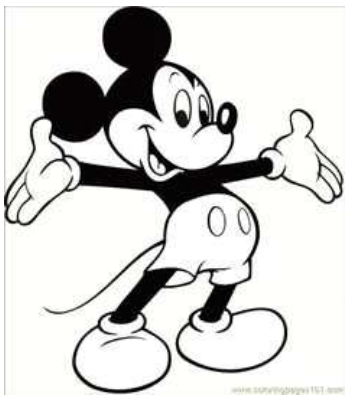
1. Multi-marker tracking

1-1.실제 사용한 마커



1-2.실제 마커사이의 거리 : 11-12cm

1-3. Homography로부터 카메라 좌표계에서 마커좌표계로의 변환 행렬 추정과정

1단계 :

```
cv::Mat T1, T2;
bool t1 = false;
bool t2 = false;
```

먼저 각각의 마커에대한 T행렬을 저장할 변수 선언.

2단계 :

```
///< 호모그래피로부터 마커에서 카메라 좌표로의 변환 행렬 추정
if (calculatePoseFromH(H, R, T)) {
    R.copyTo(E1.rowRange(0, 3).colRange(0, 3));
    T.copyTo(E1.rowRange(0, 3).col(3));

    static double changeCoordArray[4][4] = { { 1, 0, 0, 0 },{ 0, -1, 0, 0 },{ 0, 0, -1, 0 },{ 0, 0, 0, 1 } };
    static cv::Mat changeCoord(4, 4, CV_64FC1, changeCoordArray);

    E1 = changeCoord * E1;
    //첫번째 마커에 대한 T벡터set
    std::cout << "T1:" << std::endl;
    t1 = true;
    T.copyTo(T1);
    std::cout << T1.rows << " " << T1.cols << std::endl;
    std::cout << T1.at<double>(0) << " " << T1.at<double>(1) << " " << T1.at<double>(2) << std::endl;
    //gMatFlipImage.at<cv::Vec3b>(i, j)
```

첫 번째 마커에 대하여 호모그래피로부터 마커에서 카메라 좌표로의 변환행렬을 구하여 T1 matrix에 복사한다.

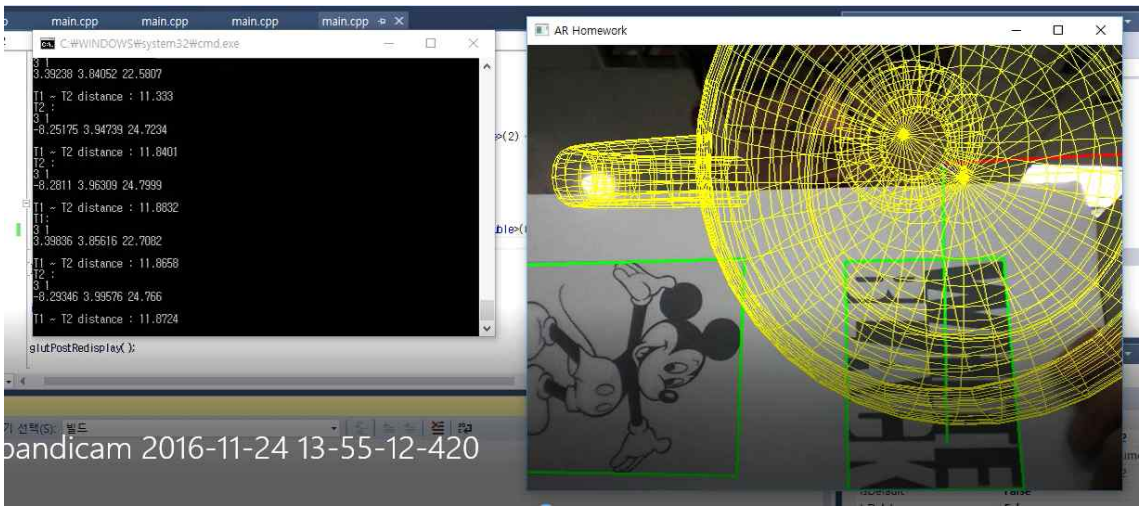
```
///< 호모그래피로부터 마커에서 카메라 좌표로의 변환 행렬 추정
if (calculatePoseFromH(H, R, T)) {
    R.copyTo(E2.rowRange(0, 3).colRange(0, 3));
    T.copyTo(E2.rowRange(0, 3).col(3));

    static double changeCoordArray[4][4] = { { 1, 0, 0, 0 },{ 0, -1, 0, 0 },{ 0, 0, -1, 0 },{ 0, 0, 0, 1 } };
    static cv::Mat changeCoord(4, 4, CV_64FC1, changeCoordArray);

    E2 = changeCoord * E2;

    //두번째 마커에 대한 T벡터set
    std::cout << "T2 :" << std::endl;
    t2 = true;
    T.copyTo(T2);
    std::cout << T2.rows << " " << T2.cols << std::endl;
    std::cout << T2.at<double>(0) << " " << T2.at<double>(1) << " " << T2.at<double>(2) << std::endl;
    //gMatFlipImage.at<cv::Vec3b>(i, j)
```

