A Genetic Programming Formulation to Evade Proportional Navigation

Daniel Howard
Biocomputing and Developmental Systems Group,
Computer Science and Information Systems
University of Limerick, Ireland

ABSTRACT

Genetic Programming evolves a manoeuvre plan for a single evader with constant velocity and a single pursuer employing proportional navigation with lag in a planar region. During evolution GP was required to discover the most effective evasion strategy which generalized across multiple training scenarios, i.e. different initial pursuer positions. GP was trained to evade interception for each of the multiple scenarios by controlling the evader's turn rate throughout pursuit.

1. PROBLEM DOMAIN

Game Theory [1] finds application in Economics and in many other fields. The pursuer evader game is a zero-sum game relevant to our observations in nearly all branches of Biology (predator and prey). It also has many interesting variants which have been studied analytically[1].

Machine Learning algorithms can reverse engineer a plan of evasion or a plan of pursuit (or both alternatively). Examples abound in the literature, particularly in Evolutionary Robotics, which has standardized its experimentation by introducing the Khepera robot [2]. Two artificial intelligence techniques are at the cutting edge of research for the investigation of pursuer-evader strategies: (a) genetic algorithms and (b) Q learning. A recent proposal to combine these for advantage is called 'QGP' [3] [4].

This paper considers discovering the evasion strategy of an aircraft from a missile. The missile pursues it using proportional navigation (PN) with a single lag. PN is a method of homing navigation in which the missile turn rate is directly proportional to the turn rate in space of the line of sight to the target. PN is a popular and effective algorithm of missile guidance [5].

The assumption that is made in this paper that the aircraft does not know the algorithm of pursuit. Machine Learning discovers a program of manoeuvres that defeats the missile (stops it from destroying the aircraft). The search for the evasion algorithm is based on a number of simulated encounters, each with a different starting position (and other differences). The machine learning method discovers the aircraft manoeuvres. The solution is generic (it generalizes) for the encounters. It is submitted that if the supervised learning scheme can learn a generic way to defeat the missile in PN, then it has a good chance of doing so for more complex simulation.

The proof of concept problem that is presented here is a 2D encounter but the real encounter is in 3D space. A complex and realistic simulation must consider other aspects of reality: (a) maximum thrust speed, weight, and acceleration characteristics of the aircraft; (b) timely deployment of counter-measures: flares, chaff, infrared jammer; (c) pursuits involving more than one missile, e.g. Starstreak [6] where three darts hold formation to blanket the target; (d) the relationship between the ability of a missile to generate enough lift to maneuver and its speed, wing size, and altitude; (e) atmospheric drag, aerodynamics, payload, and propellant weight; (f) parasitic effects such as missile radome; saturation; (g) noise corruption of the seeker measurement of the line of sight rate signal - e.g. Kalman filter models; (h) uncertainty about the make of missile, and the missile's guidance systems dynamics: order and time constants for seeker, noise filter and flight control system; (i) combat uncertainty, e.g. certainty of the time of fire of the missile(s) but uncertainty concerning the location of fire of the missile(s). The same technique and software might be used to satisfy quite different requirements, providing a solution over a sliding scale of uncertainty [17]. Moreover, an evasion policy is required that can ensure high probability of survival in two extreme cases: (j) against many different types of missiles or (k) against a specific type of missile. In case (j) it would learn from a great number of simulated encounters involving different types of missiles. This would result in a successful policy of evasion that would not need to know the type of pursuing missile. In case (k), it would use a number of simulated encounters for a specific type of missile, and the resulting evasion policy would only apply for pursuit by that specific missile.

The influence of time constants on augmented proportional navigation (APN) [5] could also be investigated with such a tool because it is a general method for obtaining near optimal manoeuvres and near optimal guidance laws for both strategic and tactical missile-target encounters. Countries developing Unmanned Air Vehicles (UAVs), drones and other unmanned aircraft, might use a similar tool to discover

[1]For example, the "suicidal chauffeur" problem was analyzed mathematically in a 1967 Stanford PhD thesis.

43rd ACM Southeast Conference, March 18-20, 2005, Kennesaw, GA,

requirements for autonomous systems and unmanned platforms.

