double find\_distance\_between\_two\_points(LatLon point1, LatLon point2) {

double x1, y1;

double x2, y2;

double averageLatitude;

// find the average latitude between two points (convert to radian)

averageLatitude = ((point1.lat + point2.lat) / 2)\* DEG\_TO\_RAD;

y1 = point1.lat \* DEG\_TO\_RAD;

y2 = point2.lat \* DEG\_TO\_RAD;

x1 = point1.lon \* cos(averageLatitude) \* DEG\_TO\_RAD;

x2 = point2.lon \* cos(averageLatitude) \* DEG\_TO\_RAD;

double distance = EARTH\_RADIUS\_IN\_METERS \* sqrt(((y2-y1)\*(y2-y1))+((x2-x1)\*(x2-x1)));

return distance;

}

unsigned find\_closest\_intersection(LatLon my\_position) {

map<vector<LatLon>, unsigned> mymap;

map<vector<LatLon>, unsigned>::iterator closest;

double distance;

unsigned closest\_intersection;

if (getNumberOfIntersections() == 0) {

closest\_intersection = 0;

} else {

for (unsigned i = 0; i < getNumberOfIntersections(); i++) {

if (closest != mymap.end()) {

vector<LatLon> intersectionposition = closest->first;

intersectionposition.push\_back(getIntersectionPosition(i));

mymap.insert(std::make\_pair(intersectionposition, closest\_intersection));

if (i == 0) {

distance = find\_distance\_between\_two\_points(my\_position, intersectionposition);

} else if (distance > find\_distance\_between\_two\_points(my\_position, intersectionposition)) {

distance = find\_distance\_between\_two\_points(my\_position, intersectionposition);

closest->second = i;

}

} else if (closest == mymap.end()) {

vector<LatLon> intersectionposition;

intersectionposition.push\_back(getIntersectionPosition(i));

mymap.insert(make\_pair(intersectionposition, closest\_intersection));

}

}

}

return closest->second;

}

vector<unsigned> find\_intersection\_street\_segments(unsigned intersection\_id) { //my part its id not numbers

/\*

map<unsigned, vector<unsigned>>::iterator intersection = intersection\_database.find(intersection\_id);

vector<unsigned> new\_vector = intersection->second;

//if (street.oneWay == true) {

for (unsigned i = 0; i < getIntersectionStreetSegmentCount(intersection\_id); i++) {

bool found = false;

if (new\_vector.empty()) {

if (intersection != intersection\_database.end()) {

vector<unsigned> new\_vector = intersection->second;

new\_vector.push\_back(getIntersectionStreetSegment(intersection\_id, i));

intersection\_database.erase(intersection\_id);

intersection\_database.insert(make\_pair(intersection\_id, new\_vector));

} else {

vector <unsigned> new\_vector;

new\_vector.push\_back(getIntersectionStreetSegment(intersection\_id, i));

intersection\_database.insert(make\_pair(intersection\_id, new\_vector));

}

} else {

for (unsigned j = 0; j < new\_vector.size(); j++) {

if (new\_vector[j] == getIntersectionStreetSegment(intersection\_id, i)) {

found = true;

}

}

if (found == false) {

if (intersection != intersection\_database.end()) {

vector<unsigned> new\_vector = intersection->second;

new\_vector.push\_back(getIntersectionStreetSegment(intersection\_id, i));

intersection\_database.erase(intersection\_id);

intersection\_database.insert(make\_pair(intersection\_id, new\_vector));

} else {

vector <unsigned> new\_vector;

new\_vector.push\_back(getIntersectionStreetSegment(intersection\_id, i));

intersection\_database.insert(make\_pair(intersection\_id, new\_vector));

}

}

}

}

return intersection->second;

vector<unsigned> street\_id;

for(unsigned i=0; i < getIntersectionStreetSegmentCount(intersection\_id);i++){

street\_id.push\_back(getIntersectionStreetSegment(intersection\_id, i));

}

return street\_id;

\*/

return segment\_database[intersection\_id];

}