Deep Learning Asset Location & Mapping with the Turtlebot

Michael Georgiev

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

8th September 2017

Abstract

The objective of this project has been to create a system that uses the Turtlebot to autonomously map an unknown environment. While the map is being made, objects are detected and subsequently classified with their location noted in the map. The system is capable of mapping small to medium rooms with a variety of objects contained inside. It is also able to distinguish objects which are more important than others. Through experiments and evaluation the system is deemed to have successfully fufilled its objective while the modular design allows future projects to be build upon it.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic format. **Please note that you are under no obligation to sign this declaration, but doing so would help future students.**

Name: _____ Signature: _____

Acknowledgements

Thanks to Dr. Paul Siebert and Mary Nemeth.

Contents

1 Introduction **5**

1.1 Motivation 5

1.2 Objective 6

1.3 System Overview 6

1.4 Outline 6

2 Background Survey **8**

2.1 Mobile Robots 8

2.1.1 Turtlebot 2 8

2.1.2 Cameras 9

2.2 Robot Operating System (ROS) 10

2.2.1 How does ROS work? 10

2.3 SLAM (Simultaneous Localisation and Mapping) 11

2.3.1 RTABMap 13

2.4 Object Detection and Classification 14

2.4.1 Introduction 14

2.4.2 Detection in the Natural World 15

2.4.3 Extracting Features 15

2.4.4 Object Classification with Deep Learning 16

2.4.5 Regions with Convolutional Neural Networks (R-CNN) 18

2.5 Conclusion 19

3 System Design **20**

3.1 Requirements . . . 20

3.1.1 Functional Requirements . . . 21

3.1.2 Functional Requirements . . . 21

3.1.3 MOSCOW TABLE . . . 22

3.2 Design Approach . . . 22

3.2.1 Alternative Approaches . . . 22

3.3 System Architecture . . . 23

3.4 Conclusion . . . 24

4 Implementation **25**

4.1 Mapping . . . 25

4.2 Frontier Exploration . . . 25

4.3 Object Detection and Recognition . . . 26

4.3.1 Detection . . . 27

4.3.2 Classification . . . 31

4.4 Conclusion . . . 33

5 Evaluation **35**

5.1 Testing . . . 35

5.1.1 Experiments . . . 35

5.2 Conclusion . . . 40

6 Conclusion **41**

6.0.1 Future work . . . 41

6.0.2 Reflection . . . 41

A Appendix **42**

A.1 User Manual . . . 42

Chapter 1

Introduction

A system was developed to allow a Turtlebot to autonomously map an unknown environment using SLAM as well as create an inventory of items found using a deep neural network.

1.1 Motivation

Mobile robots such as the Turtlebot are becoming increasingly more affordable and in turn accessible. Developers and researchers are now able to create robotic software and easily test it on actual real-world robots instead of in simulation or in restricted labratories. At the same time commercial applications of mobile robots are beginning to become more advanced in scope and application. The self-driving car, for example is being developed by Google, Tesla and others, with the 2015 Tesla Model S featuring a semi-autonomous autopilot system that can be driven on limited access highways in some US states.[1]

However, autonomous mobile robots have been performing complex tasks particularly since the 1990s when solutions to the simultaneous localization and mapping (SLAM) problem were developed [2]. This allowed robots to create maps of unknown environments while localising themselves within them. This can be seen in the successful use of Unmanned aerial vehicles (UAVs) in combat arenas around the world but also for applications such as mine exploration or bomb disposal where rugged mobile robots create SLAM maps of mine shafts and caves which are too unsafe for humans to access. [3]

The recent advances made in graphics processing units (GPUs) has increased the speed and complexity of Deep Learning models. This has resulted in increasingly powerful applications from speech recognition to recommender systems become ubiquitous. Robotic systems are now able to utilise highly descriptive image recognition services which are improving their ability to interact with humans and their surroundings.

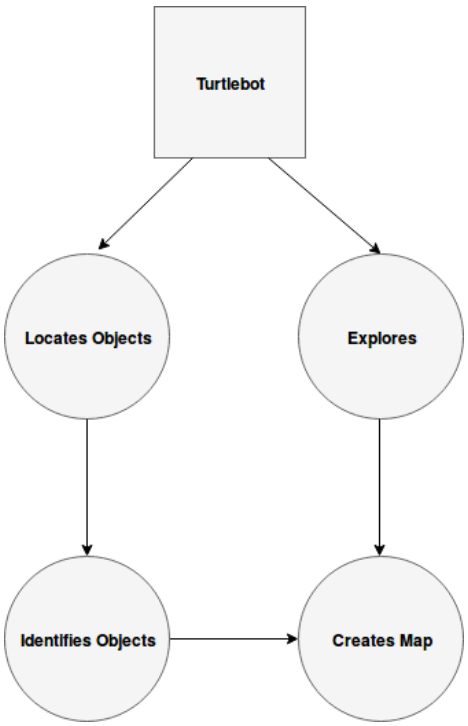
Much of the development for the Turtlebot has been on focused and isolated components which have not been combined into useful applications and tested in real-world situations. A Turtlebot that can map environments and recognise objects can be used as a basis for more advanced projects such as creating an expanding database of 3D objects and structures in interiors or even be used for navigating hazardous disaster environments to locate survivors.

1.2 Objective

The objective of this project is to build a prototypical system which combines pre-built SLAM software with frontier exploration and implements a bespoke object detection and recognition service on the Turtlebot. When placed in an unknown room the Turtlebot will autonomously create a 3D map of the environment while creating an inventory of classified and localised objects in the map.

1.3 System Overview

Figure 1.1: High-Level System Process Diagram. Describing the Turtlebots actions in an environment.



The Turtlebot autonomously explores an unknown environment. While this happens a 2D and 3D map is created and objects detected in the environment are located in this map. These objects are identified using a classifier. The project was implemented successfully.

1.4 Outline

This dissertation explains the process undertaken to create a system to solve the project objectives. A background survey was undertaken to investigate what is required to develop for robots as well as to gain an understanding of integral areas of the project such as SLAM and object detection. This is followed by an explanation of the system design while considering requirements and different approaches. A detailed report of how the design was implemented follows and an evaluation chapter

considers whether this implementation was successful by discussing a series of tests taken. Finally, a conclusion examines potential future work for the project and any exisiting limitations.

Chapter 2

Background Survey

This chapter provides research into the hardware and software required to run the Turtlebot and perform autonomous SLAM mapping. It will consider the processes of object detection and recognition by first considering them in the context of natural visual systems and then in their application in software.

2.1 Mobile Robots

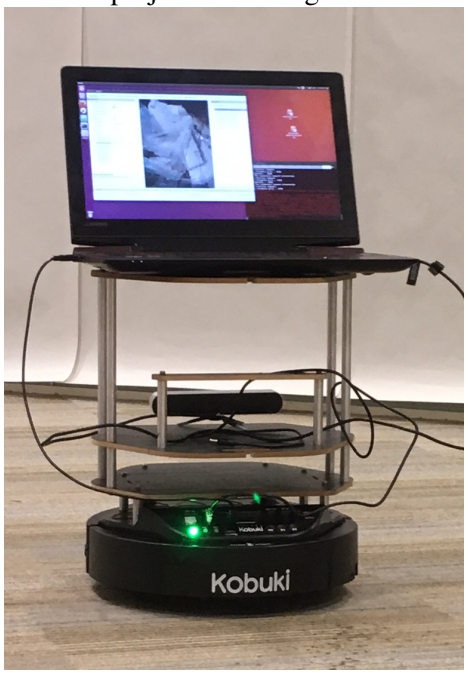
Mobile robots are robots that are not fixed to one location and are capable of moving around their environments. Movement is achieved generally through legs, wheels or tracks but aerial and nautical robots may use propellors. The robot can be controlled through manual tele-operation involving a human driver, line-following which follow visual cues such as painted lines to navigate or autonomously. These robots have many applications in areas such as the military, agriculture, rescue, transport and domestic use and include Unmanned Aerial Vehicles (UAVs or more commonly known as drones)and self-driving cars.

2.1.1 Turtlebot 2

The Turtlebot 2 is a part of a series of low cost, open-source mobile robots first developed at Willow Garage by Melonee Wise and Tully Foote in 2010. These robots are intended for educational and research purposes and provide a 'low-barrier-of-entry' platform for development with suitable hardware for SLAM and navigation[4]. The construction of this robot is very flexible featuring a modular support structure that encourages adding additional hardware but the core of the Turtlebot consists of a Kobuki base, a computer running ROS and some form of camera particularly RGB-D or stereo cameras such as the Xbox Kinect or Zed Camera.

The Kobuki base contains sensors and actuators as well as a power supply and vehicular functions. This provides a centralised unit of hardware that can be accessed via the Robot Operating System (ROS). This means that hardware processes are sufficiently abstracted as to be approachable for software developers and other high-level users without an engineering or robotics background. The base has cliff, wheel drop and bumper sensors as well as a gyroscope and highly accurate odometry.

Figure 2.1: The Turtlebot used for the project. Featuring a Zed Camera and Lenovo Ideapad Y700.



As it requires a computer running ROS to communicate with, a laptop can simply be attached to the top of the base connected by a USB cable. However, in some cases it may be more convienent to attach a small netbook or Raspberry Pi to the base and then use a separate computer to remotely communicate with the host machine. The Turtlebot has many bespoke packages for use within ROS, these include standard applications such as tele-operation or navigation and more specialised packages such as frontier exploration.