2. GENETIC PROGRAMMING

Genetic Programming (GP) [7]-[9] is a machine learning method belonging to the family of Evolutionary Computation methods. GP maintains a population of candidate solutions (computer programs) in memory. Each population member is initially assembled at random. GP has a discrete time-stepping concept, and applies changes to members of the population at each time step. Thus, the population of candidate solutions changes as the computation proceeds.

GP uses the concept of a fitness (by analogy to Darwinian fitness) to determine whether population members are better or worse candidate solutions. At each time step, it selects some of the more successful computer programs in the current population based on this criterion. GP applies operators (by analogy to mutation and crossover events in population genetics) to modify the selected programs to make new programs, perhaps by combining pieces of two of the selected solutions. The resulting "progeny" is not necessarily an improved solution (monotonic convergence is not guaranteed) but improvements do invariantly occur (by analogy to Natural Evolution).

GP needs a representation for the computer program, by default this is a *variable length* tree representation with leaf nodes (the terminals) and internal nodes (the functions). If so desired, GP can use one representation for the evolution of solutions and another for the implementation of solutions. GP is able to exploit all of the functionality that is available to computer programmers: e.g. recursion, iteration, memory, variables, parameterized functions, etc. [8]. Solutions lend themselves to 'white box' analysis (can be interpreted to discover the component strategies).

GP is not a greedy but a patient search. Usually the simplest model that fits the data is discovered first. Solutions grow in size automatically, usually to accommodate more complex solutions but also pathological solutions invariably occur (by analogy to "junk DNA" and the C-value enigma [10]) as this is a property of the variable length representation [11]. GP is scalable to large problem sets, and able to handle many variables to discover their information value. It is a superset of other AI methods. The choice of functions, terminals, allowable Computer Science constructs and combination rules allows GP to represent a wide variety of methods: neural networks, support vector machines, etc. This can be clearly understood when considering the version of GP that evolves grammars (by reference to the Chomsky hierarchy of grammars and Turing machines) and known as *Grammatical Evolution (GE)* [12]. In addition, GE implements silent mutation and handles different rules of combination of variables with ease (typing is not a problem because it is part of the grammar). A popular GP practice is to separate what is being evolved from its expression (by analogy to the genotype and phenotype distinction that is part of the basic cellular technology of life). GP is run on a number of computers in parallel independent runs with different initial random populations. This enables the practitioner to select an outstanding solution from one run and perhaps a slightly less outstanding solution that is more amenable to simplification and interpretation from another and selected to shed light on solution principles.

The population based approach confers GP with many advantages. For example, there exist many different ways of maintaining diversity in the search [13]-[15], and this is particularly suitable to multi-objective optimization, or to the concurrent search for multiple possible explanations to a given under-determined problem. Moreover, Evolutionary Computation and GP offer another advantage: co-evolution [16]. In principle, GP can co-evolve the technology race between evaders and pursuers.

This paper applies the steady state version of Genetic Programming with tournament selection [7] to evolve the evasion of an aircraft from a missile that pursues it. An objective of this work is to present computations that can be easily validated and rules that can be easily interpreted unlike those in [17]. GP works by incremental improvement or *evolution* in the evasion capability of a set or *population* of evasion policies. Central to this process is to assign a Darwinian fitness to each candidate evasion policy, which can only be obtained from the solution to a mathematical model of the encounter - real encounters are not an option because thousands of policies must be evaluated. An option would be to use the closed form solutions to the 2D linearized proportional navigation equations in [18] to predict the resulting miss distance when the aircraft either undertakes a single manoeuvre from a given range, or when the initial missile velocity vector points at some heading error. Though extremely useful to demonstrate guidance principles these analytical solutions contain too many assumptions to provide the Darwinian fitness measure. Instead, a numerical solution of the non-linear differential equations of pursuit and evasion is the preferred option to judge the performance of the evasion policy. The particular fitness measure used in this paper is novel for this problem.

The remainder of the paper presents the method and results for the 2D encounter proof-of-concept problem.

