Universidad Nacional de Cuyo - Facultad de Ingeniería Mecatrónica
Prof. Marchetta Martin
Alumnos: Pino, Jeremias ; Elorga Eliseo; Correa, Elias

Inteligencia Artificial II
2019

Trabajo Práctico II

1 Knowledge Base

Dominio del problema: "Evaluación y mantenimiento de una válvula de seguridad de una estación de reducción de presión de gas."(Ver fig.1)

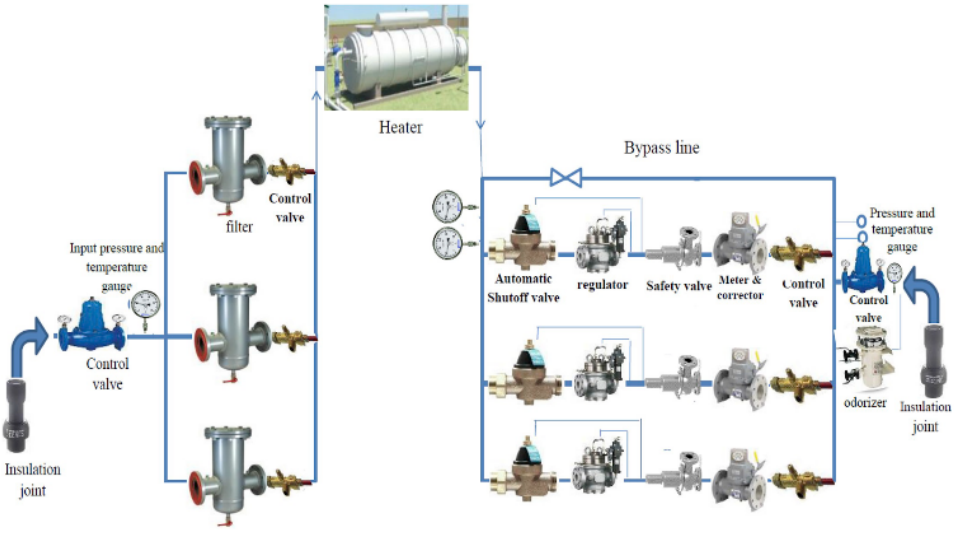

Figure 1: Estación de reducción de presión de gas

Para dicho problema desarrollamos una base de conocimiento(KB) en **Prolog**, en la cual implementamos preguntas cerradas del tipo "verdadero/falso", preguntas abiertas en las cuales la KB me indica que acción tomar y por último además de estas reglas axiomáticas incluimos algunos Ground Facts(hechos) donde modificamos la KB mediante el uso de predicados dinámicos.

1.1 Introduccion a Prolog

Prolog es un lenguaje de programación que se basa en el lenguaje de la Lógica de Primer Orden y que se utiliza para resolver problemas en los que entran en juego **objetos y relaciones** entre ellos. Una de las ventajas de la programación lógica es que se especifica qué se tiene que hacer (programación declarativa).

Además, es lenguaje, incluye algunos predicados predefinidos "meta-lógicos", ajenos al ámbito de la Lógica de Primer Orden, (var, nonvar, ==, ...), otros "extra-lógicos", que tienen un efecto lateral, (write, get, ...) y un tercer grupo que nos sirven para expresar información de control de como realizar alguna tarea (el corte, ...).

La Lógica de Primer Orden analiza las frases sencillas del lenguaje (fórmulas atómicas o elementales) separándolas en Términos y Predicados. Los términos hacen referencia

a los objetos que intervienen y los predicados a las propiedades o relaciones entre estos objetos.

1.2 Implementación

A continuación anexamos algunas implementaciones:

- Ejemplo de preguntas de carácter "abiertas" (Ej: ¿Qué debe hacerse ante determinada situación?).

Implementamos para "simular" lo que un operario observaría en la planta física con **procedimientos de manejo de listas** que incorpora el lenguaje. En este caso de tipo random.

