Combining Learning from Past Missions and Prediction Designs for Multi-AUV Situation Understanding

Emmanouil Orfanoudakis, Nikolaos Kofinas, and Michail G. Lagoudakis
Telecommunication Systems Institute (TSI)
Technical University of Crete (TUC)
Chania, Crete 73100, Greece

Table of Contents

1 Objective

In the context of the *Noptilus* project, Situation Understanding (WP6) is defined as:

> Cognitive ability of inferring high-level descriptions and representations of the current state of the environment.

The objective of this work package is summarized as follows:

> Automatically detecting, recognizing, understanding and predicting static underwater features and patterns as well as highly dynamic phenomena and events taking place and influencing the underwater operations.

The project deliverable of this work package is being developed as a system that supports the following functions, aligned with the three tasks within WP6:

On-Board On-Line function [**Task 6.1**] The on-board on-line system functions as an *event recognizer* that aims to discover high-level discrete phenomena within vast amounts of raw sensor data streams. The goal of the system is to provide (i) abstraction from raw data to facilitate decision making at higher levels, (ii) robustness against sensor noise and environmental changes in different scenarios, and (iii) compression of sensor information as a result of abstraction. It should be noted that the description of recognized events has a much shorter representation and, therefore, can be utilized to communicate the abstract state of the AUV in an efficient way over the limited bandwidth of acoustic communication modems. Thus, this system can enhance coordination within the multi-AUV team.

Off-Board Off-Line function [**Task 6.2**] The off-line system serves as a training process for generating automatically the configuration of the on-line system (grammatical structure, probabilities, events, etc.). It is intended to serve as the *learning* platform, which will utilize past mission logs, possibly annotated by humans, to automatically derive patterns associated with events of a certain type, such as normal and abnormal behavior. The goal is to off-line generate a robust event recognition scheme, capable of effective on-line recognition.

Integrated Event Recognition function[**Task 6.3**] The integration of the off-line system with the on-line system requires the adoption of a common representation and certain conventions for grammars, symbols, events, data quantization, parsing window modes, etc. In addition, the integration of event recognition within the Dune software architecture poses a few constraints on flexibility, but on the other hand allows for simultaneous execution of several event recognizers on diverse types of events. The ultimate goal is to enable the operators of the AUV team to generate event recognizers from recorded logs (off-line) and seamlessly transfer and activate them onboard for event recognition (on-line). Provided communication with the mission operation center, any transmitted information on recognized events can be presented to operators at land/boat stations in ways that enhance awareness about the mission and simplifies the supervision of individual vehicles or real-time monitoring of the whole mission itself at a high level of description.

The present document, Deliverable 6.3, titled "*Combining Learning from Past Missions and Prediction Designs for Multi-AUV Situation Understanding*", according to the Noptilus DoW, *provides the details of the evaluation of the designs of Tasks 6.1 and 6.2, as well as the final NOPTILUS situation understanding module (that combines the two designs), along with experimental results.* More specifically, this document describes the research results obtained within Task 6.3, where focus is placed on the integration of the parser used for on-board on-line event recognition (documented in Deliverable 6.1) with the grammars learned from past mission logs (documented in Deliverable 6.2).

2 Definitions and Notation

Observation a measurement obtained by a sensor

Data/Observation Stream a temporally-ordered list of observations

Symbol a discrete representation of an observation (or several observations)

(Symbol) Sequence/Word an ordered list of discrete symbols

Grammar a collection of syntactic rules for producing sequences of symbols

CFG Context-Free Grammar (syntactic rules apply independently of the context)

PCFG Probabilistic Context-Free Grammar (each rule carries a probability value)

Parser software that builds the syntactic tree of a sequence produced by a grammar

Syntactic Tree tree showing how a sequence of symbols is produced by a grammar

Input Sequence sequence/word of symbols fed into a parser

Output Symbol the output of the parser on a given input sequence

Grammar Hierarchy output sequences serve as input sequences at higher levels

- $a, b, c, \ldots$ represent symbols
- Σ represents an alphabet, that is, a set of symbols
- a^* means that the symbol a can appear zero or more times
- a^+ means that the symbol a can appear one or more times
- a^n means that the symbol a appears exactly n times

- w is a word, that is, any sequence of symbols
- $|w|$ denotes the length of a word w
- w^R is the reverse word of a word w
- O is a corpus, that is a finite set of words
- F^* represents the set of all words with zero or more symbols from F
- Σ^* represents all the words over some alphabet Σ
- $\mathcal{L}$ represents a language, that is, any subset of Σ^*
- G is a grammar over some alphabet
- $|G|$ is the length of grammar, the number of symbols required to be represented
- Lower-case letters in a grammar represent terminal symbols
- Upper-case letters in a grammar represent non-terminal symbols

3 Task 6.3 Motivation

Our approach to Event Recognition relies on the use of Probabilistic Context-Free Grammars (PCFGs) [1] operating on streams of discrete symbols generated from sensor measurements through a quantization procedure. These grammars are learned using data from past missions, namely collections of patterns or corpora of words (sequences of symbols) corresponding to normal AUV operation. The learned grammars are subsequently used to detect patterns that cannot be interpreted by the grammar and, thus, are reported as abnormal events. Alternatively, grammars may be handcrafted, since they are human-understandable and intuitive structures, and fed into the parser. In this case, the outcome may any desired discrete symbol (not just normal/abnormal) corresponding to the most probable derivation at the top of the syntactic tree.

Work within the project on on-line parsing and off-line learning was conducted primarily by two researchers, Emmanouil Orfanoudakis and Nikolaos Kofinas respectively, who worked independently to some degree due to the different nature of the corresponding problems. The integration of their work into a coherent product required the adoption of a common representation and certain conventions for grammars, symbols, events, data quantization, parsing window modes, etc. Additionally, several implementation issues had to be finalized to bring these two pieces of software together and perform experiments. Task 6.3 was introduced into the project exactly to assist this integration.

