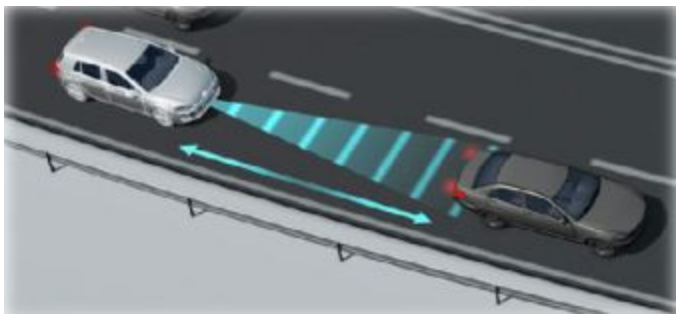
PRÁCTICA 2: FUZZY-CLIPS

Control de crucero de un vehículo



Pascual Andrés Carrasco Gómez

Índice

Introducción

El sistema se basa en un control automático de frenado de un coche, lo hace mediante dos parámetros que son la distancia relativa y la velocidad relativa entre el coche y el coche que tiene situado delante. Los parámetros se los damos como entrada por consola, pero perfectamente podrían ser dos sensores situados en el coche.
El sistema también dispone de un parámetro que representa la presión que hemos de proporcionar al freno dependiendo de la distancia y la velocidad relativa proporcionada.
Este sistema es un ejemplo de sistemas que se tienen que implementar con lógica difusa ya que contiene conocimiento que no es preciso, ya que cuando decimos que un coche está cerca de otro que entendemos por “cerca”, 2 metros, 5 metros, …
También ocurre con la velocidad relativa entre ambos coches ya que por ejemplo si el coche se está alejando es porque tienen una velocidad relativa de -30 km/h, -20 km/h, …
Observamos que el conocimiento es impreciso y necesitamos de una herramienta como es “Fuzzy-clips” que nos permite utilizar variables difusas con sus respectivos valores difusos, aplicar un proceso inferencial difuso, pasar valores crisp a valores difusos (mediante una función externa como es fuzzify) y finalmente defusificar el valor difuso obtenido a un valor crisp que es el que proporcionaremos al usuario.

Variables difusas

En este apartado vamos a explicar las variables difusas que hemos utilizado y como las hemos implementado en “Fuzzy-clips”. Como he comentado anteriormente el sistema utiliza tres variables difusas que se llaman “distancia”, “velocidad-relativa” y “presion-freno”. A continuación vamos a ver como están implementadas en el código y ha explicar su significado:

```
(deftemplate distancia
    0 50 metros
    ((cerca (0 1) (15 0))
    (medio (10 0) (25 1) (35 1) (40 0))
    (lejos (35 0) (50 1)))
)
```

```
(deftemplate velocidad-relativa
    -30 30 kmh
    ((alejando (-30 1) (0 0))
    (constante (-10 0) (0 1) (10 0))
    (acercando (0 0) (30 1)))
)
```

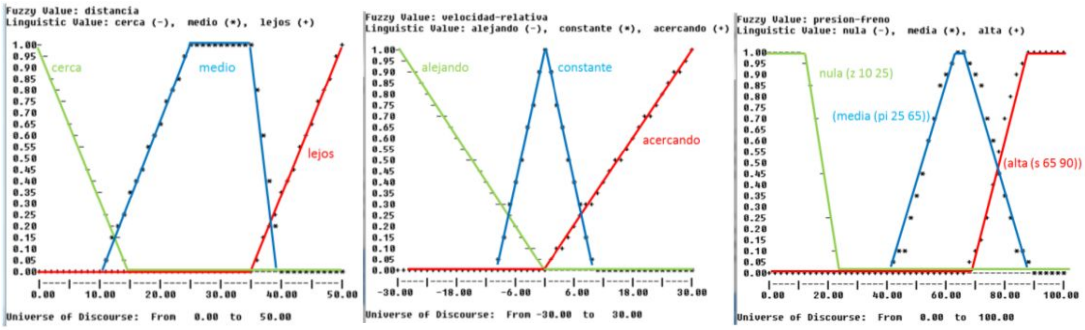
```
(deftemplate presion_freno
    0 100 %
    ((nula (z 10 25))
    (media (pi 25 65))
    (alta (s 65 90)))
)
```

La variable difusa “distancia” está definida en un universo que va desde 0 hasta 50 y está representado en metros, tiene tres valores difusos que son “cerca”, “medio” y “lejos”, los parámetros que van a continuación de cada valor difuso definen su función de pertenencia dentro del universo.