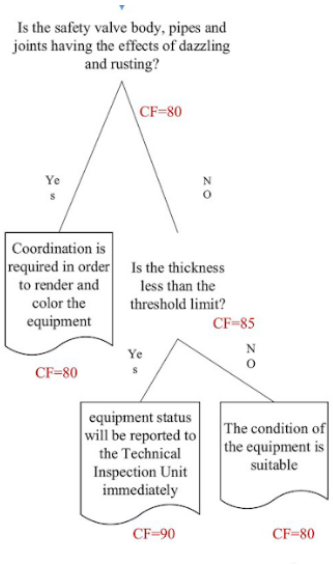

Figure 2: Rama 1 del problema general

Declaramos los predicados en donde los componentes del sistema corresponden al mismo "tipo" o "clase" denominada elementos(). (Ver fig. 3)

```
16  %Predicados
17  elementos(valvulaSeguridad).
18  elementos(juntas).
19  elementos(tuberias).
```

Figure 3: Declaración de predicados

Implementamos una lista con los todos los estados en los que se podía encontrar cualquiera de los elementos del sistema y seleccionamos alguno de ellos de manera random. (Ver fig.4)

```
31  %Consulto el estado de alguno de los componentes
32  consultarEstado(X,Y):- elementos(X), estado_random(Y).
33
```

Figure 4: Consulta de estado

Una vez especificado el estado en el que se encuentra, el operario de consultar que hacer en cada caso. (Ver fig.5)

```
43  valvulaSeguridadEsta(deslumbrante):- write('Se requiere coordinacion para renderizar y colorear el equipo'),!.
44  valvulaSeguridadEsta(oxidado):-write('Se requiere coordinacion para renderizar y colorear el equipo'),!.
45  valvulaSeguridadEsta(buenEstado):-write('Debe consultar su espesor!'),!.
```

Figure 5: Consecuentes

Si el estado de algún elemento correspondía a "buenEstado", se debía consultar por su espesor. Manteniendo la lógica que veníamos trabajando, generamos una lista de valores de espesores para seleccionar de manera random alguno de ellos. En caso de que se cumpliese o no las condiciones establecidas se le indica al operario como actuar. (Ver fig.6)

```
47  %Hacemos una lista de espesores. Para luego verficar los GroundFact
48  lista_espesores([27,28.5,27.6,30,29.5,28,30,29]).
49
50  %ConsultarEspesor(X,Y) :- el espesor de elemento X es Y
51  espesor_random(X) :- lista_espesores(L), random_member(X,L).
52  verificarProblema(E,X):- espesor(E,Y), X < Y, write('Reporte una inspeccion tecnica de inmediato!'),!.
53  verificarProblema(E,X):- espesor(E,Y), X==Y, write('La condicion del elemento es adecuada!').
54
55  consultarEspesor(E,X) :- elementos(E), espesor_random(X), verificarProblema(E,X).
56
```

Figure 6: Consulta de espesor

- Ejemplo de predicados dinámicos como Ground Facts. Modificaciones en la KB.

