제목 : Node의 이동에 의한 Handoff발생 Simulation

IT대학 전자공학부 2016114203 이정훈

Ⅰ. 요약 : Mobility를 갖는 node가 기지국의 coverage를 벗어난다면 어떻게 해야 할까? 이런 경우 다른 기지국으로의 handoff는 불가피할 것이다. 그렇다면 한번 Call generation이 일어났을 때 평균적으로 몇 번의 handoff 가 일어날까에 대한 궁금증이 생긴다. 물론 대답은 "상황에 따라 다르다" 이겠지만 여러 가지 확률변수(R.V. : Random Variable)를 이용함으로써 일반적인 접근을 통해 simulation을 구현해 보았다. 또한 여러 가지 조건들을 변경해 가면서 경향성을 파악해 보았다. 즉, 특정 변수와 handoff 발생 횟수와의 관계를 그래프를 통해 파악해 보았다.

Ⅱ. 서론 : hexagonal cell environments에서 handoff simulation을 2가지 프로그램 코드를 작성함으로써 simulation을 구현하였다. Code1은 Cell의 반경 길이[m], 방향 전환 사이 평균거리[m]등 총 4가지 parameter를 입력받는다. 그다음 육각형으로 근사 된 Cell 내부에서 Random하게 발생 된 Call이 어디로 움직일지, 얼마나 통화할 지, 얼마의 속도로 움직일지 등에 따라 handoff가 몇 번 일어나는지 조사한다. 또한 이러한 Call generation을 Random 하게 10000번 일으키고 각각의 handoff 횟수를 기록함으로써 입력 한 조건에서 평균적으로 몇 번의 handoff가 발생하게 되는지 알 수 있다.
두 번째 코드는 첫 번째 프로그램과 전반적으로 유사한 코드를 가지지만 경향성을 파악할 수 있다는 점에서 차이를 가진다. 다른 parameter 값들은 고정시켜 둔 채 하나의 parameter에만 변화를 주면서 그 parameter의 값이 평균 handoff 발생 횟수에 어떠한 영향을 끼치는가에 대해 그래프를 통해 확인할 수 있도록 했다.

Ⅲ. 본론 :

