JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 20, No. 3, pp. 647-655, Mar. 2019

이종 운영체제에서 UART 통신을 이용한 라즈베리파이 리듬 게임 구현

양희준 · 김범준 · 김동회*
강원대학교 IT대학 전기전자공학과

Implementation of Raspberry Pi Rhythm Game using UART Communication in the different Operating Systems

Hee-Jun Yang · Beom-Jun Kim · Dong-Hoi Kim*
Electrical and Electronic Engineering, IT College, Kangwon National University, Chuncheon, Korea

[요 약]

본 논문에서는 리눅스(Linux) 운영체제(Operating system)를 사용하는 라즈베리파이 3(Raspberry pi 3)와 윈도우(Windows) 운영체제를 사용하는 PC(Personal computer)의 이종 운영체제상태에서 UART(Universial asynchronous receiver/transmitter) 통신을 이용하여 리듬 게임을 구현하였다. 구현된 리듬 게임은 라즈베리파이에서 작성한 소스를 이용해 입력 값을 받아서 윈도우 운영체제인 PC에서 동작한다. 구현한 리듬 게임에 대한 실험들을 통하여 UART 통신을 위해 필요한 보율(Baud rates)값 중에서 가장 빠른 반응속도를 가지는 보율 값을 찾았고, PC 성능에 따라 달라지는 반응 속도를 비교 분석하였다. 결과적으로 다른 운영체제를 가진 라즈베리파이와 PC에서 UART 통신을 이용하여 빠른 반응속도를 가지는 리듬 게임을 구현하였고 원하는 정상적인 동작을 확인하였다.

[Abstract]

In this paper, we implement the rhythm game using UART communication in the different Operating systems of Raspberry pi 3 using Linux Operating system and PC using Windows Operating system. The implemented rhythm game is operated in PC of Windows Operating system which receives the source made in the Raspberry pi 3 as input value. Through the experiments of the implemented rhythm game, we can find the baud rate with the fastest response time among baud rate values which are necessary for UART communication and can explore and analyze the response times which are varied by PC performance. As a result, we implemented a rhythm game with the fast response time by using UART communication in Raspberry pi 3 and PC with different Operating systems and confirmed the desired normal operations.

색인어 : 라즈베리파이, 파이썬, 리듬 게임, UART 통신, 보율

Key word : Raspberry Pi, Python, Rhythm game, UART communication, Baud rates

http://dx.doi.org/10.9728/dcs.2019.20.3.647

Received 16 February 2019; **Revised** 24 March 2019
Accepted 26 March 2019

***Corresponding Author; Dong-Hoi Kim**

Tel: +82-33-250-6349
E-mail: donghk@kangwon.ac.kr

Ⅰ. 서 론

과거부터 오늘날까지 게임 산업은 점차 발전하고 있다. 여러 종목별로 다양한 게임들이 개발되고 있고 그에 따라 여러 해가 지날수록 게임 역시 진화해 가고 있다. 과거에는 2D 중심의 게임과 PC 게임이 주를 이루는 시대였다면 스마트폰이 개발되고 난 뒤 모바일 게임과 콘솔게임 등 3D 중심게임들이 주를 이루는 시대가 되었고 오늘날에는 PC 게임과 모바일 게임들이 서로 공존하는 게임 산업이 되어 가고 있다. 이러한 게임 산업 중심에서 다양한 게임을 개발할 때 이용되는 것 중 하나가 바로 라즈베리파이이다. 라즈베리파이는 게임 분야 뿐만 아니라 IoT 기술 분야 및 여러 IT 분야 등 현대에 많은 분야에 활용되고 있다.

본 논문은 이처럼 여러 분야에 활용되는 라즈베리파이를 이용하여 실제 리듬 게임을 구현하는 것을 제안하였다. 현재 여러 웹사이트에서 공유되고 있는 리듬 게임들은 윈도우(Windows) 운영체제에서 개발되어 윈도우 운영체제에서 실행되는 프로그램이 대다수이고, 본 논문에서 구현한 리듬 게임 프로그램도 윈도우 운영체제에서 개발되어 EXE 확장자명을 가진 응용프로그램이기 때문에 리눅스(Linux) 운영체제를 사용하는 라즈베리파이 단독으로 구성하게 되면 리눅스 운영체제를 사용하는 라즈베리파이에서 윈도우 운영체제에서 맞춰져 컴파일되어 있는 프로그램을 실행할 수 없기 때문에 리듬 게임을 구동시킬 수 없다. 이를 해결하기 위해서 리눅스 운영체제를 사용하는 라즈베리파이와 윈도우 운영체제를 사용하는 PC를 서로 UART(Universial asynchronous receiver/transmitter) 통신[1]을 통해 라즈베리파이에서 조작을 하고 조작한 값을 보내고 윈도우 운영체제를 사용하는 PC에서 받아 리듬 게임을 구동시켜 라즈베리파이 리듬 게임을 구현하였다. 또한 라즈베리파이를 이용하여 게임을 실행하기 위한 여러 소스들을 작성할 때 C 언어 기반은 아무래도 코드가 너무 길고 복잡하며 이로 인해 다른 사람들이 한 눈에 보기에 어려움이 있으며 개발자 입장에서도 좀 더 간결하고 한 눈에 보기에 편리한 파이썬을 이용하여 작성하기로 결정하였다.

