Viterbi-based Trajectory Optimization

Jan Bernhard
Cornell Tech
jhb353@cornell.edu

Jinlong Hong
Cornell Tech
jh2695@cornell.edu

Abstract

The Viterbi algorithm is a popular dynamic programming algorithm. It has been applied to find an optimal state sequence in a variety of applications. In this work, we apply the Viterbi algorithm to the field of trajectory optimization. The goal is to find a path state sequence that results in a minimum distance or minimum time trajectory. We compare the optimization results to that of an optimization method used in a car racing game. The Viterbi-based approach achieves optimization results that match or approach the results of the reference method. This finding is interesting because the Viterbi-based method has a smaller runtime complexity compared to the reference method.

1 Introduction

Traditionally, the Viterbi-algorithm (VA) has been used in communication systems [3]. However, its application spread into many different areas, such as map-matching [12]. Inspired by the VA's versatility, we examine the VA's application to yet another field: trajectory optimization. The VA's objective is to optimize the state sequence of a time-discrete Markov process, referred to as trellis graph [2]. Trajectory optimization problems consist of two quintessential steps: the possible paths' parameterization and the path optimization. Using a trellis graph to parameterize the path allows us to apply the VA to optimize the path state sequence.

In this work, we investigate the effectiveness of applying the VA to estimate an optimal vehicle trajectory along a 2D track. Our experiment uses the VA to optimize for minimal distance in the first part of the investigation and minimal time in the second part. To estimate the time for a trajectory, we run the trajectory's path segments through a simulation with a constraint dynamic point model.

Our experiment aims to contextualize the VA's optimization results and, thus, simplify the interpretation and runtime complexity trade-off. To consider the algorithm's performance in context, we compare the method's results to the track's centerline and inner contour trajectory and the results from a trajectory optimization method used in video games [8]. We expect the VA to find trajectories that improve upon the centerline and inner contour trajectories. However, more interestingly, we investigate whether the VA-based approach can reach comparable optimization results to the reference method despite its lower runtime complexity.

The remaining sections of this report review a selection of related literature to our approach, outline the setup and code necessary to perform our experiment, explain our experiment, and discuss the results and possible conclusions. The code to run this report's experiment can be found on a public GitHub repository [1].

[1] Link to code repository of this report `https://github.com/janhenrikbern/VT_OPT`

CS 5112: Algorithms and Data Structures for Applications Project Report. Fall 2020.

2 Related Work

The VA was first introduced to operate on convolutional codes within the field of communication systems [11]. Inspired by Viterbi's work, [2] follow-up work shows that the VA is capable of finding the shortest path through a trellis. According to [3], another early follow-up to Viterbi's work originated from the field of control theory [7]. This is interesting because control theory is closely related to trajectory optimization. To the best of our knowledge, there has been little work applying the VA to the field of trajectory optimization. We found two relevant papers from the field of robotics: [6] the VA to optimize a line-following trajectory, and [9] to find the optimal sequence of maneuvers along a trajectory. Our approach is inspired by [6]. Similar to [6], our VA implementation adds the transition scores between states, our trellis graph is a direct mapping into physical space, and we extract our path from image data. Nevertheless, we extend [6] approach by applying it to a full loop path instead of just small line segments.

In the context of traditional trajectory optimization, our approach is most similar to that of a linear spline collocation method [5]. We verify whether we satisfy the vehicle and trajectory constraints at each state of the trellis and approximate the trajectory between sites through linear interpolation, as illustrated in Figure 2.

3 Method: Code & Experiments

This section explains the code and algorithms at the core of our experiment and the experiment's procedure.

3.1 Data Acquisition

The track is a closed-loop drawing, as shown in Figure 1. To convert the image to track data, we squeezed the RBG 3D-image into a 2D-image containing binary values. Additionally, we use the topological structural analysis of a digitized binary images algorithm from OpenCV to get the inner and outer contours of the track [10, 1]. These contours represent the inner and outer boundaries of the track. Additionally, the inner contour is used as a baseline for the minimum distance optimization.

3.2 Track Parameterization

All of our algorithms use the trellis graph as the underlying path parameterization. The trellis graph consists of equally distributed lines that vertically intersect with the inner and outer contours. All of the candidate points lie on the trellis lines, as shown in Figure 2. It is much easier to track the vehicle's state using a trellis than an approach that directly uses a pixel based map or continuous states.

