AI Application in Housekeeping – Shoe Sorting Robot

Yuan Zhang
Faculty of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
vanilla.zhang@mail.utoronto.ca

YuHaoWang
Faculty of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
billyuhao.wang@mail.utoronto.ca

Yujie Hua
Faculty of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
yujie.hua@mail.utoronto.ca

Wenhao Li
Faculty of Mechanical & Industrial Engineering
University of Toronto
Toronto, Canada
wenhaowh.li@mail.utoronto.ca

Abstract**—***This paper presents an AI-based robot that is able to detect shoes, pair shoes, plan its path to the shoe rack and plan its manipulator trajectory in the task space. For each task, the team evaluates different potential methods and finally chooses one specific approach in the area. The implemented algorithm for each task is YOLO v3 for shoe detection, SIFT algorithm for shoe pairing, SLAM for map generation, RRT* for path planning and RRT for robot arm trajectory planning. By integrating the code online with the project, the group came up with a design of the shoe sorting robot that can assist the user to place the unsorted shoes into a shoe rack or a designated area that is previously assigned by the user in pairs.*

Keywords**—***shoe shorter, YOLOv3, SIFT, RRT, RRT**

1.INTRODUCTION

Artificial intelligence techniques are commonly applied in people's lives. The use of artificial intelligence can help people in a variety of scenarios. Usually, using AI, robots are trained for given tasks. After a series of training, the AI agent is able to finish the assigned tasks individually. Some of those tasks can be as easy as a simple command: taking an object and transferring it to the targeted destination, while some of the tasks can be complex. Considering the working environment, an AI robot can even be designed to do eyeball surgeries that humans are not able to finish by dsoctor's hands in the future. Actually, AI robot applications are ubiquitous.

On the other hand, creating a housekeeping robot that can help users sort messy shoes and placing the shoes on a shoe rack to improve life experience is in demand. In this project, the group has designed a shoe sorting robot, applying various AI techniques, to accomplish assigned tasks. The major challenges that the group faced are extracting shoes' 3D orientations from 2D images, dealing with moving obstacles while navigating through the environment, handling various types and shapes of shoes, optimizing overall processing speed.

In the project, AI techniques are used for image processing, path planning, autonomous navigation, and robot arm control. For each aspect, different algorithms and methods are proposed and compared to finalize the decision of the best-fit techniques for this project. The first task is detecting the target unsorted shoe and finding its grasping point location. The Second task is pairing shoes and saving pairing data for the robot arm control algorithm to place shoes in order. Then, generating a collision-free path that minimizes the length of the route is defined as the third task. The fourth task is using autonomous navigation techniques to perform real-time adjustments, considering the dynamics of the surrounding space, to avoid collision with unexpected obstacles. Finally, the last task that the robot needs to perform is grasping the detected shoe using the gripper and placing it on a shoe rack.

2.LITERATURE REVIEW

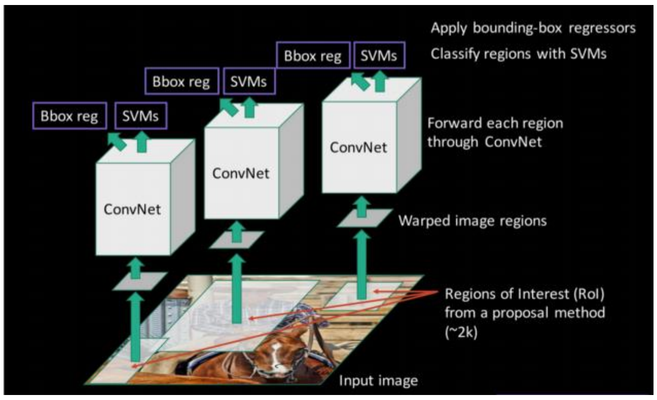
2.1 Shoe Detection

2.1.1 R-CNN

The recent research on object detection all starts with Region-Based Convolutional Neural Networks (R-CNN) which, as the name suggests, is based on generating regions and feeding into a convolution neural network. The algorithm will produce a set of bounding boxes as output, where each bounding box contains the object and its category.

R-CNN was proposed by R. Girshick in 2014 [1]. They combined region proposals with CNNs, which boosted the mean average precision (mAP) by more than 30% compared to their last solutions.

R-CNN consists of three basic modules. The first module generates region proposals which divide the input pictures into thousands of regions that are recognizable for the detector. The method used is selective search, and compared with other generating region proposals methods such as objectness, category-independent object proposals, constrained parametric min-cuts (CPMC), selective search is more simplified and is able to capture all scales [2]. The second module is the Convolutional Neural Network that extracts all kinds of feature vectors from the previous region proposals and then passes them to the next module to classify. The last module is a set of class-specific linear support vector machines to perform the classification as well as the bounding box generator. The framework diagram of R-CNN is shown as follows in figure 1.



Apply bounding-box regressors
Classify regions with SVMs
Bbox reg
SVMs
Forward each region through ConvNet
ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Fig. 1. A simplified framework diagram of R-CNN. Three important modules' functions are region proposal generation, feature extraction using network and classifier.

Overall, there are four steps during the R-CNN algorithm shown in Fig. 2. below: The first step is the input of the test image; after that, the system uses Selective Search algorithm to extract around two thousand region proposals per image; and then the proposed regions are resized into a 227*227 pixel size form which is compatible with CNN and use a specific layer 'fc7' as output, and each fc7 output is further used as input to the SVM to perform the classify; finally the system calculates the bounding box regression and generates the predicting bounding box as well as the predicted class as final output.

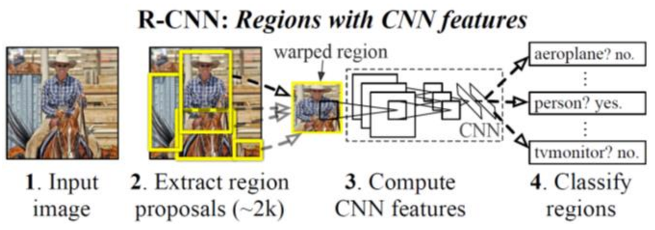
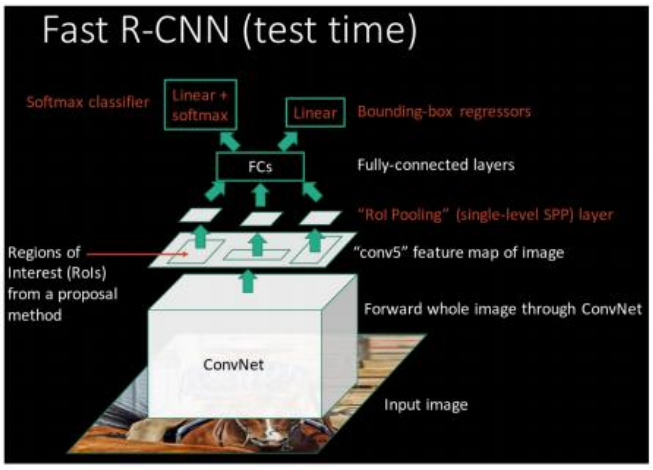


Fig. 2. The system (1) takes input image, (2) extracts around two thousand region proposals, (3) input those proposals into CNN to generate features, and (4) uses SVM to classify the image [3]

However, the drawbacks of the R-CNN method are obvious: since CNN, SVM, and bounding box regression all require training, it means that the training time could be enormous. In fact, the training time for their R-CNN is 84 hours. Each picture extracts thousands of region proposals which then creates thousands of feature vectors, it needs a huge amount of hard disk storage space and time. Because of the high level of calculation, the average time for R-CNN to classify a single picture is around 47 seconds which is impossible for real-time detection.

2.1.2 Fast R-CNN

With these crucial drawbacks, Ross Girshick came up with an enhanced version of R-CNN called fast R-CNN in 2015 [4]. In R-CNN, when extracting features, it uses every reshaped form of the proposal region as input which is required to calculate repeatedly for thousands of times. However, in fast R-CNN, it only uses the original pictures as input to extract features and thus significantly reduces the feature vectors' production time. Fast R-CNN unified the entire image input and fed it into the convolution network. The fifth pooling layer of the network is changed into a special Region of Interest (RoI) pooling layer. The function of this layer is it can change the size of the input regions to the same size of regions. After the RoI layer, it will then extract the features of the input and then generate the feature map. In a word, the R-CNN needs to calculate thousands of times to get the final feature map of the input while the Fast R-CNN only needs to be calculated once. Thus, the detection time of Fast R-CNN is greatly reduced. Follow is the structure of the Fast R-CNN in Fig. 3.



Fast R-CNN (test time)
Softmax classifier
Linear + softmax
Linear
Bounding-box regressors
FCs
Fully-connected layers
"RoI Pooling" (single-level SPP) layer
Regions of Interest (RoIs) from a proposal method
"conv5" feature map of image
Forward whole image through ConvNet
ConvNet
Input image

Fig. 3. A simplified framework diagram of Fast R-CNN. Notice that one distinct change is the switching between the region proposals generator and the network. Without the calculation of two thousand times, the speed of the algorithm will be greatly improved [4].

Moreover, without the process of extracting every single proposal region, it does not need to write into the hard disk frequently and thus reduces both time and storage space. Another improvement is the integration of the bounding box regression into CNN. It forms a multi-task model and throughout the test session, it turns out the final result is satisfying as well. As a result, it can train the CNN with bounding box regression together which reduces the training time. It even uses the SoftMax function to substitute the SVM classifier.

With the above vital changes, the final outcome is quite satisfying with a reduction of detection time to around 3 seconds. Also, the training time was reduced to 9.5 hours.

2.1.3 Faster R-CNN

Although the detection time is greatly reduced, is it still impossible to make real-time detection. The bottleneck of the algorithm so far is due to the selective search method which requires a huge amount of time to generate region proposals. As a result, in the same year, Ross Girshick with Shaoqing Ren developed a faster R-CNN which has the final detection rate of 5 fps and is almost achievable for real-time detections [5].

The faster R-CNN uses a complete network to generate region proposals: Region Proposal Network (RPN). The implementation of an entire network is much faster than the selective search algorithm.

The input of the RPN is the feature map of the image and the algorithm can output the region proposals to the RoI pooling layer. There are three steps inside the RPN: The first step is to generate the anchor boxes. The anchor box is a set of coordinates that contains 9 different rectangular boxes around every point in the feature map. With the 9 different boxes for each point, it will then classify into two classes which are foreground and background. The foreground is the object that needs to be detected. The last step is to adjust the anchor boxes that contain the foreground to the ground truth in terms of Interest over the Union.

The traditional methods of generating region proposals are all rather time-consuming because they all use Sliding Windows over the input or Selective Search. However, with the RPN, the detection speed is greatly reduced. With the region proposals output of the RPN, these region proposals are then reshaped using an RoI pooling layer which is then used to classify the image and predict the offset values for the bounding boxes. Following is the simplified structure diagram of the Faster R-CNN in Fig. 4.

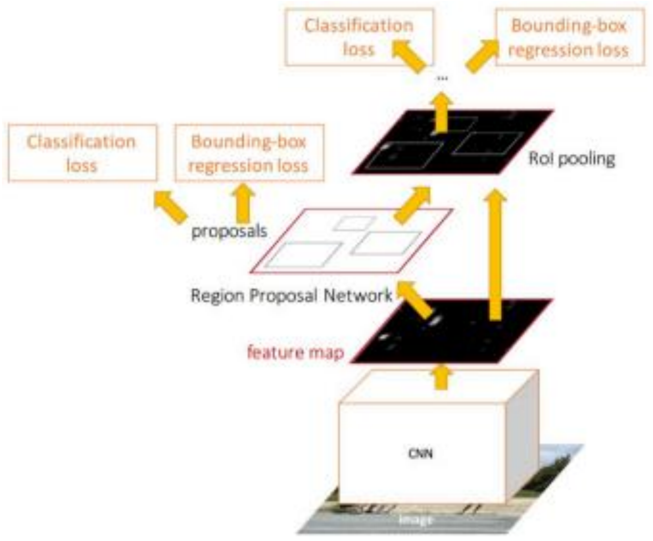


Fig. 4. The simplified framework diagram of Faster R-CNN. It uses a network instead of a Selective Search algorithm to generate region proposals hence enhancing the speed [5].

As a result, Faster R-CNN has a detection speed of 5 fps which is nearly a thousand times faster than the original R-CNN algorithm. With the development of object detection using machine learning, the researchers are able to keep pushing the envelope in the computer vision area.

