UNCUYO
UNIVERSIDAD NACIONAL DE CUYO

FACULTAD DE INGENIERÍA

Inteligencia Artificial II

U2: Razonamiento

Grupo N°2

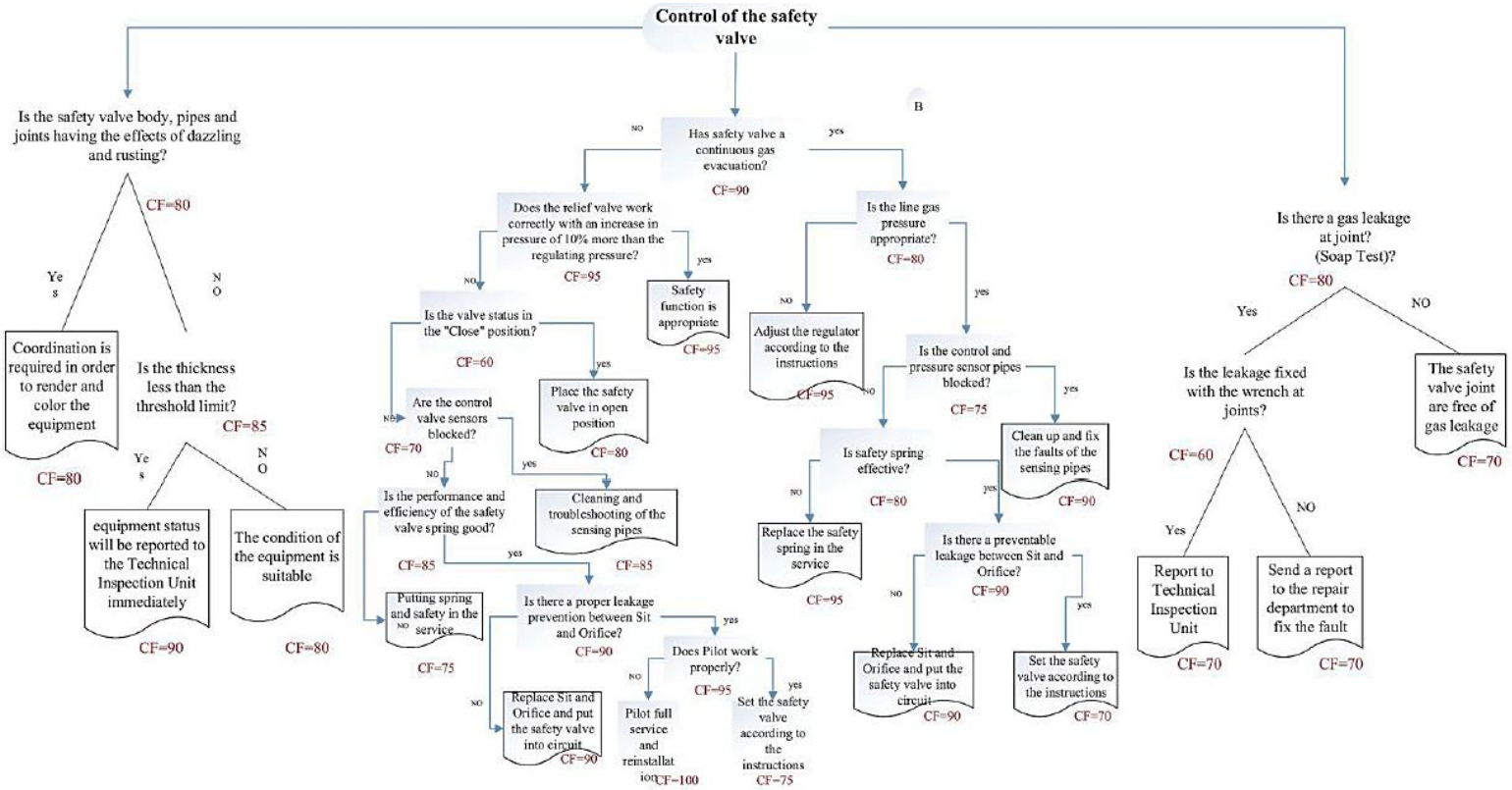
Alumno:	**Legajo:**
Cantú, Maximiliano	11294
Costarelli, Agustín	10966
Lage Tejo, Joaquín	11495

