Robot Can Imagine: Object Detection using deep Text-to-Image Generative Adversarial Networks

Bhart Gupta[1*], Aditya Srichandan[2*], Prakasam P[3†], Noor Mohammed V[4]

School of Electronics Engineering, Vellore Institute of Technology, Vellore, India

[1]bhart.gupta2017@vitstudent.ac.in, [2]aditya.srichandan2017@vitstudent.ac.in,
[3]prakasamp@gmail.com, [4]noorb4u@gmail.com

Abstract

Assistive and home service robots have been important as well as lucrative applications in burgeoning robotics technology. In this paper, we propose an object detection pipeline for home service mobile robots to detect and fetch objects using GANs. Similar to how humans teach about objects to younger ones using imagination, we aim to make the robot imagine the object to be fetched from the surroundings based on user's speech. Robot Operating System (ROS) is used to develop and simulate the robot in a home environment. The robot detects objects from different rooms by imagining the object using text-to-image GANs (generated image) based on the user's speech command and compares similarity score of generated images with real images in the environment using Deep Image Similarity (DIS). SLAM, and path planning of robot is incorporated to achieve the proposed overall object detection and fetching pipeline. Finally, we compare our object detection pipeline with YOLOv4 on object detection and fetching success.

Keywords: Generative Adversarial Networks, Text to image synthesis, Computer Vision, mobile robots, Object Detection

1. Introduction

Robot assistance applications are used in many fields such as hospitals for assisting in surgeries, factory automation and industry 4.0 for assembling parts, space robotics to assist humans with images and data to research on different planets, iRobot that assists by working as a vacuum cleaner, etc. With the rise of AI, Deep Learning, and big data, there have been a lot of improvements to make applications more robust and unique. In this paper, we aim to explore one of the applications of home service robots that help in fetching items and objects from different locations in the house. We introduce an approach in applications of service robots to assist users to fetch objects in the environment by making it imagine objects that it has to fetch as per the given speech command by the user. There has been a lot of work going on for object detection using deep learning [1][2], path planning and navigation of robots, etc. that are commonly used in these applications and most of the work discusses solutions for specific topics for the same [3]. Our work focuses mainly on a novel approach for detecting and fetching objects from the surroundings by imagining objects based on the user's speech and present an end-to-end solution of robot assisting humans to fetch objects using text-to-image GANs. The robot consists of a camera and a 2D lidar laser scanner as a part of its perception. Our home gazebo simulated environment consists of 4 rooms, (bedroom, living room, kitchen & drawing room) with few objects. We use fruits as objects for the implementation of the paper. The user commands the robot to fetch certain items from one of the rooms. The robot will imagine the objects based on the user's speech and will start navigating to the desired location and try to search the object in the environment by matching it with its imagined image. In our paper, we address the "Imagining objects" part using text to image GANs and speech recognition [4][5]. The user's speech is converted to text, the text is converted to an image and the robot uses this image and compares it to match with the surrounding objects in the environment and fetches if the match was successful. Our novelty lies in the approach to implement an end-to-end application of assistive home service robots using the imagination of objects based on the user's speech followed by path planning towards its goal, search object, match the surrounding desired object after reaching the goal and compare it with imagined ones. This would ensure robot has a vague idea of what it is fetching even if the object in the environment is not same as imagined one. For example, If the user asks robot "bring me a green apple from the living room", the robot tries imagining

[*]The authors contributed equally to this work.
[†] Corresponding author

a green apple from the text-to-image GAN models and fetch the same even if it was trained on red apples and different green coloured objects. Though in this paper we implement by training the text-to-image GANs model using the Rotten Fresh fruits dataset, the same approach can be applied on any dataset for imagining objects. In this paper, we implement the application on Gazebo real-time simulation, with the same parameters and domain randomization for different lighting conditions [6] we can implement in real robots.

The major contributions of this research are

- We devised a pipeline for robots to detect and fetch objects using imagination for which we used Text-to-Image GANs and algorithm 1-5.
- We employed self-attention along with residual connections in both generator and discriminator networks.
- Validated the proposed model with fresh and rotten fruits dataset for which we wrote the captions manually and compared with the existing models.

2. Related Works

Robotic vision/perception is a key part of the fetching process of the application. Object detection Deep learning models such as YOLO, RCNN are used but require lots of training images and complex CNN architectures. They tend to perform well under different lighting conditions with region proposals and anchor box techniques. GANs based object detection for fetching has recently come into the application of object detections [7]. Text to an image is widely being researched upon and GAN [8] is the most popular model employed for this [9] made one of the earliest attempts to synthesize an image from text with the architecture of conditional GAN which makes use of a character-level convolutional-recurrent neural network [10] to encode text description and generate 64x64 images based on the text description. For images with higher resolution, HDGAN [11] was used which makes use of a generator that hierarchically synthesizes multiscale images ranging up to 256x256 resolution from input text. [12] uses a condition text description along with the image to the generator and couple it with a discriminator to attain fine-grained attribute distribution. Further StackGAN [13] and SpeckleGAN [14] structures are introduced which use the technique of stacking up multiple generators in sequence to generate an image from coarse to fine, each generator has its discriminator for training. These models optimize match-aware loss. Based on the stacked structure [15] StackGAN++ proposed a generative network based on attention to assist generation between word-level text features and image features [16] proposed a structure with one stage consisting of only one generator and discriminator for text to image generation. The generator consists of a series of Up Blocks designed for upsampling the image features for generating high-resolution images. AttnGAN [17] makes use of cross-modal attention to have a word-context vector for every small region in an image and use concatenation on them for text to image fusion. It also introduces Deep Attentional Multimodal Similarity Model to calculate the similarity between image and text at the word as well as sentence level to have a fine-grained loss for generation of the image. All this is done to make the image semantically consistent with the text. ControlGAN [18] introduces channel-wise attention and word-level spatial block for the synthesis of features that correspond to the most relevant words in the process of generation. DMGAN [19] makes use of a memory network to select essential text information dynamically based on the image currently being generated to refine the image features. Semantic-conditioned batch normalization [20] conducts a batch normalization conditioning on global sentence vectors and batch normalization is made to condition on word vectors. MSGAN [21] introduced an effective mode-seeking regularization term that can be used in arbitrary conditional adversarial networks in tasks to remove the issue of mode collapse and hence improve diversity. Keeping in mind about all previous works in object detection/image generation and since no notable work is published with text-to-image GANs as object detectors, in this paper we introduce an approach to use text-to-image GANs for object detection and fetching object process instead of deep learning object detection models and compare the performance of such approach with existing ones.

3. Materials and Methods

The proposed research introduces a more human-level approach to teach a robot to fetch objects and use them as a home service robot and focuses on a novel way of teaching the robot through imagination. Similar to how humans learn based on imagination, we aim to teach the robot to imagine the objects we ask it to fetch, and the imagined image is compared with its surroundings to get the match. The overall process flow of the proposed method with example of *"get me a fresh ripe orange from kitchen"* asked by the user is illustrated in Fig 1.

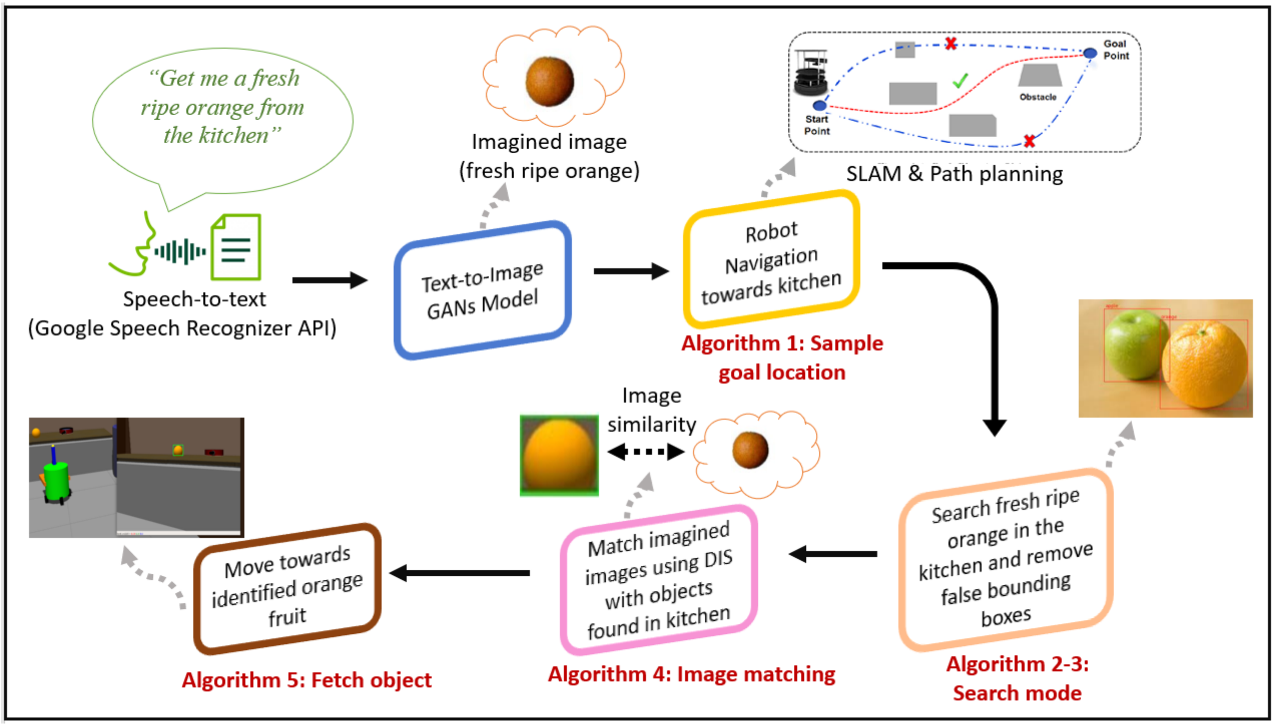

"Get me a fresh ripe orange from the kitchen"
Speech-to-text (Google Speech Recognizer API)
Imagined image (fresh ripe orange)
Text-to-Image GANs Model
Start Point
Goal Point
Obstacle
SLAM & Path planning
Robot Navigation towards kitchen
Algorithm 1: Sample goal location
Image similarity
Search fresh ripe orange in the kitchen and remove false bounding boxes
Algorithm 2-3: Search mode
Match imagined images using DIS with objects found in kitchen
Algorithm 4: Image matching
Move towards identified orange fruit
Algorithm 5: Fetch object

Figure 1: Overall Pipeline with example of fetching orange fruit from kitchen

The pipeline consists of six parts.