두 번째 마커에 대하여 호모그래피로부터 마커에서 카메라 좌표로의 변환행렬을 구하여 T2 matrix에 복사한다.

3단계:

T1~T2 사이의 거리를 구한다.

1-4. 실행결과



11~12cm 사이의 값이 나옴.(실제 자로 잰 길이와 거의 유사함)

시연영상 : https://youtu.be/BNuda_bG984

2. Interaction

2-1. 구현설명

1단계: 변수선언

```
//각도관련변수들
float angle1 = 30.0f; //주전자 객체를 회전할 각도를 저장
float angle2 = -30.0f; //큐브객체를 회전할 각도를 저장
bool isSetAngle1 = false; //주전자 객체에 각도 수정이 완료되었으면 true 아니면 false
bool isSetAngle2 = false; //큐브객체에 각도 수정이 완료되었으면 true 아니면 false
float radius = 5.0; //회전할 반지름

//색깔관련변수들
char currentColor1 = 'r'; //주전자의 현재색상을 저장할 변수
char currentColor2 = 'r'; //큐브의 현재색상
bool isSetColor1 = false; //주전자 객체 색상 수정 완료되었으면 true 아니면 false
bool isSetColor2 = false; //큐브객체에 색상 수정이 완료되었으면 true 아니면 false

//현재모양관련 변수들
char currentShape1 = 's'; //주전자의 현재모양을 저장
char currentShape2 = 's'; //큐브의 현재모양 저장
bool isSetShape1 = false; //주전자 객체 모양 수정완료되었으면 true 아니면 false
bool isSetShape2 = false; //큐브 객체 모양 수정완료되었으면 true 아니면 false

int change_cnt = 0; //색변화횟수를 저장 - 가위 : 4, 바위 : 2, 보 : 10
```

2단계 : teapot과 cube에 각도설정

teapot:

```
if (angle1 >= 360.0f) //360도보다 크거나 같으면 0으로 reset
{
    angle1 = 0.0f;
}
if (RockCnt >= 2) //바위가 5번이상 인식되면
{
        isSetAngle1 = true;
        glRotatef(angle1, 0.0f, 1.0f, 0.0f); //마커를 중심으로 공전
        glTranslated(radius, 0, 0);
        glRotatef(angle1, 0.0f, 1.0f, 0.0f); //자전
        angle1 += 30.0f;

    if (isSetAngle1 && isSetAngle2) //두객체에 대한 각도가 모두 수정되었으면
    {
        isSetAngle1 = false;
        isSetAngle2 = false;
        RockCnt = 0;//바위가 나온 횟수를 0으로 초기화
    }
}
```

cube :

```
if (angle2 <= -360.0f)
{
    angle2 = 0.0f;
}

if (RockCnt >= 2) //바위가 5번이상 인식되면
{
    //glTranslatef(1.5f, 0.0f, -7.0f);
        isSetAngle2 = true;
        glRotatef(angle2, 0.0f, 1.0f, 0.0f); //마커를 중심으로 공전
        glTranslated(radius, 0, 0);
        glRotatef(angle2, 0.0f, 1.0f, 0.0f); //자전
        angle2 -= 30.0f; //-30도 회전

    if (isSetAngle1 && isSetAngle2) //두 객체모두 각도가 모두 수정되었으면
    {
        isSetAngle1 = false;
        isSetAngle2 = false;
        RockCnt = 0;
    }
}
```

