Teaching Modern C++ with Flipped Classroom and Enjoyable IoT Hardware

Ulrich Sch¨afer
Ostbayerische Technische Hochschule Amberg-Weiden (OTH-AW)
Faculty of Electrical Engineering, Media and Computer Science
Amberg, Germany
Email: u.schaefer@oth-aw.de

***Abstract*—The C++ programming language has been transformed into a modern one over the past decade, and has gone through considerable improvements. So have teaching methods for undergraduate students. We describe a novel approach to teaching modern C++ (from C++11 onwards) to electrical engineering and computer science students. The teaching concept uses flipped classroom and the practical exercises are centered around an Internet of Things (IoT) device. The main motivations are (1) guide students to a life-long learning mode, (2) increase the time to practice software development in presence of a trainer, (3) motivate programming practice by using attractive IoT hardware, (4) focus on modern C++ with shared and unique pointers, templates, standard library, containers and concurrency.**

Keywords*—*software development; flipped classroom; inverted classroom; Internet of Things; modern C++; electrical engineering; computer science; sensors; embedded systems; SBC; single-board computer

I. Introduction and Motivation

The C++ programming language has been transformed into a modern one over the past decade, and has received considerable improvements [1]. It is important that students nowadays learn this new flavor which differs considerably from old coding styles. At the same time, new teaching methods have been developed. This paper describes how both can be combined in a novel approach to teaching modern C++ (from C++11 onwards) to electrical engineering or computer science students.

The teaching concept uses flipped classroom [2], [3]. The practical exercises are centered around an Internet of Things (IoT) system: by using attractive hardware such as Raspberry Pis with SenseHAT, a hardware attached on top (HAT) with sensors, buttons plus an 8x8 RGB LED matrix, students can be better activated and motivated. The HAT carries everything on one board, no soldering or wiring is required.

The main motivation for using flipped classroom is to guide students to a life-long learning mode. It also helps to increase the time to practice software development in presence of a trainer instead of passively sitting in a lecture. Finally, the contents of a modern C++ course focus on C++ with shared and unique pointers, templates, standard library, containers, strings and concurrency, and combines them with the physical feedback of the IoT hardware with sensors.

It has to be pointed out that our concept focuses on *learning the C++ programming language*, not on the Internet of Things as such. IoT devices are used here as a means for doing interesting things related to hardware, because this is one of the main applications of C++ and for the addressed audience.

Although IoT hardware nowadays is often even capable of running higher-level programming languages such as Python, Java or JavaScript, the C++ language has some advantages such as very low memory footprint at runtime, explicit control of memory allocation if wanted as well as closeness to hardware (low-level drivers for sensors etc.). Therefore, and because the described course also serves as a prerequisite to an embedded systems lecture, C++ was chosen.

The paper is structured as follows. In Section II, we briefly sketch the idea of flipped (inverted) classroom. In Section III, we discuss related work and some basic learning theory. In Section IV, we present the overall course plan, followed by the reading plan in Section V. In Section VI selected exercises that combine modern C++ programming and IoT hardware are described. Section VII shows a first evaluation of the course. In Section VIII, we describe a peer instruction as supporting activity example and finally give a conclusion and outlook to planned future course structure in Section IX.

II. Flipped Classroom

To save time to discuss questions and spend more time on practical programming exercises, the course has been, after many years of classical lecture plus exercises, evolved into a flipped classroom course.

Flipped classroom [2], [3] (or synonymously *inverted classroom*) means that reading a textbook, instead of passively sitting in lecture, was introduced as a mandatory, preparatory homework. Presence time at university became time to ask, discuss and answer questions concerning the previously read book chapters, interleaved with practical programming sessions where appropriate.

In contrast to [3], we conceive flipped classroom in its traditional sense which includes reading texts. We reject the sometimes uttered thesis that ”students do not read”. Given motivating programming tasks, they do. However, continuous contact and feedback by the teacher is necessary.

Moreover, in case of programming courses, watching videos can be an asset for some topics, but never completely replace a book. Electronic versions of books where examples can be directly copied and pasted from e.g. the PDF into the

978-1-5386-9506-7/19/$31.00 ©2019 IEEE 9–11 April, 2019 - American University in Dubai, Dubai, UAE
2019 IEEE Global Engineering Education Conference (EDUCON)

development environment (if copy & paste works correctly), we consider more useful than separate downloadable archives of source code.

Another rationale for not putting video lecture too much in front is life-long learning: there will not be a video lecture for every subject the then graduated students will have to learn during their lives. Often, reading (e)books, draft documents or manuals is part of the work of a developer. Therefore, reading is essential for live-long learning and should be practiced at university.

III. Related Work

Using IoT elements as a means to get students more engaged has e.g. been proposed by [4]. Moreover, we had extremely positive feedback from own numerous *Physical Computing* student projects over four years with IoT SBC (single-board computer) devices. Therefore, we decided to adopt the idea of using IoT hardware, but, to put programming in front, without wiring or soldering.

[5] is a useful textbook for developing device drivers for Raspberry Pi, and even covers some modern C++. However, we found it too low-level for explaining many more aspects of modern C++, especially to C++ beginners. Therefore, all exercises below had been developed independently of this book.

There is few published work on teaching C++ with the flipped classroom method, most works are of the style of [6] on computer science or programming in general, none covering modern C++ explicitly. However, there is literature on related subjects such as teaching Matlab with flipped classroom [7]. This article also nicely explains the rationale behind flipped classroom: the two stages of learning, *transmission* and *assimilation*.

The lecture part (called *transmission* in learning theory) in a classical lecture is cognitively easier but also inflexible: some students understand faster, some find it more difficult to follow what is said by the teacher and need more time. This could be better handled by reading at individual speed. Watching video is similar: a video can be paused, rewinded and repeated as often as necessary.

In contrast, the exercise part (called *assimilation* in learning theory) is cognitively more challenging, but, especially in case of programming software or doing math exercises, would benefit much more from the presence of a trainer who can answer questions or give hints individually. According to this rationale, classical lecture is odd if there is not enough time left to solve at least parts of the exercises in presence of the trainer.

