Learning from Past Missions Design for Multi-AUV Situation Understanding

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

NOPTILUS

Table of Contents

1 Objective 1

2 Definitions 2

3 Grammars 3
3.1 Context-Free Grammars . . . 3
3.2 Probabilistic Context-Free Grammars . . . 4

4 Event Recognition using PCFGs 9

5 Grammatical Parsing 10

6 Hierarchical Processing 12

7 Proof-of-Concept Case Study 13
7.1 Preprocessing . . . 14
7.2 Grammars . . . 15
7.3 Mission Log . . . 19
7.4 Results . . . 20
7.5 Performance . . . 23

8 Generalization 25

9 On-Board, Real-Time Event Recognition 27

10 Summary 30

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 two major 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.

Apart from the main two functions described above, the system delivered can be utilized in other meaningful ways. An off-board on-line use could be the real-time monitoring of the mission at a high level of description. Any information on recognized events transmitted by the on-board on-line system (acoustically or through Wi-Fi communication) can be presented to human operators at land/boat stations in ways that enhance awareness about the mission and simplifies the supervision of individual vehicles or monitoring of the whole mission itself. The two lines of research corresponding to Tasks 6.1 and 6.2 will be combined and merged in the context of Task 6.3, whose aim is to integrate the outcomes of the first two tasks.

The present document, Deliverable 6.1, titled "*Learning from Past Missions Design for Multi-AUV Situation Understanding*", according to the Noptilus DoW, *provides the details of the NOPTILUS situation understanding design that employs the "Learning from Past Data" methodology, as well as the results obtained through the application of the learning procedure using available human-operator data.* This deliverable describes the results obtained within Task 6.1, where focus is placed on the efficient, real-time operation of the recognition system on the AUVs. It turns out that the proposed methodology, based on parsing using formal grammars, relies heavily on the availability of past mission data, which have been analyzed and annotated carefully by human operators of the multi-AUV team in order to construct appropriate grammars. Learning is restricted only in automatically acquiring the numerical parts of a grammar, namely the probability distributions associated with each production rule. A more general approach to learning the structural parts of a grammar, a problem that falls under the umbrella of structured prediction, is investigated within Task 6.2 and is documented in Deliverable 6.2.

2 Definitions

Observation a measurement obtained by a sensor

Observation Stream a temporally-ordered list of observations

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

(Symbol) Sequence 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)

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

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

Input Sequence sequence of symbols generated from observations fed into a parser

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

Output Sequence a sequence of output symbols delivered by a parser

Grammar Hierarchy output sequences serve as input sequences at higher levels

3 Grammars

The power of *Probabilistic Context-Free Grammars (PCFGs)* has been demonstrated in various applications, mainly in the field of Natural Language Processing [1]. Previous work has shown that PCFGs can be utilized effectively [2, 3] for a variety of purposes [4]. Moreover, previous work has demonstrated that for the general problem of *event recognition*, the use of grammars is a viable approach [5–8] and comes with some appealing features:

Expressive Power Most approaches make use of *Markov Models* to express and capture useful events. PCFGs, as a generalization of these models, provide much better ability to describe structure and sequence compared to Markov Models. This property can be illustrated by showing that every event detectable by a (Hidden) Markov Model can also be detected by a PCFG formulation [9].

Compositional Power Given the fact that every parse operation reasons over an input sequence and produces an output sequence of recognized items, it is possible (and desirable) to compose complex, hierarchical recognition schemes, whereby the output sequences of multiple lower-level parsers are fed as input sequence to a higher-level parser, giving rise to PCFG hierarchies.

Robustness The PCFG approach is inherently robust against sensor noise, given the capability of expressing a probabilistic view of all possible input sequences into the rules. Moreover, the parsing operation itself is a *maximum a-posteriori* detector over all possible parses described by the grammar. Therefore, provided that the grammatical and probabilistic description of input scenarios in the PCFG is correct, the output of the parser is an *optimal detector*.

Reusability As long as the environmental assumptions incorporated into the grammar remain valid, a PCFG generated automatically by a learning process utilizing a large amount of past knowledge remains correct and can be reused in future missions for event recognition in similar environments.

3.1 Context-Free Grammars

Context-Free Grammars (CFGs) are formal tools used widely in Computer Science and Language Processing for specifying the syntax of programming and natural languages and for parsing sequences of symbols to identify their syntactic correctness and structure. Each CFG is characterized by a set of terminal symbols, which are the base symbols forming sequences, and a set of non-terminal symbols, which are intermediate symbols that characterize structural components of valid sequences. A CFG G is formally defined as a 4-tuple: (V, Σ, R, S), whereby V is a set of non-terminal symbols, Σ is an alphabet of terminal symbols, R is a set of syntactic rules defining how non-terminal symbols can be replaced by sequences of terminal and non-terminal symbols, and S is a start (non-terminal) symbol.

It is possible to construct valid sequences (*strings*) belonging to the language defined by a grammar through a process known as *derivation*. Derivation begins with the start symbol of the grammar and iteratively transforms the current symbol sequence of the derivation by applying rules of the grammar to non-terminal symbols until the sequence contains only terminal symbols. The characterization *context-free* implies that rules can be applied to any non-terminal symbol in a derivation sequence, regardless of the context, that is, the symbols preceding or following the chosen symbol. The power of CFGs is that, given an arbitrary sequence of terminal symbols, a *syntactic tree* can be constructed through a procedure known as *parsing* that describes the structural elements of the input (the combination of rules which yields that sequence from the start symbol), provided that the input can be derived from the grammar. If that input cannot be derived from the grammar, parsing yields no syntactic tree.

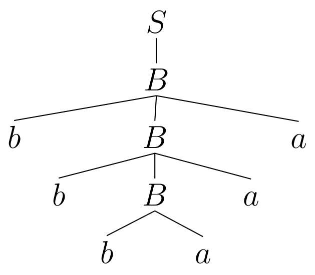
A simple grammar that encodes and produces all sequences of the form[1] $a^n b^n$ or $b^n a^n$ for $n \geq 1$ is the following:

$$\begin{aligned}
&G = (V, \Sigma, R, S)\\
&V = \{S, A, B\}\\
&\Sigma = \{a, b\}\\
&R = \{S \to A,\ S \to B,\ A \to aAb,\ A \to ab,\ B \to bBa,\ B \to ba\}
\end{aligned}$$

The sequence *bbbaaa* can be derived as follows

$$S \Rightarrow B \Rightarrow bBa \Rightarrow bbBaa \Rightarrow bbbaaa$$

and its syntactic tree is the following



In contrast, the sequence *aabbbb* cannot be derived from this grammar and, thus, no syntactic tree exists in this case.

3.2 Probabilistic Context-Free Grammars

Probabilistic Context-Free Grammars (PCFGs) extend CFGs by adding a probability value to each rule, so that the values over all rules for a certain non-terminal symbol form a valid probability distribution. During derivation, the choice of the rule for

[1]The notation a^n denotes n repetitions of a, for example $a^5 = aaaaa$, whereas a^* implies 0 or more repetitions of a. For a set F, F^* denotes the set of all strings with 0 or more symbols from F.

replacing a non-terminal symbol is drawn from the corresponding probability distribution. A PCFG implicitly defines a joint probability distribution over all possible sequences derived by the grammar. This probability value for any given sequence denotes "*how probable it is for the grammar to generate the given sequence*" and, therefore, how well the grammar *fits* the sequence and vise versa. Sequences that cannot be derived from the grammar have a probability value of 0.

A formal definition of a PCFG G can be given as a 5-tuple:

$$G = (V, \Sigma, R, P, S)$$

whereby:

V is a finite set of *items* (typically represented as alphanumeric strings) that comprise the vocabulary of *non-terminal symbols* of the grammar. They can appear on both sides of a production rule $r \in R$.

Σ is a finite set of *items*, known as the *terminal symbols* of the grammar, disjoint from V. These symbols form the alphabet for writing sequences derived from the grammar or provided as input to parsing. They can appear only on the right side of a production rule $r \in R$.

R is the set of production rules of the grammar. Formally, R is a finite relation between V and $(V \cup \Sigma)^*$. Each rule takes the form $l \to x$, where $l \in V$ is a non-terminal symbol and $x \in (V \cup \Sigma)^*$ is an ordered set of terminal or non-terminal symbols. Intuitively, each rule allows to *substitute* l with x in any derivation sequence.

P describes the *production probabilities* of each rule $r \in R$, so that the values over all rules with the same non-terminal symbol on their left side form a valid discrete probability distribution. These probabilities reflect the likeliness of selecting some rule to substitute a non-terminal symbol during a derivation.

S denotes the start symbol of the grammar. All valid derivation trees have this symbol as root. Conversely, parsing will have to *reduce* the input sequence to S, when applied to a valid (with respect to the grammar) sequence.

Derivations Every valid (with respect to a grammar) sequence can be generated (derived) by iteratively applying the rules of the grammar to a string w, which is initialized with the start symbol S, until w contains only terminal symbols. During the derivation, every non-empty intermediate string w can be written as:

$$w = \alpha v \beta, \text{ where } \alpha, \beta \in (V \cup \Sigma)^* \text{ and } v \in V$$

A non-terminal symbol $v \in V$ contained in w can be substituted by the right-hand side x of some rule $v \to x$. We say that string $w = \alpha v \beta$ *yields* string $z = \alpha x \beta$ through rule $v \to x$ and we write:

$$w \overset{v \to x}{\Longrightarrow} z$$
$$\alpha v \beta \overset{v \to x}{\Longrightarrow} \alpha x \beta$$

We can generalize this notion, using the closure of the *yields* relation. We say that w yields z if and only if z can be generated from w by applying zero or more rules of the grammar and we write:

$$w \overset{*}{\Rightarrow} z$$

Similarly, if z can be generated from w by applying one or more rules of the grammar, we write:

$$w \overset{+}{\Rightarrow} z$$

Every *recognized* string of a grammar is a sequence $w \in \Sigma^*$ of purely terminal symbols that can be yielded from the starting symbol S:

$$S \overset{*}{\Rightarrow} w$$

A particular derivation tree t of a "$S \overset{*}{\Rightarrow} w$" relation *models* a particular string w.

$$t \models w : \; S \overset{t}{\Longrightarrow} w$$

Due to grammar ambiguities (see below), there could be multiple trees that model the same string. A simple grammar, typically used for illustration, is the following:

$$G_p = \big(\{S\}, \{a, b\}, R_p, P_p, S\big)$$

The rules R_p and their corresponding probabilities P_p (in brackets) are defined as:

$$S \rightarrow aSa \qquad [p_1] \qquad (1)$$
$$S \rightarrow bSb \qquad [p_2] \qquad (2)$$
$$S \rightarrow bb \qquad [p_3] \qquad (3)$$
$$S \rightarrow aa \qquad [p_4] \qquad (4)$$

whereby $p_1 + p_2 + p_3 + p_4 = 1$. One can easily see that the strings produced and recognized by the grammar G_p are all palindrome strings in $\{a, b\}^*$, such as "abaaba", "abba", and "babaabab". Figure 1 shows the derivation trees for these examples. The derivation probabilities of these strings are p_1p_3, $p_1p_2p_4$, and $p_2p_1p_2p_4$ respectively, taking into account the rules used in each derivation (see below). Strings containing other symbols or strings that are not palindromes cannot be part of the language $L(G)$ of grammar G and have zero derivation probability.

