Brian Delgado, Justin Moustouka, Dhaamin Buford, Caiden Puma 3-21-2022

ENGN0040 Dynamics and Vibrations Design Project: "Non-Trivial Pursuit" Report

Abstract:

Purpose: The purpose of this design project is to create and implement effective predator and prey strategies into MATLAB. Our goal in programming the predator is to generate a strategy that can catch prey that utilize various strategies of their own, within the time allotted for each simulation. Our goal in programming the prey is to create a universal strategy that can evade different predators for the same allotted time.

Ideas/Methods: As a collective, we generated many ideas to implement into our complete predator or prey strategies. Beginning with the prey, our main focus was to travel directly away from the predator's position and velocity vector. We added some cyclic variations using sine and cosine functions to alleviate the inferior straight-line speed of our prey in relation to the predator. For our predator, a similar method was attempted, which was to predict the motion and velocity of the prey being chased and travel in that direction. The methods used to determine these predictions included motion and derivative relations. Also, there were components added to the force inputs to our predator and prey that would allow them to refuel safely and carry out the other commands with a level of ground avoidance to prevent crashing.

Results: Testing for our design consisted of a head-to-head competition between another group's code and strategy. The results for both our predator and prey against this group were fairly successful, winning most of our matchups in a relatively short time interval. Whether it was a defect in the other group's code or an unintended aspect of our strategy, the opposing group's predator had crashed multiple times in pursuit of our prey. Also, we tested our code against Prof. Bower's aggressive code for predator and prey. This testing was less successful, only getting two wins out of around eight attempts, one for each of our predator and prey.

Conclusion: From our results of testing our group has come up with many notes about our performance including things we should change about our code and strategy. First, we are satisfied with the performance of our code against a peer group because the majority of attempts were successful. We are also pleased with the balance of our strategies, between our predator and prey, although in more than half of our internal testing, the prey was caught by the predator. Lastly, we believe some components of our predator code that could've been changed is lowering the time interval for predictions and including a small component of motion going directly toward the position of the prey. For our prey, we believe a more calculated (rather than random) rate of direction changes would have increased success.

Introduction:

Problem Statement

In this design project, our team is tasked with creating and implementing effective strategies for a predator and prey particle in a MATLAB simulation exercise. Our goal is to code for a predator that can catch prey and a prey that can evade predators, both throughout the time interval of $0 < t < 250$ sec.

Design Requirements:

For this project, there were a number of design constraints meant to ensure fairness in competition and ensure smooth running of the ode solver function in the script. Most of these constraints are built into the code used to run the simulations and some are as follows: Max propulsive forces must not exceed a given magnitude (Predator: 1,275.3 N, Prey: 137.34 N), mass of Predator is 100kg and mass of Prey is 10 kg, finite fuel reserve (Predator: 500kJ , Prey: 50 kJ), random forces, drag force, and gravitational force will all act on the particles during simulation, and crash speed on impact with ground (Predator: 15 m/s , Prey: 8 m/s). In the driver code, there are fail safes in place that would correct any propulsive force over the limit amount.

Contents Covered in Report

The contents to be covered in the report include aspects of the design process, final design code and strategy, and results and conclusions that consider our team's success. We have decided to define success for this project as a victory over the opposing group during testing, also factoring in oversights that could be corrected, like crashing to the ground for either particle.

Strategies:

Pursuit Strategy:

The strategy we built for the predator, in ideal conditions, involved the prediction of the position of the prey relative to itself a few seconds in the future assuming a constant velocity for both the predator and the prey at time t. The predator would then direct its force to accelerate towards that predicted position in order to reach the position at the same time as the prey, hopefully catching it at the same time. The closer the predator and prey were, the smaller the time intervals of prediction would be, therefore more accurately predicting the prey's motion in order to precisely catch it. Additionally, another component to the force was always applied in the upward direction in order to mitigate any velocity in the negative y-direction, and increased as the predator approached the ground level, therefore avoiding many crashing incidents. Finally, as the predator moves, its loss of energy will require the use of a strategy which implements landing and refueling. In our strategy, once the predator reaches a certain energy 'threshold', its primary goal will be to land and refuel, by first allowing freefall then applying an upward force in order to slow the predator, before rising again to wreak havoc on its prey.