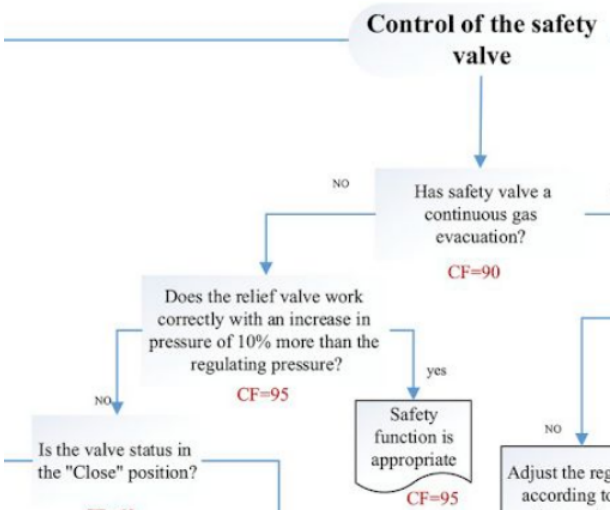

Figure 7: Rama central del problema general

Declaramos el predicado de la presión de tipo "dinámico", esto quiere decir que el operario va a poder modificar el valor del mismo, según lo requiera, para que Prolog pueda inferir como se debe proseguir en diferentes circunstancias. (Ver fig.8)

```
104  %GroundFact (de tipo dinámico)
105  :-dynamic(presion/1).    %Le declaro al compilador que va a ser de tipo dinamico
106  presion(30).
```

Figure 8: Predicado dinámico

Definimos una lista booleana de dos valores "si/no" para eventualmente usarla como emular preguntas de carácter cerrado. (Ver fig.9)

```
62    %Lista booleana para el estado Pierde/No pierde
63    list_bool([si,no]).
64
65    bool_random(X):-list_bool(L), random_member(X,L).
66
```

Figure 9: Lista booleana

Vemos que a Prolog podemos preguntarle si hay evacuación continua de gas. En donde si la respuesta aleatoria es "no", se indica que debe consultarse si funciona la válvula de alivio para 10 % mas de la presión regular. (Ver fig.10)

```
119  verificaEvacuacion(X):- X==si, write('Por favor, consulte si la presion de la linea de gas es la apropiada!'),!.
120  verificaEvacuacion(X):- X==no, write('Por favor, consulte si funciona la valvula de alivio con un 10% de aumento de la presion regular',!).

     consultarEvacuacionContinuaGas(X):- bool_random(X),verificaEvacuacion(X).
```

Figure 10: Consulta de evacuación de gas

Es aquí donde se "simula" la lectura de un sensor, es decir, el aumento adicional de presión viene dado por la lectura de un sensor. Como carecemos del mismo, el incremento lo puede ingresar el usuario por pantalla.(Ver fig.11)

```
126   calculaPresion(Adicional,Z):- presion(Y), Z is Y + Y*(Adicional/100).
127
128   adicionaPresion:- write('Cual es el adicional (en %): '),read(Adicional),
129                     calculaPresion(Adicional,Z),
130                     %Cambio el GroundFact  ("SIMULO" variable global)
131                     retract(presion(Anterior)),
132                     assert(presion(Z)).
```

Figure 11: Adición de presión

- Por último, agregamos un screenshot de todas las preguntas que podemos hacerle a la KB.

```
consultarArregloPerdida          consultarFugaEvitable          consultarPresionGas              consultarSensorControl
consultarEspesor                 consultarFuncionamientoPilot   consultarPrevencionAsiento_Orificio  consultarSensorControlPresion
consultarEstado                  consultarPerdidaGas            consultarResorteSeguridad        consultarValvulaAlivio
consultarEvacuacionContinuaGas   consultarPosicionValvula       consultarResorteValvula
```

Figure 12: Consultas

2 Fuzzy Logic

Dominio del problema: Sistema de inferencia difusa para controlar un péndulo invertido. (Ver fig.13)

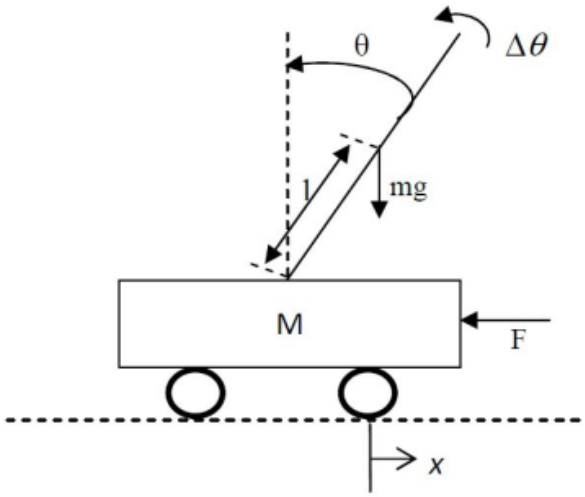

Figure 13: Diagrama del problema

El donde el modelo físico del problema es: (Ver fig.14)

