2020 4th International Conference on Automation, Control and Robots

Linear Genetic Programming-Based Controller for Space Debris Retrieval

E. Gregson
Mechanical Engineering Dept.
Dalhousie University
Halifax, Canada
e-mail: ed565337@dal.ca

M. L. Seto, *Ph.D., P.Eng., SMIEEE*
Mechanical Engineering Dept.
Dalhousie University
Halifax, Canada
e-mail: mae.seto@dal.ca

***Abstract*—In this paper, we investigate the use of linear genetic programming to evolve a controller that can guide a debris removal chaser spacecraft to match the motion of an uncontrolled target debris object. The problem is treated in 2D, and the controller is required to apply forces and torques to the chaser such that it approaches the target and matches a "hand" point in the chaser-fixed frame to a "handle" point in the target-fixed frame. The training simulations are extensively parameterized, and as the population of controllers evolves, the population of training scenarios also changes through both coevolution and scheduled changes. This allows the controller population to be gradually taught the full task after starting with a simpler version. The resulting evolved controllers show promise but would benefit from a more sophisticated GP implementation than monolithic linear GP.**

Keywords-genetic programming; controller; autonomous system; chaser spacecraft

I. INTRODUCTION

We report on a method to deorbit arbitrarily-sized and shaped space debris to reduce the collision threat to orbiting operational space systems. Space debris can be derelict spacecraft, discarded rocket stages and fragments in Earth orbit. Moving at orbital velocities, debris poses a substantial risk. Impact with even very small debris is sufficient to destroy an operational space system. An impact with a piece of debris, or between two debris objects, can generate more space debris. In the worst case, a chain reaction, the Kessler Syndrome, occurs where the space debris population grows exponentially through self-collisions creating a very large debris population and rendering the orbit unusable [1].

A proposed solution is autonomous debris removal (ADR), where a robotic spacecraft is deployed to secure select debris and deorbit it. This slows the debris growth by preventing the eventual target breakup in orbit. Strategies have been proposed to secure and deorbit debris, including capture with robotic arms, nets, or harpoons. This paper reports on a robot arm capture development as it is a flexible and re-usable solution. In concept, the ADR spacecraft (hereafter, chaser) flies up to the debris and grabs it with its robot arm end effector, then applies forces and torques from its thrusters to detumble and deorbit the debris (hereafter, target).

The automated debris removal task encompasses both low-level control and higher-level guidance. Genetic programming / genetic programs (GP) has been applied to both tasks for several decades now. GP approaches used can be based on evolving tree-structured programs, neural nets [2], or sequential programs. A distinction can be made between genetic algorithm (GA) approaches, where a controller architecture is chosen beforehand and evolution is used to optimize its numerical parameters, versus a more undefined GP approach where the structure of the program itself is evolved.

Lewis *et al.* [3] use genetic programming to construct a neural net for control of a legged robot. They preset the neural net architecture and use evolution to tune the weights, using a two stage training process in which the first stage's fitness function prompts the evolution of a general movement, and the second stage's fitness function refines that movement. Nordin and Banzhaf [4] explore the use of linear GP in robot online learning; their robot has a population of GP controllers, all of which are given a chance to control the robot and be selected based on their outcomes, allowing the robot to learn obstacle avoidance and object following online. Kadlic *et al.* [5] looks at water turbine control using a form of Cartesian genetic programming (CGP) that evolves connections between functional blocks, whereas Khan *et al.* [6] examines the use of CGP to evolve weights and topology for neural network controllers, testing them on the double pendulum problem. Luo *et al.* [7] investigate benchmarking linear GP on prediction and control tasks, including chaotic time series prediction and pole balancing. Linear GP (used for this work) is further described by Brameier and Banzhaf [8]. Hein *et al.* describe a means to generate interpretable reinforcement learning policies by training a GP population on a pre-existing neural-net-based world model, applying this to the mountain car, pole balancing and industrial benchmark tasks [9].

In the aerospace domain, Oh and Barlow applied GP to the problem of guiding a UAV to find, approach and circle a radar source [10]. GA and GP have been frequently applied to the problem of attitude control for satellites and spacecraft. Dracopoulos studies evolving satellite attitude controllers with GP [11]. Silva *et al.* describe a more GA-based approach of evolving rocket attitude controllers by evolving the gains for linear quadratic controllers [12]. Recently, Marchetti *et al.* demonstrate a controller for a launching rocket that trains a population of genetic programs online to adapt to hardware failures or unexpected events [13].

For the specific problem of controlling a spacecraft to match the motion of an uncontrolled tumbling object,

978-1-7281-9207-9/20/$31.00 ©2020 IEEE

solutions have generally been based around less machine-learning based methods. Aghili describes an optimal control-based approach to docking to and securing space debris [14]. Li *et al.* present a solution to the same problem based on model predictive control [15].

A GP solution poses challenges in autonomous planning and control (the contribution here) of the ADR vehicle. The target will be tumbling with arbitrary (approximately) torque-free motion. For most targets, only certain sections are suitable as grapple points. For example, a rocket engine nozzle might be a good grapple point whereas the edge of a solar panel would not be.

The chaser must track a candidate grapple point on the tumbling target then after deliberation attempt to grasp that point. If the target tumbles relatively quickly with a fair amount of rotational inertia, the chaser may have to fly a path that matches (station-keeps) to the candidate grapple point, to decrease stress on the arm and/or give it time to extend the arm. The attitude of the chaser during such maneuvers must be controlled; the chaser side with the arm must face towards the target grapple point. The paper reports on a proposed controller methodology for a chaser that station-keeps to an arbitrarily sized and shaped transiting and tumbling target such that a chaser-based arm can grasp its grapple point. Afterwards, the thrusters would provide corrective forces and torques to detumble and deorbit the target. The posed question is how to design adaptive control laws for the chaser's thrusters, given the target motion, to allow capture. The controller would maintain the chaser-debris formation and autonomously correct in real-time for uncertainties in the debris or chaser dynamics. This is the problem of the spacecraft rendezvous and docking to a tumbling target.