La variable difusa “velocidad-relativa” está definida en un universo que va desde -30 a 30 y está representado en km/h, los valores difusos de esta variable son “alejando”, “constante” y “acercando” con a continuación sus valores de función de pertenencia.

La variable difusa “presion-freno” está definida en un universo que va desde 0 a 100 y está representado en tanto por cien (%), sus valores difusos son “nula”, ”media” y “alta” con sus valores de función de pertenencia respectivamente.

A continuación observamos las funciones de pertenencia de cada valor difuso de cada variable difusa representadas gráficamente:



NOTA: existen dos formas de especificar los valores de las funciones de pertenencia de cada valor difusos, mediante valores en rangos de 0 y 1 o mediante funciones definidas como z, s o pi.

Reglas

Las reglas de nuestro sistema vienen definidas por la siguiente tabla que nos proporciona el problema:

Distancia *vs.* Velocidad relativa	Alejando	Constante	Acercando
Cerca	Nula	Media	Alta
Medio	Nula	Nula	Media
Lejos	Nula	Nula	Media

Observamos que las reglas definidas por el sistema son reglas que contienen variables difusas y la inferencia necesaria es una inferencia difusa que nos proporciona “Fuzzy-clips”, la definición de las reglas sigue el patrón de una R y el número de regla, a continuación vamos a comentar la definición de una regla ya que las demás reglas son iguales.

```
(defrule R6
    (declare (salience 100))
    (distancia medio)
    (velocidad-relativa acercando)
=>
    (assert (presion_freno media))
)
```

Observamos que el antecedente de las reglas tienen valores difusos y el consecuente inserta el valor difuso para la presión de freno.

Las reglas como he comentado anteriormente vienen definidas por la tabla proporcionada por el problema que para cada distancia y velocidad nos proporciona un valor aplicable al freno.

NOTA: El salience 100 lo tienen todas las reglas del tipo Rx para que se ejecuten antes de la regla de defusificación, ya que la defusificación es el último paso y tiene que ir después de la inferencia.

Como podemos observar estamos trabajando con variables difusas en todo momento, pero el usuario necesita un valor en porcentaje de la presión que le tiene que aplicar al freno, ese valor como podemos entender no puede ser un valor difuso sino que tiene que ser un valor crisp, para ello tenemos una regla de defusificación que nos permite obtener el valor crisp obtenido del valor difuso obtenido por la inferencia difusa del sistema. La regla de defusificación es una parte muy importante del sistema y la vamos a comentar a continuación:

```
; DEFUSIFICACION
(defrule obtener_presion
    (declare (salience 50))
    (presion_freno ?pf)
=>
    (bind ?aux1 (maximum-defuzzify ?pf))
    (printout t "Criterio-Maximo: La presion a aplicar es :" ?aux1 (get-u-units ?pf) crlf)
    (bind ?aux2 (moment-defuzzify ?pf))
    (printout t "Criterio-Momento: La presion a aplicar es :" ?aux2 (get-u-units ?pf) crlf)
    (bind ?media (/ (+ ?aux1 ?aux2) 2))
    (printout t "Media: La media de los criterios obtenidos es: " ?media (get-u-units ?pf) crlf)
)
```

Para la defusificación utilizamos e imprimimos dos estrategias que son las del máximo y la del momento (maximum-defuzzify y moment-defuzzify respectivamente), para obtener un resultado más realista realizamos la media de ambos valores, la función get-u-units nos proporciona el valor que tiene el universo de la variable difusa “presion_freno” que es como he comentado anteriormente el porcentaje de presión (%) a aplicar. Observamos que la regla de defusificación no lleva la instrucción “halt” esto se debe a que “Fuzzy-clips” para automáticamente cuando no tiene reglas a aplicar, lo veremos con las ejecuciones posteriormente.