4 Task 6.3 Contribution

The implementation of the parser within Dune was finalized and additionally the integration with the off-line grammar learning procedure was performed. The Event Recognition task (Dune term for independent software modules) is now able to run within Dune either in simulation or on the real AUVs in multiple instances, each one performing parsing on some data stream communicating their outcome using the IMC protocol. The software repository for the on-line parser, as well as usage instructions, are also documented in this text.

The implementation of the off-line procedure for learning probabilistic context-free grammars appropriate for normal/abnormal event recognition was also finalized. The proposed approach is based on a local search procedure using a Bayesian approach for inferring the complete structure of a grammar (non-terminal symbols, rules, probabilities) according to the principles of structured prediction. The software repository for the off-line learner, as well as usage instructions, are also documented in this text.

Finally, the outcomes of Task 6.1 and Task 6.2 were fully integrated so that grammars obtained through the off-line learning procedure in Task 6.2 can be fully utilized within the on-line parsing procedure in Task 6.1. The AUV team operator is now able to learn such grammars over any specific data stream (sensor measurements) and subsequently use these grammars on-board the AUV for on-line, real-time event recognition. A common representation for symbols, quantization, and parsing modes was adopted towards this end. Experiments were also conducted on both the learning and the parsing sides using data from real AUV missions. Some of these missions were executed by FEUP specifically for the purposes of event recognition.

5 Representation

This section introduces the representation we adopted for grammars, quantization rules, and corpora of words. Our choices aim to maximize flexibility, portability, and simplicity.

5.1 Grammars

Each grammar is fully represented using a text-based configuration file. In order to demonstrate the way we generate such a configuration file, we will use the following toy grammar for language $L = \{a^n b^n : n \geq 1\}$ over the alphabet $\Sigma = \{a, b\}$:

$$\begin{aligned} G &= (V, \Sigma, R, S, P) \\ V &= \{S, A, B\} \\ \Sigma &= \{a, b\} \\ RP &= \{S \to ASB\ (0.2),\ S \to AB\ (0.8),\ A \to a\ (1.0),\ B \to b\ (1.0)\} \end{aligned}$$

First, all Terminal and Non-Terminal Symbols of the grammar are numbered with unique numeric (integer) ids, starting from 0. In our example, a, b, A, B, and S are

```
4              % The id of the start symbol S
2              % Number of Terminal Symbols
0 2            % 'a' -> 0  A -> 2
1 3            % 'b' -> 1  B -> 3
2 1            % The representation of A -> 'a' [one production]
1 0 1.0 1      % "One symbol" "'a'" "probability" "Has terminal? 1"
3 1            % The representation of B -> 'a' [one production]
1 1 1.0 1      % "One symbol" "'a'" "probability" "Has terminal? 1"
4 2            % The representation of S -> ASB  AB [two productions]
3 2 4 3 0.2 0  % "Three symbols" "A" "S" "B" "probability" "Has terminal? 0"
2 2 3 0.8 0    % "Two symbols" "A" "B" "probability" "Has terminal? 0"
```

Figure 1: Sample Grammar Configuration File

mapped to 0, 1, 2, 3, and 4 respectively. Below, are the rules we follow to create the grammar configuration file:

1. Write the id of the start symbol
2. Write the total number of all Terminal Symbols
3. For each Terminal Symbol do:
   (a) Write the id of the Terminal Symbol
   (b) Write the id of the Non-Terminal Symbol which yields this Terminal
4. For each Non-Terminal Symbol do:
   (a) Write the id of the Non-Terminal Symbol
   (b) Write the total number number of productions for this Non-Terminal
   (c) For each Production of the Non-Terminal Symbol do:
      i. Write the total number of symbols in the production
      ii. Write the ids of all symbols as they appear in the production
      iii. Write the probability of this production
      iv. Write `1` (or `0`) if the production contains (or not) a Terminal

Figure 1 shows the configuration file for the grammar presented above. Note that anything after the `%`'s are just comments added for clarity and need not be in the configuration file.

5.2 Quantization

Quantization is the procedure of constraining a continuous set of values to a relatively small discrete set [2]. In our approach, this small set is the alphabet of the grammar

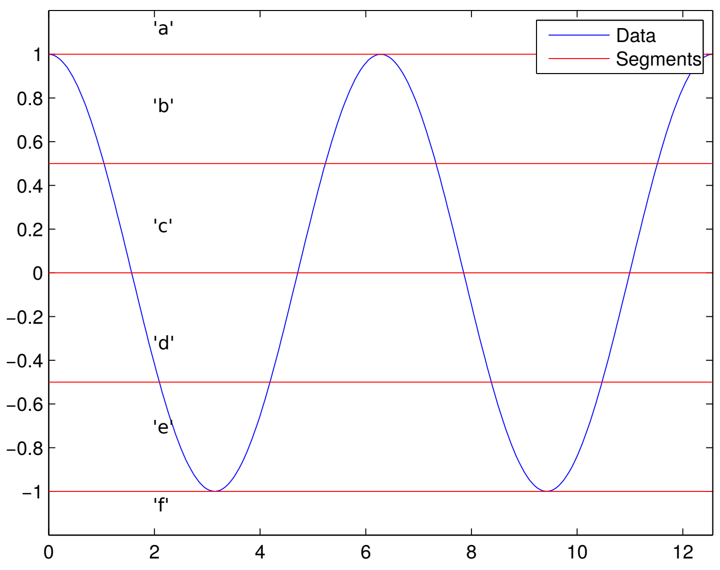


Figure 2: Data and quantization regions

or the set of terminal symbols. The quantizer is responsible for this discretization and maps the input values to the corresponding symbols from the alphabet according to the specification provided by a quantization configuration file. The importance of storing the quantization configuration must be stretched here; the quantization used when learning the grammar, must also be used when parsing a stream of data with the learned grammar for consistent and valid results.

