Decision-time Adaptive Replanning (DAR) Behavior for Benthic Environment Monitoring using AUVs*

Eric Guerrero[1], Francisco Bonin-Font[1] and Gabriel Oliver[1]

Abstract— **This work presents the development and field testings of a novel active exploration framework for mapping benthic underwater environments using AUVs. This framework is based on a *decision-time adaptive replanning* (DAR) behavior that works together with a *sparse Gaussian process* (SGP) for online environmental modeling and a *Convolutional Neural Network* (CNN) for semantic image segmentation. The SGP uses semantic data obtained from stereo images to probabilistically model, online, the spatial distribution of certain species of seagrass that colonize the sea bottom forming extend meadows, and provide a measure of sampling informativeness to the adaptive behavior. The adaptive behavior (DAR) has been designed to execute successive informative paths, without stopping, considering the newest information. We solve the *information path planning* (IPP) problem by means of a *depth-first* (DF) version of the *Monte Carlo tree search*(MCTS). The DF-MCTS method has been designed to explore the state-space in a depth-first fashion, provide solution paths of a given length in an *anytime* manner, and reward smooth paths for field realization with non-holonomic robots. The complete active exploration framework has been integrated in a ROS environment as a high level layer of the COLA2 software architecture. We prove the effectiveness of the proposed active exploration framework by providing the results obtained during field test.**

I. INTRODUCTION

Many applications in robotics share the common objective of exploring an unknown environment for recording data to represent it. *Information Gathering* (IG) algorithms guide such exploration using an information metric which represents the informativeness of the environment variable under study in particular locations, and is used to drive the data recording process towards the more informative spots whilst minimizing a cost. The methods developed for such exploration are often differentiated by four components: (1) the technique for modeling the environment; (2) the information function; (3) the *Informative Path Planning* (IPP) strategy, and (4) the adaptive strategy for replanning.

One way to obtain environmental models is applying a *Gaussian Process* (GP) fed with the data collected by an AUV [1]. GP are a powerful nonparametric technique that can handle a large variety of problems, and have the ability to learn spatial correlation with stochastic noisy measured data. The key feature of GP for IG algorithms is their ability to handle both, data uncertainty and data incompleteness. The *information function* (I) is used to point out the more relevant spots to visit (or revisit), and it is directly related to the environmental variable under study. Whilst the authors in [2] propose the use of an interpolated version of the GP model predicted variance, most of the methods use either *Differential Entropy* (DE) or *Mutual Information* (MI) as information function. Nonetheless, using DE results in lower computation times and higher uncertainty reduction than using MI when computed from a GP prediction in a stationary setup [3].

Informative planners define the most informative trajectories for IG, applying constraints such as the planning time or the travelling cost. Whilst graph-based solutions are usually time expensive, they have been used for small state-space scenarios [4], [5]. In contrast, sampling and evolutionary-based strategies, have been used for adaptive frameworks, and have yield high performance. For instance, [6], [3] base their strategies in modified versions of standard sampling-based strategies [7], considering the information gain. In particular, *Hollinger et.al* [6] presented a rapidly-exploring IG tree (RIG-tree), that was the base of a further developed two-step planning process presented by *Viseras et.al* [3]. They proposed a solution based on *Rapidly-exploring Random Trees* (RRT) and RRT* [8] to find a goal position providing a high information path under a budget constraint; using a GP for environment modeling, and DE as information function. Moreover, evolutionary-based strategies solve the IPP problem by means of a path parametrization and optimization. Hitz and Popovic *et al* [9], [10] propose the use of a GP to model the environmental variable and a CMA-ES optimizer to optimize the path to be followed by the robot, with a fixed length.

This work is based on the GP modelling described in [11], yet it extends the latter by developing a novel: (1) IG framework, (2) *decision-time adaptive replanning* (DAR) behavior, and a, (3) *depth-first Monte Carlo tree search* (DF-MCTS) strategy for IPP.

The IG framework coordinates the parallel execution of the data processing, map estimation and replanning modules. A data processing module that integrates an encoder-decoder based *convolutional neural network* (CNN) for image segmentation to discriminate the seagrass from the background, and a map estimation module build on a sparse GP (SGP) that generates a predictive model of the environment (in this case, an stocastic predictive model of seagrass distribution). The replanning module has been designed using a novel *decision-time adaptive replanning* (DAR) behavior for adaptive mis-

*This work is partially supported by the Spanish Ministry of Economy and Competitiveness under contract TIN2014-58662-R (AEI, FEDER, UE) and DPI2017-86372-C3-3-R (AEI, FEDER, UE) and by the Government of the Balearic Islands through grant FPI/2031/2017 (CAIB)

[1]Departament de Matemàtiques i Informàtica, Universitat de les Illes Balears, Ctra. Valldemossa Km 7.5, 07122 Palma, Spain (`e.guerrero@uib.eu`, `francisco.bonin@uib.eu`, `g.oliver@uib.eu`)

sion replanning, integrating a new IPP strategy that consists in a *depth-first* (DF) version of the *Monte Carlo tree search* (MCTS). The DF-MCTS is a reinforcement learning (RL) strategy guided by: 1) a value function that considers the sampling informativeness and the recorded data density, and, 2) by a function that rewards smooth trajectories. Finally, the proposed framework is validated in field tests to prove the effectiveness of the method for autonomous exploration of underwater environments covered with *P. oceanica*.

