3D Printer Calibration Program

2학년 김신건

본인은 학교 3D Printer 방과 후 수업을 진행하던 중, 3D Printer Calibration 과정에 대한 불편함을 가지게 되었다. 그리고 Calibration을 수동으로 진행하고 감으로 맞추는 것이 비효율적이라고 생각하게 되었다. 이를 계기로 3D Printer Calibration 프로그램을 제작하고자 하였다. 먼저, 3D Printer 방과 후 수업에서 사용하던 da Vinci 1.1 Plus 의 조정나사와 위치 측정 센서에 대해서 알아본 뒤, 간단한 삼각비의 원리를 이용해 Calibration 계산을 만들었고 이를 이용해 프로그램을 제작하였다. 그 후, 모든 3D Printer의 Calibrtion 프로그램을 제작하고자 조정나사와 위치 측정 센서의 위치와 그 위치들의 집합에 대한 일반화를 하였고 조정 나사의 위치에 따른 명령어의 집합을 일반화하였다. 일반화된 정보를 이용해 3D Calibration program을 제작하고자 했으나 3D Printer들이 사용하는 높이의 단위와 Calibration 완료 조건이 다르다는 점에 의해 Calibration Program의 일반화 과정까지 진행하였다. 이 연구를 통해서, 3D Printer Print Bed Calibration 과정에 편의성을 가져올 수 있었고 3D Printer 교육에 있어서 교사와 학생 모두에게 편의성을 가져올 수 있을 것으로 보인다.

1. 서론

1.1 연구배경

이 연구는 본인의 고등학교 3D Print 방과 후 수업 내용 중 XYZ printing 사의 da Vinci 1.1 plus 3D Print를 이용하여 3D Printer의 Print Bed를 Calibration하는 수업 내용에서 시작되었다. 이 수업을 통해 본인은 3D Printer의 Print Bed에 조정 나사의 위치와 높이 측정 지점의 위치가 다른 경우,Calibration을 직접 하는 사용자가 많은 어려움을 겪을 수 있다는 생각을 하게 되었다. 본인은 이 문제점을 컴퓨터 프로그램을 통해 da vinci 1.1 plus에 국한되지 않고 널리 사용가능한 Print Bed Calibration을 최소 횟수로 조정 나사를 조작하여 3D Priner 사용자가 편하게 Calibration을 할 수 있는 방법을 찾는 것을 목표로 연구를 하게 되었다.

1.2. 연구의 이론적 배경:

1.2.1. 3D Printer

1981년 일본 나고야 시 공업 연구소의 고다마 히데오가 이론으로 만들고 1986년 미국의 척 헐이 특허를 얻어 설립한 3D System사에서 제품을 출시한 것이 3D Printer의 시초이다. 3D Print 기술은 기존의 종이에 프린트하는 2D 프린터의 원리와 비슷하다. 단, 2D 프린터는 종이에 잉크를 쌓는 반면에 3D 프린터는 다양한 재료를 Print Bed에 쌓는다. 이 적층 가공 방식은 시간이 오래 걸리고 인쇄물에 휘어짐 현상이 일어난 다는 단점이 있다. 현재 3D Printer는 적층 가공이 아닌 절삭 가공의 형태도 있지만 적층 가공에 비해 인쇄물의 표면이 매끄럽지 못해 품질이 떨어진다.

1.2.2. Print Bed Calibration

Print Bed란 2D 프린터에서 잉크를 종이에다 쌓듯이 3D Printer에서 3D Print 소재를 쌓는 공간을 말한다. Calibration이란 저울로 무게를 제기 전에 영점을 조절하거나 모니터를 사용하기 전에 색 온도나 색 균형 등을 설정하는 행위를 의미한다. 이 연구에서 Print Bed Calibration은 Print Bed가 3D Print 소재가 나오는 사출기와 평행하게 만드는 행위를 의미한다. 이 Print Bed Calibration이 필요한 이유는 사출기와 Print Bed가 평행하지 않다면 3D 인쇄물을 뽑는 과정에서 3D 소재가 휘거나 인쇄가 되지 않는 경우가 생기기 때문이다.

1.2.3. 프로그래밍과 알고리즘

프로그래밍이란 프로그래머가 원하는 수식이나 행위가 있을 때, 특정 명령 코드를 입력하여 정해진 순서에 따라 컴퓨터가 연산하게 만드는 것이다. 알고리즘이란 어떠한 문제를 해결하려 하는 명령, 절차, 수식 등을 순서 있게 배치되어 있는 집합이다.

1.2.4. BFS (Breadth-First-Search)

BFS란 우리말로 너비 우선 탐색이라 한다. 너비 우선 탐색은 주어진 자료구조를 탐색하는 방법 중 하나이다. 탐색 방식은 먼저 시작 지점을 방문한 후, 시작 정점에 인접해 있는 정점을 모두 우선적으로 방문하고 더 이상 방문하지 않은 정점이 없을 때까지 방문한 정점의 방문하지 않은 인접한 정점을 방문한다. Fig 1에서 방문하는 순서가 보기 쉽게 나타내어 있다. BFS를 통해 그래프나 트리 등의 자료구조를 탐색할 수 있으며 BFS를 프로그램으로 구현할 때는 큐라는 자료구조가 사용된다.

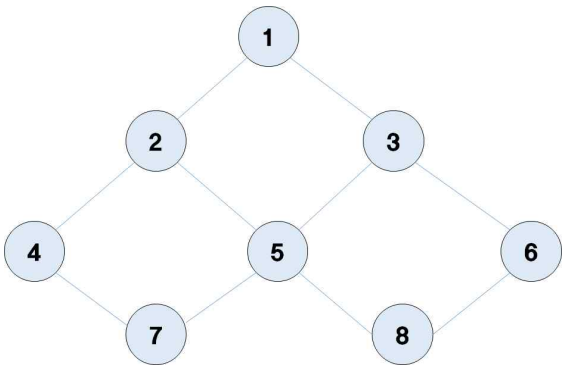

Fig. 1. BFS 알고리즘의 그래프 내의 탐색 순서

1이라 명시되어 있는 부모 노드를 시작지점으로 하여서 8이라 적혀 있는 정점까지 BFS 방식으로 탐색하였을 때 정점에 방문하는 순서를 표시하였다.

