22. Fuzzy Logic

- **Fuzzy Logic**
- **Odometry**

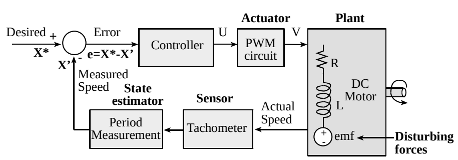
***"I think there is a world market for maybe five computers"* Thomas Watson, chairman of IBM, 1943**

Things that can go bad

- Hitting the wall
  - Think of three ways to tell if you hit the wall
  - Corrective measures
- Wrong-way Dayo
  - Think of ways to reduce the chances
  - Three repairs -> disqualification
- Other robots in the way
  - Can you distinguish a robot from a wall?
  - Strategy for passing

Comparison

- Incremental
  - \+ Simple, stable
  - \- Slow response



- PID or PI
  - \+ Theory, fast response
  - \- Needs empirical tuning, depends on load
- Fuzzy Logic Maps human intuition into rules
  - \+ Fast, good when you have expert knowledge
  - \+ Abstractive approach
  - \- Needs empirical tuning

Fuzzy Membership Set

- Membership set, variable, set
  - Value specifying levels of truth
  - Collection describes the entire system

0........32........64.........96.........128........160........192........224........255
Not at all...a little bit...somewhat...mostly...pretty much...definitely

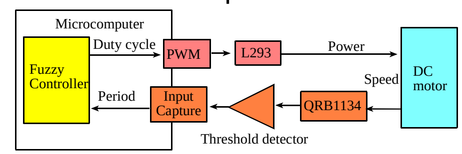
- Examples for a speed control system
  - TooSlow
  - SpeedOK
  - TooFast
  - SlowingDown
  - SpeedConstant
  - SpeedingUp

Fuzzy Membership Set

- Lab 7 example membership sets
  - Too close to the right wall
  - Distance to the right wall is ok
  - Too far away from the right wall
  - Too close to the left wall
  - Distance to the left wall is ok
  - Too far away from the left wall
  - Open space to 30 degrees to the right
  - Open space to straight ahead
  - Open space to 30 degrees to the left

Speed Controller

- Desired state
  - X* is the desired tach period
- Physical plant
  - X real state variable, actual period
- State estimator, data acquisition
  - X' measured tach period



Fuzzy approach

- Preprocessor
  - Crisp inputs (variables with units)
- Fuzzification
  - Input membership sets
- Fuzzy rules
  - Output membership sets
- Defuzzification
  - Crisp outputs (variables with units)
- Postprocessor and actuator output

Fuzzy approach

- Preprocessor, crisp inputs
  - E = X*-X' error in motor period
  - D = X'(n) – X'(n-1) acceleration

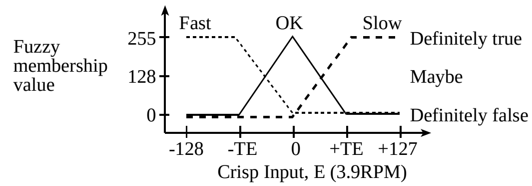
```
unsigned char Ts;     // Desired Speed in 3.9 rpm units
unsigned char T;      // Current Speed in 3.9 rpm units
unsigned char Told;   // Previous Speed in 3.9 rpm units
char D;               // Change in Speed in 3.9 rpm/time units
char E;               // Error in Speed in 3.9 rpm units
void CrispInput(void){
   E=Subtract(Ts,T);
   D=Subtract(T,Told);
   Told=T;    /* Set up Told for next time */
}
```

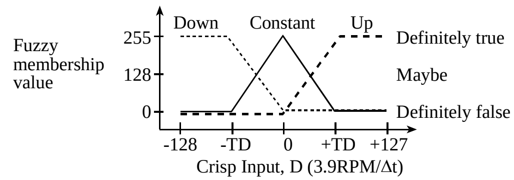
Fuzzy approach

- Preprocessor, crisp inputs
  - E = X*-X' error in motor period
  - D = X'(n) – X'(n-1) acceleration
- Fuzzification

Slow	True if the motor is spinning too slow
OK	True if the motor is spinning at the proper speed
Fast	True if the motor is spinning too fast
Up	True if the motor speed is getting larger
Constant	True if the motor speed is remaining the same
Down	True if the motor speed is getting smaller.