- Speech to text
- Text to image Generative Adversarial Networks
- Robot environment
- Object search
- Image Matching
- Fetching

3.1 Speech to text

The first part of the pipeline includes communicating with the robot to fetch an object from the environment. There are several improvements in recent years for speech recognition with the release of Deep speech [22], Since these models require a lot of training and computation, and as per our requirement, the robot should have less computation from its end for faster user interactivity. Therefore, we used speech_recognition API (python) which uses google_recognizer for speech to text, and for this, the API provided faster and convenient results.

3.2 Text-to-Image Generative Adversarial Networks (TI-GANs)

For the training of generator network, we use both the text embeddings and noise in a concatenated form and train the model so that it can model noise based on the text input provided. The concatenated input is first passed to a convolution layer and then passed through an activation layer. The generator network can be considered as a series of two cells, a residual cell similar to [23] and an up-sampling cell.

3.2.1 Generator Residual Block (GRB)

One generator residual block consists of one convolution layer, one batch normalization layer and one PRelu activation function followed by another convolution layer, batch normalization layer, with input feature map (X) as in figure added to F(x), as shown in figure 2.

3.2.2 Generator Residual Cell (GRC)

The residual block is passed through a self-attention layer and this in whole is known as one generator residual cell. See figure 2.

3.2.3 Self-Attention Module (SA Module)

Since we have to move down in network for convolutional kernel's receptive field to become large enough to capture things, we use self-attention which looks at every pixel in feature map and find what we must pay attention to. We project every input feature into three vectors, key, query and value. The value represents input features, to have a closer look in the local regions of input, value has reduced dimensions from input features both in channel numbers and size of activation map. The number of channels is reduced by 1x1 convolution. Key and queries compute importance of features of self-attention map and are obtained using 1x1 convolution. For calculation of output feature at a particular location we take query at that location and compare it with key at all locations. For example, for feature 0, we calculate q_0 x k_0 to q_0 x k_{N-1}, where q_i and k_i represents query and key at i^{th} feature which is normalized using SoftMax. This is used as weight to perform element-wise multiplication of value, to output attention weights. Another 1x1 convolution is used to restore input channels. Furthermore, a residual connection is used to add input to the output (see figure 2).

3.2.4 Discriminator Network

3.2.4.1 Discriminator Residual Block (DRB)

Similar to generator residual block, one block consists of a convolution layer, a batch normalization layer, and a PReLu activation function followed by another convolution layer, batch normalization layer, with input feature map (X) as in figure is passed through another convolution layer, and its output is added to F(x) and thus the output from DRB can be interpreted as – F(x) + Conv(X) as shown in figure 2.

3.2.4.2 Discriminator residual cell (DRC)

The output from residual block is passed to one PReLU activation function and one self-attention layer, and thus called one discriminator residual cell, illustrated in figure 2.

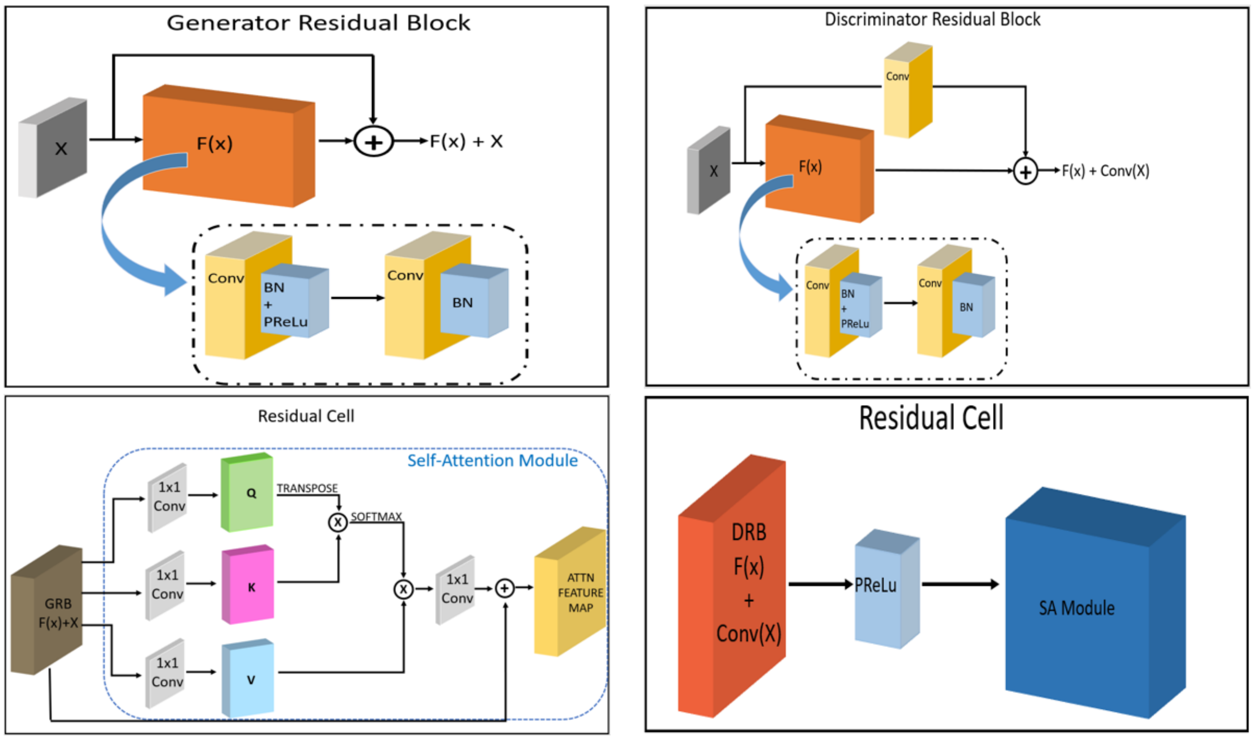


Figure 2: Generator Residual Block (GRB) architecture (top left); Discriminator Residual Block (DRB) architecture (top right); Generator Residual Cell (GRC) with attention module architecture (bottom left); Discriminator Residual Cell with same attention module as used in GRC (SA Module)

3.2.4.3 Loss Function

The text and image pairs are concatenated and fed to discriminator. As shown in Figure 8 and its output shown in equation (1), (2), and (3) is used to calculate Loss function [9] as described in equation (4) and (5)

$$d_{rr} = D(x,h) \ \{real\ image, right\ text\} \quad \ldots.\ (1)$$
$$d_{rw} = D(x,h') \ \{real\ image, wrong\ text\} \quad \ldots.\ (2)$$
$$d_{gw} = D(x',h) \ \{generated\ image, right\ text\} \quad \ldots.\ (3)$$

Where D(x,h) is the discriminator output with image x and text h as input

Loss of discriminator network, $L_D = \log(d_{rr}) + (\log(1 - d_{rw}) + \log(1 - d_{gw}))/2$ . . .. (4)

Loss of generator network, $L_G = \log(d_{gw})$ . . .. (5)

The loss that is to be optimized is $= L_G + L_D$ . . .. (6)

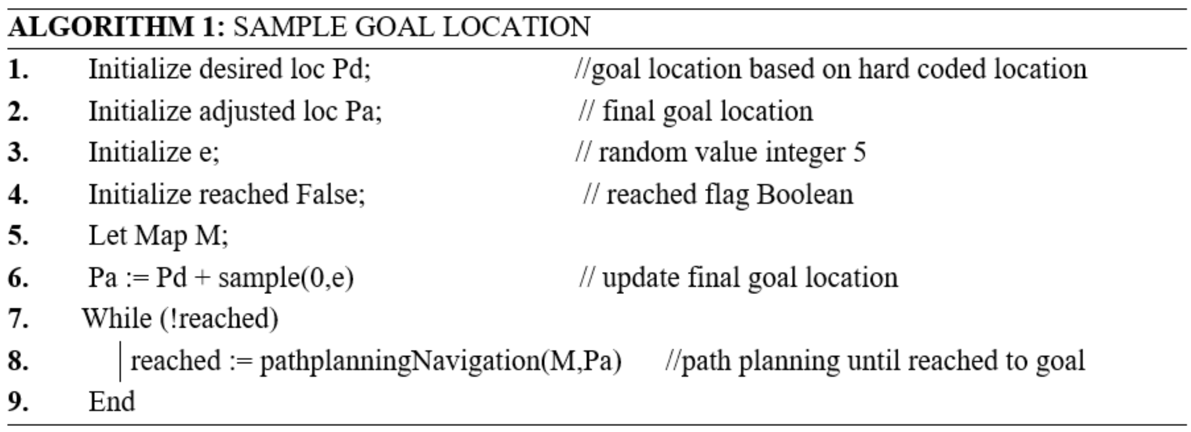
3.3 Robot Environment

Previous steps will all be running in the robot environment that uses ROS. We made a real-time simulation on the gazebo with all the pipeline logic in ROS. As the Imagined image from the previous pipeline is available to the robot, it starts the search in the environment to complete the given task of fetching the object from a given location. We do the process in two parts -

