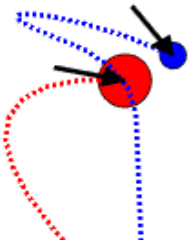
ENGN0040 Dynamics and Vibrations
Design Project: "Non-Trivial Pursuit"

Synopsis

You will devise algorithms to control an autonomous drone, with a view to both steering the drone to intercept a target object, as well as to escape a pursuer, and implement your algorithms as MATLAB code. Your programs will be tested in a competition against those designed by other teams.

1. Introduction

Pursuit problems are ubiquitous in engineering, mathematics, and computer science. The 'Homicidal Chauffeur Problem' is a famous example. For a more humane example from biology, you could watch this famous movie of a white blood-cell chasing a bacterium. In the classic pursuit problem, a fast, but poorly maneuverable hunter chases a slower, but highly maneuverable prey. The hunter and prey both seek optimal strategies to catch, or to escape, their competitor.



In this design project you will work on a computerized version of a pursuit problem that represents, approximately, a large drone in pursuit of a smaller one. Both predator and prey move only in the vertical plane. They experience (i) a force from their propulsion system (they can adjust the direction of this force freely, and can vary its magnitude within set limits); (ii) gravity (iii) an aerodynamic drag force; and (iv) a random fluctuating force representing wind gusts. The predator will choose the magnitude and direction of its propulsive force to try to catch the prey; while the prey will try to escape.

Both predator and prey have a finite reserve of energy, which is consumed at a rate that depends on the magnitude of their propulsive force. They can refuel by landing. A hard landing will cause them to crash, however. If the predator crashes, the prey wins the contest; if the prey crashes, the predator wins.

The goal of the design project is to write MATLAB scripts that determine (based on the positions and velocities of the two vehicles) the forces that must act on the predator and the prey to achieve their objectives. Your design team should write codes for both predator and prey. Your codes will then be tested against those of a competing group. The winning team will be chosen based on a combination of how long your prey can escape the competing predator, together with how quickly your predator can catch the prey designed by the competing team.

The main challenge involved in the design will be to use your physical intuition, and knowledge of particle dynamics, to devise a strategy for computing the direction (and magnitude) of the propulsive force acting on the predator and prey, given the position and velocity vectors of the two objects, and their energy reserves. The problem is completely open-ended, has no 'right' answer, and many possible strategies.

You will be able to think of pursuit, escape, and refueling strategies that are cleverer than anything we can think of, but if you are stuck, the video introduction to the project has some suggestions for strategies for the predator, prey, for avoiding the ground, and for refueling.

Don't try to create a random path by using the random number generator to determine your propulsive force – this makes the ODE solver crash)

2. Project Deliverables:

1. A written group report (not to exceed 10 pages in length, plus one page self-evaluation per team member), which describes your solution, and summarizes the process that led to your choice. A format for the report is suggested in the Appendix. Each group member should also provide a short description (not to exceed one page) of their role in the design process.
2. An oral presentation (including appropriate visual aids in the form of powerpoint slides, movies, or other creative media) to faculty, TAs and the competing team describing your design solution and its merits
3. A matlab script that will be tested in competition. **Matlab code must be submitted on Canvas by 6pm on Thurs July 1. Please ensure that your code passes the test described in Appendix 2 before you upload it.**

To be compatible with the code that is used to run the competition, your MATLAB function must have the following form (The function name and arguments must appear *exactly* as shown, and the file you submit should contain nothing other than this function)

```
function F = compute_f_groupname(t,Frmax,Fymax,amiapredator,pr,vr,Er,py,vy,Ey)
% Test time and place Enter the time and room for your test here
% Group members: list the names of your group members here
%  t:   Time
%  Frmax: Max force that can act on the predator
%  Fymax: Max force that can act on the prey
%  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 vector with current position of predator eg pr = [x_r;y_r]
%  vr - 2D vector with current velocity of predator eg vr= [vx_r;vy_r]
%  Er - Energy remaining for predator
%  py - 2D vector with current position of prey py = [x_prey;y_prey]
%  vy - 2D vector with current velocity of prey py = [vx_prey;vy_prey]
%      NB:pr,vr,py,vy are all COLUMN VECTORS
%  Ey - Energy remaining for prey
%  F - 2D 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)

      if (amiapredator)
          % Code to compute the force to be applied to the predator
      else
          % Code to compute the force to be applied to the prey
      end
end
```

The *groupname* in your function should be the name you choose for your group (please come up with a name that will be unique!)

