EE491 - EE492

Senior Design Project Semester Report

Continuous Unsupervised Learning of Object's Classes While Moving

Yiğit Korkmaz

Department of Electrical and Electronic Engineering

Boğaziçi University

Bebek, Istanbul 34342

Project Advisor: H. Işıl Bozma

January 4, 2020

Abstract

Segmentation is an important capability for robotic systems, especially in object detection and scene recognition. With the development of cameras capable to store color and depth images together, it is possible to use RGB-D data as the main source of the segmentation. With combined information, the accuracy and speed of segmentation processes are greatly increased. In this part of the project, we implemented and tested different RGB-D methods on a dataset to compare their speed and efficiency in our application.

Contents

List of Figures

List of Tables

1 Introduction

In today's world, many applications including robotics require understanding the objects present in one's surroundings. The resulting analysis is often represented by an objects' map - namely a graph of objects and their spatial relations [1]. The categorization of objects is a challenging problem. In most of the approaches, categorization has been treated as a supervised problem. Most approaches including deep convolutional neural network methods are supervised methods which require the training object candidates to be labeled. Although they perform impressively, they impose a limit on robots' autonomy. Furthermore, since the number of all possible objects is large with varying appearances and sizes, it is not feasible to achieve this object with off-line training sets. With new data coming in, learning should be done from the beginning. As such, approaches that consider life-long unsupervised learning of objects' categories are being developed. One such approach has been developed in the Intelligent Systems Laboratory [2, 3]. This approach presents a complete end-to-end unsupervised pipeline from the generation of object candidates through simply looking around to accumulating the knowledge of object categories in an open-ended manner. In this approach, only RGB data is used. However, if there is accompanying depth information, this data also needs to be taken into account. However, since robot is recording frames and creates object candidates by comparing segmented images, not only accurate but also fast segmentation algorithm is needed. Furthermore, while the generation of object candidates takes the temporal coherence of the incoming visual stream into account through utilizing a tracking method, it is not considered in the learning of objects' categories. However, such information can also be valuable to the learning process. Furthermore, the problem of integration of knowledge as the robot moves has not been also studied. This project will focus on these aspects of the problem.

2 Problem Statement

The goal of this project composes of three steps:

1. Generation of object candidates while taking both RGB and depth data into account, in a fast and accurate way,
2. To evolve the knowledge of objects' categories while taking the temporal nature of the determined object candidates into account.
3. To evolve an objects' map as the robot moves.

3 Method

The method is planned to consist of the following stages:

- Understanding the proposed approach: The proposed approach consists of three stages: generation of object candidates through simply looking around, reasoning regarding object categories and building objects' map. In the proposed approach, generation of object candidates are done in a turtlebot with a kinect mounted on it, where reasoning object categories and objects' map are currently done on another computer by using MATLAB. Implementation of the first part is done in ROS environment. Thus, it is important to fully understand ROS and its utilities as well as the algorithms used in the approach.

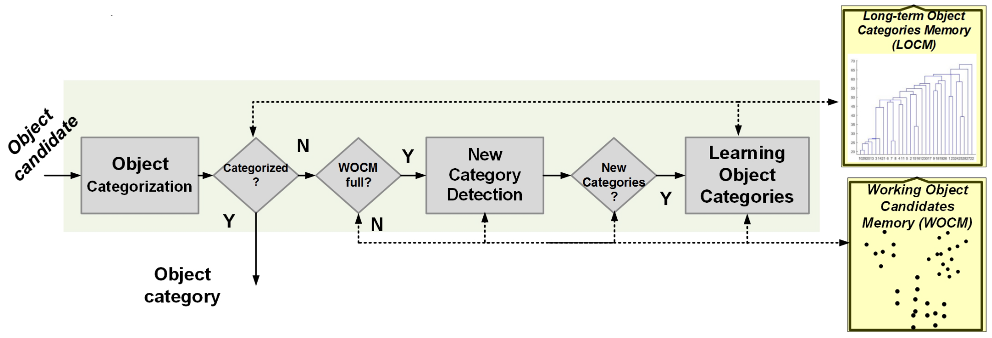


Figure 1: Life-long unsupervised learning of object categories [3].

