Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Hyperloop Network Design

Master Thesis

Sibylle Jeker

sibylle.jeker@gmail.com

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Manuel Eichelberger, Roland Schmid
Prof. Dr. Roger Wattenhofer

February 16, 2019

Abstract

We present a tool that automatically computes Hyperloop routes between any start and end destination in terms of minimizing building cost per travel time. The routes can be generated anywhere in the world. In order to reduce the building cost the Hyperloop routes are mainly planned above motorways and railroads, such that no new land is occupied. Travel time is reduced by building the routes as straight as possible.

The route finding is done on a bitmap consisting of existing railroads and motorways. A website shows a map with a precomputed Hyperloop network consisting of preselected routes with two different settings of cost parameters. The Hyperloop routes can be compared to straight-line underground routes and the existing SBB train routes. The user can interactively adjust cost, speed and acceleration parameters to recompute the travel time and the building cost of the Hyperloop routes.

Contents

Chapter 1

Introduction

In this thesis, we want to analyze a cheaper and faster way of travelling. The most common public transport systems at the moment are buses, metros, trains and planes. What if we could introduce a new system that has a maximum speed in the range of planes, but is built on the ground? Such a transport system, called Hyperloop Alpha [1], is possible in theory and was introduced by Elon Musk in 2013. It operates in almost vacuum tubes such that the air resistance can be minimized, and higher speeds can be achieved.

We use the design ideas of Hyperloop Alpha, and build a tool that automatically computes Hyperloop routes, only given a bitmap of existing rails and motorways and a start and end destination. The goal is to find the optimal route in terms of minimizing the building cost per travel time. The building cost is minimized by planning the routes mainly above railroads and motorways, such that traffic routes can be reused and no new land has to be occupied. Travel time is reduced by avoiding sharp curves, such that the Hyperloop pods can have high velocity while maintaining a low acceleration to ensure that passengers feel comfortable.

1.1 Related Work

In 1907, Robert Goddard first proposed a vacuum train and in 1972 the *RAND Corp.* developed an ultrasonic underground railway system called the *Vactrain* [2]. These concepts are similar to the Hyperloop. However, Elon Musk accomplished to get a lot attention, especially with the Hyperloop Pod Competition [3] from his company SpaceX, where research teams design and build fast pods and compete against each other. Several commercial companies started to develop their own Hyperloop systems as well, the biggest are Virgin Hyperloop One [4] and Hyperloop Transportation Technologies [5].

Another highspeed transport idea was introduced under the name Swissmetro [6] in Switzerland. It consists of an underground train connecting the largest cities in Switzerland. Although it went into liquidation in 2009 be-

cause of missing financial support, SwissMetro was rebranded into EPFLoop [7]. EPFLoop is a research team at EPF Lausanne, that works on building Hyperloop pods.

In the next two sections the concepts Hyperloop Alpha and Swissmetro are explained in more detail. Afterwards, the related work of existing transportation systems and pathfinding is discussed.

1.1.1 Hyperloop Alpha

Hyperloop Alpha [1] is Elon Musks proposal of a high speed transport system. People are transported in pods that can reach speeds of up to 1220 km/h and carry up to 28 people. The pods operate in almost vacuum tubes such that the air resistance can be minimized. To make sure there is no friction with the tube, several air skis are attached at the bottom of the pods. The skis use two mechanisms to prevent friction using pressurized air. First, the front of the air ski is slightly elevated relatively to the back of the ski. This results in a thin pressurized air film between the ski and the tube walls. Second, the air skis use external pressurized air to support carrying the weight of the capsule. The external pressurized air is provided by the compressor, that collects accumulated air in the front of the pod. An illustration of the Hyperloop pods and air ski is given in Figure 1.2. To accelerate and decelerate the pods, magnetic levitation is used.

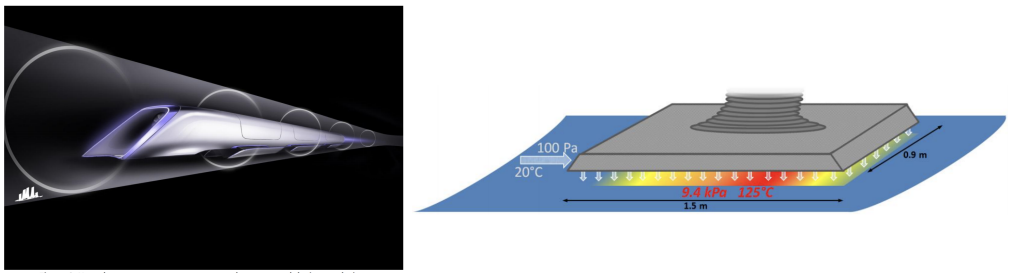


Figure 1.1: Illustration of a Hyperloop pod and an air ski.

The tubes can be built either under- or above ground. To reduce cost, Elon Musk proposed to mainly build the routes above streets or rails, such that no new land has to be occupied and expensive cost for tunnels can be saved [1]. Pylons are used every 30 meters to stabilize the tubes that are built on average 6 meters above ground. Figure 1.2 shows Hyperloop tubes with pylons.

A basic route design of the Hyperloop is presented in the paper and consists of a route from San Francisco to Los Angeles. They did a detailed cost, speed and duration analysis of the route. In contrast to this fixed route design, our tool automatically computes Hyperloop routes in terms of minimizing the building

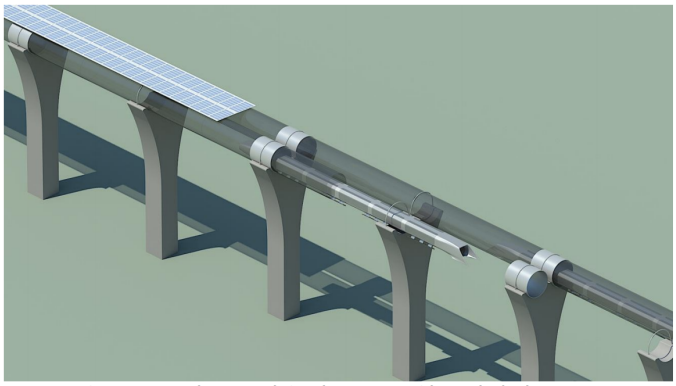


Figure 1.2: Hyperloop tubes with pylons.

costs per travel time and it can be extended to compute routes anywhere in the world.

A detailed collection of parameters, such as maximum possible acceleration, or tunnel cost is provided in the paper. We use these parameters for the route computation.

1.1.2 SwissMetro

SwissMetro [6] is an idea of a high-speed transport system in Switzerland. It consists of a similar concept to the Hyperloop, because it uses partial vacuum tunnels and magnetic levitation to achieve high speed. In contrast to the Hyperloop it was planned completely subterraneous and uses trains that can transport up to 400 people and operate with a frequency of 6 minutes at rush hours [8]. This design is different to Hyperloop Alpha [1], where the pods can operate every 30 seconds at rush hours, but carry only 28 people. Comparing these two design ideas, the rush hour throughput of Hyperloop is a bit smaller than the SwissMetro rush hour throughput. However, the speed of SwissMetro trains was planned to reach about 500 km/h [8], which is only half as high as the planned speed for Hyperloop pods.

A proposed network design from the main study of SwissMetro [9] is shown in Figure 1.3. It consists of a main route from Geneva to St. Gallen with intermediate stops in Lausanne, Bern and Zurich, and a second main route from Basel to Lugano with intermediate stops in Zurich and Lucerne. Potential extensions in Switzerland and to Italy, Germany and France are included as well. The SwissMetro network it is different to our design since it operates completely subterraneous, whereas we are mainly considering above ground routes, which are cheaper. However, the proposed SwissMetro network connects the same destinations in Switzerland as our precomputed Hyperloop network does.

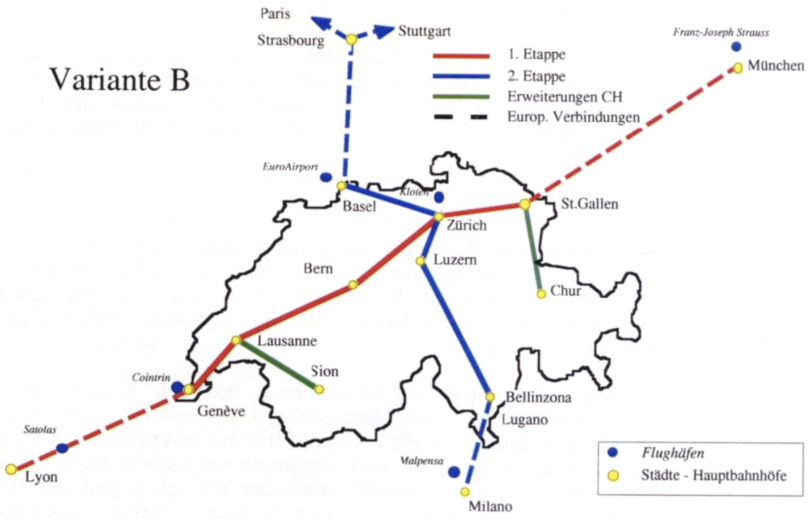


Figure 1.3: The proposed network from the main study of SwissMetro.

1.1.3 Existing Transportation Systems

In order to design Hyperloop routes, the curves of the routes have to be constrained in their radii. For example, when a Hyperloop pod reaches a speed of 1220 km/h, a minimum curve radius of 4.8 km is required [1]. A smaller radius would result in too large of a lateral force, which would be uncomfortable for the passengers.

For planes and high speed trains, the curve radius is important as well. However, planes can mostly fly on straight routes and do not depend on fixed constructions. High speed train networks are similar to the Hyperloop, since they are also restricted to curve radii [10]. One of the fastest high speed trains at the moment is the Shanghai-Hangzou Maglev Line [11], which has a peak speed of 431 km/h and an average speed of 268 km/h. The required curve radius for the peak speed is 2.8-3.5 km, and for the average speed between 2.1-2.8 km [10]. Compared to the Hyperloop, this results in a smaller curve radius restriction. Also, high-speed trains often connect large distance cities and pass through non-occupied land. In Switzerland, there currently exist no high speed trains.

The curve radius constraint also exists for all other transport systems, but it has less impact during the design. For example, trains and cars operate at lower speed, therefore the curve radii can be smaller. For underground train systems it is easier to design straight routes, because is often the cheapest option to keep the tunnel distance short.