3단계 : teapot과 cube에 색깔설정

teapot :

```
if (SissorsCnt >= 10) //가위가 10번이상 인식되면
{
    if (currentColor1 == 'r')
    {
        currentColor1 = 'g';
    }
    else if (currentColor1 == 'g')
    {
        currentColor1 = 'b';
    }
    else if (currentColor1 == 'b')
    {
        currentColor1 = 'r';
    }
    isSetColor1 = true;
    if (isSetColor1 && isSetColor2) //두객체에 대한 색깔이 모두 수정되었으면
    {
        isSetColor1 = false;
        isSetColor2 = false;
        SissorsCnt = 0;//가위가 나온 횟수를 0으로 초기화
    }
}

//현재 상태에따라서 색깔을 설정
if (currentColor1 == 'r')
{
    glColor3f(1.0f, 0.0f, 0.0);
}
else if (currentColor1 == 'g')
{
    glColor3f(0.0f, 1.0f, 0.0f);;
}
else if (currentColor1 == 'b')
{
    glColor3f(0.0f, 0.0f, 1.0f);
}
```

cube:

```
if (SissorsCnt >= 10) //가위가 10번이상 인식되면
{
    if (currentColor2 == 'r')
    {
        currentColor2 = 'g';
    }
    else if (currentColor2 == 'g')
    {
        currentColor2 = 'b';
    }
    else if (currentColor2 == 'b')
    {
        currentColor2 = 'r';
    }
    isSetColor2 = true;
    if (isSetColor1 && isSetColor2) //두객체에 대한 색깔이 모두 수정되었으면
    {
        isSetColor1 = false;
        isSetColor2 = false;
        SissorsCnt = 0;//가위가 나온 횟수를 0으로 초기화
    }
}

//현재 상태에따라서 색깔을 설정
if (currentColor2 == 'r')
{
    glColor3f(1.0f, 0.0f, 0.0);
}
else if (currentColor2 == 'g')
{
    glColor3f(0.0f, 1.0f, 0.0f);;
}
else if (currentColor2 == 'b')
{
    glColor3f(0.0f, 0.0f, 1.0f);
}
```

4단계 : teapot과 cube에 모양설정(wire/solid)

teapot :

```
if (PaperCnt >= 1) //보가 1번이상 인식되면
{
    if (isSetShape1 == false)
    {
        if (currentShape1 == 's')
        {
            currentShape1 = 'w';
        }
        else if (currentShape1 == 'w')
        {
            currentShape1 = 's';
        }

        isSetShape1 = true;
    }

    if (isSetShape1 && isSetShape2) //두객체에 대한 모양이 모두 수정되었으면
    {
        isSetShape1 = false;
        isSetShape2 = false;
        PaperCnt = 0;//보가 나온 횟수를 0으로 초기화
    }
}

//현재 상태에따라서 객체의 모양을 설정
if (currentShape1 == 's')
{
    glutSolidTeapot(2.5);
}
else if (currentShape1 == 'w')
{
    glutWireTeapot(2.5);
```

cube:

```
if (PaperCnt >= 1) //보가 1번이상 인식되면
{
    if (isSetShape2 == false)
    {
        if (currentShape2 == 's')
        {
            currentShape2 = 'w';
        }
        else if (currentShape2 == 'w')
        {
            currentShape2 = 's';
        }

        isSetShape2 = true;
    }
    if (isSetShape1 && isSetShape2) //두객체에 대한 모양이 모두 수정되었으면
    {
        isSetShape1 = false;
        isSetShape2 = false;
        PaperCnt = 0;//보가 나온 횟수를 0으로 초기화
    }
}

//현재 상태에따라서 객체의 모양을 설정
if (currentShape2 == 's')
{
    glutSolidCube(2.5);
}
else if (currentShape2 == 'w')
{
    glutWireCube(2.5);
}
```

