Human-Agent Interaction Model Learning based on Crowdsourcing

Jack-Antoine Charles
ISAE-SUPAERO,
Université de Toulouse,
France

Caroline P. C. Chanel
ISAE-SUPAERO,
Université de Toulouse,
firstname.lastname@isae-supaero.fr

Corentin Chauffaut
ISAE-SUPAERO,
Université de Toulouse,
France

Pascal Chauvin
ISAE-SUPAERO,
Université de Toulouse,
France

Nicolas Drougard
ISAE-SUPAERO,
Université de Toulouse,
France

ABSTRACT

Missions involving humans interacting with automated systems become increasingly common. Due to the non-deterministic behavior of the human and possibly high risk of failing due to human factors, such an integrated system should react smartly by adapting its behavior when necessary. A promise avenue to design an efficient interaction-driven system is the mixed-initiative paradigm. In this context, this paper proposes a method to learn the model of a mixed-initiative human-robot mission. The first step to set up a reliable model is to acquire enough data. For this aim a crowdsourcing campaign was conducted and learning algorithms were trained on the collected data in order to model the human-robot mission and to optimize a supervision policy with a Markov Decision Process (MDP). This model takes into account the actions of the human operator during the interaction as well as the state of the robot and the mission. Once such a model has been learned, the supervision strategy can be optimized according to a criterion representing the goal of the mission. In this paper, the supervision strategy concerns the robot's operating mode. Simulations based on the MDP model show that planning under uncertainty solvers can be used to adapt robot's mode according to the state of the human-robot system. The optimization of the robot's operation mode seems to be able to improve the team's performance. The dataset that comes from crowdsourcing is therefore a material that can be useful for research in human-machine interaction, that is why it has been made available on our web site.

CCS CONCEPTS

• **Information systems → Crowdsourcing**; • **Human-centered computing → Human computer interaction (HCI)**; • **Computing methodologies** → *Planning under uncertainty*; *Supervised learning by classification*; Maximum likelihood modeling; • **Computer systems organization** → Robotics;

KEYWORDS

Human-Robot Interaction, Mixed-Initiative Mission, Crowdsourcing, Markov Chain Learning, Markov Decision Process, Classification

ACM Reference Format:
Jack-Antoine Charles, Caroline P. C. Chanel, Corentin Chauffaut, Pascal Chauvin, and Nicolas Drougard. 2018. Human-Agent Interaction Model Learning based on Crowdsourcing. In *6th International Conference on Human-Agent Interaction (HAI'18), December 15–18, 2018, Southampton, United Kingdom.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3284432.3284471

1 INTRODUCTION

Human and artificial agent interaction is an actual research track that covers various disciplines. Artificial intelligence, human factors and sociology are few examples of involved topics. Due to the increase of the decisional autonomy of artificial agents, *e.g.* robots, autonomous cars and unmanned aerial vehicles (UAVs), the role of the human operator is reduced regarding direct control, and concentrated on higher level decisions, that are not automated for practical, ethical or legal reasons. The use of automated planning for artificial agents actions has been amplified by the recent technical advances in artificial intelligence and machine learning. As an example, convolutional networks led to artificial vision [31] and popularized deep learning techniques, which played an important role in the latest successes of decision making algorithms based on Reinforcement Learning [33] and Planning under Uncertainty [18].

However, human operators are still vital in numerous scenarios. In particular they are able to produce tactical, moral, social and ethical decisions [22]. Such decisions are not (yet) assigned to machines. For instance, legal regimes need people for responsibility assessment issues, encouraging human supervision of automated systems.

On the other hand, this drastic change of the human operator role in favor of system's autonomy results in a new paradigm also known as mixed-initiative [15]. Mixed-initiative human-robot interaction considers human operators and artificial agents as a team [22], in which each agent can seize the initiative from the other. From the human operator's point of view it is not always bearable or acceptable that such an artificial system could seize the initiative, except if human cognitive capabilities or performance are degraded. A study reports that human factors are involved in 80% of autonomous aerial vehicles accidents [36]. This fact is due to several constraints experienced by humans during their missions. Stress, high workload, fatigue or boredom, which can be induced respectively by pressure (e.g. cause by a danger), complexity, hardness or duration of their tasks, are some of the main problems encountered by humans. As a result, an intelligent supervision system could lend a strong hand in order to help the human operator and the human-robot team to perform better.

Hence, an appropriate supervision strategy has to manage the information given to the human operator, the task allocation between the human and the machine, as well as the machine policy during its own tasks. In other words, the supervision strategy goal is to drive human-machine teams firstly by allocating the tasks that can be carried out by both the human and the machine, secondly, by providing, or not, appropriate alarms to the human operator, and finally by adapting actions of the machine according to the human behavior.

Considering that the human behavior is not deterministic, as well as environment dynamics, events occurring in a mixed-initiative mission can be considered as uncertain. A classical automated planning framework for probabilistic domains can be used to handle such a mixed-initiative human-robot interaction problem: the Markov Decision Processes (MDPs) [25]. MPDs allow to define the goal of a given mission in terms of rewards valuating states of the system. The optimization of MDPs consists in computing a strategy maximizing the expected sum of rewards over time [2].

The drawback of using MDPs is the need for a precise transition model, the which must faithfully represent the dynamics of the system. On one hand, Reinforcement Learning (RL) [35] can be explored in cases only a generative model is available to learn the optimal actions during repetitive mission realizations. On another hand, if a sufficient number of missions have been carried out before hand, it is possible to learn the MDP's parameters, and so, MDP optimization algorithms can be applied to obtain the optimal strategy.

This work is built based on this second approach. For this purpose, a mixed-initiative mission, called *Firefighter Robot* game [8], has been designed to reveal some general problems that occur in human-machine interaction. This mission, which simulates a remotely operated robot, is available on an opened website[1]. Advertising was done in the authors' (professional and social) networks to encourage Internet users to carry out the mission in order to collect as much anonymous data as possible. This crowdsourcing platform has collected more than a thousand mission realizations, allowing the application of machine learning techniques, namely classification and Markov Chain learning, to define the parameters of the MDP that models the mission.

This paper is organized as follows: firstly the designed mixed-initiative mission available on the crowdsourcing platform is presented as well as the produced dataset. Then the methodology used to learn a human-robot interaction model is given, followed by a complete mission model definition in the form of an MDP. Finally simulations are performed, evaluation results are presented and future work is discussed.

