**Persistent Autonomy through Learning, Adaptation, Observation and Replanning**

**DELIVERABLE 3.3**

**Multi-dimensional RL skill learning for AUV**

Distribution: Pandora Partners, Project Officer, Reviewers, PANDORA website

Supplementary notes:

<table>
<thead>
<tr>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>2015-07-31</th>
<th>Final Version</th>
<th>53</th>
<th>IIT</th>
<th>IIT</th>
<th>IIT</th>
<th>HWU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2014-09-02</td>
<td>Second Draft Issued</td>
<td>53</td>
<td>IIT</td>
<td>IIT</td>
<td>IIT</td>
<td>IIT</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2014-08-31</td>
<td>First Draft Issued</td>
<td>53</td>
<td>IIT</td>
<td>IIT</td>
<td>IIT</td>
<td>IIT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*This document is the property of the Pandora partnership. - Any unauthorised use of the same will be prosecuted*
# Contents

## Summary

### I Reinforcement Learning

1. **Introduction**
2. **Background**
   1. Multi-objective reinforcement learning
   2. Multi-dimensional reward vector in RL
3. **Multi-objective reinforcement learning**
   1. Methodology
      1. AUV Model
      2. Fault detection Module
      3. Policy representation
      4. Vectorized Reward Function
      5. Multi-Objective Optimization Problem
      6. Multi-objective Optimization Algorithms
   2. Experimental Setup
   3. Results
      1. A Fully Broken Thruster
      2. A Partially Broken Thruster
      3. Improving the Performance
   4. Conclusion

4. **Multi-dimensional reward vector in RL**
   1. The testing platform
   2. Methodology
      1. The optimization algorithm
      2. The policy representation
      3. The AUV model
      4. The reward vector
   3. Results
   4. Conclusion

## II Reactive Learning

1. **Introduction**
2. **Methodology**
3. **Imitation Learning**
4. **Force Control Strategy**
   1. The Force and Motion Controller
      1. Valve Turning with Stationary Base
      2. Valve Turning with Moving Base
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Learning of Reactive Behavior</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5.1 Design of the Fuzzy System</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>5.2 Tuning the Fuzzy System</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>5.3 Experimental Result of the Reactive System</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>5.3.1 Lab Experiments</td>
<td>34</td>
</tr>
<tr>
<td></td>
<td>5.3.2 Underwater Experiments</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>Conclusions</td>
<td>35</td>
</tr>
<tr>
<td>1</td>
<td>Reinforcement Learning</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>1.1 Multi-objective RL</td>
<td>46</td>
</tr>
<tr>
<td></td>
<td>1.2 Multi-dimensional reward vector in RL</td>
<td>46</td>
</tr>
<tr>
<td>2</td>
<td>Reactive Learning</td>
<td>46</td>
</tr>
</tbody>
</table>
Summary

This deliverable describes work conducted as part of the PANDORA project. It is divided into two parts: multi-dimensional reinforcement learning (Part I) and reactive learning (Part II). The multi-dimensional reinforcement learning is further divided into two parts: multi-objective reinforcement learning and multi-dimensional reward vector in reinforcement learning.

In the reinforcement learning (RL) part we investigated, designed and developed multi-dimensional techniques that helps in learning control policies for an autonomous underwater vehicle (AUV) on the fly. In the reactive learning part we researched new methods and developed a controller that takes in to account uncertainties in the environment and reacts in time to avoid any damage to the system or the environment in which the robot is operating, for example, when the AUV is turning a valve, the proposed method reacts to disturbances in the environment to retract the arm to avoid damage to the valve and the AUV.

In Part I, we present research that investigates the advantage of a multi-dimensional reward vector over the conventional scalar rewards in RL. We show that the added information improves the action selection strategy, producing better policies. We apply the developed method to a scenario in which an AUV has to recover from a thruster failure. The goal of the AUV is to reach a desired location. At the point of failure, the robot uses the model of the AUV to learn a policy that uses the remaining assets of the AUV. We use a parameterized policy to control the AUV. The proposed method uses vector optimization using genetic algorithms. The optimization algorithm generates a pool of policies of an arbitrary size, n. In each iteration, a new generation of polices are generated using evolutionary mechanisms such as mating and mutation. The policies are executed, in simulation, using the model of the AUV to evaluate their fitness using a multi-dimensional reward vector, then the top n polices are selected. The process is repeated until the solutions converge. Since the reward function consists of a vector, the optimizer maintains a Pareto optimal set. A Pareto optimal set is defined as the set of vectors in which it is impossible to make one objective better without making at least one other objective worse off. We use a reward vector that consists of the difference between final and desired x-coordinate and the y-coordinate. We also include the length of the trajectory. We tested the proposed method by running the task twenty times in simulation. We compare our results with the scalar reward approach. We show that in the worst case, we produce a policy that is as good as the scalar approach and in the best case, the multi-dimensional approach performs better than the scalar approach.

Furthermore, we investigated multi-objective reinforcement learning. We propose a model-based direct policy search for discovering fault-tolerant control policies for thruster failure recovery in AUVs. The approach learns a fault-tolerant policy on an on-board model of the vehicle and then executes the optimal policy on the real AUV. In this approach, when a thruster fails, the robot has three objectives: reach a desired location, with a desired velocity and a desired orientation. Since after the thruster failure we have an under-actuated robot, not all of the objectives can be attained. We extend our previously developed algorithms to learn multi-objective policies that utilizes the reconfigured model to discover a set of optimal solutions. Each optimal solution can be used to generate a trajectory that is able to navigate the AUV towards a specified target while satisfying multiple objectives. We show that our approach can deal with partially and totally broken thrusters.

In Part II, an approach is proposed for the challenging task of autonomous robotic valve turning in uncertain environments. The proposed system comprises two main phases: reaching and turning. In the reaching phase the manipulator learns to generate trajectories to reach or retract from the target. The learning is based on a set of trajectories demonstrated by the operator. The turning phase is accomplished using a hybrid force/motion control strategy. Furthermore, a reactive decision making system is devised to react to the uncertain behaviors of the system during the valve turning process. The reactive controller
monitors the changes in force, movement of the arm with respect to the valve, and changes in the distance to the target. Observing the uncertainties, the reactive system modulates the valve-turning task by changing the direction and rate of the movement. A real-world experiment with a robot manipulator mounted on a movable base is conducted to show the efficiency and validity of the proposed approach.
1 Introduction

In this report we investigate multi-dimensional reinforcement learning. We divide the problem into two categories: multi-objective reinforcement learning and multi-dimensional reward vector for reinforcement learning.

Multi-objective reinforcement learning is an optimization problem that learns a compromise between multiple, often conflicting, objectives to reach a Pareto optimal solution. A Pareto optimal solution is one in which it is impossible to make any one objective better without making at least one other objective worse off.

In multi-dimensional reward vector for reinforcement learning, however, the reward vector represents only one objective. By design, the vector is non-conflicting. Hence, the approach leverages the extra information to increase the convergence speed and improve the action selection strategy by allowing the algorithm to analyze the effect of different actions on the individual components of the reward vector. We develop techniques that allow reinforcement learning algorithms to use multi-dimensional reward vectors as opposed to conventional scalar rewards.

A combination of these two methods can give autonomous underwater vehicles a powerful tool to successfully trade-off between conflicting goals and learn a control policy fast.

2 Background

The background is divided into two parts, namely, multi-objective reinforcement learning and multidimensional reward vector.

2.1 Multi-objective reinforcement learning

In multi-objective reinforcement learning, often, the objectives are conflicting. There are a number of solutions proposed in the literature for multi-objective reinforcement learning. These include use of lexicographic ordering [1] of objectives, convex hull value iterative algorithm [2], linear scalarization function [3] and hypervolume-based multi-objective methods [4].

One approach to solving a multi-objective problem is to keep a Pareto optimum set, for example, in the reinforcement learning framework, we can keep a Pareto optimum set of actions that will result in positive reward for all objectives. During the exploration stage, when a new action is encountered, a decision has to be made whether it belongs: to the current Pareto set; or to the dominated region.

Moffaert et al. [4] propose a multi-objective reinforcement learning algorithm where they use hypervolume unary indicator to evaluate action selection. Hypervolume indicator calculates the volume of the area between a reference point and the Pareto set. It is the volume in which the objective value is dominated by at least one Pareto-optimal objective. The area or volume under the Pareto-optimal front is used as an indicator of dominance. The term ‘hypervolume indicator’ was coined by Zitzler et al. [5], while the idea was earlier introduced and employed by the same author [6]. Any objective with the value falling in this area is considered to be sub-optimal and rejected.

Standard reinforcement learning optimizes an implicit objective function, whereas, evolutionary methods optimize an explicit objective function. Hence, these approaches lend themselves well to multi-objective reinforcement learning [7]. Stochastic search algorithms, which includes evolutionary algorithms, consist of three parts: 1) a working mem-
ory that contains the currently considered solution candidates, 2) a selection mechanism—mating and environmental selection—, and 3) a variation mechanism—mutation—. Most stochastic algorithms have been designed for single objective optimization and only consider one solution at a time. Hence, there is no mating. In contrast, evolutionary algorithms can have multiple objectives and mating selection mechanism is applied. Two fundamental goals of multi-objective evolutionary algorithms are: guiding the search towards the Pareto-set, and keeping a diverse set of non-dominated solutions.

