2014 IEEE Congress on Evolutionary Computation (CEC)
July 6-11, 2014, Beijing, China

Online Generation of Trajectories for Autonomous Vehicles using a Multi-Agent System

Garrison W. Greenwood
Department of Electrical & Computer Engineering
Portland State University
Portland, OR 97207 USA

Saber Elsayed, Ruhul Sarker, Hussein A. Abbass
School of Engineering and Information Technology,
UNSW-Australia at Canberra
Canberra 2600 ACT, Australia

***Abstract*—Autonomous vehicles are frequently deployed in environments where only certain trajectories are feasible. Classical trajectory generation methods attempt to find a feasible trajectory that satisfies a set of constraints. In some cases the optimal trajectory may be known, but it is hidden from the autonomous vehicle. Under such circumstance the vehicle must discover a feasible trajectory. This paper describes a multi-agent system that uses a combination of reinforcement learning and differential evolution to generate a trajectory that is ε-close to a target trajectory that is hidden.**

I. Introduction

Trajectory generation involves defining a set of feasible motions that will move a body (vehicle, robot, etc.) from a starting point to an end point subject to a set of constraints. These constraints arise from any number of sources. For example, a trajectory may have obstacles or no-fly zones that must be avoided [1]. Trajectories should not require a movement—e.g., navigation around a sharp turn—that exceeds a vehicle's capability [2]. In still other cases the trajectory should provide a low probability of detection [3]. All of these examples have the same thing in common: they assume at least one desirable trajectory exists that satisfies all constraints, but it is not known *a priori*. The question then is how can a desirable trajectory be found?

Classical trajectory generation would model the problem as an optimization problem. A solution is a complete trajectory that satisfies all constraints and possibly optimize an objective function, such as minimization of energy consumption. In some applications—as in surveillance operations—the autonomous vehicle does not need to exactly fly-by the way points on this target trajectory. It is sufficient that the vehicle flies within some distance ε of each desired way point. The dynamic nature of the operational environment also makes it difficult for the vehicle to identify all target way points in advance and before take-off. We will call this problem online trajectory generation to differentiate it from the classical trajectory generation problem.

Online trajectory generation is difficult to do in a centralized way because this type of trajectory generation involves several interacting and competing components. This is particularly true with fixed-length trajectories. For instance, changing the morphology (shape) of a fixed-length trajectory moves at least one endpoint location. Similarly moving an endpoint changes the morphology. The generated trajectory must closely match a target trajectory but specific details about the target trajectory are not known.

Since centralized methods are often impractical, a reasonable approach is to decompose the generation processes into a collection of sub-systems or agents that work together to generate the trajectory. A generated trajectory is called *feasible* if the start and stop points are within some distance ε of the desired endpoint locations and its shape is isomorphic to the target trajectory[1]. The agents collaborate to construct segments of the trajectory and adapt their behavior as needed to satisfy local goals. Their collective actions must satisfy the global goal, which is to generate a feasible trajectory.

This paper describes how a *multi-agent system* (MAS) can generate feasible trajectories in $\Re^n$. The target trajectory is hidden in the sense its start point, stop point, length and shape are not known beforehand, but must be discovered through an algorithmic process. The proposed MAS discovers a target trajectory by using differential evolution (DE) to find the start/end points and then generating candidate trajectories. A form of reinforcement learning rates the quality of those trajectories.

The paper is organized as follows. Section II gives a precise definition of the trajectory problem. Section III describes how a candidate and a target trajectory can be compared to measure endpoint locations and path isomorphism. Section IV is particularly important because it describes the MAS approach for generating isomorphic candidate trajectories in a piecemeal manner. Experimental results are presented in Section V. Lessons learned about the MASON agent toolkit (used to design the MAS) are discussed in Section VI.

II. Problem Description

Let $\Theta \subset \Re^\ell$ where Θ is the union of m closed subsets $\{\theta_1, \theta_2, \ldots, \theta_m\}$. Each θ_i represents regions in ℓ-space that are feasible or areas of interest. Let $x_0 \in \theta_i$ and $x_n \in \theta_j$, where $i \neq j$, be a particle moving from x_0 to x_n which follows some trajectory $T : x_0 \rightarrow x_n$. A trajectory that satisfies certain properties (which are problem dependent) is called the *target trajectory* and is denoted by T^*.

[1]"Isomorphic" in this context means any two corresponding sample points in the generated trajectory and the target trajectory are at most distance ε apart.