Another difference is that the Hyperloop operates in a vacuum tunnel. It can be built either in tubes above ground or underground. Some underground transportation systems, for example metros, already exist, but so far there is

no system that operates in a tube above ground. Compared to existing train and road networks, we want to avoid occupying expensive land, and instead try to plan the tubes mostly above streets or railroads, which are already used for transportation.

1.1.4 Pathfinding

In this thesis we do pathfinding on a precomputed bitmap consisting of existing rails and motorways. Pathfinding on bitmaps is often used and well-researched for autonomous systems [12], especially for object avoidance. However, we have not found literature for automatic route generation with bitmaps where large radii need to be preserved and the speed may depend on arrival direction. We use the A-Star algorithm [13] to do pathfinding and developed a customized heuristic function in order to improve the run-time performance.

Chapter 2

Design

The aim of this project is to develop an algorithm that automatically computes routes between user defined start and end destinations in terms of minimizing cost per time. We therefore use the proposal from Hyperloop Alpha [1] to reduce cost by building the routes above railroads and motorways. In order to reduce time, we try to build the routes as straight as possible, which is required to allow Hyperloop operation at high speed.

In this chapter we first give an overview of the computation of time and cost of routes. Then, we describe three approaches for the route computing. The first, naive approach uses the current traffic routes and then tries to straighten the routes at sections, where high curvature occurs. The second approach is similar, but allows switching between both, rail and street routes. In the third and final approach we bypass shortcomings of the first two approaches, by minimizing cost per time during the pathfinding. A precomputed bitmap with marked motorways and railroads is built and then A-Star is used to compute routes. In the end we provide a detailed explanation of two post-processing steps, that are applied to all computed routes in the same way.

2.1 Time and Cost Computation

2.1.1 Time

The overall travel time on a route, is determined by computing the maximum speed for equidistant points on the route. Based on these speeds and the corresponding forward acceleration a_{fwd}, which is the acceleration in the direction of travel, the time between two points with speeds v_0 and v_1 is calculated as:

$$time = \frac{v_0 - v_1}{a_{fwd}}$$

The speed of travel is restricted by three factors:

1. **Maximum Speed:** The speed must always be smaller than a predefined maximum possible speed v_{max}.
2. **Lateral Acceleration:** The lateral acceleration must always be smaller than a predefined maximum lateral acceleration $a_{lat,max}$. We define the maximum possible speed only considering lateral acceleration as v_{lat}.
3. **Forward Acceleration:** The forward acceleration must always be smaller than a predefined maximum forward acceleration $a_{fwd,max}$. We define the maximum possible speed only considering forward acceleration as v_{fwd}.

Considering these restrictions, the speed at each point on the route is defined as the minimum:

$$v = min(v_{max}, v_{lat}, v_{fwd})$$

While the first restriction is a simple parameter definition of v_{max}, the second and third restrictions require more computation and are further described below.

Lateral Acceleration

For passenger comfort it is important to keep the lateral acceleration below a certain threshold. High lateral acceleration may occur when the pods pass a curve at high velocity. One method to reduce the lateral acceleration is to tilt the pod inside the tube during the curve, transforming it to gravitational force that we assume to impose less discomfort to passengers.

The lateral acceleration a_{lat} at a certain point on the route can be computed with the speed v, tilting angle α, curve radius r and the gravitational constant g:

$$a_{lat} = \frac{v^2 \cdot cos(\alpha)}{r} - g \cdot sin(\alpha) \quad ([14])$$

The maximum speed v_{lat} that the pod can have in a curve, without exceeding the maximum lateral acceleration $a_{lat,max}$ is therefore:

$$v_{lat} = \sqrt{\frac{(a_{lat,max} + g \cdot sin(\alpha)) \cdot r}{cos(\alpha)}}$$

The minimum radius of the curve is the inverse of the curvature which is computed with the first and second derivative of the points on the route:

$$curvature = \frac{\partial x \partial^2 y - \partial y \partial^2 x}{(\partial x^2 + \partial y^2)^{\frac{3}{2}}} = \frac{1}{radius}$$

Forward Acceleration

Similar to the lateral acceleration, passengers may experience discomfort if the forward velocity it too high. Thus, the maximum forward acceleration further restricts the time and distance we need, in order to increase or decrease the speed of a pod. The maximum possible speed at a certain point on the route depends on the starting speed v_0 and the distance s between the starting point and current point:

$$v_{fwd} = \sqrt{v_0^2 + a_{fwd,max} \cdot s \cdot 2}$$

2.1.2 Cost

The amount of tunnels that have to be built makes up the greatest composition of the costs. Using the overall tunnel length and the total length of the route, we can compute the tunnel cost and tube cost:

$$\begin{aligned} cost(tunnel) &= length(tunnel) \cdot cost_{tunnel/km} \\ cost(tube) &= (length(route) - length(tunnel)) \\ &\quad \cdot (cost_{tube/km} + cost_{pylon/km}) \end{aligned}$$

The tunnel and tube cost per km are defined as $cost_{tunnel/km}$ and $cost_{tube/km}$. For the tubes we must add additional cost for the pylons that are built every 30 meters to carry the tubes. The pylon cost $cost_{pylon/km}$ is stored per km as well.

Additionally, the overall cost for stations and capsules are computed. The cost for stations is the same for every route. However, the capsule cost depends on the frequency of the pods, since we must know how many pods operate on a single route. As pod scheduling lies outside the scope of this thesis, we assume a constant number of capsules per km $capsule_{km}$ for the following computations:

$$\begin{aligned} cost(station) &= 2 \cdot cost_{station}, \\ cost(capsule) &= cost_{capsule} \cdot capsule_{km} \cdot length(route) \end{aligned}$$

The total cost of the route is the sum:

$$cost(route) = cost(tunnel) + cost(tube) + cost(station) + cost(capsule)$$

2.2 Rail Routes

We try to compute routes by following the existing train route. This enables us to build a path completely above ground, except when the existing railroad path passes through a tunnel. But since the rails are mostly built above ground, this approach keeps the construction of tunnels to a minimum, which concludes a low cost of the route.

Because we also want to minimize time, it is important that the route does not contain tight curves. This is not always the case for existing rail routes and we therefore implement a straightening algorithm that has the purpose to straighten sections of the route, where high curvature occurs.

After the straightening, the route may pass through residential areas or cross mountains. Residential areas must be passed underground, since most people would complain about Hyperloop tubes being constructed above their houses. To pass mountains, tunnels must be constructed as well, since the slope of the route is constrained by a maximum allowed vertical acceleration. The vertical acceleration depends on the speed of the pod and slope of the elevation on the route. Too high of a slope and high speed result in too high of a vertical acceleration. In most cases the cost is increased after the straightening, because more tunnels are added.

In the following subsections the route computation, the straightening algorithm and the tunnel computation is explained in detail.

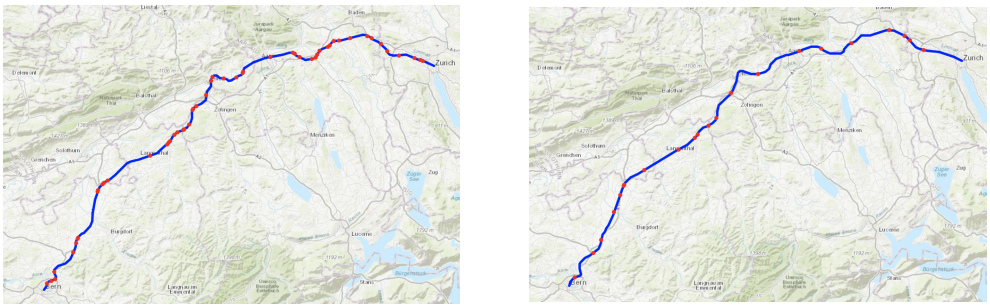
2.2.1 Route Computation

The train route is represented as a list of coordinates on the route. By connecting the points, a path that closely follows the rails is built. However, only approximating the path with straight lines between subsequent points is not a good solution, because we are interested in a smooth, continuous path with low curvature, such that the pods can achieve high velocity.

A common method to fit smooth lines through data points, is the cubic spline fitting [15] through data points. This is done by fitting piecewise third-order polynomials through the points. Since cubic splines have the property that the first and second derivative is continuous, it is guaranteed, that the splines are smooth at each point. After evaluating equidistant points on the fitted cubic splines, the cost and time of the route can be computed according to Section 2.1.

2.2.2 Route Straightening

In order to reduce travel time, sections of the route with high curvature are straightened. Since pods have to slow down in tight curves, we know that the



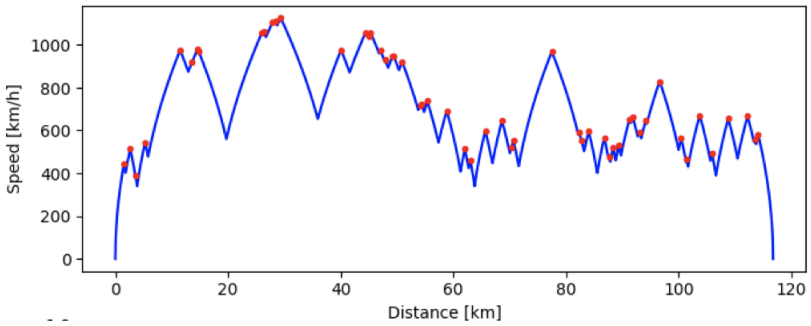
(a) Original route. (b) Straightened route.

Figure 2.1: Illustration of the result of the straightening algorithm.

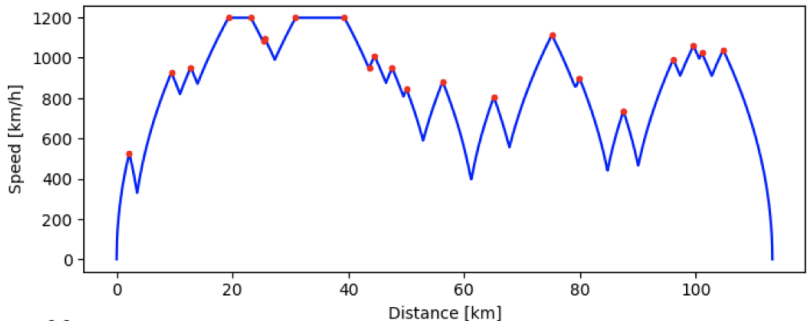
local minima of the speed of the pod may represent curves that we want to straighten. Given the speed on equidistant points on the route, we compute the local maxima of the speeds. To do the straightening, the splines are refitted only through points with a local maxima of speed, such that the local minima are discarded. This step can be applied several times, until the desired amount of curves is straightened.

