2_나비효과 최종보고서

Project Title : 영상인식기술을 활용한 동작인식 시스템

Lecture Class	컴퓨터공학종합설계_01		
Team Member Students	Student ID No.	Name	Email
	2011112162	한승범	aronix@naver.com
	2012111659	나선엽	coinnip@gmail.com
	2015113455	김상연	sy0814k@gmail.com

영상인식기술을 활용한 동작인식 시스템

1. 프로젝트 설명

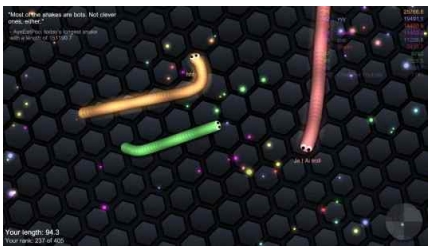
1.1 기존 프로젝트

큰 TV화면에 웹 카메라를 연결하고 사용자는 카메라 앞에 서서 게임을 시작할 수 있다. 웹 카메라가 촬영하는 영상을 서버로 보내주면 서버는 사용자의 모션을 분석하여 TV화면에 반영해준다.

<그림 1.1>

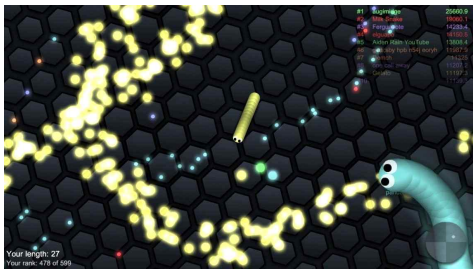
처음에 구상했던 프로젝트 예상 모습은 <그림 1.1>과 같았다. 사용자 여러 명이 카메라 앞에 서면 화면에서는 각각의 사용자가 각각의 나비에 연결이 된다. 사용자의 움직임에 따라 나비는 상하좌우로 움직일 수 있으며, 게임의 목표는 주변에서 날아오는 돌멩이나 새 등의 위험으로부터 움직이며 피하는 것이었다.

1.2 수정된 프로젝트



<그림 1.2.1>

담당 교수와 상담하면서 기존 프로젝트는 사용자의 흥미도가 많이 떨어질 수 있다는 조언을 받았다. 팀원과의 회의에서도 단순히 위험물을 피하는 것은 흥미도가 떨어질 수 있다고 생각이 들었기에 피하는 것에서 더 흥미있는 내용이 추가 되어야 한다고 생각하였다. 그래서 <그림 1.1> 의 게임같이 상대를 잡아 먹을 수도 있는 경쟁 시스템을 넣기로 하였다. <그림 1.1>은 slither 라는 게임으로 화면에 뿌려진 빛 알갱이를 먹으면 먹을수록 몸집이 커진다. 또 자신의 머리가 상대의 몸통에 부딪히면 죽는 게임이다. 죽은 대상은 <그림 1.2.2>처럼 수많은 빛 알갱이로 변하게 되기 때문에 상대를 가두는 등 전략적인 방법으로 상대를 공격하여 내 자신의 몸집을 더 키울 수도 있다.



<그림 1.2.2>

우리팀은 최종적으로 slither와 흡사한 게임을 만들기로 하였다. 그림<1.2.3>과 같이 사용자가 카메라 앞에 서 있으면 각각의 사용자에 해당하는 지렁이가 화면에 나타나고, 사용자의 동작에 따라 연결된 지렁이가 움직이는 방식이다.

<그림 1.2.3>

지렁이의 움직임은 사용자의 손바닥을 이용해서 할 계획이다. 사용자가 화면 앞에 손바닥을 내비치고 있으면 웹 카메라가 인식을 하여 그 이후부터는 사용자가 손을 상하좌우로 움직임에 따라서 지렁이가 움직일 계획이다. 화면 곳곳에 랜덤하게 나오는 빛을 먹으면 지렁이의 몸집이 점점 커지며 지렁이의 얼굴이 상대 지렁이의 몸통에 부딪히면 그 지렁이는 게임에서 탈락시킬 계획이다. 화면에 지렁이가 다 죽어 더 이상의 지렁이가 없으면 게임은 종료된다.

2. 아키텍처, 개발계획

2.1 요구 사항

Server는 Openpose에서 의미있는 분석 결과를 얻을 수 있는 해상도의 이미지 또는 영상을 처리할 수 있도록 고성능 GPU 및 CPU를 사용할 필요가 있다.