Some of the prerequisites in [7] differ from our setup: the audience is very homogeneous and appropriate textbooks on modern C++ exist for the reading part. However, the list of three findings towards the end of that paper is useful:

- provide pre-class assignments with clear learning objectives and good structure,
- give a simple assessment of a "pre-class assignment" as an "entry ticket" to the in-class session, e.g. a five-minute clicker quiz. This can also generate an important instantaneous feedback for the students as well as for the instructor (cf. Section VIII below),
- set clear ground rules about what will happen in presence time: no lecture, no "re-teaching".

We would like to point out that our concept focuses on learning the modern version of the C++ programming language, while other approaches using IoT hardware described in literature focus on physical computing or IoT as such, using other or even different programming languages in one course as examples how programming works [8], [9]. Our proposed SenseHAT hardware which will be described in Section VI makes it easy to concentrate on programming: No wiring is required because sensors, keys and displays are integrated on the circuit.

IV. Course Plan

Our course concept foresees a one-semester course of approx. ten weeks with 90 minutes for discussion and questions (replacing the former lecture) and 180 minutes per week for practical programming exercises, followed by another five weeks with 270 minutes each for a programming project. These times are presence times, about the same amount of time is foreseen for self study, for reading the text book and preparation of exercises, project and exam.

The students in our case had two preceding lectures on C programming and software development basics so that the C++ course can concentrate on the object-oriented programming paradigm, templates and newer C++ features such as standard type library, smart pointers, containers, lambda, strings and algorithms. The first week mainly contained a repetition of the C language (data types, expressions, control structure, etc.), cf. Table I.

There is a special issue of the MagPi magazine ("Essentials: Learn to Code with C"; free PDF download at [10]) that introduces the basics of the C programming language with the gcc compiler. The examples are demonstrated on a Raspberry Pi but without specific libraries or hardware requirements. This textbook can be used as preparation for those C++ textbooks that require previous knowledge of the C programming language. It is more convenient for beginners to read than [11] though not a full replacement.

The course and reading plan largely follows the structure of the text book [12]. This is a new book that provides a fresh and modern approach to introduce C++ 14, yet without any relation to IoT SBC devices. The book omits C++ legacy as much as possible and does not require prior knowledge of C or C++. An English counterpart may be [13] which however requires prior knowledge of (legacy) C++.

V. Reading Plan

In the beginning (actually two months before the semester started), a reading plan was published on the local instance of the open source learning management system Moodle (http://moodle.com) for the course. The reading plan indicated which exact chapter was to read until which week.

TABLE I
COURSE PLAN

Week	Topics
1	I/O, data types, literals, operators, control structure, comparisons, auto, const, constexpr, enum
2	files, declaration and definition, interface and implementation, header files, preprocessor directives, name spaces, reference parameters, recursion, extern, array, vector, assert
3	object-oriented programming paradigm, classes, objects, constructors, (element) initialization, visibility, explicit, copy constructor, destructor
4	static (class variables and functions), pointer, pointer arithmetic, C arrays, dynamic memory allocation, parameters, new and delete, smart pointers: unique_ptr and shared_ptr, inheritance, polymorphism, abstract classes, virtual, pure virtual functions, multiple inheritance
5	error handling, exceptions, operator overloading, deep and shallow copy
6	template functions and classes, function objects, lambda
7	pairs, iterators, containers (vector, list, set, map, etc.), threads
8	bit vectors, algorithms, complex numbers
9	UML, Doxygen, OOP design patterns 1
10	design patterns 2, regular expressions
11-15	programming project

The textbook was made available as chapter-wise downloadable licensed PDFs in Moodle but a cheap printed copy is also available [12]. Each chapter ends with control questions, the solutions are in the book.

The book also contained separate chapters in which a game was developed. Most of this material was skipped in favor of programming exercises with IoT hardware which we will describe in the next section. These exercises are not part of the book but had been developed by the paper author.

A more comprehensive textbook [14] was recommended as an additional reading source for details and advanced topics such as threads or atomic variables. It was also made available through the learning management system with links to the relevant chapter PDFs. A comparable, but less up-to-date English counterpart is [15].

VI. Practical Exercises with IoT Hardware

The motivation behind using IoT hardware with sensors is to provide more interesting programming exercises that function in a physical environment and are close to hardware that students like.

At the same time, in an information technology world that becomes more and more complex, it is important to learn and train the abstraction facilities which are provided by modern C++ such as templates, containers, algorithms, shared pointers and that are safer and superior to low-level C programming in many cases.

A. Raspberry Pi

In our practical exercise setup, we used desktop computers as graphical terminals to Raspberry Pis (Model 3B) running the Raspbian operating system which is a variant of Debian Linux (http://debian.org).

To enhance IoT capabilities, the Pis were equipped with SenseHAT extension boards attached on top (Fig. 1). We will present more details on SenseHAT in Section VI-B.

Fig. 1. A Raspberry Pi with the SenseHAT hardware module attached on top, connected to a desktop computer via USB network adapter

For connectivity with the desktop PCs, USB LAN adapters and the RDP (remote desktop protocol) were used to display and control the graphical user interface (X-Windows) of the Pi on the PC screen. It is of course also possible to directly connect keyboard, network and HDMI screen to the Raspberry Pis themselves. Moreover, alternative network protocols such as VNC or the pure X-Windows protocol could be used instead of RDP.

On the Pi, Code::Blocks (Fig. 2; http://codeblocks.org), an integrated development environment, was used for C++ development. If crashes of Code::Blocks (version 16.01) occur during typing, a quick fix is to disable the symbols browser by enabling the checkbox in menu Settings→Editor→Code Completion→Symbols Browser→Disable Symbols Browser. Otherwise, the IDE works very reliably.