978-1-4799-1488-3/14/$31.00 ©2014 IEEE

x_0, x_n and T^* are known but not disclosed to a user. Instead, the user must try to identify T^* through an algorithmic process. In abstract terms this algorithm has two phases: in the first phase the algorithm searches for likely start and end points of T^* and in the second phase tries to deduce the shape. It does this by repeatedly generating a candidate trajectory $\widehat{T}$ between the end points found during the first phase. Each $\widehat{T}$ is input to a black box and the black box then outputs a number λ indicating how closely $\widehat{T}$ and T^* match. λ decreases as the degree of matching gets better. λ^{-1} thus corresponds to fitness in an evolutionary algorithm. The objective is to repeat this algorithm until the target trajectory is reconstructed.

III. Approach

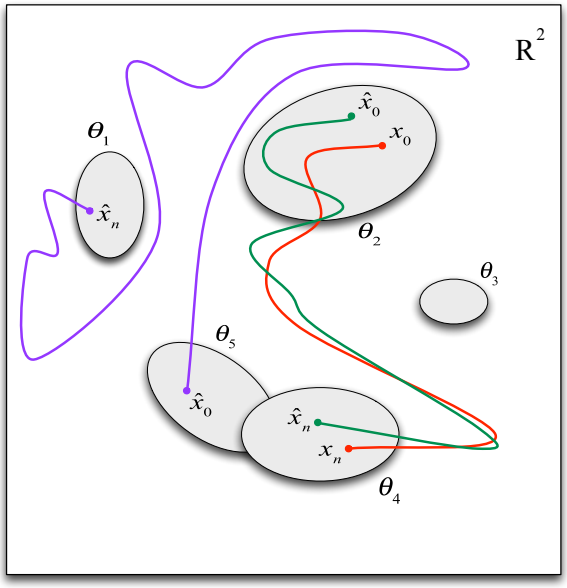

Fig. 1. An example trajectory problem in $\Re^2$. In this example the target trajectory (red line) and the first candidate trajectory (blue line) have considerably different start/end points and the trajectory shapes are completely different. That candidate trajectory would have a large λ value. Conversely, the second candidate trajectory (green line) has relatively accurate start/end points and the trajectory shape is much closer to the target trajectory. The corresponding λ value is much smaller.

Figure 1 shows a simple case in $\Re^2$ with $m = 5$. Of course, as shown in the figure, $\hat{x}_0$ may be located nowhere near x_0, $\hat{x}_n$ nowhere near x_n and $\widehat{T}$ may look nothing like T^*. The λ equation must consider all of these factors. With $x \in \Re^n$, define a vector function

$$F(x) = \sum_{\theta_i \ni x} |\theta_i| \tag{1}$$

where $|\theta_i|$ represents the volume of θ_i in $\Re^n$. Then the equation for λ is

$$\begin{aligned}\lambda = \alpha_1 \left|F(\hat{x}_0) - F(x_0)\right| \; + \; \alpha_2 \left|F(\hat{x}_n) - F(x_n)\right| \\ + \; \alpha_3 D(T^*, T)\end{aligned} \tag{2}$$

where α_j is user defined and $D(\cdot,\cdot)$ is a dissimilarity measure. The first term in Eq. (2) penalizes the difference between the candidate and target trajectory start points. The second term penalizes differences in the end points. The last term reflects the difference in shape between the two trajectories. $D(\cdot,\cdot) = 0$ implies the two trajectories are identical in shape; $\lambda = 0$ means the trajectories are identical in shape and their start/end points coincide.

A number of trajectory distance measures have been proposed to check the similarity of two trajectories. These measures include Euclidean distance, dynamic time warping [4] and a variety of edit distance measures. Euclidean distance has only $\mathcal{O}(n)$ complexity (n is the trajectory length) while the other methods have $\mathcal{O}(n^2)$ complexity. But Euclidean distance requires both trajectories to be of the same length, which does not frequently occur. Also, Euclidean distance is susceptible to noise (as is dynamic time warping). The edit distance measures can handle noise and will work with sequences of different length but the complexity is much greater than that of Euclidean distance. So which distance measure is best? It turns out the answer is analogous to the No Free Lunch Theorem in optimization: no distance measure works best for all situations [5]. Nevertheless, the edit distance measures seem to be gaining in popularity.

The idea behind edit distance is simple. Given two trajectories S and R, how many insert, delete or replace operations are needed to transform S into R? Each time an operation is needed, the edit distance increases by one. For example, consider the two 1-D trajectories $S = \{1\ 101\ 2\ 7\ 4\}$ and $R = \{1\ 2\ 3\ 4\}$ with threshold parameter $\varepsilon = 1$. Two elements are considered "equal" if their difference is less than or equal to ε. Notice, $S \equiv R$ if the 2nd element of S is deleted and the element '7' is replaced by '3'. Hence, the edit distance between S and R is two. The smaller the number of operations required, the smaller the edit distance and the more similar the trajectories.