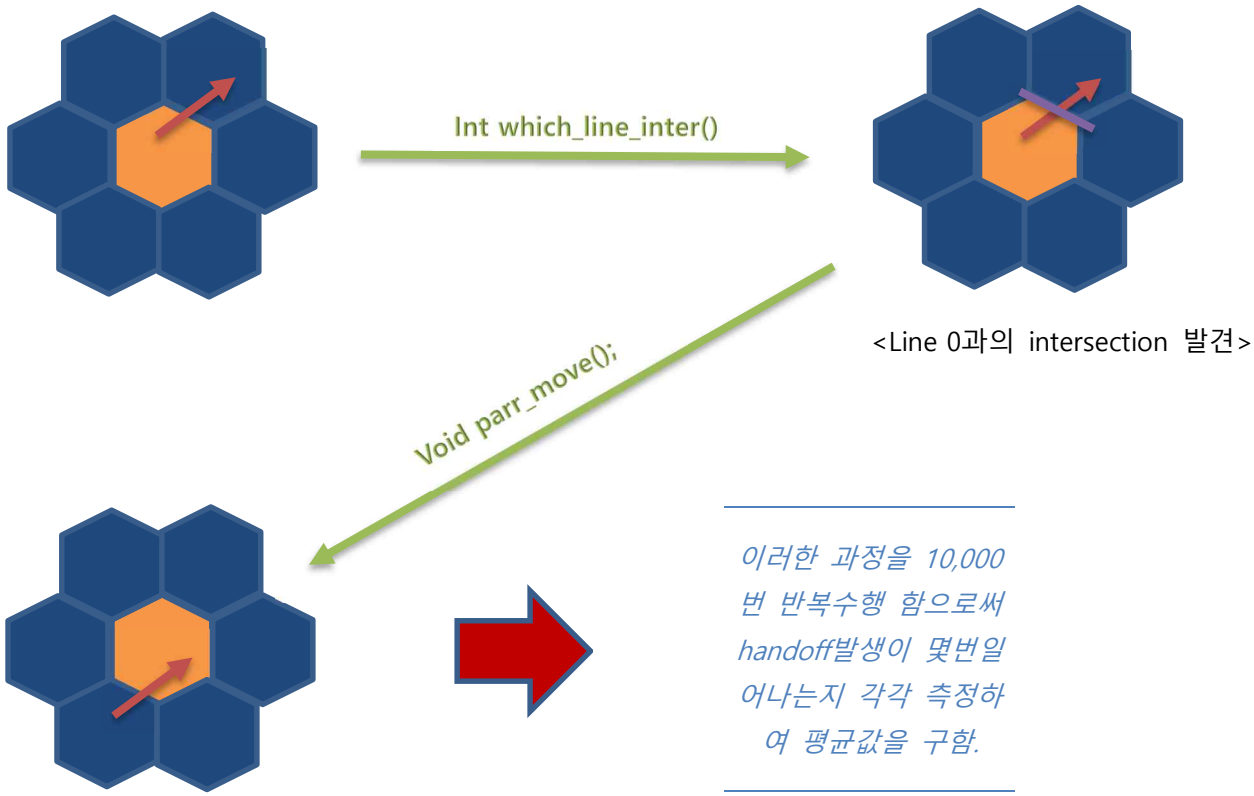
<Code 1>

첫 번째 코드는 4가지 parameters(Cell의 반경길이[m], node이동속도의 범위[Km/h], 방향전환사이 평균거리[m], 평균전화시간[min])를 입력 받은 후 Tc(Call Time)가 음수가 될 때까지 반복해서 node를 이동시키며 handoff 횟수를 측정하는 방식이다. 이때 node의 이동방향, node의 위치, 통화시간 Tc, node의 이동속도는 R.V.의 값을 따라가며 node의 이동방향은 [0,2*PI]인 uniform R.V.이고 이동속도, 통화시간은 exponential R.V.값을 지닌다.

```
double Rand_0to1() {                    // 0~1 Randomly generation
            return (double)rand() / 32767;
}
double Rand_uniform(double start, double end) {
            double uniform = Rand_0to1() * (end - start) + start;
            return uniform;
}
double Rand_velocity() {                // 속도 2~6km/h uniformly R.V.
            double velocity = Rand_0to1() * 4 + 2;
            return velocity;
}
double Rand_angle() {                   // 0 ~ 2PI Randomly generation
            double angle = Rand_0to1() * 2 * M_PI;
            return angle;
}
double Rand_x() {
            double x = Rand_0to1() * 500 * sqrt(3) - 250 * sqrt(3);
            return x;
}
double Rand_y() {
            double y = Rand_0to1() * 1000 - 500;
            return y;
}
double exponential(double lambda) {
            double first = log(1.0-0.9999999999999*Rand_0to1());
