

Flexible Techniques for Fast Developing and Remotely Controlling DIY Robots, with AI flavor

Dimitrios Loukatos^{1,2}, Ken Kahn³, Dimitris Alimisis²

¹Agricultural University of Athens - Department of Natural Resources Management & Agricultural Engineering, Greece

²EDUMOTIVA - European Lab for Educational Technology, Greece

³University of Oxford, Department of Education, UK

dloukat@gmail.com, kenneth.kahn@it.ox.ac.uk,
alimisis@edumotiva.eu,

Abstract. During the last years we are witnessing a very successful osmosis between innovative and cost-effective credit card - sized computers and education. These computers, equipped with low cost sensors or actuators, can be the “heart” of various DIY robotic artefacts. This environment allows for a mixture of thinking and making activities that can be very meaningful in terms of pedagogy and science. Indeed, similar practices, usually referred as STEM or STEAM activities, are applied in many educational institutions, from primary schools up to universities, with most of the effort to focus on secondary school students. The overall process, although promising at the beginning, is not always straightforward to keep up with. More specifically, as students get more experience, they develop a hunger for more complicated scenarios that usually demand features like remote interaction with simple Artificial Intelligence – A.I. capabilities or sophisticated control of their robotic artefacts. At this moment, trainers should be able to propose simple and stable techniques to their students for implementing such features in their constructions. This paper proposes flexible methods for this to be done by exploiting the very popular MIT App Inventor and Snap! visual programming environments, in conjunction with a modified tiny web server module, written in Python, that runs on a Raspberry Pi credit card - sized computer. Furthermore, this paper reports on simple techniques being used to make robust enough robots by low cost everyday / recyclable materials like cardboard, wood, plastic bottles or broken toys.

Keywords: DIY Robots, Remote Control, Visual Programming Tools, AI, App Inventor, Snap!, Raspberry Pi.

1 Introduction

The fast growth of cell phone and tablet industry made possible the production of high-end electronics in vast numbers and thus at very affordable costs. “Exotic” devices with features like wireless connectivity, cameras, GPS, motion sensors, high resolution screens are offered at price levels that seemed unbelievable just few years ago. This fact gave a boost to the computer industry that started producing credit card

- sized devices equipped with powerful features. Indeed, these devices are much like cell phones without their cover and provide bigger General Purpose Input Output (GPIO) pins and other connectivity ports. Such devices, of diverse capabilities, ranging from Arduino [4] units to Raspberry Pi [15] ones, can be used in many embedded systems or IoT projects.

The benefits that similar systems might have in education weren't left unexploited. Indeed, during the last years, we are witnessing a very successful osmosis between innovative and cost-effective credit card - sized computers and education. These computers, equipped with low cost sensors or actuators, can be the "heart" of various DIY robotic artefacts made by low cost everyday / recyclable materials like cardboard, plastic bottles or broken toys. This environment allows for a mixture of thinking and making activities that can be very meaningful in terms of pedagogy and science [1][2]. Indeed, similar practices, usually referred as STEM or STEAM activities, are applied in many educational institutions, from primary schools up to universities, with most of the effort to focus on students of secondary education age.

Being in line with this movement, the eCraft2Learn EU H2020 [8] research project (duration: years 2017 and 2018) aims to introduce digital fabrication and maker movement in formal and informal education settings. The eCraft2Learn/H2020 research project researches, designs, pilots and validates an ecosystem based on digital fabrication and making technologies for creating computer-supported artefacts. The project aims at reinforcing learning and teaching in science, technology, engineering, arts and math (STEAM) education and the development of 21st century skills that promote inclusion and employability for youth in the EU. The eCraft2Learn ecosystem aspires to support both formal and informal learning.

In the frame of the eCraft2Learn research project a learning ecosystem is being designed offering technologies such as robotics, DIY electronics, 3D modeling & printing and visual programming tools supported by a constructionist learning methodology [7][13]. The eCraft2Learn ecosystem is used to support pilot activities with students targeting on how to make robots and other computer – supported artefacts from scratch [16]. In the framework of the eCraft2learn project, local labs were established and run project pilots in Athens (Greece) and Joensuu (Finland) in formal and informal education settings. In addition to student pilots, the eCraft2Learn labs are used for training teachers who act later as coaches in student pilots.