Different fitness assignments have been used to select a candidate. These include aggregation based, criterion based, and Pareto based. In the aggregation based method, the objectives are combined in a single parametrized objective function. The parameters of this function are systematically varied during the optimization run in order to find a set of non dominated non-dominated solutions instead of a single trade-off solution. For example, weighted-sum aggregation [8, 9]. Criterion based methods switch objectives during the selection based on a criterion, for example, adding individuals in the mating pool such that the proportion of the individuals per objective is equal [10]. The Pareto dominance method uses Pareto ranking, that is, counting the number of individuals dominated by an individual as its fitness [11]. The advantage of the Pareto method is that it considers the entire population.

It is important to maintain the diversity of the population. To this end, an individual’s chance of being selected is inversely proportional to the density of other individuals in its neighborhood. Silvervman et al. [12] discuss different methods, namely, Kernel method, nearest neighbor and histogram to maintain diversity. In kernel methods, for each individual a function that decreases in value as it goes away from the individual is calculated, which is called the kernel function. The density of an individual is calculated by summing the values of all individuals in the population. In histogram method, the search space is divided into bins, the density is estimated by the fraction of individuals in the bin. In the nearest neighbor technique the distance of an individual of interest to its k nearest neighbors is used to estimate the local density.

2.2 Multi-dimensional reward vector in RL

As described earlier, in multi-dimensional reward vector approach for reinforcement learning we investigate advantage of using a non-scalar reward vector. We envisage that the extra information provided to the reinforcement learning algorithm will help in increasing the rate of convergence. Moreover, it may help in alleviating the problem of local maxima/minima.

A reinforcement learning problem consists of:
- a state,
- actions that can be performed in a given state,
- a policy, which is a mapping from a state to an action
- a reward function – a mapping from an action to an immediate reward value,
- a value function – a mapping from an action to the expected reward if the policy is followed from that point on.

In classical reinforcement learning the reward function and the value functions output a scalar value. If there is more than one factor that contributes to the reward, the values of these factors are combined, for example, weighted sum of the reward due to each factor. In this section we introduce the idea of multidimensional reward vectors. For example if we want to get an AUV to reach a target position, the output of the reward function is the Cartesian distance between the AUV and the target position. Another approach will be to give the algorithm a reward vector consisting of the relative value of each coordinate.

The hypothesis is that by adding extra information, we can get the RL algorithm to converge to the solution faster. That is, using less trials. In robotics one less trial equates to less wear and tear in the robot. It is different from multi-objective optimization because the
factors considered belong to the same objective and hence there are not conflicts between the factors.

The ARCHER algorithm proposed by Kormushev et al. [13] makes an attempt in using a multidimensional reward value. The reward is the x and y coordinate of the distance from the target. However, the authors use the scalar distance value to order the solutions. In the reward space, they calculate weights for each component using chained regression, putting higher weights to rewards that are closer to the target. The weights calculated in the reward space is then used to calculate a new policy by combining the parameters of the corresponding polices according to the weights calculated in the reward space.
3  Multi-objective reinforcement learning

3.1  Methodology

As can be seen in Fig. 1 when a thruster is deemed faulty, the fault detection module sends a signal to the fault recovery module. This module’s task is to discover a fault-tolerance control policy using the remaining assets of the system. The discovered control policy can safely bring the AUV to a station where it can be rescued.

The proposed fault recovery module is framed in the context of model-based direct policy search for reinforcement learning. This framework comprises a dynamic model of the vehicle, a parameterized representation for the control policy, a reward function, and an optimization algorithm. The dynamics model of the system is reconfigured according to the current situation of the system. In the employed model-based policy search approach the trials are performed on the on-board dynamic model and not directly by the vehicle. For AUVs this is not a practical limitation, as their dynamics have been modeled accurately. The direct policy search utilizes a function approximation technique and an optimization heuristic to learn an optimal policy that can reach the goal specified by the reward function. The optimization heuristic can be treated as a black-box method because in policy search over a finite horizon, the particular path followed by the agent in the state-space can be ignored. In this section, all the components of the fault recovery module depicted in Fig. 1 are explained.

The rest of this section is organized as follows: First, the dynamics model of the AUV and the parameters of the model are described. Then, the fault detection module is explained. We show the policy representation used in our approach and define the vectorized reward function. The employed single-objective and multi-objective algorithms are explained in details. Finally, the conducted experiment and the results are illustrated.

Figure 1: The diagram shows the fault detection and fault recovery modules. The fault recovery module includes a number of elements such as policy representation, reward function and dynamics model of the system. (see Section.3.1 for more details).
3.1.1 AUV Model

In this section a dynamic model of the AUV is formed using a set of equations and a set of parameters. This model is used to find the optimal solutions that are executed on the real robot later.

The dynamics equations of a 6-DoF rigid body subject to external forces and torques while moving in a fluid environment can be generally formulated as

\[
\dot{\eta} = J(\eta)v
\]

\[
(M_{RB} + M_A)v + (C_{RB}(v) + C_A(v) + D(v))v + g(\eta) = B\tau
\]

where \(\eta \triangleq [x \ y \ z \ \phi \ \theta \ \psi]^T\) is the pose (position and orientation) vector with respect to the inertial frame and \(v \triangleq [u \ v \ w \ p \ q \ r]^T\) is the body velocity vector defined in the body-fixed frame. \(J(\eta)\) is the velocity transformation matrix, \(M_{RB}\) is the rigid body inertia matrix, \(M_A\) is the hydrodynamic added mass matrix, \(C_{RB}(v)\) is the rigid body Coriolis and centripetal matrix, \(C_A(v)\) is the added mass Coriolis and centripetal matrix, \(D(v)\) is the hydrodynamic damping matrix, \(g(\eta)\) is the hydrostatic restoring force vector, \(B\) is the actuator configuration matrix, and the vector \(\tau\) the control input vector or command vector.

The AUV used in our experiments is the Girona 500 AUV [14]. Girona 500 is a reconfigurable AUV equipped with typical navigation sensors (e.g. DVL), survey equipments (e.g. stereo camera) and various thruster layouts. The selected thruster layout in this work consists of five thrusters: 2 in heave direction, 2 in surge direction, and 1 in sway direction.

So far the dynamics equations of the system in Eq.1 are used to build the model of the AUV. In order to complete the dynamic model of the system, we use the hydrodynamic parameters of Girona 500, which have been extracted using an online identification method and reported in [15].

3.1.2 Fault detection Module

The process of monitoring a system in order to recognize the presence of a failure is called fault detection. We only consider the case of thruster failure that can take place due to thruster blocking, rotor failure, flooded thrusters, etc. In a real underwater vehicle sometimes the thruster may still work but not as a fully functional module. For instance, some sea plant may twist around the propeller of the thruster and reduce its efficiency by a percentage. In this research, we consider a generic case in which a thruster can be fully functional, partially broken or totally nonfunctional. Furthermore, since the process of detecting failure in AUVs and ROVs has been extensively studied [16, 17, 18, 19], investigating the fault detection process is not the main focus of this research. Therefore, we assume that the fault detection module is placed in a higher layer in the architecture of the system (see Fig. 1). This module continuously monitors the existing thrusters of the AUV and sends information about the coefficient of functionality (healthiness) of each thruster when a change in the behavior of the system is observed. The output of this module is a vector of functionality coefficients in range \([0, 1]\), where 0 indicates a totally nonfunctional thruster, 1 represents a fully functional thruster, and for instance, 0.7 indicates a thruster with 70% efficiency.

3.1.3 Policy representation

Reinforcement learning (RL) in continuous state-space requires function approximation. In this technique a parameterized representation of the final solution is formed. The goal of the RL algorithm is to find a set of parameters that lead to an optimal solution. In direct policy search, which is an alternative method for searching directly in the policy space, a policy representation, a reward function, and a stochastic optimization heuristic is used to achieve the goal.
Using linear function approximation, a policy can be represented as a weighted linear sum of a set of features (known as basis functions).

\[ \Pi = \sum_{i=1}^{n_f} \theta_i \phi_i = \Theta^T \Phi \]  

(2)

where \( \Pi \) is the policy, \( n_f \) is the number of features, \( \phi_i \in \Phi \) is the \( i^{th} \) feature, \( \Phi \) is the set of features, \( \theta_i \in \Theta \) is the \( i^{th} \) parameter, and \( \Theta \) is the parameter vector. The most common choices for basis functions include polynomial basis, radial basis, Proto-value, and Fourier basis schemes. In this work we use Fourier basis scheme because they are easy to compute accurately even for high orders, and their arguments are formed by multiplication and summation rather than exponentiation. In addition, the Fourier basis seems like a natural choice for value function approximation [20]. To deal with multiple variables in the representation of the policy the \( n^{th} \) order Fourier expansion of the multivariate function \( F(x) \) is used. Since a full Fourier expansion includes both sin and cos terms, the number of basis functions for the \( n^{th} \) order expansion with \( d \) variables is \( 2(n+1)^d \). This number can be reduced to \( (n+1)^d \) by dropping either of terms for each variable [20]. Thus the \( n^{th} \) order Fourier expansion of \( d \) variables can be formulated as

\[ \phi_i(x) = \cos (\pi \varsigma^i . x) \]  

(3)

where \( \varsigma^i = [1, ..., \varsigma^d], \varsigma_j \in [0, ..., n], 1 \leq j \leq d. \) For more details about the function approximation using Fourier basis refer to [20].

