Joining the Game and the Experiment in Peer-to-Peer Remote Laboratories for STEM Education

Ananda Maiti[1], Andrew D. Maxwell[1], Alexander A. Kist[1] and Lindy Orwin[2]
[1]School of Mechanical and Electrical Engineering, University of Southern Queensland, Australia
[2]ADFI, University of Southern Queensland, Australia
anandamaiti@live.com, andrew.maxwell@usq.edu.au, kist@ieee.org, Lindy.Orwin@usq.edu.au

***Abstract*— Remote Access Laboratories (RAL) provide access to experimental setups from remote locations. These experimental setups are composed of controller units programmed to gather data and interact with user inputs. A distributed version of RAL can be maker oriented i.e. the experiment rigs are designed by individuals and shared among each other. This paper presents the programming aspects and activity user interface (UI) design and organization of experiments in a distributed RAL aims at STEM education. The user interface must be interactive to increase engagement and motivation for the user. Being designed for school students, the environment to create the control logic of a rig created by the student needs to be on a homogenous platform. The programming language has to be easy to understand and use. Characteristics and requirements of a graphical programming language SNAP, which is modified and used as the programming platform for RAL, is studied in this paper.**

Keywords — remote laboratories; e-learning; computer programming; progarmming languages; quest-based learning

I. INTRODUCTION

The educational disciplines of science and engineering typically require learners to demonstrate proficiency bridging the theoretical and experimental world. As part of these experiential learning experiences, online Remote Access Laboratories (RAL) may be used for demonstrations of actual event and experiment [1]. A novel project, "RALfie – Remote Access Laboratories for fun, innovation and education", is used as an example of a Peer-to-Peer (P2P) environment for the deployment of remote access laboratories where users create lab activities and associated programs and share them through the Internet (see Fig. 1). This system aims to promote Science, Technology, Engineering and Maths (STEM) education among young learners by enabling collaboration and increasing 'hands on experience' [2].

Computer-based *games* are fundamentally designed for quick, colorful and creative fun and entertainment. Other than entertainment, games have also been used to create environments for the students (players) to acquire knowledge and skills [3]. Gamification of learning environments can take many forms. In context of RALfie a quest-based approach is taken. Students access experiments through quests, which provide context and guidance. The content of quests is presented as a set of instructions and associated resources. It guides the interaction between the students and the U*ser Interface* (UI) of the experiment. Quests are organized into hierarchical groups as a larger game-based learning environment [8] where individual users can accomplice bigger goals by completing multiple quests. In addition, experiments themselves can be designed as interactive games.

The key innovation of the RALfie system is that users are able to design and host experiments themselves. The environment has two types of users – *Makers* (experiment providers), who create experiments and the *learners*, who run the experiment for learning purposes [5]. Any user can be both maker and learner. The creation of experiments by the Makers involves programming to develop a user interface and to control the experiment. This often involves setting parameters and retrieving data. However, learning programming languages can be challenging for new users as they have their own syntax and semantics to describe complex functionalities.

Several methods have been suggested to teach programming to young learners using either library based or visual methods [3]. In order to provide a unified, consistent, and easily understood programming interface to represent the states of the experimental activity, this paper outlines the requirements of a programming language and supporting technical tools for a P2P RAL environment and evaluates the feasibility of using a graphical languages as the Integrated Development Environment (IDE) to create a Human-Machine interface for experiments.

The rest of the paper is organized as follows: Section 2 discusses related works in robotics and programming for RAL. Section 3 shows how P2P RAL operations. Section 4 presents the P2P RAL programming and storage requirements. Section 5 and 6 presents the RALfie implementations and users' feedback.

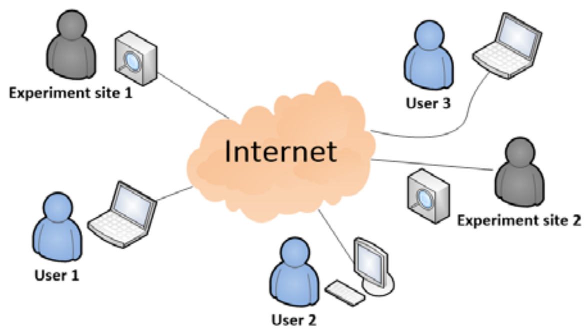


Fig1. The RALfie Communication System Architecture

II. RELATED WORK

This section describes the teaching practices for robotics and programming languages.