The remainder of the paper is organized as follows: Section II introduces some background about GP models and RL algorithms, Section III describes the IG framework, and Sections IV and V describe the developed DAR and DF-MCTS algorithms, respectively. Section VI describes the tests performed in simulation and in field and ilustrate the results. Finally, Section **??** summarizes the conclusions.

II. BACKGROUND

In this section we define the basis of the map estimation using Gaussian processes, and the basis of the IPP using reinforcement learning algorithms.

A. Gaussian processes (GP)

A GP is defined as a collection of random variables which have a joint Gaussian distribution [12]. The random variables represent the value of the non-observable function value $f(\mathbf{x})$ at location $\mathbf{x}$. Such function is specified by its mean function $m(\mathbf{x})$ and covariance function $k(\mathbf{x}, \mathbf{x}')$ such that,

$$f(\mathbf{x}) \sim \mathcal{GP}\Big(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')\Big) \tag{1}$$

Assuming a linear regression model, $f(\mathbf{x}) = \phi(\mathbf{x})^\top \mathbf{w}$, where $\phi(\mathbf{x})$ represents a basis function vector, or kernel, and the weights $\mathbf{w}$ follow a zero mean normal distribution $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma_p})$. Since $\mathbb{E}[\mathbf{w}] = 0$, $f(\mathbf{x})$ results in a zero mean:

$$\begin{aligned} m(\mathbf{x}) &= \mathbb{E}[f(\mathbf{x})] \\ &= \phi(\mathbf{x})^\top \mathbb{E}[\mathbf{w}] = 0 \end{aligned} \tag{2}$$

In addition, the covariance of $f(\mathbf{x})$ is derived from using the zero mean weights assumption such that:

$$\begin{aligned} k(\mathbf{x}, \mathbf{x}') &= \mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x}))\,(f(\mathbf{x}') - m(\mathbf{x}'))] \\ &= \mathbb{E}[f(\mathbf{x})\, f(\mathbf{x}')] \\ &= \phi(\mathbf{x})^\top \mathbb{E}[\mathbf{w}\mathbf{w}^\top]\phi(\mathbf{x}') \\ &= \phi(\mathbf{x})^\top \mathbf{\Sigma_p} \phi(\mathbf{x}') \end{aligned} \tag{3}$$

It results in that the GP covariance directly depends on the basis function vector, the query locations and the variance in the weights.

B. Reinforcement learning (RL)

The basic idea behind RL is to learn from interaction. The learning is based on trial and error; in real or simulated environments. In this work we will use RL in order to explore the space of possible paths to be executed by a mobile robot, by interacting with the environment model obtained from a GP. The selection of the best path to follow is considered a sequential decision process.

1) Markov decision processes (MDP): MDPs are a classical representation of sequential decision processes [13], and are characterized by the Markov assumption: the decisions taken only depend on the current state. Moreover, such processes are called Finite MDP (FMDP) when the states and actions are finite, and Partially Observable Finite MDP (POFMDP) when the state can not be completely observed. An MDP is characterized by four main sub-elements; a policy, the reward signal, the value function and, optionally, a model of the environment:

- The *policy* determines the action to be taken from a particular state.
- The *reward signal* comprises the goal of the RL problem, defining the good or bad events to the agent in an immediate sense.
- The *value function* estimates how good is for the agent being in a given state. It considers the total amount of reward that the agent can expect to accumulate over the future. Whilst rewards are given directly from the environment, values take into account the future rewards of the states that are likely to follow the actual state.
- The *model* describes the behavior of the environment. It predicts the transitions between states given an action; given an state and an action, it predicts the next state and the reward.

2) State-value function: We represent the IPP as a POFMDP, in which the sets of states, actions and rewards ($\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}$) have a finite number of elements, and where the state can not be fully observed. The *state-value function* is defined as,

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_\pi(s')\Big], \tag{4}$$

, for all $s, s' \in \mathcal{S}$ and $r \in \mathcal{R}$, $a \in \mathcal{A}(s)$; where the policy $\pi(a|s)$ provides the probability of choosing a particular action a from a state s. This is the expected return on the long run. Where $\gamma \in [0, 1]$ is the *discount rate*. With a $\gamma = 0$ the agent would be myopic, considering only immediate rewards, whereas with a higher γ values, the agent would have a stronger consideration on further rewards.

3) Bellman equation: The Bellman equation II-B.3 is used to find the optimal policy π^* that provides an optimal value function $v_*(s) \doteq \max_\pi (v_\pi(s))$, provides an iterative expression to find an optimal solution to the RL problem as $k \to \infty$. This equation can be solved with either *Dynamic programming (DP)*, *Monte-Carlo (MC)* or *Temporal difference (TD)* methods. DP can be used to incrementally compute optimal policies when a perfect model of the environment is available. However, they require a fully observable environment. The value update for MC and TD methods is performed using the expressions 6 and 7, respectively, where the λ parameter defines the *learning rate*. Whilst MC learns after trial, using the expected reward value G_t, TD learns from guesses, using the discounted value of the next step $v_k(s')$, it *bootstraps*

$$v_{k+1}(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma v_k(s')\Big] \quad (5)$$

$$MC : v_{k+1}(s) \doteq v_k(s) + \lambda\Big(G_t - v_k(s)\Big) \quad (6)$$

$$TD : v_{k+1}(s) \doteq v_k(s) + \lambda\Big(r + \gamma v_k(s') - v_k(s)\Big) \quad (7)$$

4) Decision-time planning: Decision-time planning methods are used to plan a series of decisions from a root state. These methods use simulated experience from a model to improve a policy or a value function. Instead of solving the whole MDP, they focus on solving a sub-MDP. They plan using the MDP model to look ahead from the current state. Rollout algorithms are a type of decision-time planning methods, based on Monte Carlo control. They sample multiple trajectories in a depth-first fashion from a root state. Each trajectory (or rollout) consists in taking successive decisions according to a given default policy until a terminal state is reached. They are used to obtain near-optimal decisions by taking random samples in a decision space and building a search tree according to the results. They are useful for AI applications with large states and action spaces. Moreover, this kind of methods are a good fit for our applications since they are *anytime*, providing always a solution where more computing power leads to a better performance.

