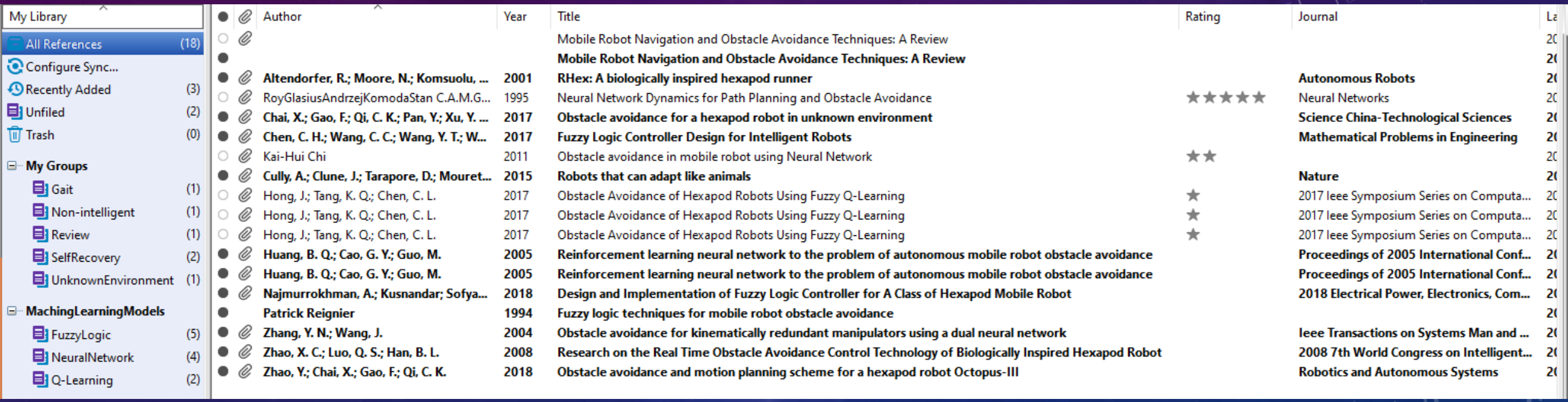
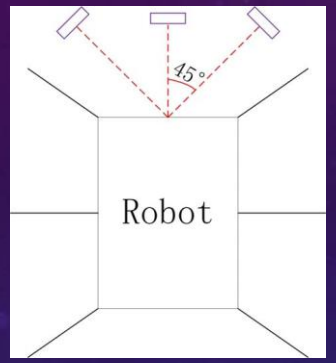
HEXAPOD ROBOT CONTROL SYSTEM