Without loss in generality, we assume 2-D trajectories for now. Let R and S be trajectories of length m and n, respectively. Then $R = [(t_1, r_1), \ldots, (t_m, r_m)]$, where $r_i = (r_{i,x}, r_{i,y})$. To compute the distance between two trajectories R and S, we use *Edit Distance on Real Sequences* (EDR) [6]. This edit distance measure looks for long common subsequences in the two trajectories. There will be gaps between these subsequences and EDR assesses a penalty proportional to the size of these gaps. The EDR between R and S is given by

$$EDR(R,S) = \begin{cases} m & n = 0 \\ n & m = 0 \\ \min(\gamma_1, \gamma_2, \gamma_3) & \text{otherwise} \end{cases} \tag{3}$$

where

$$\begin{aligned}\gamma_1 &= EDR(\text{REST}(R), \text{REST}(S)) + SubCost \\ \gamma_2 &= EDR(\text{REST}(R), S) + 1 \\ \gamma_3 &= EDR(R, \text{REST}(S)) + 1\end{aligned}$$

and *SubCost* = 0 if $match(r_1, s_1) = true$ and 1 otherwise. Two elements r_i and s_j match if $|r_{i,x} - s_{j,x}| \leq \varepsilon$ and $|r_{i,y} - s_{j,y}| \leq \varepsilon$ where ε is a threshold parameter. The edit distance is calculated using dynamic programming. Thus the complexity of computing edit distance is $\mathcal{O}(mn)$. Notice EDR does not require m and n be equal.

IV. MAS Design Philosophy

The general concept is to build a minimum edit distance candidate path using a 3-agent system. The candidate path is built in a piecemeal manner, one dimension at a time. Before describing the details, it is worthwhile reviewing a concept regarding edit distance. Edit distance equals the number of operators (replace, insert or delete) needed to convert a candidate trajectory into the target trajectory. The replace operator is needed if two corresponding points in the two trajectories differ in magnitude by a value greater than ε. The two trajectories do not have to be the same length because the insert operator (or conversely the delete operator) can be applied to make the candidate and target trajectories equal length. This latter point is key to the piecemeal construction technique.

The objective is to create a candidate trajectory that matches point for point with a target trajectory in N-space. The idea is to create a candidate trajectory that is ε-close to a target trajectory at (typ.) $k = 10$ sample points. The candidate trajectory is built piecemeal one dimension at a time.

A number of agent-based modeling (ABM) software toolkits exist and several papers that compare them exist. We chose the MASON [7] toolkit for this work primarily because it has a straightforward scheduler for the agent tasks and, unlike some ABM toolkits, it does not require a spatial orientation for the agents.

TABLE I
Agent Definitions

Agent	Notation	Action
1	A_ε	adjust ε
2	A_s	adjust segment length
3	A_c	retrieves edit distance; changes path generation dimension

A. Agent Behavior

The candidate path is generated one segment at a time. It begins by adding a line segment with one end-point at $\hat{x}_0$. This segment lies in only one dimension (say x). Three agents are needed to generate the candidate path (See Table I.) The process begins with the first agent (A_ε) making ε initially large and the second agent (A_s) creating a short trial line segment towards the next sample point. A large initial value for ε makes the initial line segment length ε-close to the x coordinate of the next target trajectory sample point. That means $k - 1$ replacement operations are needed to make the candidate and target trajectories of equal length. Hence, the edit distance is $k - 1$. The first agent then slowly decreases ε until the segment end-point is no longer ε-close. This event occurs when the edit distance suddenly increments to k. The second agent now increments the segment length until end-point is again ε-close to the target sample points indicated by the edit distance drop back to $k - 1$. The first agent reduces ε again and the two agents work back and forth, alternating between reducing ε and increasing the line segment length. This process continues until the segment end-point is within some user-defined minimum ε_0 to the next target sample point. This iterative process is shown in Figure 2.

The third agent is a control agent. At each time step, this agent obtains the location of the line segment end-point from the second agent. It then sends the end-point location to the black box and retrieves the edit distance. The edit distance from the previous time step is stored so it is easy to see if the edit distance incremented, decremented or remained the same. In either of the first two cases an event is generated (see below). Eventually ε becomes less than a predefined lower bound ε_0. At that time the control agent informs the other two agents no further updates are needed in that dimension. The control agent then tells the other two agents to move to the next dimension (say y) and repeat the process. The process continues until all dimensions are covered. The candidate trajectory now has one line segment with one end-point at the target trajectory start (x_0) and the other end-point ε_0-close to the first target trajectory sample point in all N dimensions. This latter end-point becomes the end-point of the next line segment that will be generated. The process repeats until the entire candidate trajectory has been placed.

