Title of project placed here

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

Date of submission placed here

Abstract

abstract goes here

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

Yo! If you are reading this... I love you < 3 You will be my acknowledgement.

Contents

Chapter 1

Introduction

1.1 Sentence

I completed a program which allowed a turtlebot to autonomously map an environment using SLAM as well as create an inventory of items found in the environment.

1.2 Importance/Context/Motivation

Mobile robots are becoming more and more affordable and accessible which has allowed developers to take advantage of their applications in many different ways.

SLAM has it's application in rescue robots which http://www.aaai.org/Pressroom/Releases/release-02-0910.php these robots required realtime control and utilised only video streams to identify and rescue people.

This compares to this which utilises rugged mobile robots to create SLAM maps of mine shafts with minimal supersvision. This can then be applied to areas which are too unsafe/ small for humans to access. https://miningrox.informatik.tu-freiberg.de/en/

More affordable Drones can also be used to increasingly accurate create maps of property as well

Object detection can be used to detect humans via heat sensors etc. Aswell as identifying bombs etc.

1.3 Objectives/Problem statement

While the Turtlebot has been increasingly used by developers and researchers to create robotic software much of this development has been in focused and isolated components and not combined into useful applications. Similarly much work has primarily been developed using simulated environments and not tested using the Turtlebot or cameras.

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.

1.4 System Overview

1.4.1 System Diagram

1.4.2 How I achieved it

1.5 Outline of the dissertation

Chapter 2

Background

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 considered them in the context of natural visual systems and then in application in software.

2.1 Mobile Robots

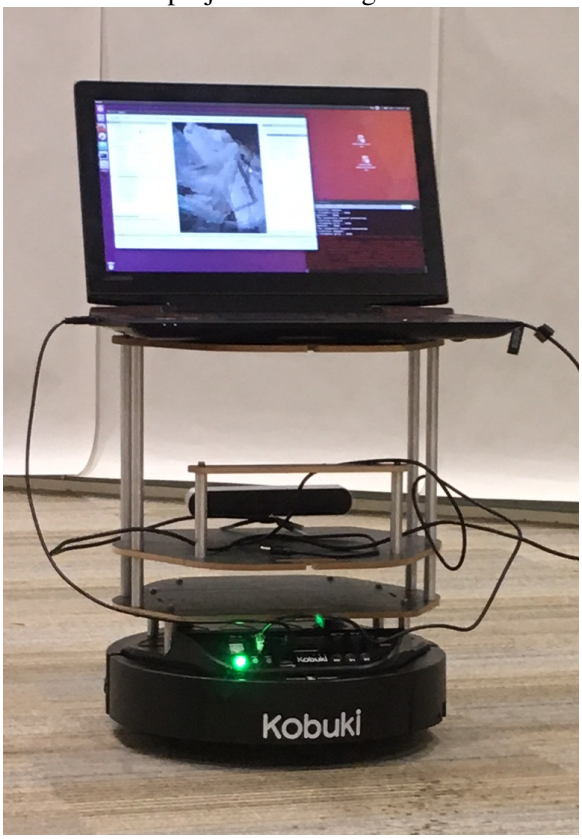
Mobile robots are robots that are not fixed to one location and are capable of moving around their environments. Movement is achieved generally through legs, wheels or tracks but aerial and nautical robots can use propellors. The robot can be controlled through manual-tele operation involving a human 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 an 'low-barrier-of-entry' platform for development while suitable hardware for SLAM and navigation[4]. The construction of the 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 seperate computer to remotely communicate with the host machine. The Turtlebot has many bespoke packages for use within ROS, these include standard applications such as teleoperation or navigation and more specialized packages such as frontier exploration.

2.1.2 Cameras

For nearly all use cases, the Turtlebot requires the use of 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. The Kinect is an active camera which utilises a time-of-flight sensor to estimate the distance of points in the 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.

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 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.

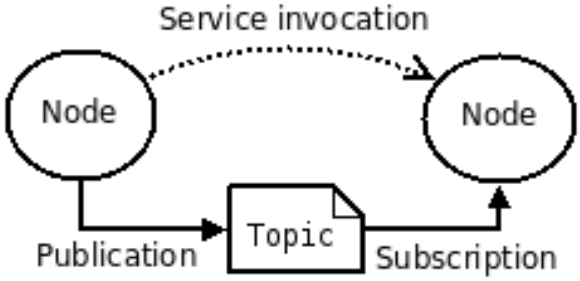
2.2.1 How does ROS work?

