2016 European Control Conference (ECC)
June 29 - July 1, 2016. Aalborg, Denmark

Waypoint planning with Dubins Curves using Genetic Algorithms

Karl D. Hansen, Anders la Cour-Harbo

Abstract— **Mission planning for aircraft is often done as waypoint planning. A sequence of waypoints describing the three-dimensional positions that the aircraft must visit. A common approach is to plan the sequence of the waypoints such that the Euclidean distance between them is minimized. When the high-level waypoint planning is finished, a finer grained planning is executed to obtain a trajectory that the aircraft must follow. When the waypoints in a plan are distributed far apart compared to the turning radius of the aircraft, the two-step planning approach works well, but when the waypoints are closer, the kinematics of the aircraft ruins the plan. This work describes an approach that uses a genetic algorithm to solve the waypoint planning problem while considering the kinematics of the aircraft in one single step. This approach entails the addition of a heading and target speed along with the position in the waypoint definition. The kinematics of the aircraft is modeled with Dubins curves, which are extended to allow variable turning radii.**

I. INTRODUCTION

When planning a route through a set of waypoints for a mobile robot, the optimization criterion is often the traveled length or time spent traveling. A straight forward method of performing this planning is to optimize Euclidean distance of the route. However, following straight lines between waypoint might easily violate kinematic or dynamic constraints, in fact such a path might be very far from a feasible route for the mobile robot. The inertia of a moving robot implies that some sort of 'turns' might be quite useful for planning a route.

The geometrically simplest of turns is (part of) a circle, which when connected with straight line forms the basis for Dubins curves [1]–[3]. The Dubins curve is a path composed of line and circle segments, connecting two points with corresponding headings in the plane. Dubins [1] showed that any two points with arbitrary headings can be connected by a combination of two circle segment with a straight line segment in between (CSC), three circle segments (CCC), or any subset of these, with the circle segments C being either left- or right-turning (L or R). The circles in the paths are all constrained to one given radius.

The Dubins curves have also been considered for applications to aircraft [3], [4]. Here the curves to some extent fit the kinematics of an aircraft moving at a constant speed. This is because of the bounded turning speed of an aircraft, where a constant forward speed translates to a constant turning radius. The traditional Dubins curves, defined in the plane, have been generalized to three dimensions [5], where the circle segments no longer are defined as left and right turns, but defined on a doughnut shaped manifold located around the orientation vector of the waypoints. That approach may lead to very steep inclines, a problem that has been treated in [4] where a bound is set on the rate of change in altitude, leading to 'corkscrew' maneuvers to increase or decrease altitude. A nice review of methods for path and trajectory following of Dubins curves in aircraft autopilots is presented in [6].

The traditional way of planning a waypoint-based path with Euclidean performance function is solving a traveling salesman's problem (TSP), i.e. finding an ordering of the waypoints that produces the shortest possible accumulated distance. The TSP is an $\mathcal{NP}$-hard problem [7] thus exact algorithms are not efficient and heuristics have been developed [8], [9]. Typical approaches include neighborhood-search meta-heuristics like ant-colony optimization, tabu-search, and genetic algorithms (GA).

It seems obvious to combine Dubins curves with a TSP where the performance function is traveled distance along the curves (rather than Euclidean distance). However, the Dubins TSP introduces the continuous variable of heading for each waypoint, and these variables have to be determined somehow in order to determine the Dubins curves connecting the waypoints. Since the length of the path is very much influenced by the headings these variables are part of the optimization. Work has been done to solve this problem by discretizing the headings [3] thus producing a generalized TSP with clusters of waypoints with fixed headings where only one of the waypoints in the cluster should be visited. A well-known heuristic method is known as the alternating algorithm [2], which sets every second heading to point in the direction of the next waypoint in the sequence and the next heading to the same value. This produces a route of alternating straight segments and Dubins curved segments. Another approach is to manage the headings in a genetic algorithm by randomly selecting headings in the neighborhood function [10].

It should be noted that the optimal solution for a Dubins TSP tends to the optimal solution for the Euclidean TSP as turning radius of the aircraft becomes smaller in relation to distance between waypoints. This is because the turning then becomes an increasingly smaller part of the total traveled distance. Conversely, solving the Dubins TSP when the waypoints are close together compared to the turning radius leads to "cluttered" solutions because the aircraft will have to make wide turns to "get back to" the waypoints. Within certain limits, this can be countered by lowering the forward speed of the aircraft to reduce the turning radius. Therefore, we have extended the Dubins curves to work with varying radii, while also considering the heading associated with each

K. Hansen and A. la Cour-Harbo is with the Department of Electronic Systems, Aalborg University, Denmark, [kdh, alc]@es.aau.dk.

waypoint. Thus, we present a Dubins curve based trajectory generation and waypoint planning with variable radii, which is based on a variable forward speed. The planning is based on a genetic algorithm that modifies the both the continuous heading variable as in [10] and also the target speed in each waypoint.