본 논문은 기존의 리듬 게임을 리눅스 운영체제인 라즈베리파이와 윈도우 운영체제인 PC[2]를 UART통신을 이용하여 리듬 게임의 해당키 입력을 리눅스 운영체제인 라즈베리파이[3]에서 작성한 소스를 이용해 입력 값을 보내면 윈도우 운영체제인 PC에서 값을 받아 PC에서 실행중인 리듬 게임의 키를 조작하게 하여 리듬 게임을 구동하였다. 몇 가지 테스트를 통하여 가장 빠른 반응속도를 가지는 보율(Baud rates)값과 PC 성능을 찾아 적용함으로써 라즈베리파이와 PC를 UART 통신을 이용하여 빠른 반응속도를 가지는 리듬 게임을 구동할 수 있었다. 본 논문은 서론, Ⅱ장에서는 라즈베리 파이를 이용한 기존게임들, Ⅲ장에서는 UART통신을 이용하여 구현시킨 리듬 게임, Ⅳ장에서는 리듬 게임 제작 및 실험결과, 마지막 Ⅴ장에서는 본 논문의 결론을 맺는다.

Ⅱ. 라즈베리파이를 이용한 기존 게임들

과거부터 오늘날까지 게임 산업은 꾸준히 발전해오고 있다. PC게임, 모바일 게임, 여러 콘솔게임 등 이와 같은 다양한 게임의 개발에 사용되는 것 중 하나가 라즈베리파이이다. 라즈베리파이는 게임분야 뿐 아니라 최근에 큰 관심을 받고 있는 IoT 서비스 분야에도 사용되고 있는 것처럼 다양한 분야에 사용되고 있다. 우리는 이러한 여러 분야에 사용되고 있는 라즈베리파이를 이용하여 다양한 음악들을 가지고 플레이하는 리듬 게임을 제작해보기로 결정하였다. 라즈베리파이를 이용하여 기상청에서 무료로 제공하는 RSS(Rich Site Summary) 서비스로부터 얻어진 날씨 정보를 이용하는 새로운 탐사 로봇 속도 제어 방법도 있다[3]. 이처럼 라즈베리파이는 여러 분야에서 사용될 수 있다.

기존의 라즈베리파이를 이용한 기존 게임 운영체제는 현재 크게 3가지가 존재하고 있다. LibRETRO(RetroARCH)를 기반으로 구동되는 레트로파이(RetroPie), 리콜박스(Recalbox), Lakka가 그 예시이다. 라즈베리파이에 이 3가지 중 하나의 운영체제를 설치하여 게임 롬을 넣어주고 사용하면 된다. 하지만 이 3가지 모두 리눅스로 만들어진 운영체제이다. 우리가 하려고 하는 리듬 게임(DDR)은 윈도우 환경에서 동작하기 때문에 우리가 선택한 방법은 UART 통신을 이용하여 리눅스 운영체제를 사용하는 라즈베리파이와 윈도우 운영체제를 사용하는 PC를 연결하여 라즈베리파이에서 작성한 소스를 통해 입력 값을 보내면 PC에서 UART 통신으로 값을 받고 PC에서 작성한 소스를 통해 실행중인 리듬 게임의 조작키를 제어하고자 한다[4].

2-1 Linux 운영체제 와 Windows 운영체제

리눅스는 자유 소프트웨어와 오픈 소스 개발의 가장 유명한 표본으로 들 수 있다. 리눅스는 다중 사용자, 다중 작업(멀티태스킹), 다중 스레드를 지원하는 네트워크 운영 체제(NOS)이다[1]. 엄밀하게 따지면 이 '리눅스'라는 용어는 리눅스 커널만을 뜻하지만, 리눅스 커널과 GNU 프로젝트의 라이브러리와 도구들이 포함된, 전체 운영 체제(GNU/리눅스라고도 알려진)를 나타내는 말로 흔히 쓰인다. 윈도우는 마이크로소프트에서 개발하는 컴퓨터 운영체제이다. 데스크톱에 쓰는 운영체제 중에서는 사실상 표준 수준으로 가장 많은 점유율을 가지고 있다.

리눅스의 장점으로는 네트워크를 이용하는 컴퓨팅에서 리눅스가 강점을 가지며 오픈 소스의 특징으로 인해 누구나 버그 수정이 가능하고, 커널을 비롯한 각종 프로그램을 사용하기에 편리하게 묶어 놓은 배포판이라는 장점이 있다. 단점으로는 커널과 여러 소프트웨어가 배포될 때 정리가 잘 되지 않았고, 실시간 처리가 약하며, 시스템 보안에 취약하다는 단점이 있다.

Ⅲ. UART 통신을 이용하여 PC에서 구현시킨 리듬 게임

3-1 UART란?

무선 센서 네트워크는 일반적으로 엔드 장치에 연결된 센서가 측정한 데이터를 코디네이터에게로 전달하는 자원이 한정된 장치들로 구성된다. 지그비는 무선 센서 네트워크에서 가장 많이 사용되는 프로토콜 중의 하나로서, 적정한 데이터 전송률을 가지면서 저전력, 저비용의 무선 통신을 지원하기 위해 표준으로 도입되었다[5]. UART란 범용 비동기화 송수신기라고 부르며 병렬 데이터의 형태를 직렬 방식으로 전환하여 데이터를 전송하는 컴퓨터 하드웨어의 일종이다[2]. UART의 U는 범용을 가리키는데 이는 자료 형태나 전송 속도를 직접 구성할 수 있고 실제 전기 신호 수준과 방식이 일반적으로 UART 바깥의 특정한 드라이버 회로를 통해 관리를 받는다는 뜻을 의미한다. 또한, 통신 데이터는 메모리 또는 레지스터에 들어 있어 이것을 차례대로 읽어서 직렬 형태로 바꾸어서 통신한다. 단위로는 8비트가 기본 단위이다.

