THESIS ON MECHANICAL ENGINEERING E72

Long-Range Navigation for Unmanned Off-Road Ground Vehicle

ROBERT HUDJAKOV

TUT PRESS

TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Mechanical Engineering
Department of Mechatronics

Dissertation was accepted for the defense of degree of Doctor of Philosophy in Engineering on December 12, 2012.

Supervisor: Prof. Mart Tamre
Department of Mechatronics, Tallinn University of Technology

Opponents: Prof. Petri Kuosmanen
Aalto University, Finland

Prof. Johannes Steinbrunn
Hawassa University, Ethiopia

Prof. Jüri Vain
Tallinn University of Technology, Estonia

Defense of the thesis: January 16, 2013

Declaration:

Herby I declare that this doctoral thesis, my original investigation and achievement, submitted for the doctoral degree at Tallinn University of Technology has not been submitted for any academic degree.

Robert Hudjakov

ISSN 1406-4758
ISBN 978-9949-23-396-0 (publication)
ISBN 978-9949-23-397-7 (PDF)

MEHHANOTEHNIKA E72

Kaugmaa navigatsioonisüsteem maastikuvõimekusega autonoomsetele liikuritele

ROBERT HUDJAKOV

TTÜ
KIRJASTUS

Contents

Abbreviations

UGV	Unmanned Ground Vehicle
UAV	Unmanned Aerial Vehicle
LIDAR	Light Detection And Ranging
GIS	Geographic Information System
ANN	Artificial Neural Network
CPU	Central Processing Unit
GPU	Graphics Processing Unit
API	Application Programming Interface
SDNN	Space Displacement Neural Network
FPGA	Field-Programmable Gate Array
DARPA	Defense Advanced Research Projects Agency
LAGR	Learning Applied to Ground Robots
A*	A-star algorithm
D*	D-star algorithm

Introduction

The last decade has seen introduction of multiple unmanned ground vehicles (UGV) for both road and off-road use. The development of road driving autonomous vehicles fueled by DARPA Grand Challenges has led to usable prototypes with more than 300 000 accident free driving miles on public roads [1]. The Head of General Motors research division, Alan Taub, has promised self-driving cars to be in production by 2020 [2]. The progress of off-road capable UGV systems has been modest. There is a need for off-road capable UGV in both military and civil applications: autonomous environment monitoring, load bearing or as a platform for complex machinery. A project started by Estonian Defense Forces with an intention to build an off-road capable UGV platform supported by UAVs serves as the motivation of the current thesis.

Navigating in an unstructured off-road environment leads to a set of challenges not present in structured environments, such as compressible obstacles (tall grass, bushes) and negative obstacles (ditches). Due to the changing nature of unstructured environments the availability of up to date navigation maps is limited, forcing UGV to rely on the onboard navigation system that is restricted by a vehicle's perception range. In addition, the existing navigation maps will be rendered useless in crisis situations that introduce a significant amount of obstructions over large areas, such as hurricanes, earthquakes or military conflicts. The limited climbing ability of UGV robots forces them to wander in search of passage.

The autonomous navigation capability of an off-road UGV is hindered by the short perception range of its onboard sensors. The direct perception range of an UGV is mostly limited by its height and sensor quality. Reliable perception distance of stereo cameras is in the order of magnitude of 10 m, the distance can be increased to 100 m by guesswork. LIDARs have higher perception distance, but they too are limited by height of the sensor. Obstacle detection using stereo vision or LIDAR alone gives insufficient information for navigation in natural environments. Thus, it is required to understand the surroundings. Compressible obstacles, such as grass or mud, demand extra sensors to be detected. In addition to finding the obstacles, the UGV must categorize them based on their properties – this capability can be utilized for developing a long-range navigation system.

The aim of this study is to reduce wandering in unknown terrain by generating ad hoc navigation maps using overhead imagery and data gathered by an UGV. We propose a system that can be used for map generation and path planning on an unknown terrain using orthorectified UAV or satellite imagery. The system learns from labels applied by the UGV and extrapolates the gathered knowledge on overhead imagery onto a wider area, effectively improving the perception range of the robot beyond immediate range of its onboard sensors. The created ad hoc map will help to shorten UGV path, to reduce its energy consumption, to increase average speed and – most importantly – to reduce breakdowns.

Before an UGV mission begins, an UAV makes a pass over the designated target area returning fresh terrain imagery; optionally satellite imagery can be used. An UGV operator will then be able to use the imagery to predict time and UGV energy requirements for a predefined mission. If the full mission is not attainable with the current UGV battery charge, the operator can choose between aborting the mission or adjusting its objectives.

After the UGV is dispatched, it has to navigate autonomously through a set of waypoints using onboard sensors for obstacle avoidance and the long-range navigation system for route selection. The long-range system is adaptive to new input and is capable of learning from the new data gathered by the UGV as it navigates through the waypoints. This adaptiveness allows the UGV to generate maps in unknown environments and to navigate in changing environments such as flooding or forest fires.

Main Objective of the Thesis

Scientific Objectives

- Development of a long-range navigation method for an off-road capable UGV that utilizes aerial and satellite imagery
- Development of an aerial imagery classifier using deep learning approach
- Development of a self-assessment algorithm for graceful failure

The objective of the thesis is to develop an intelligent long-range navigation method for an off-road capable UGV that utilizes monocular aerial or satellite imagery. The navigation system must be able to cope with unknown and changing terrain. The off-road environment is both unstructured and unpredictable, requiring the navigation system to be adaptive, able to learn new kinds of obstacles on the go. The environment is also changing (flooding, forest fires); the navigation system must be able to reevaluate the traversability of terrain and update the generated route when fresh data arrives.

An extra constraint is that the system will be deployed on a small battery powered UGV. The analysis of large-scale terrain for long-range navigation can be computationally expensive. The size of the batteries, however, limits available processing capacity. The resulting navigation system must be computationally efficient to be viable.

Structure of the Thesis

The body of the thesis starts with a review of literature, which describes some previous studies in the field and in the fields related to this thesis. The review section is by no means exhaustive, it only refers to cherry picked works that are relevant to this thesis. A deficiency in an off-road UGV navigation system and current solutions are described. In the final part of the chapter problems

encountered in current solutions are summarized and arguments in favor of our alternative are proposed.

The review of the literature is followed by a section dedicated to theoretical work: it covers orthophoto analysis, cost map generation, path planning and a usage scenario. For orthophoto analysis we use a classifier that combines feature extraction and image classification. The first part of the theoretical section describes in detail the classifier and its properties that make this specific classifier particularly suitable for the task.

The following two chapters focus on cost map generation and path planning. The overhead imagery classifier takes a small pattern from the imagery as an input and returns a feature vector as an output. Each element of this vector represents a likelihood that a corresponding feature is detected on the input pattern. The cost map can be generated solely using the list of detected features, but for added robustness we use prior knowledge to extract the confidence of the classifier for each feature. At path planning the used method is briefly described and challenges that are unique to our system are discussed. Namely, the aerial imagery classification is a relatively expensive step. Reversing the processing pipeline and evaluating aerial imagery on need-to-know basis, increases the efficiency of the system in well defined environments.

To conclude the theoretical section we describe a usage scenario that explains how the system can be used for completing a mission. The time and energy requirements of the mission can be roughly estimated prior to the mission, using existing aerial imagery and a generic classifier, allowing the operator to adjust the objectives and the extent of the mission as needed. After the UGV is dispatched it will gather fresh information about the environment. Cyclic update mechanism allows the classifier to learn from the fresh data gathered by the UGV, improving its classification capability and enabling better path generation. The cyclic update keeps the system adaptive, able to recognize new obstacles and to reroute around those obstacles.

The theoretical section is followed by a practical part. We made a series of experiments using manually labeled aerial imagery from the Estonian Land Board database and satellite imagery from the Google Earth database. The main objective of these experiments was to analyze the capability of the system to perform on a known terrain and its ability to cope with an unknown terrain. At the end of the practical sections we describe the performance challenges of applying the system to overhead imagery covering large territories and propose a solution that utilizes heterogeneous computing hardware.

In the system analysis, first, the classifier is analyzed. The capability of the classifier is checked against a manually labeled dataset, on both structured and unstructured terrains, using aerial and satellite imagery. The measured results are further validated against a GIS database.

In addition to measuring the peak capability of the classifier we want to extract its confidence information. The confidence of the system can be

conveyed to the UGV – on high certainty clear areas it can use high speed movement profile, while on low certainty areas it should use caution and rely solely on local navigation capabilities. To obtain the confidence information we utilize prior knowledge – the classifier is executed on a dataset with known labels and an estimate of its capability is calculated from the measured results. To verify the classifiers capability of expressing its confidence we prepared intentionally weakened classifiers for both structured and unstructured terrain.

The classifier tests are followed with combined tests that include cost map generation and path planning. The features detected by the classifier, along with associated certainties, are converted to traversal costs, which are utilized by the path planner. The objective of the combined tests is to demonstrate the capability of the proposed system in both structured and unstructured environment. We demonstrate path planners' ability to generate a safe route from start to finish with both a well- and an under-performing classifier. These tests are set up to be easily verifiable – the generated path must follow a road avoiding obstacles even in low confidence scenarios.

The practical section is concluded by a short description of the developed software and assessment of system performance. Processing of a large overhead image with the proposed classifier on a CPU is a prohibitively expensive operation for a battery powered UGV (in terms of time and used energy). To overcome performance related challenges we explored different ways of utilizing heterogeneous computing hardware and present a solution that reduces the classification time and energy consumption by two orders of magnitude in comparison to a CPU implementation.

1. Review of the Literature

1.1 UGV Navigation

Majority of current UGV navigation systems use expert systems that rely on various pre-programmed criteria for distinguishing the features of the surrounding environment. The traversability of the features is evaluated and used for driving. The algorithms typically begin with ground level extraction, followed by detection of objects above the ground – those are classified either as obstacles or soft barriers. The worst drawback of the expert systems is the lack of flexibility, the robots that depend on manually pre-programmed algorithms tend to bog down in unstructured environments [3]. Furthermore, the reliable perception range of UGV mounted instruments that is severely limited by the jolting of the moving vehicle is not a helpful. The effective perception range of LIDAR sensors is reduced to the range of stereo cameras: below 30 meters.

In order to increase UGV flexibility DARPA (Defense Advanced Research Projects Agency) launched LAGR (Learning Applied to Ground Robots) project targeted at creating intelligent UGV navigation systems. The robots using intelligent navigation systems are capable of learning from the past experiences and from activities of instructors. Perhaps the ability of the intelligent robots to perceive the environment beyond the stereo range of the cameras is most important.

"Creating a machine that can navigate autonomously and intelligently outdoors and off-road is an enormous challenge whose difficulty is well appreciated by all those who try. While it is fairly simple to program a vehicle to go from a waypoint to a waypoint over a smooth ground, as soon as potential obstacles block the straight-line path, the task becomes much more difficult. The autonomous navigation system has to determine which objects in the path of the vehicle can be safely traversed and at what speed, and which objects need to be avoided completely" [3].

1.2 UGV and UAV Collaboration

The concept of fusing UAV and UGV sensors for navigation purposes was first proposed by Stentz *et al*. [4, 5] to discover obstacles that are hard to detect in time from a fast moving ground vehicle. A notable hazard is a negative obstacle, such as a pothole or a ditch, which is occluded to mobile vehicle cameras until the vehicle is close. They proposed to use a "flying eye" sensor that scouts ahead of the ground vehicle detecting hazards. Their motivation was to increase an off-road mobile vehicle autonomy from human operator during the DARPA funded PerceptOR program: "The PerceptOR program seeks to remedy the poor performance of extended tele-operation by introducing significant autonomous perception, reasoning, and planning onboard the UGV, while directed by a remote human operator who can assist the UGV when it is unable to determine the best course of action. By introducing this autonomy, it is expected that the frequency and intensity of

human involvement at the remote command post can be greatly reduced, enabling a single operator to control multiple vehicles and reducing the bandwidth required between the UGV and the operator" [4].

The flying eye gathers 3D geometry (point cloud acquired by LIDAR or stereo cameras) and geo-references it using pose estimation sensors. The geo-referenced 3D geometry is sent back to the UGV over wireless link and fused with the UGV onboard sensors for navigation. The data provided by the UAV was used to extract "penetrability" and "compressibility" information about terrain using stochastic algorithms – the penetrability correlates to availability of geometric obstacles and is softened by compressibility criteria. For example, tall grass is categorized as an obstacle by penetrability analysis but softened to traversable terrain by compressibility criteria.

They made four experiments to validate the concept [4]: UGV without prior knowledge, UGV with prior knowledge, UGV with outdated prior knowledge, and UGV with online UAV sensor. Comparison of the UGV mission with prior knowledge to the mission without it shows 40% reduction in the mission time and 20% reduction in the traveled distance. Even the mission with outdated prior knowledge (an obstacle was introduced on a desired path) was 10% shorter (distance and time) than the mission without prior knowledge. The comparison of mission with online flying eye and prior knowledge to the mission with only prior knowledge shows significant but not dramatic improvement in the traveled distance (Figure 6 in [5]) – usage of flying eye altered the traveled path when new obstacles were detected but after the obstacle was bypassed, the UGV returned to the predetermined path.

1.3 Terrain Classification Using Aerial 3D Point Cloud

Focus in terrain classification for an off-road unmanned ground vehicle is mainly on interpreting 3D point cloud returned by LIDAR or stereo cameras. An helicopter equipped with a LIDAR flying at height of 400m can acquire a point cloud with a density of 1 to 52 points per square meter with a range resolution of 1cm and point positional accuracy between 10 and 30 cm [6].

Stefanik *et al*. [7] used stereo vision for the 3D point cloud generation and for environment mapping. Unlike LIDAR solution, the stereo camera approach requires no helicopters to sweep the entire area. Stereo cameras can produce the point cloud even on a hovering vehicle, helping with vertical takeoff and landing. Stereo cameras produce a dense point cloud (2200 points/m^2) using cameras attached to the autonomous helicopter from a low height of 40 m. The cameras have a resolution of 1600x1200 pixels covering 846 m^2 area with each shot. The best case vertical accuracy of the resulting point cloud is approximately 0.6 m across the 33 m x 25 m area. The article speculates that the system can run in real time with 5 frames per second, consuming 10W when implemented on FPGAs.

Off-road terrain classification for navigation using 3D point cloud generally emphasizes extracting explicit features, such as ground surface, vegetation [8], manmade structures (buildings, roads) [9 – 11] or manmade

obstacles (concertina wire) [12]. Cost map for path planning generally considers ground smoothness (lack of trenches, large rocks and vertical walls) and vegetation penetrability (grass vs. trees). Load bearing (ground) surface extraction from an UAV point cloud is important for it is the primary input for ground vehicle path planning. For a high-speed run a smooth surface with a small slope angle is preferred and all trenches and walls should be avoided.

Vandapel *et al*. [13] have proposed two methods for ground surface extraction: multi echo based filtering and cone based filtering. Multi echo based filtering utilizes a property of a LIDAR – it returns multiple results when it measures sparse foliage. The results are clustered by height using k-means clustering and the lowest cluster is assumed to be a ground surface for any given area. Cone based filtering is based on a fact that volume below ground point must not contain any other point (important for distinguishing tree canopy from ground). Everything above ground is considered to be vegetation.

Secondary input in their work for cost map generation is "vegetationess" of the ground, defined as the ratio of data points above ground level to total data points in a given area. The vegetationess of terrain is related to the penetrability of terrain – grasslands have low vegetationess and are traversable for robot but dense bushes have higher vegetationess and are thus harder to penetrate. The total cost of each node on a cost map is the combination of traversing cost (from ground surface) and vegetationess; the uneven surfaces and dense foliage are avoided, resulting in safe path.

Generally, LIDAR data acquired by a flying sensor is classified in few discrete steps: the point cloud is divided into small regions in the scale of 1 square meter. From the point cloud that falls into the region, a set of parameters is extracted: maximum height difference between the points, standard deviation, average value, ground surface, etc. The parameters are then used as inputs for linear (such as k-means) [13, 14] or nonlinear (such as neural networks) [15 – 18] classifier. In addition to point cloud parameters, the classifier may also use other inputs, such as RGB values from cameras or LIDAR signal reflectance [10, 11, 15].

Terrain classification methods that directly classify each point in point cloud using its neighbors [19, 20] are infrequently used because of computing requirements they impose. In addition, the density of the point cloud is higher (some points are few centimeters apart) than essential for global path planning.

Using 3D point cloud for terrain classification has multiple drawbacks [7]. The price of LIDARs is high compared to cameras. LIDARs are heavier than cameras, especially those with suitable resolution, which makes them unsuitable for a small UAV. The point cloud produced by stereo cameras has high resolution, but has to be acquired from relatively low height to maintain the resolution and to keep the precision under control. For navigation it is preferable to acquire data about a wide area around a desired UGV path and this can only be done from a height of hundreds of meters at least.

The long-range navigation systems developed are intended for low hanging fruits and they utilize a high definition high precision 3D cloud for terrain classification, but the difficulty of acquiring the point cloud for a large area severely limits their usefulness. It is far more practical to use monocular imagery for navigation as it is more readily available and can be easily acquired. High resolution satellite imagery covering most of the world is freely available from Google and Bing maps databases or from commercial providers such as TerraServer. Relatively up-to-date aerial imagery can be downloaded from GIS databases. Fresh aerial imagery can easily be acquired by an UAV, the imagery can be updated during the mission to detect changes in the environment.

1.4 Aerial Image Classification

Using monocular aerial imagery, terrain is mostly classified in the context of automation of GIS database creation. Mayer reviews a long list articles [21] focused on building extraction and Mena [22] lists nearly 250 publications covering road extraction for GIS database update. However, GIS data extraction methods for navigation are non-adaptive. They rely on hand-crafted algorithms that are architected to find a very specific feature on aerial imagery. UGV navigation systems that rely on pre-programmed algorithms for feature detection fall apart in unstructured environments. Jackel *et al*. report, "The off-road world is so complex that it is virtually impossible to pre-program a vehicle to successfully deal with every environmental condition it might encounter. Unfortunately, most systems today tend to be preconfigured for the expected environment and cannot adapt to unforeseen circumstances. For example, suppose that a vehicle will be driving through a mature forest. In such an environment, much of the ground is either bare or littered with impassable downed trees and branches. If the vehicle is programmed to avoid all objects on the ground that are higher than 20 cm, it will be unable to traverse a sunlit clearing where there is compressible tall grass" [3].

Heidarsson *et al*. [23] use aerial imagery in HSV and CIELab color spaces for obstacle detection on water surface. The obstacles are used for unmanned surface vehicle navigation. They use a set of simple filters (average, deviation, edge detection and H channel entropy from HSV image) as feature extractors and a small fully connected neural network layer as a classifier. The classification is done pixel by pixel, the context of the pixel is largely discarded, e.g. the edge detection filter transmits the steepness of the edge but discards the shape of the edge. The neural network is trained using labels produced by a surface vessel, which uses sonars for obstacle detection. In overhead imagery, they use satellite data from Google and Bing maps database.

Heidarsson reported excellent classification capability (over 90%) on a calm lake and a calm harbor. The results are expected because the calm water is nearly featureless, especially, when compared to feature rich surrounding environment in a chosen experiment. Edge detection and standard deviation

filters return blank image for water area and light up on feature rich ground area, high classification results can be achieved by simply labeling featureless areas as water. The method falls apart when water surface contains features, such as wave crests or vegetation. Finding a good set feature extraction filters for a wide variety of environments and weather conditions is a difficult task, testing the filters in various conditions is even a more laborious task.

1.5 Cost Map Generation & Path Planning

In order to execute path planning, the features detected on aerial imagery have to be transformed into a form that enables comparison. For instance, if a path planner has to route a robot through an area that contains both sand and solid ground, then it is not sufficient to say that the solid ground is preferable for driving but it is also necessary to specify how much better the solid ground over sand is. The selected area is segmented into nodes for path planning, each node is assigned a "traveling cost". The job of the path planner is to route the UGV through the nodes so that the total traveling cost is minimized. The exact unit of the traveling cost depends on the objective: "energy" [16] can be used to increase power efficiency or "probability of getting stuck" [24] can be used to reduce risks, a mixture of both can be used for safe and efficient planning. Assigning 10 times higher traveling cost to sand than to solid ground usually means that the path planner is willing to make 10 m deroute on solid ground in order to avoid 1 m of sand.

The level of detail used during cost map generation varies from roughly predicting the terrain type [16, 24] to estimating the position of each UGV tire on top of the ground surface and calculating UGV pitch/roll [25]. The trend is to use multi level navigation systems where global planner roughly predicts the terrain traversability and generates a rough path to destination [26-28]. The short-range decision making on how to traverse each segment of the generated path is left on the local navigation system.

Once the cost map is generated, the path can be planned. A good search algorithm for fully classified areas is an A* algorithm [29], which is an extension to Dijkstra's algorithm [30]. The drawback of the A* algorithm is its performance cost in partially classified areas where the path has to be updated as new data becomes available. Stenz proposed a D* algorithm [31] to overcome the necessity to reevaluate the whole area when new information becomes available. The D* algorithm, however, is largely replaced by a simpler and more efficient D*-lite algorithm from Koenig *et al*. [32]. Gerkey *et al*. [33] claimed the D* algorithm to be slow for small scale path planning and suggested usage of a gradient based planner instead [34].

1.6 Synchronizing UAV and UGV Maps

One of the challenges in using aerial imagery for ground vehicle navigation is coherency of UGV local maps and overhead imagery. The UAV and UGV positions can not be estimated precisely because the inertial navigation systems tend to drift. The global navigation systems have errors caused by a

long list of sources, such as atmospheric effects, clock errors and signal reflections. Position estimate can be improved by using receivers that support Differential GPS, but the availability of DGPS stations is not guaranteed. The errors in UAV and UGV location and pose estimation cause mismatch between UGV local map and overhead imagery. The UGV local map and global map generated by overhead imagery must be synchronized, the most common technique is fitting of the maps using distinct key points from both maps.

In their survey paper, Martial *et al*. [35] cover localization and mapping solutions for both 3D input (point cloud) and 2D input (camera images). All the methods described focus on defining distinct signature points on both maps, finding a set of corresponding points and using them for defining coordinate transformation. In the case of 3D point cloud, a local neighborhood is used for computing the local shape descriptors or "signatures" [8]. In the case of 2D images a set of features or "landmarks" [36] that can be detected from various camera angles are recorded along with camera pose estimates. An alternative for fitting with 2D images that is especially suitable for overhead imagery is usage of a multi frame Structure From Motion (SFM) methods. The SFM interprets multiple images taken with a monocular camera at different positions as stereo imagery and uses it for point cloud generation. The generated point cloud can then be matched using techniques developed for 3D sensors. However, the point cloud generated using the SFM method is inferior in quality to stereo cameras and can not be used for terrain classification for navigation.

The major limiting factor in synchronizing the two maps is the computing capacity. Given large data sets, searching a corresponding match for each point in both maps is computationally infeasible. Choosing a good algorithm for signature point selection will greatly reduce the count of points to be matched, bringing the required computing capacity to manageable levels. After a preliminary correspondence between the two maps is established, the transformation can be improved by using detailed data.

The established transformation function between the two maps enables fusing data of the two. The maps are, however, not equal. The local map is more valuable in areas that are explored by a robot whilst global maps are probably a more reliable source for the rest of the area. Fusing of the maps with attached uncertainty information has been described by Elfes *et al*. [37] and Oriolo *et al*. [38].

1.7 Chapter Summary

Off-road UGV robots with the capability of environment labeling are available. However, their navigation system is short-sighted, which is a serious problem for an off-road capable UGV. The short-sightedness can be cured by using a flying eye sensor, the existing solutions rely on 3D point cloud provided by an UAV. There are two major problems with existing

solutions: they are rigid in their functionality and acquisition of 3D point cloud is difficult.

The rigidity in functionality manifests, in particular, in the inability to learn from experiences. The rigid, pre-programmed, algorithms do not work well in off-road terrain. It is practically infeasible to create algorithms that account for all the circumstances the off-road robot will meet. The fixed-functionality algorithms tend to break down in unstructured environments where the properties of obstacles are unpredictable. The adaptive algorithms that are taught by example or from experience are superior in natural environments.