In this work the policy represents the control input vector \( \tau \) of the AUV which is a function of parameters \( \Theta \) and an observation vector \( \Omega \subset \eta \). The observation vector is the subset of state variables that we observe during the learning process (\( \Omega = [x y \psi u v] \)).

### 3.1.4 Vectorized Reward Function

The performance of the vehicle is measured through a reward function:

\[ R = \sum_{t=0}^{T} r_t(\eta_t) / \Pi \]  

(4)

where \( r_t \) is the immediate reward vector, and depends on the current state \( \eta_t \), which in turn is determined by the policy and its parameters. Therefore, the aim of the agent is to tune the policy’s parameters in order to maximize the cumulative reward vector \( R \) over a horizon \( T \). Many different definitions of the immediate reward are possible. We defined a vectorized reward function \( r_t \) including three reward components \( r_{1t}, r_{2t}, \) and \( r_{3t} \).

\[ r_t = [r_{1t} r_{2t} r_{3t}] \]

\[ r_{1t} = \frac{1}{\|P_t - P_d\| + \varepsilon} \]

\[ r_{2t} = \frac{1}{\|V_T - V_d\| + \varepsilon} \]

\[ r_{3t} = \frac{1}{|\psi_t - \psi_d| + \varepsilon} \]

(5)

where \( P_t \) and \( P_d \) are the current and the desired position vectors \( [x y] \) respectively. \( V_T \) and \( V_d \) are the current and the desired linear velocity vectors \( [u v] \) respectively. \( \psi_t \) and \( \psi_d \) are the current and the desired yaw angles respectively. The first reward component, \( r_{1t} \), is defined to navigate the AUV towards the target. The second component, \( r_{2t} \), ensures the AUV reaches the target with minimum final speed. And the third component, \( r_{3t} \), keeps the orientation of the AUV always heading forward. If the reward components do not conflict a single-objective algorithm can be applied to find a single optimal solution by scalarizing the reward function (i.e. \( r = \sum \omega r_t \)). Since a thruster is deemed faulty,
the reward components conflict. Multi-objective optimization techniques are used to solve problems with conflicting objectives. In the next section we describe the definition of a multi-objective problem.

### 3.1.5 Multi-Objective Optimization Problem

A multi-objective optimization problem can be formulated as

\[
\text{Minimize } \mathbf{F}(\mathbf{x}) = [f_1(\mathbf{x}), \ldots, f_m(\mathbf{x})]^T \\
\text{s.t. } \mathbf{x} \in \Omega,
\]

where \( \Omega \) is the decision space and \( \mathbf{x} = [x_1, x_2, \ldots, x_n]^T \) is a decision vector, and \( f_i : \mathbb{R}^n \rightarrow \mathbb{R} \), \( i = 1, \ldots, k \) are the objective functions.

Single-objective optimization problems may consist of a number of objective functions, as far as the objectives don’t conflict with each other. In this case a single solution exists, which can optimize all the objectives. On the other hand, if the objectives conflict, improvement of one may lead to deterioration of another. The best solutions in such case are called *Pareto* optimal solutions.

The following definitions are used in the concept of *Pareto* optimality:

**Definition 1.** A vector \( \mathbf{u} = [u_1, \ldots, u_m]^T \) is said to dominate another vector \( \mathbf{v} = [v_1, \ldots, v_m]^T \), denoted as \( \mathbf{u} \prec \mathbf{v} \), iff \( \forall i \in \{1, \ldots, m\}, u_i \leq v_i \) and \( \mathbf{u} \neq \mathbf{v} \).

**Definition 2.** A feasible solution \( \mathbf{x}^* \in \Omega \) is called a *Pareto* optimal solution, iff \( \exists \mathbf{y} \in \Omega \) such that \( \mathbf{F}(\mathbf{y}) \prec \mathbf{F}(\mathbf{x}^*) \).

**Definition 3.** *Pareto* set (PS) is the set of all the *Pareto* optimal solutions and can be denoted as

\[\text{PS} = \{ \mathbf{x} \in \Omega | \exists \mathbf{y} \in \Omega, \mathbf{F}(\mathbf{y}) \prec \mathbf{F}(\mathbf{x}) \}\]

The *Pareto* front (PF) is the image of the PS in the objective space

\[\text{PF} = \{ \mathbf{F}(\mathbf{x}) | \mathbf{x} \in \text{PS} \}\]

### 3.1.6 Multi-objective Optimization Algorithms

Among several existing optimization algorithms, evolutionary algorithms are considered as a powerful alternative. These algorithms are very effective to solve complex search problems, including single-objective [21] and multi-objective optimization problems [22]. Dealing with a group of candidate solutions, makes them effective to find a group of optimal solutions.

Differential Evolution [23] is currently one of the most popular and efficient evolutionary algorithms [21]. The backbone of the algorithm is based on weighted difference between solutions to perturb the population and to create candidate solutions. The performance of DE was compared with two other population based algorithms in our previous works [24, 25, 26]. In order to solve the described multi-objective problem in this work, a multi-objective differential evolution algorithm is utilized. In this section, firstly the standard differential evolution algorithm is briefly explained. Furthermore, a multi-objective extension of the differential evolution is discussed in more details.

**Differential Evolution**

The differential evolution (DE) algorithm [23] is a highly efficient yet simple optimization algorithm over a continuous domain. DE was initially designed for scalar objective optimization. The mutation operator works based on differences between pairs of solutions with the aim of finding a search direction using the distribution of the solutions in
the current population. Similar to other Evolution Strategy (ES) algorithms, DE is a population based approach and its recombination and mutation operators are the variation operators used to generate new solutions. Unlike, Genetic Algorithm (GA) and several ES approaches, solutions in DE are encoded with real values. Moreover, DE doesn’t use a fixed distribution as the Gaussian distribution adopted in ES approaches; instead, the current distribution of the solutions in the search space determines the search direction and even the stepsize for each individual.

DE utilizes $NP$ $D$-dimensional parameter vector $x_{i,G}$, $i = \{1, 2, \ldots, NP\}$ as a population for each generation $G$. In order to cover the entire parameter space, the initial population is sampled using a uniform probability distribution.

DE generates new parameter vectors by adding the weighted difference between two population vectors to a third vector. The mutant vector is generated as

$$v_{i,G+1} = x_{p_1,G} + F_c \cdot (x_{p_2,G} - x_{p_3,G})$$

(7)

with random indices $p_1, p_2, p_3 \in \{1, 2, \ldots, NP\}$, and $F_c > 0, F_c \in [0, 2]$ is a real and constant factor which controls the amplification of the differential variation. In the next phase, the crossover operation is introduced to increase the diversity of the perturbed parameter vector. The trial vector $u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1})$, is formed according to

$$u_{j,G+1} = \begin{cases} v_{j,G+1} & \text{if } \text{rand}(j) \leq CR \text{ or } j = \text{rnbr}(i) \\ x_{j,G} & \text{if } \text{rand}(j) > CR \text{ and } j \neq \text{rnbr}(i) \end{cases}$$

(8)

where, $\text{rand}(j)$ is the $j^{th}$ evaluation of a uniform random number. $CR \in [0, 1]$ is the crossover constant. $\text{rnbr}(i)$ is a randomly chosen index which ensures that $u_{i,G+1}$ gets at least one parameter from $v_{i,G+1}$.

The selection operator is used to decide whether or not the trial vector should become a member of the next generation. By comparing the trial vector to the target vector, the selection operator selects the vector with smaller cost.

In order to classify the different variants of DE the notation $\text{DE/a/b/c}$ is proposed by Storn and Price [23]. $a$ specifies the vector to be mutated. Two examples are ‘rand’ and ‘best’. $b$ is the number of difference vectors used in the strategy. $c$ denotes the crossover scheme (e.g. ‘bin’). The explained DE strategy can be written as: $\text{DE/rand/1/bin}$. The effect of the constant parameters of the algorithm and related tuning procedures were studied in recent years [27].