Figure 2.1 shows the results of the straightening algorithm on the route from Bern to Zurich. On the left hand side we show the original route and on the right hand side the smoothed route is illustrated. The corresponding speed plots with maxima respresented as red dots are shown in Figure 2.2.

By comparing the plots, one can see that several curves are straightened after the smoothing and a higher overall velocity is achieved. Especially the local minima at distance of about 20 km and 40 km are smoothened, which allows the pods to reach the maximum speed of 1200 km/h in this section of the route. However, the algorithm does not always behave as we intend at sections with several consecutive curves. For example at distance 60 km, a local minimum of about 400 km/h remains, because this point was a maximum before the straightening was applied. As we fit the splines through all maxima, this point can not be straightened by doing the straightening only once. It is not always clear how many times the smoothing must be applied to a route and the perfect amount of smoothing may not be the same for all segments of a route.



(a) Speed on original route.



(b) Speed on the straightened route.

Figure 2.2: Illustration of the result of the straightening algorithm.

2.2.3 Tunnel Evaluation

Tunnels are required when the route passes either mountains or residential areas. These restrictions are due to the incapability of the pods to overcome mountains and the missing land ownership.

To know whether it crosses mountains, the slope of the route is computed. As soon as it is higher than 6%, a tunnel must be planned. The threshold of 6% is adopted from Hyperloop Alpha [1], where it is defined as the maximum slope a Hyperloop pod can pass.

To find intersections of the route with residential areas, we intersect the routes with residential area polygons. As soon as it intersects with a residential area, a tunnel is planned. This is also important when residential areas lie next to railroads. In this case, even though the route might only slightly deviate from the rails, a tunnel is necessary.

2.3 Combination of Rail and Motorway Routes

The routes of the previous section are restricted to rail routes. It might occur that motorways, that might fit better as Hyperloop route for certain areas, are missed. We therefore show a possibility to generate routes that combine rails and motorways.

The tunnel computation and straightening algorithm is identical to the previous Sections 2.2.2 and 2.2.3 and therefore skipped. The computation of the route is explained in the next section.

2.3.1 Route Computation

We build a graph containing all motorways and railroads of a given area, for instance Switzerland, and then find the routes by computing the shortest path between two destinations with the Dijkstra algorithm [16].

Nodes in the graph represent intersections of two rails or two motorways. To enable the route to switch between railroads and streets, nodes are also added at intersections of streets and railroads. Two nodes are connected with an edge, if there exists a direct path between them, and the weight of the edge represents the length of the connecting path.

A drawback of this approach is that we might miss potential intersection points of close streets and railroads, when they operate parallel to each other, but never intersect.

2.4 Route Finding on Bitmap

The route finding on a bitmap addresses the shortcomings of the route straightening algorithm from Section 2.2.2. By computing the cost and time during the pathfinding, tunnels can automatically be added at sections with high curvature, and therefore straightening is not necessary anymore.

The pathfinding applies the A-Star algorithm [13] on a bitmap file that contains motorways and railroads. The resulting route on the bitmap file consists of pixel coordinates and is curvy, when it follows each pixel exactly. As we are interested in straight routes, we use a pixel smoothing method, that can be applied to a list of pixels.

In the following, it is first explained how the bitmap is built, and then how pixel smoothing is achieved. After that the A-Star algorithm on the bitmap is explained in detail.

2.4.1 Bitmap

The bitmap file is a binary map of a certain region, stored in a sparse matrix. All motorways and rail tracks are represented as ones, and the remaining parts as zeros. As soon as a path crosses a pixel with a value equal to zero, a tunnel has to be built. As long as the path follows only pixels with values equal to one, the route is built above the railroads or streets.

The bitmap file can contain data from any region in the world and the resolution of the path is dependent on the pixel size.

2.4.2 Pixel Smoothing

Because we do pathfinding on a bitmap file, the output path is a list of pixels, which results in a curvy path when it goes through each pixel.

To reduce curvature, we process the path. The path must follow the pixels, but still be able to approximate the path up to some degree. P. Diearckx [17] proposed a method to do curve fitting with B-Splines and added a smoothing factor s, that defines how much it is allowed to diverge from the pixels. The defintion of s is:

$$s \leq \sum (w \cdot (y - g))^2$$

where *g(x)* are the smoothed interpolation points and *y(x)* are the pixel points. Weights w can be defined for each pixel separately. We want to treat the smoothing of each pixel with the same importance and therefore set all weights w equal to 1. By applying this to the formula, w gets canceled and we obtain:

$$s \leq \sum (y - g)^2$$

In order to find the right smoothing factor, the desired squared deviation per pixel has to be multiplied with the number of pixel points.

To fit splines through the points, we use a parametric approach, because the x or y values of the pixels may not be strictly increasing. We define an additional variable z_i for each pixel i that is defined as the length of the route from the starting pixel p_0 until the pixel p_i:

$$z_i = length(route(p_0, p_i))$$

The z values are strictly increasing, as the distance between two subsequent points is always greater than zero.

2.4.3 A-Star Algorithm

In order to find Hyperloop routes between a start and a target destination on the bitmap, we use the A-Star Algorithm [13].

Algorithm 1 A Star

```
1: init open_list = {start_node}
2: init closed_list = {}
3: while open_list not empty do
4:     current_node = open_list.get()
5:     if current_node is goal_node: then
6:         path found, start backtracking!
7:     closed_list.put(current_node)
8:     for child in current_node.children do
9:         if child not in closed_list then
10:            open_list.put(child, child.cost)
```

Algorithm 1 shows the pseudo code of the A-Star algorithm. The search starts at the start node and the cost of its neighbor nodes are computed. The neighbor nodes are then added into an open list with their corresponding cost. Next, the node with the lowest cost in the open list is expanded by computing its neighbors and adding them to the open list as well. This is repeated, until the expanded node is the target node. In order to make sure that each node gets expanded at most once, every expanded node is added to a closed list.

The cost function from the start node to the current node is computed incrementally by dynamic programming. The computational complexity of the cost computation is therefore for each node the same and does not depend on the distance to the start node.

In contrast to the Dijkstra Algorithm [16], A-Star uses an additional heuristic cost function, which has the purpose to guide the search into the direction of the goal. This has the benefit that less nodes are expanded and the computational complexity of the algorithm can therefore be further reduced.

The cost function f of a node v is computed as the sum of the cost $g(v)$ to reach v and the heuristic function $h(v)$, which is an underestimation of the cost to get from the current node v to the destination node. In order to minimize cost per time, we compute both a cost function f_{cost} and a time function f_{time}. They are computed similar to the function f:

$$f_{cost}(v) = g_{cost}(v) + h_{cost}(v), \tag{2.2}$$

$$f_{time}(v) = g_{time}(v) + h_{time}(v) \tag{2.3}$$

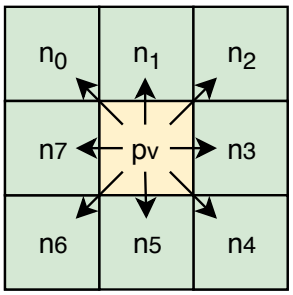


Figure 2.3: The current pixel p_v is shown in yellow and the neighbor pixels are shown in green.

By adding a constant t_{train}, that represents the travel time for the route by train, we combine the two cost functions to a new cost function f that minimizes the cost while maximizing the time that we are faster than the train:

$$f = \frac{f_{cost}}{(t_{train} - (f_{time}))}$$

In the following sections it is explained, how the neighbors of nodes are defined and how the g and h cost functions are computed.

Neighbor Definition

We assume that we are currently at pixel p_v and want to expand the current path to the neighbor pixels. In general, it is allowed to move to any pixel, even to go backwards. This is because there are no obstacles in our map, just more and less expensive pixels. In Figure 2.3 the direct neighbors of pixel p_v are shown.

Going backwards does not seem reasonable in this scenario, as we are interested in a path that has a low curvature, such that a high speed can be achieved. A change of direction of more than 45 degrees would result in more curvature, which is not a result we are looking for. To improve performance, the algorithm expands the pixel to only three neighbors instead of eight. These neighbors are dependent on the predecessor p_{v-1} of the node p_v and change the direction from p_{v-1} to p_v at most by 45 degrees. In Figure 2.4 the neighbor pixels are illustrated for both, a vertical and a diagonal predecessor pixel.

Actual Cost and Time

To compute cost and time from the start node to the current node, we use the methods described in the previous Section 2.1. However, for performance reasons, storing the whole pixel path from the start node to the current node and recomputing the cost functions for each neighbor must be avoided. Since

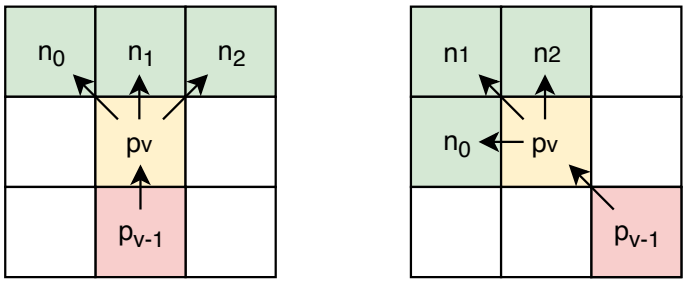


Figure 2.4: The current pixel p_v is shown in yellow and its predecessor pixel p_{v-1} in red. The neighbor pixels following the direction of the predecessor are shown in green.

the curvature and therefore the attainable speed is dependent on the previous nodes, parts of the previous path must be stored and recomputed.