2.1.4 YOLO

In 2016, Joseph Redmon presented a new approach based on the R-CNN family for object detection: You Only Look Once (YOLO) [6]. Although the mAP of YOLO is around 58% which is slightly lower than its ancestors, the first version of YOLO can achieve an unprecedented detection speed of 45 fps which makes object detection in real-time become possible.

The detection of YOLO can be divided into three parts as shown in the following Fig. 5.: YOLO resize the input pictures to 448*448 pixel size; process the image to a single convolution layer which predicts both multiple bounding boxes and class probabilities at the same time; and finally use the model's confidence to threshold the detections. One of the reasons why YOLO can achieve a high speed of detection is that it only has one simple convolution network evaluation process.

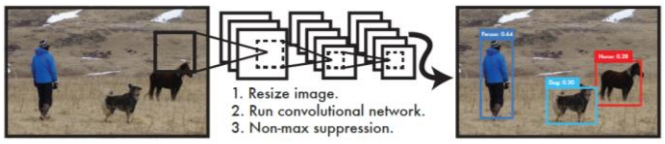


Fig. 5. YOLO's three steps in detecting objects: (1) resize the input into fixed size, (2) input those images to the network, and (3) use the object confidence to classify [6].

Modeling the detection process as a regression problem, YOLO is much faster than the traditional methods of object detections since no complex pipeline is needed. Not using the traditional sliding windows and region proposal method as its ancestors, YOLO sees the entire image during the process, thus it can relate the contextual information together with their appearance. Another result of seeing the entire image is that it can separate the background finely from the object. Also, YOLO's ability of generalization outperforms R-CNN and other detection methods. It even can extract special features in the artwork. However, YOLO still falls behind in terms of accuracy, especially the small items. This might be one of the drawbacks of seeing the problems as a simple regression problem and discarding the sliding windows method.

Following is the convolution structure of YOLO, inspired by the GoogLeNet model [7]. The input is the resized 448*448 pixel picture. There are several convolution layers as well as max pool layers which transform the input into a 7*7*1024 tensor. After that, there are two fully connected layers that continually convert the tensor into a 7*7*30 tensor as the final output of the network. This process is shown in Fig. 6 below.

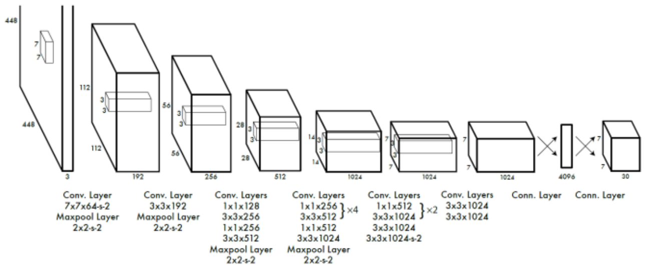

448
448
3
112
112
192
56
56
256
28
28
512
14
14
1024
7
7
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }×4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }×2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer

Fig. 6. The architecture of the convolute network in YOLO. There are two fully connected layers at the end after 24 convolute layers [7].

In the following Fig. 7, the YOLO system divides the input image into an S × S grid. If the center of an object falls in a grid cell, that grid cell is responsible for detecting that object. All the other grid cell that contains only part of the object will not be used for detection. For every single detection cell, there are (B*5 + C) parameters that are required for detection. B is the number of bounding boxes that need to be predicted, which is 2 in this case. 5 is the number of predictions of every bounding box which is x, y, w, h, and confidence. X and y coordinates are the center of the bounding box relative to the bounds of the grid cell. The w (width) and h (height) are predicted relative to the whole image. Finally, the confidence prediction represents the intersection over union (IoU) between the predicted box and any ground truth box. C is the conditional class probabilities.

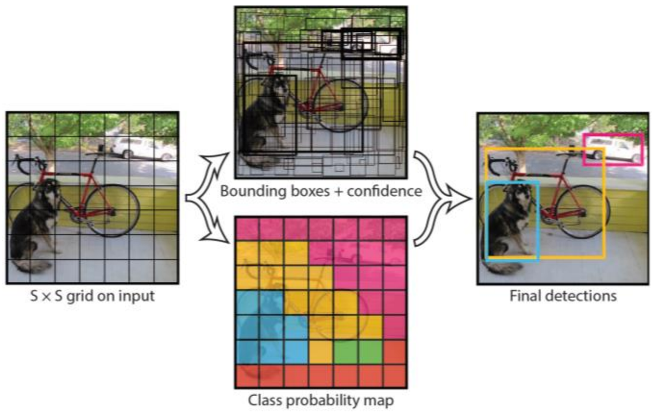


Fig. 7. The model regards the problem as a simple regression task. The input is divided into an S*S grid and for each cell grid. It predicts two bounding boxes, each with 5 parameters. C is the class probability. The output of the network is an S × S × (B* 5 + C) tensor [7].

Since YOLO treats the problem as a regression problem, there is a loss function that needs to be minimized for the algorithm. Following is the loss function for YOLO, where the first term is the loss for x and y coordinates, the second loss is for h and w, the third and fourth are the loss for IoU and the last term is the classification loss.

$$L = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i\,j}^{obj} \left[\left(x_i - x_j\right)^2 + \left(y_i + y_j\right)^2 \right]$$
$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i\,j}^{obj} \left[\left(\sqrt{\omega_i} - \sqrt{\hat{\omega}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right]$$
$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i\,j}^{obj} \left(C_i - \hat{C}_i\right)^2$$
$$+\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{I}_{i\,j}^{noobj} \left(C_i - \hat{C}_i\right)^2$$
$$+ \sum_{i=0}^{S^2} \mathbb{I}_{i\,j}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2 \quad (1)$$

However, YOLO still has some of its limitations: since each grid cell in the model only predict two bounding boxes and only one class, it limits the number of detection objects. YOLO struggles with small objects as well. Another limitation exists from the loss function. It did not take the relative size of the error in different sizes of bounding boxes. A small error in a big bounding box might be negligible, but the same size of an error in a small bounding box could have a serious effect in terms of IoU.

2.1.5 YOLOv3

As time goes by, the author keeps improving the model and releasing YOLO v3 in 2018 [8]. One of the big improvements is that it changes the base net to Darknet-53 based on Darknet-19 in YOLO v2, which already improves the prediction compared with the original version. Darknet-53 has 53 convolutional layers, it's deeper than YOLOv2 and it also has residuals or shortcut connections. It's much more powerful than the Darknet -19.

Without the limit of the input in the second version as well as the third version, YOLO can detect different sizes of input. The network even randomly selects different sizes of images as input every 10 batches during the training process which makes the model more capable for different image inputs.

In order to improve its detectability for a small object, the YOLO v3 uses three different scales for prediction which is 13*13, 26*26, and 52*52, as shown in the following figure 3.1.5.1. With multi-scale prediction boxes, YOLO v3 is able to predict smaller objects which is hard for its ancestor.

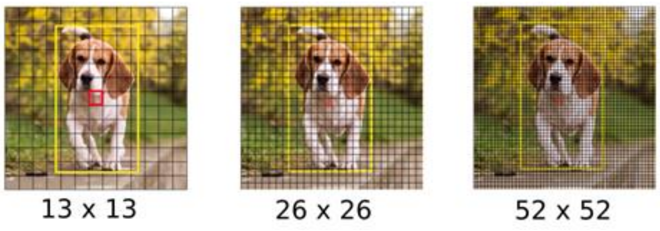
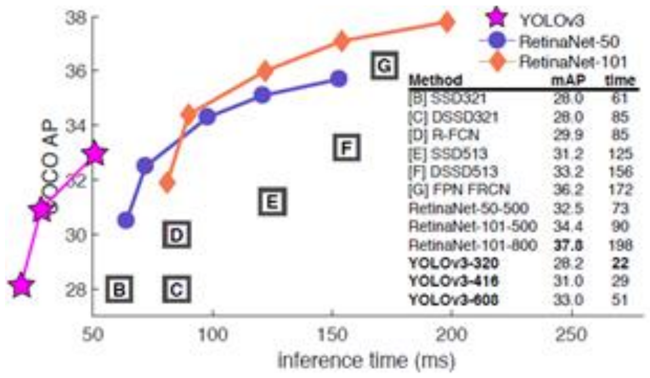


Fig. 8. Three different scales of YOLO v3 detection. With different scales, YOLO v3 gains the ability to better predict at varying scales [8]

With other adjustments such as changing the loss function from SoftMax loss to logistic loss to produce multiply class during detection, activation function change from SoftMax to sigmoid function and changing the number of bounding boxes in every grid cell from 5 to 3 hence increasing the IoU, the final result of YOLO v3 is rather satisfying. Under the same mAP, YOLO v3 detection time is 3 times less than Single Shot Multibox Detector (SSD) method. The comparison of different methods as well as YOLO v3 is shown as following Fig. 9.



Method	mAP	time
[B] SSD321	28.0	61
[C] DSSD321	28.0	85
[D] R-FCN	29.9	85
[E] SSD513	31.2	125
[F] DSSD513	33.2	156
[G] FPN FRCN	36.2	172
RetinaNet-50-500	32.5	73
RetinaNet-101-500	34.4	90
RetinaNet-101-800	**37.8**	198
YOLOv3-320	28.2	**22**
YOLOv3-416	31.0	29
YOLOv3-608	33.0	51

Fig. 9. Comparison of different methods [8]

2.2 *Shoe Pairing*

We use the SIFT algorithm to perform shoe pairing based on local key points.

SIFT stands for Scale Invariant Feature Transform, it was proposed by David Lowe in 1999 [9]. This kind of approach can generate key locations that are invariant from image translation, scaling, rotation, and transform input images into scale-invariant coordinates relative to local features.

There are four major steps in this algorithm: First is to detect extrema in scale space using a Difference-of-Gaussian function (DoG). The difference-of-Gaussian function is a special type of Gaussian Pyramid. The Pyramid is a set of images that are downsampled from an initial image and the

Gaussian Pyramid uses Gaussian blur with different standard deviations to further change each layer of downsampling image. By subtracting the adjacent layer of the Gaussian Pyramid, we can get the difference-of-Gaussian function as the following Fig. 10.

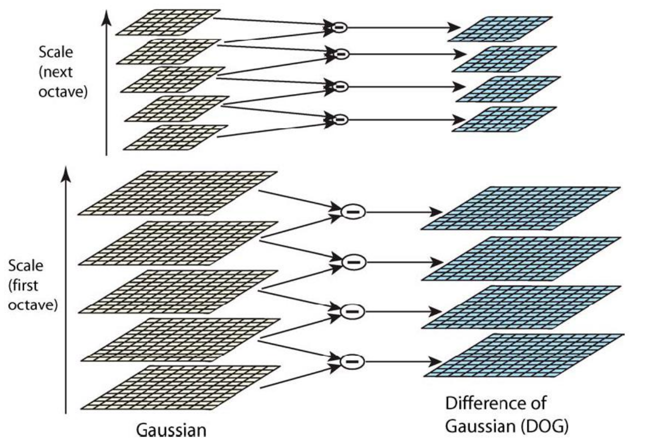


Fig. 10. The structure of difference-of-Gaussian function. Each octave is the difference between two adjacent layers in the Gaussian Pyramid [9].

When the DoG of the input image is found, the algorithm detects the local extrema. In the following Fig. 11, the X represents local minima or maxima in its 3*3*3 pixel surroundings and thus this pixel is picked out for further calculations.

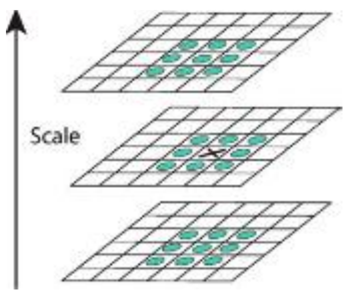


Fig. 11. Extrema of the difference-of-Gaussian images are detected by comparing a pixel to its 26 neighbors in 3×3 regions at the current and adjacent scales [9].

The second step is to localize the key points. We can only find the relative position of the extrema in the discrete scale space. Thus, we need to fit a detailed model to the points so that all the points we find become consistent. Also, the low contrast point, noise-sensitive points, or edge points can be deleted during this process.

After that, the algorithm assigns the orientations. In order to make the feature invariant to orientation, it required a consistent orientation for each key point. Based on the gradient direction of the image, one or more consistent orientations are assigned.

Finally, the key points are described. By gridding the adjacent area of the key point, the algorithm calculates the gradient histogram and generates a unique representation vector, thus representing the unique feature of this area.