Proper Grammars Not all valid grammars are considered proper. Improper grammars have fallacies in their description, meaning that their formulation is not *minimal*, in the sense of Minimum Description Length. Proper grammars have the following properties:

1. **no inaccessible symbols**: $\forall\, v \in V,\ \exists \alpha, \beta \in (V \cup \Sigma)^* : S \overset{*}{\Rightarrow} \alpha v \beta$
2. **no unproductive symbols**: $\forall\, v \in V,\ \exists w \in \Sigma^* : v \overset{*}{\Rightarrow} w$
3. **cycle-free derivations**: $\nexists\, v \in V : v \overset{+}{\Rightarrow} v$

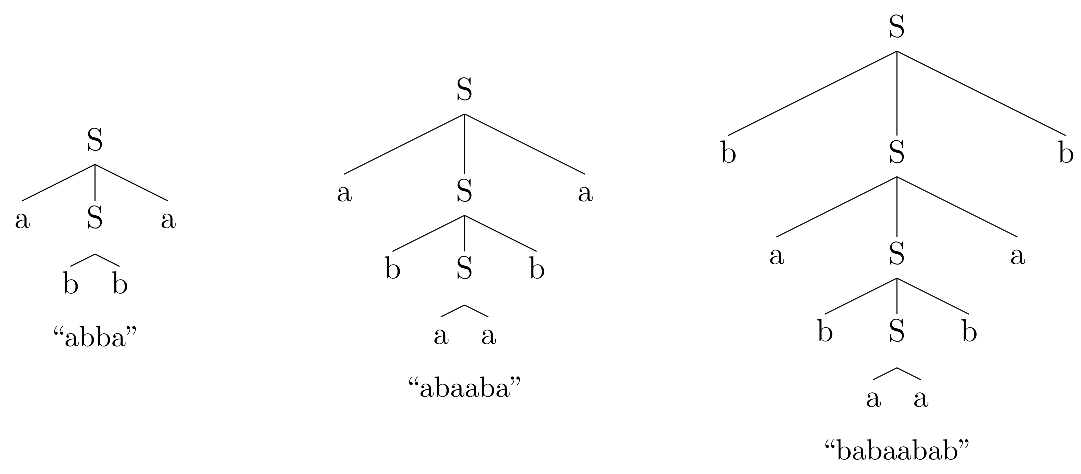


Figure 1: Derivation of various strings from grammar G_p.

Inaccessible and unproductive symbols can (and should) be removed from the vocabulary V causing no alteration in the set of input sequences (strings) that a particular grammar derives (or parses). The detection of cycles requires close examination of a grammar and there is no general rule on how to break them. The important subtle detail that arises from cycles in grammars is that, if a cycle exists in the derivation tree of a particular string (or a set of strings), then an infinite number of valid parse trees exist for that particular grammar by recursively introducing the cyclic structure into the parse tree.

Parsing Ambiguities It is possible to construct grammars able to derive a particular string in different ways leading to multiple syntactic trees. Note that the order in which rules are applied during the derivation does not affect the structure of the resulting tree. Therefore, ambiguity is a property with deeper causes in the rules of the grammar. To provide an example, let us define another simple grammar

$$G_a = \big(\{S\}, \{a\}, R_a, P_a, S\big)$$

with the following rules R_a and the corresponding probabilities P_a:

$$S \to SS \qquad [q_1] \tag{5}$$

$$S \to a \qquad [q_2] \tag{6}$$

This particular grammar can generate arbitrary sequence of a's, but there are more than one derivation trees for each sequence. For example, there are two possible parse trees for the string "aaa" that fully comply with the grammar, as shown in Figure 2. Our ultimate goal of event recognition allow us to ignore any difficulties that arise from such ambiguities. When considering the derivation probability of a string (see below), we take into account the probabilities of all possible derivations of this particular string by the grammar.

Sentence and Tree Probabilities PCFGs can be used for modeling stochastic environments in an intuitive and accurate way. Each rule $r \in R_p$ is associated with

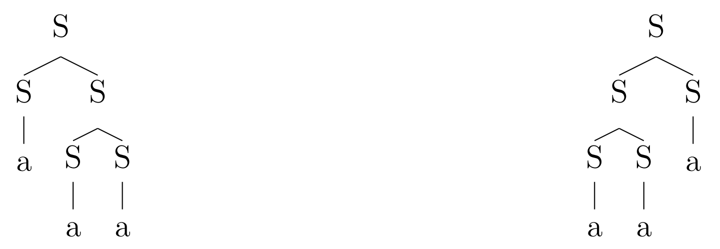


Figure 2: Multiple derivation trees for string “aaa” by grammar G_a.

a probability value $P(r)$. Due to the context-freeness property of the grammars, one can compute the probability of a particular derivation tree for a given string as the product of the probabilities of all the rules used in the derivation. For example, the strings “abaaba”, “abba”, “babaabab” are derived from grammar G_p as shown in Figure 1. The trees are generated from S by applying specific rules:

abba rules 1, 3: $P\big(\text{“}abba\text{”} \mid G_p\big) = P(1)P(3) = p_1p_3$

abaaba rules 1, 2, 4: $P\big(\text{“}abaaba\text{”} \mid G_p\big) = P(1)P(2)P(4) = p_1p_2p_4$

babaabab rules 1, 2 (twice), 4: $P\big(\text{“}babaabab\text{”} \mid G_p\big) = P(2)P(1)P(2)P(4) = p_1p_2^2p_4$

Grammar G_p is unambiguous, therefore the trees shown in Figure 1 are unique for each string and the probabilities shown above are the correct derivation probabilities for these strings. For the ambiguous grammar G_a, described earlier, one can easily see that the probability of each derivation tree of a particular string of length n is:

$$P\Big(t \;\Big|\; t \models a^n, G_a\Big) = q_1^{n-1}q_2^n$$

Essentially, probability values can be assigned to all trees, whether partial (containing at least one non-terminal symbol) or complete (containing terminal symbols only). The probability of a particular string a^n being generated by grammar G_a is the sum of the probabilities of all its possible derivation trees:

$$P\big(a^n|G_a\big) = \sum_{t:t\models a^n} P\Big(t \;\Big|\; t \models a^n, G_a\Big) = \sum_{t:t\models a^n} q_1^{n-1}q_2^n = \frac{(2n)!}{(n+1)!n!}\, q_1^{n-1}q_2^n$$

To understand this result, note that, due to our grammar rules, all syntactic trees are binary and the number of binary trees with exactly n leaves is given by the n-th Catalan number [10]. In general, the derivation probability of any given string is computed as the sum of the probabilities of all possible complete derivation trees. Obviously, strings that cannot be parsed by a particular grammar are assigned zero probability, for example $P\big(\text{“}abab\text{”} \mid G_p\big) = 0$.

The derivation probability of a string is useful information in the context of parsing. Given any input string, we can find the probability that this string can be derived by the grammar. If there are multiple derivation trees, we can even find the most probable tree. In general, we can utilize this information to classify several input sequences into different classes based on the probabilities of their respective derivation trees. This property is exploited for the purposes of event recognition.

4 Event Recognition using PCFGs

Utilizing the formal structure of a PCFG, one could create a grammatical structure that can distinguish between different interesting input structures and arrange them according to the probability of occurrence. Let us assume that we need to identify N different events and that a given input sequence w must be *classified* to one of these events. Let us assume a causality relation between the observed sequence and the event that occurred. In other words, we assume that a particular event generates (causes) the particular observed sequence. We shall assume N different grammars G_i, $i = 1, \ldots, N$, each one of which assigns a probability value to any particular input w. Intuitively, if the grammars are indeed *representative* of these N events, then we could simply extract the most probable event E_w that triggered the observation of the input w based on the parse probabilities of each of the grammars. Using a Bayesian approach, we can define the probability of an event E_i given an input sequence w as:

$$P(E_i|w) = \frac{P(w|E_i)P(E_i)}{P(w)}$$

and the extracted event is:

$$E(w) = \arg\max_i P(E_i|w) = \arg\max_i \frac{P(w|E_i)P(E_i)}{P(w)}$$

The probability of the input word $P(w)$ can be ignored in the maximization, since it only serves the normalization of the probability values and remains constant across the different events. Additionally, the probabilities $P(E_i)$ correspond to the *prior probabilities* of the considered events. In the absence of prior knowledge all event can be taken equally probable.

To express the above probabilities in terms of parse probabilities, a relatively weak assumption is made: an event is fully described by the corresponding grammar and no other grammar is correlated to that particular event:

$$P(G_i, E_j) = \begin{cases} 1 & \text{, iff } i = j \\ 0 & \text{, otherwise} \end{cases}$$

Therefore, we can simply substitute E_i with G_i in the decision rule above:

$$E_w = \arg\max_i P(w|G_i)P(G_i) \tag{7}$$

The simplicity of this approach allows an easy interpretation of an input sequence, by simply utilizing the parse probabilities. One can easily verify the following:

- If a particular event cannot generate the observed sequence (or equivalently, the input sequence cannot be derived by the corresponding grammar), then the assigned probability value is zero.
- If two or more events can generate the observed sequence (or equivalently, the input sequence can be derived by two or more grammars), then the most probable event is selected (recognized).

In addition to the above, an important and straight-forward transformation can be used; all event grammars can be combined into a single grammar as follows. Initially, a trivial renaming of the non-terminal symbols of all grammars takes place to ensure that no two grammars have common non-terminal symbols. Then, a single grammar G is constructed by taking the union of each of the elements of the N grammars. A new starting symbol S is created and the set of non-terminals, terminals, and rules is the union of the corresponding sets of the grammars G_i, $i = 1, \dots, N$. This new grammar is augmented by a set of N rules, which can (must) be applied to expand the new starting symbol to a particular event E_i, thus enforcing the application of the rules of the corresponding grammar G_i through its own starting symbol S_i:

$$S \to S_i \qquad\qquad [P(E_i)] \qquad\qquad i = 1, \dots, N$$

Note that the prior probabilities of the events are incorporated into the new grammar as the probabilities of selecting the rule applied to the root of any syntactic tree, effectively selecting the corresponding event.

Given the new global grammar, a parser must be designed to output which of the N rules was applied at the root of the derivation, thus the one corresponding to the highest parse probability. This rule yields a one-to-one mapping to the N detectable events and the parse probability is the event detection probability. The parser may select greedily the most probable parse tree among the possibly multiple N derivation trees.

It is important to note that in the general case, the grammars G_i can be ambiguous. Also, if the grammars describe totally independent sets of words (non-overlapping terminal symbols), then the events can be deterministically distinguished and the proposed formulation is a bit redundant. If a grammar G_i is the only grammar that can generate (and parse) a particular sequence, then no probabilistic parsing is needed to identify the event; all other grammars cannot parse the string, and, therefore, generate a parse probability equal to zero. We can deduce that all non-trivially *useful* grammars are *ambiguous*, therefore we need a parser that is capable of handling ambiguity.

5 Grammatical Parsing

A parser has been designed and optimized for on-line, real-time operation. Since our grammars are PCFGs, only two families of parsers need to be considered: Earley [11] and Cocke-Younger-Kasami (CYK) [12] parsers. Typically, Earley parsers are considered to be superior to CYK parsers, since for a given input of length n they exhibit an $\mathcal{O}(n)$ complexity. However, for ambiguous grammars, Earley parsers present the same complexity as CYK parsers, $\Theta(n^3)$.

Given the comparable performance of both families, we chose to design a CYK (or chart) parser, shown in Algorithm 1. Since our observation streams have unconstrained length, only a finite sequence from the stream can be considered for interpretation (parsing) at any time. A parse operation on a sequence of length w (i.e.