When a problem is complex and its parameters only partially observable or unknown because it changes rapidly or is stochastic in nature, machine learning methods can be considered. An apparently high-dimensional and complex problem / data set has a lower-dimensional representation which distills out the key features of interest. These features could be learned through a machine learning approach. However, machine learning implementations with neural networks are computationally demanding on an embedded platform that carries all its mission energy on-board. Canonical linear genetic program (GP) solutions are a type of machine learning which produces programs that are more computationally light weight [4] but can still address the stochastic nature of some of the parameters. This motivated an interest to determine the extent of an adaptive control solution from GP. At a high level, the intent is to take the chaser and target state vectors, for particular chaser and target initial conditions and performance goals (scenarios), and apply a GP controller to determine the control forces and torques to guide the chaser to a rendezvous and grapple with the target.

The rest of this paper is as follows. Section II presents background on GP and how it was implemented. Section III covers the details of the task and simulator. Section IV describes the training process and how the simulated scenarios were used to teach the controllers. Section V is an overview of the results, Section VI discusses them, and Section VII discusses possible future work. Section VIII lists the contributions, and Section IX concludes with a few remarks.

II. Background

A. Genetic Programs

Generally, a linear GP takes a set of registers and an array of input values, and returns the register values altered, based on the input values. One line in a linear GP program is a list of numbers which identifies a chosen register, an operation to apply to that register value, and a second register value or input to be an additional operand to the operation. It outputs the altered value of the chosen register. A stack of lines (represented as a 2D array) forms a program. A canonical linear genetic program (GP) was implemented in MATLAB.

A population of programs (controllers) are represented as a 3D array of 2D program arrays. To improve the programs, a Breeder model (as described in [16] and similar to the Breeder GA in [17]) is employed, where (in every generation) each program is tested on its task and assigned a fitness based on its performance. The gap number of worst performing programs are discarded, and the remaining programs (called parents) are used as input to variation operators, which generate child programs with random modifications and combinations of the parents. These parents and children form the new population.

The Breeder model is elitist; the best performing program is never discarded, so the best fitness demonstrated by the program population only increases, if the test cases to which they are applied remain the same [16]. The variation operators used here to evolve the controllers were single-point crossover and mutation.

Single-point crossover takes two randomly selected parent programs, splits each at a randomly selected line number and swaps the lower blocks of lines to create two new child programs. It allows growth in program length. Mutation used here takes a randomly selected parent program and moves through the values in each line of the program, altering the values with probability P_m to create a child program. The values are altered by adding or subtracting 1 with equal probability, with a value-wrapping condition to keep the values within the allowed range. The mutation probability P_m was set to 0.1 during training. When applied to a program, it was scaled by $i/(i + \frac{L}{L_{MAX}}(L - i))$, where i is the current line number, L is the number of lines in the program and L_{MAX} is the maximum number of lines allowed in a program. This means that for short, early programs mutation is relatively uniform, but for long programs the mutation probability is reduced for earlier lines where changes could have a greater impact. About half of the child programs were generated by crossover, and half by mutation. Further details are omitted for brevity.

III. Simulator Implementation

A. Simulation Models to Assess Controllers

The objects modelled are the chaser and the target, simulated in 2D for simplicity and lower computational demand. Both are assumed to be in a zero-gravity environment. Each object has a mass, concentrated at its centroid, and a moment of inertia about its centroid. The chaser and target masses and moments of inertia have the same values throughout. The specific chaser and target state values supplied to the controller during a control step varied across training runs as experiments were performed, but every value supplied has additive zero-mean white Gaussian noise scaled by 0.05, to represent sensor and state uncertainty.

Standard 2D dynamics and kinematics give the chaser's response to applied forces and torques which come from the controller. Gravity is zero, so the chaser responds only to the forces and torques outputted by the controller. The target is assumed to have a constant angular velocity and is not transiting – the chaser transits towards it. This is true for the entire duration of a simulation.

Both objects are represented as rectangles. The chaser is a 1 m × 1 m square whereas the target can have more arbitrary rectangular dimensions (could evolve up to a side length of 6 m in either dimension). A fixed point in the chaser's reference frame, 2 m from its centroid in the *x*-direction, is designated the robot end effector or "hand". Similarly, a fixed point in the target's reference frame is designated the grapple point, or "handle". The handle location can be varied but is constrained to be outside the boundaries of the rectangle that represents the target (Fig. 1). If the chaser controller can keep its hand point within a specific minimum distance of the target's handle point for a set time duration, the chaser is considered to have captured the target. Ideally, the controller avoids collisions and minimizes fuel use while achieving a capture.

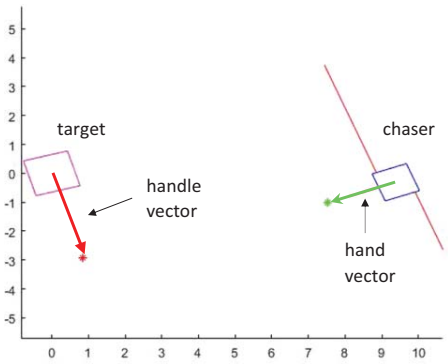

Figure 1. Models used in the simulation. Chaser (blue square) with its hand (green dot) and target (magenta rectangle) with its handle (red star). The thin red lines visualize the chaser's hypothetical jets.

To simplify the computational requirements for this proof-of-principle, it was reduced to two-dimensions from three. In two-dimensions, the dynamics are linear. Each object has three degrees-of-freedom which define its state; its position, *(x,y),* and angle relative to a reference. The controller sets the value of force and torque on the chaser. The set force and torque values could easily be translated into thruster forces; with 6 jets (3 pairs of opposing jets) the 2D chaser would be fully actuated and holonomic.

