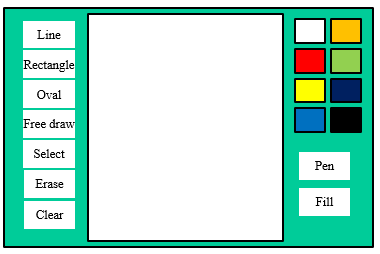
**Touch screen Lab**

1. **과제**



User Interface

* Touch-screen을 활용하여 그림판을 만든다.
* Touch-screen을 입력장치로 하여 LCD 상에 그림을 그릴 수 있다.
* Line, Rectangle, Oval은 rubber-band를 지원한다.
* 버튼

1. Line : 두 점을 잇는 선
2. Rectangle : 두 점을 맞꼭지점으로 하는 네모
3. Oval : 두 점을 맞꼭지젓으로 하는 네모에 접하는 타원
4. Free draw : 입력되는 점으로 이루어진 임의의 도형
5. Select : 그려진 객체를 선택하여 이동
6. Erase : 지우개
7. Clear : 전체 화면 지우기
8. Pen : 테두리 색
9. Fill : 도형 채우기
10. 8가지 색
11. **과제수행**

이제까지 배운 CLCD와 TFT LCD의 내용을 활용하여 이번 Lab을 수행한다.

* 1. **내용 연구**

1. 도형을 그리고 지웠다가 다시 디스플레이에 나타내기 위해서 그려진 도형을 저장할 자료구조가 필요하다. 구조체 listnode 안에 다음으로 연결된 노드를 표시해주는 포인터를 넣어 도형을 움직이고 지우고 다시 출력할 때 사용한다.
2. Free draw를 노드 안에 넣을 수 있지만, start와 end만 넣으면 다시 출력할 수 없다는 문제가 있다. 이를 해결하기 위해 bool total[196][196]을 사용한다. 선을 그릴 때마다 그 위치에 true를 넣어준다.
   * 1. **개발 환경**

Linux version : Ubuntu 18.04.4 LTS

실행 명령어 : gcc lab4.c -o lab4 -lwiringPi -lwiringPiDev -lpthread -lcrypt -lm -lrt

* + 1. **디스플레이**

1. 8가지 색은 makepixel()을 사용하여 만든다. white, orange, red, green, yellow, navy, blue 그리고 black이 있다.
2. clear\_and\_print()

이 함수는 그림을 그릴 수 있는 영역(64, 22, 260, 218)을 지워준 후 바뀐 위치에 따라 도형을 다시 그려주는 역할을 한다.

1. 이것의 또 다른 역할은 Select를 통해 도형을 움직일 때 흰 영역을 벗어났다가 다시 돌아왔을 때도 영역을 벗어난 부분이 사라지지 않고 유지되도록 하는 것이다.

void clear\_and\_print(listnod\* head) 중 Select에 영향을 주는 부분

for (i = start.y; i < end.y; i++) {

for (j = start.x; j < end.x; j++) {

if (i < 22 || i > 218 || j < 64 || j > 260)

{

continue;

}

if (total[i - start.y][j - start.x] == true) continue;

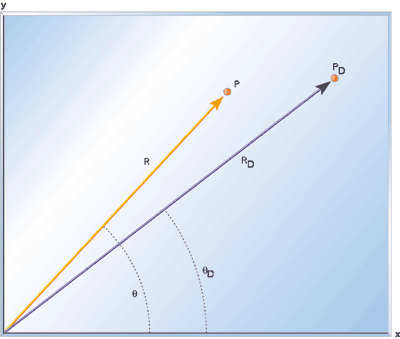
offset = i \* 320 + j;

\*(pfbdata + offset) = white;

}

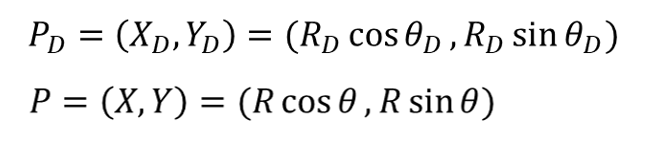
}

1. 리스트에 저장된 순서에 따라 노드에 저장된 id를 확인하며 도형을 그려준다.
2. Free draw는 규칙적인 모양을 갖고있지 않기 때문에 노드에 start와 end만으로 저장할 수 없어서 total 배열을 사용한다. 이 배열에는 사용자가 흰 화면을 터치했을 때 그 셀이 true로 바뀐 값이 들어가고, 눌린 부분을 다시 출력할 수 있다.
   * 1. **터치 보정**
3. setCalibration()을 사용하여 LCD와 터치스크린을 보정한다.