A. *Teaching Programming Languages*

A computer game was used as a tool for teaching Object-Oriented programming (OOP) methodologies and paradigms in a computer science course [12]. This was a character based role-playing game where the player's character has to follow a storyline and clear some objectives. In doing so, the character (or object as in OOP) acquires traits (properties as in OOP) and performs tasks (methods as in OOP). The player gets experience points or rewards for finishing the given set of objectives. Game oriented procedures have been implementing in STEM fields [14]. Student's motivation mainly includes intrinsic goals and tasks of the game.

Natural Language has been used to teach programming fundamentals [11]. It has been shown to be a good alternative to traditional programming languages defined by context free grammar. The natural language although attractive, may not be directly applied to RAL, due to its complex use of ports [15] used to control peripheral. A visual drag and drop language like SCATCH [7] which is a simple language used to teach programming concepts to K12 students is more suitable. The drag enabled programing building blocks allows the pedagogical principles of teaching programming with a low threshold for entry.

B. *STEM and RAL*

The effects of applying remote laboratories based education have been studied before [1, 2]. The main objective of a STEM RAL is to provide experiments in the STEM fields to school level students. These experiments involve lesser complicacy in building or running the experiments and focus more on the visual actions of the rigs and appropriate pedagogical tools to teach the STEM concepts.

RALfie aims to be a STEM based RAL [2] that can be effective in creating engagement for students to continue with STEM field in higher studies. This requires certain features in RAL such as collaboration and hands-on-experience to build a rig. Collaboration can help with to exchange of ideas about a STEM concept while self-made designs increase engagement.

C. *Robotics and programming in RAL*

Robotics and automation are integral parts of online laboratories. Robotics components are added to a localized version of the experiment setup to make it accessible from remote locations [6]. LEGO based robotics is designed for teaching K12 students about robotics. These have been part of many school based STEM initiatives [13].

RAL programming uses various programming languages although often it is LabVIEW. Pastor et al [10] describe user based custom programming. This approach uses XML to specify the components and the corresponding functions which are then recompiled as Java programs. The students rely on using a XML based Laboratory Experimentation Description Markup Language for creating the laboratory modules and joining them to form an experiment. This form of language is not suitable for STEM students..

III. P2P RAL CHARATERISTICS AND ARCHITECTIRE

This section describes the RAL learning objectives in context of the STEM education, the P2P RAL operation and the pedagogical advantages of the P2PRAL.

A. *RAL Learning Objectives*

The learning is RALfie happens in following steps:

1. As in regular RAL, the experiments have an underlying curriculum concept, in this case a topic from STEM. For those using experiments, the learning objective is to run the experiments as per given instruction and observe the action on the experimental rig.
2. In-experiments data collection/observation is important part of the activity. Each activity will have at least one question to be answered based upon the observation of the experiment run.
3. Post-experiment data analysis is less common but can accompany an experimental activity. Learners are then required to collect large amount of data during the experiment run and after completing it, analyze it to answer a question.

The innovative part of the RALfie is the maker aspect. Some users are also developers of the experimental rigs. This aspect has the following learning objectives:

1. Designing and creating an experimental setup to express the STEM concept.
2. Adding automation and programing it to operate using Remote UI. This allows the Makers to gain experience in prototyping innovative ideas.

Thus, a distributed P2P RAL engages user in two distinct activities - assembling and programming the experiment setup and then sharing it with others.

B. *P2P RAL Operation*

The operation of the P2P maker-learner experimental rig sharing is shown in Fig 2. The process starts with the developer identifying/given a STEM problem. Once it is decided on what is to be built, the corresponding experimental setup is prepared. The experimental rig uses automation components such as actuators into the experimental rig that enables its computer-based/remote control. The additional of the automation tools may require minor re-design of the rigs. These two steps are a repeated until a satisfactory control interface and the rig is setup.

Once the setup is ready, it needs to be stored as a *published* experiment in a repository where other users can search them. This storage mechanism id modelled around the quest-based learning [8].

After the experiments are published, it is available to the learners. They run the experiments, collect data and complete activities to gain experience points and collect badges in the quest based system. The creation to publishing affects the users experience with the system in the reverse order –

- Search is affected by storage policy,
- Experiments run and answering the questions is affected by automation and programming of the rig and its interface.
- The learning outcome and 'game incentives' in the form of badges, eXperience Points (XP) or achievements gained by the learner is dependent on the type of rig and the experiments chosen by the experiment developer (Makers).