**Multi-objective Differential Evolution**

Because of its efficiency for solving problems, several multi-objective DE algorithms have been developed in recent years [28, 29, 30]. We use the multi-objective differential evolution (MODE) algorithm proposed in [31]. This algorithm is inspired from elitist non-dominated sorting genetic algorithm (NSGA-II). In a multi-objective domain, the goal is to identify the Pareto set (PS). In MODE, a population of size $NP$ is generated randomly and the fitness functions are evaluated. The population is then sorted based on dominance concept. In the next step, DE operations are carried out over the individuals of the population. Then the fitness of the trial vectors are evaluated. Unlike DE, in MODE the trial vectors are not compared with the corresponding parent vectors. Instead, both the parent vectors and the trial vectors are combined to form a global population of size, $2NP$. Then the global population is ranked by the distance calculation. The best $NP$ individuals are selected based on their ranking and distance and act as the parent vector for the next generation. Algorithm 1 shows a simplified pseudo code of the MODE algorithm.
Algorithm 1 Multi-Objective Differential Evolution

1: $P_{\text{parent}} \leftarrow$ Initialize random population
2: $J_{\text{parent}} = \text{eval}(P_{\text{parent}})$
3: for $i = 1$ to numGenerations do
4:     for $j = 1$ to numPopulation do
5:         $P_{\text{Mutant}} = \text{Mutation}(P_{\text{parent}})$
6:         $P_{\text{child}} = \text{crossover}(P_{\text{parent}}, P_{\text{Mutant}})$
7:     end for
8:     $J_{\text{child}} = \text{eval}(P_{\text{child}})$
9: for $j = 1$ to numPopulation do
10:    $[P_{\text{parent}}, J_{\text{parent}}] = \text{selection}(P_{\text{parent}}, P_{\text{child}})$
11: end for
12: $PF = J_{\text{parent}}$
13: $PS = P_{\text{parent}}$
14: end for
15: $[PF_{\text{out}}, PS_{\text{out}}] = \text{DominanceFilter}(PF, PS)$

3.2 Experimental Setup

All the experiments are based on dynamics model of Girona 500 (presented by Eq. 1) which its hydrodynamics parameters have been identified in [15]. All of the experiments, are designed so that the thruster failure occurs in the horizontal plane, while the heave movement of the AUV is always controlled by its original controller. In all the experiments the right surge thruster is deemed faulty. We assume the fault detection module not only detects the faulty thruster but also estimates its coefficient of functionality ($\alpha \in (0, 1]$). Thus, in different experiments $\alpha_2$ indicates the measure of the functionality of the second surge thruster. However, the other two thrusters are considered fully functional ($\alpha_1 = \alpha_3 = 1$). Unlike our previous work [24], our proposed approach benefits from the remaining useable power of the faulty thruster together with the other healthy thrusters. For each experiment we specify a final time ($T$) for each episode (e.g. $T = 60s$). The final time is selected somehow that the target is reachable in $T$ seconds. We also designed the vectorized reward function so that when the AUV reaches an area close enough to the desired position, $\|P_t - P_d\| < 0.2m$, the current episode is terminated. In all the experiments the policy depends on 5 state variables of the system (i.e. position, orientation, together with linear and angular velocities). Employing a 3rd order Fourier basis to represent the features of the policy, the number of optimization parameters is equal to 16 for each thruster. So, for the navigation in 2D plane including 3 thrusters (including functional or partially functional), the total number of the optimization parameters equals to 48.

3.3 Results

In this section a set of simulated experiments are performed. In the first experiment we consider a case in which the right surge thruster is fully broken. This experiment shows that the proposed method can deal with under-actuated vehicles. In the second experiment a partially functional thruster with various coefficient of functionality is considered. This experiment shows that even if the thruster is partially broken, the proposed approach can benefit from the useable remaining assets of the system to reach the specified target. In the third case, all the experiences in the previous experiments are used to increase the performance of the approach.
3.3.1 A Fully Broken Thruster

In the first experiment, we consider the case in which the right surge thruster is fully broken ($\alpha_2 = 0$). In such situation the AUV becomes under-actuated and any attempt to reallocate the actuator configuration matrix would be ineffective. The target is located 4m in front of the AUV and the algorithm tries to find a set of policies to satisfy the multiple-objectives defined in section 3.1.4. The set of optimal trajectories generated by employing the discovered Pareto optimal policies are depicted in Fig. 2. The related velocities are shown in Fig. 3. The effect of the multiple conflicting objectives can be seen in the figures. For instance, in Fig. 2 the final yaw orientation of the AUV in the light blue trajectory is better (less) than the other trajectories. On the other hand, in Fig. 3 the light green profile shows a better (less) final velocity.

3.3.2 A Partially Broken Thruster

In the second experiment, we consider the case in which the right surge thruster is partially broken ($\alpha_2 \neq 0$). In such situation the AUV is considered as redundant, but the dynamic behavior of the system has changed. Similar to the previous experiment, the target is located 4m in front of the AUV and the algorithm tries to find a set of optimal policies to satisfy the multiple-objectives defined in section 3.1.4. We repeated this experiment 11 times and each time the coefficient of functionality for the right surge thruster ($\alpha_2$) is increased by 10% from 0 to 1. The set of optimal trajectories generated by the discovered Pareto optimal policies in each case are depicted in Fig. ???. Since we kept the maximum number of episodes in each experiment fixed (50), in some cases the algorithm discovered more Pareto optimal policies while in some others it found less solutions. Increasing the maximum number of episodes leads to solutions with bigger rewards. In addition, the number of function evaluation in each case is equal to 1000 times. All the discovered control policies in this experiment are stored in the memory and is used in the next experiment.

Figure 2: Three optimal trajectories produced by employing the discovered Pareto optimal solutions. The arrows show the orientation of the AUV (see Section 3.3.1 for more details).
3.3.3 Improving the Performance

The idea behind this experiment is to benefit from the previous experience and increase the performance of the learning approach. The Pareto optimal solutions for different values of $\alpha$ are stored and are depicted in Fig. 8.

In Fig. 8, each row is related to a thruster and in each row 16 parameters of the policy $\theta$ are depicted. Each $\theta$ is plotted versus $\alpha$ ($\alpha = 0: 0.1: 1$). Since the behavior of the parameters seems to be nonlinear, using regression techniques to find a model would be ineffective. However, we want to investigate the effect of using previous experience by employing the nearest existing solution as initial population in the MODE algorithm.

A new experiment is performed where $\alpha_2$ is not exactly one of the previously experienced values. we initialize the approach with the nearest existing solution to increase the efficiency of the approach. In this experiment, $\alpha_2 = 0.45$ is assumed. All the other assumptions are similar to the previous experiments. We first initialized the algorithm with random population to discover an optimal control policy. Then we repeated the same simulation, starting from the existing optimal solution for $\alpha_2 = 0.4$. To be consistent, the whole experiment was repeated 20 times. The result is depicted in Fig. 9. In the left the number of function evaluations and in the right the time to reach the first optimal solution are compared between two cases. All experiments took place on a single thread on an Intel Core i3-2350M CPU 2.30GHz. The result suggests that starting from a neighbour solution can increase the performance of the approach.

3.4 Conclusion

We proposed a model-based direct policy search reinforcement learning approach for discovering fault-tolerant control policies for thruster failure recovery in AUVs. The approach learns a fault-tolerant policy on an on-board model of the vehicle and then executes the optimal policy on the real AUV. The model of the AUV is first reconfigured according to the
detected and isolated fault. A multi-objective reinforcement learning approach utilizes the reconfigured model to discover a set of optimal solutions. Each optimal solution can be used to generate a trajectory that is able to navigate the AUV towards a specified target while satisfying multiple objectives. To increase the persistent autonomy of the AUV, our approach can deal with partially and totally broken thrusters. In addition, the proposed approach is applicable when the AUV either becomes under-actuated or remains redundant in the presence of a fault. Finally, the efficiency of the approach is increased by taking advantage of the previous experiences.
Figure 4: Acquired results for the 2nd experiment with various coefficient of functionality for the right surge thruster for {0, 0.1, 0.2}. The trajectories are generated by employing the discovered optimal control policies. (see section 3.3.2 for more details.)
Figure 5: Acquired results for the 2\textsuperscript{nd} experiment with various coefficient of functionality for the right surge thruster between \{0.3,0.4,0.5\}. The trajectories are generated by employing the discovered optimal control policies. (see section 3.3.2 for more details.)
Figure 6: Acquired results for the 2nd experiment with various coefficient of functionality for the right surge thruster between \{0.6, 0.7, 0.8\}. The trajectories are generated by employing the discovered optimal control policies. (see section 3.3.2 for more details.)
Figure 7: Acquired results for the 2nd experiment with various coefficient of functionality for the right surge thruster between \( \{0.9, 1.0\} \). The trajectories are generated by employing the discovered optimal control policies. (see section 3.3.2 for more details.)
Figure 8: The discovered control policies for the 2nd experiment. Each row in the plot is related to a thruster. In each row 16 parameter are depicted (parameters of the policy representation, $\theta$). And each parameter is plotted versus $\alpha$. (see Section 3.3.3 for more details).
Figure 9: This plot shows the effect of employing previously experienced knowledge in the performance of the approach. The first experiment starts from random values, while the second experiment uses the nearest neighbor solution. The left plot shows the number of function evaluations and the right plot depicts the time needed to reach the goal. Both experiment has been repeated 20 times. (see Section.3.3.3 for more details).
4 Multi-dimensional reward vector in RL

In this section we will discuss a multi-dimensional reward vector approach for reinforcement learning. Conventional reinforcement learning uses a scalar value to evaluate the fitness of a policy. In this section we investigate an approach that uses multi-dimensional reward vector. A multi-dimensional reward vector has several advantages. Firstly, it can help in reducing chances of a search algorithm getting trapped in a local minima/maxima. Secondly, it can help in speeding up the rate at which the optimal policy is discovered. Thirdly, the algorithm produces multiple policies that favor a particular feature, for example, in the thruster failure task investigated in the PANDORA project, it might be favorable if the robot misses the target by a short distance, but takes a shorter route, hence, saving time and energy. A multi-dimensional reward vector facilitates in discovering such policies with less effort.