1. Go to Goal Location
2. Search Mode

3.3.1 Go to Goal Location

The proposed planning algorithm for sample goal location is shown in Algorithm 1. One assumption however we consider before the task execution is that a 2D map has been provided to the robot or it should be created through any of the 2-D SLAM techniques. Once we have the map, we used algorithm 1 to reach the goal location. ROS navigation stack provides required tools and applications to use for path planning and obstacle avoidance. We hard code the location with coordinates and sample it with some probability whenever the robot is asked, such that the coordinate of the location is in the vicinity of the desired goal location by some value $|e| < 5$ (found by trial and error) see algorithm 1. This introduces the bias that whenever the robot reaches the goal it is not in the same state as previous and thus brings more variability and flexibility for the robot to search for objects in the environment, as it ends up at a slightly different location than the previous time and can be observed in *'Pa'* adjusted goal pose. After the robot has reached the goal location the search mode of the robot is triggered up.



ALGORITHM 1: SAMPLE GOAL LOCATION

```
1.  Initialize desired loc Pd;          //goal location based on hard coded location
2.  Initialize adjusted loc Pa;         // final goal location
3.  Initialize e;                       // random value integer 5
4.  Initialize reached False;           // reached flag Boolean
5.  Let Map M;
6.  Pa := Pd + sample(0,e)              // update final goal location
7.  While (!reached)
8.      | reached := pathplanningNavigation(M,Pa)      //path planning until reached to goal
9.  End
```

3.3.2 Search Mode

The robot will stop and start the search mode by executing and slowly rotating in place for one complete round and try to match the surrounding objects with the imagined one with the camera mounted on the top. The proposed search mode algorithm is shown in Algorithm 2. Our goal is to go near the matched object with the imagined one to fetch. The robot does again an in-place rotation if the first attempt to detect an object is unsuccessful but now with zoomed/magnified frames to deal with vicinity issues. If again unsuccessful with the attempt, the robot tries to go to the nearest location to all corners of the room and repeat the process of in-place rotation till it finds the object. Upon completing all the processes if it still does not find the object, the robot declares the task to be done but with no matched objects in the room.

ALGORITHM 2: SEARCH MODE

```
1.  Initialize angularVelocity ph;                 //set angular velocity of robot
2.  Initialize pi  22/7;
3.  Initialize tick t;                             //set time counter
4.  Initialize imageFrameStream img;               //set image frame stream
5.  Initialize rotationNo rot;          // integer representing rotation number of the robot
6.  Initialize found flag;
7.  flag: = False                                  // set found object flag to false
8.  While (not flag)
9.      While (rot<2)                              // in place rotation if object not found in first pass
10.         if (rot: ==1)
11.             img: = magnify(img)                // zoom image if not found in 1st in place rotation
12.         End
13.         while ((2*pi/ph)>t)                    //for one complete rotation
14.             box: = ObjectSearch(img))          //get bounding boxes from image frame stream
15.             if(box)                            // if detect bounding box detected
16.                 stopRobot()
17.                 For each box                   // for each box do algorithm 4.
18.                     if (ImageMatch(img[box]))
19.                         flag:= True
20.                         fetch(img)             // fetch object if desired object detected
21.                         Break
22.                     End
23.                 End
24.             End
25.             startRobot()
26.             t: =t+1                            //update 't' to check one complete rotation
27.         End
28.         if(flag)                   //if desired object found and fetched in 1st in place rotation
29.             Break
30.         End
32.         rot: =rot+1                // if object not found, go for 2nd in place rotation
33.     End
34.     if(flag)                       //if object found in 1st or 2nd pass
35.         Break
36.     End
37.     do for all corners    // repeat both rotations for all corners until object found and fetched
38.     display (object not in the desired location)   // if not found after all the trials
39. End
```

3.4 Object Search

The robot after reaching the desired destination needs to find the object it was asked and, in our work, we consider fetching tasks simply by going towards the object. The algorithm used to do object search is shown in Algorithm 3. The object search process is divided into Contour detection, bounding box, and Most common colors for easy understanding.

3.4.1 Contour detection and bounding box

To differentiate our desired object from the background and obstacles we need to detect objects in the environment, for this, we have employed the technique of contour detection and selective bounding box search, where the robot will scan through the environment and detect the various contours and try to draw a bounding box around it. For contour detection, the robot will first process the image it gets by applying Gaussian blur to reduce the noise and smoothen the image then using a canny edge detector which uses a multistage algorithm to determine all the edges, and then we use dilation to structure element to probe and expand the shapes in the input image for better analyzing of the shapes and detecting all the shapes the environment [26]. This image is then used to generate all the contours and bounding boxes. Now the generated contours and their bounding boxes would have a lot of contours which will have along the required contours a lot of false contours and thus corresponding false bounding boxes.

3.4.2 Most common colors

To overcome the problem of false contours and bounding boxes, we only look into those boxes whose most common color is similar to the imagined image from our GAN. For example, we make use of the generated image received from GAN i.e., imagined image, and find the most common color in the generated image by going through every pixel of the image and storing the color of that pixel in a dictionary then we take the top colors to generate a mask, now this mask is applied to all the images within the bounding boxes. All the fake contour's bounding boxes not containing the desired object will have colors other than the desired ones, so we apply the mask within the generated contour's bounding boxes and those bounding boxes which have the desired colors in a good ratio are still stored and the rest of the bounding boxes are filtered.

ALGORITHM 3: OBJECT SEARCH

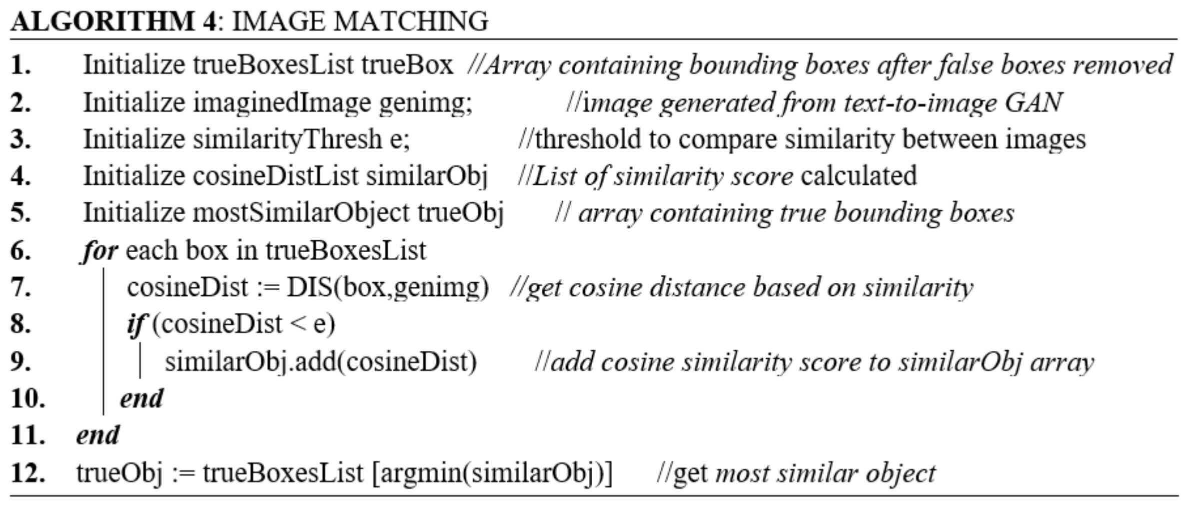
```
1.  Initialize imageFrameStream img;            // image frame stream
2.  Initialize imaginedImage genimg;            // image generated from text-to-image GAN
3.  Initialize clusters numCluster;             // number of clusters for k-means
4.  Initialize ratioThresh e;                   // threshold value
5.  Initialize trueBoxesList trueBox            // array containing boxes without false bounding box
6.  contours: = Contour(img)                    // get contours from image
7.  KmeansCenter: =  Kmeans (genimg. pixelsArray, numCluster) //get center of clusters
8.  commonColor: = histogram (Cluster (genimg, KmeansCenter), length (KmeansCenter))
9.  for each contour in contours
10.     box: = Drawbox (img, contour) // draw bounding boxes on contours
11.     ratio: = Mask(box, commonColor)  // mask out the most common colors within box
12.     if (ratio>e)                  // if ratio is greater than threshold then add to array trueBox list
13.         trueBox.add(box)
14.     End
15. End
```

3.5 Image Matching

The next part of the pipeline after successfully shortlisted a list of bounding boxes with the most common colors compared with the imagined image. The main aim is to find the object from the list which resembles most of the imagined image since there would be a few false positives in the list which could not be the desired object but some other object whose most common color could be similar to imagined ones. The algorithm used for the image matching process is illustrated in Algorithm 4. To achieve the same, we pass through each bounding boxed image and imagined image (generated image) into the DIS (deep image similarity) to extract features and calculate cosine similarity distance and store it in a list. To get the most similar desired object from the list of cosine distances, we first compare the cosine distance with a threshold to check if the box contains an object like our desired object, and then to get the most similar desired object from this, we take the minimum value cosine distance argument from the list and get the corresponding bounding box coordinates. Hence, we now know the desired object location in the camera frame and its bounding box. Next and the final part of the pipeline would be to track and move towards the object.