Fuzzification





Fuzzification

```
#define TE ???
long Fast, OK, Slow, Down, Constant, Up;
#define TD ???
long Increase,Same,Decrease;
#define TN ???
void InputMembership(void){
 if(E <= -TE) {        /*  E <= -TE */
   Slow = 255;
   OK = 0;
   Fast = 0;}
 else
   if (E < 0) {        /* -TE<E<0 */
     Slow = (255*(-E))/TE;
     OK = 255-Slow;
     Fast = 0;}
   else
     if (E < TE) {      /*  0<E<TE */
       Slow = 0;
       Fast =(255*E)/TE;
       OK = 255-Fast;}
     else {             /* +TE <= E */
       Slow = 0;
       OK = 0;
       Fast = 255;
 }

if(D <= -TD) {          /* D <= -TD  */
   Up = 255;
   Constant = 0;
   Down = 0;}
 else
   if (D < 0) {         /* -TD<D<0   */
     Up = (255*(-D))/TD;
     Constant = 255-Up;
     Down = 0;}
   else
     if (D < TD) {      /*  0<D<TD   */
       Up = 0;
       Down = (255*D)/TD;
       Constant = 255- Down;}
     else {             /* +TD <= D    */
       Up = 0;
       Constant = 0;
       Down = 255;
   }
}
```


Fuzzy rules

If *OK* and *Constant* then *Same*
If *OK* and *Up* then *Decrease*
If *Fast* and *Constant* then *Decrease*
If *Fast* and *Up* then *Decrease*
If *OK* and *Down* then *Increase*
If *Slow* and *Constant* then *Increase*
If *Slow* and *Down* then *Increase*

E \ D	*Down*	*Constant*	*Up*
Slow	*Increase*	*Increase*	
OK	*Increase*	*Same*	*Decrease*
Fast		*Decrease*	*Decrease*

Fuzzy rules

Same=(*OK*and*Constant*)
Decrease=(OKand*Up*)*or*(*Fast*and*Constant*)or(*Fast*and*Up*)
Increase=(*OK*and*Down*)or(*Slow*and*Constant*)or(*Slow*and*Down*)

- and operation is minimum
- or operation is the maximum

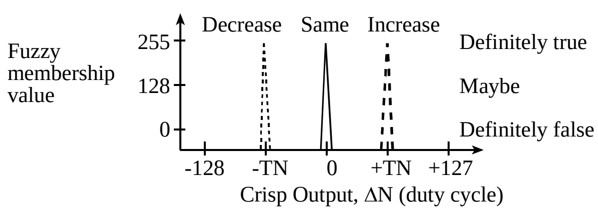
```
unsigned char static min(unsigned char u1,unsigned char u2){
  if(u1>u2) return(u2);
  else return(u1);
}
unsigned char static max(unsigned char u1,unsigned char u2){
  if(u1<u2) return(u2);
  else return(u1);
}
```


Fuzzy rules

```
void OutputMembership(void){
  Same=min(OK,Constant);
  Decrease=min(OK,Up)
  Decrease=max(Decrease,min(Fast,Constant));
  Decrease=max(Decrease,min(Fast,Up));
  Increase=min(OK,Down)
  Increase=max(Increase,min(Slow,Constant));
  Increase=max(Increase,min(Slow,Down));
}
```

E \ D	*Down*	*Constant*	*Up*
Slow	*Increase*	*Increase*	
OK	*Increase*	*Same*	*Decrease*
Fast		*Decrease*	*Decrease*

Defuzzification



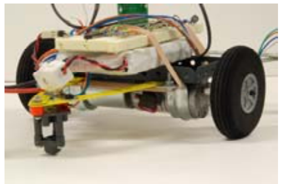
ΔN=(*Decrease*•(-TN) + *Same*•0 +*Increase*•TN)/(*Decrease*+*Same*+*Increase*)

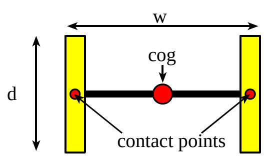
```
long dN;
void CrispOutput(void){
 dN=(TN*(Increase-Decrease))/(Decrease+Same+Increase);
}
```