            double time = first / (-lambda);
            return time;
}
```

Rand()함수를 이용해 0~1사이의 수를 Randomly generate하는 함수를 만들었고 이 함수를 이용해서 다른 Uniform R.V.와 exponential R.V.들을 생성하였다.

<전체적인 코드 흐름>



<Line 0과의 intersection 발견>

이러한 과정을 10,000번 반복수행 함으로써 handoff발생이 몇번일어나는지 각각 측정하여 평균값을 구함.

<Intersection line의 number에따라 parallel move 함>

```
do {//균일하게 육각형 내에서 Call generation 하기위한 반복문
        node.x = Rand_x();
        node.y = Rand_y();
} while (Nodecheck(&node,Cell_length) == 0);
```

```
int Nodecheck(Node* ptr,double Cell_length) {
        if (ptr->x < (-Cell_length/2) * sqrt(3)) { return
0; }
        else if(ptr->x > (Cell_length / 2) * sqrt(3))
{ return 0; }
        else if ((ptr->y) >= ((ptr->x) / (-sqrt(3)) +
Cell_length )) { return 0; }
        else if ((ptr->y) <= ((ptr->x) / (-sqrt(3)) -
Cell_length)) { return 0; }
        else if ((ptr->y) >= ((ptr->x) / (sqrt(3)) +
Cell_length)) { return 0; }
        else if ((ptr->y) <= ((ptr->x) / (sqrt(3)) -
Cell_length)) { return 0; }
        else { return 1; }
        return 1;
}
/*Nodecheck라는 함수를 만들어서 hexagonal외부에 있다면 0을
return하고 내부에 존재한다면 1을 return하도록 하였다. 또한 이
함수는 이후에도 유용하게 사용된다.*/
```