In a node v_i, the previous $t + c$ pixels, and the cost and time at v_{i-t} are stored. The cost and time of the proceeding node v_{i+1} are then computed as:

$$g_{cost}(v_{i+1}) = cost(v_{i-t}) + cost(v_{i-t}, v_{i+1})$$
$$g_{time}(v_{i+1}) = time(v_{i-t}) + time(v_{i-t}, v_{i+1})$$

The path from v_{i-t} to v_{i+1} is therefore fully recomputed for each node, whereas the cost until the node v_{i-t} is stored. The constant c is added, such that the splines are not refitted starting from the node v_{i-t}, but from the node $v_{i-(t+c)}$ to avoid boundary errors.

Heuristic of Cost and Time

The heuristic functions estimate the cost and time from the current node to the destination node. First, it is important that the heuristic function is an underestimation, since otherwise the optimal solution may not be found. Second, it should also be a good approximation such that the search space can be reduced and a good run-time performance is achieved. The computation of the heuristic cost and time functions are explained in the following.

Heuristic Cost: To compute the heuristic function of the cost, first the Euclidean distance between the current node and the destination node is computed. This distance is always smaller or equal to the optimal route, since there is no shorter way. Second, it is assumed that no tunnels are built. This is an underestimation as well, since we assume tunnels to be more expensive than tubes.

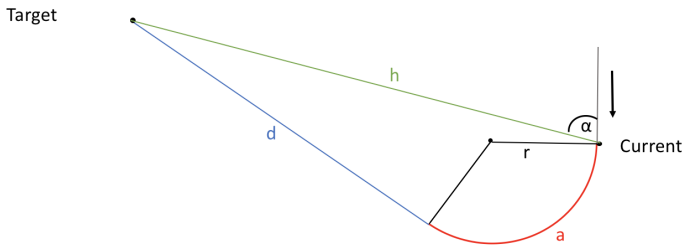


Figure 2.5: Illustration of the heuristic function of travel time.

The heuristic cost h_{cost} of a node v_i with destination v_{dest} is summarized as follows:

$$h_{cost} = eucl\_distance(v_i, v_{dest}) \cdot (cost_{tube/km} + cost_{pylon/km})$$

Heuristic Time: The heuristic function of the time not only depends on the current and the target node, but also on the arrival direction, meaning the angle α between the previous, current and destination node. An illustration is shown in Figure 2.5. The Euclidean distance from the current node to the target node is marked in green. While it denotes the shortest distance, the pod would have to slow down to nearly zero, in order to make such a tight curve without exceeding the maximum lateral acceleration. A more realistic path is shown in red and blue. The pod first makes a curve with constant radius r on the red arc, afterwards it continues straight on the blue line d. The idea is, that the pod has a constant speed on the arc, and then accelerates to maximum speed on the straight path d.

The radius r is dependent on the angle α. For a large α the radius can be large too, since the pod can almost continue straight. On the other hand the radius must be smaller for a large change of direction. The speed on the arc a changes with the radius. Larger radii enable travel in higher speed.

Since the heuristic function must be an underestimation, we compute the constant speed a pod should have in the red curve, such that the overall time to the destination is minimized. We precomputed this function for every angle and assumed a constant, large overall distance h to the goal.

Figure 2.6 shows the results of the precomputation. One can see the optimal speed that a pod should have on the arc a, for every angle α. The optimal speed increases with the angle, which is because for a larger α the pod must change the direction less and can therefore reach a higher speed. In this example we defined the maximum possible speed of Hyperloop pods as 333 m/s, which is why the speed stops increasing at an angle α of about 70 degrees and stays constant at 333 m/s.

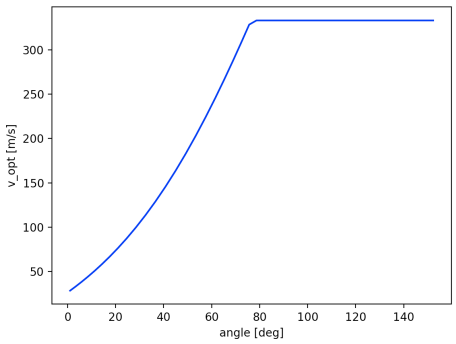


Figure 2.6: Illustration of the optimal constant speed a pod should have in a curve, such that the total time to the destination is minimized.

The heuristic function is then computed by assuming the constant precomputed speed from Figure 2.6 during the arc a, and then afterwards the pod can accelerate to maximum speed until it reaches the target. The radius of the arc is defined as the smallest possible radius the curve can have given the speed on the arc and the lateral acceleration limit that must not be exceeded.

2.5 Route Post-processing

After a route is computed, we apply two post-processing steps. The first step addresses the fact that we want the passengers to have a comfortable travel experience, and applies acceleration smoothing. Second, we want to plan realistic tunnel sections. Without post-processing the tunnels may consist of small tunnel sections that are close to each other, even though in this case it is more realistic to combine the small sections to one, longer tunnel. In the next two sections, the methods are explained in detail.

2.5.1 Acceleration Smoothing

By computing the maximum possible speed on a route regarding lateral and forward acceleration, we enable instant switching between maximum acceleration and maximum deceleration. Although this does not exceed the maximum or minimum allowed accelerations, instant switching of the forward acceleration will result in discomfort for passengers because of sudden braking and accelerating.

To assure a more comfortable experience for passengers, post-processing on the acceleration is necessary. This is achieved by only allowing continuous acceleration and deceleration with a predefined constant slope. This not only avoids abrupt changes in acceleration, but also ensures a smooth speed curve without sharp peaks. We show an example in Figure 2.7. On the left hand side, the for-

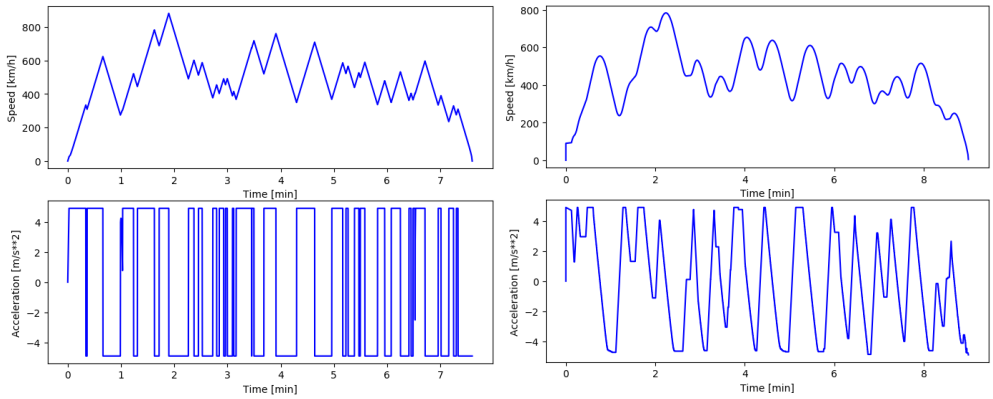


Figure 2.7: Illustration of the acceleration smoothing.

ward acceleration and speed are illustrated before the acceleration smoothing. We can see that there are several peaks in the speed plot and the corresponding forward acceleration mostly alternates between maximum acceleration and maximum deceleration. On the right hand side we show the plots after the acceleration smoothing. The speed plot is a lot smoother and the acceleration is continuous with a constant slope.

2.5.2 Tunnels

Whether a tunnel is built is defined either by the bitmap entries, or by checking the slope of the elevation and intersections with residential areas. However, it can happen that the path has several small tunnel segments with a few non-tunnel segments in between. In this case it is better to build one, longer tunnel instead of several short tunnels. To achieve this, we post-process the tunnels and combine tunnels that are closer than a predefined distance.

It is important to consider to add some additional distance at the beginning and the end of each tunnel. To see this, imagine that on the route there is a residential area with several buildings. In order to not exceed the maximum allowed vertical acceleration, we have to start building the tunnel before, such that by arriving at the residential area, the tunnel is deep enough to pass the area underground. The amount of distance we have to add is dependent on the speed and the maximum allowed vertical acceleration. This is computed similar to the lateral acceleration, that was explained in Section 2.1.1.

Because the postprocessing modifies the calculated tunnels from the pathfinding, it is possible that the tunnels contain unnecessary curves. In order to straighten the tunnels, we refit the splines through all non-tunnel pixels, such that the splines pass as straight as possible through tunnel-sections.

In Figure 2.8 we show an example of the tunnel post-processing. In the left

subfigure the route contains several small tunnel segments. In the middle, these segments are combined to one larger segment and in the right picture, the tunnel gets straightened, to allow a higher speed.

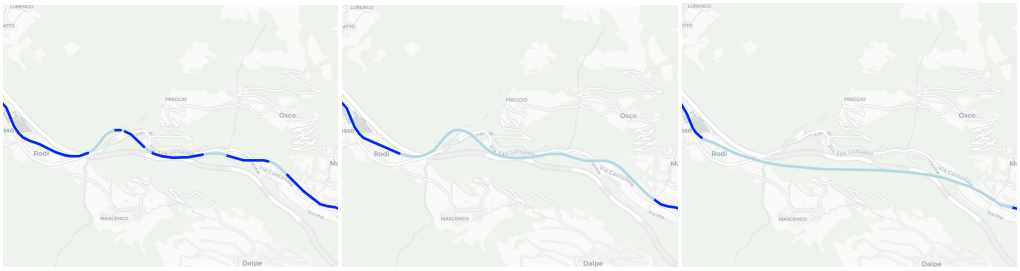


Figure 2.8: Illustration of the tunnel post-processing.

Chapter 3

Implementation

We implemented the rail routes, the combination of rails and motorways, and pathfinding on a bitmap in Python. In the following sections we provide details about the implementation and explain the data we used for each approach. After that, an overview of the website is shown.

3.1 Rail Routes

An overview of the implementation where the route follows the existing rail path is given in Figure 3.1. In the main method, the route is accessed from the Google Directions API [18]. After that, the splines are fitted and speed, acceleration and time is computed in the script *route_computation*. In order to compute the cost, the script *tunnel_check* is accessed, to know where we have to take account of tunnels. Finally, the acceleration smoothing is done as post-processing step.

The data of the route and the computation of the tunnels is further described in the next two subsections.

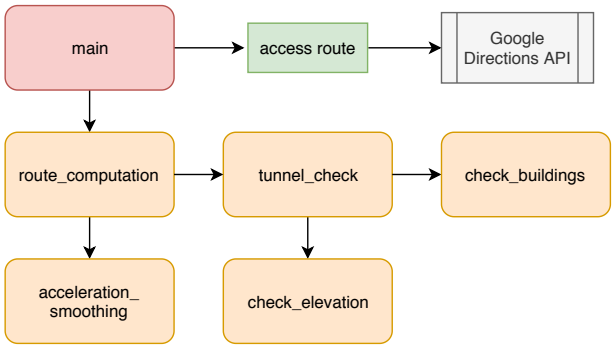


Figure 3.1: Implementation overview.