Algorithm 1 CYK algorithm for PCFG

```
Require: CNF PCFG $G = (V, \Sigma, R, P, S)$ and word $= s_1 \cdots s_n$ where $n \geq 2$
Ensure: $P(word)$ (if word $\in \mathcal{L}(G)$ then $P(word) > 0$, else $P(word) = 0$)
 1: $N[i,j] := \emptyset, i = 1, \ldots, n, j = 1, \ldots, n$
 2: for $i := 1$ to $n$ do
 3:   $N[i,i] := \{A,\ P(A \to s_i) : (A \to s_i) \in R\}$
 4: end for
 5: # End of span
 6: for $j := 2$ to $n$ do
 7:   # Start of span
 8:   for $i := j - 1$ to 1 do
 9:     # Partition of span
10:     for $k := i$ to $j - 1$ do
11:       if $(A \to BC) \in R$ and $B \in N[i,k]$ and $C \in N[k+1,j]$ then
12:         $prob := P(A \to BC) * P_{N[i,k]}(B) * P_{N[k+1,j]}(C)$
13:         if $A \in N[i,j]$ and $P_{N[i,j]}(A) < prob$ then
14:           $P_{N[i,j]}(A) := P_{N[i,j]}(A) + prob$
15:         else if $A \notin N[i,j]$ then
16:           $N[i,j] := N[i,j] \cup \{A, prob\}$
17:         end if
18:       end if
19:     end for
20:   end for
21: end for
22: if $S \in N[1,n]$ then
23:   return $P_{N[1,n]}(S)$, $N$
24: else
25:   return 0, $N$
26: end if
```

w consecutive symbols) would yield an event that describes the time interval corresponding to the w symbols in this sequence. Therefore, after an initial mute interval over the first w symbols in the stream, the parser will be able to start recognizing events using the w most recent symbols for any chosen length w. Therefore, our chart parser operates on a *sliding/rolling window* of fixed length w and generates an output (event) to characterize the corresponding time interval. The output sequence of the parser is almost as long as the observation stream (w less symbols at the very beginning).

Our parser exploits the fixed length of the parsing window and the overlap between consecutive windows, namely the fact that at each iteration the oldest symbol is discarded, while the new one is appended at the end of the sequence. Careful bookkeeping is used to store most of the intermediate parse results over a parse window and re-utilize them while parsing the next window. This simple optimization, when applied to the CYK algorithm, reduces the complexity from $\Theta(w^3)$ to $\Theta(w^2)$ at each iteration. Only the parsing of the first window will cost $\Theta(w^3)$, since there is no previous overlapping window. Therefore, in the limit of an infinitely-long observation stream the amortized complexity of each iteration is $\Theta(w^2)$ for some parsing window of length w. Since the rolling window has a fixed length, the complexity of the parser is constant and independent of the length of the observation stream.

After an initial cost of $\Theta(w^3)$, the CYK table N is filled up, containing all intermediate parse sub-trees for length n describing the observed sequence $s_1 s_2 \ldots s_N$. The table can be reused effectively, when a new symbol y is acquired. The first symbol is discarded (to keep the total length constant) and the resulting word becomes $s_2 \ldots s_N y$. Since the parse operation is defined on all sub-sequences, if $s_k \ldots s_l = s_p \ldots s_q$, then $N[k, l] = N[p, q]$. Each entry $N[i, j]$ previously described the parse sub-trees for $s_i \cdots s_j$. After the new symbol is added, the same information will be generated by the naive CYK parser in entry $N[i-1, j-1]$ for $1 < i < j < N$, since the sub-word $s_i \ldots s_j$ is simply shifted to $s_{i-1} \ldots s_{j-1}$. Therefore, a simple shift of all $N[i, j]$'s from the previous iteration to $N[i-1, j-1]$'s, for all $i, j > 1$, results in a correct partial sub-matrix for the next iteration. Only a single column, $j = N$, becomes invalid and needs to be recalculated. The rolling parser algorithm, shown in Algorithm 2, requires the sub-tree parse matrix N of the previous parse operation and only the new symbol y to perform the parse operation on the next window. It reuses all, but the first, rows of matrix N and computes only its last column.

Based on the grammar structure model described above, the output of the parser is essentially a symbol indicating the rule and, thus, the corresponding event that needs to be applied to the starting symbol of the grammar for generating the parse tree that maximizes the derivation probability of the input. The proposed optimizations on the proposed parser enable implementations for real-time operation on embedded systems, such as the Noptilus AUVs.

6 Hierarchical Processing

The output of a parser is a sequence of discrete symbols. Therefore, a straightforward extension of a parsing process is to use its output sequence as input sequence in another, higher-level, parsing process. Even better, several output sequences derived from parsing independent input streams could be combined to form a single input sequence for a higher-level parser, thus giving rise to hierarchies of grammars and/or parsers. Consider, for example, two distinct parsers (built from two different grammars) that operate independently on the observation streams generated by two different sensors. An interesting question that immediately arises is how one could use a third, higher-level grammar to generate a combined interpretation of the two sensor streams by parsing the outputs of the independent parsers. Given the discrete nature of the output of the two parsers, it is trivial to generate a new symbol stream, whereby the symbols are all possible combinations of outputs from the two parsers. This stream essentially contains processed, abstracted information from both sensors and, therefore, the third grammar operating on this stream could seek to recognize *higher-level* abstract phenomena or events related to the state of the system, based on the information from both sensors.

Algorithm 2 CYK rolling parser algorithm for PCFG

```
Require: CNF PCFG $G = (V, \Sigma, R, P, S)$ , symbol $y$, and matrix $N'$ of size $n \times n$
Ensure: $P(word)$ (if word $\in \mathcal{L}(G)$ then $P(word) > 0$, else $P(word) = 0$)
 1: $N[i, n] := \emptyset$, $i = 1, \dots, n-1$
 2: $N[n, n] := \{A,\ P(A \to y) : (A \to y) \in R\}$
 3: # Shift sub-trees
 4: $N[i, j] := N'[i+1, j+1]$, $i = 1, \dots, n-1$, $j = 1, \dots, n-1$
 5: # Start of span
 6: for $i := n-1$ to 1 do
 7:   # Partition of span
 8:   for $k := i$ to $n-1$ do
 9:     if $(A \to BC) \in R$ and $B \in N[i, k]$ and $C \in N[k+1, n]$ then
10:       $prob := P(A \to BC) * P_{N[i,k]}(B) * P_{N[k+1,n]}(C)$
11:       if $A \in N[i, n]$ and $P_{N[i,n]}(A) < prob$ then
12:         $P_{N[i,n]}(A) := P_{N[i,n]}(A) + prob$
13:       else if $A \notin N[i, n]$ then
14:         $N[i, n] := N[i, n] \cup \{A, prob\}$
15:       end if
16:     end if
17:   end for
18: end for
19: if $S \in N[1, n]$ then
20:   return $P_{N[1,n]}(S)$, $N$
21: else
22:   return 0, $N$
23: end if
```

Let us denote the possible generated events from the lower-level parsers as $\mathbf{E}^1 = \{E_1^1, E_2^1, \dots, E_M^1\}$ for the first parser (M events) and $\mathbf{E}^2 = \{E_1^2, E_2^2, \dots, E_L^2\}$ for the second one (L events). A grammar residing on top of these two parsers would accept as input symbols the Cartesian product of these two sets:

$$\Sigma = \mathbf{E}^1 \times \mathbf{E}^2 = \{E_1^1 E_1^2, \dots, E_M^1 E_1^2, E_1^1 E_2^2, \dots, E_M^1 E_L^2\}$$

This modular approach can accommodate any grammatical parsing scheme and any event can be "described" structurally using either sensor data or the output of another grammar.

7 Proof-of-Concept Case Study

To verify and demonstrate the performance of the proposed approach, an off-line, off-board simulation of the proposed event recognition system was implemented in MATLAB, utilizing real sensor data for generating the observation streams. Sensor data were captured directly by the AUV during a past mission, which was eventually aborted after a collision of the AUV with the sea floor[2].

The goal in this case study is to recognize the following categories of events:

Depth Events These events derive from the depth sensor measurement stream. The goal is to come up with a grammar/parser that describes abstractly the motion of the vehicle with respect to depth only. The output could be used for assessing the navigation performance of the platform with respect to depth.

Pitch Events These events derive from the Euler angle (pitch) sensor measurements of the vehicle. The goal is to find a grammar/parser that yields a high-level description of the state of the vehicle in terms of pitch. The output could be used for assessing the vertical orientation of the vehicle.

Diving Behavior Events These events derive from the combination of the depth and pitch observation streams, seen at the abstract level of the outputs of the two independent level-1 parsers discussed above. Note that depth and pitch are highly correlated, since the propulsion of the vehicle (propeller) is not directed and any diving actions (ascend or descend) will have to raise or lower the pitch. The goal here is to come up with a higher-level grammar that parses the combination of the two input streams. The output could serve the detection of faults, such as collisions, in the vertical (downward or upward) motion of the vehicle.

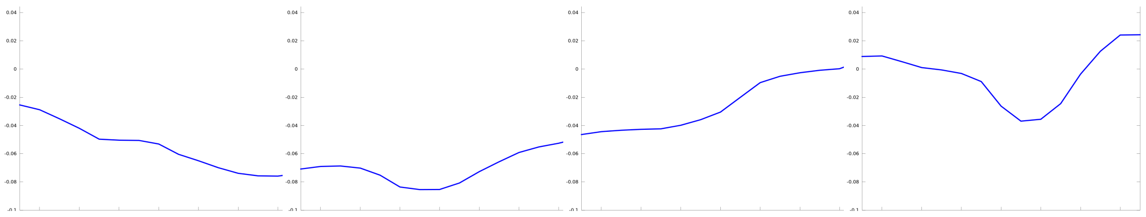
7.1 Preprocessing

For the purposes of our work, we assume that both sensors operate in a roughly synchronous fashion. Indeed, although the two sensors generate values at different rates, the ratio of these rates is fixed: five pitch values are generated by the *inertial measurement unit* in the same time period that one depth value is generated by the *depth sensor*. A preprocessing step is applied to the raw continuous measurements to generate discrete symbols (observations) as inputs to the parser.

For the depth data, since there is unlimited range, we chose to extract the rate of change instead of the actual depth. To this end, a simple differentiator with respect to time estimates the rate of change at each time using the previous value and the time interval between time steps. In particular, data are preprocessed as follows:

- A total of s consecutive raw measurements are collected.
- For each one of them the differentiator generates a rate of change.
- The s rates are averaged and a single rate is produced.
- This rate is quantized: negative ($-$), (approximately) zero (0), positive ($+$).
- The discrete symbol ($-$, 0, or $+$) is fed to the level-1 parser for depth.

[2]The entire log of the mission was kindly provided by our Portuguese partner FEUP.



+	0	–	0
0.037474	0.024209	-0.019689	0.028816
0.032219	0.017864	-0.017810	0.035222
0.028115	0.014133	-0.016547	0.041991
0.025039	0.013886	-0.022303	0.049693
0.022243	0.015512	-0.027118	0.050414
0.022903	0.016860	-0.032096	0.050569
0.026767	0.014031	-0.033769	0.053115
0.035741	0.009555	-0.030667	-0.006621
0.037393	0.003580	-0.019416	0.007136
0.034305	-0.000692	-0.000125	0.021382
0.028398	-0.002861	0.008617	0.029188
0.023691	-0.007767	0.017316	0.040740
0.024289	-0.012192	0.025421	0.043735
0.026708	-0.017634	0.028816	0.042784
0.027268	-0.019469	0.035222	0.042784

Figure 3: Example of depth data preprocessing (quantization into discrete symbols)

An example of this kind of processing on the depth measurement stream with $s = 15$ is shown in Figure 3. For the pitch data, since there is limited range $[-\pi, +\pi]$, we chose to simply quantize its value. In particular, data are preprocessed as follows:

- A total of $5s$ consecutive raw measurements are collected.
- The s values are averaged and a single pitch value is produced.
- This value is quantized: negative $(-)$, close to zero (0), positive $(+)$.
- The discrete symbol $(-$, 0, or $+)$ is fed to the level-1 parser for pitch.

This preprocessing procedure generates a sub-sampled stream of discrete symbols for each of the two sensors. Since the sub-sampling is synchronous, both parsers generate output symbols synchronously at a macroscopic time scale. Some degree of jitter is obviously expected to be present in the real-time system, but the variation of the rate of the sensors is in the sub-second time scale and is effectively smoothed out by preprocessing. The size s of the sampling window can be adjusted to meet the required frequency of symbol generation. The higher-level grammar operates on top of the (discrete) outputs of the two level-1 parsers, therefore there is no need for any kind of data preprocessing for its input.