2.1.2 Cameras

For nearly all use cases, the Turtlebot requires a camera. While this can be something as simple as a webcam a greater number of applications can be created for it with some form of depth or range measurement received from RGB-D and stereo cameras. Both the discussed cameras are commonly used in ROS and for the Turtlebot.

The Kinect is a very affordable RGB-D camera developed for use on machines running Microsoft Windows, though open-source drivers exist for other operating systems. It's affordability has meant that it is frequently used by other developers whose open-source software is readily accessible. As an active camera the Kinect utilises a time-of-flight sensor to estimate the distance of points in its field of view. The Zed Camera is a stereo camera with a very user-friendly SDK and wide driver support.

Table 2.1: Comparison of the Zed Camera and Xbox Kinect v2.

Feature	Kinect	Zed Camera
Resolution	1080p at 30fps	1080p at 30fps
Depth range	0.5 - 8m	0.5 - 20m
FOV	70°horz. and 60°vert.	110°
Power	12V Adapter	5V USB

2.2 Robot Operating System (ROS)

ROS is a meta-operating system which provides a collection of tools and conventions to aid the writing of robot software. This includes an abstraction of hardware and communications as well as many tools for tasks such as message passing and package management[5].

ROS features open-source licenses and a large growing collection of packages which contribute to a vibrant ecosystem of developers and researchers working on applications. Some of these packages provide frameworks and algorithms for areas such as SLAM and kinematics as well as useful visual and modelling tools. These include:

- **RViz** - a visualisation tool which provides 3D visualisations of sensor data and robot states, such as cameras, lasers and joint states.
- **Gazebo** - a modelling tool which provides a simulated environment to build and test applications in ROS complete with robot sensors and accurate physics.

The software engineering principles of loose coupling and abstraction are encouraged throughout design which simplifies software integration and reuse as well as meaning that ROS is generally independent of hardware specifics. ROS features standard implementations in popular programming languages such as Python and C++ which has further increased the community of users.

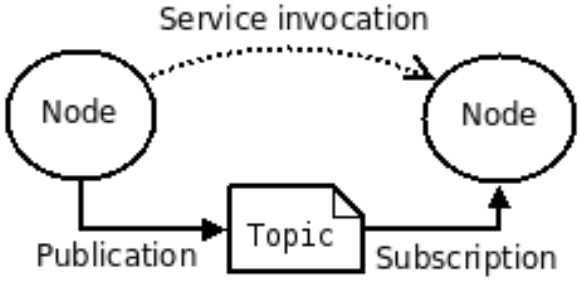
2.2.1 How does ROS work?

ROS consists of some key components within the communications infrastructure.

- **Nodes** are modular processes performing computation which can communicate with each other using topics or services. A system may consist of many nodes which perform many different tasks.
- **Topics** are named buses which provide a clear message passing interface between nodes. This is achieved through an anonymous publish/subscribe mechanism that allows many-to-many transport.
- **Messages** are a data structure which are published by nodes to topics. They support strongly typed fields such as primitives or arrays and are either predefined or user-defined.
- The ROS core or **Master** provides a centralised node which locates and negotiates communications between nodes as well providing naming and registration services.

- **Services** forgo the many-to-many paradigm and provide a request/reply interaction via servers and clients.

Figure 2.2: Illustration of general ROS communication concepts



For example a simple robot vehicle system can consist of three nodes. The first node controls the robot's ultrasonic range finder and is publishing a stream of range messages taken from the sensor along the `sensor/sonar` topic.

The second node controls the robot's movement hardware and is publishing the robot's orientation on the `geometry_msgs/pose` topic. It also contains an actionlib server which responses to requests to change the robot's positioning.

The third node controls the robot's movement and is subscribed to the `sensor/sonar` topic and contains a client of the actionlib server. While processing range messages, if a range is detected which indicates that the robot is very close to something else the actionlib service can be used to request an adjustment in the orientation of the robot thus ensuring the robot will avoid collisions.

2.3 SLAM (Simultaneous Localisation and Mapping)

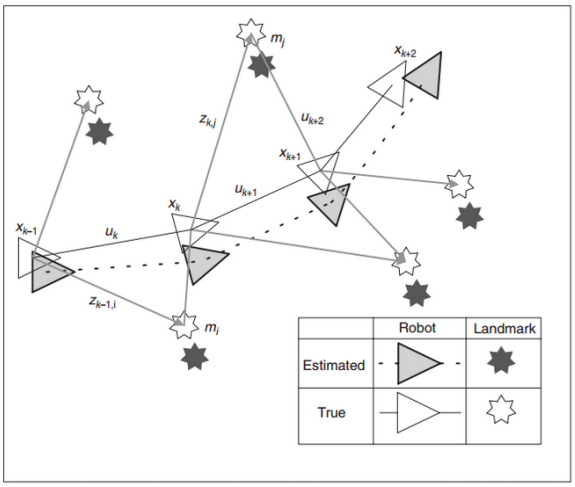
SLAM is the problem of simultaneously localising (finding the pose and orientation) of a camera within it's surroundings at the same time as mapping the structure of the environment. This requires a robot or camera with the ability to produce odometry readings as well as a camera with a range measurement device.

SLAM forms the basis of navigation in most mobile robots, meaning unknown environments can be explored and mapped without the need for imprecise technology such as GPS. It has applications in a range of manned and autonomous robots. This includes UAVs, underwater robots and domestic robots such automatic lawnmowers.

SLAM is also a key component in the development of self-driving cars These cars are driven along routes while performing SLAM capturing location, feature and obstacle data. Once the map is completed it is processed and the cars are driven autonomously along these routes updating the map as necessary [6].

This is considered to be a solved problem in Computer Science and their are many different approaches for finding a solution [2]. But at a high level SLAM is solved by using the environment to update the pose of the robot. Using odometry as the sole measurement of localisation has an

Figure 2.3: Illustration of the SLAM problem.



element of uncertainity due to extraneous factors such as wheels slipping on different environments meaning that a stated distance given by an odometry reading may be over or under estimated.

Therefore laser scans or other forms of depth readings are used to correct the robot's position by extracting features from the surrounding environment. These are called *landmarks* and can be extracted by various methods such as Random Sampling Consensus (RANSAC) and provide a growing map of the environment. Recognising previously visited landmarks is process known as loop closure detection where matches are found between new observations and regions of the map determined by the uncertainty associated with the robots position[7].

Most SLAM solutions are probabilistic and for example, can use a Kalman Filter to track the uncertainity of the robot within the map using erroneous range observations and robot controls such as odometry over time.

Defined formally as:

$$P\left(x, m | z_{1:t}, u_{1:t}\right) \tag{2.1}$$

Where given variables are:
Robot controls

$$u_{1:t} = \{u_1, u_2, u_3 ... u_t\} \tag{2.2}$$

Observations

$$z_{1:t} = \{z_1, z_2, z_3 ... z_t\} \tag{2.3}$$

Where unknown variables are:
Map of the Environment

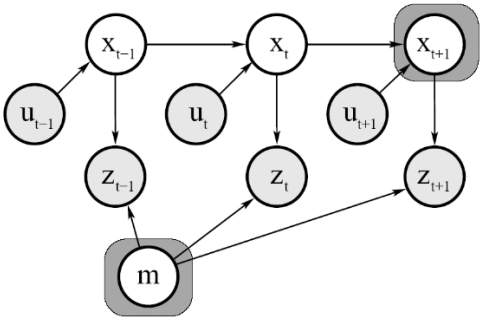
$$m \tag{2.4}$$

Path of the robot

$$x_{0:t} = \{x_1, x_2, x_3 ... x_t\} \tag{2.5}$$

SLAM has been considered a 'chicken and egg' problem [8] because these variables cannot be fully decoupled as can be observed in Figure 2.4. For example a map is required for localising landmarks which requires pose estimates.

Figure 2.4: Graph illustrating the dependencies between SLAM variables.

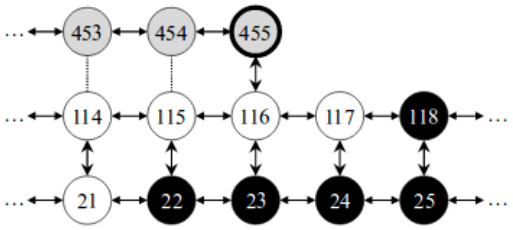


2.3.1 RTABMap

ROS features many different applications of SLAM including a popular wrapper for OpenSlam's GMapping which produces a 2D occupational grid map.[9] Another approach is RTABMap (Real-Time Appearance-Based Mapping) which produces a 3D point cloud map as well as a 2D occupancy grid map. It is a RGB-D Graph-Based approach based on an incremental appearance-based loop closure detector.

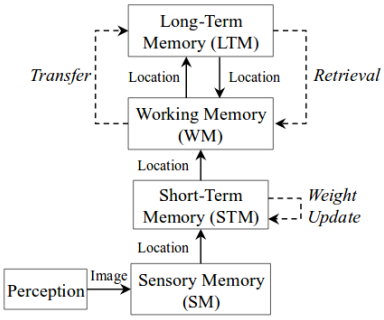
In most probablistic SLAM approaches loop closure detection compares newly found landmarks to previously visited landmarks. This takes an exponential amount of time as the time required to

Figure 2.5: Graph representation of locations. Vertical arrows are loop closure links and horizontal arrows are neighbour links. Dotted links show not detected loop closures. Black locations are those in LTM, white ones are in WM and grey ones are in STM. Node 455 is the current acquired location



process new landmarks increases with the number of visited landmarks found on the map. A delay is introduced if this process is greater than the time it takes to find a new landmark thus deteriorating the accuracy of the map produced.

Figure 2.6: RTAB-Map memory management model.