II. METHODS

The varying radii in Dubins curves and the application of a genetic algorithm to search for the optimal path with time as the performance measure are two distinct methods that we will address in this section. First, we present the generation of point-to-point Dubins curves with variable radii. This is followed by the formulation of a genetic algorithm that optimizes over the combinatoric sequence (solving the TSP) along with optimizing the continuous heading and target speeds of the waypoints.

A. Variable Speed Dubins Vehicle

The Dubins Curves that we will present here is an extension of the constant speed, bounded angular speed vehicle to a bounded speed, bounded angular speed vehicle. The forward speed v will be bounded

$$v_{\min} \leq v \leq v_{\max} \ , \tag{1}$$

and is subject to an acceleration that may be either $\pm a_{\max}$ or 0. It is assumed that the vehicle is able to turn with an angular velocity ω, which may take on only maximal values $\pm\omega_{\max}$ or 0. The forward speed and the angular speed defines the turning radius

$$r = \frac{v}{\omega} \ . \tag{2}$$

Further, it is assumed that the vehicle is only able to accelerate on straight stretches where the angular velocity is 0.

B. Point-to-Point Variable Radii Dubins Curve

We want to determine the length of a Dubins curve between two waypoints given the heading and speed in both waypoints. There are six possible combinations of segments forming a Dubins curve

$$\{\text{Right-Straight-Right}\}\ \{\text{LSL}\}\ \{\text{RSL}\}$$
$$\{\text{LSR}\}\ \{\text{RLR}\}\ \{\text{LRL}\} \ .$$

The length of each of the three segments are denoted t, p, and q for the first, middle, and last segments, respectively. The heading is any real number between 0 and 2π, and the speed is any positive, real number, and defines the radius of the circle segment associated with the waypoint.

In [11], Shkel and Lumelsky derive formulas for computing the lengths of the different Dubins curve types along with a classification method to select the right type before the actual computation, this is opposed to the 'traditional' compute-and-compare approach. The approach in [11] is to translate, rotate, and scale the problem into canonical form before computation. The canonical form places the initial waypoint in the origin, is scaled with the reciprocal of the turning radius, and rotated to place the final waypoint on the x axis.

The same idea of transforming the original problem into a canonical form will be used here. However, since the radii of are not equal in our case, a slightly different transformation is appropriate. In stead of transforming the problem to put the waypoints on the x axis, it is transformed to put the two centers of rotation on this axis. Note that this means that different transformations must be applied for the different path types.

We want to travel from one waypoint at $\mathbf{w}_1 = (x_1, y_1)$ with heading ϕ_1 and radius r_1 to another waypoint at $\mathbf{w}_2 = (x_2, y_2)$ with heading ϕ_2 and radius r_2. The centers of rotation then become

$$\mathbf{c}_{\text{right},i} = \mathbf{w}_i + r_i \begin{pmatrix} \sin(\phi_i) \\ -\cos(\phi_i) \end{pmatrix} \ , \tag{3}$$

$$\mathbf{c}_{\text{left},i} = \mathbf{w}_i - r_i \begin{pmatrix} \sin(\phi_i) \\ -\cos(\phi_i) \end{pmatrix} \ . \tag{4}$$

These parameters are all illustrated on Figure 1 for the case where the Dubins curve is made as Right-Straight-Right. The task now is to determine the lengths t, p, and q of the three segments, and for this we will use the transformation described above. The rotation angle is given by the angle $-\Theta$, the scaling by $1/r_1$, and the translation by $-\mathbf{c}_1$. This will transform the original configuration to the one seen in Figure 2. Now define the scaled radius of the second circle $\rho = r_2/r_1$ and the scaled distance between circle centers $d = \|\mathbf{c}_1 - \mathbf{c}_2\|/r_1$. The length of the short leg of the right triangle (gray in the figure) is $1-\rho$ and the length of the hypotenuse is d. Thus, the angle at which the straight segment intersects the two circles is given as $\phi = \arccos\big((1-\rho)/d\big)$ and the length of this same segment is $p' = \sqrt{d^2 - (1-\rho)^2}$. Now, finally, let α_i be the angle between the x axis and the transformed waypoint $\mathbf{w}_i'$, which is given as $\alpha_i = \phi_i - \Theta + \pi/2$. Then $t' = \alpha_1 - \phi$ and $q' = \rho(\phi - \alpha_2)$ (constrained to the interval 0 to 2π). For this to make sense we need only require that $d \geq 1-\rho$, meaning that none of the circles are properly inscribed in the other (in which case no tangent exists between the two).