Each agent is reactive, which means it maps sensors input into actions. The behavior of the first two agents is described by the state diagrams depicted in Figure 3. Note these are not finite state machine diagrams because the arc labels are events and not inputs. For example, in Figure 3(a) the agent transitions from a state s_1, where ε decreases at each time step, to a state s_2 where it remains constant. This transition takes place when the event "edit distance increments" ($ED \uparrow$) is detected. It transitions back to state s_1 where it starts decrementing again once the event "edit distance decrements" ($ED \downarrow$) is detected.

B. Generating Trajectories

Generating a trajectory consists of two phases. In the first phase, a search is conducted to find the trajectory end-points. The second phase involves the path generation between the end-points. The objective is to find end-points within some ε_0 of the target trajectory (T^*) end-points and then find a path shape that matches the T^* shape. Referring back to Eq. (2), the first two terms depend on the end-point locations while the third term depends on the edit distance between a candidate path and T^*.

During the first phase an evolutionary algorithm called *differential evolution* (DE) was used to search for the two trajectory end-points [8]. DE is a population based stochastic

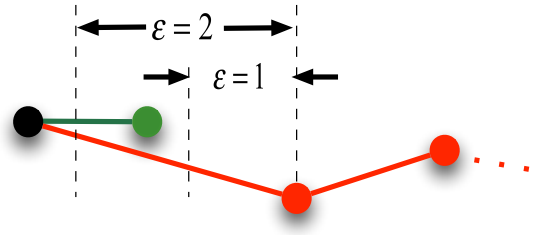

Fig. 2. An example of agent-based path generation. The black node is the starting node ($\hat{x}_0$) and the red path is the target path. The green node is a candidate node. One agent moves the green node while a second agent controls the threshold ε. The location of the green node decrements the edit distance when $\varepsilon = 2$ but increments the edit distance when $\varepsilon = 1$. A new line segment is then generated from $\hat{x}_1$ and the process described above is repeated. The difference is now the edit distance switches between $k-1$ and $k-2$. Once completed a candidate trajectory with k sample points will have been generated. All points in the candidate path are ε_0-close to the corresponding sample points in the target trajectory.

search algorithm. The population consists of N individuals where in trajectory problems each individual I_j, $j = 1 \ldots N$, consists of a d-dimensional real vector representing a location in d-space. The initial population is randomly initialized and the algorithm runs for a fixed number of iterations. During each iteration, new individuals are created by a stochastic process. Specifically,

$$I'_{i,j} = \begin{cases} I_{a_1,j} + \xi.(I_{a_2,j} - I_{a_3,j}) & \text{if } (rand \leq CR \\ & \text{or } j = j_{rand}), \\ I_{j,j} & \text{otherwise.} \end{cases} \quad (4)$$

where a_1 is a random integer number $\in [1, N/2]$ [9], while a_2 and a_3 are random integer numbers $\in [1, N]$, $a_1 \neq a_2 \neq a_3$, $rand \in [0, 1]$, ξ is an amplification factor $\in [0, 1]$, the crossover rate (CR) $\in [0, 1]$, and $j_{rand} \in$ 1,2,...,number of variables is a randomly selected index.

Fitness is a measure of the quality of a solution. For this problem the fitness of an individual is computed by extracting the coordinate location and then computing λ^{-1}. High fitness means the proposed location is near the true end-point of T^*. Suppose we are searching for the starting point $\hat{x}_0$. Referring to Eq. (2), only the first term is of interest. By assuming $\hat{x}_n = x_n$ the second term equals zero. This also forces all candidate trajectories to have the same number of insert operations so the third term is constant. Thus only the first term affects λ. Once the $\hat{x}_0$ is found the first term equals 0 so a search can begin for $\hat{x}_n$ again with a constant third term; only the second term now affects λ.

As stated above, during each iteration each individual I_j creates an offspring I'. The fitness of the parent and offspring are computed and the higher fit individual is kept. Typically 400 iterations are sufficient to find a good end-point location although the algorithm can terminate early if an end-point is found within ε_0 of the target trajectory end-point. The best fit individual in the final population is chosen as the end-point.

The candidate path generation is conducted a dimension at a time by extending a line segment from the previous fixed point This previous fixed point is ε-close to the corresponding target trajectory point. Since this previous fixed point is on the real number line, the line segment either extends to the left (decreasing values) or to the right (increasing values)[2].