RTAB-Map uses an efficient memory management technique to minimise this problem by keeping the most frequent and recently observed landmarks in the robot's Working Memory (WM) and transfers the others into a Long-Term Memory (LTM). [7] Locations in LTM are not used for loop closure detection and thus a weighting system using a graph of landmarks is used. When landmarks need to be transfered from WM to LTM the landmark with the lowest weight is selected.

2.4 Object Detection and Classification

2.4.1 Introduction

Object detection is the task of finding the potential real-world objects within images or videos while classification involves giving these objects the correct label. Alongside robotics, such technology has applications in areas such as surveillance, medical image analysis and human computer interaction. An intelligent agent that interprets objects in scenes can be modelled on natural visual systems as many organism's vision systems have much less resolution, sensitivity and field of vision than a modern camera yet are able to perform incredibly complex tasks successfully.

2.4.2 Detection in the Natural World

For most visual systems while everything is seen for the first time our senses do not keep telling us things we already know, this is because between scenes, most important environmental information stays constant. This process is a form of sensory adaption where perception is temporarily changed when exposed to new stimuli in an attempt to normalise visual experience.[10] This happens at a retinal and neural level, where information provided by past experience have a greater say on how a scene has been interpreted than immediate information provided by external organs. For example an ellipse projected into a human retina can be interpreted as a circle due to our experiences with perspective[11].

Many species exhibit a form of pattern or feature detection to trigger neural responses to visual changes. In contrast to humans which detect generic image features such as shapes, many lower organisms utilise goal-based feature detection. For example arthropods such as honey bees readily distinguish features such as flowers in the environment that pertain to the goal of gathering food. Similarly a frog's eye is stimulated when a black disc moves in an arc rapidly within the receptor field indicating that a flying insect is near and triggers the frogs feeding response [11] thus satisfying the goal of feeding.

This is useful to the field of Computer Science because it allows us to model intelligence agents on such processes, which are typically simple and linear in that there is only one way of achieving the goal and that task assumes the agents complete engagement.

2.4.3 Extracting Features

What determines if an area of a scene is worth triggering neural activity for is different for different species. In image processing software attempts are made to find features which are unique, that can be easily tracked and can be easily compared. These features can be used for many different types of processing such as calculating a Histogram of Oriented Gradients (HOG) which can then used for object classifiers.

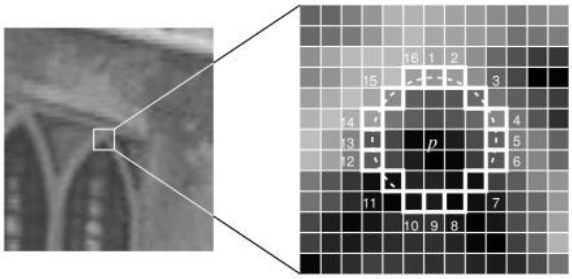
FAST Feature Detection

There are many software applications of feature detection, this includes SIFT (Scale-Invariant Feature Transform) which performs extremely well in most contexts[12]. However for applications in real-time image processing such as SLAM and Augmented Reality it is often not fast enough. FAST (Features from Accelerated Segment Test) [13][14] avoids the costly difference of Gaussians (DoG) method found in SIFT and is subsequently much faster.

Using an appropriate threshold T which is usually 12, the algorithm selects a pixel P which is the centre of a circle with a circumference of 16 pixels, n and the pixel's P intensity is I_p. The pixel P is determined to be a corner if n are all brighter than $I_p + t$ or are all darker than $I_p - t$.

If a threshold T of 12 or greater is used a high speed test is performed to reject a large number of non-corner points which involves examining the pixels at 1, 9, 5, 13. If either pixel 1 or 9 are brighter or darker than T then 5 and 13 are checked. Therefore, if three of these pixels are either all darker than the threshold or all brighter then P is determined to be a corner.

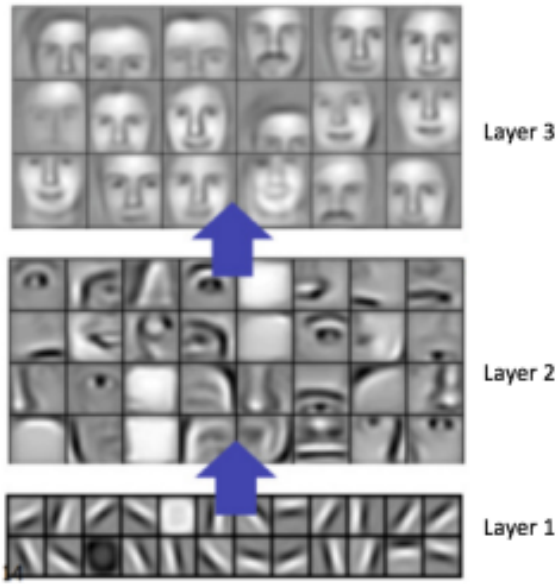
Figure 2.7: Illustration of the FAST Feature.



2.4.4 Object Classification with Deep Learning

Determining what an object is from within an image generally requires a form of classifier which utilises a training set of identified images and a validation set. One such approach is Deep Learning which is based on the way the human brain processes information and learns. Deep Learning is a form of machine learning that involves feeding data through neural networks composed of many layers. This allows data to be processed in a hierarchical way where each layer recognizes higher or more abstract features than the previous layer.

Figure 2.8: Hierarchical layers of a deep neural network.



Convolutional Neural Networks (CNN)

Convolutional Neural Networks (CNN) are a type of deep neural network which since 2012[15] has been increasingly popular and suited for object recognition and classification in images. This is namely because:

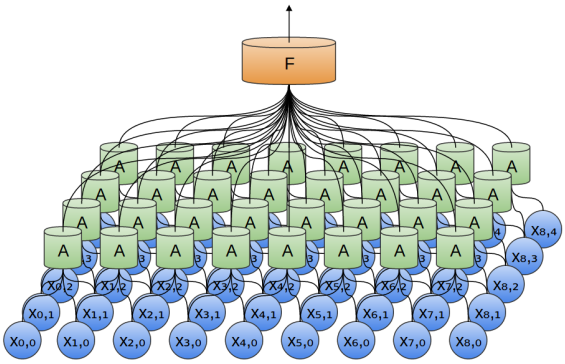
- It is rugged to distortions such as different lighting and occlusion.

- Training can be spread across several GPU's resulting in very deep networks.
- It features fewer parameters than standard neural networks meaning training time is substantially reduced.

A CNN uses many identical copies of the same neuron, meaning the network can have a large amount of neurons expressing computationally large models while keeping the parameters the neurons have to learn very small. [16]

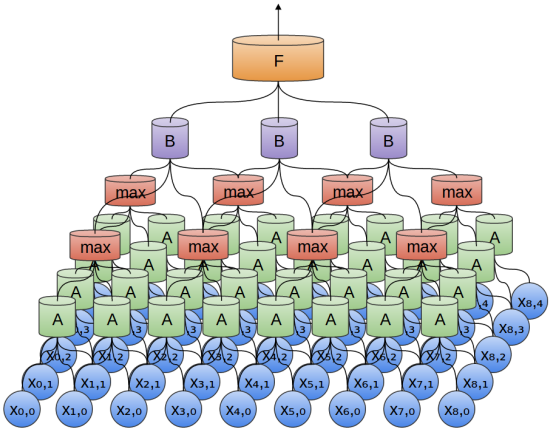
An approach called *symmetry* is used to look for properties in the data at each segment. Therefore a group of neurons, A can be introduced to a neural network that computes certain features such as the presence of an edge and then the output of this layer is fed into a fully-connected layer, F.

Figure 2.9: The convolutional layer is represented by the group of neurons A.



These convolutional layers are composable, thus you can feed the output into the input of another layer detecting more abstract features. A max-pooling layer is often used which takes the maximum features over segments of a previous layer, this allows further convolutional layers to work on larger sections of the data, this works because it is not wholly useful to know the exact pixel position of an edge but it is enough to know its location within a few pixels.

Figure 2.10: Featuring a max-pooling layer.



At every convolution an element-wise operation called Rectified Linear Unit (ReLU) is performed that replaces negative pixel values with zeros. This introduces non-linearity into the network because most real world data is linear.

MobileNet with TensorFlow

TensorFlow is an open source software library produced by Google Brain for machine learning[17]. It is frequently used due to the fact that it identifies and uses GPUs and multiple cores by default allocating 100% of GPU RAM for each process, this results in a very high performance. In TensorFlow, computations are represented as a graph, nodes are representations of operations and edges are multi-dimensional arrays called tensors (represented as numpy arrays in Python). This graph occurs within a session and is executed on the CPU or GPU.

MobileNet is a is a deep convolutional neural network for image recognition found in TensorFlow[18]. The model has been trained on the 2012 ImageNet dataset and is highly optimised for efficiency and size at the expense of accuracy. However the error rate of not providing a correct classification within the top five results is only 10.5% which for most applications is adequate. Like most pre-trained models it can be retrained to solve specific problems. This network has been used for real-time classifications on memory scarce devices and is particularly applicable to robotics.