UART는 컴퓨터나 주변 기기의 일종으로 병렬 데이터를 직렬형태로 바꾸어서 통신하는 개별 접적회로이다. 또한 비동기 통신이기 때문에 동기 신호가 전달되지 않는다. 따라서 수신 쪽에서 자체적으로 동기신호를 찾아내어 데이터의 시작과 끝을 시간적으로 알아 처리할 수 있도록 설정되어 있다. 디지털 회로는 자체의 클럭 신호를 이용하여 정해진 속도로 수신 데이터로부터 비트구간을 구분하고 그 비트의 논리 상태를 결정하여 데이터를 통신한다.

3-2 UART 데이터 송 · 수신 형태

Bit count	1	2	3	4	5	6	7	8	9	10	11
	Start bit	Data bit								Parity bit	End bit
	start	data 0	data 1	data 2	data 3	data 4	data 5	data 6	data 7	parity	stop

그림 1. UART 데이터의 송·수신 형태
Fig. 1. Transmission/reception mode of UART data

데이터 송, 수신은 먼저 각 데이터 비트의 시간에 대해 클럭 신호를 이용하여 시작 비트로부터 각 비트의 경계를 찾아낸다. 클럭 신호는 보드 설정 및 프로그래밍에 의한 레지스터 설정에 따라 변경될 수 있다. 그렇기 때문에 경계를 찾아낸 뒤 통신 양쪽에서 미리 설정하고 클럭 신호 발생부의 레지스터를 같은 속도로 설정해야 통신이 원활하게 이루어지게 된다. 다음은 각 비트에 관한 간단한 설명이다.

시작 비트는 통신을 시작하는 것을 뜻하며 하나의 비트 시간 길이만큼 유지한다. 시작 비트를 기준으로 정해진 규칙대로 통신을 시작한다. 데이터 비트는 5비트~8비트의 데이터 전송을 뜻한다. 몇 비트를 사용할지는 레지스터의 설정에 따라 결정된다. 패리티 비트는 오류 검증을 하기 위한 패리티 값을 생성하여 송신하고 수신 쪽에서 오류를 검정한다. 레지스터 설정에 따라 사용하지 않음, 홀수, 짝수 패리티 세 가지 옵션을 선택할 수 있다. 종료 비트는 통신 종료를 알리며, 레지스터 설정에 따라 1, 1.5, 2비트 세 가지의 정해진 비트만큼 유지해야 한다[6].

3-3 UART를 사용할 때 하드웨어와 연결방법

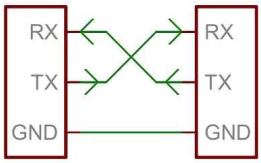

그림 2. UART를 사용할 때 하드웨어의 연결방법
Fig. 2. Connection method of hardware when UART is used

UART 통신을 이용할 때 하드웨어 연결방법은 그림에서 보는것과 같이 데이터 수신(RX)과 데이터 송신(TX)은 서로 교차로 연결하여야 한다. 또한 UART 통신은 보율을 설정하기 위해 1초에 몇 개의 신호가 전송되는지를 나타내는 단위로 비동기식이기 때문에 두 프로세서 간의 속도를 맞춰주어야 한다.

3-4 UART통신을 이용한 조작키 제어(전체적인 알고리즘)

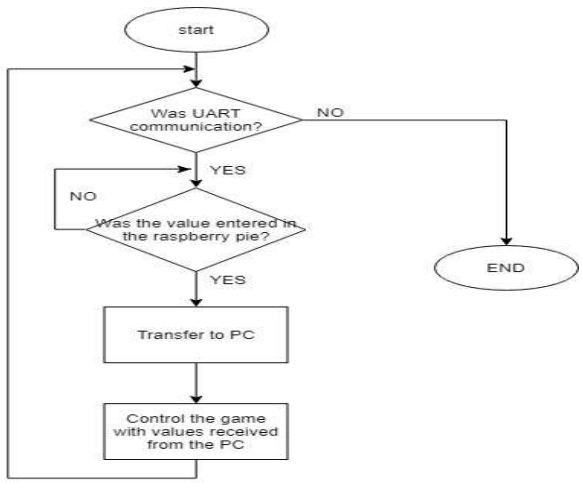

그림 3. UART 통신을 이용한 조작키 제어 (전체적인 흐름도)
Fig. 3. Operation key control using UART communication (overall flowchart)

그림 3은 UART 통신을 이용한 조작키 제어에서 전체적인 동작 흐름을 보여주고 있다. 라즈베리파이와 PC가 UART 통신이 이루어졌는지 확인하고 정상적으로 통신이 이루어졌다면, 라즈베리파이에 입력 값이 들어올 때 까지 기다리다가 값이 들어오면 PC로 그 값을 UART 통신을 이용해 전송해 준다. PC에서 UART 통신을 통해 전송된 값을 받으면 그 값과 맵핑되어 있는 키보드 값을 입력하여 실행되고 있는 리듬 게임을 제어할 수 있게 된다[7].