MCTS is a particular rollout method, improved to bias the growing of a decision tree towards highest valued regions. A tree is started from a root node, and grows iteratively following a *Selection*, *Expansion*, *Simulation* and *Backpropagation* steps. Once the search is interrupted, an action of the root node is selected according to some predefined criteria. For instance select the action that conduces to the highest valued child or the most visited child.

III. IG FRAMEWORK

In this section we introduce the framework used for visual information gathering in benthic underwater environments. It comprises the execution in parallel of four modules: (A) The *navigation* module executes a mission path and continuously publishes the mission status and the recorded data. (B) The *data processing* module continuously generate high level data from the robot sensors. (C) The *map estimation* module provides sequential GP models to represent the environment using the last processed data available. (D) The *planning* module uses the last environment model to generate an informative mission path with the proposed DAR strategy.

A. Navigation

Navigation includes the low level architecture of the AUV -i.e. localization, mission control and data gathering-. The proposed framework has been integrated with the ROS-based COLA2 architecture [14] [15] for AUVs control, and can be executed in any robot working with such open source software. In this work, the execution of the proposed framework is tested on the Turbot AUV, which is a $1.6m$ long torpedo-shaped AUV based on a Sparus II AUV [16]. This vehicle has three degrees of mobility (surge, heave and yaw).

1) Localization: The Turbot AUV integrates the *extended Kalman filter* (EKF) implementation discussed in [17]. Which consists on the fusion of data provided by an *inertial measurement unit* (IMU), a *Doppler velocity log* (DVL), an *ultra-short baseline* (USBL), *depth* measurements and GPS (if the AUV in on the surface) in a double EKF setup.

2) Mission control: Governs the thrusters setpoints in order to follow a desired mission path. According to the COLA2 control architecture, the mission paths can be composed by waypoints or section maneuvers. Our proposed method builds the mission paths using section maneuvers; defined by (i) an initial and final position (x,y,z) (ii) speed and (iii) tolerance . This manoeuver follows a *Line of Sight with Cross Tracking Error* (LOSCTE) strategy [18] to produce reference velocity commands.

3) Data gathering: Consists on recording stereo images from the seabed and computing their disparity image with the onboard stereo camera. Fig. 1 shows a sample of the images gathered together with the stereo disparity image obtained.

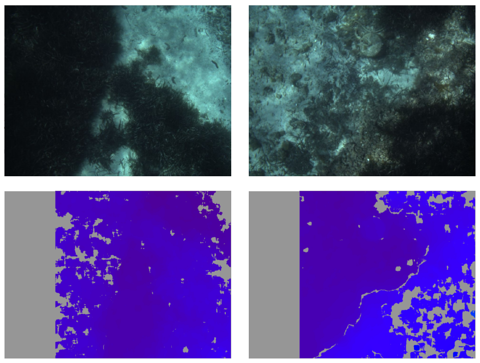

Fig. 1: Example of two images gathered during field tests together with their disparity image.

B. Data processing

The data processing consists in performing a semantic segmentation of the gathered images using a CNN, and a projection of their labeled pixels into global 3D coordinates as segmented point clouds, or *spatial distribution data* henceforth. The semantic segmentation uses the encoder-decoder based CNN arquitecture VGG16-FCN8. More specifically, it uses the pre-trained model described in [19], which is trained to discriminate the seagrass *P. oceanica* from the background, in underwater images. Moreover, as proposed by [11], the last two transposed convolutional layers are pruned to reduce the execution time at expenses of a lower output resolution. Fig. 2 shows some examples of gathered images together with their segmented image and segmented pointcloud. The segmented pointclouds are generated using the disparity image, and the transformation of such pointcloud to the global coordinates frame is done using the robot localization as described by [11].

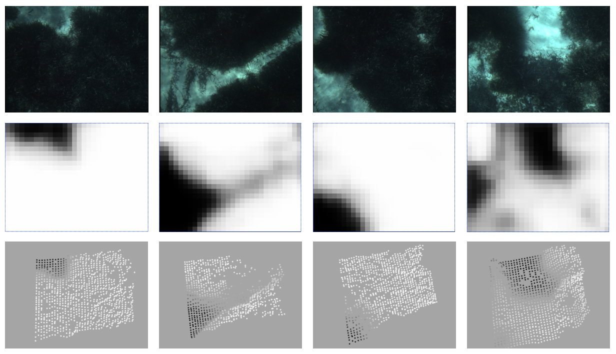
Fig. 2: Example of four images gathered during field tests together with their segmented image and segmented pointcloud.

C. Map estimation

The modeling of the seagrass spatial distribution involves two main processes: the *sampling* of the spatial distribution data, and the *learning* of the GP model hyperparameters.

