Building Markov Decision Process Based Models of Remote Experimental Setups for State Evaluation

Ananda Maiti, Alexander A. Kist, Andrew D Maxwell
School of Mechanical and Electrical Engineering, University of Southern Queensland, Australia
anandamaiti@live.com, kist@ieee.org, andrew.maxwell@usq.edu.au

***Abstract*—Remote Access Laboratories (RAL) are online environments that allows the users to interact with instruments through the Internet. RALs are governed by a Remote Laboratory Management System (RLMS) that provides the specific control technology and control policies with regards to an experiment and the corresponding hardware. Normally, in a centralized RAL these control strategies and policies are created by the experiment providers in the RLMS. In a distributed Peer-to-Peer RAL scenario, individual users designing their own rigs and are incapable of producing and enforcing the control policies to ensure safe and stable use of the experimental rigs. Thus the experiment controllers in such a scenario have to be smart enough to learn and enforce those policies. This paper discusses a method to create Markov's Decision Process from the user's interactions with the experimental rig and use it to ensure stability as well as support other users by evaluating the current state of the rig in their experimental session.**

I. Introduction

Markov Decision Process (MDP) are models to represent stochastic processes and have been applied in many fields to model partly random decision processes. The Stochastic Shortest Path MDP [1] or SSP MDP is a particular version of the MDP that specifies a set of *goal* states that must be reached from any other state. For a system modeled by MDPs, there is a decision maker or agent which decides what to do in the system. The agent is provided with a plan or policy that gives the best chances with minimal cost or delays to succeed in reaching certain *goal* states.

Remote Access Laboratories (RAL) are online environments where instruments or experimental rigs are shared with users through the Internet [4]. The experiment parameters are changed by the learner and the rig responds by altering its configuration or state. From an MDP perspective, the *human learner's* input is the random factor and they act as the *agents*. For each experiment, there is a Controller Unit (CU) that follows predefined control policies created by the *experiment provider* to operate the experimental setup (see Fig. 1) to guarantee stability and reliability. However, in some scenarios such as in Peer-to-Peer RAL [3], it is difficult for *makers,* or the experiment providers to create specific control policies by themselves. In P2P RAL, Micro-Controller Units (MCU) such as Arduino, LEGO etc. are used which have common characteristics and provide a generalized control platform [3]. Each experiment has definite *goals* to achieve and unless proper guidelines are set by the *makers*, the learners may be unaware on how to operate the rig to get to those *goal* states.

In this paper, a method to generate these control policies automatically from makers' usage of MCU based rigs is presented. To do this, a variant of the SSP MDP of the experimental rig is created from the training data from the makers interactions with it. The MDP defines the perimeters of admissible states of the rig's state space. This allows for the CUs to check the transitions of the state and blocks any command that leads it to an *unknown* or *undesired* state. The MDP also generates proper policies that indicate best ways to change the states to reach a desired *goal* state. Three indicators based on these policies to evaluate the current state of the rig with respect to the overall operation of it are also presented. Finally, the MDPs, policies and indicators are used to create a system that is able determine if the learner needs any or what guidance to perform the experiment i.e. a support system.

The rest of the paper is organized as follows: Section II describes the role of MDPs and its use in control and automation as well as describing the P2P RAL and its dependence on automation. Section III introduces state space, characteristics and models of an experimental rigs as well as user and system related constraints. Section IV presents the smart aspects of the Controller Units based on MDPs in order to evaluate the state transitions and detect unstable conditions in the experimental rig. The use of the MDP in RAL environment and an example of a rig and its application of the smart paradigms are discussed in Section

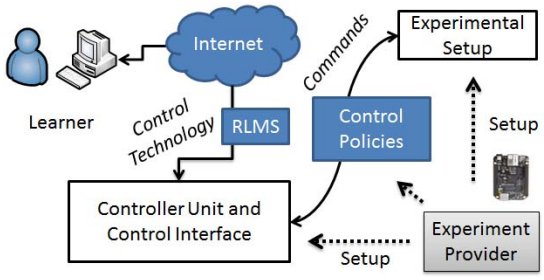

Fig. 1. The system architecture of RAL experiment.

This work is supported through the Australian Governments CRN Digital Futures program

978-1-4799-7560-0/15 $31.00 © 2015 IEEE
DOI 10.1109/SSCI.2015.65

V. The strengths and limitations of this approach are discussed in Section VII.

II. Related Work

A. Markov Decision Processes and Control

Markov decision processes are used to model systems that maintain the *memoryless* properties i.e. choosing a new system state solely based on the current state and the corresponding action chosen. However, there are some approaches that store the past information into the current state of the system and carry it forward [11]. The next best state and the corresponding action are chosen based on a prescribed policy (π) that maps each state to one action. Again the policy may not be static for every time step of the system [12] and it may be updated with variable rewards within the system. MDPs are used in artificial intelligence [2] to model and create decision-based support systems. It provides a framework to model complex problems that have large state-spaces and complex cost functions. It also provides a model to further develop learning algorithms to aid reinforcement learning corresponding to the system. There are also some well-known and efficient methods to solve MDPs such as the Value Iteration Algorithm (VIA) [8].

In [5] the MDP is used to model the aircraft's movements and autonomously avoid collision. The performances of different types of sensors are evaluated in the model. But the aircraft state vector is fixed to the properties of an aircraft, thus limited to a specific application. In the RAL scenario, the MCUs provide a generic platform to create variable experiments. Thus a method is required that can generate a MDP for any rig using the MCU based architecture [3]. Similarly another particular application to guide people with dementia is reported in [6]. It exploits the MDP's implicit capability to manage stochastic dynamics and capturing the trade-offs between multiple objectives. All of these applications are capable of handling high dimensional data and large number of states, although the computational capacity required becomes a challenge on low-cost devices like MCUs.