GPU는 benchmark에 따르면 11GB VRAM 환경에서 1280x720의 영상을 처리할 때 10fps가

나오고 있다. 직접 6GB VRAM 환경에서 320x320 해상도의 영상으로 테스트 한 결과 20fps가 나오는 것으로 보아 개발 환경을 위해서는 VRAM이 11GB 정도에 해당하는 GPU를 사용하면 될 것이다.

CPU는 벤치마크로 직접 제공된 정보는 없었으며, 위의 6GB GPU로 테스트 할 때 사용한 8core 3.4GHz clock CPU에서는 GPU 병목으로 크게 load가 걸리지 않았다. 그리고 다중 코어를 지원하기 때문에 CPU 선택에는 크게 제한을 받지 않을 것으로 예상된다.

2.2. Server 구성

Openpose가 Caffe에 최적화되어있으므로, Caffe 구동에 가장 적합한 Linux 환경으로 구성한다. 이를 만족하는 OS로 AWS의 Ubuntu SSD Volume Type을 사용한다.

Server에 필요한 사양은 AWS에서 제공하는 인스턴스중 EC2-p2.xlarge를 사용하면 될 것으로 예상된다. 최종적으로 시연을 할 때는 p2.8xlarge 인스턴스를 사용하여 Delay를 최소화 하기로 한다. 해당 인스턴스의 가격은 시간당 각각 1.465$, 11.72$이다.

서버에 Openpose를 설치한 뒤, UDP 방식으로 영상을 전송받아 영상을 처리할 수 있도록 한다. 영상을 실시간으로 전송받아 지연 없이 실시간으로 좌표를 돌려보내야 하기 때문에, UDP를 사용하기로 한다. 또한, 수신 받은 영상이 있더라도 이것이 Server의 영상 처리 속도 문제로 계속해서 지연이 된다면, Client 입장에서는 계속해서 이전의 영상에 대한 좌표 정보를 받게 되므로 크기가 제한된 Queue등을 이용하여 제 때 받지 못한 정보는 사용하지 않고 넘어가기로 한다.

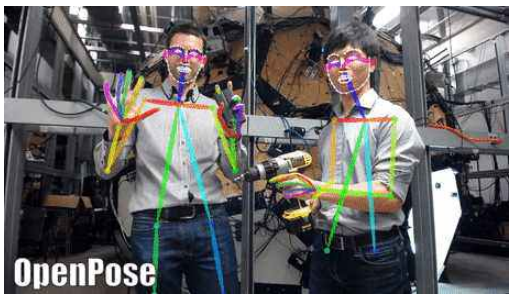
2.3. Client 구성

Server는 그래픽에 관련된 동작은 OpenCV를 이용하여 지정된 이미지를 영상위에 나타내는 정도의 동작밖에 없기 때문에, 사양에 제한받지 않는다. Client PC에는 영상을 촬영할 Webcam, 받아온 영상을 나타낼 Display를 사용한다. 영상은 UDP 방식으로 연결된 서버에 전송을 하게 된다. 게임 프로그램의 진행은 클라이언트 PC 내부에서 진행된다.

3. 사용할 오픈소스 : Openpose

3.1 오픈소스 설명

<그림 3.1>은 이번 프로젝트의 개발에서 필요한 핵심 오픈 소스다. Gines Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Hanbyul Joo, and Yaser Sheikh 에 의해 개발된 오픈 소스이고 자세한 내용은 하단의 링크에서 확인할 수 있다.



<그림 3.1>

이 오픈소스를 이용하면 단일 이미지 프레임에 대해서 인체 모형, 손, 안면 키패드(총 130개)를 공동으로 감지할 수 있다.