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 centralized 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



* DIAGRAM OF DESCRIBED SYSTEM

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 the range messages, if the range indicated is very small the node can use the actionlib service to request an adjustment in the orientation of the robot thus meaning the robot will avoid collisions.

2.3 SLAM (Simultaneous Localisation and Mapping)

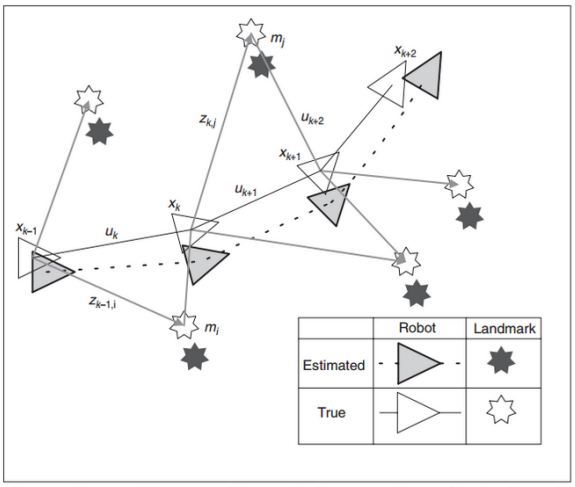
SLAM is the problem of simultaneously localising (finding the pose and orientation) of a camera within it's surroundings at the same time as mapping the structure of the environment. This requires a robot or camera with the ability to produce odometry readings as well as a camera with a range measurement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

SLAM forms the basis of navigation in most mobile robots, meaning unknown environments can be explored and mapped without the need for technology such as GPS which is not precise. 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.

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

This is considered to be a solved problem in Computer Science and their are many different approaches for finding a solution [6]. But at a high level SLAM is solved by using the environment

Figure 2.3: Illustration of the SLAM problem.



to update the pose of the robot. Using odometry as the sole measurement of localisation has an 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 enviroment. 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[11].

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

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

Where:

Given:

Robot controls $u_{1:t} = \{u_1, u_2, u_3 ... u_t\}$

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

Wanted:

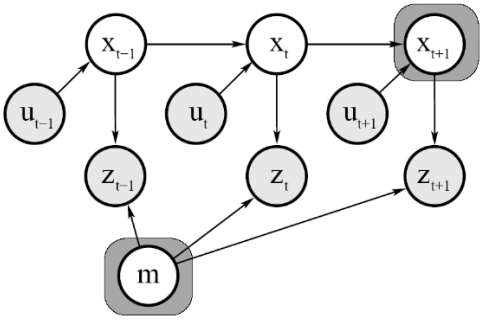
Map of the Environment m

Path of the robot $x_{0:t} = \{x_1, x_2, x_3 ... x_t\}$

(Starts at 0 to fix a coordinate frame)

SLAM was considered an especially difficult problem to solve [5] because these variables cannot be fully decoupled as can be observed in Figure []???. For example a map is required for localising landmarks which requires pose estimates.

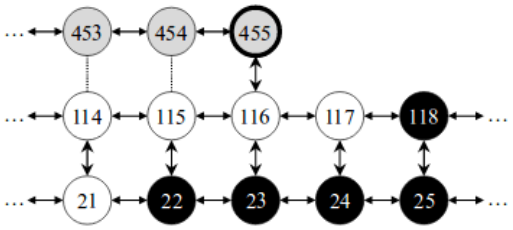
Figure 2.4: Graph illustrating the dependencies between SLAM variables.



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.[2] Another approach is RTABmap (Real-Time Appearance-Based Mapping) which produces a 3D point cloud map as well as 2D occupancy grid map. It is a RGB-D Graph-Based approach based on an incremental appearance-based loop closure detector.

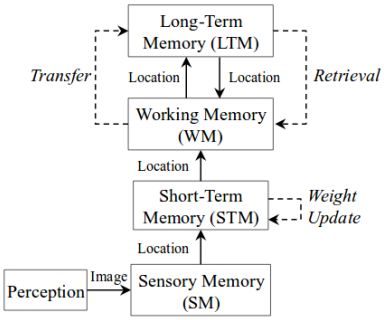
Figure 2.5: Graph representation of locations. Vertical arrows are loop closure links and horizontal arrows are neighbour links. Dotted links show not detected loop closures. Black locations are those in LTM, white ones are in WM and gray ones are in STM. Node 455 is the current acquired location



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 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.

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). [11] 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.

Figure 2.6: RTAB-Map memory management model.