B. RAL and Automation

In a centralized or federated RAL, the RLMS is managed from a select set of computer nodes. The entire RLMS is provided as a service by universities or institutions [10]. The RLMS governs the users' access aspects such as authorization of users, scheduling users when they access which rig and store any experiment setup data. The RLMS also implements the control policies that determine the exact manner of operations and the limits of parameter both inputs and outputs. The control policy differs between experimental setups depending upon the components used and their configuration. The control policies' main aims are:

- to make sure that the rig is always in a stable state by blocking or rejecting any inputs that is not within the allowed range of parameters.
- Attempt recovery, re-boot of the experimental rig, if possible and inform the RLMS administrators about any unstable state that is persistent and cannot be rectified without human intervention.

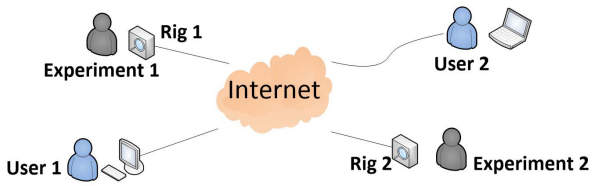

Fig. 2. The P2P RAL architecture.

Remote Access Laboratories being remote in nature must have some form of automation the experimental rigs to help guide the experiment run without the assistance of human. The automations involve a mechanical or electro-mechanical device that re-configures the experiment rigs as the current learner wants. The RLMS stores the control-interface (CI) created by the *provider*, which collects inputs for operating these rigs. The control policies regarding the inputs are hard-encoded into the control program for e.g. the definite limits of an actuator are specified in the program. This requires deep understanding of programming to create such policies. Also, it is more difficult to alter these policies in case the hardware is changed.

In a P2P RAL, individual makers are given the opportunity to create a rig and it's CI. The maker, based upon their knowledge of any particular experimental activity creates and shares an experimental setup (see Fig. 2). This setup or rig is based upon a MCU [3]. The MCU can be modelled as a special automaton that contains an instruction processing capabilities, memory to store instructions and related contents. It also has a set of ports to control or interact with the experiments environments or objects through actuators and sensors.

The MCUs gives enough flexibility to create an experimental rig along with necessary automation. Makers can connect any sensor and actuator to the ports and create the CI using a visual programming language on an online platform [9]. However, providing this flexibility comes at the cost of lower reliability of the components used in the rig. Also, the makers of the experiments have disparate backgrounds and knowledge about control and automation. Thus the experimental rigs and CI created by them are less reliable both in terms of the control policy implemented as well as the actual equipment are not guaranteed to perform accurately for a lengthy period of time especially without human supervision.

Every experiment will have a set of tasks that signifies the completion of the experiment successfully. This means the experimental rig must go through the particular states of its state space within the experiment session. These set(s) of state may be called goal states. In the P2P RAL system, it is difficult identify whether the learner have met those goals. However, the makers are able to run their rigs as they desire. This way they can train the experimental rigs to accept valid inputs only.

In this paper, the MCU based experimental rigs for P2PRAL is described as an MDP which is then used to filter out inputs that are potentially incorrect and recover from any erroneous state as well as predict the next proper steps towards the goal of the experiment. This increases the

reliability of the components and the rig as whole. It also keeps the human intervention out of the way as much as possible.

III. THE P2P RAL CHALLENGES

This section discusses the properties and constraints of the P2P RAL system.

A. *The state space of the rigs*

The state space of an experimental rig is dependent on the status of the ports. Each port signifies a variable in the experimental rig whether it is connected to a sensor or actuator. The change in the state space is caused by a change in the value of any port i.e.

$$Y_{i+1} = AY_i + BC_i \quad \ldots (1)$$

where Y represents the controlled system or the rig, C_i is the command issued by the user or the agent and A and B are constant matrices for each experiment. In terms of simple control, only the actuators have any impact on state transition as this are the only components that can directly change the orientation of the experimental rig. But, from a decision and automation point of view, the state space contains the values of the all active ports on the MCU both sensors an actuators. The state space of the rig is infinite as each actuator in the rigs can have a value between -∞ and ∞. But in reality while training and running the rigs, it will only attain a finite set of states.

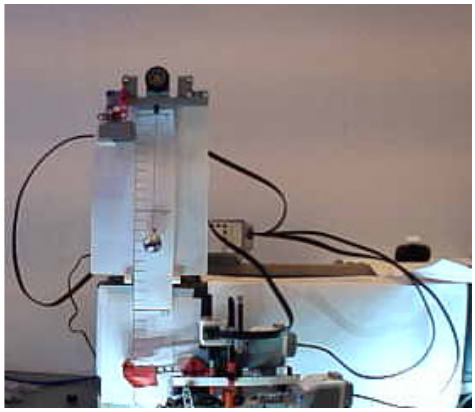

Fig. 3. (a) A pendulum experiment setup

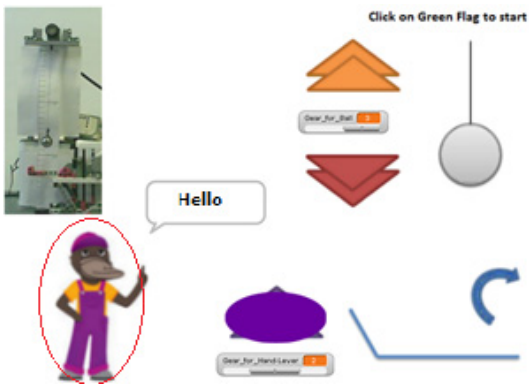

Fig 3 (b) The control interface of a RAL experiment in SCRATCH.