3. SYSTEM OF EQUATIONS

In this simplified 2D encounter we demonstrate that Genetic Programming can evolve very powerful sets of aircraft manoeuvres that allow the aircraft to take maximum advantage of the single lag dynamics of an idealized missile that pursues it with proportional navigation, i.e. to maximize the miss distance and hence to evade the missile starting from various distances. The initial conditions for this problem assume zero heading error, i.e. miss distance will depend only on evasive manoeuvres. The closed form solution of the linearized version of the single manoeuvre problem is given in [18] and will be used in section 6.1 to judge the performance of our GP solution.

The geometry of proportional navigation is as in Figure 1 and Table 1 summarizes a 2D non-linear differential model of proportional navigation with single lag [18]. Expressions involving the proportional navigation time integration have been stressed in bold type. The subscripts 1 and 2 stand for components along the x and y coordinates respectively. The locations of target and missile are given by R_T and R_M respectively. The same applies to the speeds V_T and V_M or magnitude of the velocity vectors $\vec{V_T}$ and $\vec{V_M}$ respectively. The missile target closing velocity V_c is defined as the negative rate of change of the distance R_{TM} from missile to target.

Proportional navigation issues acceleration commands that are perpendicular to the instantaneous missile-target line-of-sight angle in radians λ, and proportional to the line-of-sight

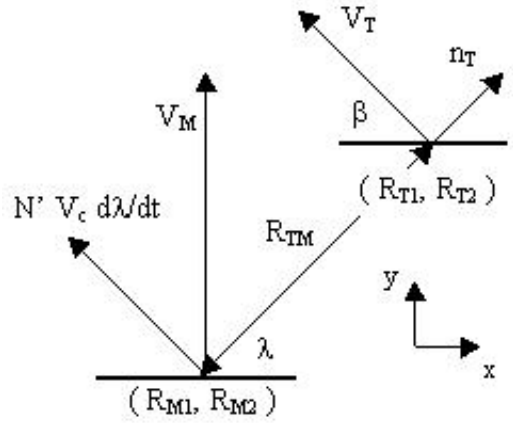

Figure 1: Proportional navigation pursuer-evader geometry: missile and target.

Table 1: Proportional navigation with single lag.

GEOMETRY:	
$R_{TM1} = R_{T1} - R_{M1}$	$V_{TM1} = V_{T1} - V_{M1}$
$R_{TM2} = R_{T2} - R_{M2}$	$V_{TM2} = V_{T2} - V_{M2}$
$R_{TM}^2 = R_{TM1}^2 + R_{TM2}^2$	
$V_c = -(R_{TM1}V_{TM1} + R_{TM2}V_{TM2})/R_{TM}$	
$\dot{\vec{x}} = \boldsymbol{f(\vec{x}, t)}$	
$\vec{x} = [\boldsymbol{\beta, R_{T1}, R_{T2}, R_{M1}, R_{M2}, V_{M1}, V_{M2}, n_L}]^T$	
TARGET:	
$\dot{\boldsymbol{R_{T1}}} = \boldsymbol{V_{T1}}$	$\dot{\boldsymbol{R_{T2}}} = \boldsymbol{V_{T2}}$
$V_{T1} = -V_T cos(\beta)$	$V_{T2} = V_T sin(\beta)$
V_T constant	n_T constant, e.g. $-3g$
$\beta = tan^{-1}(V_{T2}/-V_{T1})$	$\dot{\boldsymbol{\beta}} = \boldsymbol{n_T/V_T}$
MISSILE:	
$\dot{\boldsymbol{R_{M1}}} = \boldsymbol{V_{M1}}$	$\dot{\boldsymbol{V_{M1}}} = \boldsymbol{a_{M1}}$
$\dot{\boldsymbol{R_{M2}}} = \boldsymbol{V_{M2}}$	$\dot{\boldsymbol{V_{M2}}} = \boldsymbol{a_{M2}}$
$a_{M1} = -n_L sin(\lambda)$	$a_{M2} = n_L cos(\lambda)$
$\boldsymbol{n_L = N' V_c \dot{\lambda} - \tau \dot{n_L}}$	$\lambda = tan^{-1}(R_{TM2}/R_{TM1})$
$\dot{\lambda} = (R_{TM1}V_{TM2} - R_{TM2}V_{TM1})/R_{TM}^2$	

time rate of change $\dot{\lambda}$ and to V_c the missile target closing velocity. The instantaneous flight-path angle of the target is given by β and the angular velocity of the target is given by $\dot{\beta}$.