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. Modelling a 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.

It is useful to think of object detection software in terms of an intelligent agent which is an autonomous entity that gathers and observers information through sensors and acts upon an environment using actuators to direct its activity to achieve goals.[13] Why???

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.[17] 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.

Many species exhibit a form of pattern or feature detection to trigger neural responses to visual changes. In contrast to humans which detect generic images features such as shapes, many lower organisms utilise goal-based feature detection. 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.

For example arthopods 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 [] thus satisfying the goal of feeding.

Once objects have been detected and classified by a creature are they retained in memory...

2.4.3 Attention Systems - Memory Hierachy

Spatial temporal Attention Models

How long do organism retain visual information.

How spatial temporal reasoning - don't overload memory - Have a Buffer - reaffirm spatial temporal Attention Memory Hierarchy - 9 seconds, How long? - What is long term? - What is short term? - What is important - How to decide - Intelligence agents

Humans forget things when they leave a room.

Pre attentive queue

2.4.4 Extracting Features

What determines if an area of a scene is worth triggering neural activity 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.

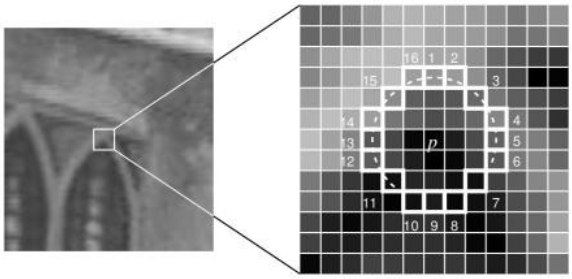
FAST Feature Detection

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) [14][15] 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.

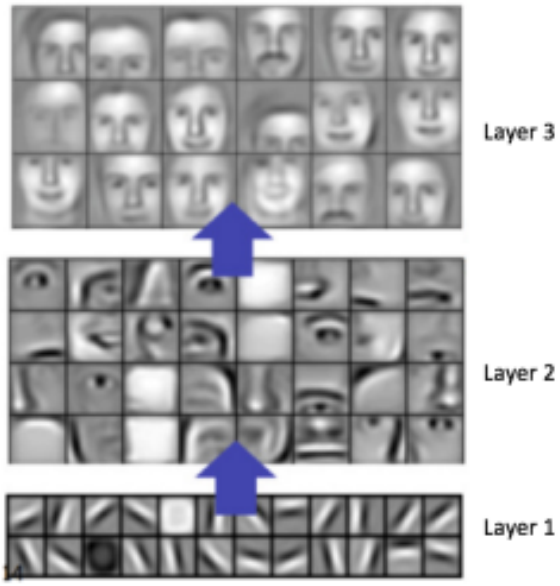
Figure 2.7: Illustration of the FAST



2.4.5 Object Classification with Deep Learning

Determining what an object is 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: Need to add******



Convolutional Neural Networks (CNN)

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

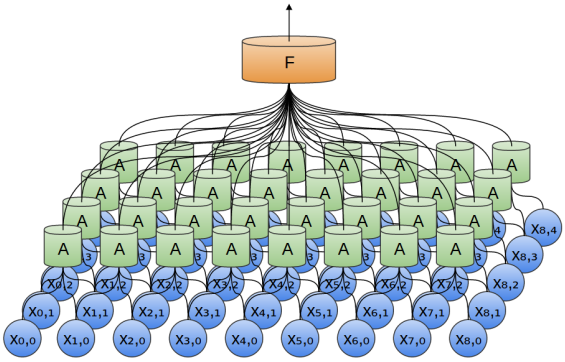
- it is rugged to distortions such as different lighting and occlusion

- 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 thus expressing computationally large models while keeping the parameters the neurons have to learn very small. [10]

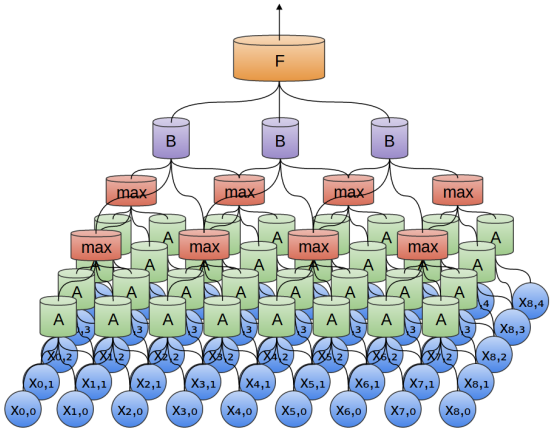
An approach called *symmetry* is used to look for properties in the data where at each segment of data we care about the same properties at the same time. Therefore a group of neurons, A can be introduced to a neural network that computes certain features such as the presence of an edge and then the output of this layer is fed into a fully-connected layer, F. ****Not convincing