3-5 UART 통신을 이용한 조작키 제어(Raspberry Pi 3 알고리즘)

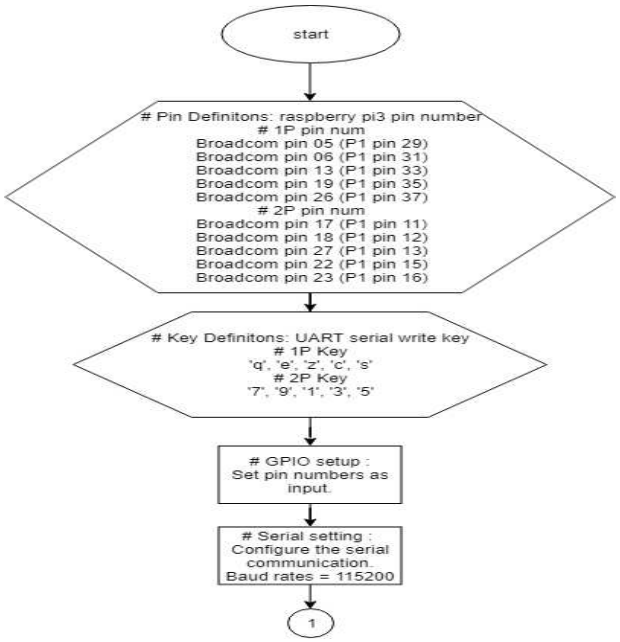

그림 4. UART 통신을 이용한 조작키 제어(라즈베리파이 부분 알고리즘 - 선언부)
Fig. 4. Key operation control using UART communication (Raspberry pie algorithm - Declaration section)

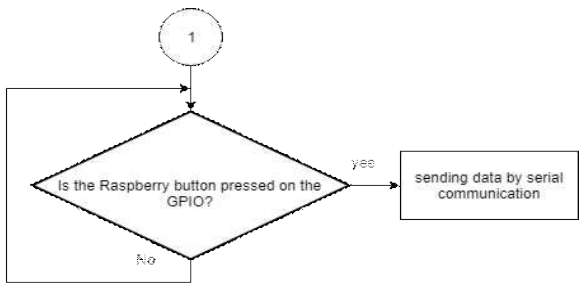

그림 5. UART 통신을 이용한 조작키 제어(라즈베리파이 부분 알고리즘 - 실행부)
Fig. 5. Key operation for UART communication (Raspberry pie algorithm - Execution section)

그림 4는 UART 통신을 이용한 조작키 제어에서 라즈베리파이 3의 소스부분(선언부)을 나타내고 있다. 선언부에서는 사용할 GPIO핀들을 선언해 주고, 그 핀에 입력이 들어왔을 때 시리얼 통신으로 보낼 데이터를 'q', 'e', 'z', ...등으로 선언해 주었다[7]. 그리고 선언한 GPIO핀 번호들을 입력으로 설정해 주고 시리얼 통신의 설정을 해준다. 시리얼통신의 보율 값은 115200으로 해주었다. 그림 5는 UART 통신을 이용한 조작키 제어에서 라즈베리파이 3의 소스부분(실행문)을 나타내고 있다. 실행문에서는 무한루프에 들어가 있어 입력으로 설정한 GPIO핀에 값이 들어오게 된다면 시리얼 통신을 통해 그 핀에 해당하는 데이터를 보내는 역할을 하고 있다. 즉 라즈베리파이는 GPIO에 연결된 버튼이 눌리는지 아닌지를 조건문으로 설정하고 GPIO에 연결된 버튼이 눌린다면 시리얼 통신으로 해당 데이터를 보내고(yes), 아니면 다시 무한루프로 인하여 처음으로 돌아가서 조건문으로 다시 들어가는 것을 반복하게 된다[8].

3-6 UART 통신을 이용한 조작키 제어(PC 알고리즘)

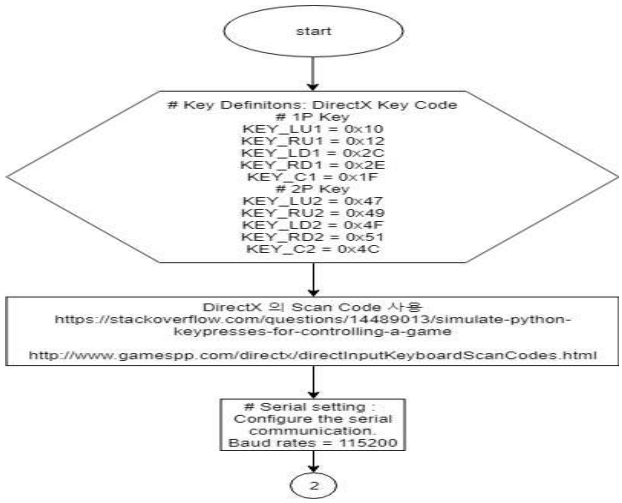

그림 6. UART통신을 이용한 조작키 제어(PC 부분 알고리즘 - 선언부)
Fig. 6. Key operation control using UART communication (PC algorithm - Declaration section)