Let us assume that we care to quantize the signal shown in Figure 2 using an alphabet of 4 symbols. As a convention, any chosen alphabet is enhanced by two extra symbols to indicate possible measurements above and below the typical range of the sensor. Such measurements may be delivered in practice due to sensor noise and/or unexpected factors and the representation should be ready to handle them. Therefore, there is a total of 6 discrete symbols in this example corresponding to specific regions as shown in Figure 2. Note that the region boundaries need not be uniformly distributed over the range.

Below are the rules we follow to create a configuration file for some quantization:

1. Write the number of all quantization threshold levels
2. For each quantization region, beginning from the first (lowest), do:
   (a) Write the lower threshold level of the region
   (b) Write the Terminal Symbol that corresponds to this region
3. Write the upper threshold level of the last (highest) region

Note that the lowest and the highest thresholds are typically $-\infty$ and $+\infty$ respectively. Figure 3 shows an example configuration file for the target quantization

```
7
-inf
f
-1.0
e
-0.5
d
-0.0
c
0.5
b
1
a
inf
```

Figure 3: Sample Quantization Configuration File

illustrated in Figure 2 (the signal takes nominal values in the window $[-1 : +1]$, but the entire $[-\infty : +\infty]$ window is covered). The reserved keyword `inf` is used for ∞.

5.3 Data File Format

Our off-line grammar training software takes as input a number of files which contain either plain parameter values or a corpus/set of words (sequences of terminal symbols). In the first case, the format of the required file is just a set of floating-point numbers (one number per line). In the latter case, the file contains multiple lines, where each one of them begins with either one (`1`) or zero (`0`), to indicate if the word belongs to the target language (positive example) or not (negative example), and then it contains all the symbols of the word separated by white space (spaces and/or tabs). An example file of a small corpus with 18 words belonging to the target language (positive examples) can be seen in Figure 4.

6 Off-Line Learning Software

6.1 Software Tools

We have implement the entire code for the unsupervised learning of PCFGs in C++. Apart from our proprietary code, we used an implementation of the Inside-Outside algorithm [3] provided by MIT[1], which is used to (re-)estimate the production probabilities in a PCFG using any corpus of words. Our software repository contains

[1] http://web.mit.edu/course/6/6.863/python/nltk_contrib_backup/mit/six863/rr4/inside-outside/

```
1 b a b a a b a a b a a b a b
1 a a a b b b b a b a b b b b a a b b b b a b a b b b b a a a
1 b b a b b b a b a b b b a b a a b a b b b a b a b b b a b b
1 b b a b b b a a b a a b b a a b a a b b b a b b
1 a b b a a b b a
1 b b b b a b b b a a b a b a a a a b a b a a b b b a b b b b
1 a b b a a b b a
1 a a a b a a a b b a a a b a b a a b b b b a a b a b a a a b b a a a b a a a
1 b b a a a a b b a a a a b b
1 b b b b a b b a a a b b a b b a b b b b b b a b b a b b a a a b b a b b b b
1 a a b b a a b b a b b a b b a a a a b b a b b a b b a a b b a a
1 b b b b b a b b a a b b a b b b b b
1 b b a a a a a a b b
1 a b b b a a b a b b b a a a b b a a a b b b a b a a b b b a
1 b b a b a a b b b b a a b a b b
1 a b b b b a a b b b a b b a a b b b b b b a a b b a b b b a a b b b b a
1 b b
1 a b a a b b b b a a b a
```

Figure 4: Sample Corpus of Words File

both codes and the user has to build both of them separately. Detailed instructions about the build procedure can be found in the `README.md` file.

6.2 Git Repository

The complete source code of our software tool for grammar learning is located at the following Git [4] code repository:

`https://github.com/eldr4d/CFG-learner`

The Git repository includes two separate branches: (a) the `master` branch, which contains the generic code that can be used in various grammar learning applications (no quantization), and (b) the `NOPTILUS` branch, which is specialized and customized for the Noptilus project (with quantization).

6.3 Usage

Our software tool for off-line grammar learning is invoked using the following terminal command line (for the `master` branch):

```
./CFG-learner -l learnFile -t testFile -e evalFile -d dirPath
-f outFile -p mode -w wordSize -s symbols -L normalization
```

where flags and arguments are as follows:

`-l learnFile` [REQUIRED] Input file containing a corpus of words belonging to the target language. Multiple files may be specified, each with its own `-l` flag; at least one such file is required.

`-t testFile` [REQUIRED] Input file containing a corpus of words for which it is known and specified that they belong to the target language; they are used for testing the generalization of the learned grammar. Multiple files may be specified, each with its own `-t` flag; at least one such file is required.

`-e evalFile` [REQUIRED] Input file containing a corpus of words for which it is known and specified whether they belong or not to the target language; they are used to evaluate the (over-)generalization of the learned grammar. Multiple files may be specified, each with its own `-e` flag; at least one such file is required.

`-d dirPath` [REQUIRED] The path to the directory which contains all data files.

`-f outFile` [REQUIRED] The prefix name for all output files.

`-p mode` [REQUIRED] Mode of word formation for parsing a stream of symbols: `0` = standard fixed nominal length, `1` = uniform distribution around nominal length, `2` = normal distribution around nominal length.

`-w wordSize` [REQUIRED] A positive integer indicating the nominal word length.

`-s symbols` [REQUIRED] The number of the terminal symbols in the grammar.

`-L normalization` [OPTIONAL] The normalization factor for $P(G)$. Default=0.001.