https://github.com/CMU-Perceptual-Computing-Lab/openpose/

3.2 오픈소스 활용 방안

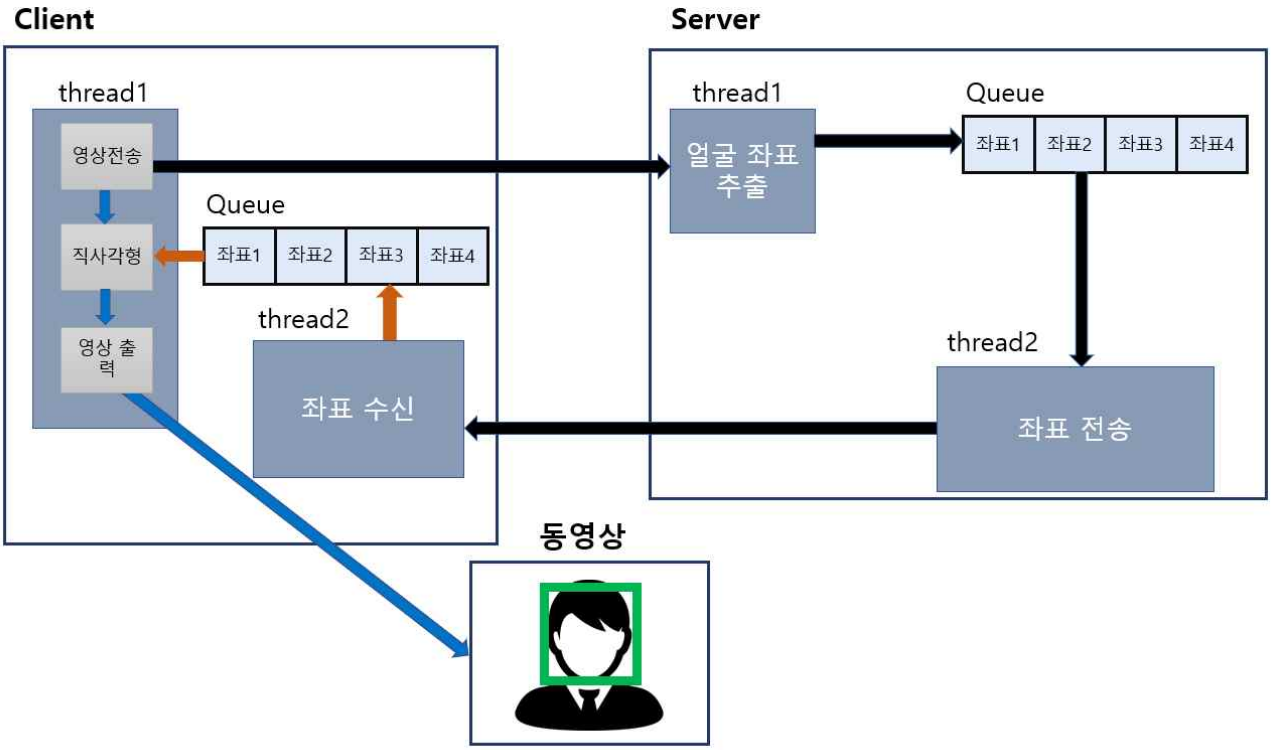
우리의 프로젝트에서 위의 오픈소스를 이용하는 가장 큰 이유는 사용자의 움직임을 인식하기 위해서이다. 우리는 각각의 사용자로부터 좌표를 얻어내어 사용자의 움직임을 인식하고, **이전 프레임과의 좌표의 차이**를 이용하여 의미 있는 행위로 인식할 계획이다. 가령 사용자의 손의 위치가 이전들의 프레임과 비교해봤을 때 왼쪽으로 이동하였다면 사용자의 손의 움직임이 왼쪽으로 이동했다고 예측하여 그에 맞는 동작을 반영하는 것이다.

3.3 오픈소스의 한계

이 오픈소스를 활용하면 특정 프레임에서의 사용자들의 위치를 알 수는 있지만 이전 프레임과 현재 프레임의 연관 관계는 알려주지 않기 때문에 중간에 사용자가 겹치거나 새로운 사용자가 추가될 경우 누가 누구인지를 오픈소스에선 알려줄 수 없다. 우리가 개발할 프로젝트에서는 이러한 부분도 해결해야하기 때문에 이전 프레임과 현재 프레임의 연관 관계를 결정해주는 알고리즘도 개발할 계획이다.

4. 진행 사항

4.1 멘토링 과제#1



<그림 4.1>

4.1.1 과제 목표

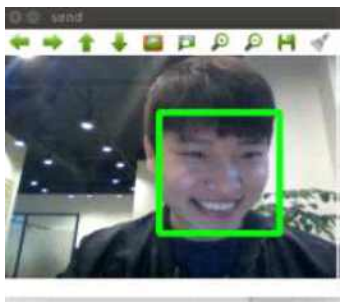
이번 과제의 목표는 최종 과제에 사용할 동영상 스트리밍의 기본을 익히고 실제로 오브젝트를 프레임에 그릴 수 있는 방법을 익히기 위해 서버와 클라이언트를 구축한다.