Google Vision API

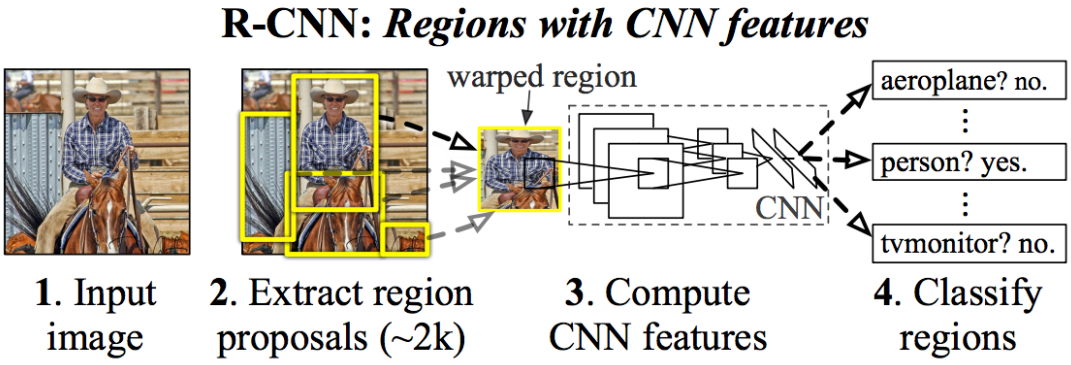
The Google Vision API is a cloud implementation of a deep convolutional neural networks trained from Google images and implemented with TensorFlow. As part of the Google Cloud platform users can make RESTful API calls from their language of choice querying images to be classified. The service provides label detection and facial detection amongst other features such as SafeSearch which filters images based on their obscene content[19].

2.4.5 Regions with Convolutional Neural Networks (R-CNN)

Detection systems utilise classifiers to evaluate potential objects. Yet there is no standard way of localising objects within an image which is often required in robotics for example to estimate the distance of such object. One such approach is called Regions with Convolutional Neural Networks (R-CNN) and is considerable more efficient than previous approaches and at the time of it's release, R-CNN had the best detection performance on the PASCAL VOC 2012 image dataset. In the paper "Rich Feature Hierarchies For Accurate Object Detection and Semantic Segmentation"[15] Girshick et al. suggest a method which involves taking an image and identifying objects using bounding boxes and then performing a classification on these areas to create a label for each object.

Roughly two thousand bounding boxes, or region proposals are created using the process of Selective Search which performs a segmentation algorithm that groups regions together by color, intensity or texture.[20] This is in contrast to using an exhaustive sliding window approach found in Deformable Parts Models.[21] Each selected region is warped to a 227 x 227 square RGB image and fed through a CNN which computes features. The final stage involves running a Support Vector Machine (SVM) on the feature vector of each region to classify and score the object within the region (if any). Greedy non-maximum suppression is used to merge the regions which share the same object resulting in accurate bounding boxes for each object.

Figure 2.11: Illustration of the stages found in R-CNN.



2.5 Conclusion

To achieve the objective of the project it was first nessessary to understand the hardware of the Turtlebot and how to communicate with it which required knowledge of ROS. The Turtlebot requires use of cameras and both the Kinect and Zed Camera have been considered, however the Kinect features the largest amount of documentation and other developers and therefore was chosen to be the default camera for this project.

Many SLAM packages exist however only RTABMap has been identified to produce 3D maps which was required to fufill the project objective. Object detection was investigated within the context of the natural world. This allowed a greater understanding of the purpose and implementation of detection systems. How to achieve these kinds of systems with computer vision was explored and two different implementations of image classifiers were identified and discussed as well as a very successful model, R-CNN. Using some of these examples of software and models helped inform the development of the object recognition software. The next chapter will discuss the requirements and system design of the project.

Chapter 3

System Design

This chapter first examines and prioritises the different requirements of the project. Then considering these requirements, this chapter identifies alternative approaches to fufill the project objective. These approaches are discussed in context of their feasibility and potential successfulness. Finally a high-level description of the system design is provided.

3.1 Requirements

The objective of this project is broad enough that it can be interpreted in many different ways. Therefore at the beginning stage of the project key system requirements were developed which help create an overaching system design and well as create a focused implementation stage. These are divided into functional and non-functional requirements. A functional requirement specifies a behaviour of the system, while a non-functional requirement describes a constraint of how the system must be developed. The defined requirements are prioritised in a MoSCoW table which states whether the requirement is Must have, Should have, Could have or Wont have

3.1.1 Functional Requirements

Requirement
Autonomously explore an unknown environment
Perform SLAM mapping
Perform autonomous SLAM mapping of an unknown environment
Locate areas of a map which may contain potential objects
Locate and map predefined objects within the environment
Locate, classify and map undefined objects within the environment
Display discovered objects in the map
Display a 3D map of the environment
Visually distinguish between predefined and undefined objects
Allow localisation of previously mapped environment
Display discovered objects from within a GUI
Perform calibration for different environments
Actively create a 3D point cloud of detected objects
Create a database of features for detected objects
Navigate to object within localised map
Save a map and detected object locations
Create a contour map of object locations

3.1.2 Functional Requirements

Requirement
Turtlebot maps a sufficiently large room with in real-time
Object detection and recognition runs concurrently to mapping
Turtlebot maps different room environments (e.g. different floor and wall types)
Produce sufficiently accurate object classification
Use ROS with Ubuntu v16.04
Use a Kinect v2 RGB-D camera
Use a ZED stereo camera
Application be able to be simulated in Gazebo

3.1.3 MOSCOW TABLE

Prority	*Requirement*	*Version*
Must have	Perform SLAM mapping	1
	Autonomously explore an unknown environment	1
	Perform autonomous SLAM mapping of an unknown environment	1
	Locate areas of a map which may contain potential objects	1
	Locate and map predefined objects within the environment	1
	Display a 3D map of the environment	1
	Perform SLAM mapping	1
Should have	Locate, classify and map undefined objects within the environment	1
	Visually distinguish between predefined and undefined objects	1
	Display discovered objects from within a GUI	1
	Allow localisation of previously mapped environment	
Could have	Perform calibration for different environments	1
	Navigate to object within localised map	
	Save a map and detected object locations	1
Would like	Create a contour map of object locations	
	Create a database of features for detected objects	
	Actively create a 3D point cloud of detected objects	

3.2 Design Approach

The project will consist of four main processes, SLAM mapping, exploration, object detection and recognition and visualisation. It will utilise a pre-built SLAM package for mapping and which will be adapted to work with the Turtlebot and the desired cameras. A pre-built frontier exploration package will also be used but will have to be adapted to work with the Turtlebot and the object detection processes. Object detection and recognition will be created from scratch for use with a map building robot and the desired cameras. Finally visualisation will be handled primarily by RViz. The stated design approach is a composite, having been informed by many different alternative approaches

3.2.1 Alternative Approaches

1. Utilise prebuilt software for all components of the system. The project would stitch them together while providing a facade for user interaction. This approach would require using a general purpose object recognition system which would not easily provide the localisation of objects in a 3D map without editing the source code. This task would involve decomposing very large and complex systems which is unnessessary. Similarly, stitching together components means the project will be highly coupled to external software which may affect futureproofing and further reuse.

2. Create a bespoke object detection package which by default does not need a map, but has functionality to locate objects within a map. This would also require incorporating external packages for SLAM etc. into the project. This approach would produce a highly reusable generic object detection service that could be used with just a RGB-D or Stereo camera.

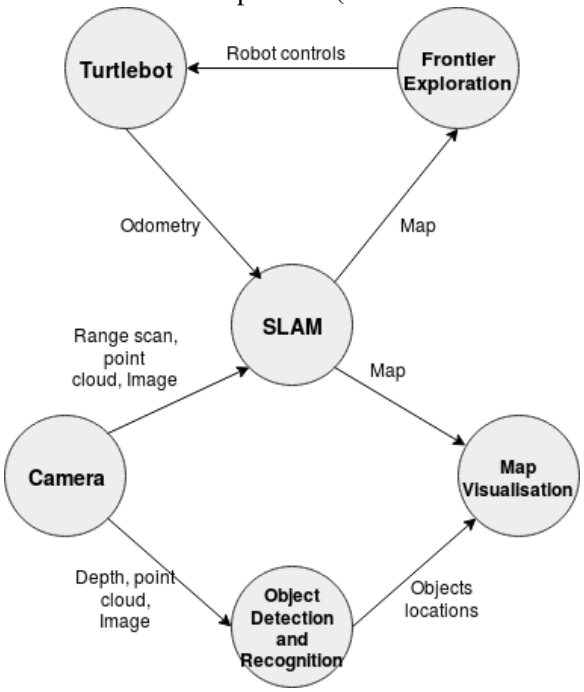
However, this is not required to achieve the project objectives and would require substantial research and time invested into developing a process of maintaining persistance when tracking objects in a 2D image.

3. Create a package from scratch which would solve simple SLAM while providing exploration and object detection. This approach would require a considerable amount of time and an advanced background in mathematics.

3.3 System Architecture

Due to there being many different ways to implement the project at a low-level either by choice of technology or algorithms etc. a high-level system architecture diagram is discussed. The system is described as a graph and this is informed by the stated approach and background research undertaken.

Figure 3.1: High-Level System Architecture. Consisting of the four main active processes of the project, as well as the passive hardware components (Turtlebot and camera).



The advantage of using this type of design is it that it maps directly to the way ROS works. A node is equivalent to a ROS node and contains a process while edges are equivalent to ROS topics. This design allows some nodes to be removed, for example frontier exploration and map visualisation and the other nodes still work as expected. This is useful because the system can developed and tested in modular stages. Similarly nodes can be replaced, for example the Turtlebot and camera can be replaced by a simulation which will provide the same edges meaning there is scope for adaption and code reuse for future projects.

3.4 Conclusion

Formulating requirements provides a clear understanding of the scope of the project. This allowed the system design to be flexible and account for an iterative process where different modules were completed and tested in stages. The high-level system design is able to capture the requirements and is sufficiently modular. The next chapter describes the implementation process of the stated design.