1) Sampling: The objective of this process is: (i) collecting efficiently the processed data in a *raw data* dataset, (ii) generating a downsampled *samples* dataset to be used for the learning process, and (iii) computing the *raw data* density for a set of query locations. The first objective is attained by a thread that continuously filters the spatial distribution data; each segmented poincloud obtained during the data processing stage is downsampled using a voxel filter with a $0.1m$ resolution and appended to a *raw data* dataset. The second objective is attained by downsampling the *raw data* dataset using a grid filter with a given *samples resolution*. This process is executed under query and returns the *samples* dataset used for environment learning. The third objective is attained by building a *k-d tree* [20] using the *raw data* dataset and a minimum leaf size of $1.0m$. This latter step is also executed under query, and returns the raw data density in the query locations.

2) Learning: The objective of this process is to build a stocastic model of the environment needed to predict the seagrass spatial distribution in punctual locations. Such predictions will be further used by the IPP planner to compute the information richness of visiting a particular locations. A GP model configuration based in [11] and feed with the vehicle position, navigation heigh and the Posidonia data segmented in the previous stage is used to obtain this predictive model. In [11], Guerrero *et al* provide an extensive study for selecting the type of GP and its best configuration for modeling *P. oceanica* spatial distribution. Such GP model configuration consists in an *Sparse Variational GP using MCMC* (SGPMC) [21] model with a Matérn function with scaling factor $\nu = 3/2$, and a beta function to represent the GP likelihood distribution. Moreover, in this case for training the GP we use a fixed *kernel lenghtscale* $\lambda = 30$. This parameter controls the correlation strength between pairs of samples depending of their distance. Fixing this hyperparameter permits to control the distance foresight of the GP model and balance the exploration-explotation tradeoff: increasing λ will result in an increased exploitation of predicted areas while decreasing λ will result in an increased exploration, or increased data coverage.

An important issue for learning the environment model is fixing the R and the induction points density IPD desired. The higher the R and IPD the more accurate will be the model. However, it will affect in longer mapping times. In order to set a bound on the mapping time, we propose to set the maximum number of samples $N_{samples}$ and a maximum number of induction points N_{IP} used for training the GP model, and compute such R and IPD using the area A of the target region as $R = (\frac{A}{N_{samples}})^{\frac{1}{2}}$ and $IPD = \frac{N_{IP}}{A}$

D. Planning

The planning module proposed uses the developed DAR method described in the next section, IV. This module retrieves the robot localization and current goal, retrieves the most recent GP model and queries the robot navigation to execute a planned path.

IV. DECISION-TIME ADAPTIVE REPLANNING (DAR)

The proposed method consists in the following four key ideas:

- Building and updating of a network of nodes $\mathcal{N}$ that results in an pre-initialization of part of the content included in the states and actions considered during decision-time planning.
- The robot is neither stopped for planning nor obliged to complete the commanded mission paths. This permits having the vehicle in constant motion, while being flexible to execute updated mission paths.
- Selecting an starting point for the next computed path considering distance to be covered during the planning time.
- Growing a search tree in a depth-first fashion following a novel DF-MCTS strategy for decision-time planning.
- Recycling part of the last search tree for successive planning executions.

A. Algorithm

The main structure of the method is depicted in Algorithm 1. It inputs a target area $\mathcal{A}$, an obstacle region $\mathcal{O}$, a GP model of the environment $\mathcal{M}$, a initial state s_0, the sampling nodes density ρ, and the nearest neighbor distances d_1 and d_2. And both, the path to execute $\mathcal{P}$ and the remaining path $\mathcal{P}_{s_0}$ are initialized empty. It starts generating a set of sampling nodes $\mathcal{N}$, and then, the algorithm iteratively: (1) updates the nodes utility using the newest environment model $\mathcal{M}$, (2) gets the next initial state s_0 if a path $\mathcal{P}$ has been previously generated, (3) gets a path $\mathcal{P}_{s_0,s_n}$ using the DF-MCTS strategy, and (4) saves the path. As the robot finishes a section maneuver, it executes the newest path proposed from the search tree.

Algorithm 1: DAR()

```
Input: $\mathcal{A}$, $\mathcal{O}$, $\mathcal{M}$, $s_0$, $\rho$, $d_1$, $d_2$;
1  $\mathcal{P} \leftarrow \varnothing; \mathcal{P}_{s_0} \leftarrow \varnothing$;
2  $\mathcal{N} \leftarrow buildNodes(\mathcal{A}, \mathcal{O}, d, d_1, d_2)$
3  while $\neg stopCondition()$ do
4      $\mathcal{N} \leftarrow updateNodes(\mathcal{N}, \mathcal{M})$
5      if $\mathcal{P} \neq \varnothing$ then
6          $s_0 \leftarrow getNextInitState(\mathcal{P})$
7      end
8      $\mathcal{P}_{s_0,s_n} \leftarrow getPath(s_0)$
9      $\mathcal{P}_{s_0} \leftarrow getRemainingPath(\mathcal{P}, s_0)$
10     $\mathcal{P} \leftarrow \mathcal{P}_{s_0} + \mathcal{P}_{s_0,s_n}$
11     $savePath(\mathcal{P})$
12 end
```

B. Node network

To build the node network, we first generate a random set of nodes $\mathcal{N}$ inside of a given target area $\mathcal{A}$ and outside of given obstacle areas $\mathcal{O}$. The number of nodes n to build is determined by a desired node density ρ and the area of $\mathcal{A}$. Then, we build a *k-d tree* using the locations of the nodes contained in $\mathcal{N}$ for quick nearest-neighbor lookup. The node network is build by quering the *k-d tree* for two sets of neighbors for each node; a first set of nodes $\mathcal{N}_1$ located at a distance $d < d_1$, and a second set of nodes $\mathcal{N}_2$ located at a distance $d > d_1$ and $d < d_2$. Being the distance thresholds $d_1 < d_2$.