3. Design Constraints:

- Predator and prey can only move in the *x*-*y* plane. The *y* coordinate is vertically upwards, and the ground is at *y=0*.
- At time *t=0* the predator and prey have position vectors $\mathbf{r}_r = 500\mathbf{i}$ $\mathbf{r}_y = \mathbf{0}$ m (i.e. they are both on the ground) and *zero initial velocity* (so you will need to be careful that your code will not bomb if the velocity is zero, eg because you divide by the speed in your code somewhere)
- Simulations will run for a time period *0<t<250* sec.
- The predator has mass $m_r = 100$ kg [1]
- The prey has mass $m_y = 10.0$ kg
- Both predator and prey will be subjected to four forces: (a) The propulsive force; (b) Gravity (with magnitude *mg* acting downwards; (c) an aerodynamic drag force; and (d) a random time-varying force.
  1. The propulsive forces will be determined by functions provided by the two competing groups. The maximum propulsive force on the predator is $F_{r\max} = 1.3 m_r g$ (g =9.81 m/s^2)
     The maximum propulsive force on the prey is $F_{y\max} = 1.4 m_y g$
     where m_r, m_y are the masses of the predator and prey, and *g* is the gravitational acceleration.
  2. The aerodynamic drag force will be computed as $\mathbf{F}_D = -cV\mathbf{v}$, where the viscous drag coefficient $c = 0.2$ Ns^2/m^2, **v** is the velocity vector of the object, and $V = \sqrt{v_x^2 + v_y^2}$ is the speed.
  3. Different random forces will act on the two objects. The two components of forces are independent pseudo-random numbers that vary between $\pm 0.2 m_r g$ N (on the predator) and $\pm 0.2 m_y g$ (on the prey). The forces will be computed using the script provided in the template posted online.
- Both predator and prey have a finite fuel reserve. At time *t=0*
  - The predator has a fuel reserve of 500 kJ (kiloJoules)
  - The prey has a fuel reserve of 50 kJ
- Predator and prey consume energy at a rate determined by their thrust.
  - The predator consumes energy at a rate $dE_r / dt = -0.1\left(\left|\mathbf{F}_r\right|\right)^{3/2}$ where $\left|\mathbf{F}_r\right| = \sqrt{F_{rx}^2 + F_{ry}^2}$ is the magnitude of the thrust applied by the predator.
  - The prey consumes energy at a rate $dE_y / dt = -0.2\left(\left|\mathbf{F}_y\right|\right)^{3/2}$ where $\left|\mathbf{F}_y\right| = \sqrt{F_{yx}^2 + F_{yy}^2}$ is the magnitude of the thrust applied by the predator.
- If predator or prey exhausts their fuel supply their thrust drops to zero.
- Predator and prey may both land, and will be fully refueled as soon as they do so (so their fuel reserves return to 500kJ and 50kJ for predator and prey, respectively). However, a hard landing will cause them to crash and burn.
  - The predator crashes if it lands with a speed exceeding 15 m/s
  - The prey crashes if it lands with a speed exceeding 8 m/s

  If the predator crashes, the prey wins the contest; if the prey crashes, the predator wins (and the contest is terminated)
- The predator is assumed to catch the prey if the distance between predator and prey drops below 1m.

[1] Since predator and prey both start with a p, we use the last letter in predator (r) and prey (y) as the subscripts distinguishing the two.

- Algorithms that cause either predator or prey to fluctuate at very high frequency will cause the ODE solver to stall. We will try to run these by applying a filter to the forces or by reducing the tolerance of the ODE solver, but if that fails we may have to ask you to modify your algorithm.

For the competition, we will provide the MATLAB code that will compute and animate the trajectories of the competitors, the code will also determine the winner of each contest and keeps score automatically. The test code will be working in SI units.

4. Scoring:

- Each team will play the role of 'predator' and 'prey' three times, with 'blind' code (i.e. without knowing what strategy the opposing team has chosen).
- Teams will then be given 5 minutes to modify their codes based on the results of the first round. The team that lost the first round may choose whether to modify their predator code or prey code. Both teams may then modify whichever the losing team selects, i.e. both teams must either modify the prey but leave predator code as-is, or both teams must modify predator and leave prey untouched. The six tests will be repeated with the modified code.
- The overall score for each group will be determined as follows

$$S = \sum_{i=1}^{6} T_s(i) - \sum_{i=1}^{6} T_C(i)$$

where $T_S(i)$ is the time that the prey survives during the *ith* test, and $T_C(i)$ is the time taken for the predator to catch the competing prey during the *ith* test. Negative scores will be rounded up to zero. If neither team is able to catch the prey from the opposition, the winning team will be the one whose predator is able to come closest to the prey from the opposing team.