3.5.1 Deep Image Similarity

To match the generated image from GANs with surrounding objects a Deep Image Similarity model (DIS) is used for image pair similarity. This model is used in the Image matching pipeline, where each bounding box image from the true boxes list (i.e., after removing all false-positive contours and bounding boxes in the previous pipeline) is passed through this model and calculate cosine similarity distance between the imagined image (generated image GANs) and bounding box image. We pass the images through a pre-trained model such as VGG, and ResNet and remove the last fully connected linear layer. The remaining network's final layer now is a CNN layer acting as a feature extractor and the output feature map is fed to cosine similarity calculation to get a distance metric between the images.



ALGORITHM 4: IMAGE MATCHING

```
1.   Initialize trueBoxesList trueBox  //Array containing bounding boxes after false boxes removed
2.   Initialize imaginedImage genimg;        //image generated from text-to-image GAN
3.   Initialize similarityThresh e;          //threshold to compare similarity between images
4.   Initialize cosineDistList similarObj   //List of similarity score calculated
5.   Initialize mostSimilarObject trueObj    // array containing true bounding boxes
6.   for each box in trueBoxesList
7.   |   cosineDist := DIS(box,genimg)  //get cosine distance based on similarity
8.   |   if (cosineDist < e)
9.   |   |   similarObj.add(cosineDist)      //add cosine similarity score to similarObj array
10.  |   end
11.  end
12.  trueObj := trueBoxesList [argmin(similarObj)]     //get most similar object
```

3.6 FETCH

After the image matching, the same is used to fetch the object through imagining the object through speech would be fulfilled in this final part, where the desired object's bounding box obtained from the previous pipeline is used to calculate how much distance and at what pose the robot has to move to reach the object. The algorithm employed for fetching is shown in Algorithm 5. Since we use a 2D laser scanner and no 3D information is known. The robot starts to calculate the center coordinates of its frame and the center coordinates of the desired object's bounding; it observes if the center coordinate of the bounding box is less/greater than the center coordinate of the camera frame. If it is less, the robot rotates towards the left with some velocity until the difference between frame and bounding box is greater than some threshold and vice versa. This means the robot is aligned with desired objects' vicinity or Line of Sight (LOS). The center coordinate of the bounding box is updated while rotating in place by using tracking algorithms that track desired object bounding boxes and return the center in each frame. The camera frame's center and returned bounding box’s center from the tracking algorithm in each frame is used to check if the center of the bounding box and camera frame has less distance pixel-wise between them, which explains that the robot is aligned with the desired object.

ALGORITHM 5: FETCH

```
1.  Initialize ImgTracker tracker;                  //OpenCV tracker
2.  Initialize imageFrameStream img;                // image frame stream
3.  Initialize mostSimilarObject trueObj            // array containing true bounding boxes
4.  Initialize pixelDistThresh pdt;                 // pixel distance threshold
5.  Initialize angularVelocity ph;                  //angular velocity of robot
6.  Initialize linearVelocity xvel;                 //linear velocity of robot
7.  Initilize laserScanDist dist;                   // distance measured by laser scanner
8.  Initialize SafeDist ed;                         // safe distance threshold set before collision
9.  xImg,yImg := calculateCenter(img)               // center x,y of image frame
10. x,y := calculateCenter(trueObj)    //center x,y of each true bounding box in the array 'trueObj'
11. while(|(x-xImg)| > pdt )       //while bounding box center and image frame center not aligned
12.     if (x < xImg)              //bounding box center left to image frame center
13.         rotate("anti-clock",ph)        // rotate opposite to get object in Line of Sight
14.     End
15.     if (x > xImg)              //bounding box center left to image frame center
16.         rotate("clock",ph);    //rotate opposite to get object in Line of Sight
17.     End
18.     ph:=0                      //stop robot
19. End
20. dist := laserscan()            //get distance from laser scan
21. While (dist>ed)                //move till safe distance from the desired detected object
22.     move("straight",xvel);
23.     dist := laserscan()        //update distance
24. end
25. stopRobot();
```

4. Experimental Setup and Results

The experimental setup, simulation environment, the deep TI-GAN, and results obtained using the proposed method are explained in this section. This section deals with the dataset used, Robot Model employed the simulation environment, proposed deep TI-GANs, SLAM & Navigation process, searching the image using search mode, and finally matching the image with image similarity. Gazebo and ROS simulation is used to demonstrate the full pipeline.

4.1 Dataset

Fruits Fresh and Rotten [‡] dataset is used for images and we have manually written one caption for each image for 6 classes with 300 images in each. The dataset contains fresh apples, fresh oranges, fresh bananas, and rotten counterparts of the same classes, totaling six classes. Each caption describes the image's color, texture, shape, and size. The user will describe the object to the robot with its location.

4.2 Robot Model

We have modeled a two-wheeled robot with a tray that can help in fetching/carrying objects. The robot model, which is illustrated in figure 3, consists of a camera with 80.0-degree horizontal FOV and a 2D laser scanner for obstacle detection and SLAM, with min/max angles of scan being +/- 90 degree, thereby scanning an area by 180 degrees in one second. The range and resolution of the laser scanner are set to 30.0m and 0.01, respectively.

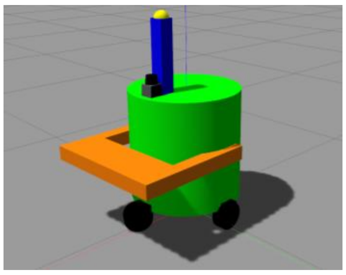


Figure 3. The robot model

[‡] https://www.kaggle.com/sriramr/fruits-fresh-and-rotten-for-classification

4.3 Environment

The home environment is modeled in Gazebo that includes A drawing room (hall), living room, kitchen, and bedroom. Our robot is placed in the home environment and is expected to fetch the objects kept in the room by understanding speech commands given by the user about what to fetch and from where. We created a few blender 3D models of apples, bananas, and oranges to match the objects with our generated imagined images. The simulated environment for the proposed research using Gazebo is shown in figure 4.

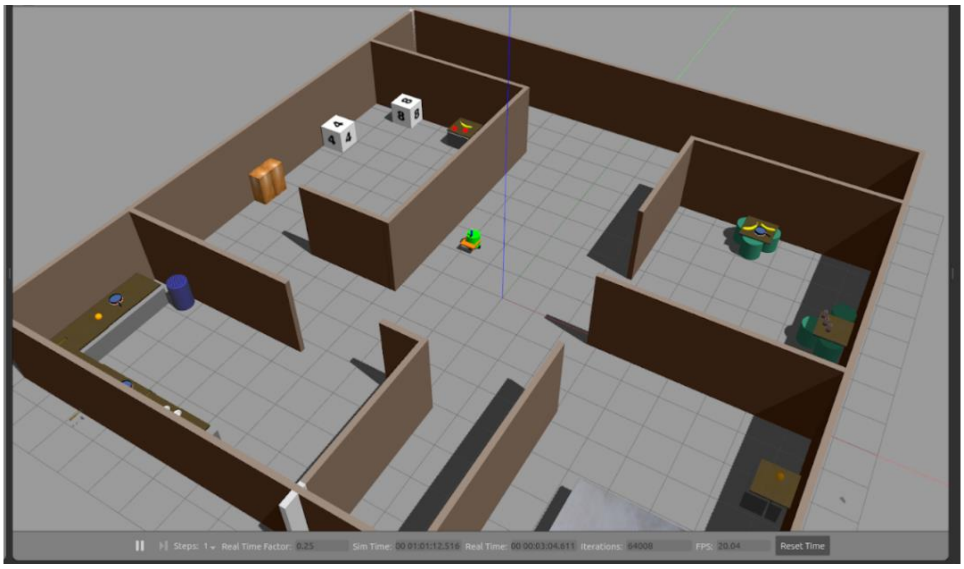

Steps: 1 Real Time Factor: 0.25 Sim Time: 00 01:01:12.516 Real Time: 00 00:03:04.611 Iterations: 64008 FPS: 20.04

Figure 4. The Environment in the gazebo

4.4 Deep Text-to-Image GANs (TI-GANs)

The deep TI-GAN is illustrated in Fig 5. In the proposed deep TI-GANs, pre-trained text embeddings Google news is used instead of normal text embeddings [32]. The text encoding consists of word embedding of size 300, the noise and text encodings are reshaped to 8x8x128 and concatenated and passed through a convolution layer with parameters (filter size, kernel size, stride) as (64,3x3,1) passed through PRelu. Then there are 4 Generator Residual Cells (GRCs), within each GRC there is a GRB and a self-attention module. The two convolutional layers parameters of GRB are set with (64,3x3,1) and (64,3x3,1) respectively. In self-attention module of each GRC we use 1x1 convolution and reduce channel number of C to C/k where we have taken k=8 [31] to get Query, Key, Value see figure Figure3. Finally, we get the self-attention feature map and forwarded to next GRC. The output of 4th GRC is added to the input of the 1st GRC which is then passed through 4 up sampling cells each consisting of conv2DTranspose layer and a batch normalization layer. The conv2DTranspose layer in the cells with parameters as (512,3x3,2), (256,3x3,2), (128,3x3,2), (64,3x3,1) respectively. It is finally passed through convolutional layer of (3,3x3,1) to produce 64x64x3 sized images.