The reason why the SIFT algorithm is scale-invariant is the implementation of DoG. Since it contains different down-sampled scales of images, it can simulate different scale conditions which can be detected and matched by the algorithm. During the orientation assignment step, the algorithm assigns a consistent orientation for each key point. As a result, the SIFT features are invariant to rotation as well.

2.3 Autonomous Navigation

2.3.1 Autonomous navigation, localization and path planning

A Star is one of the most popular path planning algorithms widely used for mobile robots. The use of the A-Star algorithm is based on a functional SLAM that feeds a map as the input. The job of A Star is to generate a path that is represented by the connections between points of interest. In this paper, several add-ons and modifications of A Star are introduced. Basic Theta* algorithm is implemented as an extension to A Star to eliminate zigzag connections between points of interest by ignoring all connections between visible cells. Phi* algorithm is used to allow the restricted definition of robot dynamics. To reduce the computational time that A Star needs, Rectangular Symmetry Reduction (RSR) was introduced to only consider cells that are near the edge of rectangle spaces, which can reduce the processing time by ten to a hundred times. Jump Point Search (JPS) was also introduced to reduce the computational time. It is done by searching the cells around corners and finding visible paths to connect them. It is much faster than the traditional A Star algorithm, though it cannot search in every angle like Basic Theta* and Phi*. Comparing these A-Star-based algorithms, JPS appears to be the most efficient one in that it always finds an almost optimal path with the least computational time and examined cells, though the paths it finds are always a little longer than the ones the other algorithms find. In conclusion, JPS is suitable for real-time path planning problems in which the length of the path is not especially important. If the length is important, Basic Theta* is more likely to be the best fit [10].

The paper “Path planning for indoor Mobile robots based on deep learning” introduces a path planning method based on the classification of obstacles [11]. GoogLeNet is used to classify obstacles into static ones like furniture, and dynamic ones like animals. Once static obstacles are detected, they are directly marked on a map. However, dynamic obstacle avoidance is completed by a proposed waiting rule or by path planning that takes the speed of the moving obstacle and the speed of the robot into account. RRT is compared with GA and PSO in a static environment, and it is compared with APF in a dynamic environment. In conclusion, using an RGB camera image of detected obstacles to classify obstacles can give the system more potential to plan better paths when there are moving objects in the area.

Rapidly random-exploring tree (RRT) is another one of the most popular path planning algorithms. It explores the free spaces very quickly and efficiently, but it takes a long time to converge to the optimal solution. An A*-RRT* algorithm was introduced by Brunner to combine the advantage of both the A* and RRT* algorithms. It uses A* to generate an optimal path but with a low resolution and possibly a zigzag one. This paper introduces another method that overcomes the disadvantages and further takes advantage of the A* and the RRT* algorithms by training a convolutional neural network model by learning many optimal paths generated from the A* algorithm. A clearance distance that sets the minimum distance the robot can be from the nearest obstacle is added as a variable to ensure safety. Eventually, neural RRT* (NRRT*) is introduced to achieve nonuniform sampling in the path planning process which performs much better than both A* and RRT*[12].

The paper “Simultaneous localization and mapping: part I,” introduces the fundamentals and historical development of simultaneous localization and mapping (SLAM). Basically, SLAM is used on an autonomous robot that is placed at an unknown location in an unknown environment to incrementally build a consistent map while simultaneously determining its location. The problem of SLAM can be considered solved already and its application is becoming more and more impressive. During the construction of a map, as more and more landmarks are being observed, the correction that SLAM makes between landmarks estimates increases monotonically, which makes the map more and more accurate and reliable. Through the use of the extended Kalman filter, EKF-SLAM records uncertainties throughout an operation to avoid the system being over-confident. Another variant of SLAM ------ FastSLAM uses Rao-Blackwellized particle filters to become more robust to incorrect associations due to its local registration of landmarks[13].

Through deep reinforcement learning, this paper introduces an end-to-end network architecture for robots to learn collision-free autonomous navigation. Visual simultaneous localization and mapping (V-SLAM) often depend a lot on good lighting conditions and a good diversity of visual landmarks. However, image-based autonomous navigation provides greater potential for 3D-environment recognition than the traditional laser range sensing method since it contains much more information. The paper proposes a model of layer normalization dueling double deep Q-network (LND3QN) that can be transferred to a real robot with little fine-tuning after being trained in a simulated environment. Using the LND3QN method, the training process is accelerated by applying layer normalization before each convolutional layer. CNN was used to extract out the features from the four consecutive depth images while Q-values were calculated from the features. The proposed LND3QN model had much better performance compared with the other baseline models in terms of reward and variance [14].

Since YOLO is a practical technique for object detection and grasp pose prediction, training the embedded CNN to learn to point out the targeted object, while in the scope of the project is shoes, with bounding boxes is implemented. The inspiration, from Tong et al. in the paper, “Keypoint-Based Robotic Grasp Detection Scheme in Multi-Object Scenes,” gripping points can also be trained under supervised learning [15]. Similar to teaching the system to detect shoes, drawing bounding boxes at the gripping points can lead to successful training for the system to tell where the location should be grasping the shoes with the robot’s gripper as well. With further discussion in this article, a visual processing method to detect multiple object types is proposed. Using the VMRD dataset, training obtained reliable detecting accuracies in most objects in that dataset. Testing this algorithm by experiment, the outcome of accuracy is 74.3% and 96.05% for general and VMRD datasets, respectively. The result shows that using the keypoint-based grasping algorithm can efficiently determine the grasping point and generate valid data for processing the following robot arm grasping task [15].

Another paper discussed a similar problem with a different solution[16]. To map any instance from a category of objects in space into a desired set of goal states, the group proposed a novel method, using “semantic 3D keypoints as the object representation.”[16]. By applying instance segmentation, 3D keypoint detection, optimization-based robot action planning and local dense-geometry-based action execution, the group has built a reliable algorithm that by taking advantage of representing key points on the input data, robust mapping accuracy result can be shown with large variation (size, shape, and color) from the dataset used for training. Furthermore, the group has also done two experiments of mugs and shoes to validate their conclusions. From these results, the method is suitable for doing the task of mapping a category of objects to their parent object.

Among all reinforcement learning techniques, Markov Decision Progress (MDPs) is known as the basic method to deal with given tasks. The agent can use MDPs to generate an optimized path in the environment. The traditional MDPs method introduces state, action, and reward functions with given expressions that are defined by the decision-maker, and by maximizing the future cumulative reward, an iterative process is performed. However, Wiesemann et al. have pointed out that MDPs are highly sensitive to distributional model parameters which are unknown and need to be estimated [17]. So, the limitation of this method explains the main error source that leads to decision-making failure. Wiesemann’s group proposed a Robust Markov Decision Processes (RMDPs) to reduce the influence of this aspect. A policy-based method is applied to improve the MDPs method, and unlike the traditional policy evaluation process in two main aspects. Firstly, the variant (RMDPs) uses observation histories instead of the less practical process, building a confidence region using transition sampling. Secondly, instead of using classic approximation, the group developed two novel approximations that avoid destroying vital characteristics of robust MDPs. The result of their experiment of a machine replacement problem shows this method can be used on a large problem scale.

Besides MDPs and Monte Carlo methods, the Kalman filter is also good at estimating an optimum solution for a measured location. In general, the real-time location can be either measured or calculated, and according to Rahman Atif, Kalman filter can be used for a dead reckoning localization process with help of GPS and IMU [18].

The result shows that with the Kalman filter, the simultaneous localization results can be obtained with higher accuracy compared to the trajectory detected by GPS. However, the limitation of using the Kalman filter is obvious. Without the initial position given, only given the position change, Kalman filter is not able to be used to calculate the new position.

2.3.2 Localization algorithms selection

In terms of localization algorithm selection, the most popular algorithms include Monte Carlo localization, Markov localization, and Kalman Filter localization. The selection of these algorithms is based on how the distribution of probability looks like, how demanding it is computationally, and whether there are any assumptions or constraints of the environment. Normally when the shoe sorting robot moves around the space without external intervention, its location can be estimated based on its previous poses, in colder readings, lidar sensor readings and possibly camera readings. In this case, the probability distribution can be represented by Gaussian distribution. However, it is possible that the robot can be moved by the user either in operation or while being powered down. From the perspective of this product, we cannot prevent this from happening and we must give the users freedom of moving it by hand either when the robot is blocking the way or for some reason the user just wanted it to go back to the charging dock without using the app to do that. In this case, the probability distribution of the robot's location can no longer be represented by Gaussian distribution since all locations on the map have equal probability. As a result, Kalman filter localization would not be a good choice as the only localization algorithm on this robot. Of course, we do expect to use only one localization algorithm for simplicity. When comparing the Markov localization algorithm and Monte Carlo localization algorithm, the accuracy, resolution, repeatability, robustness is taken into account. Firstly, Markov localization is grid-based, which means the X, Y and theta results are discretized. Each combination of XY and theta holds an estimation value. On each iteration, all estimation values of all discretized XY and Theta combinations are calculated and stored, which results in a larger and larger usage of memory space and computational power as we increase the resolution of result space. However, the Monte Carlo localization algorithm uses much less memory space since it calculates a predefined number of estimations which are basically combinations of XY and Theta and stores them. The memory usage doesn't go up as the map size is increased. The output estimation is calculated based on the distribution of estimations in the results space where the estimations concentrate at the most. Using less memory space and less computational power compared with the Markov localization algorithm Monte Carlo localization runs at a much faster speed which provides the robot with a faster localization refresh rate.

2.3.3 Path Planning Algorithm selection

The team investigated several path planning algorithms including A*, PRM, RRT and their variances. In terms of efficiency, RRT stood out to be the best because it is able to generate paths very quickly while most of the time it does not generate a short path, but a zigzag one. In terms of shortest paths planned, A* always has the shortest path generated if the resolution of A* is high enough though which can result in higher demand for memory usage and computational resources. Improved based on RRT, RRT* reconnect notes in each iteration to find a shorter path, which produces a more optimal path at the end while it does consume a bit more computational recourse to find and reconnect nodes. By simulating RRT* in MATLAB, the efficiency is proven, and, by tweaking algorithm variables such as the radius to find nodes to reconnect in order to form shorter paths, optimal paths are found with great efficiency. The results are shown in the Methodology section.

2.4 Robotic Arm Planning

Multi-tasks are required for the AI-based robot arm control design: picking and reorienting the shoes, placing them in appropriate positions without collision based on the embedded logic. Many approaches could be used for training robot manipulation arms to accomplish the mentioned tasks and sampling-based planners, multi-query planners, grasping approaches and machine learning are popular methods.

2.4.1 Sampling-Based Planners

Sampling-based algorithms save a large number of computations by avoiding explicit obstacles construction in the state space, providing information about candidate trajectories feasibility, and connecting sets of points sampled in obstacle-free space to build feasible trajectories roadmaps. The general structure/primitives of a sampling-based planner are illustrated in Fig. 12 [19]. Within the planner, the Sampling process, randomly selecting a configuration and then growing it to the roadmap or tree, is the core of the planner and the random samples could be in obstacle or free state space. The Metric returns the value/cost representing the effort to reach state space from another state space and the Nearest Neighbor is the search algorithm returning the nearest node to the new random sample based on the predefined metric function. In the Select Parent process, it chose an existing "parent" node to the new sample node and the selection of parent nodes or samples depends on the specific sample-based planners. As for the last two procedures: Collision Checking and Local Planning, the collision checking returns success or failure when it connects two state spaces and it does not intersect with the obstacle space when connecting successfully while the local planner tries to intuitively use straight-line paths to connect two configuration spaces[19][20].

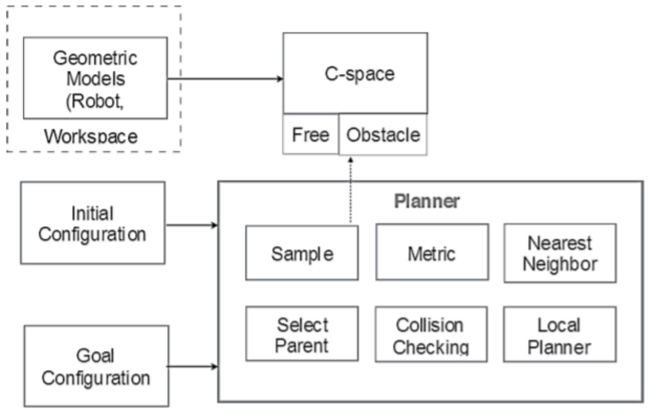


Fig. 12. A general sampling-based planner [19]

Most frequently used sampling-based motion planning algorithms such as Rapidly-exploring Random Trees (RRT), Expansive space trees (EST), Probabilistic Roadmaps (PRM) frameworks are widely used in robot arm motion planning [21][22] and the PRM and RRT are discussed in the following.