4.1.2 과제 구현

클라이언트의 웹캠을 이용하여 동영상을 촬영. **프레임을 실시간으로 서버에 전송**하여 서버 측에서 해당 프레임의 **얼굴 좌표 추출** 후 좌표 **큐에 저장**한다. 또 서버 thread2는 계속해서 큐를 검사하여 **큐에 값이 있으면 이 값을 클라이언트 측으로 전송**한다.

클라이언트 측에서는 **전송받은 좌표를 클라이언트 좌표 큐에 저장**한다. 클라이언트 thread1이 큐를 계속해서 검사하며 값이 있을 경우 이 **값에 해당하는 직사각형을 그려 출력**한다.

4.1.3 프로그램 구현 [<그림 4.1> 참고].



<그림 4.2>

서버와 클라이언트간의 원활한 전송 환경을 구축시키기 위해 클라이언트에서 프레임을 jpeg로 인코딩하여 서버에 전송하였다. 프레임을 전송하는 과정에서는 프레임 유실이 발생해도 프로그램이 다운되지 않는 **UDP 프로토콜을 사용**하였다.
그리고 서버에서 추출한 좌표를 클라이언트 측으로 전송하는 과정은 좌표값이 유실되면 안되므로 안정성이 더 높은 **TCP 프로토콜을 사용**하였다.
결과 화면은 <그림 4.2> 를 통해 확인할 수 있다.

4.2 멘토링 과제#2

4.2.1 과제 목표

우리가 최종적으로 사용하게 될 인물의 좌표 값을 추출하기 위해 오픈소스를 활용한다. 이 오픈소스를 활용하여 과제#1과 접목하여 객체를 화면에 그리고 이 객체를 움직일 수 있도록 한다.

4.2.2 진행 사항

현재 이 오픈소스를 Desktop환경에서 실행시켰고, 이를 라이브러리 형태로 우리가 이전에 만들었던 동영상 스트리밍 서버에 접목시킬 계획이다.

4.3 1차 프로젝트 개발

4.3.1 사전 환경 구성

Openpose를 이용한 Server 및 게임을 구동할 Client를 구축하기 위한 구성 과정이 필요하였다. Local에서 Openpose 설치 및 컴파일 방법을 직접 진행해보고, 테스트를 위한 AWS 서버를 구성하였다.

openpose를 활용한 이미지 처리를 진행하는 프로그램과 UDP 프로그램을 이용하여 Openpose Server – Client 환경을 구성하였다.

Openpose의 예시로 제공된 코드를 활용하여 Client 프로그램 개발을 위해 필요한 좌표만을 전송할 수 있도록 수정하였다.

위와같은 환경을 구성하여 효율적으로 분업하여 프로그램을 개발할 수 있도록 하였다.

4.3.2 게임 프로그램(클라이언트)

4.3.2.1 초기화

클라이언트 프로그램이 처음 실행되면 서버와 udp 및 tcp 연결을 수립한다. udp연결은 클라이언트가 서버로 웹캠 프레임을 전송하기 위한 것이고, tcp연결은 서버로부터 계산된 좌표를 받기 위한 것이다. 그 이후부터는 지렁이 객체 초기화, 먹이 생성등의 실제 게임 플레이를 위한 데이터를 초기화 해준다.

4.3.2.2 주요 흐름도

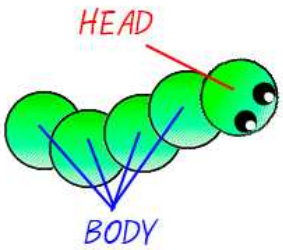
초기화가 끝나면 프로그램은 그림<4.3.2> 와 같은 로직을 반복하게 된다.