B. *The automaton based model*

The rigs and its controller can be represented as twin-finite state automata. This architecture consists of two sides - Control Interface (CI) and the CU. The CU acts upon the inputs from *CI* and the human users. The language between *CI* and *CU* is the communication protocol for the instrument control in the P2P RAL. This language consists of the very basic (or atomic) components of instrumentation -

- read (𝕣) - reading the value of a port (sensors and actuators)
- write (𝕨) - writing a value to a port (for actuators)
- wait (𝕒) - pause the CU for maintain synchronization

The program logic created by the maker, process the learner inputs for the UI to generate the corresponding symbol sets or communication commands composed of a combination of these three atomic instructions.

Fig 3(a) illustrates a typical example of a MCU based experimental rig. It is built using LEGO Mindstorms parts and based on a LEGO smart Brick as the MCU. This experiment demonstrates a pendulum with 3 actuators. The aim is to swing the pendulum and take measurements at different heights of the ball. Fig 3(b) is the corresponding web browser based CI with a number of buttons relating directly to an actuator on the rig. The users can view the outcome of the experiment via a web-camera stream as shown in Fig 3(a). There is also an animated character that can provide feedback and guidance (red circle).

C. *User and System Constaraints*

Makers have constraints in hard-encoding control policies in the CI as these require expert knowledge. They are however, capable of running the experimental rig with basic commands associated with a control interface. The systems constraints are:

1. The users and experimental rigs can be geographically located anywhere and interact with the rig via the Internet. Video feedback is used for viewing experiment outputs. As the system uses the Internet, control message between learner and experiment node are subject to delays. This means that there can be a chance that the learner may give an asynchronous wrong input depending upon what they perceive as the current state of the rig.

2. MCUs have limited computational capacity to process data per unit time [3].

3. The learners who interact with the experiment are provided with detail about the experiment and its goals. But, initially they will not be aware of the exact steps that need to be completed to achieve the experiment outcomes.

Also, as the P2P RAL is decentralized, there is no external entity to co-ordinate between the learner and the experiment rigs. This means that the experiment rigs have to be intelligent enough to create the control policies and avoid erroneous rig states. Thus creating the MDP for the rigs can help in:

1. Setting up the admissible boundary of the state space of the experimental rig. It can then always keep the rig in a *valid* state that can be obtained from the MDP. There can be two broad types of states:

- *Valid* state: a possible rig state that is stable and in the MDP, thus permissible. It is when the rig is not executing any command and the rig's parameters are not changing.
- *Error* state: a possible rig state that will break the rig and make it inactive, thus not permissible.

The valid states can be determined in the MDP. But *error* states are not possible to be identified as they are never recorded in the MDP. There can be two other types of states:

- *Interpolated* state: a possible rig state that is not in the MDP, but may be valid. Whether the state is *valid* or not may be determined by interpolating nearby *valid* state.
- *Undesired* states: A possible state that is not in the MDP, but cannot be validated in any way i.e. it may not break the rig, but not permissible either. An *undesired* state is essentially assumed to be an *error* state by the rig.

2. Predicting the next best step towards the immediate goals keeping in view any other goals depending solely upon the current state of the rig. The leaners' actions are matched against these best moves to *evaluate* their interaction and determine if any support is required for them.

IV. Smart Control and Automation in RAL CUs

This section discusses the MDP and the system state indicators that help with evaluating the system.

A. The experimental rigs as MDPs

The MDP is created from the state space of the experimental rig. An experimental rig may consist of multiple sensors (or actuators with feedback mechanisms) each of which is considered a variable in the state space. For actuators, it contains sensors to determine their current state. The conversion of the experimental rig's state space into MDP maintains a direct relationship between the MDP states and the rig's state space, i.e. rig's state that are positioned in the MDP adjacent to other states that precedes or succeeds them during the course of rig control. This can help in evaluating whether the transition in experimental rig's state is desirable.

Construction of the MDP requires a training data set containing sample input commands. The training set *X* has the set of makers' inputs during testing phase of the experiment,

$$X = \{ x_n^0,\ x_n^2,\ x_n^3\ \ \ldots\ x_n^o \}$$

where x^t_n is the state vector at time *t* of the experimental rig, *n* is the dimension of the feature vector i.e. the number of variables (or sensors) in the rig and o is a finite integer. The feature vector contains the values of all ports connected to a component, both sensors and actuators. Each x_n^t is a stable rig state when a command has finished executing and before the beginning of the next command execution. A command can be *composite* i.e. contain multiple instance of $\mathbb{r}, \mathbb{w}, \mathbb{a}$ to accomplice an action that requires to either maintain strict time intervals between multiple states transition from x_n^t to x_n^{t+1} (e.g. $\mathbb{w}_1\mathbb{a}\mathbb{w}_2$) or even change multiple ports in parallel in a single $\mathbb{w}$ command. However, each *composite* command can be broken down to its corresponding set of atomic instructions and executed in reverse order (e.g. $\mathbb{w}_2\mathbb{a}\mathbb{w}_1$) to get back to the previous state x^t_n.

Thus, the MDP for the rig system is defined as

$$Y = \{S, A, T, G, R_1, R_2 \ldots R_{|G|}\}$$

where,

S = E ∪ F. E contains all the valid states the rig can be in. F is a set of failed states corresponding to each transition t ∈ T for valid states. E represents a small subset of all possible states of the rig as most others would be error or undesired. Each element in E corresponds to one or more elements in X.