Chapter 4

Implementation

This chapter describes how the chosen system design was implemented. This is achieved by dividing the system into three sections; mapping, exploration and object detection and recognition. Mapping and exploration both required the use of external packages but these were still required to be interfaced with hardware and each other. The object section however, is bespoke and its development process is described in considerable detail.

4.1 Mapping

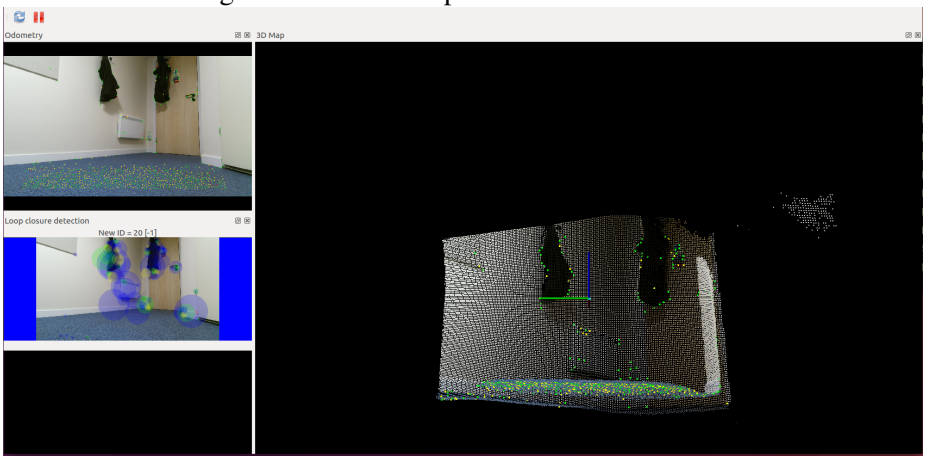
The RTABMap software was used for mapping environments with the Kinect and Zed Camera. RTABMap requires the use of odometry, RGB image, RGB camera information, depth image and a laser scan topics. Simulated visual odometry is usually provided by the RTABMap itself however for this project the Turtlebot's odometry is used which is much more accurate. The other topics are all published by camera nodes. However, each camera uses different topic names for the same thing so different remappings are required for each. The scan topic needs to created by a different node, depthimage_to_laserscan this takes a depth image and generates a 2D laser scan.

All of these nodes are combined in launch files with default arguments provided which are specific for use with the Turtlebot, for example a minimum distance is set which accounts for its manoeuvrability. This, however, allows RTABMap to be easily customized. Upon initiating the launch files the RTABMap GUI is launched as well, this provides visual notifications of loop closures which is very useful for debugging. An RViz file has also been created which allows the map and pointcloud RTABMap creates to be visualized.

4.2 Frontier Exploration

Different built in generic frontier exploration packages in ROS were tested including frontier_exploration. However, in every case they would not work with the Turtlebot. An open source package hosted on Github was instead used. [22] This package was built for usage with the Turtlebot and only required a map topic which RTABMap provides.

Figure 4.1: RTABMap Visualisation Software



It is based on the detection of frontiers within a map. A frontier is defined as is an area which is known to be neither an occupied or open area. The map topic in ROS uses the map message which consists of a 2D grid of either open, occupied or empty pixels. This is used to build an algorithm where unexplored zones are detected and ranked based on their size. The largest frontier is chosen and a path is found using an A* search. Once at the location the Turtlebot is spun in place to discover more of the map.

The package was slightly edited to slow down the speed the robot explored at. It was noticed during tests that a fast moving camera meant less features were detected by RTABMap due to a blurred image resulting in frequent losses of odometry.

4.3 Object Detection and Recognition

The purpose of the object detection and recognition package is to be able to locate potential objects within a scene, classify them and note their location within the map being built using SLAM.

This package was determined to be written from scratch as it required bespoke functionality however certain other software was consulted such as Find Object [23].Because the detection would primarily be carried out by a mobile robot certain aspects were modelled on lower organisms visual systems. Both of these pieces of research helped form the completed package.

This package was written in Python 2.7 primarily to make use of the ROS and OpenCV bindings which are required. The ROS bindings (Rospy) permits the interfacing of ROS components in Python. It allows nodes to be implemented very quickly at the expense of runtime speed, but this tradeoff was considered acceptable as this project is prototypical and can be ported to the faster C++ API in the future if required. OpenCV is a library of computer vision functions such as displaying, processing and machine learning.

4.3.1 Detection

Similar detection software was consulted and in all cases some form of blob detection was used to determine salient regions, frequently called Regions of Interest (ROI) [24]. Blob detection allows areas of a 2D image with similiar parameters such as brightness or colour to be distinguished from their surroundings. Blob detection is a sequential process and each stage is described is elaborated.

0. Initial Exploratory Blob Detection

Initially a naive method was conceived that took SIFT or FAST features as regions of interest within an image and then created a blob around clusters of these features. This method used a blob creator algorithm that determined blobs based on a closeness value in pixels to the current blob's edge.

Algorithm 1: Determine if a given feature is within the distance of a blob.

```
1  function isNearEdge (x, y);
   Input  : External feature vector x and y
   Output: Boolean
2  Where blobFeatures are current features within the blob;
3  shortestDistance ← 10000;
4  for vector in blobFeatures do
5  |  tempDistance ← euclideanDistance(x, y, vector);
6  |  if tempDistance < shortestDistance then
7  |  |  shortestDistance ← tempDistance
8  |  end
9  end
10 if shortestDistance < minimumDistance then return True ;
```

This method generally worked, however it is a very slow and costly function whereby every feature vector has to be compared against every other feature vector but moreover it fundamentally assumed that objects contained more features than their surrounding environments which is flawed.

The environment that will be encountered when the project is run is unknown but it can be assumed that objects are likely to be found in the environment. The type of environment will be variable but will typically be an office or workplace which will feature some areas of high entropy such as book shelves, exterior windows and patterned carpets. Therefore in some cases the object may have less features than it's surroundings.

1. Thresholding with Otsu's Method

Considering this, an assumption was made that potential objects will be in the foreground of an scene. This means that objects can be distinguishable even when there is a high degree of variance in the overall scene.

To achieve this the grey-scale of the image was thresholded using Otsu's method which creates a histogram of intensity and then chooses an threshold value to split the image into two classes. This method was chosen because it is very fast and assumes that an image has two kinds of pixel,

Figure 4.2: RGB Image

Figure 4.3: After thresholding with Otsu's method

foreground and background[25]. Objects in the foreground are considered to be sparce compared to the background and therefore the class with the lesser number of pixels is determined to be the foreground. In, Figure 4.3 this process correctly distinguishes the high contrast guitar in the foreground. However, the white walls distort this and constitute a great deal of the image (although there are still slightly less white pixels than black here) this could lead to the foreground being incorrectly determined.

1.5. Calibration Tool

In some cases large expanses of the same material with the same colour such as carpet or walls can be overrepresented in the mask thus inversing the foreground and the background incorrectly. To mitigate the chance of this happening a calibration tools was written. While running this program the camera can be pointed at a region of the environment that will be subtracted from subsequent image processing. This tool extracts the mean hue, saturation and luminance (HSV) values of the region which is used to subtract other areas which feature a HSV within a certain range of the mean. HSV is used as opposed to RGB or BGR image spaces because it seperates colour from luminance (intesity), this means that changes in lighting such as shadows or sunlight do not affect the detection of similar regions thus produces more accurate masks.

2. Combining with a Depth Mask

To properly seperate the foreground the Otsu method is combined with the a depth mask taken from the Kinect or Zed Camera's range finding where a depth threshold is pre-determined. This allows only locations within a certain range to be located therefore considerably improving the the clarity of the foreground. (However it is important to note the books in Figure 4.5 are just at the edge of the Kinect sensor's range and not fully rendered.) A morphological operation removeSmallObjects found in the Scikit-Image library is used to perform erosion and dilation based on a specified kernal size, this has the effect of removing small regions of spurious pixels and somewhat provides blobs with well defined edges.

3. Clustering

With the blobs sufficiently seperate from the background they can be clustered with the aim of creating segments. This allows bounding boxes to be defined around each segment forming a region

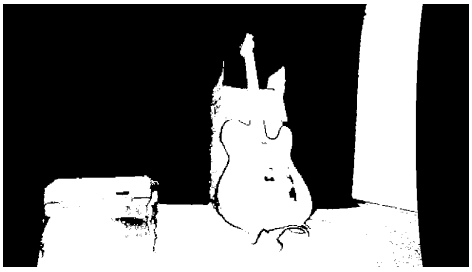


Figure 4.4: Depth image at 1.5m

Figure 4.5: Combined depth and Otsu mask

of interest.

This is similar to how R-CNN finds bounding boxes however in R-CNN the number of clusters is a known variable, usually 2000 and the Selection Search algorithm is used to find this many clusters. In this project the number of potential objects in a scene is unknown and therefore the number of clusters is also unknown, this limits the number of algorithms that can be used.

A clustering algorithm, Density-based spatial clustering of applications with noise (DBSCAN)[26] is applicable to this problem as it works with an unknown cluster number and an implementation is provided in in the Scikit-learn package. This has a best case complexity of $O(nlogn)$ which is considerably faster the inital clustering algorithm ALGORITHM 1 which has a best case of $O(n^2)$

DBSCAN has two parameters: $\varepsilon(epsilon)$ which defines the radius around a data point p and a minimum number of points that consist of a cluster $minPoints$. The algorithm can be abstractly described as thus:

Algorithm 2: High-level DBSCAN

1 Find ε neighbours of p_n and identify the core points;
2 Find connected components of core points;
3 Assign border points or otherwise outliers;
4 Where N is a neighbourhood;
5 Core points are $N_\varepsilon >= minPoints$;
6 Border points are $N_\varepsilon < minPoints$ but still reachable from a Core point;
7 Outliers are neither core or border points;

Clusters are thus created from the core and border points while outliers in can be discounted.