Extending this result to include also RSL, LSL, and LSR we get

$$p' = d\sqrt{1-\lambda^2} \tag{5}$$

$$t' = \begin{cases} (\alpha_1 - \phi) \mod 2\pi & \text{RSR, RSL} \\ (\phi - \alpha_1) \mod 2\pi & \text{LSL, LSR} \end{cases} \tag{6}$$

$$q' = \begin{cases} \rho((\phi - \alpha_2) \mod 2\pi) & \text{RSR} \\ \rho((\alpha_2 - \phi) \mod 2\pi) & \text{LSL} \\ \rho((\phi - \alpha_2 - \pi) \mod 2\pi) & \text{LSR} \\ \rho((\alpha_2 - \phi - \pi) \mod 2\pi) & \text{RSL} \end{cases} \tag{7}$$

where

$$\alpha_1 = \begin{cases} \phi_1 - \Theta + \frac{\pi}{2} & \text{RSR, RSL} \\ \phi_1 - \Theta - \frac{\pi}{2} & \text{LSL, LSR} \end{cases}$$

$$\alpha_2 = \begin{cases} \phi_2 - \Theta + \frac{\pi}{2} & \text{RSR, LSR} \\ \phi_2 - \Theta - \frac{\pi}{2} & \text{LSL, RSL} \end{cases}$$

$$\phi = \begin{cases} -\arccos\lambda & \text{LSL, LSR} \\ \arccos\lambda & \text{RSR, RSL} \end{cases}$$

$$\lambda = \begin{cases} (1-\rho)/d & \text{RSR, LSL} \\ (1+\rho)/d & \text{LSR, RSL} \end{cases}$$

$$d = \frac{\|\mathbf{c}_1 - \mathbf{c}_2\|}{r_1} \qquad \rho = \frac{r_2}{r_1}$$

$$d \geq \begin{cases} 1-\rho & \text{RSR, LSL} \\ 1+\rho & \text{RSL, LSR} \end{cases}$$

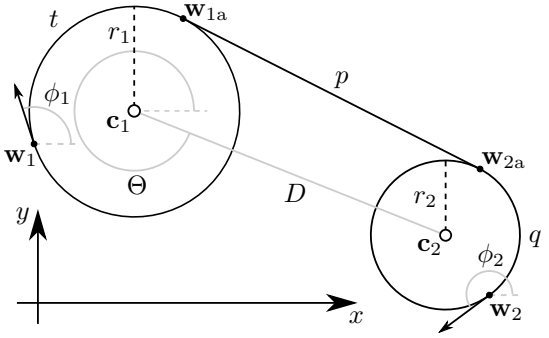

Fig. 1. Dubins curve in original form.

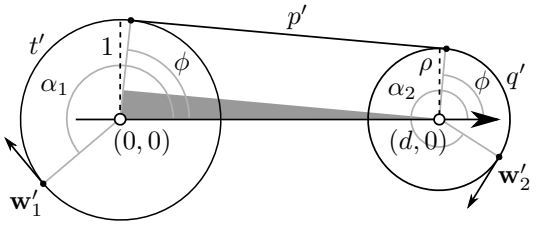

Fig. 2. Canonical representation of the path seen in Figure 1.

As the lengths are both rotation and translation invariant, the original lengths $\{t, p, q\}$ are achieved by scaling $\{t', p', q'\}$ by r_1.

Since we do not allow acceleration of the vehicle on the curving part of the path the CCC cases are of limited used, and therefore these cases are not included in this work.

C. Interpolation functions for Dubins Curves

Given the lengths of the non-canonical segments, three motion operators will translate a point along the path. Let $C : \mathbf{R}^5 \mapsto \mathbf{R}^3$ be a mapping along a circle. The input is $[x\, y\, \phi\, s\, r]^\top$ where x, y is the initial position in $\mathbf{R}^2$, ϕ is the heading angle, s is the arc distance and r is circle radius (which should be positive to make left turns and negative to make right turns). Then C is given by

$$C([x\, y\, \phi\, s\, r]^\top) = \begin{bmatrix} x - r\left(\sin(\phi) - \sin\left(\phi + \frac{s}{r}\right)\right) \\ y + r\left(\cos(\phi) - \cos\left(\phi + \frac{s}{r}\right)\right) \\ \phi + \frac{s}{r} \end{bmatrix}.$$

A translation along the straight segment is more simple, and is given by $S : \mathbf{R}^4 \mapsto \mathbf{R}^3$ as

$$S([x\, y\, \phi\, s]^\top) = \begin{bmatrix} x + s\cos(\phi) \\ y - s\sin(\phi) \\ \phi \end{bmatrix}.$$

Note that ϕ in S is the heading of the straight segment. If the vehicle starts at waypoint $\mathbf{w}_1$ with heading ϕ_1 and is following a left turn circle of radius r_1 then a) the position of the vehicle when it leaves the circle to traverse the straight segment, and b) the position when it leaves the straight segment to enter the second (right turn) circle with radius r_2, and c) the position when the entire maneuver is completed, are given by

$$\text{a)} \quad \mathbf{w}_{1a} = C\left(\begin{bmatrix} x_1 \\ y_1 \\ \phi_1 \\ t \\ r_1 \end{bmatrix}\right)$$

$$\text{b)} \quad \mathbf{w}_{2a} = S\left(\begin{bmatrix} \mathbf{w}_{1a} \\ p \end{bmatrix}\right)$$

$$\text{c)} \quad \mathbf{w}_2 = C\left(\begin{bmatrix} \mathbf{w}_{2a} \\ q \\ -r_2 \end{bmatrix}\right)$$

These functions may be useful when visualizing the curves of a computed path.

