NATIONAL AND KAPODISTRIAN UNIVERSITY OF ATHENS
SCHOOL OF SCIENCES
DEPARTMENT OF INFORMATICS AND TELECOMMUNICATIONS
POSTGRADUATE STUDIES PROGRAM

POSTGRADUATE THESIS

Autonomic tackling of unknown obstacles in navigation of robotic platform

Nefeli K. Prokopaki Kostopoulou

Supervisors:
Stasinos Konstantopoulos, Research Associate NCSR Demokritos
Panagiotis Stamatopoulos, Assistant Professor NKUA

ATHENS
MAY 2019

ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αυτόνομη αντιμετώπιση αγνώστων εμποδίων στην πλοήγηση ρομποτικής πλατφόρμας

Νεφέλη Κ. Προκοπάκη Κωστοπούλου

Επιβλέποντες:
Στασινός Κωνσταντόπουλος, Συνεργαζόμενος Ερευνητής ΕΚΕΦΕ Δημόκριτος
Παναγιώτης Σταματόπουλος, Αναπληρωτής Καθηγητής ΕΚΠΑ

ΑΘΗΝΑ

ΜΑΪΟΣ 2019

POSTGRADUATE THESIS

Autonomic tackling of unknown obstacles in navigation of robotic platform

Nefeli K. Prokopaki Kostopoulou
R.N.: M1535

Supervisors:
Stasinos Konstantopoulos, Research Associate NCSR Demokritos
Panagiotis Stamatopoulos, Assistant Professor NKUA

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αυτόνομη αντιμετώπιση αγνώστων εμποδίων στην πλοήγηση ρομποτικής πλατφόρμας

Νεφέλη Κ. Προκοπάκη Κωστοπούλου
Α.Μ.: Μ1535

Επιβλέποντες:
Στασινός Κωνσταντόπουλος, Συνεργαζόμενος Ερευνητής ΕΚΕΦΕ Δημόκριτος
Παναγιώτης Σταματόπουλος, Αναπληρωτής Καθηγητής ΕΚΠΑ

ABSTRACT

The goal of the present thesis is to develop a method for a robotic outdoor platform. The robot should discover by itself, based on its sensors and its previous knowledge, how to approach an obstacle that stands in front of it, whether it is capable of driving over the obstacle or should avoid it. Obstacle avoidance ensures the safety and integrity of both the robotic platform and the people and objects present in the same space. That is one of the reasons why current approaches mainly concentrate on maneuver to avoid obstacles rather than yield autonomous systems with the ability to self improve. There is not much work done on curiosity-driven exploration, in which there is no explicit goal, but the abstract need for the robot to learn a new environment.

In the current thesis we introduce a system that not only autonomously classifies its environment to areas that can or cannot be driven over, but also has the capacity for self-improvement. To do so, we use a pre-trained neural network for whole scene semantic segmentation. We implement a program that accepts as input images extracted from the neural network mentioned above and predicts whether the illustrated scenes can be traversed or not. The program trains itself and then evaluates its effectiveness. Our results are quite satisfactory and the error rate can be explained by the fact that the environment is not evenly distributed in obstacles and paths, while at the same time it is not always clear which one is dominant. Furthermore, we show that our model can be easily optimized with just a few modifications.

SUBJECT AREA: Traversability estimation
KEYWORDS: traversability, data collection, navigation, exploration, neural networks

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας διπλωματικής είναι η ανάπτυξη μεθόδου ώστε μια ρομποτική πλατφόρμα εξωτερικού χώρου να ανακαλύπτει μόνη της, με βάση τους αισθητήρες της και τη γνώση που έχει αποκτήσει, πώς πρέπει να προσεγγίζει το εκάστοτε εμπόδιο που βρίσκεται μπροστά της, αν μπορεί να το υπερπηδήσει ή αν χρειάζεται να το παρακάμψει. Η αποφυγή εμποδίων εξασφαλίζει την ασφάλεια και ακεραιότητα τόσο της ρομποτικής πλατφόρμας όσο και των ανθρώπων και αντικειμένων που υπάρχουν στον ίδιο χώρο. Αυτός είναι ένας από τους λόγους που οι περισσότερες προσεγγίσεις τέτοιων θεμάτων επικεντρώνονται κυρίως στους ελιγμούς για την αποφυγή εμποδίων αντί για την παραγωγή αυτόνομων συστημάτων με ικανότητα αυτοβελτίωσης. Δεν υπάρχει μεγάλη βιβλιογραφία για ρομπότ που έχουν την περιέργεια να εξερευνήσουν το περιβάλλον τους, για περιπτώσεις δηλαδή που δεν υπάρχει συγκεκριμένος στόχος, αλλά μόνο η αφηρημένη ανάγκη του ρομπότ να εξερευνήσει ένα καινούριο περιβάλλον.

Στην παρούσα διατριβή παρουσιάζουμε ένα σύστημα που όχι μόνο κατατάσσει αυτόνομα το περιβάλλον του σε προσπελάσιμες και μη προσπελάσιμες περιοχές, αλλά επίσης έχει την ικανότητα να αυτοβελτιώνεται. Για να το επιτύχουμε, χρησιμοποιούμε ένα προεκπαιδευμένο νευρωνικό δίκτυο που αναπαριστά χρωματικά τα αντικείμενα της σκηνής. Αναπτύσσουμε ένα πρόγραμμα, το οποίο δέχεται ως είσοδο εικόνες που εξάγονται από το προαναφερθέν νευρωνικό δίκτυο και προβλέπει αν το ρομπότ μπορεί να προσπελάσει τα απεικονιζόμενα αντικείμενα. Το πρόγραμμα αυτό εκπαιδεύεται και στη συνέχεια αξιολογείται η αποτελεσματικότητά του. Τα αποτελέσματά μας κρίνουμε ότι είναι αρκετά ικανοποιητικά. Το ποσοστό σφάλματος μπορεί να εξηγηθεί από το γεγονός ότι το περιβάλλον δεν είναι ομοιόμορφα κατανεμημένο σε εμπόδια και προσπελάσιμες περιοχές ενώ παράλληλα δεν είναι πάντοτε σαφές τι από τα δύο υπερισχύει. Τέλος, δείχνουμε ότι είναι εύκολο να μειωθεί το ποσοστό σφάλματος με λίγες μόνο τροποποιήσεις.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Εκτίμηση προσπελασιμότητας
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: προσπελασιμότητα, συλλογή δεδομένων, πλοήγηση, εξερεύνηση, νευρωνικά δίκτυα

To those who are always and forever
with me
D.K. & K.P.

Σε μια από τις καλύτερές μου φίλες,
που με στηρίζει πάντα
Λ.Π.

και σε έναν φίλο που μου λείπει
Ν.Π.

To the one whose thought alone
still makes me smile
S.T.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Prof. Takis Stamatopoulos for providing his expertise, support and guidance throughout all my undergraduate and postgraduate studies, as well as for giving me the chance to work on such an interesting topic.

I am also grateful to my other supervisor, Researcher Stasinos Konstantopoulos for his support and patience for the whole time that I have been working on this project. His insights and guidance were crucial throughout the research and writing of this thesis.

I would like to thank Katerina Maria Oikonomou and Christos Sevastopoulos for their help and advices. Also, thanks to my friend Spyros for his support.

Lastly, I would like to thank my parents and my grandparents for supporting me and believing in me. Without them nothing that I have accomplished would be possible.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This project has been developed since October 2018 in the University of Athens at the department of Informatics and Telecommunications as my postgraduate thesis. It has been implemented in cooperation with the National Centre for Scientific Research “Demokritos” at the Institute of Informatics and Telecommunications.

Athens, May 2019

1. INTRODUCTION

During the last years the focus of research for robotic applications evolved from well structured indoor environments to unstructured outdoor environments. With this expansion of interest, it is a crucial prerequisite to reliably classify traversable ground in the environment, especially when it comes to truly autonomous (or else self-supervised) systems. This topic is typically referred to as *traversability analysis* or *obstacle detection* [1]. The verb traverse is defined as "*to pass or move over, along, or through*". Hence *traversability* refers to the affordance of being able to traverse [2]. Failing on this task can cause great damage or restrict the robots movement unnecessarily.

So, traversability is the generic capability of a robotic ground vehicle to navigate within environments of varying complexity, while ensuring safety in terms of collisions or reaching unrecoverable states and achieving goals in an optimal mode of operation [3]. Occasionally other terms such as *mobility* [4], *drivability* [5], etc are used to describe the same concept.

Although traversability is considered a fundamental capability for mobile robots, in some cases it is limited to the problem of simple obstacle avoidance [2]. When such approaches are used, the robot tries to avoid making any physical contact with the environment, and heads only to open spaces. Its response would be the same whether it encounters an impenetrable wall or a balloon that can just be pushed aside without any damage. Therefore, methods that can automatically learn the traversability affordances from the robot's interactions with the environment are valuable for robotics.

In addition, there is the possibility that previously learned behavior is not relevant, because the visual appearance and traversability of roads may have changed due to various reasons [6]. That is probably why geometry-based analysis is the direction followed by the majority of traversability analysis methodologies in the past [3].

Supervised learning approaches are unlikely to work reliably in unknown or unstructured outdoor environments. That is because the system assesses the traversability using an off-line learned model trained with specific terrain types. For example, if a system is trained with terrain samples in a specific season, the systems might not detect traversable regions in other seasons [7].

Unsupervised, or else self-supervised, learning approaches may be the solution to this problem. They use on-line learning methods in order to exploit newly- acquired training data in making traversability predictions about unknown terrain [8]. That way the learned traversability concepts are incrementally updated with new data only. That comes with the advantage that the updated classifier is immediately available for navigation and that the memory requirements for learning are reduced, compared to off-line methods.

In the real world and its unstructured and dynamic surroundings such as vegetation landscape and terrain, the perception of a mobile robot needs to be capable of navigating this unknown environment by using sensor modalities [9].

So, if autonomous mobile robots are to become more generally useful, they must be able to adapt to new environments and learn from experience. To do so, they need a way to store pertinent information about the environment, recall the information at appropriate times, and reliably match stored information with newly-sensed data. They also must be able to modify the stored information to account for systematic changes in the environment [10].

Estimating the traversability of terrain in an unstructured outdoor environment is a core functionality for autonomous robot navigation [8]. Nevertheless, the traversability of more complex terrain, such as vegetation and sloping ground, is extremely difficult to characterize beforehand. It is difficult to find general rules which work for each vehicle's capabilities and for a wide variety of terrain types such as trees, rocks, tall grass, logs, and bushes. As a result, methods which provide traversability estimates based on predefined terrain properties such as height, shape or colour (*geometry-based* and *appearance-based* analyses) will be unlikely to work reliably in unknown outdoor environments. That is why combining data collected a priori together with the vehicle's navigation experience is more likely to work better for deciding terrain traversability (more about *hybrid* approaches in Papadakis [3]).

Last but not least, traversability should be treated as an affordance and not simply as a predefined property of different types of terrain [8]. That is because a large vehicle may be able to drive over small saplings that would present an insurmountable obstacle to a smaller vehicle. A stair that is traversable for a hexapod robot may not be traversable for a wheeled one. So, when used here, *affordance* implies the complementarity of the robot and the environment, the interaction between them [2].

In this thesis we will tackle on how an autonomous mobile robot can improve its traversability estimation method in natural environments, meaning not only on bare ground-like environment but also on terrain containing vegetation. On contrast, we will rule out high-risk applications where a single accident can be fatal to the robot like planetary or volcano exploration. We will concentrate in everyday practical situations. We will determine how to introduce a learning capability to the robot that will enable it to decide for itself the traversability of the terrain around it, based on input from its sensors and its experience of traveling over similar terrain in the past. We would also like our robot to plan further ahead and avoid entering traps that prevent it from reaching its goal.

The rest of the thesis is organized as follows:

- In Chapter 2 we give a background on the topic we are dealing with. We present already existing traversability estimation algorithms; the goals, the approaches, the methods used, the strengths and weaknesses.
- In Chapter 3 we present the rationale behind the decisions made and the algorithms used or developed.
- In Chapter 4 we perform an evaluation of our implementation and explain why its weaknesses occur.
- In Chapter 5 we give our conclusions and ideas for future work.
- In Appendix A we share some URLs of prototype implementations from researchers who kindly offered their work open-source.
- In Appendix B we share the URLs of some popular datasets used for training neural networks.
- In Appendix C one can find a brief outline on how we collected the sample data we used for our evaluation.

2. BACKGROUND

In order to have an autonomous robot improve its traversability estimation we will need to address each sub-problem individually:

1. Traversability estimation algorithms that can be improved from experience and examples.
2. Methods for collecting the data needed by the algorithm above, from the sensory input that is available to the robot. The input does not necessarily directly map to positive or negative decision.
3. Navigation strategies. There might be an explicit goal to achieve, e.g. follow the fastest or easiest route to a target. Or it could be curiosity-driven exploration, meaning the abstract need to learn a new environment.

We will now present the state of the art in all three areas of research.

2.1. Learning traversability estimation algorithms

In order for an autonomous robot to be able to safely navigate, it is crucial for it to be able to conclude on its own the terrain traversability around it. Historically, most commonly, traversability analysis is treated as a binary classification problem [3], i.e. distinguishing traversable from non-traversable terrain. But later on, it became clear that rough natural terrain is not easily partitioned into clear traversable and non- traversable classes. The need for finer classification was recognized. The new idea was to either assign a continuous traversability score or classify the terrain into the various classes that were commonly encountered within a particular application. Many papers have been published regarding traversability estimation approaches, and here we present some of the most recent and most influential.

This line of research starts with Lalonde et al. [4] that segment local three-dimensional (3D) *point clouds* using a purely geometric approach, for autonomous robot navigation purposes. A *point cloud* is a set of data points in space, generally produced by 3D scanners. The approach used is a segmentation in three terrain categories, based on scatter-ness, linear-ness, and surface-ness. That way the authors are able to represent porous volumes such as grass and tree canopy, capture thin objects like wires or tree branches, and capture solid objects like ground surface, rocks or large trunks, respectively.