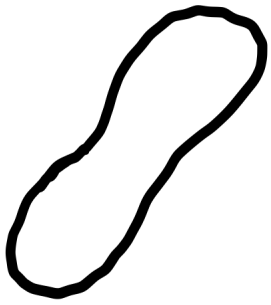


Figure 1: Track.

The trellis graph has three parameters: states m, sites n, and sample split number k. We split the inner boundary of the track into n equal segments. For each segment, we place a site by ejecting a line normal to the inner boundary segment. The line stops when it reaches the outer boundary, as shown in Figure 2. m states are equally distributed across the line between inner and outer track boundaries. Consequently, the trellis has $m \times n$ nodes in its graph. To reduce the complexity, we skipped k pixels between each site.

3.3 Vehicle Model

It is necessary to introduce some information about vehicle dynamics driving along the given trajectory to collect information about the given trajectory's lap-time. Consequently, we decided to introduce a constraint point model similar to the one used in [4]. Since our experiment's goal is to investigate the

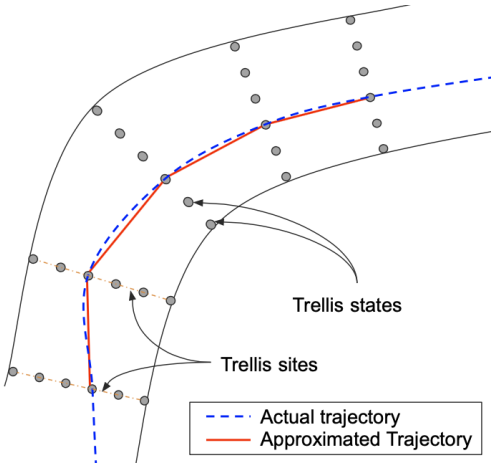


Figure 2: Track parameterization.

VA, we make a series of assumptions that simplify the dynamics vehicle. Firstly, the optimization's only decision variable is the steering target expressed as the coordinates of a possible next location. Secondly, we assume that the vehicle accelerates at its maximum possible rate, a_{max}, at all times as long as it hasn't reached its maximum speed, v_{max}. Deceleration only occurs while turning. Thirdly, we assume that the maximum turning rate, $d\theta_{max}$, is constant with speed and relative to a change in distance dx. Lastly, we ignore the effects of friction and drift and assume that the steering and heading angles are equal at all times.

We pick the constraint parameters in Table 1 rather arbitrarily. However, we thought them to be realistic for a speedy go-kart. The value for $d\theta_{max}$ corresponds to a turning radius of around 12 meters.

Based on our assumptions, we can describe the vehicle state with four variables θ, v, x, and y. x and y express the vehicle's location, v the speed, and θ the heading. The equations to update the vehicle's state from site i to $i+1$ are as follows:

$$\Delta\theta = |\arccos(\frac{|x_{i+1} - x_i|}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}) - \theta_i| \tag{1}$$

$$\Delta v = v_i c_{steering} + \frac{a_{max} c_{steering} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}{v_i} - v_i \tag{2}$$

The deceleration due to turning is included in Equation 2 by

$$c_{steering} = 1 - \frac{\Delta\theta}{d\theta_{max} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}$$

, such that $c_{steering} \in [0.0, 1.0]$. Based on Equations 1 and 2 and a given target location (x_{i+1}, y_{i+1}), the vehicle state at site $i+1$ is:

$$v_{i+1} = min(v_i + \Delta v, v_{max})$$

$$\theta_{i+1} = \theta_i + \Delta\theta + \frac{\Delta\theta}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}$$

3.4 Viterbi Implementation

The Viterbi algorithm is a dynamic programming algorithm to optimize a candidate path under a certain objective function. In our case, it is used to find the shortest distance path or the shortest time path for a vehicle model along the track. Each point in the trellis is stored in a 3D-array X of shape $n \times m \times 2$. n corresponds to the number of sites and m to the number of states in the trellis. The third

Constraint parameter	Value
a_{max}	$7.0\frac{m}{sec^2}$
v_{max}	$28.0\frac{m}{sec}$
$d\theta_{max}$	$15.0\frac{deg}{m}$

Table 1: Vehicle specifications

dimension holds the x and y coordinate values. For each trellis point $X[i, j]$, we calculate the optimal previous path to the point using the Viterbi algorithm. We gather the score and transition information in E and L, respectively. The arrays are of shape $n \times m$. The score values along the first state are set to $E[1, :] = 0$. The states' transition values along the first site are initialized as $L[1, :] = null$ pointers because the trajectory recovery on the backward pass ends at the first site. Beyond the first state, the value of $E[i, j]$ is the optimal score of point $X[i, j]$. The higher the score of point $X[i, j]$, the more desirable it is in respect to the optimization objective. In finding the optimal candidate paths at each site, we will record the transition between states of the previous and current site, as shown in Algorithm 1. Lastly, we retrieve the optimal path by selecting the highest score from $E[n, :]$, and backward tracking the corresponding transitions saved in L.