1. Hexagonal내부에서 균일한 확률로 Call을 generation

이 때 rand()함수를 이용한 uniform R.V.또는 exponential R.V.한 변수를 만드는 과정은 단순한 산수를 통해 구현할 수 있는 쉬운 프로그래밍 난이도이기 때문에 자세한 설명은 생략하였으며 자세한 구현 방식은 "handoff.c" 파일에서 확인할 수 있다.

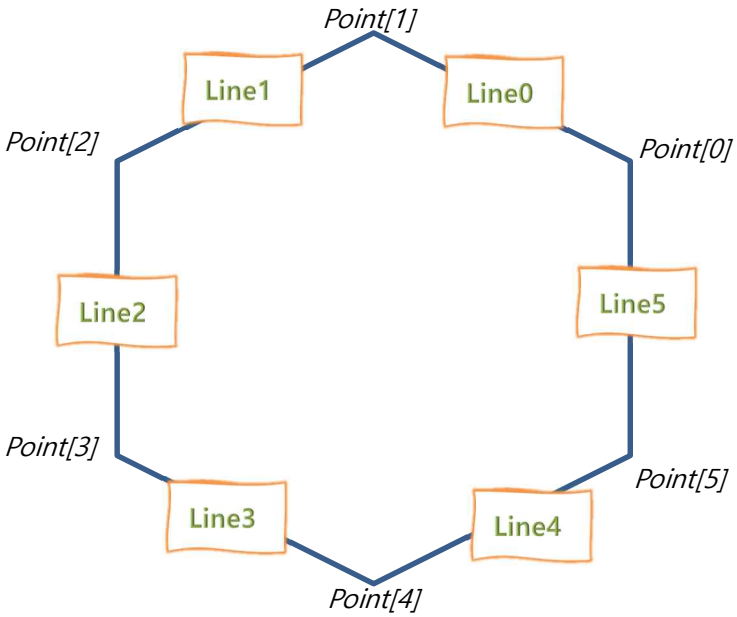
2. Tc를 R.V.로 생성시키고 Tc가 음수가 될 때까지 node의 angle, velocity, distance를 바꿔가며 이동시킴

```
int h_o_num = 0;
double Tc = exponential(1.0 / avg_call_time);       //Tc is exponential R.V.
printf("Tc : %lf₩n",Tc);
int num_of_move = 1;
do {
        node.distance = exponential(1.0 /  avg_turn_distance);       //exponential R.V
        node.velocity = Rand_uniform(vel_min, vel_MAX);     //uniform R.V.   [2000m/60min,6000m/60min]
        node.angle = Rand_uniform(0.0, (double)2 * M_PI);           //uniform R.V.
        printf("distance : %lf / velocity : %lf / angle : %lf ₩n", node.distance, node.velocity,
        node.angle);
        double Tc_before = Tc;    //Tc값 임시 저장
        Tc -= node.distance / node.velocity; //Tc값 감소시킴
        printf("%d번 이동 후 남는 Tc : %lf₩n",num_of_move , Tc);
        if (Tc < 0) {
                printf("Tc가 음수인경우에 대한 조건문 시작₩n");
                double distance_ratio = Tc_before / (node.distance / node.velocity);          //임시저장
해 뒀던 Tc와 (distance/velocity)사이 관계를 구해서 이동하는 거리를 계산 후 Nodecheck
                node.distance *= distance_ratio;
                move_node(&node);
                if (Nodecheck(&node, Cell_length) == 1) {
                        buffer += h_o_num;
                        fprintf(fp, "%d,", h_o_num);
                        printf("%d번째 Call END------------------------------₩n", i);
                }
                else {
                        h_o_num++;
                        buffer += h_o_num;
                        fprintf(fp, "%d,", h_o_num);
                        printf("handoff를 발생시킴과 동시에 %d번째 CALL END-----------------₩n", i);
                }
        }
        else {
                double before_x = node.x;
                double before_y = node.y;
                move_node(&node);          //R.V.에 따라 노드위치 이동
                printf("%d번째 직진이동 후 좌표 : (%lf,%lf)₩n", num_of_move, node.x, node.y);
                num_of_move++;
                int line = -1;   //처음에 닿는 line=-1로 초기화
                while (Nodecheck(&node, Cell_length) == 0) {//도착점이 CELL 밖에 있으면
                        which_line_inter(&line, before_x, before_y, &node, point);
                                                        //어떤 변과 만나는지 체크
                        printf("TEST : return된 *line 값 :: %d ₩n", line);
                        printf("before parr_move : (%lf,%lf)₩n", node.x, node.y);
                        parr_move(&line, &before_x, &before_y, &node, Cell_length) ;//평행이동시킴
                        h_o_num++;
                }
        }
} while (Tc >= 0);
```