Missile acceleration commands are achieved by exoatmospheric interceptors via thrust vector control, lateral divert engines, or squibes, and by endoatmospheric missiles with moving fins and control surfaces to achieve lift. The missile's flight control system must cause the missile to maneuver in such a way that the achieved acceleration n_L matches the desired acceleration $N' V_c \dot{\lambda}$. In Table 1 this has been modelled by single lag control dynamics with τ as the flight control system time constant. The Effective Navigation Ratio N' is a unitless designer gain usually in the range 3 to 5.

The missile lag dynamics gives the aircraft the opportunity to maneuver to maximize the miss distance and to evade the missile. The target has the freedom to attempt to maneuver evasively with acceleration n_T perpendicular to the target velocity vector V_T.

4. NUMERICAL INTEGRATION

This system can be integrated numerically with a second order Runge-Kutta algorithm Table 2 and Table 3. The principle of this method being to calculate a representative slope of the function in the interval by averaging the slopes at various extrapolated points. The time step δt is made smaller near the end of the flight to accurately capture the magnitude of the miss distance.

5. GP IMPLEMENTATION

GP manipulated a population of candidate solutions over successive generations in order to evolve near-optimal solutions. Each initial candidate solution was represented as a random tree-like structure of *function nodes* and *terminal nodes*. Each tree processed the current pursuer-evader state via terminal inputs and output a real numerical value which controlled the evader's turn rate, enabling each tree to be awarded a *fitness measure* by governing evader manoeuvrability throughout multiple pursuit cases.

GP advanced to a second generation of candidate solutions by selecting a member from the population and replacing it with a new member which was regenerated from the existing solutions. The probabilistic odds of being selected for replacement and for regeneration were prescribed by the fitness measure, such that members with relatively low fitness were replaced whereas members with relatively high fitness were regenerated. In this way, advancement through successive generations constitutes the Darwinian principle of *survival of the fittest*. This method enables GP to search the complex space of function and terminal nodes in order to produce near-optimal solutions which generalize across the training data.

GP requires the function and terminal nodes to be specified for a given problem and a number of parameters must be set. Our GP implementation is summarized in Table 4 where x_{PE}, y_{PE}, d_{PE} are the dimensionless x displacement, y displacement and distance from pursuer to evader, all expressed as a percentage of the maximum distance, i.e. 40000ft. All terminal and function nodes output a value between -1 and 1. Regarding the genetic operations in Table 4 the mutation with 5% probability was only carried out on constant terminals, i.e. not on terminal variables x_{PE}, y_{PE}, and d_{PE}.

Table 2: Elements of the second order Runge-Kutta numerical integration of single-lag proportional navigation.