NOTA: Por lo comentado en la anotación anterior la regla de defusificación tiene un salience más pequeño que las reglas de tipo Rx, ya que se tiene que ejecutar la última.

Por otra parte tenemos dos reglas que nos permiten obtener información del usuario:

```
; INTERACCION USUARIO
(defrule preguntar_distancia
    (declare (salience 1000))
    ?p1 <- (pregunta1 si)
=>
    (printout t "Introduzca la distancia en metros ([0 - 50])" crlf)
    (bind ?aux (read))
    (fuzzify distancia ?aux 0)
    (retract ?p1)
)
```

```
(defrule preguntar_velocidad
    (declare (salience 1000))
    ?p2 <- (pregunta2 si)
=>
    (printout t "Introduzca la velocidad en km/h ([-30 - 30])" crlf)
    (bind ?aux (read))
    (fuzzify velocidad-relativa ?aux 0)
    (retract ?p2)
)
```

Estas reglas se lanzan las primeras ya que las he declarado con un “salience” (prioridad) elevado, y solo se lanzan una vez y esto lo controlo con la variable preguntaX (donde X es el número de pregunta) que una vez realizada la pregunta se elimina el hecho con un retract, en estas funciones la parte interesante está en la función fuzzify que nos permite fusificar los valores crisp introducidos por teclado a valores difusos.
NOTA: Los valores que se insertan desde consola son valores difusos de tipo “singlenton”, lo observamos en el código al ver que el último valor introducido en la función “fuzzify” es 0.

Funciones

Para poder fusificar los valores introducidos por teclado hemos de incorporar en el código la siguiente función que viene definida como función externa (Ejemplos/fuzzify.clp) en la descarga del programa. La mostramos a continuación:

```
; FUNCION FUZZIFY
(deffunction fuzzify (?fztemplate ?value ?delta)
    (bind ?low (get-u-from ?fztemplate))
    (bind ?hi  (get-u-to   ?fztemplate))

    (if (<= ?value ?low)
      then
        (assert-string
          (format nil "(%s (%g 1.0) (%g 0.0))" ?fztemplate ?low ?delta))
      else
        (if (>= ?value ?hi)
          then
            (assert-string
               (format nil "(%s (%g 0.0) (%g 1.0))"
                           ?fztemplate (- ?hi ?delta) ?hi))
          else
            (assert-string
               (format nil "(%s (%g 0.0) (%g 1.0) (%g 0.0))"
                           ?fztemplate (max ?low (- ?value ?delta))
                           ?value (min ?hi (+ ?value ?delta)) ))
        )
    )
)
```

El primer parámetro de la función es la variable difusa en la que queremos insertar el valor difuso, el segundo valor es el valor crisp a introducir (en nuestro caso, el valor crisp introducido por teclado) y el último valor es la valor de certidumbre (en nuestro caso es 0 que hace referencia a introducir un valor “singlenton” como he comentado anteriormente).

Hechos iniciales

Nuestro sistema únicamente tiene dos hechos iniciales que observamos a continuación:

```
; HECHOS INICIALES
(deffacts datos
    (pregunta1 si)
    (pregunta2 si)
)
```

Estos hechos iniciales nos permiten que las reglas diseñadas para preguntar al usuario información acerca de la distancia y la velocidad del sistema se ejecuten una única vez, ya que permiten que las reglas se lancen y en el consecuente se retractan estos hechos para que no se vuelvan a lanzar más dichas reglas.
NOTA: Al tener hechos iniciales definidos con deffacts es necesario hacer un (reset) antes de ejecutar (run) para que se carguen los hechos.