An alternative approach would be using a modern text editor such as Sublime Text, Atom or Visual Studio Code on the PC side with `rmate` plugin (a remote editor protocol). One could also use other C++ IDEs with cross-platform compilation capabilities.

Source code management and backup was handled using gitlab (http://gitlab.com) from the command line (`git` command), but also graphical user interfaces for git exist and can be used on the Raspberry Pi.

As an alternative to the Raspberry Pi, it would be possible to use the Arduino platform (http://arduino.cc). By default, it supports C++11, and can be switched to C++14 by compiler settings, and even C++17 may be supported (e.g. in the ESP32 development tool chain). However, depending on the hardware used, there are restrictions to modern C++ usage mainly for lack of memory.

Even more game-oriented hardware suitable for the course would be the Gameboy-like ODROID-GO (http://hardkernel.com, ESP32-based), with considerably more memory and

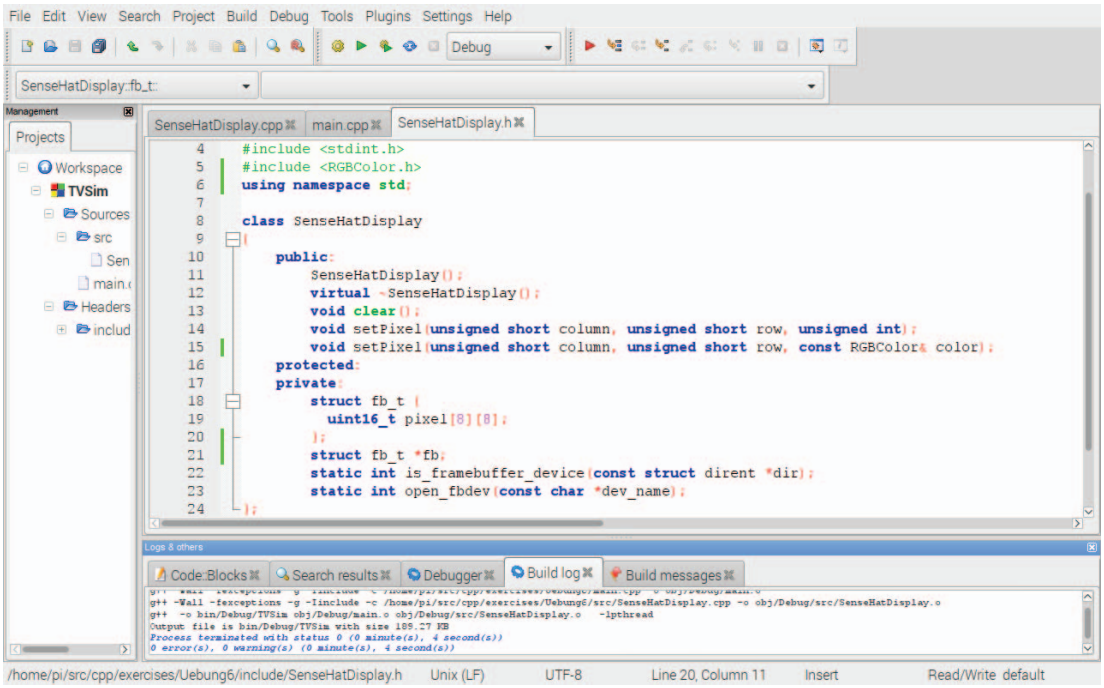

Fig. 2. Code::Blocks IDE

C++ potential than the ATmega32u4-based Arduboy (http://arduboy.com). Both can be programmed using the Arduino IDE or, alternatively, the Atom editor (http://atom.io) with a suitable plugin.

B. SenseHAT

SenseHAT is a ”hardware attached on top”, a compact circuit board to be placed on top of the Raspberry Pi board and connected via the GPIO connector (Fig. 1), with

- temperature, humidity and air pressure sensors
- 9-dof (3x3 degrees of freedom) IMU (inertial measurement units) sensors: accelerometer, gyrometer, magnetic field sensors
- 8x8 RGB LED matrix
- 4 button group.

As this board is relatively expensive (even slightly more expensive than the Raspberry Pi 3 board itself), a cheaper, yet more fragile alternative would be to buy components separately: accelerometer and temperature, humidity sensors, buttons, LED matrix with built-in I2C controller, and place them on a breadboard. While we make use of existing low-level drivers for SenseHAT, similar drivers are developed for individual components e.g. in [5].

For almost all such components, Linux drivers exist out of the box, although they may differ from the ones for SenseHAT we will demonstrate in the following. It might be a good advice to look for IMU sensors that are supported by the RTIMULIB but there are also other libraries.

Although software emulators with GUI for the SenseHAT do exist, they can, to the best of our knowledge, only be programmed in Python, not in C++, and are therefore not suitable for this course.

The following publication may be used as an inspiration to see what is possible with the SenseHAT: MagPi magazine special issue “Essentials: Experiment with the SenseHAT”; free PDF downloadable at [10]. The code examples there are in Python and Scratch, not C++. However, their display and sensor interface are based on the same low-level drivers we will describe below, so the ideas can be transferred.

RTIMULib [16] is a C++ library that encapsulates the access to SensorHAT’s IMU (inertial measurement units), a combination of accelerometer, magnetometer (compass) and gyrometer, each with an x, y and z axis. This is why it is also called a 9 dof (degrees of freedom) sensor. RTIMULib also supports other IMU hardware that is not part of the SenseHAT.

The same library also includes access to the built-in temperature, pressure and humidity sensors. The RTIMULib [16] consists of a well modeled C++ class hierarchy of which the source code itself is a good starting point to learn how (sensor) device drivers can be modeled and implemented in object-oriented C++.

Access to the 8x8 RGB LED matrix is provided through a Linux framebuffer data structure (in C, not part of the RTIMULib) that could be mapped to C++ containers as an exercise (see Section VI-F below).

Both sensors and display, together with a joystick-like button, can be used in motivating programming exercises including games or IoT applications exploiting physical data such as movements or temperature. An example is Marquee text (scrolling text) on the RGB LED matrix that automatically adapts its output direction when the device is rotated. This

application can be developed in a two or three stage exercise, as we will describe below.