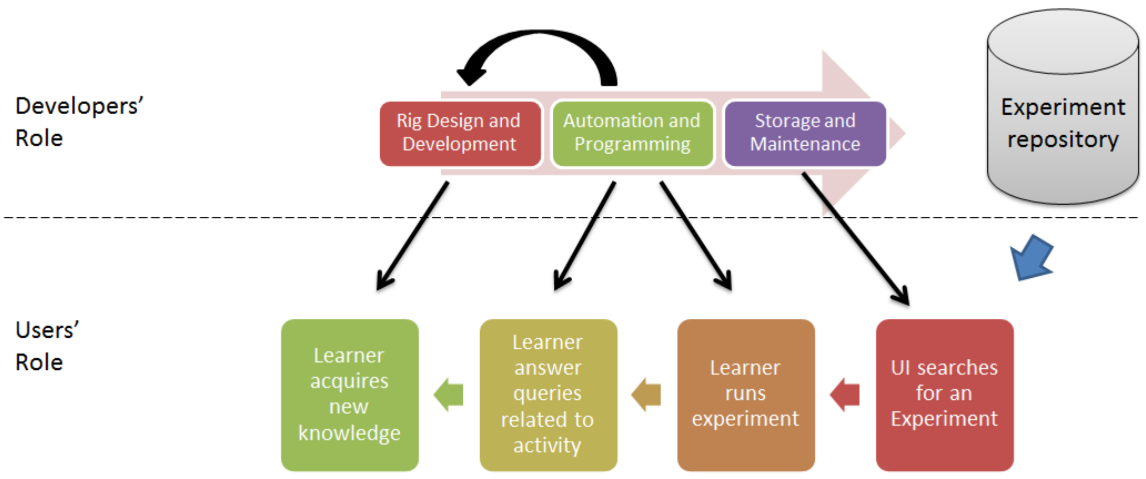

Developers' Role
Rig Design and Development
Automation and Programming
Storage and Maintenance
Experiment repository
Users' Role
Learner acquires new knowledge
Learner answer queries related to activity
Learner runs experiment
UI searches for an Experiment

Figure. 2. The P2P experiment creation, storage and usage operational steps.

For this paper, the focus is on the "Automation and Programming" and "Storage Mechanisms" from a developer's perspective.

C. *Advantages of P2PRAL*

The advantages of P2P architecture are:

- It allows collaborations or communication between individual students.
- It allows hands-on-experience of building the required experimental setup.
- It allows modularity i.e. multiple experiments can be added form different locations that re totally independent of each other.

IV. P2P RAL Programming and Storage

A proper programming language and development environment must be used to enable users to connect the instruments to the Internet in a homogenous manner.

A. *Role of Programming Language*

Once an experimental rig has been assembled, it must be programmed to communicate with the UI through Internet. From the perspective of young learners programming languages may be divided into many groups:

- *Procedural vs. Object oriented programming*: The aspect differentiates between programs that have a simple flow control with programs that associates every data to a conceptual object. Experiments in a RAL are usually operated by a small finite set of commands for a session. As such, it should be procedural in operation i.e. the code composed must start and end without initializing any object. Using objects adds higher overhead of associating each function with an object.
- *Text based vs. visual languages*: This aspect differentiates between the styles of representation of the language components. A text-based language requires more typing of code, with the associate potential for errors, while the visual languages are more colorful and primarily uses drag and drop methods. Visual languages are more appealing to the users with less to no programming background [17].
- *Declarative vs. Imperative languages*: This aspect differentiates between the structures of languages. The declarative strategy specifies the logic of the computation without specifying the manner in which it will be obtained (e.g. SQL). The imperative programming explicitly specifies the line of code. A former is more suitable for teaching young learners but requires high level of computational flexibility for interpreting the users input.

