On Simulation-Based Adaptive UAS Behavior During Jamming

Ronnie Johansson[1], Peter Hammar, and Patrik Thorén

Abstract— **We address the issue of autonomously planning a flight path for a remotely controlled surveillance aircraft when control is lost due to jamming. An optimization problem arises where we want the aircraft to continue surveying in the jammed area so that a possible attack does not go unnoticed, but we want the aircraft to leave the jammed area (and report any collected information) while there is still time to respond and take defensive measures.**

We formulate this as a stochastic approximation problem, involving state parameters and a discrete set of path options (specifying candidate Bezier curves), and train on simulated data from realistic scenarios. The result is a discussion how to acquire a policy which considers both realistic tactical scenarios with varying initial values and simulated sensor characteristic.

I. Introduction

Remotely piloted aerial systems (RPAS) are dependent on a stable data link to maintain remote control and payload data feedback. A defense task for such a RPAS is territorial surveillance where the payload is a radar system capable of detecting hostile cruise missiles.

A malevolent adversary launching an attack with cruise missiles is interested in disrupting the RPAS surveillance task by jamming the radio link to mitigate the uplink remote control and the down-link of sensor data[2]. When the communication is jammed, the aircraft effectively becomes an unmanned aerial system (UAS) and has to operate autonomously.

A standard approach for a RPAS that has lost its communication link is to travel to a pre-specified location (such as the home base) where the operators are thought to be able to reconnect with the platform. This location is preferably selected with care so that the aircraft causes as little damage as possible if it runs out of gas or battery.

This simple autonomous behavior can be very useful in many cases, but becomes problematic if the automatic abortion of the mission is the primary goal of the adversary. Hence, an adversary can take control of the RPAS (in the sense of sending it to its pre-specified location) simply by jamming the communication.

What we are considering here is a situation where the RPAS is performing a routine surveillance task over sea along a coast line to make an early detection of cross-border attacks. Human uplink control is typically not necessary but the downlink of sensor data is monitored continuously. During the task execution, the RPAS will be jammed by an adversarial force, i.e., a boat with jamming equipment at sea, and the communication link is lost and the autonomous behavior engages. About the same time, the adversary initiates an attack by cruise missiles on strategic targets in the defended country.

It would be desirable if the aircraft, while moving out of the jammed area, continue to survey for attacks. Any detected cruise missiles will be reported by the aircraft as soon as the border of the jammed area is reached. We make the simplifying assumptions that the UAS can locate the jamming boat, the transmission strength of the jammer, and that a boundary between the jammed and non-jammed area can be calculated.

UAS path planning is hardly a new problem. It has for instance been applied for cooperative search and localization [1] and offensive attacks in hostile environments [2]. However, as far as we know, our particular problem of path planning to autonomously recover from intentional jamming has not been addressed.

We approach this problem by looking for a *policy*, i.e., a function that given a situation description outputs recommended control parameters that define a flight path for the aircraft. To find a suitable policy for all circumstances is, however, challenging as many complex conditions are involved including the initial location of the aircraft when it is being jammed, how many cruise missiles are attacking, and radar features. We capture the variations in these conditions by computer simulation using the FLAMES simulator[3].

To find a policy, we propose a machine learning method based on stochastic approximation. However, given the conditions of the project, in the paper we ultimately don't recommend and evaluate a certain policy. Instead of making too many assumptions about priorities aircraft operator, we settle with reasoning about how to look for an appropriate policy in the methodology that we propose.

Section II presents our machine learning approach. In Section III, we model the problem of representing aircraft flight paths mathematically. Section IV explains the simulations. Section V, presents the results of appying the methodology, and Section VI some conclusions.

