Due 12/13/2024
ENGR-13A

Qiuyang Wang

Instructor: Jonathan Krones & Michael Norton

ENGR-13A: Modeling and Simulation

December 11, 2024

Motion Camouflage

Introduction

When I was in high school, my math teacher assigned me with an online academic article describing an intriguing phenomenon called motion camouflage (Dartnell). Unfortunately, I did not fully understand its concept at that time. Since then, the idea has lingered in my mind and sparked my curiosity. As a result, I decided to explore this concept further as my final project for this course. When we mention the term "camouflage," the first image that often comes to mind is an animal blending into its environment by changing its skin color, texture, or patterns to avoid detection. This traditional concept of camouflage emphasizes visual concealment through stillness and blending. However, there exists a fascinating and less commonly discussed type of camouflage called motion camouflage (Srinivasan and Davey). Unlike regular camouflage, which involves changing skin color or texture, motion camouflage refers to a phenomenon where a predator moves toward its prey while appearing stationary to the prey. The only noticeable change to the prey is the inevitable increase in the perceived size of the predator as it moves closer. This phenomenon exploits the prey's visual perception to mask the predator's movement, creating an illusion of optical immobility.

Many researchers have studied and made progress in modeling motion camouflage in the past 20 years. In 2004, Paul Glendinning laid the mathematical groundwork to model this concept (Glendinning). He also proves that motion camouflage can catch the prey faster than the classic pursuit. Later, some researchers argue that his work is not biologically plausible without explicitly providing feedback laws to produce the motion camouflage trajectory. Therefore, a biologically plausible feedback laws model was proposed in 2006 to analyze and interpret motion camouflage systematically (Justh and Krishnaprasad; Reddy et al.). The study subject also extends broadly from dragonflies, falcons, and other insects to octopus arm movements. In Glendinning's work, one particular step caught my attention and inspired me to consider it as a potential project idea. He mentions an essential first-order nonlinear differential equation that does not have a known solution. To address this issue, he transforms the original equation into a more manageable one. With the development of software for math and science, I believe MATLAB can solve this issue using a numerical method. Thus, in this project, I aim to use this software to implement the mathematical model of motion camouflage described in Glendinning's paper without simplifying the original differential equation. In the meantime, I can also use MATLAB to create some 2D and 3D visualizations to show the trajectory of the predator.

Methods and Results

The essence of motion camouflage is to create an illusion of minimal relative movement from the prey's perspective. Instead of directly pursuing the prey, the predator aligns its trajectory to blend into the background or a specific point in the observer's field of vision. This relies on two fundamental principles. One is the predator always lies on the line segment between a fixed reference point and the position of the prey, which is the constraint line. The other is that the angle between the predator and the prey should be carefully adjusted to change at a specific rate, simulating the motion of an object at a fixed distance. The parameters used by the model are listed below:

$z(t)$: the position of the prey;

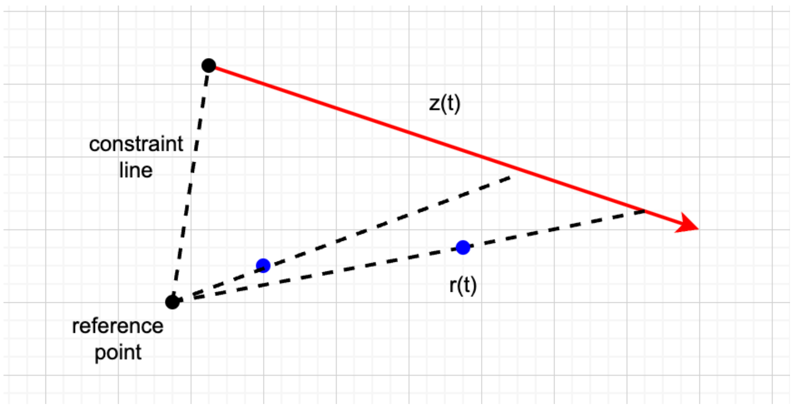
$r(t)$: the position of the predator;

$u(t)$: the ratio that the predator moves towards the prey on the constraint line;

r_0: the fixed reference point;

c: the speed of the predator;

v: the speed of the prey.