- Improving the generation of object candidates through incorporating the depth data into segmentation and/or tracking. Segmentation is a process where an image is partitioned into multiple segments. It is generally used in object detection applications. In the current version of the approach, the segmentation is done on the RGB images by creating a vector that stores RGB values of every single pixel and comparing those values of a pixel with the ones near it. That way, one can detect abrupt discontinuities in pixel values, which typically indicate edges that define a region. Although this method is generally successful in detecting object, it fails in scenes where object is nearly same color with the background or where 2 objects with similar colors overlaps. However, it is possible to get an RGB-D image by using a camera which has a built in IR sensor, such as Microsoft Kinect. An RGB-D image is simpyly a combination of RGB image and its corresponding depth image, which is an image channel where each pixel relates to a distance between the image plane and the corresponding object in the RGB image. By integrating the depth data coming from the camera, robot can figure out not only the size but also the distance of the object candidates and use the similarity of this distance between different frames with existing coherence methods to generate more accurate object candidates. This method could be very helpful especially in locating colorful single objects or overlapping objects with similar colors. In the present configuration, Microsoft Kinect is used as an input device. Therefore, both RGB and depth data of the images are recorded. Also, kinect records point cloud versions of the frames as well. In order to achieve RGB-D segmentation, different methods can be used. Segmentation can be done by either using RGB image and depth image corresponding to that specific frame or by using colored point clouds. Normal point clouds can be thought as datasets that represents objects in a scene. They represent X,Y and Z coordinates of objects with respect to an underlaying sampled surface. They are generally produced by laser scanners or LIDARs. However, point matching is a problem to overcome in point cloud data. It is the task of finding correspondences between two arbitrary sets of points. Namely, it is the process that makes the data coming from an object from different angles corresponds to the same object. [7] Generally, drivers that are used within the kinect already handles this issue and gives matched point clouds. In the case of colored point clouds, each point in the data will also have RGB values assigned to it. Also, in order to use point cloud data effective, the data should also be converted into a texture mesh.

Another method is using graphs instead of points. In this method, pixels in frames are grouped together and treated as masks. First and simpler way of doing this is, storing 2 different images (RGB and depth image) which are taken at the same time t, and after segmenting the RGB image part, adding the corresponding depth values of the pixels to segmented image. By doing that, we can also take the depth into account while checking coherence of different frames. Although this does not change the way of segmentation, we believe that checking corresponding depth values in coherence will improve the performance of object detection. Second way is instead of doing segmentation only on RGB image or depth image, creating a new image with combination of both and using segmentation algorithm on this new combined image. This way, segmentation is done considering vectors with 4(or more) elements, namely R,G,B and depth values of the pixels, instead of adding depth values to segmented images. One can find the difference between RGB and RGBD segmentation in the Fig. 2. However, this method is a complex process and it needs substantial amount of computing power, since it is needed to pre-process depth and color image, also post process segmented image to achieve better performance in object detection. [4, 5, 6]