A different line of research starts with Pfaff et al. [11] that decided to represent the environment of a mobile robot with *elevation maps*, another geometric approach. A *digital* *elevation map (DEM)* [12] is also known as a $2\frac{1}{2}$*-dimensional representation* of the environment [11]. It is a two-dimensional (2D) array of terrain elevation measurements. More concrete, it is a grid that stores in each cell the vertical distance above or below the surface, the height of the territory.

The representation of the environment with elevation maps, however, can be problematic when a robot has to utilize these maps for navigation. For example, when a mobile robot is located in front of a bridge, the underpass will completely disappear and the elevation map will show a non-traversable object.

Pfaff et al. [11] classify the cells of elevation maps into four classes: parts of terrain seen from above, vertical structures, vertical gaps and traversable areas. They also maintain a set of intervals per grid cell, which are computed and updated upon incoming sensor data. The authors use this classification for their extension to the elevation maps. The advantage here is that they can deal with vertical structures like walls of buildings, but also with overhanging structures like branches of trees or bridges. In order to determine the class of a cell, they consider the variance of the height of all measurements falling into this cell. If this value exceeds a certain threshold, they identify it as a point that has not been observed from above. Then they check whether the point set corresponding to a cell contains gaps exceeding the height of the robot. When a gap has been identified, they determine the minimum traversable elevation in this point set. So they only keep the height values for the lowest surface in each cell. As a result, the area under the bridge, in the previous example, will appear as a traversable surface, and the bridge will not be represented.

Yet, another approach works a little differently. The autonomous vehicle has also to decide for itself the traversability of the terrain around it. But it has no a priori knowledge of the kind of terrain it will traverse, so it must learn as it goes along by observing the geometry and appearance of the terrain. That is both proprioceptive and exteroceptive sensory data processing [3]. In a few words, *proprioceptive* analysis is useful in learning while the vehicle traverses a given terrain, gathering data with on-board sensors as it goes. On the other hand *extreroceptive* data processing is divided in geometry-based and appearance-based analysis.

Shneier et al. [10] follow a hybrid approach such as the above. They use a local *occupancy grid* map that scrolls under the vehicle as the vehicle moves, and cells that scroll off the end of the map are forgotten. *Occupancy grid* maps [13] are 2D arrays depicting the robot's environment with regions classified as empty, occupied or unknown. Shneier et al. [10] do not use a global map and the previous known information is forgotten once the robot moves away from that location. Considering distance above or below the ground, color, texture, and contrast, they estimate each cell's traversability. This estimation of the cost of traversing regions is used to generate models of terrain in order for the robot to learn from its own experience.

Kim et al. [8] developed a hybrid method that is based on autonomous training data collection. Their method exploits the robot's experience in navigating its environment to train classifiers without human intervention. The main idea is that image data obtained in the past is associated with traversability labels obtained in the present, the so called *on-line machine learning*. The learning process produces a classifier which makes traversability predictions for new terrain regions. Successes and failures of the navigation provide positive and negative traversability labels for cells in a grid-based representation of the terrain surrounding the vehicle. Cells under the robot footprint that can be driven over are traversable and therefore yield positive training examples, while those that hinder the robot's motion are non-traversable and result in negative examples.

Later on, Suger et al. [1] proposed a learning approach that uses a 2D occupancy grid map (like Shneier et al. [10]), where each cell stores features that provide information from the senors. Every sell is associated with at least one feature vector that is computed from the 3D point clouds (like Lalonde et al. [4]) that are mapped to the respective cell. The authors use the features mentioned bellow (mostly geometrical like Lalonde et al. [4] and Pfaff et al. [11]) to distinguish different types of terrain as well as traversability constraints of the robot.

1. Maximum height difference and
2. slope

reflect the ground-clearance of the robot as well as the motor power.

1. Roughness and
2. remission values (meaning the reflection or scattering of light by a material)

help to distinguish concrete and vegetation types.

In contrast with Kim et al. [8], Suger et al. [1] collect partially and only positive labeled training data. And then they use existing strategies [14, 15] to learn a classifier from this kind of training data.

Similarly to Kim et al. [8], Lee et al. [7] employ a self-supervised on-line learning approach. As the vehicle explores its environment, the classifier is trained incrementally with autonomously labeled training samples. Their approach determines whether unknown regions in front of a vehicle are drivable while the vehicle is in motion and without human's input. Their traversability detection method is based on *incremental nonparametric Bayesian clustering (INBC)*. In probability theory and statistics, *Bayes' theorem* (alternatively *Bayes' law* or *Bayes' rule*) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. Many approaches have used it for traversability estimation. For example, Suger et al. [1] use a naive Bayes classifier [14], and Lalonde et al. [4] use Bayesian classification to label the incoming data.

Several authors have considered the problem of *simultaneous localization and mapping (SLAM)* in an outdoor environment. Some tried to solve it with elevation maps generated from 3D range data acquired with a mobile robot [11]. But elevation maps only model a single surface, they lack the ability to represent vertical structures or even multiple levels. *Multi-level surface* maps (*MLS* maps) [16], on the other hand, store multiple heights in each grid cell. This extension allows a mobile robot to model environments with more than one surface, such as bridges, underpasses, buildings or mines.

The approach of Pfaff et al. [11] allows to deal with vertical and overhanging objects in elevation maps. Despite their efforts, they still lack the ability to represent multiple surfaces. For example, the robot can plan a path under a bridge but not over it, as mentioned before.

The attention had most often been focused on methodologies that access the traversability characteristics before actually driving over the respective region [3]. But Droeschel et al. [5] use a way for continuous mapping and localization during mission, without the necessity to map the environment beforehand or to stop for acquiring new 3D scans and to process them. Their representation consists of local maps (*multiresolution* maps as the authors call them) and a global map (called *allocentric* map).

Each local map is a robot-centered 3D grid map. It has high resolution in the vicinity of the robot and coarser resolutions with increasing distance (hence its name multiresolution map). Each cell stores 3D point measurements (including height from ground) along with occupancy information. Since the robot, hence the sensor too, is moving during acquisition of the data, individual grid cells are stored in a circular buffer to allow for shifting elements in constant time. So when the robot moves, the circular buffers are shifted whenever necessary to maintain the egocentric property of the map.

A forward-looking image alone may be insufficient for planning and navigation [12]. Robots operating in rough terrain may require knowledge of terrain that has been observed but is currently out of the sensor field of view such as terrain under and behind the robot. And that is the main reason why global maps are useful.

In this case, the global map is built from local multiresolution maps acquired at different view poses of the robot [5]. This is useful in order to overcome pose errors and to localize the robot with respect to a fixed frame. While traversing the environment, a local map is extended whenever the robot explores previously unseen terrain and optimized when a *loop closure* has been detected. The loop closure problem consists in detecting when the robot has returned to a past location after having discovered new terrain for a while. The authors localize towards this local map during mission to get the pose of the robot in the global map. They assess the traversability of the terrain by analyzing height differences in the global map and plan cost-optimal paths.

Subsequently, Wigness et al. [6] proposed another way to learn new behaviors quickly in the field with no or minimal human supervision. They propose a methodology for learning reward functions from human examples via visual perception. This means that the agent learns how to simply assign costs to distinct terrain types, and follows the trajectory with the minimum cost. This approach is more focused than this of Suger et al. [1] in following the optimum path, but less in experimenting with traversability. It also insists on dynamic environments, while Suger et al. [1] interprets the characteristic of traversability to be static, and further assume that dynamic objects are detected and removed in advance.

Hirose et al. [17] introduced a semi-supervised approach for traversability estimation, called GONet. The core of the proposed approach are *Generative Adversarial Networks (GANs)* [18]. GANs are a framework for estimating generative models via an adversarial process. They are deep neural network architectures that simultaneously train two models (more about neural nets we will see on Section 2.2.1). A generative model, let's say "a team of counterfeiters", that captures the distribution of the training data and tries to produce fake samples and use it without detection. And a discriminative model, let's call it "police", that tries to detect the fake images by estimating the probability that a sample came from the training data rather than the "counterfeiters". Competition in this framework drives both teams to improve their methods until the "counterfeits" are indistinguishable from the genuine samples.

There is no need for any *Markov chains* or approximate inferences during either training or generation of samples [18]. A *Markov chain* is a stochastic (or random) process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event. It has actually many similarities with Bayes' theorem mentioned above. Many authors use an extension of Markov chains, named *Markov Decision Process (MDP)*, to formulate the problem of autonomous navigation and allow mobile robots to make decisions [6, 19]. Others use *Markov Random Field (MRF)* [20] to i.e. enforce spatial consistency in a map or the preference that neighboring points have the same label [4].

Returning to GONet [17], intuitively, it works comparing two images, an input image and a generated one. The generated image is created with a particular class of GANs trained on positive examples only. It is similar to the input image and looks as if it came from the actual positive examples. The GONet compares the input with the generated image to decide whether the area seen through the input image is traversable. The main assumption of the approach is that when the input indeed shows a traversable area, the generated image would look very similar to it. But when the input depicts a non-traversable scenario, then the generated image would look different. The generated images not only look like traversable areas, but also resemble the input query.

We have presented recent methods on how to conduct traversability estimation models from data; noting that data needs to be labeled. We will now proceed to present how this labeled data can be autonomously acquired and which sensors are needed.

2.2. Data collection methods

For a long time until in recent years, robots have long been used in industrial environments. In industrial environments, robotic systems are pre-programmed with repetitive assignments which lack the capability of autonomy and as such operate on the basis of a structured approach [9]. Such an environment cannot be adaptive for a mobile robot since it eliminates the need for autonomy. As such, surviving and adapting in the real world is more complex for any robotic system in comparison to the industrial setting since the risk of failure, system error, external factors, obstacles, corrupt data, human error and unrecognizable environments is more prevalent.

So, for unstructured environments, a way to collect training data is to obtain them through a human operator that drives a safe trajectory that is similar to the environment where the robot should later be able to reliable operate in. This process for training data generation has the advantage that it is fairly easy to execute.

One way to do that is to label the cells of the map that intersect with the projection of the footprint of the robot as positive examples [1]. This has the drawback that the labeled data are only positive examples, leaving tons of unlabeled data to learn from.

A similar approach, inspired by the above, is to train the robot with many positive images of traversable places and just a small set of negative images depicting blocked and unsafe areas [17]. The positive examples can be collected easily by simply operating the robot through traversable spaces, while obtaining negative examples is time consuming, costly, and potentially dangerous. But small amounts of negative examples can improve traversability estimation in comparison to using only positive data.

In a variation of this, optimal trajectory examples are collected [6] in order to be used from a reward function and train the robot.

Another way is to autonomously collect data without any human supervision [8, 7]. The robot can image the terrain in front of it and store the resulting image patches in a data pool [8]. Then, each image patch is an observation of a single cell in a grid-based terrain map. Initially all of this data is unlabeled, because the robot has not yet interacted with the terrain, and its traversability is unknown. Then the robot attempts to drive over the terrain that it previously observed, thus discovering the traversability properties of the environment.

Autonomous driving in unstructured environments faces many challenges which do not exist in structured environments [9]. In unstructured environments, object attributes needed for driving cannot be defined as priori. Information concerning objects has to be gained through sensors even though these are normally ambiguous and therefore introduce uncertainty and avail information that is redundant.

In environments where the ground is not flat or contains obstacles that are not purely vertical, the basic approach of classifying based on the observed obstacles from 2D laser scanners can not be safely used anymore. In these cases, 3D range data, by i.e. stereo-cameras, radar or 3D-laser scanners, is necessary. A popular approach to collect data, either for the initial training or to use them for traversability estimation, is to use *light detection and ranging (LIDAR)* [1, 4] (also called *ladar* [4, 10]). LIDAR is a surveying method that measures distance to a target by illuminating the target with pulsed laser light and measuring the reflected pulses with a sensor. LIDAR sensors use emitted light, so they work independent of the ambient light. Night or day, clouds or sun, shadows or sunlight, they pretty much see the same in all conditions. On the other hand they often have trouble sensing highly reflective surfaces and transparent objects, such as mirrors and glass doors [17].

Other commonly used sensors are stereo cameras. They are really inexpensive, especially compared to LIDAR. Because they use reflected light, they can see an arbitrary distance in the daytime, as opposed to LIDAR whose range of vision is limited. They also have higher resolution and are able to see color, instead of just a grayscale. But they need illumination at night, and headlights might not be enough. When it comes to geometric data, stereo camera sensory might not be enough to detect all important features with the reliability necessary for safe traversing.

In occasions where sensors that can measure in all directions are needed, hardware requirements are imposed. One can use a laser scanner that rotates around a vertical axis [5]. That way the sensor can measure in all directions, except for a cylindrical blind spot around the vertical axis centered on the robot.

Another option is to not use geometric data, but instead concentrate on visual data. Instead of using LIDAR [1, 4], one can use different sensors like fisheye camera [21, 17], to estimate whether a physical space is traversable or not. This kind of approaches are mainly focused on obstacle detection and avoidance, even in dynamic environments. But they are less interested in traversability estimation for obstacles that may seem untraversable while in fact can be easily driven over by a robot, like tall grass.

More so, even without the introduction of uncertainty, sensors in themselves are ambiguous [9]. For example, a lemon and a soccer ball can look similar from a certain perspective. In addition, a cup could be invisible in case the cupboard is shut and it can be challenging to tell the difference between a remote control and cell phone is they are both facing down. These factors are all contributive to the challenges of perceiving the state of the environment.

A third choice is to use proprioceptive information, via on-board sensors such as *inertial measurement unit (IMU)*, motor current, and bumper switch [8] or even wheel encoder data [7] (like wheel odometry measurements [5]). IMU is an electronic device using a combination of accelerometers and gyroscopes, sometimes also magnetometers. It measures and reports a robot's specific force, angular rate, and the 6D robot pose (i.e. 3D location and orientation).