The predator's trajectory can be expressed using vector geometry as $r(t) = r_0 + u(t) * (z(t) - r_0)$. The core implementing this model is to find $u(t)$ with the initial condition $u(0) = 0$. Glendinning successfully derives the differential equation for $u(t)$ in his work:

$$\dot{u} = \frac{-\left[(z(t) - r_0)\cdot\dot{z}\right]u + \sqrt{\left(\left[(z(t) - r_0)\cdot\dot{z}\right]^2 u^2 - (v^2u^2 - c^2)\,|z(t) - r_0|^2\right)}}{|z(t) - r_0|^2}$$

However, this is when he cannot use this equation further because it has yet to have a known solution. Therefore, MATLAB can take over the work after that. The equation above contains three vectors. To obtain a more comprehensive equation that can be used in MATLAB, we need to split those vectors in both X and Y directions:

$$z(t) = \begin{Bmatrix} z(t)_x \\ z(t)_y \end{Bmatrix} \quad r(t) = \begin{Bmatrix} r(t)_x \\ r(t)_y \end{Bmatrix} \quad r_0 = \begin{Bmatrix} r_0x \\ r_0y \end{Bmatrix}$$

Then, substitute the following two parts that contain those vectors from the original equation:

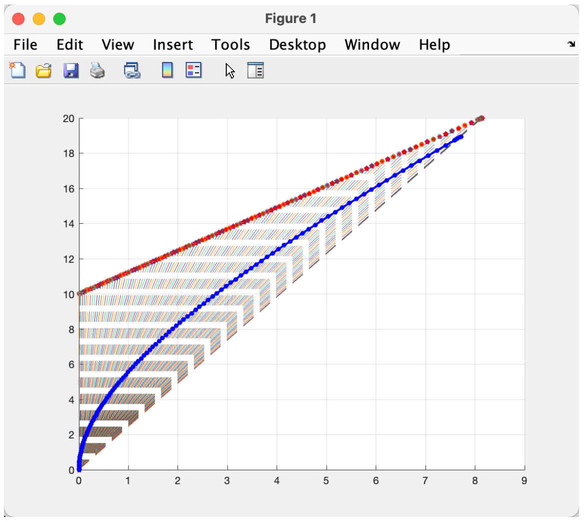
$$|z(t) - r_0|^2 = (z(t)_x - r_{0_x})^2 + \left(z(t)_y - r_{0_y}\right)^2$$

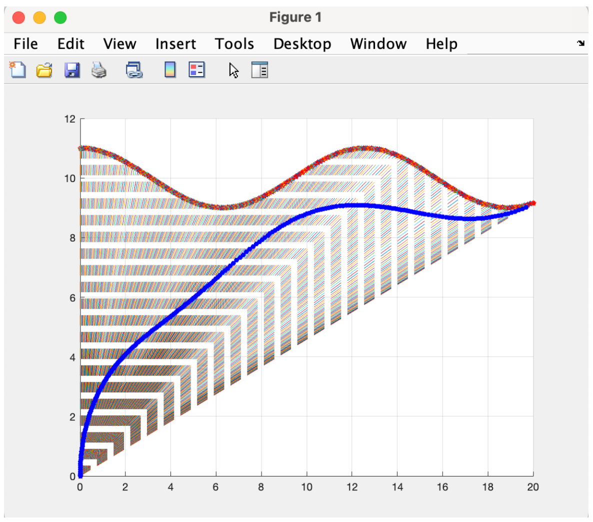
$$(z(t) - r_0)\cdot\dot{z} = (z(t)_x - r_{0_x})\,\dot{z}_x + \left(z(t)_y - r_{0_y}\right)\dot{z}_y$$