The 3D point cloud is typically acquired either from relatively low height with stereo cameras or from high height using a LIDAR. Acquisition from low height poses collision risks to an UAV and limits the area that can be scanned with one shot due to occlusion by tall objects. The high flying sensors have much wider field of view and thus can examine a larger area with less scanning, but the required high-precision LIDARs are heavy and expensive. From technical point of view it is more convenient and cost-efficient to use a monocular camera mounted on a high-flying UAV, but the required image processing task is much harder to solve.

In addition to the flying sensor, alternative ways to acquire overhead imagery are available. There are existing databases for aerial imagery, for satellite imagery, for 3D point cloud, and for preclassified GIS data. The two problems with existing databases are freshness of information and level of detail. The fresh high-fidelity data is usually available only for cities and areas with high population density. The image quality of remote off-road areas is often lower, but the situation with aerial imagery is much better than with a LIDAR point cloud. The level of detail for classified data in GIS databases for off-road remote areas is meager at best. Among the choices, the overhead imagery has best coverage and is most often updated.

There is a need for an adaptive system capable of working with monocular overhead imagery that is able to learn.

2. Theoretical Foundations

2.1 Theoretical Background

The backbone of this thesis is the overhead imagery classifier, as it is the most difficult part of the puzzle to solve. The classifier must take an orthorectified aerophoto as an input, detect features on it and return some kind of map as an output. The map is then used by a cost map generator that prepares the data for path planning.

The aerial imagery provided by an UAV is georeferenced and orthorectified; it has been adjusted for shift, scale, camera tilt angle, lens distortion, and topographical relief of the terrain. Because there is a conversion from image coordinates to geometrical coordinates, the classifier can focus on images and work within image coordinates. The desired output of the overhead image classifier is a feature vector for each pixel in the image, each element of this feature vector expresses confidence of the classifier that the corresponding feature is detected at the given pixel position. Another way to represent the output is feature masks; for every feature in the feature vector a mask can be plotted.

Aerial imagery classification algorithms typically consist of two steps: *feature extraction* from input images and *linear classification* based on the detected features [39]. Feature extraction is a step where a list of parameters is extracted from an image, such as "spectral signatures, vegetation indices, transformed images, textural or contextual information, multitemporal images, multisensor images, and ancillary data" [39]. The linear classification step will then detect objects on the image based on the detected features. Our UGV must be off-road capable, i.e. the feature extractor algorithm must perform on images acquired from wildly varying environments and the linear classifier must be able to recognize a prohibitively large set of object categories based on detected features. Given the variety in unstructured terrain it is excessively difficult to build a universal aerial imagery classifier that is independently capable of analyzing aerial imagery without assistance from additional sensors.

To classify aerial imagery for navigation purposes we can lean on UGV onboard sensors; the UGV onboard navigation system must be capable of labeling the environment for path finding and obstacle avoidance. Transferring of the labels assigned by an UGV to aerial imagery provides the classifier with prior knowledge, enabling usage of machine learning algorithms. Machine learning allows us to introduce concepts to a computer based on labeled samples: from specific (building) to generic (obstacle). Learning the overhead imagery features from an UGV greatly simplifies the classification task. Instead of detecting all possible feature classes from every input image we only have to detect the feature classes that an UGV is able to recognize –1 the classifier is trained to detect the labels provided by an UGV.

It is no longer required from the classifier to “know” all terrains but instead it can “learn” the local terrain from an UGV.

The resulting classifier is highly flexible because it adapts to terrain, weather and illumination conditions. Because of stochastic nature of machine learning we must use caution during interpreting the results. Before the UGV has gathered a sufficient amount of information about the environment, the overhead imagery classifier is unreliable and can not be trusted. Instead of defining an arbitrary threshold for “sufficient” information we decided to build our system to be able to cope with uncertainties of the classifier. We can measure the capability of the classifier by using only part of labels returned by the UGV for training the classifier and leaving the rest for testing. The capability of the classifier can then be used for weighting the contribution from the long-range navigation system – while classifier confidence is low, the UGV should rely on local navigation capabilities and gradually increase its reliance on the long-range navigation system as its confidence increases.

The machine learning algorithms in general are parameterizable functions, parameters of which are taught using known samples. As a simple solution, we could implement the linear classifier as a machine learning algorithm that uses the set of feature extraction algorithms as its input [15, 40], but that would leave us with a challenge of finding a good set of feature extraction algorithms. Instead, we suggest using a machine learning algorithm that combines feature extraction and linear classification: convolutional artificial neural network.

The convolutional ANN utilizes a key property of images: nearby pixels are more closely related than scattered pixels. Some of the additional properties of convolutional ANNs are shift invariance, rotation invariance and scale invariance [40]. Shift invariance is helpful because when we are looking for a house in the input pattern it may be in any part of the pattern, it does not have to be in the corner where it was during training. Scale invariance helps to detect all sizes of buildings or trees, not just the ones that were present in the training set. Due to shared weights in convolutional layers the variable space is smaller; the training of the classifier takes less time and can be done with smaller data sets than with conventional fully connected neural networks.

The convolutional ANN configuration proposed in this thesis is based on a configuration used for handwritten character recognition [41]. The optical character recognition was done on grayscale imagery, but aerial imagery is a 24 bit RGB. Converting the color imagery to RGB results in a loss of valuable information. To accommodate the color input we decided to extend the network by introducing three input patterns to the network instead of one – one pattern per color channel. We could have made the one input pattern larger but the mapping of the convolutional kernels to the input pattern would have been unnecessarily complicated without any benefits. An upside to our chosen approach is that the classifier can easily be extended to fit infrared imagery to our classifier when need arises – infrared is just another color channel.

The color information is invaluable because it gives cues about the material – vegetation (grass, weed, bushes) dominates in the green color channel while rigid objects (dirt, rocks, tree trunks) tend to be brown [5]. In addition, pixel-by-pixel comparison of red and near-infrared color channels can be used for measuring the chlorophyll content and thus can be used as an alternative robust approach to detect vegetation [42, 43]. The chlorophyll content strongly correlates to the compressibility of the terrain – softer materials, such as grass and leaves, tend to have high chlorophyll content while tree branches and rocks have low or none.

To describe the useful properties of a convolutional neural network it is necessary to explain its inner workings. The following chapter contains a short introduction to mathematics behind neural networks in general, followed by an introduction to convolutional neural networks accompanied by comments of how various properties are utilized in the current work. Further paragraphs describe classifier training set creation and classifier output processing and path planning.

2.2 Artificial Neural Networks

The objective of this section is to briefly introduce artificial neural networks (ANNs) and how they work. Next, convolutional neural networks are covered along with a description of why specific properties of a convolutional neural network are useful in the context of the current work. The objective, however, is not to convey the history of the artificial neural networks or its relations to human brain. For the sake of brevity, only the basic mathematical framework is covered – for a more detailed account, LeCun's "Efficient backprop" [44] or Bishop's "Pattern recognition and machine learning" [40] is recommended.

The basic computational unit of an ANN is a *neuron*. A neuron is typically connected to other neurons using *weights*. Every neuron has an output; the value of the neuron output is usually calculated as the function of the weighted sum of the connected neuron outputs. The neurons in an ANN are organized into two or more layers; every neuron in a layer is usually only connected to the neurons in the previous layer. The neurons in the first layer are called *input* neurons; their output values are set explicitly. The last layer of an ANN is called the *output* layer and the intermediate layers are *hidden* layers. To evaluate an ANN its inputs have to be set and the output values of all neurons have to be calculated one layer at a time.

The values of output neurons in an ANN depend only on the values of input neurons and network weights. The basic idea behind ANNs and artificial intelligence in general is that the weights and thus the function performed by an ANN is trained. Training of an ANN is done by evaluating a set of inputs with known output values and adjusting the weights in the ANN to minimize the difference between the ANN output and the known output.

First, let us consider a very simple dual layer fully connected feed forward network with two neurons in the input layer and one neuron in the output layer (Figure 1). The neuron in the output layer is connected to both neurons in the input layer and to an extra *bias* neuron by using weights.

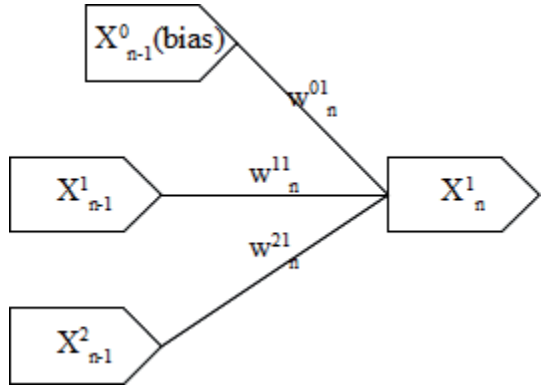

Figure 1. A simple ANN network

The value of the output neuron is calculated by eq. (1):

$$X_n^i = F\left(Y_n^i\right) = F\left(\sum_j X_{n-1}^j w_n^{ji}\right), \qquad (1)$$

where

X_n^i is the output value of the output neuron,
Y_n^i is the weighted sum before the activation function,
F is an activation function (*tanh*),
X_{n-1}^j is the value of j[th] input neuron,
w_n^{ji} is the value of weight connecting neurons X_{n-1}^j and X_{n-1}^i,
X_n^0 is the bias neuron and its values fixed to 1.

In essence the ANN executes a nonlinear function mapping input variables X_{n-1}^j to output variables X_n^i controlled by the set of variables w_n^{ji}.

We are using *tanh* function as an activation function because it is a sigmoid function that is symmetrical over the coordinate axis. The *tanh* function is easy to calculate and also the derivative of *tanh* (we need it later) is also easy to calculate using eq. (11):

$$y = \frac{d\,\tanh(x)}{dx} = 1 - tanh^2(x) \qquad (2)$$

The simple two layer network is useful for illustrative purposes but not for much else. Usually the ANNs are more complicated containing multiple inputs, multiple outputs and hidden layers. For example, the Figure 2 presents the ANN configuration most frequently used that has K input neurons, L neurons in the hidden layer and M output neurons. The ANN is an example of a *fully connected network*, where each neuron in a layer is connected to every other neuron in the previous layer. It can approximate any continuous function on a compact input domain to arbitrary accuracy, provided it has enough neurons in the hidden layer [40].

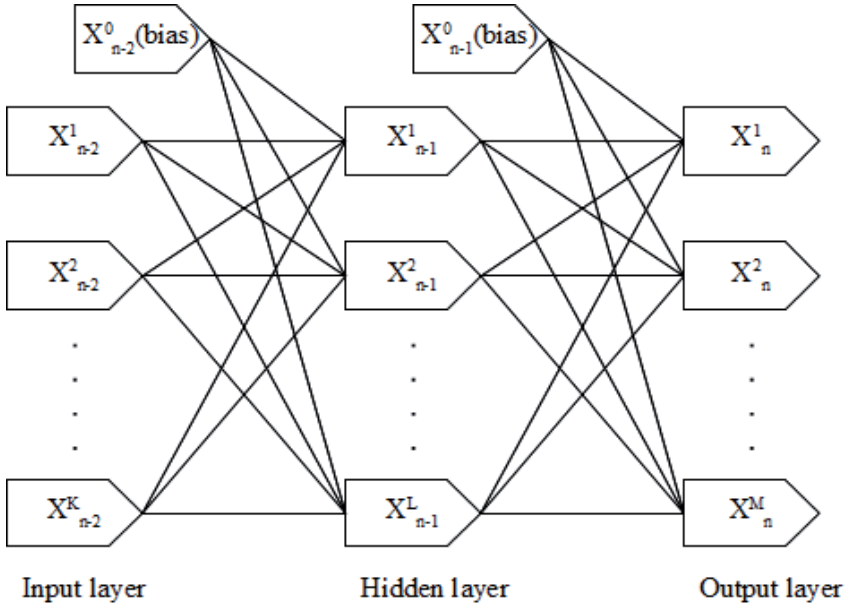

Figure 2. A simple network with one hidden layer

The values of neurons in an ANN output layer are defined by the values of neurons in the input layer and the weights through the equation (1). A popular way to find the values for weights is to train them using a *backpropagation* algorithm. The backpropagation algorithm feeds patterns with known output values to the network and propagates them through the network. Once the network output vector is calculated, it is compared to the expected output to find an error vector. The errors at the output layer are then backpropagated to previous layers and are used to adjust weights in the network.

For training we need a sample dataset of inputs with known output values. We start training by randomizing all weights in the ANN, then select samples one by one from the dataset, initialize the network inputs X_{n-1}^{j} in Figure 1 to sample and calculate the network outputs X_n^i using formula (1). Now that we have the network output, we compare it to the known output to find the error for each output neuron [44]:

$$\frac{\partial E_n^i}{\partial X_n^i} = X_n^i - T_n^i, \qquad (3)$$

where

X_n^i is the network output for a given input sample,

T_n^i is the expected value for a given input sample,

$\frac{\partial E_n^i}{\partial X_n^i}$ is the error at the output neuron.

The error of output before the activation function is

$$\frac{\partial E_n^i}{\partial Y_n^i} = F'\left(X_n^i\right)\frac{\partial E_n^i}{\partial X_n^i} \tag{4}$$

and the error at the previous layer neuron is proportional to the weight it is connected to:

$$\partial E_{n-1}^j = \frac{\partial E_n^i}{\partial X_{n-1}^j} = \sum_i w_n^{ji}\frac{\partial E_n^i}{\partial Y_n^i}. \tag{5}$$

The share of the error for a weight is proportional to the signal it is carrying:

$$\frac{\partial E_n^i}{\partial w_n^{ji}} = X_{n-1}^j\frac{\partial E_n^i}{\partial Y_n^i}. \tag{6}$$

During training we feed multiple samples through a network, backpropagate the errors and adjust the weights by the coefficient from equation (6). To achieve steady improvement during training the weights should be gradually adjusted using a learning rate:

$$w(t) = w(t-1) - \eta\frac{\partial E}{\partial w}, \tag{7}$$

where

$w(t)$	is the value of the weight after adjustment,
$w(t-1)$	is the value of the weight before adjustment,
η	is the learning rate,
$\frac{\partial E}{\partial w}$	is the adjustment factor from equation (6).

To acquire the best result out of the training dataset it should be fed through the network in multiple *epochs*. After each epoch the learning factor should be decreased to improve the accuracy of the approximation function the network performs.

This section described the basic functionality of an artificial neural network. ANNs approximate complex functions by combining a large set of simple but trainable functions. The fundamental unit of an ANN is a neuron that executes the simple function using trainable parameters. The neurons are organized into layers so that neurons in a given layer are only connected to the previous layer. The layers are executed one by one, each prior layer output is input for the next layer.

"Whole is more than sum of its parts" – Aristotle.

2.3 Convolutional Neural Networks

Even though large enough neural networks can be trained to approximate any function, they are not necessarily efficient and there are additional constraints in the task at hand that make usage of a large fully connected neural network infeasible. Because we want to be able to use overhead imagery soon after we have dispatched the UGV, the classifier must be trainable even with a small set of samples, but large fully connected neural networks have huge internal variable spaces that need equally huge sample sets for training. In addition to the data set requirement, it takes much time (in processor core hours) and energy to actually train those variable spaces.

Convolutional neural networks support specific mechanisms, such as local receptive fields, weight sharing and sub-sampling that are particularly useful for feature extraction from images. In the following samples, a small convolutional neural network is built and its useful properties are described. The samples are organized in several steps; each step introducing and describing a new feature.

In order to feed grayscale image elements (patterns) into a network we will build an artificial neural network that has an input layer, one hidden layer and one output layer. The input layer must have as many neurons as the pattern has pixels to ensure one to one relationship. In order to feed patterns into the network let us arrange the neurons in the input layer into a two- dimensional array (feature map) and copy the pixel intensity values from the pattern directly to the input layer neurons (Figure 3). For color images (RGB or YUV) we can separate the image into color channel and use a separate feature map for each color channel.

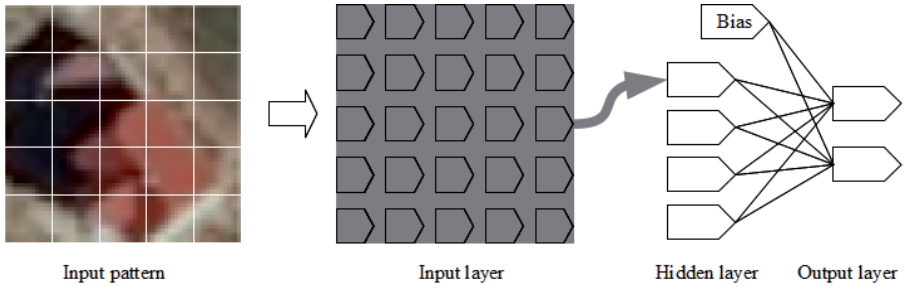

Figure 3. Image pixel intensities are directly transferred to input layer neurons. In order to illustrate convolutional kernels we append a gray background to all neurons connected by the kernel and use bold gray arrow to point where those neurons are connected to

The centerpiece of the convolutional neural network theory is a notion of a *kernel*; a convolutional kernel is a set of weights that connects multiple previous layer neurons to a next layer neuron. In the above image the neurons that are connected by a convolutional kernel are encapsulated in gray rectangle and the neuron they are all connected to is at the other end of the wide gray arrow. The ANN in (Figure 3) connects all the input layer neurons

to just one neuron in the hidden layer and therefore it is not particularly interesting.

On the following sample (Figure 4), only a subset of input layer neurons is connected to each of the hidden layer neurons by using a convolutional kernel. An important thing to note here is that all the subsets of input layer neurons are connected to the next layer by using the same shared kernel so that there are only 9 weights (actually 9 weights + 1 bias) connecting the input layer and the hidden layer in the figure. The neural network is called convolutional because in essence the kernels perform a convolution operation.

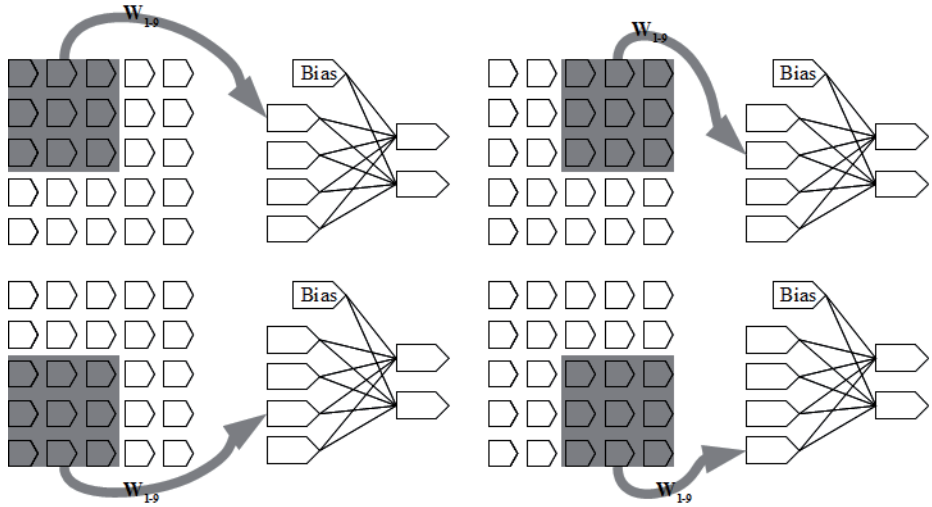

Figure 4. A convolutional layer. All neurons in the convolutional layer are connected to a set of previous layer neurons by using a shared kernel

Weight sharing has many useful properties. As there are fewer weights in the network, the variable space of the network is significantly smaller and the network can be trained using fewer samples. For an UGV it is important because it has to roam and gather local data used for training the classifier – the less data is needed the faster the long-range system can be used.

Each kernel works on a small subset of the input pattern and as such can be perceived as a local receptive field, the kernels are capable of extracting localized features from the input pattern. The network is organized in such a way that it extracts small features from the input pattern and based on the combination of those features it decides if an object is present on the input pattern or no. If the convolutional layer detects a chimney, an antenna and roofing tiles, the fully connected layer can decide that there is a building present in the input pattern.

Because the signals from different parts of the input layer are propagated to the next layer by using the same convolutional kernel, the network is shift-invariant. It helps to detect features on the input pattern, no matter in which corner of the input pattern they are found. Locating the precise position of a

feature in the input pattern potentially even hurts the classifier capability to identify the input pattern because the positions will vary for different instances of the feature [45].

To further reduce the precision of feature position detection, sub-sampling layers are used (Figure 5). The sub-sampling layers "perform local averaging and sub-sampling, reducing the resolution of the feature map, and reducing the sensitivity of the output to shifts and distortions" [45]. Each neuron in the sub-sampling layer is connected to four neurons in the previous layer; it aggregates the output of the connected four neurons by using a trainable kernel.

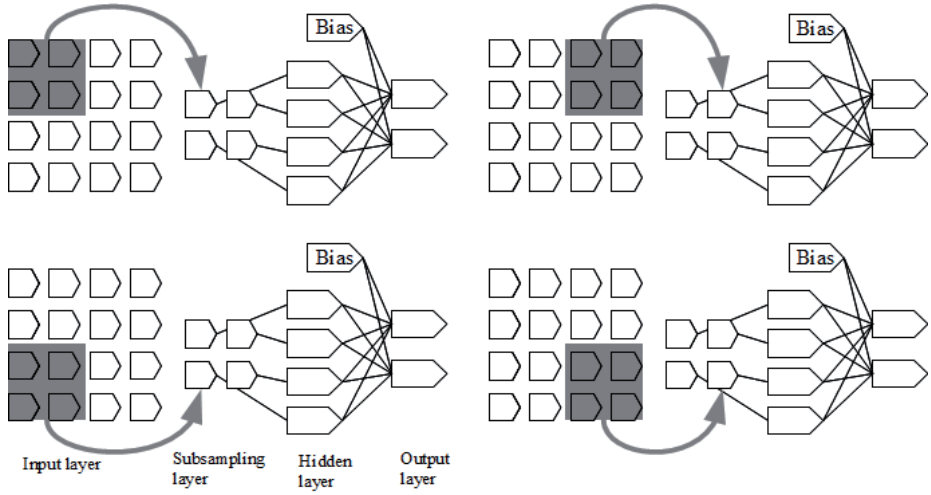

Figure 5. Sub-sampling layer, its purpose is to downsample resolution of the previous layer to reduce sensitivity to distortions

Simard *et al*. [41], however, have shown (link) that the sub-sampling layer can be combined with the convolutional layer. To illustrate that we reorganize the hidden layer neurons from Figure 4 into a convolutional feature map, as shown in (Figure 6). The neurons in the feature maps are connected to the previous layer by using convolutional kernels, but the kernels skip every other row/column in the previous layer. In the illustration the 5x5 input layer is connected to 2x2 hidden layer; without row/column skipping the hidden1 layer would have to be 3x3. While the hidden layer size is insignificant in the current example, in practical samples the input patterns and feature maps are larger and the effect becomes more dominant.

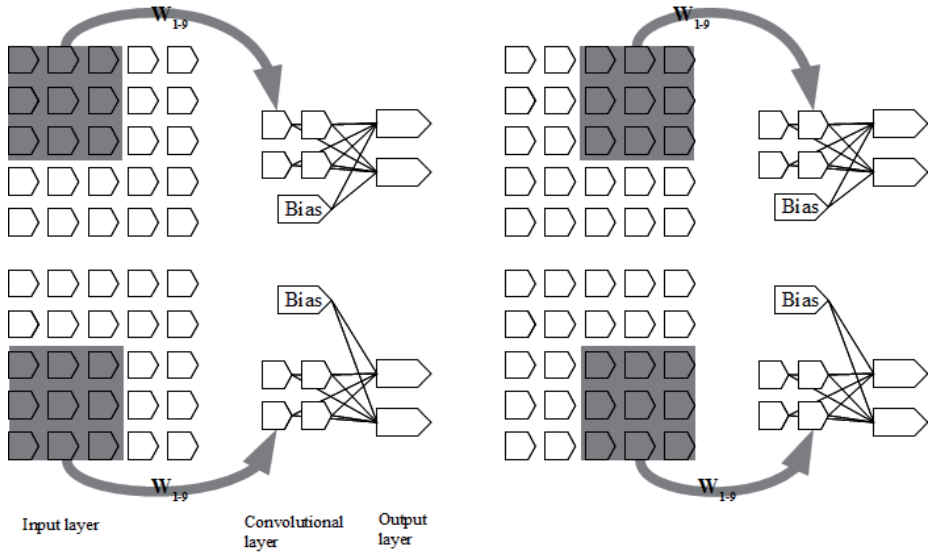

Figure 6. Integrated sub-sampling layers. To reduce the resolution of the input pattern the convolutional kernel skips every other column and row during mapping. NB! The neurons in the convolutional layer are organized into a 2D grid; not to be confused with the sub-sampling layer in Figure 5

The invariance to geometric transformations can be increased by introducing multiple alternating convolutional and sub-sampling layers into the network [45] – or in the case of Simard *et al*. [41] by stacking multiple feature maps. The progressive loss in resolution should be compensated by increasing the count of feature maps in each layer.

