate origin was manually set as a point where the robotic gripper in a closed position touches the location of the zero pixel of the image made from the home position. All consecutive moves are relative to the origin.

The gripper used in this project is Robotiq 2F-85 that has controls on opening value, speed and force of the grasp. This gripper was connected to the UR% controller through the RS-485 and USB converter. Control of the gripper was performed by using *urx.robotiq_two_finger_gripper* library in Python [16].

4. Results

Tiny YOLOv4 model shows the comparable to the full model results on an images and the videos where only the one object is located. When more than 3 objects are present the smaller model cannot recognize some of the objects or has a low confidence level. The full model performs much better than the smaller version but requires more computational resources. This can be crucial for the performance of the Raspberry PI.

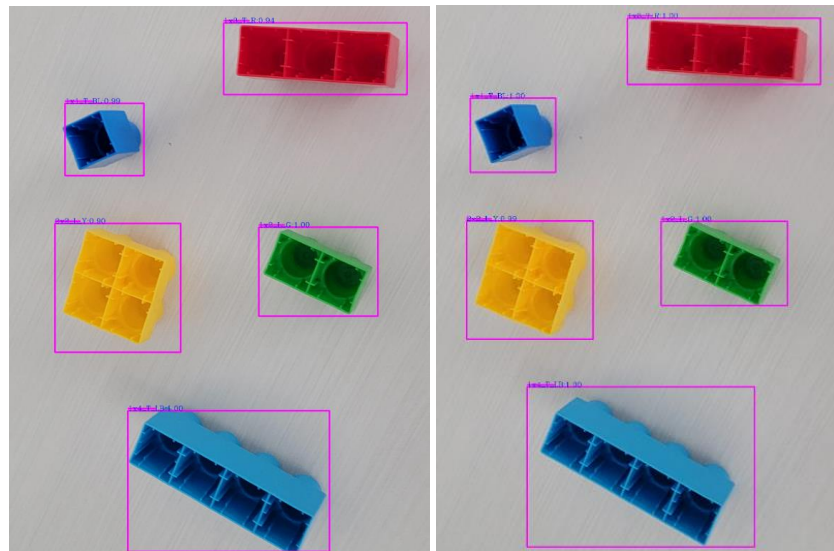


Figure 3. The result in confidence level for the YOLOv4Tiny and YOLOv4: the confidence level for the full YOLOv4 model is 100% for all the blocks except 2x2_L_Y (99%), whereas YOLOv4Tiny has 100% only for the 1x2_L_G and 1x4_T_LB (1x3_T_R 94%, 1x1_T_BL 99%, 2x2_L_Y 90%).

The results of the object detection can be used to perform the next stage – position detection. In this stage the bounding box becomes the input for the next stage of detection, where the model detects the positions of the block and gives a command to the robot to rotate its grip to align with the edges of the object.

The detector is capable of detecting the objects but there are some limitations. When the lighting changes or the background is reflective, the model has troubles in detecting the objects. Also, some blocks are not recognized when they are positioned at a specific angle. This can be eliminated by creating a much bigger dataset with a greater variance. 3D data can also be used to assist the creation of the dataset that can augment real-life dataset.

The system based on UR5 robot and Raspberry PI is capable to detect objects and position the gripper in accordance to the position of the objects. There are still some drawbacks in this approach that need to be addressed. The threshold values of the canny edge detector need to be self-adaptable to the environment, or some other approach is needed to build a robust edge detection. There is a difficulty if 2 blocks are touching: the bounding box will capture a piece of the adjacent block and the edge detector will capture it, leading to the incorrect angle calculation. This can be overcome by applying the color filtering for the blocks of different hues, but if block are of the same color it will not work.

5. Conclusions

Current level of technology is capable to give small companies an opportunity to use computer vision and robotics on the relative low price. Free machine vision frameworks and low-cost computers can be implemented in the control of the robotics. There are some difficulties that hinder the development of the robust frameworks but they can be overcome via thorough brainstorming and finding ways around to overcome the hardships.

This project shows that working model of the adaptive robot assembly is possible with a little investments. Further work in this project can lead to a useful product that can be used to upgrade robots to the cobots that will utilize the computer vision to assist human workers. There are still some improvements needed or even whole concept needs to be redesigned to create a robust system that will have critically low error rate and self-adapt the detection depending on the surrounding environment.