A = { w(P, V)} are the write commands that are issued by the user. This is the random factor in the MDP as the agent may choose to issue any command regardless of whether that is optimal or not. P is a set of port(s) the command works on and V are the values to be written.

T is a set of transition rules (or edges) that defines action allowed from a state, that would lead it to the next state(s) if the associated command is executed. t ∈ T also contains the probability of success of the transition i.e. $0 \leq T(s, s') \leq 1$.

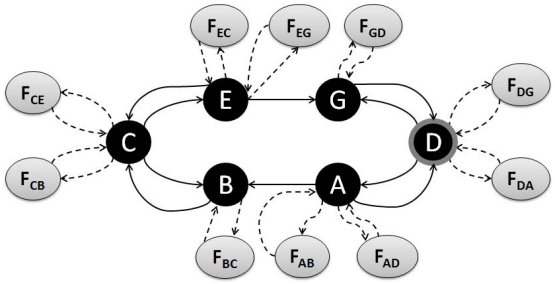

Fig. 4. Example of an MDP graph.

G is the set of goal states and G ⊂ S.

Ri is a set of rewards strategies corresponding to each goal state i ∈ G. Each reward strategy consists of a matrix of rewards for each transition t ∈ T.

The Fig 4 shows an example of the MDP where A, B, C, D, E, G are valid states, D is a goal state and F_{AB}, F_{AD}, F_{BC} ... are *failed* states corresponding to the actions of a valid states. The characteristics of the MDP graph are:

- There can be no self-loop in the MDP graph i.e. there can be no action that will keep or bring the rig to the same state.
- There can be many numbers of actions (or commands) in the MDP corresponding to the components on the CI. But any *action* from a state *s* can lead to only one other valid state *s'*. So, the actions can be represented simply by the corresponding states *(s_i, s_j)* it is between i.e. the *edges* between *valid* states represents a command. For two states s and s', if there is only one edge (s, s') during training with a command (e.g. $\mathbb{w}_1\mathbb{a}\mathbb{w}_2$), the edge (s', s) may be created by reversing the command (e.g. $\mathbb{w}_2\mathbb{a}\mathbb{w}_1$), if that is permitted.

  There a path should exist from every node to another node i.e. it cannot have an absorbing state or locking states. This means, that the rig cannot stall at any position with that learner having no control over it to bring it another state. The existences of the routes are vital as the rig may have to automatically restore itself to certain states from any other state by executing the commands or action associated with each edge in reverse. There may be two

pairs of states that have only one directed edge, but these should allow traversal from one side of the graph to another. A single edge between two nodes represents a one-way transition. This may be due mechanical constructions such as valves that operate in one-way. But being a semi-autonomous system, the rig must be able to reach to a state preceding the single edge transition i.e. there must be an alternate path.

- For every pair of adjacent state (*s, s'*) in the rig, there is a failed state $f_{ss'}$ for it connected to only s. The *failed* state represents the situation when a command fails and the rig enters a state that is basically an undesired state. It represents only the failure of its connected valid state *s* to reach s'. Thus, the aim of the rig in the fail state is to move to a valid state automatically which is always the corresponding connected *valid* state. The probability of success from each failed state to the success state is always considered 1. If it is unable to restore itself to its *valid* state then the rig is considered broken. There can be no specific command associated with the edges on $f_{ss'}$.
- There are nodes that represent the goal states. Goal state may be determined by a number of ways - the node with the highest degree or the most visited node during training. But the best way is to collect the goal states from the makers explicitly. These states signify the achievement of a target i.e. learning objective in the experiment. Goal states or task can be a single state in the MDP or there can be multiple in which case the MDP includes multiple rewards strategies (R_i). This will generate multiple policies for each reward strategy. Each reward strategy R_i allocates the maximum *utility* to the goal state *i*. Correspondingly, the a policy exists for each goal state *i* as = π_1, π_2, ... π_g. each reward strategy R_i or π_i gives most importance to the goal state $i \in G$.

Another important aspect of the MDP is its quality as MDP is trained by its maker. The makers are not expected to cover all possible states and transitions during training. For this purpose, a number of edges may be added by the rig itself depending upon conditions set by the maker. The quality of a MDP after training then can be defined as:

$$Q = \frac{number\ of\ edges\ from\ Training\ Data}{number\ of\ edges\ added\ by\ the\ rig} \quad \ldots (2)$$

B. The MDP generating Algorithm

The MDP is generated from the training data that explicitly contains the goal states. The *makers* can use the following steps to generate the MDP:

1) When the experiment is run by the maker for training, record each command and the corresponding state (x^t_n) that is reached when it finishes i.e. the values of the ports along with the goal states. This recording provides the training data set.
2) For each state transition, s_1 to s_2 recorded, add the states and action as the directed edge (s_1, s_2) in the MDP.
3) Check whether there is any node that is not reachable from any other node. If there is any such node, check whether all the transitions can be bi-directional.
   (i) If yes, for each pair of state that has one directed edge, add another edge in the opposite direction with the reversed command. The number of such addition is noted to calculate the quality of the training set.
   (ii) If no, then the training data is insufficient and more data is need. The algorithm stops here.
4) For each edge (*s, s'*) set probability $T(s, s') = 0.99$.
5) For each edge (*s, s'*), add a fail state $f_{ss'}$ with edge $T(s, f_{ss'}) = 0.01$ and $T(f_{ss'}, s) = 1$.