7.2 Grammars

All three grammars in this case study were designed and tuned by hand, using our intuition about the target events of interest. Our parallel work investigates the employment of off-line learning methods for automatically constructing grammars from data in numerous past mission logs. Such a learning scheme will eliminate to a good extent the human factor from the process in the future.

The depth parser, using the grammar for depth, outputs one of the following events:

Up The vehicle is ascending (non-terminal symbol U).

Down The vehicle is descending (non-terminal symbol D).

Hover The vehicle is at a roughly constant depth (non-terminal symbol H).

Change The vehicle is performing a vertically convex or concave maneuver, (C).

Likewise, the pitch parser, using the grammar for pitch, outputs the following events:

Up The vehicle is pointing upwards (non-terminal symbol U).

Down The vehicle is pointing downwards (non-terminal symbol D).

Hover The vehicle is roughly level (non-terminal symbol H).

The depth and pitch grammars are defined on the set of terminal symbols generated during the preprocessing step: $\Sigma_{\text{Depth}} = \Sigma_{\text{Pitch}} = \{-, 0, +\}$. The starting symbol is E (from **E**vent) and the rules of both grammars are shown in Figure 4. Note that the two grammars are independent, therefore there is no problem in having common names for some of the symbols. As explained in Section 4, each grammar is a combination of different parse structures, each capable of recognizing a distinct event. These distinct events can be seen at the first rules of each grammar, which describe the derivation of each kind of event from the starting symbol. These rules are of the form:

$$E \to E_i \qquad [P(E_i)]$$

Briefly, the intuition behind the depth grammar is that an Up event (U) is characterized by arbitrary sequences of mostly +s and some 0s, collectively described by the non-terminal symbol u. Likewise, a Down event (D) is characterized by arbitrary sequences of mostly −s and some 0s, collectively described by the non-terminal symbol d. A Hover event (H) is mainly characterized by arbitrary sequences of mostly 0s and some +s or −s, collectively described by the non-terminal symbol h, however it may also contain some pairs of Up and Down events. Finally, a Change event is a specific sequence of the other events, namely a convex (DHU) or a concave (UHD) change. Similarly, the intuition behind the pitch grammar is that an Up event (U) is characterized by arbitrary sequences of mostly +s and only a few 0s or −s, collectively described by the non-terminal symbol u. Likewise, a Down event (D) is characterized by arbitrary sequences of mostly −s and only a few 0s or +s, collectively described by the non-terminal symbol d. Finally, a Hover event (H) is mainly characterized by arbitrary sequences of mostly 0s and some +s or −s, collectively described by the non-terminal symbol h, however it may also contain some pairs of Up and Down events.

A sampling window of size s and a parsing window of size w translate to one output event being extracted from the last sw data values of the depth sensor or $5sw$ data

$E \rightarrow C$	(Event **Change**)	[0.1]
$E \rightarrow U$	(Event **Up**)	[0.3]
$E \rightarrow D$	(Event **Down**)	[0.3]
$E \rightarrow H$	(Event **Hover**)	[0.3]
$D \rightarrow Dd$		[0.5]
$D \rightarrow d$		[0.5]
$U \rightarrow Uu$		[0.5]
$U \rightarrow u$		[0.5]
$C \rightarrow UHD$		[0.5]
$C \rightarrow DHU$		[0.5]
$H \rightarrow Hh$		[0.34]
$H \rightarrow h$		[0.34]
$H \rightarrow DHU$		[0.16]
$H \rightarrow UHD$		[0.16]
$d \rightarrow +$		[0.98]
$d \rightarrow 0$		[0.02]
$u \rightarrow -$		[0.98]
$u \rightarrow 0$		[0.02]
$h \rightarrow -$		[0.02]
$h \rightarrow +$		[0.02]
$h \rightarrow 0$		[0.98]

(a) Depth

$E \rightarrow U$	(Event **Up**)	[0.3]
$E \rightarrow D$	(Event **Down**)	[0.3]
$E \rightarrow H$	(Event **Hover**)	[0.4]
$D \rightarrow Dd$		[0.5]
$D \rightarrow d$		[0.5]
$U \rightarrow Uu$		[0.5]
$U \rightarrow u$		[0.5]
$H \rightarrow Hh$		[0.34]
$H \rightarrow h$		[0.34]
$H \rightarrow DHU$		[0.16]
$H \rightarrow UHD$		[0.16]
$d \rightarrow +$		[0.98]
$d \rightarrow -$		[0.01]
$d \rightarrow 0$		[0.01]
$u \rightarrow -$		[0.98]
$u \rightarrow +$		[0.01]
$u \rightarrow 0$		[0.01]
$h \rightarrow -$		[0.01]
$h \rightarrow +$		[0.01]
$h \rightarrow 0$		[0.98]

(b) Pitch

Figure 4: Grammar rules for (a) depth and (b) pitch events.

values of the pitch sensor. The value *sw* is a design parameter and various combinations were tested to study its effect on the recognition process. Figure 5 shows an example of parsing on a stream of input symbols using the depth grammar.

On top of the two independent grammars for depth and pitch, a third higher-level grammar is designed to detect diving behavior events related to both depth and

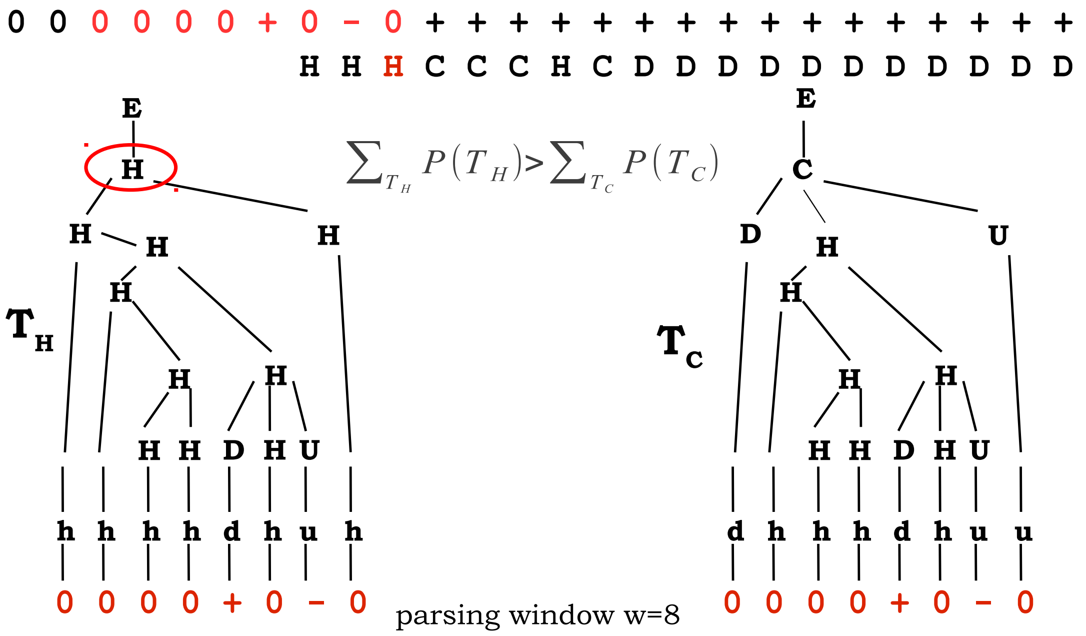


Figure 5: Example of parsing with the depth grammar on a window of $w = 8$ symbols: input and output streams (top) and syntactic trees of the Hover and Collision events for deriving the highlighted sequence of symbols (bottom). Hover is chosen.

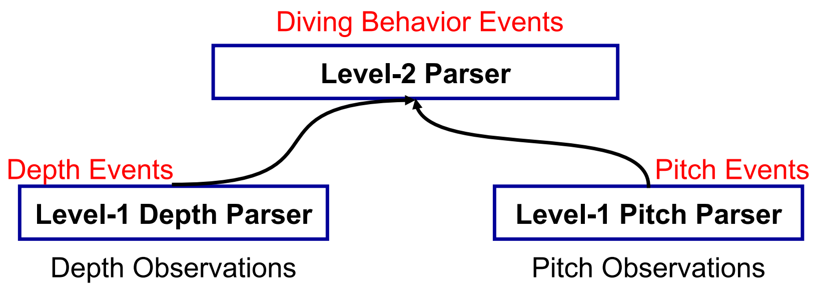


Figure 6: Two-level hierarchy of grammars for recognizing diving behavior events.

pitch. The two-level hierarchy of grammars is shown in Figure 6. This higher-level grammar is defined on an input symbol set which is the Cartesian product of the outputs of the lower-level grammars, whose parsers operate synchronously to each other. The input symbols (terminals) of this higher-level grammar are formed using concatenation of the output symbols with the depth parser output contributing the first (left) symbol and the pitch parser output contributing the second (right) symbol. Therefore, the terminal symbols of the higher-level grammar are:

$$\Sigma_{\text{Diving}} = \{UU, UD, UH, DU, DD, DH, CU, CD, CH, HU, HD, HH\}$$

Each symbol in this set denotes the events detected by the two lower-level parsers at the same time. The events detected by the parser with the higher-level grammar are the following:

OK Normal operation, no interesting event (non-terminal OK).

Collision Collision with the sea bottom (non-terminal Co).

Up Draft An upward current pushes the vehicle up (non-terminal UpD).

Down Draft A downward current pushes the vehicle down (non-terminal $DownD$).

The rules of the higher-level grammar are shown in Figure 7. This grammar is a little more involved. Intuitively, it states that an OK event is characterized by arbitrary sequences of symbols indicating that the underlying recognized events at the lower level mostly “match” each other, such as UU, DD, or HH. An Up Draft event is characterized by sequences mostly of symbols indicating either that the vehicle is hovering while it is pointing downwards (HD) or that the vehicle emerges while its pitch is level (UH). Likewise, an Down Draft event is characterized by sequences mostly of symbols indicating either that the vehicle is hovering while it is pointing upwards (HU) or that the vehicle submerges while its pitch is level (DH). Finally, a Collision event of the vehicle with the sea floor is indicated by a repeated mismatch between the recognized events at the lower level, whereby the vehicle seems to emerge while pointing downwards (UD).

7.3 Mission Log

The three grammars/parsers organized in a two-level hierarchy were tested on data from a specific logged mission aiming at verifying that we can eventually recognize a confirmed and known event contained in this log, namely a collision of the vehicle with the sea floor which occurred at around 450 seconds into the mission. The trajectory of the AUV during the named mission is shown in Figure 8 as a three-dimensional plot. The AUV started at the sea surface (green circle); after an initial descend phase, it alternated between hovering and descending taking a couple of right turns. Eventually, a collision occurred near the end of the trajectory at which point the mission was aborted. More specifically, this mission is characterized qualitatively by the following:

- The first 10-30 seconds indicate a rough agitated sea surface. The vehicle moves with the waves and the pitch sensor is quite noisy at this period.
- The first descend phase (30-160 seconds) is followed by a hovering phase (160-220 seconds) and then by a second descend phase (220-280 seconds).
- After second 280 the vehicle alternates between hovering and descending.
- The collision occurs at second 450: there is an evident “bounce” at the depth readings, and the pitch of the vehicle turns rapidly and unnaturally positive.
- After the collision the vehicle aborted the mission and surfaced; this part is not recorded in the log.

