[디지털이미징종합설계]

최종 보고서

-Thumbs up-

TAYO
20125203 김현지
20121651 박서영
20132390 오성기

목 차

1. 서론

현재 정확성과 속도 향상을 위하여 학습데이터 생성 및 분류를 기반으로 다양한 모션인식 연구가 활발하게 진행되고 있다. 모션 인식 기술은 어떤 특정한 물체의 움직임이나 위치를 인식하는 각종 센서를 이용한 기술을 통칭한다. 이 기술에서 가장 중요한 것은 바로 '센서'이다.

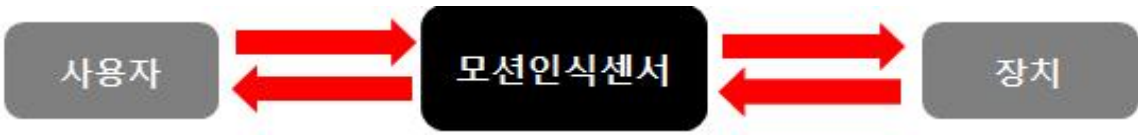


그림 1. 모션인식 기술

센서는 사용자의 신체 움직임을 인식하고, 그림 1과 같이 이를 장치와 상호 작용하는 중간 연결 고리 기능을 해준다.

현재, 모션 인식 기술은 크게 '컨트롤러 기반'과 '카메라 기반'방식으로 나눌 수 있다. 컨트롤러 기반의 모션 인식 기술에는 센서가 많이 사용된다. 특히, 가속도를 측정하여 물체의 방향 변화를 감지하는 자이로스코프와 지표면을 기준으로 가속도와 기울기를 이용하여 물체의 움직임을 측정하는 가속도 센서가 가장 많이 사용되고 있다. 카메라 기반의 모션 인식 기술에는 키넥트와 립모션이 가장 많이 사용된다.

(a) (b)

그림 2. 카메라 방식의 모션 인식 기술 (a) 키넥트, (b) 립모션

그림 2(a)의 키넥트는 적외선 카메라를 이용한 카메라 기반 방식으로 적외선을 송출하는 적외선 프로젝터와 반사되어 오는 적외선을 인식하는 깊이 인식 적외선 카메라로 구성되어 있다. 이 센서들은 20여개 이상의 관절을 인식하여 X, Y, Z 3차원의 신체 움직임을 감지할 수 있다. 그림 2(b)인 립모션은 기본적인 원리는 키넥트와 비슷하며, 마우스가 없어도 모니터에 손을 대지 않아도 화면 조종이 가능하다. 또한 그림을 그릴 수 있고, 게임도 할 수 있다.

본 과제에서는 카메라 기반의 방식인 립모션을 사용하여 모션 인식 연구에 있어서 가장 중요한 화두로 떠오르고 있는 모션 학습 트레이닝 데이터셋 생성에 있어서 정확도와 속도 향상을 위한 모션 학습 트레이닝 데이터셋 생성 방법을 제안하고, 적용한 VR게임 콘텐츠를 제작하였다.

2. 배경이론

2.1 립모션으로부터 데이터 취득

립모션은 시리얼 통신을 통해 실시간으로 손의 좌표 데이터를 출력한다. 촬영 환경에 따라 초다 20~200 frames의 데이터를 전송한다. 전송되는 초기 데이터는 각 joint들의 좌표이다. 립모션 개발 라이브러리는 관절 좌표를 통하여 팔의 위치 및 방향, 손목의 위치, 손바닥의 길이와 위치 또는 방향, 각 손가락의 길이 및 위치와 방향을 알 수 있다.

```
const Frame frame = controller.frame( );

HandList hands = frame.hands( );
for (HandList::const_iterator hl = hands.begin( ); hl != hands.end( ); ++hl) {
    const Hand hand = *hl;
```

그림 3. frame, hand 클래스

그림 3에서와 같이 frame 클래스는 hand 클래스의 상위 개념이다. frame 클래스는 0~2개(왼손, 오른손)의 hand 클래스를 가지고 있다. hand 클래스는 손바닥 위치와 방향 정보를 가지고 있으며, 손바닥의 위치는 해당 손 좌표계의 중심점이 된다.

```
Arm arm = hand.arm( );

const FingerList fingers = hand.fingers( );
for (FingerList::const_iterator fl = fingers.begin( ); fl != fingers.end( ); ++fl) {
    const Finger finger = *fl;
```