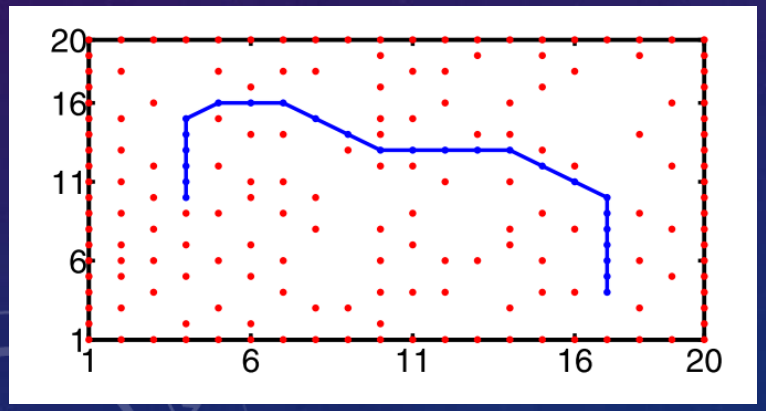
3RD MEETING – 2019/12/13

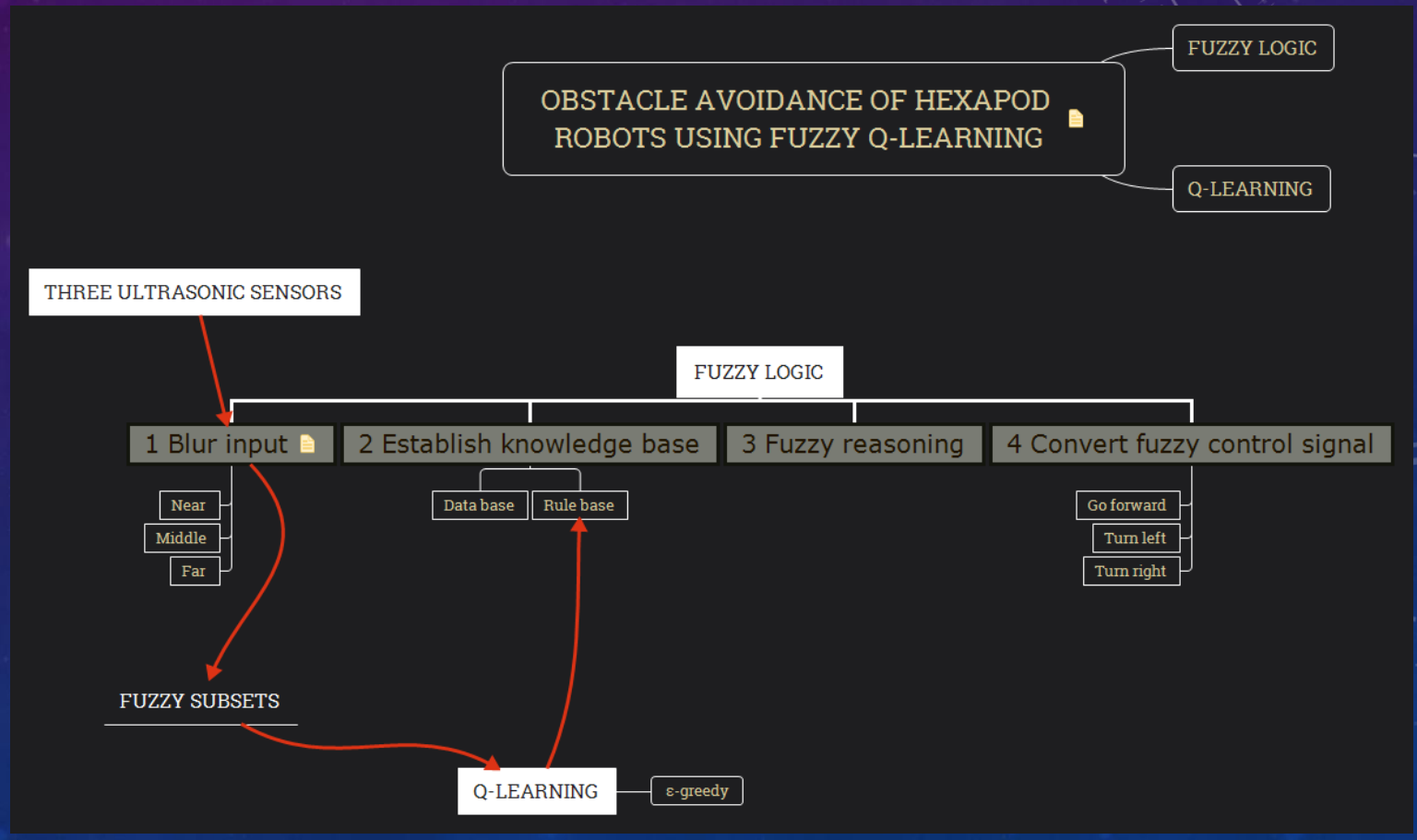


EndNote library

OBSTACLE AVOIDANCE OF HEXAPOD ROBOTS USING FUZZY Q-LEARNING

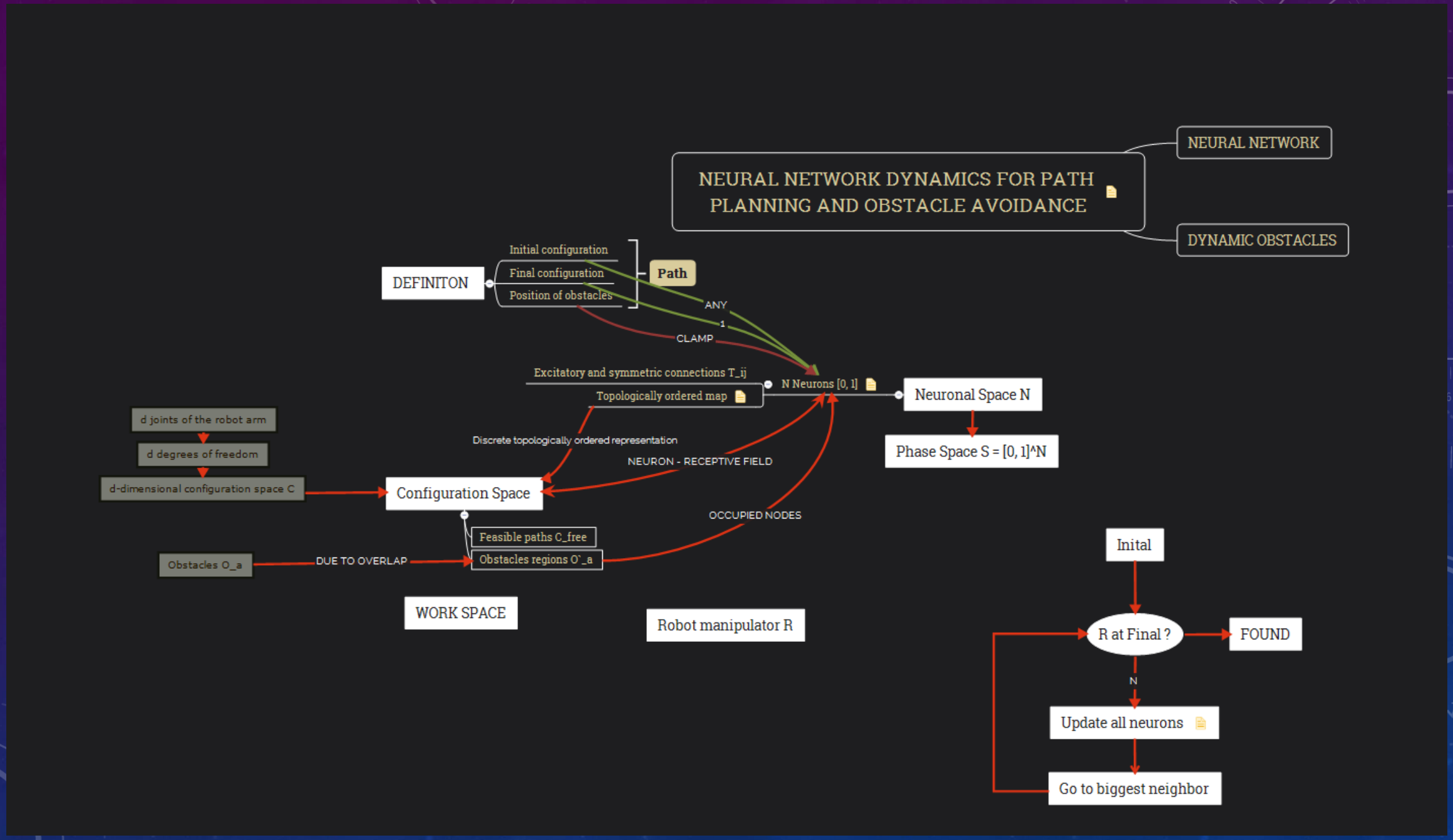






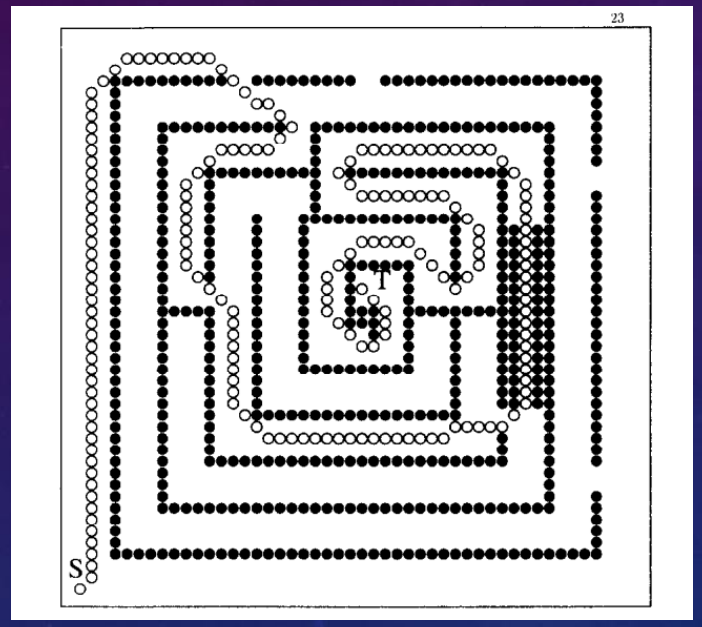
NEURAL NETWORK DYNAMICS FOR PATH PLANNING AND OBSTACLE AVOIDANCE

- Use Neuronal Space N to store the geometry information (obstacles)
- The path finding is done from both the initial position and destination
- Can deal with dynamic obstacles
- May be used to deal with unknown area with modifications to the neurons
- A lot of maths - but not intelligent ?


NEURAL NETWORK DYNAMICS FOR PATH PLANNING AND OBSTACLE AVOIDANCE
NEURAL NETWORK
DYNAMIC OBSTACLES
DEFINITON
Initial configuration
Final configuration
Position of obstacles
Path
ANY
1
CLAMP
Excitatory and symmetric connections T_ij
Topologically ordered map
N Neurons [0, 1]
Neuronal Space N
Phase Space S = [0, 1]^N
d joints of the robot arm
d degrees of freedom
d-dimensional configuration space C
Discrete topologically ordered representation
NEURON - RECEPTIVE FIELD
Configuration Space
OCCUPIED NODES
Feasible paths C_free
Obstacles O_a
DUE TO OVERLAP
Obstacles regions O`_a
WORK SPACE
Robot manipulator R
Inital
R at Final ?
FOUND
N
Update all neurons
Go to biggest neighbor