Tc(Call time)을 생성시킨 후 node 의 distance, velocity, angle 또한 R.V.로 생성 및 이동시킨다. 이동시킨 후 node 의 위치가 밖에 있을지 hexagonal Cell 내부에 있을지는 이전에 만들었던 nodecheck()함수를 통해 판별이 가능하다.

Nodecheck()함수를 통해 노드가 이동 후임에도 불구하고 Cell 내부에 있다고 판단되면 다시 node 의 distance, velocity, angle 값을 새롭게 입력받아 Tc 가 음수가 될 때까지 계속해서 이동시킨다.

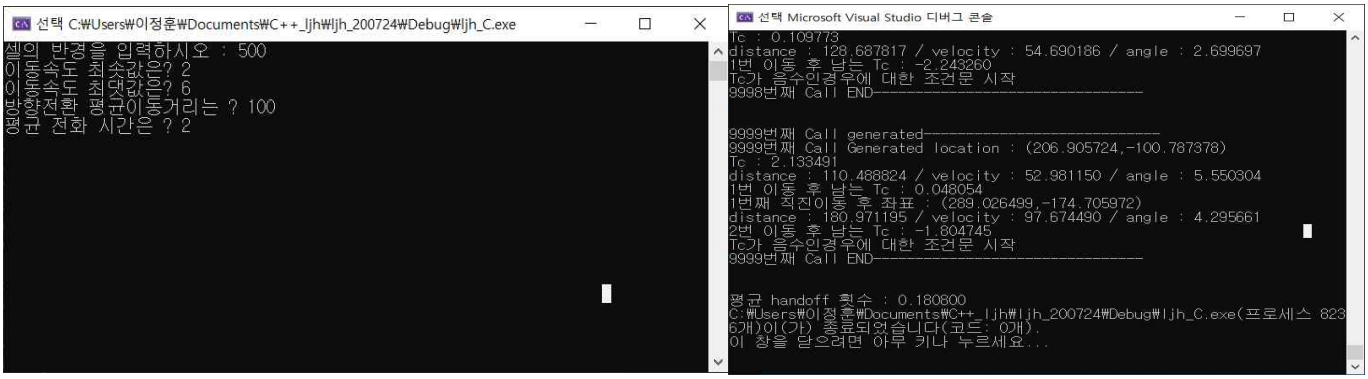
그러나 Cell 의 외부에 있다고 판단되는 경우 which_line_inter()이라는 함수를 통해 hexagonal Cell 의 변 0~5 개 중에 어떤 변과 intersection 하는지 조사한다. 만약 line0 과 intersection 한다 판단되는 경우 parr_move()함수를 통해 노드의 위치를 평행이동 시킨다. 이 때 평행이동 시킨다는 의미는 한번의 handoff 가 발생했다는 의미와 같고, 이에 따라 우리는 Tc 값이 음수가 될 때 까지 몇 번의 평행이동이 일어났는지를 통해 handoff 발생 횟수를 측정 가능하게 된다. 또한 hexagonal Cell 의 각 변의 index 는 다음과 같이 지정하였다.




```
Point point[6] = {
        {Cell_length / 2 * sqrt(3) , Cell_length / 2},
        {0,Cell_length},
        { -Cell_length / 2 * sqrt(3) , Cell_length / 2},
        { -Cell_length / 2 * sqrt(3) , -Cell_length / 2},
        {0, -Cell_length},
        {Cell_length / 2 * sqrt(3) ,  -Cell_length / 2}
};
/*point배열을 이용하여 hexagonal Cell의 각점을 좌표로 지정하였고 Cell_length라는 double형 변수에 따라 좌표가 변할 수 있도록 하여서 추후에 Cell의 size변화에 대해서도 동작할 수 있도록 프로그램을 작성하였다.*/
```