Figure 3.2: Example of a route section, where small deviation of the points leads to unnecessary curvature. The blue line shows the unprocessed route and the red line illustrates the route after discarding points.

3.1.1 Route Data

The existing transit routes are accessed from the Google Directions API [18]. The data is represented as a polyline which contains lists of coordinate points on the route. To fit the cubic splines through the data points we use the Python library SciPy [19].

The data points sometimes diverge minimally from the rails, which leads to a higher curvature of the route. An example is shown in Figure 3.2. The blue curve is the unprocessed route. Although the path passes rather straight, the route contains unnecessary curves. By fitting splines only through points that are at about 150 meters apart, and discarding the rest of the points, we achieve a smoother resulting path, that still follows the rails or streets closely, as illustrated with the red route in Figure 3.2. To compute the new polyline we use a simple method. Starting from the first point, we discard every subsequent point that is closer than 150 meters. As soon as a point is further away than 150 meters it is added to the new polyline. This is repeated until the end of the polyline is reached.

3.1.2 Tunnel Data

In order to check whether we pass mountains, meaning the slope of the elevation is too large, we use the digital elevation model from the German Aerospace Center [20]. It consists of elevation data built by satellites and is available for the whole world. It has a pixel size of 90 meters.

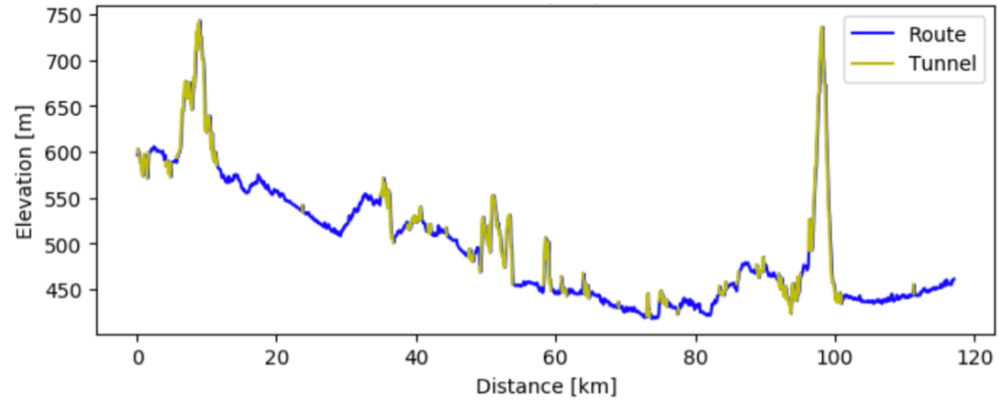


Figure 3.3: Illustration of the elevation.

The elevation is accessed for equidistant points on the route, which are 120 meters apart, and then the slope between subsequent points is computed. Since the model was built with satellite data, which means that it also considers the height of buildings, bridges, etc., the resulting slope can sometimes be higher or lower than in reality. This can result in a wrong amount of planned tunnels on a route. We did not find data about the slope on railroads, which is why we were not able to find out, how much our computation deviates from the real slope on the routes.

In Figure 3.3 we show an example of the elevation on the route from Bern to Zurich. Tunnels are marked in yellow and they are built, as soon as the slope of the elevation is higher than 6%, which is the maximum possible slope for Hyperloop routes [1]. There are two mountains, one close to the start and one towards the end of the route, where our computation correctly identifies that tunnels must be built. Also in the middle of the route, a few hills are existent and the tunnels are correctly computed. However, there are a few sections, where individual tunnel points were computed. For these cases it is not clear whether tunnels are really necessary.

Intersections of the route with residential areas are checked with the open source dataset of OpenStreetMap (OSM) [21], that contains residential area polygons. A limitation of the dataset is that sometimes, when residential areas are close to railroads, the polygon includes railroads. An example is shown in Figure 3.4, on the left hand side. The residential area is marked in orange, and the Hyperloop route, which passes along the railroads, is shown in blue. In this case, the intersection with the residential area polygon would result in false positives, as no tunnel must be built, when the route follows railroads. To avoid erroneous tunnels, we decided to also check, whether the Hyperloop route is close to railroads, and in this case, no tunnel is built. We therefore access the railroads data set from OSM. On the right hand side in Figure 3.4, the railroad dataset is

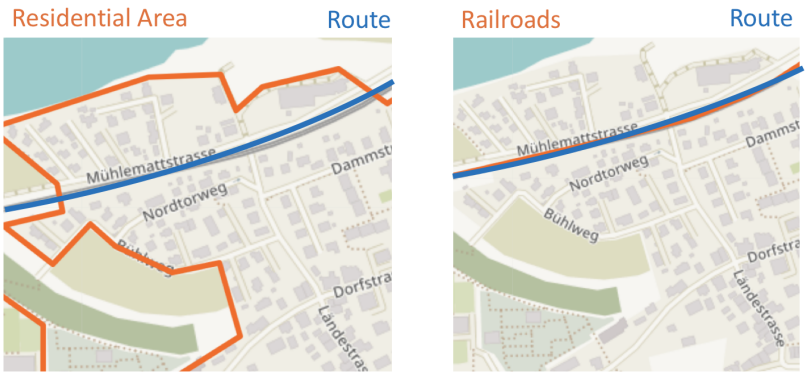


Figure 3.4: Illustration of the dataset of residential areas and railroads of OSM.

illustrated in orange, and since it is close to the blue Hyperloop route, no tunnel is constructed in this example.

In order to find polygon intersections we use RTrees [22], a hierarchical data structure. The polygons are stored as bounding rectangles and grouped in a tree. Intersection queries can then be answered efficiently by traversing the tree starting from the root. We use Rtree [23], a Python library for the implementation of RTrees. To be able to construct Polygons and LineStrings from the OSM dataset we use Shapely [24], a Python package for manipulation and analysis of planar geometric objects.

3.2 Combination of Rail and Motorway Routes

The implementation of this section is the same as in the previous, with exception of the route finding, for which we use routing on graphs instead of accessing the existing rail paths. We therefore skip the implementation overview and explain the graph construction.

3.2.1 Graph Construction

For the graph construction we use the railroads and motorways dataset from OSM [21]. In order to construct the graph, we use the library OSMnx [25], which provides functionality to automatically transform OSM data into graphs. It is built on top of NetworkX [26], a Python package to build and study complex networks. After the graph is constructed, the edge lengths can be added automatically and the functionality to compute the shortest paths on the graph is provided by NetworkX.

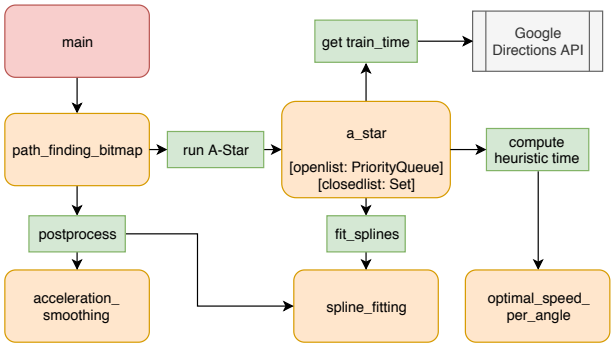


Figure 3.5: Implementation overview.

3.3 Route Finding on Bitmap

A simplified overview of the scripts we use for the route finding on a bitmap is given in Figure 3.5. The main function takes as input the smoothing parameter, the name of the bitmap file, the start and the end destination. It then calls *a_star* to apply the pathfinding on the bitmap. The *a_star* script accesses the scripts *spline_fitting* and *optimal_speed_per_angle* to compute the partial routes during the search. When the result is found, *a_star* does the backtracking and returns the pixel points of the path. *path_finding_bitmap* then starts the post-processing of the route which includes acceleration smoothing and refitting the splines.

In the next sections we explain the generation of the bitmap and the implementation of the A-Star algorithm further.

3.3.1 Bitmap Data

To generate the bitmap, it was an important decision which dataset to use. OSM [21] offers a large set of open source geodata, and is available for the whole world. Even though there exist also smaller datasets for certain areas with better resolution, we decided to go with the OSM data, such that it is straightforward to extend the route generation to any part of the world.

The data is downloaded either per country or continent. However, it is only possible to download unfiltered data for large areas, which we are interested in. After the download, the data is processed with osmctools [27]. First it is cropped with osmfilter, to extract streets and motorways without tunnels, then it is converted into GeoJson format. The streets and motorways are represented as LineStrings, which are a collection of coordinates that form a path. We transform the LineStrings into a bitmap with the library rasterio [28]. We choose the pixel size as 10 meters, and every pixel that intersects with a data LineString is set to one. Since the LineStrings do not contain data about how wide the streets

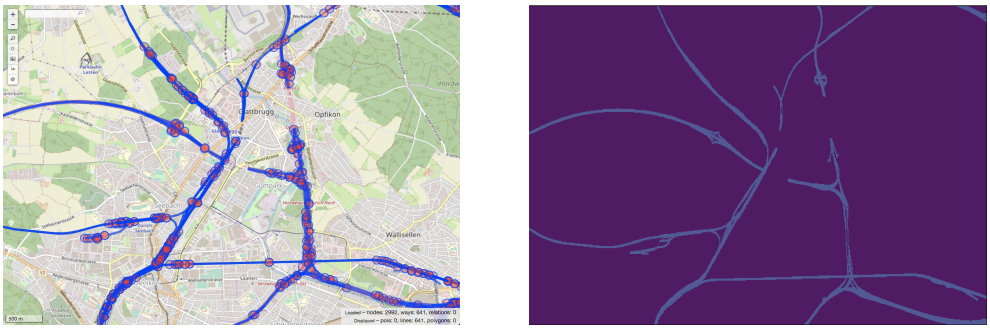


Figure 3.6: Illustration of the data in OSM (left) and as Bitmap (right).

and rails are, we mark all neighbors of marked pixels as well. This reduces the probability of missing parts of motorways or railroads.