Hence a declarative, visual and procedural language was chosen for RALfie. For a P2P RAL like RALfie, the fundamental capabilities required for its programming language are:

1. *Iterative and conditional abilities*: These are the two most commonly used programing constructs and needed to write any sort of program.
2. *Data logging abilities*: The language must be able to read and write with a range of sensors and actuators.
3. *Rapid user interface design capabilities*: A GUI and an IDE are also important to easily (re-) configure any program. The visual nature of a program is more appealing to young learners [7]. A GUI allows the users to be more expressive and it provides an easy way for setting up the actual user-interface for the experiment.
4. *Event capturing capabilities*: It must be event oriented. Capturing an event at the user interface and responding to that is vital to a RAL experiment program. Thus events must be clearly defined and a wide variety of events must be supported.
5. *Web Browser based*: the language and the corresponding IDE should preferably run in a Web Browser.
6. *Packaging*: Packaging refers to the capability of creating modular software and re-using code as much as possible. Users may share their codes and designs with others.
7. *Network Capabilities*: Obviously to communicate through the internet the language must be equipped with the best internet connectivity features. Note that this feature is not required for RALfie users. The users only create code and run it with the experiment. The underlying network infrastructure is hidden from the actual users of RALfie.

There are multiple graphical languages that fulfill some of these criteria, especially 1-4, like SCRATCH. However, Blockly and SNAP (http://snap.berkeley.edu/) has the additional capabilities of being Web Browser oriented and

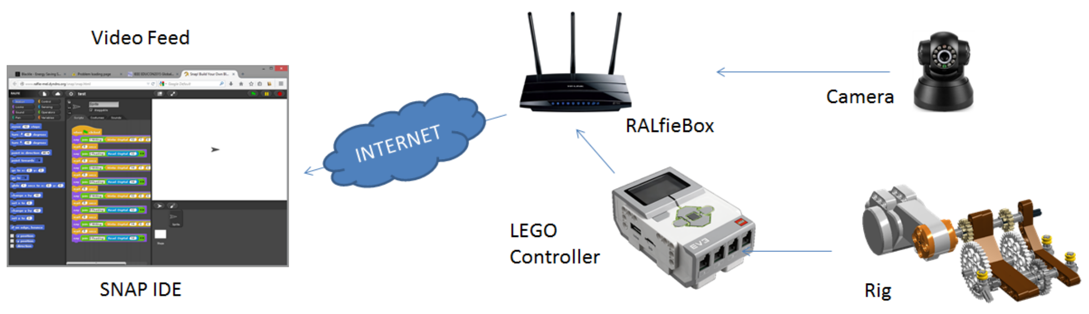


Figure 3. The RALfie Communication System Architecture

supporting HTML5. They also allow packaging. SNAP is chosen because of it's similar fetaures to SCRATCH which is a wide used language. The network capabilities are not sufficient in SNAP but an additional network module was added for the RALfie and thus it forms the basis of the RALfie platform as described in the next section.

B. *Activity as a Game*

In order to present the activity to the learner, a *quest* is created. A *quest* is basically a game with an objective that must be achieved with in game mechanics provided by the developers. To make the quest interesting and hold the attention of the learners, it is presented as a story. The storyline follows a sequence of interaction between the learner and the interface which leads to a final solution where the interface tells the learner whether the user has reached a correct stage or not.

In case of RALfie a *Narrative* approach [16] is taken where a character is used to first describe the UI environment i.e. the tools available on screen such as buttons, indicators etc. Then the learner is presented with the quest logic during which they are simply asked for a set of values through a set of questions. The answers to these questions are the input parameters to the experiments. The learners then observe any change in the experiments site through the video feedback or data feedback on the UI. At the end of the quest the learner is presented with quest question(s). The answer to these final set of questions lies within the previous interactions with the UI and will indicate the learning outcome of the quest.

C. *Storage in the Content Management System*

Once an experiment is created, it must be hosted as part of structured hierarchy so that users are able to search for them and access them in the appropriate sequence. For ease of use and ubiquitous access *Content Management System* (CMS) are often associated with RAL. These provide the learning materials and task instructions that give the context for the experiment. Traditionally these would form lessons delivered by a Learning Management System such as *Moodle* or *Blackboard*.

In order to increase communication and collaboration between learners, RALfie deploys a non-traditional, gamified approach. Content in RALfie is delivered within a *quest* [8]. Learners receive eXperience Points (XP) that accumulate to earn badges that indicate competency. Learners are members of guilds that provide an online learning community. This gamified approach has implications for the design and delivery of content and learning experiences. However, the requirements of the distributed RAL described in this paper remain constant whether a traditional lesson structure or a quest-based system is used in relation to a P2P network of user-generated RAL.

V. RALFIE IMPLEMENTATIONS

This section presents the technical implementation regarding the programming environment, communication and users feedback for the RALfie.