The measurement values of the depth and pitch sensors during the mission are graphed over time in Figure 9. The depth measurements are taken at a rate of 10Hz, whereas pitch measurements are taken at a rate of 50Hz. The rate of change

$E \rightarrow OK$	(**OK**)	[0.889]	$UpD \rightarrow UpD\ upd$	[0.5]
$E \rightarrow Co$	(**Collision**)	[0.001]	$UpD \rightarrow upd$	[0.5]
$E \rightarrow UpD$	(**Up Draft**)	[0.05]		
$E \rightarrow DownD$	(**Down Draft**)	[0.05]	$upd \rightarrow updt$	[0.9]
			$upd \rightarrow h$	[0.1]
$OK \rightarrow OK\ o$		[0.5]		
$OK \rightarrow o$		[0.5]	$updt \rightarrow HD$	[0.5]
			$updt \rightarrow UH$	[0.5]
$o \rightarrow u$		[0.3]		
$o \rightarrow d$		[0.3]	$DownD \rightarrow DownD\ downd$	[0.5]
$o \rightarrow h$		[0.3]	$DownD \rightarrow downd$	[0.5]
$o \rightarrow updt$		[0.025]		
$o \rightarrow downdt$		[0.025]	$downd \rightarrow downdt$	[0.9]
$o \rightarrow dot$		[0.05]	$downd \rightarrow h$	[0.1]
$u \rightarrow UU$		[0.9]	$downdt \rightarrow HU$	[0.5]
$u \rightarrow CU$		[0.1]	$downdt \rightarrow DH$	[0.5]
$d \rightarrow DD$		[0.9]	$Co \rightarrow Co\ co$	[0.5]
$d \rightarrow CD$		[0.1]	$Co \rightarrow co$	[0.5]
$h \rightarrow HH$		[0.88]	$co \rightarrow o$	[0.1]
$h \rightarrow CH$		[0.07]	$co \rightarrow cot$	[0.9]
$h \rightarrow DU$		[0.05]		
			$cot \rightarrow UD$	[1.00]

Figure 7: Grammar rules of the higher-level grammar for diving behavior events.

in depth estimated by our differentiator (Section 7.1) is also graphed in Figure 9 (a). Our goal is to annotate the mission with useful information about the occurred events.

7.4 Results

Initially, we run our lower-level parsers on the depth and pitch data for various values for the sampling window s and the parsing window w. The refresh interval is s times

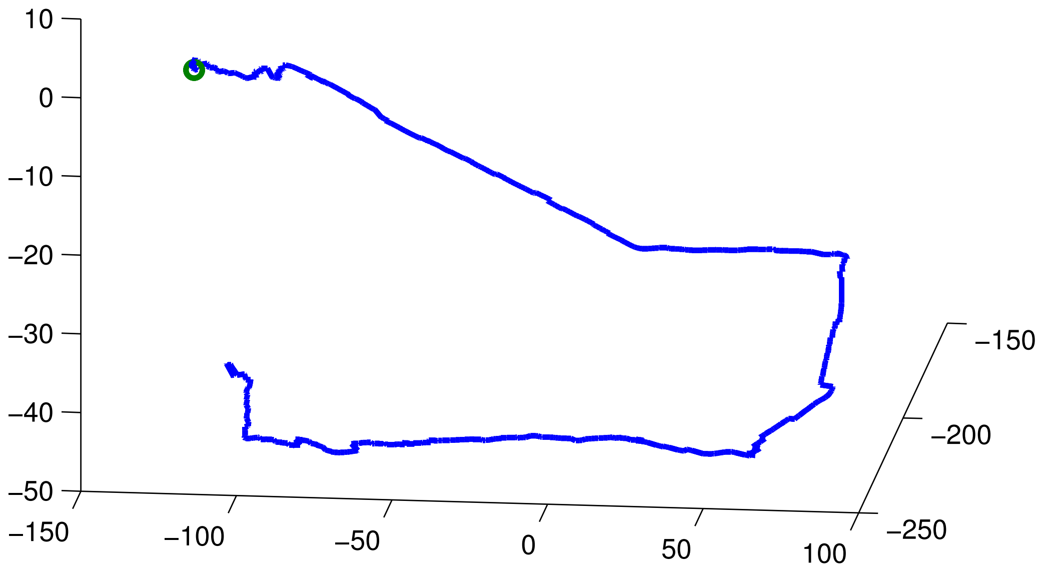
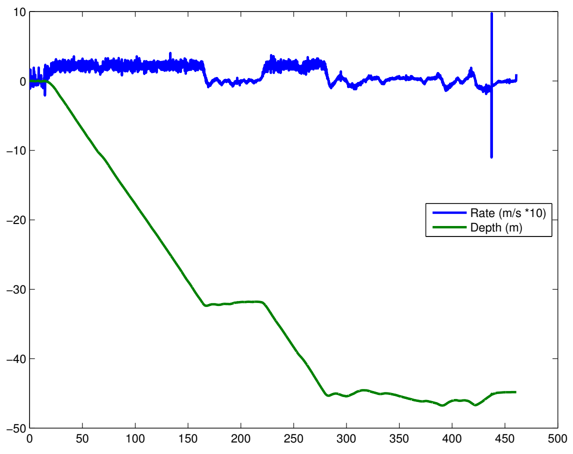
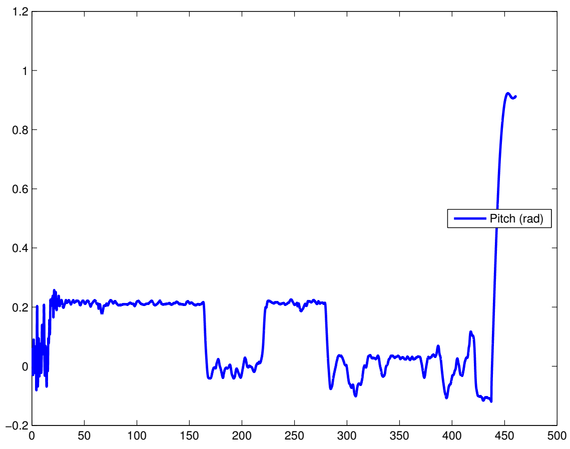


Figure 8: The AUV trajectory recorded in the mission log.



(a) Depth Measurements and Calculated Rate

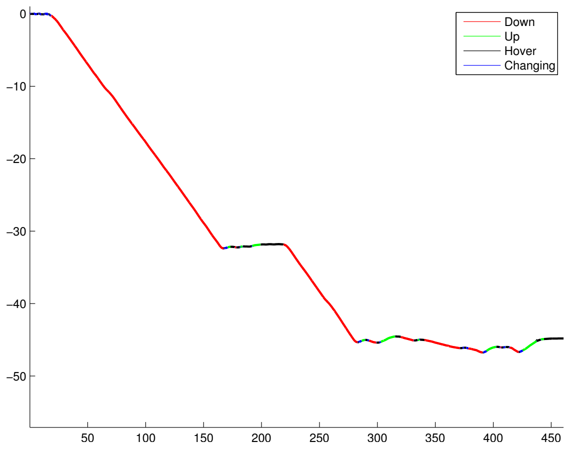


(b) Pitch Measurements

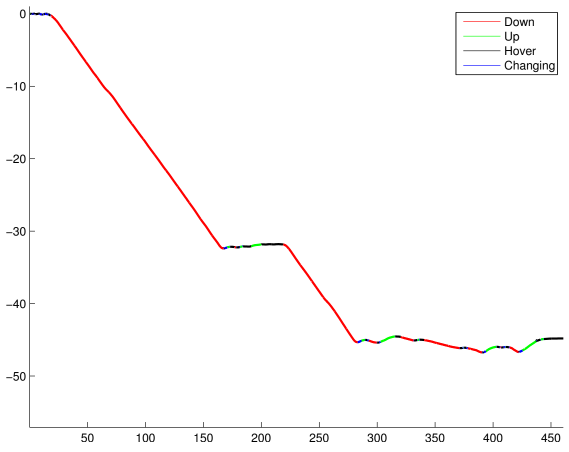
Figure 9: Depth and pitch sensor measurements from the mission log.

the period of the depth sensor (0.1 seconds) and the effective time window over which events are detected is sw times the period of the depth sensor. For various indicative values of s and w the event detected at each time step (annotated in color) for each of the depth and pitch data streams is presented in Figure 10 and Figure 11 respectively. In each case, the corresponding lower-level grammar was used within the parser. It is apparent that the parser correctly annotates the mission data and this annotation is insensitive to the precise choices of s and w, as long as the effective time window remains constant.

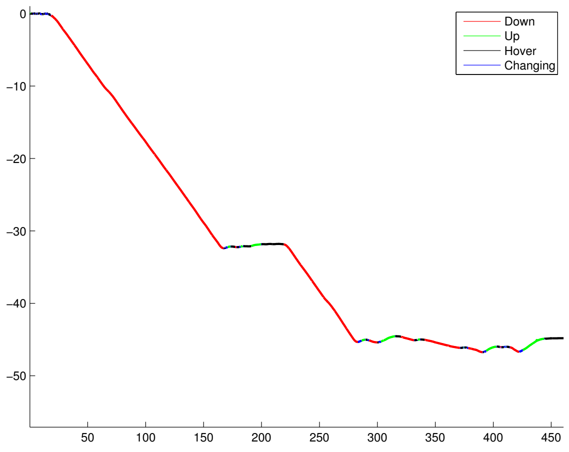
The event output of the higher-level parser for various indicative values of s and w is shown in Figure 12, whereby the depth data are now color-annotated by the high-level events. It is worth noting the accuracy of the detected events and especially



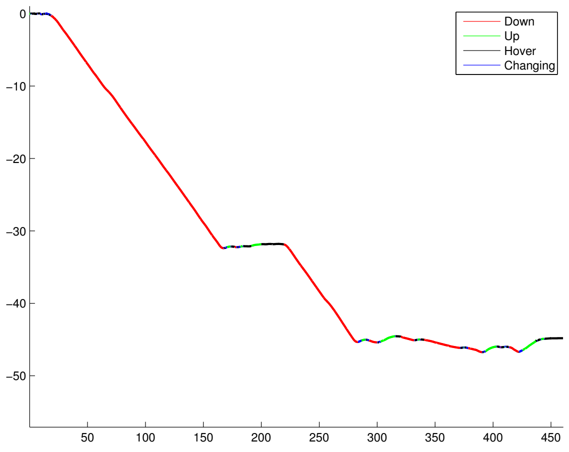
(a) $s = 2, w = 19$, eff. window 3.8 secs



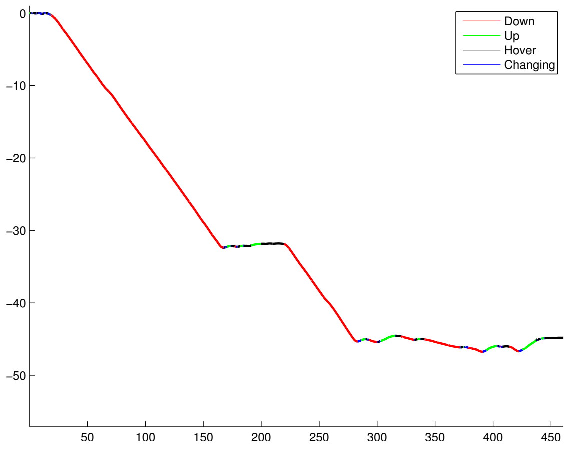
(b) $s = 3, w = 13$, eff. window 3.9 secs



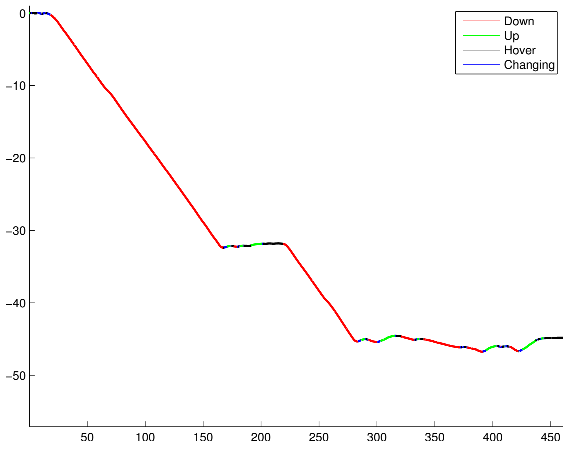
(c) $s = 4, w = 10$, eff. window 4.0 secs