<그림 4.3.2>

웹캠으로 캡처한 그림을 서버로 전송한 후(이미지 전송), 서버로부터 계산된 좌표를 수신 받는다(좌표 수신), 그 이후 수신된 좌표를 통해 지렁이의 각도를 계산하여 지렁이의 움직임에 반영한다.(지렁이 이동), 움직인 지렁이가 화면상에 존재하는 먹이와 충돌했는지 체크하고 충돌했다면 처리한다.(먹이 충돌 처리), 그 이후 지렁이와 먹이의 그래픽을 화면에 갱신한다. (그래픽 갱신)

4.3.3 지렁이

4.3.3.1 지렁이 설계



<그림 4.3.2>

```
class Worms{
public:
        Worms();
        ~Worms();
        void move();
        void setTheta(float theta);
        void increaseBody();
        float getTheta();
        vector<Pt> getBody();

private:
```

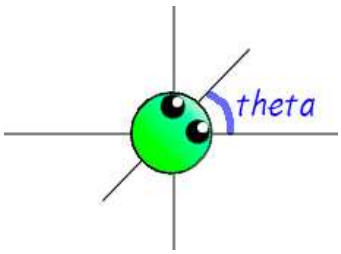
```
    float theta; //지렁이 현재 각도
    int velocity; //지렁이 속도
    vector<Pt> bodies;

};
```

지렁이는 크게 몸체와 각도, 속도를 가지게 설계하였다. 몸체(bodies)는 Point구조체를 담고 있는 vector로 선언되었으며 <그림 4.3.2>에서 보면 머리와 몸통이 있는데 bodies 0번째 요소가 지렁이의 머리, 나머지 부분(1번째 요소~마지막 요소)은 지렁이의 몸통으로 이용할 수 있게 설계하였다.

그 다음은 지렁이 머리의 각도 theta 이다. 이 theta는 지렁이가 움직일 때 방향을 결정하기 위해 사용할 변수이다.

마지막으로 지렁이의 속도 velocity 이다. 이 velocity 변수는 지렁이가 얼마만큼의 속도로 화면에서 이동하는 지에 대해 결정할 변수이다.

4.3.3.2 지렁이 이동 알고리즘



<그림 4.3.3>

지렁이의 움직임 이동 알고리즘은 지렁이 클래스의 move() 함수에서 구현되어 있다. 지렁이 움직 알고리즘의 핵심은 지렁이의 theta에 있다. 지렁이의 머리가 <그림4.3.3>처럼 theta의 방향을 보고 있을 때, 단위시간 후 지렁이의 머리좌표는 다음과 같다.

```
x += (velocity) * cos(theta);
y += (velocity) * sin(theta);
```

위에서 velocity는 지렁이가 가진 현재 속도이다. 위처럼 머리가 이동하고, 그 다음은 나머지 몸통들은 별다른 수식 계산 없이 바로 이전 앞의 몸통의 좌표로 이동하는 식으로 알고리즘을 구현하였다.

4.3.3.3 지렁이 먹이 및 크기 변화

지렁이는 화면상에 있는 먹이를 먹으며 몸의 크기를 키울 수 있다. 먹이는 간단한 Point 구조체의 vector 형태로 선언하였고, 화면 상에 랜덤한 좌표로 생성 되도록 구현하였다. 지렁이가 먹으면 지렁이 객체안의 increaseBody()메소드가 호출되며 지렁이의 몸이 증가한다.

4.3.4 이미지 처리

OpenCV에 사용자 이미지를 프레임에 붙이는 함수가 존재하지 않아, 그 함수를 사용자 정의 함수로 구현하였다. 함수 내부 구현은 먼저 이미지를 해당 각도로 회전시키기 위해 회전값을 2차원 매트릭스 형태로 얻는다. 그리고 이 회전값과 회전시킬 이미지(원본)을 Affine시켜 새로운 이미지(결과)를 얻어내었다.
그 후 이 회전된 이미지를 실제 프레임에 그려 넣기 위해 그려 넣을 영역을 프레임에서 확보해야한다. 확보를 하면 붙일 이미지를 Masking하여 실제 붙일 이미지의 데이터만 가려내는 작업을 거친다. 그 후 가려낸 이미지를 확보된 영역에 붙여 넣는 식으로 함수를 구성하였다.

4.3.4.1 이미지 오버헤드

이미지를 붙이는 함수를 구현하였으나, 붙일 이미지의 용량이 작으면 문제가 덜 하지만 용량이 클 때 이러한 이미지를 여러 개를 빠르게 그리다보니 과부하 현상으로 게임의 속도가 느려지는 현상이 발생하였다. 이는 기존의 OpenCV 라이브러리에 있던 원 그리기 함수에도 똑같은 문제가 발생함을 확인할 수 있었고, 따라서 구현된 함수에서만의 문제가 아님을 확인할 수 있었다.