Acknowledgements

The author wishes to thank Dr. Binil Starly and the ISE Department for the opportunity to use the equipment and help in overcoming the hardships in finding the solutions to the problems that were faced during the work.

References

- [1] K. O'Marah, "Mass Customization and the Factory of the Future," *IndustryWeek*, Jan. 14, 2015. <https://www.industryweek.com/supply-chain/article/22008141/mass-customization-and-the-factory-of-the-future> (accessed Mar. 11, 2021).
- [2] "The Impact of Raspberry Pi," *GeeksforGeeks*, Jan. 31, 2020. <https://www.geeksforgeeks.org/the-impact-of-raspberry-pi/> (accessed Mar. 11, 2021).
- [3] "How to easily Detect Objects with Deep Learning on Raspberry Pi," *AI & Machine Learning Blog*, Nov. 14, 2018. <https://nanonets.com/blog/how-to-easily-detect-objects-with-deep-learning-on-raspberry-pi/> (accessed Mar. 11, 2021).
- [4] "Create a real-time object tracking camera with TensorFlow and Raspberry Pi | Opensource.com." <https://opensource.com/article/20/1/object-tracking-camera-raspberry-pi> (accessed Mar. 11, 2021).
- [5] "Running TensorFlow Lite Object Recognition on the Raspberry Pi 4," *Adafruit Learning System*. <https://learn.adafruit.com/running-tensorflow-lite-on-the-raspberry-pi-4/overview> (accessed Mar. 11, 2021).
- [6] "What I learned from competing against a ConvNet on ImageNet." <http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/> (accessed Mar. 11, 2021).
- [7] R. Geirhos, D. H. J. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann, "Comparing deep neural networks against humans: object recognition when the signal gets weaker," *arXiv:1706.06969 [cs, q-bio, stat]*, Dec. 2018, Accessed: Mar. 11, 2021. [Online]. Available: <http://arxiv.org/abs/1706.06969>
- [8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *arXiv:1506.02640 [cs]*, May 2016, Accessed: Mar. 11, 2021. [Online]. Available: <http://arxiv.org/abs/1506.02640>
- [9] F. H. Zunjani, S. Sen, H. Shekhar, A. Powale, D. Godnaik, and G. C. Nandi, "Intent-based Object Grasping by a Robot using Deep Learning," in *2018 IEEE 8th International Advance Computing Conference (IACC)*, Dec. 2018, pp. 246–251. doi: 10.1109/IADCC.2018.8692134.
- [10] "cornell_grasp." <https://kaggle.com/oneoneliu/cornell-grasp> (accessed Mar. 19, 2021).
- [11] P. Tsarouchi, S.-A. Matthaiakis, G. Michalos, S. Makris, and G. Chryssolouris, "A method for detection of randomly placed objects for robotic handling," *CIRP Journal of Manufacturing Science and Technology*, vol. 14, pp. 20–27, Aug. 2016, doi: 10.1016/j.cirpj.2016.04.005.
- [12] C. KAYMAK and A. UCAR, "Implementation of Object Detection and Recognition Algorithms on a Robotic Arm Platform Using Raspberry Pi," in *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)*, Sep. 2018, pp. 1–8. doi: 10.1109/IDAP.2018.8620916.
- [13] S. Chatterjee, F. H. Zunjani, S. Sen, and G. C. Nandi, "Real-Time Object Detection and Recognition on Low-Compute Humanoid Robots using Deep Learning," *arXiv:2002.03735 [cs, stat]*, Jan. 2020, Accessed: Mar. 05, 2021. [Online]. Available: <http://arxiv.org/abs/2002.03735>
- [14] S. Pillai and J. Leonard, "Monocular SLAM Supported Object Recognition," *arXiv:1506.01732 [cs]*, Jun. 2015, Accessed: Mar. 19, 2021. [Online]. Available: <http://arxiv.org/abs/1506.01732>
- [15] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv:2004.10934 [cs, eess]*, Apr. 2020, Accessed: Apr. 18, 2021. [Online]. Available: <http://arxiv.org/abs/2004.10934>
- [16] "SintefManufacturing/python-urx," *GitHub*. <https://github.com/SintefManufacturing/python-urx> (accessed May 13, 2021).