C. Exercises on modern C++ with Raspberry Pi and SenseHAT

In the following sections, we describe some selected exercises that make use of modern C++ elements and use the SenseHAT through the above described drivers. The core idea is to abstract from hardware using advanced data structures available in STL/modern C++ such as bit sets, vectors, threads and classes to encapsulate low-level drivers.

The examples form only a small part of all exercises that also cover the standard C++ elements such as data types, constants, auto, I/O, namespaces, exceptions, etc.

For lack of space, we will skip these in most cases. We will rather sketch the exercise ideas focusing on the IoT SBC device than provide fully-fledged exercise texts and solutions. Code is provided here only for some more tricky parts. As usually, there are many possible solutions for these problems.

D. Initial Exercises: GUI, IDE, Compiler, Shell

The initial exercises comprise simple tasks such as learning the Raspbian desktop User Interface called PIXEL which combines the LXDE desktop environment (http://lxde.org) with the Openbox window manager (http://openbox.org).

Furthermore, writing, compiling, running and debugging C++ programs with the Code::Blocks Integrated Development Environment (IDE) is trained. For modern C++ code, the compiler option `-std=c++11`, `-std=c++14` or `-std=c++0x` needs to be enabled in the compiler settings menu.

Also, the Unix/Linux shell is introduced, e.g. to inspect or evaluate the return value of the `main()` function from within `bash` scripts using the `$?` variable. Furthermore, shell commands for creating, changing, renaming, moving files and directories, etc. are explored.

An introduction to the Raspbian command line shell (bash) can be found in the MagPi magazine special issue "Conquer the Command Line"; free PDF download at [10]. Further sources for shell usage introductions are are [17], [18].

E. Exercise: OOP, operator overloading

A simple initial class definition exercise is one that models color representations. In a first step, we define an abstract class Color with some (virtual) methods as well as subclasses that model specific color representations such as RGB, RGB with transparency, YUV etc.

The RGBColor color model subclass for RGB values, e.g., would contain three separate 8-bit values for its red, green and blue components. One part of the exercise is to let the Code::Blocks IDE generate setter and getter method code for the color components member variables (via the menu File - New - Class).

Next, methods and constructors would be defined to convert between different models. This is also a good place to discuss the use of static vs. non-static functions.

Technically, the SenseHAT's 8x8 RGB LED matrix is connected via an I2C bus. The driver is part of the Raspbian package `sense-hat`. It defines a framebuffer data structure that contains a 16-bit RGB value for each of the 64 color LEDs, i.e. 128 bytes in total. Each 16 bit value dedicates its highest 5 bits to the red, then 6 bits to the green and the least significant 5 bits to the blue component. This is also called RGB565 or HiColor model (Figure 3).

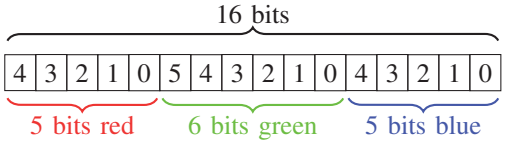

Fig. 3. RGB565 color representation ("HiColor")

Using bit manipulation operators (`&`, `|`, `<<`, `>>`) known from C, an RGB565 value can be composed and decomposed to and from separate eight-bit RGB values. In later exercises, one can study overloaded versions of these operators in the `bitset` container class.

Next, arithmetic operators would be overloaded in the color classes to add and subtract single numeric values or other colors. This would exemplify the global vs. class-attached operator functions in C++.

F. Exercise: OOP encapsulation of framebuffer

Starting point to learn how to access the RGB display is a framebuffer data structure that is demonstrated in the example program `snake.c`. This C source file is contained in the Raspbian package `sense-hat` where it is located in `/usr/src/sense-hat/examples/snake/snake.c`. Unfortunately, this file contains neither code comments nor an author name.

The Raspbian `sense-hat` package can be installed from the command line with `sudo apt install sense-hat`.

Listings 1 and 2 (separated here for paper layout reasons) show the C++ code that defines an object encapsulating the C code for framebuffer access. It is based on fragments from `snake.c`. The code mainly contains the data structure definition of the framebuffer plus a function that opens the Linux framebuffer device and maps it to the memory of the low-level driver.

Our C++ API consists of the constructor that opens the framebuffer and functions to set pixels with a 16-bit RGB565 value or a color object, and to clear the whole display. As an exercise, the C++ wrapper class/API could be developed by the students, with the C code (extracted) from `snake.c` given as input.

The `snake.c` source code also shows how to evaluate the button status of the 5 buttons on the SenseHAT (we omit the details here). Again, this code could be encapsulated in an object-oriented way, and would be useful to control games, e.g.

```
#include "SenseHatDisplay.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <fcntl.h>
#include <unistd.h>
#include <linux/fb.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <dirent.h>
#include <string.h>
#include <linux/input.h>
#define DEV_FB "/dev"
#define FB_DEV_NAME "fb"

using namespace std;

void SenseHatDisplay::setPixel(unsigned short column
    , unsigned short row, unsigned short color)
{ // set color of a single LED
  fb->pixel[column][row] = color;
}

void SenseHatDisplay::setPixel(unsigned short column
    , unsigned short row, const RGBColor& color)
{ // set color of a single LED using Color object
  fb->pixel[column][row] = color.toRGB565();
}

void SenseHatDisplay::clear()
{ // all LEDs off
  for (unsigned short c=0; c<8; c++)
    for (unsigned short r=0; r<8; r++)
      fb->pixel[c][r] = 0;
}

SenseHatDisplay::SenseHatDisplay()
{ // based on snake.c (SenseHAT examples)
  int fbfd = 0;
  fbfd = open_fbdev("RPi-Sense FB");
  if (fbfd <= 0) {
    printf("Error: cannot open framebuffer device.\n
        ");
    goto err_ev;
  }
  fb = (fb_t*)mmap(0, 128, PROT_READ | PROT_WRITE,
      MAP_SHARED, fbfd, 0);
  if (!fb) {
    printf("Failed to mmap.\n");
    goto err_fb;
  }
  memset(fb, 0, 128);
  clear();
  err_fb:
  close(fbfd);
  err_ev:
  printf("SenseHAT init done.\n");
}

SenseHatDisplay::~SenseHatDisplay()
{ // destructor
  clear();
  setPixel(0,0,0x4208);
  munmap(fb, 128);
  printf("SenseHAT memmap released.\n");
}

int SenseHatDisplay::is_framebuffer_device(const
    struct dirent *dir)
{
  return strncmp(FB_DEV_NAME, dir->d_name,
  strlen(FB_DEV_NAME)-1) == 0;
}
```