PRM is often used in multi-query planning performed in the same environment and it implements two main stages to generate a probabilistic roadmap: learning/preprocessing phase and query phase. In the learning phase, configuration space is sampled and the sample in free space attempts to connect to all neighboring nodes using a local planner while the sample is discarded if it is generated in the obstacle space[19][23]. Then the process will check for collisions, disconnect all the colliding paths and the whole process will repeat a certain number of times. In Fig. 13, a typical roadmap building in the learning phase is shown[19]. In the second stage, the start and goal configurations are added to the roadmap and then a graph search algorithm is used for the shortest path between the two configurations using the generated roadmap.

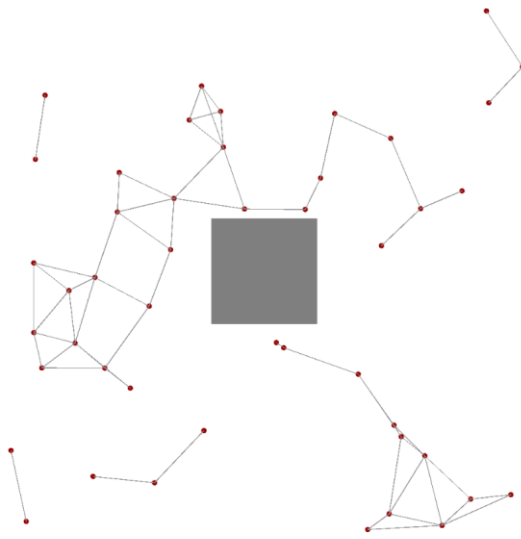


Fig. 13. Typical roadmap built in the PRM learning phase [19]

The RRT algorithm incrementally grows a search tree from an initial state space by sampling random configuration repeatedly and attempting to add the connection from the nearest tree node to it, and it is primarily aimed at single-query applications [20]. More specifically, as shown in Fig. 14, the search is initialized from the start state-space towards the goal configuration. A random configuration is generated by the planner and then the nearest node is calculated through Nearest Neighbor search. If the random configuration is in free space, then it will be connected to the nearest node returning a new node and the random configuration would be discarded if it is in the obstacle space. The collision checking is performed and the tree explores in the free space until the node is connected to the goal state-space or the new node is the goal configuration [19][22][23].

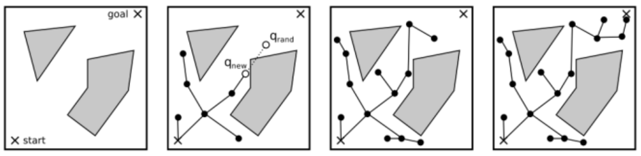


Fig. 14. Illustration of RRT process [23]

In single query problems, PRM is slower than RRT since the RRT is not required to construct the roadmap after sampling the configuration space[19]. However, PRM could be used to solve different problems in the same environment and the queries could be solved quickly since the planning time is used to generate the roadmap and the start and goal configuration in the subsequent stage are specified[20]. The following Table I lists some planner algorithm approach examples based on the PRM and RRT, and it provides a brief description of each approach.

Table I
Some planner algorithms based on PRM and RRT

Algorithms based on PRM	
Elastic Roadmap [24]	Uses both workspace information and control techniques for planning. Roadmaps generated with feedback controllers are used as local planners and the controllers can respond to environmental changes.
Flexible Anytime Dynamic PRM (FADPRM) [25]	An A* search is done from the goal state. Based on the nearby nodes' relative density and workspace desirability, samples are generated near frontier nodes. The motion plan/path can be continually improved during execution.
Reactive Deformation Roadmap (RDR) [26]	Edges as a series of particles and then builds a roadmap based on dynamic milestones/vertices. These particles are then controlled via a Newtonian physics model that vertices attract and obstacles repulse. The use of control techniques makes the roadmap adjust to changes instead of requiring re-planning and the add or remove of vertices and links maintains connectivity.
Algorithms based on RRT	
Closed Loop RRT [27]	Uses partial motion planner approaches, grows a tree input space at each time step and generates the best trajectory in the assigned time. Then re-root the tree at the end of each trajectory.
Dynamic RRT (DRRT) [28]	Edges intersecting with a notice obstacle are invalidated and then a trimming process that iteratively discards children of invalid vertices or tree branches. If the goal is not reachable by the tree, it is then regrown. An enhancement is to grow the tree from the goal to the current configuration space affected by changes.
RRT*[29]	An optimal re-planning version of RRT generates the optimal or shortest path to the goal state space in the current static environment. The tree is rewired if the newly added vertex has a decreased cost.

2.4.2 Grasping Motion Approaches

The robot grasping problem has been widely studied and is an active research field. Broadly, the stable robot grasping methods could be classified as analytical approaches and empirical approaches. Physical and mathematical geometry models, as well as dynamics and kinematics, are used to calculate grasps that are stable in an analytical approach. However, it's still hard to transfer the exact modeling result in the real world since the parameters or model state are normally assumed to be ideal in the calculation while the condition in the real world is much more complicated due to the manipulator and object interactions. On the other hand, the empirical methods are more likely to use real-world models and experience, so it could also be called the database-driven grasping approach. This method works well with known objects, associating good grasp points with an object model/shape offline database based on object classes [30][31], but this approach is hard to generalize if the target object is unknown.

The analytical approach is typically involved with leverage force-closure to make sure the stable grasp configuration. As shown in Fig. 15, many contributions have been done to compute the force-closure grasp while few have addressed the task-oriented grasps[32]. An analytical approach proposed by Y. Li et al. for object grasping position planning with a multi-fingered hand could be discussed as an example. To plan the stable GraspableFinger Position Region (GFPR), the team firstly uses the force equilibrium condition to select the graspable candidates from all the potential object edges combinations. Then the moment equilibrium condition is used to analyze the graspable finger position region of the selected candidate. According to [33], several boundary hyperplanes determine the region and to obtain the exact GFPR, two prepositions are then proposed based on the boundary hyperplanes. Furthermore, the team compared the biggest resulting inscribed hyperspheres and convex polyhedron volume and successfully found the stable finger position region for robot arm finger grasping [33].

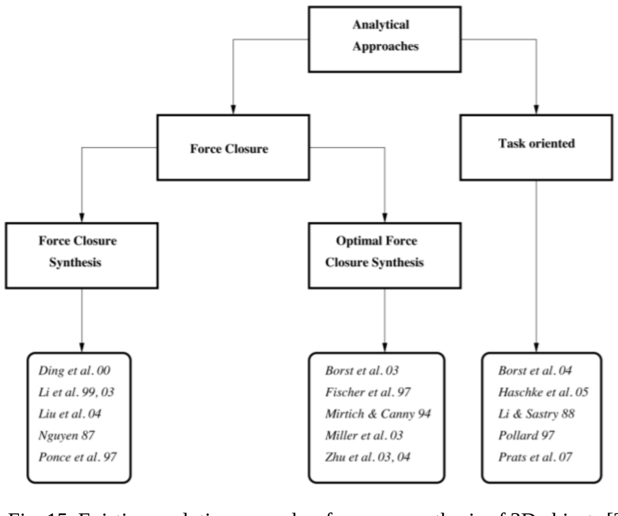


Fig. 15. Existing analytic approaches for grasp synthesis of 3D objects [32]

Empirical approaches are based on classification and human demonstration learning methods discarding the analytical computational complexity. The techniques in the empirical approach are shown in Fig. 16 and it could be broadly classified into two classes: observation of the grasped object and the observation of grasping performed by humans. In the object observation-oriented method, the system analyzes the properties and learns the relation between different robot hand shapes and target object characteristics for the task grasp computation. Some techniques identify the finger grasping region based on images and others use hand shapes, grasp parameters or geometric features to search for stable grasps [32]. For the human demonstration observation approach, the Different Learning-by-Demonstration (LbD) framework, the robot observes human behavior such as picking up a cup and then performs the task itself, the key idea. In this case, the use of stereoscopy to track human hand grasping action or recognize the hand shape from grasping image dataset, and utilization of data-gloves to map human hand to robot arm/finger workspace and learn the joint angles are most popular used techniques. To perform the human observation, commonly used sensors/trackers include magnetic trackers, dataglove-based descriptors, vision-based descriptors etc. [32].

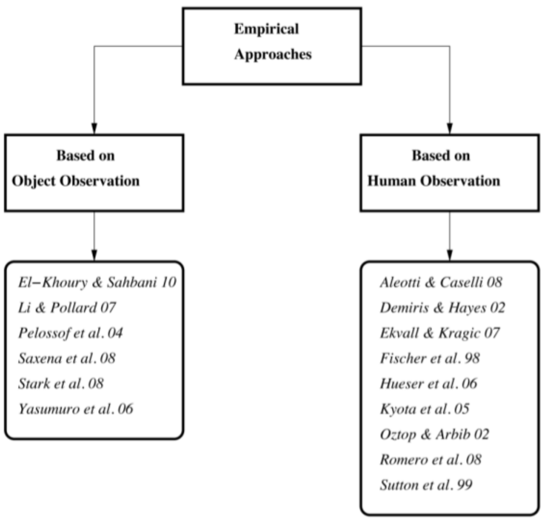


Fig. 16. Existing empirical approaches for grasp synthesis of 3D objects [32]

2.4.3 Reinforcement Learning for Robot Manipulation

Reinforcement learning applies in high-repeated task training situations enabling fast and reliable training outcomes and it has been applied in robotic control using high and low dimensional function approximators. By combining basic CNN categorization algorithms, trained robots can successfully do sorting jobs in a certain manner associated with the category to which the target belongs. Impressive improvement has been done in reinforcement learning specifically in robot manipulation and they could broadly be classified into model-based and model-free (policy-based and value-based) depending on whether there is a model of the environment that could be accessed. A general taxonomy of the RL algorithms is shown in Fig. 17 [30]. Compared to model-free algorithms which learn the value function directly from the robot environment interaction, the model-based algorithm typically has a learning-induced reward function and one problem is that the agent could potentially work well in the learning model but behave poorly in the real world if bias is introduced in the model. Value-based algorithms estimate the price to take action or be in a state. It may not be sufficient for complex problems but it could be a key block for other methods. On the other hand, using a policy-based method, reliability and stability improvement could be made, but they are less sample efficient since they calculate the policy value and use it for control simultaneously [30].

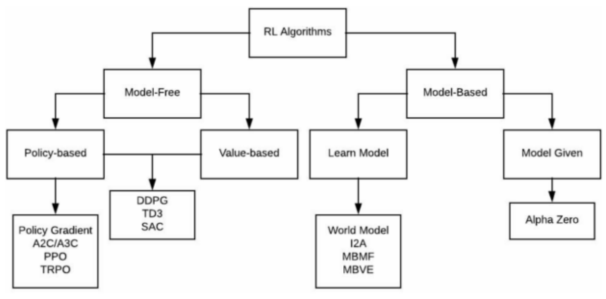


Fig. 17. RL algorithms in robot manipulation [30]

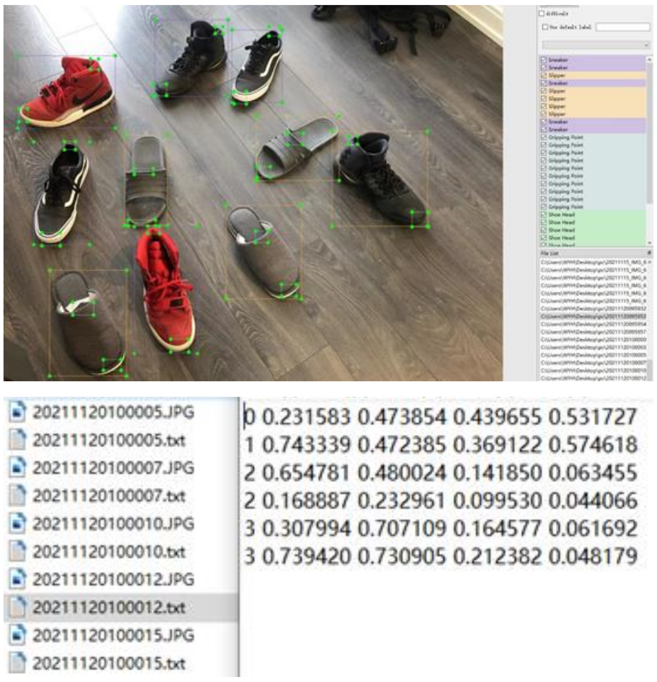
Most grasping systems use the prediction of grasp pose to determine which location is the best grasping point and then an open-loop planner is used to execute the grasping task. However, some researchers also worked on dynamic closed-loop control. For example, M. Gualtieri et al. has demonstrated the pick and place actions of a robot arm across different instances [34] using deep RL and D. Kalashnikov et al. presented a robotic reinforcement learning with the raw sensory data input framework and closed-loop vision-based control to tackle the grasping, regrasping and responses to obstacles [35].