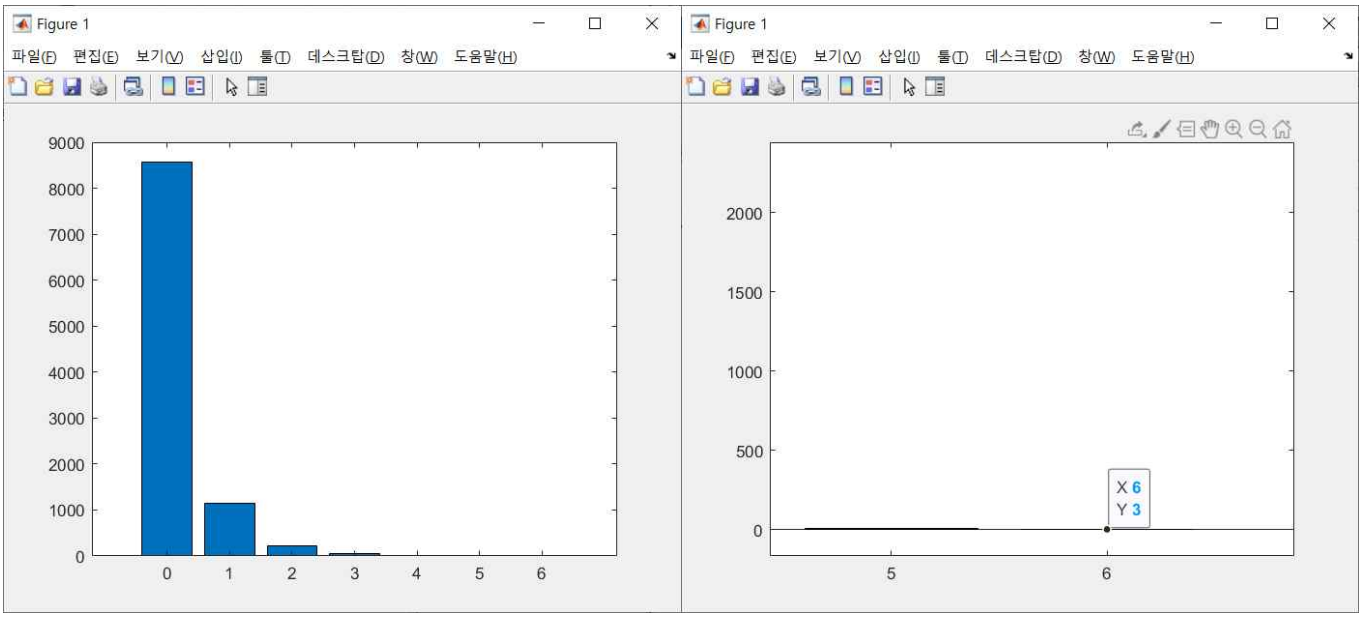
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define _USE_MATH_DEFINES
#include <math.h>
int main() {
        for (int i = 0; i < 10000; i++) {
        -----설명 한 프로그램----
        }
        printf("평균 handoff 횟수 : %lf", (double)buffer / 10000);
}
```

3. 위의 과정을 10,000번 반복수행 함으로써 handoff 평균을 구할 수 있다.



실행결과 평균 handoff 횟수가 0.18~2사이의 값이 나타나는 것을 경험적으로 알 게 됐으며 fprintf()를 통해 handoff.txt함수를 생성하도록 만들었으며 이 txt파일에는 10,000번 반복실행하는동안 각각 몇번의 handoff가 일어났는지, 즉, 10000개의 횟수가 적혀있다.

이 txt파일을 matlab의 handoff_plot.m()사용자 정의함수를 만들었고 ho_simul.m이라는 스크립트를 생성시켜서 한문장으로 visualize가 가능토록 프로그래밍 하였다. 결과는 다음 그래프와 같이 나타났다.



한 번 10,000번 실행 하였을 때 일어나는 최대 handoff횟수는 6번이었고 가장 많이 나타난 1번의 handoff횟수에 압도적으로 적은 횟수인 3번동안 3번의 handoff가 발생했음을 확인하였다. 프로그램을 여러번 실행해 봄으로써 최대 8번까지의 handoff도 확인하였다.

<Code 2>

두번째 코드는 첫번째 코드에 기반을 두고 변형하여 작성하였다. Code1이 여러가지 조건들을 입력받아 그에따른 handoff 횟수를 측정하고 평균 handoff횟수를 측정하는데 목표가 있었다면 Code2는 다른 모든 조건들은 고정한 채 하나의 parameter값만 일정하게 증감시킴으로써 그 parameter와 handoff발생과의 연관성을 파악해 보고 그 경향성을 그래프를통해 확인함에 있다. 