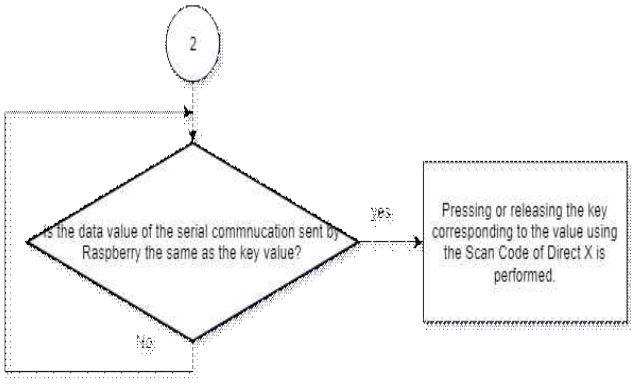

그림 7. UART 통신을 이용한 조작키 제어(PC 부분 알고리즘 - 실행부)
Fig. 7. Key operation control using UART communication (PC algorithm - Execution section)

그림 6은 UART 통신을 이용한 조작키 제어에서 PC의 소스 부분(선언부)을 나타내고 있다. 선언부에서는 게임에서 키를 제어하기 위해 Direct X의 Scan Code를 사용하였다. Direct X의 Scan Code의 정보에서 각 키에 해당하는 값들을 선언해 주고, Direct X의 Stack overflow 코드를 참조하여 소스를 작성하였다[9]. 라즈베리파이 3과 마찬가지로 시리얼 통신의 보율 값은 115200으로 설정해 주었다.

그림 7은 UART 통신을 이용한 조작키 제어에서 PC의 소스 부분(실행문)을 나타내고 있다. 실행문에서는 무한루프에 들어가 라즈베리파이 3에서 보낸 시리얼 통신의 데이터를 판별하여 그 값에 해당하는 키를 Direct X의 Scan Code를 사용하여 키를 눌렀다가 떼는 행동을 한다. 즉 라즈베리 파이에서 보낸 시리얼 통신의 데이터 값과 PC의 해당 키 값과 같다면 그 키를 눌렀다가 떼는 동작을 수행하는 것을 의미한다.

Ⅳ. UART 통신을 이용한 라즈베리파이를 사용하여 리듬 게임 제작 및 실험결과

4-1 두 종류의 언어 체계를 사용하였을 때 특징 비교

표 1. 두 언어 체계 특징 비교
Table 1. Characteristic Comparison between two language systems

	C	Python
productivity	low productivity	high productivity
complication	high complication	low complication

```
1   void Interface();       // 메인 화면
2   void Map();             // 게임 맵
3   void PrepareGame();// 게임 준비
4   void gotoxy(int x, int y);          // 좌표 이동
5   int RandNum();          // 랜덤함수
6   // 1P
7   int upL(int n);         // 방향키(왼쪽상단버튼)
8   int upR(int n);         // 방향키(오른쪽상단버튼)
9   int C(int n);           // 방향키(가운데버튼)
10  int downL(int n);       // 방향키(왼쪽하단버튼)
11  int downR(int n);       // 방향키(왼쪽하단버튼)
12  // 2P
13  int upL2(int n);        // 방향키(왼쪽상단버튼)
14  int upR2(int n);        // 방향키(오른쪽상단버튼)
15  int C2(int n);          // 방향키(가운데버튼)
16  int downL2(int n);      // 방향키(왼쪽하단버튼)
17  int downR2(int n);      // 방향키(왼쪽하단버튼)
```

그림 8. C 언어로 게임 버튼을 위한 소스 코드 예
Fig. 8. Example of source code for game buttons in C language

C 언어로 프로그램 소스 코드를 작성할 때에는 여러 가지 함수 및 코드들을 호출하여야 하고 그때 마다 형식에 맞춰서 소스 코드를 작성해야 하기 때문에 이식성이 높을지라도 보는 사람 입장에서 가독하기가 매우 힘들었다. 예를 들어 게임에 필요한 버튼 키 설정을 위한 코드를 작성할 때 파이썬(Python)의 경우에는 그림 10과 같이 변수와 그 값을 설정만 하면 되지만 C 언어의 경우에는 조건문을 쓰고 각각의 변수의 자료형도 설정해 주어야 하고 점점 소스 코드의 길이도 길어지고 그러다 보니 다른 사람이 한눈에 보기 어렵고 중간에 코드를 잘못 입력하여 오류가 나는 등 이식성도 그 만큼 떨어지게 된다. 그러나 파이썬의 경우에는 C 언어로 구현된 라이브러리를 그대로 간단하게 사용할 수 있기 때문에 C 라이브러리의 함수 호출이 굉장히 쉽다. 함수 호출이 쉽고 C에 비해 여러 가지 불필요한 코드들을 사용하지 않아도 되기 때문에 코드가 상대적으로 짧고 이에 가독성이 더욱 높은 특징을 확인할 수 있었다. 그림 8은 C언어로 게임에 필요한 버튼 및 메뉴를 설정하는데 변수 및 함수를 선언하는 그림이며, 파이썬처럼 간단하게 키를 설정하는 코드를 작성하는 것 대신 C 언어로는 게임에 사용할 버튼 키 설정을 하는데 있어서 여러 함수들을 설정하는 모습이다. 그림 8과 10을 비교해 보면 C 언어로 작성한 것이 훨씬 복잡하고 한눈에 보기에도 어렵다는 것을 확인할 수 있어서 그만큼 가독성과 이식성이 낮다는 것을 표 1을 통해 정리할 수 있다[10].

