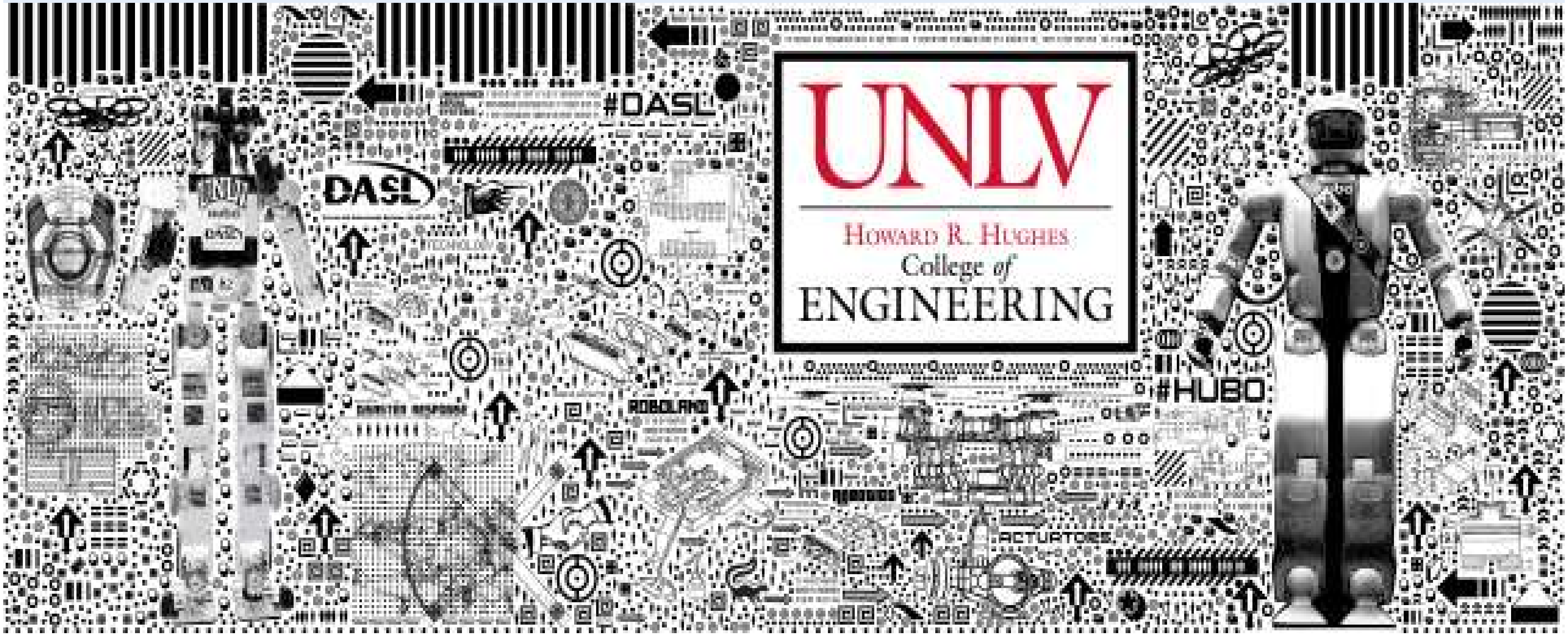
UNIVERSITY OF NEVADA LAS VEGAS

DRONES AND AUTONOMOUS SYSTEMS LAB (DASL@UNLV)



Navigation and Path-planning [DASL-102]

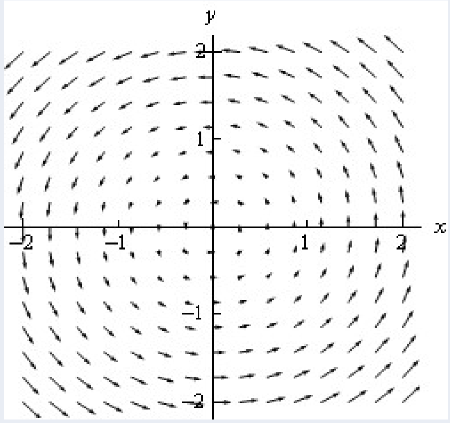
WEEK 1 – Potential Fields

INSTRUCTOR: Dylan Wallace

- Vector Fields
- Potential Functions
  - Gradient functions
- Navigation
  - Updating robot position
  - Non-holonomic constraints
- Local Minima
- Algorithm
  - Tuning constants
- MATLAB Code
- Homework
- References