It is also important to have multiple feature maps in each layer because the feature maps connect to the previous layer by using shared kernels. Each kernel is roughly capable of extracting a single feature from the previous layer, i.e. to increase the count of detected features we need to increase the count of feature maps. It should be noted, however, that the features a kernel will extract are not high level, such as tree or chimney but rather low level items, such as cross-hatching of specific pattern of pixels. Multiple high level objects may share the same low level features.

The reference configuration chosen as a classifier in our project was based on suggestions made by Simard *et al*. [41]. We chose to use three or four 29x29 pixel patterns for RGB and infrared color channels in the input layer, one for each color channel. The features on an aerial imagery with a resolution of 0.8 m per pixel fit comfortably into the input 29x29 input pattern, for higher resolution imagery we might need to increase the input pattern size and kernel size. The first convolutional layer contains six 13x13 feature maps, the second convolutional layer contains fifty 5x5 feature maps and the third hidden layer is a linear fully connected layer with 100 neurons (Figure 7). Both convolutional layers use 5x5 kernels.

Due to the used sub-sampling method, the layer sizes are linked. If a feature map edge size on the previous layer is x, then the next layer edge size can be calculated using formula (x – kernel_size) / 2 + 1. Hence the feature maps in the network have sizes 29x29, 13x13 and 5x5. Because the sub-sampling layers are integrated into the feature map, the first hidden layer feature maps are only 13x13 pixels, without sub-sampling it would be 25x25 pixels (we use 5x5 kernels). Similarly, the second convolutional layer feature maps are 5x5 pixels instead of 9x9 pixels.

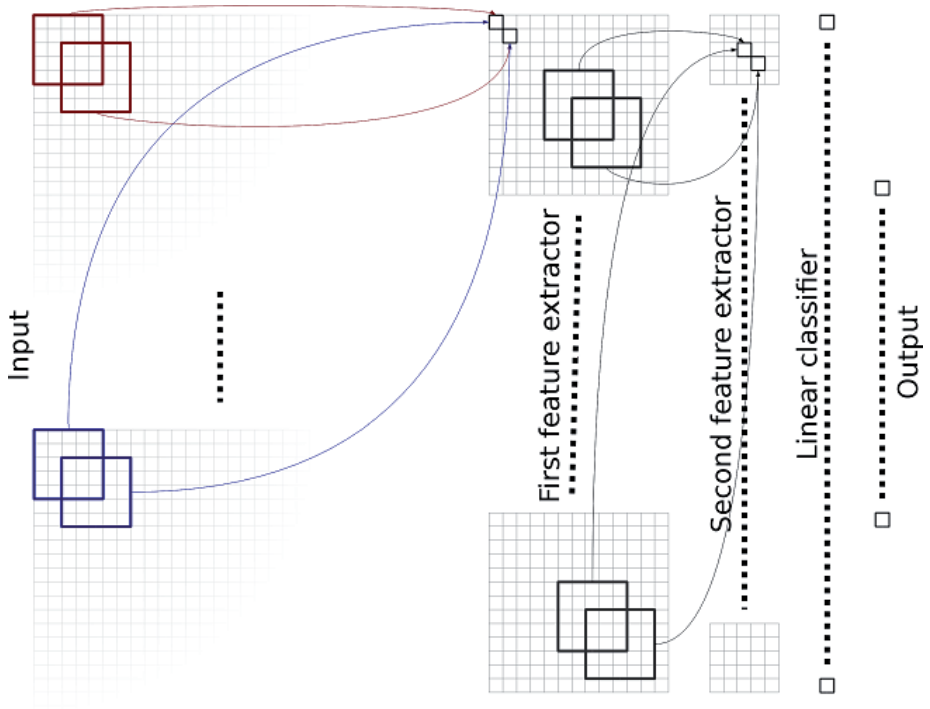

Figure 7. Classifier structure

In essence the reference ANN is a fusion of two separate ANNs with three layers both – the first three layers can be viewed as a dedicated feature extractor and the last three layers are a linear classifier. The third layer in this case is shared; it is the output layer of the feature extractor and the input layer for the linear classifier.

2.4 Training

An artificial neural network is a generic tool; even the specific convolutional ANN configuration described above is a relatively generic tool that can be trained to solve arbitrary image processing problems. The specific capabilities of the classifier must therefore be described by the structure of the network and by the data set used for training. More specifically it is important to describe the method of data gathering used to build the training data set.

We have two methods of training data set preparation: one is derived from a handwritten character recognition field [41, 45] (*Method A*) and the other is inspired by an UGV navigation system [46] (*Method B*). Both methods can be used for training the classifier, but they result in classifiers that have distinctly different properties.

2.4.1 Method A

Method A is based on the idea that the convolutional ANN is capable of extracting distinct features from the input pattern and the linear classifier layer can then be trained to report presence of objects in the input pattern. In layman's terms each classifier output is trained to answer simple questions, such as “Is there a building in the input pattern?”, “Is there a tree in the input pattern?” or “Is there some grass in the input pattern?”. All the classifier outputs are independent, there can be multiple object classes simultaneously present in the input pattern.

During construction of the training set special care must be taken to ensure that each object is represented in the input pattern by an adequate number of pixels, e.g. every object in the input pattern has to be on at least x pixels or not be present at all. The classifier must have enough data to extract features from and if the training set contains patterns where an object is represented only by few pixels on the input pattern, it will only confuse the classifier diminishing its capabilities.

For numerical experiments with 29x29 input patterns we used a threshold of 100 pixels – patterns where objects were represented by less than 100 pixels were discarded. Overall about 40% of the generated samples were discarded because of this criterion – to obtain a training set of 30000 patterns we had to generate about 50000 patterns.

2.4.2 Method B

Method B will train the classifier to detect the category of a single pixel in the input image, particularly the pixel in the center of the pattern. The classifier will then answer the question “Which category does the pixel in the center of this pattern belong to?”. The categories are exclusive and thus the classifier outputs are dependent on each other.

During construction of the training set we discarded patterns where the selected pixel was over-specified (labeled to belong into multiple categories). We also balanced the training set by making sure that all categories were

represented by an equal amount of patterns. Overall about 46% of generated samples were discarded because of those criteria – to obtain a training set of 30 000 patterns we had to generate about 56 000 patterns.

It is important to note that the classifier outputs are trained to be independent, but when evaluating unknown patterns the classifier may declare that some of the patterns have multiple features present, particularly in the feature border regions. While we could use the Bayes' theorem to decide which feature class has the highest probability and suppress all other outputs, it is better to forward all the output values to the cost map calculation algorithm presented in section 0. For instance, if a pixel is classified to be simultaneously road and grass with 50% probability for both classes, the traveling cost at this pixel will be somewhere between road and grass.

2.4.3 Method A versus Method B

The main advantage of Method A is the input quality requirement, during composition of the training set the expected output values for each input pattern have to be provided. The training method deemphasizes exact locations of features on the input pattern only requiring a feature to be present to be detected. It is useful when the features on the input patterns are ill defined, i.e. it is very difficult to produce pixel precise definitions of features, especially at feature border region. Image and label coherency is difficult to achieve and having an additional invariance built into the classifier is beneficial.

The second advantage of Method A is the computing performance of the classifier; the output of Method A gives information about the whole 29x29 pixel input pattern while the output of Method B only gives information about a single pixel. By using Method A it is possible to obtain preliminary results by classifying an aerial image at a coarse level and then to refine the results by choosing overlapped patterns, but with Method B one must either classify the aerial imagery pixel by pixel or choose to skip pixels risking missing of important features.

The main advantage of Method B is spatial resolution of the output as illustrated in Figure 8. It allows us to classify every pixel in the aerial image (apart from some pixels on the image edge) while the spatial resolution of Method A is limited to its pattern size. It is possible to increase the resolution of Method A by choosing overlapped patterns but the output will still be smudgy because Method A does not specify the location of a feature in the input pattern.

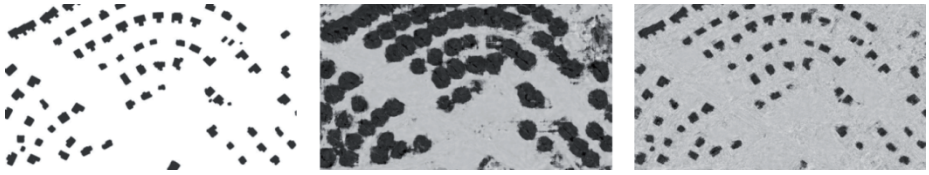

Figure 8. Comparison of classification methods: hand-classified mask for buildings (left), classifier output with Method A (center) and Method B (right)

2.4.4 Training Procedure

The actual training procedure is quite straightforward; it is done using a training data set that contains a set of patterns with known (expected) output values. Before training all network weights are randomized and learning rate is initialized to a predefined value (0.01). During training the patterns are loaded from the training data set one by one, color channels are separated and the ANN input layer neurons are initialized to the pixel intensity values. After the input layer is initialized, the first hidden layer can be evaluated by using equation (1), the first hidden layer outputs are then used as second hidden layer inputs, which are also evaluated and it proceeds until the last layer is evaluated.

After the forward propagation step is complete, the last layer output (ANN output) is subtracted from the expected value to find the network error. The errors at the output layer neurons are backpropagated to the penultimate layer neurons using (4) and (5), from where they are backpropagated to previous layers until the errors are defined for the first hidden layer neurons. Next, the errors are divided between the weights using equation (6), for shared weights the errors from different sources are added together. Finally, the weights are adjusted by the fraction of the error using equation (11).

The patterns in the training set are introduced to the classifier for learning in multiple epochs. After each epoch the learning rate is slightly reduced, stabilizing the learning process. To avoid dependence on the order in which the patterns are introduced to the classifier we randomize the training set before each epoch.

To increase the learning rate we used a gradient based method that will assign an additional learning rate parameter to each weight. The parameter, called a diagonal hessian, was calculated before each epoch in a “boosting” phase – 500 patterns were evaluated to find the diagonal hessian value. The method increases the learning rate of the network threefold without introducing much computing overhead [44].

The training procedure is identical for both Method A and Method B – the difference is only in how the training data set is generated. Resulting classifiers, however, have distinctly different properties. Method A is more robust to input errors while Method B has higher output resolution. After a classifier has been trained, it can be used for object detection on overhead imagery and for long-range path planning. The classifier output is not directly usable for path planning, the following section will describe the challenges and solutions to converting the classifier output into a form that can be used as path planner input.

2.5 Cost Map Generation

The path planner algorithm we are using is A*. The A* algorithm requires a set of connected nodes with predetermined positive costs for moving between the nodes and results with minimal cost path through the nodes from start position to destination. This section will describe how we use the classifier to generate the set of nodes and calculate the associated costs. In addition, we propose a simple cost function that takes into account the uncertainty of the classifier.

To generate the nodes we take an aerial image, divide it into a grid of 29x29 patterns and feed the patterns into the classifier. The classifier returns a feature vector for each pattern that can be used for cost map generation. For the simplest approach we take each 29x29 pattern as a node and calculate the cost of moving into that node based on the feature vector. The problem with dividing the input image into 29x29 pixel patterns is that it reduces both the horizontal and the vertical resolution of input by 29 times. It is especially problematic with a classifier that uses training Method B, then we obtain information about every 29th pixel. To increase the resolution we use overlapping patterns: the first pattern origin is at coordinate (0; 0), the second pattern is n pixels further away at (n; 0), etc. (where n is the scaling factor).

The feature vectors returned by the classifier are raw classifier outputs and thus not directly usable for cost map generation, we need to convert the classifier outputs to feature presence probability estimations. In theory the classifier outputs are trained to be 1.0 when a feature is present on the input pattern and -1.0 when the feature is not present. If features are well distinguishable from the background, the classifier output values tend to acuminate around -1.0 and +1.0 (Figure 9) and feature probability can be estimated using a simple threshold function: if the output value is over 0, we can say the feature is there and if not, we can assume it is missing on the input pattern.

However, a more prosaic case is encountered when the searchable features blend into the background and are hard to detect, resulting in a lower classification rate. In that case the classifier outputs are in the range of 1.7 to 1.7 (limited by used sigmoidal activation function), the density functions will coalesce (Figure 10) and it is difficult to differentiate if the feature is present or not. The low classification rate can be caused by a large number of variables, such as a too small training set, spatial discord between training data and overhead image or distortions in the overhead image. Robust cost map generation solution needs information about the features detected on the input pattern coupled with continuous confidence information – not just discrete probabilities of 0 and 1.

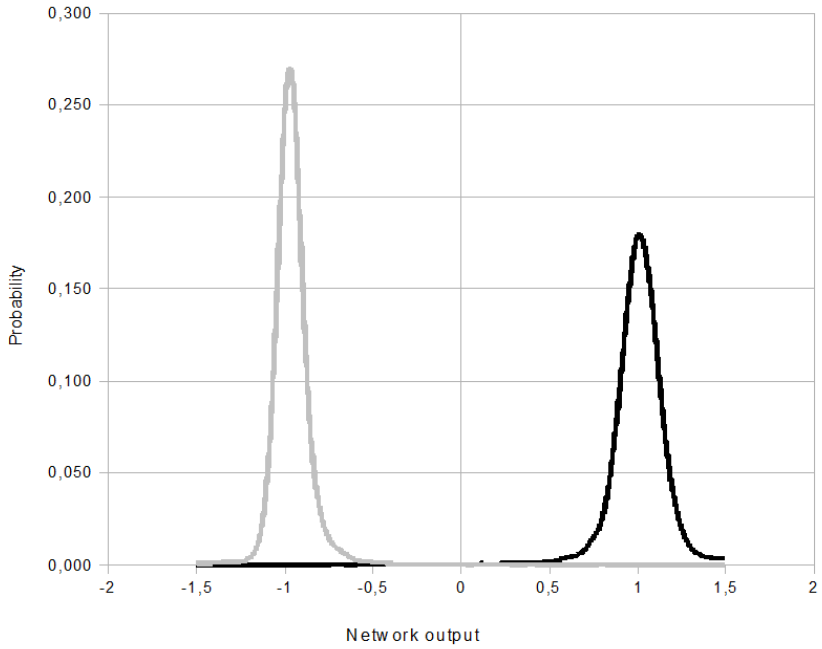

Figure 9. Density functions of network output when detecting buildings. Black line represents the density of the classifier output when the input pattern contains a building and gray line is the density of the classifier output, when the input does not contain a building

We need to convert the feature vector into a probability vector, each element of which represents the probability that the feature is present on the input pattern. In this case Bayesian approach that uses prior knowledge suits best. The Bayes' theorem states that

$$p(C_k|x) = \frac{p(x|C_k)p(C_k)}{p(x)}, \qquad (8)$$

where

$p(C_k|x)$ is the probability of feature class k when the classifier output value is x,

$p(x|C_k)$ is the probability of output x when it is known that the input belongs to feature class k,

$p(C_k)$ is the probability of class k,

$p(x)$ is the probability of classifier output x (can be calculated using cm 2).

The $p(x)$ in Bayes' theorem the denominator can be calculated using the sum rule:

$$p(x) = \sum_k p(x|C_k)p(C_k), \qquad (9)$$

where

$p(x|C_k)$ is the probability of output x when it is known that the input belongs to feature class k,

$p(C_k)$ is the probability of class k.

To find the class-conditional densities $p(x|C_k)$ and the prior class probabilities $p(C_k)$ for each feature class we divide the training set into two. The training set contains the known output values for each pattern, so it is easy to extract the patterns that contain a given feature (class C_1) and the ones that do not (class C_2). After evaluating both data sets with the classifier we can plot out the class conditional densities for both classes $p(x|C_1)$ and $p(x|C_2)$.

Figure 10 illustrates the class conditional densities $p(x|C_1)$ and $p(x|C_2)$ for a feature class that is not easily distinguishable from the background. The black line is the density of the classifier output on the data set where the corresponding feature was present, i.e. probability of the network output x when it is known that all input patterns contained feature class C_1. The gray line represents the density of the network output x on the dataset where the feature was missing on the input pattern (class C_2).

The probabilities of classes $p(C_1)$ and $p(C_2)$ are easy to calculate – they are the ratios of patterns in the divided data set to the full training set. For instance, if we have 30 000 patterns in a training set and 10 000 of them contained the given feature, then the probability of a feature being present on the pattern is $p(C_1)$ =10 000/30 000=0.33 and the probability of a feature missing is $p(C_2)$ = 20 000/30 000=0.66. Using class-conditional densities and probabilities of classes, the probability of output can be calculated using eq. (11) and from there the probability for each feature being present on the input pattern can be calculated using eq. (8). Specifically, we are interested in the probability of the feature being present $p(C_1)$ on the input pattern given the network output x: $p(C_1|x)$.

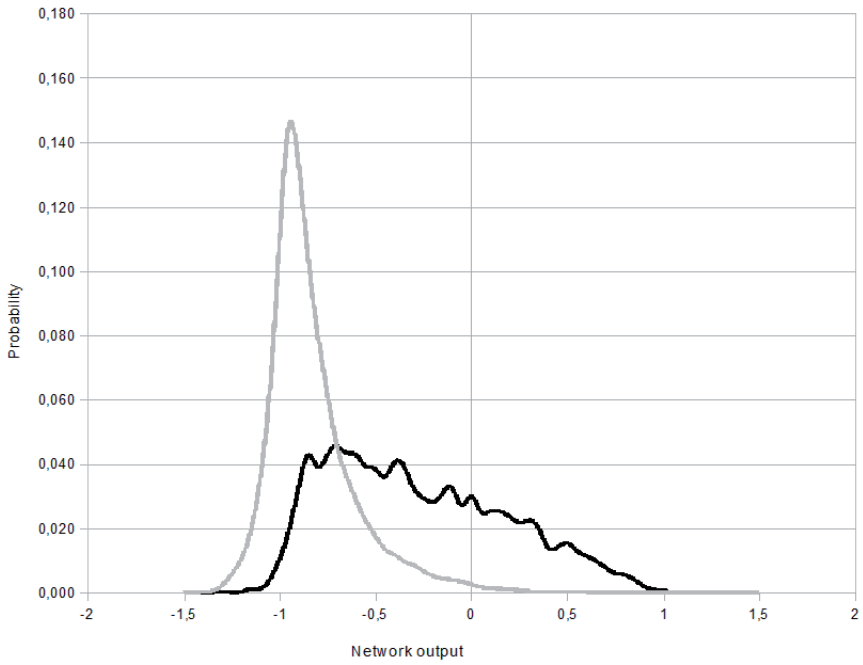

Figure 10. Class-conditional densities $p(x|C_1)$ (black) and $p(x|C_2)$ (gray) for a classifier output

In order to use the uncertainty of the classifier (probability of the feature being present) in the node cost calculation we need to define the cost of the unknown input. It will be used as the cost of the node if the certainty of presence for all features in the input pattern is 0. We want to have the cost of an easily traversable terrain (roads) to be lower than an unknown terrain and the cost of an impassable terrain (buildings) to be higher than that of an unknown terrain. The equation we are proposing has the following form:

$$cost = \sum_i B(o_i)w_i + c_u, \qquad (10)$$

where

O is the classifier output vector,

B is the Bayesian function that maps the classifier output to probability,

W is the weight vector,

c_u is the cost of the unknown terrain.

To make the cost of roads smaller than c_u we use negative weights for terrain features that are easily traversable and positive weights for features that are difficult to traverse. Finally, we add extra check to make sure the cost for the node is positive (11); the nodes with negative cost may lock A* algorithm into a loop where it keeps returning into the same node, thus breaking the algorithm.

$$cost = Max\left(\sum_i B(o_i)w_i + c_u, c_{min}\right), \qquad (11)$$

where

c_{min} is the minimal traversal cost that must be a positive number.

In addition to Bayes' function, we explored the usefulness of cumulative distribution functions created from density functions (Figure 9, Figure 10). One of the distribution functions gives us a probability that a feature class is present on the input pattern, depending on the classifier output. The other distribution function represents the probability that the feature class is not missing on the input pattern, also depending on the classifier output. Both of the distribution functions represent a probability of a feature being present, by choosing various combination methods we can bias the cost map generation. Multiplication of the distribution functions, for example, gives us probability mapping that requires detection of the feature "presence" and an extra confirmation that the feature is "not missing" on the input pattern. The multiplication function thus emphasizes the correct feature detection. The best results, however, are achieved by using Bayes' theorem.

The objective of converting the classifier output to probabilities and generation of cost map is path planning. We use a training data set to evaluate class conditional densities of classifier outputs, and use them in Bayes' theorem to estimate the probability of feature presence depending on the classifier output. The feature presence information along with confidence is used for cost map generation; the next section will demonstrate how it can be used for path planning.

2.6 Path Planning

Path planning is done by the A* algorithm, which is an improvement over Dijkstra's algorithm [30]. It generates the lowest cost path from a given start point to the destination point. As mentioned above, the A* algorithm requires a set of nodes with an associate cost of moving from one node to the neighboring node.

The A* algorithm is best summarized by a tutorial written by Patrick Lester:

1) Add the starting square (or node) to the open list.

2) Repeat the following:

a) Look for the lowest F cost square on the open list. We refer to this as the current square.

b) Switch it to the closed list.

c) For each of the 8 squares adjacent to this current square …

- *If it /.../ is on the closed list, ignore it. Otherwise do the following.*
- *If it is not on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.*
- *If it is on the open list already, check to see if this path to that square is better, using G cost as the measure. A lower G cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. /.../*

d) Stop when you:

- *Add the target square to the closed list, in which case the path has been found /.../, or*
- *Fail to find the target square, and the open list is empty. In this case, there is no path.*

3) Save the path. Working backwards from the target square, go from each square to its parent square until you reach the starting square. That is your path. [47]

The algorithm maintains a list of working items (open list) and a list of processed items (closed list). Each item has two costs associated: H is an heuristic value that estimates the distance of the current node from the destination and G is the cost of moving from the start node to a given item, the F cost is the sum of H and G. The G cost of the start node is 0; the cost of moving from the start node to the neighboring node is taken from the cost map calculated using equation (11). By working from the start node towards the end node we assign the G cost of each neighboring node as the cost of the current node + the value from the cost map calculated using equation (11), as specified in step c.

Development of a path planner is a separate field of research not covered in this thesis. We chose to use the A* algorithm [29] because it generates the lowest cost path from the start point to the end point, allowing us to demonstrate our system. When the system has to deal with highly changing

environments and continuous replanning, the D* [31, 48] or D* Lite [32] algorithms may be more appropriate. The A* and D* algorithms may be too slow for UGV local navigation, but they are suitable for planning in large-scale environments [33].

One thing to note, however, is that the path planning algorithm does not use the cost of all nodes. To evaluate the cost of a node we need to trigger the classifier which is a relatively costly process in terms of computer cycles. There is a possibility to integrate the path planner, cost map generator and terrain classifier for greater performance. Instead of evaluating full aerial image with a classifier for a cost map generator and a path planner, we could reverse the pipeline and evaluate the aerial imagery on a need-to-know basis. When the path planner needs a cost of a node that it does not have, it will request it from a path planner which in turn triggers the classifier. The classifier then evaluates the corresponding portion of aerial imagery, returns the result to the classifier which in turn calculates the cost for the path planner.

The classification of aerial imagery pattern by pattern on a need-to-know basis, however, neglects the benefits we receive from batch processing (section 3.5). Processing of a large set of patterns in parallel on a heterogeneous hardware is significantly more efficient than the classification of individual patterns.