$$\ddot{\theta} = \frac{g\sin\theta + \cos\theta\left(\dfrac{-F - ml\dot{\theta}^2 Sin\theta}{M+m}\right)}{l\left(\dfrac{4}{3} - \dfrac{m\cos^2\theta}{M+m}\right)}$$

$$\theta' = \theta' + \theta''\Delta t$$

$$\theta = \theta + \theta'\Delta t + (\theta''\Delta t^2)/2$$

Figure 14: Modelo cinemático

2.1 Introducción

La lógica difusa es un conjunto de principios matemáticos basados en grados de membresía o pertenencia, cuya función es modelar información. Este modelado se hace con base en reglas lingüísticas que aproximan una función mediante la relación de entradas y salidas del sistema (composición). Esta lógica presenta rangos de membresía dentro de un intervalo entre 0 y 1, a diferencia de la lógica convencional, en la que el rango se limita a dos valores: el cero o el uno.

Ventajas del uso de lógica difusa en los sistemas de control:

- Diseño de control simplificado para modelos complejos.
- Autonomía.
- Adaptabilidad.
- En el caso del control difuso, no es necesario un modelo matemático de la planta.

2.2 Implementación

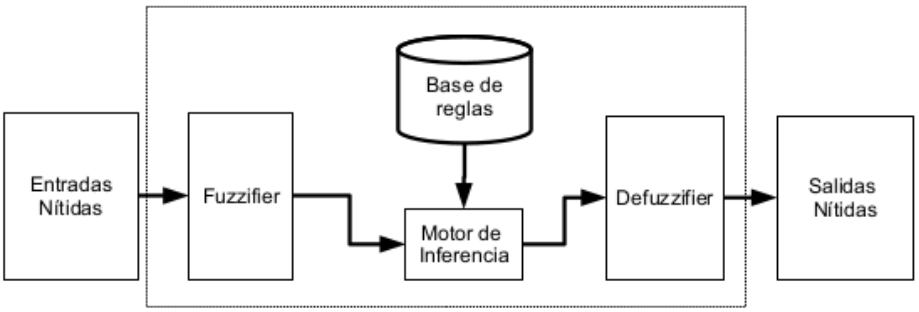

Figure 15: Estructura de un controlador difuso

Basándonos en el esquema de la fig.(15) las entradas nítidas a nuestro controlador difuso fueron las posición, velocidad y aceleración angular del sistema. De las cuales por practicidad como variables lingüísticas solo trabajamos con la posición y la velocidad ya que nos fueron suficientes para definir una base de reglas para poder inferir la fuerza.

Se consideraron cinco particiones borrosas **"MN, N, Z, P, MP"**, para las variables lingüísticas de entrada. Ver fig.(16), (17).