If the command is issued from the `NOPTILUS` branch, the input files contain raw measurements instead of words of symbols. These measurements must be quantized appropriately before proceeding. Quantization is specified implicitly; given the desired numbers of symbols through the `-s` flag, the internal quantizer splits the nominal range of measurement values (found by iterating over all data in the `learnFile` files) into `symbols` uniform regions and adds two extra symbols assigned to the regions above and below the nominal range. At the same time, using this information, a quantization configuration file is generated automatically to accompany the output grammar and support quantization in future uses of this grammar. Besides quantization, words in the input files are formed using the `mode` and `wordSize` parameters and subsequently learning proceeds similarly to the `master` branch.

7 On-Line Parsing Software

7.1 Integration in Dune

Integrating Tasks 6.1 and 6.2 into a functional event recognition module required the creation of an efficient parser within the Dune software architecture [5] of the

Noptilus AUVs. The parser takes as input either a handcrafted or an automatically learned grammar, as well as a specification for quantization and performs on-line event recognition (acquisition of measurements, quantization to symbols, grammatical parsing, reporting of events). Using the adopted representation formats described in Section 5, a software module capable of providing this functionality was coded consistently with the Dune requirements. This software module (*task*, in Dune terminology) can be executed transparently either off-board within a simulation environment or on-board a physical Noptilus AUV. This flexibility is due to the design and development principles of the Dune architecture.

7.1.1 Event Recognition Messaging

Tasks in Dune are executed asynchronously in parallel and provide their designated functionality, either by interacting with other tasks within Dune or by interacting with system resources (files, network sockets, sensors, actuators, etc). All tasks communicate with each other through the Inter-Module Communication (IMC) bus by exchanging small concrete pieces of information, termed *messages*, as per the message passing architecture paradigm. The IMC component handles routing and thread synchronization requirements transparently, typically with a small computational overhead.

Incoming information to a parser task for event recognition is provided by existing message types within IMC, corresponding to sensor measurements or processed outputs from other Dune tasks. Given the diversity of message formats with respect to the fields they contain, a small extractor function must be written manually within the parser task for each new type of incoming data stream considered for parsing. This need to be done only once and requires recompilation of the code. Any of the implemented extractors can be invoked later without need for recompilation. Currently, our code contains such extractors for a few input data streams: depth (message name `Depth`), pitch (message name `EulerAngles`) and roll (message name `EulerAngles`).

A new message type has been introduced within the IMC to store event recognition information. This message, named `ParserOutput`, contains two text fields:

`name` The source entity name, describing the origin of the result contained in the message; it is identical to the Entity Label of the parser task instance (see details in the next section) that produced the message.

`value` The value of the parser output; either a `true`/`false` value for normal/abnormal events or the id of the recognized event (as enumerated in the grammar configuration file).

Note that the output of some parser task may be used as input to another parser task, as explained below.

```
[Monitors.GrammarParser/PCFG]

#Simulation, Always, Hardware
Enabled             = Simulation

Entity Label        = PCFG

Execution Frequency = 5

#Which messages are used as inputs,
#must be in the correct order
Bind to =           ParserOutput.PCFG
#        Depth
#        EulerAngles.Pitch

Quantization = parser/SampleQuant.txt

Grammar = parser/SamplePCFG.txt

NormalAbnormal = true
Normal Threshold = 1e-5

Window Size = 8
Rolling Window = true
```

Figure 5: Sample Dune Task Configuration File

7.1.2 Event Recognition Dune Task

To support on-line event recognition, a Dune task that feeds input data to the parser and outputs the outcome of the parse operation in the form of messages was implemented. Each instance of this Dune parser task requires a set of configuration parameters to function. A sample configuration file for such a task is shown in Figure 5. Note that the symbol # is used to insert comments.

The configuration begins with a section name that designates the compile-time assigned name of the parser (`Monitors.GrammarParser`). The `Enabled` flag specifies whether this task instance will be activated only on simulated AUVs (`Simulation`), only on real AUVs (`Hardware`), or in both cases (`Always`). The `Entity Label` assigned to each instance serves as the instance id and is used to mark the output messages. Therefore, each parser output message is tagged with the name of the particular parser instance that produced it.

The parser is run periodically at the specified `Execution Frequency` (in Hertz) and, if the input stream has changed, parsing takes place, and a new output message is produced. The input stream is selected as particular message bindings, specified in the `Bind to` flag. In other words, a particular message type is used to produce the input symbols for the parser. Each message type must be dealt with independently and a compatible quantizer (specified by `Quantization`) must be assigned. When the input bindings (messages to be used as inputs) are determined to be produced by another parser instance, then no quantizer is needed, instead they are fed directly to the loaded grammar parser. Each input message generates a symbol asynchronously. These are fed to a concatenation module, which handles the synchronization issues. Each concatenated symbol is generated by the most recent outputs, thereby ensuring that all information sources are as closely synchronized as possible.

The configuration file of the supplied grammar used by the parser in each instance is specified using the `Grammar` label. When operating in normal/abnormal event recognition mode (details in Deliverable 6.2), as indicated by the `NormalAbnormal` flag, a probability threshold (`Normal Threshold`) is used to decide whether a parsed word will be classified as normal or abnormal, based on the derivation probability of the observed sequence according to the supplied grammar, and a `true`/`false` value is stored in the output message of the task instance. When operating in arbitrary event recognition mode (details in Deliverable 6.1), the id of the most probable derivation (event) is stored in the output message of the task instance. An arbitrary parsing window of size `Window Size`, which may be rolling or not (as indicated by `Rolling Window`) can be specified, but, in any case, these choices must be compatible with the supplied grammar.

7.2 Parallel/Hierarchical Event Recognition

Given the fact that we need to accommodate hierarchies of grammar parsers, the software provides the option to concatenate multiple inputs (each delivering one symbol) into a single hyper-symbol, provided that all individual symbols originate from `ParserOutput` messages. A parser instance can be configured to bind to `ParserOutput` messages, and an arbitrary number of bindings can be configured, all of which are concatenated into a single hyper-symbol, which is directly fed to the grammar without any form of quantization. This flexibility enables the construction of parallel grammar parsers, where some parser instances operate asynchronously on top of other parser instances. At the same time, these parser instances can be interconnected to form hierarchies and more complex structures.