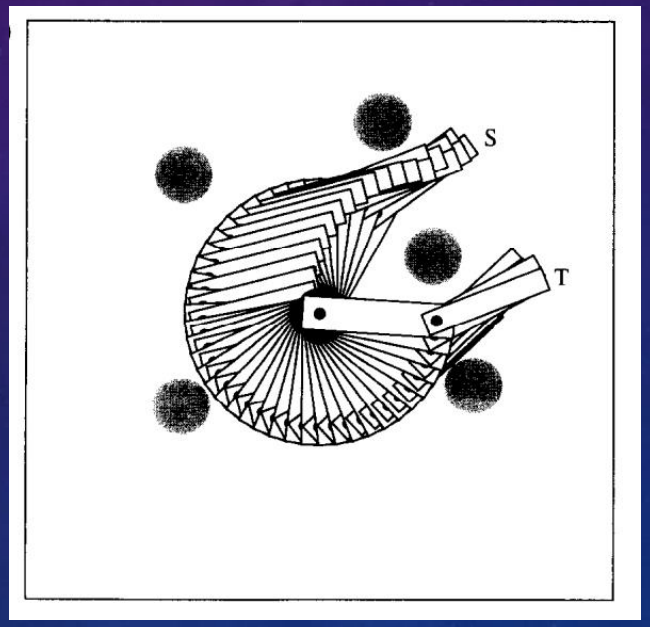
NEURAL NETWORK DYNAMICS FOR PATH PLANNING AND OBSTACLE AVOIDANCE

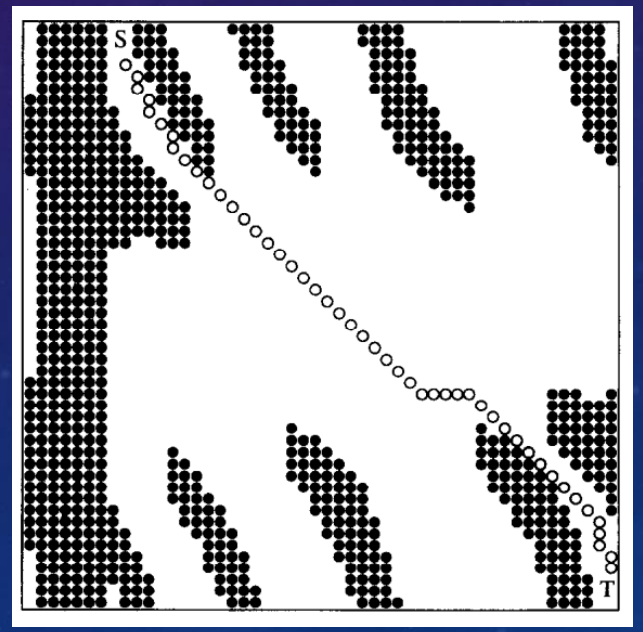
Complex as well as dynamic



Shouldn't go randomly at first ?

2D Robot arms (can rotate)





How to generate neuronal space ?

How to quantify arm behavior?

OBSTACLE AVOIDANCE IN MOBILE ROBOT USING NEURAL NETWORK

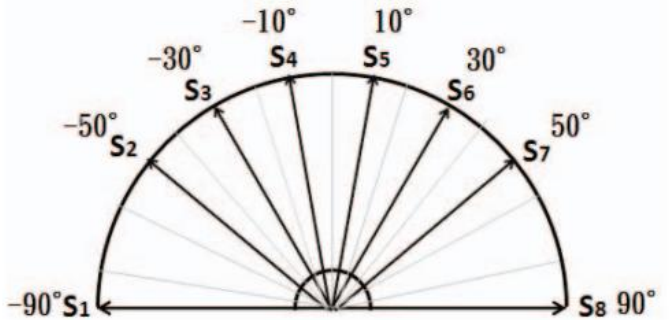
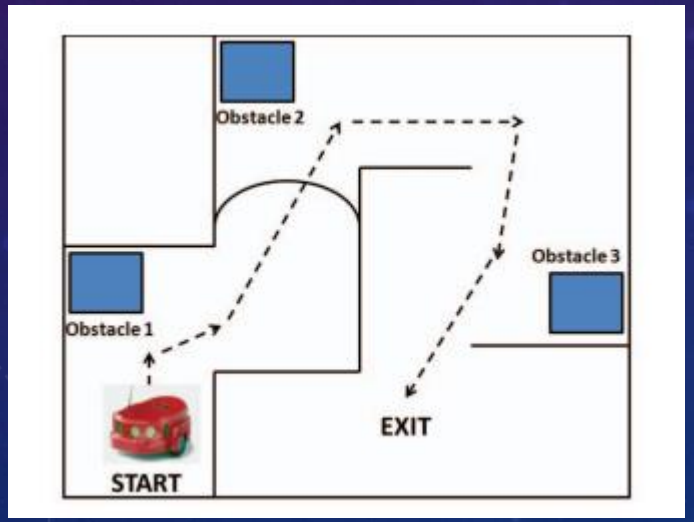


Figure 2. sonar direction and degree in P3DX



- For data training, we construct all obstacle possibilities and define respond of the output.
- Back propagation model (to study)

Step 1 Initialize weights and thresholds to small random values.