1.2.5. 동적계획법(Dynamic Programming)

동적 계획법이란 하나의 문제를 풀기 위해서 여러 개의 하위 문제로 나누어 푼 다음, 그것을 결합하여 하나의 문제를 해결하는 것이다. 각 하위 문제를 해결할 때마다 계산된 결과를 저장하여 계산횟수를 저장하고 동일한 하위 문제가 등장했을 때 다시 불러오는 방식으로 하위 문제의 수가 굉장히 많을 때, 빠른 시간 내에 해결할 수 있다.

1.3. 이 연구의 필요성

이 연구는 한민고등학교 방과 후 수업 진행 중에 생긴 문제점과 이를 해결하기 위해 시작되었다. 이는 학교 수업을 진행하면서 학생이 직접 느낀 문제점을 해결하기 위해 연구가 진행되었다는 것을 의미한다. 따라서 학생들에게 3D Printer 관련 교육을 할 때, 좀 더 학생들에게 편한 교육 환경을 제공할 수 있다. 물론, 3D Printer 사업에 있어서 Calibration 분야 발전을 이끌어낼 수 있고 3D Printer 사용자의 편의성을 높일 수 있다.

2. 본론

2.1. 연구 준비

2.1.1. da vinci 1.1plus(3D Printer)

Fig. 2. da vinci 1.1plus 3D Printer 사진

XYZprinting 사의 da vinci 1.1모형의 사진이며 이 3D Printer의 성능은 아래의 Table 1에 명시되어 있다. 이 3D Printer는 프로그램에서 20x20 사이즈의 Print Bed를 입력한 후 정확성을 실험하기 위해 사용되었다.

Table. 1. da vinci 1.1plus 3D Printer Specification

da vinci 1.1 plus 3D printer의 사양 중 연구와 관련 있는 사항만 명시한 표이다.

항목	사항
Model name	da Vinci 1.1 plus 3D Printer
Print Head	Single Head

Build Volume	20 x 20 x 20 cm

2.1.2. C++ IDE

본 연구에서는 두 정류의 C++ IDE를 사용하였다. 첫 번째 C++ IDE는 DEV C++이고 두 번째 C++ IDE는 Visual Studio Community이다. 그리고 프로그램 실행 환경은 Windows 7 구버전과 Windows 10 최신 버전이다.

2.2. 연구 방법

본 연구의 큰 순서는 다음과 같다.

첫째, da Vinci 1.1 Plus의 Calibration 방식을 분석해 Calibration을 할 때 필요한 계산을 분류하고 da Vinci 1.1 plus Print Bed Calibration에 적용할 수 있는 프로그램을 제작한다.

둘째, da Vinci 1.1 Plus를 이용해 분류한 계산들과 프로그램을 일반화하여 모든 3D Printer에 적용한다.

셋째, 일반화된 계산을 C++ 콘솔 프로그램으로 제작하고 실제 3D Printer에 적용하여 효율성을 분석한다.

넷째, C++ 콘솔로 제작된 프로그램을 JAVA나 HTML 등을 이용하여 3D Printer 사용자에게 편리한 형태의 애플리케이션을 제작한다.

2.3. 연구 내용

2.3.1. da Vinci 1.1 Plus의 Calibration 방식

2.3.1.1 Print Bed 높이 측정

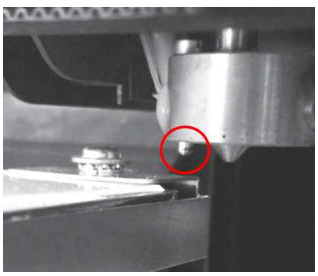

Fig. 3. da Vinci 1.1 Plus의 높이 측정 센서 사진

da Vinci 1.1 Plus의 높이 측정 방식은 압력으로 눌러서 반응하는 터치센서방식이 아닌 빨간색으로 표시되어 있는 부분과 Print Bed에서 측정되는 부분에 닿는 순간 전기가 흐르면서 그때의 높이가 측정되는 방식이다.

2.3.1.1 높이 측정 센서와 조정나사의 위치

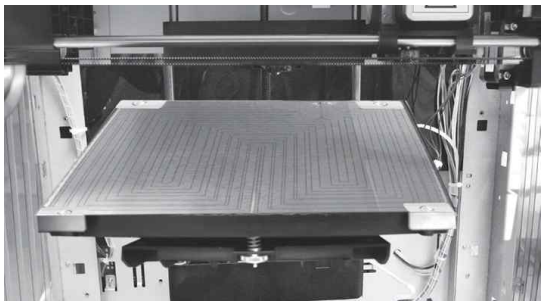

Fig. 4. da Vinci 1.1 Plus의 Print Bed 사진

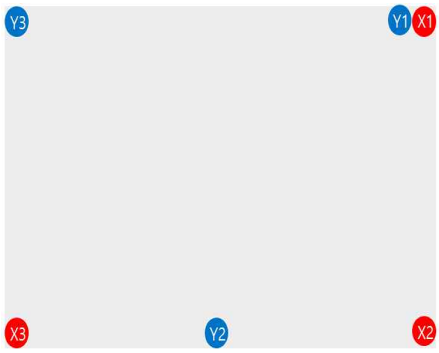

Fig. 5. da Vinci 1.1 Plus의 높이 측정 위치와 조정 나사의 위치를 나타낸 그림ㄴ

높이 측정 센서에 의해 Print Bed의 높이가 측정되는 부분은 빨간색 원(X)으로 표시되었고 3D 프린터 사용자가 Print Bed의 높이를 조정할 수 있는 조정나사의 위치는 파랑색 원(Y) 표시하였다. 그림 상, Y1과 X1이 옆에 나란히 있는 것으로 되어 있다. 하지만 실제로는 높이측정 위치와 조정나사의 위치는 Print Bed의 위아래로 위치되어 있기 때문에 Print Bed의 오른쪽 위 모서리에 Y1,X1이 둘 다 위치한다.