The overall process, although promising at the beginning, is not always straightforward to keep up with. More specifically, as students get more experience, they develop a "hunger" for more complicated scenarios that usually demand features like remote interaction with simple Artificial Intelligence [5] capabilities or sophisticated control of their robotic artefacts. At this moment, trainers should be able to propose simple and stable techniques to their students for implementing such features in their constructions, which is not always easy. Indeed, although there are many quite cost - effective radio link modules available for boards like the Arduino, it is not always straight forward for the inexperienced user to make the necessary wiring as these modules usually demand extra electronic components to be added. Furthermore, ordinary tablet or cell phone users, are not familiar with radio links different than the trivial Wi-Fi (i.e., the IEEE 802.11 [22]) and are unwilling to spend extra money, in

order to buy equipment (at least in pairs) to remotely control their robotic artefacts. But, even when using ordinary Wi-Fi modules, one can be far from being efficient. Indeed, connection methods through VNC (i.e., Virtual Network Computing [21]) although straightforward to use, require a large amount of computational power and bandwidth to run flawlessly, but this is not an optimal choice when creating autonomous robotic artefacts.

This paper proposes flexible methods for this remote interaction and control process to be done flawlessly, via the familiar Wi-Fi links, by exploiting the very popular MIT App Inventor [11] or the Snap! [18] visual programming environments, in conjunction with a modified tiny web server module, written in Python [14], that runs on a Raspberry Pi credit card - sized computer.

The crafting experience that has been gained during the eCraft2Learn project is also reported as good practices have been tested to make robust enough robots by low cost everyday / recyclable materials like cardboard, wood, plastic bottles or broken toys.

Although the eCraft2learn project has initially been used to support pilot activities with students aged from 13 to 17, the techniques being studied and refined during this project are applicable to students of universities as well, after small necessary adaptations. Such successful tests have initially been carried out with student-teachers who attend the post-graduate course STEM Education, co-organized by the University of Patras and the National Kapodistrian University of Athens [3], and lately with students of the Agricultural University of Athens (Laboratory of Agricultural Engineering).

The rest of the paper is organized so as to provide the architecture being followed to support a flawless and easy remote interaction / control of the robotic artefacts (in section 2), to highlight useful implementation details (in section 3), to present some characteristic examples (in section 4) and, finally, to conclude on the overall process and give directions for future work (in section 5).

2 Architectural Overview

One of the most crucial components of the eCraft2Learn ecosystem is the Raspberry Pi (model 3 B) single board computer (RPi3). This unit appears to have more than one important role.

Indeed, a set of Raspberry Pi 3 units are serving as experimental development computers (workstation units) for the students. These units are equipped with TFT screens and keyboard-mouse sets. The adoption of this solution tackles the problems related to the vast software and hardware diversity characterizing a typical School Lab and reduces the Lab's overall energy consumption and, unlike computers and tablets controlled by a school's IT staff, typically teachers may have full control over their Raspberry Pis. Furthermore, a second set of Raspberry Pi 3 units are used for the potential implementation of the designed artefacts (the more complex ones). Of course one or more Arduino boards, connected with a variety of electronic compo-

nents and/or the Raspberry Pi 3 units, to assist in artefact implementation, are provided as well.

The overall ecosystem, in terms of hardware, includes tablet devices to facilitate remote interaction scenarios and various DIY electronic components (e.g. photoresistors, potentiometers, servomotors, LEDs) that will be used in conjunction with the RPi3 and the Arduino units to implement the designed artefacts and various DIY modified parts brought from home during a recycling process, like broken toys, plastic bottles, pieces of paperboard, computer fans, speakers, etc. Moreover, this ecosystem includes 2D and 3D printing equipment, the necessary networking equipment to facilitate the interconnection of the pilot sites and provide further access to the Internet and eCraft2Learn resources, and, finally the necessary power supply equipment like power banks (or small solar panels). The selection of all the hardware components has been done to minimize the cost, the size, the power consumption and maximize the reusability of materials and electronics.

The eCraft2Learn ecosystem, in terms of software, includes a set of tools of diverse capabilities that have been selected to meet a set of very welcome characteristics. More specifically, they can run on a RPi3 environment, they have reduced need of installation or update of software elements, have a user-friendly interface, are pedagogically meaningful, are easy to integrate with the external hardware, are open source and free (or at least have low cost) and, finally, are preferably capable of supporting Artificial Intelligence cloud services. This basic set of software tools is extensible with more tools dedicated to more advanced users.

Amongst the above mentioned software tools are visual programming tools like the Snap! (and its variants) that allows custom visual block creation, provides easily exportable/importable XML code and cooperates with many web-based tools for useful data exchange. As the vast majority of today's students are very familiar with smart phones / tablets, another outstanding tool is the MIT App Inventor that allows for rapid mobile application creation even by inexperienced users.