[1]http://robot-isae.isae.fr

2 CROWDSOURCING FOR MASSIVE DATA COLLECTION

As explained earlier, a crowdsourcing platform has been set up to allow users to perform the mission called Firefighter Robot game. This human-robot mission immerses the user in a scenario where he plays a fireman who must cooperate with a robot that is present in a small area with few trees. These trees have a weird tendency to self-ignite for some unknown reason. Through the graphical user interface (GUI) shown in Figure 1, the human operator gets the position of the robot in a map (bottom center), as well as the video streaming from its camera (top right).

The battery charge level of the robot decreases with time. However, when the robot is in the charging station, represented by a red square on the ground, the battery recharges. If the battery is empty and the robot is not on the red square, the mission fails and is finished. All the information related to the robot is summarized at the bottom right of the GUI.

The volume of water contained by the robot is not unlimited: to recharge in water, the robot has to be in the water station represented by a blue square on the ground and the associated tank has to contain enough water. For that, the human operator has to fill this ground tank using the buttons on the left-side of the interface: a tap, which can move horizontally by actuating a wheel (top buttons), fills the tank when it is in the middle (which is an unstable equilibrium). To actually fill the tank, the button bellow (black tap) turns on the tap for few seconds. Leaks may appear on the tank during the mission causing it to lose water: the button below (black wrench) can be used to fix them.

With the help of this robot, the goal of the mission is to fight as many fires as possible in a limited amount of time (ten minutes). The temperature of the robot increases when it is too close to flames and the mission terminates when it is too hot. The presence of fires is supposed to be felt as a danger by the operator. The robot, when its mode is "manual", is controlled by the arrows (navigation) and the space bar (shoot water) of the keyboard. In "autonomous" mode, the robot drives itself with a hard-coded strategy, including shooting water and the recharge of water or battery when necessary.

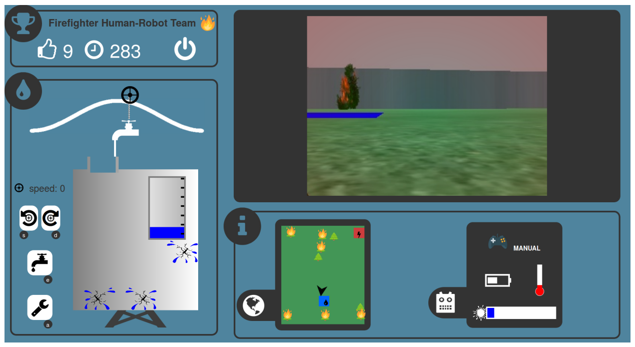

Figure 1: Graphical user interface (GUI) of the Firefighter Robot mission. The score and remaining time are displayed at the top left and the video from the robot is available at the top right, above the robot position and status information. The water reserve management task is displayed at the bottom left of the interface.

Integer	Associated alarm
−1	No alarm displayed
0	"Low battery"
1	"Too-high temperature"
2	"Less than one minute before the end of the mission"
3	"The robot's tank will soon be empty (2 shoots left)"
4	"The robot is in autonomous mode"
5	"The robot is in manual mode"
6	"The ground tank's water level is low"

Table 1: The different alarms that can be displayed during the mission as well as the associated integers, *i.e.* the numbers representing each of the alarms in the dataset.

For model learning purposes, the robot's operating mode can change randomly every ten seconds: it can be autonomous or it can need for manual control. This uniform sampling technique is used in order to get a balanced dataset (as much data under both supervision actions) needed to learn model probabilities for each action. Like the robot mode, the display of an alarm is also considered as a supervision action. When an alarm can be displayed, *i.e.* when the information it provides is true, the action of displaying this alarm is also randomly selected, in order to get data from both conditions (with and without alarms). The complete list of alarms is given in Table 1.

Temperature, battery and external tank management, as well as the score (number of extinguished fires) and the remaining time should imply stress and pressure. Pretests revealed that both tasks (robot control and water management inspired from MATB [6]) are complex enough to generate cognitive workload in the human operator, notably highlighted by variations in her engagement [9]. Such a (degraded) mental state can impair the human operator's cognitive abilities, and thus increase the risk of mission failure.

2.1 A Human-Robot Mission Dataset

The total time of recorded missions reaches more than 85 hours. In addition, in at least 55% percent of the whole samples, the human operator interacts with the interface. For instance she clicks on the interface, or uses the keyboard. Note that, each sample represents the amount of data collected during one second.

All anonymous data collected is available online[2]. The history of each mission is saved in a comma-separated values (CSV) file *i.e.* in the form of a table. The system status, that is the mission context and human-system interaction data, is recorded at every second. The first line of the file is always the same since the mission always starts in the same initial state. Thus, the information actually generated by the human-machine system goes from line 2 to line $H + 1$ (line 601 if no premature game over occurs) where $H \in \{1, \ldots, 600\}$ is the process horizon.

The remaining mission time r_t in seconds is the first column and takes values from 600 to $600 - H$. If $H = 600$ the mission is completed until the end and without premature game over. Since the missions are not always successful, not all missions have the same duration.

[2]https://personnel.isae-supaero.fr/isae_ressources/caroline-chanel/horizon/

The next two columns correspond to the potential actions of the human-robot team's supervision system. Indeed the second column contains the different operating modes taken by the robot during the process: $a_t^m = 1$ if the robot is autonomous during the second t, and $a_t^m = 0$ if it is in manual mode. The third column contains the alarms that can be triggered during the mission and is denoted by a_t^a. Please refers to Table 1 for alarm encoding details.

The next three columns describe the robot pose: $\forall t \in \{0, \ldots, H\}$, $(x_t, y_t, \theta_t) \in ([-20, 20]^2 \times] - \pi, \pi])$. For instance, on the map such as displayed on the GUI, or in Figure 2, the x_t (resp. y_t) increases when the robot moves to the right (resp. goes up the map). The angle θ_t is zero when the robot is oriented to the right and grows in the trigonometric direction.

Then comes the column describing the condition of the trees over time, *i.e.* which trees are on fire and which are not. A number is assigned to each of the nine trees and their coordinates are given in Table 2. By noting $f_t^i \in \{0, 1\}$ the state of the tree i (1 for *on fire* and 0 otherwise) at time $t \in \{0, \ldots, H\}$, the value given in the CSV file is the forest state $f_t = \sum_{i=1}^{9} f_t^i \cdot 10^{9-i}$. Thus, the resulting binary number at i^{th} digit, beginning from the left, denotes the state of tree f_t^i.