따라서 program code에는 큰 변화가 없으며 몇가지 함수를 좀 더 추가했다는 점이 Code1과 Code2의 차이라 할 수 있다.

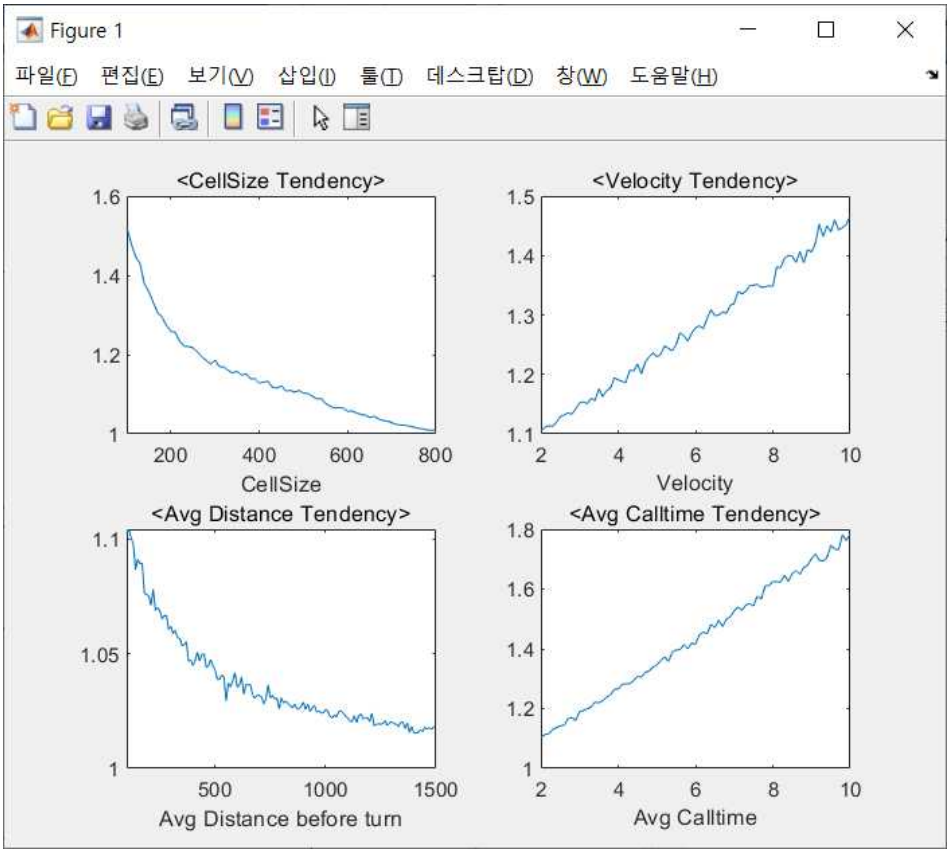
Program 전반적인 내용은 Code1에서 설명했기 때문에 어떻게 Code2를 실현하였는지는 간단하게 설명 하겠다. Code1에서 main함수가 하던 역할을 새로운 함수 operate10000()함수가 하도록 했다. 이젠 operate10000()함수에 조건을 입력만 해 준다면 한번의 함수 호출로 10,000번의 실행이 가능하게 된 것이다. 이 코드는 Code1과 매우 유사하므로 자세한 사항이 궁금하다면 "handoff_tendency.c"에서 operate10000()함수를 직접 확인하길 바란다.

Code2에서 새롭게 추가된 함수는 `void Cellsize_tendency(Node* node,FILE* fp)`, `void velocity_tendency(Node* node, FILE* fp)`, `void distance_tendency(Node* node, FILE* fp)`, `void Calltime_tendency(Node* node, FILE* fp)` 총 4가지가 전부이다. 이름을 통해서도 알 수 있겠지만 각각의 parameter가 변화함에 따라 평균 handoff횟수에 어떠한 영향을 미치는지 조사하기 위해 data를 작성하는 역할을 해 주는 함수이다. 매우 단순한 역할을 갖는 함수이므로 2가지 정도만 display하겠다.

for문을 통해 반복적으로 하나의 parameter값만을 변화시키면서 평균handoff횟수를 txt파일에 입력하는 역할을 가진다. 따라서 4개의 함수를 통해 4개의 .txt.파일이 생성되며 이는 Matlab을 통해 visualize하였다.