D. Velocities for Time-Optimal Traversal of the Curves

The straight segments are the only places where the vehicle is allowed to accelerate to match the initial velocity v_1 in (x_1, y_1, ϕ_1) with the final velocity v_2 in (x_2, y_2, ϕ_2). The time-optimal traversal of the segment is achieved by moving as fast as possible. This means that the vehicle must accelerate from v_1 to maximum velocity and later reduce it to v_2.

Assuming first that there is no upper limit on the velocity, then on the segment p the maximum velocity that can be achieved v_m, subject to a maximum allowed acceleration of a, must satisfy

$$p = \left(v_1 + \frac{v_\text{m} - v_1}{2}\right)\tau_1 + \left(v_2 + \frac{v_\text{m} - v_2}{2}\right)\tau_2$$

where $\tau_i = (v_\text{m} - v_i)/a$ are the times it take to reach v_m from v_i. Solving this gives

$$v_\text{m} = \sqrt{\frac{2pa + v_1^2 + v_2^2}{2}}. \tag{8}$$

Now, if this velocity is higher than the maximum allowable velocity v_max then we simply impose this limit and the total travel time on segment p becomes

$$\begin{aligned} \tau_p &= \tau_1 + \tau_2 + [\text{delay due to speed limit}] \\ &= \frac{v_\text{m} - v_1}{a} + \frac{v_\text{m} - v_2}{a} + \frac{(v_\text{m} - v_\text{max})^2}{a v_\text{max}}. \end{aligned}$$

In case the velocity limit is not reached the third term is (defined to be) zero. The time for traversal of the entire path is then

$$T = \tau_t + \tau_p + \tau_q = \frac{t}{v_1} + \tau_p + \frac{q}{v_2} \ . \tag{9}$$

Because of the bounded acceleration the path may be infeasible if the difference between the initial and final speeds is too large to realize over the p-segment. When this is the case, $v_{\mathrm{m}} < \max\{v_1, v_2\}$.

E. Genetic Algorithm

To find the fastest path through the waypoints we employ a genetic algorithm. In this case there are three parameters that determine the fitness of the solution when varied. The parameters are the sequence of the waypoints, the heading in each waypoint, and the velocity in each waypoint. In the traditional case where the Euclidean distance between waypoints is used the choice of connecting edges in the TSP graph are decoupled resulting in a entirely combinatorial problem. However, in the Dubins TSP case, the headings of the waypoints couples the segments so that each Dubins path cannot be optimized separately from the order of the waypoints. The same applies to the velocity in this variable-speed Dubins TSP.

Much work has been done in the genetic algorithms for Euclidean TSP, for a review see [12]. Yu and Hung [10] extends some of the traditional methods for use in Dubins TSP. Much of their representation is used here to adapt the genetic algorithm to the specific problem at hand. The algorithm is presented in Figure 3.

1) Encoding: The path representation, described in [12], encodes the problem in an ordered list of indices to the waypoints. In the jargon of genetics, the encoded solution to the problem is called the genome and the entries of the genome is called the genes. This way, the ordered list of indices is the genome and each index is a gene. Yu and Hung [10] extends the path representation to include the heading in each gene. The same approach is taken here; extending the representation with the velocity, so that a genome is a sequence of genes of the form (i, ϕ, v). The genome example

$$\{(1, 1.4, 2.3), (3, 3.6, 2.7), (2, 4.1, 1.6), (4, 1.1, 1.2)\}$$

starts in the waypoint indexed 1 with the heading 1.4 radians and the velocity 2.3, and move through waypoints 3, 2, and 4 with their associated headings and velocities.

2) Fitness: The fitness function evaluates how fit each individual in the population is. As the objective is to minimize the time spent, individuals with a lower time cost should have a higher fitness than slower individuals. This is achieved here by letting the fitness equal to the time of the slowest genome of the population minus the time of the evaluated genome.

3) Selection: A roulette wheel selection is used. Here each bin of the roulette wheel has a width equal to the fitness of the corresponding individual. This way the roulette ball selects more fit individuals more often.