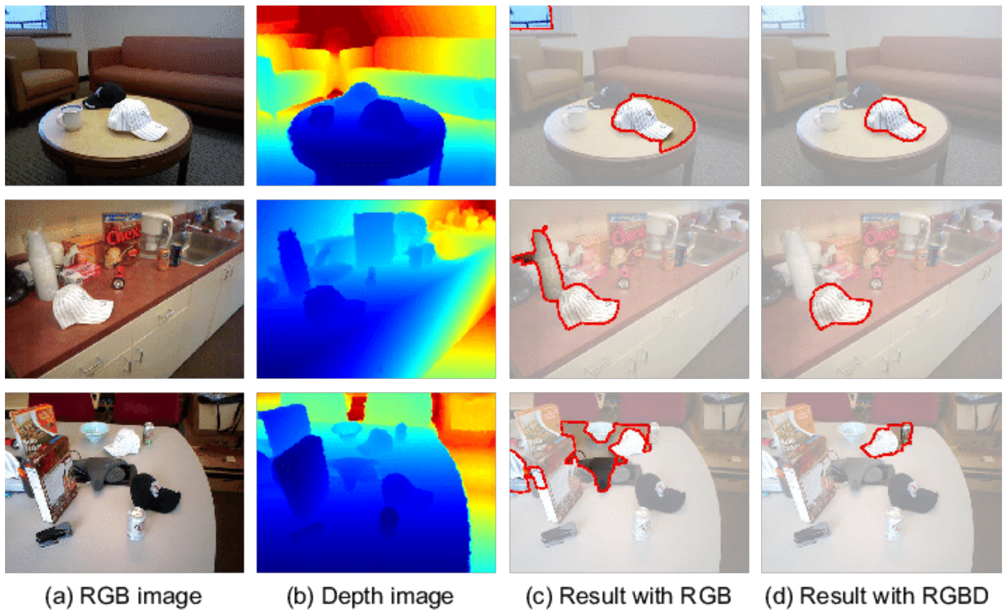


Figure 2: ”Co-segmentation with RGB vs. RGBD images. (a) A set of RGB images which all contain a white cap in common. (b) The corresponding depth maps. (c) Co-segmentation results based on RGB images, which exhibit errors due to similarly colored background objects (second row) or illumination change (third row). (d) Co-segmentation results with RGBD images. Our use of depth cues notably improves co-segmentation quality. ” [9].

- Improving the learning part through considering the temporal nature of the generated object candidates. The learning of objects' categories is currently based on an approach as shown in Fig. 1. Here, a long-term object categories' memory(LOCM) stores learned object categories in a hierarchical manner. The structure of LOCM is free and evolve when needed with visual data coming from robots movement and camera. There is also a working object candidates memory (WOCM) that accumulates object candidates which have not been categorised. Then, clusters of those object candidates in the WOCM are determined and leveraged unsupervisingly in order to update categories of LOCM. That way, robot will be able to update its LOCM throughout its operation. Note that, WOCM acts like a temporary buffer for accumulated uncategorised object candidates. Throughout the operation, filling up of WOCM is waited, and after that, new object categories are detected and LOCM is updated.

- Constructing complete objects' graphs: This will require the tracking the objects' graph as the robot moves to different locations and then consolidating those from different locations. Since an object graph is a view of an object system at a particular point in time. Throughout robot's movement, instantaneous objects' graphs will change. As such, it will be possible to use a tracking scheme to generate a complete objects' graph that encodes the objects and their relative spatial positions with respect to the space that the robot has navigated through.

4 Summary of Work Completed

Currently, the progress is behind the schedule planned. The plan was to complete RGB-D segmentation process in this semester.
Since this project is the continuation and an improvement of the proposed approach, it is very crucial to completely understand the methodology and the processes behind the proposed approach. Thus, a user manual document is created in order to clear ambiguities about projects work flow.
As mentioned above, current approach consists of a Turtlebot that has a Kinect camera mounted on it. The camera is mounted by using a motor which is controlled by an Arduino. This way, camera angle and position can be changed to a desired position. Robot captures the view via Kinect, then it segments the frames by using the RGB data sent from it. The aim is to do unsupervised learning of objects' categories and creating an objects' graph by using segments and their coherence in the frame. The Robot uses ROS as the operating system. ROS is an open-source, meta-operating system created for controlling robots. It allows the communication of different nodes (processes) with each other. The Robot has a brain ROS hydro installed. After making the required connections, one should expect a diagram like below.

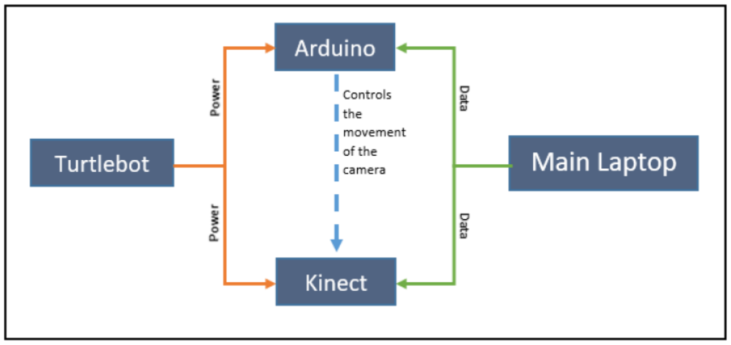


Figure 3: Diagram of Attentive Robot Configuration

After establishing the connections, the user should follow the procedure below.

- First of all, the user needs to enter command "roscore". This command needs to be used before entering any other command. It composes of nodes and programs that are pre-requisites of a ROS based system.

- After that, the user needs to initialize the communication between Arduino and the laptop. In order to do this, user needs to open up a new terminal while roscore is running on the first one. Then, he/she needs to type "rosrun apes_arduino apes_arduino_node" after this command, if the previous connections are made, the robot should connect to Arduino with no problem.
  "rosrun" allows you to run an executable in an arbitrary package from anywhere without having to give its full path or cd/roscd there first.