4.1 The testing platform

The testbed for our approach is the Girona500 AUV [32]. Figure 10 shows the Girona 500 AUV. It is a reconfigurable AUV, we used a layout that has 5 thrusters: two vertical thrusters for the heave, one lateral for the sway, and two horizontal for the yaw and surge. The reinforcement learning is performed in simulation, we use the model of the Griona500 AUV in an underwater simulator (UWSim – an UnderWater Simulator for marine robotics research and development). To test the multi-dimensional reward approach, we simulate a thruster failure in the Girona 500. Note that the approach can be applied to any reinforcement learning problem where the reward vector consists of multiple non-conflicting goals.

AUVs operate in harsh underwater environments where recovery from a system failure is critical in the recovery and return of the AUV back to the base. At the point of failure the failure is identified, then the robot uses simulations to learn a policy that uses the remaining thrusters to achieve its goal. In our case the goal is to move the AUV to a predetermined recovery point.

4.2 Methodology

We frame the problem as a vector optimization problem. In vector optimization multiple objectives are simultaneously optimized. As mentioned earlier, this is similar to multi-objective optimization. However, we only consider objectives that are non-conflicting.
Figure 11 shows an overview of the method. It consists of a parameterized policy representation, the AUV model and an optimization algorithm. The policy produces thruster commands, which is fed into the AUV model. After the model is executed, based on the final state of the AUV a reward vector is generated. The reward vector is then used as the input of an optimization algorithm that generates a new set of parameters. This is repeated until the optimal policy is discovered. In the rest of this section we will discuss in detail the components of our method.

4.2.1 The optimization algorithm

In the vector optimization framework multiple objectives are optimized simultaneously subject to a given ordering. If the objectives are non-conflicting, there exists a single optimal vector. However, in cases where there is a conflict between the objectives, it is not possible to find a single optimal vector. In such cases a Pareto optimum set of vectors is calculated. A Pareto optimal solution is defined as the set of vectors in which it is impossible to make one objective better without making at least one other objective worse off. Since in this section we are investigating a non-conflicting reward vector, the optimization algorithm should contain one best solution. Evolutionary optimization algorithms lend themselves well to the problem at hand. The advantage of the evolutionary algorithms is that they generate sets of solutions, which approximates the Pareto front.

We use the MATLAB implementation of a multi-objective evolutionary optimization algorithm, gamultobj, which is based on non-dominated sorting genetic algorithm II (NSGA-II) by Deb et al. [33]. The GA is an evolutionary optimization algorithm that works on the principles of natural selection. GAs can be used to search through a set of policy parameter to compute an optimal solution. Table 1 shows an overview of the GA algorithms. It consists of a population of policy parameter sets, a selection mechanism that selects the fittest candidates for reproduction, a crossover mechanism that combines the two candidate individuals into an offspring and finally a mechanism to mutate the offspring with a low
Table 1: Summary of the components of genetic algorithms

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Population:</strong></td>
<td>current pool of policy parameter sets.</td>
</tr>
<tr>
<td><strong>Selection:</strong></td>
<td>randomly select candidates from the population for reproduction based on a fitness function.</td>
</tr>
<tr>
<td><strong>Crossover:</strong></td>
<td>randomly select individuals to combine into an offspring, by choosing random parameters from the parents.</td>
</tr>
<tr>
<td><strong>Mutation:</strong></td>
<td>modify random parameters.</td>
</tr>
</tbody>
</table>

probability to reduce the likelihood of being trapped in a local optima.

Figure 12 shows a flowchart of the execution of a GA algorithm. The algorithm starts by choosing a random population of n individuals. In the next step the fitness of each individual policy is calculated. This is followed by selection, crossover and mutation that generates a new generation. At this point the algorithm checks if the solution has converged or maximum allowable iterations has been exceeded. In the affirmative case, the algorithm is terminated. Otherwise the process is repeated.

![GA flowchart](image-url)

Figure 12: GA flowchart.
4.2.2 The policy representation

The policy, \( \pi \), is represented with a linear function approximator that depends only on time, \( t \):

\[
u(t) = \pi(t | \theta) = \theta^T \phi(t)
\]

where \( u \) is the control input of the AUV model, i.e., the thruster commands, the functions \( \phi_i(t) \) are called basis functions, or features. We used a third order Fourier basis function. There are four optimization parameters for each thruster. In the scenario considered, there are two thrusters that can be controlled to reach the target position. Hence, we have eight parameters in total.

4.2.3 The AUV model

The model of the AUV is represented as a rigid body subject to external forces and torques. The 6 degree-of-freedom equations of the AUV are given by:

\[
M \dot{v} + C(v)v + D(v)v + g(\eta) = \tau
\]

\[
\dot{\eta} = J(\eta)v
\]

\[
\tau = Bu
\]

where \( M \) is the mass matrix; \( C \) is the Coriolis matrix; \( D \), is the drag matrix; \( g(\eta) \) is the hydrostatic restoring force vector; \( J(\eta) \) is the Jacobian matrix transforming the velocities from the body-fixed to the earth-fixed frame; \( \eta = [x \ y \ z \ \phi \ \theta \ \psi]^T \) is the pose (position and orientation) vector; \( v = [u \ v \ w \ p \ q \ r]^T \) is the body velocity vector; \( \tau \) is the force/torque vector; \( u \) is the input vector and \( B \) is the thruster reconfiguration matrix. The hydrodynamics parameters of the AUV are identified using an on-line identification algorithm [34, 35].

4.2.4 The reward vector

The reward vector used for our experiments consists of three components, namely, proximity of the AUV to the desired location in the x coordinate, proximity of the AUV to the desired location in the y coordinate, and the total distance traveled by the AUV.

\[
R = \left[ \begin{array}{c} \|x_{AUV} - x_{desired}\| \\ \|y_{AUV} - y_{desired}\| \\ \int_{t=0}^{T} p_{AUV}(t) \end{array} \right]
\]

where \( x \) and \( y \) are the coordinate locations, \( p_{AUV} \) is the AUV distance with respect to its starting position, \( t \) is time in seconds, and \( T \) is the time when the simulation is terminated, which takes place either when the robot reaches the desired location within a given error, \( \varepsilon \), or the simulation time limit, \( t_{\text{terminal}} \) is reached. In our experiments \( x_{desired}, y_{desired}, \varepsilon \) and \( t_{\text{terminal}} \) are set to 0 meters, 3 meters, 0.2 meters and 60 seconds, respectively. The \( x_{desired} \) and \( y_{desired} \)

4.3 Results

To measure the average performance of the proposed multi-dimensional reward we executed the algorithm twenty times. We also conducted twenty simulations with a scalar reward function. The length of the trajectory is crucial to getting the robot to a safe location at the quickly. Table 2 summarizes the results, comparing the multi-dimensional reward method to the scalar reward method. The results show that on average the multi-dimensional method does better than the scalar method.
Table 2: Summary of the length of the trajectories (meters). Comparing the multi-dimensional and the scalar reward approaches.

<table>
<thead>
<tr>
<th></th>
<th>Multi-dimensional</th>
<th>Scalar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>4.41 ± 0.69</td>
<td>4.44 ± 0.86</td>
</tr>
<tr>
<td>Longest trajectory</td>
<td>6.61</td>
<td>7.43</td>
</tr>
<tr>
<td>Shortest trajectory</td>
<td>2.82</td>
<td>2.83</td>
</tr>
</tbody>
</table>

The average trajectory produced by the proposed method is 4.41 ± 0.69 meters. The longest trajectory was 6.61 meters and the shortest trajectory was 2.82 meters. The direct distance to the desired location in the experiments is 2.80 meters. In comparison policies learned using the scalar method produced on average a trajectory of 4.44 ± 0.86. The longest and the shortest trajectories were 7.43 meters and 2.83 meters, respectively. Figure 13 and Figure 14 show the plot of the longest and the shortest trajectories for multi-dimensional approach and the scalar approach, respectively.

It is not surprising that the shortest trajectory produced both methods, that is, the scalar method and the multi-dimensional method, are similar. It is at the limit of the theoretical shortest possible path. However, we notice that the multi-dimensional method performs better on average and the longest trajectory produced by the multi-dimensional methods is much shorter than the longest trajectory produced by the scalar method.

4.4 Conclusion

We have proposed a method that uses multi-dimensional reward for reinforcement learning. We applied the method to help an AUV recover from a thruster failure. We showed that the policies produced by the method can successfully guide the AUV to a desired location. We also compared the method with the classical scalar method and showed that the current method is on average better than the scalar method. The worst case performance of the proposed method is also better than the scalar method. We also showed that the best case scenario is same as the scalar method.
Figure 13: Multi-dimensional reward method. The trajectories produced by the learned policies. When the robot gets within 0.2 m of the target, we terminate the policy.
Figure 14: Scalar reward method. The trajectories produced by the learned policies. When the robot gets within 0.2 m of the target, we terminate the policy.
Part II

Reactive Learning

1 Introduction

Robotic valve turning is a challenging task specially in unstructured environments with increasing level of uncertainty (e.g. underwater). The existing disturbances in the environment or the noise in the sensors can endanger both the robot and the valve during the operation. For instance, the vision system may be occluded and thus introduce a delay in updating the data, or even providing the system with wrong information. Exerting huge forces/torques on the valve by the robot, is another hazardous and highly probable situation. In such cases an autonomous system that is capable of observing the current state of the system and reacting accordingly, can help to accomplish the mission successfully even in the presence of noise.