   When an MDP is created for the first time, Step 4-5 assigns static values for the probabilities of transition. In subsequent training sessions, the success rate of any existing edge (s, s') may be recorded and the values for probabilities in $T(s, s')$ and $T(s, f_{ss'})$ can be updated.
6) Calculate the degree deg(s) of each state $s \in S$. For each reward strategy, $R_i \in R$
   (i) Initially assign for each edge (*s, s'*) in the MDP a reward value $R_i(s, s') = deg(s')^2$.
   (ii) Then the reward for goal state *i*, corresponding to R_i, $R_i(s, i) = 2 \times max\ (deg(s')^2)$.
   (iii) For all fail states f_s from state *s*, $R_i(s, f_s) = 0$.
7) Once the MDP is constructed, the Value Iteration Algorithm (VIA) is used to determine the best policies (π_i) for the MDP.

The VIA(r) starts with the value function $V_0^r(s) = 0$ for all state $s \in S$. Then the following is repeated until for all $s \in S$, $V^r_{(i+1)}(s) - V_i^r(s) \leq 0.001$ i.e. it converges.

$$V^r_{i+1}(s) = \max_{s' \in E} T(s, s')[R(s, s') + \gamma V_i^r(s')]\} \quad \ldots (3)$$

for i = 1, 2, ..., where, γ is the decay rate. In each iteration for each state s, the policy (π_r) records the state *s'* as the best next state for which $V^r_{i+1}(s)$ is the highest. The VIA itself is repeated for all rewards strategies $R_i \in R$ to generate corresponding policy plan for each goal state. The algorithm generates the MDP and a set of optimal policy plans corresponding to each goal state.

V. ANALYSIS

Step 1-5 creates the MDP according to the properties discussed earlier and adds the fail states. Step 4 makes sure that the probability of transition is never 1 for transitions between valid states. Even if no failure is recorded, there is always a chance of failure. Step 6 calculates the reward matrix (R_i) for each goal state in G. It ensures that each of the edges leading to the goal state has the highest reward value. Step 7 calculates the best reward possible i.e. value function for each R_i. Note there is no summation as in the regular VIA [8] because every successful command or action leads to only one *valid* state and *fail* states are not counted .

Different MCUs have different computational capacities and power resources. The number of states generated in the MDP may be very large and edges between each state even larger. Processing the VIA to calculate the best policy map takes the largest computational effort. This algorithm has a decay rate (γ) that may be altered to increase or decrease the speed of the algorithm. Fig. 5 shows the effect of changing γ from 0.99 to 0.5. While the iterations

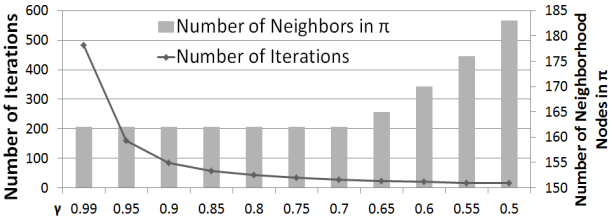

Fig. 5. Relationship between decay factor and the accuracy of the policy.

reduces exponentially, the quality of the policy will drop at certain point (in this case $\gamma = 0.65$). Thus, the a suitable value for γ may be determined for a particular rig for a given training data set. This will change respective to MCUs and the experiments built around them and useful when the policy needs to be recalculated quickly such as during an experiment session.

A. Indicators in the MDP

The quality progress in the transition of the states is measured with the following values :

1. d_{ij} and Δd - The primary indicator is the raw distance (d) between the current state and the goal state. This distance is the length of the shortest path in the corresponding policy π_j for current state i to goal state j. With each new state of the rig, the change in value of d indicates whether the learner has moved away from the goal state or not. $\Delta d < 0$ when the rigs state moves towards a goals state and vice versa.

2. $ə$ and $\Delta ə$

$$ə_{ij} = Length(P_{ij}) \,/\, [Length(P_{ij}) + n_a(ij)] \quad \ldots(4)$$

where P_{ij} is the shortest path in the MDP graph, between current state i and the goal state j and $n_a(i,j)$ is the number of pairs of adjacent nodes in the path P_{ij} for which are not adjacent in the policy π_j. This value indicates the relative distance so the rigs state forms the goal state. A value of 1 indicates that the rig can transit to the goal state and it is right on track. A lower value indicates, that in order to reach the goal state the rigs has to go through some suboptimal paths in the policy which indicates it is off-track. Varying probabilities of success in the MDP means that sometimes the shortest transition may not always be the most preferable option in π.

3. τ and $\Delta\tau$

$$\tau_{ij} = Length(P_{ij}) \;/\; Length(P_{ji}) \quad . \quad \ldots(5)$$

This indicates the weight of the immediate action of the MDP. For a rig MDP that may have at least two directed edges that does not have an opposite directed edge but still maintaining a directed path between all states, this indicator ($\Delta\tau$) shows any sudden change in the rig state that is very faulty in terms of getting to the goal state. For instance, in Fig 4 if the rig is in position A, then going to D takes is 1 step, but if there is one bad decision of moving to B, then the feasible path length immediately increases suddenly. This sudden change could indicate severe learner's mistakes particularly for one-way transitions of states. Note that only when a learner makes a wrong choice and chooses a wrong one-way path, then $\Delta\tau > 0$. For all other type of transitions before and after that, $\Delta\tau = 0$. Thus it is very easy to detect such a mistake.

The change in indicators may be used in multiple ways, depending upon the system it is being used in, to obtain a value representing all three changes. The rig can then automatically decide whether it should intervene in the agents control commands and how much it should govern itself.

VI. Using the MDP in RAL

This section discusses the application of the MDP in case of RAL experiments.