3. Methodology

3.1 Shoe Detection

To train the model, the first step is to label the image. Due to time constraints, we only use 74 images to train the model. There are four classes in our model which is Sneaker, Slipper, Gripping Point, and Shoe Head. The reason why we detect the shoe head is it can assist the robot for path planning. The robot will stop at the back of the gripping point from the shoe head to perform the gripping. The gripping point for the sneakers is at the back of the shoe while for the slipper, it is on the top. For sneakers, when the robot detects both shoe head and gripping point, it will first alter its initial path to the back of the gripping point. When the robot comes to a stop, it will plan the trajectory of the manipulator and stop right above the gripping point, with the gripper head straight down and perpendicular to the gripping point. After the axis of the grippier is aligned with the bottom of the gripping point, the gripper will move straight down for the grip. For the slippers, the grippier will be aligned horizontally with the gripping point and reach out toward the shoe to perform gripping.

Using the label application labelIng-master, we label the image in the YOLO v3 format shown in following Fig. 18. For each image we use, it will generate a txt file containing the class, x-y coordinates, width, and height of the object.

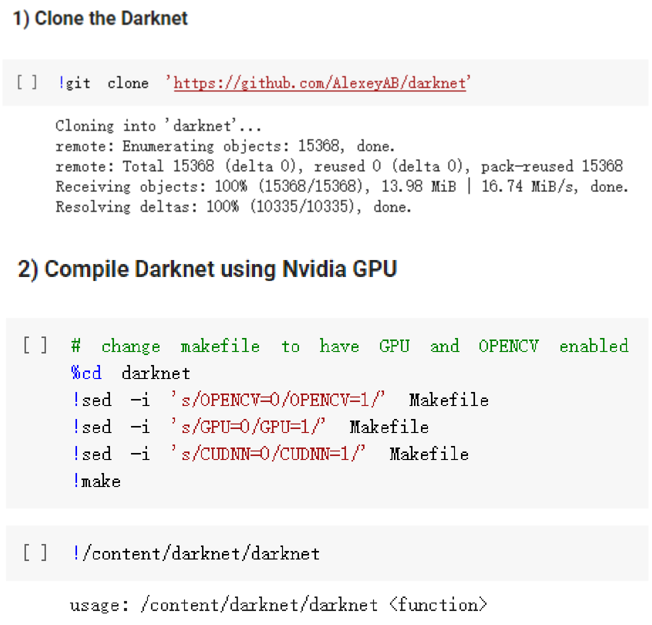


```
20211120100005.JPG
20211120100005.txt
20211120100007.JPG
20211120100007.txt
20211120100010.JPG
20211120100010.txt
20211120100012.JPG
20211120100012.txt
20211120100015.JPG
20211120100015.txt
```

```
0 0.231583 0.473854 0.439655 0.531727
1 0.743339 0.472385 0.369122 0.574618
2 0.654781 0.480024 0.141850 0.063455
2 0.168887 0.232961 0.099530 0.044066
3 0.307994 0.707109 0.164577 0.061692
3 0.739420 0.730905 0.212382 0.048179
```

Fig. 18. A screenshot of the label application and the label file.

We decided to train the model online since we can use GPU to accelerate our training process in Google Colab. We get the code from Github (https://github.com/AlexeyAB/darknet) and compile the Darknet using GPU. Then we config the file for our model. The classes, as well as the filters, need to change accordingly with our task. After that we extract the images as well as the label file and generate the core file of the training, 'training.txt', which contains all the paths of the dataset, config files, and the initial weights. The Fig. 19 below presents the training process.

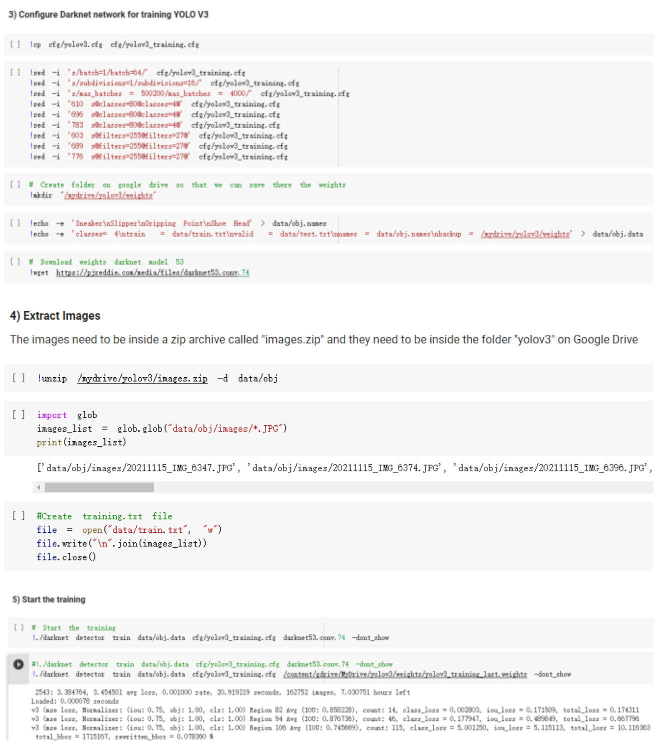


1) Clone the Darknet

```
[ ]  !git clone 'https://github.com/AlexeyAB/darknet'

     Cloning into 'darknet'...
     remote: Enumerating objects: 15368, done.
     remote: Total 15368 (delta 0), reused 0 (delta 0), pack-reused 15368
     Receiving objects: 100% (15368/15368), 13.98 MiB | 16.74 MiB/s, done.
     Resolving deltas: 100% (10335/10335), done.
```

2) Compile Darknet using Nvidia GPU

```
[ ]  # change makefile to have GPU and OPENCV enabled
     %cd darknet
     !sed -i 's/OPENCV=0/OPENCV=1/' Makefile
     !sed -i 's/GPU=0/GPU=1/' Makefile
     !sed -i 's/CUDNN=0/CUDNN=1/' Makefile
     !make

[ ]  !/content/darknet/darknet

     usage: /content/darknet/darknet <function>
```



3) Configure Darknet network for training YOLO V3

```
[ ] !cp  cfg/yolov3.cfg  cfg/yolov3_training.cfg
```

```
[ ] !sed  -i  's/batch=1/batch=64/'  cfg/yolov3_training.cfg
    !sed  -i  's/subdivisions=1/subdivisions=16/'  cfg/yolov3_training.cfg
    !sed  -i  's/max_batches  =  500200/max_batches  =  4000/'  cfg/yolov3_training.cfg
    !sed  -i  '610  s@classes=80@classes=4@'  cfg/yolov3_training.cfg
    !sed  -i  '696  s@classes=80@classes=4@'  cfg/yolov3_training.cfg
    !sed  -i  '783  s@classes=80@classes=4@'  cfg/yolov3_training.cfg
    !sed  -i  '603  s@filters=255@filters=27@'  cfg/yolov3_training.cfg
    !sed  -i  '689  s@filters=255@filters=27@'  cfg/yolov3_training.cfg
    !sed  -i  '776  s@filters=255@filters=27@'  cfg/yolov3_training.cfg
```

```
[ ] #  Create  folder  on  google  drive  so  that  we  can  save  there  the  weights
    !mkdir  "/mydrive/yolov3/weights"
```

```
[ ] !echo  -e  'Sneaker\nSlipper\nGripping  Point\nShoe  Head'  >  data/obj.names
    !echo  -e  'classes=  4\ntrain   =  data/train.txt\nvalid   =  data/test.txt\nnames  =  data/obj.names\nbackup  =  /mydrive/yolov3/weights'  >  data/obj.data
```

```
[ ] #  Download  weights  darknet  model  53
    !wget  https://pjreddie.com/media/files/darknet53.conv.74
```

4) Extract Images

The images need to be inside a zip archive called "images.zip" and they need to be inside the folder "yolov3" on Google Drive

```
[ ]  !unzip  /mydrive/yolov3/images.zip  -d  data/obj
```

```
[ ]  import  glob
     images_list  =  glob.glob("data/obj/images/*.JPG")
     print(images_list)

     ['data/obj/images/20211115_IMG_6347.JPG', 'data/obj/images/20211115_IMG_6374.JPG', 'data/obj/images/20211115_IMG_6396.JPG',
```

```
[ ]  #Create  training.txt  file
     file  =  open("data/train.txt",  "w")
     file.write("\n".join(images_list))
     file.close()
```

5) Start the training

```
[ ] #  Start  the  training
    !./darknet  detector  train  data/obj.data  cfg/yolov3_training.cfg  darknet53.conv.74  -dont_show
```

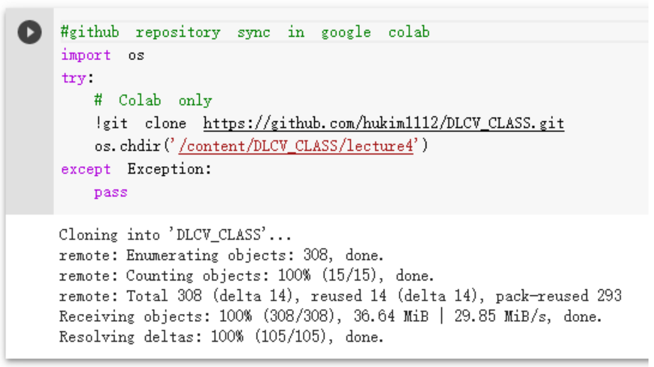
```
#!./darknet  detector  train  data/obj.data  cfg/yolov3_training.cfg  darknet53.conv.74  -dont_show
!./darknet  detector  train  data/obj.data  cfg/yolov3_training.cfg  /content/gdrive/MyDrive/yolov3/weights/yolov3_training_last.weights  -dont_show

 2543: 3.384764, 3.454501 avg loss, 0.001000 rate, 20.919219 seconds, 162752 images, 7.030751 hours left
Loaded: 0.000078 seconds
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 82 Avg (IOU: 0.858228), count: 14, class_loss = 0.002803, iou_loss = 0.171509, total_loss = 0.174311
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 94 Avg (IOU: 0.876736), count: 46, class_loss = 0.177947, iou_loss = 0.489849, total_loss = 0.667796
v3 (mse loss, Normalizer: (iou: 0.75, obj: 1.00, cls: 1.00) Region 106 Avg (IOU: 0.745669), count: 115, class_loss = 5.001250, iou_loss = 5.115113, total_loss = 10.116363
 total_bbox = 1715167, rewritten_bbox = 0.078360 %
```

Fig.19. Colab code for training the model

As we can see, we trained for 2500 epochs and approximately three days due to the limitation of free usage of GPU. The final loss is around 3.45. The final results are in the next Chapter.