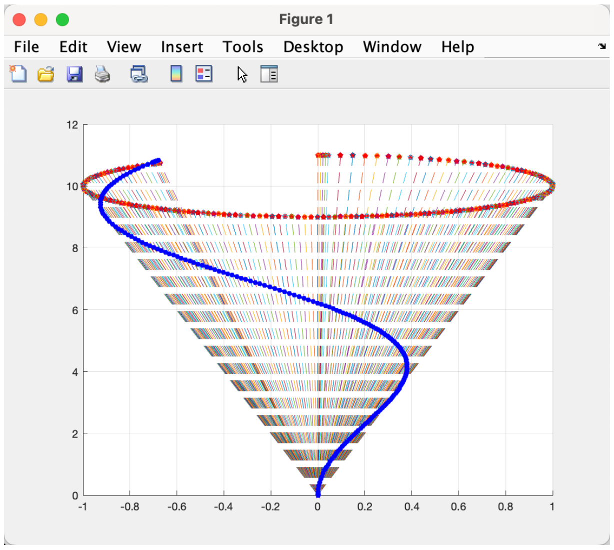
Plug them back to the original differential equation. The updated equation becomes:

$$\dot{u} = \frac{-\left[(z(t)_x - r_{0x})\dot{z}_x + (z(t)_y - r_{0y})\dot{z}_y\right]u + \sqrt{\left(\left[(z(t)_x - r_{0x})\dot{z}_x + (z(t)_y - r_{0y})\dot{z}_y\right]^2 u^2 - (u^2v^2 - c^2)\left[(z(t)_x - r_{0x})^2 + (z(t)_y - r_{0y})^2\right]\right)}}{(z(t)_x - r_{0x})^2 + (z(t)_y - r_{0y})^2}$$

With the help of applying ode45 in MATLAB, some examples of the trajectory of motion camouflage are listed below:



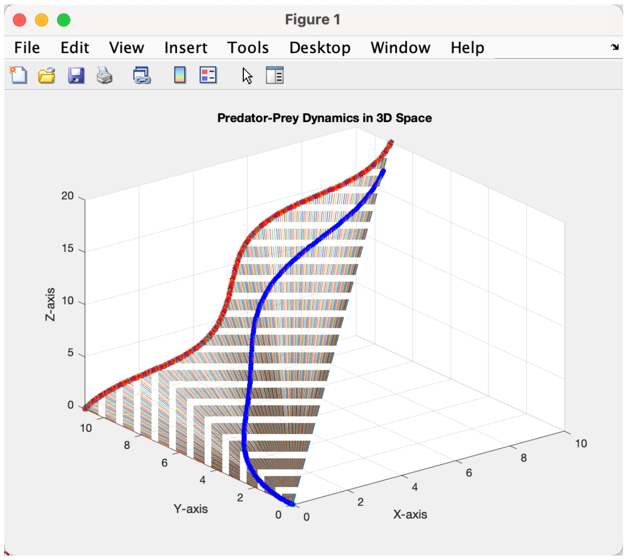


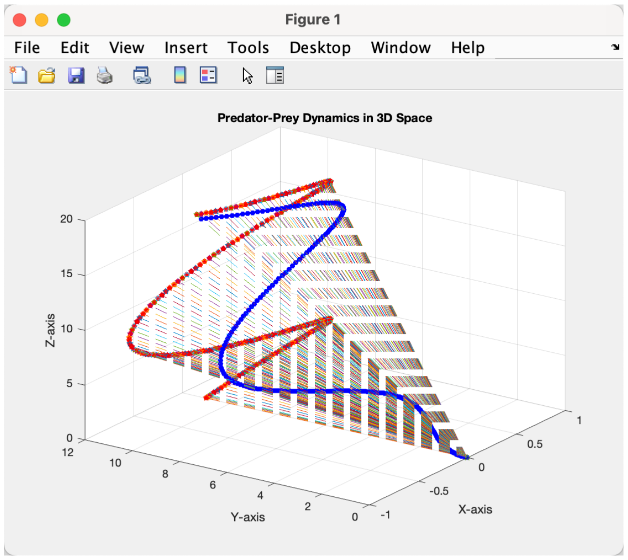


Although these graphs clearly represent the motion camouflage trajectory, they are still in 2D space. Here, if we can add one more dimension, Z, to the differential equation we used in MATLAB, the ultimate differential equation becomes:

$$\dot{u} = \frac{-[(z(t)_x - r_{0x})\dot{z}_x + (z(t)_y - r_{0y})\dot{z}_y + (z(t)_z - r_{0z})\dot{z}_z]u + \sqrt{\left([(z(t)_x - r_{0x})\dot{z}_x + (z(t)_y - r_{0y})\dot{z}_y + (z(t)_z - r_{0z})\dot{z}_z]^2 u^2 - (u^2v^2 - c^2)\left[(z(t)_x - r_{0x})^2 + (z(t)_y - r_{0y})^2 + (z(t)_z - r_{0z})^2\right]\right)}}{(z(t)_x - r_{0x})^2 + (z(t)_y - r_{0y})^2 + (z(t)_z - r_{0z})^2}$$

By following the same workflow in MATLAB, now the graph can visualize motion camouflage in 3D space:





Limitations and Future Work

While the mathematical model provides an elegant framework, several limitations hinder its direct application. One of the primary issues is this model is not biologically plausible. In the current approach, the movements of the prey are predefined with arbitrary functions, allowing derivatives to be taken to calculate future movements. However, in real scenarios, predators cannot predict the prey's future movements with such precision. Instead, they respond dynamically to the prey's actions over time and adapt their strategies based on observations. Another limitation lies in the assumption of using constant speed for predators and preys. This constraint simplifies the modeling process and aligns with Glendinning's work on ideal motion-camouflage paths, which requires constant speed to create a mathematically unique trajectory. However, this idealization does not account for natural variations, where acceleration and deceleration play significant roles in predator-prey interactions. Lastly, while plans were made to implement this model in Unity to simulate motion camouflage and observe the behavior from the prey's perspective, these efforts were postponed due to time constraints. The Unity implementation would have included attaching a virtual camera to the prey object. This can create a first-person view of how the predator's movement appears to the prey during motion camouflage.

Future work is necessary to address these limitations. The model can be improved by incorporating real-time feedback systems, which would also enable the simulation of dynamic speed situations. In the meantime, completing the implementation in Unity remains a priority for future work. By addressing these limitations and pursuing these future directions, the motion camouflage model can evolve into a more comprehensive one.

Works Cited

Dartnell, Lewis. *They Never Saw It Coming*.

Glendinning, Paul. “The Mathematics of Motion Camouflage.” *Proceedings: Biological Sciences*, vol. 271, no. 1538, 2004, pp. 477–81.

Justh, E. W., and P. S. Krishnaprasad. “Steering Laws for Motion Camouflage.” *Proceedings: Mathematical, Physical and Engineering Sciences*, vol. 462, no. 2076, 2006, pp. 3629–43.

Reddy, P. V., et al. *Motion Camouflage in Three Dimensions*. arXiv:math/0603176, arXiv, 8 Mar. 2006. *arXiv.org*, https://doi.org/10.48550/arXiv.math/0603176.

Srinivasan, Mandyam V., and Matthew Davey. “Strategies for Active Camouflage of Motion.” *Proceedings: Biological Sciences*, vol. 259, no. 1354, 1995, pp. 19–25.

Appendix

AI tools

I used ChatGPT to help me rephrase my sentences while writing this report. All intellectual and critical thinking work are done by myself.