그림 4. arm, finger 클래스

그림 4의 arm, finger 클래스는 hand 클래스에 종속되어 있는 하위 개념이다. arm 클래스가 finger 클래스의 상위호환에 속한다. arm 클래스는 손목의 위치, 팔꿈치의 위치, 팔의 방향 정보를 가지고 있다.

```
// Get finger bones
for ( int b = 0; b < 4; ++b) {
  Bone::Type boneType = static_cast<Bone::Type>(b);
  Bone bone = finger.bone(boneType);
  std::cout << std::string(6, ' ') <<  boneNames[boneType]
            << " bone, start: " << bone.prevJoint( )
            << ", end: " << bone.nextJoint( )
            << ", direction: " << bone.direction( ) << std::endl;
}
```

그림 5. bone 클래스

그림 5는 손가락 정보의 핵심을 담고 있는 bone 클래스이다. 각 finger마다 4개의 bone 클래스를 가지고 있으며 (단, 엄지손가락은 3개) 총 19개의 bone 클래스가 존재한다. 한 손가락 내에서 서로 다른 두 개의 bone 클래스는 연결된 경우 관절 좌표 하나를 공유한다. 즉, bone 클래스를 통해서만 손가락 관절 좌표에 직접적으로 접근할 수 있다. 관절 좌표에 접근하는 방법은 Bone[x].nextJoint()(끝관절) 또는 Bone[x].prev.Joint(시작관절) 함수를 이용하며 3차원 벡터값을 출력한다. 이 값은 Leap::Vector형 변수로 저장된다.

2.2 실시간 통신

립모션은 데이터를 전송해주는 server가 되고, 컴퓨터는 받아들이는 client역할을 한다. 병렬실행 시 무한루프로 실행되는 callback함수를 통해서 frame이 전송될 때마다 실행되어 정보를 처리한다.

3. 제안방법

본 과제의 목표는 손가락 모션을 이용한 핑거 보드 게임을 주제로 하여 보드손가락 모션 학습 트레이닝 데이터셋 생성에 있어서 첫 번째로, 차원 내림을 통한 좌표 재구성과 동작 구분을 위한 변수 정의한다. 두 번째로, 빠른 처리 속도를 위한 학습 데이터의 차원을 줄이기 위한 학습 데이터를 가공한다. 그리고 마지막으로 생성된 데이터셋을 이용한 VR게임 콘텐츠 제작을 목표로 한다.