II. Machine learning approach

In this paper, we choose to approach UAS flight path planning by learning a policy. Alternatively, a tactical expert

[1]Swedish Defence Research Agency (FOI), Unit of Decision Support Systems, 16490 Stockholm, E-mail: ronnie.johansson@foi.se

[2]We could consider three types of jamming: communication up and down-link, aircraft GPS, and air craft radar sensor. In this work, we address the jamming of the communication link, but assume the radar sensor to be functional. If also the GPS is jammed, the current location which is known can be used and dead reckoning be used to estimate the aircraft location

[3]http://www.ternion.com/

or aircraft operator could possibly specify a policy manually. However, conditions are complex, even for experts: the aircraft may be equipped with advanced sensors with hard-to-grasp properties, and style of attacks may vary. In the learning approach, experts have a possibly easier and more appropriate contribution. They can help design attack scenarios, by specifying adversarial doctrine and selecting sensor types for simulation.

A policy is a function $p : S \rightarrow A$, where S is a set of system states and A a set of actions. In our case, we look for a policy that maps world states (S) to flight path parameters (A).

A popular machine learning method to use for policy learning is *reinforcement learning* (RL) [3]. RL prescribes a sequential decision making, i.e., that the system makes repeated decisions, and that for each decision only the current state is considered (not the earlier states and decisions), i.e., the Markovian property. Now, say that states in our case mean locations of the aircraft and actions mean aircraft heading. Then, since past actions are not considered, if we were to use the RL policy we would risk ending up with erratic paths. To avoid erratic paths, instead of selecting a sequence of small heading corrections, we suggest to select a full flight path directly in just one decision (from a restricted family of reasonable paths).

We find RL to be on the one hand unnecessarily expressive and complex to solve our problem and on the other hand, as argued, we may end up with unpredictable flight times to the border of the jammed area. Instead we turn to a conceptually more simpler *stochastic approximation* formulation [5]. The objective with stochastic approximation is to solve optimization problems when sampled functional values are noisy. It could involve a function $F(x, \xi)$ which represents a utility function, where x is selectable value and ξ a random or unknown value which affects the utility value.

In our case, we define the generic utility by $F(\xi; s, a)$, where

- F - utility of a flight path out of the jammed area
- s - UAS state
- a - flight path parameters
- ξ - random variable affecting the outcome

The utility of a flight path will not be deterministic given only the state of the UAS and a selected action, since other factors such as difference in attack scenario affects the result.

We now define the policy in the following way:

$$p(s) = max_{a \in A} \mathbb{E}^{p(\xi)} \left[F(\xi; a, s) \right], \qquad (1)$$

i.e., given a UAS state s, select an action that maximizes the expected utility. Since $p(\xi)$ is unknown, we use simulations to pick samples of F to approximate Eq. 1.

When defining the state and action sets, we first note that the learning time will increase with the number of options of A and S. Hence there is a trade-off between a large number of options (which provides flexibility and richness) and a small number (faster learning).

When it comes to the number UAS states, although there surely are advantages to divide the flight area of the UAS into several different regions, we didn't see a natural way to divide the region into subregions, especially since the autonomous flight path will be created dynamically based on the location of the jammer. Instead we decided to remove the parameter s from F and, hence, simplify it to $F(\xi; a)$, which results in the same path parameters regardless of the state of UAS. Although the learned policy action is constant and indifferent to the current location of the UAS, the actual path generation uses the selected option a will integrate the UAS location, as we will discuss in the next section.

III. Flight path modeling

In the previous section, we suggested learning a policy described in Eq. 1. The policy should be applied in the case the UAS loses its radio link to its base station. In this section, we describe the set of options for the set of flight path parameters, A.

We first consider the type of flight paths to consider. Different types of paths are considered in the literature, some included in [4]. Among the possible options, we select one (the family of Bezier curves [6] named after Renault engineer Pierre Bézier) which turn out to be useful for our purpose.

Bezier curves are characterized by their *control points*, $P_0, \ldots, P_n$, where P_0 and P_n define the end points. A Bezier curve of the n-th order can be denoted by $B^n(t), 0 \leq t \leq 1$, B^n, and $B^n(0) = P_0$ and $B^n(1) = P_n$. Bezier curves (B-curves) typically don't pass through any of the other control points. An example of a 2nd order B-curve, a so called *quadratic* B-curve, is shown in Fig. 1.