The objective is updating the sampling utility at the $\mathcal{N}$ locations. The model $\mathcal{M}$ is queried to get the data density and GP prediction in the $\mathcal{N}$ locations. Then, computes the information gain and utility from the prediction values. The information gain I is computed from the prediction obtained with the GP model at the node location. We have considered two options to compute such information, using either the differential entropy (DE) I_{DE} 8 or the upper confidence bound (UCB) I_{UCB} 9functions.

$$I_{DE} = \frac{1}{2} \ln(2\pi e \sigma^2) \tag{8}$$

$$I_{UCB} = \mu + 1.96\sqrt{\sigma^2} \tag{9}$$

Whilst I_DE uses the variance σ^2 provided by the GP, I_{UCB} also uses the mean semantic label μ. I_{UCB} provides increased information values to higher mean label locations, which increases exploitation (coverage) in such locations at expenses of reducing exploration in lower mean label locations. Moreover, instead of using the information gain directly for planning we propose the use of the sampling utility. We propose to leverage the information gain I with the neighboring data density D such as:

$$U = I'(1 - D')^{\alpha} \tag{10}$$

in order to attenuate the utility of areas that do not present a reduction on the predicted information I, even though they have been repeatedly recorded (high D). The objective is to avoid getting trapped in local minima, high information areas that do not reduce the variance of the GP model, even if they are repeatedly recorded. This situation may happen in areas with heterogeneous data, such as meadow boundaries. Moreover, since the resolution of the GP model is bounded by the *samples resolution* value and the *induction points density* of Section III-C, recording more data on a high information location does not implies improving the fitness of the GP model. I' and D' represent the normalized information and density values for all the nodes to the range $[0, 1]$ using a *min-max* normalization.The α parameter works as a weight on the density factor, the higher the α the bigger is the impact of the density D.

C. Next initial state

In order to allow a continuous navigation and a flexible path execution, we take into account the time expend in planning in order to set the initial state used for next planning iteration. The distance between the actual position and the next initial state s_0 has to be larger than a minimum distance $d_{min} = v_{max} \cdot t_{plan}$, where v_{max} is the maximum speed of the robot and t_{plan} is the planning time. The next initial state s_0 used for next planning is selected taking into account the distance covered by the robot while the planning process is executed.

1) Reuse tree: For successive IPP executions we use a part of the last computed search tree $\mathcal{T}$. More precisely, we keep the tree structure that hangs from the next initial state s_0, from now on called $\mathcal{T}'$. In order to do so, $\mathcal{T}'$ is traversed to update the distance cost, extend the leave states and update the state values. All these, prior to execute the IPP.

In order to generalize to different target areas, we propose to set by default the distance budget $\mathcal{B}$ to the half of the target area perimeter, and the neighbor distance $\mathcal{N}_2$ to a tenth of $\mathcal{B}$.

V. DEPTH-FIRST MCTS (DF-MCTS)

Considering the high number of possible states and actions available in our field robotics application and the necessity of an online realization, a decision-time planning method, formalized as POFMDP, has been developed to solve the IPP problem. DF-MCTS is a decision-time planning method different from MCTS in one key aspect: it keeps all the states traversed during the rollout in the search tree. This provides a faster growth of the tree, and guarantees a solution path of a given length. The limitation of MCTS for our application is that whereas MCTS would explore the environment in a depth first manner, the decision tree grows exhaustively resulting in a shallow tree if a short planning time is given or a high discount factor is used.

We propose to solve the IPP problem using a RL-based algorithm using a finite number of non-fully observable states. For that end, a POFMDP configuration represents the sequential decision problem. A *state* S_k is defined by an associated *node* N, a parent state S_{k-1}, a set of action candidates $\mathcal{A}(S_k)$, a state value V, a distance cost c from init state, and an orientation θ. The nodes are used to provide a discrete representation of the possible sampling locations in the target area. A *node* N is defined by a *north-east* position, a neighboring data density D, an information gain I, a sampling utility U, and two sets of neighbor nodes $\mathcal{N}_1^N$ and $\mathcal{N}_2^N$. The actions are directly associated to the transition to a given neighbor node. So, the selection of an action directly means the selection of a neighbor node to be visited next. Taking a particular action from a given state S_k will result on the transition to the neighbor node considered in such action, and in the creation of a new state S_{k+1}.

The structure of the proposed method is represented in Fig. 3. Starting from a root state s_0, it iteratively selects a high valued node, expands the tree with multiple rollouts, and backpropagates the state-values.

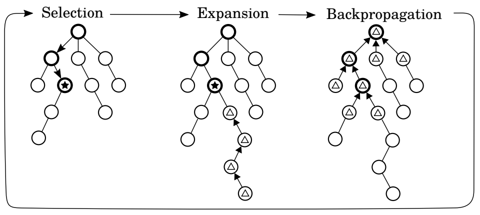

Fig. 3: One iteration of the proposed tree search algorithm. The stared state represents the selected state for expansion, the states with a triangle correspond to the states used for update the tree values for the expansion and the backpropagation steps. The darker circles represent the states whose values are updated.