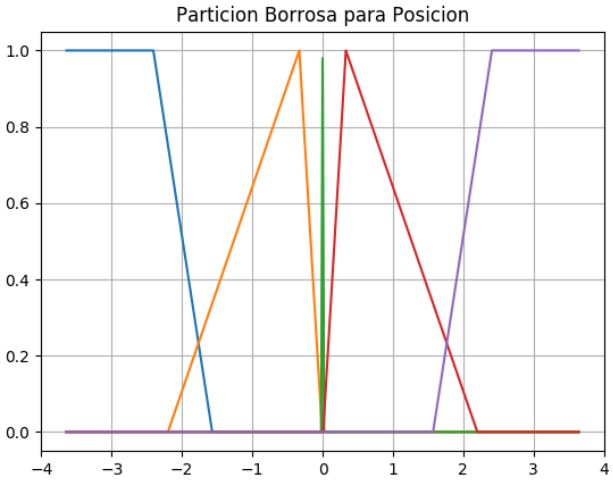

Figure 16: Particiones borrosas posición

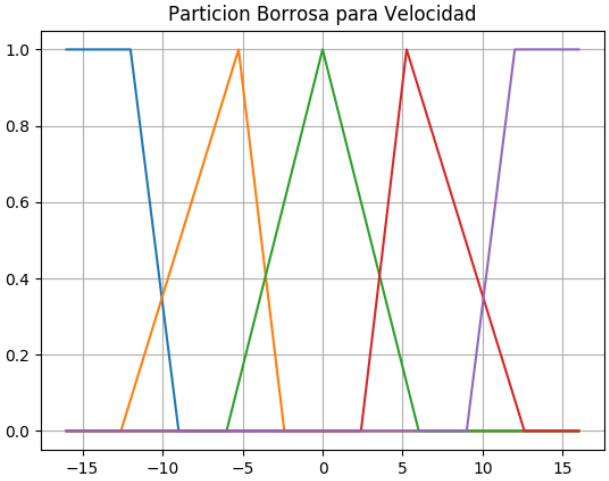

Figure 17: Particiones borrosas velocidad

Para confeccionar estos conjuntos borrosos, se utilizó como función de membresía (pertenencia), una función triangular. Ver fig.(18)

```
def membership(x, x_, x_med=0, fin=0):
    """
    Funcion usada para calcular en grado de pertenencia a un conjunto borroso
    :param x: absisas, universo del discurso
    :param x_: vector, extremos del soporte
    :param x_med: punto medio (punto para el cual la pertenencia es maxima (1))
    :param fin: variable para saber si queda en 1 o 0 el valor del conjunto borroso
    :return y: ordenadas o grado de pertenencia
    """
    y = [0, 1]  # Valor minimo y maximo de pertenencia
    if x_med == 0:
        x_med = (x_[0] + x_[1]) / 2
    if x_[0] <= x <= x_med:  # p_1 = [x_i, 0] , p_2 = [x_med, 1]
        y_ = (y[1] - y[0]) * (x - x_[0]) / (x_med - x_[0]) + y[0]
    elif x_med < x <= x_[1]:  # p_1 = [x_med, 1] , p_2 = [x_f, 0]
        y_ = (y[0] - y[1]) * (x - x_med) / (x_[1] - x_med) + y[1]
    else:
        y_ = 0

    if fin == -1 and x < x_med:
        y_ = 1
    elif fin == 1 and x > x_med:
        y_ = 1

    return y_
```

Figure 18: Membership function

Se efectuó en pequeño sesgo de las particiones "N" y "P" hacia "Z", tanto en la posición como en la velocidad.

Una vez borrosificados los valores de entrada, definimos la base de regla necesaria para poder inferir la salida que, en este caso es la fuerza. Ver tabla (1)

		Velocidad				
		MN	N	Z	P	MP
Posicion	MN	MP	MP	MP	MP	MP
	N	MN	MN	N	N	N
	Z	MN	N	Z	P	MP
	P	P	P	P	MP	MP
	MP	MN	MN	MN	MN	MN

Table 1: Base de reglas

Calculamos la pertenencia de los antecedentes mediante la Norma-T (o intersección). Ver fig.(19)

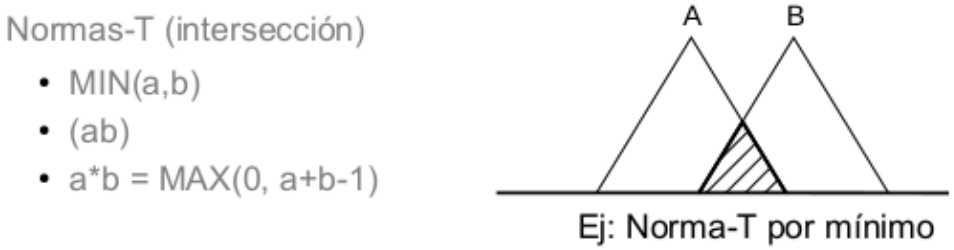

Figure 19: Norma-T

Combinando los antecedentes y resolviendo la implicación, obtenemos el único conjunto de salida **Fuerza**. Ver fig.(20)