The performance of a controller is assessed by simulating the actions of the chaser under the control of that controller for each of a population of chaser-target scenarios. A scenario specifies the target initial conditions as well as parameters for a successful capture (Table I). The time history of the chaser forces and torques, as well as chaser and target poses, in response to a chaser controller, are simulation outputs. The simulations were designed to train the GP controller to capture the target.

TABLE I. Input Scenario Vector

	parameters	***description***
	maxcapdist	maximum distance between hand and handle sustained for *mincapture* time steps; defines a successful capture
	mincapture	minimum consecutive time steps that hand and handle points have to be within *maxcapdist* to consider the target captured
	maxang	cosine similarity between the hand and handle vectors; successful capture ⇨ < *maxang* (Figure 1)
	maxcollisio n	count of chaser and target overlaps in time steps; if exceeded, terminate simulation in a collision
unscheduled	target	handle position in target-fixed frame, $[x_{h,k}, y_{h,k}]$
		target dimensions in the target-fixed frame: $[x_{t,k}, y_{t,k}] \in \{[1.6]\ m, [1,6]\ m\}$ for scenario, k
semi-	target	initial angle
		angular velocity

B. Simulator Predicts Performance of GP Controller

The GP algorithm is shown in Fig. 2. A simulation applies one controller to one scenario for 20 seconds, the duration for the chaser to capture the target. The simulation time step is 0.01 seconds. Simulations start with the target at the inertial reference frame origin and the chaser, at rest, 10 m from the target in the *x*-direction. In some training runs, the chaser started with its hand facing away from the target, while in later ones the chaser hand and target handle face one another.

Every 0.1 seconds within a simulation the controller accesses the chaser state, target state and scenario vector, applies the controller which outputs the chaser forces and torque to update the relative chaser and target poses. The (*x,y*) force components each have a maximum absolute value of 10, while the torque has a maximum absolute value of 5. At each time step, these absolute values were multiplied by the time step and summed to represent fuel use.

Each generation applies a population of 100 controllers to the 100 scenarios which results in 10, 000 simulations. Over 12, 000 generations (a training run) this results in 120 M simulations to yield an evolved controller.

In reality, the force translating the chaser and the torque rotating it would be actuated by differential jet pairs on its periphery. A more accurate fuel use model would simulate these jets and track how long they fired and at what thrust, which would be proportional to fuel use. For the simulations however, the controller calculated a force and torque, and tracking them was considered to be adequately analogous to fuel use in a system with jets.

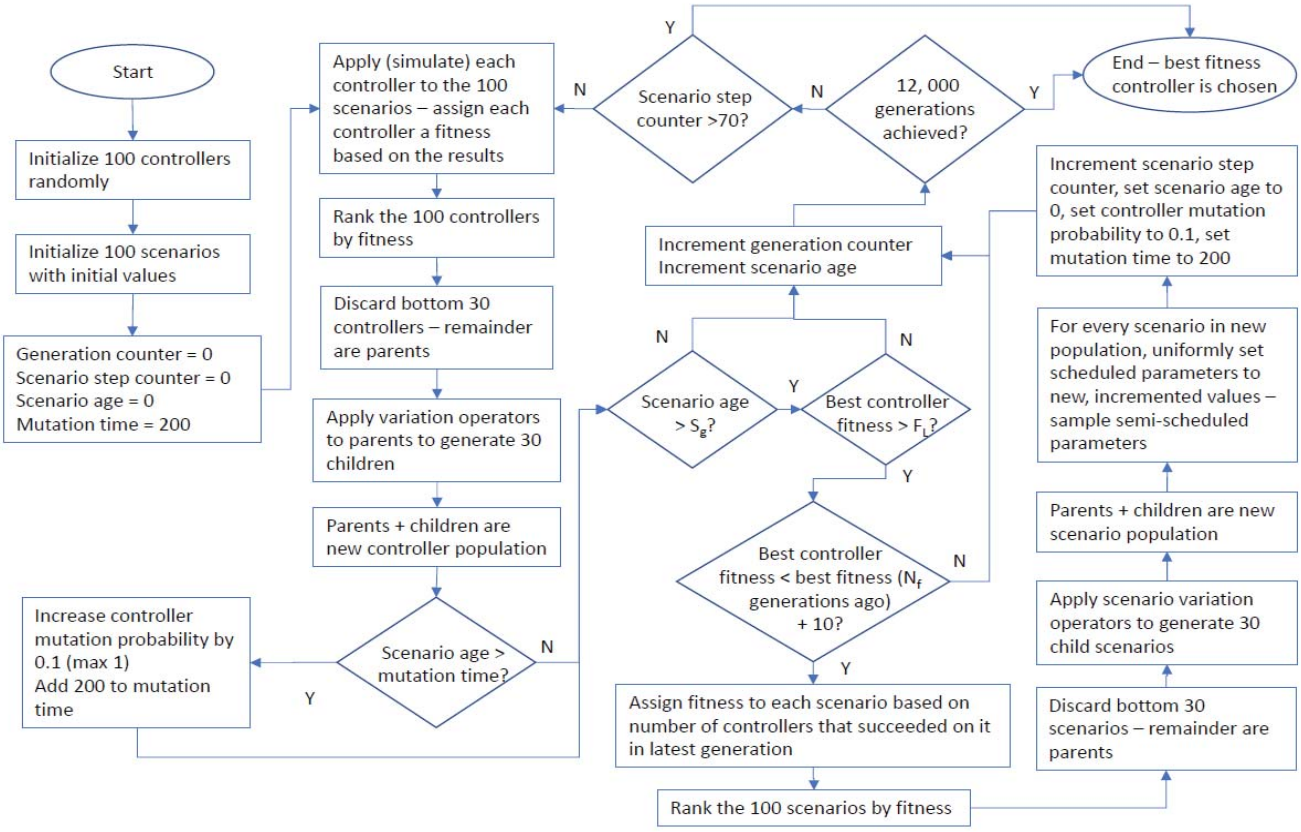

Figure. 2 General arrangement of the genetic program controller designed and implemented. Shown, is the process that occurs every 12, 000 generations. This constitutes one training run which has 120 M simulations.