4-2 리듬 게임 제작(하드웨어&소프트웨어)

그림 9. 리듬 게임 제작(하드웨어)
Fig. 9. Rhythm Game Production (Hardware)

그림 9는 리듬 게임의 하드웨어를 제작한 모습을 보여주고 있다. 하드웨어는 게임을 실행하기 위한 핵심요소인 실행 버튼을 의미하는데 버튼 안은 각각 LED 소켓으로 이루어져 있다. 또한 GPIO표를 이용하여 밑의 리듬 게임 제작을 하기 위해 사용자(Player) 1, 2를 위한 각각의 버튼 설정 소스를 참고하여 소켓에 전선을 연결하였다.

사용자1은 GPIO의 핀번호 29, 31, 33, 35, 37에 해당되는 GPIO5, GPIO6, GPIO13, GPIO19, GPIO26을사용하였으며 사용자2는 GPIO의 핀번호 11, 12, 13, 15, 16에 해당되는 GPIO17, GPIO18, GPIO27, GPIO22, GPIO23을 사용하였다. 사용자1의 LU1은 왼쪽 상단 버튼, RU1은 오른쪽 상단 버튼, LD1은 왼쪽

하단 버튼, RD1은 오른쪽 하단 버튼, C1은 가운데 버튼으로 구성되어 있으며, 사용자2의 LU2은 왼쪽 상단 버튼, RU2은 오른쪽 상단 버튼, LD2은 왼쪽 하단 버튼, RD2은 오른쪽 하단 버튼, C2은 가운데 버튼으로 구성되어 있다. 또한 앞서 정리한 그림 8의 C를 이용하여 버튼을 위한 소스를 작성하는 것보다 훨씬 간편하고 한눈에 보기에도 편리한 점을 확인할 수 있다.

```
1   # Pin Definitons:
2   # 1P pin num
3   LU1 = 5    # Broadcom pin 05 (P1 pin 29)
4   RU1 = 6    # Broadcom pin 06 (P1 pin 31)
5   LD1 = 13   # Broadcom pin 13 (P1 pin 33)
6   RD1 = 19   # Broadcom pin 19 (P1 pin 35)
7   C1 = 26    # Broadcom pin 26 (P1 pin 37)
8   # 2P pin num
9   LU2 = 17   # Broadcom pin 17 (P2 pin 11)
10  RU2 = 18   # Broadcom pin 18 (P2 pin 12)
11  LD2 = 27   # Broadcom pin 27 (P2 pin 13)
12  RD2 = 22   # Broadcom pin 22 (P2 pin 15)
13  C2 = 23    # Broadcom pin 23 (P2 pin 16)
```

그림 10. 리듬 게임 제작(Python을 이용한 게임 버튼, 소프트웨어)
Fig. 10. Rhythm game creation (game button with Python, software)

표 2. 라즈베리파이 GPIO Header
Table 2. Raspberry Pie GPIO Header

Python (BCM)	WiringPi GPIO	Name	P1 Pin Number		Name	WiringPi GPIO	Python (BCM)
		3.3v DC Power	1	2	5v DC Power		
	8	GPIO02 (SDA1, I2C)	3	4	5v DC Power		
	9	GPIO03 (SCL1, I2C)	5	6	Ground		
4	7	GPIO04 (GPIO_GCLK)	7	8	GPIO14 (TXD0)	15	
		Ground	9	10	GPIO15 (RXD0)	16	
17	0	GPIO17 (GPIO_GEN0)	11	12	GPIO18 (GPIO_GEN1)	1	18
27	2	GPIO27 (GPIO_GEN2)	13	14	Ground		
22	3	GPIO22 (GPIO_GEN3)	15	16	GPIO23 (GPIO_GEN4)	4	23
		3.3v DC Power	17	18	GPIO24 (GPIO_GEN5)	5	24
	12	GPIO10 (SPI_MOSI)	19	20	Ground		
	13	GPIO09 (SPI_MISO)	21	22	GPIO25 (GPIO_GEN6)	6	25
	(no worky 14)	GPIO11 (SPI_CLK)	23	24	GPIO08 (SPI_CE0_N)	10	
		Ground	25	26	GPIO07 (SPI_CE1_N)	11	
	30	ID_SD (I2C ID EEPROM)	27	28	ID_SC (I2C ID EEPROM)	31	
5	21	GPIO05	29	30	Ground		
6	22	GPIO06	31	32	GPIO12	26	12
13	23	GPIO13	33	34	Ground		
19	24	GPIO19	35	36	GPIO16	27	16
26	25	GPIO26	37	38	GPIO20	28	20
		Ground	39	40	GPIO21	29	21

표 2는 라즈베리파이의 GPIO Header를 나타낸 표이다. 이 표를 참고로 라즈베리파이에서 입력, 출력 핀들의 위치와 소스코드를 맵핑시킬수 있게 된다. 표에서 Python 열은 파이썬언어를 사용했을 때의 GPIO번호이고, WringPi 열은 C언어를 사용했을 때의 GPIO번호이다.