The use of the sensors above make it possible to assess the progress of the robot automatically and estimate its motion [5]. That way successes and failures of the navigation provide positive and negative traversability examples [8]. This kind of approaches can make predictions about the traversability of the terrain based on the robot's past experiences and navigation sensor values.

In some cases all three choices are used [8, 10]. While geometric data provide information about the traversability of the terrain, they are not always sufficient to measure the affordance of traversability. For example, a short (non-traversable) tree trunk and a patch of tall (traversable) grass will result in a similar height. However, they differ in visual appearance.

Likewise, a white vertical flat surface may be an impenetrable wall in one environment whereas in another environment a similar surface may be a door that can just be pushed to open [2]. But appearance data may not be enough to distinguish the two cases mentioned above. So, a robot can be equipped with stereo vision cameras which collect visual and geometric data from the environment, but also with a bumper switch at the front of the vehicle that can be used along with motor current sensors to recognize situations like getting stuck because of an obstacle or slipping, respectively [8].

The reason why proprioceptive sensory may not be sufficient on their own is that bumpers, for example, do not prevent robots from falling off edges and can fail to detect small obstacles [17]. That is why all geometric, appearance and haptic information are useful.

Since most of the approaches use at least some visual information, we will deepen a bit more on this.

2.2.1. Visual information

From the perspective of intelligence, dealing with input directly to generate output without further processing of input information is a kind of low-level intelligence [22]. It would be more satisfactory if a mobile robot could imitate the way human beings deal with such a task. Fortunately, deep learning, with its advantage in hierarchical feature extraction, provides a potential solution for this problem.

Machine learning is a subset of *artificial intelligence (AI)* that studies the design of algorithms that can learn. It is used to effectively perform a specific task without using explicit instructions, but relying on models and inference instead. The idea of using machine learning to control robots needs humans to show the willingness to lose a certain measure of control [9]. This is seemingly counterintuitive in the beginning although the gain for doing this is to allow the system to begin learning on its own.

Artificial Neural Networks (ANNs) or just *neural networks or nets (NNs)* are computational processing systems which are heavily inspired by the way biological nervous systems (such as the animal brain) operate [23]. Basically, it is a simplified model of the way the brain processes information. The neural network itself is not an algorithm, but rather a framework for many different machine learning algorithms to work together and process complex data inputs. It learns to perform tasks by considering examples, generally without being programmed with any task specific rules. Neural networks are mainly comprised of a high number of interconnected computational nodes, referred to as *neurons*. These neurons are aggregated into layers. Different layers may perform different kinds of transformations on their inputs. The neurons work entwine in a distributed fashion to collectively learn from the input in order to optimize the final output.

Deep learning is commonly introduced as a means of making sense of data with the use of multiple abstraction layers [9]. The difference between deep learning and machine learning is that the former place emphasis on the subset of machine learning resources and method and uses them to solve any difficulties that need thought, whether human or artificial. Deep learning has revolutionized computer vision and is the core technology behind capabilities like autonomous mobile robots.

There are many different neural network architectures [24], such as *Convolutional Neural Networks (CNNs or ConvNets [25])*, *Generative Adversarial Networks (GANs)*, *Deep Residual Networks (DRNs)*, etc.

Convolutional neural networks are analogous to traditional neural networks in that they are comprised of neurons that self-optimize through learning [23]. The only notable difference between convolutional and regular neural networks is that convolutional networks are primarily used to solve image-driven pattern recognition tasks (but can also be used for other types of input such as audio). A typical use case for a convolutional network is to feed it images in order for it to classify the data, e.g. outputs “cat” if it is fed with a cat picture and “dog” if it is fed with a dog picture. Thus, compared to other regular, deep, feed-forward neural networks with similarly-sized layers, convolutional networks have much fewer connections and parameters and so they are easier to train [26]. Convolutional neural networks are perfectly suitable for computer vision (i.e. robot navigation), so below we are going to concentrate mainly on them.

But when we come to adversarial examples, these are basically the images that fool convolutional networks [27]. Let’s take an example image and apply a perturbation, or a slight modification, so that the prediction error is maximized. The difference between the original and the altered content may be imperceptible to humans, but the network might make drastic errors in classification. So the goal is to train a network to understand the differences between real content and artificially created one.

That is what generative adversarial networks [18] do. As we have already mentioned in Section 2.1, they are two networks working together. The one is tasked to generate content and the other has to judge the same content. The discriminative network receives either training data or generated content from the generative network. This creates a form of competition where the discriminator is getting better at distinguishing real data from generated data and the generator is learning to become less predictable to the discriminator. Generative adversarial networks can be quite difficult to train. There are two networks that have to be trained and either of which can pose it's own problems. Also their dynamics need to be balanced. If prediction or generation becomes too good compared to the other, the neural network will not converge as there is intrinsic divergence.

Networks that are very effective at learning patterns up to 150 layers deep, much more than the regular 2 to 5 layers one could expect to train, are the so called deep residual networks [28]. Basically, these networks add an identity to the solution, carrying the older input over and serving it freshly to a later layer. This essentially drives the new layer to learn something different from what the input has already encoded.

The section below provides a brief outline on how convolutional neural networks work. If you are already familiar with basic information about convolutional neural networks, proceed to Section 2.2.1.2 where some of the most popular convolutional neural networks are described.

2.2.1.1. Outline on Convolutional Neural Networks

Based mainly on Deshpande [29] beginner's guide, we give a brief description on convolutional neural networks.

When a computer takes an image as input, it sees an array of pixel values. This array is sized depending on the resolution and size of the image. Let's say we have a color image in JPG form and its size is 32×32. The representative array will be $32 \times 32 \times 3$ (the 3 refers to RGB values). Each of these numbers is given a value from 0 to 255 which describes the pixel intensity at that point. These numbers, while meaningless to us when we perform image classification, are the only inputs available to the computer. The idea of neural networks is that we give the computer this array of numbers and it outputs numbers that describe the probability of the image being a certain class (e.g. 80% for cat, 15% for dog, 5% for bird etc).

So what we want the computer to do is to be able to differentiate between the images it is given and figure out the unique features that make a dog a dog or that make a cat a cat. This is what human minds do subconsciously as well. Roughly, we can classify a dog in a picture if it has identifiable features such as paws or snout or four legs. In a similar way, the computer is able perform image classification by looking for low level features such as edges and curves, and then building up to more abstract concepts like paws and beaks, through a series of convolutional layers. This is a general overview of what a convolutional neural network does.

The first layer in a convolutional neural network is always a *convolutional layer*. The best way to explain a convolutional layer is to imagine a flashlight that is shining over the top left of the image. Let's say that the light this flashlight shines covers a 5×5 area. And now, let's imagine this flashlight sliding across all the areas of the input image. In machine learning terms, this flashlight is called a *filter* (also referred to as a *neuron* or a *kernel*) and the region that it is shining over is called *receptive field*. This filter is also an array of numbers called *weights* or *parameters*. A very important note is that the depth of this filter has to be the same as the depth of the input (in order for the math to work out), so the dimensions of this filter is $5 \times 5 \times 3$.

As the filter is sliding, or *convolving*, around the input image, it is multiplying the values in the filter with the original pixel values of the image (aka computing *element wise multiplications*). Mathematically speaking, this would be 75 multiplications in total for the first position the filter is in. Then these multiplications are all summed up to a single number. The next step is to move the filter to the right by one unit and repeat. This process is repeated for every location on the input volume and produce a number.

There are 784 different locations that a 5×5 filter can fit on a 32×32 input image. So after sliding the filter over all the locations, the result is an $28 \times 28 \times 1$ array of numbers, called an *activation map* or *feature map*. Using more filters makes possible to preserve the spatial dimensions. If there are two $5 \times 5 \times 3$ filters used instead of one, the output volume would be $28 \times 28 \times 2$. The more filters, the greater the depth of the activation map, and the more the information about the input image.

Each of these filters can be thought of as *feature identifiers*, for features like straight edges, curves, simple colors. For example, let's assume one filter is a curve detector. If there is a shape in the input image that generally resembles the curve that this filter is representing, then all of the multiplications summed together will result in a large value. And if not, the value will be much lower. This will happen if there is nothing in the image section that responds to the curve detector filter. So the activation map will have large values in the areas where is most likely to have curves, and low values on the least likely.

Moving to the next layer, its input would be the output of the previous layer. In this case the input of the second layer will be the activation maps that result from the first layer. So the input is basically describing the locations in the original image where certain low level features appear. Applying a set of filters on top of that, as it passes through the second convolutional layer, the output will be activations that represent higher level features. Types of these features could be semicircles (i.e. combination of a curve and straight edge) or squares (i.e. combination of several straight edges).

As going through the network and through more convolutional layers, the activation maps represent more and more complex features. By the end of the network, there might be some filters that activate when there is handwriting in the image, when they recognize green objects, etc. An interesting thing to note is that going deeper into the network, the filters begin to have a larger receptive field. That means they are able to consider information from a bigger region of the original input volume.

Finally, at the end of the network there is a *fully connected layer*. This layer takes as input the output from the layer preceding it, and outputs a vector with dimensions the number of classes that the program has to choose from. Each value of the vector represents the probability of a certain class. For example, let's take a digit classification program. The output is a dimensional vector such as [0 0.1 0.1 0.75 0 0 0 0 0 0.05], since there are ten digits. This represents a 10% probability that the original image is a 1, a 10% that it is a 2, a 75% that it is a 3, and a 5% that it is a 9. Basically, a fully connected layer take the activation maps of the previous layer and determines which features most correlate to a particular class.

Now that we have seen how these things work roughly, let's come across some of the most well known and frequently used convolutional neural networks. In the next section many types of neural network models will be discussed. Models that recognize scenes or objects present in the input image, models that detect the location of different objects, even models that try to understand and describe the whole scene within an image. Each type was thoroughly examined to find out which one is the most compatible with the purpose of this thesis. For those not already familiar with common models for object classification, localization and detection, as well as scene understanding and segmenting, the next chapter will introduce these concepts. The others feel free to proceed to Section 2.3 where navigation strategies are described.

2.2.1.2. Convolutional Neural Network models

One of the most remarkable feats of the human visual system is how rapidly, accurately and comprehensively it can recognize and understand the complex visual world [30]. The various types of tasks related to understanding what we see in a visual scene is called *visual recognition*. In computer vision, visual recognition has enjoyed some great success in recent years. Particularly in single object categorization (i.e., object classification, object localization) like in the example with the classification of the cat and the dog mentioned above, in page 26. While recognizing isolated objects is a critical component of visual recognition, a lot more is needed to be done in order to recognize multiple objects (i.e., object detection, object segmentation), let alone reach a complete understanding of visual scenes (i.e., scene segmentation).

2.2.1.2.1. Classification and Localization

Convolutional networks have recently enjoyed a great success in large-scale image and video recognition [25]. This has become possible due to the large public image repositories, such as *ImageNet* [31] and high-performance computing systems, such as GPUs or large-scale distributed clusters. In particular, an important role in the advance of deep visual recognition architectures has been played by the *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)* [32], an annual competition organized by the ImageNet team since 2010 (basically, the annual Olympics of computer vision [27]). There research teams evaluate their computer vision algorithms with various visual recognition tasks such as *Object Classification* and *Object Localization*, as shown in Figure 2.1.

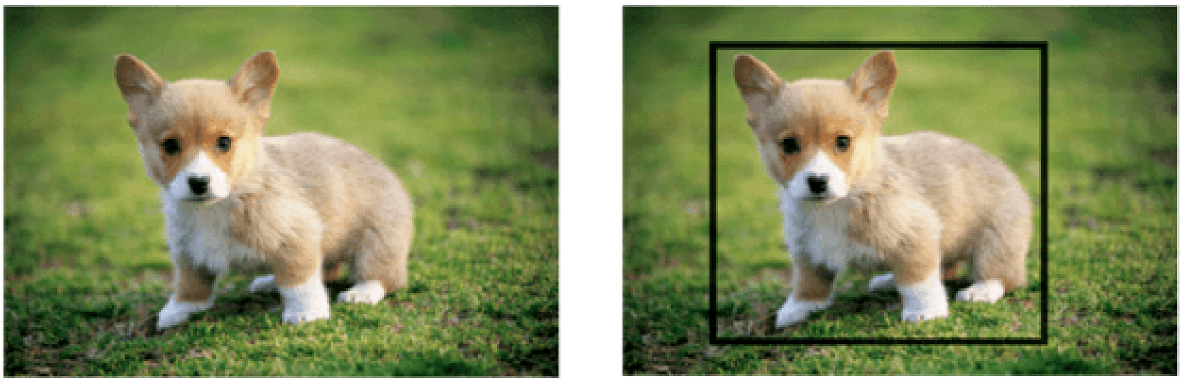


Figure 2.1: Object Classification is identifying that picture as a dog (left). Object Localization involves as well a bounding box to show where the object is located, apart from the class label "dog" (right). Both are suitable for single object.

Many research groups have been very generous in releasing their models to the open-source community. Some of the most well-known models, such as AlexNet, VGGNet, Inception, ResNet, Xception [26, 25, 33, 28, 34] won in the ILSVRC. And others, like SqueezeNet, MobileNet [35, 36] etc participated in it.

Krizhevsky et al. [26] are widely regarded to have done one of the most influential publications in the field. Because of AlexNet, 2012 marked the first year where a convolutional network was used to achieve a top 5 test error rate of 15.3%. The next best entry achieved an error of 26.2%, which was an astounding improvement that pretty much shocked the computer vision community. *Top 5 error* is called the rate at which, given an image, the model does not output the correct label with its top 5 predictions. AlexNet really illustrated the benefits of convolutional networks and backed them up with record breaking performance in the ILSVRC. Since then the use of convolutional neural networks for classification has dominated the field.

Simonyan and Zisserman [25] reinforced the notion that "*convolutional neural networks* *have to have a deep network of layers in order for this hierarchical representation of visual data to work*". VGGNet is characterized by its simplicity, using very small (3×3) convolutional filters, instead of AlexNet's (11×11), stacked on top of each other in increasing depth from 16 to 19 layers. It was best utilized with its 7.3% error rate. But it is painfully slow to train.