Require: $N_p, P_c, P_{inv}, P_{ex}, P_{dis}, P_\phi, P_v$
  procedure GENETIC ALGORITHM
    Initialize $population \leftarrow N_p$ random genomes
    $doTerminate \leftarrow False$
    while not $doTerminate$ **do**
      $newPopulation \leftarrow$ empty population
      Make ROULETTEWHEEL selector from $population$
      for $i \leftarrow 1, N_p$ **do**
        repeat
          if RAND(0, 1) $\leq P_c$ **then**
            $parent_1 \leftarrow$ ROULETTEWHEEL
            $parent_2 \leftarrow$ ROULETTEWHEEL
            $child \leftarrow$ CROSSOVER($parent_1, parent_2$)
          else
            $child \leftarrow$ ROULETTEWHEEL
          end if
          if RAND(0, 1) $\leq P_{inv}$ **then**
            $child \leftarrow$ INVERSE(child)
          end if
          if RAND(0, 1) $\leq P_{ex}$ **then**
            $child \leftarrow$ EXCHANGE(child)
          end if
          if RAND(0, 1) $\leq P_{dis}$ **then**
            $child \leftarrow$ DISPLACE(child)
          end if
          if RAND(0, 1) $\leq P_\phi$ **then**
            $child \leftarrow \phi$-RANDOM(child)
          end if
          if RAND(0, 1) $\leq P_v$ **then**
            $child \leftarrow v$-RANDOM(child)
          end if
          Add $child$ to $newPopulation$
        until VALID($child$)
      end for
      Evaluate $newPopulation$
      if $bestNewGenome > allTimeBest$ **then**
        $allTimeBest \leftarrow bestNewGenome$
      else
        $worstNewGenome \leftarrow allTimeBest$
      end if
      $population \leftarrow newPopulation$
      $doTerminate \leftarrow$ evaluate termination criterion
    end while
  end procedure

Fig. 3. The algorithm for the genetic algorithm.

4) Crossover: The crossover operator combines parts from two parent genomes to produce an offspring. In this work, the order crossover [13] is used. This crossover tries to combine segments of the parents while conserving their order.

The procedure is to choose a subsection from one parent, which is copied to the offspring, and then fill in the blanks with the sequence from the other parent less the already used indices. For example, if the genomes

$$\{1, 2, 3, 4, 5, 6, 7\} \qquad \text{and} \qquad \{4, 6, 2, 7, 1, 3, 5\}$$

are combined, choosing the subsection $\{3, 4, 5\}$ from the first genome, they will produce the offspring

$$\{6, 2, 3, 4, 5, 7, 1\} \ .$$

5) Mutation operators: Mutation operators operates on a single genome, modifying the representation in the hope that this may produce a more fit individual. Yu and Hung [10] adopt and extend the inversion and exchange mutations and introduce the shift mutation, which modifies the continuous heading value. The method of their shift mutation is used here to randomly choose headings and velocities. Here, the mutations are called ϕ-random and v-random. The extended inversion mutation of Yu and Hung is adopted here along with their shift mutation, whereas the traditional exchange is

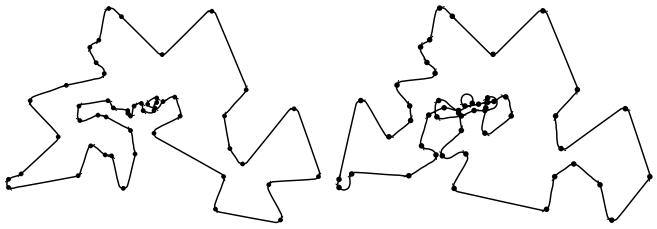

Fig. 4. Two instances of Dubins genetic algorithm solutions to the Berlin52 problem corresponding to the results listed in Table I. Left: Minimum initialized. Right: Maximum initialized. Note that the direction of travel is unimportant, as the acceleration profile is symmetrical around 0.

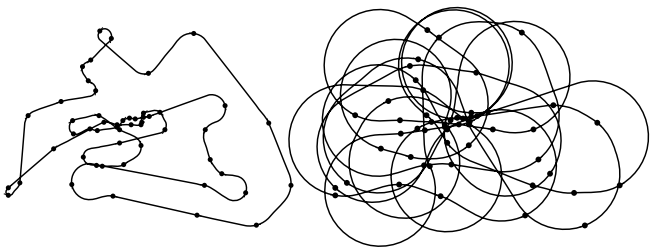

Fig. 5. Two instances of Dubins genetic algorithm solutions to the Berlin52 problem corresponding to the results listed in Table II. Left: Minimum initialized. Right: Maximum initialized.

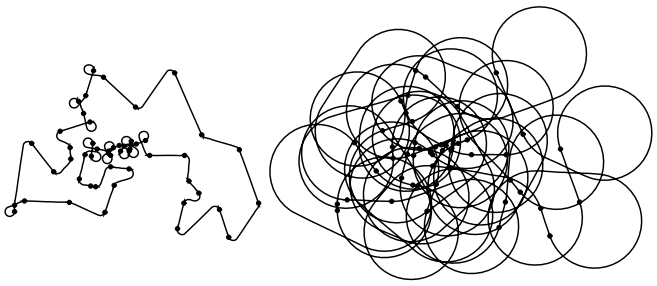

Fig. 6. Two examples of Alternating Algorithm–Euclidean genetic algorithm solutions to the Berlin52 problem, left and right corresponding to the instances in Table I and Table II respectively.

used instead of their extension. Furthermore the traditional displace mutation is also used.