A. Using the indicators

While the indicators may be used in multiple ways to evaluate the system and the agent's status during the operation of the rig, in case of RAL a simple binary evaluation is sufficient. The purpose of the intelligence in the MCU using the MDP is to guide the learner automatically and decide whether helping the learner is required or not. Thus the evaluation result (κ) can have only the values '*yes*' if the state change was profitable in some way or '*no*' otherwise., depending upon the rigs position in the MDP. However to make the decision, the last few transitions must be monitored and recorded accordingly.

Whenever a command is executed, the Algorithm *Evaluate(i)* is run for the current state *i*. If the current state is a goal state, then all variables are reset. There are three variable to monitor the d, $\Delta\tau$ and the total number of transitions. The number of time d increases for all goal states, number of time $\Delta\tau \neq 0$ for any goal state and the number of transitions are recorded in q, r and p. If any of these goes over a threshold r_c, q_c or p_c respectively without reaching a goal state, then the users is in need of assistance and κ return 'yes'. The threshold p_c is determined by the distance to the closest goal state at the beginning of the experiment session or when a goal state is reached. A tolerance of ε that maybe added to p_c along with the values of r_c and q_c are system settings put by the system administrator or the maker changing the difficulty level of getting any help for the learner.

Algorithm Evaluate(i) with global counter variables p, q, r

if $i \in G$ *then*

$p \leftarrow 0, \quad q \leftarrow 0, \quad r \leftarrow 0, \quad p_c \leftarrow \min_{j \in G \vee i \neq j} d_{ij}$

else

if $\exists\ k \in G, \Delta\tau_{ik} \neq 0$ *then*

$q \leftarrow q + 1$

if $\nexists\ j \in G, \Delta d_{ij} < 0$ *then*

$r \leftarrow r + 1$

$p \leftarrow p + 1$

if $p > p_c + \varepsilon$ *or* $q > q_c$ *or* $r > r_c$ *then* $\kappa \leftarrow$ *yes*

else $\kappa \leftarrow$ *no*

B. MDP Inputs

There are few variable components that need to be defined or acquired from the makers to be able to create or use the MDP properly. These are:

1. Initial state: As mentioned earlier, in the MDP there exists a path between every pair of states. Thus the maker can choose an initial state. The experimental rig will revert back to the initial state at the start of each experimental session by traversing through all the intermediate states.

2. Whether all transitions can be treated as bi-directional?

3. Whether state interpolation is allowed? The MDP defines a boundary which is at the best partially known as maker is not expected to cover all possible feasible state of the rig in the training. It is in theory possible for the MCU to transit to an intermediate state that lies on the path between any two known states. However, such interpolation may not be allowed at all if the rig is not completely free to operate and may break down at certain states that are in between known stable states. Allowing interpolation assumes bi-directionality is allowed.

C. Rig Operation

The rig operates by first receiving the incoming commands and then processing it. With the MDP architecture, the rig is intelligent enough to make decisions on its own whether the learner/agent is following a feasible chain of control commands. The rig operation goes through the following steps:

Step 1. At the beginning of each learner's experiment session, the rig reverts back to the initial position and the values of d, τ and initial p_c are calculated.

For every new *write* instructions, the steps 2-7 are repeated. Other instructions (*read* and *wait*) are executed immediately.

Step 2. When a write command $\mathbb{W}(P, V)$ is received in state s_i the corresponding expected state is calculated. It is checked whether executing this command will lead to a state (s_{i+1}) such that, s_{i+1} is valid and (s_i, s_{i+1}) exists in the MDP. If it is not valid and interpolation is allowed s_{i+1} is checked if it can be interpolated.

For a state (s_{i+1}) to be an interpolated state, there must exist at one other state $s_2 \in \{S - G\}$ such that,

- s_i, s_2 exists in MDP and
- the feature vector i.e. the values of all the n ports must be exactly the same except only one port (say n_j) in s_i, s_{i+1}, s_2 and the value for n_j in s_{i+1} should lie between n_j in s_i and s_2 i.e.

$s_i[n_j] < s_{i+1}[n_j] < s_2[n_j]$

This ensures that the interpolated state is actually traversed by the changing port, but not recorded. This may occur when there are different parameters, like speed of the servos (see Fig. 3), used while training from the one used by the learner. If it is not valid and interpolation is not allowed then s_{i+1} is *undesired*.

Step 3. If the new state is not *undesired* i.e. *valid* or *interpolated*, then the command is executed. Otherwise the command is rejected. After multiple rejections of write commands, the rig can decide that the user needs support.

Step 4. The resultant state (s'_{i+1}) of the command execution is matched with the expected state (s_{i+1}). If $s_{i+1} \not\equiv s'_{i+1}$, then the rig is in the *failed* state. In this situation, the rig tries to recover back to the previous state s_i by trying to write the value to the ports as in s_i.. If the rig cannot restore the states of all ports to the earlier values, it is considered broken and requires the makers intervention.

Step 5. Once a state is successfully changed, the values for Δd and $\Delta\tau$ are calculated. In the RAL scenario, the probability of success of an action from any given state is very high and generally equal for all transitions. So the value of $\Delta\text{ə}$ is not useful in context of RAL. However, the other two indicators, Δd and $\Delta\tau$ are very important. The value for κ is then calculated with *Evaluate()* for the resultant state. If κ is 'yes', the learner is provided with hints to the next feasible state towards the nearest goal state. The nearest goal state j is the one for which the path is the shortest in corresponding policy π_j from the current state for all active goal states.

Step 6. Once a goal state is achieved, it is considered done and from the learner's perspective there is lesser incentive to re-approach that state. Thus, once a task state j is reached, its corresponding values for Δd and $\Delta\tau$ are not considered for calculating κ in *Evaluate()* i.e. removed from G.