The next columns are dedicated to battery level $b_t \in [0, 100]$ and temperature $T_t \in [20, 240]$. The mission fails and the interface displays a game over when the battery is empty, $b_t = 0$, or when the temperature is too high, $T_t \geqslant 240$. Then, the successive water levels of the tank embedded on the robot $w_t^r \in \{0, 10, \ldots, 90, 100\}$ and of the one on the ground $w_t^g \in [0, 100]$ are given in the following two columns. This last tank allows the robot to be filled during the mission if containing enough water.

Leaks can appear at nine different points on the ground tank. These points follow a 3×3 grid pattern. By ordering these points from left to right then from top to bottom, the state of the point $i \in \{1, \ldots, 9\}$ at the second $t \in \{0, \ldots, H\}$ is denoted by $l_t^i \in \{0, 1\}$, with 1 for *leak at this point* and 0 otherwise. As like the states of the trees (f_t), the values in this new column are $l_t = \sum_{i=1}^{9} l_t^i \cdot 10^{9-i}$.

Finally, the last four columns concern the operator's actions on the interface. Note that, in one second, several keyboard keys could be pressed or several clicks on buttons (say $n \in \mathbb{N}$) could be performed. Thus, these three columns contain word sequences

tree	x	y
1	4.55076	14.66826
2	−0.7353	14.75052
3	−15.10146	15.76476
4	−2.6019	6.7425
5	−1.33158	10.02042
6	16.58292	−12.5847
7	16.87086	−16.01952
8	0.6078	−16.23906
9	−16.65378	−16.23906

red square (battery)

x	y
0.0	−10.0

blue square (water)

x	y
16.0	16.29

Table 2: Location of the trees and the blue and red squares in the map (see Figure 2). When considering the map displayed on the interface, the x-axis points to the right, the y-axis to the top.

separated by dashes. The first contains the keyboard key sequences used to control the robot,

$$k_t \in \{\text{"front"}, \text{"back"}, \text{"left"}, \text{"right"}, \text{"space"}\}^n, \text{ with } n \in \mathbb{N}.$$

These keys correspond respectively to the top, down, left and right arrow keys, and to the space key. For example consider the user's actions when the mission duration is between $t-1$ and t seconds. If the "front" key is pressed, then the "right" key to turn the robot and finally the "space" key to throw water, then the $(t+1)^{th}$ line of this column will contain for instance k_t = "front-front-front-right-space". The second column concerning human behavior contains the sequence of clicks on the interface buttons,

$$c_t \in \left\{\text{"left"}, \text{"right"}, \text{"push"}, \text{"wrench"}, (\text{"leak\_i"})_{i=1}^{9}, \text{"rm\_alarm"}\right\}^n$$

with $n \in \mathbb{N}$. Buttons "left" and "right" control the tap of the water management task, while "push" turns it on, and "wrench" turns the mouse pointer into a wrench. When the user then clicks on the leak $i \in \{1,\dots,9\}$, it disappears and adds "leak_i" to the clicks sequence, *e.g.* c_t = "wrench-leak_2-wrench-leak_6". When a visual alarm occurs, and the operator clicks on the button "I got it" to remove it from the interface, it leads to the addition of the word "rm_alarm" in this word sequence. The third column related to human behavior records clicks and keys pressed by mistake, as:

$$e_t \in \{\text{"down"}, \text{"up"}, \text{"click"}\}^n, \text{ with } n \in \mathbb{N}.$$

If e_t ="down-up-click" it means that, during second t, a key with no effect was pressed, then such a key was released, and finally an useless click was made. Finally, the last column contains the number of keyboard shortcuts used in the second $t \in \mathbb{N}$. Indeed, the use of some buttons (namely "left", "right", "push" and "wrench") of the GUI (*cf.* Figure 1) can be replaced by keyboard shortcuts (respectively, "s", "d", "e" and "a" keys). Thus, this column contains the number of times a shortcut has been used during the associated second: $\sigma_t \in \mathbb{N}$.

Note that the notation -1 for "No alarm displayed" (see Table 1) is also used in these last four columns. If, in a time step t, no useful key is pressed (resp. no click on buttons is performed, no useless action has been taken, no keyboard shortcut is used), *i.e.* if $n = 0$, then $k_t = -1$ (resp. $c_t = -1$, $e_t = -1$, $\sigma_t = -1$). In the same way, if the value in the dataset is -2, it means that the data is missing.

In summary, the number of columns of this Human-Robot Interaction dataset is equal to 16, and any line of such a CSV file describing a Firefighter Robot mission is organized as follows:

$$m_t = (r_t, a_t^m, a_t^a, x_t, y_t, \theta_t, f_t, b_t, T_t, w_t^r, w_t^g, l_t, k_t, c_t, e_t, \sigma_t)$$

where, m_t is the line number $t+1$ describing events occurring during the time segment $]t-1, t]$ in seconds (or a constant initial state if $t = 0$). As mentioned above, each mission (CSV) file is a $(H+1) \times 16$ table. Each line is a 16-dimensional vector m_t, thus the dataset can be seen as the realization of a random variable sequence of size $H+1$ representing the mission process $(m_t)_{t=0}^{H}$. Note that only k_t, c_t and e_t are not numbers.

In a few words, the following work assume that this sequence is a Markov chain in order to benefit from the associated learning and planning techniques. Next section describes the formal model used in this paper, the methodology applied to learn a human-agent interaction model, as well as the overall mission model.

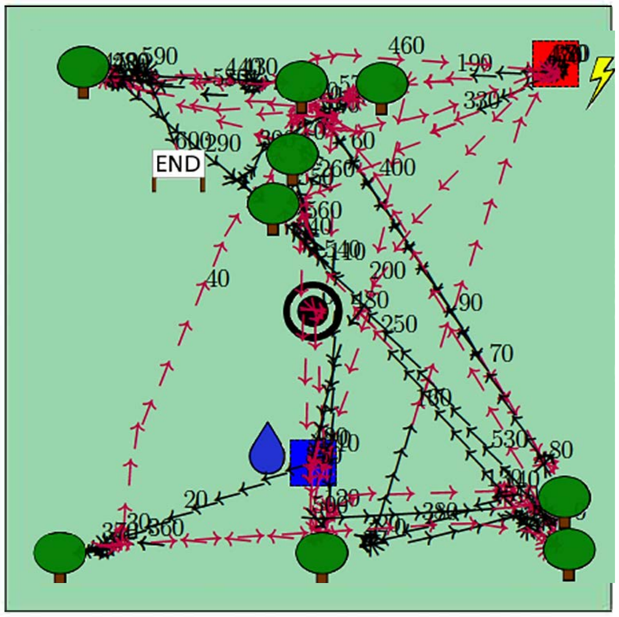

Figure 2: Map of the Firefighter Robot mission on which appear data coming from a mission carried out on the website and acquired by crowdsourcing. The target symbol in the center is the initial location of the robot and the black (resp. red) arrows are the successive robot's locations when the robot is autonomous (resp. in manual mode). The current time steps of the mission are the numbers displayed at the current location of the robot. Trees and locations of squares are also displayed, as well as the terminal robot's location with an "End" flag.