A. Selection

First, select a state from the tree $\mathcal{T}_{ne}$ to expand, where $\mathcal{T}_{ne}$ includes the non-exhausted states of $\mathcal{T}$–i.e. state containing non-tested actions–. For the selection we follow a *tree policy* based in an $\epsilon - greedy$ method; get a random number r, if it is bigger than ϵ (given by configuration) get the highest valued state s^* from $\mathcal{T}$, otherwise get a random state. Being,

$$s^* = \operatorname*{argmax}_{s \in \mathcal{T}} v(s) \tag{11}$$

B. Expansion

Second, expand the tree following a given *default policy*. This step is different from the expansion step in MCTS algorithms in that DF-MCTS: (a) performs multiple MC rollouts from the same selected state, and (b) the states traversed are kept in the tree and are considered for further tree expansions.

1) MC rollout: Is build by iteratively: (a) search for candidate actions $\mathcal{A}(s)$ to the current state s, (b) select an action $a \in \mathcal{A}(s)$ according to a given *default policy* $\pi(a|s)$, (c) perform action a and (d) get next state s' and reward r, until a distance budget $\mathcal{B}$ is exhausted. We define the default policy $\pi(a|s)$ with an uniform distribution, and the deterministic action-state transitions $p(s'|s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s)$. The execution of action a being in state s results in a new state s' located on a neighboring node position pointed by action a.

2) Action candidates: The set of action candidates $\mathcal{A}(s)$ of a given state s is built when a state is visited for the first time. It includes visiting the nodes that can be reached in one step ahead. We build the set of actions of a given state $\mathcal{A}(s)$ checking the nodes network built in algorithm DAR initialization. In order to get smoother rollout trajectories we propose the use of two sets of action candidates: $\mathcal{A}_1$ and $\mathcal{A}_2$ represented in Fig. 4. The set $\mathcal{A}_1(s)$ includes the priority actions. The default policy $\pi(a|s)$ will use the set $\mathcal{A}_1(s)$ if it is not empty, otherwise it will use the set $\mathcal{A}_2(s)$. Besides, afterwards an action candidate a is used, it is removed from its action candidate set. As a result from this as the tree grows some states get exhausted of action candidates and are removed form the non-exhausted states tree $\mathcal{T}_{ne}$. Such states will not be longer available for selection.

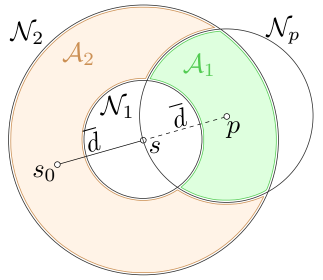

Fig. 4: Two sets of action candidates for a given state s: $\mathcal{A}_1 = \mathcal{N}_2 \cap \mathcal{N}_p$ and $\mathcal{A}_2 = \mathcal{N}_2 - \mathcal{N}_p$. Where $\mathcal{N}_p$ includes the nodes located at a distance d from p and $\overline{s\ p} = \overline{s_{parent}\ s} = \overline{d}$

3) State-value update: The values $v(s)$ of the states traversed during rollout are updated following a MC criterion, using 12, where $r(s)$ represents the reward generated in state s.

$$v(s) = r(s) + \gamma v(s') \tag{12}$$

4) Reward function: In order to generate smooth trajectories we also consider the relative turns between consecutive states in the reward function. The reward function $r(s)$ is defined in , where $\theta_r(s)$ is the relative angle between the orientations of s and its parent state. Since $u \in [0, 1]$ (Section IV-B) the reward will be unitary $r \in [0, 1]$. Computing the value of the rollouts is a mater of computing sequentially the discounted sum of rewards.

$$r(s) = u(s) \cdot \cos^w \frac{\theta_r(s)}{2} \tag{13}$$

C. Backpropagation

Finally, we back-propagate the rollout values upwards. We use for this a value function based on the TD approach of Equation 7 in Section II-B.2, the difference is that in this case we use the mean value of all child states $s_c \in \mathcal{S}_c(s)$ for the value update.

$$v_{k+1}(s) = v_k(s) + \lambda \left[r + \gamma \left\| \sum_{s_c \in \mathcal{S}_c} v_k(s_c) \right\| - v_k(s) \right] \tag{14}$$

VI. EXPERIMENTS

Here we present results obtained after field testing. The experiments have been carried out in Mallorca Island, in a shallow water region with maximum depth of 3.5m to be able to compare the online estimated maps with the existing aerial images of the area and assess the performance of such mappings. The objective of the test was to get a fast representation of the target area represented in Fig. 5, that has an extension of $13.558m^2$. And the default configurations used for navigation, map estimation and IPP are given given in the Table I.

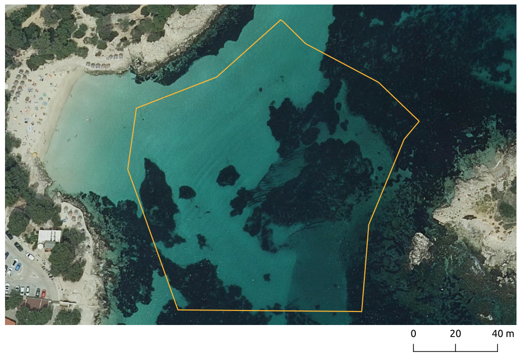

Fig. 5: Target area for field testing ($13.558m^2$) on top of an aerial image from the *Instituto Geodesico Nacional* (PNOA 2018 campaign $39.534372, 2.590594$).