An input message type configuration entry of the form: `ParserOutput`.<Entity Label> designates that another instance with an assigned Entity Label is to be used as input. When using `ParserOutput` messages as input to other parser instances, the corresponding quantization configuration is not used, instead the symbols are fed forward to the grammar without modification.

7.3 Git Repository

Our event recognition task was developed as a clone of the original Dune repository to avoid interference with ongoing development in the Dune software architecture. The modified Dune source code is available as a Git repository [4]:

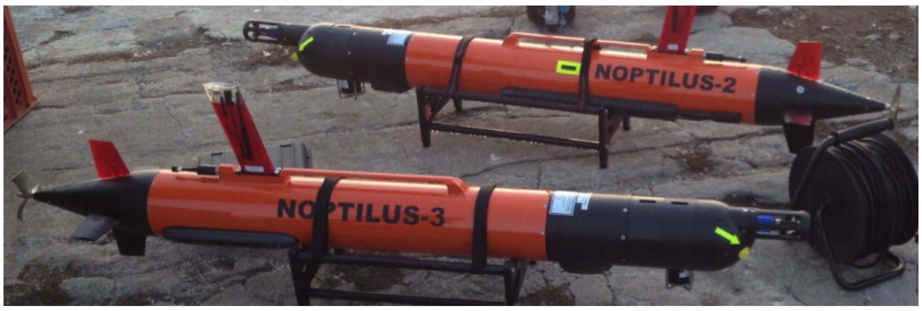


Figure 6: The Noptilus AUVs

https://github.com/vosk/dune

The Git repository includes the `master` branch, which contains the current development code, and the `staging` branch, a stable and tested version that can be readily used at any time.

7.4 Usage

To make use of the on-line event recognition Dune task, the user simply needs to follow these steps:

- prepare a configuration file for each task instance
- write code for new message extractors and recompile (only if necessary)
- place the grammar configuration file(s) in `etc/parser/`
- place the quantization configuration file(s) in `etc/parser/`
- place the task instance configuration file(s) in `dune/etc/`
- include the task instance file into the master configuration, e.g. `lauv-noptilus-1`
- start Dune

8 Experiments

The event recognition approach developed within WP6 was tested on data from the real AUVs, which were developed within the Noptilus project. Two of these Noptilus AUVs are shown in Figure 6. The missions were conducted in Porto de Leixões, Portugal by our FEUP partners. This section presents results obtained on both parts (off-line and on-line) of the proposed approach.

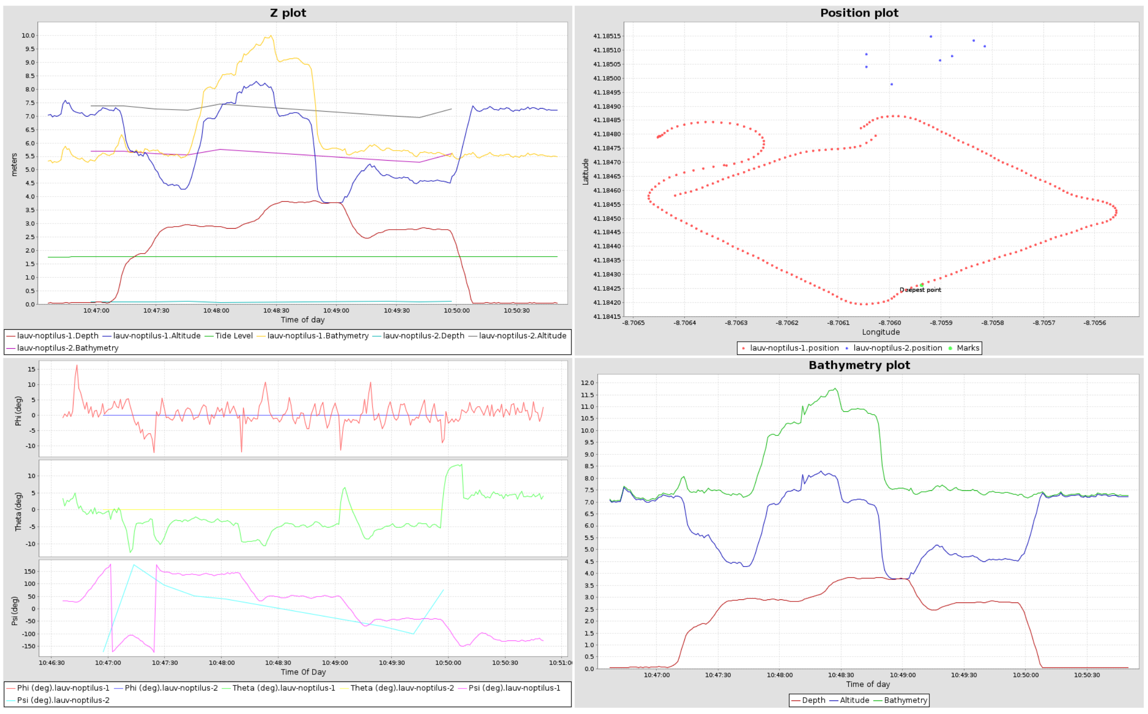


Figure 7: A mission log containing only normal events: z-plot (top-left), $x - y$-plot (top-right), Euler angles (bottom-left), depth (bottom-right).

8.1 Experimenting with Off-Line Learning

We used AUV data from previous mission logs in order to learn grammars for normal/abnormal event recognition and, thus, test and evaluate our approach. Figure 7 presents the relevant data from the bathymetry, altitude, and depth sensors, as well as the Euler angles measurements, during a normal mission. We have used four such missions, which were produced and were made available to us by our FEUP partners during the recent integration week[2]. We will rely on these data in order to train a PCFG, which should be able to recognize events that do not "match" the ones that occurred during these normal missions.

