SPAOFE - Sonar Perception Avoiding Objects Fuzzy Engine

SPAOFE is an implementation of the General Perception Concept in the XR – The First robot. The document that will guide this implementation is “Fuzzy Logic Wall Following of a Mobile Robot Based on the Concept of General Perception” – Reinhard Braunstingl – Pedro Sanz – Jose Manuel Ezkerra (1)

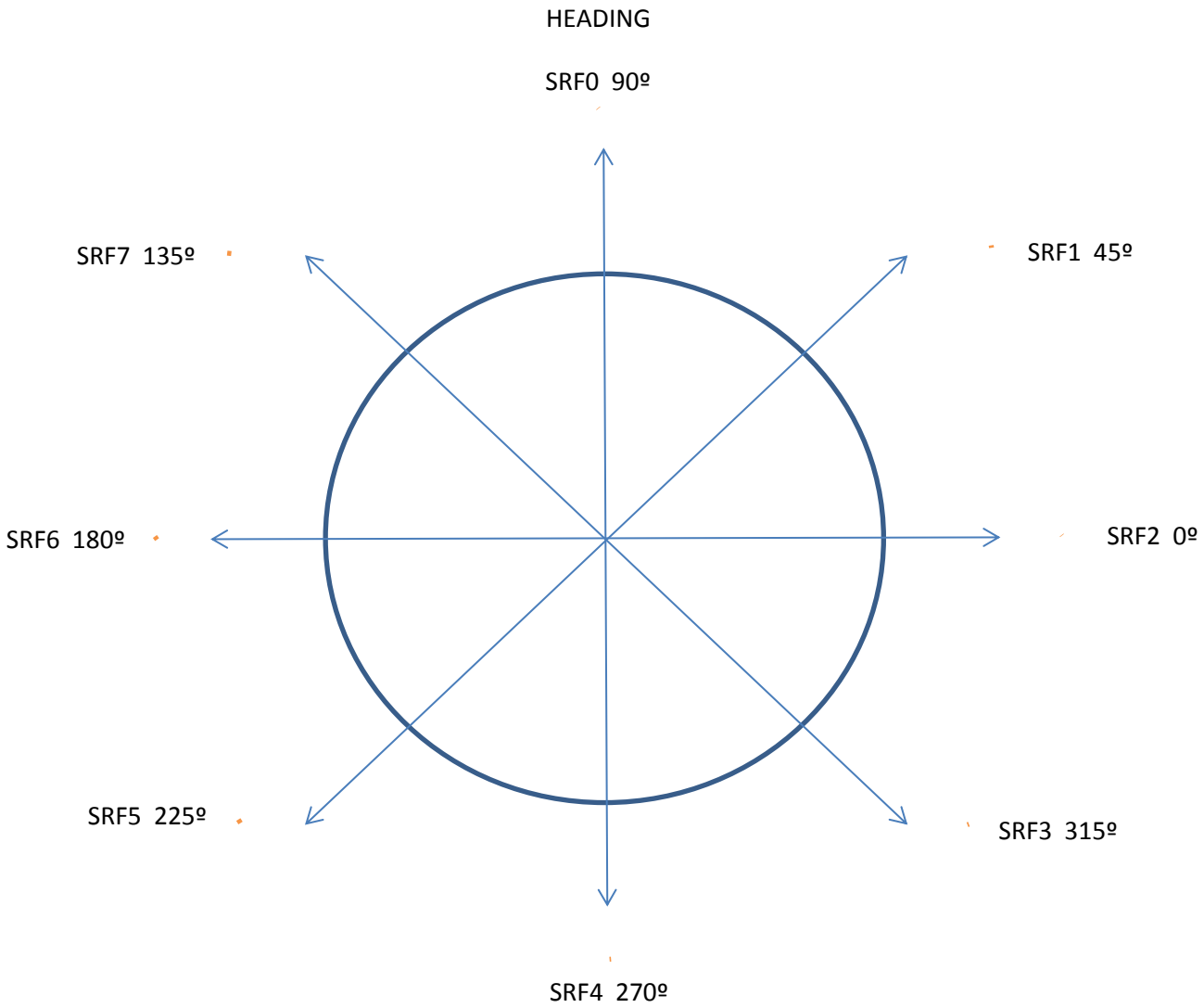


FIGURE 1

Will be used eight SRF08 ultrasonic rangers at 45º intervals as in FIGURE 1. The ultrasonic ranger's ring is fixed to a plane of the robot (this implies that the output value $\boldsymbol{\varphi}$· orientation-turn do not apply) and must be 30cm or more over the floor. The direction angle of each sensor (SRF0 … SRF7) refers to the axis of SRF2 (which has direction angle 0º) and increases in the counter clockwise direction. The heading (and velocity) is at 90º (SRF0 axis).

Each sensor $\boldsymbol{i}$ (i=0… 7) has a Perception Vector $\boldsymbol{p_i}$ associated to it.

$\boldsymbol{p_i}$
- direction => sensor direction angle
- length => $(\boldsymbol{d_{max}} - \boldsymbol{d_i}) / (\boldsymbol{d_{max}} - \boldsymbol{d_{min}})$
  - => 0 for $\boldsymbol{d_i} > \boldsymbol{d_{max}}$
  - => 1 for $\boldsymbol{d_i} < \boldsymbol{d_{min}}$

where $\boldsymbol{d_i}$ is the distance measured by the sensor, $\boldsymbol{d_{max}}$ and $\boldsymbol{d_{min}}$ are the longest and shortest distance at which the sensor can reliably detect an object. In this case I will use $\boldsymbol{d_{min}}$ = 30cm and $\boldsymbol{d_{max}}$ = 400cm .

The General Perception vector $\boldsymbol{P}$ is composed of all individual perception $\boldsymbol{p_i}$.

$\boldsymbol{P}$
- direction => direction of the vectorial sum of all individual perception $\boldsymbol{p_i}$