A. *The Instrument Programming Interface*

The system components are shown in Figure 3. The backbone of the P2P RAL communication is the VPN or overlay connection between users. Especially designed/programmed routers connect each experiment node to the VPN. Each experiment setup has one such router. One router is ideally associated with one controller although it may connect to multiple controllers.

A web-browser based IDE of SNAP [4] is used as the programming interface. SNAP is a graphical programming interface that allows drag and drop of commands to form the program. The interface is exactly same in syntax and structure as that of SCRATCH [9]. This allows quick understanding of the user interface. The only difference between SNAP and SCARTCH are that SNAP is written in JavaScript allowing it to be executed on any browser. SNAP also allows creating custom block which are essentially subroutine or custom functions.

The RALfie re-deploys these tools based on SNAP with the additional requirements of RAL hardware interaction. This adds to new programming paradigms that need to be implemented and used by the providers.

The controllers for the experiments are low-cost microcontrollers units (MCUs) e.g. LEGO, Arduino etc. with multiple ports/pins for controlling sensors and actuators. These MCUs have an interpreting program [6] that runs the command coming from the SNAP based UI. One controller can however run multiple setups that are part of different experiment activities.

B. *Programming Paradigms of RALfie Experiments*

The following are the advanced features in SCRATCH/SNAP that are used extensively for RALfie:

1. *Network Capabilities*: When the SNAP IDE is opened, it establishes a WebSocket connection to the target controller on the VPN. Henceforth, each new command for the rig is sent through a WebSocket.
2. *Sprites as Objects/Components:* SNAP uses specific images called 'sprites' that represent each component of the

user interface. These represent aspects of 'object oriented programming'. Each sprite in the interface may be regarded as objects with its associated code. But the program is written in a functional manner and no object is ever explicitly used.

Every object in the UI is a sprite that can initiate its own code execution or perform a particular function. This implicitly implements the concurrency between execution driven by user generated events such as clicks and key-press, but the concurrency need not be part of the program logic. The most common sprites in the UI are:

- *The Narrator*: This object tells the objective of the experiments (see Fig 4a). It does not take any input either for the UI or the experiment, but simply presents a set of instructions and waits for the users' actions.
- *The input components*: These include anything like a button that may be clicked to generate an event (see Fig 4b). Any image file can be used as the input components. Upon an event, these take an input either as numeric or text value or the click itself.
- *The output components*: These are those components of the UI which simply change state depending on the output received from the experiment. The output components on the SNAP interface may be optional as there is always a video feedback and certain experiments may solely rely on the video for showing the output.

All other functional blocks available in SNAP are used related with the sprites.

1. Ports as variables: Each controller is equipped with ports/pins and each pin is connected to a sensor or actuator of the experimental rig. Additional READ and WRITE components have been written for RALfie for interacting with hardware at different ports/pins of microcontrollers. These were created under the control and sensor block in the SNAP. The READ commands take an input of a port number to return the value of sensor at that port. The WRITE command takes a port and value parameter to be written at that port to operate an actuator. These commands are put into other command structures to create the program logic of the rig operation (see Figure 5).

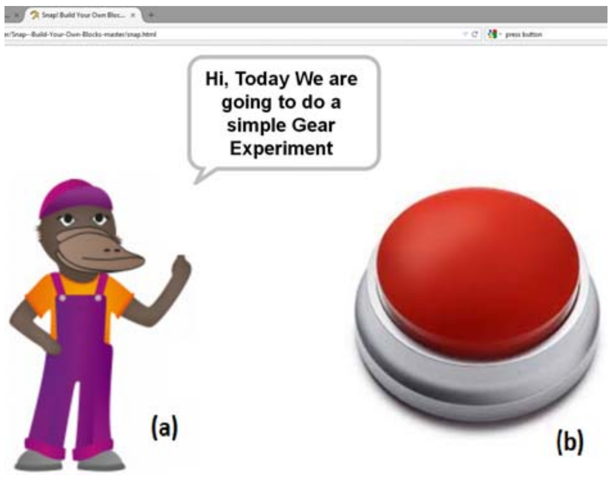
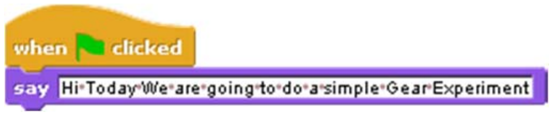
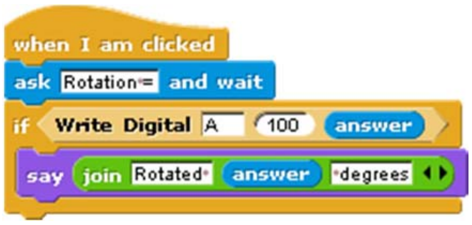