4-2 라즈베리파이 설정에 따른 통신 속도 비교

표 3. 보율 값에 따른 대략적인 응답 속도
Table 3. Approximate response rate according to the baud rate values

Category	serial Baud rates value	Response time (Approximate time)
Case 1	115200	0.2 sec
Case 2	57600	0.4 sec
Case 3	38400	0.5 sec
Case 4	19200	1.1 sec
Case 5	4800	1.5 sec
Case 6	2400	2 sec
Case 7	1200	2.5 sec

표 3는 직렬 통신의 보율 값에 따른 직렬 데이터를 전송하고 받는 데 걸리는 대략적인 시간을 나타낸 표이다. 위 표에서 serial Baud rates value열에 있는 값들은 표준 보율 값이다. Response time열은 시리얼 통신의 표준 보율 값에 응답하는 속도 인데 단위는 sec이다. 즉, 보율 값이 높아지면 높아질수록 응답속도는 빨라지는것을 이 표를 통해 알 수 있다. 여기서 응답속도는 게임을 실행하였을 때 버튼을 눌렀을 때의 속도와 밀접하게 연관되어 있다.

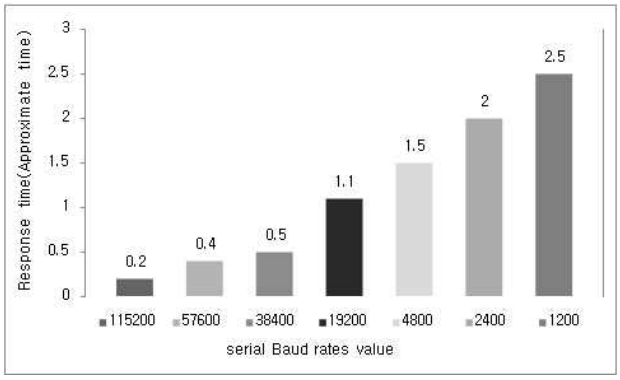

그림 11. 보율 값에 따른 대략적인 응답 속도 그래프
Fig. 11. Graph of approximate response rate based on the baud rate values

그림 11은 본 논문에서 구현한 UART 통신을 이용한 라즈베리파이 리듬 게임에 대한 실험을 통해 응답속도의 측정을 알 수

가 있다. 먼저 가장 높은 숫자인 115200일 때 응답 속도가 가장 빠르게 측정되었다. 반대로 점점 낮아질수록 응답 속도가 느려지게 측정되었고, 보율 값이 1200이 되었을 때는 115200일 때보다 10배이상 느려진 속도를 나타내게 된다. UART통신은 응답 속도가 빠른 만큼 버튼을 누르고 있는 같은 시간동안 더 많은 양의 데이터를 시리얼 통신을 통해 보내기 때문에 많은 양의 데이터를 더 빠르게 처리할 수 있다. 하지만 받는 쪽에서는 성능이 낮으면 직렬 버퍼(serial buffer)를 비우는데 더 많은 시간이 소요되게 된다. 받는 쪽의 성능이 좋지 않는다면 버퍼에 계속해서 데이터가 쌓이다가 결국 데이터가 손실되는 현상이 나타날 수 있다. 따라서, 성능을 고려해 이 시간들을 잘 조율하여 적당한 보율 값을 찾으면 효과적으로 데이터를 처리할 수 있지만 리듬 게임과 같은 반응속도가 중요한 작업에서는 가장 빠른 보율 값인 115200을 사용하여 본 논문의 리듬 게임이 잘 구동이 됨을 확인할 수 있다.

4-3 PC 성능에 따른 통신 속도 비교

표 4. PC 성능에 따른 대략적인 응답 속도
Table 4. Approximate response speed according to PC performance

Category	PC performance (Baud rates value = 115200)	Response time (Approximate time)	cpu (GHZ)
Case 1	laptop	0.2 sec	1.7
Case 2	Desktop(Low performance)	0.1 sec	3.5
Case 3	Desktop(High performance)	0.1 sec	4.1

표 5. 비교한 PC들의 각 스펙 및 정보
Table 5. Each specification and information of the compared PCs

Option	Specifications and Information
laptop	Processor : Intel(R) Core(TM) i3-4005U CPU @ 1.70Hz 1.70GHZ Installed memory(RAM) : 4.00GB
Desktop (Low performance)	Processor : Intel(R) Core(TM) i5-8500 CPU @ 3.5Hz(12 CPUs), ~ 3 .7GHZ Installed memory(RAM) : 14GB
Desktop (High performance)	Processor : Intel(R) Core(TM) i5-8500 CPU @ 4.10GHz 4.10GHZ Installed memory(RAM) : 16GB

표 4은 시리얼 통신의 보율 값이 같을 때 PC 성능에 따른 시리얼 데이터를 전송하고 받는 데 걸리는 대략적인 시간을 나타낸 표이다. PC performance열은 보율 값이 모두 동일한 값인 115200이라는 전제하에 노트북(laptop), 저 사양 PC(Low performance desktop), 고 사양 PC(High performance desktop)를 사용하였다. PC의 성능은 표 5를 보면 알 수 있듯이 각각 다른 CPU와 다른 크기의 RAM을 사용하였다. Response time열은 응답시간을 나타내며 단위는 sec이다. cpu열은 CPU의 속도를 나타내고 있다.

그림 12은 본 논문에서 구현한 UART 통신을 이용한 라즈베리파이 리듬 게임에 대한 응답 속도는 위의 표를 보았을 때와 같이 컴퓨터 성능에 따라서 응답 시간이 달라짐을 알 수 있다. 성능이 제일 안 좋은 노트북이 가장 느린 응답 속도를 가지고, 저 사양과 고 사양 PC의 속도는 둘 다 빠르기 때문에 차이를 확인하기 어렵다. 0.1초와 0.2초는 육안으로는 큰 차이를 확인할 수 없다. 그렇지만 수치상으로는 PC의 응답 시간이 노트북보다 2배가량 좋은 결과를 나타내 주고 있다. 결론적으로 노트북을 사용할 때와 PC을 사용할 때는 육안으로 확인하기 힘든 차이를 가지고 있지만 수치상으로는 2배의 성능차이를 보이기 때문에 PC을 사용하는 것이 더 반응이 빠른 리듬 게임을 할 수 있다.