Figure 3(b) shows the second agent, which controls the segment line length. Two behaviors are depicted. In most cases the agent's behavior will follow the state diagram with the green nodes. That is, the segment length extends from the sample point to the right until the edit distance decrements. This length then will not change until the event $ED \uparrow$ is detected. However, it is entirely possible the target trajectory switches direction (doubles back), for instance, to avoid an obstacle. In that situation the line length should be extending to the left instead. The brown set of nodes covers that particular case. The dashed red line shows that the agent's behavior can switch between the two state diagrams depending on which direction the segment extends from the sample point.

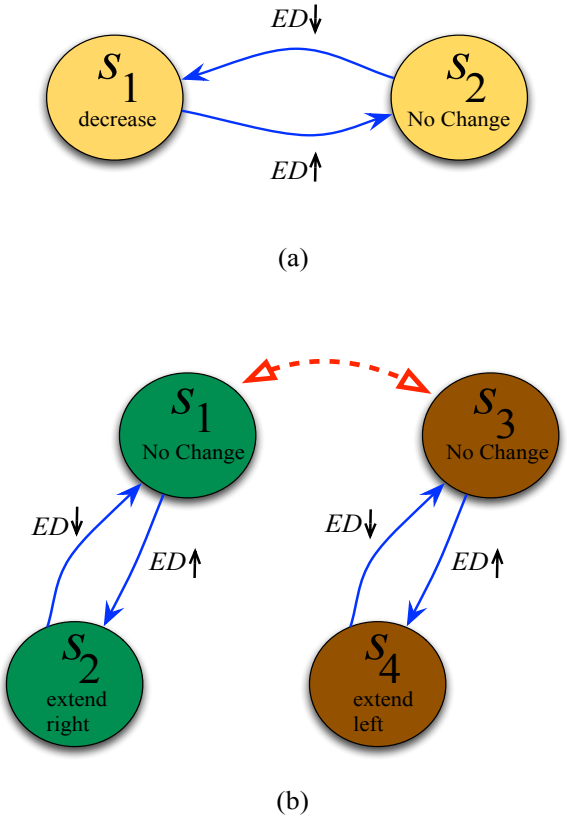

Fig. 3. Agent definition state diagrams: (a) agent controls ε size and (b) agent controls candidate path segment length. Each state has a state ID and the action performed by the agent when in that state. Labels on the arcs indicate events that cause the state transitions. Agent in (b) has two potential behaviors (see text).

All agents have access to a boolean event register. This register, depicted in Figure 4, has 5-bits for each dimension in

[2]Consider the case where the target trajectory "doubles back" to avoid an obstacle. In that case the value of the next sample point will be less than that of the current sample point, which means the line segment should extend to the left.

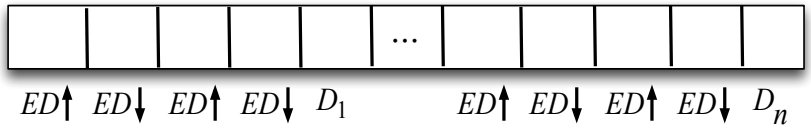

Fig. 4. Event register used in MAS. There are 5-bits in the register for each dimension in the trajectory.

the trajectory: an $ED\uparrow$ and $ED\downarrow$ event bit for the two agents and a D_j event bit for the j-th dimension. Initially, these bits are cleared indicating no event has occurred. The respective bit is set to indicate an event occurred. For instance, when the edit distance increments, the $ED\uparrow$ bits are all set. The two agents monitor their respective $ED\uparrow$ event bits each time step to see if occurred; the event bit is cleared after reading it. Thus, the control agent sets the edit distance event bits whereas the other two agents clear those event bits automatically by reading them. Only one D_j event bit at a time is set by the control agent indicating which dimension the other two agents work in. Only the control agent can set or clear the D_j bits.

The control agent manipulates the event register and sends the current candidate path segment location to the black box to get the edit distance. Since the edit distances from the previous time step is stored it is trivial to see if an $ED\uparrow$ or $ED\downarrow$ event has occurred. The control agent is also responsible for outputting the candidate path sample points. The behavior of the control agent is described in Procedure 1.