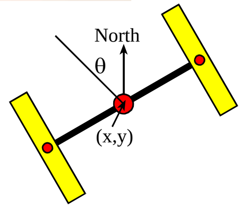

Fuzzy Logic Controller

```
void Timer0A_Handler(void){
 T = SE();                  // estimate speed, set T, 0 to 255
 CrispInput();              // Calculate E,D and new Told
 InputMembership();         // Sets Fast,OK,Slow,Down,Constant,Up
 OutputMembership();        // Sets Increase,Same,Decrease
 CrispOutput();             // Sets dN
 N = max(0,min(N+dN,255));
 PWM0_Duty(N);              // output to actuator, Program 8.4
 TIMER0_ICR_R = TIMER_ICR_CAECINT;// ack
}
```

Odometry

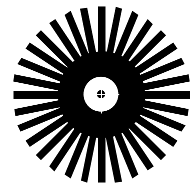






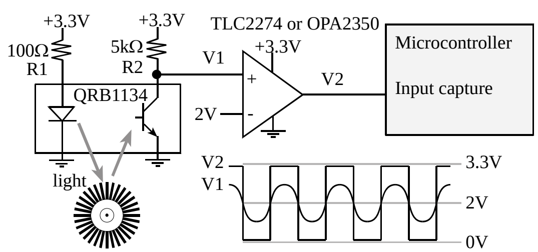
Constants

- Number of slots/rotation, n=32
- Wheel diameter, d=886 (0.01cm)
- Wheelbase, w=1651 (0.01cm)
- Wheel circumference, c=πd=2783 (0.01 cm)



Measurements

- LCount the number of left slots in Δt
- RCount the number of right slots in Δt
- Counts vary from -28 to +28 each Δt

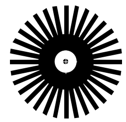


Simple cases

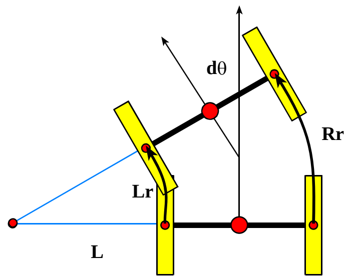
- -28 ≤ m ≤ +28 each Δt

LCount	***RCount***	*Motion*
m	m	straight line motion in the current direction
0	m	pivot about stopped left motor
m	0	pivot about stopped right motor
m	-m	pure rotation about cog

Derivations



c = circumference
n = number of slots per rotation
w = wheelbase
Lr = LCount*c/n the arc distance traveled by the left wheel (0.01cm)
Rr = RCount*c/n the arc distance traveled by the right wheel (0.01cm)

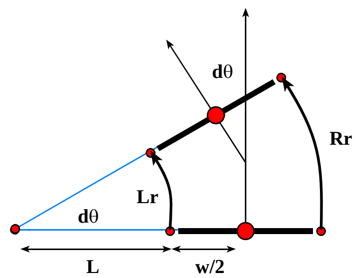


Assume Lr, Rr positive
Assume Rr>Lr

L/Lr = (L+w)/Rr
L/Lr - L/Rr = w/Rr
L Rr - L Lr = w Lr
L = w Lr/(Rr - Lr)

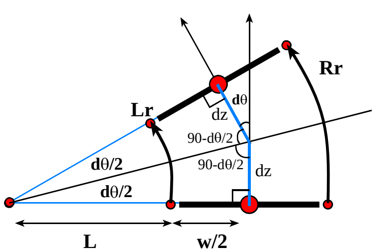
Derivations

dθ = Lr/L = Rr/(L+w)



Derivations

We can divide the change in position into two components



dz = (L+w/2)*tan(dθ/2)

if dθ is small

dz = dθ/2*(L+w/2)

Odometry

- Needs very accurate sensors
- Errors accumulate
- OK for relative travel from known position
  - periodic absolute knowledge of position

Lr = LCount*c/n	**(0.01cm)**
Rr = RCount*c/n	**(0.01cm)**
L = (w*Lr)/(Rr - Lr)	**(0.01cm)**
dθ = (100*Lr)/L	**(0.01 radians)**
dz = ((dθ/2)*(L+w/2))/100	**(0.01cm)** ***approximation***
x = x + dz*cos(θ)	**(0.01cm)**
y = y+ dz*sin(θ)	**(0.01cm)** ***first part of move***
θ = θ + dθ	**(0.01 radians)**
x = x + dz*cos(θ)	**(0.01cm)**
y = y+ dz*sin(θ)	**(0.01cm)** ***second part of move***