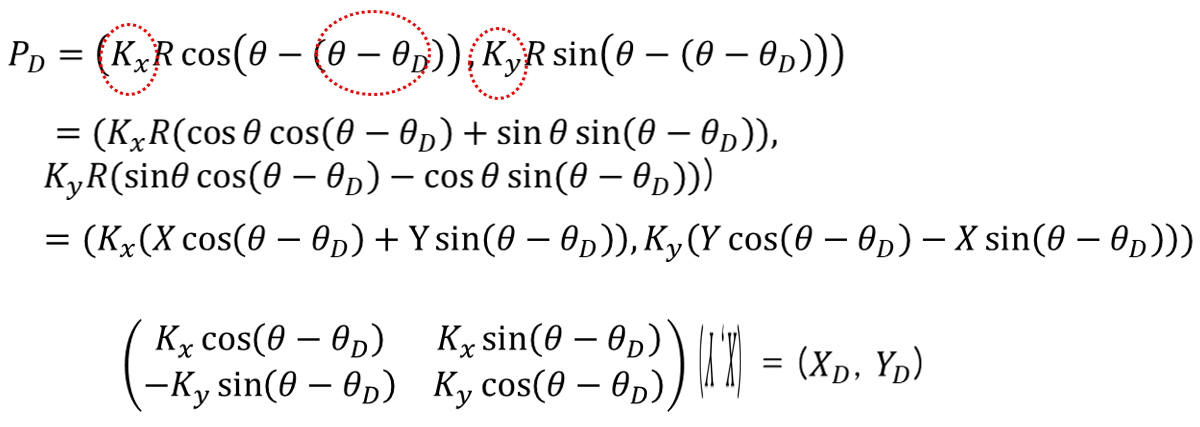


1. LCD의 좌표는 240x320이고, 터치스크린의 좌표는 따로 존재한다. LCD보다 터치스크린은 훨씬 큰데, 터치스크린과 LCD의 좌표가 다르기 때문에 매핑시켜줘야 한다. 이 차이를 줄여주는 함수가 setCalibration()이다.
2. 여기에서 사용되는 원리는 다음과 같다.

두 점이 있을 때 원점으로부터 선을 긋고 이 둘의 사이각을 알고 x와 y 좌표가 움직인 정도를 알면 한 점을 다른 점으로 이동시킬 수 있다.



각각의 점은 코사인과 사인으로 표현된다.



행렬 연산을 통해 두 점을 매핑시킬 수 있다.

1. 터치 보정 이후 setting()을 통해 터치스크린에 그림판 모양을 나타낸다.
   * 1. **자료구조**

* 연결리스트 구조체

도형을 그린 후에 연결리스트에 넣어 Select와 Erase를 수행할 수 있다.

이 연결리스트는 listnode 구조체로 만들어져 있다.

typedef struct listnode {

int id;

Point start;

Point end;

unsigned short linecolor;

unsigned short fillcolor;

struct listnode\* link;

}listnode;

1. id는 어떤 타입의 도형인지 나타낸다. 1은 Line, 2는 Rectangle, 3은 Oval 그리고 4는 Free draw이다.
2. start와 end은 도형의 시작점과 끝점이다.
3. linecolor와 fillcolor는 테두리 색과 채우기 색이다.
4. link는 다음에 그려진 도형을 가리킨다.

* listnode를 사용한 함수

1. Add()

사용자가 스크린에서 손을 떼었을 때 그린 도형을 listnode에 넣어준다.

listnode\* Add(listnode\* head, int id, Point start, Point end, unsigned short line, unsigned short f)

{

if (head->link == NULL)

{

head = insert\_first(head, id, start, end, line, f);

return head;

}

listnode\* cur = head->link;

listnode\* p = (listnode\*)malloc(sizeof(listnode));

p->id = id;

p->start.x = start.x; p->start.y = start.y;

p->end.x = end.x;p->end.y = end.y;

p->link = NULL;

p->linecolor = line; p->fillcolor = f;

while (cur->link != NULL)

cur = cur->link;

cur->link = p; // 마지막에 추가

return head;

}

1. Select

selection()을 통해 움직이는 동안 출력한다.

1. 선택한 부분에 도형이 없으면 움직이는 도형이 없다.
2. 도형이 있다면 id, linecolor 그리고 fillcolor를 가져와서 움직이는 start와 end를 반영하여 출력한다.

Change.x = click.x;

Change.y = click.y;

dx = (Change.x - Center.x);

dy = (Change.y - Center.y);

loc\_s = temp->start;

loc\_e = temp->end;

loc\_s.x += dx; loc\_e.x += dx;

loc\_s.y += dy; loc\_e.y += dy;

int selection(listnode\* head, Point click, unsigned short color, bool checkPen)의 일부

1. Erase()

Erase()는 search()부터 시작한다. 사용자가 클릭한 부분에 그림이 그려져 있는지 확인한다.

1. delete == NULL이라도 Free draw는 total에 따로 저장되어있기 때문에 total에 남아있을 수 있다. 만약 total에서 true인 곳이 있다면 그 부분을 white로 바꾸고 total을 지운다.
2. delete != NULL이라면 저장된 도형이 있다는 것이다. 지우려는 도형의 id를 보고 해당해는 도형에 따라 흰색으로 바꾸고 노드에서 삭제하는 작업을 한다.

int search\_pixel(Point start, Point end, Point click) {

int i, j;

int c\_x = (int)(end.x - start.x) / 2;

int c\_y = (int)(end.y - start.y) / 2;

if ((abs(start.x + c\_x - click.x) < MAX\_PIXEL\_SIZE) && (abs(start.y + c\_y - click.y) < MAX\_PIXEL\_SIZE))

return 1;

return 0;

}

도형의 중심에서 일정 거리 안에 있을 때 클릭한 부분에 도형이 있음을 인지한다.

listnode\* search(listnode\* head, Point click)

{

printf("In search function!!\n");

listnode\* p = head->link;

int i = 0;

while (p != NULL) {

i = search\_pixel(p->start, p->end, click);

if (i == 1) {

printf("Find!! \n");

return p;

}

p = p->link;

}

return NULL;

}

search\_pixel()의 반환값이 1이면 찾은 것이다.

