MEG Documentation

Main.m

Environment.m

Here we define a class called Environment.

```
properties
    motion_space_dimension % dimension of motion space
    evader_numbers % number of evaders in the game
    pursuer_speed % pursuer speed
    evader_speeds % array of evader speeds
    target_position % position of the target
    timestep % timestep value
    captured_evaders % boolean array containing capture status of evaders
    pursuer % pursuer object
    evaders % evader object array
    capture_tolerance % tolerance for point capture
    alpha % speed ratio array
end
```

- These are the properties it has notice there is only one pursuer
- It has a constructor function. There we assign all values to the env object.
- Classes for pursuer and evader objects also exist. They require the position and speed of the objects. env.pursuer has the pursuer object, whereas env.evaders is an array that has all the evaders.
- There is also the target_position variable(what does it do?)
- alpha is defined which is the ratio of evader_speed/pursuer_speed.

Now we shall inspect the functions

0.1 update_target(env, target_position)

- It simply updates the present target position.

0.2 reset

- This function seems to be passing the wrong parameter to the updatePos function. It is passing an object of the evader class, instead of simply passing the position

0.3 Update_Environment

- Updates positions of pursuer, evaders and even the target position, given the input positions

0.4 barrier_value

- We define it as the difference of distance squared between target and evader, and the $alpha^2$ product with distance squared between target and pursuer.

0.5 win or check_initialization

- Depending on the barrier we determine if the pursuer or the evaders win.
- If evaders are winning, then we will simply reset environment. If pursuers are winning then we can proceed.

0.6 evader_names

- this is a function that returns all the evader names, in an array.

0.7 plot_current_positions

This is used to plot the positions of evaders and pursuers on the screen.

0.8 return_evader_positions

This is a function that returns an array that has positions Similarly there exists a function that returns the velocities. now the velocities are obviously vectors, so they are represented as a column vector. This is found by taking difference of the pursuer and target positions, then using alpha(pursuer velocity is taken 1)

0.9 updateTermination

Here we will set captured status of the ith evader to true, when the distance between them is lesser than a treshold.

0.10 step

If all the evaders have been captured then it makes done=true.

0.11 obtain_trajectories

- The outputs are win, pursuerpositionstraj(over time), evaderpositionstraj(over time). Inputs are the environment object and the objective_function
- Initially plot the positions of all the players and target. Then hold the plot.
- while done is false, update the pursuerpositiontraj(A 2D matrix. One dimension is the posn the other dimension is time.) and evaderpositiontraj(A 3D matrix. Time, position, and evaderIndex i are the three dimensions. Each time t is a multiple of 10, then plot the trajectories.
- plot_trajectories to plot pursuer and evader trajectories.

Pursuer.m

There is a new class called Pursuer

```
classdef Pursuer < handle
    properties (SetAccess = public)
        position % 2x1 vector representing current position
        speed % scalar representing maximum speed
        motor1 % motor variables
        motor2
        motor3
        pos_vrpn
        ori_vrpn
        wheel_radius
        wheel_centre_radius
    end
```

Constructor Pursuer

Simply assigns the initial values

updatePos

Simply updates the position of the pursuer given input position

getPos

Outputs the position of the pursuer

optimal_headings_Ei

- Takes as input [p, evader_position, target_position] and outputs [psi_star, theta_star, m, xm, ym]
- m is slope of arrow from the pursuer to the evader. xm,ym refer to the midpoint of the line joining the evader and the pursuer.
- xt and yt are the target positions
- $xI = (m * (y_m - y_t) + x_m + m^2 * x_t)/(1 + m^2)$
  $yI = (m * (x_m - x_t) + y_t + m^2 * y_m)/(1 + m^2)$
- Geometrically, (xI, yI) refers to the foot of the perpendicular dropped from the target position to the line joining pursuer and evader.
- **psi_star** - This is the heading angle for the evader towards the intersection point to avoid the pursuer.
  theta_star - This is the heading angle for the pursuer towards the intersection point, to optimally intercept the evader.

objective_Ei

- objective_Ei takes input p, evader_position, target_position, r, theta and output is d_i.
- the scalar cost value d_i is used for evader Ei to evaluate how good a given angle theta(angle of the pursuer is).
- First use the function optimal_headings.
- perpendicular distances Tx and Ty from target and evader to the line joining evader and pursuer. Now this Ty should be 0 right ideally?
- k is half of the distance between the pursuer and evader.
- delta is the angle made by line joining evader and pursuer with the x-axis.
- a cost function $d_i = T_x + (r/(2 * k)) * sqrt(T_y^2 + k^2) * (-1 - cos(theta + psi_{star} - 2 * delta))$

concave_domain

- inputs are p, evader_positions, target_position. Outputs are theta_min, theta_max, min_evader, max_evader.
- Left due to difficulty in understanding

Evader.m

This is the Evader class

```
classdef Evader < handle
    properties
        position % 2x1 vector representing current position
        speed % scalar representing maximum speed
        index % integer indexing of each evader
        name % evader name string
        motor1 % motor variables
        motor2
        motor3
        pos_vrpn
        ori_vrpn
        wheel_radius
        wheel_centre_radius
    end
```

- The constructor initializes these values like name, index, speed, position, wheel_radius and wheel_centre_radius in jacobian.)
- updatePos similar to all other updatePos.
- getPos just gets the Position

heading_velocity

- inputs are pursuer_position, target_position, win, alpha. The outputs are velocity, psi.

alpha=1, win=true

- xc, yc refers to the midpt of the line joining pursuer and evader.
- m is the slope of line joining pursuer to evader.
- x_intercept, y_intercept refers to the perpendicular drawn from the target onto the line joining evader and pursuer.
- it sets the velocity.

alpha $<$ 1, win=true

- xc,yc,rc define the Apollonius circle for the given evader and pursuer positions. This separates the regions where the evader can and cannot escape.
- Rc is the distance from the center to the target.
- $x_{intercept}, y_{intercept}$,refer to the intercepts on the Apollonius. In essence, they are the perpendiculars dropped from the target to the circle, which obviously has to pass through the center.
- now again use these intercepts and calculate the heading velocity of the evader

else

just assign some random velocity

Now finally we will normalize velocity and return the heading angle of the velocity.

Later there are some functions to set the motor speeds, start motor and stop motor.

Learning

Topics Involved

- Differential Game Theory
- **Barrier Functions** - Present in the barrier method of env class, determines if the pursuer has any chance of winning.
- **Geometric Control Theory** - Used in heading velocity of evader that perpendicular bisector and appolonius circle.
- **Optimal Control** Each agent optimizes its own trajectory. As in direction optimization of individual agents.

- **Multi Agent systems** Coordination between **multiple evaders** and **one pursuer.**
- **Heuristic Algorithms** - Used in the heading_direction_heuristic() implements weighted average of optimal headings.