Algorithm 1 A_c Agent Behavior

Require: $D \leftarrow dim$ {# of path dimensions}
1: **while** $D \neq 0$ **do**
2: Clear all events
3: generate dimension D event
4: $ED_{old} \leftarrow$ initial edit distance
5: **repeat**
6: $ED_{new} \leftarrow$ current edit distance
7: **if** $ED_{new} > ED_{old}$ **then**
8: generate $ED\uparrow$ event
9: **end if**
10: **if** $ED_{new} < ED_{old}$ **then**
11: generate $ED\downarrow$ event
12: **end if**
13: **until** $ED_{new} == 0$
14: $D = D - 1$
15: **end while**
Ensure: Candidate path sample points

The last issue to consider is direction of each segment. Suppose the location of trajectory point x_j has just been found. The question is should the next segment to the next point x_{j+1} extend to the right ($x_{j+1} > x_j$), extend to the left ($x_{j+1} < x_j$) or remain the same ($x_{j+1} = x_j$)? The answer is determined by a collaboration between agents A_c and A_s.

The process begins by generating a trial segment with a point x' slightly greater than x_j. This then defines two candidate paths: path #1 with $x_{j+1} = x_j$ and path #2 with $x_{j+1} = x'$. Agent A_c sets ε small so the edit distance for both paths is the same. ε is then slowly increased until the edit distance decreases. If the edit distance for path #1 is less, then the next segment extends to the left. If path #2 is less then the next segment extends to the right. If the edit distance is the same for both paths then $x_{j+1} = x_j$ for the next segment.

V. Experimental Results

In this research, we used $\xi = CR$ =0.95 during the search for the trajectory endpoints. In these initial experiments each individual in the population is a 3-D real vector although the extension to arbitrary dimensions is trivial. The population size was 25. There are two possible stopping criteria: (1) the DE found endpoints within distance $\varepsilon = 0.1$ of the target trajectory endpoints, or (2) a maximum of 400 generations were processed. In almost all cases, the DE terminated because the first criterion was satisfied.

The fitness function is λ^{-1}. Although Eq. (2) has three terms only, the first two terms are relevant for assigning fitness to an individual in the population. However, the third term (the edit distance) does vary depending on how the search progresses. Assume the trajectory has n sample points (two endpoints plus $n-2$ intermediate waypoints). The DE conducts a search for both trajectory endpoints simultaneously so, effectively, we are computing the edit distance between a target trajectory of n points and a candidate trajectory of 2 points. Hence the edit distance will have a lower bound of $n-2$ because $n-2$ 'insert' operations are needed. The edit distance term could be one or two more depending on whether the endpoints are within ε of the target trajectory endpoints because, if not, this adds one or two 'replace' operations. The fitness landscape itself is bi-modal but the DE was able to find the endpoints without incorporating anything special to promote niching.

To test the algorithm, we generated test cases with different number of segments (*i.e.* trajectory length) varying between 2 and 20 segments in a step of 2. The generation of each test case was repeated 10 independent times to generate a total of 100 test problems. All problems were generated within the same fixed airspace volume.

The average running time of the algorithm over the 10 problems generated for each test case was then computed and is shown in Figure 5. An exponential curve was found to best approximate the relationship between the number of segments and computational time with $R^2 = 0.98$, where R^2 is the coefficient of determination indicating how well the data points fit the curve. The equation of the curve is

$$p = 4.823 \times e^{0.0959 \times s} \quad (5)$$

where p is the average processing time, and s is the number of segments in a trajectory.

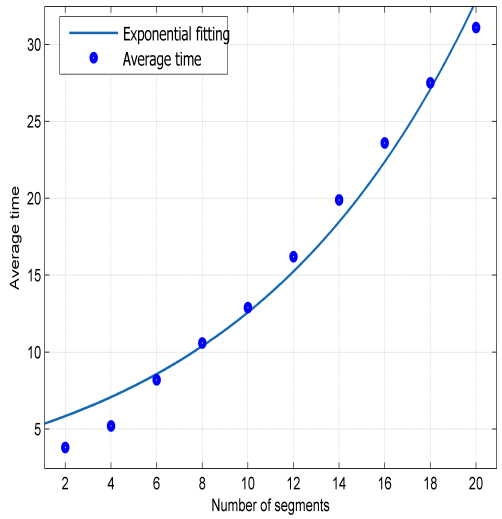

Fig. 5. The relationship between the number of segments in a trajectory and computational time.

In Figure 6, we also studied the impact of ϵ on error, and the relationship between error rate and computational time. Here, "error" is defined as the area between the found trajectory and the target trajectory. This area is zero when both trajectories match exactly. It is obvious that this error will decrease when ϵ decreases, because ϵ is in effect a tolerance factor. However, what is important in Figure 6 is the trade–off between computational time and error. This trade–off is critical in real–world situations and the choice of an appropriate level of trade-off is a context–dependent decision.