4.3.4.2 오버헤드에 대한 해결책

오버헤드는 프로그램의 성능에 중대에 영향을 미치므로 최소화를 해야만 한다. 따라서 다음과 같은 2가지 해결책을 도출해내었다.

1. 지렁이의 몸통을 좌표마다 그리는 것이 아닌 좌표값에 해당하는 하나의 다각형을 그림으로써 오버헤드를 줄이는 방법.

2. 지렁이의 몸통을 계속해서 모두 그리지 않고 띄엄띄엄 간격을 두어 그리는 방법.

우선 현재 프로그램에 적용하기 편한 2번을 적용하여 오버헤드 문제를 해결하였고, 추후에 1번을 구현하거나 새로운 해결을 도출해 낼 계획이다.

5. 향후 개발 계획

5.1 서버 기본 환경

개발 및 개발 테스트 과정에서는 Amazon Web Servic(AWS)에서 제공하는 Amazon EC2. P2.xlarge 인스턴스를 이용한다. 실제 시연을 위한 테스트 때는 P2.8xlarge를 사용하여 Delay를 최소화 하는 방향으로 개발한다.

Ubuntu OS에 caffe, CUDA, cudnn 등을 설치하여 Openpose를 구동할 수 있도록 구성한다. p2.xlarge의 요금은 시간당 1.465$이며, p2.8xlarge의 요금은 11.72$이다. 더 저렴한 가격에 나은 환경을 제공하는 Naver 클라우드 플랫폼 또한 고려해본다.

5.2 서버-클라이언트 데이터 송수신 환경

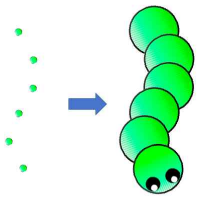
UDP 방식으로 영상을 받는데에 있어서 손실된 데이터를 받았을때의 처리가 확실하게 정의되어있지 않다. 이를 개선하는 방법을 구상하고 적용하도록 한다.

5.3 다수의 유저에 대한 처리

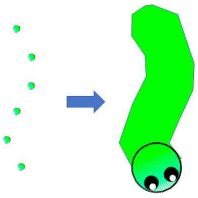
배경에 제약받지 않는 Openpose의 장점을 살리기 위하여 다수의 사용자가 참여할 수 있도록 코드를 개선한다. 이를 위해 Server에서는 일정 크기 이상의 사용자만 보낼 수 있도록 개선해야한다. Openpose는 분석한 사진 상의 사람이 이전 분석 결과와 비교하여 몇 번째의 사람인지 분석할 수 있는 기능이 없다. 따라서 이 기능은 Client에서 해결하도록 한다. 지렁이마다 사용자의 현재 좌표를 지정하여, 다음 수신 받은 좌표와 비교하여 해당 지렁이를 조종하던 사용자를 인식하도록 한다.

5.4 그래픽 처리

OpenCV를 이용하여 지렁이를 출력할 때, 지렁이의 몸통이 늘어날수록, 사용자가 늘어날수록 상당한 속도의 저하가 일어난다. 이는 상당히 많은 횟수의 draw를 수행하기 때문이다.