There are 2 conditions where a simulation terminates early. The first, is a successful capture. If the hand and handle points are within distance, *maxcapdist*, for *mincapture,* consecutive time steps, then the chaser has captured the target. In some simulations, there were further constraints on a successful capture. For an actual capture maneuver, the chaser arm should face the capture point, so a condition was added to ensure this. For each time step, the cosine similarity between the hand and handle vectors was calculated, in the global inertial coordinate frame, and if this is not < *maxang*, the capture was unsuccessful. This enforces the chaser orientation to the target during capture.

The second condition where a simulation terminates early is a collision. At every time step the simulator checks if the chaser and target centres of mass are within a minimum threshold distance, and if so, checks for overlap between the chaser and target. As both are modelled as rectangles, this was simple. The simulator tracks the number of time steps with overlaps and if it exceeds *maxcollision*, the simulation is terminated by a collision. Unlike for captures, the overlaps did not have to be consecutive to count as a collision. Each simulation returns the number of time steps that satisfied the capture and overlap criteria, fuel use, simulation time at termination and *result* which is 0 for a timed-out simulation, −1 for a collision and 1 for a capture.

Although the canonical linear GP training set up was implemented in MATLAB, the simulator itself was subsequently realized as a C++ MEX file accessible by MATLAB [18]. The simulator input takes a linear GP controller as a 2D array, and a 2D array of scenarios and performs a simulation with that controller for each scenario.

C. Introns

During training, each simulation runs the controller every control step (0.1 s), for a maximum 200 (20 second / sim at 10 Hz) GP evaluations / simulation. This is a significant part of the simulation's running time. To lower the computational cost, structural intron detection was implemented. Introns are lines of the GP program that have no effect on the program output, and thus do not need to be executed.

An extra value was added to every GP line: a '1' if the line should be executed and '0' if it is an intron. Every generation, after the creation of children, the intron detection routine checks the GP controller population, and toggles this

bit as appropriate for each line of every GP. Structural intron detection is performed as in [19]. The intron detection routine monitors which registers were used as controller outputs and puts them in a list of important registers. It moves through the GP lines from the end to the beginning. If the output of a line (the register that is changed) is a register on the list, that line is marked as a non-intron. Any registers that the line uses to generate that output are added to the list.

D. GP Operations and Limits

The basic operators available to the GP are {+, −, ÷2, ×2}. To this was added the nonlinear conditional operator **if** $X < Y$ **then** $X = -Y$, and in later tests the cosine operator $X = cos\ (X)$ and the exponential and log operators $X = e^X$ and $X = ln(X)$, where X is the output register value and Y is another register or an input value. The exponential and log operators have the potential to cause errors or make values diverge, so conditions were added to guard against this: if the input to the exponential operator was greater than 22, the output was capped at 3, 584,912,856, and if it was less than -22, output was capped at 0.000000000279. If the input to the log operator was less than 0.00000000206, the output was capped at -20, and if it was greater than 1,586,013,452,000,000, output was capped at 35. Further issues with divergence were noted, so a condition was added so that if any GP line's output was greater/less than a threshold, it would be capped.

E. Registers and Memory

The number of registers the GP accesses is set to 12. The first 3 register values are the controller force (x, y) and torque output, subject to the force and torque limits. It was thought the controllers might benefit from retaining inputs and outputs, so the last 4 registers are set as memory; their values were initialized to 0 at the start of the simulation like all the other registers, but were not reset to 0 between control steps, giving the controller an ability to record values it could use.

The GP training is discussed next.

IV. Genetic Program Training

This section describes the scenario vector (Table I) in more detail and how its elements evolve. A scenario step is when all members of the scenario population are changed. Some scenario parameters change on a schedule and those that do not are evolved (like the controllers). A scenario generation, S_G, spans multiple controller generations (nominally, 60). The specific number of controller generations per scenario generation varies, as a scenario step can only occur when certain criteria are met.

A. Scheduled Parameters

The training evolves a GP controller to guide the chaser to capture the target without colliding with it. Ideally, the chaser keeps its hand co-located with the handle for at least a second without overlapping the target. This is unachievable as an initial training goal. Newly initialized GP controllers produce random outputs, and thus when generating the control force and torque would guide the chaser on a random path through an unlimited physical space. It is impossible for a randomly moving chaser to keep its hand coincident with the moving target's handle long enough to achieve a capture by chance. If the initial GP controllers do not achieve a capture, then there is no basis to rank and evolve them towards the goal.

Instead, the GP controllers are progressively taught the goal, by evaluating them initially on easier goals that can be achieved by chance, and then incrementally changing the evaluation criteria towards the ultimate (capture) goal. This was the purpose of the scheduled scenario parameters *maxcollision*, *mincapture, maxcapdist,* and *maxang*. If initially *maxcollision* is set to 50 time steps, *mincapture* to 1 time step, and *maxcapdist* to 8 m, then the task is easier, and most controllers in the population are likely to achieve it. The hand only has to be within *maxcapdist* of the handle for 1 time step to achieve a capture, the chaser and target have to overlap for 50 time steps before it is considered a collision, and the hand only has to be within 8 m of the handle. The chaser and target start 10 m apart, and the hand and handle are often closer than that, so many controllers in the population could randomly achieve this capture.

Once the controller population evolves through sufficient generations so that most achieve target capture, the scenario parameters are made more stringent – decrease *maxcollision*, increase *mincapture* and decrease *maxcapdist* incrementally, in a scenario step. The controllers that succeed under the less stringent parameters become the seed to evolve controllers that could succeed under the more stringent conditions. Over a training run, these scheduled parameters change incrementally until the goal is achieved, i.e.: *maxcollision* = 1 such that any overlap is a collision; *mincapture* of large enough time steps to be at least 1 second, and *maxcapdist* that is a small fraction of the chaser's dimensions.