Escape Strategy:

To begin the simulation and create repeatable, constant conditions, the prey first directs its full force in the positive y-direction for the first few seconds to get airborne. The following strategy for the prey involved a similar strategy to our predator's, in which the prey directed its force away from the predator at every time interval. However, in order to add a greater degree of variation, and therefore beneficial unpredictability, a sine and cosine component was added to

this aspect of the force, which made its motion less predictable, and not necessarily in the exact opposite direction away from the predator, which may be more easily predicted by a sophisticated predator algorithm. The prey also had a similar code in order to avoid the ground by applying a greater positive component of force, and to refuel after crossing a threshold in energy levels.

Design and Testing Procedures:

Matlab Coding:

To initially get both particles off of the ground, we set the force acting on the particle within the first 5 seconds of the simulation to be equal to the maximum allowable force vertically using a conditional statement. Then, before determining the thrust force on our particles, the issues of refueling and crash prevention needed to be addressed. Since both the predator and prey have finite energy sources, the amount of energy that is used by the thrust force can be derived through the work-kinetic energy relation. To prevent the particle from running out of fuel and plummeting to the ground, we used a conditional statement to cause the particle to begin a refueling procedure after the fuel level drops below a certain amount. For the predator, this procedure is to fall freely until it reaches some critical height, determined by the maximum vertical force and the maximum landing velocity. The maximum vertical force is then applied to the predator, slowing its descent and allowing it to reach the ground at an appropriate speed to refuel. For the prey, the procedure was to "nose dive" using the maximum allowable thrust force downward to make the refueling process as quick as possible so as to avoid the predator or even cause it to crash into the ground in its pursuit of the prey. Since the random force may increase the velocity of the particle horizontally during its descent, we set the direction of the force to be equal to the negative of the unit vector in the direction of the prey's motion (vy/-norm(vy+0.25)) Using a conditional statement similar to that of the predator, the maximum allowable force would then begin to act in opposition to the prey's direction of motion to slow the prey after having reached a critical height, determined by its current velocity, the maximum thrust force, and the maximum landing speed.

To prevent the particles from crashing by any other means, the vertical force acting on the particles needed to grow as the altitude decreased. This inverse relationship between altitude and force could be easily represented using a fraction with altitude in the denominator. This fraction would then be multiplied by the vertical unit vector so its direction was restricted. The magnitude of the constant over the altitude was more or less arbitrary and, through the process of trial and error, we chose 1000. This allowed the force on the particle by the ground to be large enough so as to repel the particles, while also serving to keep both particles high enough above the ground, further reducing the risk of crashing. However, since both particles have an altitude of zero at the start of the simulation and whenever they reach the ground to refuel, the vertical force on the particles would approach infinity and cause the simulation to crash. To account for this, a constant 0.25 was added to the denominator so the force remained real at all times during the chase.

Following from the description of our predator strategy, we found it necessary that our predator be in "hot pursuit" of the prey. To accomplish this we set the force to be applied equal to the product of the maximum thrust force of the predator and a unit vector we defined as "predictdiry" (we also included a constant vertical force equal to the weight of the predator and a force that gradually increased as the altitude decreased to prevent crashes.) This unit vector uses

the current position of the predator and the prey as well as their current velocities to calculate the trajectory of the prey particle with an accuracy determined by the discrete time interval dt. In order to prevent the predator from overshooting the prey, we created a conditional statement such that dt would decrease, and thus the predicted position of the particle would approach its true path, assisting the predator in closing in on the prey.

For the prey, we saw it most effective to have the prey move away from the predator while also being capable of dodging when the distance between the particles became too small. To do so, we began by setting the force on the prey equal to act periodically in the i and j directions using trigonometric functions sine and cosine as functions of time. This caused the particle to execute small circular motions and change its height rapidly when the predator was far from it. This made it more difficult for predators with code similar to ours to be avoided. When the predator begins to close in on the particle, the force on the prey changes directions. When the distance between the predator and prey decreased beyond some amount, the prey would then use the maximum allowable force to move in a direction that was generally away from the predator with a trajectory that changed periodically as a function of time. This allowed the prey to continue to move with a more unpredictable trajectory, making it even more difficult for predators to predict the exact position of the prey at any given time. Upon nearing the predator even further, the prey was set to resume executing small circles and rapid changes in altitude, allowing the prey to dodge the predator at the last second or cause the predator to overshoot and create distance between the two particles.

Final Testing Procedure:

To test our final designs (aside from using the function tester provided in the project description), we saw fit to run our simulation and make note of the time it takes for the predator to catch the prey, for either of the particles to crash, or whether or not our prey escaped. If the number of crashes was minimal and the results of the simulation showed that the predator and prey were fairly evenly matched, then our assumption was that our code was ready for competition. In our final testing stage, we also took the opportunity to adjust some of the quantities used in our code. For instance, our initial dt was around 8 and would reduce to 4 when the predator neared the prey. However, we found that dt values of 5 and 3 worked best against our prey particles and made the predator more aggressive. After optimizing these quantities through the painful process of trial and error, we felt our code was ready to compete against the code of our peers.

