Problem of this programm:

It is too slow if you use a high acuracy. Further it has just been applied for clothoids with radiusstart == infinity and where x and y greater than zero (in relativ-coordinate-system).

Attention: the program was designed for clothoids, but it handles clothoids not in the best manner, e.g. it uses a mirror if clockwise is true. Better would it be to change the direction of Hend.y().

What the program does:

It devides a spiral in parts of 90 degrees, while the last part has the angel <= 90 degrees. Then for every of this parts a bezier-curve is computed and compared with the original curve. If the biggest difference is larger than the toleranz, this part devides into 2 (then 4 then 8 then 16...) parts. For each of them a bezier curve is computed and compared with the original curve.

Interesting at this algorithm is, that it uses former values to reduce computing-time. (See start\_clot\_n, start\_clot\_o and line1,line2).

if (he->getAlignmentType() == buw::eHorizontalAlignmentType::Clothoid)

{//this loop approximates a clothoid through many bezier-curves so that the error is smaller than double border = 1e-4 meter

buw::vector2d end, pointofIntersect, x\_axis = { 1.0, 0.0 };

double \*maxi, clothoidconstant, length, R, radiusend, radiusstart, border = 0.001;

std::vector <double> tau\_part\_start, tau\_part\_end, bezier;

std::vector <buw::vector2d> start\_part, end\_part, clot\_start\_n, clot\_mid\_n, clot\_end, help\_pi;

bool clockwise;

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::Clockwise, &clockwise);

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::PI, &pointofIntersect);

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::ClothoidConstant, &clothoidconstant);

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::Length, &length);

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::EndPosition, &end);

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::RadiusEnd, &radiusend);

he->genericQuery(BlueFramework::Infrastructure::eAlignmentElementQueryId::RadiusStart, &radiusstart);

class compare

{

public:

bool operator()(double A, double B) const

{

return A < B;

}

};

R = radiusend;

if (radiusstart != std::numeric\_limits<double>::infinity())// this is area one, it swaps start and end if the clothoid is not normally normally radiusstart = infinity

{

buw::vector2d swap\_variable = end;

end = start;

start = swap\_variable;

R = radiusstart;

output += "<g>\n<path d=\"M " + std::to\_string(start.x()) + "," + std::to\_string(-start.y()) + " ";//test necessary,in the moment no test done

}

double tau = length / (2 \* R);

//area two: a spirale with more than tau==90 degree can not be approximated so it has to be divided into parts, this is done now

int zu = 1;//zu counts the parts of the spiral //curent maximum of parts is 3

tau\_part\_start.push\_back(0);

while (tau - zu \* M\_PI / 2 > 0)

{

tau\_part\_start.push\_back(zu\*M\_PI / 2);

tau\_part\_end.push\_back(zu\*M\_PI / 2);

zu++;

}

tau\_part\_end.push\_back(tau);// for the next line: the differenz between cloth\_direct and start\_direct is that the last depends on clockwise

buw::vector2d cloth\_direct = pointofIntersect - start;// the current function "create\_clothoid" from mainwindow.cpp can not draw spirals with more than 270 degrees but it is enough for civil-engineering

//if the scalarproduct of supposed clothoid-direction and mirror-even is negativ, the startdirection of the clothoid is multiplied with -1 so that the startdirection is correct

buw::vector2d mirror = (end - start).normalize();

buw::vector2d mirror\_direct = buw::createRotationZ22d(M\_PI / 2)\*mirror;

if (mirror.x()\*cloth\_direct.x() + mirror.y()\*cloth\_direct.y()<0)

{

cloth\_direct = -cloth\_direct;

}

double lineangle = buw::calculateAngleBetweenVectors(x\_axis, cloth\_direct);

start\_part.push\_back(start);//this part compute start\_part[ii], end\_part[ii] with M\_PI / 2;

for (int ii = 1; ii<zu; ii++)

{

buw::vector2d HendPoint;

HendPoint.x() = buw::HorizontalAlignmentElement2DClothoid::computeX(clothoidconstant\*sqrt(2 \* ii \* M\_PI / 2), clothoidconstant);

HendPoint.y() = buw::HorizontalAlignmentElement2DClothoid::computeY(clothoidconstant\*sqrt(2 \* ii \* M\_PI / 2), clothoidconstant);

if (clockwise == true)//computes the points if clockwise is true with a mirror formular: s(x)=x-2(x\*a)\*a // a is orthogonal to mirror

{

HendPoint = HendPoint - 2 \* (HendPoint.x()\*mirror\_direct.x() + HendPoint.y()\*mirror\_direct.y())\*mirror\_direct;

}

buw::vector2d endPoint = start + buw::createRotationZ22d(lineangle)\*HendPoint;

end\_part.push\_back(endPoint);

start\_part.push\_back(endPoint);

}

end\_part.push\_back(end);

for (int iii = 0; iii<zu; iii++)//area three: draws for every part of the spiral the bezier-approximation by computing the correct middlepoint, the bezierpoint=PointofIntersection and comparing

{ //the wrongest bezier-values with the clothoidvalues

int iv = -1;

do

{

iv++; // now the start\_part\_direction is generated

buw::vector2d start\_part\_direct = buw::createRotationZ22d(tau\_part\_start[iii])\* cloth\_direct;//this part generates the first helpline

if (clockwise == true)//computes the points if clockwise is true with a mirror formular

{

start\_part\_direct = start\_part\_direct - 2 \* (start\_part\_direct.x()\*mirror\_direct.x() + start\_part\_direct.y()\*mirror\_direct.y())\*mirror\_direct;

}

buw::Line2d line1(start\_part[iii], start\_part[iii] + start\_part\_direct);

std::vector <buw::vector2d> clot\_start\_o, clot\_mid\_o;//this part generates the OLD start and middle of the clothoidparts

if (iv > 0)