Algorithm 2 This implementation of the VA closely resembles the one described in [6].

Input: Trellis graph
Output: Score and transition arrays

function VITERBI(X, E, L)
  for each $site\ i \in [2, n]$ **do**
    for each $state\ j \in [1, m]$ **do**
      $E[i, j] = -infinity$
      for each $state\ k \in [1, m]$ **do**
        $score = E[i - 1, k] + scoring\_function(X[i, j])$
        if $E[i, j] < score$ **then**
          $E[i, j] = score$
          $L[i, j] = k$
        end if
      end for
    end for
  end for
end function

The operation of the scoring function depends on the optimization objective. For the minimum distance objective, the score is the negative Euclidean distance between two states of consecutive sites. For the minimum time objective, the score is the negative time required to reach state k of site i from state j of site $i - 1$. The time to reach the next state dt is derived from the vehicle's speed v at sites i and $i + 1$:

$$dt_{i,j\to k} = \frac{\sqrt{(x_{i-1,j} - x_{i,k})^2 + (y_{i-1,j} - y_{i,k})^2}}{\frac{1}{2}(v_{i-1,j} + v_{i,k})} \tag{3}$$

We add instead of multiply the scores in this implementation because the score values are negative and not guaranteed to be > 1. Each of those properties could lead to undesirable behaviors if consecutive scores are multiplied. Additionally, we check if a state satisfies the vehicle's turning constraint $d\theta_{max}$ during the scoring process. If a state doesn't satisfy the turning constraint, it is skipped.

The runtime complexity of the VA is $O(sites * states^2)$; the memory usage is $O(sites * states)$.

3.5 Reference Optimization Method

To investigate whether the VA-based approach is a viable solution, we compare it to the reference trajectory optimization algorithm underlying the car racing game "Mantis Burn Racing" [2] [8]. Its

[2] `https://store.steampowered.com/app/446100/Mantis_Burn_Racing/`

implementation evaluates all possible waypoints on the trellis from the finish line to the start line. This backward, forward evaluation approach has the advantage of knowing what is ahead when gathering the optimal trajectory from a given position. The scoring function of the algorithm has parameters α and β, which express a preference for a minimum distance or minimum curvature path, respectively. In Equation 4, we adjusted the scoring function recommended in [8] to equally scale distance and curvature values. The scoring function takes as input three location coordinates $\vec{x}_{i-1}$, $\vec{x}_i$, and $\vec{x}_{i+1}$ for some states of sites $i-1$, i, and $i+1$. When considering $\vec{d_1} = \vec{x}_i - \vec{x}_{i-1}$ and $\vec{d_2} = \vec{x}_{i+1} - \vec{x}_i$, we can express the scoring function as follows:

$$f(\vec{d_1}, \vec{d_2}) = (\beta * \frac{\vec{d_1} \cdot \vec{d_2}}{||\vec{d_1}|| * ||\vec{d_2}||} - \alpha) * (||\vec{d_1}|| + ||\vec{d_2}||) \tag{4}$$

This algorithm computes a score for every combination of states of consecutive sites $i-1$, i, and $i+1$. Consequently, the runtime complexity of the algorithm is $O(sites \times states^3)$ with a memory usage of $O(sites \times states^2)$.

3.6 Experiments

In our experiment, we apply the VA-based and reference optimization method on the same track and compare the results with the centerline and inner contour trajectories as baselines. We run this experiment twice: First, with a minimum distance objective, and, second, with a minimum time objective. Distance and time are collected for the second full lap. The underlying parameterization (trellis) of the track is kept constant.

3.6.1 Minimum Distance Experiment

The baseline for the minimum distance experiment is the path along the inner contour of the track. The minimum distance objective depends on the euclidean distance between sites i and $i+1$:

$$\min_{x,y} \sum_{i}^{n} \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}$$

To optimize this objective function, we use the VA method's distance-specific scoring function. For the reference method's scoring function, we set $\alpha = 1$, $\beta = 0$, as shown in Equation 4.