3.1 차원 내림을 통한 좌표 재구성과 동작 구분을 위한 변수 정의

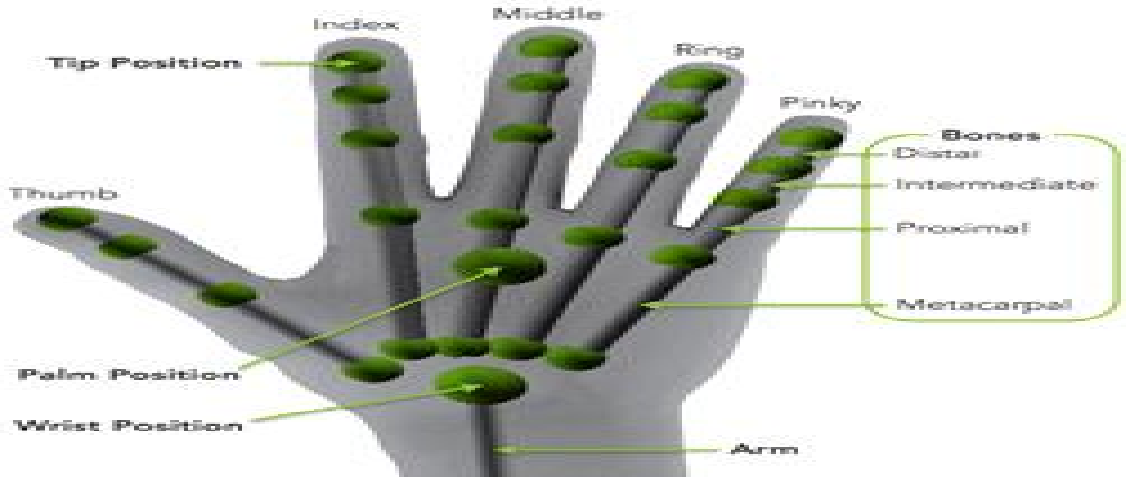


그림 6. 기본 손가락 관절

손은 그림 6에서와 같이 외관적으로 엄지부터 약지까지 Thumb, Index, Middle, Ring, Pinky로 불린다. 또한 각각 마디는 Metacarpal, Proximal, Middle, Distal로 이루어져 있고 서로를 연결해주는 joint를 가지고 있다. 립모션은 내장된 3개의 적외선 카메라를 통해 매 frame마다 아래 표시된 요소들의 3차원 좌표 값을 출력하며(초 당 20 ~ 200 frames), 개발자는 해당 데이터를 근거로 프로그램에서 손의 형태를 재구성 할 수 있다.

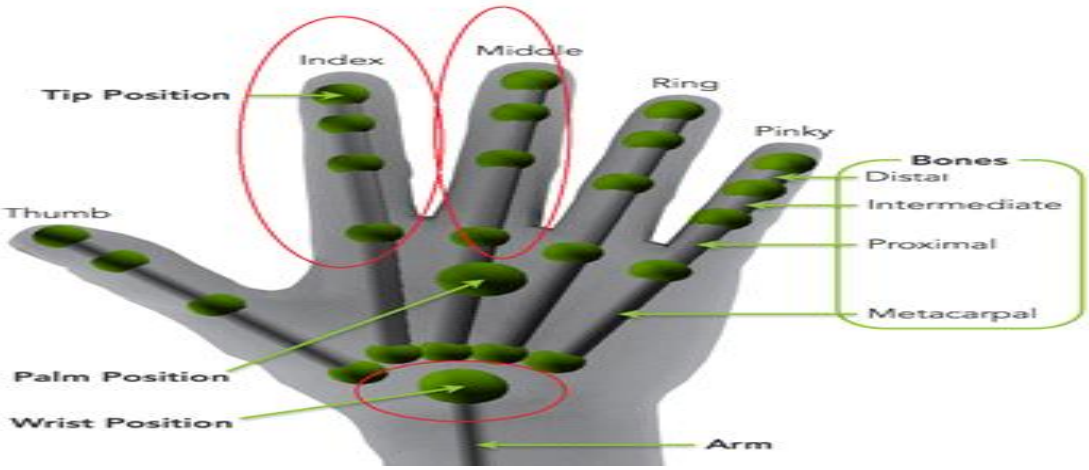


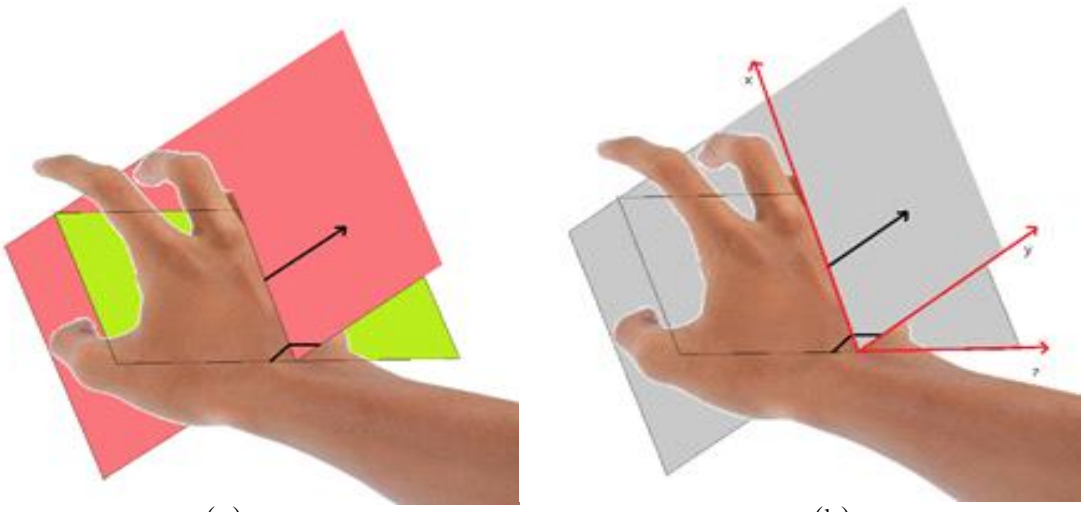
그림 7. 본 과제에서 이용하는 9개의 joints

본 과제에서는 아래에 그림 4에서 표시된 Index, Middle, Wrist를 구성하고 있는 9개

의 joints 데이터를 이용하여 학습 파라미터를 생성한다.