5단계 : 이전과제에서 구현한 interection 함수와 multi marker tracking을 thread를 이용해서 동시에 실행

```
int main(int argc, char ** argv)
{
        std::thread t1(&interaction);

        MultiMarkerTracking(argc, argv);
        t1.join();


    return 0;
}
```

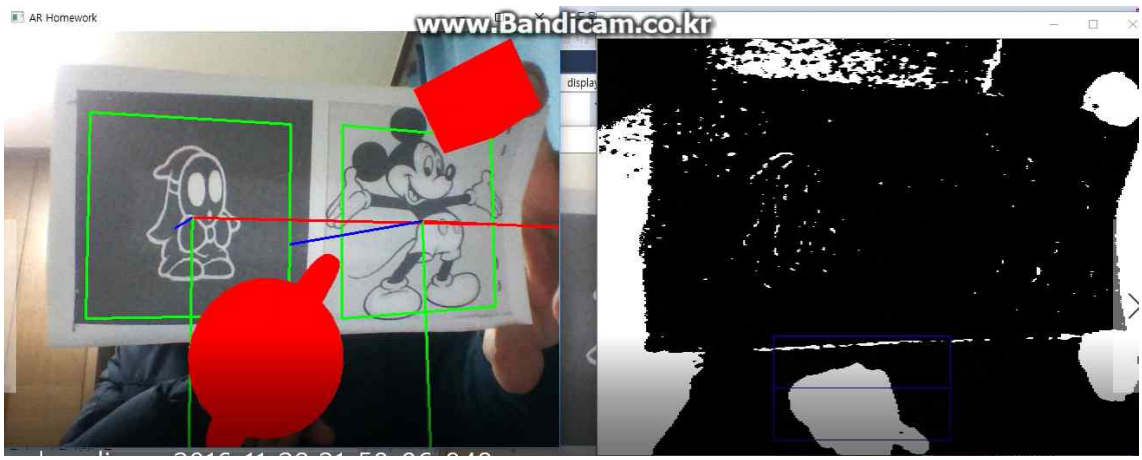
*interection 함수에 추가된부분

```
//가위 바위 보 판단
if (change_cnt == 2)
{
    std::cout << "Rock" << std::endl;
    RockCnt++;
}
else if (change_cnt == 4)
{
    std::cout << "Sissors" << std::endl;
    SissorsCnt++;
}
else if (change_cnt == 10)
{
    std::cout << "Paper" << std::endl;
    PaperCnt++;
}
else
    std::cout << "Unknown" << std::endl;
```

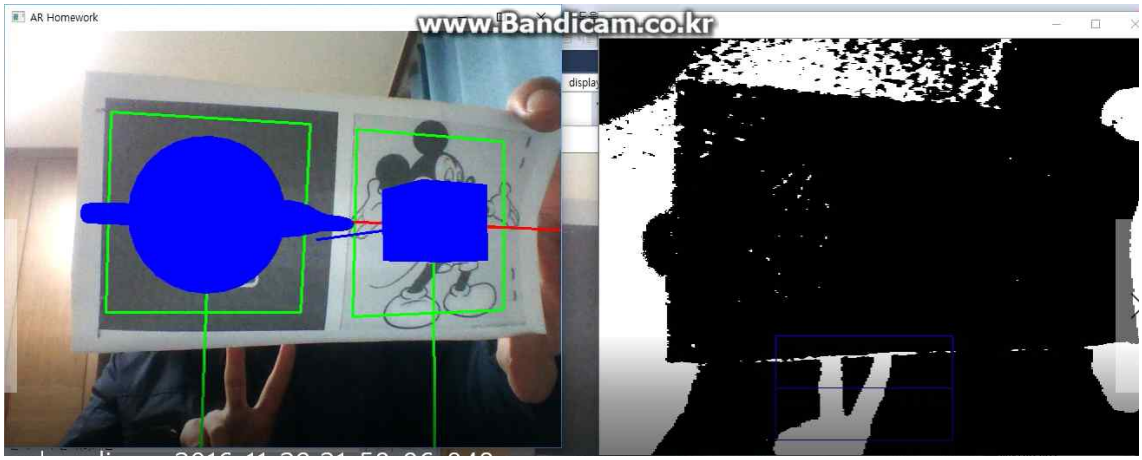
가위,바위,보 인식에 delay를 주기위해서 각각에 count를 올리고 그 카운트 이상이 될 때 interection이 되도록 구현.