In Figure 6, we fixed the number of segments to 20 and only varied ϵ. Time is reported in seconds.

Three example trajectories were evaluated. Figures 7, 8 and 9 show the results for target trajectories with 10, 14 and 20 segments, respectively. The trajectories were restricted to a 3D-box of size 25×25×25. Notice in all cases the candidate trajectory is isomorphic to the target trajectory despite the presence of several radical turns. This illustrates the MAS can correctly generate trajectories that avoid obstacles.

Our decision to pick these radical turns was done with a specific purpose in mind. In surveillance applications, an autonomous vehicle may require to hoover over an object, suddenly change its direction to comply with a complex maneuver strategy, or revisit close–by areas in a random manner. Therefore, the morphology of the generated trajectory is normally different from a classical aircraft trajectory.

VI. Discussion

We have described a MAS that generates a trajectory matching an unknown target trajectory using reinforcement learning. Although the examples shown are 2-D trajectories, the extension to trajectories in an arbitrary number of dimensions is trivial requiring no real modifications.

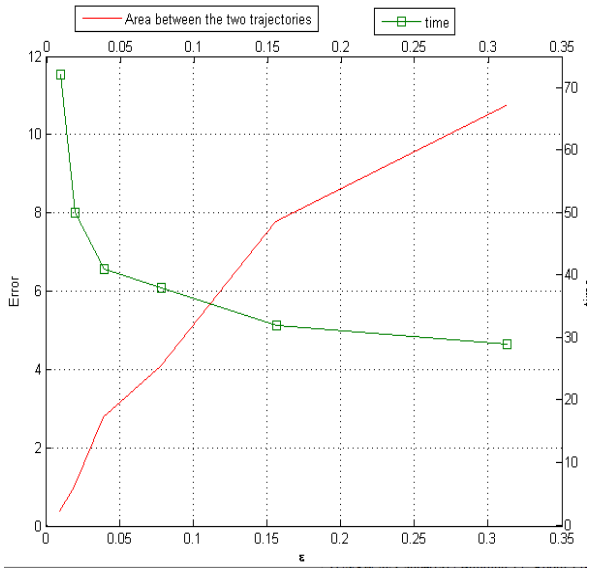

Fig. 6. The relationship between deviations from target trajectory and computational time as ϵ varies.

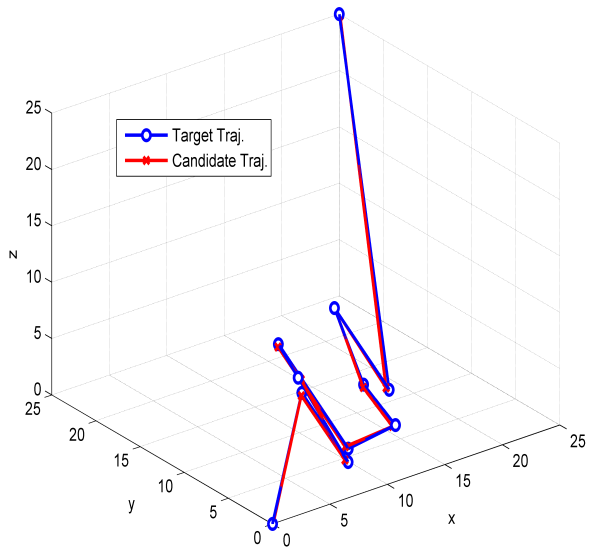

Fig. 7. Results for a test problem with a 10–segment target trajectory. Blue line represents the target trajectory, while the red line represents the found trajectory.

Normally calculating the edit distance between two trajectories of length m and n takes order $\mathcal{O}(mn)$ time. Most of this cost is due to insert and delete operations and running a dynamic programming algorithm. However, in our experiments $m = n$ so no insert/delete operations are required and a dynamic programming algorithm is unnecessary. All that is required when $m = n$ is a series of $match(\cdot, \cdot)$ calculations, defined in Section III, which can be done in $\mathcal{O}(m)$ time. However, in many situations the two trajectories will not be