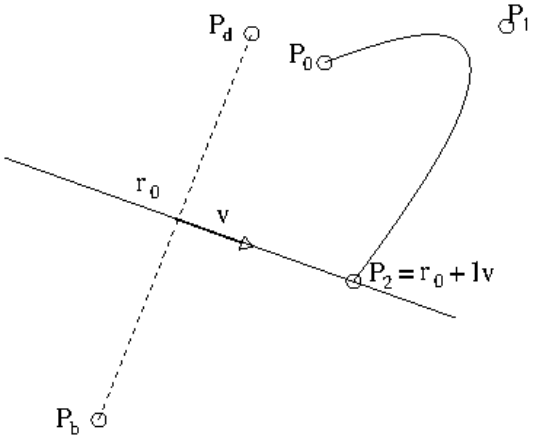

Fig. 1. The boundary of the jammed area is modeled as the line $r(l) = r_0 + lv$. We plan a flight path, a quadratic Bezier curve, from the aircraft's current location (P_0), using a learned direction (P_1), and the end point (P_2) constrained by the line $r(l) = r_0 + lv$.

In our work, we only consider quadratic B-curves (Eq. 2). Higher order B-curves can express more complex paths, but are at the same time more difficult to use with their additional degrees of freedom. A consequence of higher-order B-curves is that there might be multiple solutions for the same flight parameters which introduces additional uncertainty in the learning process.

$$B^2(t) = (1-t)^2 P_0 + 2(1-t)tP_1 + t^2 P_2 \qquad (2)$$

Now we finally arrive at the problem of defining the action parameters of A. Note that in our case, P_0 denotes the current location of the aircraft at the time it is beginning to be jammed, and P_2 should be on the boundary of the jammed area. We have however no obvious selection of P_1 and the boundary of the jammed area is not known until the jamming starts. Also the distance between the aircraft and the boundary is not known in advance and it is therefore not appropriate to learn P_2 from data as they can only be determined at the time of jamming.

We include the control point P_1 as part of the action option which we believe is suitable as the attack can only come from a few different directions. This is illustrated in Fig. 2 with a red colored jamming boat in the middle and the aircraft with a planned path in the left bottom corner. Surrounding the surveillance area is a circle (the whole circle is not shown here) consisting of a limited number of directions. In our experiments, we use ten direction points, i.e., $Directions = \{D_0, \ldots, D_9\}$.

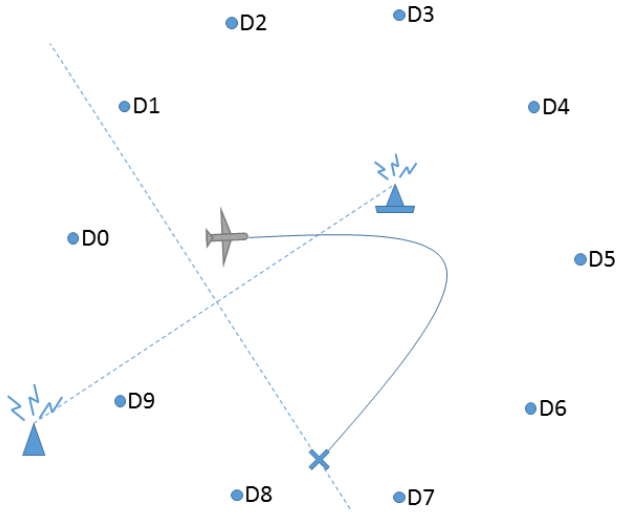

Fig. 2. Illustration of simulation scenario. $D0$-$D9$ corresponds to direction points, the triangle in lower left corner is home base antenna, and the triangle on right hand side is an adversarial jamming antenna. The dashed lines visualize the separation vector from the base antenna to the jamming antenna, perpendicular to this (located along the vector according to the relative strength of the antennas) is the separation line between jammed and non-jammed areas. The UAV is illustrated heading on a Bezier curve towards the cross, according to principles of Fig. 1.