First Point:
$R^O_{T1} = R_{T1} \quad R^O_{M1} = R_{M1} \quad V^O_{M1} = V_{M1} \quad \beta^O = \beta$
$R^O_{T2} = R_{T2} \quad R^O_{M2} = R_{M2} \quad V^O_{M2} = V_{M2} \quad n^O_L = n_L$
Second Point:
$R_{T1} = R_{T1} + \delta t V_{T1} \quad R_{M1} = R_{M1} + \delta t V_{M1}$
$V_{M1} = V_{M1} + \delta t a_{M1} \quad \beta = \beta + \delta t \dot{\beta}$
$R_{T2} = R_{T2} + \delta t V_{T2} \quad R_{M2} = R_{M2} + \delta t V_{M2}$
$V_{M2} = V_{M2} + \delta t a_{M2} \quad n_L = n_L + \delta t(N^{'} V_c \dot{\lambda} - n_L)/\tau$
Third Point:
$R_{T1} = 0.5(R^O_{T1} + R_{T1} + \delta t V_{T1})$
$R_{M1} = 0.5(R^O_{M1} + R_{M1} + \delta t V_{M1})$
$V_{M1} = 0.5(V^O_{M1} + V_{M1} + \delta t a_{M1})$
$R_{T2} = 0.5(R^O_{T2} + R_{T2} + \delta t V_{T2})$
$R_{M2} = 0.5(R^O_{M2} + R_{M2} + \delta t V_{M2})$
$V_{M2} = 0.5(V^O_{M2} + V_{M2} + \delta t a_{M2})$
$\beta = 0.5(\beta^O + \beta + \delta t \dot{\beta})$
$n_L = 0.5(n^O_L + n_L + \delta t(N^{'} V_c \dot{\lambda} - n_L)/\tau)$
Discretized Equations:
$R_{TM1} = R_{T1} - R_{M1}$
$R_{TM2} = R_{T2} - R_{M2}$
$R_{TM} = \sqrt{R^2_{TM1} + R^2_{TM2}}$
$V_{TM1} = V_{T1} - V_{M1}$
$V_{TM2} = V_{T2} - V_{M2}$
$V_c = -(R_{TM1} V_{TM1} + R_{TM2} V_{TM2})/R_{TM}$
$\lambda = tan^{-1}(R_{TM2}/R_{TM1})$
$\dot{\lambda} = (R_{TM1} V_{TM2} - R_{TM2} V_{TM1})/R^2_{TM}$
$a_{M1} = -n_L sin(\lambda)$
$a_{M2} = n_L cos(\lambda)$
$V_{T1} = -V_T cos(\beta)$
$V_{T2} = V_T sin(\beta)$
$\dot{\beta} = n_T/V_T$

Table 3: Second order Runge-Kutta numerical integration - pseudo-code for Table 2.

```
Initialise β, n_L, V_T1, V_T2, R_TM1, R_TM2, R_TM, λ,
           V_M1 and V_M2;
BEGIN LOOP;
 if V_c < 0 then STOP;
 time step change?: IF R_TM < 1000
    THEN δt = 0.0002 ELSE δt = 0.01;
 calculate:  First Point;
 update:  Discretized Equations;
 calculate:  Second Point;
 update:  Discretized Equations;
 calculate:  Third Point;
 update:  Discretized Equations;
END LOOP.
```

Table 4: GP parameters.

Parameter	Description
Terminals	x_{PE}, y_{PE}, d_{PE}; +ve reals (0.01 to 1.0 in 0.01 steps); and -ve reals (-1.0 to -0.01 in 0.01 steps).
Functions	min, max; if $x_{PE} < 0$ then arg_1 else arg_2; if $y_{PE} < 0$ then arg_1 else arg_2; if $d_{PE} < abs(arg_1)$ then arg_2 else arg_3;
Population	5000
Mate radius	5000
Max nodes	1000
Kill tourn	size 2: in steady-state GP
Breed tourn	size 4: in steady-state GP.
Regeneration	95% cross-over; 5% mutation.
Max generations	50

At each time interval or *epoch* each GP tree responded to the pursuer to evader displacements x_{PE}, y_{PE} and distance d_{PE} by output of a percentage turn rate, such that -1 represented a maximal left turn and 1 represented a maximal right turn. At time zero the fitness evaluation for a given tree gave the initial turn rate for the evader. The turn rate was recomputed at the end of each epoch. The epoch lasted for 200 milliseconds a value much greater than the Runge-Kutta time step δt. The raw fitness f_i for the i^{th} training scenario was calculated as:

$$f_i = t_s + \min(d/d_{kill}, d_{max}) \tag{1}$$

and was designed to give a boost to achieving evasion. Here t_s was the evader's survival time which was set to $t_{max} = 20$s upon evasion, d was the achieved minimum distance between the pursuer and evader in feet, d_{kill} was the 'kill distance' or what discriminates an evasion from a hit and $d_{max} = 3$ is a cap for the contribution of the distance component of the reward. Repeating this process for each of the N training scenarios enabled the overall fitness to be calculated as:

$$\text{fitness} = \frac{\sum_{i=1}^{N} f_i}{N(t_{max} + d_{max})} \tag{2}$$

thus giving low fitness measures tending towards 0 and high fitness measures tending towards 1. These functions cause GP to maximize the evader's survival time and the pursuer to evader distance.