3 FORMAL MODEL

In this work, and as often in the context of human-robot interaction, the evolution of the system cannot be considered as deterministic. Indeed, human behavior and environmental dynamics are uncertain and therefore require adequate modeling. The Markov Decision Process (MDP) framework [25] is a convenient choice for planning under uncertainty. This famous stochastic control process is an elegant way to model and solve probabilistic planning problems. Once the possible actions and system states have been identified, the goal of the problem is defined using a reward function that evaluates the utility of a state-action pair. This makes possible to define the utility of an action sequence as the expected sum of the rewards obtained over time given an initial state. The optimal sequence of actions is the one that maximizes such an expected sum of rewards.

Formally, the (finite horizon) MDP model is defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, H)$, where:

- $\mathcal{S}$ is the finite set of states;
- $\mathcal{A}$ is the finite set of actions;
- $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is the transition function, which defines the probability $p(s' \mid s, a) = T(s, a, s')$ of reaching the state $s' \in \mathcal{S}$ given that the action $a \in \mathcal{A}$ is performed in state $s \in \mathcal{S}$;

- $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function that values any state-action pair;
- $H \in \mathbb{N}$ is the horizon, that is the duration of the process in terms of discrete time steps[3].

A policy $\pi \in \Pi$ is a function that associates an action $a \in \mathcal{A}$ to each possible context, that is, in the case of finite horizon MDPs ($H < +\infty$), to each state $s \in \mathcal{S}$ and time step $t \in \{0, \ldots, H\}$: $\pi : \mathcal{S} \times \{0, \ldots, H-1\} \to \mathcal{A}$. Solving an MDP is finding a policy that maximizes the expected amount of rewards up to time step H:

$$\pi^* = \underset{\pi \in \Pi}{\operatorname{argmax}} \, \mathbb{E}\left[\sum_{t=0}^{H-1} R\big(s, \pi(s,t)\big) \,\middle|\, s_0 = s \right]. \tag{1}$$

Such a policy defines the optimal action $a \in \mathcal{A}$ to perform in a given state $s \in \mathcal{S}$ and time step $t \in \{0, \ldots, H-1\}$, and can be computed by *Dynamic Programming* [2] making use of the Bellman operator applied to the optimal value function V^* defined by induction as shown in Equation 2: the last optimal decision rule is $\pi^*(s, H-1) = \operatorname{argmax}_{a \in \mathcal{A}} R(s,a)$, $V_1^* = \max_{a \in \mathcal{A}} R(s,a)$, and $\forall h \in \{2, \ldots, H\}$,

$$\pi^*(s, H-h) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ R(s,a) + \sum_{s' \in \mathcal{S}} T(s,a,s') \cdot V_{h-1}^*(s') \right\}$$

$$V_h^*(s) = \max_{a \in \mathcal{A}} \left\{ R(s,a) + \sum_{s' \in \mathcal{S}} T(s,a,s') \cdot V_{h-1}^*(s') \right\}. \tag{2}$$

Recent MDP algorithms [4, 16] explore dynamic properties of the model (mainly using Monte-Carlo methods) to optimize actions only for reachable states. Such methods help to decrease the time and memory needed to solve MDPs with large state space.

In the case of the Firefighter Robot mission optimization, the actions to choose over time are the supervision actions, *i.e.* robot mode $a^m \in \{0,1\}$ and alarm display $a^a \in \{i\}_{i=-1}^{6}$ (see Table 1). As a result, the action space is $\mathcal{A} = \{0,1\} \times \{i\}_{i=-1}^{6}$. The remaining time is linked to the time step and the horizon by the equality $r_t = H - t$ with $H = 600$. The remaining values in m_t, *i.e.* after removing a_t^m, a_t^a and r_t, constitute the current state $s_t \in \mathcal{S}$.

Note now that the state space defined in this way is not finite as imposed in the definition. For instance the robot pose (x_t, y_t, θ_t), the temperature T_t and the water level of the ground tank w_t^g are three continuous variables. The battery level b_t, the state of the forest f_t and the leaks on the ground tank l_t are not continuous data, but they have a large number of possible values (100, 2^9 and 2^9 respectively). Since the operator can interact at any speed via keyboard and mouse, if the connection is good, the variables encoding the keystroke sequence k_t, the click sequence c_t and the error sequence e_t can be very long word sequences, and so the number of possible word combinations is also very large. The number of keyboard shortcuts σ_t can also be very large for the same reason. Only the water level w_t^r in the robot tank has a limited number of possible values which is equal to $\#w_t^r = 11$.

It is therefore necessary to discretize these state variables in order to be able to treat the problem with the desired tools. In addition, the discretization must be coarse enough to keep the state space small so that the learning and planning algorithms can solve the problem without too much memory or computation time.

[3]Note that in the most general definition, T and R could also depend on time t.

In this study, the learning process consists in the estimation of the transition probability values, *i.e.* the transition function T, that are still missing to fully define the MDP. Crowdsourcing data will be used to estimate this function, but here again, a rough discretization allows for the learning algorithms to give more reliable estimates.

3.1 Variable Selection and Trade-off in Granularity

In this work we make the hypothesis that a rough discretization can allow a better estimation of the transition function T. Indeed such a processing increases the number of occurrence - in the database resulting from crowdsourcing - of the states thus defined. In this way the learning is based on a larger number of samples which should improve the accuracy of T.

Moreover the curse of dimensionality [3] prevents us from starting our supervision study of human-robot team with too many variables otherwise the learning and planning problems will not be practically solvable. Hence, this sub-section is dedicated to the description of the MDP actually learned and solved for this first study on the Firefighter Robot dataset.

The selected state variables are limited to discretized versions of x_t, y_t, θ_t, b_t, w_t^r, f_t, and k_t. These variables seem to be the most relevant to take into account in order to optimize supervision to drive the human-robot system. Indeed, the state of the forest is the state of the system on which the goal of the mission depends. The robot's pose and water level are system's states that are quite directly related to fire extinction, so they were also chosen. The level of battery can lead to the end of a game, and therefore a sub-optimal mission given the chosen reward function. Finally, the sequence of keyboard keys used is a source of information about the operator that must be taken into account to allow the supervisory system to adapt to human behavior.