Conclusion:

Results:

During our test against another group, our trials were overwhelmingly successful. Our predator caught the opposing team's prey significantly more than it escaped. Our prey was either caught or crashed to the ground in less than half of its attempts. While our predator was being tested, our group realized that when it got too close to the prey, the prediction mechanisms used did not work effectively enough to catch the prey, just follow very closely. This tactic turned out to be successful only when the prey needed to fall to refuel, then our predator would catch it on the ground. Our prey strategy worked well also and usually kept the other team's predator much further away then they did to ours. Many times, our prey would turn, making their predator turn

the opposite way or hesitate long enough for our prey to increase distance from the predator. Our predator was very close to their prey even on the attempts we didn't catch the prey.

With this consistent close following while sometimes coming short, we used our chance to edit for the predator code. To combat this issue, we decreased the amount of time (dt) used in calculating the predicted motion of the opposing prey when the prey was within a certain distance to our predator (10m). This change would make our turns and pursuit more sensitive when it comes close enough to the prey, hopefully making it close the gap during times it gets really close. After testing with this new code, our predator followed even closer to the opponent's prey and caught it faster as it reached the ground for refueling. However, this change did not really fix the issue in the air with our predator because it still often was not able to get close enough to catch, though it followed even closer behind.

Feedback:

The feedback received from Prof. Bower and the TA present at testing were overwhelmingly positive. We are aware that most groups were likely never to be told that their design stunk, however with the general success against an opposing group and competitiveness with Prof. Bower's code, we believe this feedback is fairly honest. Afterwards, we took a look at the code Prof. Bower had used against us and despite some differences, the main functions of the code were similar, as noted by the TA.

Ideas for Future Iterations:

After watching our predator and prey in action against an outside opponent, we noticed some things in our strategy that could be altered to find even greater success in that situation. For example, in the issue we highlighted earlier about the predator only following closely behind, we have seen in Prof. Bower's simulation that the predator could be much more effective than that. For one, we could consistently attack from above the prey, making it continuously evade closer down to the ground. This alteration would increase the likelihood the opposing prey would crash and it would also not allow the prey much room to escape, being trapped between our predator and the ground. Also, we thought of a fix that would not drastically change our code and would follow closely with our current strategy. This alteration would be to use a small component of the max propulsive force directly toward the prey's position. This alteration would work best if the closer our predator got, the smaller the time intervals should be when gathering data on the prey's position. In theory, this idea should help our predator fully close the space between itself and the prey, allowing it to catch it in the air.

We also developed some ideas for our prey to more effectively escape the opposing predator. One of those ideas would be to be more precise about the times at which we would have our prey change directions. Since the prey can change directions faster than the predator, it's best to use this attribute in the most effective way possible to combat the superior straight-line speed of the predator. This could mean calculating the time after a turn that the predator, at full force, could get close to our prey (~5m) and then change directions again. If the predator was not programmed well enough, this could be a way to waste time while the predator is simply following the curved path of our prey.

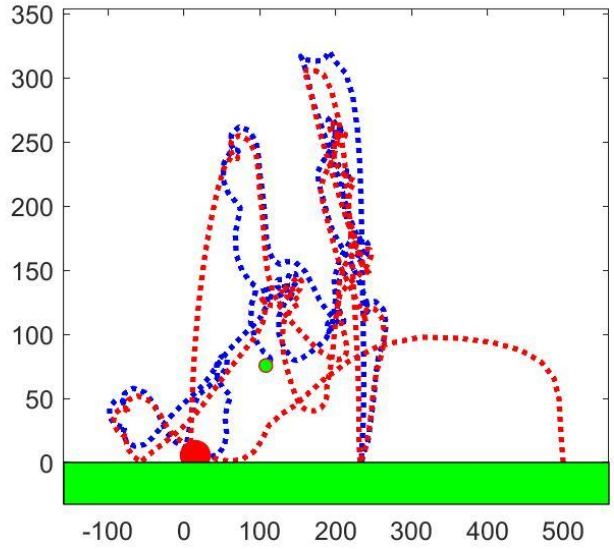
Appendix:

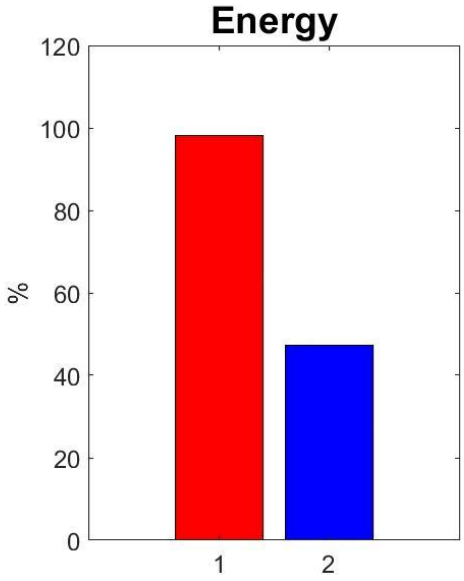
MATLAB Code:

```
function F = compute_f_BowerPower(t,Frmax,Fymax,amiapredator,pr,vr,Er,py,vy,Ey)
% Test time and place: 1:30pm @bh 096
% Group members:Dhaamin Buford,Caiden Puma,Justin Moustouka,Brian Delgado
%   t: Time
%   Frmax: Max force that can act on the predator
%   Fymax: Max force that can act on th eprey
%   amiapredator: Logical variable - if amiapredator is true,
%   the function must compute forces acting on a predator.
%   If false, code must compute forces acting on a prey.
%   pr - 2D column vector with current position of predator eg pr = [x_r;y_r]
%   vr - 2D column vector with current velocity of predator eg vr= [vx_r;vy_r]
%   Er - energy remaining for predator
%   py - 2D column vector with current position of prey py = [x_prey;y_prey]
%   vy - 2D column vector with current velocity of prey py = [vx_prey;vy_prey]
%   Ey - energy remaining for prey
%   F - 2D column vector specifying the force to be applied to the object
%   that you wish to control F = [Fx;Fy]
%   The direction of the force is arbitrary, but if the
%   magnitude you specify exceeds the maximum allowable
%   value its magnitude will be reduced to this value
%   (without changing direction.
    g = 9.81;
    mr = 100; % Mass of predator, in kg
    my = 10.; % Mass of prey, in kg
    predator_crash_limit = 15; % Predator max landing speed to survive
    prey_crash_limit = 8; % Prey max landing speed to survive
    Max_fuel_r = 500000; % Max stored energy for predator
    Max_fuel_y = 50000;  % Max stored energy for prey
    h_crit_y = 1.5*(0.5*my*norm(vy)^2)/(Fymax-(my*g));
    h_crit_r = 1.5*(0.5*mr*norm(vr)^2)/(Frmax-(mr*g));
    dt = 5;
    predictdiry = ((py+vy*dt)-(pr+vr*dt))/norm((py+vy*dt)-(pr+vr*dt));

  % Code to compute the force to be applied to the predator

  if (amiapredator)
     if (t<5)
        F = Frmax*[0;1];
     else
       if (Er<150000)
        if (pr(2)>h_crit_r)
           F = 0*[0;1];
           if (pr(2) == h_crit_r)
              F = Frmax*[0;1];
           end
        elseif (pr(2)<h_crit_r)
```

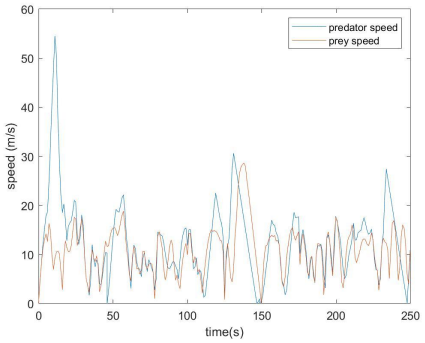
```
            F = Frmax*(vr/(-1*norm(vr+0.25))); %Force opposes current motion in i and j
            if(pr(2) == h_crit_r)
               F = [0;0];
            end
     if (norm(pr-py)<10)
        dt = 3;
     end
        end
       else
          F = Frmax*predictdiry + (mr*g + 1000/(pr(2)+0.25))*[0;1];
       end
     end
  else
   if (t<5)
        F = Fymax*[0;1];
   else
       if (Ey<14000)
        if (py(2)>h_crit_y)
           F = Fymax/2*[0;-1];
           if (py(2) == h_crit_y)
              F = Fymax*[0;1];
           end
        elseif (py(2)<h_crit_y)
           F = Fymax*(vy/(-1*norm(vy+0.25))); %Force opposes current motion in i and j
           if (py(2) == h_crit_y)
              F = [0;0];
           end
        end
       else
          F = Fymax/sqrt(2)*cos(t)*[1;0] + (Fymax/sqrt(2)*sin(t) + my*g +
1000/(py(2)+0.25))*[0;1];
        if (norm(py-pr)<50)
           F = Fymax*((py+vy*cos(t))-(pr+vr*sin(t)))/norm((py+vy*cos(t))-(pr+vr*sin(t))) +
(my*g + 1000/(py(2)+0.25))*[0;1];
          if (norm(py-pr)<15)
             F = Fymax/sqrt(2)*cos(t)*[1;0] + (Fymax/sqrt(2)*sin(t) + my*g +
1000/(py(2)+0.25))*[0;1];
          end
        end
       end
   end
  end
  end
```