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



These convolutional layers are composable, meaning 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 convultional 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 it's 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.

http://colah.github.io/posts/2014-07-Conv-Nets-Modular/

MobileNet with TensorFlow

TensorFlow is an open source software library produced by Google Brain for machine learning. It is frequently used due to the fact that it discovers 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 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[9]. The model has been trained on the 2012 ImageNet dataset and is highly optimized 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

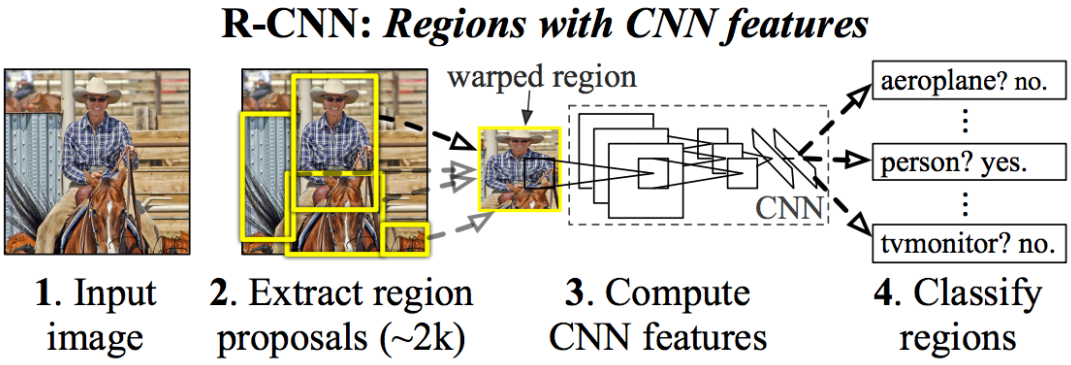
The Google Vision API is a cloud implementation of a deep convolutional neural networks trained from Google images and implemented with TensorFlow. As part of the Google Cloud platform users can make RESTful API calls from their language of choice querying images to be classified. The service provides label detection and facial detection amongst other features such as SafeSearch which filters images based on their obscene content[1].

2.4.6 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 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"[8] the authors 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.[16] This is in contrast to using an exhaustive sliding window approach found in Deformable Parts Models.[7] 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 Summary and Discussion

2.5.1 What currently exists for completing the objectives

2.5.2 What are the my knowledge gaps, What do I have to do to fill the gaps

2.5.3 What methods are not appropriate

Chapter 3

System Design

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

3.1 Requirements

The problem statement/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

- 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 Non-Functional Requirements

- 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

Platform Requirements

- Use ROS with Ubuntu v16.04
- Use a Kinect v2 RGB-D camera
- Use a ZED stereo camera
- Use a Intel Realsense ZR300 camera
- Application be able to be simulated in Gazebo

3.1.3 MOSCOW TABLE

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

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 would 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 futher 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.

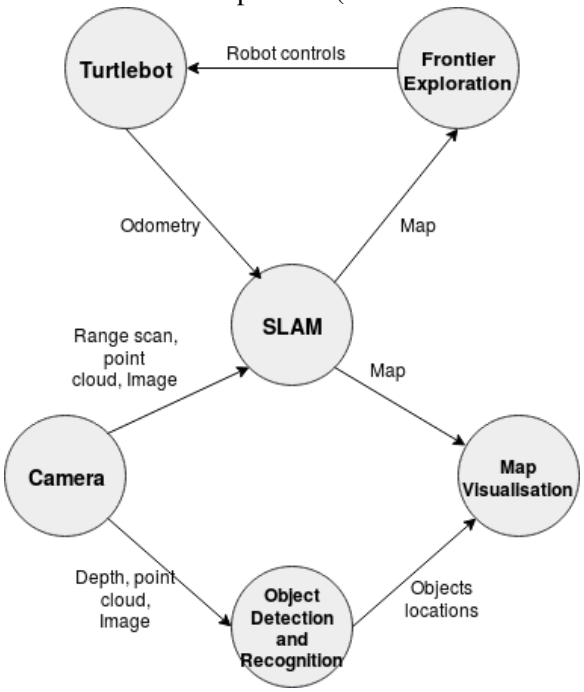
However, this is not required to achieve the project objectives and would require substantial research and time invested into developing a process of maintaining persistance when tracking objects in a 2D image.