Parameter *maxang* limits the maximum cosine similarity between the hand and handle vectors. It is initially 0, meaning the hand and handle vectors could be perpendicular, and then gradually decreases to narrow the range of permissible chaser capture angles. *maxang's* initial value depends in part on the state of other starting variables. The best solution was to hard-code the chaser to start with its hand facing the target so it was relatively easy to achieve initial successes with *maxang* set to 0, as little chaser rotation was required.

The choice of the schedule to change these parameters, i.e. how much change in each scenario step, whether one parameter should approach its final value faster than others, etc., had an impact on the quality of the solutions. Investigating ways to schedule the parameter changes was part of the training experimentation.

B. Unscheduled Parameters

The other parameters that define a scenario are the target (x, y) dimensions and the (x, y) coordinates of its handle point in the target frame and the target initial angle and angular velocity. The scheduled parameters *maxcollision*, *mincapture*, *maxcapdist* and *maxang* are fixed goals; the chaser must get its hand close to the target's handle for a certain length of time without colliding with the target. For the remaining parameters it is easy to evaluate a parameter

value's quality, but uncertain what quality could be reasonably achieved. For example, a controller that could achieve capture with a faster rotating target would be better than one that could not, but it was unknown what rotation levels a controller could ultimately evolve to handle, and a faster rotating target was secondary to achieving consistent, precise captures. It was decided to adopt a coevolution strategy for these parameters, similar to that described in [16] and [20].

Every scenario vector in the same generation *had the same values for the scheduled parameters*, but the unscheduled parameters varied. When the scheduled parameters were changed, each scenario was given a fitness score based on how difficult it was for the GP controllers to perform a capture. The scenarios were ranked, the lowest ranking *gap* scenarios were discarded, and the highest ranking ones retained as parents to generate child scenarios with scenario variation operators. The fitness for a scenario was based on the success rate of the GP tested on it and on a distance heuristic that quantified how different it was from other scenarios. As the goal was to evolve a general controller, if the scenario population remained diverse, the evolved controllers would be more general (desired). The scenarios were subject to ranking and child generation only occurred when the scheduled parameters were changed; this was so the scenarios would not change at the same rate as the controllers so the controllers would have time to adapt to a new scenario population.

The shape and grapple point of space debris will vary significantly, and some combinations may be more difficult than others to achieve capture. A target that is a square and has a handle far from its periphery would intuitively be easier to capture than a target that is a long rectangle with a handle aligned with its short dimension and close to the periphery. This would pose greater timing challenges for the chaser, as it would have to avoid colliding with protruding parts as it approached the handle. All unscheduled parameters had limits to prevent the training run from evolving difficult but unrealistic scenarios.

C. Semi-scheduled Parameters

The target initial angle and angular velocity were unscheduled parameters initially, but it was observed they did not change much from 0, and it was desired that the controller be applicable for different target orientations with rotation. So, the initial target angle and angular velocity became semi-scheduled parameters. When the scheduled parameters changed and the scenarios were ranked and evolved, a new value for initial angle and angular velocity were drawn from either a scaled uniform or normal distribution, but the scaling factors increased from 0 to a maximum on a schedule. So, the initial scenarios all had the same target angle with no rotation, but over training, as the scheduled parameters changed, the range of angles and angular velocities across scenarios grew larger.

D. GP Controller Fitness

The fitness for each GP controller is based on the *result* array returned by the simulator. This array has a value, for each scenario in the population, that is either 1, 0 or −1 depending on whether the GP resulted in a capture, timeout, or collision, respectively. Summing the array gives a representative GP fitness measure. The overarching intent is that however fitness was assigned, captures are better than timeouts and timeouts are clearly better than collisions, as a failed removal is better than a chaser-target collision. Minimizing fuel use also contributed to a GP's fitness. The fuel factor in (1) was subtracted from each *result* array value.

$$\text{fuel factor} = \frac{\text{fuel used}}{\text{max possible fuel use}} \times 0.01 \tag{1}$$

Then, the *result* array was summed for the GP fitness. GPs with similar numbers of captures, timeouts and collisions are further ranked by fuel use, but the scaling factor of 0.01 in the fuel factor prevents a GP with fewer captures and more timeouts or collisions from beating one with more captures and fewer collisions based on fuel consumption.

E. Scenario Fitness

The fitness assigned to a scenario reflects the success rate of all GPs for that scenario, and a heuristic quantifying its difference from other scenarios. Given target dimensions, $\mathbf{T}_k = x_{t,k}\hat{\boldsymbol{\imath}} + y_{t,k}\hat{\boldsymbol{\jmath}}$ and handle position, $\mathbf{H}_k = x_{h,k}\hat{\boldsymbol{\imath}} + y_{h,k}\hat{\boldsymbol{\jmath}}$ for scenario k, the distance, $D_{k,l}$ between scenarios k and l becomes:

$$\mathrm{D}_{k,l} = \{|\mathbf{T}_k - \mathbf{T}_l| + |\mathbf{H}_k - \mathbf{H}_l|\} \times 0.5. \tag{2}$$

In this way, a scenario was compared to other scenarios and these distance measures were further summed for a scalar representative of a scenario's overall difference from the population.

To generate the fitness for a scenario, the Breeder model tracked the results of each GP controller for each scenario in the population. It set the scenario fitness as the number of simulations with that scenario where the chaser did not capture the target. If the chaser did not capture the target in any simulations, it sets the fitness to 0, as in the point fitness in [16] (this ensures the model selects scenarios that are difficult but not impossible for the GP population to solve). Then, it scaled the fitness by the scenario's scalar difference measure (2). To reiterate, only unscheduled parameters are evolved this way; the scheduled parameters are constant across all the scenarios in a generation. Scenario fitness was based only on the GP success rate for the GPs in the current generation in which scenario selection was taking place, not on the success rate of GPs in previous generations.