In order to evaluate our grammar, we will use data from another set of four missions, during which abnormal events were manually introduced. These abnormal events were manually-introduced disturbances in the diving behavior of the AUV, namely the AUV performed an unexpected emerge followed by a sudden submerge. Data from one of these missions are presented in Figure 8; the abnormal event occurred roughly between 10:59:40 and 11:01:00. It is clear that the introduced disturbance/abnormality can hardly be distinguished from normal operation, so this mission set is not as good as one would expect to demonstrate event recognition. However, given the complexity of obtaining such data, we will still present event recognition on this set of missions.

We initially focused on the depth measurements; altitude and bathymetry measure

[2]In all logs, a second, almost idle, AUV was also recorded, but was ignored in our experiments.

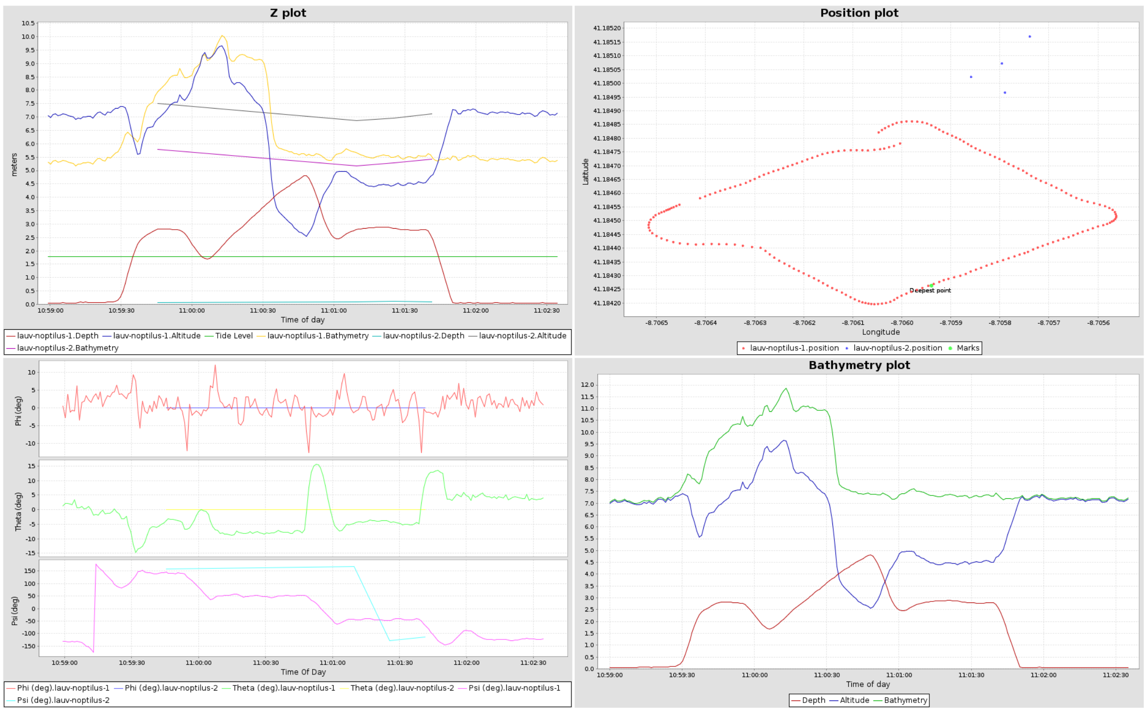


Figure 8: A mission log containing some abnormal event: z-plot (top-left), $x-y$-plot (top-right), Euler angles (bottom-left), depth (bottom-right).

properties of the environment the AUV is currently in, therefore they are not useful in recognizing the behavior of the AUV[3]. As mentioned before, our grammar learning approach takes some corpora of words and some parameters as input in order to learn a PCFG. Experimenting with the depth data, we found out that the best results were obtained when we used the standard split method (fixed word length) in order to break the data stream into words of symbols. The size of the word was set to 12, the size of our alphabet was set to 12, and the normalization factor λ was set to 0.05.

Figure 9 shows the events that the learned PCFG recognized as abnormal on a sample evaluation set from an abnormal mission. The abnormal event (approximately between time steps 500 and 1100) is marked indeed as abnormal (red color) in a large part. As we can see, it recognizes correctly the sudden submerge, but it only recognizes the beginning of the sudden emerge, mainly because emerging is part of the normal operation. Additionally, it reports correctly as abnormal the sudden transition from emerge to submerge. It can be easily seen that it correctly recognizes the beginning and the ending of the abnormal event, which has a unique footprint compared to the training set. Finally, the PCFG recognizes sporadic normal events, such as the initial submerge, as abnormal; the reason behind these false-negatives is that the training set is relatively small and does not contain a sufficient number of normal operation data, which can generalize the learned PCFG even more and make it more tolerant against outliers and noise.

[3]Bathymetry and/or altitude could possibly be used for event recognition, if the goal was to recognize abnormalities in the sea bottom.