The robot positions are discretized according to a grid 3×3 whose cells have the same size: $pos \in \{NE, N, NW, E, C, W, SE, S, SW\}$. Thresholds are defined for the battery and water level to discretization into two values: $(bat, wat) \in \{\text{"nominal"}, \text{"low"}\}^2$. The total description of the condition of the trees requires a variable with 2^9 possible values. We therefore keep only the number of fires in progress $fire = \sum_{i=1}^{9} f_t^i \in \{0, \ldots, 9\}$. A variable *space* is also introduced to encode if the user presses the space key during the current second: $space = \mathbb{1}_{\{\text{"space"} \in k_t\}} \in \{0,1\}$. The variable $end = \mathbb{1}_{\{b_t = 0\}} \in \{0,1\}$ symbolizing game overs is also introduced. The state variables necessary to define the reward function we had in mind have already been introduced. Since the goal of the human machine team is to keep the trees in good condition, the reward function can be defined as: $R(s,a) = R(s) = R(end, fire) = (1 - end) \cdot (9 - fire)$. Note that, the reward function depends only on *end* (no more reward if the mission fails) and on $fire$, but it does not depend on the action $a \in \mathcal{A}$.

The following section presents a technique for discretizing the remaining variables, namely the sequence of pushed keys k_t and the orientation θ_t. This leads to a variable called *int* (for "human's intention") and taking nine possible values. We thus have seven state variables $s_t = (pos, wat, bat, fire, space, end, int) \in \mathcal{S}$ forming a state space of size $\#\mathcal{S} = 9 \times 2 \times 2 \times 10 \times 2 \times 2 \times 9 = 12960$. In this way, the size of the state space will not prevent the problem from beeing solved, *i.e.* optimized strategy computation using a state-of-the-art MDP solver will be possible.

3.2 Interaction Model Learning

Although the discretization of the rotation θ_t would be easy to implement, the discretization of the keyboard key sequences k_t is less direct. There is no intuitive and simple partition to make. While there are many clustering algorithms for continuous and numerical data [19, 32], the literature on clustering of less structured data is quite poor. Thanks to the technique presented in this section, these two variables will in fact be used to estimate the movement that the operator wants (intends) the robot to perform.

Knowledge of the human operator's intentions can only be beneficial to a supervision system of a human-machine team. For example, suppose that the robot is in autonomous mode in an area where it generally has more difficulty moving than when the human controls it. If the operator presses the directional keys and seems to want to move the robot in a satisfactory direction, a good supervision system would change the robot mode from autonomous to manual. Remember that the robot's hard-coded strategy when in autonomous mode is not optimal, so the human operator could also observe that a better strategy is possible and thus manipulate the keyboard keys to regain control over the robot. On the other hand, if the supervision system detects that the operator's intention is to move in a direction that is suboptimal for the mission, for example to move away from the charging area when the battery is low, it would be optimal by switching to autonomous mode.

Thus, the variables θ_t and k_t will be discretized into the operator's intentions about the robot's movements. To do this, we will use a sub-dataset from our crowdsourcing database: the robot poses (x_t, y_t, θ_t) and keyboard key sequences k_t during the time steps when the robot is in manual mode. In this mode, the keyboard keys have an effect on the robot's movement. Apart from possible transmission delays, packet losses, obstacles on the way and the shape of the robot's velocity, it seems possible to reliably predict the robot's movement from its orientation and the sequence of directional keys used. Since, in this mode, most of the effects on the robot are those desired by the operator, such predictions provide us with a function of θ_t and k_t, having as a value a probable intention of the user. It is then sufficient to discretize the movements of the robot to be predicted to obtain a discretization of the couple (θ_t, k_t). Indeed the latter will be replaced by its prediction. For instance, even when it has no effect on the robot (*e.g.* obstacle or autonomous mode), a sequence of "front" keys will be replaced by the operator's intention to move forward.

In this work we consider the changes in the robot's position: $(x_{t+1} - x_t, y_{t+1} - y_t) \in \mathbb{R}^2$. These motions are simply discretized into the following values:

$$mot \in \mathcal{L} = \{\text{"NoMot"}, \text{"N"}, \text{"S"}, \text{"O"}, \text{"E"}, \text{"NE"}, \text{"SE"}, \text{"SO"}, \text{"NO"}\},$$

i.e. the fact the robot does not move much, the cardinal and the inter-cardinal directions. These values will be used as labels to learn the prediction function with a supervised classification algorithm. Data concerning the keyboard keys k_t are provided to the classifier as follows. For each $key \in \{\text{"front"}, \text{"back"}, \text{"left"}, \text{"right"}\}$ and for each line of the sub-dataset (corresponding of a time step t of a mission), a truncated number of occurrences of the key in the sequence is computed: $key_t^{oc} = \min(\sum_{key \in k_t} 1, 10)$. The input data of the classifier are then $(front_t^{oc}, back_t^{oc}, left_t^{oc}, right_t^{oc}, \theta_t) \in \{0, \ldots, 10\}^4 \times [-\pi, \pi]$: these values will be used to predict the user's intent.

Using Gradient Boosting algorithm [5, 11] available in scikit-learn library [24] the resulting classifier $c : \{0, \ldots, 10\}^4 \times [-\pi, \pi] \rightarrow \mathcal{L}$ reached an accuracy of 87.5% to predict the motion associated to the robot's orientation and the keystrokes sequence. Using the prediction $int = c(k_t, \theta_t) \in \mathcal{L}$ of this classifier instead of the couple (k_t, θ_t) allows to consider directly the intention of the human operator at the level of the MDP state s_t, while benefiting from a coarse discretization of this couple of variables. The partition defining the discretization of (k_t, θ_t) is $\left\{ c^{-1}(\{l\}) \right\}_{l \in \mathcal{L}}$, that is $\#\mathcal{L} = 9$ subsets.

4 LEARNING DYNAMICS AND COMPUTING AN OPTIMAL STRATEGY

Now that the choice of variables and their discretization has been implemented, it remains to define the transition function from the crowdsourcing data *i.e.* for each current state-action pair $(s, a) \in \mathcal{S} \times \mathcal{A}$, and for each next state $s' \in \mathcal{S}$, the probability value of such a transition: $T(s, a, s') \in [0, 1]$ with $\sum_{s' \in \mathcal{S}} T(s, a, s') = 1$. Indeed, once this function T is estimated, the desired MDP is fully defined and a planning algorithm can be used to optimize the supervision strategy. The next subsection deals with learning, while the next one deals with planning.