As students are getting more and more familiar with the capabilities and strengths of the abovementioned ecosystem they usually ask to create more sophisticated artefacts and tend to move to more composite scenarios of human – robot interaction. At this point, both MIT App Inventor and Snap! can provide satisfactory solutions, in conjunction with the RPi3 units and a few lines of code written in Python (that might remain hidden from the final users or not), without increasing complexity or relying upon hiring “exotic” radio interfaces. Furthermore, the utilization of a quite powerful unit (like the Raspberry Pi) inside the robotic artefacts allows for more composite human – robot interaction.

The key factor to achieve that is to take advantage of the HTTP request / reply messaging mechanism that most today's systems are supporting as a common feature. More specifically, both MIT App Inventor and Snap! support such a mechanism and, in order the RPi3 unit to be able to intercept and handle HTTP requests, Python implemented HTTP server code has been modified appropriately and adjustments have been made so as this Python code to be executed as a service on the RPi3 unit (i.e., to start automatically after powering up the RPi3 unit).

Snap! and many of its variants such as Snap4Arduino, NetsBlox, and BeetleBlocks can read the contents of URLs. New blocks can be defined to behave just like the Snap4Arduino [19] blocks that read and write digital pins of Arduinos. For example the Raspberry Pi's digital pin 5 can be written by a block that can write any pin by constructing URLs such as localhost/writePin/5/on. The small Python web server then turns on the pin. To read a pin a URL such as localhost/readPin/5 is generated and depending upon whether the contents of the URL are 1 or 0 will report the appropriate Boolean value.

Instead of localhost one can use raspberrypi.local instead. The Raspbian operating system treats this the same way as localhost, when fetching pages from the Raspberry Pi. But it also enables access to the Python-implemented web server from any device on the local area network. This means, for example, that a laptop on the local network can read and write the Raspberry Pi pins using the Snap! blocks via Wi-Fi. This enables programming the Raspberry Pi without connecting a display, keyboard, and mouse to it. Furthermore, if the project involves machine learning it can rely upon the browser's access to the GPU to speed up computations over one hundred times over the speed of the Raspberry Pi.

Fig. 1. depicts the interoperation among the various modules supporting the robot control actions.

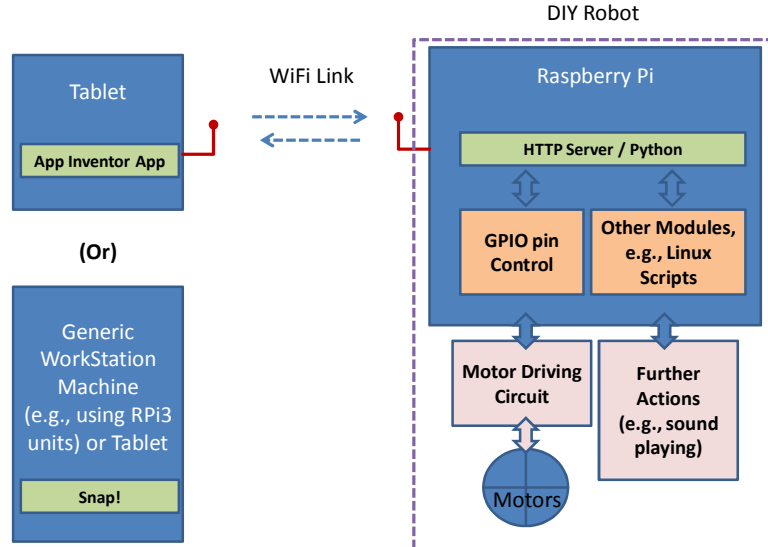


Fig.1. Diverse modules interoperation to support the robot control functions.

3 Implementation Details

In further detail, a simple HTTP server code like the one described in [6] has been combined with the RPi.GPIO Python library [16] so as to control the GPIO pins of the RPi3 board. This Python related part, most of which may remain transparent for the

learners, provides the ability to handle the necessary HTTP requests (of GET, PUT or POST type) into real (remote) interaction cases. Fig. 2. depicts indicative code (in Python) that controls the status of GPIO pins of the RPi3 unit (e.g., by handling the `http://127.0.0.1:42101/GPIO_18_ON` request). The Python code includes separate cases that can be further extended so as instead of simply altering the status of some GPIO pins to call custom Linux (bash) scripts performing even more sophisticated actions like sound/music invocation or battery status data acquisition. Indeed by importing `os` [12] library in python code we can invoke any system command (or script) to be executed via the `os.system()` method. For instance by including the `"os.system("mpg321 ScorpionsWind.mp3")"` command in our python code we invoke the song `ScorpionsWind.mp3` to start playing (under a specially designed remote command of course).