The original incarnation of the Inception architecture used in Szegedy et al. [33] submission to ILSVRC 2014 is called GoogLeNet. It is a 22 layer convolutional neural network. It actually uses 12 times fewer parameters than the winning architecture AlexNet, from two years ago. It is also significantly more accurate, with a top 5 error rate of 6.7%. Inception was one of the first models that introduced the idea that convolutional layers do not always have to be stacked up sequentially, but could perform many operations in parallel. Finally, this new model places notable consideration on memory and power usage.

Depending on their skill and expertise, humans generally hover around a 5%-10% error rate [27]. But He et al. [28] came up with the ResNet architecture that has an incredible error rate of 3.6%. Aside from the new record in terms of error rate, ResNet is well-known due to its extreme depth of up to 152 layers. This is 8 times deeper than VGGNet. ResNet is also a great innovation for the idea of residual learning.

2.2.1.2.2. Detection

Image classification is a lightweight form of object detection. The difference between them is that in classification algorithms the goal is to label an image with a category of the object it belongs (or at least the most likely predictions). While in detection algorithms the aim is to draw a bounding box around the object of interest to locate it within the image, as shown in Figure 2.2. Also, it is not necessary to draw just one bounding box in the object detection case. There could be many bounding boxes representing different objects of interest within the image, the number of which is known a priori.

A naive approach to do that would be to take different regions of interest from the image by sliding windows from left and right, and from up to down. And then use a convolutional network to classify the presence of the object within the chosen region. The problem with this approach is that the objects of interest might have different spatial locations and aspect ratios. Hence, the number of regions would be huge leading to computationally blow up. Therefore, algorithms like SSD, YOLO and R-CNN [37, 38, 39] have been developed to find these occurrences in the fastest way possible. The models mentioned above are also released to the open-source community.

To bypass the problem of selecting a huge number of regions, Girshick et al. [37] proposed a method called *R-CNN: Regions with CNN features*, one of the most impactful advance-

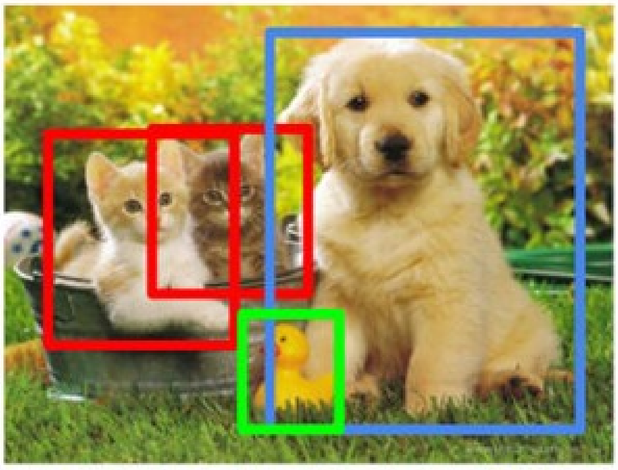


Figure 2.2: Object Detection locates and identifies multiple objects and all their instances (cat, dog, duck) within the image.

ments in computer vision [27]. The process can be split into two general components, the region proposal step and the classification step. For the former step, a selective search [40] is used to extract just 2000 regions from the image that have the highest probability of containing an object. The authors called them region proposals. These 2000 candidate region proposals are warped into a square and fed into a trained convolutional network (VGGNet or AlexNet in this case) that acts as a feature extractor. In the latter step, the extracted features are used to classify the presence of the object within that candidate region proposal and to refine the boundary box with the most accurate coordinates. The main disadvantage here is that it still takes a huge amount of time to train the network as there are 2000 region proposals per image to classify. Also, the selective search algorithm is a fixed algorithm. Therefore, no learning is happening at that stage, something that could lead to generating bad candidate region proposals. That is why *Fast R-CNN* and *Faster R-CNN* followed up.

But, as an alternative, is a separate region proposal step needed? Can both boundary boxes and classes be directly derived from feature maps, in one step? Redmon et al. [38] use a single convolutional network to predict the bounding boxes and the class probabilities for these boxes. They use a *single shot object detection* algorithm called *YOLO: You Only Look Once*, that is much different from the *region based detection* algorithm mentioned above. YOLO divides every image into a S $\times$ S grid and every grid predicts N bounding boxes and class probability for them. The bounding boxes having the class probability above a threshold value are selected and used to locate the object within the image. Note that at runtime, the image runs on the convolutional network only once. Hence, YOLO is super fast and can be run real time. It sees the complete image at once as opposed to looking at generated region proposals. However, a limitation for YOLO is that it only predicts one type of class in each grid. Hence, it struggles with very small objects.

Yet another algorithm was developed by Liu et al. [39]. *SSD: Single Shot Detector* differs from others single shot detectors due to the usage of multiple layers that provide a finer accuracy on objects with different scales. SSD runs a trained convolutional network (for example VGGNet model) on input image only once and calculates a feature map. Then it runs a 3×3 sized convolutional filter on this feature map to foresee the bounding boxes and classification probability. It attains a good balance between speed and accuracy.

2.2.1.2.3. Semantic Segmentation

Nowadays, *semantic segmentation* is one of the key problems in the field of computer vision [41]. Looking at the big picture, semantic segmentation is one of the high-level task that paves the way towards complete scene understanding. Although humans perform scene segmentation with apparent ease, automatic scene segmentation is a very challenging problem [42]. Segmentation refers to the process of mapping each pixel in an image to an object class. Each object class has to be segmented separately. For example see Figure 2.3. As one can imagine, this is a much more complex problem as compared to the classification or even detection problem.

The importance of scene understanding as a core computer vision problem is highlighted by the fact that an increasing number of applications nourish from inferring knowledge from imagery. Some of those applications include self-driving vehicles, human-computer interaction, virtual reality etc. With the popularity of deep learning in recent years, many semantic segmentation problems are being tackled using deep architectures, most often convolutional neural networks, which surpass other approaches by a large margin in terms of accuracy and efficiency.

Probably the most well known models concentrating on scene segmentation are *Fully Convolutional Neural Networks (FCNs)* [43]. The original fully convolutional network is trained for pixel-wise prediction, without extracting the region proposals. The authors adapt contemporary classification networks (AlexNet, VGGNet and GoogLeNet) into fully convolutional networks and transfer their learned representations by *fine-tuning* to the segmentation task. Fine-tuning is a process in which a network model that has already been trained for a given task, is modified in order to perform another similar task. The main idea of fully convolutional networks is to make classical convolutional networks take as input arbitrary-sized images. Contrary to them, fully convolutional networks only have convolutional and pooling layers which give them the ability to make predictions on arbitrary-sized inputs. *Pooling layers* are in charge of reducing the number of parameters when the images are too large. The results of the original fully convolutional network trained on PASCAL VOC are shown in Figure 2.4.

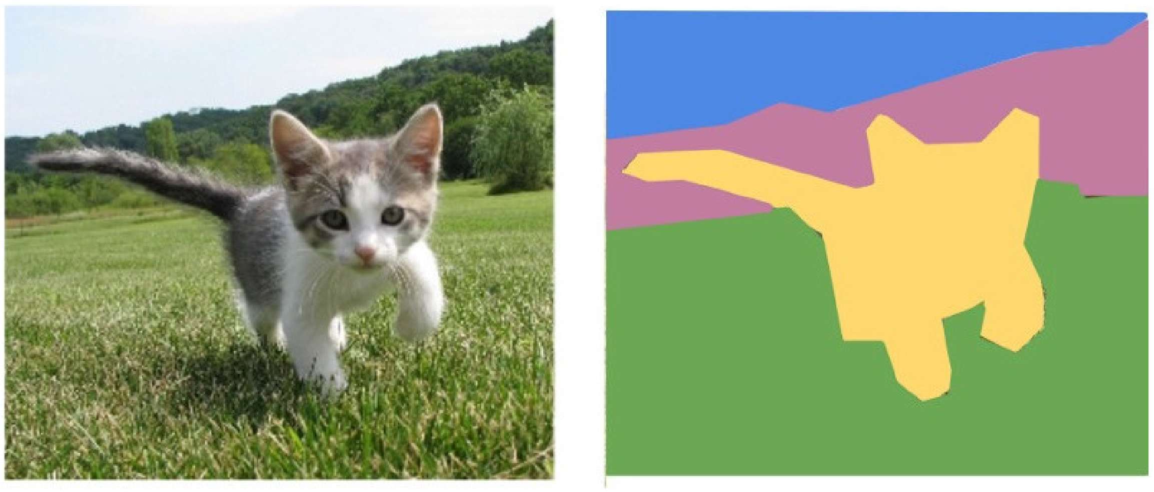


Figure 2.3: Semantic Segmentation makes a prediction at every pixel within an image. That means there is a label for each pixel, instead of object. The predictions are grass, cat, tree and sky. Original image (left), output of the semantic segmentation (right).

Semantic understanding of visual scenes is one of the holy grails of computer vision [44]. The emergence of large-scale image datasets like ImageNet [31] and COCO [45], along with the rapid development of the deep convolutional neural network approaches, have brought great advancements to visual scene understanding.

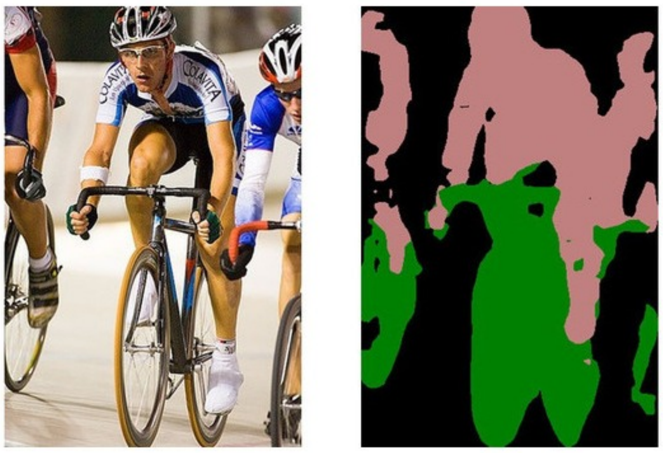


Figure 2.4: Figure taken from Fully Convolutional Neural Networks for Semantic Segmentation, trained on PASCAL VOC: Original image (left), predicted label map (right). Notice the predictions are only for the foreground, the entire background in the left image appears black on the right.

But what is the strategy for deciding where to go next? Is there a specific goal for the robot to reach? Should it find the best trajectory? Explore the environment? A discussion of this topic is made in the following section.

2.3. Navigation strategies

A natural thing would be to let the robot learn about the traversability of the environment, while another perspective would be to concentrate on more preservative situations, such as go or no-go [21]. Even though the former method allows the robot to autonomously learn a model of the environment, encourages exploration and consequently improves the learning and generalization performance [19], the trial and error part of it involves a high risk to damage the robot. The latter, on the other hand, has as main priority to prevent robots from colliding with objects, injuring people, getting stuck in constrained spaces, or falling over an edge.

A third approach would be to let the autonomous vehicle navigate from a defined start point to a fixed goal point [10, 19]. That way the robot has to build a model of the world around it and plan a path from the start to the goal. A way to do that is to enable the robot to learn which regions to avoid and which to seek out, in early runs, so that in later runs it can determine the most efficient path [10]. In cases where there is no knowledge of the map of the robot's current environment, the designated goal location can be acquired via cheap localization solutions, such as visible light localization, Wi-Fi signal localization or GPS [19].

Generally, *intrinsic motivations* are related to curiosity, exploration, the interest for novel objects and surprising events and the interest in learning new behaviors. They come in contrast to *extrinsic motivations* which are to obtain biologically relevant resources to avoid damage and vouch survival.

In robotics, extrinsic motivations can be related to the accomplishment of the user tasks assigned to the robot. In contrast, intrinsic motivations drive behavior and actions to gain knowledge: e.g., to explore the world, to learn to predict novel or surprising stimuli, to acquire a higher competence to change the environment with action.

In our knowledge, most approaches concentrate on robot navigation in order to make sure they do not collide with other objects and they follow a route to reach a specific goal. There is not much work done on curiosity-driven exploration, in which there is no explicit goal, but the abstract need for the robot to learn a new environment. The traversability estimation is most commonly performed in order for the robot to move safely rather than explore the environment and learn which areas are traversable and which are not.

2.4. Conclusions

Intrinsic motivations fully entered the field of machine learning and robotics only recently. That is despite some early pioneering computational explorations, and although they were investigated by psychologists for a long time. Intrinsic motivations are very important for autonomous robotics. They allow the acquisition of knowledge and skills, in the absence of tasks directly established by the robot users, e.g. to learn to manipulate different objects without being instructed to do so. Also they are task-independent. That means they can drive the robotic system to acquire skills re-usable to accomplish many possible tasks in a certain class of environments.

It has become clear to researchers in robotics and adaptive behavior that current approaches are yielding systems with limited autonomy and capacity for self-improvement. To learn autonomously and in a cumulative fashion is one of the hallmarks of intelligence. So that is what we want to achieve in the current thesis. The traversability estimation can become a goal of action based on intrinsic motivations.

We explored many ways for a robot to be able to decide for itself the traversability of the terrain around it and plan paths to avoid any obstacles. We also investigated ways to improve the robot's traversability estimation method in everyday practical situations. So, this thesis will be focused on the one thing that is missing; intrinsically motivated learning. It will mainly concentrate on how to explore the environment around the robotic platform and improve its knowledge of traversability.

Enough of background, let's now see how our autonomous mobile robot takes input from its environment and handles the newly acquired knowledge to improve its traversability estimation method.

3. AUTONOMOUS NAVIGATION

The goal is for the robot to be able to autonomously navigate in natural environments. To do so, we could use a pre-trained neural network in order to be able to distinguish traversable from non-traversable terrain.