F. Scenario Variation Operators

The variation operators for the scenario evolution are one-point crossover and mutation. In one-point cross-over, two scenarios are randomly drawn from the population of parent scenarios. An index between 1 and the scenario vector length minus 1 is randomly drawn with uniform probability. Two children are created from the parents by swapping the scenario vector sections below the randomly selected index.

In mutation, a single parent scenario is selected. For each non-scheduled parameter in the scenario vector, the difference between the maximum and minimum allowed values is divided by 20 to give an increment. If a number drawn from a uniform random distribution is less than a scenario mutation parameter, $Pscene$, that parameter is selected for mutation. The increment is added to, or subtracted from, the parameter with equal probability. If the addition/subtraction puts the parameter outside its maximum/minimum allowed value, the increment is instead subtracted/added.

G. Scenario Step Condition

In one step of scenario evolution, the scheduled parameters change according to the preset schedule and the unscheduled parameters are evolved. The controller population is ranked and evolves every generation, but scenario evolution steps occur much less frequently – whenever a specific condition is met. The choice of condition was a subject of experimentation during the training runs. Initially, a scenario step was taken if the current scenario population was at least S_g generations old and the current best GP controller fitness (~ number of successes) was > F_L. However, the GP tends to improve in sudden jumps, with the best fitness improving due to a sudden chance improvement, and then the mean GP performance gradually catching up as improvements from the best GP propagate through the population with the usual crossover and selection. Ideally, the best GP evolves to perform as well as it can on the current scenarios, and the other GP controllers acquire its successful traits, which allows the population to have a good starting point for new scenarios. Instead, the best fitness would make a sudden jump above the limit F_L, and the scenarios would change, even though the full extent of the improvement may not have been realized and the improvement was likely not well established in the population. Therefore, a further condition was added. The best fitness a number of generations, N_f, before the current generation had to be worse than the current best fitness by at most 10 for the scenario change to occur. This condition requires that the performance plateau before a scenario step, ensuring that chance improvements which allow the population to attain a new best fitness > F_L, have been fully explored and established in the population.

H. Training Termination Criteria

Each training run has a maximum number of generations (in later runs, 12,000) with which to evolve the controller population. Usually this was sufficiently large, and S_g sufficiently small, that if the scenario steps were taken every S_g generations and not delayed, the scheduled parameters would reach their final values before reaching the maximum generations. Then, scenario steps would continue to be taken, but the scheduled parameters would no longer change. The unscheduled parameters continue to evolve, and the semi-scheduled parameters would continue to be redrawn from distributions with scheduled change in the scaling factors. It was of benefit to continue training after the final scheduled values were reached, as it exposes the controller population to more diversity from the unscheduled parameters. The schedules were such that all scheduled parameters reach their final values after 50 scenario steps; if the training reached 70 scenario steps, this caused an early training stop.Controller Inputs and Coordinate Systems

One of the main experimentation topics during training runs was the input state vectors to the GP controller during a control step. This was expressed in a polar coordinate frame centered on the chaser with the 0-radians direction aligned with the chaser hand. Coordinates are a distance and an angle, with the hand coordinates constant since it is fixed relative to the chaser. Target velocity values were also expressed in this chaser-centric polar coordinate system (3), where $(x_{t,k}, y_{t,k})$ and (v_x, v_y) are the position and velocity components, respectively. (r, θ) and $(\dot{r}, \dot{\theta})$ are the polar coordinates and their first derivatives.

$$r = \sqrt{x^2 + y^2}, \theta = \text{atan2}\left(\frac{y}{x}\right) \tag{3a}$$

$$\dot{r} = \frac{x \cdot v_x + y \cdot v_y}{\sqrt{x^2+y^2}}, \dot{\theta} = \frac{-y \cdot v_x + x \cdot v_y}{x^2+y^2} \tag{3b}$$

It is easier to evolve a controller that navigates to the handle when its directions are from the state vector. The controllers were given a state vector of length 14 consisting of the target position and velocity, the target angle minus the chaser angle, the target angular velocity minus the chaser angular velocity, the global angle and angular velocity of the chaser, the polar position of the hand and handle and the target dimensions. All values had additive zero-mean white Gaussian noise. The first 3 register values at the end of the GP controller execution were the applied force (in polar coordinates) and torque. The force was transformed to global inertial Cartesian coordinates, and both force and torque were capped by their respective limits.

V. Testing and Results

The initial target angle and angular velocity were semi-scheduled. At each scenario step they were sampled uniformly from a range that increased according to a schedule. This was so the GP controller population would learn to work with higher target angles and rotation rates.

A series of training runs were conducted to experiment with parameter settings and strategies. There are many possible parameter combinations and strategies to explore, so these experiments to date are not exhaustive. Achieving the best performance from the GP controllers is on-going, and the results reported are general observed patterns.

In the first experiments, the decision to take a scenario step was solely based on S_g, which was set to 30. In those experiments, the best fitness stayed near 100 for 20-30 scenario steps, then started to fall and went to 0 before the end of the training run. A condition was added so that the scenario could only change if the current best fitness was over 20 (~ 20% capture rate). This prevented the best fitness

from going to zero, but also prevented the training run from reaching the limit of the scenario steps. The best fitness declined with the scenario steps until it hit a point where it was < 20 from which it could not improve, preventing the training run from taking further scenario steps. The best fitness would spike and then immediately trigger a scenario change, so the condition was added to ensure the best fitness plateaued before the scenario changed. S_g was changed to 60, N_f was set to 40, and fitness limit F_L was changed to 30 from 20.

The first training runs evaluated the controller fitness simply by summing the *result* output. The controllers did make progress learning the task, but the chaser moved in an erratic manner, and when it missed a capture, it would accelerate and rotate to high speeds, and move away from the target. Adding fuel use to the GP fitness metric trained the chaser controllers to move in a much less abrupt manner; they tended to perform a gentle arc when they missed their capture, rather than accelerating away. None evolved a try-again behaviour. This is the subject of future work.