The direction point is one part of our action parameters, the other is flight time, i.e., how long time the aircraft travels in the jammed area before it leaves it can report any observations to the base station. Also in this case, we discretize and let $time = \{17, 19, \ldots, 33\}$, where the numbers represent flight time in minutes. Hence, our set of action options A consists of all combinations of direction points and flight times, i.e., $A = Directions \times time$.

In summary, if the UAS becomes jammed, it selects the learned action parameters (consisting of flight direction D^* and time τ^*) that was learned through training and calculates a flight path based on those parameters.

Since the number of action options is limited by our definition of A, viz. $|A| = |Directions| \cdot |time| = 90$, we decided not to explore the action space (as would be typical for reinforcement learning) but to exhaustively investigate the performance of each possible action option. For each attack scenario and action option investigated, we ran 50 simulations[4] (with pre-generated random scenario parameters) and measured the number of detected cruise missiles.

We still need to show how the path is generated from selected action parameters. As mentioned, the P_0 B-curve endpoint is directly given by the current aircraft position, P_1 is the direction point of the selected action a, but we still need to calculate the end point P_2. We furthermore assume that the jamming source can be located, P_d in Fig. 1, and the base station, P_b. We make the simplifying assumption that boundary of the jammed area is in the middle between the jammer and the base station, represented by the middle point r_0 and the perpendicular vector v. This constrains the possible values of P_2 to the line $r(l) = r_0 + lv$, where the selection of P_2 is defined by the value of $l \in (-\infty, \infty)$.

A pseudo-algorithm for calculating P_2 (Alg. 1) is included in the Appendix. Finding the parameter l is dealt with in a separate algorithm (Alg. 2) after transforming the selected allowed travel time τ^* to a path length c^* (assuming an average speed). Then the l^* that generates the sought path length can be found.

For quadratic B-curves, an analytic solution for their length exists but the solution is complicated and difficult to differentiate (we need to differentiate in order to find a solution for l^* as explained below) [7]. Instead, we approximate the B-curve by a polygon which intersects the B-curve at certain points. $L(B, k)$ represents this approximation for the length of the B-curve B which sums the distances between k equidistant points from $B(0)$ ending in $B(1)$. P_2 is replaced by $r_0 + lv$, where l is the line parameter that we are looking for (to derive a P_2). The 2-D points have components with x and y subscripts, e.g., $P_0 = (P_{0,x}, P_{0,y})$. We let $L(l; B, k)$ denote the approximated length of a B-curve B divided into k segments (and where P_2 has been replaced by $r(l)$).

$$
\begin{aligned}
L(l; B, k) &= \sum_{i=0}^{k-1} ||B(\frac{i}{k}) - B(\frac{i+1}{k})|| = \\
&= \ldots \\
&= \frac{1}{k^2} \sum_{i=0}^{k-1} \left(\left(\alpha_x^i + l\beta_x^i\right)^2 + \left(\alpha_y^i + l\beta_y^i\right)^2 \right)^{1/2}, \\
\text{where} \quad & \alpha_\bullet^i = P_{0,\bullet}(1 - 2(k-i)) + \ldots \\
& \ldots + P_{1,\bullet} 2(k - 2i - 1) + r_{0,\bullet}(2i+1) \\
\text{and} \quad & \beta_\bullet^i = v_\bullet(2i+1)
\end{aligned}
\tag{3}
$$

Note that all parts of $L(l; B, k)$ in Eq. 3 are constant except for the line parameter l. Now that we can calculate the length of the path to the jamming boundary for each possible B-curve that ends on $r(l)$, we will look for an l^* that satisfies the equation $f(l) = L(l; B, k) - c^* = 0$.

[4]Hence, in total 4500 simulation runs per attack scenario.

The equation cannot be solved analytically, so we apply the numerical approximate Newton-Raphson method [8] which uses the derivative of the function (i.e., $f'(l)$) and looks iteratively for a solution until an acceptable one is found (Eq. 4).

$$l_{n+1} = l_n - \frac{f(l_n)}{f'(l_n)} \tag{4}$$

IV. Simulation

Estimating the policy expressed in Eq. 1 involves running simulations to approximate the estimated value of the utility function F.