For the purposes of this thesis we will basically deal with pre-trained models. By borrowing a little story from Gupta [46] we are going to explain why. Imagine two people, Mr. Potato and Mr. Athlete. They sign up for soccer training at the same time. Neither of them has ever played soccer and the skills like dribbling, passing, kicking etc. are new to both of them. Mr. Potato does not move much but Mr. Athlete does. That is the core difference between the two even before the training has even started. As you can imagine, the skills Mr. Athlete has developed as an athlete (e.g. stamina, speed and even sporting instincts) are going to be very useful for learning soccer even though Mr. Athlete has never trained for soccer. Mr. Athlete benefits from his pre-training. Mr. Potato on the other hand will have to develop all these skills from scratch, something that will cost him much more energy and time.

In this chapter we describe the primary thoughts and the actual strategy on how to reach the goal mentioned above.

3.1. Selecting a convolutional neural network

Primarily, a neural network is needed in order to convert images that depict the environment seen by the robot, to a form more recognizable by it. Ideally, after the conversion, all objects would be distinguished from all traversable areas. But the idea of finding such a neural network is probably not realistic. Thus, the goal for this section is to find a neural network that distinguishes all objects from one another.

3.1.1. Object classification, localization, detection

The first attempt was to use one of the most-well known models from ILSVRC. With a little help from the code of Rosebrocke [47] we experimented on pre-trained VGGNet, ResNet, Inception and Xception. We fed them images and, as expected, they returned classification predictions about them. With the contribution of ImageNet a list of human-readable labels and the probability associated with them was printed.

Images containing just one object (e.g. soccer ball, couch) led to predictions that were satisfactory in their entirety. But being fed with images containing multiple objects (e.g. book and glasses) the networks got confused. They gave some decent predictions (like envelope or book jacket, in the previous example) but also some that were a little bit off (like lighter or birdhouse), within their top-5 list.

Images that contain vegetation, which are of particular interest to us in this work, were the worst case scenario. For example, the networks above, after being fed with an image of a tree, gave as most likely predictions kinds of seeds like lemons. This probably happened because the networks concentrated on the one part of the whole image that was most recognizable by them. Finding out what the predictions with smaller probabilities were, did not help. As the networks kept predicting, they got desperate and started giving all sorts of predictions.

Even though these networks are proven to be great on object classification, when it comes to scene recognition there is no trivial way to make them work correctly.

Similarly, models specialized in object detection like SSD or YOLO, do not seem to be able to generalize on scene segmentation issues. As dictated by their name, they detect all objects within an image, but ignore the rest of the scene.

3.1.2. Related work

Many papers have been published regarding robot navigation approaches that use already existing deep neural networks, or modify them in order to meet their needs. Unfortunately some of these researchers had not made their code publicly available, in order for us to rely on part of their work and extend it with our own ideas. And while others kindly released their code online, their networks were not trained compatible to the needs of this thesis.

Some papers considering traversability estimation concentrate on go or no-go situations, [17, 48]. They use generative adversarial networks and train them to estimate whether the space seen through the given image is traversable or not. They are trained in indoor environments, which means that we would have to train them from scratch to work in natural outdoor environments.

Likewise, Tai et al. [22] further extended the concept of deep neural networks to not only perception but also decision-making. Basically, they used a structure that fuses several convolutional neural network layers with decision-making process, in order to explore an unknown environment. According to the authors, in traditional computer vision applications each label of the output represents either an object or scene categories. The outputs of their model are control commands that show the platform which route to follow.

Other researchers use neural networks to classify the area in front of the robot according to traversability and level of confidence [49, 50]. These neural networks assign class labels to parts of the input image. Classes which show that "*only ground or only obstacle is seen in the area*", are of high confidence. While the rest inspire lower confidence. These classes are separated to "*ground and obstacle may be seen*", "*obstacle is seen but does not fill the area*", "*location where an obstacle meets the ground*".

Some approaches identify only a few class labels to classify the whole image [51, 52, 53]. While others have as their main goal to find and follow a path [54, 55]. The first category usually includes scene labels that can be used in outdoor environments (such as sky, road, tree, grass, building), as their title declares. The second one, while also being able to recognize such class labels, identifies them as obstacles.

3.1.3. Scene segmentation

So, the aim is on total scene segmentation rather than single or even multiple object categorization. Semantically meaningful image understanding is a relatively recent topic in computer vision. That explains why, compared to recognition, far fewer papers address scene segmentation in neural networks [42]. A general semantic segmentation architecture can be broadly thought of as an *encoder network* followed by a *decoder network* [41]. The encoder is usually a pre-trained classification network like VGGNet or ResNet that outputs a feature map. The task of the decoder is to semantically project the lower resolution features learned by the encoder, onto the higher resolution (pixel space) to get the best closest match to the original input.

Given a visual scene of, let us say, a living room, a robot equipped with a trained convolutional network can accurately predict the scene category. However, to freely navigate in the scene and manipulate the objects inside, the robot has far more information to digest [44]. It needs to recognize and localize not only the objects like sofa, table, and television but also to segment the stuff like floor, wall and ceiling for spatial navigation. It probably needs to recognize also object parts, e.g. a seat of a chair or a handle of a cup, to allow proper interaction.

Following the instructions of Le [41], on his guide on how to do semantic segmentation using deep learning, we attempted to implement the most popular architecture for semantic segmentation, fully convolutional networks. We had in mind that the encoder, VGGNet in this case, would be pre-trained, but we would have to train the decoder from the beginning on KITTI [56]. But fully convolutional networks, at least those trained on KITTI dataset, do semantic segmentation only on foreground and ignore the background (Figure 2.4, in page 34). In order for us to be able to decide terrain traversability, whole scene segmentation is necessary.

3.1.4. Datasets

Scene parsing, or recognizing and segmenting objects in an image, remains one of the key problems in scene understanding [44]. Going beyond the image-level recognition, scene parsing requires a much denser annotation of scenes with a large set of objects.

However, the current datasets have limited number of objects, e.g. COCO [45], PASCAL VOC [57]. In many cases those objects are not the most common objects one encounters in the world like frisbees or baseball bats. Or the datasets only cover a limited set of scenes, like urban sceneries, e.g. Cityscapes [58] and KITTI [56]. Most of the large-scale datasets typically only contain labels at the image level or provide bounding boxes, e.g. ImageNet [31], PASCAL VOC and KITTI. ImageNet has the largest set of classes, but contains relatively simple scenes. Compared to the largest annotated datasets COCO and ImageNet, ADE20K [59] comprises of much more diverse scenes.

Finally, existing datasets with pixel-level labels typically provide annotations only for a subset of foreground objects, and no background, e.g. PASCAL VOC and COCO. That is probably why fully convolutional networks trained on KITTI, as mentioned before, do not give class labels to the background pixels. But, generally, pixel appearance features allow to perform well on classifying (amorphous) background classes. ADE20K categorizes semantic classes present in the scene into three super classes: stuff (sky, road, building, etc), foreground objects (car, tree, sofa, etc), and object parts (car wheels and door, people head and torso, etc).

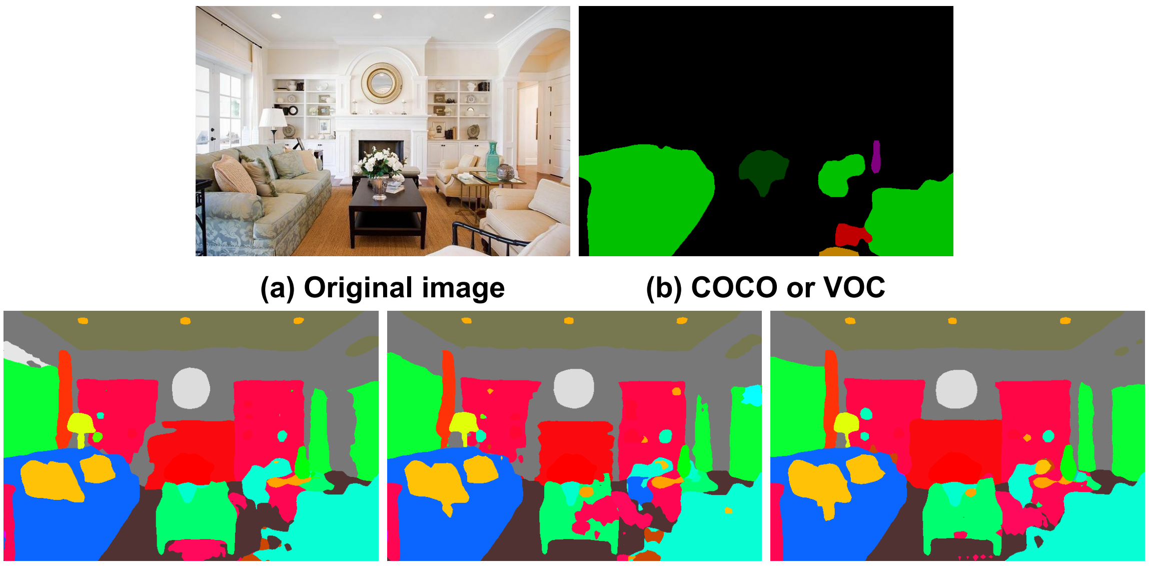
3.1.5. Pre-trained models on scene segmentation

After finding some pre-trained models [1] we experimented on different algorithms and training datasets. The backbone network in all those cases was ResNet. Be reminded that successful deep neural network architectures for image level classification like AlexNet, VGGNet and ResNet are a natural precursor to, and often a direct part, of semantic segmentation architectures.

The algorithms mentioned previously are the fully convolutional network [43], *Pyramid Scene Parsing Network (PSP)* [60] and DeepLab [61]. All three of them when trained on COCO or PASCAL VOC do not perform complete scene semantic segmentation. When they are fed with indoor images they recognize only foreground objects. No class labels are given to the background pixels. Our hypothesis that the part of the image they ignore

[1] `https://gluon-cv.mxnet.io/model_zoo/segmentation.html`

is the background is further intensified by the way outdoor images are handled. Their result on outdoor input is a black image. On the contrary, when trained on ADE20K all tree networks transact semantic segmentation on the whole scene. The outputs from all tree algorithms trained on each of the three datasets are depicted in Figure 3.1 for indoor inputs, and Figure 3.2 for outdoor inputs.



(a) Original image **(b) COCO or VOC**

(c) FCN with ADE **(d) PSP with ADE** **(e) DeepLab with ADE**

Figure 3.1: Results given from indoor input. Fully convolutional network, Pyramid scene parsing network and DeepLab trained on COCO, PASCAL VOC and ADE20K dataset.

Many deep learning architectures have been proposed for image segmentation. As far as we know, in order to be able to semantically segment the whole image we can use any of the three algorithms mentioned above as long as they are pre-trained on the ADE20K dataset. But how can we use them to help us distinguish traversable from non-traversable terrain?

3.2. Estimating traversability

We implemented a program that when given an image predicts whether the illustrated scene is traversable or not. The program trains itself and then evaluates its effectiveness. All the input values are images extracted from a neural network for whole image semantic segmentation. These images, as shown in Figure 2.3, in page 34, consist of colors depicting objects and parts of the scene. Each color is recognized by the computer as a triad of numbers representing the amount of red, green and blue present to produce the original color.

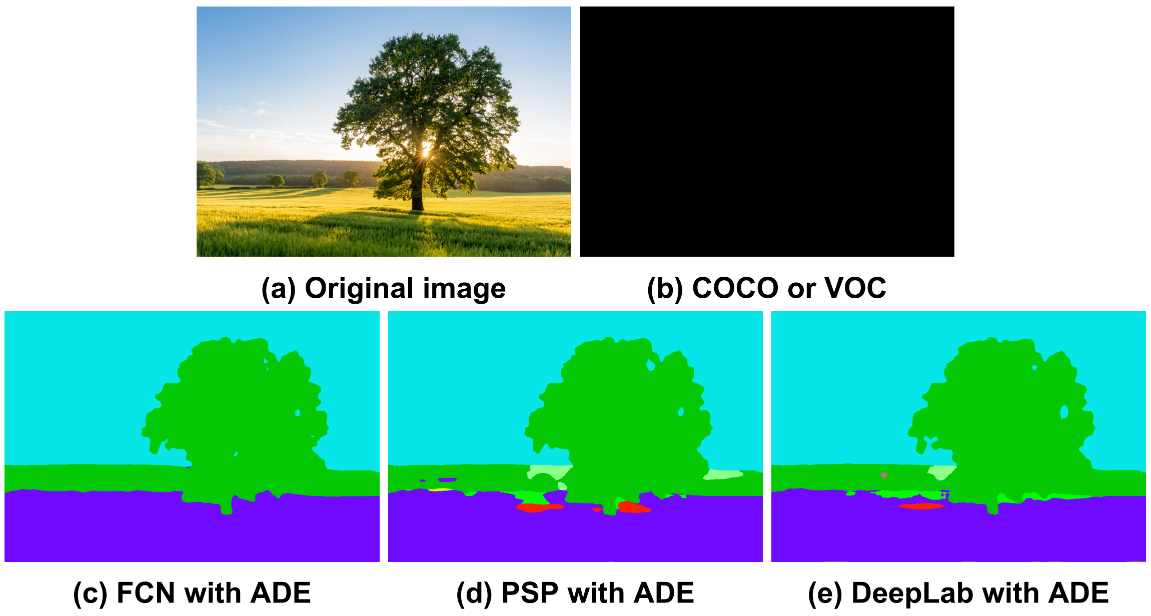


Figure 3.2: Results given from outdoor input. Fully convolutional network, Pyramid scene parsing network and DeepLab trained on COCO, PASCAL VOC and ADE20K dataset.