6. EVOLVED MANOEUVRES

GP was trained using various initial pursuer positions as given in Figure 2 and Table 2. The table shows that the pursuer was always launched from the same x direction but from both y directions. These positions represent 'worst cases' because the initial evader velocity vector had a negative x component and zero y component. The last scenario represents a 'head on' pursuit. The table also shows that GP was required to generalize across a number of initial pursuer to evader distances. The initial velocity of the pursuer was 3000ft/s. The magnitude of the evader's velocity was fixed at 1000ft/s.

6.1 Results

Manoeuvre plans were evolved for a number of pursuit configurations. The evader's maximum turn rate was fixed

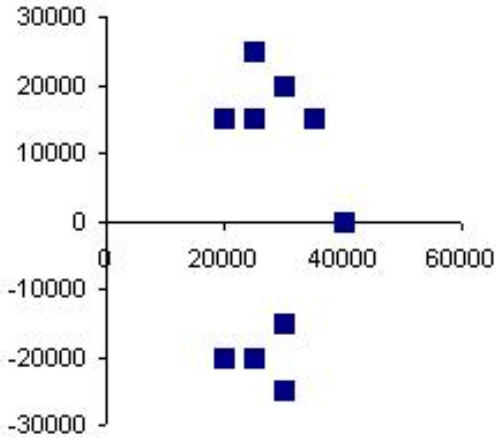

Figure 2: Training scenarios: pursuer to evader distances in feet.

Table 5: Training scenarios: pursuer to evader distances in feet.

scenario	x	y	overall
A	20000	15000	25000
B	20000	-20000	28284
C	25000	15000	29154
D	25000	-20000	32015
E	30000	-15000	33541
F	25000	25000	35355
G	30000	20000	36055
H	35000	15000	38078
I	30000	-25000	39051
J	40000	0	40000

at 4g but the pursuer's effective navigation ratio, N', was set at 3 and 4. The pursuer's kill radius, d_{kill}, was varied between 30ft and 50ft. The following results were stored for each configuration and training scenario: the pursuit time t_p in seconds (i.e. the time from pursuer launch until either intercept or evasion) and the evasion distance d_e (i.e. the minimum distance beyond the kill radius d_{kill} of the evader). Interception was deemed to occur when the pursuer to evader distance was less than d_{kill}, i.e. $d_e = 0$.

At least five evolution runs were carried out for each pursuit configuration, differing only in randomizer seed. The results were very similar for each seed showing that the method, although based upon random processes, produces consistent manoeuvre plans. Table 6 presents an example of the best results for $N' = 3$, showing that successful evasion was achieved for all but the last scenario when $d_{kill} = 50$ft. Evasion distances were generally shorter for higher d_{kill}, as expected, and often for the last scenario which corresponded to a 'head on' pursuit. The reason for this is given in Section 6.2.

The evolved manoeuvre plans were less successful when N' was increased to 4, as shown in Table 6. Evolution runs using other randomizer seeds failed to evade for all scenarios even when d_{kill} was set to 30ft. No runs succeeded when d_{kill} was 50ft but GP did considerably increase the pursuit time, i.e. although interception could not be avoided the time to interception could be prolonged. This can be seen in the table by comparing the results for a given scenario at different d_{kill}. It is thought that these results reveal the limit of generalized evasion with fixed velocity in this demanding problem.

6.2 Interpreting manoeuvre plans

An advantage of GP over other AI approaches (e.g. artificial neural networks) is that evolved solutions can be interpreted to discover the constituent building blocks. The results presented in Section 6.1 corresponded to relatively simple GP representations of manoeuvre plans, consisting of less than 50 nodes. The results in Table 6 with $N' = 3$ for a d_{kill} of 40ft were produced by the following manoeuvre plan which was represented as a 9 node GP tree:

```
  if y_PE < 0 then
   if d_PE < abs(−0.22) then -0.94 else 0.98
else
   if d_PE < abs(−0.12) then 0.98 else -0.83
end if
```

This typical result shows that GP discovered that the pursuer to evader x displacement was redundant for the current problem. This GP tree (formula) is described in more detail in the following paragraphs.