2.3.1.3 da Vinci 1.1 Plus의 조정나사를 움직였을 때의 Print Bed의 움직임

1) 삼각비

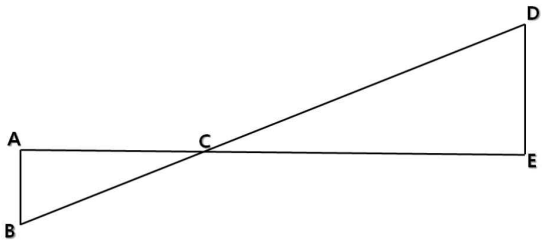

Fig. 6. AA닮음인 2개의 삼각형

각CAB와 각 CED는 직각으로 같고 각 ACB와 각 BCE는 맞꼭지각이므로 서로 같으므로 두 삼각형은 닮은 도형이다. 따라서 (선분 AC의 길이) : (선분 CE의 길이) = (선분 AB의 길이) : (선분 DE의 길이)가 성립하게 된다.

2) Y1을 조정한 경우

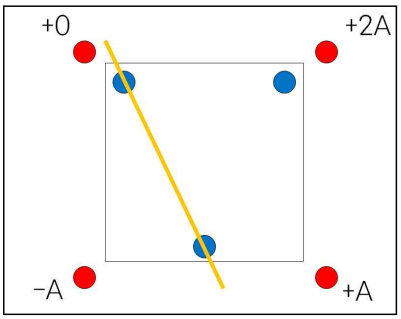

Fig. 7. da Vinci 1.1 Plus의 Print Bed에서 Y1을 조정한 경우

Y1을 +2A만큼 조정하는 경우 Fig 8에 나타나 있는 계산에 따라서 X1은 +2A, X2는 +A, X3는 -A, X4는 +0만큼 높이의 변화가 나타나게 된다.

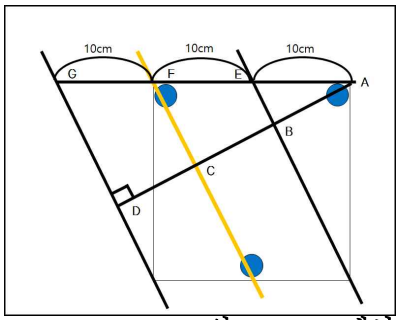

Fig. 8. da Vinci 1.1 Plus의 Print Bed에서 Y1을 조정한 후의 변화에 대한 계산

오른쪽 모서리의 Y1을 조정하는 경우 Y2,Y3가 고정되어 있기 때문에 선분 Y2Y3가 축 역할을 하게 된다. Y2와 Y3를 이은 노란색 선분을 기준으로 삼각형 ADG를 그리면 선분 AB, 선분 BC, 선분 CD가 같은 길이인 것을 알 수 있다. Fig 6에서의 삼각비에 따라 Y1을 2A만큼 조작하면 X1은 +2A, X2는 +A, X3는 -A, X4는 +0 만큼 높이의 변화가 나타나게 된다.

3) Y2를 조정한 경우

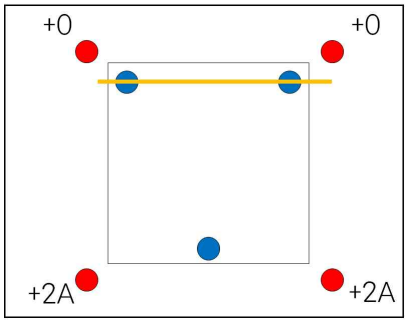

Fig. 9. da Vinci 1.1 Plus의 Print Bed에서 Y2를 조정한 경우

Y2를 +2A만큼 조정하는 경우 Fig 10에 나타나있는 계산에 따라서 X1은 +0, X2는 +2A, X3는 +2A, X4는 +0만큼 높이의 변화가 나타나게 된다.

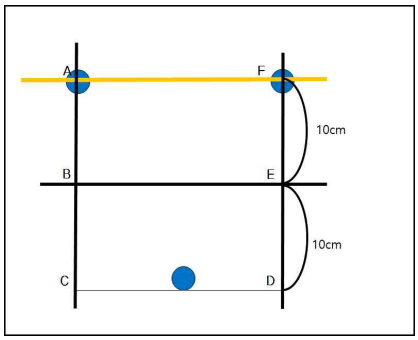

Fig. 10. da Vinci 1.1 Plus의 Print Bed에서 Y2를 조정한 후의 변화에 대한 계산

아래쪽 모선의 중점인 Y2를 조정하는 경우 Y1,Y3가 고정되어 있기 때문에 선분 Y1Y3가 축 역할을 하게 된다. 선분 AB와 선분 AC의 길이는 같기 때문에 Fig 6에서의 삼각비에 따라 Y2를 2A만큼 조작하면 X1은 +0, X2는 +2A, X3는 +2A, X4는 +0 만큼 높이의 변화가 나타나게 된다.

4) Y3를 조정한 경우

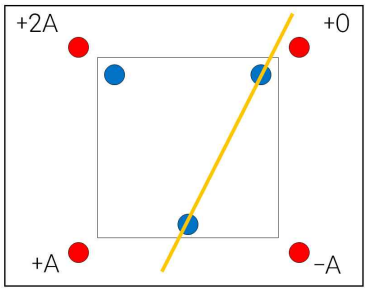

Fig. 11. da Vinci 1.1 Plus의 Print Bed에서 Y3를 조정한 경우

Y3를 +2A만큼 조정하는 경우 Fig 12에 나타나있는 계산에 따라서 X1은 +0, X2는 -2A, X3는 +A, X4는 +2A만큼 높이의 변화가 나타나게 된다.

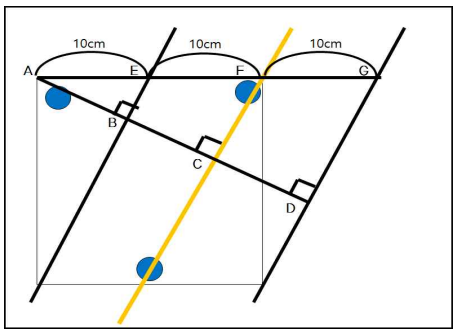

Fig. 12. da Vinci 1.1 Plus의 Print Bed에서 Y3를 조정한 후의 변화에 대한 계산

왼쪽 모서리의 Y3를 조정하는 경우 Y1,Y2가 고정되어 있기 때문에 선분 Y1Y2가 축 역할을 하게 된다. Y1과 Y2를 이은 노란색 선분을 기준으로 삼각형 ADG를 그리면 선분 AB, 선분 BC, 선분 CD가 같은 길이인 것을 알 수 있다. Fig 6에서의 삼각비에 따라 Y3를 2A만큼 조작하면 X1은 +0, X2는 -2A, X3는 +A, X4는 +2A만큼 높이의 변화가 나타나게 된다.