The algorithm was initially tested with $(x, yandthecolourimage)$ features however this resulted in a very slow runtime. Offseting this by reducing the step size and desampling the image gave poor overfitted clusters. Therefore $(x, y, depthandgrey - scaleimage)$ was used which produces satisfactory clusters.

Using a low ε value results in many clusters similar to the results found in R-CNN's Selection Search, however, R-CNN classifies every bounding box created from the the cluster. This is not feasible for this project which must be implemented in real-time. Therefore a larger ε is used which produces larger clusters that fit the blobs closer to their real world images.

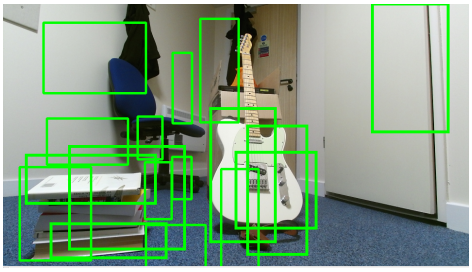
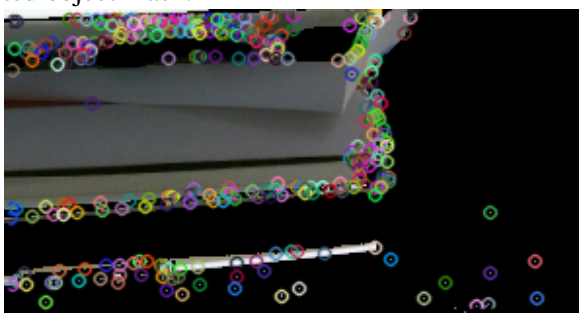


Figure 4.6: Clusters shown using a low value for ε

Figure 4.7: Clusters shown using a higher value for ε

Figure 4.8: FAST features for the stack of books. Note the features are extracted from a RGB image combined with the detected object mask.



4. Creating Bounding Boxes

Given the (x, y) points of each cluster a bounding box determined giving a region of interest. This area can then be classified with the assumption that at least part of an object is within the area. Unfortunately, as can be seen in Figure 4.7 some blobs detected are not that interesting notably the area of the door in the right side of the image. This area has very few edges or corners meaning very few noteworthy features. Therefore a minimum threshold of features can be used to determine whether a region of interest is actually interesting thus removing the part of the door. Features will have to be extracted every frame and thus FAST feature detection is used primarily because it is more speed efficient than SIFT.

These objects contained in the bounding boxes exist in a 3D plane meaning that the width and height of each box is arbitary and variable according to the scale and angle of which the object is represented as in 2D. Instead a simplistic of method surrounding the object in a circle is used. The center point of this circle is calculated as the mean point of the previously extracted features for the bounding box, this ensures that the central pixel (u, v) is very likely to be part of the object and not located in the background. The radius of the circle is calculated from the width and height of the bounding box providing a wide location in which the object can be contained.

In future iterations of the project the features (possibly using highly discriminate descriptors such as SIFT) of each salient region can be used to create a bag of visual words and be recorded with the label given by the classification process. Over time the histogram of words for each label would increase and become more developed meaning that eventually the system can learn to recognise objects using feature matching.

4.3.2 Classification

The next step after having detected salient regions within an image is to classify them. Image classification is typically achieved using a CNN which is available in cloud based APIs such as Google Vision and also in able to be trained and used locally using Tensorflow. However initially an alternative Haar Feature-based Cascade Classifier was considered before examining a robotic visual memory model was and more successful CNN based classifiers used.

Haar Cascade

A Haar Cascade is based on Viola-Jones face detection and has the advantages of being very fast and suitable for real-time applications[27] as well as having a framework built into OpenCV. The model is trained with positive images of the desired object in different environments at different scales and sizes and with negative images of anything other than the positive image. Creating the positives first required extracting the object so that the background is transparent. This process required different methods for different objects but was tested with a Coca-Cola can. OpenCV contains a script to take these transparencies and impose them at different perspectives and scales within negative background images which were taken from ImageNet[28]. The training stage was also handled by OpenCV and was trained to 10 Adaboost stages. This process took an unbelievably long time and had to be aborted after 9 stages having taken more than 16 hours despite using the maximum amount of memory. The classifier could still be tested after the completed stages and while it did detect the Coca-Cola Can with no occurence of false positives, the training process is too long particularly if a selection of objects need to be detected.

Robot Visual Memory Model

Instead a model was developed inspired by natural visual memory systems, that allowed a simple hierarchy of objects to be built from exploring an environment. This limited the need to recognise a large amount of objects while also recognising that not all regions of interest are actually important objects within the room. A frogs neural responses are triggered by goal based features where features are remembered more readily if they constitute part of or all of a goal. For example the goal of feeding is associated with black ellipsical objects indicating flies, while being carnivorous a seed pod shaped object would less likely be remembered.

This model is invoked after a salient object has been detected. Once an object is classified it is determined to either exist in the robot's long term (LTM) or short term memory (STM). The long term memory constitute of objects which have been defined. These are objects which are necessary for a goal, where the first step of achieving this goal is correctly identifing these objects. This means the objects are distinct and important such as a computer, coffee maker or a guitar. When the Turtlebot explores a room the first question asked when a region of interest has been detected is "Is this one of the goal-based objects?" If the result is affirmative then the object is remembered within it's environment perpetually in long term memory.

On the otherhand if the object is not pre-defined it is stored in short term memory where it will be forgotten after time. When a mapping and exploration session is finished and all objects have been

classified, objects which currently exist in STM are forgotton while objects in LTM are associated and assumed to exist with the completed map.

Because the project intends to map whole rooms, the manner in which STM is dumped after completion is comparable to the concept of event boundaries or the "Doorway effect" [29]. This phenomena occurs when human's leave rooms they have a tendancy to forget what they were previously doing. This exists because spatial changes such as entering a bathroom often correspond with physical and mental changes such as needing to use the bathroom and thus can readjusting our hierarchy of memories.

Implementing the Memory Model

Two different kinds of CNN classifiers are used to determine if an object will be stored in LTM or STM.

The first is the TensorFlow MobileNet CNN which is trained with a selection of objects that if found will be stored in LTM. MobileNet was chosen over other types of image classifiers because it can operate in real-time with performance testing indicating that it takes between 0.1 and 0.3 seconds to classify from six potential objects. Using it through TensorFlow simplified the training process which only took roughly 30 minutes. The training sets were chosen by bulk from Google Images. This allowed regions of interest to be scored from the defined objects, if the score was less than a confidence level (such as 0.9) then the object is considered to be important and is not stored in LTM.

The second classifier is the Google Vision API. A cloud based classifier was chosen over training one from scratch as training one to contain all the types of objects in office based environments would take too long. Similarly using a pre-trained classifier locally such as Inception [30] would potentially schew the classifications with irrelevant labels such as animals or landscapes. An image is send to the API and the response contains the most confident single label and score of the object. This compares to the IBM Watson classifier [31] which returns all labels with all the scores resulting in situations where an object can be labelled a car and a tomato and both have a score of 0.9 which is not as useful for the purposes of this project.

Another advantage of the Google Vision API was discovered through sustained use in that the labels it provides are very vague. For example a chair is labelled as furniture and bottles are labelled as products. As this classifier is determining short term memories it is enough for the robot to know what an area of an environment vaguely is as it is not concerned with the goal of finding important objects. Unfortunately the Google Vision API is slow, with classifications taking from 0.3 to 1 second depending on internet connectivity and speed. However, a buffer of requests is formed while also noting the location where the object is found meaning the robot can operate in real-time and correctly classify objects at locations in the map being built.

Adding Object Persistance

Objects need to be detected in 3D and previously classified objects need to persist meaning that the same object is identified between frames. This is achieved through the use of ROS markerArray. A marker is an (x, y, z) point which exists in a coordinate frame in this case the map frame. There

are two cases that need to be accounted for: 1) A new object is detected and 2) An existing object is detected.

If a new object is detected and classified an approximate location of this object is calculated by finding the mean feature point of the object and finding the corresponding value from the point cloud image. This is transformed into an (x, y, z) point in the map coordinate frame and is published as a marker with an associated label in ROS.

For every 2D region of interest found a test is made to see if a real-world marker exists within it's vincinity. This is calculated by assuming an object has a sector where it is highly likely this object is the sole occupant. This avoids seperate markers from being created for what is the same object.

> Considering a point $P(x, y, z)$ and a sphere with a centre $C(x_0, y_0, z_0)$ and radius r.
> P exists within C if $(x - x_0)^2 + (z - z_0)^2 + (z - z_0)^2 < r^2$

The complete algorithm for the process of classification can be can be seen in Algorithm 3. This process provides a way remember locations of discovered objects in a map and to minimize costly classifications to only when new objects are discovered.