- length => strongest individual perception $\boldsymbol{p_{i,max}}$

The General Perception Change in time $\boldsymbol{P}^{\cdot *}$ is the maximum value of the Perception Change in time of all sensors $\boldsymbol{p}^{\cdot *}_{i,max}$.

$\boldsymbol{p}^{\cdot *}_{i} =$
- $- \triangle \boldsymbol{d_i} / (\triangle \boldsymbol{t} . \boldsymbol{V_{max}})$ for $\triangle \boldsymbol{d_i} < 0$
- 0 for $\triangle \boldsymbol{d_i} >= 0$

In this case $\boldsymbol{V_{max}}$ = 40cm/sec and $\boldsymbol{V_{min}}$ = 2cm/sec

Fuzzy Controller

I. Input Values

- **α** => angle between the General Perception vector and the heading
- ***P*** => General Perception length
- $\dot{P}^{*}$ => General Perception Change in time

II. Output values

- $\dot{\Psi}$ => direction (steer)
- $\dot{V}$ => speed (acceleration)

III. Rule Base

There are two groups of rules giving a total of 27 rules. The groups are for: change of direction $\dot{\Psi}$ and change of speed $\dot{V}$

1. **Rules for change of direction $\dot{\Psi}$**

α \ *P*	**VLP**	**LP**	**MP**	**HP**	**VHP**
RB	HR	HR	R	R	C
RF	C	L	L	HL	HL
LF	C	R	R	HR	HR
LB	HL	HL	L	L	C

2. Rules for change of speed $\dot{V}$

$\dot{P}^{*}$ \ ***P***	**VLP or VHP**	**LP or HP**	**MP**	
ZE	Z	P	P	
LO	EB	B	Z	
HI				EB

3. **Adjectives**

- RB - Right Back
- RF - Right Front
- LF - Left Front
- LB - Left Back
- VLP - Very Low Perception
- L - Left
- R - Right
- LP - Low Perception
- MP - Medium Perception
- HP - High Perception
- VHP - Very High Perception
- HR - Hard Right
- C - Center
- HL - Hard Left
- ZE - Zero
- LO - Low
- HI - High
- Z - Zero
- P - Positive
- EB - Emergency Breaking
- B - Breaking

4. **The rules are in the form:**

IF <condition> THEN <consequence>

More than one condition with the AND operator are combined via minimum. With the OR operator are combined via maximum.