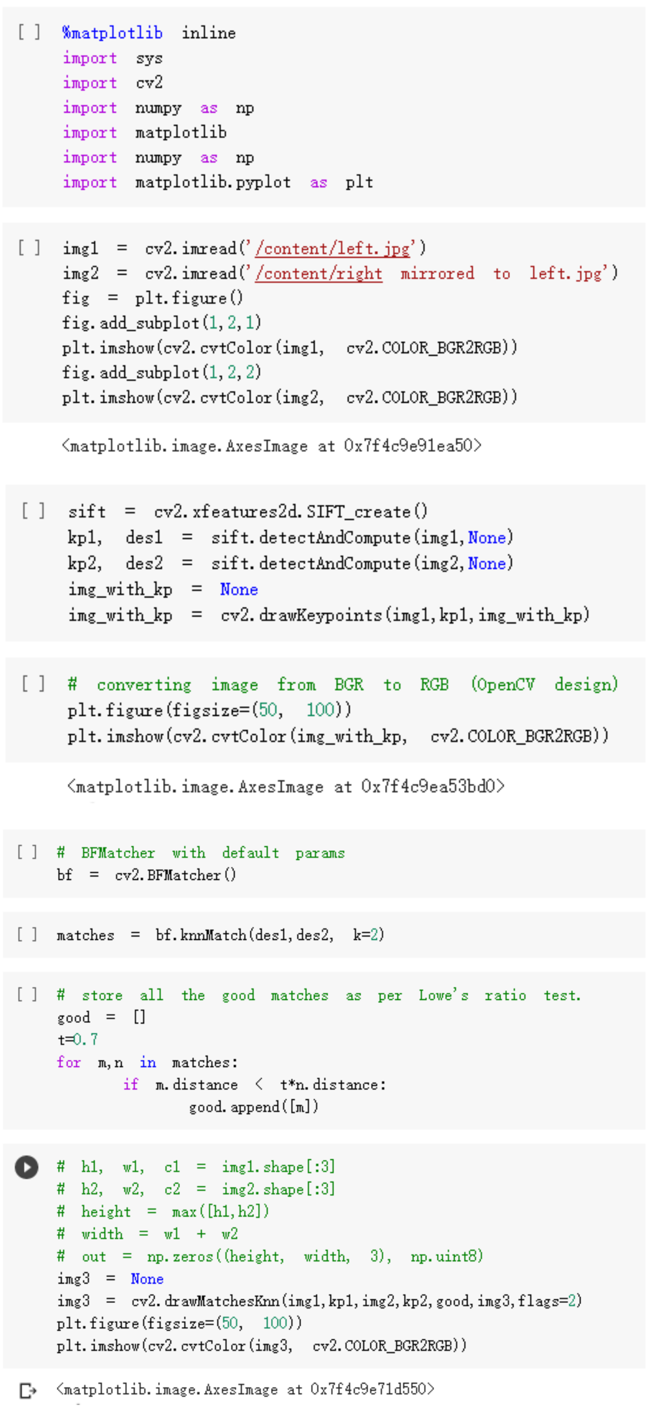
3.2 Shoe Pairing

Following is the code for the SIFT algorithm to pair the shoe. The code we used in Fig. 20 (a) and (b) is adapted from (https://github.com/hukim1112/DLCV_CLASS.git), and then our image was uploaded.



```
#github  repository  sync  in  google  colab
import  os
try:
    #  Colab  only
    !git  clone  https://github.com/hukim1112/DLCV_CLASS.git
    os.chdir('/content/DLCV_CLASS/lecture4')
except  Exception:
    pass

Cloning into 'DLCV_CLASS'...
remote: Enumerating objects: 308, done.
remote: Counting objects: 100% (15/15), done.
remote: Total 308 (delta 14), reused 14 (delta 14), pack-reused 293
Receiving objects: 100% (308/308), 36.64 MiB | 29.85 MiB/s, done.
Resolving deltas: 100% (105/105), done.
```

Fig. 20. (a). The screenshot for the SIFT algorithm.



```
[ ]  %matplotlib  inline
     import  sys
     import  cv2
     import  numpy  as  np
     import  matplotlib
     import  numpy  as  np
     import  matplotlib.pyplot  as  plt
```

```
[ ]  img1  =  cv2.imread('/content/left.jpg')
     img2  =  cv2.imread('/content/right  mirrored  to  left.jpg')
     fig  =  plt.figure()
     fig.add_subplot(1,2,1)
     plt.imshow(cv2.cvtColor(img1,  cv2.COLOR_BGR2RGB))
     fig.add_subplot(1,2,2)
     plt.imshow(cv2.cvtColor(img2,  cv2.COLOR_BGR2RGB))

     <matplotlib.image.AxesImage at 0x7f4c9e91ea50>
```

```
[ ]  sift  =  cv2.xfeatures2d.SIFT_create()
     kp1,  des1  =  sift.detectAndCompute(img1,None)
     kp2,  des2  =  sift.detectAndCompute(img2,None)
     img_with_kp  =  None
     img_with_kp  =  cv2.drawKeypoints(img1,kp1,img_with_kp)
```

```
[ ]  #  converting  image  from  BGR  to  RGB  (OpenCV  design)
     plt.figure(figsize=(50,  100))
     plt.imshow(cv2.cvtColor(img_with_kp,  cv2.COLOR_BGR2RGB))

     <matplotlib.image.AxesImage at 0x7f4c9ea53bd0>
```

```
[ ]  #  BFMatcher  with  default  params
     bf  =  cv2.BFMatcher()
```

```
[ ]  matches  =  bf.knnMatch(des1,des2,  k=2)
```

```
[ ]  #  store  all  the  good  matches  as  per  Lowe's  ratio  test.
     good  =  []
     t=0.7
     for  m,n  in  matches:
             if  m.distance  <  t*n.distance:
                     good.append([m])
```

```
     #  h1,  w1,  c1  =  img1.shape[:3]
     #  h2,  w2,  c2  =  img2.shape[:3]
     #  height  =  max([h1,h2])
     #  width  =  w1  +  w2
     #  out  =  np.zeros((height,  width,  3),  np.uint8)
     img3  =  None
     img3  =  cv2.drawMatchesKnn(img1,kp1,img2,kp2,good,img3,flags=2)
     plt.figure(figsize=(50,  100))
     plt.imshow(cv2.cvtColor(img3,  cv2.COLOR_BGR2RGB))

     <matplotlib.image.AxesImage at 0x7f4c9e71d550>
```

Fig. 20. (b) The screenshot for the SIFT algorithm.

Then is the implement part of the SIFT algorithm for both inputs. Noticed that for parameter t in good matches, we varied to see the results. It turns out that a suitable value for t is around 0.7. The outcome is shown in the next chapter.

3.3 SLAM

The SLAM algorithm is used to build a map when the robot is in a new environment or when the localization algorithm is not able to compute a stable result which is when the map needs to be updated, which is elaborated in the localization methodology part. The use of the SLAM algorithm in this project is based on LiDAR sensor inputs and encoder data of the wheels. We combined these data to try to create complete loop closures in the environment in order to reduce the effect of encoder errors. The output of SLAM is a probabilistic occupancy grid map that contains cells that are black, grey, and white. These black, grey, and white cells represent map boundaries, uncertain areas, and open spaces, respectively. After eliminating all openings of the map boundary, the map is saved for localization and path planning algorithm to reference.

As introduced in Simultaneous Localization and Mapping: Part I, SLAM is already a very effective and working algorithm. What we can do to improve the performance of SLAM is by improving the quality of the input data. Lidar sensor data quality depends on the quality and good calibration of the sensor. Based on our application environment, a management distance range from 0.2m to 4m and a minimum scan frequency of 10Hz should be guaranteed. encoder error can consist of many factors including the type of tire used, the smoothness of the robot's motion control, and the suspension system used on the wheels. by iteratively testing the motion control and improving the mechanical design of the robot, the encoder error can be reduced so that the SLAM would be responsible for making fewer corrections while aligning environment features.

3.4 Localization

The localization of the robot is based on the saved map previously created by the SLAM algorithm. It is assumed that the map is accurate enough to represent the boundaries and static obstacles of the environment.

While in operation, by comparing with a predefined confidence threshold of the current localization result, the robot will perform a localization routine if the confidence level is too low. The localization routine collects LiDAR sensor data while commanding the robot to run in circles where it is clear of obstacles. the radius of the circle increases until the confidence level reaches this threshold. If no bigger circle can be performed, limited by the space, the robot will do a random walk to another location and move in circles again until it is confident enough of its location. If the confidence level just cannot be reached by repeating this sequence, it is most likely that the environment has been changed, for example, the robot may have been moved to a completely new environment while being powered down, or the arrangement of the space has been greatly changed. In this case, the SLAM algorithm will be performed again to build a new map before this independent Monte Carlo localization algorithm is run.

The localization of the charging dock can easily be achieved by having an Infrared light source on the front of the charging dock and an infrared detector on the front of the robot. By rotating the robot, whenever it detects the infrared light from the charging dock, it stops and goes straight to the light source and at the same time corrects its orientation in case the light is not seen. The charging dock will be marked on the map after the robot reaches the charging dock and confirms it by starting charging. As for the shoe rack, it is expected that the shoe rack's dimensions are fixed which means only the shoe racks that are compatible with the robot can be used in the system for organizing shoes. This way, the dimensions and the structure of the shoe rack can be controlled, and it is feasible to make the shoe rack easily detected by designing a unique pattern or placing marking stickers.

3.5 Path Planning

The path planning algorithm of this robot is required to plan the global path using the saved map from SLAM and re-plan local passes whenever it runs into obstacles on this way. While planning global paths, the algorithm is expected to generate the optimal or near-optimal path very quickly based on its current pose, the target pose and the map. While replanning local paths, though the obstacles that we expect the shoe sorting robot will encounter are divided into static obstacles and dynamic obstacles, since the robot is moving at a relatively low speed and the motion of dynamic obstacles that includes humans and pets can be very random and very hard to predict, they can be treated the same as static obstacles for simplicity. The obstacle data comes from LiDAR readings and is helped by the camera image to distinguish whether it is an obstacle or a shoe that needs to be sorted.

The robot is usually expected to be activated from the charging dock to go to an unsorted shoe location or start a search for unsorted shoes. A path needs to be planned to get to the target location and use the robot arm to grab it. Then a path needs to be planned to get to the shoe rack. When the robot gets to the shoe rack the arm will put the shoe on the shoe rack. If there is another shoe in the area the robot will repeat this sequence until the area is clear of unsorted shoes. it will eventually go back to the charging dock. The task sequence is illustrated as shown in Fig. 21.

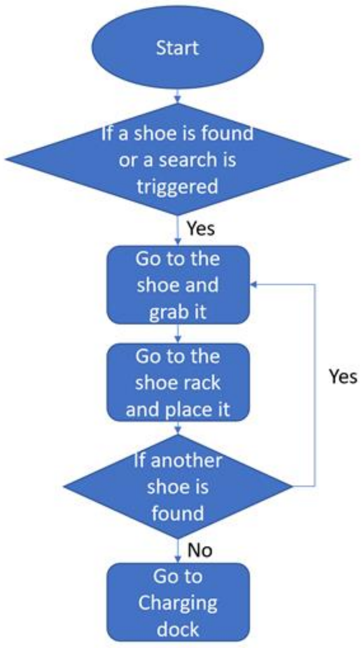

Start
If a shoe is found or a search is triggered
Yes
Go to the shoe and grab it
Go to the shoe rack and place it
Yes
If another shoe is found
No
Go to Charging dock

Fig. 21. Task sequence

By using RRT* path planning algorithm, the shoe sorting robot is able to plan near-optimal paths either at the start globally or quickly after detecting a new obstacle on its way. What's also great about using RRT* is that the near-optimal path not only to the target pose but also to most locations on the map is found in one run, which means it is possible to save computational resources by saving these path sets from the most commonly visited locations to anywhere on the map and reuse them if the map is not changed, though this is not necessary since RRT* algorithm executes very efficiently.

3.6 Path Planning Demonstration

RRT* path planning algorithm is set up and tested using MATLAB. An occupancy map is defined as one of the inputs using a 26*27 matrix. An occupancy-map-based state validator is created using state space with a default validation distance of 0.01. The built-in function plannerRRTStar is used to create a planner.

Most commonly, the expected working area of the shoe sorting robot is a rectangle with possibly shoe racks placed near one of the edges and other decorations or articles of daily use in the area. However, occasionally the area would become more complicated. To efficiently visualize the use of RRT*, the area was defined to be an L-shape with obstacles that are not on the original map but detected when the robot moves close. The charging dock and the shoe rack is defined near the door area. The target unsorted shoe's location is defined at the other end of the map as illustrated in Fig. 22.

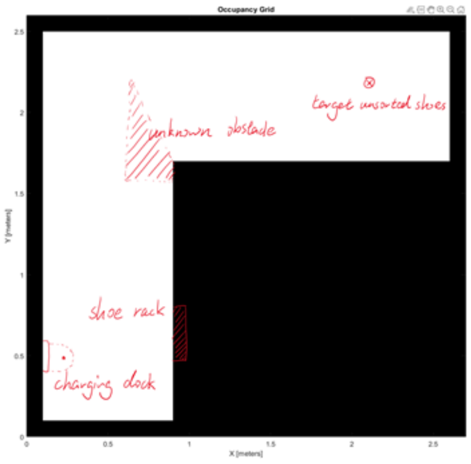


Fig. 22. Path planning demonstration set up

As shown in Fig. 23, firstly, the map needs to be inflated so that the robot can be represented by only a dot and the path of the robot can be represented as a line. The amount of inflection is equal to the maximum length from the robot turning center to its edge. The principle of inflation applies to all occupied areas on the map including obstacles that are found while operating.