Robotic valve manipulation contains a number of complex and challenging subtasks. There seem to be few published description of attempts directly related to this task. Prior works in industrial robotic valve operation, generally use nonadaptive classical control and basic trajectory planning methods. In [36], Abidi et al. tried to achieve inspection and manipulation capabilities in the semi-autonomous operation of a control panel in a nuclear power plant. A 6-DoF industrial robot equipped with a number of sensors (e.g., vision, range, sound, proximity, force/torque, and touch) was used. The main drawback is that their approach is developed for static environments with predefined dimensions and scales. For instance, the size and position of the panel, the valve, and other objects in the room are manually engineered into the system. More recent approaches generally use sensor-based movement methods which implies that the robot trajectories have not been programmed off-line [37]. In their experiments, the robot is equipped with a torque sensor and the valve handle is equipped with a proximity sensor. The valve is detected using a vision sensor. The authors focus on a model-based approach to avoid over-tightening/loosening of the valve. The other phases of the valve manipulation process are accomplished using classical methods. In their next experiments [38] the authors develop the valve manipulation task in outdoor environment. The vision sensor is replaced with a thermal camera, and the (round) valve is replaced with a T-bar valve, which is easier for the robot to manipulate. The main focus of this research is detecting the valve and avoiding the over-tightening/loosening of the valve in an early stage using a model-based technique.

Compared to our previous research [39] this work provides the following three new contributions.

1) In our previous research the turning phase was done by programming the turning motion into the robot. Here a force control strategy is proposed for controlling the turning phase instead; 2) Similar to the previous research a Reactive Fuzzy Decision Maker (RFDM) system is designed in order to react to external disturbances and sudden movements. The reactive system in the previous work monitors only the relative movement between the gripper and the valve. In addition, the new reactive system, takes into consideration the distance between the gripper and the valve, and the exerted forces to the end-effector. The resulting RFDM system shows more efficiency and better sensitivity which results in a safer valve turning process; 3) in our previous research an Optitrack system was used which captures real-time 3D position and orientation of a rigid body using a number of motion capture cameras and a set of markers. Although the Optitrack system is very precise, it cannot be used in outdoor environment. In this work, on the other hand, a single RGBD sensor is used that makes the experiments more realistic because nowadays, RGBD sensors are being used widely even in the underwater robotic tasks [40].

The proposed reactive layer in this work is evaluated in two different environments, in the lab and in underwater. The experimental set-up for all the experiments in the lab is
shown in Fig. 15. The underwater experiments are done on the GIRONA 500 AUV.

2 Methodology

The valve turning task comprises two main phases: reaching and turning. First, the robot have to learn how to reach the valve. Imitation learning approach which is designed specially to learn trajectory-based tasks, is a promising choice to learn the reaching skill [41]. In order to reproduce the reaching skill the robot utilizes feedbacks from an RGBD sensor. When the robot is able to reproduce the reaching skill a hybrid force/motion control strategy handles the turning phase. Hybrid force/motion control is a well-established method [42, 43, 44]. Using such hybrid strategy, the force controller can maintain the contact between the valve and the gripper while the motion controller turns the valve. The hybrid force/motion controller utilizes feedbacks from a force/torque sensor mounted between the end-effector and the gripper. Finally, the robot employs the reaching skill in reverse to retract from the valve.

In order to develop an autonomous system, the robot needs to deal with uncertainties. So we manually apply disturbances to the system. The disturbances during the execution of the task are monitored and handled by a Reactive Fuzzy Decision Maker (RFDM). Although such reactive system can be implemented using a thresholding method, the fuzzy system is chosen. The reason is that the fuzzy system provides a continuous decision surface and it infers from linguistic rules. The RFDM module, monitors the position of the gripper and the valve together with the magnitude of the forces and moments applied on the end-effector from the valve. Using this information, RFDM generates decisions that regulates the dynamics of the valve turning process. For example, RFDM can stop the process when the magnitude of the force increases due to an undesired movement. In addition, RFDM can also control the rate of the motion. For instance, when there is no external disturbance, the robot can reach the valve faster.

Fig.16 illustrates a flow diagram of the proposed approach. As can be seen in Fig. 15 the experimental set-up for all the conducted experiments in the lab consists of a 7-DoF KUKA-LWR manipulator mounted on a movable (wheeled) table, a (T-bar shaped) mock-up valve mounted on the wall in the robot’s workspace, a gripper designed for grasping and turning the valve, an ATI Mini45 Force/Torque (F/T) sensor which is sandwiched between the gripper and the robots end-effector, and an ASUS Xtion RGBD sensor for detecting and localizing the valve.

The experimental set-up for underwater environment consists of the Girona 500 AUV, equipped with a 4-DoF manipulator, a gripper designed for grasping and turning the valve, an ATI Mini45 Force/Torque (F/T) sensor, and an Bumblebee RGBD sensor for detecting and localizing the valve.

3 Imitation Learning

Imitation learning enables manipulators to learn and reproduce trajectory-based skills from a set of demonstrations [41]. In order to learn the reach-and-grasp skill, we utilize Dynamical Movement Primitives (DMP) [45] which is designed for modeling attractor behaviors of autonomous nonlinear dynamical systems. DMP consists of three main steps: demonstration, learning, and reproduction.

In order to create a pattern for reach-and-grasp behavior, multiple desired trajectories in term of position, velocity and acceleration have to be demonstrated and recorded by a human operator \([y_{\text{demo}}(t), \dot{y}_{\text{demo}}(t), \ddot{y}_{\text{demo}}(t)],\) where \(t \in [1, \ldots, P]\). A controller should convert desired position, velocity, and acceleration \(y, \dot{y}, \text{and} \ddot{y}\) into motor commands.

DMP employs a damped spring model that can be modulated with nonlinear terms such that it achieves a desired attractor behavior. The dynamics of damped string system
are represented in Equation. 11.

\[
\begin{align*}
\tau \dot{z} &= \alpha z (\beta (g - y) - z) + f + C_f \\
\tau \dot{y} &= z
\end{align*}
\]

(11)

where \( \tau \) is a time constant and \( \alpha \) and \( \beta \) are positive constants. With \( \beta = \alpha / 4 \) the system can be made critically damped in order for \( y \) to monotonically converge toward the goal \( g \).

This system is called transformation system because the forcing term is chosen to be nonlinear in the state of the differential equations and it transforms the simple dynamics of the unforced systems into a desired nonlinear behavior.

During the demonstration step, the trajectories are recorded independent of the explicit time, instead a first-order linear dynamics is defined as:

\[
\begin{align*}
\tau \dot{x} &= -\alpha x x + C_c
\end{align*}
\]

(12)

where \( \alpha \) is a constant. Starting from some arbitrarily chosen initial state \( x_0 \) such as \( x_0 = 1 \) the state \( x \) converges monotonically to zero. \( x \) is a phase variable, where \( x = 1 \) indicates the start of the time and the \( x \) close to zero means that the goal \( g \) has been achieved. Equation. 12 is called canonical system because it models the generic behavior of the model equations. In order to utilize the canonical system, an implicit timing equation can be defined, for instance: \( t = -\log(x) / \alpha \).

In this work we introduce, one of the capabilities of the implicit timing that is the reversible behavior of the system. For example, when the goal is reached, the arm can start retracting by switching the timing equation into: \( t = t_{final} + \log(s) / \alpha_c \). Such behavior is also employed when the reactive system sends a retract command (e.g. \(-0.75\)) to the manipulator.

The forcing term in Equation. 13 is chosen as follows:

\[
\begin{align*}
f(x) &= \sum_{i=1}^{N} \psi_i(x) \omega_i \bigg( g - y_0 \bigg) \\
\psi_i(x) &= \exp(-h_i(x-c_i)^2)
\end{align*}
\]

(13)

where \( \psi_i \) are fixed basis functions and \( w_i \) are weights, with \( N \) exponential basis functions \( \psi_i(x) \) where \( \sigma_i \) and \( c_i \) are the width and centers of the basis functions respectively and \( y_0 \) is the initial state \( y_0 = y(t = 0) \). The parameter \( g \) is the target (i.e. the position of the valve) coinciding the end of the movement \( g = y_{demo}(t = P) \), \( y_0 = y_{demo}(t = 0) \). Also the parameter \( \tau \) must be adjusted to the duration of the demonstration.

The learning process of the parameters \( w_i \) is accomplished with locally weighted regression method, because it is very fast and each kernel learns independent of others.

The set of demonstrations is depicted in Fig. 17 (the black curves). Following the described approach, the system learns a set of attractors which can be seen in the 2D plots in Fig. 17 and 18. Using the learned attractor model the robot is able to reproduce a new trajectory from an arbitrary initial position. Each red trajectory in Fig. 17 and 18 illustrates a reproduction. In both figures the goal, (the valve in our experiments), is shown in yellow. While the robot is moving along the generated trajectory, it can reverse the motion and move backwards if required. Such behavior can be seen in Fig. 18. It has to be commented that, by executing the reverse motion, the robot goes back to the center of the first attractor.