Listing 1. Class `SenseHatDisplay` encapsulates framebuffer access and defines a C++ API to access it; corresponding header definition in Fig. 2

```
int SenseHatDisplay::open_fbdev(const char *dev_name
    )
{ // based on snake.c (SenseHAT examples)
  struct dirent **namelist;
  int i, ndev;
  int fd = -1;
  struct fb_fix_screeninfo fix_info;
  ndev = scandir(DEV_FB, &namelist,
      is_framebuffer_device, versionsort);
  if (ndev <= 0)
    return ndev;
  for (i = 0; i < ndev; i++)
  {
    char fname[64];
    snprintf(fname, sizeof(fname),
      "%s/%s", DEV_FB, namelist[i]->d_name);
    fd = open(fname, O_RDWR);
    if (fd < 0)
      continue;
    ioctl(fd, FBIOGET_FSCREENINFO, &fix_info);
    if (strcmp(dev_name, fix_info.id) == 0)
      break;
    close(fd);
    fd = -1;
  }
  for (i = 0; i < ndev; i++)
    free(namelist[i]);
  return fd;
}
```

Listing 2. Rest of class `SenseHatDisplay`

G. Exercise: Marquee text, OOP Design Patterns

In this section, we describe an exercise that combines various container data structures (`string`, `map`, `vector`, `bitset`) as well as file handling (reading from a text file).

The goal is to implement Marquee (scrolling) text on the SenseHAT display: Text, e.g. the current time and temperature (the latter from RTIMULib), can be passed to a function that takes the string to be displayed. It would start an infinite loop which scrolls 8x8 (or 5x7) LED character patterns column by column horizontally from right to left, with a delay of some fraction of a second, say 100ms, between the columns.

The first step for the participants is to implement reading in the LED patterns for all possible characters and numbers, e.g. the ASCII character set, from a text file. The patterns for letters A and B are shown here. Each character corresponds to 9 lines in the text file to be read:

```
A           B
00000000    00000000
00111000    01111000
01000100    01000100
01000100    01000100     ...
01111100    01111000
01000100    01000100
01000100    01000100
01000100    01111000
```

The first line contains the character itself, the subsequent eight lines contain the eight line patterns: 0 for LED off, 1 for LED on in some color that can be defined elsewhere and multiplied with the pattern bit to obtain the RGB565 value for the SenseHAT display framebuffer, accessed via the C++ methods defined in an earlier exercise as described above.

In sum, the file would contain $9n$ text lines, where n is the number of characters defined.

Building the mapping from characters to LED patterns can be done via the constructor in the class, named e.g. `SenseHatTextDisplay`. The target data structure could be a C++ `map` container that maps each single character such as `'A'` to an `array` or `vector` of `bitset` of length 8.

Many variants are possible and it is advisable to discuss advantages and disadvantages of the different approaches. An advantage of `bitset<8>` is that it allows to use the overloaded bit shift and or operators.

In a first step, only a single character would be displayed without scrolling. For debugging, the matrix pattern from the map could also be output to the console, i.e. there could be another class that outputs character patterns to the console, cf. right side of Figure 4. If done in a specific way (or optionally using the `ncurses` library), this will later also look like scrolling big (8x8) character text on the a console window.

It is possible to use overloaded shift `<<` and or `|` operators of the `bitset` class to implement scrolling on a `bitset` model which will be flushed to the framebuffer after each scroll step.

Such an in-memory model of the display can also be used as a showcase for various Design Patterns [19] such as the Facade, Proxy, Observer and Model-View-Controller pattern. This would also be the opportunity (if not done earlier) to explain and exemplify the Iterator design pattern, e.g. on C++ containers.

In a later exercise, the code can be extended with IMU sensor evaluation to turn the display content and scrolling direction according to physical movement instantly. This will be a good opportunity to write a simple concurrent program with two threads that run in parallel, one for reading the IMU sensors, one for scrolling the text (Section VI-K) displaying sensor values.

H. Exercise: In situ matrix rotation

A nice example that motivates the use of a matrix model is in situ (in place) rotation: to follow a turned device (by `angle` degrees), the model data structure is modified, and after rotation by 90, 180 or 270 degrees, the new matrix can then be "flushed" into the display's framebuffer. Function `rot90` in Listing 3 shows a possible implementation, based on a Java example from [20] where also diagrams show how the algorithm works. Nicely, brackets are, once more, overloaded index operators in `array`, `vector`, and `bitset`.

I. Exercise: TV simulator

The idea of this exercise is to combine the display code developed and used so far with the `random` class of modern C++. It is also a simple example on how to use object-oriented libraries, `random` and `SenseHatDisplay` in this case.