4.1 Independence Assumptions and Markov Transition Learning

We used the *Pomegranate*[4] library [30] to learn the parameters of the MDP based on the seven state variables described above. In fact this library is meant to learn the transition matrix of Markov Chains (MCs) by using maximum likelihood estimates. The trick here is that when subjected to a constant strategy, an MDP becomes a Markov Chain. In this paper, we propose to build a strategy to decide only on the robot mode $a_t^m \in \{0, 1\}$. Future work will deal with alarms $a_t^a \in \{i\}_{i=-1}^{6}$. Two classes of sample transitions should then be differentiated. Those starting from manual mode ($a_t^m = 0$), and those starting from autonomous mode ($a_t^m = 1$). They could be separately given as input to the MC parameters learning algorithm which should return two transition matrices: one is $[T(s, 0, s')]_{(s,s') \in \mathcal{S}^2}$ and the other $[T(s, 1, s')]_{(s,s') \in \mathcal{S}^2}$, both of size 12960^2.

Unfortunately, this number of system states prevents the learning algorithm of the Pomegranate library from being used directly, and it is necessary to trick once again to compute this transition function. The trick here consists in computing transition matrices on a sub-group of variables. This method is made possible by first assuming that the variables at time step $t + 1$ are independent of each other conditionally to the variables in step t (*i.e.* conditionally to the past since the Markov property is already assumed), regardless of the action chosen before the transition. An MDP whose variables have this independence property is called a *factored MDP*

[4]https://pomegranate.readthedocs.io

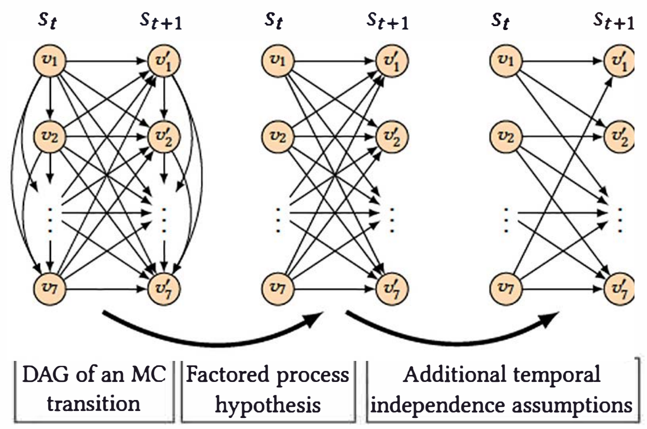
s_t
s_{t+1}
v_1
v_2
v_7
v'_1
v'_2
v'_7
DAG of an MC transition
Factored process hypothesis
Additional temporal independence assumptions

Figure 3: Without any assumption of independence, the Directed Acyclic Graph (DAG) defining the Bayesian network representing the variables during a transition is complete (left). Assuming that the variables of the same time step are independent of each other, conditional on the past, the DAG becomes a complete bipartite graph (center). Finally, other independence hypotheses also make it possible to delete some arrows (right).

[13, 14]. Note that this formalism is used in the *International Probabilistic Planning Competition*[5] (IPPC), using the *Relational Dynamic Influence Diagram language* (RDDL [29]) that we'll use in the next subsection for strategy optimization. Complying with this framework is not a very strong assumption. Indeed, if the initial MDP has n variables, it is always possible to subdivide the time steps into n sub-steps during which only one variable makes its transition while the others remain constant: thus, by considering the MDP equipped with these new intermediate time steps, the variables are indeed independent conditional on the past. In our case we simply assume that the variables are conditionally independent, without subdividing the time steps.

More formally, let us denote the seven considered variables by $(v_t^i)_{i=1}^7$, *i.e.* $s_t = (v_t^1, \ldots, v_t^7)$. The independence properties assumed in the factored framework implies that it exists seven functions $(T_i)_{i=1}^7$, one for each variable, such that $T(s_t, a_t, s_{t+1}) = \prod_{i=1}^7 T_i(s_t, a_t, v_{t+1}^i)$. The function T_i is the transition function of variable v'_i. Thus, by representing uncertainty dynamics with Directed Acyclic Graph (DAG) [23], the resulting graph is bipartite and complete as shown in Figure 3, such that the two sets of nodes constituting the bipartition of the bipartite graph are the set of variables at time t and the set of variables at time $t+1$. In addition to enabling a more efficient optimization of the MDP strategy in different uncertainty models [7, 10, 14] (by decreasing the time spent to compute the optimal policy), this characteristic will enable the learning algorithm to compute an estimation of the transition function T. For this purpose, however, it is necessary to identify additional variable independence, this time in temporal terms.

The transition probability of each variable does not depend on all previous variables. For example, the battery level depends on the previous battery level, but not on the previous water level (and vice versa). In other words, conditional on the other variables of the same time step t, the battery level at time t is independent of the water level at time $t+1$. By introducing this kind of expert knowledge on the independences of variables, some arrows can be removed from the bipartite graph representing the transition uncertainty. Figure 3 illustrates also this last pruning operation. More concretely, for each transition function T_i (associated with variable v_i), there is a subset of variables $W_i \subset \{v_1, v_2, \ldots, v_7\}$ such that $T_i(v_1, v_2, \ldots, v_7, v'_i) = T_i(W_i, v'_i)$, where primed variables represent next variable while unprimed ones stand for current variables. In the resulting DAG, this subset W_i is the set of parents of v'_i. The idea used here is that the calculation of the transition function T_i only requires transition samples concerning only the variables in W_i, so the learning algorithm is run several times ($\forall i \in \{1, \ldots, 7\}$), but on fewer variables (because $\#W_i < 7$) and therefore with dimensions that can be handled. Note that the learning algorithm returns the values of $T(W_i, a, W'_i) = p\left(W'_i \,\middle|\, W_i, a\right), \forall a \in \mathcal{A}$, so the transition function of variable v'_i can be computed by summing over variables $Z_i = W_i \setminus \{v_i\}$:

$$T_i(W_i, a, v'_i) = \sum_{Z'_i} T_i(W_i, a, W'_i) = \sum_{Z'_i} p\left(v'_i, Z'_i \,\middle|\, W_i, a\right).$$