motion_camouflage.m (2D)

```
close all
clearvars

% Define time span
ti = 0;
tf = 10;

% Define Z_x(t) and Z_y(t)
a = rand();
b = rand();
syms t Z_x(t) Z_y(t)
Z_x(t) = sin(t);     % X-component of Z(t)
Z_dot_x = diff(Z_x, t);
Z_y(t) = cos(t) + 10;     % Y-component of Z(t)
Z_dot_y = diff(Z_y, t);

% parameters
u0 = 0;              % Initial condition for u(t)
r0_x = 0;            % Reference position x-coordinate
r0_y = 0;            % Reference position y-coordinate
c = 2;               % Speed of the predator
v = 1;               % Speed of the target
r_initial = [0, 0]; % Initial position of the predator


%%

% Solve the differential equation using ode45
options = odeset('Events', @(t, u) convergence_event(t, u, Z_x, Z_y, r0_x,
r0_y),'RelTol',1e-10,'AbsTol',1e-10);
[t, u] = ode45(@(t, u) cal_ut(t, u, r0_x, r0_y, c, v, Z_x, Z_y, Z_dot_x,
Z_dot_y), [ti, tf], u0, options);

% Compute Z(t)
Z_x_vals = Z_x(t);
Z_y_vals = Z_y(t);

% Compute r(t) using u(t)
```

```
r_x = r_initial(1) + u .* (Z_x_vals - r0_x);
r_y = r_initial(2) + u .* (Z_y_vals - r0_y);

% Plot Z(t)
plot(Z_x_vals, Z_y_vals, 'r-', 'LineWidth', 2);
scatter(Z_x_vals, Z_y_vals, 25, 'r', 'filled');
hold on;

% Plot constraint lines
for i = 1:length(Z_x_vals)
    plot([0, Z_x_vals(i)], [0, Z_y_vals(i)], 'p--'); % Dashed line to [0, 0]
end

% Plot r(t)
plot(r_x, r_y, 'b-', 'LineWidth', 2);
scatter(r_x, r_y, 25, 'b', 'filled');
grid on;
```

cal_ut.m

```
function u_dot = cal_ut(t, u, r0_x, r0_y, c, v, Z_x, Z_y, Z_dot_x, Z_dot_y)
    % Evaluate Z_x, Z_y, Z_dot_x, Z_dot_y at time t
    Z_x_t = Z_x(t);
    Z_y_t = Z_y(t);
    Z_dot_x_t = Z_dot_x(t);
    Z_dot_y_t = Z_dot_y(t);

    % Calculate |z(t) - r0|^2
    a = (Z_x_t - r0_x).^2 + (Z_y_t - r0_y).^2;

    % Calculate (z(t) - r0) . z_dot
    b = (Z_x_t - r0_x) .* Z_dot_x_t + (Z_y_t - r0_y) .* Z_dot_y_t;

    % Define the numerator and denominator for the equation
    numerator = -b * u + sqrt((b.^2) * (u^2) - ((u^2) * (v.^2) - c^2) .* a);
    denominator = a;

    % Define the equation for u_dot
    u_dot = double(numerator / denominator);
return
```

convergence_event.m

```
function [value, isterminal, direction] = convergence_event(t, u, Z_x, Z_y,
r0_x, r0_y)
    % Compute Z(t) at the current time
    try
        Z_x_t = double(Z_x(t)); % Ensure Z_x is evaluated numerically
        Z_y_t = double(Z_y(t)); % Ensure Z_y is evaluated numerically
    catch
        error('Error evaluating Z_x or Z_y at time t = %.2f.', t);
    end

    % Compute r(t) at the current time
    r_x_t = r0_x + u * (Z_x_t - r0_x);
    r_y_t = r0_y + u * (Z_y_t - r0_y);

    % Compute the distance between r(t) and Z(t)
    distance = sqrt((r_x_t - Z_x_t)^2 + (r_y_t - Z_y_t)^2);

    threshold = 0.1;

    % Event condition
    value = distance - threshold; % Stop when distance < threshold
    % disp(['t = ', num2str(t), ', value = ', num2str(value)]);
    isterminal = 1; % Stop the integration
    direction = 1;  % Detect all zero crossings
end
```