A TV simulator is a small device that is less power-consuming than a TV screen, but generates similar colorful, randomly changing light with some RGB LEDs in such a way that observers from outside, e.g. potential burglars, may think that one's house is not empty because it looks as if somebody would be watching TV.

```
void MatrixModel::rot90(matrix & matrix, int angle =
     90, int L = 8)
{ // L is number of LEDS in each row and column
  for (int i=0; i < ((int)angle/90+1)%4; i++)
    for (int i = 0; i < L / 2; i++)
      for (int j = i; j < L-i-1; j++) {
        int temp = matrix[i][j];
        matrix[i][j] = matrix[L-j-1][i];
        matrix[L-j-1][i] = matrix[L-1-i][L-1-j];
        matrix[L-1-i][L-1-j] = matrix[j][L-1-i];
        matrix[j][L-1-i] = temp;
      }
}
```

Listing 3. `L×L` matrix model rotation in 90 degree steps according to `angle`

To make the color shades look like on TV, it may be useful to generate rectangles of random size up to length 8×8 pixels in random colors for some random time period. Maybe also still scenes may be mixed with fast changing scenes, both scenarios lasting for long times to simulate action and documentary movies.

To make the task even more interesting, the sample code in Listing 4 uses threads that concurrently write into the display model to simulate independent actions on the TV "screen".

This exercise leaves a considerable amount of freedom to the students. Moreover, a competition to nominate the most realistic or creative solution would be appreciated by the participants for sure.

J. Exercise: Templates, STL, shared pointers

It is convenient to learn writing template classes by implementing the `stack` data structure, although a ready-to-use implementation of this template is already part of the standard C++ library.

The class `LEDStack` to define would inherit from both `Stack` (or `stack`) and `SenseHatDisplay`:

```
template<class T>
  class LEDStack : public Stack<T>, public
       SenseHatDisplay {...
```

The next step would be to visually simulate a stack data structure that contains `shared_pointers` to different SenseHAT color objects instead of copies of color objects, add operations on them such as `push green`, `push orange` or `pop`. Visual simulation here means that each stack element corresponds to an LED with the color specified in the stack element, `push` stacks one on top of another, up to eight, and `pop` means switch off the topmost.

In this exercise, the use of `try...catch` and `throw` could also be practiced to cover the various border cases.

Finally, the stack and its methods could be used to implement another OOP design pattern [19], namely the Command pattern. Its goal is to encapsulate complex operations, in this case stack operations plus color parameters. The program to be implemented would read in text commands such as `push red`, `pop` or `top` from console and execute the commands in the visual stack simulation.

```
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>
#include <random>
#include <functional>
#include <SenseHatDisplay.h>
using namespace std;

void runner(const size_t max_cycles, const short L,
    SenseHatDisplay & shd) {
  random_device rd;
  mt19937 generator(rd());  // new seed
  //random numbers for pos, size, color, duration
  uniform_int_distribution<> offset_x_dist(0, L);
  uniform_int_distribution<> offset_y_dist(0, L);
  uniform_int_distribution<> duration_dist(0, 1000);
  uniform_int_distribution<> rgb565color_dist(0,
      65535);

  for (size_t i=0; i< max_cycles; i++) {
    auto offset_x = offset_x_dist(generator);
    auto offset_y = offset_y_dist(generator);
    // cut at the borders
    uniform_int_distribution<> size_x_dist(0, L-
        offset_x);
    uniform_int_distribution<> size_y_dist(0, L-
        offset_y);
    auto size_x = size_x_dist(generator);
    auto size_y = size_y_dist(generator);
    auto duration = duration_dist(generator);
    auto rgb565color = rgb565color_dist(generator);
    // draw random rectangle
    for (short x=0; x<size_x; x++)
      for (short y=0; y<size_y; y++)
        shd.setPixel(offset_x+x, offset_y+y,
            rgb565color);
    this_thread::sleep_for(chrono::milliseconds(
        duration)); // wait
  }
}

int main() {
  SenseHatDisplay shd;
  const size_t max_cycles = 500;
  const short L = 8; // number of LEDs per col, row
  // Start threads
  thread t1 (runner, max_cycles, L, ref(shd));
  thread t2 (runner, max_cycles, L, ref(shd));
  thread t3 (runner, max_cycles, L, ref(shd));
  thread t4 (runner, max_cycles, L, ref(shd));
  // ... and join when finished
  t1.join(); t2.join(); t3.join(); t4.join();
  cout << "All threads finished." << endl;
}
```

Listing 4. Short version of a TV simulator with threads; class `SenseHatDisplay` is defined in Figure 2, Listing 1 and Listing 2

K. Exercise: Parallel execution, atomic variables, sensor data

Reading sensor data for (mainly) IMU sensor products by various vendors is made easy through the RTIMULib already mentioned in Section VI-B

Here, the documentation is slightly better than for the framebuffer code. For SenseHAT, it is sufficient to look into a sample program that comes with the library in [16], located in https://github.com/RPi-Distro/RTIMULib/blob/master/Linux/RTIMULibDrive11/RTIMULibDrive11.cpp. RTIMULIB also defines a constant `RTMATH_RAD_TO_DEGREE` to convert angle radians to degrees.

This program contains an infinite loop that reads all SenseHAT sensors. This code could be transferred into a `thread` that runs in parallel to another thread responsible for displaying things or running a game, using the physical data provided by the other thread.

An ugly solution would be to transport sensor data via global variables. C++'s `atomic` is a much more elegant device. It can be used to share variables between concurrent code. As the sensor provides read-only data, this remains also simple code-wise.

Here are some example exercises. The first and third are larger programming tasks. They try to combine many of the C++ features learned before. Two similar, definitely not too hard tasks have been assigned as final programming projects in the two courses taught so far.

1. scrolling (Marquee) text display of Section VI-G that changes direction appropriately in 90 degrees steps when the device is turned. In situ matrix rotation can be applied as described in Section VI-H. Detecting turning of the device can be implemented using the Observer design pattern.
2. change color (of the whole display, e.g.) according the values of the IMU's x, y, z axes: The x axis would influence the red components, y the blue, and z the green components.
3. a dice roller application would react only if the device (the SenseHAT) was shaken. Shaking would be interpreted as heavy accelerometer value change, cf. Figure 4. This can be modeled using a second Observer layer. Layer 1: accelerometer value change, layer 2: heavy accelerometer value change (high Euklidian vector length difference within a certain amount of time).
   When shaking is detected, a random number is computed and displayed by the SenseHAT LED matrix. The dice eyes patterns can be defined in a text file similar to the character set of the scrolling text application described in Section VI-G.
4. balancing a single ball (LED) using accelerometer.
5. sensor data recording into a file (text, binary). Generating time stamps to this aim is a good opportunity to learn about time libraries in C++, namely `ctime` and `chrono`.