그 후, 각도 변수 정의와 Index, Middle의 상대 위치를 통한 동작 구분을 위한 변수를 정의하며 마지막으로 손의 절대 위치를 정의한다.

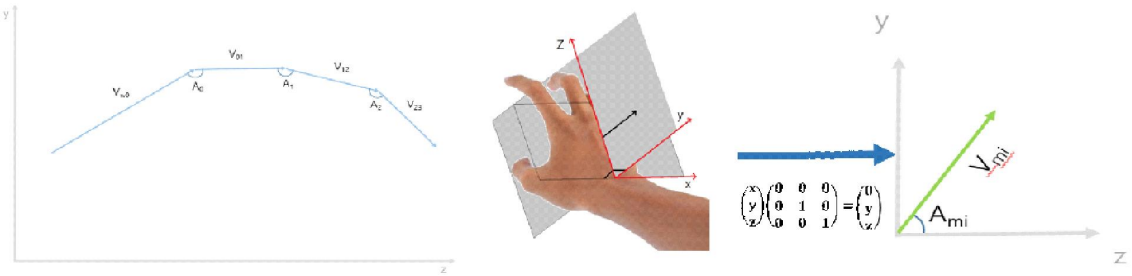
먼저, 각도 변수 정의는 meta carpals와 proximal을 연결하는 joint는 3차원 운동을 하며 그 외의 joint들은 한 평면상에서 2차원 운동을 한다. 여기서 설명하는 평면은 손바닥의 법선 벡터로 이루어지는 평면에 직교하는 평면들 중 법선 벡터를 포함하는 평면으로 정의한다.



(a) (b)

그림 8. 좌표계 재구성

그림 8(a)에서 녹색 평면은 손바닥의 법선 벡터로 이루어진 평면이며, 적색 평면이 앞에서 정의한 평면이다. Index와 Middle의 joint들은 해당 평면으로의 차원 내림을 통해 좌표를 재구성한다. 그림 8(b)는 해당 평면을 이용해 재구성한 좌표이다.

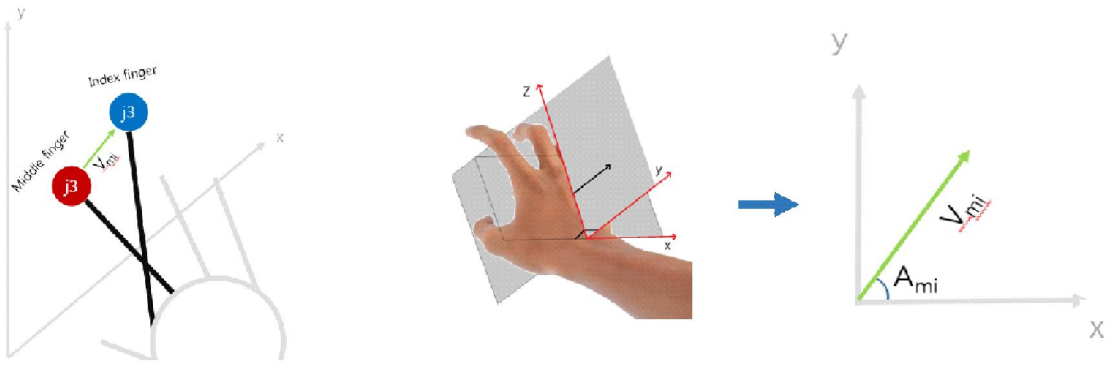


(a) (b)

그림 9. (a) 손가락 1개 기준 벡터와 각도 변수, (b) 좌표계 재구성

그림 9(a)는 손가락 1개 기준 벡터와 각도 변수들을 정의한 것이다. 손가락 1개 당 생기는 각도 변수는 3개이며, 총 6개의 각도변수가 발생한다. 해당 변수들은 서로 의존적이며 그림 9(b)와 같이 X, Y 성분들만을 가지는 2차원 변수이다.

두 번째로는, Index와 Middle의 상대위치를 정의한다. 본 과제에서는 핑거보드의 모션을 정의하여 트레이닝 셋을 생성한다. 따라서 Index와 Middle의 상대 위치를 통하여 보드의 좌, 우의 방향 전환 동작을 정의한다.