In the case that several rules are valid simultaneously, a defined control command has to be generated using contradictory rule outputs. We will use the "*correlation-product encoding with sum combination and centroid defuzzification"* (2).

IV. The process

1. Read the sensors ($\boldsymbol{p_i}$ for i = 0 to 7).
2. Calculate **α**
3. Calculate ***P***
4. Calculate $\boldsymbol{P}^{*}$
5. Fuzzify the input values using the membership functions defined in (1).
6. Apply the Rule Base.
7. Defuzzify using (2) and the membership functions defined in (1) to obtain crisp values for the output values.
8. Apply the output values via the motion control of the robot. The objective is to keep the speed vector normal to the General Perception vector and the length of the General Perception vector at the value **MP**.

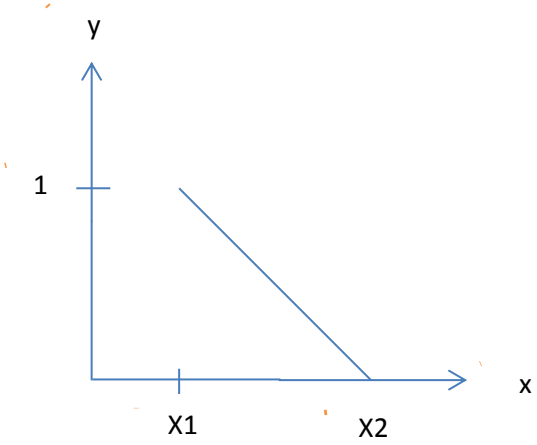
V. Implementation

The **SPAOFE** first version is implemented simulating the sensors reading and through the functions:

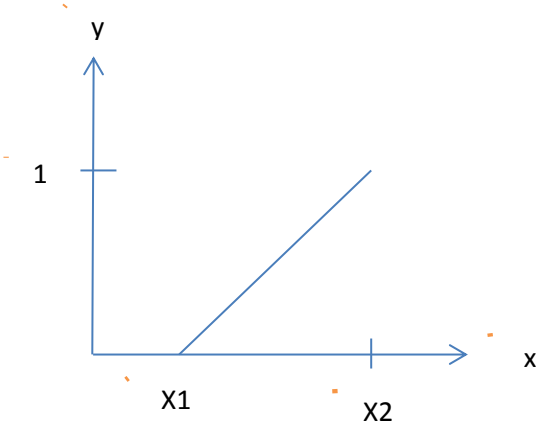
1. Range_sonar_X
2. General_Perception_Vector
3. General_Perception_V_Change
4. Perception_Angle_Membership
5. Perception_value_membership
6. Perception_Change_Membership
7. Rules_Steer
8. Rules_Acceleration
9. Cutted_Defuzzy_Steer
10. Scaled_Defuzzy_Steer
11. Cutted_Defuzzy_Acceleration
12. Scaled_Defuzzy_Acceleration
13. fHR
14. fR
15. fC
16. fL
17. fHL
18. fEB
19. fB
20. fZ
21. fP

For these functions I will use:

1. **Stright line equations:**



$y = 1 – (x – x1)/(x2 – x1)$



$y = (x – x1)/(x2 – x1)$

2. **Sectors index number:**
   - Percetion angle
     - Right_back => 0
     - Right _Front => 1
     - Left_Front => 2
     - Left_Back => 3
   - Perception value
     - Very_Low => 0
     - Low => 1
     - Medium => 2
     - High => 3
     - Very_High => 4
   - Perception change
     - Zero => 0
     - Low => 1
     - High => 2

- Steer
  - Hard_Right => 0
  - Right => 1
  - Center => 2
  - Left => 3
  - Hard_Left => 4
- Acceleration
  - Em_Brake => 0
  - Brake => 1
  - Zero = > 2
  - Positive => 3