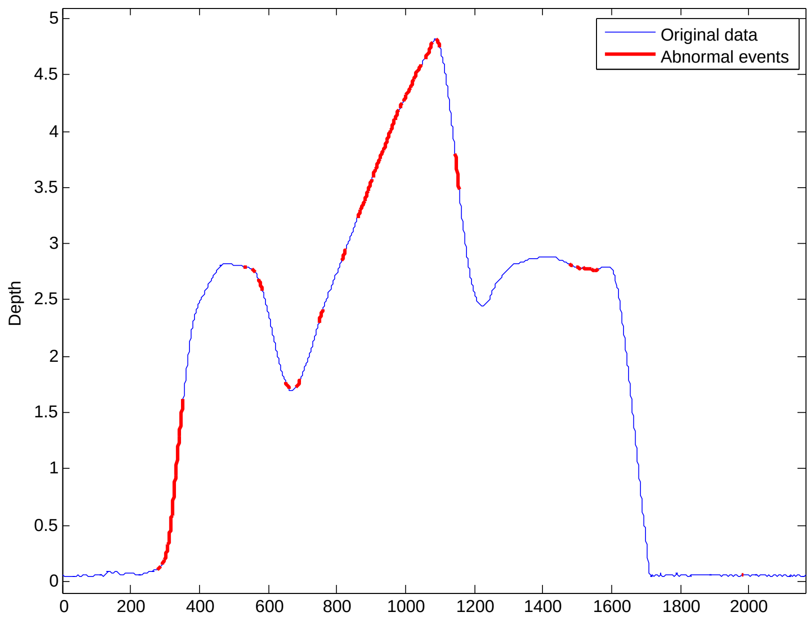


Figure 9: Annotated AUV depth data stream

Apart from the depth sensor data stream, we tried to recognize this kind of normal/abnormal events using the pitch sensor data stream. Data from the pitch sensor had to be sub-sampled due to their high frequency in order to make them sparser closer to the rate of event recognition (about 5 Hz). Figure 10 presents the results of normal/abnormal event recognition using the learned grammar on the pitch sensor data stream. In this experiment we used the standard split method (fixed word length) in order to break the data stream into words of symbols. The size of the word was set to 24, the size of our alphabet was set to 4, and the normalization factor λ was set to 0.05. Unfortunately, the results are not as good as those from the depth sensor data stream. The main reason for this may be that the abnormal event does not induce clear patterns within the pitch data stream and thus it goes occasionally unnoticed. Nevertheless, a good portion of the abnormal event (approximately between time steps 1000 and 2200) is marked indeed as abnormal (red color). Again, a number of false-negatives show up in normal parts of the mission for the reasons explained above.

8.2 Experimenting with On-Line Parsing

An example run of the on-line event recognition Dune task within the Neptus graphical environment is shown in Figure 11, where the hand-crafted grammar on the depth data stream (presented in Deliverable 6.1) is used to parse measurements of the depth sensor of the `lauv-noptilus-1` AUV in real time. The reported messages from the output of the event recognizer (`Up`, `Hover`, `Down` events) are displayed on the computer terminal that appears in the foreground, while the mission is ongoing and is being displayed within Neptus in the background. It should be noted that the use of the parser with Dune is transparent whether on simulation or on the real AUVs.

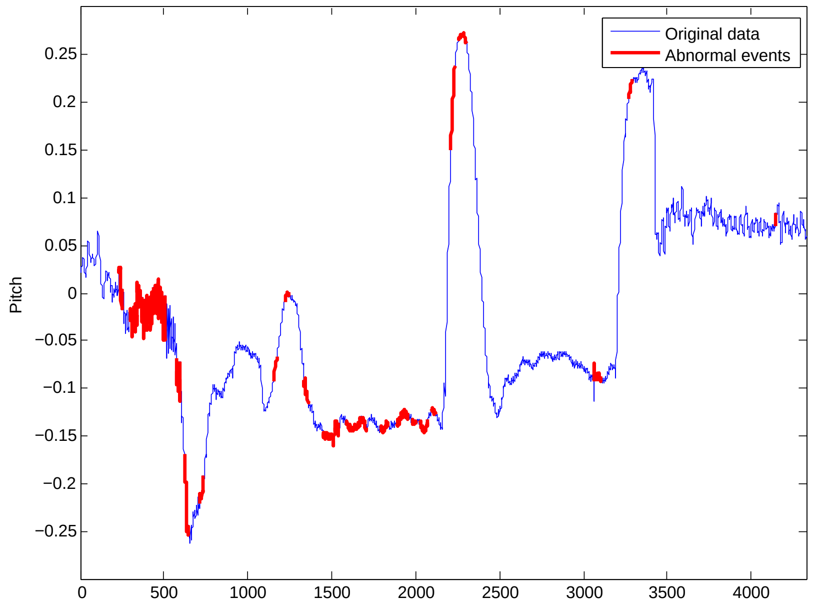


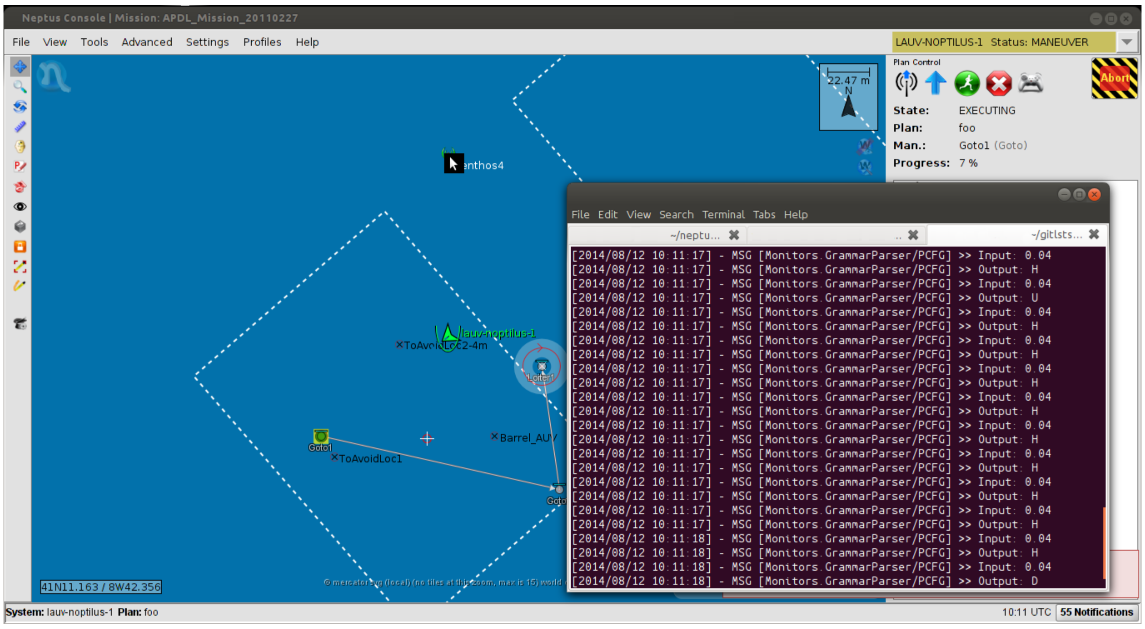
Figure 10: Annotated AUV pitch data stream