{

for (int v = 0; v < pow(2, iv - 1); v++)

{

clot\_mid\_o.push\_back(clot\_mid\_n[v]);

clot\_start\_o.push\_back(clot\_start\_n[v]);

}

}

clot\_start\_n.clear(); clot\_mid\_n.clear(); clot\_end.clear(); help\_pi.clear(); bezier.clear();

for (int vi = 0; vi < pow(2, iv); vi++)//this part generates the NEW start, middle and end of the clothoidparts

{

if (iv == 0)

{

clot\_start\_n.push\_back(start\_part[iii]);

clot\_end.push\_back(end\_part[iii]);

}

else

{

if (vi % 2 == 0)

{

clot\_start\_n.push\_back(clot\_start\_o[vi / 2]);

clot\_end.push\_back(clot\_mid\_o[vi / 2]);

}

else

{

clot\_start\_n.push\_back(clot\_mid\_o[(vi - 1) / 2]);

if (vi < pow(2, iv) - 1)

{

clot\_end.push\_back(clot\_start\_o[(vi + 1) / 2]);

}

else

{

clot\_end.push\_back(end\_part[iii]);

}

}

}

double taumiddle = tau\_part\_start[iii] + (tau\_part\_end[iii] - tau\_part\_start[iii])\*(1 + 2\*vi) / pow(2, iv + 1);

buw::vector2d H\_middle\_Point;

H\_middle\_Point.x() = buw::HorizontalAlignmentElement2DClothoid::computeX(clothoidconstant\*sqrt(2 \* taumiddle), clothoidconstant);

H\_middle\_Point.y() = buw::HorizontalAlignmentElement2DClothoid::computeY(clothoidconstant\*sqrt(2 \* taumiddle), clothoidconstant);

if (clockwise == true)

{

H\_middle\_Point = H\_middle\_Point - 2 \* (H\_middle\_Point.x()\*mirror\_direct.x() + H\_middle\_Point.y()\*mirror\_direct.y())\*mirror\_direct;

}

buw::vector2d middlePoint = start + buw::createRotationZ22d(lineangle)\*H\_middle\_Point;

clot\_mid\_n.push\_back(middlePoint);

// this part generates the direction of the end

buw::vector2d end\_direct = buw::createRotationZ22d(tau\_part\_start[iii] + (tau\_part\_end[iii] - tau\_part\_start[iii])\*(1 + vi) / pow(2, iv))\* cloth\_direct;

if (clockwise == true)

{

end\_direct = end\_direct - 2 \* (end\_direct.x()\*mirror\_direct.x() + end\_direct.y()\*mirror\_direct.y())\*mirror\_direct;

}

buw::Line2d line2(clot\_end[vi], clot\_end[vi] - end\_direct);

buw::vector2d help\_pipo;

auto Po = buw::computeIntersection(line1, line2, help\_pipo);

line1 = line2;// line1 is just at the beginning equal to (start\_part[iii], start\_part[iii]+start\_part\_direct) in all other cases line1 is the old line2

help\_pi.push\_back(help\_pipo);

//for the solution of the bezier-function. Search for t, because of the quadratic structure of bezier it is not possible for me to compute t out of given x

double \*mini, g = 0;

std::vector<double> prepreselection;

std::map<double, double, compare> preselection;

if (clot\_start\_n[vi].x()<clot\_mid\_n[vi].x() && clot\_mid\_n[vi].x()>clot\_end[vi].x() || clot\_start\_n[vi].x()>clot\_mid\_n[vi].x() && clot\_mid\_n[vi].x()<clot\_end[vi].x())// this seldom case is posible, when t gives to solutions for one x(because of the quadratic kind), but then there will be just one fitting y value

{

for (double t = 0.1; t < 1; t = t + 0.1)

{

prepreselection.push\_back(fabs((1 - t) \* (1 - t)\*clot\_start\_n[vi].y() + 2 \* t \* (1 - t)\*help\_pi[vi].y() + t \* t \* clot\_end[vi].y() - clot\_mid\_n[vi].y()));

preselection[prepreselection.back()] = t;

}

mini = std::min\_element(&prepreselection.front(), &prepreselection.back());

g = preselection[\*mini];

bezier.push\_back(fabs((1 - g) \* (1 - g)\*clot\_start\_n[vi].x() + 2 \* g \* (1 - g)\*help\_pi[vi].x() + g \* g \* clot\_end[vi].x() - clot\_mid\_n[vi].x()));

}

else

{

for (double t = 0.1; t < 1; t = t + 0.1)

{

prepreselection.push\_back(fabs((1 - g) \* (1 - g)\*clot\_start\_n[vi].x() + 2 \* g \* (1 - g)\*help\_pi[vi].x() + g \* g \* clot\_end[vi].x() - clot\_mid\_n[vi].x()));

preselection[prepreselection.back()] = t;

}

mini = std::min\_element(&prepreselection.front(), &prepreselection.back());

g = preselection[\*mini];

bezier.push\_back(fabs((1 - g) \* (1 - g)\*clot\_start\_n[vi].y() + 2 \* g \* (1 - g)\*help\_pi[vi].y() + g \* g \* clot\_end[vi].y() - clot\_mid\_n[vi].y()));

}

}

maxi = std::max\_element(&bezier.front(), &bezier.back());

} while (\*maxi > border);////here means \*maxi > border \*maxi>=border because of the while sounds funny but is true

for (int vi = 0; vi <pow(2, iv); vi++)

{

output += "Q " + std::to\_string(help\_pi[vi].x()) + "," + std::to\_string(-help\_pi[vi].y()) + " " + std::to\_string(clot\_end[vi].x()) + "," + std::to\_string(-clot\_end[vi].y()) + " ";

}

}