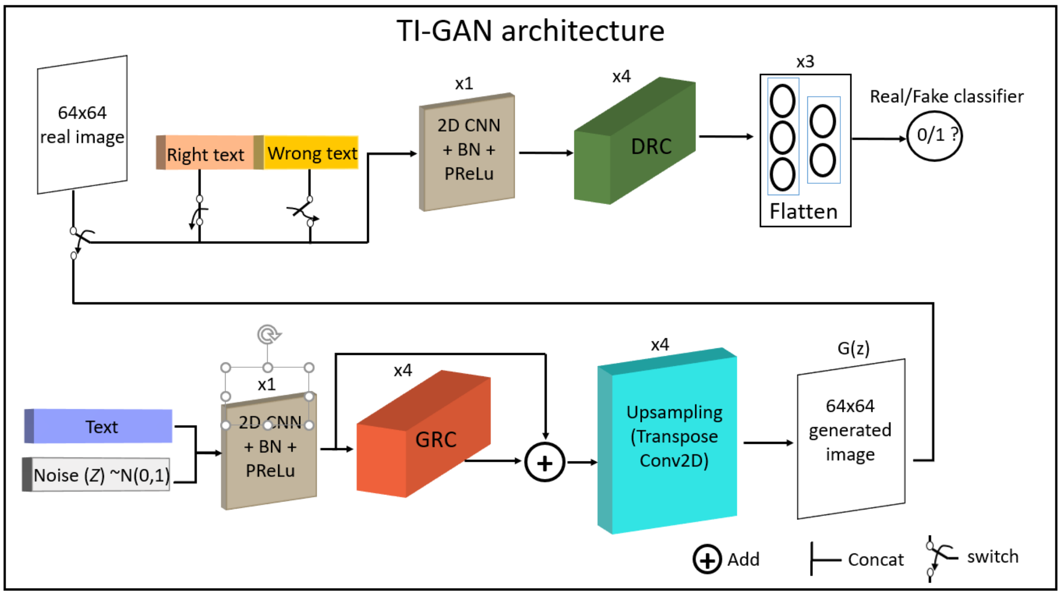
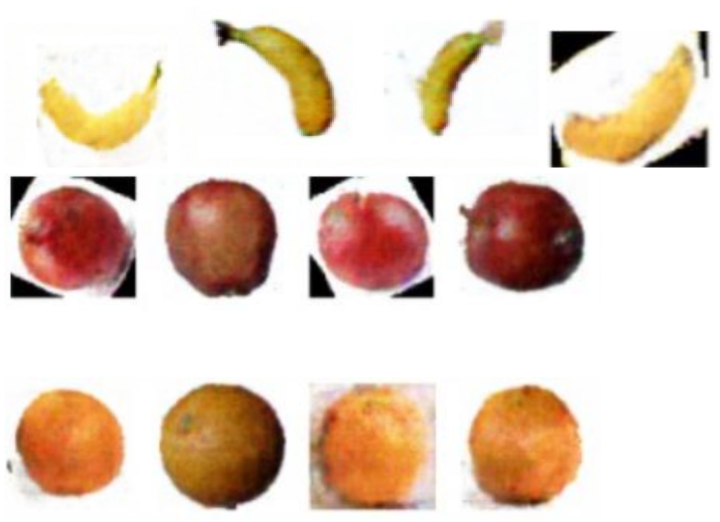
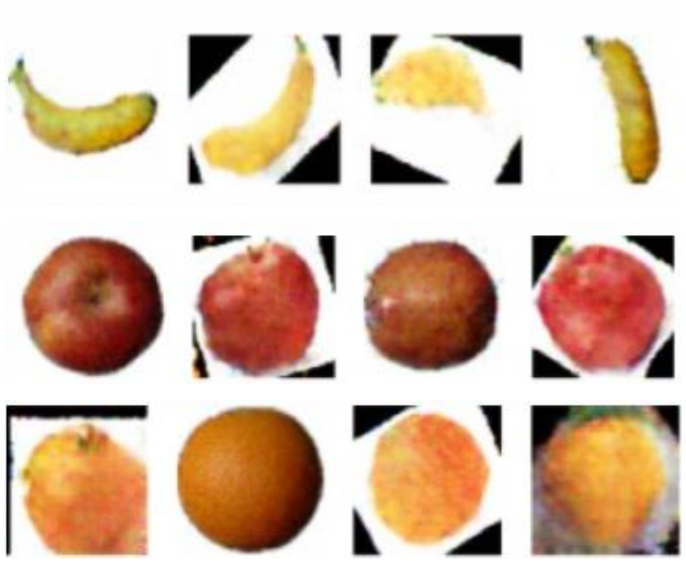
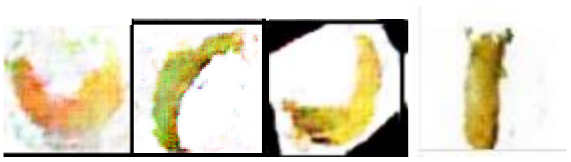
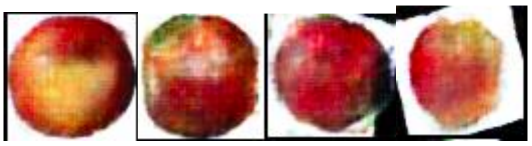
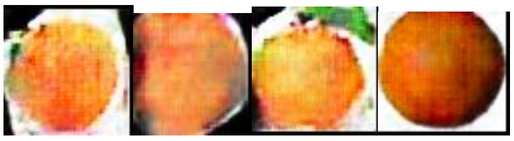


Figure 5: Proposed Deep Text to image GAN architecture (TI-GAN)

In the Discriminator network, we concatenate (text, image) pairs and input them as (real image, right text), (real image, wrong text), and (generated image, real text) [9] where the text embedding size is 300 and image size of (64x64x3). The switch in Figure 8 lets us to pass in the image, text pairs in further network as described above. Further the output is passed through a convolution layer with (filter size, kernel size, stride) as (64,3x3,1) and then passed through batch normalization, leaky Relu. Then there are 4 Discriminator Residual Cells (DRCs), within each DRC there is a DRB and a self-attention module. Self-attention module is similar to GRCs. Within each DRB there are two convolution layers with a parallel convolution layer's output added to the output of DRB see Figure 4 In the first DRC the two convolution layers has parameter (64,3x3,1) and (64,3x3,2) while the parallelly connected convolution layer is set with (64, 3x3,2) within the DRB. In the further remaining DRBs of each DRCs, the parameters of two convolution layers are set with [(128,3x3,1),(128,3x3,2)], [(256,3x3,1),(256,3x3,2)], [(512,3x3,1),(512,3x3,2)] respectively within the DRBs while the parallelly connected convolution layers are set with (128,3x3,2), (256,3x3,2), and (512,3x3,2) respectively. Finally, we flatten and pass-through dense layer to give a 1x1 output, classifying whether the image is fake or real. The TI-GAN used various simulation parameters such as optimizer as Adam, learning rate as 0.0002, momentum as 0.5, and a batch size as 64 [9]. The images that have been generated from the text using the proposed model are compared with existing MS-GAN [21] and GAN-CLS [9]. The same is shown in table 1.

Table 1 Comparison of the generated images from the text with existing models

Generated Images from text descriptions	**Text descriptions**	**Model**
	*A fresh ripe yellow banana* *A red round apple* *A fresh orange-colored fruit*	Proposed Deep TI-GANs
	*A fresh ripe yellow banana* *A red round apple* *A fresh orange-colored fruit*	MS-GANs [27]

	*A fresh ripe yellow banana*	
	*A red round apple*	GAN-CLS [12]
	*A fresh orange-colored fruit*	

From table 1, it is observed that proposed deep TI-GANs generates the more accurate image based on the text descriptions as compared with MS-GAN [21] and GAN-CLS [9] models.

4.5 SLAM & Navigation

Our work assumes that the map of the environment is given to the robot. The map can be provided in many ways to the robot, either by directly providing the known map of the area and localizing it or asking the robot to build the map and localize simultaneously. We rather take the latter approach in our demonstrated work. We used a ROS package called *gmapping*. Once the map is generated, the robot can now do navigation since it knows its position in the area. In our work, we simply made a dictionary mapped to words that are probable from users with their locations in the map by hard coding it. The navigation map for robot moving is shown in figure 6 for the following example. "Get me a fresh ripe orange from *the kitchen*" the speech-to-text from google API that we used to translate the speech to text. The location here as "*kitchen*" is stored in our dictionary mapping to its coordinates and its description "fresh ripe orange" is passed through our GAN model to get an imagined image.