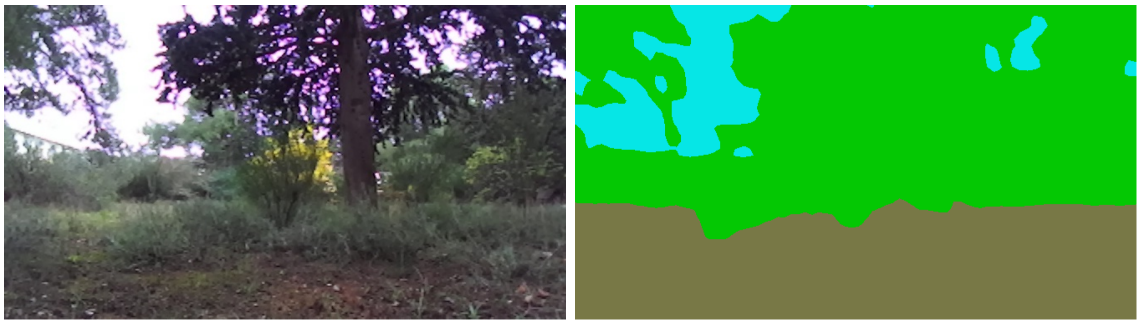
For humans to be able to observe the program's functionality we keep a default correspondence between colors and class labels. Also, to ease comprehension, all operations are rounded to the second decimal place.

During the program training period, it reads previously annotated images. These annotations are penetrable region or obstacle. In this thesis, the term *penetrable* will be used interchangeably with the term traversable. The program keeps track of how many times each color is found in traversable and how many in non-traversable images. So, it can compute the traversability percentage of each color. For example, let's suppose that brown corresponds to "earth". Brown color exists in 70 images from which 56 are traversable and the other 14 non-traversable. So the traversability percentage of the earth is

$$56 \times 100 \div 70 = 80$$

Therefore, the earth (and, consequently, the color brown) has 80% chance of being traversable.

During evaluation we implement two different calculating methods. Both of them are responsible for deciding whether the given images are traversable or not. Every image with chance of being traversable greater or equal to 50% is considered to be traversable (and therefore with chance less that 50% is considered to be non-traversable). Let's take Figure 3.3 and try to explain the two methods. For the purpose of this example, we suppose that brown has 80% chance of being traversable, as is found to be penetrable 8 times out of the total of 10. Green 9 out of 30 (30%) and blue 9 out of 20 (45%). To ease our understanding let's say that brown corresponds to "earth", green to "tree" and blue to "sky".



(a) Original image **(b) Image resulting from PSP**

Figure 3.3: Example for explaining the differences between the two calculating methods - earth (80% chance of being traversable, found to be penetrable 8 times out of the total of 10), tree (30%, 9 out of 30), sky (45%, 9 out of 20)

Intuitively, the difference between the two methods is that the former considers all classes as equal, while the latter pays attention to how often a class has been seen. More technically:

1. the first method determines the probability of an image to be penetrable, as the average of the traversability percentages for each color contained within. Figure 3.3, which contains brown, green and blue, has a probability of 51.67% being traversable.

$$(80\% + 30\% + 45\%) \div 3 = 51.67\%$$

2. the second method uses the number of times each color was found in traversable and in non-traversable images. It assumes that the likelihood of an image being traversable is in direct dependence of the number of times each color found within, is penetrable. In other words, it specifies that the traversability percentage of an image, is the weighted average of the traversability percentage of all the colors within it. This time, the probability of Figure 3.3 being traversable is 43.33%.

$$(8 + 9 + 9) \times 100\% \div (10 + 30 + 20) = 43.33\%$$

The first method is an obvious way to find the probability of an image being penetrable. The second one, however, emphasizes on statistics, in the sense that a one-time event may be a coincidence, but the more often it happens the more secure it is to become a rule. In the previous example class earth was found to be traversable 8 times out of the total of 10. This means that 2 out of 10 times, it was found to be non-traversable. Let's suppose the first given image containing earth was non-traversable. So, the traversability percentage of earth would be 0%. While the first method will use it as a certainty, the second one will have low confidence on it.

It is known in advance whether the testing images are penetrable. It is trivial to discover whether the previously described methods' decisions are right or wrong. In the example above the first method decides that the image is non-traversable, while the second method determines the opposite. Note that in other cases the results of the two methods may concur.

As we have introduced a program that decides whether the input images are traversable or not, we will proceed in the next section to evaluate the effectiveness of our approach.

4. EXPERIMENTAL VALIDATION AND COMPARISON

In this chapter we will evaluate our program. We will describe the model-validation technique we use, present the obtained results and discuss them.

4.1. Evaluation setup

We evaluate our machine learning model with a procedure called *cross validation* [62]. It is also known as *rotation estimation* or *out-of-sample testing*. Cross validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions. In a prediction problem, a model is usually given two datasets. First a dataset of known data on which training is run (*training set*). And then a dataset of unknown data (or first seen data) against which the model is tested (*validation set* or *testing set*). The goal of cross validation is to test the model's ability to predict data not used during the training of the model.

The procedure has a single parameter N that refers to the number of folds that a given data sample is to be split into. As such, the procedure is often called *N-fold cross validation*.

In this thesis our data sample consists of twenty directories containing images. Ten of these directories contain 778 traversable images and the other ten include 945 non-traversable. For more information about how we gathered these datasets, see Appendix C.

In order to evaluate our model we split our data sample in five folds and perform 5-fold cross validation. Because we have twenty directories, each fold has an equal number of four directories. During each fold we use two traversable and two non-traversable directories for testing and the rest for training. In Table 4.1 one can see the numbers of images used for training and testing for each fold.

Let's see next the evaluation results of our model.

4.2. Results

As mentioned in Section 3.2 we implemented a program that predicts whether a scene is traversable or not. It accepts as input images outputted by a neural network for semantic

Table 4.1: Number of images for each testing and training set in 5-fold cross validation. Total number of images: 778 traversable and 945 non-traversable.

Fold	Traversable images		Non-traversable images	
	Testing set	Training set	Testing set	Training set
1	153	625	184	761
2	169	609	221	724
3	164	614	225	720
4	109	669	165	780
5	183	595	150	795

segmentation. In Section 3.1 we established that the ADE20K dataset is the best fit for whole scene semantic segmentation. And that, in theory, any of the

1. fully convolutional network (FCN),
2. pyramid scene parsing network (PSP) and
3. DeepLab

can be used equally effectively. So we run our program with images that have emerged by each one of them to check which has the best results.

We gave as input to each of the three neural network models the images contained in the directories mentioned in Section 4.1. And then fed our program with their outputs. The results of the first calculating method, described in Section 3.2 are shown in Table 4.2. And the results of the second method in Table 4.3. In both tables it is quite obvious that the PSP has clearly better results than the other two models, for both individual and global trials.

Table 4.2: 1st calculating method - Percentage of evaluation images whose traversability was found correctly. In the second column are the results from images derived from the FCN, the third from PSP and the forth from DeepLab.

Success on 1st calculating method			
Fold	with FCN (%)	with PSP (%)	with DeepLab (%)
1	57.57	77.45	56.97
2	65.9	94.36	56.67
3	66.58	87.15	58.87
4	84.31	86.86	72.26
5	75.98	94.89	75.08
Average	69.3	88.33	63.26

Table 4.3: 2nd calculating method - Percentage of evaluation images whose traversability was found correctly. In the second column are the results from images derived from the FCN, the third from PSP and the forth from DeepLab.

Success on 2nd calculating method			
Fold	with FCN (%)	with PSP (%)	with DeepLab (%)
1	54.6	62.61	54.6
2	57.69	93.33	56.67
3	57.84	80.46	58.35
4	63.5	79.2	60.58
5	45.05	45.05	45.05
Average	55.6	72.84	55.02

Consequently, we decided to continue our research using the PSP model. In Table 4.4 we gathered the results from both methods when using data from the PSP algorithm. Even thought we think that the second calculating method is more representative of the overall sample, in this case the first one gives better results. That happens probably because the non-traversable images of the training set are much more that the traversable ones, as shown in Table 4.1.

Table 4.4: PSP - Percentage of testing images whose traversability was found correctly with the 1st and the 2nd calculating method.

PSP trained on ADE20K dataset		
Fold	Success with 1st method (%)	Success with 2nd method (%)
1	77.45	62.61
2	94.36	93.33
3	87.15	80.46
4	86.86	79.2
5	94.89	45.05
Average	88.33	72.84

Therefore, as is obvious, we chose to deepen into the first method of deciding on image traversability. In Table 4.5 we describe how the success rates of the first method arose.

- In column 1 each fold is shown.
- In columns 2 and 3 we give the number of traversable images and the percentage of those that were successfully predicted traversable.
- In columns 3 and 4 the same for non traversable images.
- Finally, column 5 gives the total success rate of the first method, summarizing the results of both traversable and non-traversable images.

Table 4.5: PSP 1st method - Number of traversable and non-traversable images for each testing set, with their success rate in finding traversability. Observe that the success rate for non-traversable images is always 100%, while in traversable images is much lower.

Testing images on 1st method with PSP					
Fold	Traversable		Non-traversable		Average (%)
	Number	Success (%)	Number	Success (%)	
1	153	50.33	184	100	77.45
2	169	86.98	221	100	94.36
3	164	69.51	225	100	87.15
4	109	66.97	165	100	86.86
5	183	90.71	150	100	94.89
Average	778	74.16	945	100	88.33

4.3. Discussion

In Table 4.5 we can see that the prediction for non-traversable images is always correct. The foresight on penetrable images is not equally great though. Also one can easily observe that fold 1 has the worst results compared to the other folds. Followed by fold 4 and fold 3. But why does this happen? Why non-traversable images are easier to predict than traversable ones? And why does each fold have different success rate? These are some questions we will try to answer in this section.

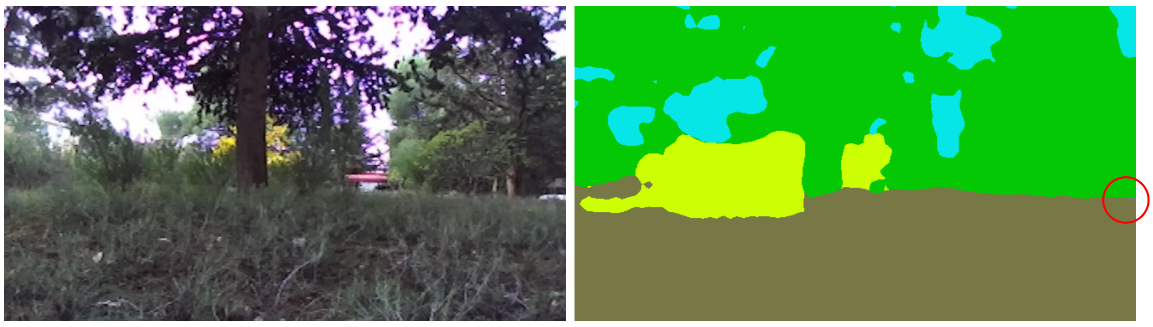
In Tables 4.8 we have gathered all class labels appearing in each one of the traversable images for validation. And in Tables 4.9 same thing for the images perceived as obstacles. Each subtable contains the information for each fold.

- Column 1 mentions all class labels appearing in the current fold.
- Column 2 keeps track of the number of each label appearances in the current fold (in how many images is the color participating?).
- In column 3 appears the traversability percentage of each label, after the system is trained with the images in the remaining directories (after training, how likely is for this color to be traversable?).
- Finally, column 4
  - shows all classes with traversability percentage $>$ 50%, in Tables 4.8, and
  - those with $\leq$ 50% chance of being traversable, in Tables 4.9.

Have in mind that in this work we do not care about the annotations representing each color within the image. These labels are available to facilitate reading. We do not stick to the fact that they do not always represent what is depicted in the image. This thesis is not about judging the proper functioning of the neural network. It is about estimating the traversability of images resulting from the neural network. More technically, similar and often confused objects tend to be similarly traversable or non-traversable. So, any mis prediction of the neural network does not affect the proper operation of our program.

As mentioned in the beginning of this section, we can see in Table 4.5 that the predictions for non-traversable images are much better than those for traversable images. Let's try and find out why. Let's take for example class labels **sky**, **earth** and **tree**. These three classes appear in most, if not in all, of the testing images. In Tables 4.8 (folds 1, 2, 3 and 5) the instances of these classes are equal to the number of the traversable images. And in Table 4.8d are only slightly fewer. The probability of each of them being traversable is between 43 and 48%. So **sky**, **earth** and **tree** are treated as impenetrable. And therefore contribute to seeing the image in which they belong as non-traversable. Same thing happens for the non-traversable images. Class **tree** appears in all images in Tables 4.9 and **sky** and **earth** in most of them.

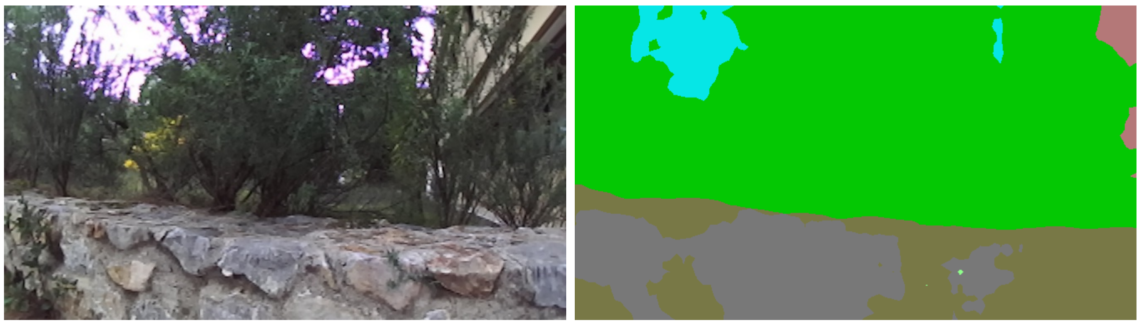
To visualize it, we give a traversable and a non-traversable image in which these classes appear. The ground in Figure 4.1 is translated as **earth**. Similarly, the short wall that makes Figure 4.2 impenetrable is recognized from PSP as a mix of **earth** and **wall**. In both cases we can see that the class **tree** is on the background. Both images are found to be non-traversable. The difference is that this prediction is correct for the second image but wrong for the first one.