```
void Cellsize_tendency(Node* node,FILE* fp) {
            fp = fopen("C:₩₩Users₩₩이정훈₩₩Documents₩₩Cellsize_tendency.txt", "w");
            double vel_min = 2 * 1000 / 60.0; double vel_MAX = 6 * 1000 / 60.0;
            double avg_turn_distance = 100.0;
            double avg_call_time = 2.0;
            for (double Cell_size = 100;Cell_size <= 800; Cell_size += 10) {
                        Point point[6] = {
                        {Cell_size / 2 * sqrt(3) ,Cell_size / 2},
                        {0,Cell_size},
                        { -Cell_size / 2 * sqrt(3) , Cell_size / 2},
                        { -Cell_size / 2 * sqrt(3) , -Cell_size / 2},
                        {0, -Cell_size},
                        {Cell_size / 2 * sqrt(3) ,  -Cell_size / 2}
                        };
                        fprintf(fp, "%lf,", operate10000(node, Cell_size, avg_call_time, avg_turn_distance, vel_min, vel_MAX, point));
            }
            fclose(fp);
}
```

```
void velocity_tendency(Node* node, FILE* fp) {
            fp = fopen("C:₩₩Users₩₩이정훈₩₩Documents₩₩Velocity_tendency.txt", "w");
            Point point[6] = {
                        {500.0 / 2 * sqrt(3) ,500.0 / 2},
                        {0,500.0},
                        { -500.0 / 2 * sqrt(3) , 500.0 / 2},
                        { -500.0 / 2 * sqrt(3) , -500.0 / 2},
                        {0, -500.0},
                        {500.0 / 2 * sqrt(3) ,  -500.0 / 2}
            };
            double avg_turn_distance = 100.0;
            double avg_call_time = 2.0;
            for (double vel = 2; vel <= 10 ; vel += 0.1) {
                        fprintf(fp, "%lf,", operate10000(node, 500.0, avg_call_time, avg_turn_distance, vel*1000/60.6, (4 + vel) * 1000 / 60.6, point));
            }
            fclose(fp);
}
```



결과적으로 변화하는 parameter의 값에 따라 평균 handoff가 비례 또는 반비례하는 그래프를 확인할 수 있었고 이 결과 그래프를 통해 몇가지 직관적인 해석을 하려 노력했다.

Code2에대한 직관적인 해석의 결과는 다음과 같다.

1. Cell_size_tendecy : Cell의 반경이 커짐에 따라 Handoff 횟수가 줄어드는 결과는 자연스럽다. 다른 parameters들 ,즉, 속도나 이동거리 값들은 고정된 채 Cell의 넓이만 넓어지는 경우에는 node가 Cell을 벗어날 확률도 줄어들기 때문이다.
2. Velocity_tendency : Cell의 넓이 등이 고정되어 있을 때 node의 이동속도가 빨라진다면 당연히 handoff횟수가 증가할 것이고 이는 그래프를 통해서도 확인이 가능하다. 예를 들어 통화를 하며 걷고있는 상황과 통화를 하면서 운전을 하고있는 경우에 대해 생각해보면, 어떤 경우에 handoff가 더 많이 일어날지 쉽게 추측이 가능하다.
3. **Distance_tendency** : 많은 고민을 했던 부분은 방향전환이전의 평균 이동거리와 handoff사이의 관계이다. 이 부분이 직관적으로 이해가 어려웠다. 방향전환사이 평균이동거리가 길어진다는 뜻은 node가 방향전환을 자주하지 않고 길게 직진이동을 하는 경우가 더 많이 발생한다는 뜻이다. 이러한 경우 평균 handoff횟수가 점점 더 줄어든다고 생각했다. 그 이유는 만약 방향전환사이 평균거리가 짧아진다면 node의 방향이 자주 변하는 것을 뜻하고, 극단적인 경우 cell boundary근처에 있는 node를 생각하면 handoff횟수는 훨씬 많이 발생할 수 있기 때문이다. 물론 나의 예상대로 그래프가 나왔지만(subplot의 index : 3번째 그래프이다.) 다른 인턴의 그래프는 handoff횟수가 증가하다가 특정한 값에 수렴한다는 결과를 보이면서 차이를 보였으며 수식적 해석이 아닌, simulation 결과를 비교했기 때문에 누구의 결과가 잘못된 지 직관을 통해 파악이 어렵다는 해석의 아쉬운 점을 남겼다.
4. Call_time_tendency : 통화시간이 길어질수록 handoff를 할 확률인 훨씬 더 많아진다 볼 수 있다. 물론 대부분의 일반적인 경우 통화시간이 길어지면 집안에서 전화하는 경우, 앉아서 통화하는 경우가 대부분이지만, 현재 simulation에서는 node는 항상 R.V.에 따라 각도를 가지며 이동한다 가정하였기 때문에 이러한 가정을 고려한다면 handoff가 증가하는 것도 쉽게 이해가 가능하다.

IV. 결론

기지국이 cover할 수 있는 반경을 hexagonal하게 근사화 시키고 여러가지 확률변수 개념을 도입함을 통해 handoff가 몇 번 일어나는지에 대해 조사 할 수 있다는 점이 새롭고 재밌었다. 한가지 아쉬운점이 있다면 어디까지나 simulation이기 때문에 현실반영에서의 한계점을 코딩하면서 느꼈다는 것이다. 실제로 통화시간이 길어진다면 방안에서의 통화, 회의중이거나, 카페에 앉아서 친구와 통화 하는것과 같이 정적인 상태에서의 통화가 많다고 생각한다. 그러나 우리 simulation의 가정을 node는 쉴 새 없이 방향을 바꿔가며 움직이며 그것은 Tc(Call time)가 0이 될 때까지 지속된다. 이러한 점에서 simulation의 한계점을 느꼈다. 그러나 처음으로 C프로그래밍을 활용하여 simulation을 해 보고, 알고리즘을 고민하는 데에 있어 재미를 느꼈고 이동통신에 더욱 관심을 갖게 되었다.