3.6.2 Minimum Time Experiment

The baseline for the minimum time experiment is the path along the centerline of the track. The minimum time objective takes into consideration the vehicle's linearly interpolated speed v at sites i and $i+1$:

$$\min_{x,y} \sum_{i}^{n} \frac{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}}{\frac{1}{2}(v_{i+1} + v_i)}$$

Due to the forward first pass of the VA, it is possible to consider the vehicle dynamics in the time-specific scoring function, as illustrated in Equation 3. The reference algorithm assigns the scores from the finish to the start line. Consequently, it is not possible to use the vehicle dynamics in the score calculations. For the reference function, we set $\alpha = 1 - \epsilon$, $\beta = \epsilon$ and search for the minimum time trajectory by iteratively increasing ϵ from 0.0 to 1.0 in steps of 0.02. $\epsilon = 0.46$ achieves the minimum time trajectory for this specific combination of track and vehicle dynamics.

3.6.3 Baselines

There are two baselines: the centerline and the inner contour trajectory. The centerline is the "average" implementation of all of the path candidates. We directly used the middle point of each trellis line to build the centerline trajectory. The inner contour trajectory is an intuitive way to build a short path from the trellis graph. However, it will not be the shortest trajectory if the track contains concave corners. In the case of a track with concave corners, it is faster to follow a straight line than following the inner boundary.

4 Results & Discussion

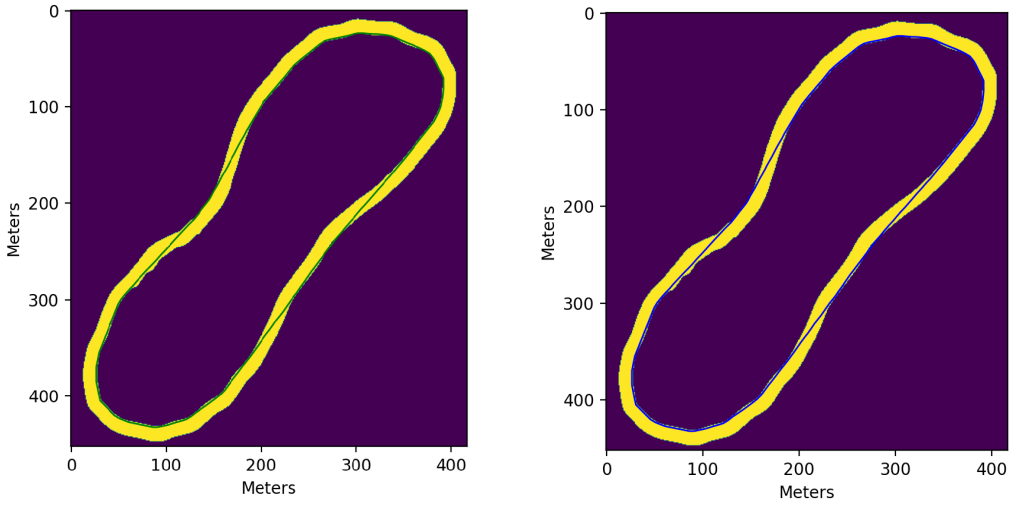
This section reports the optimization results for the minimum distance and minimum time objectives and discusses the trade-off between runtime performance and optimization accuracy. The trellis graph resolution for the experiments is 200 sites with 30 states per site.

4.1 Minimum Distance Results

	Best Distance [m]	Best Time [sec]
Viterbi Algorithm	**1155.46**	47.319
Reference Optimization	**1155.46**	**47.014**
Center line	1215.11	57.082
Inner contour	1168.09	58.615

Table 2: Optimization results.

The middle column of Table 2 indicates that the VA-based approach and the reference method tie for the best minimum distance result. Visualizing the corresponding trajectories in Figure 3 shows that the minimum distance trajectories are identical. Additionally, both methods outperform the centerline and inner contour trajectories by a significant margin.