(a) Original traversable image **(b) Image resulting from PSP**

Figure 4.1: Class labels tree, sky, earth, plant, wall (in red circle) in a traversable image that was incorrectly predicted.

As humans, we are able to identify earth as traversable and tree as an obstacle. The sky does not belong to either of the two categories. On the contrary, our model judges traversability depending on the occurrence of objects in traversable and non-traversable images. But the traversable images of the training sets are much fewer than the non-traversable, by at least 100 images on each fold (columns 3 and 5 on Table 4.1). Therefore, since these classes exist in almost all of the images (penetrable or not), they end up being perceived as impenetrable (traversability percent $\leq 50\%$). And thus this leads to a



(a) Original non-traversable image **(b) Image resulting from PSP**

Figure 4.2: Class labels tree, sky, earth, wall in a non-traversable image that was correctly predicted.

greater likelihood for traversable images to be found as non-traversable.

The more the non-traversable classes, the fewer the chances for an image to be found as traversable. Tables 4.6 and 4.7 show the numbers. In these tables we count the traversable classes in the traversable images (from Tables 4.8) and the non-traversable classes in the non-traversable images (from Tables 4.9) for each fold. And we present which percentage of the total classes is penetrable in the former case and impenetrable in the latter. The best case scenario for traversable images is fold 5 in which, however, only 50% of the classes included are correctly identified. While the worst case scenario for non-traversable images is fold 2 with 73.33% of the classes included to be correctly identified. Having that in mind one can understand why traversable images have a much lower percentage of success in being identified than the non-traversable.

Table 4.6: Percentage of classes in traversable testing images found to be traversable, for each fold.

Traversable testing images			
Fold	Traversable classes	All classes	Success (%)
1	4	13	30.77
2	4	9	44.44
3	4	10	40
4	3	9	33.33
5	6	12	50

Let's take Figure 4.1 for example. This image belongs to the testing set of fold 1. It contains the classes **plant**, **tree**, **sky**, **earth** and **wall**. According to the first calculating method, its traversability result is computed from the average of the sum of the traversability rates of its classes. Specifically, by adding the traversability rates of the above classes from

Table 4.7: Percentage of classes in non-traversable testing images found to be non-traversable, for each fold.

Non-traversable testing images			
Fold	Non-traversable classes	All classes	Success (%)
1	9	10	90
2	11	15	73.33
3	12	14	85.71
4	10	12	83.33
5	9	11	81.82

Table 4.8a and dividing the result with the number of these classes:

$$\frac{73\% + 45\% + 46\% + 46\% + 24\%}{5} = 46.8\% \tag{4.1}$$

So Figure 4.1 has a 46.8% probability to be traversable. Which is lower than 50% and therefore is declared as non-traversable.

Now that we have seen why non-traversable images are easier to predict than traversable ones, let's concentrate on why each fold have different success rate for the traversable images.

As established before, fold 1 has worse results than all the other folds (see Table 4.8a). First of all notice that the class label **house** appears in 10 out of the 153 traversable images for testing in fold 1. This class is missing from all the images in the training set. In this case our model does not count the specific class neither as traversable nor as non-traversable and completely ignores it. As can be inferred from the other Tables 4.8 class **house** only appears in the images of the testing set of fold 1 and is found to be 100% traversable. But it is ignored when calculating the traversability of the images containing it. And therefore this class does not contribute to the final result.

As can be seen in Table 4.6 fold 1 has the fewer traversable classes in proportion to the rest folds, followed by fold 4 and fold 3. And many validation images in fold 1 contain classes with very low traversability rate. For example, class label **floor** exists in 19 images and has 0% traversability rate. Class **building** has 101 instances with only 8% traversability rate and **wall** has 97 with 24%. And these are not the only three classes with many instances in the testing images and low traversability percentage (Table 4.8a).

Figure 4.1 belongs to the testing set of fold 1. Even though this image basically contains the classes **tree**, **sky**, **earth** and **plant**, PSP detected a small area of **wall**. Trees, sky and earth have all been decided to be impermeable. Nevertheless having only the first four classes, the image would be presumed as permeable, with 52.5% traversability percent (equation 4.2). But adding the extra class **wall** the rate reaches only 46.8% (equation 4.1), resulting to name the image non-traversable.

$$\frac{73\% + 45\% + 46\% + 46\%}{4} = 52.5\% \tag{4.2}$$

So what can be done in order to improve the success rate of recognizing the traversable images? In real life, the environment is not evenly distributed in penetrable and impenetrable areas. So, we cannot limit the number of the training images in a way so that traversable and non-traversable images are equal in number. Nonetheless, some ideas in how to achieve better results are discussed in the next chapter.

Table 4.8: Traversable images for testing - Labels, number of their instances and their traversability percentage as emerged from training. The last column shows all classes with traversability percentage $>$ 50%. (continuing in next page)

(a) Fold 1 (153 traversable images)

Fold 1			
Class label	Instances	Traversability (%)	Traversable
grass	52	100	✓
person	47	30	
plant	153	73	✓
tree	153	45	
signboard	11	18	
sidewalk	68	22	
path	60	100	✓
earth	**153**	**46**	
sky	153	46	
car	93	100	✓
building	101	8	
house	10	-	-
wall	**97**	**24**	
floor	19	0	

(b) Fold 2 (169 traversable images)

Fold 2			
Class label	Instances	Traversability (%)	Traversable
grass	1	100	✓
person	6	58	✓
plant	169	72	✓
tree	169	46	
flower	17	100	✓
earth	**169**	**46**	
sky	169	47	
rock	13	22	
house	-	100	✓
wall	13	30	

Table 4.8: (continued) Traversable images for testing - Labels, number of their instances and their traversability percentage as emerged from training. The last column shows all classes with traversability percentage $>$ 50%. (continuing in next page)

(c) Fold 3 (164 traversable images)

Fold 3			
Class label	Instances	Traversability (%)	Traversable
grass	18	100	✓
plant	161	73	✓
tree	164	46	
sidewalk	29	65	✓
path	22	100	✓
earth	**164**	**47**	
sky	164	47	
rock	27	20	
building	68	12	
house	-	100	✓
wall	72	28	

(d) Fold 4 (109 traversable images)

Fold 4			
Class label	Instances	Traversability (%)	Traversable
grass	49	100	✓
plant	86	84	✓
tree	109	46	
signboard	1	25	
earth	**83**	**48**	
sky	109	47	
rock	33	17	
house	-	100	✓
wall	21	30	
fence	2	100	✓

Table 4.8: (continued) Traversable images for testing - Labels, number of their instances and their traversability percentage as emerged from training. The last column shows all classes with traversability percentage $>$ 50%.

(e) Fold 5 (183 traversable images)

Fold 5			
Class label	Instances	Traversability (%)	Traversable
person	14	62	✓
plant	182	84	✓
tree	183	43	
flower	9	100	✓
signboard	17	13	
path	1	100	✓
earth	**183**	**43**	
sky	183	43	
car	117	100	✓
rock	99	10	
pole	11	-	-
house	-	100	✓
wall	100	23	
fence	94	100	✓

Table 4.9: Non-traversable images for testing - Labels, number of their instances and their traversability percentage as emerged from training. The last column shows all classes with traversability percentage $\leq$ 50%. (continuing in next page)

(a) Fold 1 (184 non-traversable images)

Fold 1			
Class label	Instances	Traversability (%)	Non-traversable
plant	1	73	
tree	184	45	✓
water	1	0	✓
rock	74	20	✓
building	184	8	✓
earth	**184**	**46**	✓
sky	183	46	✓
mountain	42	0	✓
wall	146	24	✓
floor	32	0	✓

(b) Fold 2 (221 non-traversable images)

Fold 2			
Class label	Instances	Traversability (%)	Non-traversable
person	2	58	
plant	1	72	
tree	221	46	✓
rock	219	22	✓
building	221	19	✓
windowpane	168	0	✓
earth	**195**	**46**	✓
sky	221	47	✓
mountain	47	0	✓
wall	137	30	✓
floor	192	7	✓
sidewalk	37	59	
skyscraper	43	0	✓
crt screen	48	0	✓
signboard	82	100	

Table 4.9: (continued) Non-traversable images for testing - Labels, number of their instances and their traversability percentage as emerged from training. The last column shows all classes with traversability percentage $\leq$ 50%. (continuing in next page)

(c) Fold 3 (225 non-traversable images)

Fold 3			
Class label	Instances	Traversability (%)	Non-traversable
plant	7	73	
tree	225	46	✓
water	1	0	✓
rock	218	20	✓
building	202	12	✓
windowpane	93	0	✓
earth	**220**	**47**	✓
sky	225	47	✓
mountain	8	0	✓
wall	225	28	✓
floor	70	5	✓
ceiling	1	-	-
sidewalk	68	65	
hill	51	-	-
skyscraper	97	0	✓
crt screen	36	0	✓
television	27	-	-
microwave	72	-	-

Table 4.9: (continued) Non-traversable images for testing - Labels, number of their instances and their traversability percentage as emerged from training. The last column shows all classes with traversability percentage $\leq$ 50%.

(d) Fold 4 (165 non-traversable images)

Fold 4			
Class label	Instances	Traversability (%)	Non-traversable
person	30	81	
plant	98	84	
tree	165	46	✓
rock	122	17	✓
building	165	18	✓
windowpane	94	0	✓
earth	**165**	**48**	✓
sky	165	47	✓
mountain	43	0	✓
wall	162	30	✓
floor	69	5	✓
computer	8	-	-
crt screen	35	0	✓

(e) Fold 5 (150 non-traversable images)

Fold 5			
Class label	Instances	Traversability (%)	Non-traversable
person	14	62	
plant	119	84	
tree	150	43	✓
rock	150	10	✓
building	150	18	✓
windowpane	68	0	✓
earth	**127**	**43**	✓
sky	122	43	✓
wall	140	23	✓
floor	66	5	✓
streetlight	20	-	-
crt screen	31	0	✓
plate	9	-	-

5. CONCLUSIONS AND FUTURE WORK

In this thesis we concentrated on traversability estimation methods. In Chapter 2 we explored many ways for a robot to autonomously decide the traversability of the terrain in front of it. We also found ways for it to be able to learn from the past and improve its traversability estimation. We investigated approaches to autonomously acquire labeled data and the sensors needed to do so. Finally we discussed about the most common strategies for the robotic platform to decide where to go next.

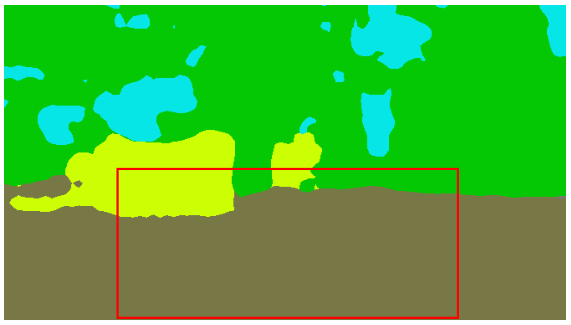
In Chapter 3 we implemented a program that when given images, predicts whether they are traversable or not. These images are extracted from a convolutional neural network for total scene semantic segmentation. We chose Pyramid Scene Parsing Network (PSP) pre-trained on ADE20K dataset for this purpose, as it seemed to have the best results. We fed PSP with images from natural outdoor environments, containing vegetation. It resulted with images consist of colors depicting objects and parts of the scene.

Lastly, in Chapter 4 we evaluated our implementation and explained how the traversable images are more difficult to predict than the non-traversable. The main reason for this is that the data sample used for training, consists of many more non-traversable than traversable images. That is something we are not able to control because we want our system to be autonomous. So what can be done in the future to improve the results?

The program can be modified to ignore specific classes. Possibly the classes that appear in a large amount on both traversable and non-traversable training images. For example, class labels earth, sky and tree seem to appear in almost all of the images used for training and validating the system (as can be seen in Tables 4.8, 4.9). All of them are predicted as non-traversable, because of the large number of non-traversable training images. However, as we know, earth is traversable, sky is neither of the two, and only trees are non-traversable.

Another way would be to concentrate on the part of the image that is more likely to be on the foreground. For example, see Figures 4.1a, 4.2a. Both of them contain the classes tree and sky. And in both cases these classes appear only on the background. An approach could be to calculate the image area that is accessible by the robot in its next move, according to its height and width. And then check whether this smaller area is traversable or not. A rough example of the minimized Figures 4.1b, 4.2b is shown in Figure 5.1. In other words, we suggest a way to perceive depth without the use of geometric data.

A further step, though, could be to actually combine appearance with geometric data. That way the robot would be able to visually recognize the environment around it, while at the


(a) Minimized Figure 4.1b

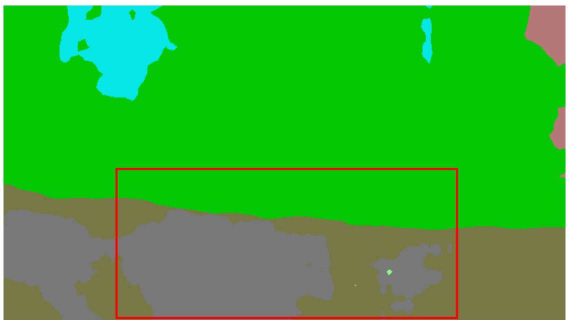

(b) Minimized Figure 4.2b

Figure 5.1: Two minimized images (one traversable and one non-traversable) so that the robotic platform fits marginally within their frame.

same time gather geometric information about it, like height or depth. We believe that with a combination like this, the results would be significantly improved.

However, we believe that our contribution has been noteworthy. As said before, it has become clear to researchers in robotics that current approaches are yielding systems with limited autonomy and ability for self-improvement. We managed to create a system that not only autonomously learns the traversability of its environment, but also has the capacity to self improve. With only a few modifications our program could be ready to be used on an actual robotic platform to explore its surroundings and improve its knowledge of traversability.

APPENDIX A. REFERENCE IMPLEMENTATIONS

Many authors published their code open-source so that other researchers may use their work. In Table A.1 we give some URLs and comments about the work mentioned above, in Section 2.1, Section 2.2 and Section 3.1. Some of them were actually used as our experimental basis.