- After this command, the user should run the command "rosrun apes_head apes_head_node". This command will establish the connection between the camera control motor and the robot.

- Then, the user should open up a new terminal, and run the command "rosrun attentive_robot attentive_robot_node". This command is the main line. It controls the robot's camera and moves it according to the Gaussian surface created. Segmentation process also occurs here. Objects are detected with this command. In other words, this command starts the process while other ones before and after this command, establishes the connections.

- Finally, the command "roslaunch freenect_launch freenect.launch" should be entered. This command initializes the Kinect and starts image recording.
  "roslaunch" is an important tool that manages the start and stop of ROS nodes. It takes one or more .launch files as arguments.

With this summary of the document, the user should be able to start the Robot and compelete object detection process.

One of the main objectives of the proposed approach is to achieve all steps (generation of object candidates, creating object categories and an objects' map) in one place. To achieve this, MATLAB part of the approach is being recomposed for suitable use in ROS environment. ROS is an operating system that makes use of the communication between different nodes. As mentioned in the ROS website: "Nodes are processes that perform computation. ROS is designed to be modular at a fine-grained scale; a robot control system usually comprises many nodes. The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services." Nodes communicate with each other with messages. Those messages are routed via publish/subscribe semantic based transport system. A node have to publish the message into a topic in order to send it to another node. Same way, other node has to subscribe to the appropriate topic to get the message.

The publisher/subscriber diagram in the current approach is shown below. In this version, generation of the object candidates is implemented as a whole package, this package will be split into 2 parts: segmentation and object candidate generation, which is shown in other diagram. Moreover, implementation of learning part will also be a separate package, which subscribes to generation of object candidates part, and will download/upload data from/to WOCM and LOCM database.

RGB-D segmentation methodology is being implemented. Currently, 2 different methods are focused on. The first method is done by using colored point cloud data. The algorithm is based on region-growing segmentation. The main purpose of the algorithm is to merge the points that are