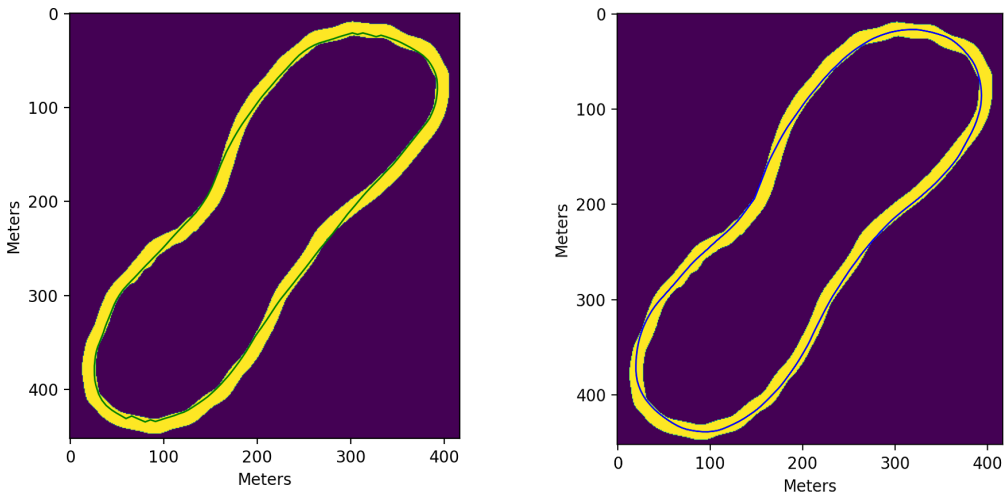
(a) Viterbi Algorithm

(b) Reference Method

Figure 3: Minimum distance paths computed by the different optimization methods.

4.2 Minimum Time Results

Considering the results for the minimum time objective presented in the last column of Table 2, the reference optimization method generates a faster trajectory than the VA-based method. Noticeable are the differences in the proposed racing line trajectories between Figure 4a and 4b. The VA-based trajectory remains relatively close to the minimum distance trajectory shown in Figure 3a. In contrast, the reference-based trajectory takes the curves a lot wider, which causes it to deviate from the minimum distance trajectory significantly. This longer trajectory of the reference method results in a lap time that is around 0.3 seconds faster than that of the Viterbi-based method.



(a) Viterbi Algorithm

(b) Reference Method

Figure 4: Minimum time paths computed by the different optimization methods.

4.3 Viterbi and Reference Trajectory Comparision

The results for the minimum distance objective favor the VA-based approach because it achieves an identical distance to the reference method while requiring a smaller runtime complexity. The minimum time trajectory of the VA shows some limitations: First, it is slower than that of the reference method, and, second, it shows a slight zick-zack pattern around the turns of the trajectory, as illustrated by Figure 4a. These artifacts negatively impact vehicle speed, which negatively impacts the lap time. A possible explanation for these artifacts is the lack of foresight in the VA-based optimization method.

4.4 Runtime-Accuracy Trade-off

The centerline and inner contour path are reasonable paths. However, the trajectories are generated without consideration for lap distance and time. Computing the baseline trajectories takes one iteration over all n sites to collect their coordinate points, which means the computation's time complexity is $O(n)$. A valid solution alone is insufficient for many applications; thus, an optimized solution is necessary. The results of Table 2 show that both the VA and reference methods provide such solutions. The VA optimizes the trajectory, which increases its runtime complexity and memory usage to $O(n * m^2)$ and $O(n * m)$, respectively. The reference method is more complex than VA with a runtime of $O(n * m^3)$ and $O(n * m^2)$ memory usage. The reference method generates a better time solution since its implementation allows it to compute a trajectory with information about what is ahead. We also considered an exhaustive search strategy to get an absolute optimal solution. However, the exhaustive search's time complexity is $O(n^m)$, making it too time-consuming to run on the available hardware. Ultimately, the VA and the reference method appear to be algorithms with a reasonable optimization accuracy to runtime trade-off.

To further consider the trade-off between our VA-based optimization method and the reference method, Figure 5 visualizes the percentage the VA-based approach is above the best solution of the reference approach. First, we see that the VA-based method matches or closely approximates the minimum distance and minimum time solutions of the reference method for as few as 20 states per trellis site; increasing the number of trellis states beyond 20 yields diminishing returns. Second, the plot of the VA-based results for finding a minimum time trajectory level off before matching the results of the reference method because the VA-based approach only considers previous information when evaluating which state to choose next. This result underscores the importance of foresight when optimizing the trajectory for a dynamic vehicle model. The reference-based approach considers information about what lies ahead when selecting the next state. This key difference illustrates a fundamental limitation of the VA when it comes to trajectory optimization. Nevertheless, Figure 5