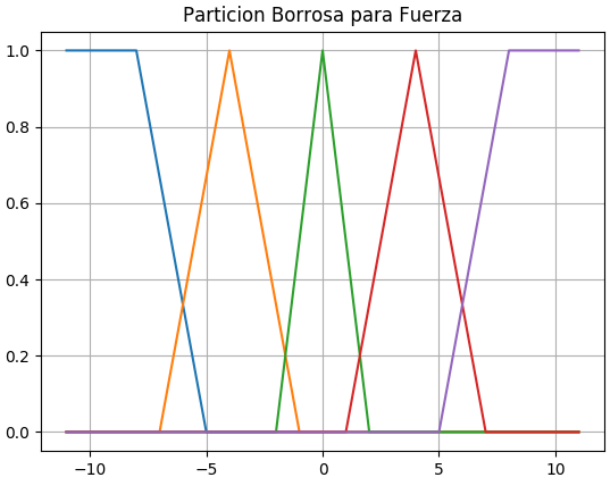

Figure 20: Conjunto de salida

La desborrosificación de este conjunto la efectuamos mediante el método de (Weighted Average). Ver fig.(21)

```python
def desborrosificador(f):
    """
    Desborrosificador por media de centros (Weighted Average)
    :param f: Son los valores del conjunto de salida fuerza
    :return num / den: devuelve el valor nitido
    """
    s_f = [[-11, -5], [-7, -1], [-2, 2], [1, 7], [5, 11]]

    num = 0
    den = 0
    for i, y in enumerate(f):
        med = (s_f[i][1] + s_f[i][0]) / 2
        num += y * med
        den += y

    return num / den
```

Figure 21: Desborrosificador

Mostramos en la fig.(22) la respuesta del sistemas frente a distintos valores de posición inicial $\{\frac{\pi}{2}, \frac{\pi}{4}, \frac{\pi}{6}, \frac{\pi}{8}\}$