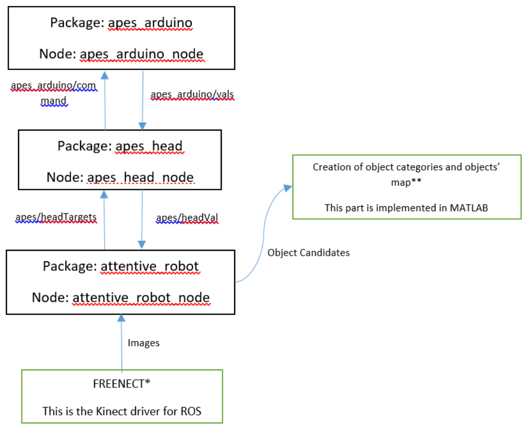


Figure 4: Current ROS Architecture

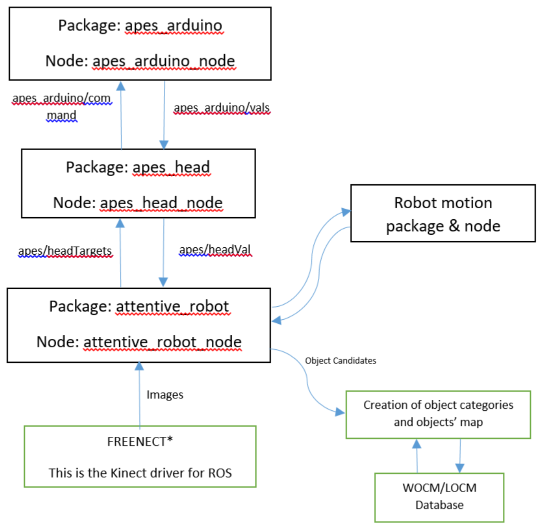


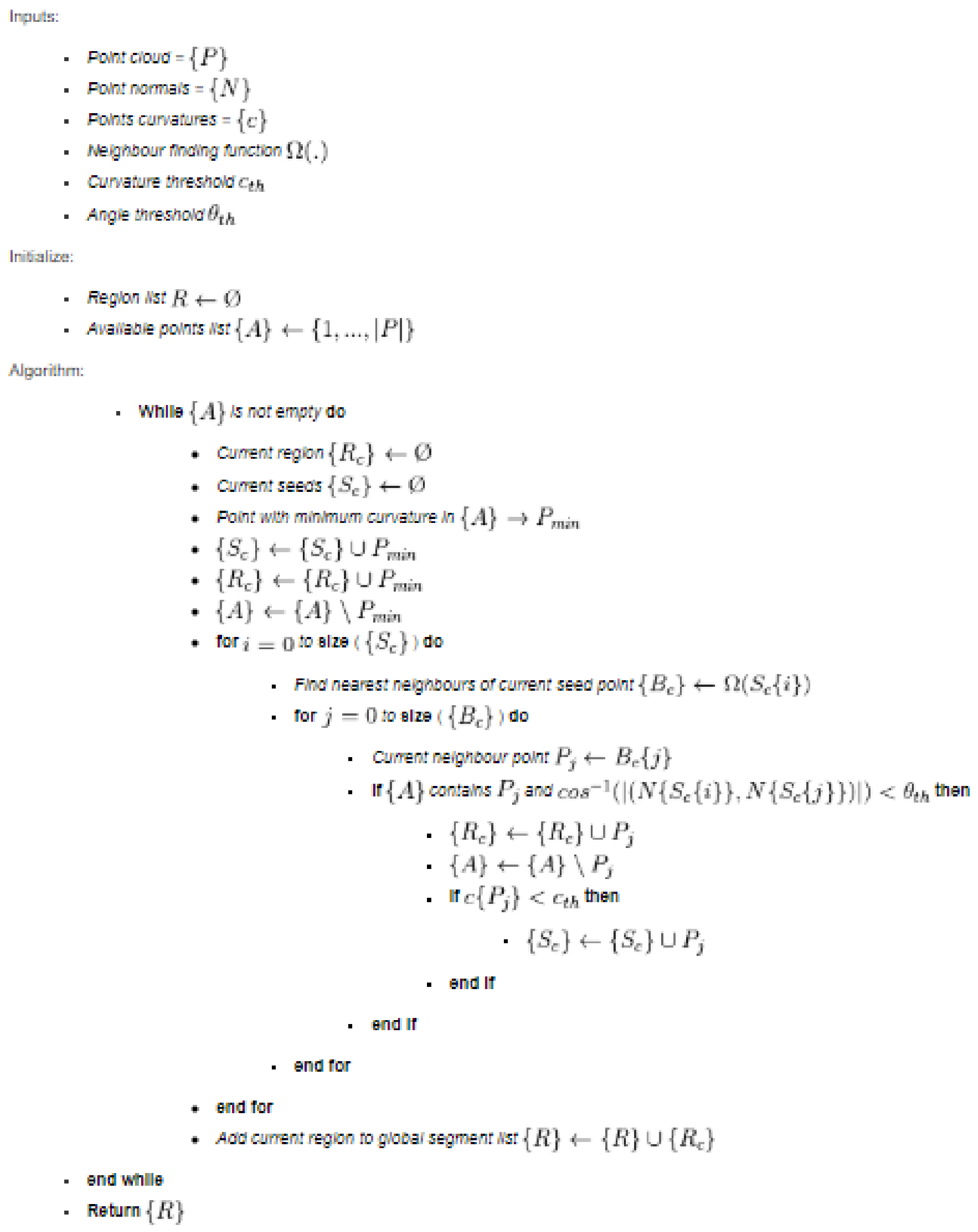
Figure 5: Planned ROS Architecture

close to each other in terms of the smoothness constraint. Therefore, the output of the algorithm is a set of clusters where points in the same cluster are considered in the same surface. Then the angles between point normals are compared.

Since the region growth begins with the point which has the minimum curveture value, the points are needed to be sorted. Because, the point with the minimum curvature is located in the flat area (growth from the flattest area allows to reduce the total number of segments). After sorting the point cloud, algorithm picks up the point with minimum curvature value and starts the growth of the region until there are not any unlabeled points. The procedure for this is as follows;

- The picked point is added to the set called seeds.
- For every seed point algorithm finds neighbouring points.
  - Every neighbour is tested for the angle between its normal and normal of the current seed point. If the angle is less than threshold value then current point is added to the current region.
  - After that every neighbour is tested for the curvature value. If the curvature is less than threshold value then this point is added to the seeds.
  - Current seed is removed from the seeds.
- If the seeds set becomes empty this means that the algorithm has grown the region and the process is repeated from the beginning. Pseudocode of this algorithm can be found below.