(d) $s = 5, w = 8$, eff. window 4.0 secs



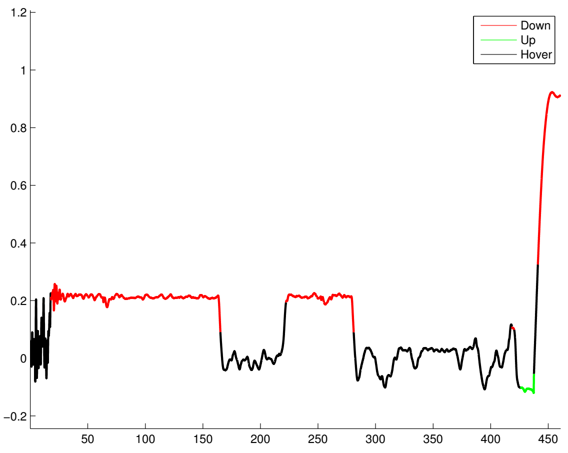
(e) $s = 6, w = 7$, eff. window 4.2 secs



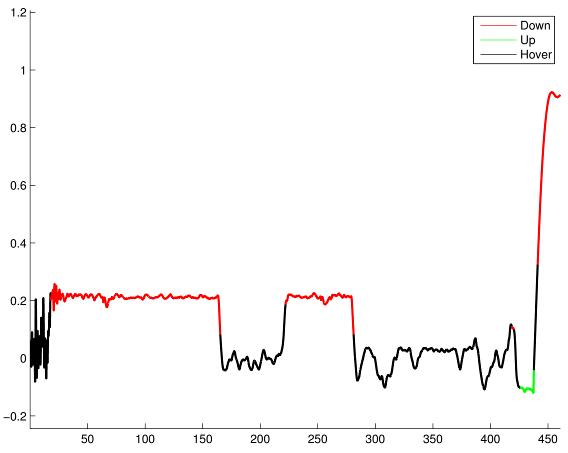
(f) $s = 7, w = 6$, eff. window 4.2 secs

Figure 10: Event annotation of the depth data stream for various time windows.

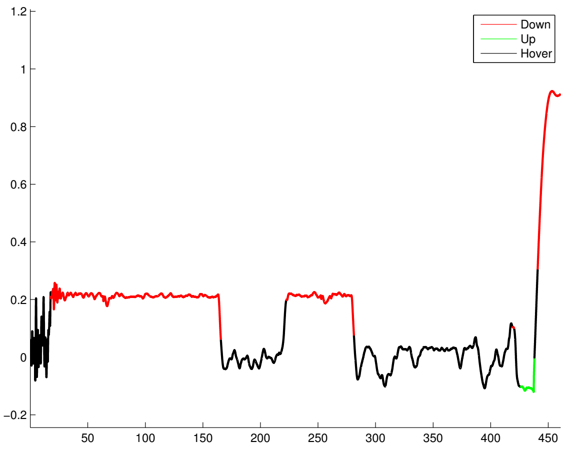
the detection of the collision at around 450 seconds (shown in red in Figure 12). Once again the results are insensitive to the precise choices of s and w, as long as the effective time window remains constant.



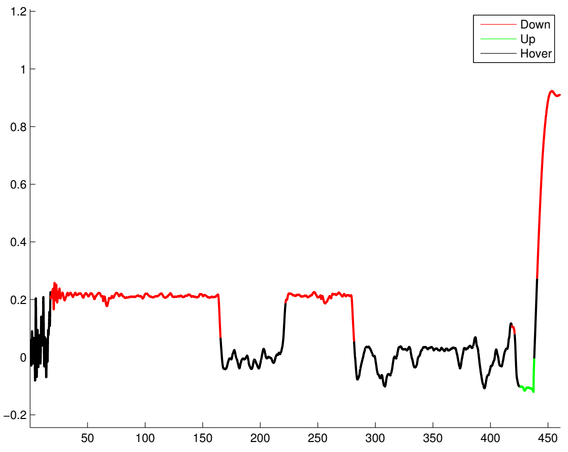
(a) $s = 2, w = 19$, eff. window 3.8 secs



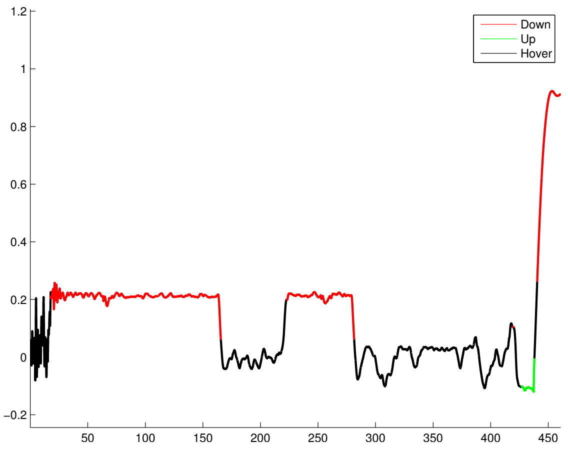
(b) $s = 3, w = 13$, eff. window 3.9 secs



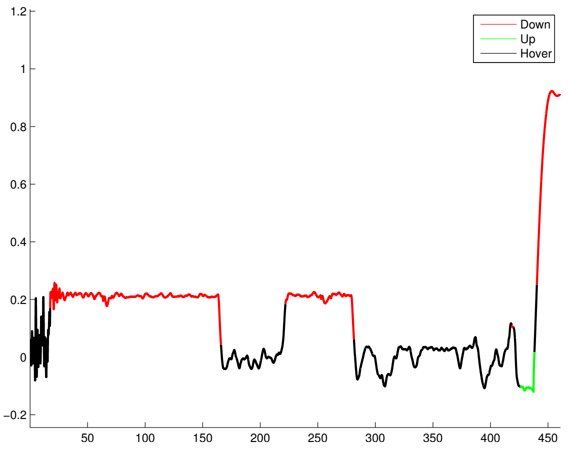
(c) $s = 4, w = 10$, eff. window 4.0 secs



(d) $s = 5, w = 8$, eff. window 4.0 secs



(e) $s = 6, w = 7$, eff. window 4.2 secs

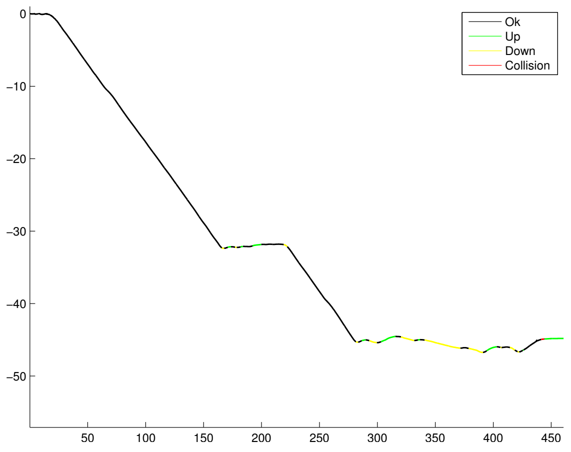


(f) $s = 7, w = 6$, eff. window 4.2 secs

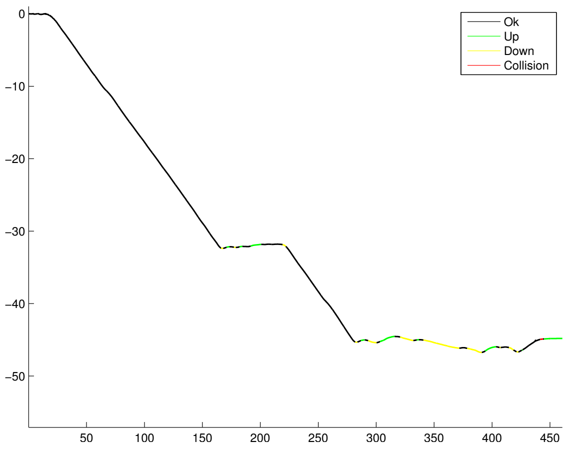
Figure 11: Event annotation of the pitch data stream for various time windows.

7.5 Performance

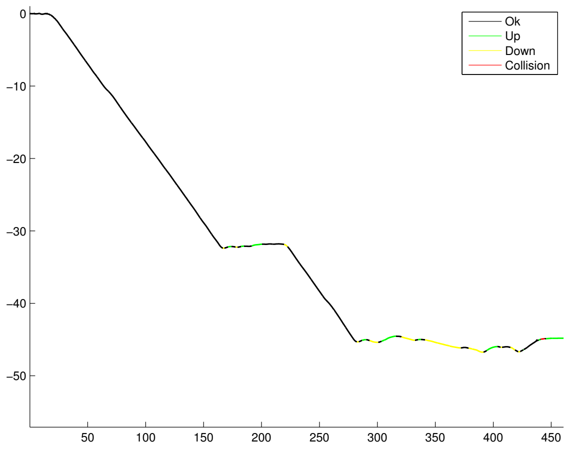
The results obtained on this log were quite promising. The annotation is robust to noise and to the selection of the parameters s and w. The desired time abstraction



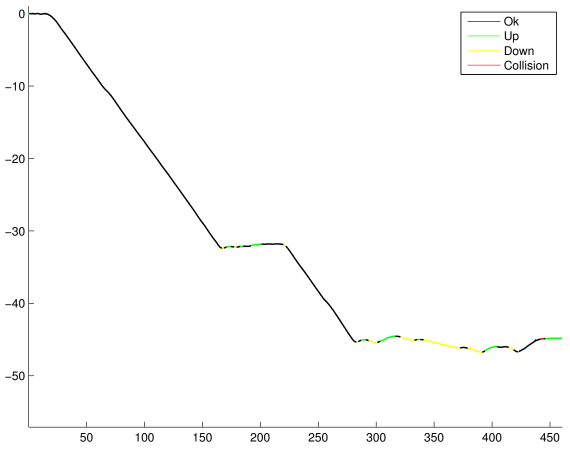
(a) $s = 2, w = 19$, eff. window 3.8 secs



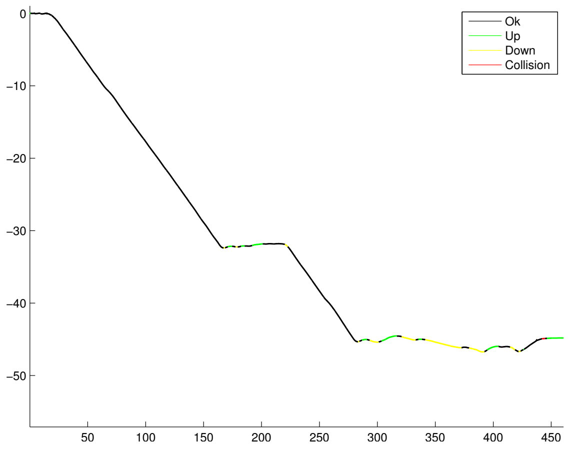
(b) $s = 3, w = 13$, eff. window 3.9 secs



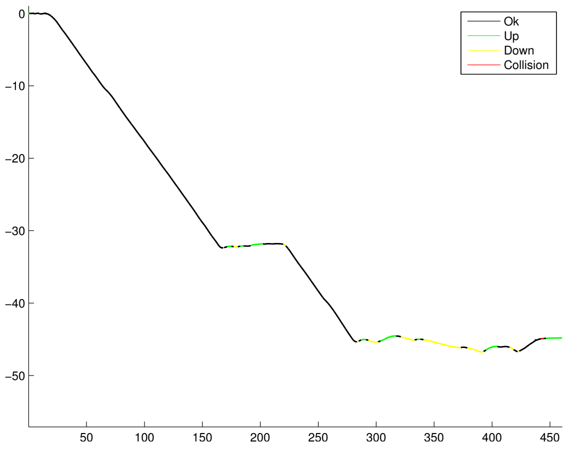
(c) $s = 4, w = 10$, eff. window 4.0 secs



(d) $s = 5, w = 8$, eff. window 4.0 secs



(e) $s = 6, w = 7$, eff. window 4.2 secs



(f) $s = 7, w = 6$, eff. window 4.2 secs

Figure 12: Event annotation of the depth data using the higher-level grammar.

and granularity can be achieved by selecting these two values appropriately, however it is an open question what the right level of time abstraction is. The answer is certainly dictated by the physical aspects of the system and the time extent of the events that need to be recognized. In any case, our approach can accommodate any set of choices and can be applied repeatedly to identify the right level.