(a) (b)

그림 10. (a)벡터 변수 변환, (b) 좌표계 재구성

그러므로 보드의 방향 전환 동작을 감지하기 위하여 Index와 Middle의 손끝의 위치 차이를 $\overrightarrow{V_{mi}}$라는 벡터 변수로 변환한다. 두 손가락 끝 사이의 방향성을 변수로 지정하며, 앞에서 정의하였던 평면으로의 차원 내림을 통하여 2차원 벡터 변수로 지정한다. 해당 변수는 상황에서 오직 1개만 존재하는 독립 변수이다.

마지막으로, 손의 절대위치 정의는 Wrist의 좌표 그대로 학습되며, 독립 변수이다.

3.2 빠른 처리 속도를 위한 학습 데이터 차원을 줄이기 위한 학습 데이터 가공

학습 데이터의 차원을 줄이기 위하여 진행되는 과정이다. 학습 데이터의 차원을 낮출수록 대략적인 상황에 대해서 빠른 속도로 처리가 가능하다는 장점을 가지고 있다. 하지만 단점 역시도 존재한다. 세부 개념에 대한 구분하는 정도가 상대적으로 떨어질 수 있다.

학습 데이터 가공은 각도변수, Index, Middle의 상대위치 변수, 손 절대위치 변수 총 3가지로 구분하였다.

먼저, 각도변수 이다. 각도변수는 움직임에 따라 증가하는 방향이 같은 의존적 변수들이다. 의존적 변수들을 통합하여 하나의 독립적인 데이터로 가공한다. 본

$$\text{param1} = \sum A_k\,(Index\ \ finger)$$

$$\text{param2} = \sum A_k\,(Middle\ \ finger)$$

과제에서는 손가락 별 각도변수들의 총 합을 독립변수로 지정했다.$(A_0 + A_1 + A_2)$

두 번째로는, Index, Middle 상대위치 변수이다. 벡터 변수는 두 점의 좌표를 통하여 정의 된다. 이 경우 한 개념을 위하여 4개의 데이터가 입력되어야 한다는 것을 의미한다. 이것은 2차원 평면에서 한 축과 어느 각도로 이루어지는 정규화 벡터는 유일하다는 성질을 이용하여 하나의 각도변수로 가공한다.

$$\text{param3} = angle\{\overrightarrow{(p_{mx} - p_{ix},\ p_{my} - p_{iy})}\} = angle(\overrightarrow{V_{mi}}) = \cos^{-1}(\overrightarrow{V_{mi}}, \vec{z})$$

마지막으로, 손의 절대위치 변수이다. 바닥까지의 최단 거리는 실제 좌표계 Y축

으로의 정사영 벡터 길이로 정해진다.

$$\text{param4} = length(w, f) = length(proj_{\overrightarrow{y_{real}}}\overrightarrow{w - f})$$

손의 수평 이동량은 Wrist의 기준점으로부터의 거리이다.