In the case of color based version of this algorithm, there are 2 main differences. Firstly, it uses color information instead of normals. Secondly, it uses the merging algorithm for over- and under-segmentation control. After the segmentation, clusters with close colors is merged. Two neighbouring clusters with a small difference between average color are merged together. Then the second merging step takes place. During this step every single cluster is verified by the number of points that it contains. If this number is less than the user-defined value than current cluster is merged



```
Inputs:
    Point cloud = $\{P\}$
    Point normals = $\{N\}$
    Points curvatures = $\{c\}$
    Neighbour finding function $\Omega(.)$
    Curvature threshold $c_{th}$
    Angle threshold $\theta_{th}$
Initialize:
    Region list $R \leftarrow \emptyset$
    Available points list $\{A\} \leftarrow \{1, ..., |P|\}$
Algorithm:
        While $\{A\}$ is not empty do
            Current region $\{R_c\} \leftarrow \emptyset$
            Current seeds $\{S_c\} \leftarrow \emptyset$
            Point with minimum curvature in $\{A\} \rightarrow P_{min}$
            $\{S_c\} \leftarrow \{S_c\} \cup P_{min}$
            $\{R_c\} \leftarrow \{R_c\} \cup P_{min}$
            $\{A\} \leftarrow \{A\} \setminus P_{min}$
            for $i = 0$ to size ( $\{S_c\}$ ) do
                Find nearest neighbours of current seed point $\{B_c\} \leftarrow \Omega(S_c\{i\})$
                for $j = 0$ to size ( $\{B_c\}$ ) do
                    Current neighbour point $P_j \leftarrow B_c\{j\}$
                    If $\{A\}$ contains $P_j$ and $cos^{-1}(|(N\{S_c\{i\}\}, N\{S_c\{j\}\})|) < \theta_{th}$ then
                        $\{R_c\} \leftarrow \{R_c\} \cup P_j$
                        $\{A\} \leftarrow \{A\} \setminus P_j$
                        If $c\{P_j\} < c_{th}$ then
                            $\{S_c\} \leftarrow \{S_c\} \cup P_j$
                        end if
                    end if
                end for
            end for
            Add current region to global segment list $\{R\} \leftarrow \{R\} \cup \{R_c\}$
    end while
    Return $\{R\}$
```

Figure 6: Pseudocode of region growing algorithm

with the closest neighbouring cluster.

As mentioned above, 2nd approach employs graph approach instead of point cloude data. In the algorithms that implement this, color image and depth image is taken separately [4] . The segmentation begins with determining edges, verticies; and dividing the image to subgraphs. Each edge has a weight which represents the similarity between connecting verticies. This weight is determined by the RGB values of the verticies. After that, a smoothing is applied on the depth image in order to reduce noise and quantization errors. This smoothing amount changes by the depth of the pixel. This is done by another algorithm which analyzes pixels' distance to object's boundary. Ensuring that smoothing is done more on the points which are far from camera and not done on the areas that are close to object boundaries[5]. Also, a saliency map is created from the color image to obtain high quality features[6]. In this algorithm, a single parameter is used in computations of all filter windows on a single integral image. Finally, before combining 2 images, a process for removing texture edges is used in order to eliminate edges caused by strong textures by calculating the difference in depth of two pixels on opposite sides of an edge pixel. After these preprocessings, the outputs of the algorithms are used in segmentation with a weight function. This function, changes the dominant factor in segmentation throughout an image by considering color, depth and saliency differences. [4]

The basic method on the other hand, is not changing the existing segmentation algorithm such that after segmentation process, the depth data of the pixels should be added to segmented image. Since depth and color image are taken separately but at the same time, one should store the depth image taken at time t, time when the color image is taken as well. That way, data about depth of pixels is not lost. After that, the rest is just resizing a vector -from 3x1 to 4(or more)x1 for each pixel-.

Currently, the first 2 methods discussed above are implemented in C++ and will be tested with robot@home dataset. This data set composes of raw and processed data from five domestic settings compiled by a mobile robot equipped with 4 RGB-D cameras and a 2D laser scanner. The main purpose of it is to create a test environment for segmentation and mapping algorithms. [8] Both of the algorithms will be tested on the same frames and the results will be compared.