To combine benefits of the integrated pipeline and batch processing we could take a hint from computing memory architecture, namely from caching. When a CPU needs to load an operand for an instruction that is not present in the on-die cache, it has to make a request for the relatively slow random access memory (RAM). Loading a single byte from RAM is a relatively slow operation, but loading subsequent bytes from RAM is much faster and it is very often a case when an argument is needed to be loaded from RAM the processor will soon need another argument from nearby memory location. Thus, to optimize the memory access when a single argument is needed from slow RAM memory, the whole page of about 1024 bytes is loaded into on-die cache instead.

In the case of path planning, when a path planner needs the cost of a node that has not been classified yet it is very likely that it will soon need the cost of its neighboring nodes, and their neighbors too. Instead of classifying a single node from aerial image, the cost map generator could trigger batch classification of a rectangular area with an edge length of a few hundred meters.

The path planner takes the cost map as an input and generates the lowest cost path from the start node to the destination node. Integrating the path planner with the cost map generator and terrain classifier may significantly reduce the area that is needed to be evaluated on aerial imagery and thus improve the performance of the system.

2.7 Cyclic Update and Self-Supervised Learning

In the previous theoretical section we described a classifier that is capable of extracting features from aerial imagery; the classifier can be trained offline using manually labeled input for training and then used on an aerial imagery for cost map generation and path planning. Relying on manual labeling, however, will impose significant limitations to the system as the manual labeling is a time consuming process and having an operator classifying aerial imagery would largely defeat the purpose of having autonomous machinery. For practical uses we must acquire the training labels autonomously utilizing the UGV onboard sensors; preferably in a cyclic manner to keep the system dynamic, capable of adapting to a changing environment.

Before the UGV is dispatched, the operator may need to analyze the feasibility of the mission. For preliminary analysis, we propose a generic overhead imagery classifier trained in a similar environment to be applied to either aerial or satellite imagery for cost map generation. In case of events that have significantly affected the passability of the environment, such as earthquakes or flooding, the area should be scanned by an UAV for fresh imagery. The generated cost map can then be used for path planning through target objectives helping to estimate mission energy requirement and duration. Having an estimation of energy requirements will also help to plan ahead the extent of the mission and adjust the objectives to match the UGV battery charge.

The UGV mission can be planned in a set-and-forget mode, where the UGV will autonomously execute the mission, or in an assisted mode, where the operator can contribute to path planning. The natural operator inputs for our system are mid-term waypoints and exclusion zones on aerial imagery. The midterm waypoints allow for an operator to adjust the generated path as necessary and insert explicit knowledge into the system about fords or bridges over rivers or mountain passes. The exclusion zones can be used to divert the UGV from dangerous zones such as minefields, and as extra safety inputs – rivers, lakes or steep slopes on mountains can be marked as exclusion zones to minimize the risk of classifier misjudgments. Alternatives to exclusion zones are safety zones – regions where UGV may travel. Safety zones limit the UGV traversal to the specified area, in this case the robot may only travel on the terrain that is explicitly enabled by the operator.

The UGV can be dispatched with an untrained or a generic classifier. As the UGV navigates the environment, it will create a local map by applying labels to the surrounding area that can be used together with overhead imagery to train the classifier. Introducing new labeled patterns to the classifier training set will reinforce its capability of detecting known features and recognizing new features. The updated classifier can in turn be used to evaluate the overhead imagery for updated cost map generation and for planning an updated path to the next waypoint. The data gathering, classifier training, cost map generation and path planning should be executed in cycles

with period of few minutes (Figure 11); it will keep the system adaptive, capable of learning new obstacles and pathways.

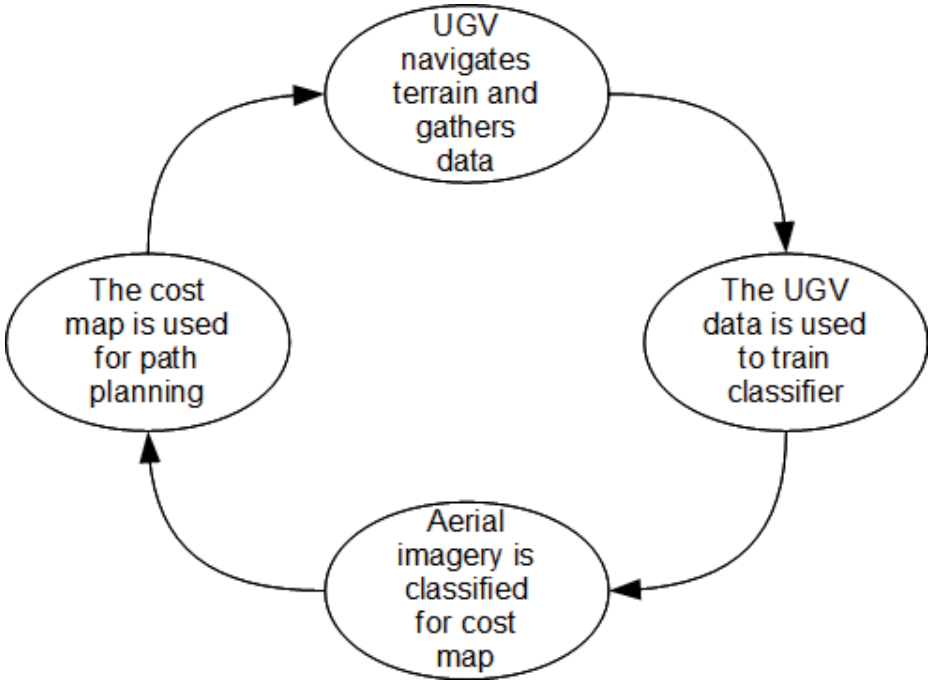

Figure 11. Cyclic update

It is important to maintain the diversity of the classifier training set [49]. If the UGV travels on a monotonous terrain for an extended period it may saturate the training set with similar patterns degrading the capability of the classifier to work in other types of terrain. The training set should keep an even amount of patterns representing each feature in addition to patterns that hold no feature. As new data is gathered it should substitute outdated patterns from the training set but not completely. The training set should keep representatives from older periods as "permanent memory" to be able to detect the old features when they reoccur.

The generic classifier has some classification capability while an untrained classifier has zero confidence in the output. The confidence of the classifier output can be used as a long-term navigation contribution factor – with an untrained classifier the UGV should rely on the on-board navigation system and gradually give the control over to the long-range system as its capability increases. In addition, the confidence of the classifier is reflected in the cost map generation – the cost of low confidence nodes is close to the cost of unknown terrain C_u. In the areas where the terrain traversal cost is low, the UGV can use the high-speed traversal mode. For the areas where the cost is close to C_u, the UGV should use the normal mode – as if the overhead imagery classifier was missing. Finally, in the areas where the cost is high, the UGV should take extreme caution and use slow speed.

3. Numerical Experiments

3.1 Test Setup

To validate the capability of our terrain classifier, cost map generator and path planner, we prepared two aerial photographs from the Estonian Land Board database: one image from the outskirts of Tallinn (Figure 12) and the other from a fen (Figure 13) and classified them manually for reference. The outskirts image covers approximately a four-square kilometer area and the fen image covers approximately two square kilometers. We used the Estonian Land Board xGIS database because their images and imagery acquired by UAV had similar resolution and because the images are already orthorectified and georeferenced.

Figure 12. Outskirts of Tallinn

The outskirts area was chosen because it represents a suburban area with low population density. In addition to roads, there are also large batches of grass that can be used to shorten the path, buildings that should be avoided and rubble piles that also pose a threat to our UGV. For classifier training and evaluating we chose a small area from the center of the image with a size of approximately 5% of the whole image (source image has a size of 3114x1994 pixels and the hand-classified area had a size of 706x401 pixels) and manually labeled it producing a mask for each feature we were looking for. In total, we created four masks: one for buildings, roads, grass and rubble (Figure 14).

Figure 13. The fen area

The weights used for cost map generation were selected such that the UGV would prefer roads for navigation avoid buildings and occasionally take shortcuts over grass when detour is inexpedient.

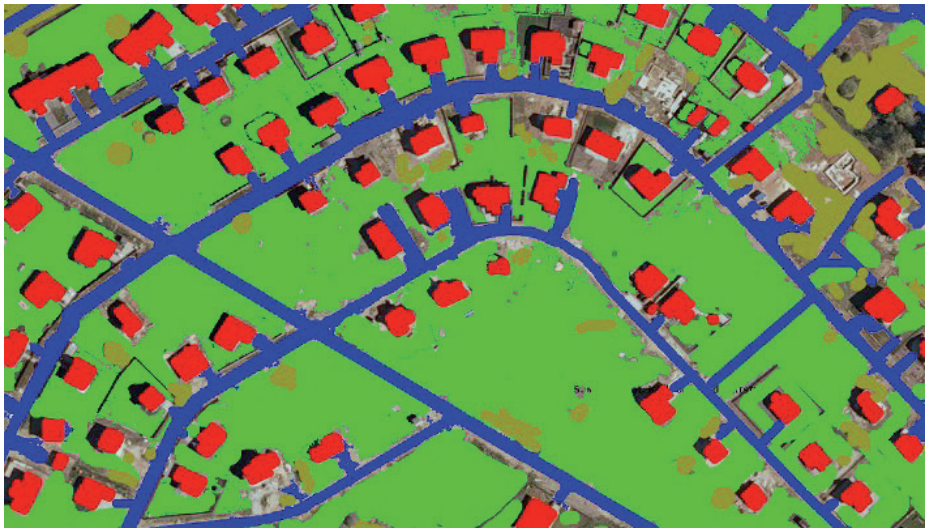

Figure 14. Classified area from the outskirts (top) and the feature masks (bottom): red for buildings, blue for roads, green for grass and brown for debris

The fen area (Figure 15) was chosen because it represents a low and wet natural terrain that is very hard to traverse for an UGV. The area contains multiple drainage canals, a low river that should be avoided and a few grassy maintenance tracks. At the manual labeling step we created three feature masks for the whole fen area – roads, water and trees. During cost map generation we focused on the track, avoiding trees and especially water.

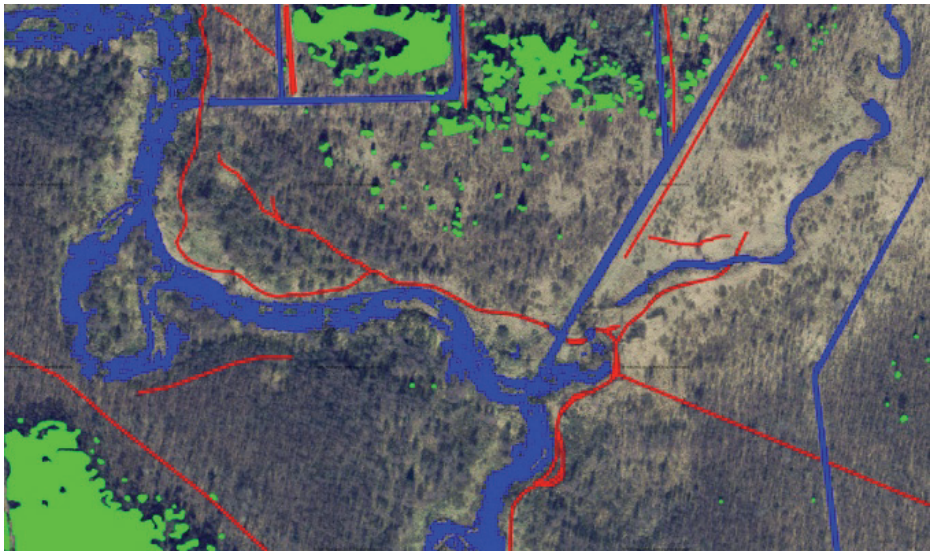

Figure 15. Fen (top) and its feature masks (bottom): blue for water, red for tracks and green for trees

In addition to aerial imagery, we prepared a satellite imagery of the suburban area for comparison. The satellite imagery was acquired from Google maps database, upscaled to identical resolution with aerial imagery and the same area that was used for aerial imagery data set creation was classified manually to obtain a comparable dataset. The satellite imagery, however, was taken on a different year, at a different time of the year and different time of the day and it had approximately 4x lower resolution (pixels per meter). The satellite imagery covers the same territory, but because it was taken from a different year, it contains some of the objects that aerial imagery did not, i.e. some of the construction projects were completed and related debris was cleaned up. Also, because the aerial imagery was acquired in summer, it has high grass content where the satellite imagery, which was acquired in early spring, has mud (Figure 16).

Figure 16. Comparison of aerial (top) and satellite (bottom) imagery

During the manual labeling process, it was difficult to precisely identify the feature borders due to limited resolution of the input – for dirt roads without no lining it is impossible to be pixel precise; we had difficulties even with well-defined structures such as buildings. To reduce the amplitude of the errors produced by manual classification we produced the masks on higher resolution imagery and then scaled both the input image and masks down by

2x. The resulting image resolution (0.8 m/pixel) is well suited for the classifier 29x29 pixel input pattern, as the features of the input image will fit comfortably into a pattern. Using lower resolution image would make details too small to detect and would make it difficult to detect higher-order objects based on those features. Higher resolution images would make the individual features too large, thus the input image would only contain a few of the features.

As mentioned above, we used a *reference configuration* for the classifier in our numerical experiments; it had three 29x29 neuron input patterns for RGB color channels, six 13x13 neuron feature maps on the first hidden layer, fifty 5x5 feature maps on second hidden layer, 100 neurons on the third fully connected hidden layer and three (fen) or four (outskirts) neurons on the fully connected output layer. The feature maps were connected to the previous layer using 5x5 convolutional kernels. Choosing a reference network allowed us to have a steady baseline against which to compare the effect of improvements, as we kept updating our system.

3.2 Classification Capability

It is important to measure the capability of the classifier because it determines the capability of the whole long-range navigation system. The faster the classifier learns an environment the faster we can use the navigation system for path planning. High certainty output from classifier enables UGV to use of high speed driving profiles in known areas. This section covers the various aspects related to the utilization of the classifier.

With a training set of 30 000 patterns the classification rate converges around optimum at around 20 epochs of training, the Figure 17 illustrates convergence rates of a large set of classifiers with the count of internal variables ranging from 35 000 to 535 000. The classification rate for reference configuration, containing 135 000 internal variables, converged after about 15 epochs of training. All the numerical experiments reported in this thesis were performed with the reference classifier configuration; to analyze the capability of the classifier we trained it for at least 20 epochs on each data set.

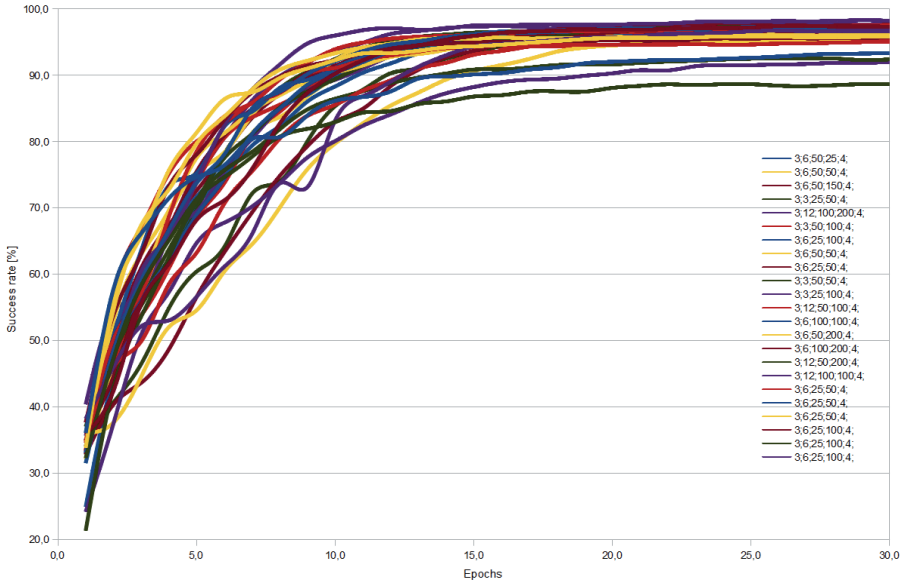

Figure 17. Convergence of the classification rate. The chart plots the correct classification rate of a set of classifiers with the internal variable count varying from 35 000 to 535 000 on the outskirts data set

For training Method A on the data set generated from the outskirts aerial imagery we achieved the combined classification rate of 97.9%; on 97.9% of the patterns the classifier could correctly detect all four feature classes. For training Method B on the same data set we achieved the classification rate of 88%. Seemingly Method A produces better classification results but in reality the two methods are not directly comparable because they produce different outputs (Figure 8); Method A will detect if the features are present on the

input pattern, while Method B will categorize on pixel on the input pattern. Because Method B has a more detailed output, it also has a higher error rate.

On the data set generated from the fen area, we achieved the combined classification rate of 73.7% with Method A and 83.5% with Method B. The much higher road detection capability of Method B can be attributed to training data set construction: with Method B we construct a balanced data set where all features are equally represented. For Method A we pick patterns randomly into the training set; the tracks make up a small area of fen image and thus they are under-represented in the training data set. To validate the claim we produced an ad hoc balanced data set for the fen region using Method A; each of the three features were present on at least 25% of the patterns and 25% of the patterns were featureless. On this balanced data set, we achieved 84.4% road detection capability of Method A – better than 79.2% of the unbalanced data set but still short of 90.2% of Method B. The ad hoc experiment indicates that the training set balancing contributes to the increased classification rate of Method B on the fen data set, but does not fully explain it.

Comparison of classification results for aerial and satellite imagery shows (Table 1) that the classifier works equally well on both. The classification rate of satellite imagery is even higher, which is unexpected, because the aerial imagery has higher resolution and more details. We also made tests to see if we can substitute aerial imagery for satellite imagery and found that we can use a classifier that was trained on aerial imagery for path planning on satellite imagery but not vice versa. The classification rate, when trained on one dataset and used on the other, e.g. when trained on aerial imagery and used on satellite imagery, is low, but we observed dramatic improvement when we retrained the classifier on the correct data set for one epoch. Hence, use of a previously trained classifier as a starting point for different image source classification is a worthwhile endeavor.

The classifier is trained using an aerial imagery and manually created feature masks. To evaluate the classifier capability visually we can classify the same aerial imagery and plot the masks by using classifier output. The feature masks produced by the classifier can then be compared against manually created labels. Figure 18 presents both manually created and classifier generated feature masks for three features from outskirts area: buildings, roads and grass. The feature masks have been generated with both training Method A and Method B for comparison.

Table 1. Comparison of classification rates for different classification methods on aerial and satellite datasets [%]

Feature Class	Buildings	Grass	Debris	Roads	Trees	Water	**Total**
Fen, aerial imagery, Method A				79.2	99.4	92.8	**73.7**
Fen, aerial imagery, Method B				90.2	98.2	93.6	**83.5**
Outskirts, aerial imagery, Method A	99.4	99.8	98.8	99.7			**97.9**
Outskirts, satellite imagery, Method A	99.7	99.9	99.6	99.8			**98.9**
Outskirts, aerial imagery, Method B	97.9	94.1	97.9	97.1			**88**
Outskirts, satellite imagery, Method B	97.4	96.3	98.6		96.9		**89.9**

The Figure 18 illustrates one of the challenges with the classifier training Method A: low spatial resolution. With pixel edge size of 29 pixels and image resolution of 0.8 m/px the classifier input pattern covers 23x23 m area. If grass is detected in any part of the input, the corresponding output will have high value; even if a small patch of grass is detected, it will be smudged out to an area that has a radius of 32 m (input pattern diagonal). The input of the Figure 18 contains a set of narrow roads and small buildings surrounded by grass, because all features will be smudged out by 32 m and grass is abundant in the input pattern, the feature mask for grass is oversaturated.

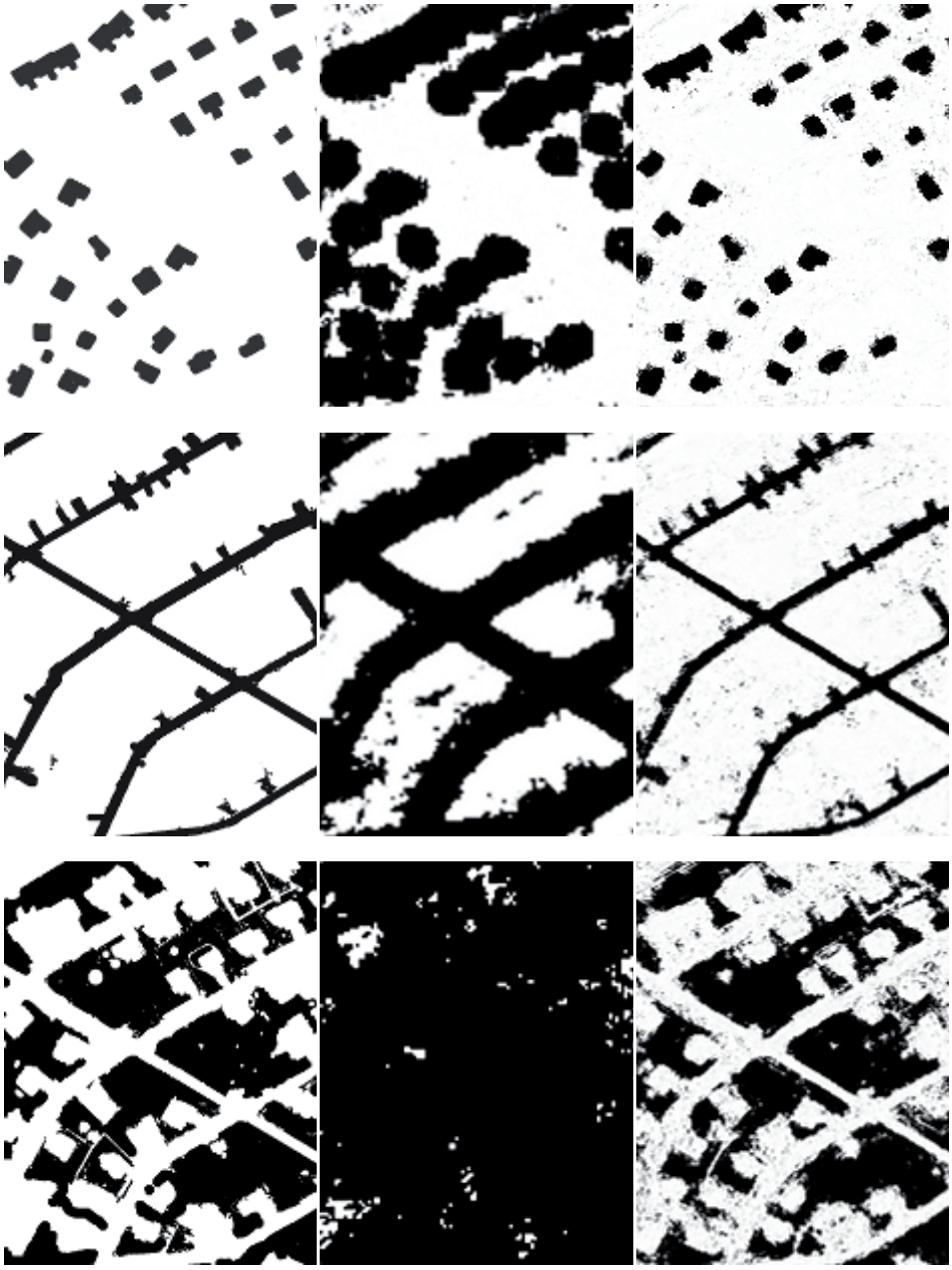

Figure 18, Comparison of a well-trained classifier outputs with reference. The first column has reference feature masks for buildings, roads and grass. The following columns have feature masks generated by the classifier with training Method A and training Method B