- Solutions which attempt to win or force a draw by devious means (putting MATLAB into an infinite loop, hacking the MATLAB ODE solver, etc) will be disqualified at the discretion of the judges.

5. Calculations and computations

You should test and optimize your MATLAB functions as far as possible. Depending on the size of your group, one strategy might be to divide your group into teams that compete (internally) against one another, and then submit the winning designs to the final competition. Another would be to try some basic ideas for predator/prey forces – then if the predator wins, try to change the prey algorithm until it escapes – then go back and fix the predator to catch the improved prey, and so on. You will also need to work on strategies for your predator and prey to land and refuel. These are independent of the pursuit strategies, so you could assign some subset of your team members to work on refueling, while others work on the pursuit strategy.

To test your ideas, you will need to

1. Derive the equations of motion for the system consisting of the predator and prey together (if you are not sure how to approach this the appendix has some suggestions);
2. Write a MATLAB script that will compute (and, if you like, animate) the motion of the system. Since managing the events associated with landing/refueling is tricky, a lot of this script has been written for you. You should start by downloading a template for your code from the course website.
3. Test this MATLAB script with some very simple propulsive forces applied to the prey and predator. For example, you could simply apply a constant vertical force to both predator and prey,

and check to make sure they behave as expected. Then you could apply a constant vertical force to the prey, and use the approach described in HW3, 2012 to compute the propulsive force on the predator.

4. You will need to think of a way to stop your vehicles from hitting the ground too hard – one way to do this would be to add a vertical component to the propulsive force that gets bigger as the vehicle gets closer to the ground.
5. If they apply their maximum thrust, both predator and prey will need to land at least once to refuel during the 250s interval. You will need to (i) choose a strategy to decide when to land; and (ii) find a way of controlling the force to estimate the forces you need to apply to slow down your vehicle before it hits the ground (you may need to fine-tune the forces afterwards by trial and error, to compensate for the random force).
6. Try to think of other strategies to determine the propulsive forces. You can use conditional statements that check the time value and/or positions of predator and prey to change behavior. To test your ideas, run your MATLAB script using your candidate functions, and experiment with your designs to optimize them. Quite simple strategies often turn out to work quite well.

Warning: You will find that some strategies make the forces on either predator, or prey, or both, fluctuate very rapidly. This happens if you activate a large vertical force when you get within a certain distance of the ground, for example, and then turn it off again when you get above the critical altitude again. This would make the particle bounce up and down around the critical position with rapid fluctuations of vertical force. Rapid fluctuations in the propulsive forces cause the ODE solver to run very slowly because the time steps need to become very small (or possibly zero). The same problem arises if both predator and prey choose to calculate their forces based mostly on the velocities of their competitor – this can make the velocities of both change rapidly, and causes rapid force oscillations. If this happens it's usually best to switch to some other strategy that accomplishes the same thing – for example make the force increase linearly (but with a steep slope) below your desired altitude; or make the forces less dependent on the velocity of your competitors. To speed up testing, you can also reduce the accuracy of the ODE solver to make the code run faster, but low accuracy can sometimes give misleading predictions. Use the 'odeset' function to set the 'RelTol' parameter to a larger value (eg 0.01) to reduce the accuracy of ode113 (see the MATLAB manual for more details).

Appendix 1: Testing your function before submitting it

Before submitting your code:

1. Write your function, compute_f_*mygroupname*.m (where *mygroupname* is the name of your group)
2. Download the encrypted MATLAB file called function_tester.p from the website, and store the file in the same directory as your compute_f_*mygroupname*.m function. You will not be able to open or read the file, but MATLAB will run it.
3. To test your code, use MATLAB to navigate to the directory containing your function and the function_tester code, then type

```
function_tester('mygroupname')
```

   in the MATLAB command window.
4. If MATLAB produces any errors or takes an infinite time to run it will not work in competition and must be corrected or modified before it is submitted. A common error is to attempt to run the entire matlab script that you used to run test competitions instead of just the function

```
function F = compute_f_groupname(t,Frmax,Fymax,amiapredator,pr,vr,Er,py,vy,Ey)
```

   This function must be saved to a file by itself, and the file must be named compute_f_*groupname*. It must also be in the same directory as the function tester.p code.