5) Y1,Y2,Y3를 움직이는 모든 경우

Table. 2. 조정나사의 종류에 따라 Print Bed 높이 측정 부분의 높이 변화

Y1,Y2,Y3를 움직이는 경우에 따라 X1,X2,X3,X4에서 측정되는 높이의 변화를 정리한 표이다.

조정하는 조정나사(+2a)	X1의 변화	X2의 변화	X3의 변화	X4의 변화
Y1	+2a	+a	-a	0
Y2	0	+2a	+2a	0
Y3	0	-a	+a	+2a

2.3.2. da Vinci 1.1 Plus Calibration program

2.3.2.1 Calibration Program 전제

프로그램 전제1: da Vinci 1.1 Plus에서는 조정나사를 1바퀴 조작할 경우 해당 부분의 높이가 0.5cm 변화하고 1/2바퀴 조작할 경우 해당 부분의 높이가 0.2~0.25cm 변화한다. 따라서 프로그램에서는 더 미세한 조작을 위해 1/4바퀴씩 돌렸을 때 0.12cm가 변화하는 것을 기준으로 프로그램을 제작하였다.

프로그램 전제2: da Vinci 1.1 Plus에서는 측정된 모든 높이가 2.3 ~ 2.6cm 범위 안으로 적용되어야 하며 측정된 각 높이는 0.2cm 내의 오차만 있어야 한다. 따라서 이 기준이 성립하고 가장 적은 이동횟수를 가진 Calibration 방법을 최적해로 출력한다.

2.3.2.2 Calibration Code

실제 C++ 콘솔 프로그램을 제작하면서 사용된 코드는 4. 부록에 명시하였다. 총 코드는 402줄이며 다음 함수들로 나뉘어 프로그램이 진행된다. Pic_PrintBed(), Calibration() , fns_check(info X) , init() , nodetracking() , Print_Sit(int Z, int ZCnt) , Pic_PrintBed2() , line() , main()

2.3.2.3 Program에서 사용된 함수 설명

init()

: 주요 함수들을 실행하기에 앞서서 memo배열을 매우 큰 수로 채워놓는 등의 프로그램 실행 준비 함수이다.

Calibration()

: BFS 탐색을 하는 함수로서 조정나사 3개를 각각 위와 아래로 조정하는 6가지의 방향으로 엣지를 설정해서 도달할 수 있는 노드의 거리를 저장하는 함수이다.

fns_check(info X)

: BFS 탐색을 통해서 어떠한 노드에 도달하였을 때, 그 노드가 Calibratiom 조건에 적합한지 검사하는 적합도 검사 함수이다.

nodetracking()

: fns_check(info X)를 통해서 어떠한 노드가 적합한지를 최종으로 알게 되었을 때, 그 노드에서 최초로 시작한 노드로 역행하면서 최종 노드까지의 스택에 저장한다.

Print_Sit(int Z, int ZCnt)

: nodetracking()를 통해서 저장된 스택에 있는 정보들을 하나씩 꺼내가며 Calibration을 하기 위해 사용자가 수행해야하는 행동으로 전환해 출력하는 함수이다.

Pic_PrintBed2()

: X1,X2,X3의 위치와 Printbed에 대한 그림을 콘솔창에 띄워주는 용도의 함수이다.

Pic_PrintBed2()

: Y1,Y2,Y3의 위치와 PrintBed에 대한 그림을 콘솔창에 띄워주는 용도의 함수이다.

line()

: 프로그램 UI에서 단락 구분 줄을 출력해주는 용도의 함수이다.

main()

: 앞서 말했던 모든 함수를 포함하는 주요 실행 함수이다.

2.3.2.4 Program 예시 결과 사진

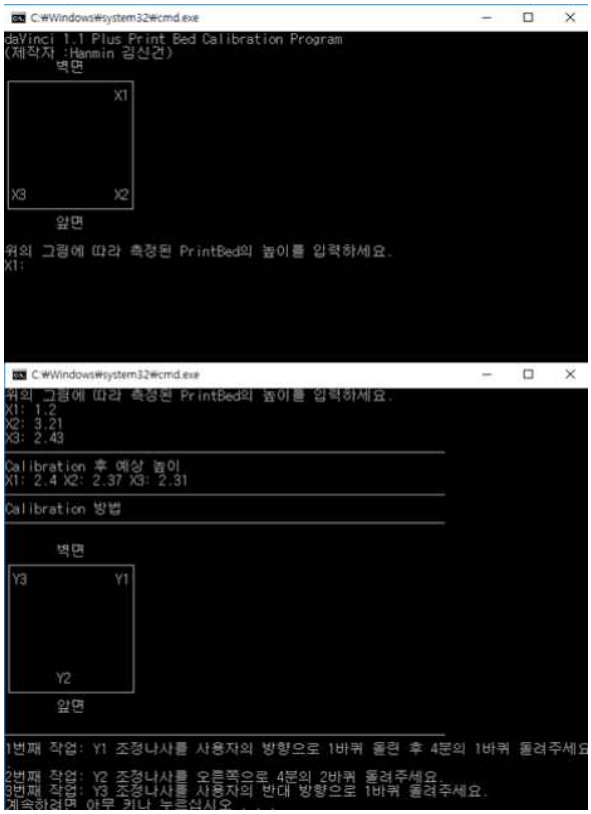

Fig. 13. da Vinci 1.1 plus calibration program 실행 사진

실제로 Visual studio를 통해 제작한 콘솔 응요 프로그램 실행사진이다. 예시 결과로는 X1, X2, X3에 1.20, 3.21, 2.43을 각각 입력하였고 3번의 조정나사를 조정하는 작업을 거치면 Calibration이 완료되는 것으로 프로그램 결과가 나타났다. 이 프로그램의 실행 파일(exe)는 다음 제시되어 있는 구글 드라이브 주소에 업로드되어 있다.

https://drive.google.com/file/d/0B-NMfjR104pWamZjZzI5R2IxUHc/view?usp=sharing

2.3.3. 3D Printer Print Bed Calibration 일반화

2.3.3.1 3D 프린터의 조정나사와 높이 측정부의 위치 일반화

3D Printer Print Bed Calibration을 일반화하기 위해서는 조정나사의 위치와 높이 측정 위치를 지정할 필요가 있다. Fig 10에서는 조정나사가 올 수 있는 경우를 나타내었다.