## 4 Force Control Strategy

### 4.1 The Force and Motion Controller

Once the robot learns the reach-and-grasp skill, the turning phase begins. In this phase, the goal of the robot is to turn the valve (by 180° from its initial configuration) while remaining
the position of the gripper. To control the forces and moments applied to the end-effector, a hybrid force/motion control approach is used [42, 43, 44].

Hybrid force/motion controller is a well-established method, and is preferred to be used in this application because during the turning phase a zero force controller can reduce the undesired forces and torques. The proposed hybrid controller consists of a five axes force controller and a motion controller around the turning axis.

The 5-DoF force controller is designed which minimizes the applied forces/torques exerted on the valve. The left 1-DoF is used to turn the valve. The force controller controls the forces along all the translational axes, and the torques around the \(x\) and \(y\) axes of the robot’s end-effector. A motion controller is used around the \(z\) axis of the end-effector.

The hybrid force/motion controller can be useful especially for autonomous underwater valve turning. In such unstructured environment the valve can be rusty and sensitive to high forces/torques. Also the force controller deals with external disturbances from the environment. As mentioned before, In order to measure the forces/torques applied to the end-effector a force/torque sensor is mounted between the gripper and the end-effector.

A coordinate frame is set with respect to the initial pose of the gripper. The \(z\)-axis (surge and roll) is considered to be normal to the end-effector’s palm. The \(y\)-axis (sway and pitch) is considered to be perpendicular to the \(z\)-axis and pointing sideways. And the \(x\)-axis (heave and yaw) is perpendicular to the \(z-y\) plane [46]. We specify the forces and moments as follows

\[
F_{\text{con}} = F_{\text{des}} + k_p (F_{\text{des}} - F_{\text{act}}) \\
M_{\text{con}} = M_{\text{des}} + k_p (M_{\text{des}} - M_{\text{act}})
\]  

(14)

where \(F\) and \(M\) denote forces and moments respectively, and subscripts \(\text{des}, \text{act}, \text{con}\) denote the desired, actual, and control parameters respectively. In the following sections, some experimental results are explained.

### 4.1.1 Valve Turning with Stationary Base

In the first set of experiments in the lab, the base of the robot remained stationary during the valve turning task. So no external disturbance is applied to the valve or the gripper. In this case, to ensure proper contact between the valve and the gripper during the turning phase a 20 N force along the \(z\)-axis is applied, \(F_{\text{z(des)}} = 20\) N. The other desired forces and torques are set to zero \(F_{\text{y(des)}} = F_{\text{x(des)}} = M_{\text{x(des)}} = M_{\text{y(des)}} = 0\). The desired roll angle is specified to be 180° clockwise with respect to the current orientation of the end-effector. The duration of turning is set equal to 10 sec. The joint angle displacements through the valve turning execution are shown in Fig. 19, where the last joint moved from \(-90\)° to 90° angle, while the rest of the angles move slightly in order to dissipate the forces generated at the end-effector during the execution of the task.

It can be seen in the graphs that the first 5 sec of the operation is used for the gripper to make contact with the valve and then dissipate the impact force. From 5 sec to 15 sec, the valve is turned by the motion controller and the generated forces and moments at the end-effector are recorded. Fig. 20 (top figure) shows that the force along the \(z\)-axis is maintained at \(+20\) N right after the impact forces have been dissipated. The controller can maintain the magnitude of the force before, during, and after the valve-turning successfully.

The exertion of the necessary moment to turn the valve and the required force to maintain the contact between the valve and the gripper, generates residual forces and moments at the end-effector. In order to dissipate these reaction forces along the \(x\) and \(y\) axes, a set of forces in range −5 to +5 N are generated. During the turning phase until the 20 sec of the task, the \(z\)-axis moment averaged at around \(+1\) Nm. This is shown in Fig. 21 (top figure).

On the other hand, the moment around \(x\)-axis is maintained at \(−0.5\) Nm at the start of the contact until the end of the turning phase, and then decreased to 0 Nm by the end of the
Deliverable 3.3.3

task. Around the y-axis, from its initial moment, (after impact that is around $-0.5 \text{ Nm}$), it moved to $-1 \text{ Nm}$ in an attempt to dissipate the residual moments. However, due to the physical limitation of the valve, (i.e. the gripper has a slot along the y-axis), it cannot create the necessary moment-resistance in order to dissipate the residual moment.

4.1.2 Valve Turning with Moving Base

In the next experiment in the lab the base of the manipulator, that is mounted on a wheeled table, is moved manually to create a disturbance during the turning task. These oscillatory disturbances simulate the dynamics of currents in the underwater environment. A demonstration of disturbance rejection during task execution is also shown in [47]. In this experiment, the duration of the valve turning process is set to 30 sec to give enough time to the operator to disturb the manipulator. The recorded force and moment during the valve turning with perturbed base are shown in bottom sub-figures in Figs. 20 and 21. The perturbation is generated from 3 sec up to 33 sec of the overall time.

5 Learning of Reactive Behavior

In robotic valve turning in the real world, a sudden movement of the arm can endanger both the valve and the manipulator. Also, if the robot exerts huge and uncontrolled amount of force/torque during the turning phase, it may break the valve off. In addition, a delay from the vision sensor can lead to unpredicted behaviors during the execution of the task. In order to prevent such behaviors and developing a more autonomous and reliable system, a reactive decision maker is designed. This system, which is a Reactive Fuzzy Decision Maker (RFDM), evaluates the dynamic behavior of the system and regulates the robot’s movements reactively.

We chose fuzzy systems because they are based on linguistic rules and the parameters that specify the membership functions have clear physical meanings. Also, there are methods to choose good initial values for the parameters of a fuzzy system [48].

The RFDM system in our previous research [39], monitors the relative movement between the valve and the end-effector according to the defined linguistic rules. One of the drawbacks of the previous reactive system is that, it is independent of the distance between the gripper and the valve. This means that despite the distance between the gripper and the valve, the robot shows the same behaviors. The proposed RFDM in this work, on the other hand, observes the distance between the gripper and the robot’s end-effector. This extra information gives the RFDM the capability to behave more adaptively. For instance, when the gripper is about to grasp the valve, the new RFDM generates more watchful decisions. In addition, the new RFDM monitors and reacts to the uncertainties in the force/torque applied to the gripper.

5.1 Design of the Fuzzy System

The proposed fuzzy system comprises three inputs: a) the relative movement between the valve and the gripper (in x-y plane); b) the distance from the gripper to the valve (the norm of the distance vector); and c) the force applied to the valve from the robot.

All the inputs are first normalized in range $[0, 1]$ and then are sent to the RFDM system. The third input (the force) is provided by the force/torque sensor and it includes three forces and three torques. In this case, the moment is multiplied by a factor to be numerically comparable to the value of the force. The normalizing equation is as follows

$$\gamma = \frac{\|f\| + s\|m\|}{f_{\text{max}}}$$

(15)
where $\gamma \in [0, 1]$, the factor $s = 10$ to level-off the range of values between the forces and the moments, and $f_{\text{max}} = 30$ is set as the maximum threshold.

Monitoring the relative movement of the valve and the gripper, the system can detect oscillations with different amplitudes and frequencies. For instance, if the end-effector is reaching the valve, and the system senses an oscillation with Medium amplitude the fuzzy system reacts to that by stopping the arm. To simulate such behavior, the operator manually moves the table of the robot back and forth. Moreover, Considering the distance between the gripper and the valve, the system can change its sensitivity. For example, if the gripper is Far from the valve, even in the presence of a disturbance, the robot still can move towards the valve. On the other hand, if the robot is in the vicinity of the valve it should react to smaller oscillations and wait or even retracts the arm. Furthermore, measuring the force/torque magnitudes applied to the gripper, generated by colliding either to the valve or other objects, the system reacts according to the defined rules.

The output of the RFDM system is the reactive decision, a real number in $[-1, 1]$, which $-1$ means retract with 100% speed, 0 means to stop, and 1 means to approach with 100% speed. Therefore, the RFDM system not only decides the direction of the movement, but also specifies the rate of the movement.