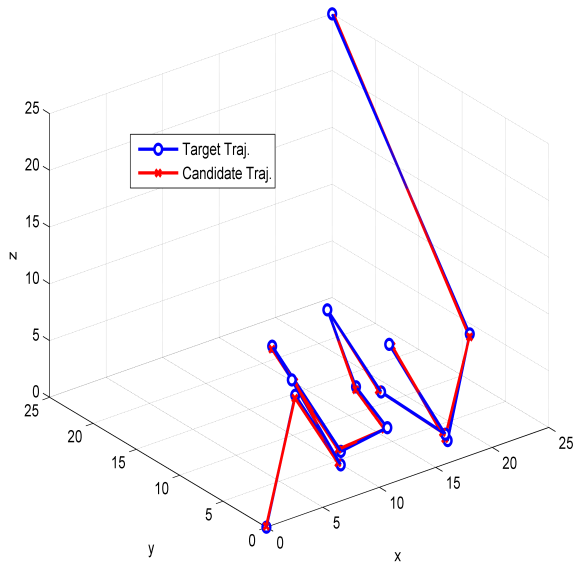

Fig. 8. Results for a test problem with a 14–segment target trajectory. Blue line represents the target trajectory, while the red line represents the found trajectory.

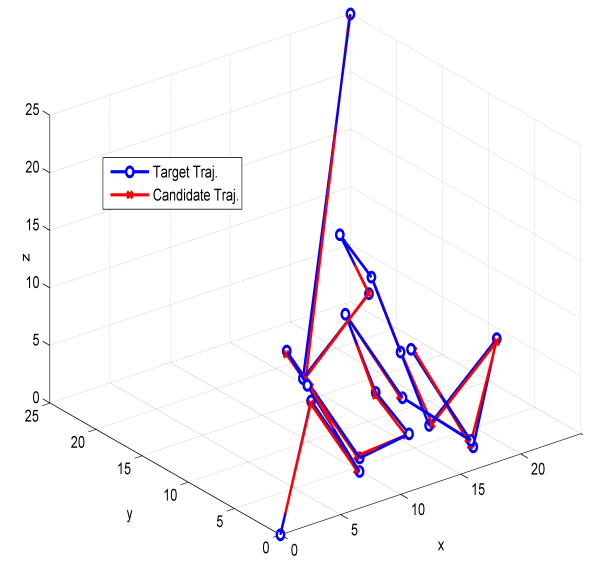

Fig. 9. Results for a test problem with a 20–segment target trajectory. Blue line represents the target trajectory, while the red line represents the found trajectory.

the same length. For instance, in some cases many T^* sample points may be available, but only a few of them are actually needed to construct a feasible trajectory for an autonomous vehicle.

We used the MASON [7] ABM toolkit to develop the MAS. Although the scheduling of agent tasks is simpler than other toolkits, MASON has no inherent communication mechanism. It is for this reason the event register was created. Using an event register provides an easy form of inter-agent communication. Agents can monitor bits in this register to communicate not only with each other but to form precepts about the environment. Only minor modifications are needed to allow multiple agent behaviors so adaptive agent behavior was straightforward.

We demonstrated the performance of the algorithm on a number of synthetic test problems. The complexity grows exponentially. However, the exponent is a very small number (0.09); therefore, the exponential growth is not too bad for practical applications. The algorithm is unique in its ability to incrementally build–up the trajectory. In our future work, we will extend this research effort to include operations for guidance and control of autonomous vehicles for specific surveillance purposes.

REFERENCES

[1] Y. Xie, L. Liu, G. Tang, and W. Zheng, "Highly constrained entry trajectory generation," *Acta Astro.*, vol. 88, pp. 44–60, 2013.

[2] T. Howard and A. Kelly, "Optimal rough terrain trajectory generation for wheeled mobile robots," *Int'l. J. Robotics Res.*, vol. 26, no. 2, pp. 141–166, 2007.

[3] T. Inanc, K. Misovec, and R. Murray, "Nonlinear trajectory generation for unmanned air vehicles with multiple radars," in *Proc. 43rd IEEE Conf. on Decision & Control*, 2004, pp. 3817–3822.

[4] A. Kassidas, J. MacGregor, and P. Taylor, "Synchronization of batch trajectories using dynamic time warping," *AIChE J.* , vol. 44, no. 4, pp. 864–875, 1998.

[5] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh, "Querying and mining of time series data: experimental comparison of resentations and distance measures for time series data," in *Proc. PVLDB*, 2008, pp. 1542–1552.

[6] L. Chen, M. Ozsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proc. SIGMOD 2005*, 2005, pp. 491–502.

[7] S. Luke, "Multiagent simulation and the MASON library," http://cs.gmu.edu/˜eclab/projects/mason/.

[8] S. Das and P. Suganthan, "Differential evolution: a survey of the state-of-the-art," *IEEE Trans. Evol. Comp.*, vol. 15, no. 1, pp. 4–31, 2011.

[9] R. Sarker, S. Elsayed, and T. Ray, "Differential evolution with dynamic parameters selection for optimization problems," *IEEE Trans. Evol. Comp.*, (accepted, in press).