ESLP report

Erik de Graaf, Mehdi Ghanimifard

January 2015

1 Wall-e: Introduction

For this project we have used ROS (Robot Operating System) for the management of different components. These components are the dialogue manager, the sensor of the Kinect camera, and the processing application for learning and recognition. The sensor information taken care of by the Freenect library[1]. The processing application is written my Mehdi and Erik, both authors contributed equally to the work presented in this paper and the associated programming code [2]. The dialogue manager is written by David. Due to material constraints we (Medhi and Erik) worked together on each part of the project, and it is not possible to detail who wrote which code. We were further constrained by a time limit, which has influenced some of our design decisions.

This processing application takes the information of both the colour and the depth sensors of the Kinect camera. With the use of the OpenCV libraries in python we manipulate the color image to only show objects that are 70 centimeter to 1 meter from the camera.

2 ROS

ROS (The Robot Operating System) is a system that allows for multiple parts of whatever runs a robot to be programmed well modularly, where they'll be able to communicate with each other in a non-complicated way. Each part of the system is called a node, which sets up a topic. Over this topic they can communicate with each other. Nodes subscribe to topics to get data sent over them, and publish on topics if they want to send data. Our project contains a node for freenect (kinect drivers), with a couple of publishers, a node for the dialogue manager that communicates with the main node that has both a publisher (to reply to questions and other information; 'this is a book, confidence 70') and a subscriber (to receive a question; 'what is this object?'"). Each ROS node gets its own terminal window to be started and is run separately, communicating with the core manager of ROS: roscore.

[1] Using freenect_stack in ROS: `http://wiki.ros.org/freenect_stack`

[2] Source code repository: `https://github.com/mmehdig/wall-e/`

3 Language application and dialog with robot

In early stages of project development we designed simple dialog with the robot, in 'Wizard of Oz' method.

H 'Kinect, do you know this object?'

- robot captures 1 frame, finds object in the scene (region of interest), tries the object recognition method

R 'I do not know, please tell me' —— R: 'This is . . . , am I correct?'

H 'This is a . . . ' —— H: 'You are correct' —— H: 'No, this is . . . '

- robot updates models with the positive or negative sample.

R 'Thank you'

The development of this dialog needs more implemenatation on dialog management node in ROS. Including the speech-to-text and text-to-speech capabilities for that specific node.

4 Image processing

4.1 OpenCV

OpenCV (Open Source Computer Vision) is the library we used for processing both the depth and the colour in real time that are sent through the freenect ROS node. In order to work with OpenCV in ROS we needed to use CV_bridge, which coOpenCVnverts the kinect's camera images in ROS to OpenCV compatible image formats. We have chosen to use OpenCV because it is an extensive library for vision processing. Attempts before OpenCV include trying to implement our own object detection based on depth only. All of these attempts were too CPU intensive and on top of that it was very hard to get rid of bugs. We learnt with this project that camera's are not very precise, and even when averaging multiple frames there can remain differences between sets of frames even in a static frame.

4.2 Region of interest

The first step of our processing is grabbing the depth frames, removing all the points that are NaN (Not a Number - when the vision is limited by camera induced shadows) or more than 1 meter away and setting these to 1 meter. This video feed is printed to the screen. The left over points are then mapped onto the colour frame, leaving only the points where the point seen is more than 70 centimeter and less than a meter away. We do this to minimize the area that we have to search in for an object. Because we got access to the depth camera, we do not have to rely on more common ways to minimize search area,

such as clustering. This significantly improves the recognition results, as most of the surroundings and most noise is filtered out. We did also try background removal in the normal 2d way which took a lot of computational processing, which introduced lag.

4.3 Feature extraction (Object recognition)

For feature extraction we tried multiple algorithms. The algorithms we tried were edge detection, ORB detection and SIFT feature detection. Edge detection gave poor results, ORB gave acceptable results, but we found that SIFT produced the best results. The SIFT features are calculated on this frame and the features are saved to the object database, in our case a list of tuples. When a user then asks what the shown object is, the program runs a matching algorithm (FLANN) on each of the saved SIFT features for the objects and returns the object with the highest score. FLANN has built in algorithms that can also deal with objects shown in a slightly rotated manner.