We use the FLAMES simulation tool (version 12.0) to create those simulations we need to learn the UAS policy. FLAMES provides flexibility to tailor a scenario and those models that we need for our simulations and learning. Some basic useful simulations models are provided with FLAMES, that can be adapted by custom C/C++ coding. This flexibility allowed us to create our own models for, e.g., the autonomous behavior and radar, and other aspects of our scenario such as adversarial cruise missiles and communication jamming.

The purpose of this work has been on the development of autonomous behavior and the simulation modeling has focused on making reasonable rather than realistic models. Some selected simulation values follow here:

- The aircraft was modeled as a MALE (medium altitude long endurance) platform having a top speed of 100 m/s, operating at an altitude of 8 km.
- The jammer breaks the radio communication, but the radar is unaffected and can detect targets of interest up to 100 km distance.
- The cruise missiles have a radar cross section that is 0.5 m^2 and a speed of 240 m/s. The missiles fly in five groups of three missiles each (hence a maximum of 15 can be detected by the UAS).
- Three different attack scenarios are modeled.

V. Results

As stated in previous sections, for each attack scenario we ran 50 simulations for each of the 90 possible action options, hence 4500 simulations for each scenario. The number of detected cruise missiles during each simulation was logged (the maximum number of detectable missiles were 15 in each simulation). In some simulations, results could not be achieved, because either no possible path existed (within the given time limit) or possible paths had to go outside the jammed area. The raw results of the simulations for each attack scenario is shown in Fig. 3, 4, and 5. In this paper, we leave it open to the responsible operator to decide what would consistute an appropriate utility (and, hence, policy). It could be, e.g., to detect as many missiles as possible during the time spent in the jammed area or, alternatively, to report the first detected missile as soon as possible.

The results in Fig. 3, 4, and 5 are presented in the same way. In the front, along the x-axis (of the x-y-plane) the action options (e.g., direction P_1 and time 31 mins) are enumerated. Along the x-axis all options which yielded some detection are included, others may be left out. Along the y-axis (moving away from the reader) are discrete slots for the number of missiles detected stacks from 0 to 15. The stacks in each position (stretching out along the z-axis) show the ratio of simulations for the option in question that resulted in that exact number of detections. The darker stacks depict simulations with few detections and the light ones more detections. Note that some options have no stacks at all due to that they did not result in any flight paths.

For scenario 1 (Fig. 3), $(P_3, 33$ mins$)$ would be a policy that optimizes Eq. 1 if the utility is expressed in only the number of detected missiles. That option, however, requires a lot of travel time before the results can be reported. Instead, if reporting time is of essence, the option $(P_2, 25$ mins$)$ is a better choice with many observations in shorter time and with few failed attempts to plan a path.

The result in Fig. 3 can be compared to using the non-adaptive default behavior of simply flying straight back to the home base which results in (our simulations) all missiles being detected in 24% of the simulations while none is detected in 58% of the cases which is worse than the presented cases in Fig. 3.

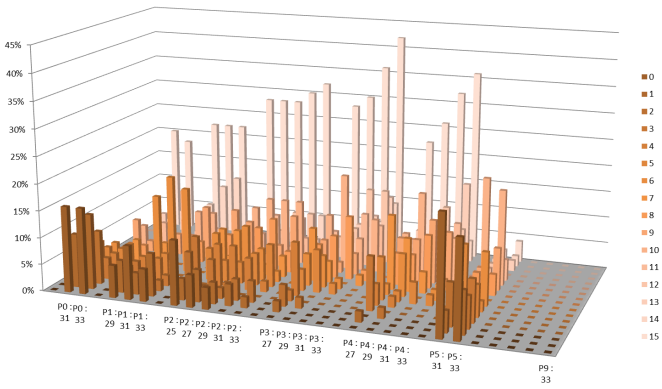

Fig. 3. The simulation result for Attack scenario 1.