In general, the performance of the controllers was promising, but simple. They did not tend to exhibit much adaptability, but instead learned relatively simple, stereotyped movements that would allow them to perform a capture the minimum allowed number of times. They usually had the chaser approach the target in a gentle arc, always from the same side and performing a slow rotation to match the hand to the handle at roughly the same position in space. These programs were not simply memorizing movements, as they managed to match under different initial conditions, but they seemed to tune the same basic movement rather than planning an approach more deliberatively. The controllers had clearly adapted their behaviour based on whether the training simulations were hard-coded to start with the chaser facing away from the target or towards it.

In initial runs, *maxcollision* had an initial value of 50 and a final value of 1, *mincapture* was initially 1 and went to 50, and *maxcapdist* started at 8 m and had to work down to 0.05 m. The increments were chosen so all reached their final values within 50 scenario steps. *maxang* had an initial value of 0 and a final value of −0.96, meaning an allowed relative angle range for the chaser during capture of 180° initially and ~30° at the end. After experimentation, it was decided that the training runs should reach the final *maxcapdist* value relatively early in training, so that the controllers would learn the basic desired behaviour, and then be able to adapt to higher target angles, angular rates and dimensional aspect ratios. So, the scheduled changes in *maxcapdist* were increased so that it reached its final value in 10 scenario steps rather than 50.

There were minor changes evolved during training runs in the handle location, and the target dimensions, initialized as a square of side length 1. Variation of between 1 and 1.5 were observed. The maximum allowed side length was 6, so the targets could have evolved to be more rectangular than they were but did not. There was also not much change noticed in the target angle and angular velocity, which stayed close to their initial 0 values when treated as evolved unscheduled parameters rather than sampled semi-scheduled parameters.

It was important that the controllers develop the ability to adapt to varying target motion, which was why initial target angle and angular velocity were changed to semi-scheduled parameters. The scaling factor on the sampled angle and angular velocity distributions always started as 0 but increased up to π radians for the angle and 1 radian/s for the angular velocity. The scheduled incremental changes in these scaling factors were usually set so that they approached their final values slower than the other scheduled parameters.

To prevent training runs from stalling before a scenario step, a condition was added. If the training run has been on the same scenario population for > 200 time steps, 0.1 is added to the mutation probability $Pscene$ in the mutation operator during controller child generation. This repeats every 200 time steps, up to a maximum $Pscene = 1$. When the training run takes another scenario step this probability resets to 0.1.

TABLE II. SUMMARY OF GP WITH BEST TRAINING PARAMETERS

	parameters	***values***
GP	12 registers	last 4 are memory
	8 operators	
	400 lines	343-389 non-intron lines
	100 controllers	14-component noisy input state vector
chaser	$1m \times 1m$	initially, 10 m from target
	hand 2m from centroid	hand faces the target
target	evolved target dimension	$1 \rightarrow 6\ m$
	initial angle drawn from uniform random dist'n	scaled to 0 at 1st scenario step and π afterwards; 1 scenario step to final value
	1 component of handle location	$0 \rightarrow 4\ m$ from target periphery
	ang velocity drawn from uniform random dist'n	Scaled $0 \rightarrow 1\ rad/s$; 50 scenario steps to final value
scenario	100 scenarios	
	maxcollision	$50 \rightarrow 1$; 50 scenario steps to final value
	mincapture	$1 \rightarrow 100$ 50 scenario steps to final value
	maxcapdist	$8 \rightarrow 1\ m$; 10 scenario steps to final value
	maxang	$0 \rightarrow -0.96\ radians$ (30°); 25 scenario steps to final
training	length of scenario gens	$S_G = 60$ controller gens
	fitness limit	$F_L = 30$
	best fitness, N_f gens prior to current one, be less by at most 10 to trigger a scenario change	$N_f = 40$

The best training conditions are summarized in Table II. Fig. 3 shows the results of one such training run. The best fitness starts close to 100 (i.e. nearly all simulations end in captures) and quickly decreases. Eventually the training run hits a point where even the best controllers can not get a fitness better than about 20, so no further scenario steps can be taken, and the training run makes no further progress.

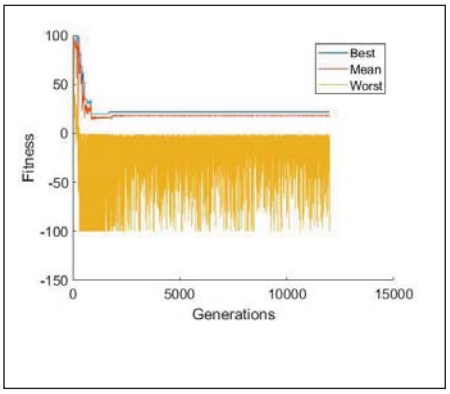

Figure 3. Best, mean and worse fitness for a sample training run.

The controllers show qualitatively similar stereotypical behaviours for these training conditions as described previously, but they made more scenario steps in training and thus can handle a greater range of target motions, so these training conditions and choice of controller input format were beneficial.

VI. Discussion

This method would benefit from further development but there are some promising results. The evolved controllers achieve capture about 20% of the time on scenarios of middling difficulty. It is likely that since F_L was set to 30 it explains why the controllers approach from the same side, because if they fail at all scenarios that require the opposite approach direction, they can still get up to half correct. This is not a real issue as the symmetry in the problem means a simple condition could be added to mirror the response.

While the controller calculates output forces in polar coordinates, due to the lack of a coordinate change it must learn to calculate them in a polar frame aligned with the global coordinates, which likely negates some of the value of the chaser-centric inputs.

The *maxcapdist* parameter appears to set the maximum performance of the controllers. When *maxcapdist* is small enough that the controllers must track the moving handle, not just get nearer to the target, difficulty increased markedly. This could be improved with *maxcapdist* decaying exponentially rather than linearly. Chaser-centric coordinates help the controllers achieve more difficult captures. It was expected that the scenario step at which *maxcollision* and *mincapture* crossed might be associated with a significant delay, as cases that were previously captures may become collisions. Fortunately, this was not observed. The best controllers in training, rarely, if ever show collisions.