Step 2 Choose an input-output training data set $(x(k),t(k))$

Step 3 Compute NN signals from input to output:

$$O_p^{(l)} = f(\sum_i W_{pi}^{(l)} O_i^{(l-1)}) \quad (1)$$

Step 4 Compute output error E and Back propagation parameter at the output layer (L)

$$\delta_i^{(L)} = [t_i - o_i^L][f'(tot_i^{(L)})] \quad (2)$$

$$E = \frac{1}{2}\sum_{i=1}^{n}(t_i - o_i^l)^2 + E \quad (3)$$

Step 5 Update the weights using:

$$\Delta W_{if}^{(l)} = \eta \delta_i^{(l)} o_j^{(l-1)} \quad (4)$$

With Back propagation:

$$\delta_i^{(l-1)} = f'\left(tot_i^{(l-1)}\right)\sum_R \delta_p^{(l)} w_{pi}^{(l)} \text{ form } l = L \text{ to } 2 \quad (5)$$

Step 6 Repeat steps 2-5 for another training data set and compute error.

Step 7 After using all training data sets (i.e., one epoch), if final error E is less than a predetermined tolerance. The network has been trained. If not, repeat the process for another epoch.

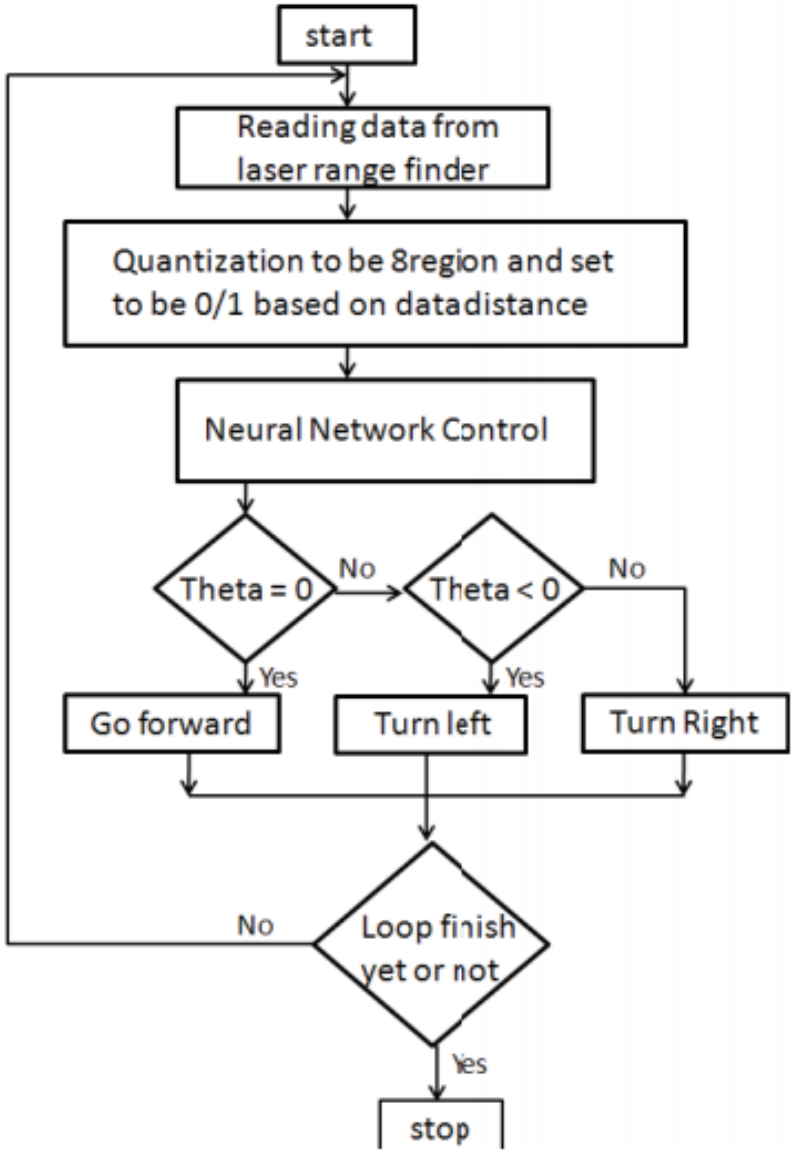
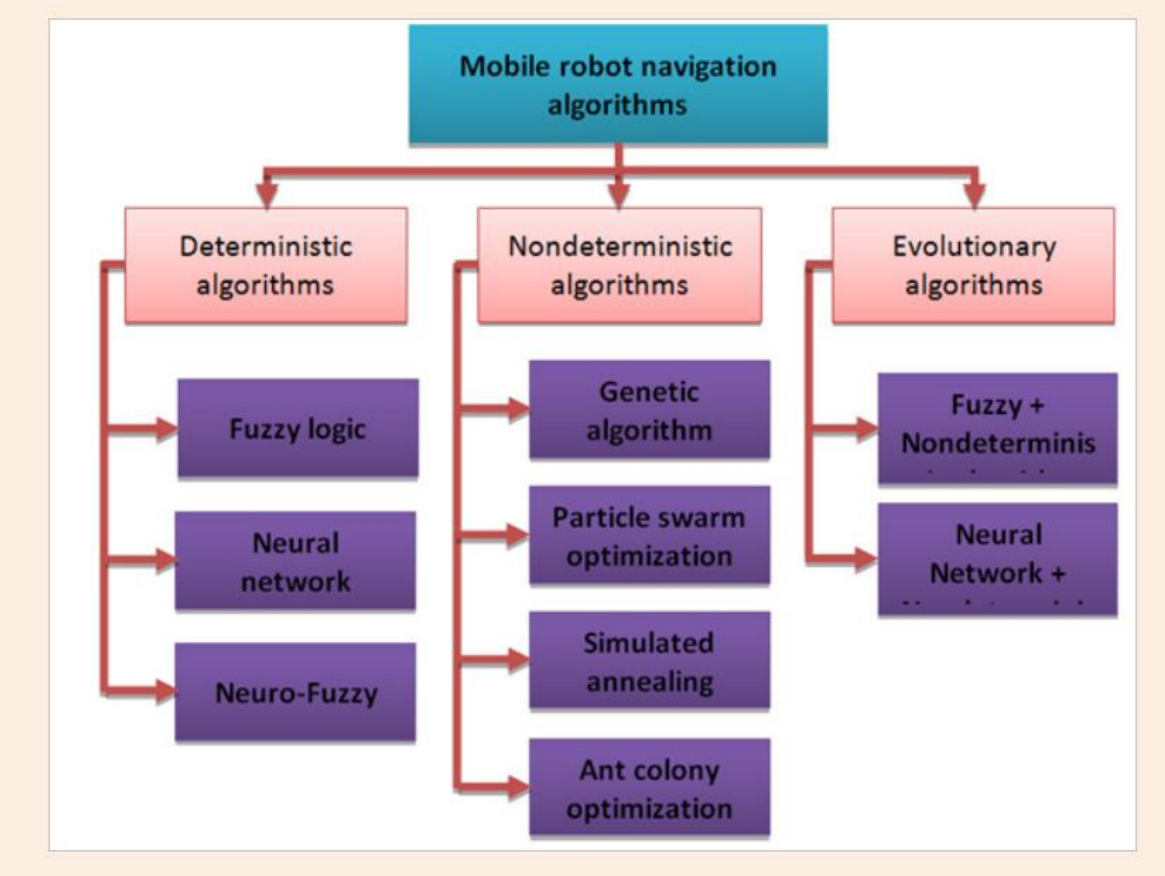


Figure 7. Flow chart of main algorithm structure

REVIEW



Mobile Robot Navigation and Obstacle Avoidance Techniques: A Review