실제로 3D Printer에서 사용될 수 있는 조정나사의 경우의 수는 2.3.3.2에 나타나 있다.

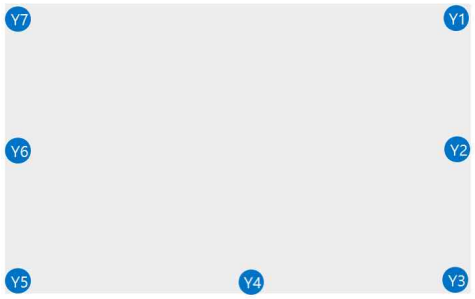

Fig. 14. 모든 3D printer에서 조정나사가 올 수 있는 경우를 나타낸 그림

이 연구에서는 조정나사가 올 수 있는 위치를 프린트 베드의 각 꼭짓점과 각 모서리의 중점으로 설정하였다. 하지만 위쪽의 Y8이 되어야 할 부분은 3D 프린터의 구조 상 벽면이 위치하기 때문에 배제하였다.

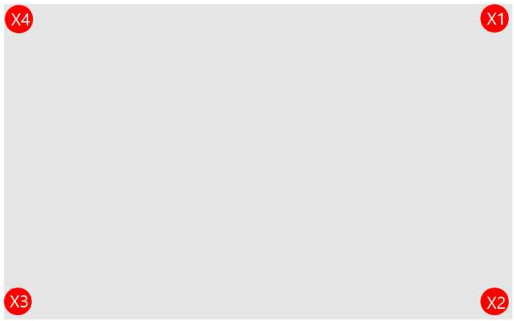

Fig. 15. 모든 3D printer에서 높이를 측정하는 부분이 올 수 있는 경우를 나타낸 그림

3D Printer의 Print Bed가 평행하게 유지되어 있는지를 검사하기 위해서는 Print Bed의 네 꼭짓점의 높이를 측정하는 것이 효율적이라고 고려되었고 높이를 측정하는 부분을 네 꼭짓점으로 설정하였다.

2.3.3.2 3D 프린터의 조정나사의 위치 집합의 경우의 수

Calibration 프로그램을 제작하기 전, 3D 프린터에서 사용하는 조정나사의 개수는 3개로 한정지었다. 그 이유는 조정나사를 1개 혹은 2개로 했을 경우에는 Print Bed의 균형이 안 맞아 Calibration의 유무에 상관없이 3D Print가 성공적으로 이루어지지 않는다. 그리고 조정나사를 4개 이상으로 하는 경우에는 Print Bed의 균형이 맞을 수는 있으나 Calibration을 하는 과정에서 조정 나사를 2개 이상을 동시에 돌려야 하기 때문에 사용자의 불편함을 가져올 수 있고 Print Bed의 영구적 손상을 불러올 수 있기 때문이다.

다음의 순서쌍들은 아무런 조건 없이 Print Bed에 조정나사가 있는 경우이다. (Xn=n으로 표기하였다.)

(1,2,3),(1,2,4),(1,2,5),(1,2,6),(1,2,7),(1,3,4),(1,3,5),(1,3,6),(1,3,7),(1,4,5),(1,4,6),(1,4,7),(1,5,6),(1,5,7),(1,6,7),(2,3,4),(2,3,5),(2,3,6),(2,3,7),(2,4,5),(2,4,6),(2,4,7),(2,5,6),(2,5,7),(2,6,7),(3,4,5),(3,4,6),(3,4,7),(3,5,6),(3,5,7),(3,6,7),(4,5,6),(4,5,7),(4,6,7),(5,6,7)

총 $_3$ 35 개로 나타난다. 하지만 조정나사 2개가 서로 이웃한 경우에는 fig 9와 같이 무게 중심이 무너지는 경우가 생긴다. 따라서 다음의 경우들은 배제해야 한다.

(1,2,3),(1,2,4),(1,2,5),(1,2,6),(1,2,7),(1,3,4),(1,4,5),(1,5,6),(1,6,7),(2,3,4),(2,3,5),(2,3,6),(2,3,7),(2,4,5),(2,5,6),(2,6,7),(3,4,5),(3,4,6),(3,4,7),(3,5,6),(3,6,7),(4,5,6),(4,5,7),(4,6,7),(5,6,7) 다음과 같이 조정나사 2개가 서로 이웃하여 균형이 무너지는 경우는 총 25개로 나타난다. 따라서 조정나사가 구현 될 수 있는 경우의 수는 10개로 나타난다. 10개의 경우의 수는 (1,3,5), (1,3,6), (1,3,7), (1,4,6), (1,4,7), (1,5,7), (2,4,6), (2,4,7), (2,5,7), (3,5,7)이다.

따라서 이 10가지의 경우가 아닌 경우가 입력되면 3D Printer의 조정나사 설계가 잘못된 것이나 사용자의 조정나사 위치 입력이 잘못된 것이므로 재입력을 하도록 프로그램을 제작한다.

2.3.3.3 조정나사의 위치에 따라 명령어 집합

Fig 14에서처럼 볼트는 어느 방향으로 돌리느냐에 따라서 너트의 이동방향이 달라진다. 그리고 Y2와 Y6을 비교하자면 Y2는 사용자로부터

반대쪽 방향으로 밀면 Print Bed가 아래로 내려온다. 반대로 Y6을 사용자로부터 반대쪽 방향으로 밀면 Print Bed가 아래로 내려온다. 따라서 사용자가 같은 방향으로 조정나사를 조작하더라도 조정나사의 위치에 따라 Print Bed의 움직임이 다르다.