In order to design the fuzzy system, we consider the inputs to be $x = [x_1, x_2, x_3]^T$ and the output as $r(x)$. Firstly, $N_i (i = 1, 2, 3)$ fuzzy sets, $A_1^1, A_2^1, \ldots, A_N^1$, are defined in $[0, 1]$, which are normal, consistent, and complete with Gaussian membership functions $\mu_{A_1^1}, \mu_{A_2^1}, \ldots, \mu_{A_N^1}$. Then, we form $M = N_1 \times N_2 \times N_3$ ($3 \times 4 \times 3 = 36$) fuzzy IF–THEN rules as follows:

\[
\text{IF } x_1 \text{ is } A_{1i} \text{ and } x_2 \text{ is } A_{2i} \text{ and } x_3 \text{ is } A_{3i} \text{ THEN } y \text{ is } B_{ij}^{i_{ij}}
\]

Moreover, 7 constant membership function in range $[-1, 1]$ is set for the output. Finally, the TSK fuzzy system $r(x)$ is constructed using product inference engine, singleton fuzzifier, and center average defuzzifier [48]:

\[
r(x) = \frac{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \sum_{i_3=1}^{N_3} y_{ij}^{i_{ij}} (\mu_{A_{1i}^1}(x_1)) \mu_{A_{2i}^2}(x_2) \mu_{A_{3i}^3}(x_3))}{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \sum_{i_3=1}^{N_3} (\mu_{A_{1i}^1}(x_1)) \mu_{A_{2i}^2}(x_2) \mu_{A_{3i}^3}(x_3))}
\]

Since the fuzzy sets are complete, the fuzzy system is well-defined and its denominator is always non-zero. The designed fuzzy system consists of three inputs and one output and cannot be illustrated in a single 3D graph. But we plotted the the fuzzy surface of the 2nd and 3rd inputs for a single value of the 1st input. So each surface is related to a fixed value of the 1st input (Fig. 22). It can be seen from Fig. 22 that the RFDM shows more sensitive and cautious behaviors as the distance from the valve decreases.

### 5.2 Tuning the Fuzzy System

One of the existing tuning methods which is applicable for fuzzy systems is called apprenticeship learning. In apprenticeship learning the subconscious knowledge of a human expert is derived and utilized for tuning the parameters of the system. In such case, the human expert knows what to do but cannot express exactly in words how to do it. In order to extract the subconscious knowledge of the human expert, a tutor simulates the effect of the disturbances (e.g. underwater currents) by moving the wheeled table, while the robot tries to reach and turn the valve. Simultaneously, using a slider button, another tutor can regulate the movements of the robot arm while it is following the reproduced trajectory or turning the valve. The tutor applies appropriate continuous commands in range $[-1, 1]$, to the system, where $-1$ means go backward along the trajectory with 100% speed and 1 means go forward along the trajectory with 100% speed. For instance, when the base of the robot is being oscillated say with a Big amplitude, the tutor smoothly moves the slider backwards to retract the arm and prevent it from any collision with the valve or the
Deliverable 3.3.3

Panel. All data, including the position of gripper and the valve, and the tutors commands are recorded during the learning process. The recorded data is then used to tune the RFDM in off-line mode.

The error between the recorded data from the tutor, which is a fuzzy surface, and the output of the un-tuned fuzzy system is used to make the objective function. The objective function can be minimized using various optimization algorithms. In [39], we applied four different optimization algorithms including gradient-descent, cross entropy method [49], covariance matrix adaptation-evolution strategy (CMA-ES) [50], and modified Price algorithm [51]. The number of optimization parameters in our problem is equal to the number of membership functions multiplied by two (center and standard deviation), plus the number of constant outputs. In our design the number of optimization parameters is equal to 79 (36 Gaussian Membership Functions × 2 parameters for each Gaussian + 7 constant outputs).

In this work we use CMA-ES for the tuning task, because CMA-ES is typically applied to search space dimensions between three and a hundred [50].

5.3 Experimental Result of the Reactive System

In this section, the behavior of the proposed RFDM system is investigated during a real-world valve turning experiment.

5.3.1 Lab Experiments

The result for experiments conducted in the lab environment are presented in this section. All three inputs of the reactive system are illustrated in the left side of Fig. 24. The first input, the distance, shows that initially the gripper was located far from the valve and gradually approached towards it. The second input, the relative movement, shows that, at some point a relative movement between the gripper and the valve is occurred. The relative movement was created manually by moving the robot’s base. And the third input, the force, shows small values during the process but at the end a sudden jump in the force was occurred. The jump in the force magnitude was generated manually by pushing the manipulator towards the valve during the turning phase. The generated decision commands by the RFDM system is plotted in the right sub-plot of Fig. 24. The effect of both the manual oscillation of the base and the manual push on the gripper is observed by RFDM and proper decisions are generated. During the manual oscillation of the base, the RFDM system decreases the rate of the motion towards zero. However, it does not retract the arm because at this point the gripper has not reached the valve yet. Also by sensing the force created by the sudden push, the RFDM system retracts the gripper from the valve.

5.3.2 Underwater Experiments

The result for experiments conducted in the underwater environment are presented in this section. In the top row of Fig. 25, the first subplot illustrates the output of the RFDM system. The middle subplot shows the distance of the gripper from the valve, which is the first input of the RFDM system. The third subplot shows the measured relative movement between the end-effector and the valve. In the bottom row of Fig. 25, the first subplot shows the delay of the sensor. It can be seen that at some points the delay is Big and the reactive system generates negative decisions and commands the robot to go back. The second subplot shows the relative movement of the gripper with respect to the end-effector. The reactive system reacts to bigger relative movements with more sensitivity.

The relative movement was created manually by moving the AUV using the joystick. The effect of both the manual oscillation of the base and the delay of the sensor is observed by RFDM and proper decisions are generated. During the manual oscillation of the base,
the RFDM system decreases the rate of the motion towards zero. However, it does not retract the arm because at this point the gripper has not reached the valve yet.

6 Conclusions

We have proposed a learning method for reactive robot behavior to deal with the challenging task of autonomous valve turning. The autonomous valve turning consists of two main phases: reaching and turning. Imitation learning is used to learn and reproduce the reaching phase. A hybrid force/motion controller is devised to accomplish the turning phase. In order to increase the autonomy of the system a reactive fuzzy decision maker is developed. This module evaluates the dynamic behavior of the system and modulates the robots movements reactively. The validity and performance of our approach is demonstrated in a real-world valve-turning experiment both in the lab and in underwater environment.
Figure 15: The experimental set-up for the valve-turning task. The valve is detected and localized using an RGBD camera through an AR-marker. The manipulator is equipped with a gripper and is mounted on a movable (wheeled) table. During the execution of the task, a human can create a random disturbance by perturbing the base of the robot.
Figure 16: A high-level flow diagram illustrating the different components of the proposed approach.
(a) Trajectories and learned attractors in 2D plane.

(b) Trajectories in 3D space.

Figure 17: The recorded trajectories that form the set of demonstrations (black), and the re-produced trajectory from an arbitrary initial position (red) towards the target are illustrated. The blue ellipses show the attractors and the yellow square shows the target (the valve).
Figure 18: The recorded trajectories that form the set of demonstrations (black), and the reproduced trajectory from an arbitrary initial position (red) are illustrated. The robot retracts from the middle of the path by receiving a command from another layer.
Figure 19: Joint angles during the valve turning task with stationary base.
Figure 20: End-effector forces during the valve turning task.
Figure 21: End-effector moment during the valve turning task.
Figure 22: Fuzzy inference system surface including three inputs. The input specifying the distance between the robot and the valve $x_1$ affects the sensitivity of the designed fuzzy system. Each surface shows a fixed value of the 1st input for the whole range of the 2nd and 3rd inputs.
Figure 23: Fuzzy membership functions for each input.

Figure 24: The recorded set of inputs and the generated decision commands by the RFDM system during a real-world valve turning experiment in the lab environment.
Figure 25: The recorded set of inputs and the generated decision commands by the RFDM system during a real-world valve turning experiment in underwater environment.
Conclusions

In this part we give a snapshot of the achievements and conclusions of this deliverable.

1 Reinforcement Learning

1.1 Multi-objective RL

We proposed a model-based direct policy search reinforcement learning approach for discovering fault-tolerant control policies for thruster failure recovery in AUVs. The approach learns a fault-tolerant policy on an on-board model of the vehicle and then executes the optimal policy on the real AUV. The model of the AUV is first reconfigured according to the detected and isolated fault. A multi-objective reinforcement learning approach utilizes the reconfigured model to discover a set of optimal solutions. Each optimal solution can be used to generate a trajectory that is able to navigate the AUV towards a specified target while satisfying multiple objectives. To increase the persistent autonomy of the AUV, our approach can deal with partially and totally broken thrusters. In addition, the proposed approach is applicable when the AUV either becomes under-actuated or remains redundant in the presence of a fault. Finally, the efficiency of the approach is increased by taking advantage of the previous experiences.

1.2 Multi-dimensional reward vector in RL

We have proposed a method that uses multi-dimensional reward for reinforcement learning. We applied the method to help an AUV recover from a thruster failure. We showed that the policies produced by the method can successfully guide the AUV to a desired location. We also compared the method with the classical scalar method and showed that the current method is on average better than the scalar method. The worst case performance of the proposed method is also better than the scalar method. We also showed that the best case scenario is same as the scalar method.

2 Reactive Learning

We have proposed a learning method for reactive robot behavior to deal with the challenging task of autonomous valve turning. The autonomous valve turning consists of two main phases: reaching and turning. Imitation learning is used to learn and reproduce the reaching phase. A hybrid force/motion controller is devised to accomplish the turning phase. In order to increase the autonomy of the system a reactive fuzzy decision maker is developed. This module evaluates the dynamic behavior of the system and modulates the robots movements reactively. The validity and performance of our approach is demonstrated in a real-world valve-turning experiment both in the lab and in underwater environment.
References


[38] David A Anisi, Charlotte Skourup, and ABB Petrochemicals. A step-wise approach to oil and gas robotics. In IFAC Workshop on Automatic Control in Offshore Oil and Gas Production, Trondheim, Norway, volume 31, 2012.