3. Global data

```
#define zone 1.0

#define zzero 0.0

#define zor(a,b) ((a) > (b) ? (a) : (b))

#define zand(a,b) ((a) < (b) ? (a) : (b))

#define znot(a) (zone+zzero-a)

#define N_Sonar 8

#define infinito 65535

#define reta_up(n,x) (x - x1[n])/(x2[n]-x1[n])

#define reta_down(n,x) 1-(x - x1[n])/(x2[n]-x1[n])


//Actualized via timer interrupt every millisecond

unsigned long int relogio = 0;

// 1 - True; 0 - False.

unsigned char Brake;

// Sectors VLP,LP,MP,HP,VHP.

float PV[5];

// PV sector index.
```

```
unsigned char iPV[5];
// Number of iPVs.
unsigned char nPV;
// Sectors RB,RF,LF,LB.
float PA[4];
// PA sector index.
unsigned char iPA[4];
// Number of iPAs.
unsigned char nPA;
// Sectors ZE,LO,HI.
float PC[3];
// PC sector index.
unsigned char iPC[3];
// Number of iPCs.
unsigned char nPC;
// Steer sector index.
unsigned char iSteer[10];
// Corresponding cut values.
float CSteer[10];
// Number of iSteer.
unsigned char nSteer;
// Acceleration sector index.
unsigned char iAcc[10];
// Corresponding cut values.
float CAcc[10];
// Number of iAcc.
unsigned char nAcc;
```

```
// Steer crisp value.
float Steer_Crisp;
// Turn direction (-1 right; +1 left).
int Direcao;
// Turn radius (cm) to obtain Steer_Crisp (degrees/s).
float Rg;
// Acceleration crisp value.
float Acceleration_Crisp;

struct sonar
{
        // Number of sensors in the ring .
        unsigned char Sonar_Number;
        // Distance previous reading instant of time
        unsigned long int Ti_1[N_Sonar];
        // Time elapsed since previous distance reading.
        unsigned long int DTi[N_Sonar];
        // Sensors I2C addresses.
        unsigned char Sonar_Addr[N_Sonar];
        // 0 if > Minimum_Distance; 1 if <= Minimum_Distance).
        unsigned char Sonar_Condition[N_Sonar];
        // Sensor readed distance.
        unsigned int Sonar_Distance[N_Sonar];
        // Distance change
        int Sonar_Distance_Change[N_Sonar];
        // Sensor individual perception direction (degrees).
        float Sonar_Angle[N_Sonar];
```

```
        // Normalized individual Perception vector value.
        float Perception_Value[N_Sonar];
        // General Perception change in time.
        float General_Perception_Change;
        // General Perception direction. (-180 to 180) degrees to heading
        float General_Perception_Angle;
        // Normalized General Perception Value (0 to 1).
        float General_Perception_Value;
        // Maximum acceleration (cm/s^2).
        float Maximum_Acceleration;
        // Maximum velocity (cm/s).
        float Maximum_Velocity;
        // Minimum velocity (cm/s).
        float Minimum_Velocity;
        // Maximum distance (cm).
        float Maximum_Distance;
        // Minimum distance (cm).
        float Minimum_Distance;
        // Normal velocity (cm/s) corresponding to acceleration sector ZERO.
        float Velocity;
};
```

4. Initializations

```
struct sonar SPAOFE =
{
.Sonar_Number = N_Sonar,
.Sonar_Addr = {0xE0,0xE2,0xE4,0xE6,0xE8,0xEA,0xEC,0xEE},
.Sonar_Angle = {90.0,45.0,0.0,315.0,270.0,225.0,180.0,135.0},
.Maximum_Acceleration = 20.0;
.Velocity = 15;
.Maximum_Velocity = 40.0,
.Minimum_Velocity = 2.0;
.Maximum_Distance = 400.0,
.Minimum_Distance = 30.0,
.Sonar_Distance = {infinito,infinito,infinito,infinito,infinito,infinito,infinito,infinito},
.Ti_1 = {relogio,relogio,relogio,relogio,relogio,relogio,relogio,relogio}
};
```