Algorithm 3: Classify regions

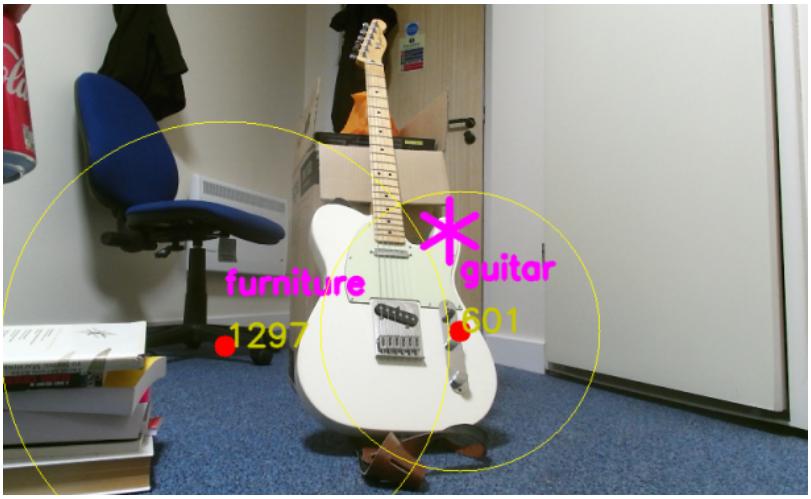
```
for region in regions do
    if region.corners > minCorners then
        existingMarker ← addNewMarker(region.meanx, region.meany);
        if notexistingMarker then
            label, score ← queryTensorflow(region);
            if score < 0.9 then
                label, score ← queryGoogleVision(region);
                If score < 0.9  continue;
            end
        end
    end
end
```

These markers are displayed in RViz as text at the relevant point in the map, STMs are coloured yellow while LTMs are coloured red. A simple GUI in OpenCV was created which displays the object detections within an camera frame. The object circumference is shown as well as the centre location, object label and number of FAST features found within the image. LTM objects are displayed with a * next to their label.

4.4 Conclusion

Implementing the system design required the use of two external packages, both of which handled complex and performance optimisation. Once these parts were implemented the object detection and recognition section was implemented in Python. Considerable experiments into different low-level methods such as using a Haar Cascade were tested, while these did not work they helped

Figure 4.9: The GUI displaying classified objects with their feature count. LTM objects are indicated with a *.



inform the subsequent successful processes developed. The final implementation of the object detection and recognition section features system which detects regions of interest, classifies them and gives persistance to the recognised objects. The next chapter records experiments and testing to evaluate the successfulness of the project.

Chapter 5

Evaluation

A testing stategy was developed which would allow the project to be tested in its entirity this involved a set of three expriments to ensure the project was successful or highlight areas for improvement.

5.1 Testing

5.1.1 Experiments

Three experiments were developed to determine whether the project objective has been achieved and to what extend, due to the fact that specific hardware was required and ongoing issues with the Turtlebot the experiments were completed in one sitting at the end of the project development period.

> **The project objective:**
> The Turtlebot will autonomously create a 3D map of an unknown room while creating an inventory of classified and localised objects in the map.

Five qualitive measures are used determine if parts of an experiment were successful, a more quantitive assessment would require more advanced software to compare completed maps and pointclouds with precise ones and is out of the scope of the project.

- **Adequate Performance** - The map is produced with no noticable delay in the robot's movements
- **Accuracy of a map** - The map produced is complete and the shape of the room is easily distinguishable
- **Accuracy of point cloud** - The point cloud does not feature any repition or gaps and objects are distinguishable.
- **Successfulness of classification** - The guitar is identified once.

- **Successfulness of object detection** - Some objects are detected and classified with vaguely relavent labels.

Experiment Setup

A small-medium enclosed room was chosen to perform testing. This allows the room to be mapped in a reasonable time due to issues pertaining to the Turtlebot battery life and temperamental connectivity between the laptop and the Kobuki and is free from human disturbances. The room was adequately lit with no natural light. A guitar is placed in the room, this has been chosen as the goal of the robot with the MobileNet having been trained to detect guitars. It is the only significant object in the room with the rest consisting of a chair, bag, bin, wardrobe, bed and desk. An Xbox Kinect v2 was used as the Turtlebot camera sensor and was connected the Lenovo Ideapad Y700 which had the RTABMap Visualisation tool and RViz open.

Experiment 1

Does the system autonomous map the environment?

Hypothesis: The Turtlebot will successfully explore the two sections of the room, the map produced will be relatively accurate but may miss areas out in the narrower part of the room.

Result A:

- Adequate Performance - n/a

- Accuracy of a map - Poor

- Accuracy of point cloud - Poor

- Successfulness of classification - n/a

- Successfulness of object detection - n/a

Process: The Turtlebot was placed 60-70 cm from the wall Figure A. The Turtlebot span less than 90 degrees and RTABMap indicated that odometry had been lost and mapping would not continue. The process was terminated.

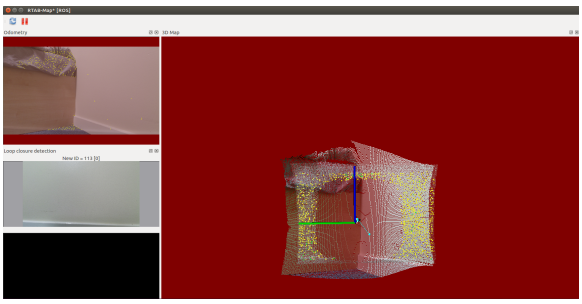
Comments: The Turtlebot was so close to the wall that the field of vision for the Kinect was entirely consumed by a white wall and no features and thus no landmarks could be extracted for SLAM to work. This indicates that the start position of the robot is very important.Unfortunately, nothing can be done to mitigate the program and thus the user must decide the start position.

Result B:

- Adequate Performance - ✓

- Accuracy of a map - Acceptable

Figure 5.1: Experiment 1: RTABMap visualisation indicating that no landmarks can be found and odometry has been lost.



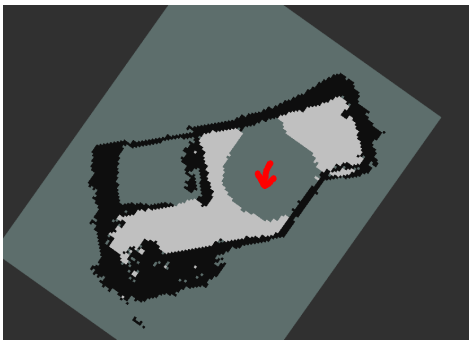


Figure 5.2: Experiment1: Red arrow indicating the start pose

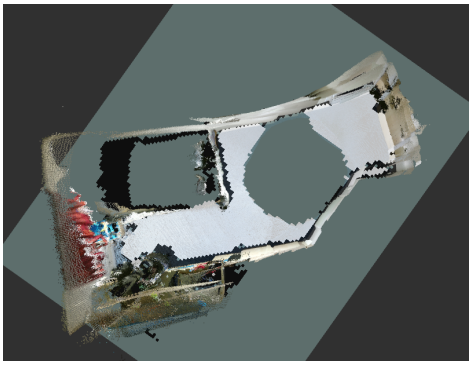


Figure 5.3: Experiment1: Completed point cloud

- Accuracy of point cloud - Good

- Successfulness of classification - n/a

- Successfulness of object detection - n/a

Process: The Turtlebot was placed in the center of the room equidistant from walls. The Turtlebot proceeded to begin explore and map the large section of the room. The narrow part of the room was visited but was not traversed the whole length. The process terminated naturally.

Comments: The Turtlebot produced an accurate point cloud of the image. The map is generally accurate but with some walls slightly distorted and a shadowy region underneath a desk producing a cluster of thick barriers in this area (Black region at bottom of map).The map is missing the circumference where the Turtlebot began as the Kinect range scanner has a minimum threshold. The frontier exploration algorithm relized leased this area was still enclosed so did not explore it. However, this could be completely solved by editing the frontier exploration source code to account for non-discovered regions within an enclosed area.

Experiment 2

Can objects be detected while mapping with manual teleoperation?

Hypothesis: The Turtlebot will successfully map the room including the start position. The guitar will successfully be detected and classified. Other objects will be detected in the room and classification should be somewhat accurate.

Results:

- Adequate Performance - ✓

- Accuracy of a map - Good

- Accuracy of point cloud - Good

- Successfulness of classification - Good

- Successfulness of object detection - Acceptable

Process: The Turtlebot was controlled using teleoperation to all areas of the room. The guitar was approached intially head on and then from the sides and was classified successfully and only once. Other regions of the room were classified.

Comments: Using teleoperation resulted in a map with more coverage than Experiment 1. However, due to non fluid angles that the Turtlebot was moving in there are some incorrect doubled barriers. The guitar is successfully located and classified. Other objects are detected but while most of these are correctly identified such as wall, others are given very vague descriptors such as blue. (Presumeably because the furniture is a blue colour). This may be because the bounding box used to enclose the object was too restrictive thus and it can only be identified as a whole

---

Experiment 3

Can objects be detected while autonomously mapping in real-time?

Hypothesis: The Turtlebot will successfully map the room but not the starting location. The guitar will successfully be detected and classified. Other objects will be detected in the room and classification should be somewhat accurate.