The grammars annotate the data in an intuitively correct way. The two lower-level grammars generate a clean and robust abstraction from the raw data. This abstraction should prove indispensable in the long run. Despite the fuzziness of the pitch sensor data, the grammar manages to filter out the rapid motion of the vehicle and infer a hovering event.

One important omission of this case study is the positive buoyancy and inertia of the vehicle: the pitch value of the vehicle is out-of-phase with respect to the depth rate of change. This results in miss-detection of upward/downward water draft. This does not relate to the performance of the proposed scheme: it is an omission in the manually-constructed rules of the higher-level grammar. Such phenomena will be accounted for by the automated learning process. Despite this fact, the verified collision which occurred at around $t = 450$ seconds of the mission is *never missed*.

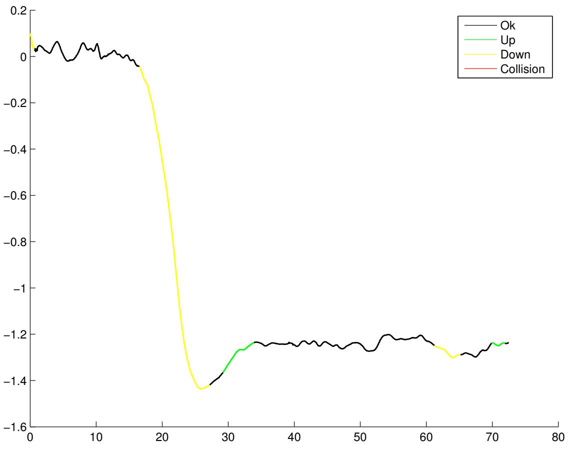
The performance, simplicity, and scalability of the hierarchical scheme are apparent. The real-time performance of the current parsers (implemented in MATLAB) appears to be feasible. Although the processing time on an embedded cannot be expected to be trivial, we estimate that the gain in robustness, compression, and modularity will justify the allocation of some memory and CPU time to event recognition.

8 Generalization

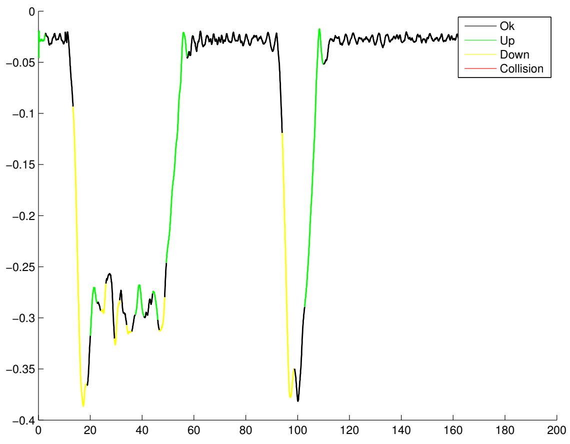
The proposed approach to recognition of depth, pitch, and diving behavior events using with the hand-crafted grammars was tested also on a variety of mission logs with and without collisions to confirm that events are correctly recognized and the developed grammars do not simply overfit the data in the initial log. Figure 13 shows the event annotation of the higher-level grammars on the depth data of various missions. The annotation of the two short missions indicates that occasionally the descend or ascend of the AUV is recognized as Up Draft or Down Draft, which may or may not be correct. Nevertheless, correctly no Collision event is reported. In the four long missions, a Collision event is reported only where it occurs. These results are quite promising, given the fact that the grammars employed in the parsers were only empirically crafted. In the course of the case study and during the additional experiments on other missions, the rules of the grammar did not really changed, but only the rule probabilities were tuned.

It should stretched out that event recognition in this study focused only on the vertical diving behavior of the vehicle measured by depth and pitch. Vertical collisions are correctly recognized, however other types of collisions, such as lateral ones, cannot be identified by the provided grammars. Figure 14 shows the annotation of two mission logs in which lateral collisions occurred (indicated by the spikes in the measurements), yet no collision was reported.

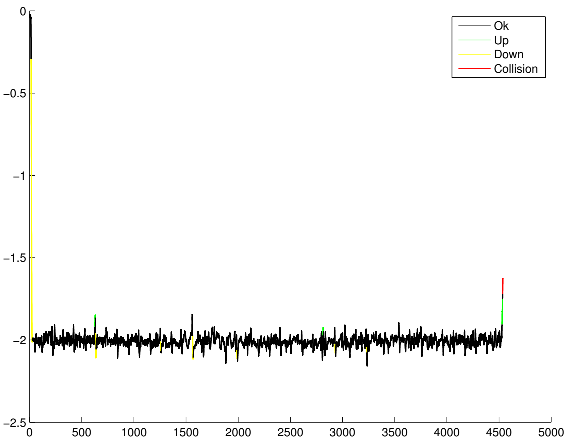
The proposed methodology, based on parsing using formal grammars, relies heavily on the availability of past mission data, which have been analyzed and annotated



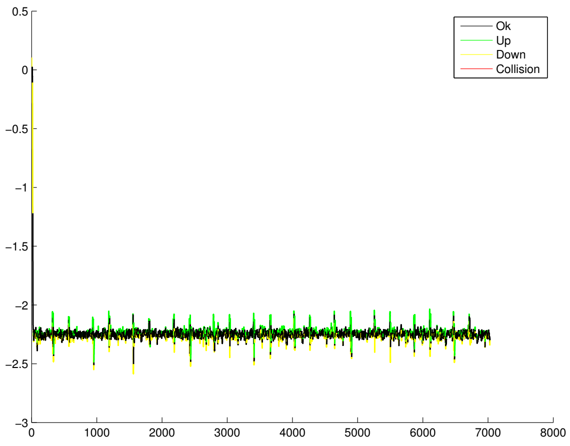
(a) Short log with no collision



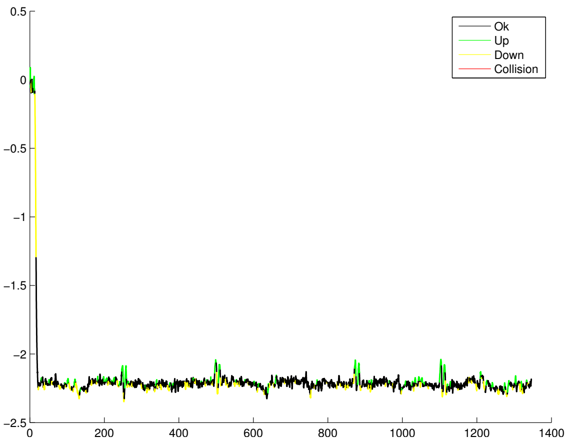
(b) Short log with no collision



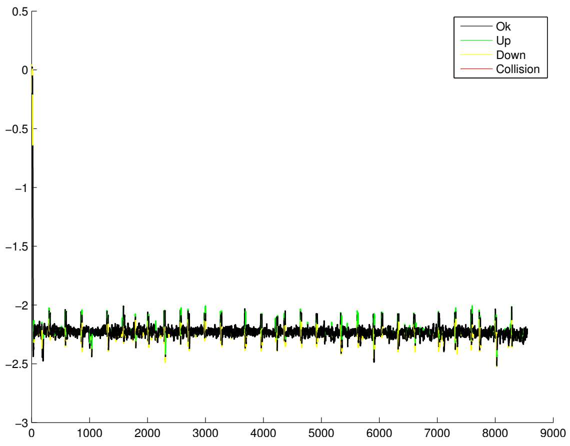
(c) Mission with a collision near the end



(d) Mission with no collision



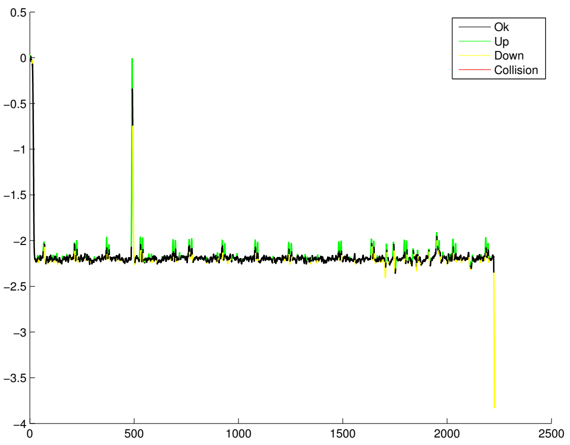
(e) Mission with no collision



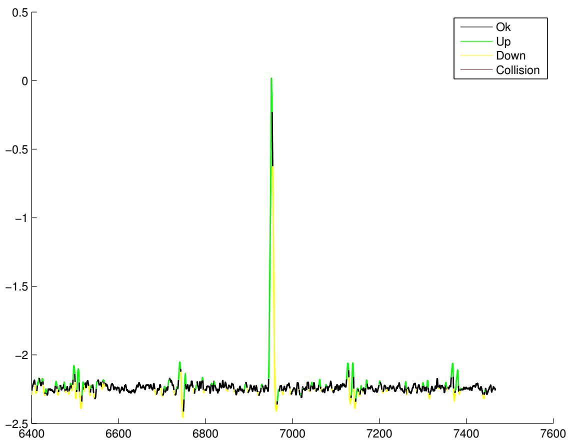
(f) Mission with no collision

Figure 13: Event annotation of depth data in various mission logs ($s = 10$, $w = 4$).

carefully by human operators of the multi-AUV team in order to construct appropriate grammars. A full machine learning approach in this context, which automates the construction of the entire grammar structure from past mission data, falls beyond the scope of this focused investigation on recognizing events. Such a comprehen-



(a) Mission with lateral collision



(b) Mission with lateral collision

Figure 14: Event annotation of depth data in various mission logs ($s = 10$, $w = 4$).

sive approach to learning the full grammar structure, a problem that falls under the umbrella of structured prediction, is investigated in parallel within Task 6.2 of the project. Nevertheless, we have experimented with a limited form of learning that fits within the present line of work. In this case, learning is restricted only in automatically acquiring the numerical parts of a grammar, namely the probability distributions associated with each production rule. Once the rules of the grammar have been written, a large training corpus of words associated (by experts) with each event included at the top rule of the grammar must be collected. Then, using the Inside-Outside algorithm [13], the probabilities of the rules can be estimated to reflect the frequencies in the training corpus. The Inside-Outside algorithm comes with a complexity of $\mathcal{O}(n^3)$, where n is the number of production rules of the grammar, but it needs to run only once off-line. Unfortunately, even this limited learning approach requires significant amounts of data, more so for the rare events we seek to recognize; such data are simply not available, unless one annotates by hand numerous missions, which is a rather tedious task. In light of these observations, the problem of learning was "pushed" into the parallel line of research, where a different, more general, approach to event recognition, which also facilitates learning, is considered.