Program length impacts computational cost, but it may be unavoidable that the controllers require more lines. The GPs used most of their allocated 400 lines, with only a small fraction being structural introns.

VII. Future Work

The simulator should be converted to simulate the chaser's 3 pairs of opposed jets required to control the motion from the forces and torques calculated. The register values returned by the controllers could be calibrated as jet thrust values. Alternatively, the jets could be given a fixed, on-off thrust, and the register values' softmax values used to decide which of the 27 possible permutations of jet firings should be chosen. Additionally, the simulator could be extended to 3 dimensions. One interesting change would be to change the controllers from the basic canonical GP, to more flexible one. like symbiotic bid-based GP (SBB GP) [16], possibly training the SBB controllers on a variety of simpler tasks to build a library of call-able actions to assemble into a more complex behaviour [21].

The problem could be extended into 3 dimensions to see if the training methodology can handle more complex and realistic dynamics. On-Earth testing can use 2D tests as proofs-of-concept for space systems; Dalhousie University is currently constructing an air-bearing based testbed that will allow testing of space systems in a effectively frictionless 2D environment, and the results of the GP work to-date could be immediately applied and tested with this set-up.

VIII. Contributions

We apply linear GP to the complex task of controlling a 2D spacecraft to perform a rendezvous with an uncontrolled spinning debris object and precisely match its own manipulator point with a moving grapple point fixed to the debris object. Our training methodology of parameterizing the task with scheduled values allows us to vary the complexity level of the task continuously and holistically train an open-ended program on the entire task, rather than splitting it into human-chosen subcomponents. Pairing this with a coevolution framework allows less important task goals to be pursued to an extent subject to the more important goals reflected in the scheduled parameters.

IX. Conclusions

The canonical linear GP was applied to a simplified version of the spacecraft guidance and control problem for debris removal. The evolved controllers showed promise but would require further experimentation to achieve the full project goals. It is likely that a more modular form of GP than the canonical linear GP would be required to achieve better results.

References

[1] D. Kessler, N. L. Johnson, J. C. Liou and M. Matney, "The Kessler Syndrome: implications to future space operations," Advances in the Astronautical Sciences, vol. 137, 2010.

[2] K. Stanley, B. D. Bryant and R. Miikkulainen, "Real-time neuroevolution in the NERO video game.," IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, pp. 653-668, 2005.

[3] M. Lewis, A. Fagg and A. Solidum, "Genetic programming approach to the construction of a neural network for control of a walking robot," in Proceedings of the IEEE International Conference on Robotics and Automation, 1992.

[4] P. Nordin and W. Banzhaf, "An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming," Adaptive Behavior, vol. 5, pp. 107-140, 1996.

[5] B. Kadlic, I. Sekaj and D. Pernecký, "Design of continuous-time controllers using Cartesian genetic programming," IFAC Proceedings Volumes, vol. 47, pp. 6982-6987, 2014.

[6] M. M. Khan, G. M. Khan and J. F. Miller, "Evolution of optimal ANNs for non-linear control problems using Cartesian genetic programming," in In Proceedings of IEEE International Conference in Artificial Intelligence, 2010.

[7] X. Luo, M. Heywood and A. Zincir-Heywood, "Benchmarking a recurrent linear GP model on prediction and control problems," in Intelligent Control and Automation. Lecture Notes in Control and Information Sciences, D. Huang, K. Li and G. W. Irwin, Eds., Berlin, Springer , 2006, pp. 845-850.

[8] M. F. Brameier and W. Banzhaf, Linear Genetic Programming, 1st ed., Springer Publishing Company, Incorporated, 2010.

[9] D. Hein, S. Udluft and T. A. Runkler, "Interpretable policies for reinforcement learning by genetic programming," Engineering Applications of Artificial Intelligence, vol. 76, pp. 158-169, 2018.

[10] C. K. Oh and G. J. Barlow, "Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming," in Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), 2004.

[11] D. C. Dracopoulos, "Evolutionary control of a satellite," in Second International Conference on Genetic Programming, 1997.

[12] P. Silva, I. Abreu, P. Forte and H. M. Amaral, "Genetic algorithms for satellite launcher attitude controller design," Inteligencia Artificial, vol. 22, pp. 150-161, 5 2019.

[13] F. Marchetti, E. Minisci and A. Riccardi, "Towards intelligent control via genetic programming," in IEEE World Congress on Computational Intelligence, Glasgow, 2020.

[14] F. Aghili, "Pre- and post-grasping robot motion planning to capture and stabilize a tumbling/drifting free-floater with uncertain dynamics," in 2013 IEEE International Conference on Robotics and Automation, 2013.

[15] Q. Li, J. Yuan, B. Zhang and C. Gao, "Model predictive control for autonomous rendezvous and docking with a tumbling target," Aerospace Science and Technology, vol. 69, pp. 700-711, 2017.

[16] P. Lichodzijewski and M. Heywood, "Symbiosis, complexification and simplicity under GP," in Genetic and Evolutionary Computation Conference, Portland, 2010.

[17] H. Mühlenbein and D. Schlierkamp-voosen, "Predictive models for the Breeder Genetic Algorithm," Evolutionary Computation, vol. 1, no. 1, pp. 25-49, 2 1993.

[18] MathWorks, "MATLAB Documentation: MEX File Functions>," [Online]. Available: https://www.mathworks.com/help/matlab/call-mex-file-functions.html.

[19] M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining.," IEEE Transactions on Evolutionary Computation, 2001.

[20] W. D. Hillis, "Co-evolving parasites improve simulated evolution as an optimization procedure," Physica D: Nonlinear Phenomena, vol. 42, pp. 228-234, 1990.

[21] S. Kelly and M. I. Heywood, "Knowledge transfer from keepaway soccer to half-field offense through program symbiosis: building simple programs for a complex task.," in Genetic and Evolutionary Computation Conference, 2015.