The results for scenario 2 (Fig. 4) are less spread out for each option, i.e., either a lot of missiles are detected or none. That directions P_0 and P_1 yielded rich paths has to do with that those paths are close to the intended missile target. This result suggests that expected time to impact could be estimated and be part of the utility. Scenario 3 (Fig. 5) yielded very few successful paths. It could be that our B-curve paths are insufficient for this scenario and that other types of path are necessary.

VI. Conclusions

Our suggested approach consists of applying a machine learning approach to find options for path planning parameters. Although our scenario is military surveillance, there is no reason that it shouldn't be useful to jamming in commercial or other civilian applications. Our approach of simulation-based policy estimation allows a subject matter

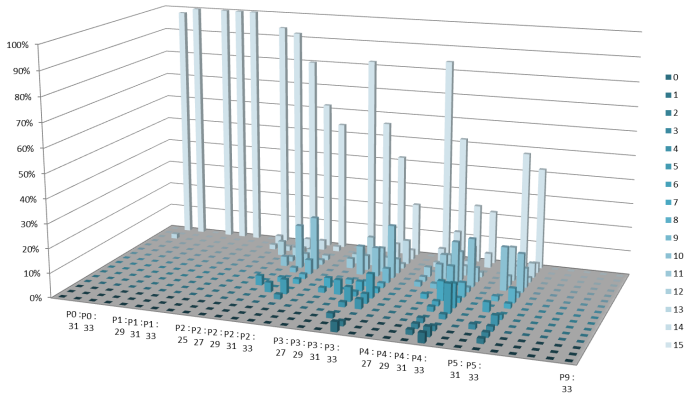

Fig. 4. The simulation result for Attack scenario 2.

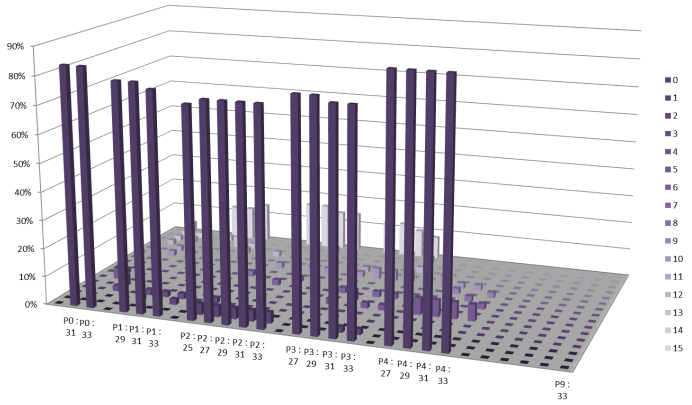

Fig. 5. The simulation result for Attack scenario 3.

expert to design the relevant scenarios to study as well as the mission objective function.

A large amount of simulations are used to form a decision basis where the subject matter expert can select a trade off between the different goal parameters. In our case, such goal parameters could involve to 1) assure to report at least one cruise missile (if any exist) in as short time as possible; 2) report as many cruise missiles as possible; or 3) to maximize the number of detections for a given flight time.

APPENDIX

Alg. 1 describes how to calculate the end point, P_2, of the quadratic Bezier curve. A part of that algorithm is to find the l-parameter of the line that constrains the possible values of P_2. Finding l is dealt with separately in Alg. 2.

Algorithm 1: Calculate P_2 of the Bezier curve
Input: P_0 Bezier start position. P_1 Bezier control parameter. P_d is the position of the radio jammer. P_b position of the homebase. $time$ the allowed time.
Output: A 2-D point P_2

CALCULATEP2$(P_0, P_1, P_d, P_b, time)$

 # find start position to look for P_2

(1) $r_0 \leftarrow \dfrac{P_b + P_d}{2}$

 # define rotation matrix

(2) $rot_{-\pi/2} \leftarrow \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$

 # vector for P_2 solutions

(3) $v \leftarrow \dfrac{P_d - P_b}{||P_d - P_b||} \cdot rot_{-\pi/2}$

 # travel time to curve length transformation

(4) $len \leftarrow time \cdot avgspeed$

 # Call to Alg. 2

(5) $l \leftarrow$ FINDL(r_0, v, P_0, P_1, len)