TABLE I: Default configurations for the LOSCTE, GP model and DF-MCTS

LOSCTE	**Value**
Maximum speed $[m/s]$	0.5
Minimum speed $[m/s]$	0.05
Lookahead distance $[m]$	4.0
Speed transition distance $[m]$	3.0
Maximum acceleration module $[m/s^2]$	0.1
GP	
Model	SGPMC
Kernel	Matern 32
Kernel scale	30
Likelihood	Beta
Likelihood scale	0.5
Optimizer	Scipy
Iterations	2500
Max number of samples	2000
Max number of inducing points	200
DF-MCTS	
Planning time $[s]$	15
Information function	DE
Alpha (Utility)	1.0
Beta (Reward)	1.0
Discount factor	0.8
Learning rate	0.9
Rollouts number	32
Epsilon	0.01
Number of nodes	20000
Neighbor distance d_1	2.0

The mean vehicle speed in the advancing direction resulted in $0.51m/s$. And the image segmentation achieved a frequency of $0.461Hz$. Which provided sufficient overlap between successive segmented pointclouds.

Fig. 6 illustrates the resulting mapping time and coverage percent. It shows a clear correlation between coverage and mapping time, which seems to converge to $40s$ for a 65% of coverage. A good consequence of this behavior is that at the beginning of the exploration the mapping frequency is higher, allowing fast adaptation of the IPP planner to the changing map estimation. Since at the last part of the exploration the map is partially known, and the high information areas are already located, a low frequency can be tolerated.

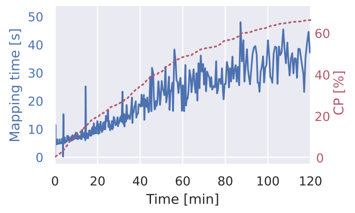

Fig. 6: Resulting mapping time and coverage percent obtained during an active exploration execution in field.

Moreover, Fig. 7a show the resulting vehicle navigation and the density of the recorded data on top of a groundtruth image hand-labeled from an aerial image; the higher density areas are located on heterogeneous data regions. Looking at Fig. 7b notice here the distance between the prediction (purple shaded area) and the groundtruth (black line).

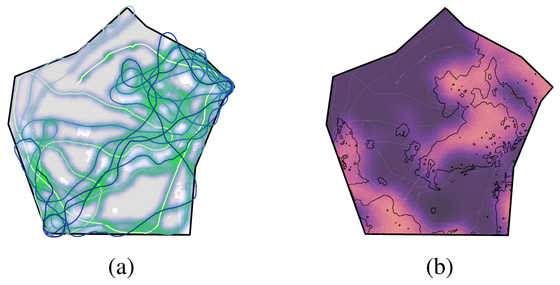

Fig. 7: (a) Path followed and raw data density obtained during field testing. The color in the path represents time, starts in yellow and ends in dark blue. The raw data density is represented with the green shade. (b) Mean prediction value with groundtruth contour on top (black) and followed path (white).

Fig. 8 show respectively the *Mean of the Differential Entropy (MDE)* and *Standard deviation of the Differential Entropy (SDE)* of the online estimated maps along the path.

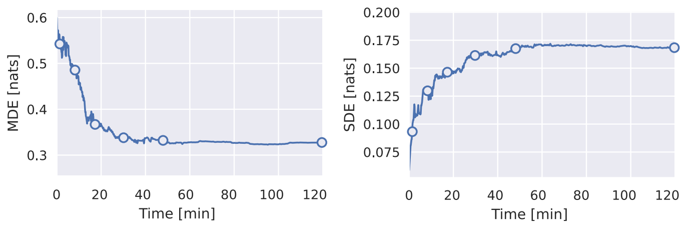

Fig. 8: Results obtained in field test. The circles on the plot represent the six time instants ilustrated in the Fig. 9.

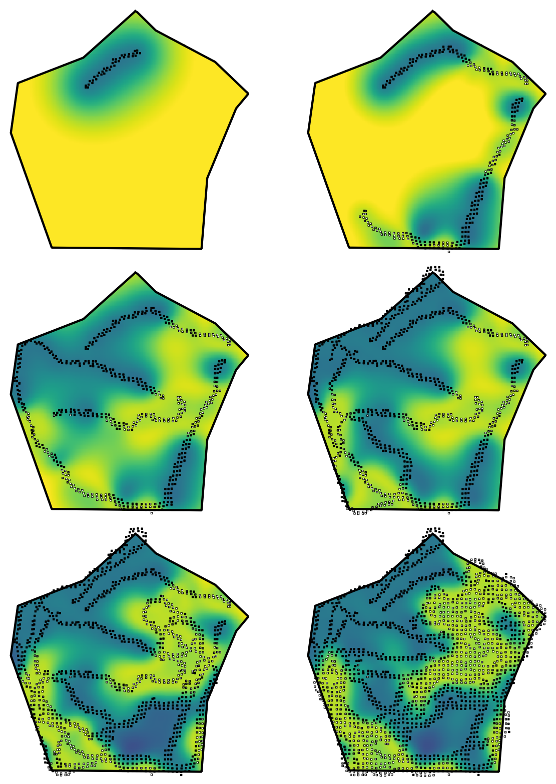
Fig. 9: Representation of 6 different instants during the execution of the DAR algorithm test A5 and A6. The squares represent the spatial distribution samples used to train the GP and the colormap represent the predicted variance map; blue for low variance regions and yellow for high variance regions. From left to right and top to bottom the representations belong to the instants; 1' 36", 8' 33", 17' 27", 29' 49", 48' 5", and 120'.