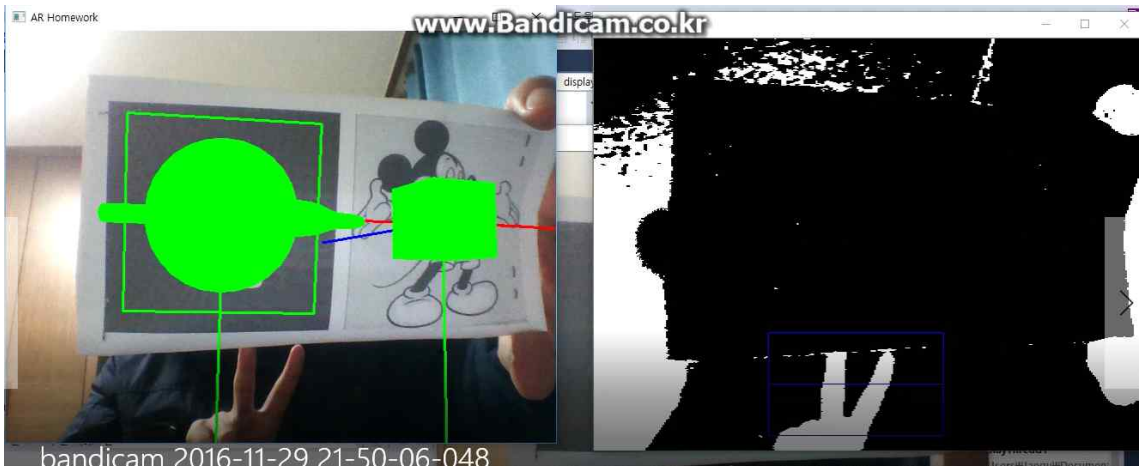
2-2. 실행결과

회전:

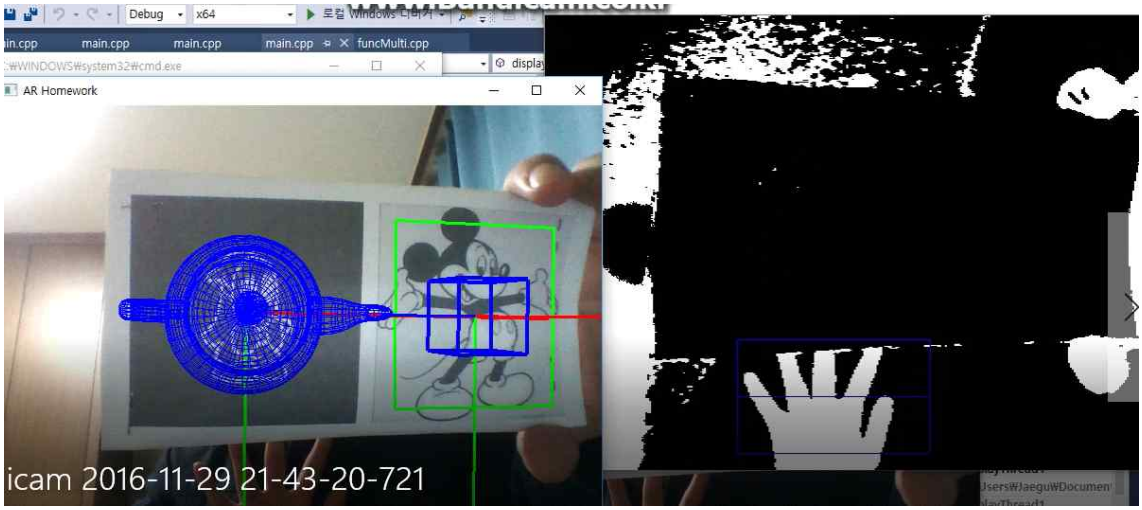


색깔변환 :


AR Homework
www.Bandicam.co.kr


AR Homework
www.Bandicam.co.kr
bandicam 2016-11-29 21-50-06-048

모양변환 :



시연영상 : https://youtu.be/EsMldvsfpaA