다음 표는 Yn에서 n에 따라 달라지는 사용자가 수행해야 하는 행동을 정리한 것이다.

Table. 3. Yn에 따른 사용자의 행동

Yn에서 n이 달라지면 사용자가 조정나사를 같은 방향으로 돌리더라도 Print Bed의 높이 변화 방향이 달라질 수 있다. 사용자 방향은 조정나사를 사용자를 향해 돌리는 방향이고 사용자 반대 방향은 조정나사를 사용자로부터 먼 방향으로 돌리는 방향이다.

조정하는 조정나사	조정 방향	해당 부분의 높이 변화
Y1,Y2,Y3	사용자 방향	높이가 증가한다.
	사용자 반대 방향	높이가 감소한다.
Y4	왼쪽	높이가 증가한다,
	오른쪽	높이가 감소한다.
Y5.Y6,Y7	사용자 방향	높이가 감소한다.
	사용자 반대 방향	높이가 증가한다.

3. 결론

3.1. 연구 결과 및 오류 해석

본 연구에서는 사람이 직접 조정나사를 돌리기 때문에 조정나사를 얼마나 돌리는 지를 구분할 수 있는 정도의 1/4바퀴를 기준으로 하고 프로그램을 제작하였다. 하지만 da Vinci 1.1 Plus Calibration program에 실험 데이터를 넣다보면 예상 높이가 도출이 되지 않고 결과 값이 나오지 않는 경우가 있었다. 이 오류는 Calibration program에서 조정나사를 돌리는 기본 단위를 1/4로 설정하면서 높이 변화가 0.12~0.24씩 나타났고 이보다 작은 숫자는 조절이 불가능하면서 생긴 것으로 보인다.

따라서 기본단위를 1/8 또는 1/16로 적용하면서 사용자가 직접 조정나사를 돌리는 것이 아닌 3D printer가 조정나사를 돌리면 정확도를 더 높이고 사용자의 편의 또한 증진시킬 수 있을 것으로 보인다.

조정나사와 높이 측정부의 위치와 이 위치들이 오는 집합의 경우의 수에 대해서와 조정나사의 위치에 따라 달라지는 명령어의 집합 같은 3D printer Calibration Program의 일반화가 이루어졌다. 이를 통해서 모든 3D Printer에 대한 Calibration 제작 프로그램을 제작하려 했으나 3D Printer마다 높이 측정 단위가 다르고 Calibration 완료 조건이 다르기 때문에 본 연구에서는 3D Printer Calibration의 일반화만 다루었다.

3.2. 기대 효과

이 연구를 통해서 기대되는 효과는 다음과 같다.

1) 3D Printer Calibration 과정 간편화
실제로 da Vinci 1.1 plus를 Calibration을 하다보면 시간이 굉장히 오래 걸린다. 따라서 이 연구를 통해서 이 Calibration 시간을 단축시키고 3D printer를 처음 접하는 사람에게 편의성을 제공할 수 있다.

2) 3D Printer 교육 접근성 증가
이 연구를 시작하게 된 계기는 3D Printer 방과 후 수업을 진행하던 중 Calibration 과정에 불편함을 느껴서이다. 이 연구를 완료함에 따라 방과 후 수업에서 Calibration이 더 쉽게 진행되고 학생들이 더 쉽게 3D Printer 수업을 배울 수 있다.

3.3. 도움주신 분들

1) Facebook 그룹, 생활코딩

Fig 16.Facebook 그룹인 생활코딩에 연구자가 올린 게시 글과 글의 댓글의 스크린 샷 이 게시 글의 링크는 https://www.facebook.com/groups/codingeverybody/?multi_permalinks=1855935464446981&notif_id=1507378631774363&notif_t=like 이다.

참고문헌

[1] 위키 백과 우리 모두의 백과사전 , https://ko.wikipedia.org/wiki/3%EC%B0%A8%EC%9B%90_%EC%9D%B8%EC%87%84

[2] 하성욱, 정보 올림피아드를 준비하는 초중고생을 위한 알고리즘.1 , 좋은땅, 2008.07.30.