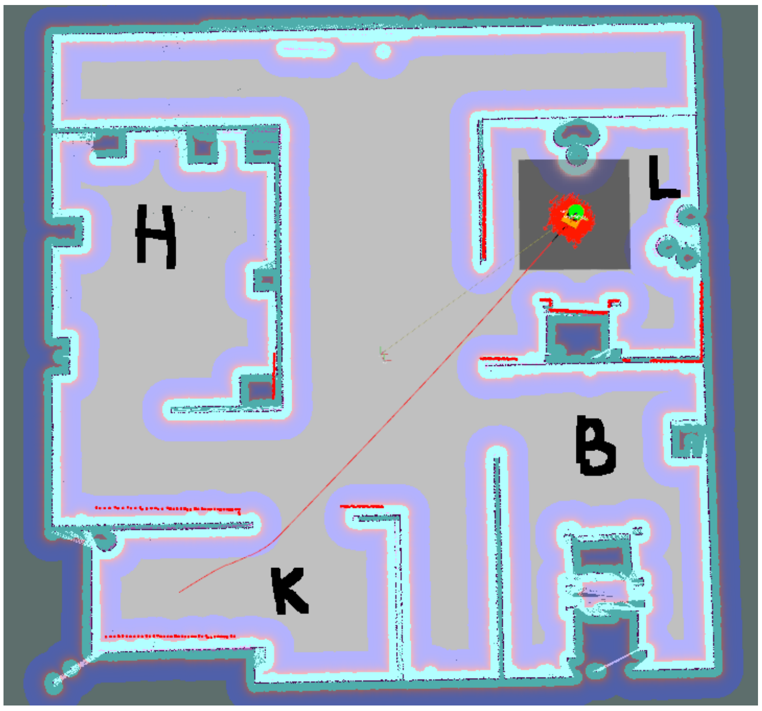


Figure 6. Navigation map for a robot moving to the kitchen (K) on command

"Get me a fresh ripe orange from the kitchen"

4.6 Search Mode

Once the robot reaches the location based on the command given by the user, now the robot stops and follows algorithm 2. It does in-place rotation two times to find the object from its current location. The frame update rate of the camera is set to 30Hz. The latter part of the in-place rotation is done by zooming the image frame, this was done taking into consideration of occlusion and other factors which could hinder the robot's vision while searching from the current location. If the robot fails to get anything from its current location (current location means the location after the robot reached its goal location given by the user) it will try to go four corners of the room and repeat algorithm 2 every step. If it succeeds in finding the object using Image Similarity matching, the robot simply goes towards the object by tracking the found object. In our work, we consider reaching towards the correct/desired object from the user as fetching and task successful execution. Figure 7 show the bounding box generation to search the required target with and without applying algorithm 3. The figure after applying algorithm 3 shows the better bounding boxes. Therefore, Algorithm 3 discusses how we implemented the false bounding box and contours elimination since while the robot is searching by doing in-place rotation, it can come across various objects and things not related to our goal or desired objects. We remove these false contours by filtering the boxes based on sizes and the most common color found in the box and make sure to filter out the boxes that are not similar or comparable with the imagined images by GANs.

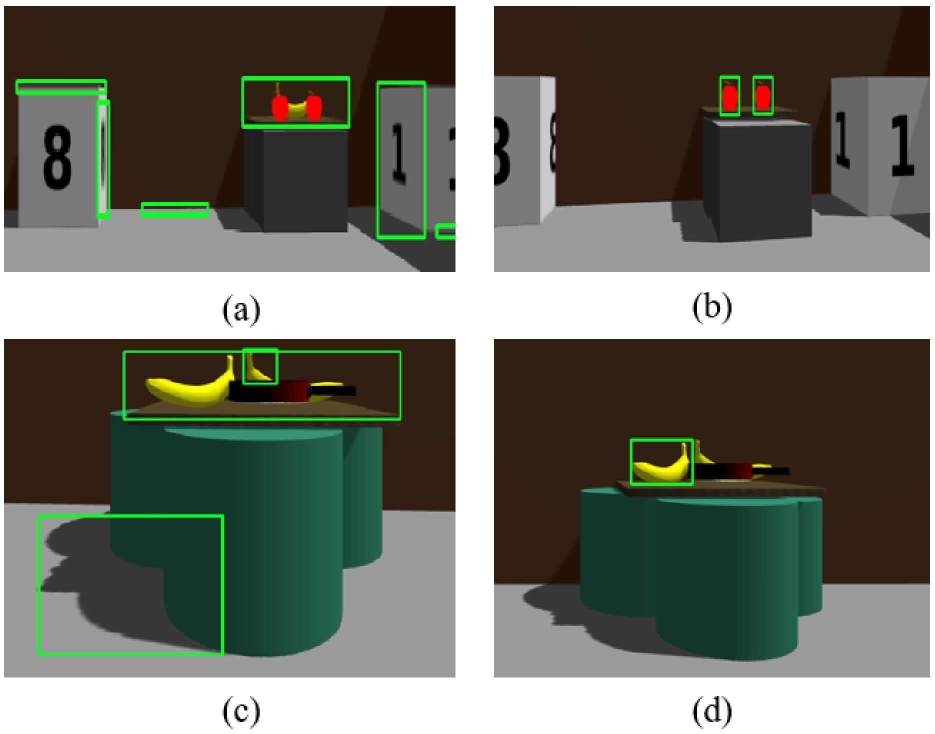


Figure 7: False Bounding boxes without Algorithm 3 (a), (c); filtered bounding boxes with Algorithm 3 (b), (d).

Once the correct and filtered bounding boxes have been found, still there could be some bounding boxes that could have similar most common color that is in imagined images. All the bounding boxes in the current image frame are passed through DIS which outputs the similarity score of each image passed from bounding boxes in the frame compared with imagined images (generated images). This is done to make sure that the good bounding boxes indeed contain the object that we are looking for, hence the deep image similarity model checks this using algorithm 4. After the DIS successfully identifies with reasonable similarity scores, the next thing is how does the robot know the location of the object in its map frame to go and fetch it? As the object's bounding boxes are in image frame coordinates. For this we use OpenCV’s CSRT tracker [28], there are a lot of other tracking options such as Kalman filter, KCF, Moose. We aim to get higher accuracy tracking rather than speed or FPS throughput since if we get lower accuracy our robot might not track objects well and not reach the desired location. Hence CSRT tracker fits better to our application compared to others though it comes with a cost of lower FPS speed. The illustration of the robot tracking the object and moving towards it using Algorithm 5 is shown in Fig 8. If the desired object is on the left, as explained in algorithm 5, the robot will turn left until the camera image frame and bounding box image frame's pixel-wise distance is less than some threshold value. This process can be considered as aligning the robot with an object, equivalent to Line of Sight, now the robot simply moves straight till it reaches some safe distance from the object which is calculated from the laser sensor.

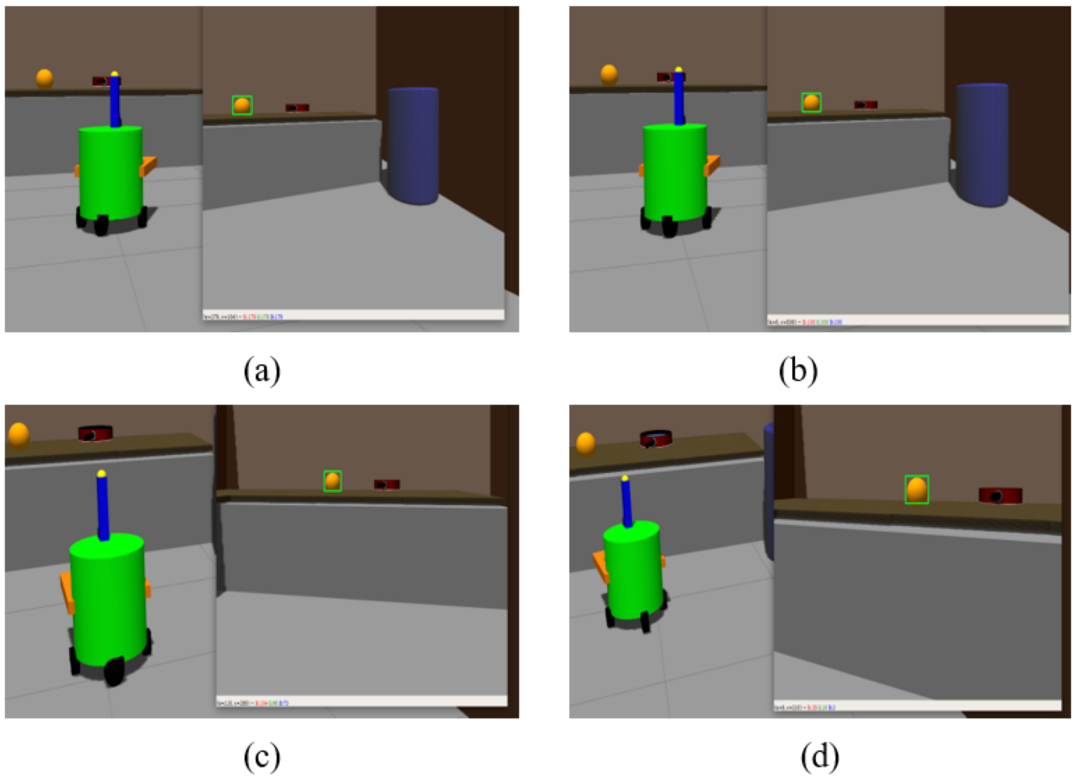


Figure 8: Illustration of robot tracking the object and moving towards it using Algorithm 5 (a)-(d).

4.6 Object Detection Model

We train YOLOv4[30] using the images of same 6 classes that we used for text-to-image GANs. Our purpose is to compare the object detection using our approach of imagining objects with existing deep learning object detection models.