$$\text{param5} = length(w, p_0)$$

3.3 분류기 제작

본 과제에서는 동작인식을 1초에 30 frames 동안의 데이터의 시간적 흐름에 따른 변화를 통하여 진행된다. 각 frame에 대한 데이터는 크기 30의 큐에 순차적으로 입출력 되며 분류기는 처리 시점 큐에 저장되어 있는 데이터들을 통하여 상황을 판단한다. 30frame 동안의 파라미터별 최대 변화량을 측정하며, 1차원적 방향성을 가진다. 측정 기준은 파라미터별 값이 최소인 경우와 최대인 경우를 기준으로 그 차이를 통해 변화량을 측정한다.

```
void classfier(params param, int fps){//param : 현재 프레임에 대한 파라미터 클래스 , fps : 초당 프레임 수
    const vector<string> sets = { "stable","run","jump","stop" };// 동작 set들의 이름들
    const int qsize = 30;//큐의 크기
    static deque<Mat> frames;//frames : 큐 형태의 행렬 변수, 현재 프레임 기준 과거 30프레임 동안의 정보를 담고 있다.
    static vector<motionsets> motions;//motions : 동작 set들의 클래스

    if (motions.empty())//동작 set들의 데이터가 비어있을 경우
        for (int i = 0; i < sets.size(); i++){
            motions.push_back(motionsets(sets[i]));//sets[i]에 해당하는 이름을 가진 동작 set들의 데이터를 불러온다.
        }

    if (frames.size() < qsize)//큐의 크기가 30보다 작을 경우
    {
        frames.push_back(param.cvtMat());//pop명령은 내리지 않고 push만을 시행한다.
    }
    else {
        frames.push_back(param.cvtMat());
        frames.pop_front();
        //큐의 30보다 같거나 클 경우 가장 오래된 프레임에 대한 정보를 제거하고 현재 프레임의 정보를 추가한다.


        Mat a_set;
        for (int i = 0; i < frames.size(); i++){
            a_set.push_back(frames[i]);
        }
        //30개 프레임들의 파라미터들을 a_set으로 모은다 a_set은 5 x 31 크기의 행렬이 된다.
        Mat sorted;
        cv::sort(a_set, sorted, CV_SORT_EVERY_COLUMN + CV_SORT_DESCENDING);
        //최소 최대값을 알기 위해 내림차순으로 정리, 결과는 sorted에 입력됨
        Mat gradient = sorted.row(0) - sorted.row(sorted.rows - 1);
        //최상위값(최댓값) - 최하위값(최소값)을 현재에서의 변화량 벡터로 지정
        //파라미터 5개의 변화량 정보를 담은 1 x 5 행렬
        Mat dists, idxes;

        for (int i = 0; i<motions.size(); i++){
            dists.push_back(motions[i].calcdist(gradient));
        }//모션 갯수만큼, 해당 동작 set의 중심점과 현재 변화량 벡터와의 차이를 구한다.

        cv::sortIdx(dists, idxes, CV_SORT_EVERY_COLUMN + CV_SORT_ASCENDING);
        //차이가 가장 작은 동작 set의 인덱스를 얻기 위해 오름차순으로 정렬

        //첫 번째에 있는 값이 분류기가 판단한 최 근거리 집합으로, 해당 현재 동작으로 판단한다.
        int itsclass = idxes.at<int>(0, 0);
        //cout << "this motion is " << sets[itsclass] << endl;
        frameshow(itsclass);//판단한 동작의 애니메이션 실행
    }
}
```

그림 11. 분류기 실행 코드

그림 11에서 확인할 수 있듯 분류기는 frame 입력에 따른 callback함수로써, 크기 30의 행렬의 큐 형태로 파라미터 정보를 저장하며, 학습 중이 아니면서 프레

임 정보가 입력될 때 마다 실행된다.

본 과제에서의 한 동작에 대한 분류기의 예를 들어보자면 발구름 동작의 경우 param2의 의존도가 높은 경우이다. 동작에 걸리는 시간과 각도 변화량을 통하여 속도를 측정한다.

그림 12. 발구름 동작의 경우 분류기

이와 같이 변수가 동작에 크게 영향을 주는 상황은 점프의 경우 Wrist의 높이 변화량 (param4), 브레이크의 경우 Wrist의 수평위치 변화량(param5), 좌 우 핸들링은 param3의 변화량이다.