This Python code can be set up to run as a service that start automatically each time the RPi3 units is powered on, by adding an entry in the `/etc/rc.local` file and/or the Linux crontab.

```
import time
import BaseHTTPServer
import cgi
import RPi.GPIO as GPIO

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.cleanup()
GPIO.setup(17, GPIO.OUT)
GPIO.setup(18, GPIO.OUT)
GPIO.setup(22, GPIO.OUT)
GPIO.setup(23, GPIO.OUT)
#more GPIO pins can be configured here ...

HOST_NAME = '0.0.0.0'
PORT_NUMBER = 42101

class MyHandler(BaseHTTPServer.BaseHTTPRequestHandler):

    def do_HEAD(s):
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()

    def do_GET(s):
        category, pin, action = s.path.split("_");
        pinnum = int(pin)
        print category, pinnum, action
        if action=='ON':
            GPIO.output(pinnum, GPIO.HIGH)
        elif action=='OFF':
            GPIO.output(pinnum, GPIO.LOW)
        ...

    ...
    if action=='FD':
        GPIO.output(17, GPIO.HIGH)
        GPIO.output(18, GPIO.LOW)
        GPIO.output(22, GPIO.HIGH)
        GPIO.output(23, GPIO.LOW)
    elif action=='BK':
        ...
    elif action=='ST':
        GPIO.output(17, GPIO.LOW)
        GPIO.output(18, GPIO.LOW)
        GPIO.output(22, GPIO.LOW)
        GPIO.output(23, GPIO.LOW)
        #response2client
        s.send_response(200)
        s.send_header("Content-type", "text/html")
        s.end_headers()
        return

if __name__ == '__main__':
    server_class = BaseHTTPServer.HTTPServer
    httpd = server_class((HOST_NAME, PORT_NUMBER),
    MyHandler)
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()
```

Fig.2. Indicative code in Python handling HTTP (GET) requests and activating the corresponding GPIO pins on the Raspberry Pi unit of the robot.

The eCraft2Learn learners were now able to create simple HTTP requests, allowing for direct access/control of the GPIO pins of either local or remote RPi3 units. For this to be done, the HTTP block functionality in Snap!, that is available under the “Sensing” category via the “url ...” block, was necessary. By forming suitable HTTP requests, a LED can be turned on and off or a robot commanded to move back and forth, as depicted in Fig. 3.



Fig.3. Suitable HTTP requests in App Inventor, to turn a LED on and off or to command a robot to move back and forth.

Multiplayer games and projects can be created by arranging for multiple Snap! instances to communicate with the same Raspberry Pi. This relies upon the fact that the Raspbian operating system supports the detection of Raspberry Pis on the local area network unless default settings have been overridden.



Fig. 4. The definition of blocks in Snap! compatible with Snap4Arduino's digital pin blocks.

Unlike Snap4Arduino that requires access to the serial port connected to an Arduino the use of local web servers means that the Snap! blocks for accessing the Rasp-

berry Pi's digital pins work in all modern browsers without installing any extensions. Moreover, by defining the Snap! Raspberry Pi blocks to match exactly the digital pin blocks of the Snap4Arduino, projects can be developed so as to work equally well without custom editing on Arduinos or Raspberry Pis. Fig. 4 shows an implementation of blocks compatible with Snap4Arduino (it uses a slightly different local web server).

During the eCraft2Learn project very important Artificial Intelligence (AI) features have been implemented [9][10]. These features like voice command reception can be easily combined with the remote control capabilities described here to provide pedagogically fruitful examples of human – robot interaction. This relies upon the speech recognition API implemented in the Chromium browser. Image recognition can also be performed in Snap! from the browser if the appropriate API keys are provided.

Snap! machine learning blocks can run on a Raspberry Pi but is typically slow. However, more powerful computers can use machine learning and control the pins of a Raspberry Pi using the raspberrypi.local domain feature.