5 Future Work

Note that, the current user-manual document only covers the proposed approach, i.e. only the movement of the camera is considered. However, in the current project, the robot itself will be moving as well. That is why, "Kobuki" package will be added and its communication with other nodes will be established as well. It is a package for Kobuki, Yujin Robot's mobile research base. By using this package, the user will be able to move the robot to a desired position, either teleoperated or autonomously.

Currently, the segmentation methods are written in C++, but not available to use in a ROS environment (because of publisher/subscriber architecture). Although they are testable in computer with pretaken images, their communication with kinect drivers and usage of the images coming from the camera should be implemented.

After comparing and deciding on the final segmentation method, robot motion part will be implemented. So that, segmentation performance will be tested while moving. This time, it will be a video, instead of stationary images and therefore, the usage of coherence will not be enough to determine object candidates. In the current approach, the images are taken while the robot is stationary, the objects position with respect to robot is not changing. However, in the updated model, the images will be taken at different positions and angles.

Apart from them, the table below will be followed.

Topic
Implementing robots motion code
Implementing image processing methodology while moving
Implementing LOCM methodology while moving
Bug testing and Presentation preparation

Table 1: Project schedule for semester-2.

6 Conclusion

6.1 Social, Environmental and Economical Impact

The importance of robotic systems in sensation and perception cannot be ignored. When talking about perception and understanding the surroundings, object detection and categorisation is a crucial point, and a good object detection is possible with good image segmentation and implementation of those segments. With this project, the aim is to create a more reliable way of object detection by issuing a strong object categorisation algorithm. In a world, where robots are getting more proactive in easing people' lives, we believe that, this project will play an important role.

6.2 Cost Analysis

Since this project is mainly a software project. The cost of it will not be high. However, the system consists of a turtlebot and a computer, which acts as the brain of the system, with a solid computational power.

References

[1] Kostavelis, I. and A. Gasteratos, “Semantic mapping for mobile robotics tasks: A survey,” Robotics and Autonomous Systems, vol. 66, pp. 86 – 103, 2015.

[2] Patar, D. and H. I. Bozma. ”Sahnelerin Keşfedilerek Nesne Adaylarının Belirlenmesi.” TORK 2018.

[3] Patar, D. and H.I. Bozma. ”Life-Long Unsupervised Learning of Visual Objects’ Categories”. (submitted to) IEEE Int’l Conf. on Robotics and Automation, 2020.

[4] G. Toscana, S. Rosa, and B. Bona, “Fast Graph-Based Object Segmentation for RGB-D Images,” Proceedings of SAI Intelligent Systems Conference (IntelliSys) 2016 Lecture Notes in Networks and Systems, pp. 42–58, 2017.

[5] S. Holzer, R.B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, pages 2684–2689, Oct 2012. doi: 10.1109/IROS. 2012.6385999.

[6] Sebastian Montabone and Alvaro Soto. Human detection using a mobile platform and novel features derived from a visual saliency mechanism. Image and Vision Computing, 28(3):391–402, 2010.

[7] Denton, Jason and Beveridge, J. Ross. An Algorithm for Projective Point Matching in the Presence of Spurious Points. Pattern Recognition, 40:586-595, 2007.

[8] Ruiz-Sarmiento, J. R. and Galindo, Cipriano and González-Jiménez, Javier. Robot@Home, a Robotic Dataset for Semantic Mapping of Home Environments. International Journal of Robotics Research, 2017.

[9] Fu, Huazhu & Xu, Dong & Lin, Stephen & Liu, Jiang. (2015). Object-based RGBD image co-segmentation with mutex constraint. 4428-4436. 10.1109/CVPR.2015.7299072.

A Reference Manual for Attentive Robot

A.1 Introduction

Attentive robot is a Turtlebot that has a Kinect camera mounted on it. The camera is mounted by using a motor which is controlled by an Arduino. Both Arduino and Kinect draws power from the Turtlebot (Arduino 5V DC, Kinect 12V DC). Both of them are connected to power board mounted on the Turtlebot. This way, camera angle and position can be changed to a desired position. Robot captures the view via Kinect, then it segments the frames by using the RGB data sent from it. The aim is to do unsupervised learning of objects' categories and creating an objects' graph by using segments and their coherence in the frames. In this document, how to start up the robot and how to bring it to working configuration will be explained.

A.2 Background Information & Configuration of the Robot

Before explaining the configuration of the robot in detail, following terms should be explained.