The individual mutations are shortly described here. For a more in-depth review of mutation- and crossover-operators, see [12].

The inversion mutation chooses a subsection of the mutatee and replaces it with itself reversed, the extended version used here also shifts the heading of each of the affected indices by π, reversing the direction of travel.

The exchange mutation simply chooses two genes and exchanges them. Here, the heading is not shifted as proposed in [10].

The displace mutation chooses a subsection of the genome and moves it forwards or backwards in the sequence, also here, the heading is untouched.

The random mutations chooses a gene and sets the continuous values to a random value. The ϕ-random sets the heading to a value in $[0, 2\pi[$, and the v-random set the velocity to a value in $[v_{\min}, v_{\max}]$.

III. RESULTS

The Berlin52 problem from the TSPLIB package [14] serves here as demonstration. It contains 52 locations of interest in Berlin, Germany, spread out in the interval $[(0, 0), (1740, 1175)]$ with about 20 of the points located relatively close around the center and the rest in the periphery. Our proposed method is well suited to this problem because it has both widely spaced and closely spaced waypoints. The traditional Dubins methods with fixed turning radius cannot adapt to the need for both slow and fast turns to accommodate the close and widely spaced waypoints.

A. Simulation results

We have examined two instances of a genetic algorithm optimization. One where the maximum speed is low and one where it is high. The results presented in Table I are for an instance where the possible speeds are in the interval $[0, 100]$ units per second with an acceleration of 10 units per second squared and rotational speed of 3 radians per second. In Table II, the results are given for a setup with the maximum speed and acceleration increased by a factor 10.

This result is compared to solutions obtained by the Alternating Algorithm (AA) of Savla et al. [2]. Such a solution consists of two steps; 1) find the desired order of the waypoints (we use both a Nearest Neighbor and a Euclidean GA), and 2) set the headings of the waypoints in pairs so that the segments connecting the waypoints alternates between a line segment and a Dubins path. The fitness is measured in the same way as for the Dubins GA, that is, acceleration to maximum allowable speed is used on any straight segment, while turns have a constant velocity linearly proportional to the turning radius. Two examples of AA solutions are shown in Figure 6. Here the sequence of waypoints are found using the genetic algorithm with a Euclidean distance cost function. To allow a fair comparison, several instances of the AA solution is evaluated, varying the (fixed) forward speed in the turns from the minimum to the maximum speed, so that the turning radii varies from least to largest. In both examples the maximum speed yielded the fastest solution. Note that setting the forward speed in the turns to zero reduces the turning radii to nothing and equates to a Euclidean solution where the vehicle slows down to a stop, rotates on the spot (i.e. center turns like a tank or hovering helicopter), and speeds up along the next line section towards the next point.

While the first instance with low maximum speed and acceleration gives reasonable results for the Euclidean GA with AA, it has a bit of trouble in the crowded center. The second instance has a turning radius of 333.3 units at the speed of 1000 units per second, which is a lot compared to the individual distances between the points. This large turning radius proves difficult and gives considerable trouble in the entire region. The solutions can be seen in Figure 6.

Six methods are listed in table I. First the nearest neighbor heuristic solution with an AA smoothing with a forward speed of 0 (center turns) and next with a forward speed of 100, which translates to turns with radius 33.3. The next two rows are also smoothed with the AA with forward

TABLE I

COSTS FOR SOLUTIONS TO THE BERLIN52 PROBLEM.

Method	Length	Time	
Nearest Neighbor + 0 AA	8,980.9	541.7	
Nearest Neighbor + 100 AA	11,796.3	118.0	
Euclidean GA + 0 AA	7,835.7	501.8	
Euclidean GA + 100 AA	10,607.0	106.1	*Figure6*
Dubins GA (min. init.)	8,476.1	102.2	*Figure4*
Dubins GA (max. init.)	9,254.3	92.6	*Figure4*

Speed: [0, 100], Acceleration: 10, Rotational Speed: 3

speeds of 0 and 100 respectively, but here the underlying solution is obtained with a genetic algorithm (GA) based on a Euclidean cost function. Lastly, two instances of the Dubins GA are listed. The difference between the two is the way of initialization. The first (minimum initialization) is initialized with a population of identical solutions obtained with the nearest neighbor heuristic, setting the speed and headings of the waypoints to 0. The next entry is initialized just like the other, but with the speeds set to maximum.