Although candidate manoeuvre plans produced arbitrary percentage turn rates in initial generations, GP discovered in early generations that 'hard turns' was a useful evasive tactic. The most successful plans involved initially turning away from the pursuer in order to maximize pursuit time and then pulling a hard turn when the pursuer to evader distance decreased below some threshold (e.g. changing the output from 1 to -1 to change the evader's turn rate from maximum right to maximum left). This strategy is encoded in the above plan with different turn rates depending on the pursuer to evader y direction.

GP consistently discovered that the optimal time to pull a hard turn was when the distance was about 20% of the maximum distance. The above plan uses a threshold of 22% and 12% for negative and positive pursuer to evader y displacements respectively. This accounts for the greater evasion distances for scenarios B, D, E and I in Table 6 with $N' = 3$ for a d_{kill} of 40ft, as these scenarios corresponded to negative y displacements (shown in Figure 2). Continuing the evolution run beyond 10 generations further optimized the time to perform the hard turn manoeuvre for both negative and positive y displacements.

The output of the above manoeuvre plan (i.e. fractional turn rate changes) during the course of pursuit is displayed in Figure 3 for scenarios A, i.e. the minimum initial pursuer to evader distance.

Evasion distances were generally shorter for scenarios J, the maximal pursuer to evader distance corresponding to a 'head on' pursuit, because the evader performs more hard turn manoeuvres as the pursuer to evader y direction changes during the pursuit.

7. CONCLUSIONS

The preliminary results reported here demonstrate the promise of a GP approach to planning tactical manoeuvres. GP successfully evolved optimal manoeuvre plans to evade pursuer interception under challenging pursuit conditions. Furthermore, these plans generalized across multiple pursuit scenarios.

Interpretation of the GP solutions revealed that GP had discovered important general principles of evasive tactics,

Table 6: Pursuit times and evasion distances for various N' and d_{kill} after 10 generations.

N'=3							N'=4						
scenario	d_{kill} = 30ft		d_{kill} = 40ft		d_{kill} = 50ft		scenario	d_{kill} = 30ft		d_{kill} = 40ft		d_{kill} = 50ft	
	t_p	d_e	t_p	d_e	t_p	d_e		t_p	d_e	t_p	d_e	t_p	d_e
A	6.474	38.650	6.475	2.659	6.469	18.988	A	6.625	5.464	6.670	0.621	7.448	0.000
B	7.428	40.607	7.438	26.418	7.437	19.619	B	7.422	8.066	7.757	0.000	9.114	0.000
C	7.468	42.666	7.469	5.327	7.471	22.327	C	7.519	10.478	7.541	2.388	8.604	0.000
D	8.331	42.766	8.295	30.732	8.294	22.730	D	8.284	6.135	8.437	1.054	10.140	0.000
E	8.586	41.061	8.578	28.742	8.575	22.305	E	8.576	3.612	8.575	6.426	9.916	0.000
F	9.260	43.300	9.273	6.664	9.265	23.786	F	9.342	10.874	9.352	0.911	12.103	0.000
G	9.300	42.264	9.288	6.894	9.311	21.564	G	9.271	14.274	9.263	2.171	11.369	0.000
H	9.802	35.884	9.759	6.519	9.837	13.147	H	9.642	15.892	9.647	0.979	11.369	0.000
I	10.458	38.510	10.161	29.547	10.385	17.807	I	10.141	2.105	10.149	7.888	13.237	0.000
J	10.034	3.268	10.011	0.303	10.041	0.000	J	10.025	12.130	10.028	6.821	10.855	0.000

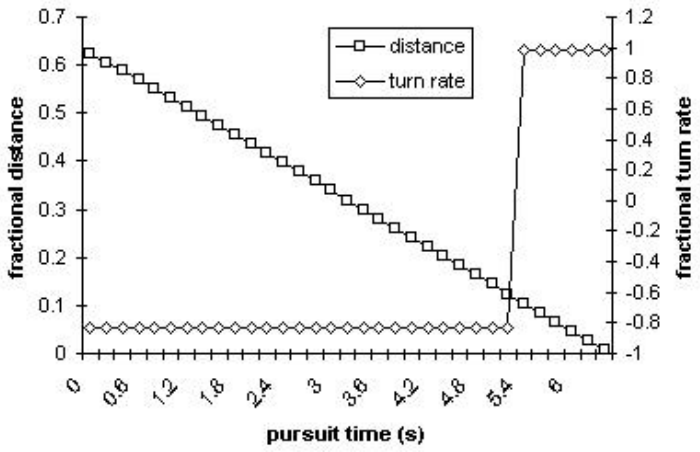

Figure 3: Scenario A: fractional evader turn rate and fractional pursuer to evader distance during the course of pursuit.