4.2 Strategy Optimization and Simulation Results

The MDP that is finally obtained is illustrated in the form of a Dynamic Bayesian Network (DBN) [21] in Figure 4. This is a minimal

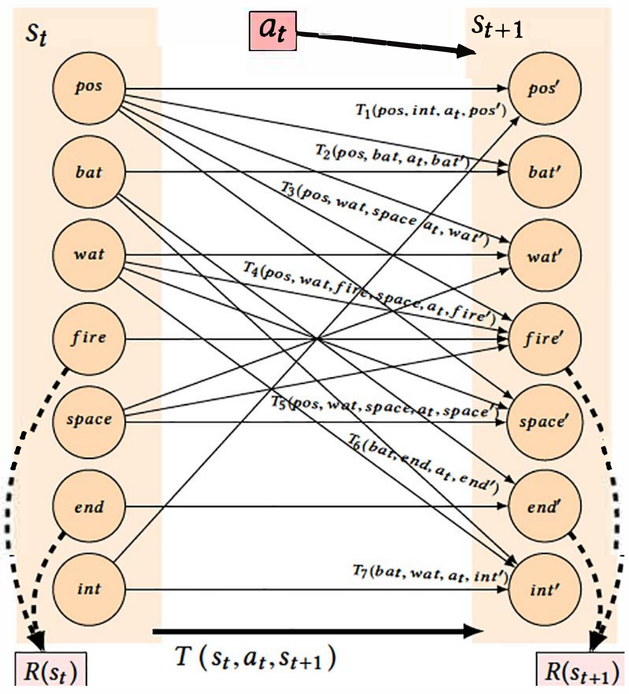
s_t
a_t
s_{t+1}
pos
bat
wat
fire
space
end
int
pos'
bat'
wat'
fire'
space'
end'
int'
$T_1(pos, int, a_t, pos')$
$T_2(pos, bat, a_t, bat')$
$T_3(pos, wat, space, a_t, wat')$
$T_4(pos, wat, fire, space, a_t, fire')$
$T_5(pos, wat, space, a_t, space')$
$T_6(bat, end, a_t, end')$
$T_7(bat, wat, a_t, int')$
$T(s_t, a_t, s_{t+1})$
$R(s_t)$
$R(s_{t+1})$

Figure 4: The DBN of the learned MDP for the Firefighter Robot game. Primed (resp. unprimed) variables represent next (resp. current) variables.

[5] https://ipc2018-probabilistic.bitbucket.io/

modeling choice while remaining understandable for human being and tractable considering learning and policy optimization. In order to evaluate our model, we simulated the process thus defined in order to estimate the expected total reward, *i.e.* the criterion that we want to maximize (see Equation 1). To do this, the problem was written in RDDL format using the transition functions $(T_i)_{i=1}^{7}$ learned previously.

These simulations were carried out under four different conditions. In each condition, 300 simulations were performed: with a random strategy (at each time step t, $p(a_t = 0) = p(a_t = 1) = 0.5$), with a constant manual strategy ($\forall t,\ a_t = 0$), with a constant autonomous strategy ($\forall t,\ a_t = 1$) and with an online strategy optimization provided by *PROST*[6] [16]. This algorithm implements *Trial-based Heuristic Tree Search* (THTS) [17] with the settings of IPPC 2014. The action selection of the solver UCB1 is used, described by [1] for *Multi-armed Bandit problems* and used in the solver UCT [18]. It also uses the *Partial Bellman* backup function [17] and samples unsolved outcomes using its probability. Finally, the results obtained during these simulations are compared with those obtained using data collected via the crowdsourcing platform, in which the policy is random. More precisely, the average of total reward was computed with 678 missions that players have not left voluntarily (by leaving the interface) but performed entirely or failed because of a game over.

The results (expectation of the total reward) are illustrated in Figure 5, with the standard errors of the data coming from the 300 simulations. The estimation of the expected total reward is 1409.08 when crowdsourcing data is used. This value is close to that obtained by simulating the mission with a random policy (1465.52), which is reassuring as to the adequacy of reality with the MDP learned. The strategy of setting the robot mode to manual mode at all times leads to the lowest performance in terms of average total reward on simulations (1206.29). Simulations with a constantly autonomous robot give a much higher average total reward (1693.73), which is not contradictory to reality: the mission being multi-task, the operator cannot control the robot all the time. Thus, an autonomous robot is more beneficial for the mission than a manual robot. Finally, the optimized online strategy provides slightly better results (1803.56). Note that this strategy is not the optimal strategy, and that offline (and time consuming) resolution can provide a much higher total reward.

These initial results confirm us in the idea that a supervision strategy, based on the states of the human-robot system and calculated using a decision model under uncertainty, can be used and improved in order to obtain better team performance.

5 CONCLUSION AND FUTURE WORK

Instead of defining a probabilistic model with expert or arbitrary parameters, this paper proposes a methodology to learn an human-agent interaction model from crowdsourcing data collection. Our probabilistic interaction model takes into account human actions on the keyboard and random environment changes using the transition function T of an MDP defined thanks to machine learning techniques. A reliable T function should enable MDP solvers to

[6]https://bitbucket.org/tkeller/prost/wiki/Home

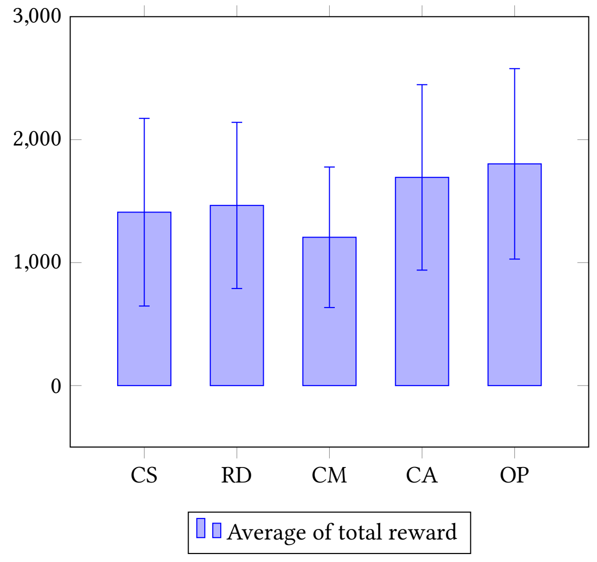

Figure 5: Estimation of expected total reward $\mathbb{E}\left[\sum_{t=0}^{H} R(s_t)\right]$. On the left (CS), the average on the crowdsourcing data. Other values are the average of rewards computed based on 300 simulations of the MDP with a random policy (RD), a constant manual policy (CM), a constant autonomous policy (CA) and a policy optimized with an MDP solver (OP).

optimize the supervision policy in accordance with the real human-robot interaction and mission considered.