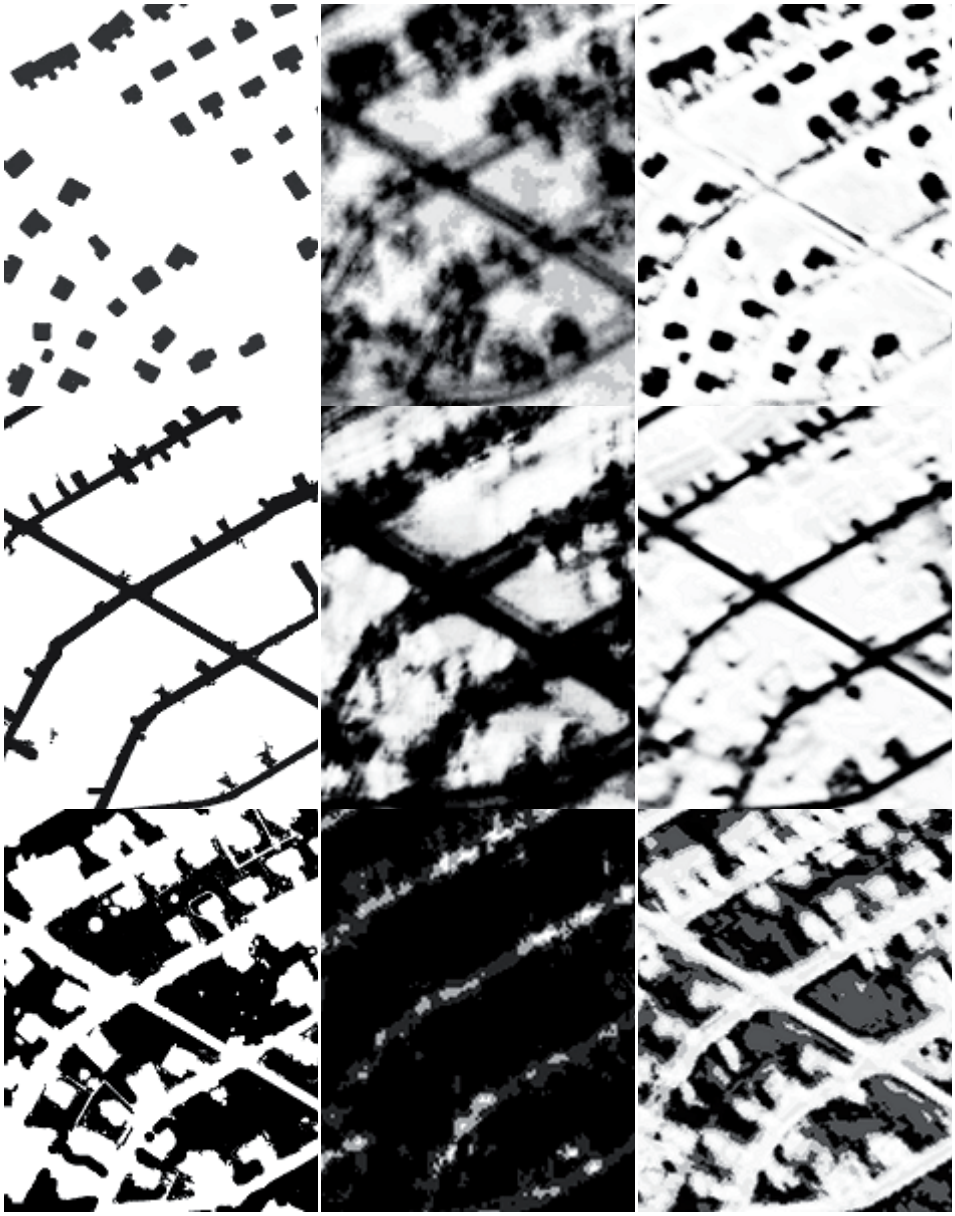

Figure 19. Comparison of under-trained classifier outputs with reference. The first column has reference feature masks for buildings, roads and grass. The following columns have feature masks generated by the classifier with training Method A and training Method B

Below in section 3.3 we assess the capability of the system to perform in low confidence scenarios. A classifier that is applied to imagery, with features that it is unable to recognize (due to insufficient training or incomplete training set), must express its uncertainty in the output. Figure 19 presents outputs of two classifiers that have received insufficient training, the certainty of the classifier output is expressed using pixel intensity – darker pixels

indicate higher certainty of feature detection and lighter pixels show lower certainty.

The feature masks presented in the Figure 19 demonstrate that the classifier is capable of expressing its uncertainty in the output. As can be seen, with Method B, on distinctly identifiable features (buildings, roads), the uncertainty is the highest at the feature borders, making feature maps look blurred. The certainty of the classifier is later utilized for cost map generation and path planning.

Our experience shows that on a training set of 30 000 patterns the peak classification rate is achieved after about 20 epochs of training, before that the classifier output expresses a significant amount of uncertainty. The classification rates for the fen data set are in an expected range, but the classification rates of over 99% on the outskirts dataset are suspicious and a hint for overlearning. The following section makes an attempt to estimate the extent of overlearning on the outskirts dataset.

3.2.1 Outskirts Overlearning Analysis

The high classification rates on the outskirts dataset hint for overlearning; the testing and training datasets might not be large enough or have too high overlap. Visual inspection of a larger classified area (Figure 20) supports the suspicion – the roads within the manually labeled area used for training are detected better than those outside that area. The current section tries to measure the scale of the overlearning problem in the outskirts dataset.

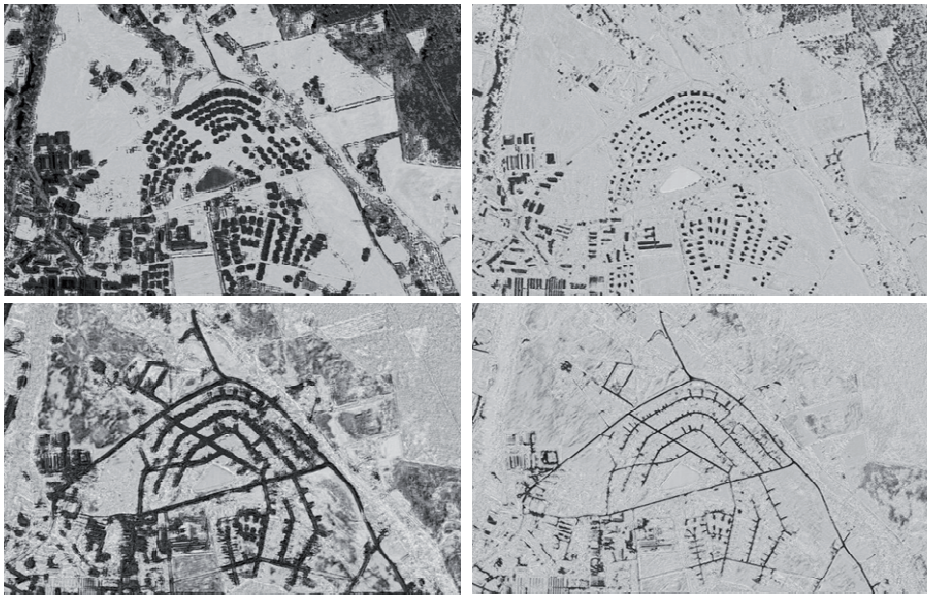

Figure 20. The top row represents buildings and the bottom row roads. The buildings (top row) are well detected within and outside the training area in the middle of the image. Some of the roads (bottom row) outside the training region, however, remain undetected, particularly in the lower left parts of the images

To estimate the extent of the problem we prepared another dataset (Figure 21) for the same outskirts area. The labeling of the new dataset was done using the Estonian Land Board xGIS database; it contains only one feature category – buildings. The reason we use only one feature category is that other features marked in the xGIS database are not coherent with the provided imagery. For example, the roads in the xGIS database do not necessarily overlap with roads on the aerial image and some pieces of roads are not present in the xGIS database at all (Figure 22). We are using the xGIS database because it allows us to label a large-scale map that is prohibitively difficult to label by hand.

Figure 21. Validation dataset with buildings marked with red tint

The xGIS data coherency with aerial imagery (Figure 22) is an important issue during classifier capability evaluation – inclusion of patterns with invalid known values into the training set will degrade the classification rate of the resulting classifier. Even if the GIS database is up-to-date and accurate it is still inferior to manually labeled input. The reason is that not all the features marked on the xGIS database are visible from aerial image: some are covered by vegetation, others are in the shadow of a larger structure or look just like another feature. For example, patterns from a large apartment building with faded tar roofing look just like patterns from a parking lot and training the classifier to categorize the roof patterns as buildings will reduce the classification rate of both buildings and roads. The manual labeling process, however, will assign labels to features based on what they look like, producing data sets that are more suitable for evaluating the classifier capabilities.

To evaluate the classifier performance hand classified input is preferred because it contains features that are actually extractable from aerial image. To

exploit the classifier the GIS database suits better, because even though it has no way of detecting some of the features using aerial image alone, including indistinguishable features in a learning set, it will reduce the confidence assigned to those features. The cost map algorithm uses the classifier uncertainty as an input and if the probability mapping functions are chosen with safe path in mind the path planner will reroute the UGV around the features whenever it is practical.

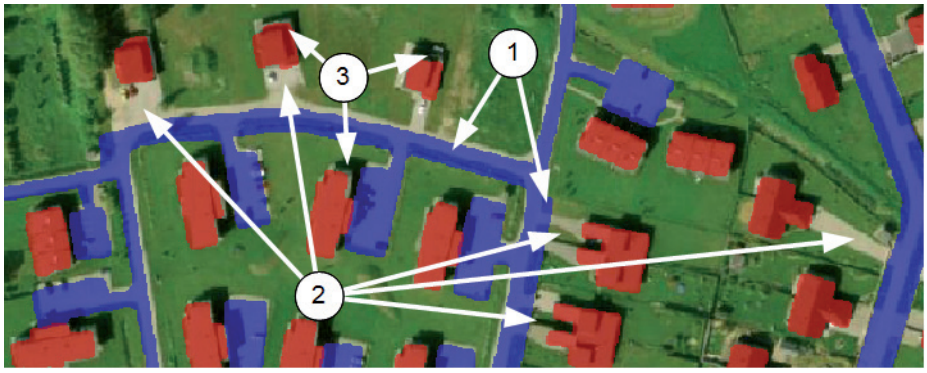

Figure 22. Discord between aerial imagery and xGIS database. The xGIS data for roads (blue) is often shifted relative to aerial imagery (1), the road data overlaps with grass. Some of the areas that could be labeled as roads during manual labeling (2) are not present on the xGIS database. The xGIS data for buildings (red) may be shifted relative to aerial imagery (3)

In our experiment we divided the image (Figure 21) into two – the top half was used for testing and the bottom half for training the classifier. Splitting the image allowed us to eliminate any overlap between the training and the testing set. Because the image is significantly larger than in previous tests we also increased the size of the training set from 30 000 samples to 100 000 to compensate for higher building diversity.

By running the experiment on the new testing set, we achieved Method A classification rate of 91.4%, lower than 99.4% than in the Table 1. The classification rate of the training set is still 98.7%, indicating that for the original outskirts dataset large pattern overlap existed between test and training datasets. The classification rate on the test set for Method B was also 91.4%, lower than 97.9% in the Table 1. The classification rate on the training set was 97.5%, again indicating the overlearning.

By classifying the xGIS outskirts control set with both Method A and Method B, we achieved an equal classification rate of 91.4%. The classification rates were lower than those for manually labeled data (Table 1) due to three reasons: overlearning, dataset quality and reduced feature count. The source of overlearning is the fact that we generated both training and testing datasets from the same area. The dataset quality means that xGIS labels are not coherent with the aerial imagery, producing invalid samples to data and increasing the classifier error rate. The low feature count increases the error rate because by including only one feature into the training set the samples with other features were under-represented. The visual inspection of

an aerial imagery classified with the xGIS classifier labeled some parking lots as buildings. If we had had a good road mask during training, that would have contained the parking lots, the classification rate would probably have been better.

The analysis shows that even though the overtraining problem is real and it exists, its extent is not excessively large. The manually labeled area in the outskirts region was small, causing overlap between training and testing samples, helping the classifier reach over 99% classification rates. The classifier, however, reached over 90% classification rate on an unfair control data set, which is plentiful for the current task.

3.2.2 Classifier Size Analysis

An important challenge with artificial neural networks is to find an optimal size of the network, increasing the size of the network improves its capabilities but also requires more samples to train and more processing power to run. It is particularly important to have training data available; large networks have large sets of internal variables that have to be trained, requiring large sets of labeled samples for training.

The proposed classifier combines two steps in one ANN: feature extraction and classification of extracted features. Our proposed ANN contains five layers, but it can be interpreted as a combination of two separate ANNs: first three layers can be seen as a convolutional ANN with one hidden layer and the last three layers as fully connected ANN with one hidden layer. The third layer of the classifier is then simultaneously an output layer for the convolutional ANN and an input layer for the fully connected classifier. The feature is extracted by the convolutional layers and classified by the fully connected one.

"The approximation properties of feed-forward networks have been widely studied (Funahashi, 1989; Cybenko, 1989; Hornik *et al*., 1989; Stinchecombe and White, 1989; Cotter, 1990; Ito, 1991; Hornik, 1991; Kreinovich, 1991; Ripley, 1996) and found to be very general. Neural networks are therefore said to be universal approximators. For example, a two-layer network with linear outputs can uniformly approximate any continuous function on a compact input domain to arbitrary accuracy, provided the network has a sufficiently large number of hidden units" [40]. Because an ANN with one hidden layer is a universal approximator, it is sufficient to have one hidden layer on the convolutional and one hidden layer on the fully connected part of the classifier. The quest for finding the optimal networks size is reduced to finding the count of neurons on the classifier layers.

The sizes of feature maps on the convolutional layers of ANN are interdependent. In order to connect 29x29 neuron input feature map to the next layer feature map with a 5x5 convolutional kernel, the second layer feature map has to have a size of 25x25 neurons. Because we are skipping every other row and column for subsampling the feature maps on the second layer must to have a size of 13x13 neurons and on the third layer 5x5. The quest for finding the optimal classifier size simplifies to finding the optimal count of feature maps on hidden convolutional layers and finding the count of neurons on the fully connected layer, given a set of training patterns.

We trained a range of classifier configurations on two datasets – a small training set consisting of 500 patterns and a large one consisting of 30 000 patterns from the outskirts area. We trained a list of classifiers on both datasets for 30 epochs and plotted the final classification rate into a chart (Figure 23). The list of classifiers was built around a reference configuration; the additional configurations were generated by halving or doubling the count of neurons on the hidden layers of the reference configuration. For instance, in addition to the reference classifier, we used ones that had 3 or 12 feature maps

on the first hidden layer, 25 or 100 feature maps on the second hidden layer, 50 or 200 neurons on the third hidden layer. We also included multiple combinations of the halved/doubled layers; one configuration where all hidden layers were halved, another one where all hidden layers were doubled, yet another one where only convolutional layers were halved, etc.

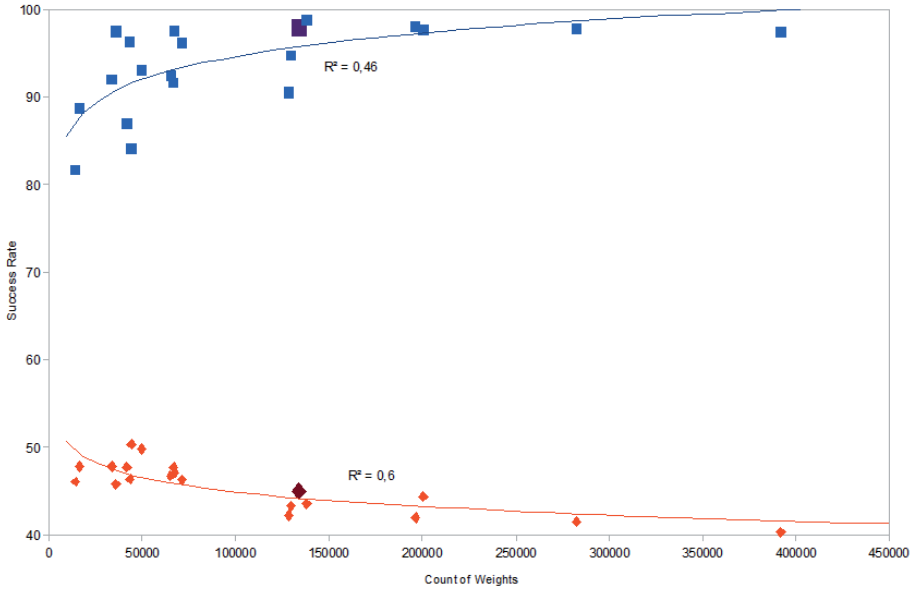

Figure 23. Effect of classifier size on classification capability. The chart contains data from two datasets – a small dataset consisting of 500 patterns (orange) and a large data set consisting of 30000 patterns (blue). The reference classifier configurations are highlighted.

Unsurprisingly, we found that the ANN configurations with lesser internal variables performed better on small datasets and the ANN configurations with large internal variable spaces performed better on larger datasets. We also found that given the fixed amount of training patterns, it is more efficient to increase the count of feature maps on convolutional layers than to increase the count of neurons on the fully connected layer because, due to shared weights, increasing the count of feature maps contributes less to increasing the variable space.

We found that halving the reference classifier size reduced its classification capability on a large training set and doubling the reference classifier size reduced its capability on a small training set without significant improvement on a large dataset. Hence, our chosen reference classifier is a good fit for various scenarios from small training sets containing few hundred patterns to large training sets containing tens of thousands of patterns.

3.3 Cost Map Generation & Path Planning

The main objective of the cost map generator is to translate the classifier output into a form that the path planner can use; not only should it transmit the information about roads and obstacles, but it should also pass the information about classifier certainty. The cost map generator uses feature vectors produced by the orthophoto classifier as an input and produces a cost map used by the path planner as an output. For cost calculation we need to convert the feature vectors to probability vectors by using Bayes' theorem in section 0. The need for the Bayes' function arises from the stochastic nature of the classification problem – the low amount of information about classifiable features leaves room for confusion, which is reflected in the classifier output value.

The conversion for each output can be found using the training dataset; to find the mapping function we split the training set into two – one where the given feature is known to be present on the input pattern and the other where the given feature is known to be missing on the input pattern. After evaluating the training set with the classifier we obtain two class-conditional density functions: $p(x|C_1)$ and $p(x|C_2)$. In addition, we can calculate the prior class probabilities $p(C_1)$ and $p(C_2)$ as ratios of the divided training set sizes to the full training set. Using the class-conditional densities, prior class probabilities, Bayes' equation (8) and the sum rule (11) we can estimate the probability of the feature being present depending on the classifier output value.

To describe the performance of the cost map generator we prepared samples at both locations (outskirts and fen), using both training methods (Method A and B) and using a well- and an under-trained classifier. The under-trained classifiers were prepared to compare the performance and capability to transmit the confidence information of various mapping functions in low-confidence scenarios. In essence, they are classifiers with reference configuration that have had limited exposure to the training set, causing them to have lower classification capability. To illustrate the cost map generator capability we plot its output by converting the cost value into pixel luminosity – black pixels have the lowest traversal cost (roads) and white pixels have the highest traversal cost (buildings, water).

The cost map generated with an untrained classifier has zero certainty and thus the nodes are set to the fixed cost of unknown node c_u. The outputs of the untrained classifier do not depend on the inputs because the randomized weights will have high enough value to saturate the sigmoid functions. The Figure 24 plots a cost map generated with the untrained classifier, it can be used as reference for evaluating the following illustrations: all pixels on "real" cost maps that are darker than the reference are marked to be traversable and all pixels lighter than the reference are obstacles.

Figure 24. The flat cost map generated using an untrained classifier. The image is meant to be used as reference for evaluating the following cost maps – this grade of gray represent the cost of an unknown terrain

The quality of the resulting cost maps and generated paths is evaluated visually. The obstacles must be highlighted on the cost maps with a light color and the roads should be marked with a black color; none of the obstacles should be marked as low cost area. The cost map generated using an under-trained classifier must have lower contrasts than the one generated using a well-trained classifier because the cost map generation algorithm eq. (11) gravitates the low confidence classifier outputs towards the cost of unknown terrain c_u. In addition, the generated path should follow roads and avoid obstacles whenever feasible.

3.3.1 The Outskirts

The outskirts sample (Figure 26, Figure 27) contains a small subset of the test image, the area was chosen small for illustrative purposes. The objective is to generate a path from one end of a cul-de-sac to the end of another cul-de-sac avoiding obstacles and seeking roads whenever possible. The safe path from start to finish follows the roads, but there is a possibility to take a shortcut over the grass which is riskier because of a pile of debris nearby (Figure 25).

Figure 25. Outskirts test case – the path starts and ends at cul-de-sac. The objective is to generate a path from start location to finish avoiding obstacles and preferring roads. Either one of the hand drawn paths are desirable outputs for the path planner

To evaluate the training method on the outskirts dataset using classification Method A we prepared a well-trained classifier with a combined classification rate (rate of correctly identifying all features on the input pattern) of 99% and a under-trained classifier with a combined classification rate of 31%; the combined classification rates with Method B were 89% and 43%, respectively. The classifier detects four feature categories on the input image: buildings, roads, grass and debris, for path planning we assigned category weights of 30, -30, -20 and 20, respectively; the unknown terrain cost c_u is set at 30. The weights are chosen so that when a classifier detects both buildings and roads at the same location with equal uncertainty, the weight of buildings (30) will cancel out the roads (-30) and the location will have the cost of unknown terrain c_u.

As explained section 0, Method A has low spatial resolution. For any given location on the input image the classifier will detect all features that are within certain radius from the location. The radius of the circle is equal to the diagonal of the classifier input pattern, for the Figure 26 the radius is 32 m. Since the input pattern consists predominantly of small buildings and roads that are surrounded by grass, it is detected almost everywhere on the input pattern causing the generated cost maps (Figure 18) to be offset by the weight of the grass (~20 m). The effect does not affect the traveling cost of roads, because they are already at minimum value, but it will bring down the traveling cost of buildings close to that of the unknown terrain. This can be observed on generated cost maps (Figure 26) – the buildings are predominantly gray, close to the unknown terrain.

As expected, the cost map generated using an under-trained classifier has lower contrasts caused by low confidence of the classifier. The obstacles stand

out from the background and the outputs from the path planner follow desired, albeit different paths.

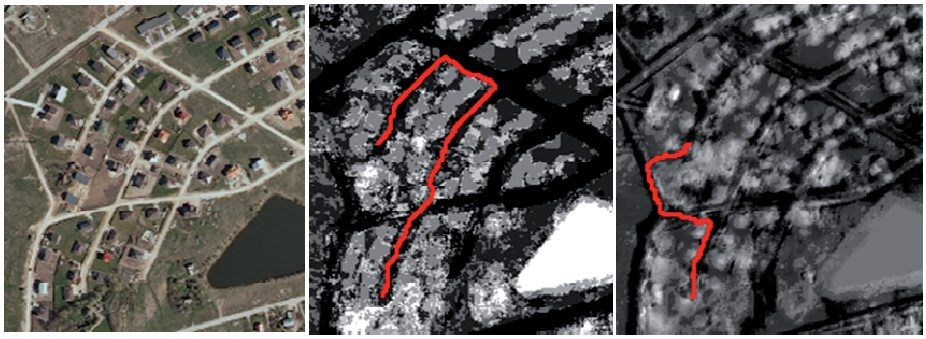

Figure 26. Input (left), generated cost maps and traveling paths for a small subset of the outskirts region using training Method A. The first cost map and path (middle) is generated using a well-trained classifier. The right cost map is generated using an under-trained classifier. Weights used for buildings, roads, grass and debris are 30, -30, -20, 20

The training Method B produces a classifier that has pixel-precise output allowing more detailed cost maps (Figure 27). All features are well detected, but the classifier has high uncertainty at the feature borders – at road edges and at building walls. The classifier uncertainty manifests correctly in the cost maps: the borders of the feature mentioned have the cost of the unknown terrain and the well-trained classifier produces higher contrasts than the under-trained classifier. The path generated using the well-trained classifier follows the desired route, but the under-trained solution took a longer route. The detour was caused by the under-trained classifier, misclassifying the beginning of a street as an obstacle. The path planner, however, fell back gracefully rerouting, using a parallel street.