Table A.1: References and their prototype implementations

Authors	URL
Droeschel et al. [5]	https://github.com/AIS-Bonn/mrs_laser_map
Hirose et al. [17]	https://github.com/NHirose/GONET
Goodfellow et al. [18]	http://www.github.com/goodfeli/adversarial
Krizhevsky et al. [26]	https://github.com/Abhisek-/AlexNet
Simonyan and Zisserman [25]	http://www.robots.ox.ac.uk/~vgg/research/very_deep/
Szegedy et al. [33]	https://github.com/google/inception
Chollet [34]	https://github.com/kwotsin/TensorFlow-Xception
He et al. [28]	https://github.com/KaimingHe/deep-residual-networks
Iandola et al. [35]	https://github.com/DeepScale/SqueezeNet
Howard et al. [36]	https://github.com/Zehaos/MobileNet
Girshick et al. [37]	https://github.com/rbgirshick/rcnn
Redmon et al. [38]	https://pjreddie.com/darknet/yolo/
Liu et al. [39]	https://github.com/weiliu89/caffe/tree/ssd
Long et al. [43]	https://github.com/shelhamer/fcn.berkeleyvision.org
Le [41]	https://github.com/udacity/CarND-Semantic-Segmentation/
Yang et al. [54]	https://github.com/DeepMotionAIResearch/DenseASPP
Oršić et al. [55]	https://github.com/orsic/swiftnet
Zhou et al. [59]	https://github.com/CSAILVision/sceneparsing
Zhou et al. [44]	https://github.com/CSAILVision/semantic-segmentation-pytorch
Chen et al. [61]	https://github.com/tensorflow/models/tree/master/research/deeplab
Zhao et al. [60]	https://github.com/hszhao/PSPNet

In the first three links appear the available source code of some researchers dealing with traversability estimation methods. Their work is described in Chapter 2.

The next seven URLs give the source code of the most popular convolutional neural networks for object classification and object localization, participated in ILSVRC. They are described in Section 2.2.1.

The next five link to the available code of other very popular convolutional neural networks for object detection and semantic segmentation. They are also described in Section 2.2.1.

The next two are used for semantic segmentation in street scenes. They are mentioned in Section 3.1.

The last four URLs are pre-trained convolutional neural networks for semantic segmentation. They are described in Section 3.1 and used in this work for experimentation. The last one, from Zhao et al. [60] was actually used for the purposes of this thesis (see Section 3.2).

APPENDIX B. DATASETS

In Table B.1 one can find some of the available datasets used for neural network training and evaluation. For our system we used ADE20K dataset to perform total scene semantic segmentation (see Chapter 3).

Table B.1: Datasets and their URLs

Dataset	URL
ADE20K [59]	http://groups.csail.mit.edu/vision/datasets/ADE20K/
ImageNet [31]	http://www.image-net.org/
Places [63]	http://places.csail.mit.edu/
COCO [45]	http://cocodataset.org/
Cityscapes [58]	https://www.cityscapes-dataset.com/
KITTI [56]	http://www.cvlibs.net/datasets/kitti/
PASCAL VOC [57]	http://host.robots.ox.ac.uk/pascal/VOC/

APPENDIX C. COLLECTING DATA

In order to have the robot autonomously fine-tune its traversability estimation, it is essential to be able to measure the traversability of a path after traversing it or having failed to traverse it [64]. One key concept put forward is that traversability can be measured as the "*error in proprioceptive localization*". That is, in localization that relies on the wheel encoder and IMU signals.

The rationale is that along an easily traversable path, encoder drift is small and when fused with IMU becomes negligible. The more difficult a path is, the more the wheels drift and this error increases. In order to calculate this error, proprioceptive localization is compared against the full 3D localization that fuses the encoders, IMU, and stereoscopic camera signals.

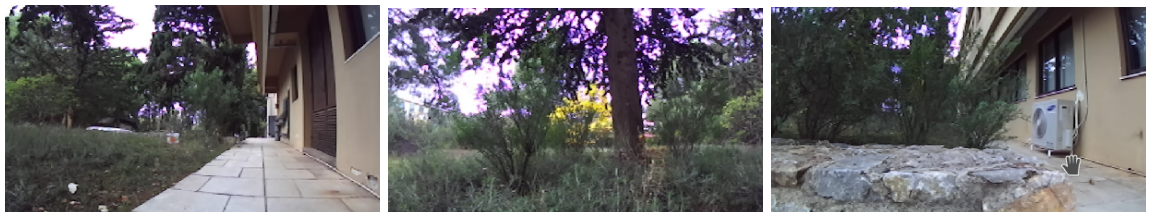

(a) Paved path **(b) Vegetation** **(c) Wall**

Figure C.1: Indicative scenes for the localization error experiments.

In order to prove this concept, several locations were selected with varying degrees of traversability ranging from paved path, to vegetation, to a wall. The vegetation appears as an obstacle in the 3D point cloud created by the stereo camera but can be pushed back and traversed. Figure C.1 shows some indicative examples.

The following experiment was executed using a DrRobot Jaguar rover fitted with a Zed stereoscopic camera. After using standard ROS [1] navigation to approach an obstacle, the robot synchronizes a secondary proprioceptive localization module with the main localization it uses to navigate. Then it circumvents normal obstacle avoidance to push against the obstacle. The velocity is the minimum velocity that the robot can obtain, guaranteeing the safety of the platform even when pushing against a wall.

The experiments have been carried out were at varying locations and approaching the obstacle from different angles. There was empirically found that for this specific platform and these specific sensors a localization error of 21cm or less signifies that the path is

[1] *Robot Operating System (ROS)* (`http://www.ros.org`) is an open source, meta-operating system for robots. It is a set of software libraries and tools that help in building robot applications.

traversable. This simple rule achieved an accuracy of 9/10 in both the traversable and the non-traversable tests. The specific threshold is likely to be different for different sensors. It would probably need to be empirically identified for each robotic platform and mix of sensors.

REFERENCES

[1] Benjamin Suger, Bastian Steder, and Wolfram Burgard. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3941–3946, 2015.

[2] Emre Ugur and Erol Şahin. Traversability: A case study for learning and perceiving affordances in robots. *Adaptive Behavior*, 2010.

[3] Panagiotis Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Engineering Applications of Artificial Intelligence*, pages 1373–1385, 2013.

[4] Jean-François Lalonde, Nicolas Vandapel, Daniel F. Huber, and Martial Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *Journal of Field Robotics*, 2006.

[5] David Droeschel, Max Schwarz, and Sven Behnke. Continuous mapping and localization for autonomous navigation in rough terrain using a 3d laser scanner. *Robotics and Autonomous Systems*, pages 104–115, 2017.

[6] Maggie Wigness, John G. Rogers, and Luis E. Navarro-Serment. Robot navigation from human demonstration: Learning control behaviors. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1150–1157, 2018.

[7] Honggu Lee, Kiho Kwak, and Sungho Jo. An incremental nonparametric bayesian clustering-based traversable region detection method. *Autonomous Robots*, pages 795–810, 2017.

[8] Dongshin Kim, Jie Sun, Sang Min Oh, James M. Rehg, and Aaron F. Bobick. Traversability classification using unsupervised on-line visual learning for outdoor robot navigation. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 518–525, 2006.

[9] Jahanzaib Shabbir and Tarique Anwer. A survey of deep learning techniques for mobile robot applications. *arXiv preprint arXiv:1803.07608*, 2018.

[10] Michael Shneier, Tommy Chang, Tsai Hong, Will Shackleford, Roger Bostelman, and James S. Albus. Learning traversability models for autonomous mobile vehicles. *Autonomous Robots*, page 69–86, 2008.

[11] Patrick Pfaff, Rudolph Triebel, and Wolfram Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 2007.

[12] In So Kweon and Takeo Kanade. High-resolution terrain map from multiple sensor data. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pages 278–292, 1992.

[13] Hans P. Moravec and Alberto Elfes. High resolution maps from wide angle sonar. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 116–121, 1985.

[14] François Denis, Rémi Gilleron, and Marc Tommasi. Text classifation from positive and unlabeled examples. *9th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, 2002.

[15] Charles Elkan and Keith Noto. Learning classifiers from only positive and unlabeled data. *14th International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 213–220, 2008.

[16] Rudolph Triebel, Patrick Pfaff, and Wolfram Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2276–2282, 2006.

[17] Noriaki Hirose, Amir Sadeghian, Marynel Vázquez, Patrick Goebel, and Silvio Savarese. Gonet: A semi-supervised deep learning approach for traversability estimation. *arXiv preprint arXiv:1803.03254*, 2018.

[18] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *27th International Conference on Neural Information Processing Systems (NIPS)*, pages 2672–2680, 2014.

[19] Oleksii Zhelo, Jingwei Zhang, Lei Tai, Ming Liu, and Wolfram Burgard. Curiosity-driven exploration for mapless navigation with deep reinforcement learning. *arXiv preprint arXiv:1804.00456*, 2018.

[20] S. Z. Li. Markov random field models in computer vision. *European Conference on Computer Vision (ECCV)*, pages 361–370, 1994.

[21] Noriaki Hirose, Amir Sadeghian, Patrick Goebel, and Silvio Savarese. To go or not to go? a near unsupervised learning approach for robot navigation. *arXiv preprint arXiv:1709.05439*, 2017.

[22] Lei Tai, Shaohua Li, and Ming Liu. Autonomous exploration of mobile robots through deep neural networks. *International Journal of Advanced Robotic Systems (IJARS)*, 2017.

[23] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015.

[24] Fjodor Van Veen. The neural network zoo, 2016. URL `http://www.asimovinstitute.org/neural-network-zoo/`.

[25] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *25th International Conference on Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[27] Adit Deshpande. The 9 deep learning papers you need to know about (understanding cnns part 3), 2016. URL `https://adeshpande3.github.io/The-9-Deep-Learning-Papers-You-Need-To-Know-About.html`.

[28] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2016.

[29] Adit Deshpande. A beginner's guide to understanding convolutional neural networks, 2016. URL `https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/`.

[30] Li-Jia Li, Richard Socher, and Li Fei-Fei. Towards total scene understanding: Classification, annotation and segmentation in an automatic framework. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2036–2043, 2009.

[31] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[32] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, pages 211–252, 2015.

[33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[34] François Chollet. Xception: Deep learning with depthwise separable convolutions. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1251–1258, 2017.

[35] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and $<$0.5mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[36] 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. *arXiv preprint arXiv:1704.04861*, 2017.

[37] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 580–587, 2014.

[38] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[39] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. *European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[40] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, pages 154–171, 2013.

[41] James Le. How to do semantic segmentation using deep learning, 2018. URL `https://medium.com/nanonets/how-to-do-image-segmentation-using-deep-learning-c673cc5862ef`.

[42] DeLiang Wang. Visual scene segmentation. *Handbook of Brain Theory and Neural Networks*, pages 1215–1219, 2003.

[43] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[44] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, pages 302–321, 2019.

[45] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context. *European Conference on Computer Vision (ECCV)*, pages 740–755, 2014.

[46] Vikas Gupta. Keras tutorial : Using pre-trained imagenet models, 2017. URL `https://www.learnopencv.com/keras-tutorial-using-pre-trained-imagenet-models/`.

[47] Adrian Rosebrocke. Imagenet: Vggnet, resnet, inception, and xception with keras, 2017. URL `https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/`.

[48] Noriaki Hirose, Amir Sadeghian, Fei Xia, Roberto Martin-Martin, and Silvio Savarese. Vunet: Dynamic scene view synthesis for traversability estimation using an rgb camera. *IEEE Robotics and Automation Letters*, pages 2062–2069, 2019.

[49] Pierre Sermanet, Raia Hadsell, Marco Scoffier, Matt Grimes, Jan Ben, Ayse Erkan, Chris Crudele, Urs Muller, and Yann LeCun. A multi-range architecture for collision-free off-road robot navigation. *Journal of Field Robotics*, 2009.

[50] Raia Hadsell, Pierre Sermanet, Jan Ben, Ayse Erkan, Marco Scoffier, Koray Kavukcuoglu, Urs Muller, and Yann LeCun. Learning long-range vision for autonomous off-road driving. *Journal of Field Robotics*, 2009.

[51] Christopher J. Holder, Toby P. Breckon, and Xiong Wei. From on-road to off: Transfer learning within a deep convolutional neural network for segmentation and classification of off-road scenes. *European Conference on Computer Vision (ECCV)*, pages 149–162, 2016.

[52] A. Bosch, X. Muñoz, and J. Freixenet. Segmentation and description of natural outdoor scenes. *Image and Vision Computing*, pages 727–740, 2007.

[53] Clément Farabet, Camille Couprie, Laurent Najman, and Yann LeCun. Learning hierarchical features for scene labeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pages 1915–1929, 2013.

[54] Maoke Yang, Kun Yu, Chi Zhang, Zhiwei Li, and Kuiyuan Yang. Denseaspp for semantic segmentation in street scenes. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3684–3692, 2018.

[55] Marin Oršić, Ivan Krešo, Petra Bevandić, and Siniša Šegvić. In defense of pre-trained imagenet architectures for real-time semantic segmentation of road-driving images. *arXiv preprint arXiv:1903.08469*, 2019.

[56] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361, 2012.

[57] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, pages 303–338, 2010.

[58] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016.

[59] Bolei Zhou, Hang Zhao, Xavier Puig, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Scene parsing through ade20k dataset. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 633–641, 2017.

[60] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[61] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, pages 834–848, 2018.

[62] Jason Brownlee. A gentle introduction to k-fold cross-validation, 2018. URL https://machinelearningmastery.com/k-fold-cross-validation/.

[63] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. *27th International Conference on Neural Information Processing Systems (NIPS)*, pages 487–495, 2014.

[64] Personal communication with Katerina Maria Oikonomou. *Institute of Informatics and Telecommunications, NCSR ”Demokritos”*.