Step 7. If the current state s' is interpolated from previous state s, then add s' to the MDP. At this point there is no edge between s' and s which could also be the case if s was interpolated by the last command. In either case add edges (s, s') and (s', s) and incorporate them with the *fail* states $f_{ss'}$ and $f_{s's}$ into the MDP by following Step 4-6 of MDP generating Algorithm (as in Section IV B) accordingly by applying the steps on the new edges/states. Then re-calculate the policies.

D. Example

The example considered to illustrate the use of MDPs is the pendulum experiment mention earlier. This experiment has three actuator i.e. the feature vector in each state contains three values of the actuator (n = 3). The values returned are integer numbers (if the servos rotates twice full circle, the value is 720 degrees; if it rotates backwards then the value is -720 degrees). The rig was trained with a sequence contains (ϱ = 79) transitions that generated 73 states in the MDP. There are 4 goal states defined in the experiment - C7, C16, C25 and C34. The learner starts with the state C0 which is the initial starting position. The learner can send commands to the rig and leave the experiment in any random position at the end of their session. The rig takes its state back to C0 for the next session. The probability of success of each command to the rig is 0.99. This is a high value as there is little probability of it failing and it is and it is equal for all transitions as all the actuators have the same reliability. The training allowed addition of bidirectional edges as all state transitions (s_1, s_2) recorded can be done in reverse (s_2, s_1).

E. Results

Fig. 6 shows the final values of each state after the VIA is run corresponding to the 4 goal states. For each of them the goal state has got the highest value followed by the state that is closest to it e.g. C7 is adjacent to C6, C8, C12 and C13. The *fail* states also get high values, but as the failed states are only connected one valid state (and the reward for the transition from the valid to the fail state is 0) for all fail state, the outgoing edge is chosen in the policy (π_i). The

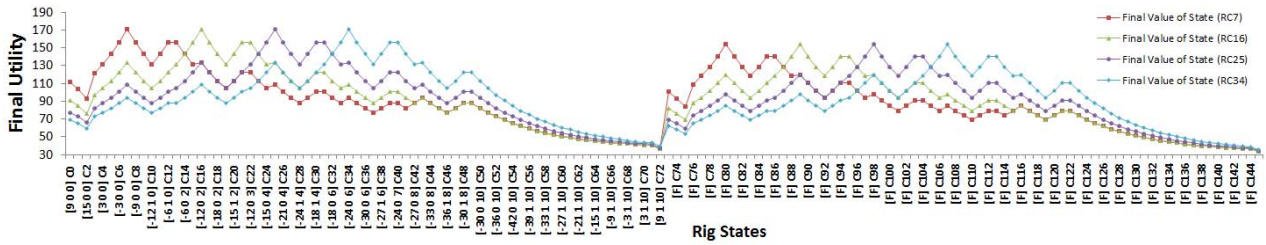

Fig. 6. The final utilities or values of the states in each R_i corresponding to the goals states C7, C16, C25, C34. For failed states only the highest value of shown for all failed states for a valid state.

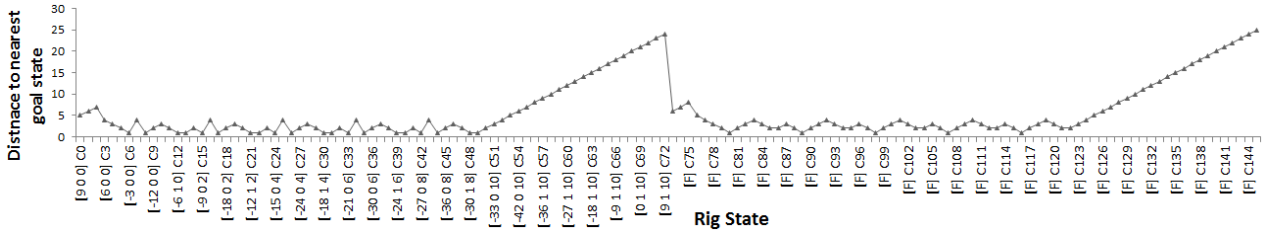

Fig. 7. The distance to the nearest goal state for each state. For failed states only the smallest value is show for each corresponding valid state.

values of all the fail state closely follow that of their corresponding valid state, but are always lesser.

Fig 7 shows the distance form any state to the nearest task state. As the probabilities of transition success are all same and the transition are bi-directional, the values for $\Delta\tau$ and $\Delta\vartheta$ always remains the same. So this experiment, the value of $\Delta\tau$ has no meaning and the evaluation (κ) is solely depended on the value of *d*. For e.g. $\Delta d > 0$ if the ball is moved up many times beyond the reach of the hand lever as in states C6 to C0 and to C3. The distance will keep increasing to all the goal states.

Note the significant increase in the distance for the states from C50 to C73 in Fig 7. This is due to bad training as these states were generated as part of the training data set. They basically represent the maker generating transitions that are not very effective towards reaching the goal. This is however important for system like RAL for teaching purposes if the maker want to make the usage of the rig as flexible as possible.

F. *Presenting the guidance*

The exact methods of providing guidance to the learner is out of scope of this paper, but once the decision to guide is made and the path to the goal state established, the rig can guide the users using any visual cue. For the MCU controlled experimental rigs, the actions are each individual commands allowed from the CI components. All actions may not be defined for all states restricting the learner and implementing the control policy. A catalog of learner-friendly *terms* may be defined for each command or MDP action $\mathbb{w}(P, V)$ by the maker and stored against the corresponding edge (s, s'). While presenting the guidance, the next steps can be presented on the interface by using the corresponding *terms* of the action required to change from current state to the next state in π_j in the shortest path towards the nearest goal state *j*.

VII. LIMITATIONS

This section discusses the advantage sand disadvantages of the MDP process.