The AI cloud service Snap! blocks range from very simple interfaces to fully functional ones with many options. For a majority of projects the simple interfaces are adequate. For example, speech recognition can be as simple as using the “the next thing spoken” reporter (Fig. 5. at top left.) which reports a text string for the next thing spoken. For more complex applications one can use a specially designed block (Fig. 5. at the middle.) where Snap! blocks can be placed in any of the “rings” to receive the appropriate inputs. Similarly for image recognition one can obtain a list of possible labels using the reporter depicted at the bottom right part of Fig. 5.

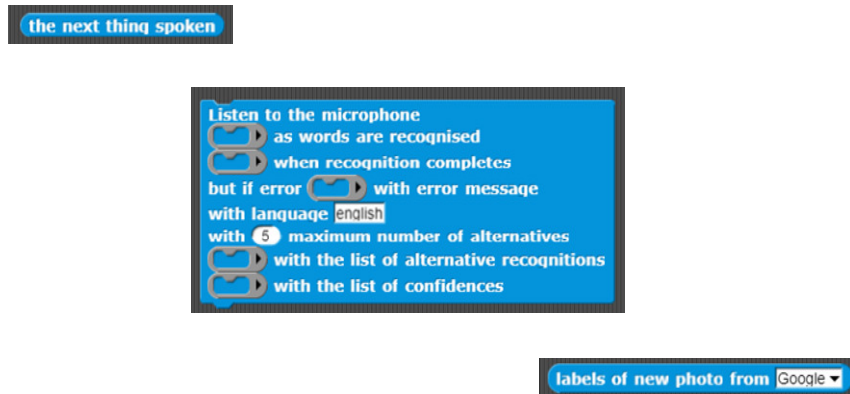


Fig.5. Indicative use of AI handling blocks designed and implemented to work with the Snap! programming environment.

The architecture we developed splits responsibilities in order for a Snap! program to access both AI cloud services and the sensors and actuators connected to the Raspberry Pi's pins. The new Snap! blocks we authored in JavaScript make HTTP(S) requests to both the local web server implemented in Python and to cloud services hosted by companies such as Google, Microsoft, and IBM. The local web server is

only used to access the pins of the Raspberry Pi. The blocks communicating with cloud services can access the camera and microphone from the browser in the standard manner that any webpage does so. Typically the first time permission must be granted. The blocks then capture images and audio and send them to the cloud service with HTTPS POSTs.

The Snap! AI blocks are described in more detail in [9]. They have been the focus of AI programming workshops in six countries involving over one hundred students ranging from 8 to 17 years old. The children 8 to 12 years old successfully used the AI blocks due to their prior experience with Scratch programming. Some of the high school students succeeded despite little or no prior programming experience.

Our architecture relies upon a typical networking setup including a router / access point device to maintain a wireless LAN (containing both the tablets and the RPi3s) and to provide access to the Internet (i.e., to the Cloud services) via a WAN line (e.g., a DSL line). In case that tablets are equipped with their own 3G/4G interface, typical tethering techniques can be used to provide access to the RPi3 unit and the Internet and thus, no extra router is needed. Typically, the single wireless network interface card, which is standard in the RPi3 units, is sufficient. The HTTP(S) requests to the local web server are handled by the Raspbian operating system, without using any dedicated network hardware, via the loopback interface (i.e., using the localhost address which is by default the 127.0.0.1). In case that Snap! runs in a different device (either a tablet or a RPi3 unit) than the RPi3 unit having the GPIO pins of interest, the localhost address should point at the IP address of this specific RPi3 unit. Apparently, more complicated networking schemas can be applied but such configurations do not further highlight the objectives of this research work and thus are omitted.

The students can use the Snap! blocks for reading and writing the pins of a Raspberry Pi without understanding how they work by making requests to the local web server. However, the idea that a URL that indicates that a pin should be read will respond with the state of the pin and another URL for writing a pin will turn on or off a device connected to the specified pin is not difficult for the students to understand as modern students, even the younger ones, are quite familiar with web environments thus to their everyday social media and gaming activities.

The simple custom mechanism that was added to the RPi3 unit to handle HTTP requests is beneficial not only for the Snap! environment but for any other tool able to generate HTTP requests in a pedagogically meaningful manner. So, as a next step, tasks similar to the above ones were practiced in the MIT App Inventor environment. More specifically, the learners generated HTTP requests in order to remotely control GPIO pins of their RPi3 unit, through their smartphones or tablets.