We observed that after the A-Star search individual points on the route sometimes minimally diverge from the rail and motorway pixels. In this case, each of this pixels would imply a tunnel section, and therefore potentially too many tunnels would be computed. To avoid this, we use a second bitmap file for the route post-processing. On the second bitmap each neighbor of marked pixels is marked as well, such that wrong tunnels sections are avoided.

In Figure 3.6 on the left hand side an example of the LineString data of OSM is shown, and on the right hand side the corresponding bitmap file is illustrated.

At this point, it is assumed that Hyperloop pods can travel above existing motorways and railroads without tunnels. However, this may not be the case if the slope of the elevation of the paths is too high for Hyperloop pods, but may be still okay for trains or cars, because they operate at lower speed. To address this problem, the digital elevation model of the German Aerospace Center [20] is used again. We filter out all pixels of the bitmap, where the slope in any direction is higher than 6%, as defined in Hyperloop Alpha [1]. Since the pixel size of the elevation data is 90 meters but our bitmap file only 10 meters, we use linear interpolation to approximate the data. However, the result is not satisfying, because too many pixels get filtered out. The reason for this is first of all that we look at the slope in all directions. This means that streets that are close to mountains are always filtered, because in the direction of the mountain the slope is high. But the street may have a low slope, when it passes parallel to the mountain. This results in lots of false positives during the filtering. Second, when for example several skyscrapers are close to a pixel, it can result in a higher computed slope, since satellite data of the model adds the height of the skyscrapers to the elevation. Because of these limitations we decided not to use the elevation model, and assume that the slope of existing streets and rails is not too large for Hyperloop pods.

To save storage, the bitmap files are sparsified, since the content of the streets

and railroads is a small fraction of the whole bitmap file.

3.3.2 A-Star

The open list of the A-Star algorithm is implemented as a priority queue that uses binary heaps. Entries of the priority list contain tuples $(cost, item)$, and by calling $get()$ on the queue, the item with the smallest cost is returned. For the closed list a set is used as data structure, since sets are fast in returning whether a index already exists.

The parameter t_{time}, which represents the travel time of trains and is used to compute the cost function, is accessed from the Google Directions API [18].

3.4 Website

We implemented a website[1] that presents an interactive map with a precomputed Hyperloop network. The routes were computed with the pathfinding on a bitmap and each Hyperloop route was computed twice, with two different cost settings. Additionally, a completely subterraneous, direct Hyperloop route and the existing SBB train route are illustrated to allow comparison.

A visualization of the Hyperloop network is shown in Figure 3.7. The SBB routes are marked in green, the suggested routes from the pathfinding in blue and red and the straight-line underground routes in grey. The tunnel segments are highlighted in lighter color.

Additionally to the map, the website shows the speed and acceleration plots of the pod during the travelling for each route, as well as an overview of the cost, time and distance of the route. The user can interactively adapt the input cost, speed and acceleration parameters, in order to recompute the overall building cost and travel times. This allows comparison of the routes with different parameters settings. An example of the plots and the parameter form is shown in Figure 3.8.

The website is implemented with Flask [29], a Python-based web framework. Additionally, we used the Python package folium [30] to visualize the routes on the map.

[1] http://hyperloop-network.ethz.ch/

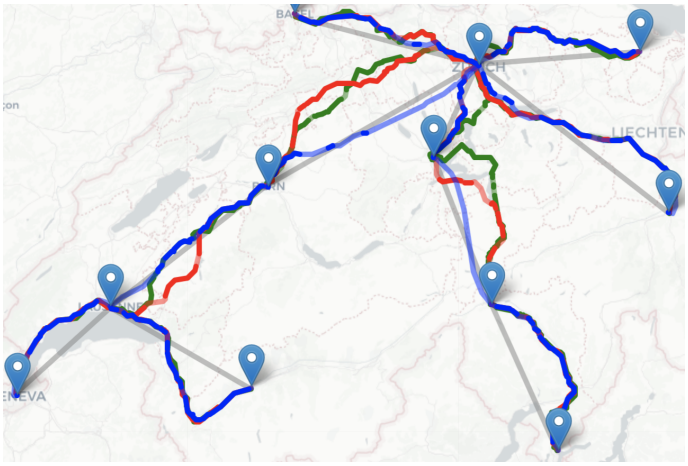
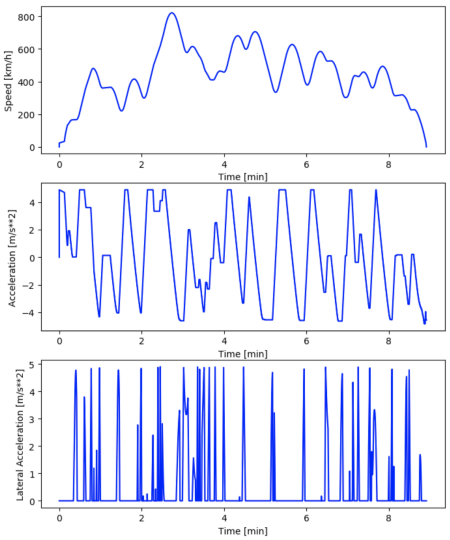


Figure 3.7: Map with the precomputed Hyperloop network in Switzerland.



(a) Example of the speed, forward acceleration and lateral acceleration plots.

Cost Factors [Mio CHF]:

Tube Cost per km:	1.8
Tunnel Cost per km:	31
Pylon Cost per km:	4.8
Capsule Cost:	1.35
Number of Capsules per km:	1
Station Cost:	125

Parameters

Max Forward Acceleration[m/s2]:**	4.9
Max Lateral Acceleration[m/s2]:**	4.9
Max Speed[km/h]:	1200
Max Tilting Angle[deg]:	45

(b) Form to change the cost, speed and acceleration parameters.

Figure 3.8: Illustration of the website.

Chapter 4

Results

In this chapter we present the results of the pathfinding algorithm on a bitmap by computing a Hyperloop network in Switzerland that is formed by computing selected individual routes. In order to evaluate these results, we compare the computed routes with existing SBB train routes and hypothetical straight-line underground routes. A detailed comparison is shown for the route from Zurich to Basel. Subsequently we discuss the performance and the optimality of the routes.

The pathfinding takes several input parameters. We set the pixel size of the bitmap to 10 meters and use a maximum squared deviation of 50 per pixel for the spline fitting. This allows the splines to deviate around 7 meters per pixel. During the execution of the A-Star algorithm, each node stores the last 50 nodes to fit the cubic splines. This is an approximate distance of 500 meters. The maximum accelerations on the passengers and the maximum possible speed are adopted from Hyperloop Alpha [1]. The threshold of the accelerations is therefore set to $4.9 m/s^2$ and the upper bound for the speed is defined as $333 m/s$. The tilting of the pod, which is important to reduce the lateral acceleration in curves, is restricted by a maximum angle of 45°.

In order to analyze the behaviour of the pathfinding with different cost parameters, we compute two solutions per route. In the first run we use the averaged cost parameters from the computation of the Hyperloop route from Los Angeles to San Francisco, given in the Hyperloop Alpha paper [1]. In the second run we reduce the cost of the stations and tunnels according to a recent statement from Elon Musk [31], where he claims that even lower costs for tunnels and stations are feasible. We call the cost parameters of the first run A and the parameters of the second run B. An overview of both settings of cost parameters are shown in Figure 4.1. Note that the tunnel costs and station costs are more than halved in the second run.

In order to compare the Hyperloop routes with the SBB routes, we require the travel time of the trains and an approximation of the construction cost of the SBB routes. Since we were not able to find official information about the average capital cost of SBB routes, we approximated them ourselves. Therefore, we an-

	A	B
Tube Cost per km	1.8 Mio CHF	1.8 Mio CHF
Tunnel Cost per km	31 Mio CHF	15 Mio CHF
Pylon Cost per km	4.8 Mio CHF	4.8 Mio CHF
Station Cost	125 Mio CHF	50 Mio CHF
Capsule Cost	1.35 Mio CHF	1.35 Mio CHF
Number Capsules per km	1	1

Table 4.1: Overview of the cost parameters of the first run (A) and the second run(B).

alyzed the cost of the railway extension project *Bahn 2000* from Wikipedia [32], where a list of newly constructed rail sections with their corresponding cost is provided. However, the cost of the routes within the project vary strongly and we did not find detailed information about the detailed composition of the costs.

The averaged cost per km of tunnel and non-tunnel segments are:

- Tunnel segments: 67,000,000 CHF per km
- Non-tunnel segments: 28,000,000 CHF per km

It is not clear, whether the SBB cost approximation is accurate. Especially since the exact composition of the cost is not known, it is possible that additional cost, like planning cost, station cost or train cost is included. This is the reason why we ignore station and train cost for the computation of the SBB building cost and it would explain why the approximation of the SBB cost is up to four times higher than the Hyperloop cost. A reason for the higher tunnel cost for SBB routes is that the diameter of Hyperloop tunnels is smaller than for train tunnels, which reduces the construction cost.

Table 4.2 shows the building costs, travel times and distances of all computed routes. The Hyperloop routes with the cost parameters from Hyperloop Alpha [32] are shown in the rows *HL-A* and the routes with cheaper tunnels and stations are called *HL-B*.