3.4 학습기 제작

학습기 제작은 분류기와 동일한 알고리즘을 사용하여 변화량 값들 31개를 모아 파일로 저장하는 역할을 수행한다. 학습기의 측정 기준으로는 Mean값과 Covariance값을 계산하여 기준을 정하며, 동작별 5개의 파라미터를 학습시켜 제작하였다.

```
bool trainer(params param, std::vector<Mat> &output, int fps, bool &trainbegin, string fname) {
    const int training_sets = 30;

    const int qsize = fps*3;//3초 동안의 데이터를 학습한다.
    static bool istraining = false;//학습 중인지 상태를 표시하는 변수
    static Mat gradients;//변화량값들을 저장하는 행렬
    static int iter_num = 0;//학습 반복 횟수
    static Mat frames;

    if (!istraining&&trainbegin) {//학습 명령이 떨어졌을 때
        istraining = true;//학습 중 변수를 참값으로 설정
        trainbegin = false;//명령 변수를 원래대로 되돌린다
        std::cout << "training begin" << endl;
    }

    if (istraining) {//학습 중 변수가 참값일 때만 실행
        if (frames.rows < qsize)
        {
            frames.push_back(param.cvtMat());
        }
        else {
            cout << "number : " << iter_num << ", training end" << endl;
            Mat sorted;
            cv::sort(frames, sorted, CV_SORT_EVERY_COLUMN + CV_SORT_DESCENDING);
            Mat gradient = sorted.row(0) - sorted.row(sorted.rows - 1);

            gradients.push_back(gradient);//변화량값 저장

            //현재 변화량 출력
            std::cout << gradient;
            std::cout << endl;
            iter_num++;
            frames = Mat();//여기중단점
        }
        if (iter_num > training_sets)//31번 학습했을 경우
        {
            Mat covmat, meanmat;//공분산, 평균 행렬
            calcCovarMatrix(gradients, covmat, meanmat, CV_COVAR_NORMAL | CV_COVAR_ROWS);
            //변화량 값들을 이용하여 공분산과 평균행렬값 계산

            string str = "trainingsets/" + fname + ".xml";//파일 이름 결정

            FileStorage file(str, FileStorage::WRITE);
            if (!file.isOpened())//에러 시 출력
                std::cout << "create file first" << endl;
            else {
                file << "covmat" << covmat;//공분산 저장
                file << "mean" << meanmat;//평균 저장
                file << "trainingsets" << gradients;//이용한 학습 데이터들을 저장
                std::cout << "train end. it saved to " << str;

                istraining = false;//학습 중지상태로 전환
                iter_num = 0;//학습 반복변수 초기화
                gradients = Mat();//변화량 저장 변수 초기화
            }
        }
        return true;//학습 중일 때 true값 리턴, 학습 중임을 출력
    }
    return false;//학습 중이 아닐 때 false값 리턴, 학습 중이 아님을 출력
}
```

그림 13. 학습기 제작 부분 소스코드

4. 실험결과

사용자와의 인터렉션을 통한 학습 트레이닝 셋 취득 후 학습오차 데이터 취득하였다.

	G1	G2	G3	G4	G5	G6
G1	0.984	0.016				
G2	0.032	0.968				
G3		0.048	0.919			0.032
G4	0.048	0.016		0.935		
G5					0.984	0.016
G6			0.097			0.903

그림 14. 학습오차 표

그림 14는 본 과제에서 립모션을 통하여 모션인식 학습을 통한 학습오차 표이다. 평균 학습오차는 0.051이며 인식률은 94.9%의 결과를 보였다.

하지만 G3(점프)와 G6(멈춤)간의 간섭이 발생함을 알 수 있어 이 두 가지의 동작을 구분할 파라미터가 필요하다.

취득한 학습 트레이닝 셋을 이용하여 제작한 VR게임 컨텐츠를 통한 실험결과이다.