They were then invited to experiment with the Accelerometer sensor which is built in any modern smart phone in order to turn on and off the GPIO18 pin of the local RPi3 or the same pin of the RPi3 unit of their neighboring team or to drive a robot as they did via the Snap! software. Fig. 6. depicts an indicative user interface layout and code part (written in MIT App Inventor) that controls two DC motors connected on pins 17,18 and 21,22 of the RPi3 respectively (using the necessary driving circuit of course, like the L293D one). The typical robotic artefact construction being made has

one independent motor per side. At the bottom of Fig. 6, code that controls one separate GPIO pin is presented.

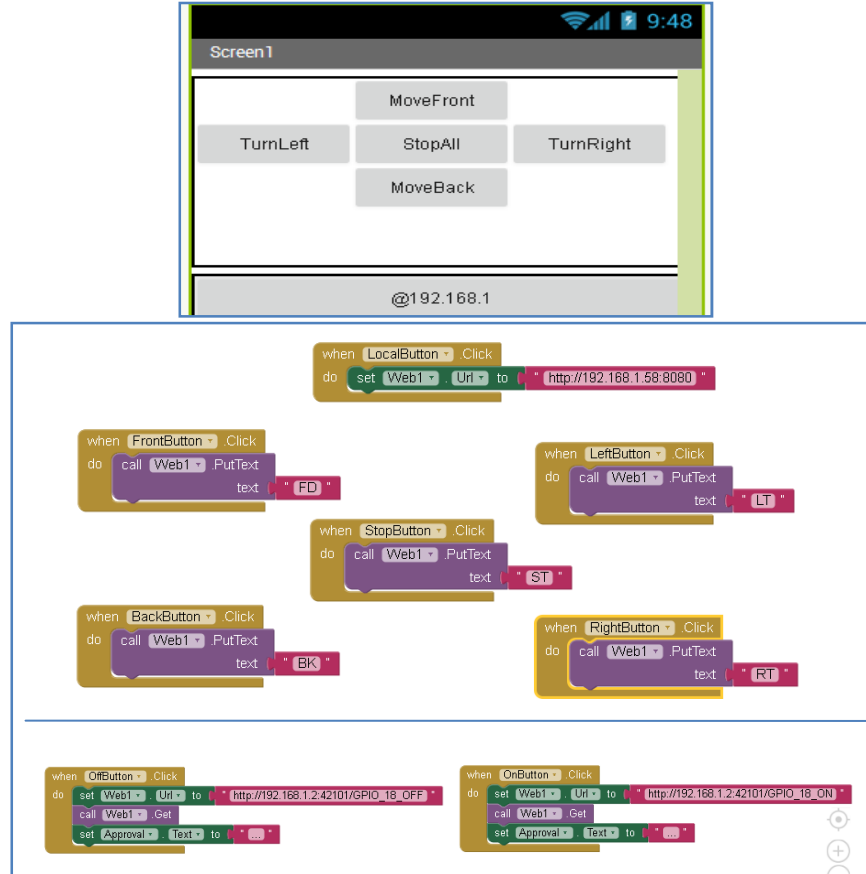


Fig.6. Indicative user interface layout and code, in MIT app inventor, sending the suitable HTTP requests each time the corresponding virtual button is pressed on the smart phone's screen.

Potential bottlenecks to the above mentioned techniques may rise due to the security settings that some browsers have eliminating access over unencrypted HTTP messages in favor of HTTPS ones. This is an apparent problem in case of Snap! and can be solved either by replacing the HTTP server in Python with an equivalent HTTPS version. The necessary security certificates should be provided then in the RPi3 unit inside the robotic artefact. An alternative solution is to implement pure UDP (i.e., User Datagram Protocol [20]) code for client / server interoperation, but this goes beyond the scope of this paper.

4 Characteristic Examples

This section is dedicated to presenting some characteristic robotic artefacts made by the eCraft2Learn learners (both secondary school students and university students), that are low cost, easy to make and combine simplicity and robustness in their construction and operation.

The 1st example is a cardboard robot having separately driven wheels by small DC motors that are glued at each side, plus a free wheel. The communication and control is carried out by a RPi3 unit and its built in Wi-Fi module. Pins 17, 18 are controlling the left side motor while GPIO pins 21, 22 the right side motor. A pair of L9110S motor control chips (on the same board) is used to amplify the signals of the GPIO pins to efficiently drive the motors. The whole system is powered by a solar power bank. This robot was easily constructed along with its client software, made using the App Inventor tool, in only two sessions of two hours each. The activities took place in Athens eCraft2Learn informal pilot site in June 2018 by a team of three secondary school students.