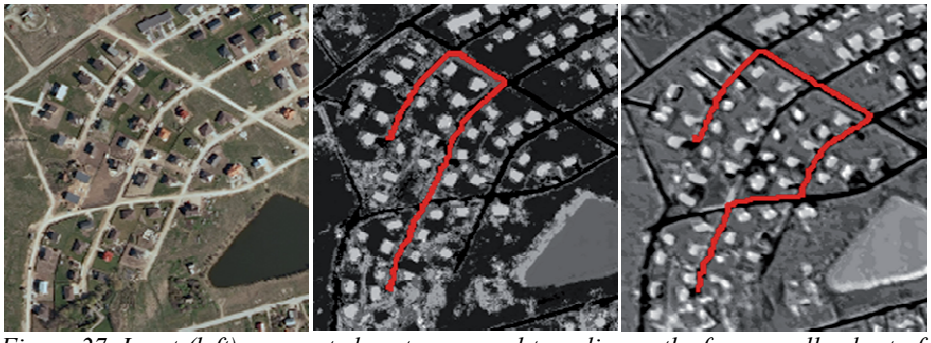

Figure 27. Input (left), generated cost maps and traveling paths for a small subset of the outskirts region using training Method B. The first cost map and path (middle) is generated using the well-trained classifier. The right cost map is generated using the under-trained classifier. Weights used for buildings, roads, grass and debris are 30, -30, -20, 20

Overall, the training Method B performs better in the suburban environment due to higher spatial resolution – the produced cost maps are easier to validate for a human operator. Both methods produced traversable if not desirable paths even in a low certainty condition.

3.3.2 The Fen

On the fen sample (Figure 30, Figure 31) we chose the start and finish to be on the two ends of a waterlogged track that crosses the area (Figure 28). The tracks are filled with water and are hard to distinguish from the background, on harder patches of land they disappear altogether. The path must cross drainage channels (leading to the stream present in the area) at two locations – preferably at fords that are not explicitly marked, but may be predicted by following tracks. The sample area is much larger than the one chosen for the outskirts covering about 1 square kilometer, the approximate length of the desired path is about 2 km.

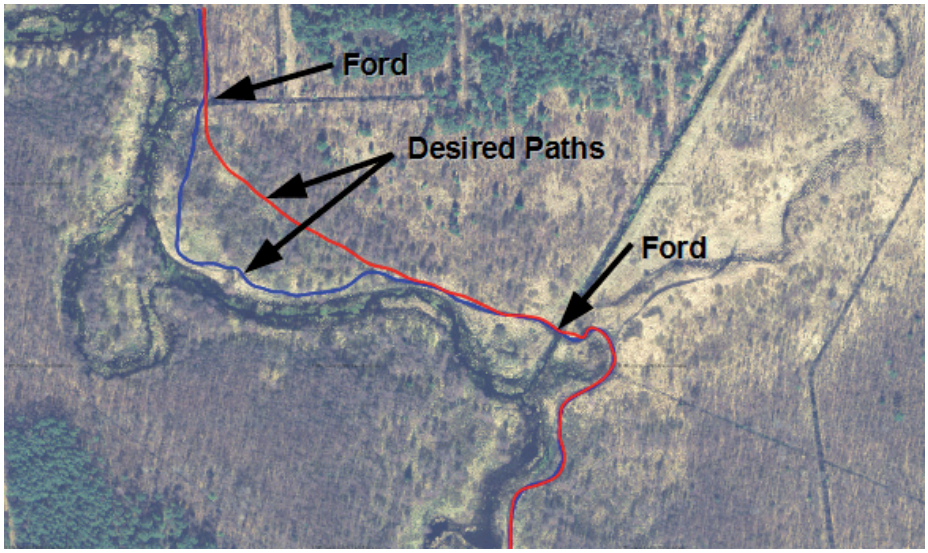

Figure 28. The fen test case. Blue and red lines are desired paths for a path planner

The well-trained classifier prepared for the experiment had a combined classification rate of 91% with Method A and 86% with Method B. The under-trained classifier had a classification rate of 52% with Method A and 73% with Method B. The classifiers were trained to detect three feature classes on the input image – water, roads and trees; the feature weights used for cost map generation were 30, -30 and 20.

Outskirts area was fully covered by feature masks of the four selected input, making the outskirts area fully defined. The three classes defined for the fen, however, cover only a small proportion of the total input. Large patches of input image contain no defined features and the cost map must associate them with the cost of the unknown terrain. Of the features present only the trees are visually well distinguishable; the stream banks are under

vegetation and thus difficult to recognize. Of the three, the tracks are most difficult to detect – during manual labeling we could pick up the track from where it is well visible and trace it from there until it disappears, but the classifier evaluates the map pattern by pattern and has to make the judgment based on local information. To detect a track on an input pattern it has to be clearly visible and distinguishable from the background.

A significant part of high error rate for tracks responsible for a large amount of false positives in Figure 30 and Figure 31, can be attributed to overlabeling of the input image during the manual labeling step. Closer examination of the manual labeling output revealed that the tracks are traced further than they are visible, to the point where they can be guessed by a gap between bushes (Figure 29). The overmarking introduced patterns to the training data that claimed track presence but contained no evidence of the tracks confusing the classifier.

Figure 29. Overlabeling of the fen: The barely visible track disappears on the harder surface, but manual labeling followed the road using extra clues from the gaps between the bushes, confusing the classifier

The cost maps generated using the classifier that was trained using Method A (Figure 30) are excellent for path planning, the low spatial resolution of Method A is not an issue on the large open environment. The stream and drainage channels are detected and the tracks are recovered with high enough certainty to use them in path planning. The produced paths followed the tracks where they were visible and crossed the drainage channels at or near the fords, avoiding crossing the stream.

Comparing the well-trained classifier output to the one of the under-trained classifier is difficult because of the large featureless regions in the input. The trees are distinctly visible on the input pattern and are correctly detected even with the under-trained classifier, but the stream is less precisely defined and the under-trained classifier has high uncertainty on the water output. Both the well-trained and the under-trained classifiers struggle with detecting tracks due to overlabeling and have noise in the output, albeit noise with low certainty.

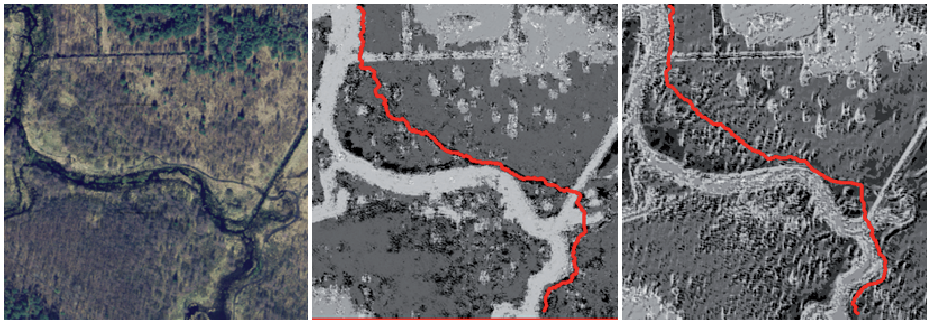

Figure 30. Cost map generation and path planning in fen using classifier training Method A

The classifier training Method B (Figure 31) exhibited similar behavior during cost map generation to Method A (Figure 30). The path generated using the well-trained classifier closely followed a desired route even though the under-classified did not cross the first channel at for but near it. The classifier again struggles with tracks producing an output with significant noise. The noise, however, was less apparent with the under-trained classifier that had low confidence in the track output.

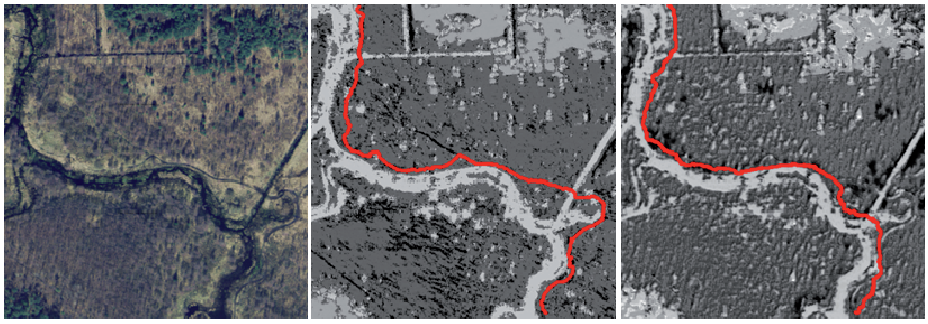

Figure 31. Cost map generation and path planning in fen using classifier training Method B

The spatial resolution of Method B might even be a disadvantage because at some patches of the stream the floating water plants were classified as unknown terrain, while the less accurate Method A marked the whole area as an obstacle.

Both classifier training methods are suitable for aerial imagery analysis. The classification Method B has an edge in the dense suburban environment while the less precise Method A has a slight edge in the natural environment. The cost map generation algorithm is capable of expressing the uncertainty of the classifier and the path planner uses that information during routing.

3.4 Reversed Processing Pipeline

Reversing the processing pipeline and classifying aerial image on demand basis may significantly reduce the area that needs to be classified. The terrain classification is an expensive step in terms of time and energy. Wasting computing resources of our battery powered UGV on unnecessary image classification takes away energy that could be used for extending the mission.

The samples in Figure 32 illustrate the area used for path planning on the outskirts imagery, the cost map for those samples are plotted in (Figure 27). The pixels actually used for path planning are highlighted with white color. For the well-trained classifier (left image in Figure 32) the area used for path planning was just a small fraction of the whole input image while for the under-trained classifier (right image in Figure 32) the area used was even larger than the plotted region. Both of the used areas are significantly (90% and 30%) smaller than the overall aerial imagery used for the input, reducing the classification time and saving energy. This falls in line with our overall experience – the higher the uncertainty on the cost map the larger area is used for path planning.

Figure 32. Path planning in the outskirts – highlighted area marks the nodes used by the path planner. The path on the left image is produced using the well-trained classifier and the path on right image is produced using the under-trained classifier

The featureless natural environments (left image Figure 33) have high classifier uncertainty and thus the savings from reversing the pipeline are minimal. Combination of featureless natural environments with the under-trained classifier (right image in Figure 33) produces worst case cost maps whereas the effect of reversing the processing pipeline is negligible – the cost map generator has to classify practically the whole mission zone.

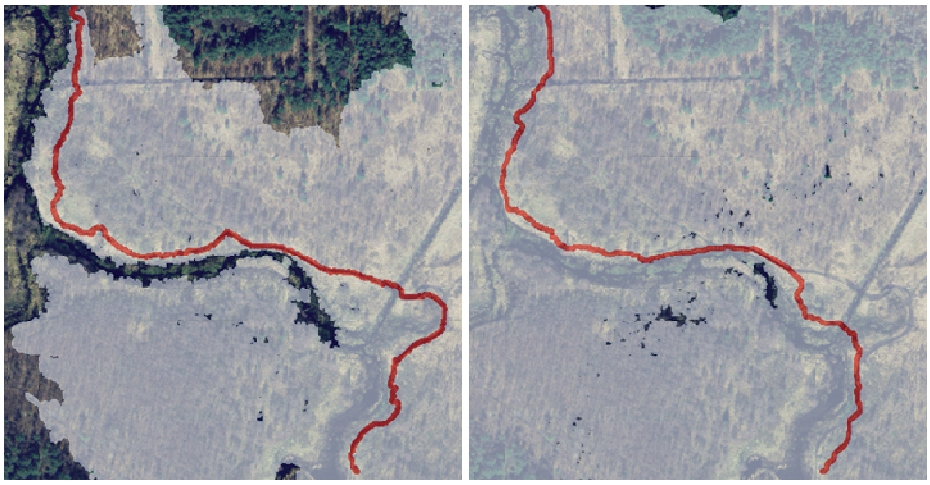

Figure 33 Path planning in fen – highlighted area marks the nodes used by the path planner. The path on the left image is produced using the well-trained classifier and the path on the right image is produced using the under-trained classifier

Our experiments have shown that the reversing of the processing pipeline has the greatest effect on well-defined environments where a clear path to the designated location exists. The optimization does not affect the worst case scenario – in natural environments with high uncertainty we still need to classify the whole aerial image for path planning. The achieved reduction of the overhead imagery classification time is over 90% in urban areas and less in natural environments.

3.5 Implementation & Performance

In the numerical experiments custom-made software was used. The software allows us to generate the training and testing datasets, to train and evaluate the classifier, to generate the cost map and to plan a path from a given start point to a given destination. The objective of this section is to sketch the structures of the prepared programs without detailed implementation, advanced functionality or specific optimization techniques used. The second objective is to describe challenges and a solution to an efficient classification of a large scale area.

3.5.1 Reference Implementation

For our numerical experiments we built an artificial neural network on a high level programming language, namely C#. For building the ANN four kinds of programming objects were used: Layers, Neurons, Connections and Weights (Figure 34). All neurons in an ANN were organized into five layers; neurons within a layer are independent of each other (not connected to each other). During the evaluation of the ANN the layers are executed one by one in a serial manner. Because the neurons within a layer are independent, they can be evaluated in parallel, achieving performance boost from multicore processors.

The neurons in different layers are connected to each other using connection objects. The connection objects are necessitated by usage of convolutional layers; they allow us to selectively connect a neuron to any other neuron by a weight object. Because a convolutional layer uses shared weights, we also need the weight objects, multiple connections may reference to the same weight object. The architecture is generic enough to be used for all layers in the classifier.

The neurons in the convolutional layers are connected to the previous layer using 5x5 shared kernels. In practice it means that each neuron in a convolutional layer maintains a list of 26 connection objects (25 + bias) per feature mask in the previous layer. The connection objects within a single kernel will all refer to the same 26 weight objects. The neurons on the fully connected layers, however, are connected to all previous layer neurons. In a program it means that each output layer neuron is connected to 101 (100 neurons + bias in a reference configuration) neurons on the previous layer, each connection object references to a unique weight object.

All the information about the ANN is contained within the objects. Neurons maintain a list of connections to the previous layer and an output value that is used during the forward propagation phase. In addition, the neurons maintain a list of connections to the next layer and an error value that is used during the back propagation step. Weights support value, learning rate and diagonal hessian properties; the latter is used by the gradient based learning technique. Connections maintain references to the previous layer

neuron, the next layer neuron and weight objects. Finally, the layers maintain a list of associated neurons.

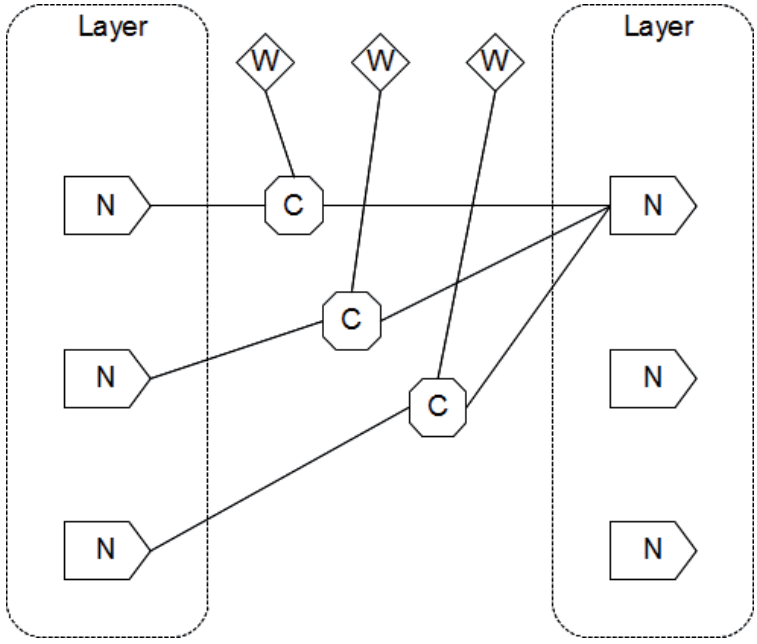

Figure 34. Classifier architecture. The neurons N are assembled into layers, the neurons on different layers are connected using connection C, using weights W

3.5.2 Heterogeneous Computing

The classifier implemented in a high level language has a respectable performance of 800 input patterns per second with a reference configuration on a quad core i7-920 processor, but for evaluating large aerial images it falls short. To classify a 1 square kilometer area with a resolution of 0.8 m per pixel we need to evaluate roughly one and a half million patterns, with the reference classifier it takes half an hour to evaluate the area. Having a 130W CPU running for half an hour at maximum power consumption to calculate few kilometers ahead is not a viable solution for a battery powered mobile platform.

A solution to our performance problems is heterogeneous computing hardware. Focus in the last 40 years of CPU and compiler development has been mostly on improving the performance of single threaded applications [50]. The first 30 years of processor development saw a steady rate of serial workload performance improvements from ever increasing processor clock speed and smarter compilers, allowing software developers to be “lazy”, implementing predominantly serial algorithms. The large legacy of applications relying on serial algorithms has forced the processor makers to seek additional ways to improve single threaded performance, besides shrinking node size and improving clock rate. Modern CPUs support a host of technologies that sacrifice die area and power for improved single threaded

performance, such as out of order, speculative, superscalar execution, pipelining, and caching. The continuous pursuit for higher serial performance has led to current situation where it takes an order of magnitude more energy to schedule an instruction than to execute it.

The last decade has seen the emergence of affordable hardware that is focused on parallel workloads – GPUs. Historically the development of GPUs was fueled by CAD/CAM applications and 3D gaming. First 3D games were running on CPUs but a desire to improve screen resolution and frame rates led to dedicated fixed function hardware. Demand for improved graphics led to introducing programmability into some steps of fixed function pipeline; the per pixel algorithms used for lighting and shadowing in particular led to a massively parallel low latency architecture. Further improvements in the field led to abolishment of fixed function pipeline in favor of fully programmable hardware that can be used for general purpose computing, supported by a few fixed function units.

Modern GPUs dedicate most of their die area to simple processors that are optimized for executing mathematical functions; the foundations of 3D gaming, after all, are mathematical. Like CPUs the GPUs have hit a power wall, but have chosen different compromises. The CPUs push high clock rates for higher single threaded performance while GPUs utilize lower clock rates and larger dies with more processing units instead (near the limit the clock rate and energy consumption are exponentially linked). For example, top-of-the-line Intel i7-3960X CPU supports 6 cores running at 3.3 GHz versus top-of-the-line AMD Radeon HD 7970 GPUs 2048 cores running at 925 MHz. Instead of using large on-die caches the GPUs utilize high bandwidth off-die memory bus.

The heterogeneous computing is an umbrella term for systems that utilize a variety of processing units. There are multiple software APIs available for exploiting parallel hardware, such as nVidia's CUDA, Microsoft's DirectCompute or AMD's Stream. We chose to use a vendor independent API supported on multiple platforms including CPUs and GPUs: OpenCL.

The OpenCL is not meant for writing applications, it is rather intended to accelerate the performance critical parts of the program by running it on parallel hardware. It can not automatically accelerate existing applications, as it requires extensive modifications and restructuring for algorithms to utilize the OpenCL capabilities.

The centerpiece of an OpenCL is a notion of computing kernel. A computing kernel is a function that implements a relatively simple algorithm and is meant to be executed in parallel. For example, to multiply two 1000x1000 matrixes we can write a kernel that computes the value of a single element in the resulting matrix and schedule 1 000 000 of those kernels to be launched on parallel hardware. Each computing kernel in this case is responsible for calculating a single element on the result matrix.

The OpenCL supports two programming models – *data parallel* and *task parallel* approaches. The data parallel approach is useful when a large

problem can be divided into smaller sub-problems that can be executed in parallel. Like in the example above where multiplication of two 1000x1000 matrixes was divided between 1 000 000 tasks, each responsible for calculating one element in the result matrix. Task parallel approach is useful when a simpler algorithm has to be applied on a large dataset. For example, to calculate the determinant of a million 10x10 matrixes we can schedule 1 000 000 threads, each responsible for calculating the whole determinant of one matrix.

To classify a large overhead image that covers multiple square kilometers, both approaches are viable – we can calculate each neuron output in a layer in parallel resulting in the data parallel approach and we can evaluate all patterns in the input image in parallel with each kernel, evaluating the whole ANN resulting in the task parallel approach. To find out which approach is better we implemented both.

For the task parallel approach we implemented a kernel that executes the whole forward propagation step of the classifier in a serial manner. The task parallel approach is very reminiscent of embedded programming: there are many hardware limitations that have to be avoided and all data structures must be explicitly allocated at compile time, but in comparison to the data parallel approach the programming is quite straightforward. Each computational kernel extracts one input pattern from the aerial image using fixed function texture hardware, initializes ANN inputs, evaluates the ANN layer by layer and saves the outputs into the pre-allocated memory area. By running the kernels in parallel we can evaluate multiple patterns at the same time.

For the data parallel approach we need to dismantle the classifier – it consists of five layers that must be executed one by one in a serial manner, but each layer itself can be evaluated in a parallel manner. We implemented the data parallel approach using four different kernels that process input image one pattern at a time. The first kernel loads the pattern from aerial image and initializes the network input layer using 29x29=841 threads; each thread is responsible for fetching one pixel from the input image and coping the individual color values to the memory buffer allocated for first layer neurons. The second kernel evaluates the first hidden layer using 13x13x6=1014 threads; each thread is responsible for evaluating one neuron output. The third kernel evaluates the second hidden layer using 5x5x150=1250 threads; again each thread calculates one neuron output. The last kernel is used to calculate the two last fully connected layers, to increase parallelism we use up to 512 threads per neuron. Each thread is responsible for multiplying a portion of connected weights and previous layer outputs, the portions are added together using three parallel reduction steps.

There is an alternative way to evaluate a large overhead image with the data parallel approach: space-displacement neural networks (SDNN) [45]. When overlapping patterns are evaluated with a convolutional ANN, some of the calculations are repeated, because the convolutional kernels that are applied to the overlapping area produce the same outputs. The SDNNs

eliminate the repeated calculations by increasing the input pattern size to contain the whole image; instead of overlapping input patterns that make up an image it has one pattern that does not have overlap (Figure 35). The input pattern size increase also affects the size of convolutional layers. The fully connected layers will transform into convolutional layers with shared weights. Effectively the SDNN is a large convolutional neural network that produces the same output that we normally would achieve by applying a small convolutional network to all locations on an input image. To train the weights in the SDNN we still use the conventional convolutional ANN, we only use the SDNN for speeding up the evaluation of a large overhead imagery.

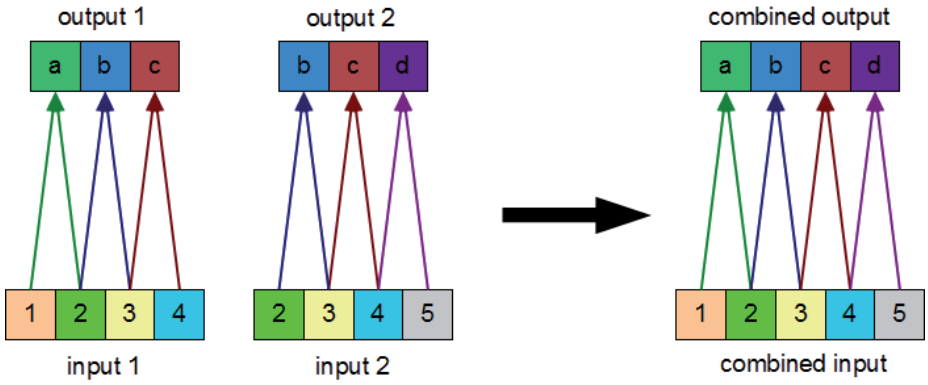

Figure 35. Evaluating overlapping patterns has repeated calculations in a convolutional layer. SDNNs reduce calculations by combining the overlapping input patterns into a single larger input pattern evaluated by a single ANN

Again, an SDNN is a convolutional ANN with the input layer as large as the input image and the output layer 4x smaller in both x and y directions due to double subsampling. The large feature maps of SDNN enable us to examine the inner workings of the convolutional neural networks. The Figure 36 enlists a few feature maps from the hidden layers for the image in Figure 14. Because the SDNN will transform the fully connected layer with 100 neurons into a convolutional layer with 100 feature maps, we can also plot feature maps from the third layer. It is easy to visually verify the features a feature layer extracts. For example, the first pattern on the leftmost column in Figure 36 extracts the roads (the features are substantially darker than the background) from the input image, the second pattern extracts grass from the input image and the third pattern extracts buildings.