5. Discussions

To analyze the performance of the proposed deep TI-GAN model, the Inception score [27] and Fréchet Inception Distance (FID) [29] are computed. The inception score is a metric that is used to measure the validation and accuracy of GANs which is used to generate the image. The optimum score of the inception scores is nothing but the class to be considered for the analysis which is present in the dataset. The FID is used to measure the quality of the generated image using GANs. FID is a distance between the generated distribution (image) and the actual distribution by inception network. The quality of the generated image using the proposed GAN is good if the FID value is low. The inception value and FID score of the proposed deep TI-GAN with existing GAN models are shown in table 2.

Table 2. The comparison of Inception score and FID score with existing models

Model	***Inception score***	***FID score***
MS-GAN [27]	3.30214705 +/- 0.14976212	305.23321907649176
GAN-CLS [12]	3.10256741 +/- 0.12743	320.14522904231166
Proposed deep TI-GAN	3.9484882 +/- 0.23692	197.37700224766064

From table 2, it is found that the proposed TI-GANs model achieved a better Inception score of 3.948 +/- 0.23692 and an FID score of 197.377 compared to other models. Also to validate the accuracy of the proposed deep TI-GANs model, precision, recall, and F1 score have been computed. Receiving a command to search the goal location and detect an object, we calculate similarity score using Deep Image Similarity and if the score is less than a certain threshold then we consider no object detected. We use a threshold of 0.4 by trial-and-error method depending on similarity values in each iteration. The two basic VGG-16 and ResNet50-V2 CNN models are employed in the proposed deep TI-GAN to classify the images for the robot. The various performance metrics are computed and compared with existing models based on the following assumptions.

- true positive: object present at goal location and detected correct class.
- true negative: object not present at goal location and not detected.
- false positive: object not present at goal location and detected wrong class.
- false negative: object present and at goal location and not detected.

The precision, recall, F1 score, and AUC has been calculated using the following formulas.

$$Precision = \frac{TP}{TP+FP} \tag{7}$$

$$Recall = \frac{TP}{TP+FN} \tag{8}$$

$$F1 - Score = \frac{2*Recall*Precision}{Recall+Precision} \tag{9}$$

$$AUC = \frac{TP+TN}{TP+TN+FP+FN} \tag{10}$$

The computed performance metrics of the proposed deep TI-GAN is tabulated in table 3 along with existing models.

Table 3 Performance comparison of Precision, recall, F1 score and AUC with other models using VGG-16 and ResNet50-V2

Metrics	***MS-GAN [21]***		***GAN-CLS [9]***		***Proposed deep TI-GAN***	
	VGG-16	*ResNet50-V2*	*VGG-16*	*ResNet50-V2*	*VGG-16*	*ResNet50-V2*
Precision	0.793	0.818	0.796	0.775	0.864	0.857
Recall	0.862	0.828	0.824	0.789	0.784	0.843
F1 score	0.826	0.838	0.810	0.782	0.8225	0.850
AUC	0.728	0.749	0.698	0.719	0.732	0.749

From table 3, it is found the proposed TI-GAN model archives the higher precision, recall, and F1 score using both baselines CNN as compared with existing models. Also, the proposed deep TI-GAN model achieved a better AUC value of 0.749 as compared with existing models.

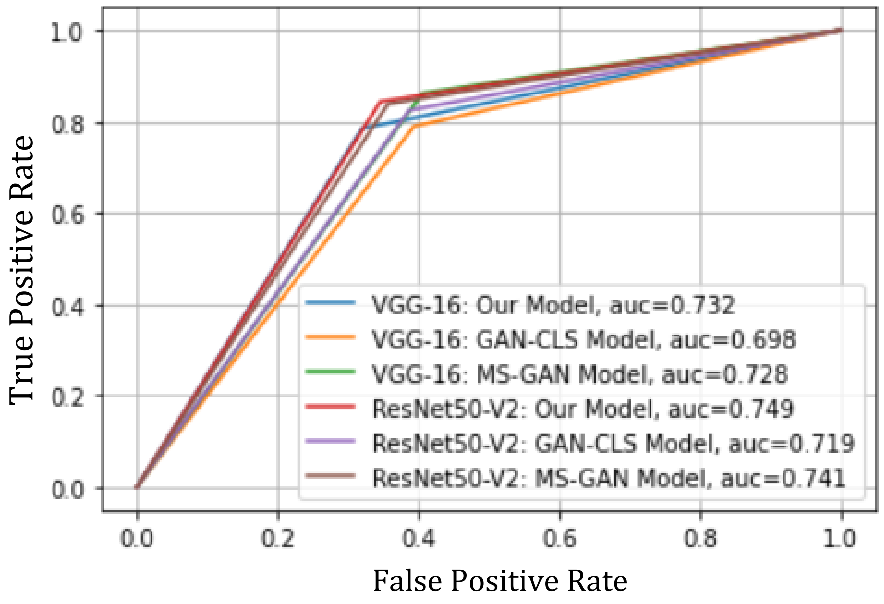


Figure 9: ROC curve comparing Text to Image models based for image similarity

The ROC curve is generated for the proposed deep TI-GAN and for the existing MSGAN and GAN-CLS models which utilize VGG-16 and ResNet50-V2 as the baseline network and is shown in figure 9. It is found that the proposed TI-GAN model performs slightly better than the existing models. Also, the similarity score has been measured for the proposed deep TI-GAN by considering both ResNet50-V2 and the VGG-16 baseline network. Based on the various speech commands, the proposed TI-GAN model generates the imagined image which almost matches the detected image in the bounding box. The similarity score of the proposed deep TI-GAN using ResNet50-V2 shown in Tables 4 and 6.

Table 4 Similarity score achieved using ResNet50-V2 between generated images based on command and object in the environment.

Sr. No	Detected Image bounding box	Imagined Image (Our GAN model)	Similarity score ResNet50-V2	Speech Command
1.			**0.689**	*Get me a fresh round orange from the kitchen*
2.			**0.697**	*Get me a yellow banana from the living room*

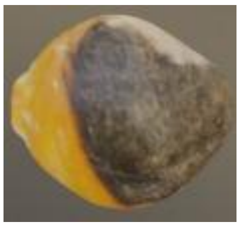
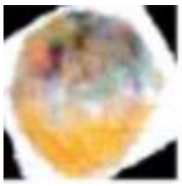
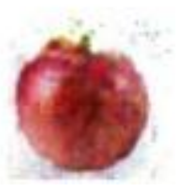
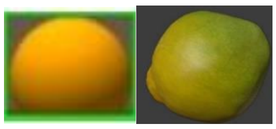
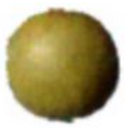
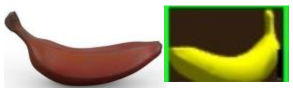
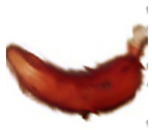
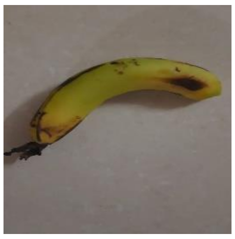
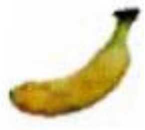
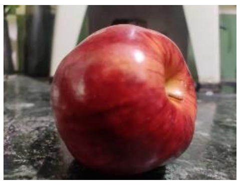
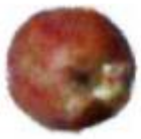
3.			**0.633**	*Throw away rotten orange from the kitchen*
4.			**0.704**	*Fetch me a fresh red round apple from Hall*

Table 5: Fetching success comparison between our model and YOLOv4

Sr. No	**Object in the environment**	**Imagined Image (Our GAN model)**	**Similarity score** **ResNet50-V2**	**YOLOv4 Prediction**	**Speech Command**
1.	 **A** **B**		**[0.390, 0.671]** **[A B]** Action: Move towards B	**[0.91, 0.772]** **[A B]** Action: Move towards A	*Bring me an orange with a greenish tinge.*
2.	 **A** **B**		**[0.702, 0.628]** **[A B]** Action: Move towards A	**[0.817, 0.904]** **[A B]** Action: Move towards B	*Get me a reddish fresh banana.*

In the above table we compare the action taken by the robot based on speech command and detected object using our model and YOLOv4. To compare object detection with TI-GANs and YOLOv4, we put two fruits of same class with different colours in the environment and test it based on action taken by the robot Table 5. We did not have greenish orange and reddish banana in the dataset. In Sr. 1 of Table 5, the environment has an orange and a greenish orange. Our model abides to speech command and generates a greenish orange, and the similarity score is higher for greenish orange compared to its counterpart. Thus, through our model, the robot moves towards greenish orange, which is the desirable action. Whereas YOLOv4 achieves higher prediction for the orange compared to its greenish counterpart. Robot moves towards higher prediction and hence it takes undesirable action as seen in table. Similarly for 2nd case with bananas, the robot through our model achieves desirable action. Since our Imagination learning approach tries to learn the representation of an image from words just as how humans imagine. The robot was able to learn about different colours, textures and other attributes from speech command and generate a representation of the image. Though YOLOv4 was able to detect fruits correctly but since it was only trained on orange, yellow banana and not to its counterpart, it was giving higher score to objects it was trained on and missed the desired action to be taken. We can say our model can understand from nuances of speech to generate images and use it to detect and fetch desirable object.

Table 6 Similarity score achieved using ResNet50-V2 between generated images based on command and real objects.

Detected Image bounding box	**Imagined Image (Our GAN model)**	**Similarity score ResNet50-V2**	**Speech Command**
		**0.648**	*Get me a yellow banana from the living room*
		**0.712**	*Get me a fresh red apple from the hall*

In the Table 4 and Table 6, we used our model's generated images to compare with the bounding box detected by the robot using the ResNet50-V2 model (since it showed the highest FID score compared to other models). We also used real bananas and apples to test the performance of the model.