Appendix 2: Deriving equations of motion and layout for a basic MATLAB script

Steps to deriving the equations of motion and writing MATLAB code

1. Write down the velocity and acceleration vectors for the white predator and prey in terms of time derivatives of their position vectors.
2. Draw free body diagrams for predator and prey, and write down the resultant force vector. The components of the propulsive forces can be specified in terms of variables, which will be computed by your function.
3. Use Newton's laws to write down the equations of motion for the predator and prey.
4. Rearrange the equations in a form that can be solved by MATLAB. Your equations should have the following general form

$$\frac{d}{dt}\begin{bmatrix} x_r \\ y_r \\ x_y \\ y_y \\ v_{xr} \\ v_{yr} \\ v_{xy} \\ v_{yy} \\ E_r \\ E_y \end{bmatrix} = \begin{bmatrix} v_{xr} \\ v_{yr} \\ v_{xy} \\ v_{yy} \\ F_{xr}^{total} / m_r \\ F_{yr}^{total} / m_r \\ F_{xy}^{total} / m_y \\ F_{yy}^{total} / m_y \\ -\left(F_{xr} v_{xr} + F_{yr} v_{yr}\right) \\ -\left(F_{xy} v_{xy} + F_{yy} v_{yy}\right) \end{bmatrix}$$

Where $(x_r, y_r), (x_y, y_y)$ are the positions of the predator and prey, $(v_{xr}, v_{yr}), (v_{xy}, v_{yy})$ are the velocities of predator and prey, $(F_{xr}, F_{yr}), (F_{xy}, F_{yy})$ are the propulsive forces applied on predator/prey, and $(F_{xr}^{total}, F_{yr}^{total}), (F_{xy}^{total}, F_{yy}^{total})$ are the resultant forces acting on the predator and prey (including gravity).

To get you started, a template for your MATLAB script is posted on the projects page of the website. To complete the code you will need to

(1) Add the section that computes the forces on the prey (the predator has been coded for you as an example)
(2) Write your compute_f_groupname function to control the forces on predator and prey.

DO NOT SUBMIT THIS SCRIPT – we only need the compute_f_groupname function.

Appendix 3: Report Instructions

Your report for this project could contain the following sections (this is just a suggestion – you can organize your report in any way you like, as long as it is clear and easy to follow):

1. *Abstract*: a few lines explaining the purpose of the report and summarizing the main conclusions
2. *Introduction*: Background information, motivation, and a brief description of the contents of the report.
3. *Pursuit strategy*: Summarize the formulas or procedures you decided to use to catch your opponent. Enough detail should be provided so that someone could duplicate your code.
4. *Escape strategy*: Summarize the formulas or procedures you decided to use to catch your opponent. Enough detail should be provided so that someone could duplicate your code.
5. *Design, and testing procedures*: Summarize candidate designs that were tested, how you made the decisions leading to your final choice, and how you tested the performance of your design.
6. *Conclusions:* summarize the results and recommendations briefly.
7. **Appendix:** list any supplemental information that is useful to the reader, but not critically important to understanding how your design works. The detailed MATLAB code could go in the Appendix, for example.

Appendix 4: Grading Rubric

1. Design Solution: 15 points.
    - Completing a basic code + 7 points
    - Success in at least one competition + 3 points
    - Unusually creative solution (this means anything you came up with on your own that is not described in the project description) + 5 points
2. Design Procedure 10 points
    - Design process did not yield a working code 3 points
    - Design process produced a working code, but only a few candidate solutions were tested, and no systematic procedure was used to test potential solutions 6 points
    - Team identified several candidate solutions, used a systematic process to test and rank them, and design procedure was documented. 10 points
3. Report 10 points
    - Basic report, adequately describes design procedure, but difficult to read, would not be comprehensible to a reader who did not read the project instructions first, did not follow required format, poor figures, etc 3 points
    - Report prepared according to instructions, generally well organized, but writing style is confusing, and minor errors (units missing off graphs, etc) 6 points
    - Publication quality report prepared according to instructions – typed, professionally formatted, perfect grammar/writing style, easily readable 10 points.
4. Oral Presentation 10 points
    - No presentation aids, presentation not planned 6 points
    - Presentation planned, all group members contributed, appropriate and well organized visual aids/demonstrations 8 points
    - Presentation planned, all group members contributed, appropriate and well organized visual aids/demonstrations, and unusually creative way of communicating content 10 points
5. Peer Evaluation 5 points