A. *Advantages*

The proposed MDP based control architecture helps in the following ways:

- The maker's inputs to the creation of control policies are minimal. Makers do need to explicitly specify the limits of changes in the system components. The MCUs can identify the most ideal sequence of activities and act accordingly.
- With the MDP rig is intelligent enough to judge the quality of the agents' use of the rig. This could help in making time-critical decisions when required. Also, the rig is able to provide support to the agent if required on what is the best control policy at the time.
- In the RAL application, the systems allow for evaluating the learner's performance. A higher number of instances where the users make wrong decisions ($\Delta d > 0$ and $\Delta\tau \neq 0$) can be recorded and a feedback may be provided on the interaction. Moreover, it helps to identify whether the users have completed the goals of the experiment at all.

All of this can be achieved without setting any specific limits in the software for the rig or the control technology. The rigs can all run the same software to create and parse the MDP regardless of the experiments.

B. *Limitations*

The limitations of the MDP approach are as follows:

1. It requires training data. The experimental rig must be used multiple times by the makers and testers to generate a sufficiently large training data set that can encompass all aspects of the experiment. This means that the rig must be used to its operational and configurational limits to ensure that the training data set as well as the corresponding MDP can cover all possible states. This is difficult to do perfectly as makers may not foresee all possible uses of the rig, thus rendering certain inputs from the users invalid with the respect to the MDP even though they may not be unstable. In case of remote laboratories, it is used for learning purposes and the

proper way to achieve the goals is as important as the learning goals themselves. Thus, following the makers' steps is acceptable when applying the MDPs to the RAL scenario.

2. The goal states may be difficult to judge. In a poorly designed rig, the actual events of the experiment result may not be captured properly from a particular state. In those cases, the effectiveness of the indicators reduces. But, this can be resolved by adding a dedicated sensor which will confirm the task events taking place. The particular variable then can uniquely identify the task state.
3. Recovering a rig is very difficult if multiple ports value were changed in parallel in the previous command. But it works fine, if only one port is changed in one atomic command i.e. there is no parallel change in the ports.

The flexibility to match intermediate states can mitigate the impact of improperly trained rigs to certain extent, but the interpolating techniques needs to be improved and the rigs, very well trained. However, with a large number of users using the system as part of the training, the MDP can be accurate.

One aspect of the goal states not addressed in the paper is the possible temporal relation between them. In some experiments it may be required to complete a set of tasks before proceeding to others. This can be handled by activating a new reward strategy at the given time once any goal state is reached.

CONCLUSIONS

Experimental rig in remote access laboratories can be represented as MDPs. These MDPs can store the states and the transition information between them. The MDPs can successfully be used to create the control policies as to what are valid and unwanted states in the rig. The quality of the policies will increase with time as more and more users use them. Three system indicators have been proposed that can evaluate the current state and correspondingly, the performance of the learner with the rigs. Certain aspects in the proposed architecture can be improved such as acquiring training data, interpolating states and recovery form a failed state.

REFERENCES

[1] Andrey Kolobov, Mausam, and Daniel S. Weld. A theory of goal-oriented MDPs with dead ends. In Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI'12), 2012.

[2] C. Boutilier, T. Dean, and S. Hanks, "Decision-theoretic planning: Structural assumptions and computational leverage," J. Artif. Intell. Res., vol. 11, pp. 1–94, 1999.

[3] A. Maiti, A. A.Kist, and A. D. Maxwell, "Real-Time Remote Access Laboratory with Distributed and Modular Design," Industrial Electronics, IEEE Transactions on, vol. 62, pp. 1-1, 2014.

[4] V. J. Harward, et al., "The iLab shared architecture a web services infrastructure to build communinities of Internet accessible laboratories," Proceedings of the Ieee, vol. 96, pp. 931-950, Jun 2008.

[5] S. Temizer, et. al, "Collision Avoidance for Unmanned Aircraft using Markov Decision Processes", AIAA Guidance, Navigation, and Control Conference 2 - 5 Aug. 2010, Toronto, Canada.

[6] J. Boger, J. Hoey, P. Poupart, C. Boutilier, et.al., "A planning system based on Markov decision processes to guide people with dementia through activities of daily living," Information Technology in Biomedicine, IEEE Transactions on, vol. 10, pp. 323-333, 2006.

[7] M. Oshita and T. Matsunaga, "Automatic Learning of Gesture Recognition Model Using SOM and SVM," in Advances in Visual Computing. vol. 6453, Springer Berlin Heidelberg, 2010, pp. 751-759.

[8] O. Madani, "Polynomial value iteration algorithms for detrerminstic MDPs", Proc. of the 18th UAI, pp.311–318, 2002.

[9] A. Maiti, A. D. Maxwell, A. A. Kist, and L. Orwin, "Joining the Game and the Experiment in Peer-to-Peer Remote Laboratories for STEM Education," in 2015 exp.at'15, Ponta Delgada, Portugal, 2015.

[10] E. G. Guimaraes, E. Cardozo, D. H. Moraes and P. R. Coelho, "Design and Implementation Issues for Modern Remote Laboratories," IEEE Transactions on Learning Technologies, vol. 4, pp. 149-161, 04/01 2011.

[11] S. Adlakha, R. Madan, S. Lall, and A. Goldsmith, "Optimal control of distributed Markov decision processes with network delays," in Decision and Control, 2007 46th IEEE Conference on, 2007, pp. 3308-3314.

[12] M. A. Wiering and E. D. de Jong, "Computing Optimal Stationary Policies for Multi-Objective Markov Decision Processes," in ADPRL 2007. IEEE International Symp. on, 2007, pp. 158-165.