The 2nd example is a minimal robot made by a small plastic box and some tiny pieces of wood. The overall architecture remains the same as in the previous case except that the driving circuit is a L293D one. The same set of RPi3 pins have been used to control the two motors. The robot has been tested using an App Inventor made mobile app and separate HTTP messages through a conventional web client (e.g., Mozilla Firefox). This example, initially started with secondary school students, served as an introductory project so as students of the Agricultural Engineering laboratory of the Agricultural University of Athens to demystify IT technologies and visual block programming.

The 3rd example of robotic construction has recently been made at the Agricultural Engineering laboratory of the Agricultural University of Athens, in order the students become familiar with techniques and machines used in modern agriculture under scale. Apart from its size, its design and construction is following the same principles as the two previous examples, with wood to be the dominant material. The driving circuit now is based on the L298N chip that is able to handle higher current values.

In all the above three cases, the typical robotic artefact construction being made has one independently controlled motor per side. This setup eliminates the need for extra mechanisms dedicated in steering tasks and leads to more robust artefacts that are able to maneuver / turn in narrow areas just by changing the rotation direction of their side wheels.

Learners, supported by their trainer, worked in high interest, realized successfully and enjoyed their projects. Team members had discrete and diverse roles (e.g., crafting, programming, presentation) and joined their powers to accomplish the project.

Fig 7. depicts these three characteristic DIY robot cases: Bottom left picture corresponds to the 1st example. Top middle corresponds to the 2nd example. Bottom right picture corresponds to the 3rd example.

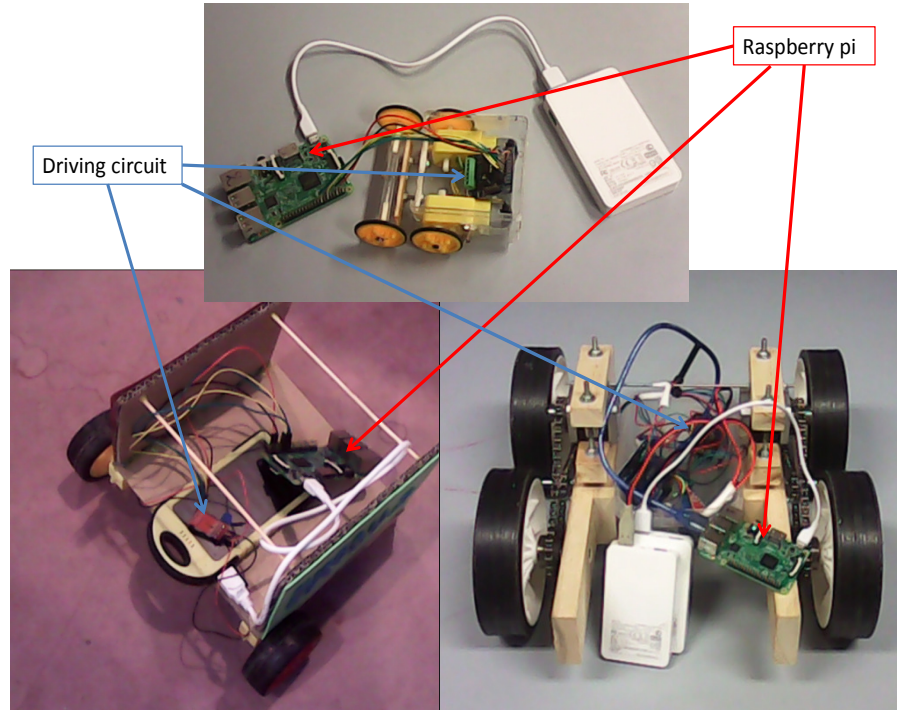


Fig.7. Examples of DIY robots of diverse size that are all sharing the same architecture and exhibit cost effectiveness and ease to use and handle.

5 Conclusions and Future Work

This paper proposed flexible methods for remotely controlling and interacting with DIY robotic artefacts by exploiting the very popular MIT App Inventor or the Snap! visual programming environment, in conjunction with a modified tiny web server module, written in Python, that runs on a Raspberry Pi credit card - sized computer. The proposed techniques provide solutions in situations where students involved in STEM / STEAM education activities, as they get more experience, want to realize more complicated scenarios that usually demand features like remote interaction with or sophisticated control of their robots.