Evaluación

En este apartado vamos a mostrar diferentes ejecuciones de nuestro sistema, explicando aquellos resultados que sean necesarios. Para poder ejecutar el sistema hemos de cargar el fichero “coche.clp”:

```
File  Edit  Execution  Browse  Window
FuzzyCLIPS> (load "Z:/home/pascu/Escritorio/coche.CLP")
FuzzyCLIPS> Defining deftemplate: distancia
Defining deftemplate: velocidad-relativa
Defining deftemplate: presion_freno
Defining deffacts: datos
Defining defrule: R1 +j+j
Defining defrule: R2 =j+j
Defining defrule: R3 =j+j
Defining defrule: R4 +j+j
Defining defrule: R5 =j+j
Defining defrule: R6 =j+j
Defining defrule: R7 +j+j
Defining defrule: R8 =j+j
Defining defrule: R9 =j+j
Defining defrule: obtener_presion +j
Defining deffunction: fuzzify
Defining defrule: preguntar_distancia +j
Defining defrule: preguntar_velocidad +j
TRUE
FuzzyCLIPS>
```

Observamos que se carga correctamente y que no produce ningún error. Una vez cargado ya podemos realizar ejecuciones, hemos de acordarnos de realizar un (reset) como he comentado anteriormente para que se carguen los hechos iniciales.

```
FuzzyCLIPS> (reset)
FuzzyCLIPS> (run)
Introduzca la velocidad en km/h ([-30 - 30])
-20
Introduzca la distancia en metros ([0 - 50])
5
Criterio-Maximo: La presion a aplicar es :8.035714285714287%
Criterio-Momento: La presion a aplicar es :9.745939166856426%
Media: La media de los criterios obtenidos es: 8.890826726285356%
FuzzyCLIPS>
```

En la ejecución anterior observamos que el sistema ha fusificado los valores crisp introducidos por consola como velocidad-relativa “alejando” y distancia “cerca” y ha ejecutado la regla R1 y la regla obtener_presion, esta última ha defusificado la variable difusa presion_freno obteniendo un valor medio de 8.89082… %.
A continuación observamos ejecuciones similares variando los parámetros de entrada:

```
FuzzyCLIPS> (reset)
FuzzyCLIPS> (run)
Introduzca la velocidad en km/h ([-30 - 30])
18
Introduzca la distancia en metros ([0 - 50])
25
Criterio-Maximo: La presion a aplicar es :65.0%
Criterio-Momento: La presion a aplicar es :65.0%
Media: La media de los criterios obtenidos es: 65.0%
FuzzyCLIPS>
```

Observamos que se lanza la regla R6 con distancia “medio” y velocidad-relativa “acercando” y el valor medio de presion_freno defusificado (mediante la regla obtener_presion) es 65 %.
Para comprobar que reglas se activan y ejecutan en la ejecución podemos indicarle a “Fuzzy-clips” que nos lo muestre por pantalla, por ejemplo para la ejecución anterior nos muestra lo siguiente:

```
FuzzyCLIPS> (reset)
FuzzyCLIPS> (run)
FIRE    1 preguntar_velocidad: f-2
Introduzca la velocidad en km/h ([-30 - 30])
18
FIRE    2 preguntar_distancia: f-1
Introduzca la distancia en metros ([0 - 50])
25
FIRE    3 R6: f-4,f-3
FIRE    4 obtener_presion: f-5
Criterio-Maximo: La presion a aplicar es :65.0%
Criterio-Momento: La presion a aplicar es :65.0%
Media: La media de los criterios obtenidos es: 65.0%
FuzzyCLIPS>
```

NOTA: Las reglas de interacción con el usuario (preguntar_x) se ejecutan siempre y las obvio en la explicación de la evaluación.

```
FuzzyCLIPS> (reset)
FuzzyCLIPS> (run)
Introduzca la velocidad en km/h ([-30 - 30])
27
Introduzca la distancia en metros ([0 - 50])
2
Criterio-Maximo: La presion a aplicar es :91.79166666666666%
Criterio-Momento: La presion a aplicar es :87.70478592568587%
Media: La media de los criterios obtenidos es: 89.74822629617626%
FuzzyCLIPS>
```