as opposed to arriving at *ad hoc* manoeuvres for particular training cases. This demonstrates the GP utility for discovering underlying principles, in addition to evolving specific solutions.

Further work is needed to improve the problem representation (e.g. improve evader and pursuer dynamics), to consider other evasion tactics (e.g. setting evader thrust) and to investigate alternative GP designs (e.g. terminal and function node definitions and fitness formulation). The GP approach can also be applied to evolving evasive manoeuvres in three-dimensional space.

8. REFERENCES

[1] Von Neumann, J., Morgensten, O., *The Theory of Games and Economic Behavior* (Princeton, NJ: Princeton University Press) (1947).

[2] `www.k-team.com/robots/khepera/`

[3] Iba, H., *Multi-Agent Reinforcement Learning with Genetic Programming*, GECCO conference, pp. 167-172 (1998).

[4] Kamio, S., Mitsuhashi, H., Iba, H., *Integration of Genetic Programming and Reinforcement Learning for Real Robots*, Proc. GECCO conference, (2003).

[5] Shneydor, N. A., *Missile Guidance and Pursuit*, Horwood Series in Engineering Science, Horwood Publishing Limited, Chichester (1998). ISBN 1-898563-43-8.

[6] `www.army-technology.com/projects/starstreak/`

[7] Koza, J. R., *Genetic Programming*, MIT Press (1992).

[8] Koza, J. R., *Genetic Programming II: Automatic Discovery of Reusable Programs*, MIT Press (1994).

[9] Koza, J. R., Bennett III, F.H., Stiffelman, O., *Genetic Programming as a Darwinian Invention Machine*. Lecture Notes in Computer Science 1598, 93–108 (1999).

[10] Gregory, T. R., Macroevolution, hierarchy theory, and the C-value enigma, Paleobiology, 30(2):179-202 (2004).

[11] Rosca, J., *A Probabilistic Model of Size Drift*, Chapter 8 in Genetic Programming Theory and Practice, 119-136, Kluwer Academic Publishers (2003).

[12] O'Neill, M., Ryan, C., *Grammatical Evolution*, IEEE Trans. Evolutionary Computation 5(4):349-358 (2001).

[13] Hu, J., Goodman, E. D., Seo, K., *Continuous Hierarchical Fair Competition Model for Sustainable Innovation in Genetic Programming*, Chapter 6 in Genetic Programming Theory and Practice, Kluwer Academic Publishers (2003).

[14] Howard, D., *Modularization by Multi-Run Frequency Driven Subtree Encapsulation*, Chapter 10 in Genetic Programming Theory and Practice, 155-172, Kluwer Academic Publishers (2003).

[15] Ryan, C., Keijzer, M., Cattolico, M., *Favorable Biasing of Function Sets Using Run Transferable Libraries* Chapter 7 in Genetic Programming Theory and Practice II, Kluwer Academic Publishers (2004).

[16] Pollack, J. B., Blair, A. D., Land, M.,*Coevolution of a Backgammon Player*, Artificial Life V: Proc. of the Fifth Int. Workshop on the Synthesis and Simulation of Living Systems, The MIT Press, Cambridge, MA, Christopher G. Langton and Katsunori Shimohara (Editors), 92–98, (1997).

[17] Moore, F. W., *A methodology for Strategy Optimization Under Uncertainty*, Ph.D. thesis, Wright State University, (1997).

[18] Zarchan, P., *Tactical and Strategic Missile Guidance*, Progress in Astronautics and Aeronautics, vol 124, A. Richard Seebass, Editor, (1990). ISBN 0-930403-68-1.