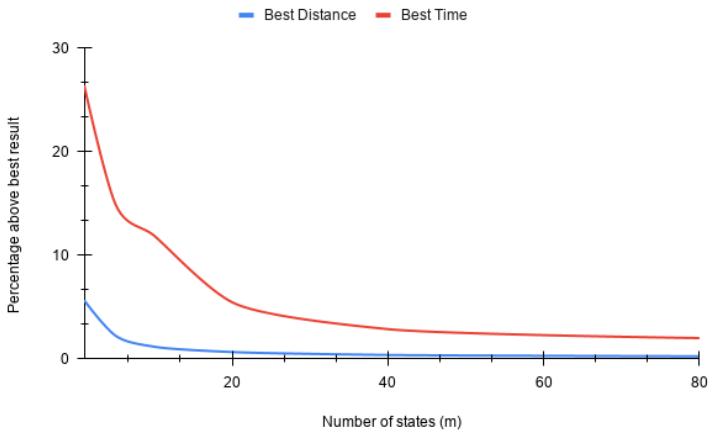


Figure 5: Viterbi-based optimization results with increasing state count. The target value is overall best distance and lap-time values achieved by the reference method at a state count of 80.

shows that the VA-based approach reaches a decent minimum time trajectory that is about 2% off of the reference method's optimization result. When considering the combined runtime complexity of the reference method and vehicle-specific iterative parameter search, this 2% improvement in lap time comes at a high computational cost. Consequently, the VA-based method might still be worth considering for minimum time objectives if computational resources are limited. An example of an application in which computation resources are limited is an online game that frequently generates random tracks and needs to evaluate the players' times with some context.

5 Conclusion

In this work, we investigated the application of the Viterbi algorithm to the field of trajectory optimization. Our experiment evaluated the algorithm by comparing it to a method used in a commercial car racing game. The results indicate that the Viterbi-based method yields matching results for the minimum distance trajectory and achieves a close estimate of the minimum time trajectory compared to the car racing game method. The main advantage of the Viterbi-based approach is the improved runtime performance in comparison to the game method. Ultimately, our Viterbi-based method may provide value for applications in which computational resources come at a premium, and a low-fidelity estimate of an optimal trajectory is sufficient.

References

[1] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[2] G. D. Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973. ISSN 1558-2256. doi: 10.1109/PROC.1973.9030.

[3] G. D. Forney. The Viterbi Algorithm: A Personal History. apr 2005. URL `http://arxiv.org/abs/cs/0504020`.

[4] W. Hsieh. First Order Driving Simulator. 2017. URL `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2017/EECS-2017-102.html`.

[5] M. Kelly. An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation. *SIAM Review*, 59(4):849–904, jan 2017. ISSN 0036-1445. doi: 10.1137/16M1062569. URL `https://epubs.siam.org/doi/10.1137/16M1062569`.

[6] P. Mazurek. Line estimation using the Viterbi algorithm and track-before-detect approach for line following mobile robots. In *2014 19th International Conference on Methods and Models in Automation and Robotics (MMAR)*, pages 788–793. IEEE, sep 2014. ISBN 978-1-4799-5081-2. doi: 10.1109/MMAR.2014.6957456. URL `http://ieeexplore.ieee.org/document/6957456/`.

[7] J. Omura. On the viterbi decoding algorithm. *IEEE Trans. Inf. Theor.*, 15(1):177–179, Jan. 1969. ISSN 0018-9448. doi: 10.1109/TIT.1969.1054239. URL https://doi.org/10.1109/TIT.1969.1054239.

[8] T. Phung Dinh. How to calculate racing lines automatically, Dec 2016. URL http://phungdinhthang.com/2016/12/16/calculate-racing-lines-automatically/?i=1.

[9] J. Roth. *Intelligent Autonomous Systems 15*, volume 867 of *Advances in Intelligent Systems and Computing*. Springer International Publishing, Cham, 2019. ISBN 978-3-030-01369-1. doi: 10.1007/978-3-030-01370-7. URL http://link.springer.com/10.1007/978-3-030-01370-7.

[10] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32 – 46, 1985. ISSN 0734-189X. doi: https://doi.org/10.1016/0734-189X(85)90016-7. URL http://www.sciencedirect.com/science/article/pii/0734189X85900167.

[11] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, April 1967. ISSN 1557-9654. doi: 10.1109/TIT.1967.1054010.

[12] H. Wei, Y. Wang, G. Forman, Y. Zhu, and H. Guan. Fast viterbi map matching with tunable weight functions. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, page 613–616, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 9781450316910. doi: 10.1145/2424321.2424430. URL https://doi.org/10.1145/2424321.2424430.