9 On-Board, Real-Time Event Recognition

Motivated by the promising results of our Matlab implementation, we focused on an implementation of the event recognition parser within the Dune software architecture of the AUVs and its optimization for on-board, real-time execution (acquisition of measurements, abstraction to symbols, parsing, event recognition, and reporting) given an appropriate grammar. To this end, a number of changes were necessary to integrate the new parser task in Dune. The implementation of the parser in Dune has been completed and is currently being tested within the Dune/Neptus simulation environment in an isolated repository to avoid interference with ongoing

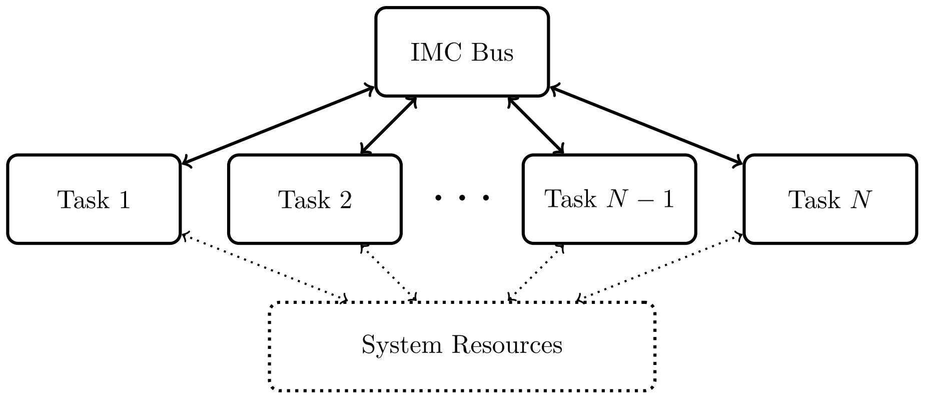


Figure 15: Information flow between Dune tasks.

work in Dune. However, to fully exploit the capabilities of the new parser task, the plan is to test it on the hardware of the real AUVs.

The Noptilus vehicles rely on the Dune software framework [14], upon which all the desired functionality is implemented. Small, contiguous pieces of software (*tasks* in Dune terminology) are executed in parallel. Each provides some kind of 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. The framework configuration of Dune is shown in Figure 15 from an information flow perspective.

The integration of a grammar parser for the purpose of event recognition into the existing platform relies on the task and message passing infrastructure to access information and make results available to the rest of the system. A new Dune task that executes the parser algorithm has been implemented. This new task feeds incoming data to the parser unit and outputs the outcome of the parse operation in the form of messages. Multiple instances of this task (each configured differently) can be executed on the same system to instantiate multiple event recognizers and accommodate a hierarchy of grammars/parsers. A new message type has been introduced within the Inter-Module Communication (IMC) component to store parsing information. At the same time, the parser requires input in the form of messages, which are transformed into symbols by some arbitrary processing and are then fed into the parser. When required by the grammar, multiple symbols can be concatenated into a single one. The processing flow of the parser task is presented in Figure 16.

The transformation of messages is handled at the preprocessing stage. Each message type requires a custom preprocessor, capable of extracting information from the data structure of the message. Each preprocessor transforms the information into a sequence of discrete symbols, ready to be parsed. The input message type can be arbitrary, coming either directly from a sensor or from the output of some

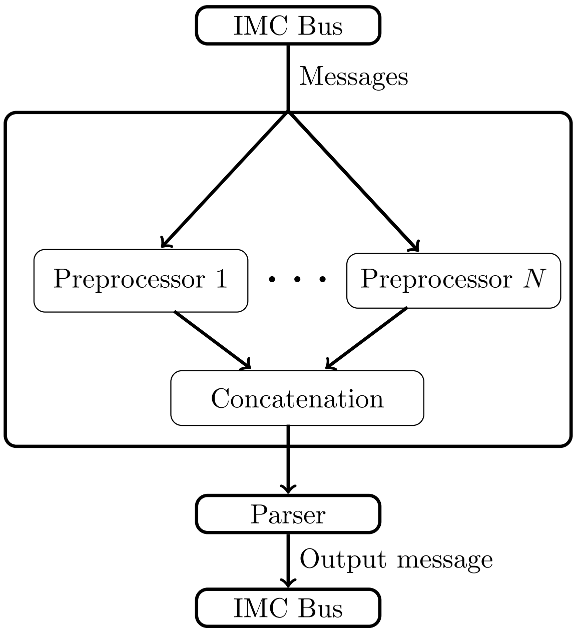
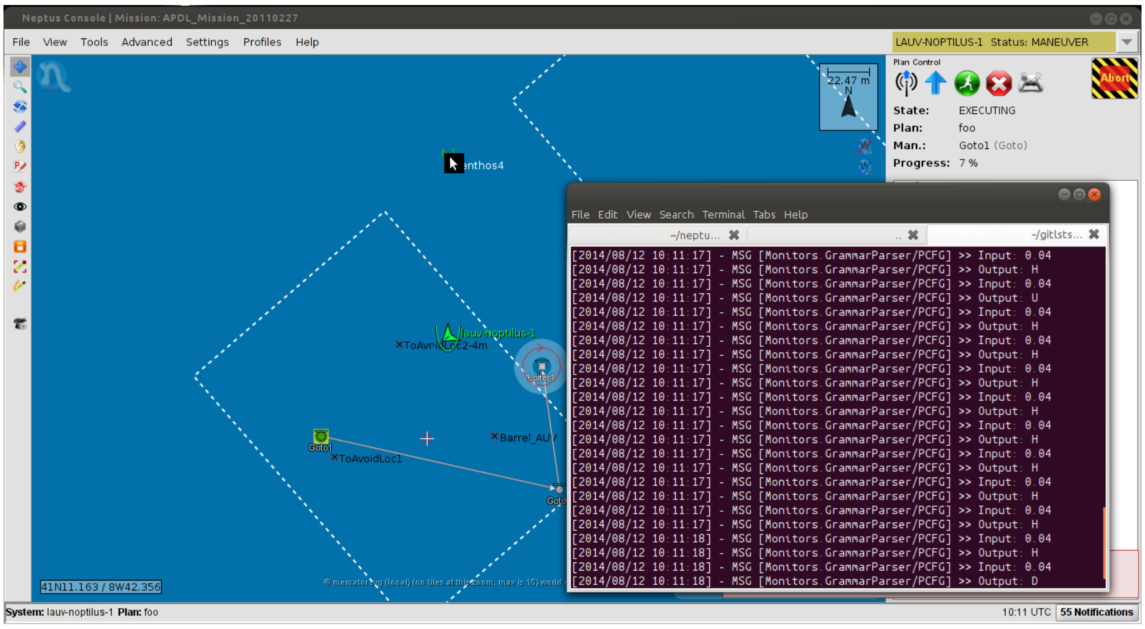


Figure 16: Information flow inside the parser.

other parser. A preprocessor can be constructed to generate symbols by integrating, differentiating, averaging, quantizing, or otherwise transforming the input data. To accommodate hierarchical grammar configurations, a simple pass-through preprocessor exists, which simply extracts the generated symbol of an incoming parser message, originating from a different parser task instance, and feeds it to the current parser.

Each preprocessor generates symbols asynchronously and these are fed to the concatenation module, which handles the synchronization issues. In the case of a single source preprocessor, no concatenation or synchronization is necessary. Each concatenated symbol is generated by the most recent preprocessor outputs, thereby ensuring that all information sources are as closely synchronized as possible. Normally, all preprocessors generate some output after some initial time, as each one receives sufficient input information in the form of messages. During initialization, not all preprocessors have sufficient input data to operate, since the buffers/windows are not filled up. In this interim period, no concatenated output can be generated and no parse operations can be executed, until sufficient data have arrived.

Each parser task instance requires a set of configuration parameters to function. First, an ordered set of preprocessors are selected. Each preprocessor is configured independently for each parser task instance. Then, a grammar is configured to the parser. The matching of input symbols (concatenated preprocessor outputs) to grammar terminal symbols is the last parameter that needs to be defined. The output message is tagged with the chosen name of the parser instance to facilitate hierarchies of grammars. The configuration scheme is integrated with the existing configuration framework of Dune, further facilitating integration and usage on the real Noptilus vehicles.



```
[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 17: A run of the event recognizer on depth data within Dune and Neptus.

An example run of the grammar-based event recognition activity within the Dune and Neptus simulation environment is shown in Figure 17, where the hand-crafted depth grammar is used to parse measurements of the depth sensor of the AUV `lauv-noptilus-1` 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.

10 Summary

In this work, we proposed an approach to the problem of event recognition based on grammatical parsing. In summary, the following qualities of the proposed event recognition system should prove invaluable in the domain of AUV missions, but also in other domains where recognition of events is required:

Representation The expressiveness of grammars provides intuitive means to express a large number of event structures, sequences, and scenarios.

Compression The detected events can be stored (and transmitted) as a dense and smooth abstraction of the raw sensor data. As a result, decision making will be greatly facilitated by operating on top of event recognition.

Reusability Grammars can be expanded and/or reused from mission to mission, generating a library of compact and understood descriptions.

Robustness Experimentation so far indicates that parsing results are quite stable against measurement noise and/or variations of design parameters.

References

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

[2] S. Petrov and D. Klein, "Learning and inference for hierarchically split PCFGs," in *Proceedings of the 22nd National Conference on Artificial intelligence (AAAI)*, 2007, pp. 1663–1666. [Online]. Available: http://dl.acm.org/citation.cfm?id=1619797.1619916

[3] A. Belz, "PCFG learning by nonterminal partition search," in *Proceedings of the 6th International Colloquium on Grammatical Inference (ICGI): Algorithms and Applications*, 2002, pp. 14–27. [Online]. Available: http://dx.doi.org/10.1007/3-540-45790-9_2

[4] G. Neu and C. Szepesvári, "Training parsers by inverse reinforcement learning," *Machine Learning*, vol. 77, no. 2, pp. 303–337, 2009. [Online]. Available: http://dx.doi.org/10.1007/s10994-009-5110-1

[5] G. Guerra-Filho, C. Fermuller, and Y. Aloimonos, "Discovering a language for human activity," in *Proceedings of the AAAI Fall Symposium on Anticipatory Cognitive Embodied Systems*, 2005, pp. 70–77. [Online]. Available: http://ranger.uta.edu/~guerra/Guerra-FilhoFS05discovering.pdf

[6] D. Lymberopoulos, A. S. Ogale, A. Savvides, and Y. Aloimonos, "A sensory grammar for inferring behaviors in sensor networks," in *Proceedings of the 5th International Conference on Information Processing in Sensor Networks (IPSN)*, 2006, pp. 251–259. [Online]. Available: http://doi.acm.org/10.1145/1127777.1127817

[7] S. C. Geyik and B. K. Szymanski, "Event recognition in sensor networks by means of grammatical inference," in *Proceedings of the 28th IEEE International Conference on Computer Communications*, 2009, pp. 900–908. [Online]. Available: http://www.cs.rpi.edu/~geyiks/geyik_infocom09.pdf

[8] A. S. Ogale, A. Karapurkar, and Y. Aloimonos, "View-invariant modeling and recognition of human actions using grammars," in *Workshop on Dynamical Vision*, 2005, pp. 115–126. [Online]. Available: http://www.cs.umd.edu/~ogale/papers/actionsICCV05WDM.pdf

[9] A. Jagota, R. B. Lyngsø, and C. N. S. Pedersen, "Comparing a hidden Markov model and a stochastic context-free grammar," in *Proceedings of the First International Workshop on Algorithms in Bioinformatics (WABI)*, 2001, pp. 69–84. [Online]. Available: http://www.springerlink.com/content/pw22u99rnv8q9p00

[10] Wikipedia, "Catalan number — wikipedia, the free encyclopedia," 2012. [Online]. Available: http://en.wikipedia.org/w/index.php?title=Catalan_number

[11] J. Earley, “An efficient context-free parsing algorithm,” *Communications of the ACM*, vol. 26, no. 1, pp. 57–61, January 1983. [Online]. Available: http://doi.acm.org/10.1145/357980.358005

[12] A. V. Aho and J. D. Ullman, *The Theory of Parsing, Translation, and Compiling*. Prentice-Hall, Inc., 1972.

[13] 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.

[14] 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