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 will be 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 will 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 their 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 also allows the system design to be flexible and account for an iterative processes where different modules can be completed and tested in stages. The high-level system is able to capture the requirements and is sufficiently modular. The next chapter will describe the implementation process of the stated design... More***

Chapter 4

Implementation

4.1 Introduction

4.2 Mapping

4.3 Frontier Exploration

4.4 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 [] and Bing[]. Why? 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.

4.4.1 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) [**?**]. 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. **** What does R-CNN do... hybrid system...

0. Initial Exploratory Blob Detection

Initially a naive method was concieved that took SIFT or FAST features as regions of interest within an image and the created a blob around clusters of these features. This method used a handwritten 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.

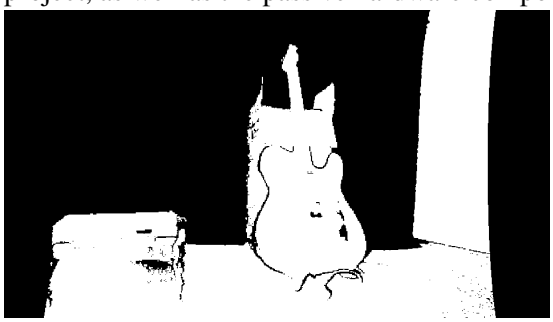
1. Thresholding with Otsu's Method

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, foreground and background[3]. 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 this process correctly distinguishes the high contrast guitar in the

Figure 4.1: High-Level System Architecture. Consisting of the four main active processes of the project, as well as the passive hardware components (Turtlebot and camera).

Figure 4.2: High-Level System Architecture. Consisting of the four main active processes of the project, as well as the passive hardware components (Turtlebot and camera).



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

No picture, but illustrate it with getting rid of white areas.

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 FIGURE 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 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)[] 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:

Find ε neighbours of p_n and identify the core points

Find connected components of core points

Assign border points or otherwise outliers

Where N is a neighbourhood:

Core points are $N_\varepsilon >= minPoints$

Border points are $N_\varepsilon < minPoints$ but still reachable from a Core point

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,y and the colour image) 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, depth and grey-scale image) 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.

4. Creating Bounding Boxes

and filter by number of features, get rid of part of the door

Figure 4.3: High-Level System Architecture. Consisting of the four main active processes of the project, as well as the passive hardware components (Turtlebot and camera).

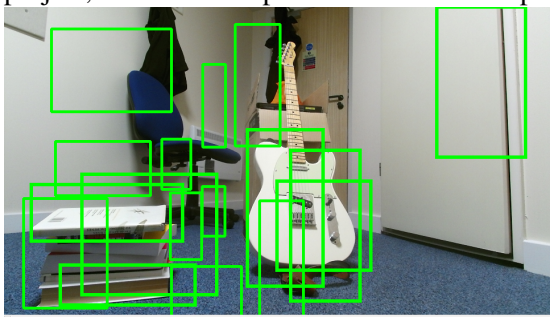


Image FAST Features

Explain the bounding mechanism

Do not use bounding boxes, as not entirely useful in mapping as this requires new detections and classifiations when changing the object pose. A simple spherical location detector is used to show the object is located within the this region of alpha

4.4.2 Classification

Haar Cascade

Using google vision means that speed does not decreae with more objects detected, statys constant

Transfer learning whereby a pre trained model is retrained for a similar problem.

I did this because of that which allows me to do this

Uses a buffer so that the robot can operate in real time asynchronous

Adding markers

Chapter 5

Evaluation

5.1 Testing

https://www.koen.me/research/pub/vandesande-iccv2011.pdf

Experiment 1 Question Experiment 2 Experiment 3

I think by doing this I will achieve... Experiments to answer questions

Chapter 6

Conclusion

6.0.1 Future work

incorporate active detection of features next time.

A signature map can be created from the locations of objects in the map. Indicating whether a floor is here. Similar to geological mapping

Appendix A

First appendix

A.1 Section of first appendix

Appendix B

Second appendix

Bibliography

[1] 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.

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

[3] Otsu's method. https://en.wikipedia.org/wiki/Otsu

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

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

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

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

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

[9] 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.

[10] 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.

[11] 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.

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

[13] Peter Norvig and Stuart J. Russell. *Artificial Intelligence*. Prentice Hall, 2003.

[14] 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.

[15] 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.

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

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