[3] XYZ Pring 사 , da Vinci 1.1 Plus Quick Guide_EN.pdf

4. 부록

다음 코드는 본 연구에서 사용된 da Vinci 1.1 Plus Print Bed Calibration의 main.cpp의 코드 내용이다.

```
#include <iostream>
#include <queue>
#include <stack>
#define Mx 999999999
using namespace std;

struct info
{
int x1, x2, x3, N, Sit;
};

info S,ANSWER;
double X[10];
info memo[400][400][400];
queue <info> cal;
stack <info> Ans;
int num = 1;

void Pic_PrintBed();
void Calibration();
bool fns_check(info X);
void init();
void nodetracking();
void Print_Sit(int Z, int ZCnt);
void Pic_PrintBed2();
void line();

int main()
{
system("mode con cols=80 lines=30");
init();
ANSWER.N = Mx
cout << "daVinci 1.1 Plus Print Bed Calibration Program" << endl << "(제작자:Hanmin 김신건)" << endl;
Pic_PrintBed();

cout << "위의 그림에 따라 측정된 PrintBed의 높이를 입력하세요." << endl;
cout << "X1: "
cin >> X[1];
cout << "X2: "
cin >> X[2];
cout << "X3: "
cin >> X[3];

S.x1 = (int)((double)100 * X[1]);
```

```
S.x2 = (int)((double)100 * X[2]);
S.x3 = (int)((double)100 * X[3]);
S.N = 0;
S.Sit = 0;
memo[S.x1][S.x2][S.x3].N = 0;
cal.push(S);
Calibration();
line();
cout << "Calibration 후 예상높이"<<endl;
cout <<"X1: "<<(double)ANSWER.x1/100 <<" X2: "<< (double)ANSWER.x2/100 << " X3: " << (double)ANSWER.x3/100 << " " << endl;
line();
cout << "Calibration 방법" << endl;
line();
Pic_PrintBed2();
line();
Ans.push(ANSWER);
nodetracking();
}

void init()
{
for (int x = 0; x < 400; x++)
for (int y = 0; y < 400; y++)
for (int z = 0; z < 400; z++)
memo[x][y][z].N = Mx,memo[x][y][z].Sit=-1;
}

void Pic_PrintBed()
{
int Size =10;
cout << "       벽면" << endl;
for (int y = 1; y <= Size; y++)
{
for (int x = 1; x <= Size; x++)
{
if (x == 1 && y == 1)cout << (char)1;
else if (x == Size && y == 1)cout << (char)2;
else if (x == 1 && y == Size)cout << (char)3;
else if (x == Size&&y == Size)cout << (char)4;
else if (x == 1 || x == Size)cout << (char)5;
else if (y == 1 || y == Size)cout << (char)6
<< (char)6;
else if (x == Size - 1 && y == 2)cout << "X1"
else if (x == Size - 1 && y == Size - 1)cout
<< "X2"
else if (x == 2 && y == Size - 1)cout << "X3"
else cout << "  "
}
cout << endl;
}
cout << "       앞면" << endl << endl;
}

void Pic_PrintBed2()
{
int Size = 10;
cout << endl << "       벽면" << endl;
for (int y = 1; y <= Size; y++)
{
for (int x = 1; x <= Size; x++)
{
if (x == 1 && y == 1)cout << (char)1;
else if (x == Size && y == 1)cout << (char)2;
else if (x == 1 && y == Size)cout << (char)3;
else if (x == Size&&y == Size)cout << (char)4;
else if (x == 1 || x == Size)cout << (char)5;
else if (y == 1 || y == Size)cout << (char)6
<< (char)6;
else if (x == Size - 1 && y == 2)cout << "Y1"
else if (x == Size / 2 && y == Size - 1)cout
<< "Y2"
else if (x == 2 && y == 2)cout << "Y3"
else cout << "  "
}
cout << endl;
}
cout << "       앞면" << endl << endl;
}

void line()
{
for (int x = 0; x < 60; x++)cout << (char)6;
cout << endl;
}
```

```
bool fns_check(info X)
{
int Gap,falseCheck=0;
if (X.x1 > 230 && X.x2 > 230 && X.x3 > 230)
{
if(X.x1 < 260 && X.x2 < 260 && X.x3 < 260)
{
if (X.x1 > X.x2) Gap = X.x1 - X.x2;
else Gap = X.x2 - X.x1;
if (Gap > 20) return false,falseCheck++;

if (X.x2 > X.x3) Gap = X.x2 - X.x3;
else Gap = X.x3 - X.x2;
if (Gap > 20) return false, falseCheck++;

if (X.x1 > X.x3) Gap = X.x1 - X.x3;
else Gap = X.x3 - X.x1;
if (Gap > 20) return false, falseCheck++;

if (falseCheck == 0) return true
}
}return false
}


void Calibration()
{

w h i l e (!cal.empty()/*&&!fns_check(cal.front())*/)
{
info Z;
int Xnum;
Z.x1 = cal.front().x1;
Z.x2 = cal.front().x2;
Z.x3 = cal.front().x3;
Z.N = cal.front().N;
Z.Sit = cal.front().Sit;

if (fns_check(Z)&&ANSWER.N>Z.N)
{
//cout << Z.x1 << " " << Z.x2 << " " << Z.x3 << " " << Z.N << endl;
ANSWER = Z;
}
// 1번상황
if (Z.x1 + 24 < 400 && Z.x2 + 12 < 400 && Z.x3 - 12 > 50)
{
if (Z.Sit == 1) Xnum = Z.N;
else Xnum = Z.N + 1;

if (memo[Z.x1 + 24][Z.x2+ 12][Z.x3 - 12].N > Xnum)
{
memo[Z.x1 + 24][Z.x2 + 12][Z.x3 - 12].x1 = Z.x1;
memo[Z.x1 + 24][Z.x2 + 12][Z.x3 - 12].x2 = Z.x2;
memo[Z.x1 + 24][Z.x2 + 12][Z.x3 - 12].x3 = Z.x3;
memo[Z.x1 + 24][Z.x2 + 12][Z.x3 - 12].N = Xnum;
memo[Z.x1 + 24][Z.x2 + 12][Z.x3 - 12].Sit = 1;

info SS;
SS.x1 = Z.x1 + 24;
SS.x2 = Z.x2 + 12;
SS.x3 = Z.x3 - 12;
SS.N = Xnum;