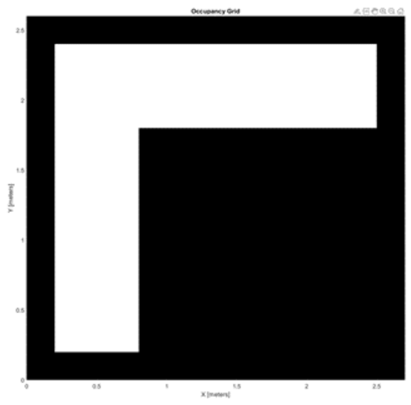


Fig. 23. Occupancy map inflation

Using the robot's current pose, the saved map and the target shoe location mapped by the result obtained from RGB image processing using YOLOv3, RRT* algorithm is executed to find a path from the current pose to the target pose as shown in Fig. 24. The algorithm is set to continue adding nodes after the target pose is achieved until a predefined number of iterations is executed to further optimize the path.

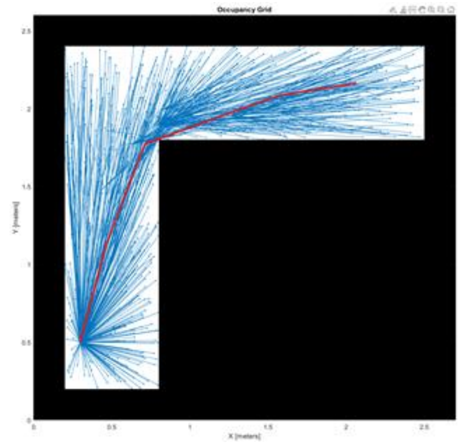
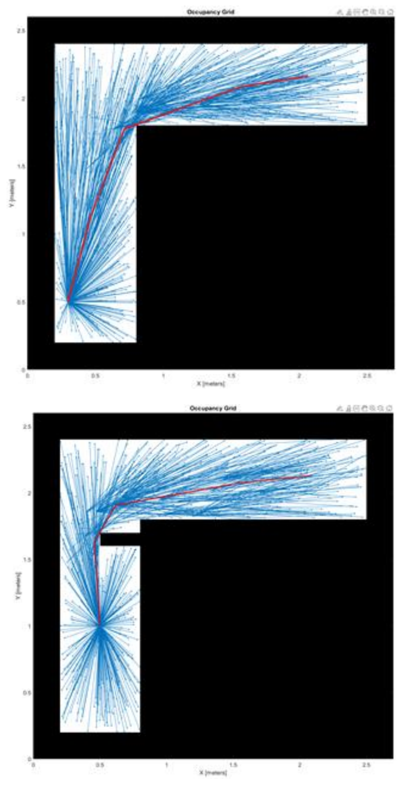


Fig. 24. RRT path planning result

While the robot is following the path to approach the target pose, the LiDAR sensor is always scanning for unexpected obstacles on its way. If a new obstacle is detected, it will temporarily be mapped onto the occupancy map. RRT* algorithm will be executed again to find a near-optimal path around the newly detected obstacle as shown in Fig. 25. The robot will then follow the new path and try to get to the target location. Also, while the robot is approaching the target unsorted shoe, YOLOv3 will find the location of the grasping point and the shoe head of that shoe and return a new desired location for the robot, RRT* will be executed again at this point of time to find a new path to the new target location. As the robot is approaching the shoe, LiDAR sensor readings will also be used for the correction of the robot's localization, if the discrepancy is too large, RRT* will be executed again to re-plan a paths.



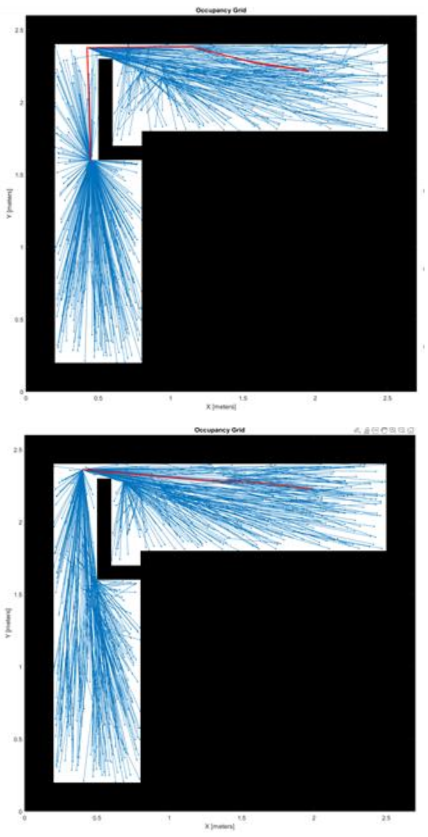


Fig. 25. Re-plan local paths

3.7 Robot Arm Planning

To start the robot arm motion planning in MATLAB, [KINOVA JACO]^®3-fingered robot with a spherical wrist robot arm was chosen as our robot arm and the demo is adapted from [37]. The first step is to create the robot arm motion environment. In the real world, the robot moving environment would be developed according to the sensory data generated in the autonomous navigation process in real-time. However, for the demonstration purpose, the team created the floor, shoe and shoe rack using the collisonBox function. As shown in Fig. 26, the shoe is represented by a red cylinder and the shoe rack is represented by the blue rectangular box with preset height. In the environment setup, RigidBodyTreeStateSpace is created to represent the configuration space. One of the sampling strategies in state space is to uniformly random sample the end-effector pose in the Workspace Goal Region around the reference goal pose and then map it to the joint space. Also, this strategy guides the RRT planner to the goal area in the task space so that the RRT could converge quicker. The WGR defines the acceptable end-effector pose and the use of the WGR increases the chance of finding a solution through the samples biasing to the goal region. After the environment setup, a customized validator is used to check the collision situation within the environment and the robot and the invalid states would be discarded.

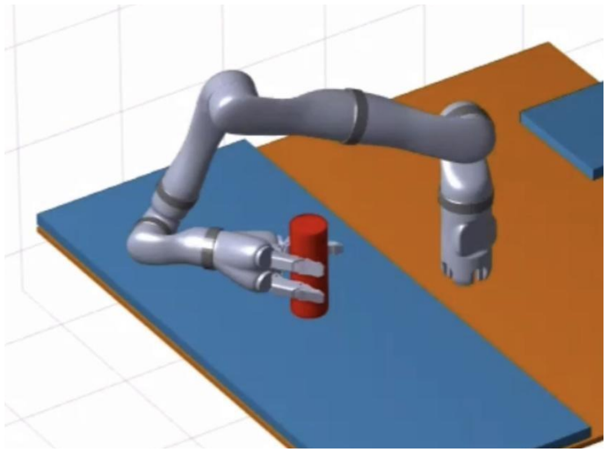


Fig. 26. Robot arm demo setting in MATLAB

The grasp motion planning is performed using the RRT algorithm which is the plannerRRT object in the demo. The inverseKinematics function is used to specify the start and goal state space, solving the configuration according to the end-effector pose. Then the constraint on the interim manipulator is turned on through the UseConstrainedSampling property and then the arms move motion planning starts. After the path planning, a recursive corner-cutting technique is used to smooth the found path and then the team visualizes the whole grasp motion including the grasp and release of the cylinder (shoe) and the robot arm move motion.

3.8 Robot Working Sequence

It is important to clarify the sequence of how the robot works. The robot is expected to be capable of being activated at any location with any orientation. Once activated, it will firstly run the localization algorithm using a previously stored map. If localization fails or there is no stored map, SLAM will be performed and a new map will be generated and stored. Once it is commanded by a user of a smart home system to start sorting shoes, the robot will use the lower camera to find any unsorted shoe in its field of view. In cases where the camera is not overlooking the whole search region, it will either rotate its orientation or move to any hidden area to perform a mobile search. A path will be planned using RRT* algorithm and a collision free movement will be performed by the agent to approach the shoe. As mentioned previously in the image processing part, after the grasping point and shoe head is found and a new destination is marked on the map, the path will be updated by the path planning algorithm. After the robot arrives at the target destination, the robot arm will move to adjust the upper camera that can clearly capture the targeted shoe with its grasping point and shoe head. Then, a motion trajectory will be planned for the gripper to pick the shoe up and hold the shoe above the robot so that the shoe will not block the lower camera and the LiDAR sensor or generate a large moment that can flip the robot forward. Then, path planning and motion planning algorithms are performed again to enable the agent to approach the shoe rack and place the shoe on it. After finishing this sequence of processes, the robot will go back to run the same process until there is no more unsorted shoe needed to be placed on the shoe rack.

3.9 3D model

The 3D model of the robot is simulated through SolidWorks shown in Fig. 27 below. Considering the work sequence mentioned above, each part is designed based on the needs of doing certain tasks and integrated as an assembly. The main body is consisted of 3 parts: the base, the robot arm, and the sensors.

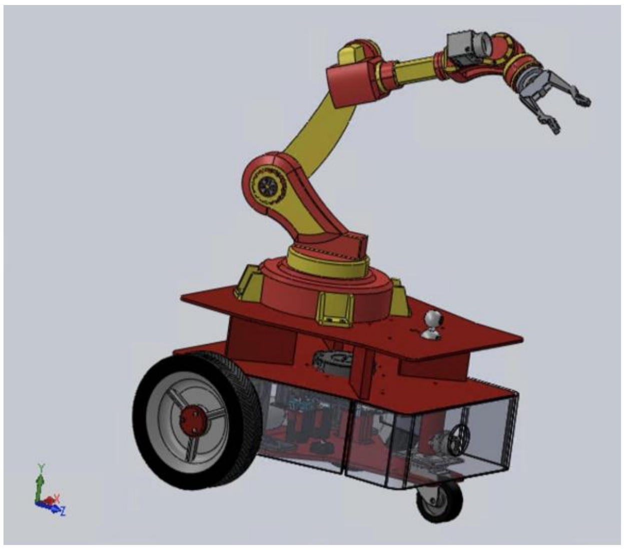


Fig. 27. 3D model of "shoe sorter" robot [37][38][39][40][41]

The base design has taken stability, mobility, and loading as main considerations. To make sure the robot can traverse from any point to the targeted destination in the environment (living room), 2 main wheels are selected with 100-mm-diameter and mm-width. This enables the robot to overcome most floor types of different roughness and unevenness with steady motion. An auxiliary wheel is placed at the front part of the base to stop tipping when the robot arm is reaching out to grab shoes. The two main wheels are controlled by two different motors that can rotate asynchronously, and thus the robot can perform a facile motion. The structure of the base is able to support a total load of 300N. So, the robot arm can work steadily on the stage above the base. The 3D model of this base is pulled from the online source from GrabCAD Community that was initially created and uploaded by Philippe Cadic on April 12th, 2020 [37]. Some modifications in shape and scale are applied to the downloaded files to fit the model in this project.

The second part of the main body, the robot arm, is designed to conduct shoe placement tasks. The size and degrees of freedom (DOF) are the top parameters in design. The total length of the robot arm is more than 1 meter so that it can retrieve shoes from a certain distance without hitting the obstacle. Furthermore, the robot arm has 6 DOFs to enable the gripper to take the shoe from any point in the space. The 3D CAD is downloaded from GrabCAD Community, and the author is Mohammed A, uploaded on August 14th, 2021 [38]. Similarly, some modifications have been done to fit the model to the real case.

The sensors include a LiDAR sensor, an IR sensor, a spherical shape camera, and a cube-shaped camera. The LiDAR sensor is used for map generation using SLAM, and IR sensor is used for the robot to find the charging dock. Cameras are used for visualizing objects and processing control based on data collected. The 3D models are downloaded from the same website as above. The authors and date uploaded of these 3D models are Larson Electronics (July, 2017) [39], Ali Salimnezhad (January, 2014) [40], and AZH (July, 2021) [41] respectively.

4. RESULT

4.1 Shoe Detection

The following Fig. 29 is a screenshot of the output video for shoe detection using YOLO v3. The final output is quite accurate despite some gripping points being missed.

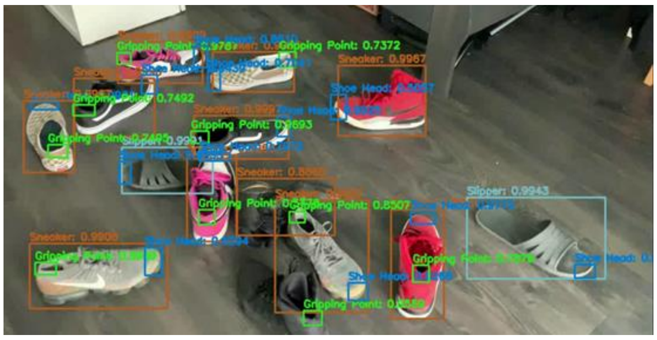


Fig. 29. Shoe detection results using YOLO v3

4.2 Shoe Pairing

The output of the SIFT algorithm is shown in Fig. 30 and Fig. 31 below. The results show that the robot has successfully used the key point to match two shoes if they are one pair.