test_3D.m (3D visualizations)

```
close all
clearvars

% Define time span
ti = 0;
tf = 10;

% Define Z_x(t), Z_y(t), and Z_z(t)
syms t Z_x(t) Z_y(t) Z_z(t)
Z_x(t) = sin(t);            % X-component of Z(t)
Z_y(t) = cos(t) + 10;       % Y-component of Z(t)
Z_z(t) = 2 * t;             % Z-component of Z(t)

Z_dot_x = diff(Z_x, t);
Z_dot_y = diff(Z_y, t);
Z_dot_z = diff(Z_z, t);

% parameters
u0 = 0;             % Initial condition for u(t)
r0_x = 0;           % Reference position x-coordinate
r0_y = 0;           % Reference position y-coordinate
r0_z = 0;           % Reference position z-coordinate
```

```
c = 2;              % Speed of the predator
v = 1;              % Speed of the target
r_initial = [0, 0, 0]; % Initial position of the predator

% Solve the differential equation using ode45
options = odeset('Events', @(t, u) convergence_event_3d(t, u, Z_x, Z_y, Z_z, 
r0_x, r0_y, r0_z), ...
                 'RelTol', 1e-10, 'AbsTol', 1e-10);
[t, u] = ode45(@(t, u) cal_ut_3d(t, u, r0_x, r0_y, r0_z, c, v, Z_x, Z_y, Z_z, 
Z_dot_x, Z_dot_y, Z_dot_z), ...
               [ti, tf], u0, options);

% Compute Z(t)
Z_x_vals = Z_x(t);  % Z_x evaluated at all time points
Z_y_vals = Z_y(t);
Z_z_vals = Z_z(t);

% Compute r(t) using u(t)
r_x = r_initial(1) + u .* (Z_x_vals - r0_x);
r_y = r_initial(2) + u .* (Z_y_vals - r0_y);
r_z = r_initial(3) + u .* (Z_z_vals - r0_z);

% Plot Z(t) in 3D
plot3(Z_x_vals, Z_y_vals, Z_z_vals, 'r-', 'LineWidth', 2);
scatter3(Z_x_vals, Z_y_vals, Z_z_vals, 25, 'r', 'filled');
hold on;

% Plot constraint lines
for i = 1:length(Z_x_vals)
    plot3([0, Z_x_vals(i)], [0, Z_y_vals(i)], [0, Z_z_vals(i)], 'p--'); % 
Dashed line to [0, 0, 0]
end

% Plot r(t) in 3D
plot3(r_x, r_y, r_z, 'b-', 'LineWidth', 2);
scatter3(r_x, r_y, r_z, 25, 'b', 'filled');
grid on;
xlabel('X-axis');
ylabel('Y-axis');
zlabel('Z-axis');
title('Predator-Prey Dynamics in 3D Space');
hold off;

% Helper functions for 3D space, similar to 2D version
function u_dot = cal_ut_3d(t, u, r0_x, r0_y, r0_z, c, v, Z_x, Z_y, Z_z, 
Z_dot_x, Z_dot_y, Z_dot_z)
    % Evaluate Z_x, Z_y, Z_z, Z_dot_x, Z_dot_y, Z_dot_z at time t
    Z_x_t = Z_x(t);
    Z_y_t = Z_y(t);
    Z_z_t = Z_z(t);
    Z_dot_x_t = Z_dot_x(t);
    Z_dot_y_t = Z_dot_y(t);
    Z_dot_z_t = Z_dot_z(t);

    % Calculate |z(t) - r0|^2
    a = (Z_x_t - r0_x)^2 + (Z_y_t - r0_y)^2 + (Z_z_t - r0_z)^2;

    % Calculate (z(t) - r0) . z_dot
    b = (Z_x_t - r0_x) * Z_dot_x_t + (Z_y_t - r0_y) * Z_dot_y_t + (Z_z_t - 
r0_z) * Z_dot_z_t;

    % Define the numerator and denominator for the equation
    numerator = -b * u + sqrt((b^2) * (u^2) - ((u^2) * (v^2) - c^2) * a);
    denominator = a;

    % Define the equation for u_dot
    u_dot = double(numerator / denominator);
end


function [value, isterminal, direction] = convergence_event_3d(t, u, Z_x, 
Z_y, Z_z, r0_x, r0_y, r0_z)

    Zx = double(subs(Z_x, t));
    Zy = double(subs(Z_y, t));
    Zz = double(subs(Z_z, t));
    % Compute r(t) using u(t)
    r_x = r0_x + u .* (Zx - r0_x);
    r_y = r0_y + u .* (Zy - r0_y);
    r_z = r0_z + u .* (Zz - r0_z);

    distance = sqrt((Zx - r_x)^2 + (Zy - r_y)^2 + (Zz - r_z)^2);
    value = distance - 0.1; % Stop when distance is less than a small 
threshold
    isterminal = 1;          % Stop the integration
    direction = 0;           % Detect all zero crossings
end
```