1. Clear

deleteAll()을 통해 리스트에 저장된 head를 제외한 모든 노드는 삭제되고, 노드가 저장된 메모리는 반환된다.

void deleteAll(listnode\* head)

{

listnode\* p = head->link;

listnode\* cur;

for (; p != NULL; )

{

cur = p;

p = p->link;

head->link = p;

printf("Memory return\n");

free(cur);

}

free(head);

}

* + 1. **그래픽 도형 알고리즘**

1. makeLine()

Start

End

일직선을 그리기 위해서 gradient를 사용한다. start와 end로 기울기를 구한 후에 함수를 만들어 이 함수에 속하는 (x, y)를 출력한다.

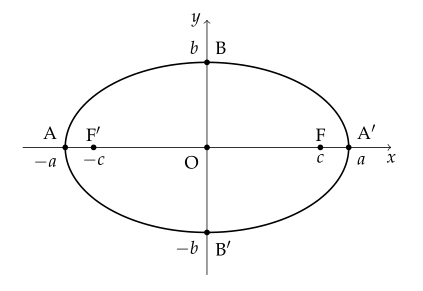
1. makeLineBox()

Start

End

사각형을 그릴 때 start와 end 사이를 돌며 점을 찍는다.

1. makeOval()



1. 타원의 방적식을 쓴다. start와 end를 통해 a와 b를 구한 후 for문을 통해 타원을 이동하여 출력한다.
2. 타원 또한 이전과 바뀐 사항에 대해서만 나타내고, 사용자가 손을 떼었을 때 listnode에 저장되며 최종적으로 저장된 도형을 출력한다.
3. Pen이 눌렸다면 checkPen은 true가 되어 테두리만 나타나고, Fill이라면 checkPen은 false가 되어 채우기 색만 보인다.
4. makefreeDraw()
5. 초기 start와 end는 같은 값이고, 이후 end에는 새로운 값이 들어간다.
6. 이 함수는 이전 좌표와 새로운 좌표 사이에 선을 긋는 함수이다. 픽셀 좌표가 연속으로 되어있지 않은 정수계 좌표라서 기울기에 따라서 선이 띄엄띄엄하게 나타날 수 있기 때문에 브레슨햄 알고리즘을 사용하여 좌표끼리 부드러운 선을 그릴 수 있다. 이 알고리즘은 기울기를 오차로 생각해 이를 누적하여 찍을 픽셀의 위치를 정한다.
7. checkPen가 true일 때 Pen이 사용되는 것이고, 반대로 false면 Fill이다. true면 사용자가 누른 색을 펜의 색으로 저장하고 아니면 채우기 색으로 저장한다.
   1. **동작**
8. Pen과 Fill 중 아무것도 선택하지 않았을 땐 검은색 펜을 기본으로 한다.
9. 무한루프를 돌며 사용자에게 값을 받는다.
10. 버튼이 눌렸을 때 위치를 확인하며, 어떤 버튼이 눌렸는지 main에 저장되는 변수인 flag를 바꿔준다. flag는 1부터 9까지 있으며, Line부터 Fill까지 순서대로 저장된다. 색이 눌렸을 때는 currentcolor의 값을 바꿔준다.
11. 사용자가 그림을 그릴 때
12. Line, Rectangle 그리고 Oval은 rubber-band를 지원하기 때문에 사용자가 입력한 값이 없을 때 end의 초깃값은 start의 값과 동일하게 한다.
13. 사용자가 손을 떼었을 때 listnode에 [Add()](#Add)로 넣어준다.
14. Select를 누를 때 (flag = 5)
15. [selection()](#selection)을 통해 도형이 클릭되었는지 확인할 수 있다. 만약 반환값이 -1이라면 도형이 제대로 선택되지 않은 것이다.
16. 사용자가 도형을 옮긴 후 놓으면 이동된 위치로 다시 출력해준다.
17. Erase (flag = 6)
18. [Erase()](#Erase)로 클릭된 도형을 찾고 지운다.
19. 함수의 반환값이 있다면 리스트에 도형이 저장되었다는 뜻이다. total에서 true로 된 부분을 출력하고, 저장된 도형을 모두 출력한다.
20. Clear (flag = 7)
21. clearLcd()로 흰 화면을 지워주고 [deleteAll()](#Clear)로 리스트에 저장된 노드를 전부 삭제한다.
22. Fill (flag = 9)
23. Fill()로 도형의 내부를 칠한다.
    1. **결론**

사용자가 누른 부분에 따라 터치스크린에 그림을 그릴 수 있다.