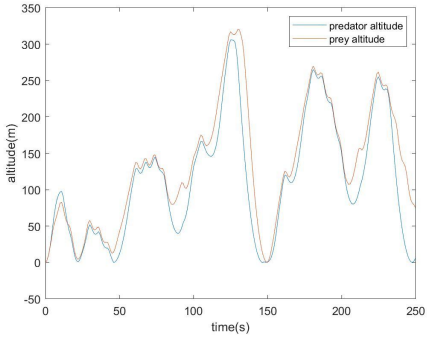
Graphs and Figures:

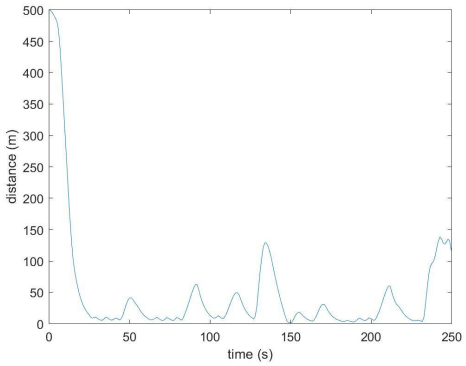


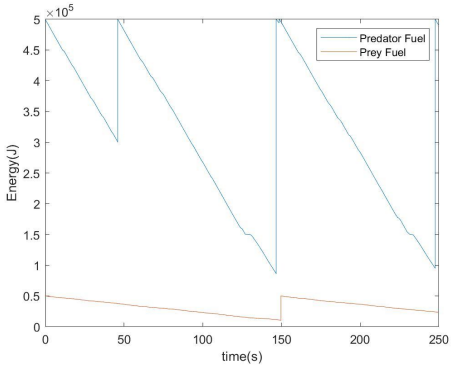


"Prey Escaped!"









Self Evaluation (Justin Moustouka):

My task in the main code portion of this project was to ensure that refueling could happen safely. Much of my time was spent deriving a method to do this, and fine-tuning this method in order to get the most efficient use of our fuel for both the predator and prey. I was also responsible for formatting/implementing this code (many if statements) into our predator and

prey sections for easy access to create and alter strategies within our code. I also assisted in the strategies for the predator and prey while at office hours with team members. Also, I had a large contribution to other aspects of the project like the presentation slides and this report. For this report, I was responsible for the abstract, introduction, and conclusion sections. I certainly believe I did my fair-share of work for this project.

Self Evaluation (Brian Delgado):

My role in this project was to assist in creating the strategies for the predator and prey, and then help my teammates in devising methods which we could use to implement these strategies into our Matlab code, which was especially difficult at some points, especially with refueling strategies. I attended office hours with my teammates and worked with them for multiple hours. I also contributed to this report, and to some of our ideas for improvements which could have been made to the code.

Self Evaluation (Caiden Puma):

My role in this project was to help with the brainstorming for the strategies for the predator and prey as well as to help troubleshoot our Matlab code. Additionally, I attended team meetings and office hours for many hours in order to facilitate the completion of our assignment. I was able to contribute to this report by checking the work of my teammates and offering suggestions on how to improve it and making edits where I saw fit to better communicate our project. Lastly, I was able to contribute ideas about how to implement our code in an efficient and effective manner in order to optimize our processes.

Self Evaluation (Dhaami Buford):

In this project, my primary role was to solve code for crash prevention and in developing the strategy for the prey particle. To help my group succeed, I attended office hours frequently during the week of the project to ensure that I was able to identify and resolve all errors in the MATLAB script and I spent several hours on my own testing the code of our prey against our predator while trying to improve the prey's ability to successfully avoid the predator. I maintained the group dynamic by keeping them informed of the changes I made to the code and sending updated versions of the script as I added to it. In addition, I also came up with our group name, "Bower Power," for which I believe I am to be commended for my exceptional creativity.