Fig. 4 (a) The Narrator of the activity (b) An example of an input component



```
when [flag] clicked
say Hi Today We are going to do a simple Gear Experiment
```

Figure 5 (a)



```
when I am clicked
ask Rotation= and wait
if Write Digital A 100 answer
    say join Rotated answer degrees
```

Fig. 5 (b)

C. Lesson/ Quest Management Interface

The language IDE and its usage must be according to the characteristics described so far. However, just satisfying the IDE requirement does not guarantee success of the system. The experiments must be stored properly. 3DGameLab, is used as the Content Management System (CMS) for RALfie. Each quest is associated with a general description of the problem and related materials. The CMS also mentions the XPs and badges one can obtain for a particular quest. Also the 3DGamlab can store the pre-requisites of the experiments. The user's final set of answers to the activity are submitted to meet quest completion criteria.. Other users' feedback on the quest, its due time and availability are also maintained by the 3Dgamelab.

The SNAP programs can be converted to XML format including the images or sprites. Once the developer is ready with a fully functional experiment and UI, they can publish the experiment by saving it on the cloud. The corresponding XML file is stored in the cloud servers and associated with the activity in the 3DGameLab. For the learners, the experiment xml file is downloaded and executed on the SNAP environment to run the experiment. They can only access the UI, but they do not have access to the associated code.

VI. User Trials and Feedback

A. The objective of the trial

A trial of the system was conducted with robotics educators, where the following sequence of activities were conducted

- Users' preliminary proficiency with Procedural Programming in SNAP.
- Users' ability to create simple activity and the usability and effects of Procedural Programming for the purpose.
- Integrating a constructed hardware robot including a MCUs and three Actuators into a small quest.

Participants were guided through the basics of the SNAP language and completed two sample example programs designed to familiarize participants with the development environment, as well as the custom output component to talk to the MCU. Participants then constructed a simple two wheel based robot, with a third flag waving actuator. A small quest was then given to the participants, to program and control their MCU based robot through a small hook turn course, and "wave" their flag upon completion. Participants were open to select either a "remote-control" type of interface, or program their wheel based robot to traverse the course by program only.

B. Observations

In the event that followed, all participants were able to gradually create the necessary program, having first established the networking to their robot, then creation of the sprites to which code would be related. Participants then built upon this with use of the SNAP output component to move each actuator in turn. This program was then built up until the robots were able to move in a controllable and predictable manner using skid steering.

All participants were successful in being able to move their robots at least partway through the track, whilst problem solving the skid steering, as well as the speed and loop parameters of their program.

During a focus group discussion afterwards, several key issues were identified:

- Whilst participants themselves were aware of the objectives of the exercise, this was not reflected in their program sprites or control interface for the activity. Participants understood the link between the software "ports" and the hardware "ports", however this was considered a threshold concept, where both ports needed to be synchronized, thus clear documentation and output component design is desirable.

- It was also identified by participants that this could also cause confusion where LEGO Mindstorm (or other MCU) hardware faults were present, particularly poor wiring connections, or improper mechanical design flaws) would cause incorrect response to the SNAP program. As such debugging systems (although not present in the trial) are desirable within the SNAP interface.

- With regards to instrumentation and sensors, participants were unsure what these devices or mechanisms were, and thus some examples or tutorials on sensors and instrumentation was requested, and although not specific to SNAP highlights the issue of open ended hardware design with novice programmers.

- Participants indicated that the organization of the SNAP interface was at first confusing, but related to familiarity with the interface. When creating the interface, participants felt a more interactive interface was required, where SNAP blocks showed or indicated what the physical object would do with any given SNAP block.

SNAP/Scratch is typically used for game development, but in this instance it was used to control the experiment; create the graphical interface for the experiment, and to direct learning of programming. Participants felt the most appealing aspect was to have a quest, and achieve a level of operation or understanding about that quest (in this case motivation of the physical robot). They also identified they could use RALfie to demonstrate someone else's rig first, to understand the capabilities of the system before building their own, and indicated that a bank of example activities would considerably help their understanding of the concepts. Additionally, it was indicated that sharing of the activities with other participants was the most memorable aspect of the trial.