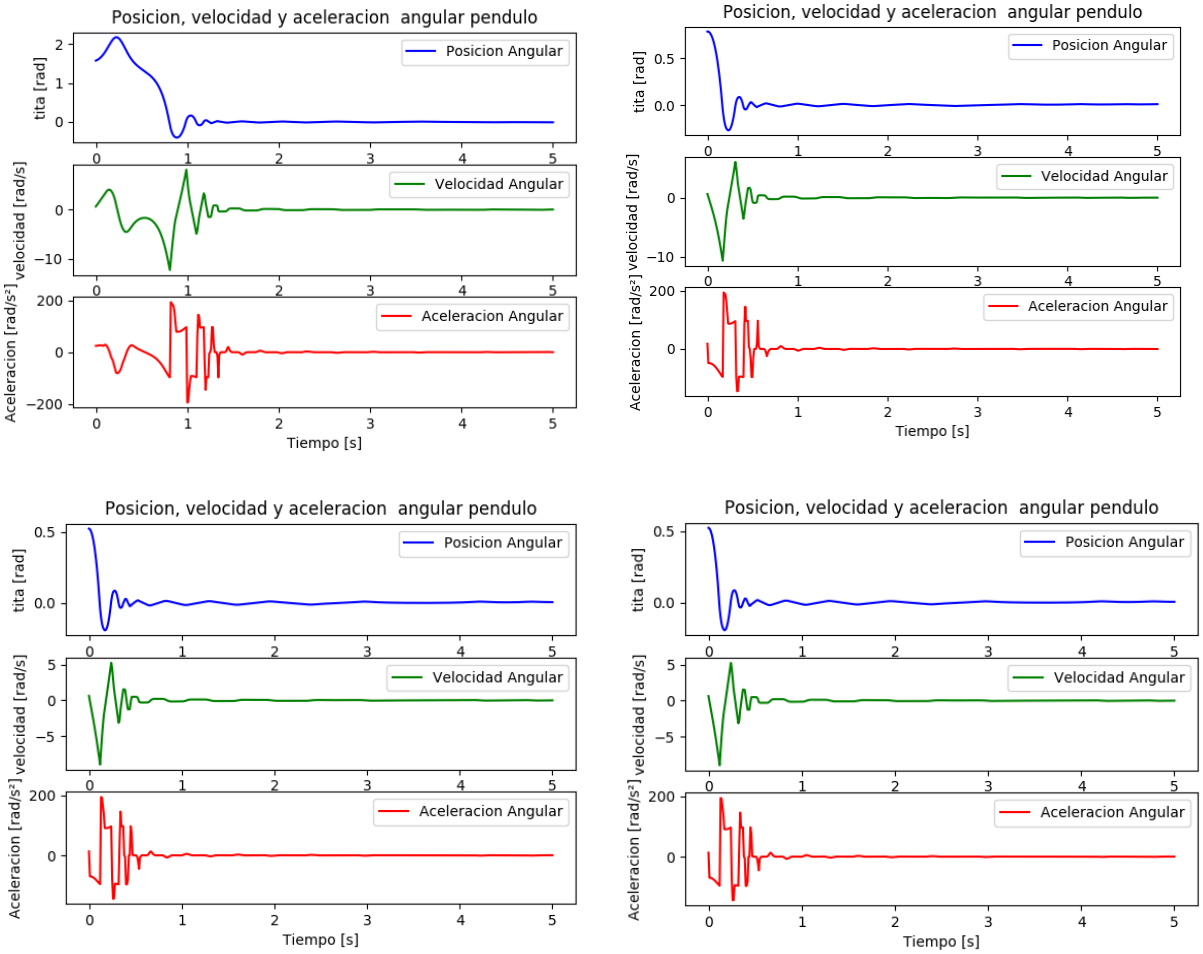

Figure 22: Respuesta frente a diferentes condiciones iniciales

3 Planificador - Fast Downward

3.1 Modelado en PDDL del dominio: Transporte aéreo de cargas

Para el problema del transporte aéreo se definió el dominio transporte_aereo(ver fig.23). En este se definen los predicados CARGO, PLANE, AIRPORT, AT y IN y las acciones load, unload y fly.

El problema transporte_aereo_problem presentado (ver fig.24). En este se definen las cargas C1 y C2, los aviones airplane1 y ariplane 2 y por ultimo las ciudad SFO y JFK.

En el estado inicial se definen los objetos con los hechos CARGO , PLANE y AIRPORT y ademas se instancia la C1 en SFO y la C2 en JFK.

Como objetivo se plantea el intercambio de los paquetes.

Dominio:

```
(define (domain transporte_aereo)
  (:predicates
              (CARGO ?x)
              (PLANE ?x)
              (AIRPORT ?x)
              (AT ?x ?y)
              (IN ?x ?y)
  )
  (:action load
     :parameters (?x ?y ?z)
     :precondition (and (AT ?x ?z) (AT ?y ?z) (CARGO ?x) (PLANE ?y) (AIRPORT ?z) )
     :effect (and (not (AT ?x ?z)) (IN ?x ?y))
  )
  (:action unload
     :parameters (?x ?y ?z)
     :precondition (and (IN ?x ?y) (AT ?y ?z) (CARGO ?x) (PLANE ?y) (AIRPORT ?z) )
     :effect (and (AT ?x ?z) (not (IN ?x ?y)) )
  )
  (:action fly
     :parameters (?p ?from ?to)
     :precondition (and (AT ?p ?from) (PLANE ?p) (AIRPORT ?from) (AIRPORT ?to) )
     :effect (and (not(AT ?p ?from)) (AT ?p ?to) )
  )

)
```