Vector Fields

- We will only deal with 2-D vectors in the xy-plane
- The gradient vector is given by taking the partial derivatives of the potential function, with respect to x & y.
  - Ex: **f**(x, y) = < x^2 + y^2, x + y >
  - ∇f(x, y) = < 2x, 1 >
- This gradient will create a vector at every point of the space
  - This is called a gradient vector field



Potential Functions

- We view the obstacles as repulsive forces, and the target as an attractive force
- Attractive:

$$U_a = \frac{\eta_a}{2}\rho_t^2(q)$$

$$\rho_t = \sqrt{(x_t - x)^2 + (y_t - y)^2}$$

η_a : attractive potential constant
(x_t, y_t) : target location
ρ_t : distance to target

  - Gradient:

$$\nabla_q U_a = \begin{bmatrix} \eta_a (x - x_t) \\ \eta_a (y - y_t) \end{bmatrix}$$

- Repulsive:

$$U_r(q) = \begin{cases} \frac{1}{2}\eta_r \left(\frac{1}{\rho} - \frac{1}{\rho_o}\right)^2 & \rho \le \rho_o \\ 0 & \rho > \rho_o \end{cases}$$

η_r : repulsive potential constant
(x_o, y_o) : obstacle location
ρ : current robot-to-obstacle distance
ρ_o : repulsive force maximum range

  - Gradient:

$$\nabla_q U_r = \begin{cases} \frac{\eta_r (x_o - x)}{\rho^3}\left(\frac{1}{\rho} - \frac{1}{\rho_o}\right) \\ \frac{\eta_r (y_o - y)}{\rho^3}\left(\frac{1}{\rho} - \frac{1}{\rho_o}\right) \end{cases} \quad \text{when} \quad \rho \le \rho_o$$

$$\nabla_q U_r = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \text{when} \quad \rho > \rho_o$$

Navigation

- We can update the robot position by using the previous position, a sampling rate, and the gradient vector:

$$q[i] = q[i-1] - \Delta T \nabla_q (U_a + U_r)$$

$$x[i] = \begin{cases} x[i-1] - \Delta T \eta_a (x[i-1] - x_t) - \dfrac{\Delta T \eta_r (x_o - x[i-1])}{\rho^3} \left\{ \dfrac{1}{\rho} - \dfrac{1}{\rho_o} \right\} & \text{when } \rho \le \rho_o \\ x[i-1] - \Delta T \eta_a (x[i-1] - x_t) & \text{when } \rho > \rho_o \end{cases}$$

$$y[i] = \begin{cases} y[i-1] - \Delta T \eta_a (y[i-1] - y_t) - \dfrac{\Delta T \eta_r (y_o - y[i-1])}{\rho^3} \left\{ \dfrac{1}{\rho} - \dfrac{1}{\rho_o} \right\} & \text{when } \rho \le \rho_o \\ y[i-1] - \Delta T \eta_a (y[i-1] - y_t) & \text{when } \rho > \rho_o \end{cases}$$