L. Games

Given the low display resolution, games with the SenseHAT display have some limitations, but on the other hand side, the accelerometer makes game programming attractive. Alternatively, the five buttons can be used for control, e.g. for classic games such as Pong or Snake.

Further popular games in this spirit are Tic Tac Toe, Four in a Row and Tetris. In case of Tic Tac Toe, an object-oriented version with text output and keyboard input was included in the textbook [12, Section 5.3]. A rather small exercise was to replace the text output of Tic Tac Toe with two-color LED output based on the above described class `SenseHatDisplay`.

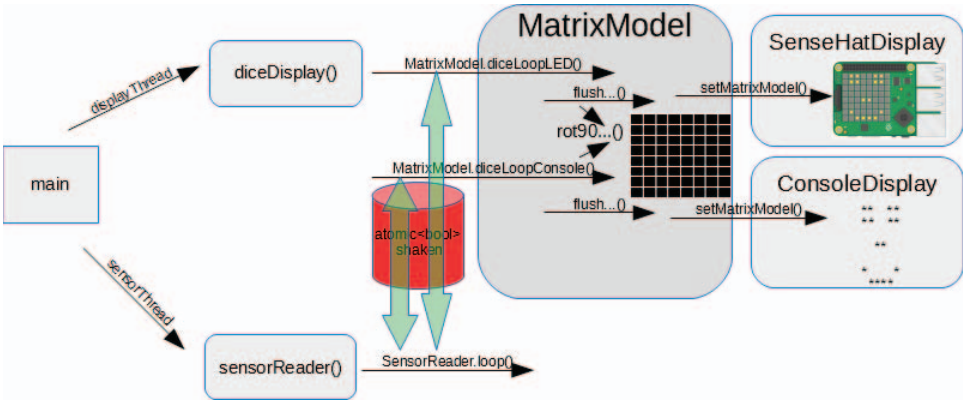

Fig. 4. Architecture of a dice roller application with accelerometer data to detect shaking of the device

VII. Evaluation

A post-course evaluation (30 written answers plus oral statements in a final retrospective meeting) showed that the time gained by the inverted classroom concept in favor of more practicing time (programming) was appreciated by the students, as well as the IoT-centered programming tasks.

The "lecture" (now meant as question and discussion time) was considered too fast and hard to understand by five students. The reason was that they did not understand that they had to read the book chapters before the practical parts and question time. However, the whole concept including a reading plan with exact dates had been communicated two months before the teaching period started so that students with tight week schedules had a chance to read the book beforehand.

In the free text fields of the written evaluation, three students complained about too much workload, and eight that the textbook would be too hard to understand while three others stated that the book was good and well understandable. Ten students explicitly stated that the practical exercises with Raspberry Pis and SenseHATs were appreciated and enjoyable.

To sum up, while the IoT hardware aspect received no criticism at all, we observed with a part of the audience what [21] described as "student resistance to novel teaching methods", especially because our course was the first and only flipped classroom course the students had seen so far. However, we think that the proposed approach is far better than the classical frontal lecture.

VIII. Peer Instruction as Supporting Activity

We had also experimented with *peer instruction* [22], sometimes referred to as "clicker questions", in an earlier C++ course. It worked very well and was extremely loved by the students. The only drawback is that it consumes a considerable amount of presence time. We can recommend it, especially to receive feedback on how many students really understood the subjects.

In the context of this paper, however, we consider it a side topic, but present an example related to modern C++ for illustration. The experiments were conducted using the Turning Point system (http://www.turningtechnologies.com).

Figure 5 shows an example of a peer instruction question on reverse iterators in C++.

What will the follow code snippet output?

```
string word;
cout << "please enter a word";
cin >> word;
cout << (word==string(word.rbegin(), word.rend()));
```

A letters of word in reversed order
B always 0 because the equality never holds
C 1 if word read forward and backward is the same, 0 otherwise
D 1 if word is identical to its copy

Fig. 5. Peer instruction question example: reverse iterators

In the first peer instruction round, students will think about the question alone, and then choose their answer using the clicker (a kind of remote control where a central voting computer counts and displays the overall result on the projector after the vote). A result is shown in Figure 6.

In the second round, after having seen the overall vote, students first discuss with their neighbor and then vote again. A result for the same question is shown in Figure 7.

Typically, and almost always, the second vote is better and shows how discussing possible answers helps to learn und correct misunderstandings among the students and without individual help by the instructor.

The advantage of the peer instruction method for the students is that they are forced to think about a problem alone, but then quickly get feedback about misconceptions in such a way that misconceptions become less manifested. Therefore, it is important that these are discussed afterwards (again) and the correct solution is clearly repeated and explained by the teacher.

IX. Conclusion and Outlook

We have presented the concept of a modern C++ course for electrical engineering and computer science students based on inverted classroom and with enjoyable IoT hardware. To ease the use of the IoT hardware, SenseHATs (hardware attached on top boards) were used so that wiring circuits could be avoided.