A.2.1 ROS (Robot Operating System)

ROS is an open-source, meta-operating system created for controlling robots. A meta-operating system is a system which is built on top of an existing operating system (generally Ubuntu in the case of ROS). It allows the communication of different nodes (processes) with each other. ROS provides different services such as, control of low-level devices, message passing between processes, and package management. Basically, it acts as the brain of the robot.

A.2.2 Kinect

Kinect is essentially a camera endowed with a depth sensor. Normal cameras convert real world scene into an image by creating an array of 2D, telling which colors are present at what XY coordinates. Basically, converting 3D to 2D. However, during this process, information about depth data, i.e. along z axis, is lost. A depth sensor comes into play at this part. It captures the data along the z-axis. Therefore, Kinect, provides us with RGB-D data, i.e. it tells the distance of the object and its distance to the background with color information.

A.2.3 Configuration of the System

Components Needed

In order to set the system, the user needs to have following ready and/or installed:

- Kinect's Connection cable: In order to start communicating with camera and power it up, this cable is needed. It has 1 input, 2 outputs. Input will be coming from Kinect where one of the outputs (USB cable) will be connected to Laptop and the other will be connected to power distrib1ution board on the robot.

- USB Cable: Since robot is using Arduino (Model) to control camera rotation. This cable is needed to establish communication between the Laptop and the Arduino.

Robot Configuration

Attentive robot is controlled via an 11" laptop with ROS hydro installed in it. One can think that the Laptop is the main controller of the robot. Since the robot will use the RGB data coming from the Kinect, the camera also needs to be connected to the laptop. Note that, the pan tilt camera is connected to a motor which is controlled by an Arduino. Thus, in order to command Arduino to configure motor, it needs to be connected to robot as well. After doing those connections, the user needs to power up the robot by using the switch located in the bottom part of the Robot. Note that, turtle bot is running on a battery. If the battery of the Robot is low, it will not be powered up. In that case, the user needs to charge up the battery by plugging in the charger into the port located next to power switch.

Terminal Configuration and Commands

After establishing the connections explained in previous section, the user needs to open up terminal in the laptop.

- First of all, the user needs to enter command "roscore". This command needs to be used before entering any other command. It composes of nodes and programs that are pre-requisites of a ROS based system.
- After that, the user needs to initialize the communication between Arduino and the laptop. In order to do this, user needs to open up a new terminal while roscore is running on the first one. Then, he/she needs to type "rosrun apes_arduino apes_arduino_node" after this command, if the previous connections are made, the robot should connect to Arduino with no problem.
  "rosrun" allows you to run an executable in an arbitrary package from anywhere without having to give its full path or cd/roscd there first.
- After this command, the user should run the command "rosrun apes_head apes_head_node". This command will establish the connection between the camera control motor and the robot.
- Then, the user should open up a new terminal, and run the command "rosrun attentive_robot attentive_robot_node". This command is the main line. It controls the robot's camera and moves it according to the Gaussian surface created. Segmentation process also occurs here. Objects are detected with this command. In other words, this command starts the process while other ones before and after this command, establishes the connections.
- Finally, the command "roslaunch freenect_launch freenect.launch" should be entered. This command initializes the Kinect and starts image recording.
  "roslaunch" is an important tool that manages the start and stop of ROS nodes. It takes one or more .launch files as arguments.

Entering those 5 commands in the order explained above will make the robot work properly. Note that, the user need to open up a new terminal in order to enter the next command. Previous commands should be running.

A.3 Ending the process & Powering off the Robot

In order to end the process, the user only needs to terminate (by pressing ctrl + c) every terminal in reverse order, meaning starting with the terminal which has "roslaunch freenect_launch freenect.launch" command running, and ending with roscore terminal. After terminating each terminal, the user can close the terminals and remove the cables. Lastly, the user should power off the computer and the robot by using power switch.

A.4 Notes

At the moment, learning part of the code is implemented in MATLAB. That is why current attentive_robot command takes samples of segmented and normal images. The user should retrieve these samples, which are located in data folder of workspace, and upload them to a computer with MATLAB code in it. Feature vectors of the object candidates in the segmented images are created in MATLAB. Then, in the main file, those vectors are used and their coherences are compared. In the end, an objects' graph is obtained. In the future, learning process will also be implemented in C++ and attentive_robot code will be updated so that learning also occurs within the robot. That is why, attentive_robot code is assumed as the main file for this project.