This research work exploited the experiences gained during the eCraft2Learn EU research project that aims to introduce digital fabrication and maker movement in formal and informal education settings, and reported on simple techniques being used to make robust enough robots by low cost everyday / recyclable materials like cardboard, wood, plastic bottles or broken toys. The techniques being presented, although initially planned for secondary school students have successfully been tested with university students as well. The experiences gained from these first tests with learners have provided encouraging indications that the proposed scenarios and techniques can

help educators and trainers introduce fast development and remote control of DIY robots and finally promote further the making culture in educational robotics.

In the near future, we are planning to study and highlight more advanced interaction scenarios involving a larger variety of robotic artefacts and humans, always keeping feasibility and cost at very affordable level.

Acknowledgement: This research was supported by the eCraft2Learn project funded by the European Union's Horizon 2020 Research and Innovation Action under Grant Agreement No 731345.

Disclaimer: This communication reflects the views only of the authors and the European Commission cannot be held responsible for any use which may be made of the information contained therein.

References

1. Alimisis, D., Moro, M., Menegatti, E., Preface in D. Alimisis et al. (eds.), *Educational Robotics in the Makers Era*, *Advances in Intelligent Systems and Computing* 560, DOI 10.1007/978-3-319-55553-9_1, Springer (2017)
2. Alimisis, D. (2013). Educational Robotics: Open questions and new challenges. *Themes in Science and Technology Education*, 6(1), 63-71.
3. Alimisis, D., Loukatos, D., STEM education post-graduate students' training in the eCraft2Learn ecosystem, 2nd International Conference on Innovating STEM Education, Athens (2018) (in press)
4. Arduino. Retrieved in May of 2018 from the site: <https://www.arduino.cc/>
5. Artificial Intelligence – AI. Retrieved in May 2018 from the site: https://en.wikipedia.org/wiki/Artificial_intelligence
6. BaseHTTPServer. Retrieved in May 2018 from the web site: <https://wiki.python.org/moin/BaseHttpServer>
7. Blickstein, P. Digital Fabrication and 'Making' in Education: The Democratization of Invention. In J. Walter-Herrmann & C. Büching (Eds.), *FabLabs: Of Machines, Makers and Inventors*. Bielefeld: Transcript Publishers (2013).
8. eCraft2Learn EU H2020 project. Retrieved in May of 2018 from the site: <https://project.ecraft2learn.eu>
9. Ken Kahn and Niall Winters, AI Programming by Children, Constructionism Conference, Vilnius, Lithuania, August 2018. (to appear in)
10. Kahn K., Winters N. Child-Friendly Programming Interfaces to AI Cloud Services. In: Lavoué É., Drachsler H., Verbert K., Broisin J., Pérez-Sanagustín M. (eds) *Data Driven Approaches in Digital Education*. EC-TEL 2017. *Lecture Notes in Computer Science*, vol 10474. Springer, Cham (2017)
11. MIT App Inventor. Retrieved in May 2018 from the web site: <http://appinventor.mit.edu/explore/>
12. OS - Miscellaneous operating system interfaces (Python). Retrieved in May of 2018 from the site: <https://docs.python.org/3/library/os.html>
13. Papert, S., & Harel I. Preface, *Situating Constructionism*, in Harel, I. & Papert, S. (Eds.), *Constructionism, Research reports and essays, 1985- 1990* (p. 1), Norwood NJ (1991).
14. Python Programming Language. Retrieved in June 2018 from the site: [https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

15. Raspberry. Retrieved in May of 2018 from the site: <https://www.raspberrypi.org/>
16. RPi.GPIO module basics (Python). Retrieved in June 2018 from the site: <https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/>
17. Schon, S., Ebner, M., Kumar, S. The Maker Movement Implications from modern fabrication, new digital gadgets, and hacking for creative learning and teaching, In Laia Canals, P.A.U. Education (Ed.) eLearning Papers Special edition 2014 p. 86-100 http://www.openeducationeuropa.eu/en/article/Learning-in-cyber-physical-worlds_In-depth_39_2
18. Snap! Retrieved in May of 2018 from the site: <https://snap.berkeley.edu/>
19. Snap4Arduino. Retrieved in May of 2018 from the site: <http://snap4arduino.rocks/>
20. UDP – User Datagram Protocol. Retrieved in June of 2018 from the site: https://en.wikipedia.org/wiki/User_Datagram_Protocol
21. VNC – Virtual Network Computing. Retrieved in June of 2018 from the site: https://en.wikipedia.org/wiki/Virtual_Network_Computing
22. Wi-Fi – IEEE 802.11 Standard. Retrieved in May of 2018 from the site: <http://www.ieee802.org/11/>