$$\rho = \sqrt{(y_o - y[i-1])^2 + (x_o - x[i-1])^2}$$

Non-Holonomic Constraints

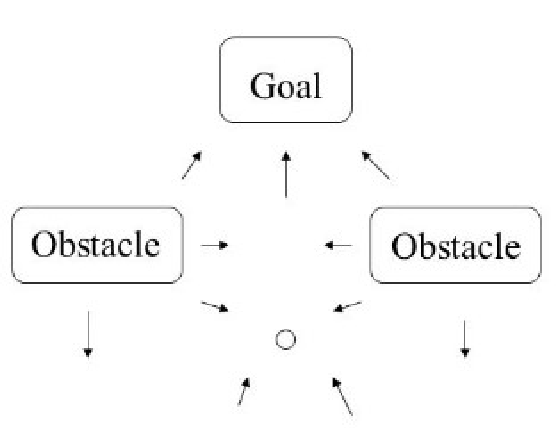
- For robots that cannot move omnidirectionally, we need to have special equations to move the robot to the desired position
- Different robot configurations will have different equations
- Two-wheeled system (no steering):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ \dot{\theta} \end{bmatrix}$$

$$\begin{bmatrix} V \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \frac{R}{2} & \frac{R}{2} \\ \frac{R}{2b} & \frac{-R}{2b} \end{bmatrix} \begin{bmatrix} \omega_r \\ \omega_l \end{bmatrix}$$

Local Minima

- The way the potential functions are created, a local minimum is created at the target, and the robot navigates toward a local minima
- Other minima can be created if the configuration space has obstacles that are placed in a way that creates a local minima
  - i.e. Repulsive forces cancel each other
- This will cause our robot to be stuck, stopping the navigation



Algorithm

- We can combine all of this math into a useable algorithm for our robot to follow.

I. Define attractive, repulsive, and minimum constants

II. Loop over a number of iterations, which define the resolution

1. Calculate distance to target and obstacles
2. If the current distance is less than the maximum
   i. Calculate repulsive gradient
   ii. If not, repulsive gradient is zero
3. Update robot position

Sunday, May 14, 2017, 13:40

MATLAB Code

```
for i = 2:iterations % Run algorithm for 100 sample periods
    t(i-1) = T*(i-1); % Time in seconds

    % Distance to target
    rhoTarget(i-1) = sqrt(((yTarget-y(i-1))^2) + ((xTarget-x(i-1))^2));

    % Distance to an obstacle
    rho(i-1) = sqrt( ((yObstacle-y(i-1))^2) + ((xObstacle-x(i-1))^2) );

    % Calculate gradient
    if rho(i-1) < rho0
        ur_x = nRepulse*(xObstacle-x(i-1))*((1/rho(i-1))-1/rho0))/(rho(i-1)^3);
        ur_y = nRepulse*(yObstacle-y(i-1))*((1/rho(i-1))-(1/rho0))/(rho(i-1)^3);
    else
        ur_x = 0;
        ur_y = 0;
    end

    % Calculate new robot position
    x(i) = x(i-1) - T*nAttract*(x(i-1)-xTarget) - T*ur_x;
    y(i) = y(i-1) - T*nAttract*(y(i-1)-yTarget) - T*ur_y;
end
```

Homework

- Use the MATLAB code to create a situation with 3 obstacles. Tune your constants accordingly. The space: 10 x 10
  - Obstacles: (3, 1), (4, 5), & (7, 9)
  - Target: (10, 10)
- Make sure to plot your results with every iteration
- You will need to tune your attractive, repulsive, and maximum distance constants to get the best results
- If you do not have access to MATLAB, you can use GNU Octave as a free alternative
- Due by next lecture

- http://tutorial.math.lamar.edu/Classes/CalcIII/GradientVectorTangentPlane.aspx
- http://tutorial.math.lamar.edu/Classes/CalcIII/VectorFields.aspx
- https://www.ijsr.net/archive/v5i1/NOV152938.pdf
- http://daslhub.org/unlv/wiki/lib/exe/fetch.php?media=alvarop:lecturexxhandouts.pdf
- http://phoenix.goucher.edu/~jillz/cs325_robotics/goodrich_potential_fields.pdf
- http://www.cs.cmu.edu/~./motionplanning/lecture/Chap4-Potential-Field_howie.pdf
- https://www.gnu.org/software/octave/
- https://www.gnu.org/software/octave/doc/interpreter/