SS.Sit = 1;
cal.push(SS);
}
}


//2번상황
if (Z.x1 - 24 >50 && Z.x2 - 12 > 50 && Z.x3 + 12 < 400)
{
if (Z.Sit == 2) Xnum = Z.N;
else Xnum = Z.N + 1;

if (memo[Z.x1 - 24][Z.x2 - 12][Z.x3 + 12].N > Xnum)
{
```

```
memo[Z.x1 - 24][Z.x2 - 12][Z.x3 + 12].x1 = Z.x1;
memo[Z.x1 - 24][Z.x2 - 12][Z.x3 + 12].x2 = Z.x2;
memo[Z.x1 - 24][Z.x2 - 12][Z.x3 + 12].x3 = Z.x3;
memo[Z.x1 - 24][Z.x2 - 12][Z.x3 + 12].N = Xnum;
memo[Z.x1 - 24][Z.x2 - 12][Z.x3 + 12].Sit = 2;

info SS;
SS.x1 = Z.x1 - 24;
SS.x2 = Z.x2 - 12;
SS.x3 = Z.x3 + 12;
SS.N = Xnum;
SS.Sit = 2;
cal.push(SS);
}
}

//3번상황
if (Z.x1 > 50 && Z.x2 + 24 < 400 && Z.x3 + 24 < 400)
{
if (Z.Sit == 3) Xnum = Z.N;
else Xnum = Z.N + 1;

if (memo[Z.x1][Z.x2 + 24][Z.x3 + 24].N > Xnum)
{
memo[Z.x1][Z.x2 + 24][Z.x3 + 24].x1 = Z.x1;
memo[Z.x1][Z.x2 + 24][Z.x3 + 24].x2 = Z.x2;
memo[Z.x1][Z.x2 + 24][Z.x3 + 24].x3 = Z.x3;
memo[Z.x1][Z.x2 + 24][Z.x3 + 24].N = Xnum;
memo[Z.x1][Z.x2 + 24][Z.x3 + 24].Sit = 3;

info SS;
SS.x1 = Z.x1;
SS.x2 = Z.x2 + 24;
SS.x3 = Z.x3 + 24;
SS.N = Xnum;
SS.Sit = 3;
cal.push(SS);
}
}

//4번상황

if (Z.x1 > 50 && Z.x2 - 24 > 50 && Z.x3 - 24 > 50)
{
if (Z.Sit == 4) Xnum = Z.N;
else Xnum = Z.N + 1;

if (memo[Z.x1][Z.x2 - 24][Z.x3 - 24].N > Xnum)
{
memo[Z.x1][Z.x2 - 24][Z.x3 - 24].x1 = Z.x1;
memo[Z.x1][Z.x2 - 24][Z.x3 - 24].x2 = Z.x2;
memo[Z.x1][Z.x2 - 24][Z.x3 - 24].x3 = Z.x3;
memo[Z.x1][Z.x2 - 24][Z.x3 - 24].N = Xnum;
memo[Z.x1][Z.x2 - 24][Z.x3 - 24].Sit = 4;

info SS;
SS.x1 = Z.x1;
SS.x2 = Z.x2 - 24;
SS.x3 = Z.x3 - 24;
SS.N = Xnum;
SS.Sit = 4;
cal.push(SS);
}
}

//5번상황

if (Z.x1 > 50 && Z.x2 - 24 > 50 && Z.x3 + 24 < 400)
{
if (Z.Sit == 5) Xnum = Z.N;
else Xnum = Z.N + 1;

if (memo[Z.x1][Z.x2 - 24][Z.x3 + 24].N > Xnum)
{
memo[Z.x1][Z.x2 - 24][Z.x3 + 24].x1 = Z.x1;
memo[Z.x1][Z.x2 - 24][Z.x3 + 24].x2 = Z.x2;
memo[Z.x1][Z.x2 - 24][Z.x3 + 24].x3 = Z.x3;
memo[Z.x1][Z.x2 - 24][Z.x3 + 24].N = Xnum;
memo[Z.x1][Z.x2 - 24][Z.x3 + 24].Sit = 5;
```

```
info SS;
SS.x1 = Z.x1;
SS.x2 = Z.x2 - 24;
SS.x3 = Z.x3 + 24;
SS.N = Xnum;
SS.Sit = 5;
cal.push(SS);
}
}

//6번상황

if (Z.x1 > 50 && Z.x2 + 24 < 400 && Z.x3 - 24
> 50)
{
if (Z.Sit == 6) Xnum = Z.N;
else Xnum = Z.N + 1;

if (memo[Z.x1][Z.x2 + 24][Z.x3 - 24].N > Xnum)
{
memo[Z.x1][Z.x2 + 24][Z.x3 - 24].x1 = Z.x1;
memo[Z.x1][Z.x2 + 24][Z.x3 - 24].x2 = Z.x2;
memo[Z.x1][Z.x2 + 24][Z.x3 - 24].x3 = Z.x3;
memo[Z.x1][Z.x2 + 24][Z.x3 - 24].N = Xnum;
memo[Z.x1][Z.x2 + 24][Z.x3 - 24].Sit = 6;

info SS;
SS.x1 = Z.x1;
SS.x2 = Z.x2 + 24;
SS.x3 = Z.x3 - 24;
SS.N = Xnum;
SS.Sit = 6;
cal.push(SS);
}
}

cal.pop();
}
}

void nodetracking()
{
//cout << endl;
while (Ans.top().N != 0)
{
info ZZ;
ZZ.x1                                    =
memo[Ans.top().x1][Ans.top().x2][Ans.top().x3]
.x1;
ZZ.x2                                    =
memo[Ans.top().x1][Ans.top().x2][Ans.top().x3]
.x2;
ZZ.x3                                    =
memo[Ans.top().x1][Ans.top().x2][Ans.top().x3]
.x3;
ZZ.N                                     =
memo[Ans.top().x1][Ans.top().x2][Ans.top().x3]
.N;
ZZ.Sit                                   =
memo[Ans.top().x1][Ans.top().x2][Ans.top().x3]
.Sit;
Ans.push(ZZ);
//cout << ZZ.x1 << " " << ZZ.x2 << " " <<
ZZ.x3 << endl;
}

//cout << endl;
int Z = -1, ZCnt = 0;
while (!Ans.empty())
{
//cout << Ans.top().Sit << endl;
if (Ans.size() == 1)
{
Print_Sit(Ans.top().Sit, ZCnt);
}
else if (Z == Ans.top().Sit)
{
ZCnt++;
}
else
{
Print_Sit(Z, ZCnt);
Z = Ans.top().Sit;
ZCnt = 1;
}
```

```
Ans.pop();
}
}

void Print_Sit(int Z, int ZCnt)
{

switch (Z)
{
case 1:
cout << num++ << "번째 작업: "
cout << "Y1 조정나사를 사용자의 방향으로"
break
case 2:
cout << num++ << "번째 작업: "
cout << "Y1 조정나사를 사용자의 반대 방향으로"
break
case 3:
cout << num++ << "번째 작업: "
cout << "Y2 조정나사를 왼쪽으로"
break
case 4:
cout << num++ << "번째 작업: "
cout << "Y2 조정나사를 오른쪽으로"
break
case 5:
cout << num++ << "번째 작업: "
cout << "Y3 조정나사를 사용자의 반대 방향으로"
break
case 6:
cout << num++ << "번째 작업: "
cout << "Y3 조정나사를 사용자의 방향으로"
break
}

if (Z!=0&&Z!=-1&&ZCnt % 4 == 0)
{
cout << ZCnt / 4 << "바퀴 돌려주세요." <<
endl;
}
else if(Z != 0 && Z != -1&&ZCnt/4!=0)
{
cout << ZCnt / 4 << "바퀴 돌린 후" << "4분의"
<< ZCnt % 4 << "바퀴 돌려주세요." << endl;
}
else if(Z != 0 && Z != -1)
{
cout<< "4분의" << ZCnt % 4 << "바퀴 돌려주세
요." << endl;
}
}
```