Results:

- Adequate Performance - ✓

- Accuracy of a map - Good

- Accuracy of point cloud - Good

- Successfulness of classification - Good

- Successfulness of object detection - Good

Figure 5.4: Experiment 2: Manual map. Yellow = STM, Red = LTM

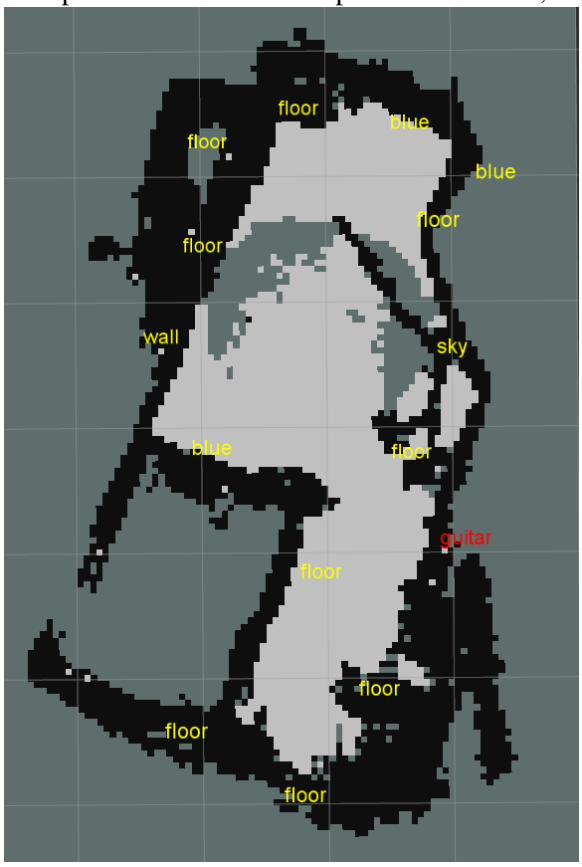
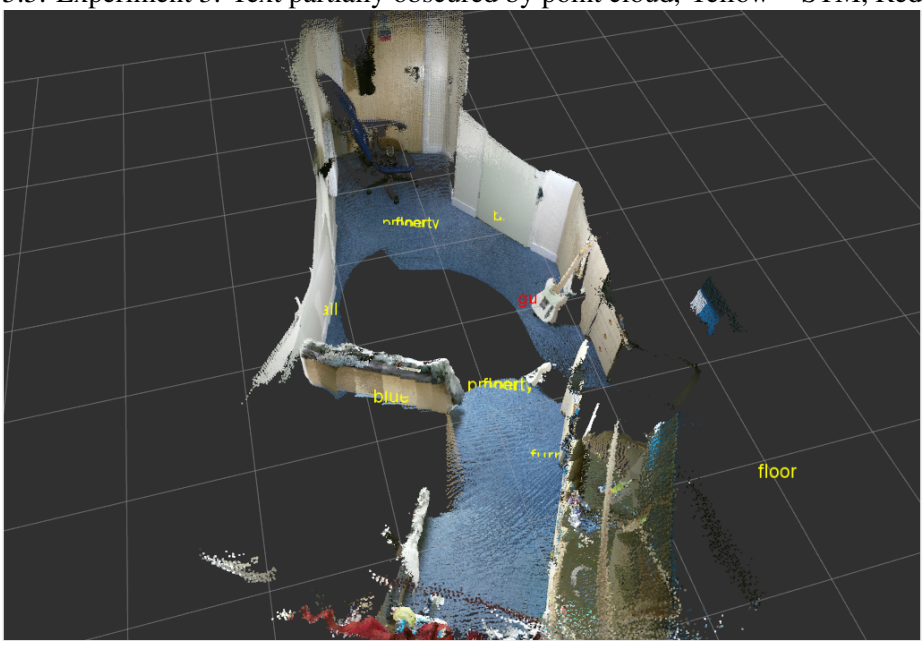


Figure 5.5: Experiment 3: Text partially obscured by point cloud, Yellow = STM, Red = LTM



Process: The Turtlebot was placed in the center of the room equidistant from walls. The Turtlebot proceeded to begin explore and map the large section of the room. The guitar was classified successfully and only once. Other regions of the room were classified.

Comments: Experiment 3 was successful it produced the most accurate map and point cloud despite missing the starting location. The guitar was located and classified correctly as well many other objects in the room such as the wardrobe and chair being given a furniture descriptor. This may be because the frontier exploration package used is very conservative with movements which means objects can be detected from a distance making them distinguishable from their background. The floor was detected outside of the map which may have been caused by the pointcloud readjusted itself during the mapping process leaving the marker at it's original location.

5.2 Conclusion

The hypothesises stated are all proved correct and thus the project objectives have been achieved, with an environment accurately mapped autonomously and the contents classified successfully. The requirements indicated for completion in the first version were all completed. Results varied across experiments and with even slight changes to the setup giving very different readings. While the individual components were tested iteratively during the implementation stage it was difficult and time consuming to do comprehensive tests of the whole project until all components were completed. The overall project could be improved by implementing solutions to some of the problems found during the testing process. This would essentially constitute a second version whereby other requirements could be completed also.

Chapter 6

Conclusion

6.0.1 Future work

During the course of development, some ideas were conceived which were outside of the scope of the project. There are also some limitations to the project. These both inform ideas to improve and add to the project.

- Incorporating active detection of objects would not be a difficult task, when an object is detected by a robot it can manoeuvre itself to build up a detailed 3D image of the object. This can be saved in a database and can form the basis for a point cloud detection service.
- Advanced map making could be achieved by adapting the current technology stack to fit the Turtlebot with a 360°field of view as well as upwards facing cameras. This would create fully rendered 3D point clouds of environments. Similarly using stereo cameras would allow the turtlebot to operate in outdoor environments.

6.0.2 Reflection

The objective of the project was to make the Turtlebot autonomously create a 3D map of an unknown room using SLAM while creating an inventory of classified and localised objects found in the map. Considering the experiments from the evaluation chapter, the project can be considered a success: the project is able to perform in real-world environments without noticable latency, maps and point clouds are created with some accuracy and object detection and classification results in correct objects being located and identified.

The project focused on quickly combining the cameras, Turtlebot and pre built software together. This task was non-trivial as much of the documentation was either out of date or misleading resulting in a considerable amount of time wasted debugging camera drivers. Similarly there were some hardware issues with the Turtlebot which only appeared near the end of the project. Developing the object detection and recognition package was a much more fufilling experience as it was essentially a green-field piece of software and therefore algorithms and designs could be developed from scratch. The developed project provides a complete navigation and mapping technology stack for the Turtlebot, providing a strong but modular foundation for any kind of future development.

Appendix A

Appendix

A.1 User Manual

Dependencies:

- For usage with the Xbox Kinect v2.: https://github.com/code-iai/iai_kinect2
- Frontier Exploration: https://github.com/bnurbekov/Turtlebot_Navigation

In a new terminal for every command. (Ensure the catkin workspace has been sourced. Usually: source /catkin_ws/devel/setup.bash)

1. roslaunch terrapin-ros kinect.launch
2. roslaunch terrapin-ros stream.py kinect2
3. rosrun final_project control.py
4. rosrun final_project mapping.py
5. roslaunch terrapin-ros map.rvizs

Bibliography

[1] Sam Abuelsamid. Tesla autopilot fatality shows why lidar and v2v will be necessary for autonomous cars. https://www.forbes.com/sites/samabuelsamid/2016/07/01/first-tesla-autopilot-fatality-demonstrates-why-lidar-and-v2v-probably-will-be-necessary/7af624982d91.

[2] Hugh Durrant-Whyte, Fellow, IEEE, and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms, 2006.

[3] Mining rox: Robotics in mining. https://miningrox.informatik.tu-freiberg.de/en/.

[4] What is turtlebot? http://www.turtlebot.com/about/. Accessed: 2017-09-5.

[5] http://www.ros.org/.

[6] https://blog.acolyer.org/2015/11/05/simultaneous-localization-and-mapping-part-i-history-of-the-slam-problem/.

[7] M. Labbe and F. Michaud. Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics*, 29(3):734–745, 2013.

[8] Hugh Durrant-Whyte. Uncertain geometry in robotics. *IEEE Trans. Robotics and Automation*, 1988.

[9] Gmapping. http://www.openslam.org/gmapping.html. Accessed: 2017-09-5.

[10] Michael A. Webster. Visual adaptation. *Annual Review of Vision Science*, 1(1):547–567, 2015.

[11] Martin A. Fischler and Oscar Firschein. *Intelligence*. Addison Wesley, 1987.

[12] K Mikolajczyk and C Schmid. A performance evaluation of local descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2005.

[13] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006.

[14] Edward Rosten, Gerhard Reitmayr, and Tom Drummond. Real-time video annotations for augmented reality. In *Advances in Visual Computing. LNCS 3840*, pages 294–302, December 2005.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2014.

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[17] https://www.tensorflow.org/.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[19] Explore the galaxy of images with cloud vision api. https://cloud.google.com/blog/big-data/2016/05/explore-the-galaxy-of-images-with-cloud-vision-api. Accessed: 2017-09-5.

[20] J. Uijlings, K. van de Sande, T. Gevers, , and A. Smeulders. Selective search for object recognition. *IJCV*, 2013.

[21] P. F. Felzenszwalb, R. B. Girshick, and D. McAllester. Discriminatively trained deformable part models, release 4. http://people.cs.uchicago.edu/ pff/latent-release4/.

[22] https://github.com/bnurbekov/.

[23] https://introlab.github.io/find-object//.

[24] Jan Hendrik Hosang, Rodrigo Benenson, and Bernt Schiele. How good are detection proposals, really? *CoRR*, abs/1406.6962, 2014.

[25] Otsu's method. https://en.wikipedia.org/wiki/Otsu Accessed: 2017-09-5.

[26] Martin Ester, Hans-Peter Kriegel, Jrg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. pages 226–231. AAAI Press, 1996.

[27] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[28] http://www.image-net.org/.

[29] Gabriel A. Radvansky, Sabine A. Krawietz, , and Andrea K. Tamplin. Walking through doorways causes forgetting: Further explorations. *THE QUARTERLY JOURNAL OF EXPERIMENTAL PSYCHOLOGY*, 64 (8), 16321645, 2011.

[30] https://github.com/tensorflow/models/tree/master/inception.

[31] http://www.image-net.org/.