6. Conclusion

In this research work, we have proposed a pipeline to make an assistive robot understand objects through their image representation and fetch them. Based on the speech command given by the user, the proposed deep TI-GANs generated images and then reach the desired location where it compared the generated images with objects at the location. The generated objects are found similar to "imagination" where the object would learn features from some dataset provided initially and generate images based on the description by using the features. We achieved this process by using attention mechanism and residual networks in TI-GANs architecture and calculated similarity scores for image comparison. Our approach works well as we saw in examples, that we can produce and detect images that are not in the dataset and compare the images based on similarity score. Comparing with existing object detection model YOLOv4, our model was able to fetch desired object based on speech command unlike YOLOv4. For future works, this method can be used as online-incremental learning and deep reinforcement learning to learn with fewer data initially to get image representation (GANs) and continuously learn about objects to fetch from its interaction with the environment.

7. References

[1] Bochkovskiy, Alexey et al. “YOLOv4: Optimal Speed and Accuracy of Object Detection.” *ArXiv* abs/2004.10934 (2020)

[2] Carion N., Massa F., Synnaeve G., Usunier N., Kirillov A., Zagoruyko S. (2020) End-to-End Object Detection with Transformers. In: Vedaldi A., Bischof H., Brox T., Frahm JM. (eds) Computer Vision – ECCV 2020. ECCV 2020. Lecture Notes in Computer Science, vol 12346. 213-219. Springer, https://doi.org/10.1007/978-3-030-58452-8_13

[3] Nandhini Abirami, R. Durai Raj Vincent, P. M. Srinivasan, Kathiravan. Tariq, Usman. Chang, Chuan-Yu. (2021) Deep CNN and Deep GAN in Computational Visual Perception-Driven Image Analysis, Complexity, Vol. 2021, 5541134, https://doi.org/10.1155/2021/5541134

[4] Alankrita Aggarwal, Mamta Mittal, Gopi Battineni, (2021) Generative adversarial network: An overview of theory and applications, International Journal of Information Management Data Insights, Vol. 1, No. 1, https://doi.org/10.1016/j.jjimei.2020.100004.

[5] Lan Lan, You Lei, Zhang Zeyang, Fan Zhiwei, Zhao Weiling, Zeng Nianyin, Chen Yidong, Zhou Xiaobo, (2020) Generative Adversarial Networks and Its Applications in Biomedical Informatics, Frontiers in Public Health, Vol. 8, DOI: http://doi.org/10.3389/fpubh.2020.00164

[6] Tobin, Josh & Fong, Rachel & Ray, Alex & Schneider, Jonas & Zaremba, Wojciech & Abbeel, Pieter. (2017). Domain randomization for transferring deep neural networks from simulation to the real world., 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 23-30, doi: 10.1109/IROS.2017.8202133.

[7] Marcelo J.G., Azcarraga A.P. (2020) Generative Adversarial Networks for Improving Object Detection in Camouflaged Images. In: Yang H., Pasupa K., Leung A.CS., Kwok J.T., Chan J.H., King I. (eds) Neural Information Processing. ICONIP 2020. Communications in Computer and Information Science, vol 1332. 426-433, Springer, https://doi.org/10.1007/978-3-030-63820-7_49

[8] Goodfellow, Ian & Pouget-Abadie, Jean & Mirza, Mehdi & Xu, Bing & Warde-Farley, David & Ozair, Sherjil & Courville, Aaron & Bengio, Y.. (2020). Generative Adversarial Networks. Communications of the ACM, 63(11), 139-144, https://doi.org/10.1145/3422622.

[9] Reed, Scott & Akata, Zeynep & Yan, Xinchen & Logeswaran, Lajanugen & Schiele, Bernt & Lee, Honglak. (2016). Generative Adversarial Text to Image Synthesis. Proceedings of The 33rd International Conference on Machine Learning, PMLR 48, 1060-1069.

[10] Xu, Y., Qian, Q., Li, H., Jin, R., & Hu, J. (2020). Representation Learning with Fine-grained Patterns. ArXiv:2005.09681 [Cs]. http://arxiv.org/abs/2005.09681

[11] Zizhao Zhang, Yuanpu Xie, and Lin Yang. (2018) Photographic text-to-image synthesis with a hierarchically-nested adversarial network. In Proc. IEEE Conference on Computer Vision and Pattern Recognition, pages 6199–6208, https://doi.org/10.1109/CVPR.2018.00649

[12] Seonghyeon Nam, Yunji Kim, and Seon Joo Kim. (2018). Text-adaptive generative adversarial networks: Manipulating images with natural language. Proceedings of the 32nd International Conference on Neural Information Processing Systems, 42–51.

[13] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. (2017) StackGAN: Text to photo-realistic image synthesis with stacked generative adversarial networks. IEEE International Conference on Computer Vision (ICCV), 5908-5916, doi: 10.1109/ICCV.2017.629.

[14] Bargsten, L., Schlaefer, A. (2020) SpeckleGAN: a generative adversarial network with an adaptive speckle layer to augment limited training data for ultrasound image processing. International Journal of Computer Assisted Radiology and Surgery 15, 1427–1436, https://doi.org/10.1007/s11548-020-02203-1

[15] Zhang, Han & Xu, Tao & Li, Hongsheng & Zhang, Shaoting & Wang, Xiaogang & Huang, Xiaolei & Metaxas, Dimitris. (2017). StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. IEEE Transactions on Pattern Analysis and Machine Intelligence. PP. https://doi.org/10.1109/TPAMI.2018.2856256.

[16] Ming Tao, Hao Tang, Songsong Wu, Nicu Sebe, Fei Wu, and Xiao-Yuan Jing. DF-GAN: Deep fusion generative adversarial networks for text-to-image synthesis. arXiv preprint arXiv:2008.05865, 2020

[17] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. (2018) AttnGAN: Fine-grained text to image generation with attentional generative adversarial networks. IEEE/CVF Conference on Computer Vision and Pattern Recognition, 1316-1324, doi: 10.1109/CVPR.2018.00143.

[18] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip H. S. Torr. (2019) Controllable text-to-image generation. https://arxiv.org/abs/1909.07083

[19] Minfeng Zhu, Pingbo Pan, Wei Chen, and Yi Yang. (2019) DM-GAN: Dynamic memory generative adversarial networks for text-to-image synthesis. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 5802–5810, https://doi.org/10.1109/CVPR.2019.00595

[20] Guojun Yin, Bin Liu, Lu Sheng, Nenghai Yu, Xiaogang Wang, and Jing Shao. (2019) Semantics disentangling for text-to-image generation. IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2327–2336, https://doi.org/10.1109/CVPR.2019.00243

[21] Q. Mao, H. Lee, H. Tseng, S. Ma and M. Yang (2019) Mode Seeking Generative Adversarial Networks for Diverse Image Synthesis, IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 1429-1437, https://doi.org/10.1109/CVPR.2019.00152.

[22] Amodei, D. Ananthanarayanan, S. Anubhai, R et al. (2016) DeepSpeech: Scaling up end-to-end speech recognition in English and mandarin, Proceedings of the 33rd International Conference on International Conference on Machine Learning, Vol. 48, 173–182

[23] He, Kaiming & Zhang, Xiangyu & Ren, Shaoqing & Sun, Jian. (2016). Deep Residual Learning for Image Recognition. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770-778, doi: 10.1109/CVPR.2016.90.

[24] Chen, Y., Zhang, H., Liu, L. et al. (2021) Research on image Inpainting algorithm of improved GAN based on two-discriminations networks. Appl Intell 51, 3460–3474, https://doi.org/10.1007/s10489-020-01971-2

[25] Wang, Y., Ye, H. & Cao, F. (2021) A novel multi-discriminator deep network for image segmentation. Appl Intell, https://doi.org/10.1007/s10489-021-02427-x

[26] Xue Li, Muying Luo, Shunping Ji, Li Zhang & Meng Lu (2020) Evaluating generative adversarial networks based image-level domain transfer for multi-source remote sensing image segmentation and object detection, International Journal of Remote Sensing, 41:19, 7343-7367, https://doi.org/10.1080/01431161.2020.1757782

[27] Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., & Chen, X. (2016). Improved techniques for training GANs, Proceedings of the 30th International Conference on Neural Information Processing Systems, 2234–2242

[28] Farkhodov, Khurshedjon & Lee, Suk-Hwan & Kwon, Ki-Ryong. (2020). Object Tracking using CSRT Tracker and RCNN. Proceedings of the 13th International Joint Conference on Biomedical Engineering Systems and Technologies, 209-212, https://doi.org/10.5220/0009183802090212.

[29] Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., & Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local nash equilibrium, Proceedings of the 31st International Conference on Neural Information Processing Systems, 6629–6640

[30] Bochkovskiy, A., Wang, C., & Liao, H. (2020). YOLOv4: Optimal Speed and Accuracy of Object Detection. ArXiv, abs/2004.10934.

[31] Zhang, Han & Goodfellow, Ian & Metaxas, Dimitris & Odena, Augustus. (2018). Self-Attention Generative Adversarial Networks.

[32] Reed, Scott & Akata, Zeynep & Lee, Honglak & Schiele, Bernt. (2016). Learning Deep Representations of Fine-Grained Visual Descriptions. 49-58. 10.1109/CVPR.2016.13.