Comparing these two Hyperloop routes, one can see that the overall length of tunnels on *HL-B* routes is always higher in our computed examples. This should also hold in general, because of the lower construction cost of tunnels for *HL-B* routes. The total distance of the routes with cheaper cost is always shorter or equal to the distance of the route with larger cost, since the tunnels avoid curves and do not make a detour. Also, by comparing the time one can see that the travel time is always shorter or equal on the routes with cheaper cost settings. This is because tunnels imply straighter and shorter routes and therefore, the speed can be increased. The cost is smaller for most *HL-B* routes as well, because

	Route	**Cost** [Mio CHF]	**Time** [Min]	**Total Distance** [Km]	**Tunnel Distance** [Km]
Genf - Lausanne	HL-A	789	8	61	2
	HL-B	797	7	61	24
	HL-Direct	1883/925	4	50	50
	SBB	1684	50	60	0
Lausanne - Bern	HL-A	1604	13	95	24
	HL-B	1057	11	83	35
	HL-Direct	2767/1371	5	77	77
	SBB	2842	72	96	3
Bern - Zurich	HL-A	1761	14	116	23
	HL-B	1565	9	99	80
	HL-Direct	3334/1656	6	94	94
	SBB	4241	56	116	25
Zurich - St.Gallen	HL-A	1137	14	81	9
	HL-B	872	12	76	19
	HL-Direct	2293/1130	4	62	62
	SBB	2677	63	87	5
Zurich - Basel	HL-A	1183	13	84	10
	HL-B	984	11	80	29
	HL-Direct	2659/1316	5	74	74
	SBB	2623	53	88	3
Zurich - Luzern	HL-A	1020	6	45	16
	HL-B	605	6	45	16
	HL-Direct	1553/757	3	40	40
	SBB	1883	45	57	7
Zurich - Chur	HL-A	1809	17	115	26
	HL-B	1441	14	113	52
	HL-Direct	3395/1688	6	96	96
	SBB	3578	74	116	7
Luzern - Airolo	HL-A	2115	10	79	50
	HL-B	1148	4	65	63
	HL-Direct	2298/1133	4	63	63
	SBB	4305	114	104	35
Airolo - Lugano	HL-A	1519	11	82	25
	HL-B	997	11	80	30
	HL-Direct	2353/1160	4	64	64
	SBB	3109	111	95	11
Lausanne - Sion	HL-A	1348	13	92	15
	HL-B	984	11	90	19
	HL-Direct	2363/1166	4	65	65
	SBB	2625	68	92	0

Table 4.2: Comparison of the cost, time and distance of all Hyperloop (HL) routes with the SBB train route. The first suggestion (HL-A) is computed with the more expensive setting of cost parameters and the second suggestion (HL-B) with the cheaper setting. For the underground route (HL-Direct) we show both, the expensive and cheap cost.

smaller cost parameters were used to compute tunnels and stations cost. If the same cost parameters were used, the cost would most likely be higher because of the larger fraction of tunnels.

The cost of the direct underground routes is computed for both cost settings such that we can compare them with both, *HL-A* and *HL-B* routes. The direct route is always faster, shorter and more expensive than the suggested Hyperloop routes. The only exception is the route from Lucerne to Airolo, where the route of the cheap cost setting is similar to the direct route. The reason is that the algorithm computed the best route to be a nearly straight route which mostly consists of tunnels. After applying the tunnel post-processing, the result was one large tunnel, except of a small non-tunnel segment at the beginning. The slightly higher overall distance and the higher cost compared to the direct route are due to a small curvature in the tunnel computed during the spline fitting, in order to attach the route smoothly to the non-tunnel segment.

Even though the SBB routes mainly consist of non-tunnel segments, they are more expensive than all Hyperloop routes except two direct underground routes with cost setting *A*. This means, to build a complete subterraneous Hyperloop network would be cheaper than to build an SBB network, which operates above ground. It is not clear whether this is true, whether the Hyperloop cost declarations are too small, or whether our approximation of the average building costs of SBB is too high. The travel time of SBB trains is on average about 5 times higher than the Hyperloop routes computed by the algorithm.

4.1 Zurich to Basel

In order to provide more information we analyze the routes from Zurich to Basel in more detail. We show in Figure 4.1 the directions of the three Hyperloop routes and the SBB route. The picture in the top shows all routes on top of each other, to be able to see where they diverge. The routes are shown in different colors, and their tunnels are drawn with less opacity. The grey route is the direct underground route and therefore the shortest route, but also the most expensive one. The green route shows the SBB train route. It contains many curves but only two short tunnel sections with an overall distance of 5 km. The red route is the Hyperloop route with the more expensive tunnel cost, *HL-A*. Starting from Zurich in the bottom right, it follows the SBB route until a larger tunnel is computed in the middle of the route to avoid tight curves. After the tunnel, the route follows the motorways and therefore deviates from the SBB route. Intuitively, we believe that the motorways were chosen because they pass straighter into the direction of Basel and contain less curves. In the lowermost picture we show the Hyperloop route with cheaper tunnel cost. By comparing it with the red Hyperloop route, it contains overall more tunnels. This is because it is cheaper to build tunnels and since tunnels can pass straighter, a higher

speed can be reached. In the beginning of the route, starting from Zurich, a tunnel is computed, to switch from the railroads to the streets. Then, two large tunnel sections are constructed, which avoids a sequence of tight curves on the railroads. After that, the route passes similar to the red Hyperloop route, apart from two small tunnel segments in the end of the route.

Overall, this comparison demonstrates that with more expensive tunnel cost, the path follows mainly the motorways and railroads, whereas with cheaper tunnels, tunnels are planned where high curvature occurs. It is therefore possible to include the straightening of the route in the pathfinding algorithm by adapting the tunnel cost. Lower tunnel cost results in straighter Hyperloop routes.

4.2 Run-Time Performance

The run-time performance of the A-Star algorithm depends on the distance between the start and target destinations. For one of the shortest routes, Geneva to Lausanne, it took the A-Star implementation about 18 hours to finish, whereas for longer routes the computations took several days. Since the number of expanded nodes is proportional to the run-time performance, we show in Figure 4.2 the area of expanded nodes during the search of the route from Geneva to Lausanne. The yellow area represents the expanded nodes, and the red circles show the location of Geneva and Lausanne. The route was computed starting from Geneva, in the lower left. Although the heuristic function guides the search into the direction of Lausanne in the upper right, one can see that still a large area is explored. The total sum of the expanded nodes is 10.35 Mio. pixels, which corresponds to about 160 nodes that get expanded per second.

The search space grows exponentially with the route cost. The reason for this is that, with each step on the optimal route towards the goal, real cost is added. However, the cost of nodes that are closer to the start node, consists of more heuristic cost, which is an underestimation of the real cost. This is why the probability is high that the nodes at the border of the search space may have lower overall cost and get expanded first.

However, due to the heuristic function of the time, which adds a penalty when the path goes in the wrong direction, the search space lies mostly between the start and end destination, and only a small part is explored in the backward direction.

4.3 Route Optimality

There are several reasons, why the final route is not always optimal in terms of minimizing cost per time. The limitations of the cost and time computations,

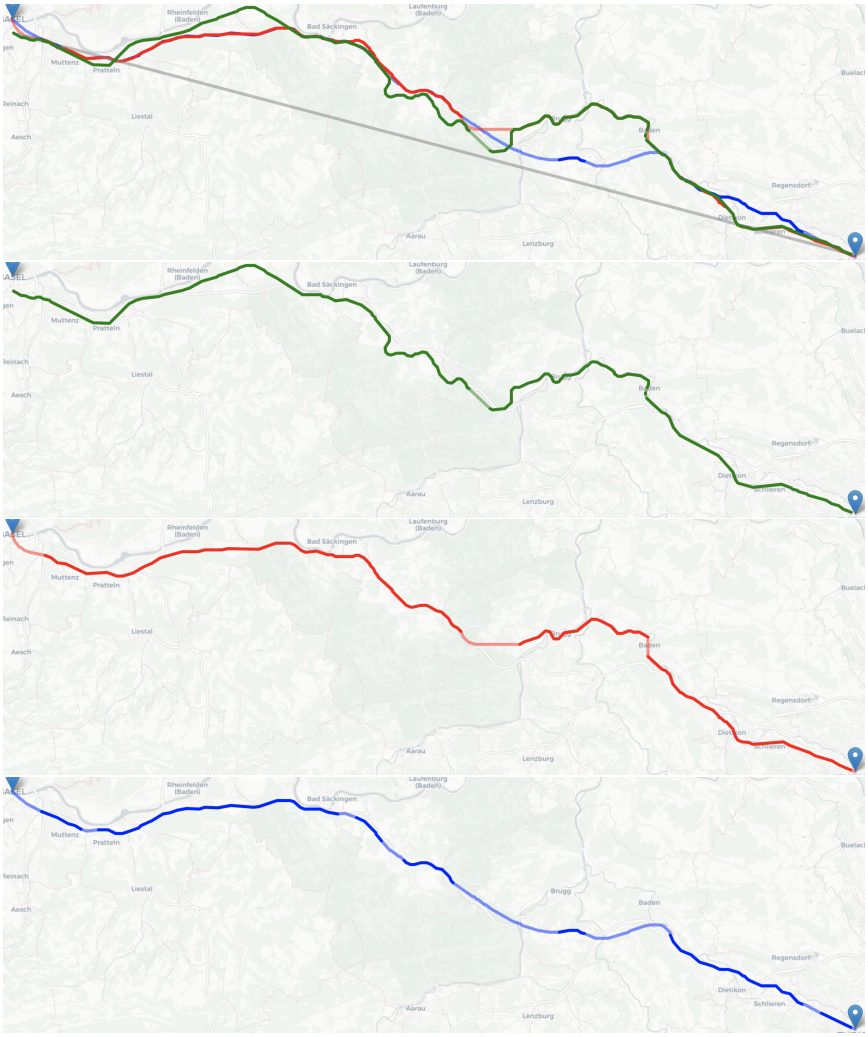


Figure 4.1: Comparison of the Hyperloop routes with the SBB route from Basel to Zurich. The first picture shows all routes on top of each other to showcase where they differ. The direct route is shown in grey and the SBB route is shown in green. The computed Hyperloop routes are shown in red (HL-A) and blue (HL-B).

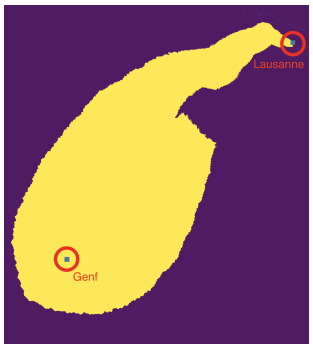


Figure 4.2: Search space of the pathfinding from Geneva to Lausanne. The expanded nodes are shown in yellow and the locations of Geneva and Lausanne are circled in red.

as well as the shortcomings of the current definition of a node in the A-Star implementation is provided in the next sections.

4.3.1 Time Computation

The computation of the time during the A-Star search can be too small, when tight curves follow a high speed section. During the pathfinding we store the previous 50 pixel points at each node, which is an approximate distance of 500 meters. This might not be enough distance to slow down from high speed when approaching a tight curve, without exceeding the acceleration limit. In this case, the algorithm thus does not consider enough time to brake, but only the time of braking within the last 50 pixels. Due to this limitation, the algorithm might underestimate the additional time for very tight curves and may miss, that it might be better to avoid the curve with a tunnel. However, to store more pixel points in the nodes would require more memory and imply even more computation time for each neighbour node. Therefore, we decided to store 50 pixels per node as a trade-off between accuracy of the estimated time and the run-time performance.