(6) **return** $r_0 + l \cdot v$

Algorithm 2: Find a parameter l (defining P_2) that results in a Bezier curve of about length len that is constrained by P_0, P_1, and $P_2(l) = r_0 + lv$.
Input: P_0, P_1 are two quadratic Bezier curve parameters, as 2-D points. r_0 and v are parameters on the line that defines possible P_2, i.e., $P_2 = r_0 + lv$.
Output: A real value l. len is the desired length of the Bezier curve.

FINDL(r_0, v, P_0, P_1, len)

 # perform Newton-Raphson to find l

 # intialize l

(1) $l \leftarrow 0$

 # Maximum number of iterations

(2) $maxiter \leftarrow 20$

 # Iteration counter

(3) $iter \leftarrow 0$

 # How large error in meters that we can accept

(4) $tolerance \leftarrow 5$

 # how many parts the Bezier should be divided into

(5) $k \leftarrow 10$

 # Algorithm does not check if a solution exists!

(6) **while** $iter < maxiter$ **and** $|L(l; B, k) - len| > tolerance$

(7) $nom \leftarrow L(l; B, k) - len$

(8) $denom \leftarrow \frac{d}{dl}(L(l; B, k))$

(9) **if** $denom = 0$

 # deal with bad start value

(10) $l \leftarrow l + 0.01$

(11) **continue**

(12) $q \leftarrow nom/denom$

(13) $l \leftarrow l - q$

(14) $iter \leftarrow iter + 1$

(15) **return** l

Finally, the required derivative derivative of the path length $f'(l) = \frac{d}{dl}L(l;B,k)$ which appears in line (8) of Alg. 2 is:

$$\begin{aligned}\frac{d}{dl}L(l;B,k) &= \left\{x_i(l) \triangleq \alpha_x^i + \beta_x^i l, y_i(l) \triangleq \alpha_y^i + \beta_y^i l\right\} \\ &= \frac{1}{k^2}\sum_{i=0}^{k-1}\left(x_i(l)^2 + y_i(l)^2\right)^{-1/2} \cdot \ldots \\ &\quad \ldots \cdot (x_i'(l)x_i(l) + y_i'(l)y_i(l))\end{aligned} \tag{5}$$

$\alpha_\bullet^i$ and $\beta_\bullet^i$ are defined in Section III.

ACKNOWLEDGEMENT

This work was funded by the Swedish Armed Forces. The authors wish to thank our coworkers Magnus Jändel for contributing to this work, Sara Linder for her consultation, and Farzad Kamrani and Tomas Mårtensson and the anonymous reviewers for their helpful comments on the manuscript.

REFERENCES

[1] J. Tisdale, Z. Kim and J.K. Hendrick, Autonomous Path Planning and Estimation using UAVs, *IEEE Robotics and automation magazine*, Vol. 16, Issue 2, June, 2009.

[2] Y.M. Chen, and W-Y Wu, Cooperative Electronic Attack for Groups of Unmanned Air Vehicles based on Multi-agent Simulation and Evaluation , *IJCSI International Journal of Computer Science Issues*, Vol. 9, Issue 2, March, 2009.

[3] R.S. Sutton and A.G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, ISBN 0-262-19398-1, 1998.

[4] A. Tsourdos, B. White, and M. Shanmugavel, *Cooperative path planning of unmanned aerial vehicles*, Wiley, 2011.

[5] Webpage: Stochastic approximation, https://en.wikipedia.org/wiki/Stochastic_approximation, accessed 2017-07-16.

[6] Webpage: https://en.wikipedia.org/wiki/B%C3%A9zier_curve, accessed 2017-07-17.

[7] Webpage: http://www.malczak.linuxpl.com/blog/quadratic-bezier-curve-length/, accessed 2017-07-18.

[8] Webpage: https://en.wikipedia.org/wiki/Newton's_method, accessed 2017-07-18.