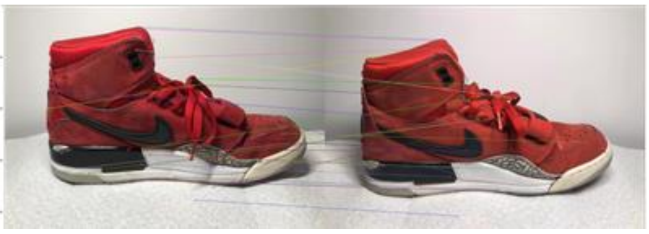


Fig. 30. Shoe pairing result using trained SIFT model for a single pair of shoes

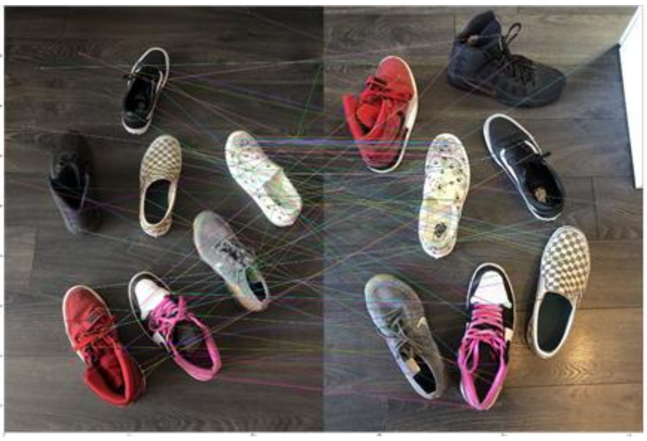


Fig. 31. Shoe pairing result using trained SIFT model for a set of shoes

4.3 Path Planning

As shown previously in Fig. 25, by using the RRT* algorithm, near-optimal paths can be generated in less than 1 second, which makes the robot capable of replan paths quickly when new obstacles are detected.

5. Discussion

For shoe detection, apparently, the dataset is not enough. We only use 74 images to train the model. And that led to the result that this model can only detect shoes on this fixed background. If there were more data for shoes, for example, with different backgrounds, different types of shoes, different orientations, and even different lighting, the outcome could be enhanced to a great extent. Another drawback for YOLO is that when running in real-time, the detection rate is around 3 frames per second. We assume that this is because it did not use GPU to accelerate the detection process. Without the assistance of the GPU, the detection speed cannot reach the designated speed of 55 fps.

For the shoe pairing SIFT algorithm, it requires two separate images as input to perform which in our case, the user needs to separate the left shoe away from the right shoe. Also, the input needs to be mirrored as well to fit the model more properly.

To improve the compatibility of the system two different types of shoe racks in the future, efforts can be put in to try to use computer vision algorithms like YOLO for identifying the shoe rack and its orientation, though a minimum requirement of the shoe rack's material, level heights, depth, angle and width will still be needed for compatibility. For example, the levels should be high enough for the robot to reach in with most shoes and retract back. Some shoes that are uncommonly higher than others can be placed on top.

Obstacles can be categorized into many different types which cannot be distinguished using LiDAR sensors. To improve the system's ability to handle different types of obstacles, we can use RGB images collected by the mounted camera to categorize obstacles based on their appearance and treat them in different ways. for example, paper trashes can be ignored and pushed away; messy cables that may strangle the wheels should be avoided; soccer balls can be slowly pushed away. The categorization of obstacles can help the robot operate more efficiently by pushing away some of the obstacles that are movable and even help the SLAM algorithm with the generation of maps by ignoring temporary obstacles.

To improve the success rate of grasping shoes, aside from distinguishing slippers from regular outdoor shoes, future work can be put into mapping shoes in 3D for a more accurate and reliable grasping point detection. In the kPAM project, the method of detecting key points and mapping them onto a pre-defined parent shoe model was a great way to find grasping points in 3D space[16]. The height and width difference between shoes is compensated by stretching all compressing the parent model. However, since we have already proven the feasibility of detecting grasping points using YOLO, we can have the robot arm end effector camera take two pictures of the grasping point from two angles. Based on the locations of the grasping point in these two pictures and the end effector locations that we have in ROS, the grasping point's relative 3D location can easily be calculated with respect to the robot. The end effector approaching orientation can be obtained the same way as now by using the relative location of the shoe head with respect to the grasping point. It is expected that there would be grasping point estimation errors that come from inaccurate end-effector location calculation, RGB image discretization and averaging, YOLO grasping point detects a discrepancy between two images, and the instability of the robot base while the arm is moving. By iteratively improving mechanical design and action sequence design, hopefully, these errors' influence would be minimized.

6. Conclusion

For each task using AI techniques in image processing, path planning, autonomous navigation, and robot arm motion planning, the team analyzed and compared each algorithm option and selected the one that best suits the application of a shoe sorting robot. Simulations were created to test and demonstrate the functionality of these algorithms. By combining these AI techniques and real-life constraints, the team came up with a working sequence and a conceptual 3D model that illustrates the primary design of the shoe sorting robot. As the system is created and iteratively improved, some assumptions have been made while many innovative ideas were also generated. Although the robot system has already been established, further insights can be gained once the mentioned improvements can be applied to the next generation of the product.

References

[1] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[2] 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.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014.

[4] R. Girshick, "Fast R-CNN," 2015 IEEE International Conference on Computer Vision (ICCV), 2015.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137–1149, 2017.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[7] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[9] D. G. Lowe, "Object recognition from local scale-invariant features," Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999.

[10] F. Duchoň et al., "Path Planning with Modified a Star Algorithm for a Mobile Robot," Procedia Engineering, vol. 96, pp. 59–69, 2014.

[11] L. Zhang, Y. Zhang, and Y. Li, "Path planning for indoor Mobile robot based on deep learning," Optik (Stuttgart), vol. 219, p. 165096–, 2020.

[12] J. Wang, W. Chi, C. Li, C. Wang, and M. Q.-H. Meng, "Neural RRT: Learning-Based Optimal Path Planning," IEEE transactions on automation science and engineering, vol. 17, no. 4, pp. 1748–1758, 2020.

[13] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part I," IEEE robotics & automation magazine, vol. 13, no. 2, pp. 99–110, 2006.

[14] M. M. Ejaz, T. B. Tang, and C.-K. Lu, "Vision-Based Autonomous Navigation Approach for a Tracked Robot Using Deep Reinforcement Learning," IEEE sensors journal, vol. 21, no. 2, pp. 2230–2240, 2021

[15] T. Li, F. Wang, C. Ru, Y. Jiang, and J. Li, Keypoint-Based Robotic Grasp Detection Scheme in Multi-Object Scenes, vol. 21, no. 6, pp. 2132–2132, Mar. 2021.

[16] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, "kPAM: KeyPoint Affordances for Category-Level Robotic Manipulation," 2019.

[17] W. Wiesemann, D. Kuhn, and B. Rustem, “Robust Markov Decision Processes,” Mathematics of operations research, vol. 38, no. 1, pp. 153–183, 2013.

[18] A. Rahman, "Navigation in Wheeled Mobile Robots using Kalman Filter Augmented with Parallel Cascade Identification to Model Azimuth Error." Queen's University (Canada), Ann Arbor, 2013.

[19] M. Elbanhawi and M. Simic, “Sampling-based Robot Motion Planning: A Review,” IEEE Access, vol. 2, pp. 56–77, 2014.

[20] S. Karaman and E. Frazzoli, “Incremental sampling-based algorithms for optimal motion planning,” Robotics: Science and Systems VI, 2010.

[21] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," Robotics and Automation, IEEE Transactions on, vol. 12, pp. 566-580, 1996.

[22] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University TR 98-11, 1998.

[23] A. Short, Z. Pan, N. Larkin, and S. van Duin, “Recent progress on sampling based dynamic motion planning algorithms,” 2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM), 2016.

[24] Y. Yang and O. Brock, “Elastic roadmap motion generation for autonomous mobile manipulation,” Autonomous Robots, vol. 28, no. 1, pp. 113–130, 2010.

[25] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou, “Anytime dynamic path-planning with flexible probabilistic roadmaps,” in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. IEEE, 2006, pp. 2372–2377.

[26] R. Gayle, A. Sud, M. C. Lin, and D. Manocha, “Reactive deformation roadmaps: motion planning of multiple robots in dynamic environments,” in Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on. IEEE, 2007, pp. 3777–3783.

[27] Y. Kuwata, S. Karaman, J. Teo, E. Frazzoli, J. P. How, and G. Fiore, “Real-time motion planning with applications to autonomous urban driving,” Control Systems Technology, IEEE Transactions on, vol. 17, no. 5, pp. 1105–1118, 2009.

[28] D. Ferguson, N. Kalra, and A. Stentz, “Replanning with rrts,” in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. IEEE, 2006, pp. 1243–1248.

[29] M. Otte and E. Frazzoli, “RRTX: Real-time motion planning/replanning for environments with unpredictable obstacles,” in Algorithmic Foundations ofRobotics XI. Springer, 2015, pp. 461–478.

[30] H. Nguyen and H. La, “Review of Deep Reinforcement Learning for robot manipulation,” 2019 Third IEEE International Conference on Robotic Computing (IRC), 2019.

[31] Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng. Robotic Grasping of Novel Objects using Vision. The International Journal of Robotics Research (IJRR), 27 (2):157–173, 2008.

[32] A. Sahbani, S. El-Khoury, and P. Bidaud, “An overview of 3D object grasp synthesis algorithms,” Robotics and Autonomous Systems, vol. 60, no. 3, pp. 326–336, 2012.

[33] Y. Li, Y. Yu, and S. Tsujio, “An analytical grasp planning on a given object with multifingered hand,” Proceedings 2002 IEEE International Conference on Robotics and Automation, 2002.

[34] M. Gualtieri, A. ten Pas, and R. Platt, “Pick and place without geometric object models,” 2018 IEEE International Conference on Robotics and Automation (ICRA), 2018.

[35] D. Kalashnikov, “Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation,” 2nd Conference on Robot Learning, 2018.

[36] MATLAB, “https://www.mathworks.com/help/nav/ug/motion-planning-with-rrt-for-manipulators.html,” Motion Planning with RRT for a Robot Manipulator. [Online]. Available: https://www.mathworks.com/help/nav/ug/motion-planning-with-rrt-for-manipulators.html. [Accessed: 28-Nov-2021].

[37] P. Cadic, “Free CAD designs, Files &amp; 3D models: The grabcad community library,” Free CAD Designs, Files &amp; 3D Models | The GrabCAD Community Library, 12-Apr-2020. [Online]. Available: https://grabcad.com/library/opensource-robot-based-on-rpipi3a-1. [Accessed: 12-Dec-2021].

[38] M. A, “Free CAD designs, Files &amp; 3D models: The grabcad community library,” Free CAD Designs, Files &amp; 3D Models | The GrabCAD Community Library, 14-Aug-2021. [Online]. Available: https://grabcad.com/library/robot-arm-171. [Accessed: 12-Dec-2021].

[39] Larson Electronics, “Free CAD designs, Files &amp; 3D models: The grabcad community library,” Free CAD Designs, Files &amp; 3D Models | The GrabCAD Community Library, 20-Jul-2017. [Online]. Available: https://grabcad.com/library/explosion-proof-motion-sensor-10-to-20-mounting-height-15-x-15-area-adjustable-timer-lv-2. [Accessed: 12-Dec-2021].

[40] A. Salimnezhad, “Free CAD designs, Files &amp; 3D models: The grabcad community library,” Free CAD Designs, Files &amp; 3D Models | The GrabCAD Community Library, 12-Jul-2014. [Online]. Available: https://grabcad.com/library/webcam-5. [Accessed: 12-Dec-2021].

[41] A. Z. H, “Free CAD designs, Files &amp; 3D models: The grabcad community library,” Free CAD Designs, Files &amp; 3D Models | The GrabCAD Community Library, 15-Jul-2021. [Online]. Available: https://grabcad.com/library/flir-flea-c-mount-2021july-1. [Accessed: 12-Dec-2021].