The MDE provides the mean map information, and is computed using the variance value of the environment model prediction V. The SDE provides the standard deviation of the pixelwise map information, also computed using the variance value of the environment model prediction V. Both metrics converge around the minute $50'$. The circle marks represented in this figure point the instant of the representations of Fig. 9. Such figure shows a set representations of the gathered data and the uncertainty map (GP predicted variance) in six time instants during the active exploration process.

These results are considered to balance the exploration/exploitation trade-off. The vehicle tends to cover the entire area during the first half of the mission, and step by step, it starts focusing in the regions with high variance. As the estimated map represents better the environment (minute $29'49$" in Fig. 8 and fourth step in Fig. 9) the planned paths focus in heterogeneous data regions.

VII. CONCLUSIONS

The proposed active exploration framework has shown promising results on field experiments. It can be executed online and provide high informative paths. Whilst it provide a fast reduction of the estimated map entropy, further experiments and comparisons with other methods would provide a more solid proof of the proper performance.

REFERENCES

[1] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005.

[2] G. A. Hollinger, A. A. Pereira, and G. S. Sukhatme, “Learning uncertainty models for reliable operation of Autonomous Underwater Vehicles,” in *2013 IEEE International Conference on Robotics and Automation*. IEEE, may 2013, pp. 5593–5599.

[3] A. Viseras, D. Shutin, and L. Merino, “Robotic Active Information Gathering for Spatial Field Reconstruction with Rapidly-Exploring Random Trees and Online Learning of Gaussian Processes,” *Sensors*, vol. 19, no. 5, p. 1016, feb 2019.

[4] J. Binney and G. S. Sukhatme, “Branch and bound for informative path planning,” *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 2147–2154, 2012.

[5] K. C. Ma, L. Liu, H. K. Heidarsson, and G. S. Sukhatme, “Data-driven learning and planning for environmental sampling,” *Journal of Field Robotics*, vol. 35, no. 5, pp. 643–661, 2018.

[6] G. A. Hollinger and G. S. Sukhatme, “Sampling-based robotic information gathering algorithms,” *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1271–1287, aug 2014.

[7] S. M. LaValle, *Planning Algorithms*. Cambridge: Cambridge University Press, nov 2006, vol. s1-9, no. 48.

[8] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, jun 2011.

[9] G. Hitz, E. Galceran, M.-È. Garneau, F. Pomerleau, and R. Siegwart, “Adaptive continuous-space informative path planning for online environmental monitoring,” *Journal of Field Robotics*, vol. 34, no. 8, pp. 1427–1449, dec 2017.

[10] M. Popović, T. Vidal-Calleja, G. Hitz, J. J. Chung, I. Sa, R. Siegwart, and J. Nieto, “An informative path planning framework for UAV-based terrain monitoring,” *Autonomous Robots*, vol. 44, no. 6, pp. 889–911, 2020.

[11] E. Guerrero-Font, F. Bonin-Font, M. Martin-Abadal, Y. Gonzalez-Cid, and G. Oliver-Codina, “Sparse Gaussian process for online seagrass semantic mapping,” *Expert Systems with Applications*, vol. 170, p. 114478, may 2021.

[12] M. SEEGER, “GAUSSIAN PROCESSES FOR MACHINE LEARNING,” *International Journal of Neural Systems*, vol. 14, no. 02, pp. 69–106, apr 2004.

[13] R. S. Sutton and A. G. Barto, *Reinforcement Leaning*, 2018.

[14] I. Robotics, “Cola2 software architecture,” https://bitbucket.org/iquarobotics/cola2_wiki, 2020.

[15] N. Palomeras, A. El-Fakdi, M. Carreras, P. Ridao, and A. C. M. R. P. Palomeras N.; El-Fakdi, “COLA2: A Control Architecture for AUVs,” *IEEE Journal of Oceanic Engineering*, vol. 37, no. 4, pp. 695–716, 2012.

[16] M. Carreras, C. Candela, D. Ribas, N. Palomeras, L. Magí, A. Mallios, E. Vidal, È. Vidal, and P. Ridao, “Testing Sparus II AUV, an open platform for industrial, scientific and academic applications,” *Instrumentation Viewpoint*, no. 18, pp. 54–55, 2015.

[17] E. Guerrero-Font, F. Bonin-Font, P.-L. Negre-Carrasco, M. Massot-Campos, and G. Oliver-Codina, “USBL Integration and Assessment in a Multisensor Navigation Approach for AUVs,” *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 7905–7910, jul 2017.

[18] T. I. Fossen and K. Y. Pettersen, “On uniform semiglobal exponential stability (USGES) of proportional line-of-sight guidance laws,” *Automatica*, vol. 50, no. 11, pp. 2912–2917, nov 2014.

[19] M. Martin-Abadal, E. Guerrero-Font, F. Bonin-Font, and Y. Gonzalez-Cid, “Deep Semantic Segmentation in an AUV for Online Posidonia Oceanica Meadows Identification,” *IEEE Access*, vol. 6, 2018.

[20] J. L. Bentley, “Multidimensional Binary Search Trees Used for Associative Searching,” *Communications of the ACM*, 1975.

[21] J. Hensman, A. G. d. G. Matthews, M. Filippone, and Z. Ghahramani, “MCMC for Variationally Sparse Gaussian Processes,” jun 2015.