Figure 23: Dominio

Tareas:

```
(define (problem transporte_aereo_problem)
(:domain transporte_aereo)
(:objects C1 C2
          airplane1 airplane2
          SFO JFK)
(:init (CARGO C1) (CARGO C2)
       (PLANE airplane1) (PLANE airplane2)
       (AIRPORT SFO) (AIRPORT JFK)
       (AT C1 SFO) (AT C2 JFK) (AT airplane1 SFO) (AT airplane2 JFK))
(:goal (and (AT C1 JFK) (AT C2 SFO)))
)
```

Figure 24: Tareas

3.2 Modelado en PDDL del dominio del problema B.

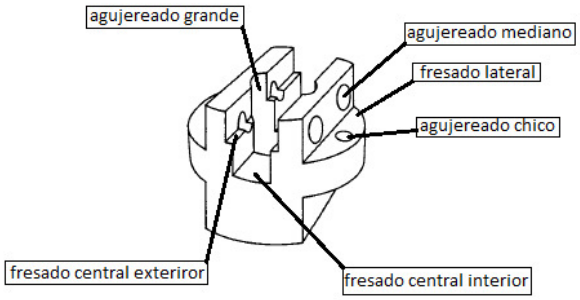

Figure 25: Ilustracion de la pieza

El dominio del problema define los tipos de datos. A diferencia de la implementación del problema del transporte aéreo en este no se necesitaran los hechos del tipo CARGO, etc. Esto ya viene dado por la sentencia types.

Los predicados son:
En : indica si un objeto esta en una maquina
Aplicado : este es el hecho que verifica que se realizo la operación en la pieza
Posición V o H : indica la posición de la pieza
Tamaño_herr : indica el tamaño de la herramienta que se va a utilizar
No_usada o Usada : indican si una maquina ya tiene una herramienta o esta libre.
Las acciones son:
colocar / quitar - herramienta : coloca o quita la herramienta de la máquina
colocar / quitar - pieza : coloca o quita la pieza de la maquina
parar / acostar - pieza : cambia de posición la pieza
fresar : define la función de fresar
agujerear : define la acción de agujerear

Para el problema se define como estado inicial a las 2 maquinas libres y todas las herramientas en el armario. Como objetivo se plantea la realización de todas las acciones.

Después de ejecutar el programa se obtuvo el siguiente plan:
Actual search time: $0.0816042[s]; t = 0.0857961[s]$

colocar_pieza_en_pieza1 fresa1 armario1 (1)
colocar_herramienta_en_ herr_f1 fresa1 armario1 (1)
fresar_central_interior_pieza pieza1 fresa1 herr_f1 fr_ce_int v ch (1)
quitar_herramienta_de_ herr_f1 fresa1 armario1 (1)
colocar_herramienta_en_ herr_f2 fresa1 armario1 (1)
fresar_central_exterior_pieza pieza1 fresa1 herr_f2 fr_ce_ex v med (1)
quitar_herramienta_de_ herr_f2 fresa1 armario1 (1)
colocar_herramienta_en_ herr_f3 fresa1 armario1 (1)
fresar_lateral_pieza pieza1 fresa1 herr_f3 fr_lat v gra (1)
colocar_pieza_en_ pieza1 taladro1 fresa1 (1)
colocar_herramienta_en_ broca1 taladro1 armario1 (1)

agujerear_c_pieza pieza1 taladro1 broca1 ag_ch v ch (1)
quitar_herramienta_de_ broca1 taladro1 armario1 (1)
colocar_herramienta_en_ broca3 taladro1 armario1 (1)
agujerear_g_pieza pieza1 taladro1 broca3 ag_gr v gra (1)
quitar_herramienta_de_ broca3 taladro1 armario1 (1)
colocar_herramienta_en_ broca2 taladro1 armario1 (1)
colocar_pieza_en_ pieza1 armario1 taladro1 (1)
acostar_pieza pieza1 armario1 v hor (1)
colocar_pieza_en_ pieza1 taladro1 armario1 (1)
agujerear_m_pieza pieza1 taladro1 broca2 ag_med hor med (1)