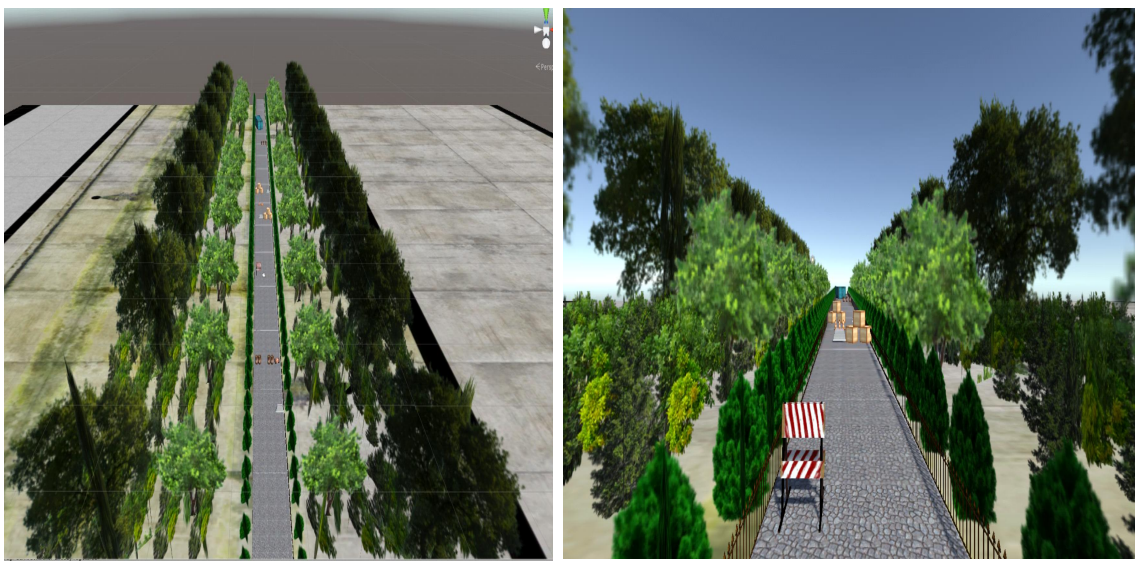


그림 15. 게임 map

그림 15에서와 같이 게임의 map은 직진형 트랙으로 제작하였다. 중간 중간 핑거보드의 기술을 이용할 수 있는 장애물들을 설치하였다. 장애물들은 Animator controller에서 skate state(기본 동작)을 진행하고 map에서 보이는 barrier tag가 되어 있는 object들과 player가 만날 경우 (부딪힐 경우) 게임이 끝난다.

그림 16. 시작 Idle 상태

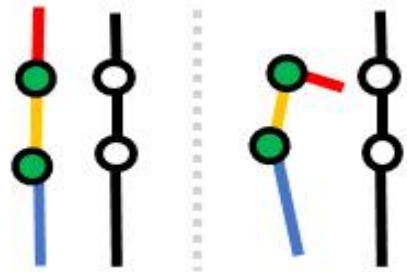


그림 17. 부스터

그림 18. 직진

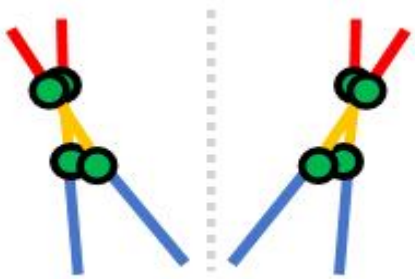


그림 19. 좌/우 방향전환

그림 20. 점프

그림 17과 같은 부스터의 경우 미리 학습시켜 놓은 트레이닝 셋과 동일한 모션이 인식될 경우 부스터로 인식되며 player의 3D vector의 속도를 증가시켜 player의 진행속도를 가속화 시킨다. 그림 18과 같은 직진은 게임 특성상 별도의 모션을 취하지 않아도 기본으로 설정한 기본속도로 player가 skating하는 것이 default값이다. 그림 19와 같이 방향전환으로 인식하면 input controller에서 좌/우의 파라미터를 true로 전환한다. default값인 skating state의 상태를 유지하며 movedirection의 Y축과 Z축의 상태는 그대로 유지하면서, X좌표의 진행에 변화값을 준다. 그림 20의 경우 점프동작이다. 검지와 중지 손가락이 바닥으로부터 일정 높이 이상 올라가면 점프라고 인식하며 특정 파라미터 값을 갖는다. 따라서 이 파라미터가 true가 될 때 skating state에서 점프 애니메이션으로 변환되도록 하는 구조를 생성한다.

5. 결론

본 과제에서는 절대좌표를 대신하여 palmplane을 도입함으로써 모션 트레이닝셋의 정확도를 향상시켰다. 하지만, 손가락만으로 표현할 수 있는 동작이 한정적이라는 한계점을 가지고 있다. 따라서, 추후 연구에서는 모든 손가락을 이용하여 모션을 학습시킨다면 정확도가 훨씬 향상되는 결과를 얻을 수 있을 것으로 기대한다.

5. 역할 분담

김현지	모션트레이닝 셋 생성 데이터 취득 Unity map 설계 Unity 게임 컨텐츠 제작 C# 스크립트 작성 및 에셋과의 연동 창의 ICT 캡스톤 디자인 발표
박서영	모션트레이닝 셋 생성 데이터 취득 Unity 게임 컨텐츠 제작 UDP 소켓 통신 패킷을 이요한 C++프로젝트와 Unity연동 창의 ICT 캡스톤 디자인 PPT작성 디지털이미징종합설계 최종보고서 작성
오성기	Opencv C++을 이용한 학습기 제작 Opencv C++을 이용한 분류기 제작 모션트레이닝 셋 생성 데이터 취득 디지털이미징종합설계 최종 발표

7. 참고문헌

1) Kinect 기반 손 모양 인식을 위한 손 영역 검출에 관한 연구, 박학훈, ISSN 2287-9137

2) Leap Motion을 활용한 학습 환경에서의 제스처 인식 인터페이스, 구본창, 성균관대학교

3) 립모션을 이용한 유니티 게임 개발, 강기태, 한국정보과학회