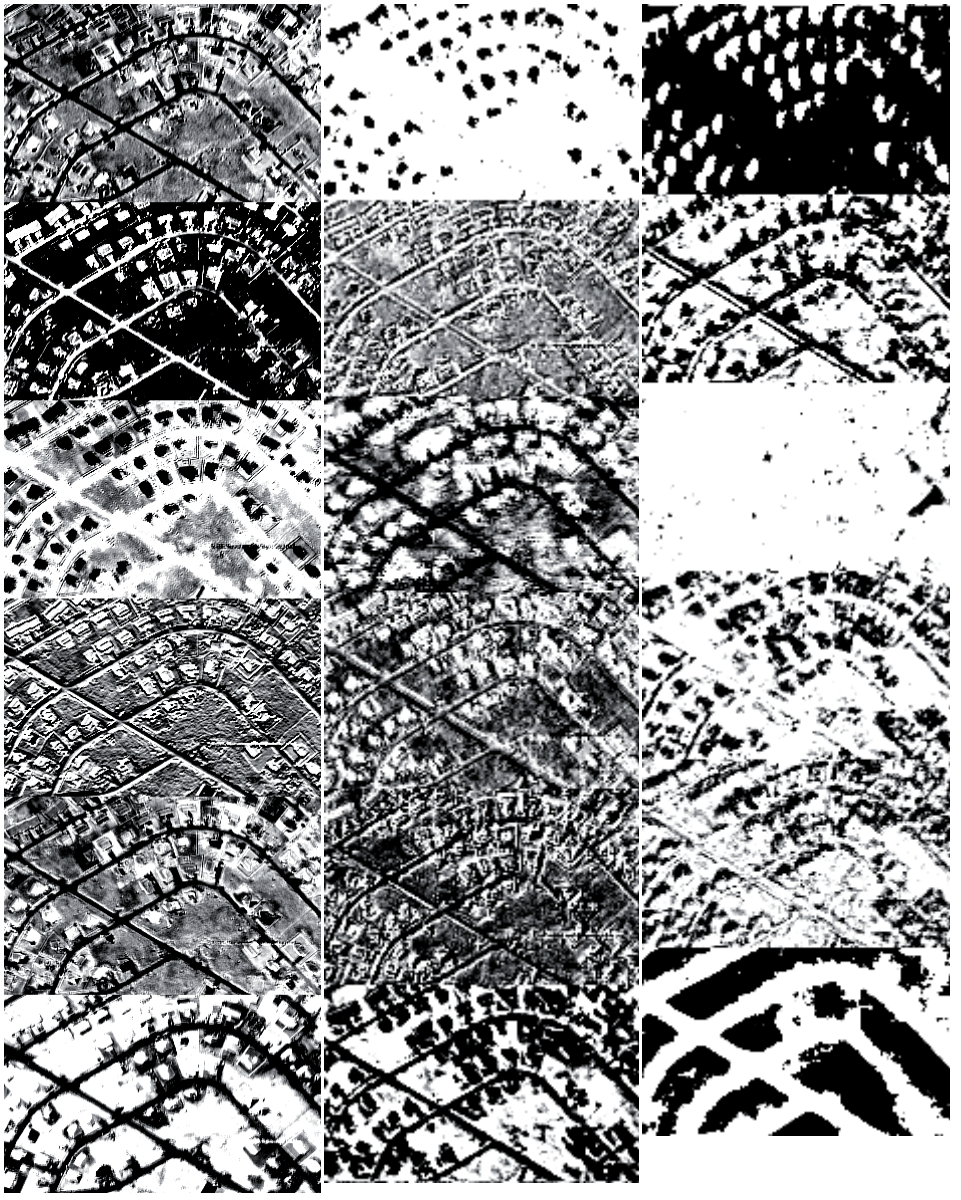

Figure 36. Internal states of an SDNN. Leftmost image plots the 6 feature maps of the first hidden layer, middle image plots 6 of 50 feature maps on the second hidden layer and the right image plots 6 of 100 feature maps on the third hidden layer. The feature maps are representative samples from each layer and are not directly related

3.5.3 Performance

For performance evaluation we measure three parameters – throughput and latency and energy requirement. The throughput is the performance of the system in terms of processed patterns per second. The throughput is main concern during the classification of a large overhead image that consists of

millions of patterns. Energy consumption, like the throughput, is relevant during the evaluation of overhead imagery; evaluating a large overhead image with a reference classifier on a 130 W CPU may take hours, making the system unsuitable for a battery powered vehicle. We define latency as time it takes from presenting the classifier with an input pattern to having the classification result. The latency is important during the training phase where inputs are presented to the classifier one by one.

We observed the throughput of the task parallel approach to be higher than the data parallel approach on a large input size (Figure 37) due to higher parallelism that helps to hide memory latencies. Because of the small caches the GPUs are very sensitive to memory access patterns and the semi-random neuron access pattern in convolutional layers diminishes the performance. On highly parallel tasks the memory latencies can be hidden – when a computing kernel thread is blocked by a memory access it will suspend the current thread and perform calculations of another scheduled thread, the suspended thread will be resumed after the memory access operation is completed. The memory latency hiding only works if there is enough scheduled threads per execution unit, this explains the dependency of the task parallel approach on the input dataset size.

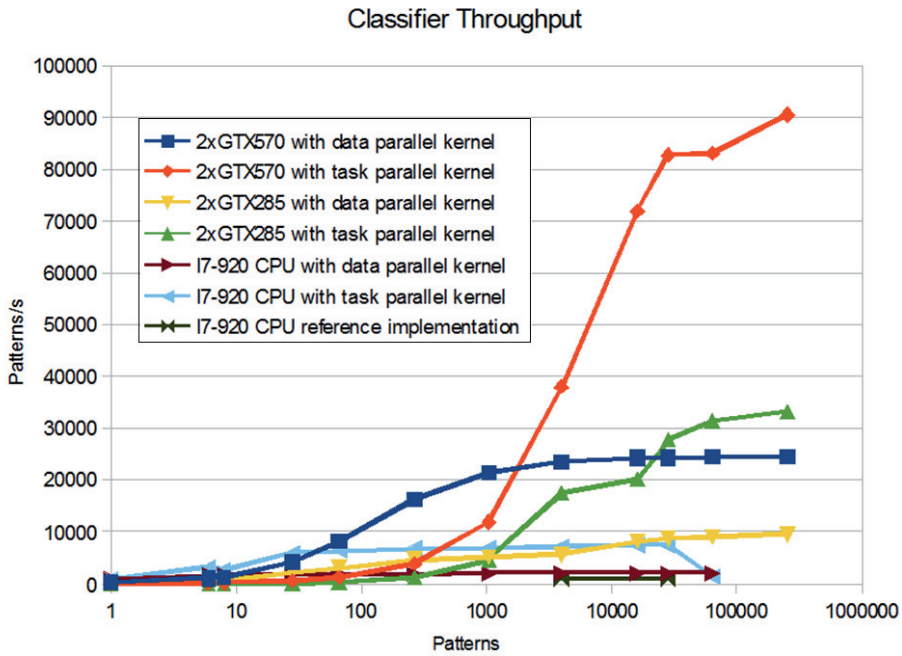

Figure 37. Classifier throughput in patterns per second on different platforms. The chart includes set up time that is spent on allocating device memory and copying data over from RAM to the device memory. The performance of the data parallel approach improves with a higher pattern count because the proportion of the setup time to the execution time decreases

The training of the classifier is done one pattern at a time, to improve the training speed we need to reduce the time it takes to evaluate a single pattern. Thus for training we can not use the approaches that rely on batch processing such as SDNN or task parallel approach. Our experiments (Table 2) show that the evaluation of patterns on a heterogeneous platform and training on the CPU is not a viable approach because of the time it takes to transfer data to and from device memory diminishes all benefits of high computing capability. To take advantage of the computing capacity the whole forward and back propagation cycle has to perform on the GPU. We do not have an optimized classifier implementation for the GPU that does the training step but according to our experience the back propagation step is about 2.2 times slower than the forward propagation on the data parallel kernel. Considering the peak evaluation rate of the data parallel kernel on a single GTX 570 GPU is about 12 000 patterns per second we should achieve the training rate of 5500 patterns per second with a reference classifier configuration.

Table 2. Time it takes to evaluate a single pattern.The values include set up time required for allocating device memory buffers and for transferring data to and from the device memory

System configuration	Latency [ms]
Dual GTX 285 with data parallel kernel	20.3
Dual GTX 285 with task parallel kernel	211
Dual GTX 570 with data parallel kernel	3.4
Dual GTX 570 with task parallel kernel	51
i7-920 CPU with data parallel kernel	12
i7-920 CPU with task parallel kernel	3.7
i7-920 CPU with reference implementation	1.3

The reference C# classifier has the throughput of 800 patterns/s on i7-920 CPU. As calculated above, it takes half an hour to classify 1 500 000 patterns required for evaluating 1 square kilometer of area at 0.8 m/pixel image resolution. The task parallel GPU classifier has 90 000 patterns/s on 2x GTX 570 GPU, to evaluate the same area on the GPU it takes only 17 seconds. By implementing the slowest step – classification of a large aerial image – on a heterogeneous computing architecture we gained over 110x improvement in the throughput.

Classification of 1 square kilometer of terrain with reference C# implementation on 130 W CPU took half an hour and 234 kJ of energy; evaluating the same area on 440 W GPUs only takes 17 seconds and 7.5 kJ of energy, reducing the energy consumption by more than 30x. The energy gain is even more significant if we take the whole computer power into account.

The throughput of the SDNN is not included in the Figure 37, but we achieved the throughput of over 65 000 patterns per second with a single GTX 285 GPU or over 360 000 patterns per second on a single GTX 570 GPU. The reported throughput is measured at the output of the SDNN, it is already

compensated for the resolution reduction from subsampling. We did not evaluate the classification rate of the SDNN on a dual GPU system, but by splitting the input image into two and evaluating the halves on separate GPUs we can achieve nearly double performance. The performance of SDNNs can be attributed to high parallelism and memory access efficiency – it has comparable parallelism to the task parallel approach and comparable memory access efficiency of the data parallel approach. Classification of a 1 square kilometer area with SDNNs using the 440 W GPUs took 2.08 s and 915 J of energy, making it over 860 times faster and 250 times more energy efficient than the CPU based reference solution, even when not considering the power consumption of the rest of the computer.

To test the concepts proposed in this thesis we built a high-performance classifier using a high level programming language. By implementing the classifier on heterogeneous computing hardware we gained over 100x classification throughput and over 30x energy efficiency over the reference implementation. By further optimizing the calculations using space displacement neural networks we gained over 800x classification throughput and over 250x energy efficiency over the reference classifier. Using GPU implementation it takes nine days to evaluate the whole 45 000 square kilometer surface of Estonia with a $1000 PC, much less than 2.7 years with the CPU based reference implementation. Classifying the same area with SDNNs takes only 1.7 hours, but results in a cost map that has 4x lower resolution in both horizontal and vertical direction.

4. Conclusions

4.1 Summary

The current thesis proposes a long-range navigation system that is capable of learning from an UGV local navigation system and is able to extrapolate the gathered knowledge beyond the UGV's immediate perception range. The system is adaptive, able to learn from the UGV on the go, making it suitable for navigating in unknown or changing environments. The "deep learning" approach has produced a system that is capable of navigating in both structured and unstructured environments using aerial or satellite imagery.

Classification of an aerial imagery is a difficult task that is prone to errors. A part of this thesis is dedicated to finding the confidence of the overhead imagery classifier and utilizing it during path planning. Our numerical experiments on the manually labeled real terrain show that the system is capable of expressing its uncertainties. The uncertainties can be used for fusing local and global navigation systems. The UGV can concentrate on local navigation capabilities on an unknown terrain that has low confidence and follow the generated route through areas with high confidence.

To use the proposed system on a battery powered UGV is challenging due to high computing capacity requirement. Processing of a large overhead imagery with the proposed classifier requires excessive amounts of computing resources that consume equally large amounts of energy. The thesis proposes a cost effective solution to the performance challenge using heterogeneous computing hardware. The resulting increase in speed and reduction of energy consumption over the CPU based implementation is more than two orders of magnitude.

Even though the lack of real world experiments with working UGV and UAV systems decreases the credibility of this work, the results obtained from manually labeled areas indicate feasibility of the proposed system for long-term navigation. The adaptiveness of the system supported by high classifier capability and built-in failsafe make it suitable for a wide range of UGV solutions.

4.2 The Main Scientific Contributions

- Development of a novel intelligent long-range navigation method for an off-road vehicle that relies on monocular overhead imagery.
- Proof of the suitability of convolutional neural networks for overhead imagery classification.
- The developed method has built-in failsafe: the uncertainty of the classifier can be measured using prior knowledge. The UGV can fall back to the onboard navigation system in areas where the certainty of the long-range system is low.

4.3 Future Work

Our UGV robot is currently incapable of labeling the environment around it, forcing us to perform numerical experiments with manually classified data. For manual labeling we used relatively specific classes (building, road, grass) that were easily identifiable for human rather than generic classes (obstacle, clearance) that are more natural for an UGV [33, 51]. However, it has been demonstrated [51, 52] that the classifier trained with Method B is capable of detecting generic classes. We suggest that the same applies to training Method A, but it has to be confirmed experimentally.

Using generic classes simplifies inclusion of new features because the feature set has a fixed size. With specific features it is required to add another output to the classifier every time the UGV detects a new feature class. With generic features it is only required to include the feature in the training set, there is no need to reconfigure the classifier. Another upside is that there is no need for the UGV to explicitly define the new feature; it only has to mark it as an obstacle or clearance. During the following update cycles the classifier will learn the feature, will be able to recognize it on the aerial imagery and use it for path updating.

The classifier using generic feature classes has constant configuration and predictable performance that does not depend on the specific count of detected features. The variable space of that classifier is fixed, there is a limit of how many features it can learn to recognize – the UGV might meet more obstacle types (buildings, trees, rocks, lakes, etc) than the fixed size classifier is capable of learning. Our proposed system has built-in failsafe: if the feature count increases beyond the limit, the classification capability degrades and the confidence of the system declines. As the confidence decreases, the UGV will de-emphasize the long-range navigation system and falls back to onboard navigation capability.

The fixed variable space necessitates fixed training set sizes, the patterns have to be managed in the training set. A classifier with the finite variable space is incapable of learning an infinite amount of feature classes, forcing us to limit the training set size. Hadsell *et al*. have suggested [49] using a balanced ring buffer as a short-term memory. This could be coupled with another buffer that would act as a long term memory. Maintenance of the long term memory, however, is an open topic.

The ultimate intention is to run the system online on an UGV in a cyclic manner. After our UGV obtains labeling capabilities we need to reassess the chosen classifier configuration, reevaluate the classifier performance and test the system in real world experiments.

References

1. **Urmson C.** *The self-driving car logs more miles on new wheels*, [Online] http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html (7 Aug 2012).
2. **General Motors**. *Self-Driving Vehicles Could be Ready by End of Decade*, [Online] http://media.gm.com/content/media/us/en/gm/news.detail.html/content/Pages/news/us/en/2011/Oct/1016_autonomous.html (16 Okt 2011).
3. **Jackel L. D., Krotkov E., Perschbacher M., Pippine J. and Sullivan C.** *The DARPA LAGR program: Goals, challenges, methodology, and phase I results*, J. Field Robotics, 2006, vol. 23, pp. 945–973.
4. **Stentz T., Kelly A. and Rander P.** *Integrated Air/Ground Vehicle System for Semi-Autonomous Off-Road Navigation*, Robotics Institute, 2002, p. 18.
5. **Stentz A., Kelly A., Rander P., Herman H., Amidi O., Mandelbaum R., Salgian G. and Pedersen J.** *Real-time, multi-perspective perception for unmanned ground vehicles*, Robotics Institute, 2003, p. 16.
6. **Vandapel N., Donamukkala R. R. and Hebert M.**, *Unmanned Ground Vehicle Navigation Using Aerial Ladar Data*, The International Journal of Robotics Research, 2006, vol. 25, pp. 31–51.
7. **Stefanik K. V., Gassaway J. C., Kochersberger K. and Abbott A. L.** *UAV-based stereo vision for rapid aerial terrain mapping*, GIScience & Remote Sensing, 2011, vol. 48, pp. 24–49.
8. **Vandapel N. and Hebert M.** *3D rover localization in airborne ladar data*, Experimental Robotics VIII, 2003 pp. 156–167.
9. **Wang O., Lodha S. K. and Helmbold D. P.** *A bayesian approach to building footprint extraction from aerial lidar data*, 3D Data Processing, Visualization, and Transmission, Third International Symposium on, 2006, 192–199.
10. **Charaniya A. P., Manduchi R. and Lodha S. K.** *Supervised Parametric Classification of Aerial Lidar Data*, Computer Vision and Pattern Recognition Workshop, 2004, p 30.
11. **Silver D., Sofman B. and Vandapel N.** *Experimental Analysis of Overhead Data Processing To Support Long Range Navigation*, Intelligent Robots and Systems, International Conference on, 2006, pp. 2443–2450.
12. **Vandapel N. and Hebert M.** *Finding Organized Structures in 3-D Ladar Data*, Intelligent Robots and Systems, International Conference on, 2004, vol 1, pp. 786–791.
13. **Vandapel N., Donamukkala R. R. and Hebert M.** *Experimental Results in Using Aerial LADAR Data for Mobile Robot Navigation*, Field and Service Robotics, 2003, pp. 103–112.

14. **Tamm T. and Remm K.** *Estimating the parameters of forest inventory using machine learning and the reduction of remote sensing features*, International Journal of Applied Earth Observation and Geoinformation, 2009, vol. 11, no. 4, pp. 290–297.
15. **Sofman B. and Stentz A.** *Terrain Classification from Aerial Data to Support Ground Vehicle Navigation*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMURI-TR-05-39, 2006.
16. **Sofman B., Ratliff E. L., Cole J. and Vandapel N.** *Improving Robot Navigation Through Self-Supervised Online Learning*, Journal of Field Robotics 2006, vol. 23, pp. 1059–1075.
17. **Dima C. S., Vandapel N. and Hebert M.** *Classifier fusion for outdoor obstacle detection*, Robotics and Automation, Conference on, 2004, vol. 1, pp. 665–671.
18. **Tamm T.** *Riigimetsa takseerandmete kasutamine eesti metsade kaugseires tehisõppe rakenduse abil*, Magistritöö, Tartu Ülikool, 2005.
19. **Hebert M. and Vandapel N.** *Terrain classification techniques from ladar data for autonomous navigation*, Robotics Institute, 2003, p. 411.
20. **Dima C. S., Vandapel N. and Hebert M.** *Sensor and classifier fusion for outdoor obstacle detection: an application of data fusion to autonomous off-road navigation*, Applied Imagery Pattern Recognition Workshop, Proceedings on, 2003, pp. 255–262.
21. **Mayer H.** *Automatic object extraction from aerial imagery a survey focusing on buildings*, Computer vision and image understanding, 1999, vol. 74, pp. 138–149.
22. **Mena J.** *State of the art on automatic road extraction for GIS update: a novel classification*, Pattern Recognition Letters, 2003, vol. 24, pp. 3037–3058.
23. **Heidarsson H. K. and Sukhatme G. S.** *Obstacle detection from overhead imagery using self-supervised learning for Autonomous Surface Vehicles*, Intelligent Robots and Systems, International Conference on, 2011, pp. 3160–3165.
24. **Hadsell R., Sermanet P., Ben J., Erkan A., Scoffier M., Kavukcuoglu K., Muller U. and LeCun Y.** *Learning long-range vision for autonomous off-road driving*, Journal of. Field Robotics, 2009, vol. 26, pp. 120–144.
25. **Vandapel N., Donamukkala R. and Hebert M.** *Experimental results in using aerial ladar data for mobile robot navigation*, Field and Service Robotics, 2006, pp. 103–112.
26. **Hadsell R., Erkan A., Sermanet P., Ben J., Kavukcuoglu K., Muller U. and LeCun Y.** *A multi-range vision strategy for autonomous offroad navigation*, Robotics and Applications, Proceeding on, 2007, vol. 1, p. 7.
27. **Kelly A. and Stentz A.**, *Rough terrain autonomous mobility part 2: An active vision, predictive control approach*, Autonomous Robots, 1998,

vol. 5, pp. 163–198.

28. **Kelly A., Stentz A., Amidi O., Bode M., Bradley D., Diaz-Calderon A., Happold M., Herman H., Mandelbaum R. and Pilarski T.** *Toward reliable off road autonomous vehicles operating in challenging environments*, The International Journal of Robotics Research, 2006, vol. 25, pp. 449–483.
29. **Hart P., Nilsson N. and Raphael B.** *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*, IEEE Transactions on Systems Science and Cybernetics, 1968, vol. 4, pp. 100–107.
30. **Dijkstra E. W.** *A note on two problems in connexion with graphs*, Numerische mathematik, 1959, vol. 1, pp. 269–271.
31. **Stentz A.** *Optimal and Efficient Path Planning for Unknown and Dynamic Environments*, International Journal of Robotics and Automation, 1993, vol. 10, pp. 89–100.
32. **Koenig S. and Likhachev M.** *Fast replanning for navigation in unknown terrain*, Robotics, IEEE Transactions on, 2005, vol. 21, pp. 354–363.
33. **Gerkey B. P. and Konolige K.** *Planning and control in unstructured terrain*, Workshop on Path Planning on Costmaps, 2008.
34. **Konolige K.** *A gradient method for realtime robot control*, Intelligent Robots and Systems, International Conference on, 2000.
35. **Hebert M., Deans M., Huber D., Nabbe B. and Vandapel N.** *Progress in 3-D Mapping and Localization*, 9th International Symposium on Intelligent Robotic Systems, 2001.
36. **Vandapel N. and Chatila R.** *Affine trackability for landmark selection in natural environment*, Intelligent Robots and Systems, International Conference on, 2002, vol. 1, pp. 37–42.
37. **Elfes A.** *Occupancy grids: A stochastic spatial representation for active robot perception*, The Sixth Conference on Uncertainty in AI, 1990, vol. 2929.
38. **Oriolo G., Ulivi G. and Vendittelli M.** *Fuzzy maps: a new tool for mobile robot perception and planning*, Journal of Robotic Systems, 1997, vol. 14, pp. 179–197.
39. **Lu D. and Weng Q.** *A survey of image classification methods and techniques for improving classification performance*, International Journal of Remote Sensing, 2007, vol. 28, pp. 823-870.
40. **Bishop C. M.** *Pattern Recognition and Machine Learning*, New York, Springer-Verlag Inc., 2006.
41. **Simard P. Y., Steinkraus D. and Platt J. C.** *Best practices for convolutional neural networks applied to visual document analysis*, Seventh International Conference on Document Analysis and Recognition, 2003, vol. 2, pp. 958–962.

42. **Bradley D., Thayer S., Stentz A. and Rander P.** *Vegetation detection for mobile robot navigation*, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-04-12, 2004.
43. **Bradley D. M., Unnikrishnan R. and Bagnell J.** *Vegetation detection for driving in complex environments*, In Robotics and Automation, IEEE International Conference on, 2007, pp. 503–508.
44. **Lecun Y., Bottou L., Orr G. B. and Muller K. R.** *Efficient BackProp*, Lecture Notes in Computer Science, 1998, vol. 1524, pp. 5–50.
45. **Lecun Y., Bottou L., Bengio Y. and Haffner P.** *Gradient-based learning applied to document recognition*, Proceedings of the IEEE 86, 1998, vol. 11, pp. 2278–2324.
46. **Hadsell R., Erkan A., Sermanet P., Scoffier M., Muller U. and Lecun Y.** *Deep belief net learning in a long-range vision system for autonomous off-road driving*, Intelligent Robots and Systems, International Conference on, 2008, pp. 628–633.
47. **Lester P.** *A* Pathfinding for Beginners*, [Online] http://www.policyalmanac.org/games/aStarTutorial.htm (9. Aug 2012).
48. **Stentz A.** *The Focussed D* Algorithm for Real-Time Replanning*, International Joint Conference on Artificial Intelligence, 1995, vol. 14, pp. 1652–1659.
49. **Hadsell R., Sermanet P., Ben J., Erkan A., Han J., Muller U. and LeCun Y.** *Online Learning for Offroad Robots: Spatial Label Propagation to Learn Long-Range Traversability*, Robotics: Science and Systems, 2007, vol. 11, p. 32.
50. **Glaskowsky P. N.** *NVIDIA's Fermi: the first complete GPU computing architecture.*, whitepaper, 2009.
51. **Sermanet P., Hadsell R., Scoffier M., Grimes M., Ben J., Erkan A., Crudele C., Miller U. and LeCun Y.** *A multirange architecture for collision-free off-road robot navigation*, Journal of Field Robotics, 2009, vol. 26, pp. 52–87.
52. **LeCun Y., Muller U., Ben J., Cosatto E. and Flepp B.** *Off-road obstacle avoidance through end-to-end learning*, Advances in neural information processing systems, 2006, vol. 18, p. 739.