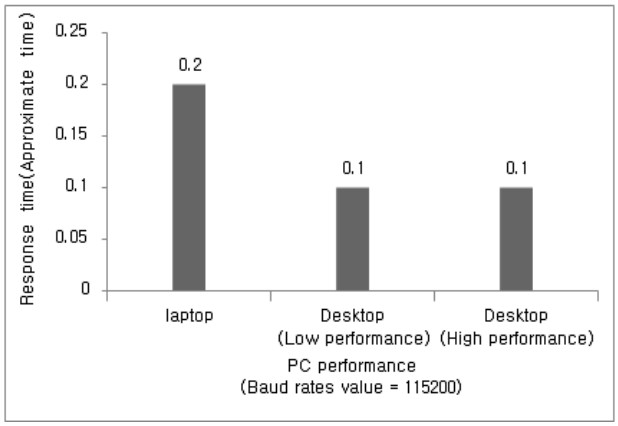

그림 12. PC 성능에 따른 대략적인 응답 속도 그래프
Fig. 12. Graph approximate response rate according to PC performance

Ⅳ. 결 론

본 논문에서는 운영체제가 다른 두 운영체제, 리눅스 운영체제를 사용하는 라즈베리파이 3과 윈도우 운영체제를 사용하는 PC를 UART 통신을 이용하여 윈도우 운영체제에서 구동 가능한 PC 게임의 조작키를 리눅스 운영체제인 라즈베리파이 3로 제어하여 리듬 게임을 구동해 보았다. 리듬 게임은 반응 속도가 중요하기 때문에 라즈베리파이 3의 GPIO에 연결되어 있는 버튼을 누를 시 즉각적으로 UART통신을 통해 PC를 제어하는 것을 중점으로 제작하였다. 이를 구현시키기 위해서 여러 보율의 값에 따른 응답 시간을 구해 비교하여 이에 가장 적합한 보율 값을 도출할 수 있었다. 이 도출된 값을 통해 라즈베리파이 3의 GPIO에 연결되어 있는 버튼을 누를 시 UART 통신을 이용하여

PC를 제어하는데 걸리는 시간을 최대한 줄일 수 있었다. 추가적으로는 라즈베리파이 3을 활용하여 할 수 있는 프로젝트들이 많이 있고, 자료들도 다양한 분야로 있기 때문에 다른 프로젝트들도 라즈베리파이 3을 이용하여 연구를 진행할 예정이다.

참고문헌

[1] Manisha Sharma, Nidhi Agarwal, SRN Reddy, “Design and development of daughter board for USB-UART communication between Raspberry Pi and PC”, International Conference on Computing, Communication & Automation, pp. 944–948, May 2015

[2] Kyung Sung, “Trend analysis of X Window used in Linux”, *Journal of Digital Contents Society,* Vol. 18, No. 7, pp. 1393-1401, Nov. 2017

[3] Young-Kyun Sang, Seong-Dong Son, Jung-Moon Lee · Dong-Hoi Kim, “Implementation of Autonomous Speed-controlled ExplorationRobot using Weather Information”, *Journal of Digital Contents Society*, Vol. 19, No. 5, pp. 1011-1019, May 2018

[4]https://ko.wikipedia.org/wiki/%EB%9D%BC%EC%A6%88%EB%B2%A0%EB%A6%AC_%ED%8C%8C%EC%9D%B4

[5] Byungsoon Kim, “Porting of Z-Stack and Implementation of UART on the TI CC2530”, *Journal of Digital Contents Society*, Vol. 13, No. 4, pp. 525-530, Dec. 2012

[6] https://ko.m.wikipedia.org/wiki/UART

[7] http://jhlblue.tistory.com/m/12

[8]https://m.blog.naver.com/PostView.nhn?blogId=hahav000&logNo=221322904700&categoryNo=9&proxyReferer=&proxyReferer=https%3A%2F%2Fwww.google.com%2F

[9] http://superkts.pe.kr/upload/helper/file1/keyCode.html

[10]http://kb.globalsoft.co.kr/web/web_view.php?notice_no=285

양희준(Hui-Jun Yang)

2014년 ~ 현재: 강원대학교 IT대학 전기전자공학과 재학

※관심분야 : 사물인터넷(IoT) 및 프로그래밍, 게임개발 등

김범준(Beom-Jun Kim)

2014년 ~ 현재: 강원대학교 IT대학 전기전자공학과 재학

※관심분야 : 사물인터넷(IoT) 및 네트워크, 프로그래밍 등

김동회(Dong-Hoi Kim)

2005년 : 고려대학교 전파공학과 (공학박사)

1989년 1월 ~ 1997년 1월 : 삼성전자 전임연구원
2000년 8월 ~ 2005년 8월 : 한국전자통신연구원 선임연구원
2006년 3월 ~ 현재 : 강원대학교 IT대학 전기전자공학과 교수
2018년 7월 ~ 현재 : 행정안전부 전자정부추진위원회 위원 등

※관심분야 : 무선 네트워크 및 사물인터넷(IoT) 등