<그림5.1>

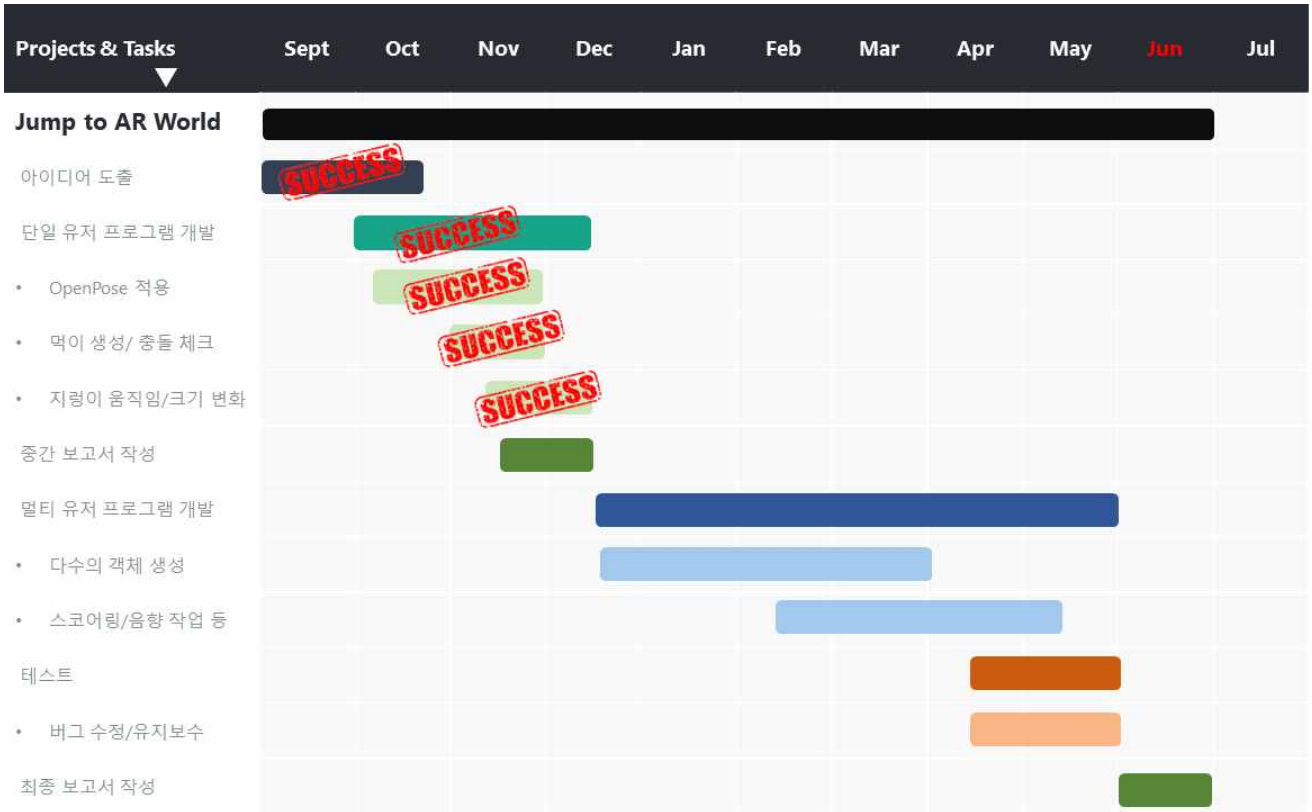


<그림5.2>

<그림5.1>은 현재의 구현 방식을 나타낸 것이며, 왼쪽의 각 점 좌표는 몸통 및 머리를 의미한다. 각 좌표마다 머리와 몸통 그림을 draw 한다. 이러한 방식의 문제점을 해결하기 위해 <그림5.2>에 소개된 것처럼 지렁이의 몸통의 좌표들을 중심으로 하여 다각형 그리기를 적용하여 draw 횟수를 줄이는 방법을 고려한다.

또는 Unity를 활용하는 방법을 고려한다.

6. 타임라인



6.1 타임라인 상세 설명

6.1.1 아이디어 도출

우리 나비효과팀은 동작인식을 이용하여 영상처리 시스템을 구축하기 위해 첫 주제로 '새들의 공격을 피하여 끝까지 살아남은 나비'로 정하였으나 지속적인 회의를 통해 '서로 서로 경쟁하는 지렁이 게임'으로 변경하였다.

6.1.2 단일 유저 프로그램 개발

6.1.2.1 OpenPose 적용

기존 UDP서버에 서브루틴을 실행하는 wrapper 함수를 스레드를 사용하여 호출하고 클라이언트에게 .jpg로 인코딩된 프레임을 처리하여 좌표 값을 다시 클라이언트에게 TCP 통신으로 보내도록 하였다.

6.1.2.2 먹이 생성, 충돌체크

개수가 정해진 먹이를 프레임에 랜덤하게 분산시키고 핸들러 함수를 통해 현재 먹이 좌표와 지렁이 좌표가 일정 범위 내에 들어와있는지 검사하여 충돌 여부를 판정하였다.

6.1.2.3 지렁이 움직임/크기 변화

지렁이의 움직임은 사용자의 몸통과 오른쪽 손의 좌표값으로 각도를 계산에 해당 각도로 지렁이를 움직이도록 하였다.

지렁이의 크기변화는 지렁이가 먹이를 먹을 때마다 지렁이 끝 쪽에 새로운 몸통을 이어 붙임으로써 지렁이의 크기를 변화시켰다.

이상으로 계획했던 일정을 모두 완수하였고 이것으로 나비효과팀의 최종보고서를 마치겠습니다.

<끝>