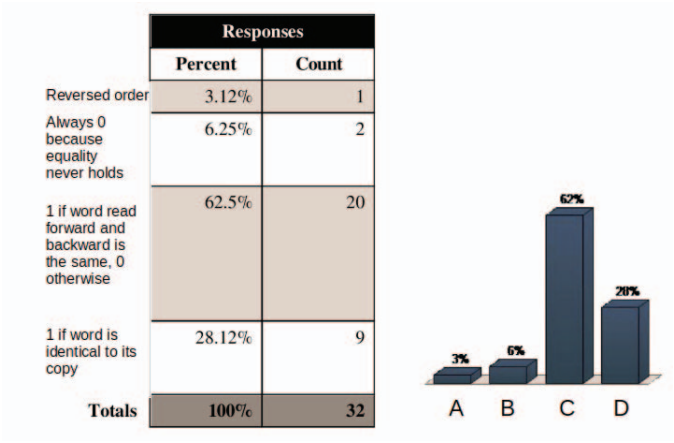

Fig. 6. Peer instruction question first round

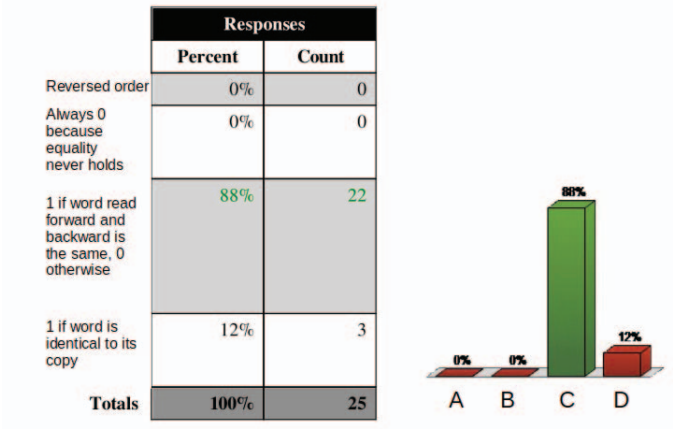

Fig. 7. Peer instruction question second round

The main goals, reducing lecture time in favor of more time for practical programming tasks as well as preparing students to autonomous, potentially lifelong learning have been achieved through flipped classroom. The new concept was appreciated by the majority of the students and also enjoyed by the instructor.

This course is under continuous development. For the next teaching period, improvements based on the feedback of the previous two courses will be:

- additional two hours of presence time per week, mainly dedicated to posing and answering questions, including again more *peer instruction* sessions
- parts of the exercises (those without IoT hardware requirements mostly towards the beginning of the course) could be run on PCs in a virtual machine (Ubuntu Linux with Code::Blocks), as some students had complained about the overhead to work on a remote Raspberry Pi with USB-LAN network configuration, remote login, slower yet acceptable CPU performance
- more control questions than those in the textbook could be added to the learning management system because they were considered important by the participants.

Finally, we would like to point out that the proposed course is meant for C++ beginners. Advanced students could for example also exploit further technologies such as the MQTT (Message Queuing Telemetry Transport) communication protocol [23] to compose sensor networks.

Acknowledgment

I would like to thank the anonymous reviewers for their helpful comments.

References

[1] "Standard for programming language C++," 2017. [Online]. Available: https://isocpp.org

[2] A. King, "From sage on the stage to guide on the side," *College teaching*, vol. 41, no. 1, pp. 30–35, 1993.

[3] J. L. Bishop and M. A. Verleger, "The flipped classroom: A survey of the research," in *ASEE National Conference Proceedings*, vol. 30, Atlanta, GA, 2013.

[4] J. Chin and V. Callaghan, "Educational living labs: A novel internet-of-things based approach to teaching and research," in *9th Int. Conf. on Intelligent Environments (IE)*. IEEE Press, New York, 2013, pp. 92–99.

[5] W. Gay, *Exploring the Raspberry Pi 2 with C++*. Apress, 2015.

[6] M. L. Maher, C. Latulipe, H. Lipford, and A. Rorrer, "Flipped classroom strategies for CS education," in *46th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '15. New York, NY, USA: ACM, 2015, pp. 218–223.

[7] R. Talbert, "Learning MATLAB in the inverted classroom," in *ASEE Annual Conference & Exposition*, San Antonio, Texas, 2012.

[8] X. Zhong and Y. Liang, "Raspberry Pi: An effective vehicle in teaching the internet of things in computer science and engineering," *Electronics*, vol. 5, no. 3, 2016.

[9] O. Hahm, E. Baccelli, H. Petersen, M. Wählisch, and T. Schmidt, "Simply RIOT: Teaching and experimental research in the internet of things," in *13th ACM/IEEE Int. Conf. on Information Processing in Sensor Networks (IPSN 2014)*. Berlin, Germany: ACM, 2014.

[10] "The Raspberry Pi Foundation, MagPi magazine issues," free PDF download. [Online]. Available: https://www.raspberrypi.org/magpi-issues/

[11] B. W. Kernighan and D. M. Ritchie, *The C Programming Language*. Upper Saddle River, NJ, USA: Prentice Hall Press, 1988.

[12] U. Breymann, *C++ – eine Einführung*. C. Hanser, Munich, 2016.

[13] B. Stroustrup, Ed., *A Tour of C++*, 2nd ed. Addison-Wesley Professional, 2018.

[14] U. Breymann, *Der C++-Programmierer*, 5th ed. C. Hanser, Munich, 2018.

[15] B. Stroustrup, *The C++ Programming Language*, 4th ed. Addison-Wesley Professional, 2013.

[16] "RTIMULib - a versatile C++ and Python 9-dof, 10-dof and 11-dof IMU library," 2015. [Online]. Available: https://github.com/RPi-Distro/RTIMULib

[17] M. Garrels, "Bash guide for beginners," 2008. [Online]. Available: http://www.tldp.org/guides.html#bbg

[18] M. Cooper, "Advanced bash-scripting guide," 2014. [Online]. Available: http://www.tldp.org/guides.html#abs

[19] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-oriented Software*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[20] I. Kadiyski, "Rotate matrix," 2013. [Online]. Available: http://algorithmproblems.blogspot.com/2013/01/rotate-matrix.html

[21] C. F. Herreid and N. A. Schiller, "Case studies and the flipped classroom," *College Science Teaching*, vol. 42, no. 5, pp. 62–67, 2013.

[22] E. Mazur, *Peer Instruction – a user's manual*. Pearson Education, 1997.

[23] A. Banks and R. Gupta, "MQTT version 3.1.1." [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html