All the solutions are evaluated according to two metrics. The first is the traveled distance, i.e. length of the solution, the next is the time it takes to traverse it. Note that the Euclidean GA uses the Euclidean distance as the cost function for optimization, but the distance listed in the table is the distance after AA smoothing. In table II the results of the same instances, but with a higher maximum speed and acceleration are listed.

TABLE II

COSTS FOR SOLUTIONS TO THE BERLIN52 PROBLEM.

Method	Length	Time	
Euclidean GA + 0 AA	7,835.7	246.9	
Euclidean GA + 1,000 AA	73,125.8	73.1	*Figure6*
Dubins GA (min. init.)	10,732.7	50.9	*Figure5*
Dubins GA (max. init.)	39,091.6	39.1	*Figure5*

Speed: [0, 1,000], Acceleration: 100, Rotational Speed: 3

The computational effort of the Dubins GA is considerable compared to the Euclidean GA. The instances of the Euclidean GA were solved in 10,000 generations, where the Dubins GA required 15,000 generations, but because of the more computationally intensive cost function of the Dubins curves, the run time of the Dubins GA was roughly six times as long. Specifically 115 seconds for the Dubins GA compared to 20 seconds for the Euclidean GA on a Intel Core i7 2.80 GHz core.

B. Observations

When examining the results, note that the optimized headings of the waypoints result in the circle segments on either side of the waypoints are of equal length, and further that no path has a left-turn when entering the waypoint and a right-turn exiting (or vice versa). See a close-up of the center section of Figure 4 in Figure 7 with the headings of the waypoints illustrated.

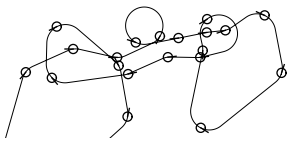

Fig. 7. Close-up of the center section of the maximum initialization instance in Figure 4.

IV. DISCUSSION

In all instances, the Dubins genetic algorithm outperforms the other solutions by matter of time. Interestingly, the method of initializing the algorithm has a strong influence on the obtained solution. The nearest neighbor solutions in Table I, gives some insight to this behavior. The solution initialized with minimum speed produces a short, but slow execution where the maximum speed initialization produces a longer, but faster execution. As the algorithm incorporates a validity check, all offspring with too large difference between the initial and terminal speeds to be realized are discarded. Thus the algorithm may obtain a solution that is trapped in a local minimum, which the mutations are not "strong" enough to break out of.

A possible way to visualize this problem is by thinking of the solution space as a hill with the best solution sitting at the top of the hill. As the solutions are mutated, they may move a step up or down the hill. The selective pressure of the genetic algorithm favors the solutions that are stepping up the hill. The validity check creates boundaries on the hill that the solutions cannot step over, this could be thought of a a form of ravine dividing the faces of the hill. The solution may get stuck on one side of the ravine in a local maximum and have to cross the ravine to get to the global maximum on the other side. If the mutation operators are not "strong" enough to step across the ravine in one step, the solution have to backtrack down the hill to a point where the ravine becomes so narrow that it can be negotiated. The genetic algorithm allows a portion of backtracking, but the probability dwindles as more steps are needed.

This is what seems to happen in the case of maximum- and minimum-initialized initialization. The ordering of the waypoints and their headings are fairly quick at converging to a nice solution, where the speed adaption process are more slow. The ravine is thus created as new changes in heading or ordering may easily become infeasible because of a too large speed. Thus, the solution have to backtrack by first reducing the speed, then changing the ordering, and finally the heading. The probability of this sequence of mutations happening is quite low.

There are different ways to overcome the problem of getting caught in local minima. Here, one of the problems is that the algorithm cannot search from one feasible domain through an infeasible domain to another feasible domain. Such problems are addressed by Ray et al. in [15], where they allow a portion of infeasible solutions in the population. They also note that optimal solutions often lie on the constraint boundaries, an observation that may be seen in this case as well: The maximum-initialized version

maintains the maximum speed throughout the entire path, thus lying on the speed constraint boundary. Conversely, the minimum-initialized version moves on the acceleration constraint boundary; the reason why the speed does not reach the maximum. The approach by Ray et al. bridges or narrows the ravines, easing the transition from one feasible domain to another.

Interestingly, the sub-optimal minimum-initialized algorithm generally produces "pleasingly" looking paths, whereas the maximum-initialized paths tend to look more complex (although faster). Rather than dismissing the minimum-initialized approach we will regard it as a conservative version of the algorithm, where the maximum-initialized can be regarded as the aggressive version. The difference between the two is very apparent in Figure 5 corresponding to the solutions in Table II.

V. FURTHER WORK

The observation that the circle segments seem to be divided on the middle seems intuitive as this results in the shortest straight segments, which definitely minimizes the length. But with the acceleration and speed constraints, the same conclusion is not so obvious. If a closed form solution to this is constructed, there would be no reason for the genetic algorithm to individually optimize the headings as they would be an implicit consequence of the ordering and speed.