No modification is required to switch between these two modalities and in fact the only difference between simulation and the real vehicle is the source of the sensor measurements. The parser operates in the same way, requiring no modification to the source code or to the configuration file. The use of the simulator is highly valued, because recorded logs of entire missions can be replayed and data can be presented to the active Dune tasks, as if they are coming from the real AUV. The mission replay functionality is extremely useful in the context of event recognition, since multiple grammars may be tested on the same data set to find a good set of parameters. Additionally, new missions can be easily produced and executed in simulation to test new Dune tasks.

Two missions replayed in simulation are presented in Figure 12. Event recognition in these two cases relies on the handcrafted grammar over the depth data stream (presented in Deliverable 6.1). The on-line event recognition results from the parser Dune task are identical to the off-line event recognition results of the initial Matlab parser.

9 Conclusion

While progress is evident, a few more steps are required to fully complete Task 6.3. Regarding recognition of environmental situations (external to the AUV), a first direction is the possibility of parsing data streams from the salinity and temperature sensors. However, these sensors are not standard equipment of the AUVs and/or the environmental conditions during typical missions do not change much along these dimensions to detect any interesting events. Nevertheless, the proposed event recognition method is applicable to any data stream(s), so it is a matter of identifying



```
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: U
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:17] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:18] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:18] - MSG [Monitors.GrammarParser/PCFG] >> Output: H
[2014/08/12 10:11:18] - MSG [Monitors.GrammarParser/PCFG] >> Input: 0.04
[2014/08/12 10:11:18] - MSG [Monitors.GrammarParser/PCFG] >> Output: D
```

Figure 11: A run of the event recognizer on depth data within Dune and Neptus.

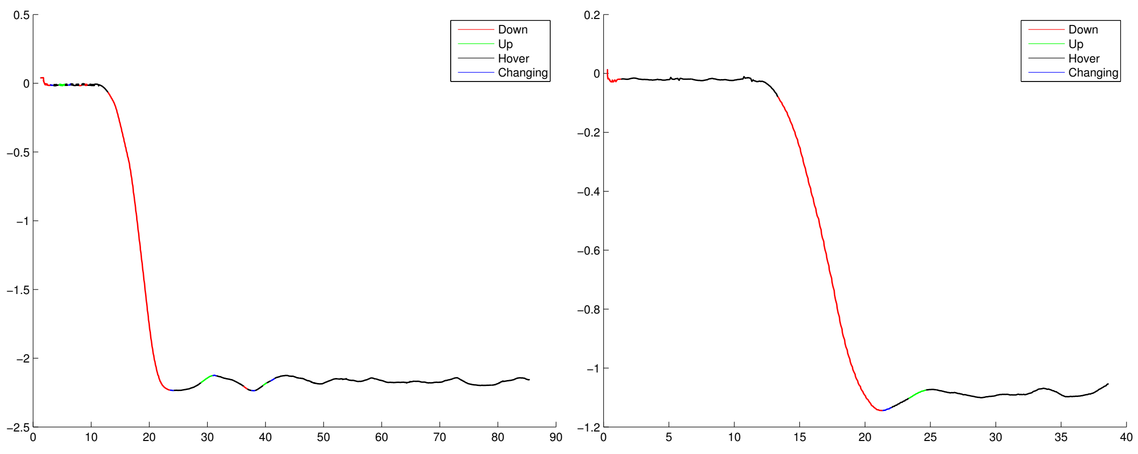


Figure 12: On-line event recognition in two missions: annotated AUV depth data

data that may carry some interest about the environment situation.

Regarding the multi-AUV nature of Noptilus, at this time multi-vehicle event recognition is rather prohibitive due to limited communication. However, the outcome of single-vehicle event recognitions can be easily communicated among the AUVs and affect decisions at team level. For example, if one AUV detects abnormal operation, the others may be notified and take over its assigned task by reassigning roles.

The ultimate goal of WP6 is to enable the operators of the AUV team to generate event recognizers from recorded logs (off-line) and seamlessly transfer and activate them on-board for event recognition (on-line). Provided communication with the mission operation center, any transmitted information on recognized events can be presented to operators at land/boat stations in ways that enhance awareness about the mission and simplifies the supervision of individual vehicles or real-time monitoring of the whole mission itself at a high level of description.

References

[1] C. D. Manning and H. Schütze, Foundations of Statistical Natural Language Processing. MIT Press, 1999.

[2] R. G. Gallager, Principles of digital communication. Cambridge University Press, 2008.

[3] K. Lari and S. J. Young, "The estimation of stochastic context-free grammars using the inside-outside algorithm," Computer speech & language, vol. 4, no. 1, pp. 35–56, 1990.

[4] T. Simonite, "The internet's innovation hub," MIT Technology Review, 2013.

[5] Laboratório de Sistemas e Tecnologias Subaquáticas (LSTS), "DUNE: Unified navigation environment, open-source on-board software solution for unmanned vehicles," 2014. [Online]. Available: http://lsts.fe.up.pt/software/dune