Offline optimization of the strategy will be performed in future work with state-of-the-art algorithms [4, 16] to achieve near optimal solutions for such an MDP model. The policy computed will be deployed in laboratory conditions and on the project's website for an evaluation of the overall system's performance. Moreover, if we consider that the difficulty of the mission leads the human operator into degraded mental states (mental fatigue, working memory load [27], attentional tunneling [26], etc.) the estimation of the operator's mental state could enrich the representation of the system described by the MDP. Clearly, a human mental state is not a fully observable state variable. In this case, the use of Partially Observable Markov Decision Process [34] could be a promise avenue to model such a human-agent interaction problem.

Finally, one could explore frameworks able to provide a generative model [12, 28], that is a simulator, based on collected data described in this paper. The development of such a simulator could be explored in order to apply Reinforcement Learning techniques [20, 35] to learn the supervisory policy based on simulated missions. Even more, automated discretization, as clustering algorithms, could be used to infer a possible smarter discretization while keeping a sufficiently low number of clusters to ensure reliable transition function estimates.

ACKNOWLEDGMENTS

This work was supported by a chair grant from Dassault Aviation (CASAC).

REFERENCES

[1] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.

[2] Richard Bellman. 1954. The theory of dynamic programming. *Bull. Amer. Math. Soc.* 60, 6 (11 1954), 503–515. http://projecteuclid.org/euclid.bams/1183519147

[3] Richard Ernest Bellman. [n. d.]. Rand Corporation (1957). *Dynamic programming* ([n. d.]).

[4] Blai Bonet and Hector Geffner. 2012. Action Selection for MDPs: Anytime AO* Versus UCT.. In *AAAI*.

[5] Leo Breiman. 1997. *Arcing the edge*. Technical Report. Technical Report 486, Statistics Department, University of California at Berkeley.

[6] J Raymond Comstock Jr and Ruth J Arnegard. 1992. The multi-attribute task battery for human operator workload and strategic behavior research. (1992).

[7] Karina Valdivia Delgado, Scott Sanner, Leliane Nunes De Barros, Fábio Gagliardi Cozman, et al. 2011. Efficient solutions to factored MDPs with imprecise transition probabilities. *Artificial Intelligence* 175, 9-10 (2011), 1498–1527.

[8] Nicolas Drougard, Caroline Ponzoni Carvalho Chanel, Raphaëlle N Roy, and Frédéric Dehais. 2017. Mixed-initiative mission planning considering human operator state estimation based on physiological sensors. In *IROS Workshop on Human-Robot Interaction in Collaborative Manufacturing Environments (HRI-CME)*.

[9] Nicolas Drougard, Raphaëlle N Roy, Sébastien Scannella, Frédéric Dehais, and Caroline Ponzoni Carvalho Chanel. 2018. Physiological Assessment of Engagement during HRI: Impact of Manual vs Automatic Mode. In *2nd International Neuroergonomics Conference*.

[10] Nicolas Drougard, Florent Teichteil-Königsbuch, Jean-Loup Farges, and Didier Dubois. 2014. Structured Possibilistic Planning Using Decision Diagrams.. In *AAAI*. 2257–2263.

[11] Jerome H Friedman. 2002. Stochastic gradient boosting. *Computational Statistics & Data Analysis* 38, 4 (2002), 367–378.

[12] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.

[13] Carlos Guestrin, Milos Hauskrecht, and Branislav Kveton. 2004. Solving factored MDPs with continuous and discrete variables. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 235–242.

[14] Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. 1999. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 279–288.

[15] Shu Jiang and Ronald C Arkin. 2015. Mixed-Initiative Human-Robot Interaction: Definition, Taxonomy, and Survey. In *Systems, Man, and Cybernetics (SMC), 2015 IEEE International Conference on*. IEEE, 954–961.

[16] Thomas Keller and Patrick Eyerich. 2012. PROST: Probabilistic Planning Based on UCT.. In *ICAPS*.

[17] Thomas Keller and Malte Helmert. 2013. Trial-Based Heuristic Tree Search for Finite Horizon MDPs.. In *ICAPS*.

[18] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *ECML*, Vol. 6. Springer, 282–293.

[19] Stuart Lloyd. 1982. Least squares quantization in PCM. *IEEE transactions on information theory* 28, 2 (1982), 129–137.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[21] Kevin Patrick Murphy and Stuart Russell. 2002. Dynamic bayesian networks: representation, inference and learning. (2002).

[22] William D Nothwang, Michael J McCourt, Ryan M Robinson, Samuel A Burden, and J Willard Curtis. 2016. The human should be part of the control loop?. In *Resilience Week (RWS), 2016*. IEEE, 214–220.

[23] Judea Pearl. 2014. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Elsevier.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[25] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[26] Nicolas Régis, Frédéric Dehais, Emmanuel Rachelson, Charles Thooris, Sergio Pizziol, Mickaël Causse, and Catherine Tessier. 2014. Formal detection of attentional tunneling in human operator–automation interactions. *IEEE Transactions on Human-Machine Systems* 44, 3 (2014), 326–336.

[27] Raphaëlle N Roy, Stephane Bonnet, Sylvie Charbonnier, and Aurélie Campagne. 2013. Mental fatigue and working memory load estimation: interaction and implications for EEG-based passive BCI. In *Engineering in Medicine and Biology Society (EMBC), 2013 35th Annual International Conference of the IEEE*. IEEE, 6607–6610.

[28] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. 2016. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*. 2234–2242.

[29] Scott Sanner. 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished ms. Australian National University* (2010), 32.

[30] Jacob Schreiber. 2018. Pomegranate: fast and flexible probabilistic modeling in python. *Journal of Machine Learning Research* 18, 164 (2018), 1–6.

[31] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).

[32] Jianbo Shi and Jitendra Malik. 2000. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence* 22, 8 (2000), 888–905.

[33] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[34] Richard D. Smallwood and Edward J. Sondik. 1973. *The Optimal Control of Partially Observable Markov Processes Over a Finite Horizon*. Vol. 21. INFORMS. 1071–1088 pages.

[35] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[36] Kevin W. Williams. [n. d.]. A Summary of Unmanned Aircraft Accident/Incident Data: Human Factors Implications. *U.S. Department of Transportation, Federal Aviation Administration, Civil Aerospace Medical Institute* ([n. d.]).