Even if no closed form is found, the observation may be used as a heuristic when constructing initial solutions for the genetic algorithm.

In this case, the vehicle is constrained to only accelerating on the straight segments. For a real aircraft, however, it is possible to accelerate during a turn. As described in (2) the radius is defined by the forward and angular speed of the aircraft. If the aircraft is to accelerate in the turns, the radius will become a function of the speed, and the path will become a spiral segment. Planning with such spirals in stead of circle segments would certainly bring the model closer to the real system, and construct even better trajectories. Spirals have been used in planning for aircraft. One often used is the clothoid, which has a curvature that is linear with arc length [16]. While the clothoid is an approximation of the spiral alluded to here, it models a constant forward speed and a bounded angular acceleration.

The differences between the two solutions from the maximum- and minimum-initialized algorithms show that there may be other factors in the optimization of this problem than simply the time consumption. The minimal-initialized solution definitely looks more pleasing to the human eye, even though it is slower. Such a factor may very well be important when motion planning for robots in human environments. This may lead to the use of a multi-objective optimization instead; with both the time and distance as optimization factors. In [17], a genetic algorithm have been developed to handle such problems.

Another approach to this multi-optimization problem is to consider the fuel usage of the trajectory. Such a parameter will combine the two other parameters, as it will strike a balance between the fastest solution and the slowest solution, choosing a speed that is probably closer to the cruising speed of the aircraft, where it has the best mileage.

ACKNOWLEDGMENT

This work is supported by the Danish Council for Strategic Research under grant no. 09-067027 (ASETA). See `www.aseta.dk` for more details.

REFERENCES

[1] L. E. Dubins, "On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents," *American Journal of Mathematics*, vol. 79, no. 3, p. 497, Jul. 1957.

[2] K. Savla, E. Frazzoli, and F. Bullo, "On the point-to-point and traveling salesperson problems for Dubins' vehicle," in *Proceedings of the 2005, American Control Conference, 2005.* Portland, OR, USA: IEEE, 2005, pp. 786–791.

[3] J. L. Ny, E. Feron, and E. Frazzoli, "On the Dubins Traveling Salesman Problem," *IEEE Transactions on Automatic Control*, vol. 57, no. 1, pp. 265–270, 2012.

[4] H. Chitsaz and S. M. LaValle, "Time-optimal paths for a Dubins airplane," in *2007 46th IEEE Conference on Decision and Control*. IEEE, 2007, pp. 2379–2384.

[5] S. Hota and D. Ghose, "Optimal geometrical path in 3D with curvature constraint," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2010, pp. 113–118.

[6] P. Sujit, S. Saripalli, and J. Sousa, "Unmanned Aerial Vehicle Path Following: A Survey and Analysis of Algorithms for Fixed-Wing Unmanned Aerial Vehicles," *IEEE Control Systems*, vol. 34, no. 1, pp. 42–59, 2014.

[7] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. New York: Plenum Press, 1972, pp. 85–103.

[8] S. Lin and B. Kernighan, "An effective heuristic algorithm for the traveling-salesman problem," *Operations research*, vol. 21, no. 2, pp. 498–516, 1973.

[9] K. Helsgaun, "General k-opt submoves for the LinKernighan TSP heuristic," *Mathematical Programming Computation*, vol. 1, no. 2-3, pp. 119–163, Jul. 2009.

[10] X. Yu and J. Y. Hung, "A genetic algorithm for the Dubins Traveling Salesman Problem," in *2012 IEEE International Symposium on Industrial Electronics*. IEEE, May 2012, pp. 1256–1261.

[11] A. M. Shkel and V. Lumelsky, "Classification of the Dubins set," *Robotics and Autonomous Systems*, vol. 34, no. 4, pp. 179–202, Mar. 2001.

[12] P. Larrañaga, C. M. Kuijpers, R. H. Murga, I. n. Inza, and S. Dizdarevic, "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators," *Artificial Intelligence Review*, vol. 13, no. 2, pp. 129–170, 1999.

[13] L. Davis, "Applying adaptive algorithms to epistatic domains," in *IJCAI'85 Proceedings of the 9th international joint conference on Artificial intelligence*, vol. 1. Morgan Kaufmann Publishers Inc., Aug. 1985, pp. 162–164.

[14] G. Reinelt, "TSPLIB–A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, Nov. 1991.

[15] T. Ray, H. K. Singh, A. Isaacs, and W. Smith, "Infeasibility Driven Evolutionary Algorithm for Constrained Optimization," in *Constraint-Handling in Evolutionary Optimization*, E. Mezura-Montes, Ed. Springer Berlin Heidelberg, 2009, pp. 145–165.

[16] A. Tsourdos, B. White, and M. Shanmugavel, *Wiley: Cooperative Path Planning of Unmanned Aerial Vehicles*. John Wiley & Sons, Ltd, 2011. [Online]. Available: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-0470741295.html

[17] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, Apr. 2002.