4.3.2 Cost Computation

The cost of the route is mainly dependent on the fraction of tunnels on the route. During the A-Star algorithm, we add a tunnel section as soon as the path crosses non-marked pixels on the bitmap. This is not completely accurate to the final shape of the route, because of the post-processing on the tunnels, which combines and straightens tunnel segments. After the post-processing the overall tunnel distance is often different than the one computed by the algorithm and the overall cost of the route can therefore also differ from the computed cost

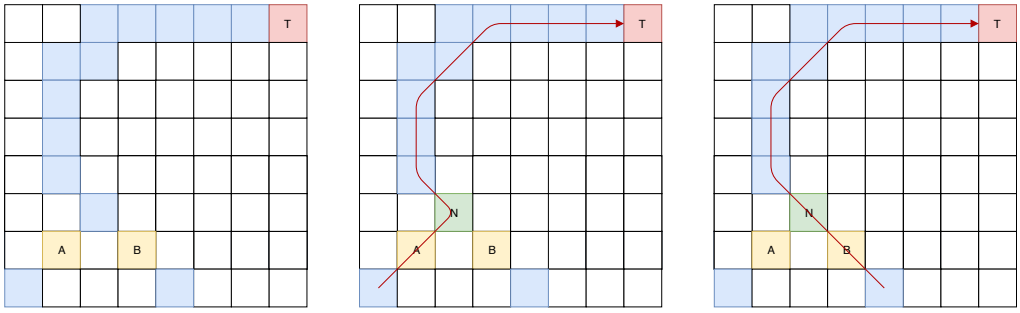


Figure 4.3: Example of a possibly non-optimal solution because pixel N is mapped to exactly one node and can therefore only expand once.

during the A-Star algorithm.

4.3.3 Node Definition

In our implementation of the A-Star algorithm each pixel corresponds to exactly one node. However, the cost function of a node does not only depend on the pixel location itself, but also on the arrival direction. To correctly adapt our pathfinding problem to A-Star it would therefore be required to define nodes as pairs consisting of pixel location and arrival direction. Although each pixel has eight direct neighbors, there are more than eight possible arrival directions. This is because the direction depends on the preceding route section, which is smoothed with cubic splines. Therefore, the possible arrival directions per pixel form a continuous set. By defining nodes as pixel and arrival direction pairs, the overall amount of expanded nodes would most likely increase and therefore, the run-time performance would get worse. This is why we decided to trade optimality of the result for better run-time performance by only defining one node per pixel.

In Figure 4.3 we show an example where the optimality of the route is not guaranteed. All colored nodes represent motorway or rail pixels, whereas in the white pixels tunnels have to be built. The red pixel is the target location, and the yellow pixels are two nodes that are expanded subsequently. Two possible paths with preceding pixel A or B are illustrated as red lines. Since the pixel N is represented by exactly one node, it can only get expanded once. If it expands the path with preceding pixel A, it will miss the path with preceding pixel B, even though this would avoid a sharp curve at node N.

Chapter 5

Conclusion

In this thesis we introduced a tool for automatic Hyperloop route generation. We developed an approach to do pathfinding on a bitmap in terms of minimizing building cost of the route per travel time. The straightness of the route can be changed by adapting the ratio between the tunnel and non-tunnel cost.

We developed a website where a precomputed Hyperloop network with two different settings of cost parameters can be analyzed. It can be compared to straight-line underground routes and the existing SBB train route. The user can adjust parameters to recompute the travel time and building cost of the routes.

There are several limitations. On the one hand the low run-time performance of the pathfinding, which makes it infeasible to use the A-Star implementation in real-time. On the other hand, it is not ensured that the optimal route regarding minimal cost per time is found.

5.1 Future Work

5.1.1 Pathfinding

Real-time performance and being able to do pathfinding in the whole world would be a great improvement. This would allow users to compute and analyze any path that they are interested in. The data can be easily adapted to the whole world. To achieve real-time performance with the algorithm, the frequency of spline fitting and path computation must be reduced. One possibility is to precompute shorter paths for subsections of the map, which could then be reused during the real-time computation. Another approach is to redefine neighbor nodes in the A-Star algorithm, by not only looking at the direct neighbors, but to include a larger area.

Since some route sections contain suboptimal paths, future work also includes improving the algorithm and ensuring optimal results. The node definition of the A-Star algorithm can be refined and the heuristic function can be further developed.

5.1.2 Network Generation

So far we built a network consisting of preselected routes. In order to automatically compute entire networks instead of just routes, the tool can be further extended such that the user defines as input an area in which he wants a Hyperloop network to be computed, instead of a start and end destination of the route.

In order to decide which cities should be connected within this area, one can access all cities with a large population. Then, by comparing the gross domestic product and population of a city with the building cost of a route that connects the city to the network, it can be decided whether it is feasible to join the city to the network. Optimal intersection points of the routes can be found by further developing the implementation of pathfinding on a bitmap.

5.1.3 Pod Scheduling

So far, we computed the construction cost of the routes and the travel time of the pods. To compute the cost we assume that one pod operates per km. With a precise pod scheduling algorithm that computes the exact number of pods per route, the cost can be computed more precisely.

Additionally, one can compare the throughput of passengers in Hyperloop networks with the throughput in current train networks. Though the capsules of Hyperloop pods are a lot smaller, they can operate more frequently than trains.

Bibliography

[1] Musk, E.: Hyperloop Alpha. (2013)

[2] Salter, R.M.: The Very High Speed Transit System. (1972)

[3] SpaceX: The Official SpaceX Hyperloop Pod Competition. https://www.spacex.com/hyperloop/. Accessed on 2019-02-03.

[4] Virgin Hyperloop One. https://hyperloop-one.com/. Accessed on 2019-02-03.

[5] Hyperloop Transport Technologies. https://www.hyperloop.global/. Accessed on 2019-02-03.

[6] Jufer, M., Bourquin, V., Sawley, M.: Global modelisation of the Swissmetro maglev using a numerical platform. (2019)

[7] EPF Lausanne: Epfloop. https://epfloop.ch. Accessed on 2019-02-03.

[8] Mossi, M., Engineering, G., Lausanne, S., Rossel, P., Lausanne, E.E.: Swissmetro: a Revolution in the High-Speed Passenger Transport System. (2019)

[9] Weidmann, U., Buchmueller, S., Rieder, M., Erath, A., Nash, A., Carrel, A.: Europaeische Marktstudie fuer das System Swissmetro. Phase I. **134** (2006) Schlussbericht, Kurzfassung, A 1 Zu- und Abgangszeiten des Verkehrssystems Swissmetro.

[10] Lindahl, M.: Track Geometry for High-Speed Railways. TRITA-FKT Report (2001)

[11] Railway Technology: Shanghai-Hangzhou Maglev. https://www.railway-technology.com/projects/shanghai-maglev/. Accessed on 2019-02-03.

[12] Barsi, A., Nyerges, A., Poto, V., Tihanyi, V.: An Offline Path Planning Method for Autonomous Vehicles. Production Engineering Archives (2018)

[13] Hart, P.E., Nilsson, N.J., Raphael, B.: A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics **SSC-4(2)** (1968) 100–107

[14] Schach, R., Jehle, P., Naumann, R.: Transrapid und Rad-Schiene-Hochgeschwindigkeitsbahn: Ein gesamtheitlicher Systemvergleich. (2006)

[15] de Boor, C.: A Practical Guide to Spline. Volume 27. (1978)

[16] Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numer. Math. **1**(1) (1959) 269–271

[17] Dierckx, P.: Algorithms for Smoothing Data with Periodic and Parametric Splines. Computer Graphics and Image Processing **20**(2) (1982) 171 – 184

[18] Google: Google Directions API. https://developers.google.com/optimization/routing/google_direction. Accessed on 2019-02-03.

[19] Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–)

[20] Wessel, B., Huber, M., Wohlfart, C., Marschalk, U., Kosmann, D., Roth, A.: Accuracy Assessment of the Global TanDEM-X Digital Elevation Model with GPS data. ISPRS Journal of Photogrammetry and Remote Sensing **139** (2018) 171 – 182

[21] OpenStreetMap contributors: Planet Dump Retrieved From https://planet.osm.org. https://www.openstreetmap.org. (2017) Accessed on 2019-02-03.

[22] Hadjieleftheriou, M., Manolopoulos, Y., Theodoridis, Y., Tsotras, V.J. In: R-Trees: A Dynamic Index Structure for Spatial Searching. Springer International Publishing, Cham (2017) 1805–1817

[23] Butler, H.: Rtree: Spatial indexing for Python. http://toblerity.org/rtree/. Accessed on 2019-02-03.

[24] Gillies, S.: Shapely. https://pypi.org/project/Shapely/. Accessed on 2019-02-03.

[25] Boeing, G.: OSMnx: New Methods for Acquiring, Constructing, Analyzing, and Visualizing Complex Street Networks. Computers, Environment and Urban Systems **65** (2017) 126 – 139

[26] Hagberg, A.A., Schult, D.A., Swart, P.J.: Exploring Network Structure, Dynamics, and Function using NetworkX. (2008) 11 – 15

[27] Weber, M.: osmctools. https://gitlab.com/osm-c-tools/osmctools. Accessed on 2019-02-03.

[28] Gillies, S., et al.: Rasterio: Geospatial Raster I/O for Python Programmers (2013–)

[29] Grinberg, M.: Flask Web Development: Developing Web Applications with Python. 1st edn. O'Reilly Media, Inc. (2014)

[30] python-visualization: folium. https://github.com/python-visualization/folium/. Accessed on 2019-02-03.

[31] Geuss, M.: Elon Musk has been pitching cheap tunnels from The Boring Company to big names. https://arstechnica.com/cars/2019/01/elon-musk-has-been. Accessed on 2019-02-03.

[32] Wikipedia Contributors: Schweizer Eisenbahnprojekte — Wikipedia, the Free Encyclopedia. Accessed on 2019-02-03.