Conclusions

The characteristics and use graphical programming interface SNAP for creating user interface and control logic of an experimental rig was discussed. This programming environment is most suitable for providing a platform independent and homogenous set of tools to connect a user created experiment to the internet. This approach of creating quests can increase motivation and engagement for the users as well as provide valuable experience on how to build a rig and encourage collaboration. The SNAP programming language is designed for primary school students and is suitable for creating experimental rigs in RALfie where the aim is to enable students to have hand-on-experience in building experiments and sharing them with others.

References

[1] D. Lowe, P. Newcombe, and B. Stumpers, "Evaluation of the Use of Remote Laboratories for Secondary School Science Education," Research in Science Education, vol. 43, pp. 1197-1219, 2013.

[2] A. Maiti, A. D. Maxwell, A. A. Kist, and L. Orwin, "Integrating enquiry-based learning pedagogies and remote access laboratory for STEM education," in IEEE EDUCON *2014*, pp. 706-712.

[3] I. F. de Kereki, "Scratch: Applications in Computer Science 1," in Frontiers in Education Conference, 2008. FIE 2008, pp. T3B-7-T3B-11.

[4] B. Harvey and Jens Mönig, Bringing "No Ceiling" to Scratch: Can One Language Serve Kids and Computer Scientists?, Constructionism 2010, Paris.

[5] A. A. Kist, *et al.*, "Overlay network architectures for peer-to-peer Remote Access Laboratories," in *Remote Engineering and Virtual Instrumentation (*REV*), 2014 11th Intl. Conf. on*, 2014, pp. 274-280.

[6] A. Maiti, A. A. Kist, and A. D. Maxwell, "Using Network Enabled Microcontrollers in Experiments for a Distributed Remote Laboratory," in REV 2014, Porto, Portugal, 2014, pp180-186.

[7] P. Gruenbaum, "Undergraduates Teach Game Programming Using Scratch," Computer, vol. 47, pp. 82-84, 2014.C. C. Haskel, "Design Variables of Attraction in Quest-Based Learning", May 2012.

[8] C. C. Haskel, "Design Variables of Attraction in Quest-Based Learning", May 2012.

[9] B. Kaucic and T. Asic, "Improving introductory programming with Scratch?," in MIPRO, 2011 Proceedings of the 34th International Convention, 2011, pp. 1095-1100.

[10] R. Pastor, R. Hernandez, S. Ros, D. Sanchez, A. Caminero, A. Robles, et al., "Online laboratories as a cloud service developed by students," in Frontiers in Education Conference, 2013 IEEE, 2013, pp. 1081-1086.

[11] O. L. Oliveira, A. M. Monteiro, and N. Trevisan Roman, "Can natural language be utilized in the learning of programming fundamentals?," in Frontiers in Education Conference, 2013 IEEE, 2013, pp. 1851-1856.

[12] W. Yoke Seng, M. H. M. Yatim, and T. Wee Hoe, "Use computer game to learn Object-Oriented programming in computer science courses," in IEEE EDUCON 2014, pp. 9-16.

[13] I. Rothe, "Organization of a Lego-robots contest offered to high school kids by engineering students within a project based learning environment," in IEEE EDUCON 2014, pp. 36-39.

[14] S. A. Nikou and A. A. Economides, "Transition in student motivation during a scratch and an app inventor course," in Global Engineering Education Conference (EDUCON), 2014 IEEE, 2014, pp. 1042-1045.

[15] A. Maiti, A. A. Kist, and A. D. Maxwell, "Real-Time Remote Access Laboratory with Distributed and Modular Design," Industrial Electronics, IEEE Trans. on, 2014, doi: 10.1109/TIE.2014.2374572.

[16] B. Mott and J. Lester, "Narrative-Centered Tutorial Planning for Inquiry-Based Learning Environments," in *Intelligent Tutoring Systems*. vol. 4053, M. Ikeda, K. Ashley, and T.-W. Chan, Eds., ed: Springer Berlin Heidelberg, 2006, pp. 675-684.

[17] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The Scratch Programming Language and Environment," Trans. Comput. Educ., vol. 10, pp. 1-15, 2010.