List of Publications

1. **Hudjakov, R.; Tamre, M**. *Comparison of Aerial Imagery and Satellite Imagery for Autonomous Vehicle Path Planning.* In: Proc.of the 8th International Conf. of DAAAM Baltic, Industrial Engineering, Tallinn, Estonia, 19–21 April, 2012. Ed. by Otto. T. Tallinn, Estonia: Tallinn University of Technology Press, 2012, 301–308.

2. **Hudjakov, R.; Tamre, M.** *Ortophoto analysis for UGV long-range autonomous navigation.* Estonian Journal of Engineering, 2011, 17(1), 17–27.

3. **Hudjakov, R; Tamre, M.** *Aerial Imagery Based Long-Range Path Planning for Unmanned Ground Vehicle.* In: 7th International Conf. Mechatronics Systems and Materials MSM 2011, Kaunas, Lithua11nia, 7–9 July, 2011. Ed. by Skiedraite, I., Baskutiene, J., Dragašius, E. Lithuania: Kaunas University of Technology Press, 2011, 1–7.

4. **Hudjakov, R.; Tamre, M**. *Aerial Imagery Terrain Classification for Long-Range Autonomous Navigation.* In: Proc. of the 7th International Conference of DAAAM Baltic Industrial Engineering, 22-24 April 2010. Ed. by Küttner, R.. Tallinn: Tallinn Technical University Press, 2010, 530–535.

5. **Hudjakov, R; Tamre, M.** *Aerial Imagery Terrain Classification for Long-Range Autonomous Navigation.* In: Proc. of International Symposium on Optomechatronic Technologies: ISOT2009. Ed. by Okyay Kaynak. IEEE, 2009, 88–91.

Abstract

The navigation capabilities of existing off-road unmanned ground vehicles are severely limited by the perception capabilities of the vehicles' on-board sensors. The reliable perception distance of the on-board sensors does not surpass 40m limiting the driving capabilities of an UGV to those of a human driver in dense fog. The objective of this thesis is to generate ad hoc navigation maps for an UGV, enabling smarter path planning, faster movement and reduced energy consumption.

The thesis proposes an intelligent long-range navigation method that learns from the UGV local navigation system and extrapolates the knowledge about local environment into wider area using overhead imagery. The proposed navigation method has built-in failsafe allowing UGV to fall back to local navigation system when the long-range system is not adequate. The method has energy efficient implementation that utilizes heterogeneous hardware, enabling deployment on battery powered vehicles.

The thesis consists of five parts: introduction, short review of literature, theoretical foundations, practical experiments and conclusion. The introduction presents the objectives of the thesis and describes its structure. The following review of the literature chapter discusses problems in existing off-road capable navigation systems and proposed solutions. In addition the review of the literature chapter offers a brief summary of works in areas related to this thesis – aerial imagery classification, path planning and combining of maps from multiple sources.

The review of the literature is followed by a theoretical section, that focuses on orthophoto analysis, cost map generation, path planning and brings forth a possible usage scenario for the results of this work. For orthophoto analysis we proposed convolutional neural networks based classifier, that combines trainable feature extractors and a linear classifier. The useful properties of this classifier are described in detail in the context of off-road navigation task.

The classifier output is further analyzed in Bayesian framework to find confidence information attached to extracted feature vectors. The classifier outputs a feature for each input pattern, each element of this vector represents the likelihood that corresponding feature is present on input pattern. In order to account this likelihood into cost map generation we need to convert it to probability. The conversion is done by extracting probability density functions for each feature class from classifier training set and using them in Bayes' theorem.

For cost map generation we first define a cost of unknown terrain, it is defined to be lower than the cost of obstacles but higher than the cost of clearances. This constant is used if classifier uncertainty is zero, for nonzero values the cost will gravitate towards it as confidence decreases. In addition we attach a weight to each feature class; the traveling cost of an area is weighted sum of feature probabilities that is offset by the cost of unknown

terrain. A special care is taken to ensure the traveling cost of an area remains positive at all times, otherwise it would break path planning algorithms.

The produced cost map is used for path planning. The path planner itself is not the subject of this thesis, but significant performance advantages can be gained from combing path planner, cost map generator and terrain classifier. The terrain classification step is a relatively expensive one performance wise, reversing the evaluation pipeline and triggering the classifier on need-to-know basis both increases the reaction time of the navigation system and reduces the energy consumption.

The theoretical section is followed by a practical one that focuses on evaluating the capabilities of proposed navigation system. The main objective of the theoretical section is to demonstrate the navigation systems path planning ability on a known terrain and its self-assessment ability on an unknown terrain. The experiments made in the course of this chapter use manually labeled aerial imagery from Estonian Land Board database and satellite imagery form Google Maps database.

The classifier capability is evaluated by comparing the classifier output against manually labeled aerial and satellite imagery and against labels from a GIS database. The classifier has shown an excellent classification ability achieving over 90% correct classification rate on all the tests. For validating the Bayesian filter a set of weakened classifiers were prepared that were trained with insufficient and noisy data. The experiments show that the processed classification result of those weakened classifiers has low confidence attached to features. The confidence information is used by cost map generation algorithm, which gravitates the cost of areas with low confidence vectors towards the cost of unknown terrain.

The classifier capability analysis is followed by a set of combined tests that cover aerial imagery classification, cost map generation and path planning. The objectives of combined tests are twofold: to demonstrate the path planning capability of the navigation system as a whole and to show the behavior of the system with weakened classifier. The combined tests show that the navigation system is able to work under bad conditions when the surroundings of UGV are unfamiliar to the classifier.

The practical section is concluded by a subsection that addresses the main weakness of the classifier – the convolutional neural networks are large systems that require equally large computing capacities to operate. Execution of the proposed classifier on a CPU is unreasonable, because it requires the full power of a multicore processor for real time operation, which is a burden for a battery powered vehicle. The thesis proposes a solution executing on heterogeneous computing architecture, which reduces the energy consumption by more than 250 times and reduces analyzing time by more than 850 times.

The body of the doctoral thesis is summarized by a conclusion chapter, which outlines the main achievements and future-oriented ideas.

Kokkuvõte

Olemasolevate autonoomsete sõidukite maastikuvõimekusele on oluliseks piiranguks pardal olevate andurite nägemisulatus. Liikuva sõiduki rappumisest tulevad häired piiravad usaldusväärse nägemiskauguse umbes neljakümnele meetrile, mis omakorda langetab autonoomse sõiduki maksimaalse võimekuse paksus udus liikuva autojuhi tasemele. Töö eesmärk on genereerida laiemat maa-ala hõlmavad *ad hoc* kaardid võimaldamaks targemat teekonna planeerimist, kiiremat sõitmist ja väiksemat energiakulu.

Käesoleva doktoritöö raames on välja töötatud intelligentne kaugmaa navigatsioonisüsteem, mis õpib sõiduki lokaalselt navigatsioonisüsteemilt ja ekstrapoleerib kogutud teadmise laiemale maa-alale, kasutades lisaks satelliit- või aerofotosid. Pakutud navigatsioonisüsteem on suuteline hindama oma võimekust vähendamaks süsteemi eksimisest tulenevaid riske – süsteemi võimekuse langedes võib sõiduk ümber lülituda lokaalsele navigatsioonisüsteemile. Töös on esitatud ka heterogeensetel arvutisüsteemidel põhinev lahendus, mis sobib patareitoitel töötavale sõidukile.

Töö koosneb viiest osast: sissejuhatusest, põgusast kirjanduse ülevaatest, teoreetilisest osast, praktilisest osast ja kokkuvõttest. Sissejuhatuses püstitatakse doktoritöö eesmärgid ning kirjeldatakse ülesehitust. Sissejuhatusele järgnev kirjanduse ülevaate peatükk käsitleb probleeme olemasolevates maastikuvõimekusega UGV navigatsioonisüsteemides ja seni pakutud lahendustes. Lisaks teeb kirjanduse ülevaade põgusa kokkuvõtte käesoleva tööga otseselt seotud valdkondadest: aerofotode klassifitseerimisest, teekonna planeerimisest ning mitmest allikast pärinevate kaartide ühildamisest.

Kirjanduse ülevaatele järgneb töö teoreetiline osa, mis käsitleb ortofoto analüüsi, maastiku läbitavuse hindamist, teekonna planeerimist ning toob esile ühe võimaliku töö tulemuste kasutamise stsenaariumi. Ortofotode analüüsiks on esitatud konvulutsioonilistel närvivõrkudel põhinev klassifikaator, mis ühendab tunnuste valimise ja näidistel põhineva järeldamise süsteemi. Teoreetilise osa esimesed alapeatükid on pühendatud valitud klassifikaatori detailse ülesehituse kirjeldamiseks, tuues eraldi välja omadused, mis teevad selle antud ülesande lahendamiseks sobivaks.

Järgnevad teoreetilise osa alapeatükid keskenduvad maastiku läbitavuse hindamisele ja teekonna planeerimisele. Ortofoto analüüs käib väike tükk korraga, klassifikaatori väljundiks on tükile vastav tunnusvektor. Iga selle vektori element on funktsioon vastava tunnuse esinemise tõenäosusest klassifikaatori sisendis. Leidmaks funktsiooni, mis teisendab klassifikaatori väljundvektori tõenäosusvektoriks, on kasutatud Bayesi teoreemi. Klassifikaatori väljundi teisendamine tõenäosuseks on oluline klassifikaatori pädevuse hindamiseks, mis võimaldab robotil vältida tundmatut maastikku või läheneda võõrale piirkonnale ettevaatlikult, kasutades aeglast kiirust ja lokaalset navigatsioonisüsteemi.

Igale tunnusele on seatud vastavusse maastiku läbitavuse hinnang. Hindamaks ortofoto tükil kujutatud maastiku läbitavust kombineeritakse

saadud tõenäosusvektor tunnustele omistatud hinnangutega. Teekonna planeerimisel jagatakse ortofoto väikesteks tükkideks ning leitakse igale tükile vastava maastiku läbitavuse hinnang. Teekonna planeerimise algoritm leiab seejärel vähima summaarse läbitavuse hinnanguga teekonna antud alguspunktist lõpp-punkti.

Iga ortofoto tüki tunnusvektori leidmine on arvutusmahukuse ja energiakulu poolest kallis operatsioon. Suurte aerofotode klassifitseerimiseks kuluv energia võib olla akutoitega autonoomsele robotile oluliseks koormaks. Integreerides teekonna planeerija ortofoto klassifikaatoriga, saab juhtida analüüsi protsessi klassifitseerimaks ainult neid ortofoto tükke, mida reaalselt kasutatakse. Teekonna planeerimise algoritmi käsitleva alapeatüki lõpus kirjeldatakse üht võimalikku meetodit planeerija ja klassifikaatori ühendamiseks.

Töö teoreetilisele osale järgneb praktiline osa, mis keskendub pakutud navigatsioonisüsteemi võimekuse uurimisele. Tehtud eksperimentide peamine eesmärk on demonstreerida navigatsioonisüsteemi suutlikkust planeerida teekonda nii robotile tuttaval maastikul kui ka võõras keskkonnas. Eksperimentides on kasutatud käsitsi sildistatud Eesti Maa-ameti aerofotosid ja Google'i kaardirakendusest pärit satelliitfotosid.

Klassifikaatori võimekuse uurimiseks on võrreldud klassifikaatori väljundit inimese poolt sildistatud aero- ja satelliitfotodega nii maastikul kui ka asulapiirkonnas. Lisaks on saadud tulemused kõrvutatud ka Maa-ameti geograafilise info andmebaasiga. Kõikides tehtud katsetes on klassifikaator näidanud suurepärast sildistamisvõimekust, saavutades üle 90% klassifitseerimise määra.

Klassifikaatori pädevuse hinnangu algoritmi kontrollimiseks on ette valmistatud meelega nõrgestatud klassifikaatorid. Nende klassifikaatorite treenimiseks on kasutatud ebapiisavalt väikest ja mürarikast andmebaasi. Katsed näitavad, et nõrgestatud klassifikaatorite väljundiks on tunnused, millele on omistatud madal leidmise tõenäosus. Seda tõenäosust kasutab maastiku läbitavuse hindamise algoritm, mis lähendab ebapädeva klassifikaatori poolt antud maastiku läbitavuse hinnangu tundmatu maastiku omale.

Klassifikaatori võimekuse analüüsile järgnevad kombineeritud testid, mis sisaldavad aerofoto klassifitseerimist, maastiku läbitavuse hindamist ja teekonna planeerimist. Komplekssete eksperimentide peamine eesmärk on demonstreerida navigatsioonisüsteemi kui terviku töövõimet ja näidata ebapädeva klassifikaatoriga süsteemi käitumist. Kombineeritud testid näitavad, et pakutud navigatsioonisüsteem on võimeline töötama ka kehvades oludes, kui ümbruskond on klassifikaatorile võõras.

Praktilise osa lõpetab konvulutsiooniliste närvivõrkude peamist nõrkust adresseeriv alapeatükk. Konvulutsioonilised närvivõrgud on väga suured ning spetsiifilise ehitusega aparaadid, mille kasutamiseks on tarvis suuri arvutusvõimsusi. Pakutud klassifikaatori täitmine traditsioonilise keskprotsessori peal on ebamõistlik, kuna reaalajas navigeerimiseks läheb tarvis suure voolutarbega mitmetuumalist protsessorit, mis koormab liikuri akut. Käesoleva doktoritöö raames on välja töötatud pakutud heterogeensetel arvutisüsteemidel

põhinev lahendus, mille arvutusvõimsus ületab 850 korda neljatuumalise keskprotsessori oma, vähendades samal ajal energiatarvet 250 korda.

Doktoritöö sisulise osa lõpetab kokkuvõttev peatükk, milles on välja toodud peamised saavutused ning tulevikku suunatud ideed.

Curriculum Vitae

1. Personal data

Name	Robert Hudjakov
Date and place of birth	14 Nov 1979, Tallinn

2. Contact information

Address	Mäepealse 16-20, 12619 Tallinn, Estonia
Phone	+372 52 45 445
E-mail	robert.hudjakov@gmail.com

3. Education

Institution	Graduation Year	Field of study/degree
Tallinn Polytechnic School	2001	Computers and Computer Networks
Tallinn University of Technology	2005	B.Sc. in Engineering Physics
Tallinn University of Technology	2007	M.Sc. *Cum Laude* in Engineering Physics

4. Language competence/skills

Estonian	Fluent
English	Fluent
Russian	Intermediate

5. Professional Employments

JOT Estonia Ltd.	2000–2008	Software Engineer
IPTE Estonia Ltd.	2008–...	Software Engineer

Elulookirjeldus

1. Isikuandmed

Nimi	Robert Hudjakov
Sünniaeg ja -koht	14. nov. 1979, Tallinn

2. Kontaktandmed

Aadress	Mäepealse 16-20, 12619 Tallinn, Estonia
Telefon	+372 52 45 445
E-mail	robert.hudjakov@gmail.com

3. Haridustee

Õppeasutus	Lõpetamise aeg	Haridus
Tallinna Polütehnikum	2001	Arvutid ja arvutivõrgud
Tallinna Tehnikaülikool	2005	B.Sc. tehnilises füüsikas
Tallinna Tehnikaülikool	2007	M.Sc. *cum laude* tehnilises füüsikas

4. Keelteoskus

Eesti	Emakeel
Inglise	Ladus
Vene	Kesktase

5. Teenistuskäik

JOT Estonia Ltd.	2000–2008	Tarkvarainsener
IPTE Estonia Ltd.	2008–...	Tarkvarainsener

DISSERTATIONS DEFENDED AT
TALLINN UNIVERSITY OF TECHNOLOGY ON
MECHANICAL AND INSTRUMENTAL ENGINEERING

1. **Jakob Kübarsepp**. Steel-Bonded Hardmetals. 1992.

2. **Jakub Kõo**. Determination of Residual Stresses in Coatings &Coated Parts. 1994.

3. **Mart Tamre**. Tribocharacteristics of Journal Bearings Unlocated Axis. 1995.

4. **Paul Kallas**. Abrasive Erosion of Powder Materials. 1996.

5. **Jüri Pirso**. Titanium and Chromium Carbide Based Cermets. 1996.

6. **Heinrich Reshetnyak**. Hard Metals Serviceability in Sheet Metal Forming Operations. 1996.

7. **Arvi Kruusing**. Magnetic Microdevices and Their Fabrication methods. 1997.

8. **Roberto Carmona Davila**. Some Contributions to the Quality Control in Motor Car Industry. 1999.

9. **Harri Annuka**. Characterization and Application of TiC-Based Iron Alloys Bonded Cermets. 1999.

10. **Irina Hussainova**. Investigation of Particle-Wall Collision and Erosion Prediction. 1999.

11. **Edi Kulderknup**. Reliability and Uncertainty of Quality Measurement. 2000.

12. **Vitali Podgurski**. Laser Ablation and Thermal Evaporation of Thin Films and Structures. 2001.

13. **Igor Penkov**. Strength Investigation of Threaded Joints Under Static and Dynamic Loading. 2001.

14. **Martin Eerme**. Structural Modelling of Engineering Products and Realisation of Computer-Based Environment for Product Development. 2001.

15. **Toivo Tähemaa**. Assurance of Synergy and Competitive Dependability at Non-Safety-Critical Mechatronics Systems design. 2002.

16. **Jüri Resev**. Virtual Differential as Torque Distribution Control Unit in Automotive Propulsion Systems. 2002.

17. **Toomas Pihl**. Powder Coatings for Abrasive Wear. 2002.

18. **Sergei Letunovitš**. Tribology of Fine-Grained Cermets. 2003.

19. **Tatyana Karaulova**. Development of the Modelling Tool for the Analysis of the Production Process and its Entities for the SME. 2004.

20. **Grigori Nekrassov**. Development of an Intelligent Integrated Environment for Computer. 2004.

21. **Sergei Zimakov**. Novel Wear Resistant WC-Based Thermal Sprayed Coatings. 2004.

22. **Irina Preis**. Fatigue Performance and Mechanical Reliability of Cemented Carbides. 2004.

23. **Medhat Hussainov**. Effect of Solid Particles on Turbulence of Gas in Two-Phase Flows. 2005.

24. **Frid Kaljas**. Synergy-Based Approach to Design of the Interdisciplinary Systems. 2005.

25. **Dmitri Neshumayev**. Experimental and Numerical Investigation of Combined Heat Transfer Enhancement Technique in Gas-Heated Channels. 2005.

26. **Renno Veinthal**. Characterization and Modelling of Erosion Wear of Powder Composite Materials and Coatings. 2005.

27. **Sergei Tisler**. Deposition of Solid Particles from Aerosol Flow in Laminar Flat-Plate Boundary Layer. 2006.

28. **Tauno Otto**. Models for Monitoring of Technological Processes and Production Systems. 2006.

29. **Maksim Antonov**. Assessment of Cermets Performance in Aggressive Media. 2006.

30. **Tatjana Barashkova**. Research of the Effect of Correlation at the Measurement of Alternating Voltage. 2006.

31. **Jaan Kers**. Recycling of Composite Plastics. 2006.

32. **Raivo Sell**. Model Based Mechatronic Systems Modeling Methodology in Conceptual Design Stage. 2007.

33. **Hans Rämmal**. Experimental Methods for Sound Propagation Studies in Automotive Duct Systems. 2007.

34. **Meelis Pohlak**. Rapid Prototyping of Sheet Metal Components with Incremental Sheet Forming Technology. 2007.

35. **Priidu Peetsalu**. Microstructural Aspects of Thermal Sprayed WC-Co Coatings and Ni-Cr Coated Steels. 2007.

36. **Lauri Kollo**. Sinter/HIP Technology of TiC-Based Cermets. 2007.

37. **Andrei Dedov**. Assessment of Metal Condition and Remaining Life of In-service Power Plant Components Operating at High Temperature. 2007.

38. **Fjodor Sergejev**. Investigation of the Fatigue Mechanics Aspects of PM Hardmetals and Cermets. 2007.

39. **Eduard Ševtšenko**. Intelligent Decision Support System for the Network of Collaborative SME-s. 2007.

40. **Rünno Lumiste**. Networks and Innovation in Machinery and Electronics Industry and Enterprises (Estonian Case Studies). 2008.

41. **Kristo Karjust**. Integrated Product Development and Production Technology of Large Composite Plastic Products. 2008.

42. **Mart Saarna**. Fatigue Characteristics of PM Steels. 2008.

43. **Eduard Kimmari**. Exothermically Synthesized B_4C-Al Composites for Dry Sliding. 2008.

44. **Indrek Abiline**. Calibration Methods of Coating Thickness Gauges. 2008.

45. **Tiit Hindreus**. Synergy-Based Approach to Quality Assurance. 2009.

46. **Karl Raba**. Uncertainty Focused Product Improvement Models. 2009.

47. **Riho Tarbe**. Abrasive Impact Wear: Tester, Wear and Grindability Studies. 2009.

48. **Kristjan Juhani**. Reactive Sintered Chromium and Titanium Carbide-Based Cermets. 2009.

49. **Nadežda Dementjeva**. Energy Planning Model Analysis and Their Adaptability for Estonian Energy Sector. 2009.

50. **Igor Krupenski**. Numerical Simulation of Two-Phase Turbulent Flows in Ash Circulating Fluidized Bed. 2010.

51. **Aleksandr Hlebnikov**. The Analysis of Efficiency and Optimization of District Heating Networks in Estonia. 2010.

52. **Andres Petritšenko**. Vibration of Ladder Frames. 2010.

53. **Renee Joost**. Novel Methods for Hardmetal Production and Recycling. 2010.

54. **Andre Gregor**. Hard PVD Coatings for Tooling. 2010.

55. **Tõnu Roosaar**. Wear Performance of WC- and TiC-Based Ceramic-Metallic Composites. 2010.

56. **Alina Sivitski**. Sliding Wear of PVD Hard Coatings: Fatigue and Measurement Aspects. 2010.

57. **Sergei Kramanenko**. Fractal Approach for Multiple Project Management in Manufacturing Enterprises. 2010.

58. **Eduard Latõsov**. Model for the Analysis of Combined Heat and Power Production. 2011.

59. **Jürgen Riim**. Calibration Methods of Coating Thickness Standards. 2011.

60. **Andrei Surzhenkov**. Duplex Treatment of Steel Surface. 2011.

61. **Steffen Dahms**. Diffusion Welding of Different Materials. 2011.

62. **Birthe Matsi**. Research of Innovation Capasity Monitoring Methodology for Engineering Industry. 2011.

63. **Peeter Ross**. Data Sharing and Shared Workflow in Medical Imaging. 2011.

64. **Siim Link**. Reactivity of Woody and Herbaceous Biomass Chars. 2011.

65. **Kristjan Plamus**. The Impact of Oil Shale Calorific Value on CFB Boiler Thermal Efficiency and Environment. 2012.

66. **Aleksei Tšinjan**. Performance of Tool Materials in Blanking. 2012.

67. **Martinš Sarkans**. Synergy Deployment at Early Evaluation of Modularity of the Multi-Agent Production Systems. 2012.

68. **Sven Seiler**. Laboratory as a Service – A Holistic Framework for Remote and Virtual Labs. 2012.

69. **Tarmo Velsker**. Design Optimization of Steel and Glass Structures. 2012.

70. **Madis Tiik**. Access Rights and Organizational Management in Implementation of Estonian Electronic Health Record System. 2012.

71. **Marina Kostina**. Reliability Management of Manufacturing Processes in Machinery Enterprises. 2012.