Observamos mediante las ejecuciones que el sistema es coherente y funciona correctamente, en esta última ejecución observamos que se ejecuta la regla R3, ya que se ha fusificado los valores distancia a “cerca” y velocidad-relativa a “acercando” y cuya defusificación de la variable difusa presion_freno es 89.74822… % en valor medio.

```
FuzzyCLIPS> (reset)
FuzzyCLIPS> (run)
Introduzca la velocidad en km/h ([-30 - 30])
18
Introduzca la distancia en metros ([0 - 50])
12
Criterio-Maximo: La presion a aplicar es :86.375%
Criterio-Momento: La presion a aplicar es :74.47755133159458%
Media: La media de los criterios obtenidos es: 80.42627566579729%
FuzzyCLIPS>
```

En esta ejecución la variable difusa distancia toma los valores “cerca” y “medio” ya que el valor crisp introducido pertenece a la función de pertenencia cerca y medio al mismo tiempo, la variable difusa velocidad-relativa toma el valor “acercando” por lo tanto se lanzan las reglas R3 y la regla R6 y por último obtener_presion, el resultado de la defusificación para la variable difusa presion_freno es 80.426275… %.

```
FuzzyCLIPS> (reset)
FuzzyCLIPS> (run)
Introduzca la velocidad en km/h ([-30 - 30])
6
Introduzca la distancia en metros ([0 - 50])
38
Criterio-Maximo: La presion a aplicar es :9.178571428571429%
Criterio-Momento: La presion a aplicar es :37.58884985476201%
Media: La media de los criterios obtenidos es: 23.38371064166672%
FuzzyCLIPS>
```

En esta última ejecución los valores crisp introducidos toman varios valores difusos para cada variable, es decir, la variable difusa velocidad-relativa toma los valores difusos “constante” y “acercando”, y la variable difusa distancia toma los valores difusos “medio” y “lejos”, esto hace que se activen y se ejecuten cuatro reglas que son R5,R6,R8 y R9 y por último se ejecuta la regla obtener_presion obteniendo como vemos el valor crisp de la defusificación 23.383710… % para la presión a aplicar.

Conclusión

Con este sistema acabamos de comprender completando lo visto en teoría que un sistema es muy habitual que no disponga de conocimiento preciso lo que nos obliga a trabajar con lógica difusa aplicando una serie de pasos que han quedado bien definidos y estructurados en la práctica que son la fusificación de valores crisp, la inferencia difusa y la defusificación de valores difusos a valores crisp. En esta práctica hemos trabajado con el lenguaje de programación “Fuzzy-clips” que es una modificación del lenguaje CLIPS que nos permite trabajar con lógica difusa. Los tipos de problemas que encontramos en la vida real tienden siempre a contener conocimiento impreciso ya que los humanos tomamos decisiones en la vida real de forma muy similar a como lo hace la lógica difusa esto nos obliga a estudiar estos tipos de sistema para poder ponerlos en práctica en un futuro.
En esta práctica hemos aprendido diferencias entre valores crisp y valores difusos de una variable difusa y en que consiste el proceso de fusificar y defusificar valores dentro de un sistema basado en lógica difusa.

Auto-crítica

La comprensión de un sistema basado en lógica difusa no me ha resultado trivial en un principio, ya que nunca habíamos trabajado con sistemas que emplean inferencia difusa y con conceptos como son “fusificar” y “defusificar” variables, los sistemas basado en reglas que habíamos visto siempre han sido (o hemos hecho que sean) con conocimiento preciso pero poco realista. El apartado que más tiempo he dedicado a comprender es el de defusificar variables difusas a variables crisp, ya que “Fuzzy-clips” lo hace internamente pero para ello primero genera una matriz de inferencia composicional y después una función de pertenencia de la cual se obtiene el valor crisp mediante dos estrategias como son “máximo” y “momento” (las hemos visto en la práctica).