2 0 2 0

Trabajo Práctico Nº 2
Razonamiento

Ejercicio 1 – Agentes Basados en el Conocimiento:

Se desarrolló una base de conocimiento en PROLOG para un sistema de evaluación y mantenimiento de una válvula de seguridad en una estación de reducción de presión de gas. La misma responde al siguiente esquema simplificado de decisión:



El algoritmo se basa en el "estado" de las distintas partes que componen el sistema de seguridad de la central. Dicho estado responde a una pregunta binaria (Verdadero o Falso) asignando "Yes" o "No" al valor del estado, o "Desconocido" en caso que no se posea información sobre esa pieza/componente en particular. En este último caso, y en caso de ser necesario dicho dato para la evaluación de un predicado, el sistema pide la verificación del estado. Se agregaron como Ground Facts, distintos estados a los componentes, de forma aleatoria para probar las distintas combinaciones posibles de funcionamiento.

Los axiomas del dominio del problema, que componen la base de conocimiento, se construyeron respetando el funcionamiento del algoritmo de búsqueda de PROLOG. En este caso es una búsqueda en profundidad hacia atrás, es decir que partiendo de una pregunta o

consulta, verifica los predicados (en nuestro caso estados) en la KB que unifiquen al predicado que necesita evaluar. Cuando logra encontrar un estado coincidente, toma una decisión, que puede terminar el proceso de búsqueda (si un paso anterior no posee un estado "Desconocido"), o propagarse la búsqueda "aguas arriba" del árbol de búsqueda, pudiendo llegar a la raíz del árbol.

El programa puede consultarse de distintas formas. Es posible acceder a un menú de opciones con la instancia "menu." por consola. Se puede pedir un diagnóstico completo sin acceder al menú, ingresando "diagnosticar.". Estas dos opciones están pensadas para el personal de mantenimiento, devolviendo oraciones que conformarían el plan de mantenimiento a realizar.

Por otro lado, se pueden consultar los estados específicos de los componentes con "estado(piloto, X)." (consultando si el piloto funciona correctamente o no). Esta última opción requiere conocer la codificación de los componentes en el código, por lo que puede dejarse para usuarios más interiorizados. Por último, es posible obtener solo la información de qué componentes deben revisarse para comprobar su estado, con "estado(X, desconocido)." y obtenemos todas las opciones al ingresar ";" una detrás de la otra.

A continuación se explicitan los componentes (como fueron ingresados en el código) y la respuesta a la cual responde su estado:

- thickness_less_than_the_threshold_limit : ¿El espesor de la válvula es menor al umbral mínimo?
- effects_of_dazzling_and_rusting : ¿Hay efectos de deslumbre u oxidación en la válvula, las tuberías o uniones del sistema?
- piloto : ¿El piloto funciona correctamente?
- leakage_prevention_between_sit_and_orifice : ¿Existe una correcta fuga de prevención entre asiento-orificio?
- safety_valve_spring : ¿Es correcto y eficiente el funcionamiento del resorte de la válvula de seguridad?
- control_valve_sensors_blocked : ¿Están bloqueados los sensores de la válvula de control?
- valve_status_closed : ¿El estado de la válvula está en "Cerrada"?
- relief_valve_ok_with_10_percent_more_pressure : ¿La válvula de alivio funciona correctamente al aumentar un 10% la presión sobre la presión nominal de regulación?
- safety_valve_has_continuous_evacuation : ¿Hay una evacuación continua en la válvula de seguridad?
- preventable_leakage_between_sit_and_orifice : ¿Hay una fuga prevenible entre asiento-orificio?
- safety_spring_effective : ¿Es efectivo el resorte de seguridad?
- control_and_pressure_sensor_pipes_blocked : ¿Las tuberías de control y sensado de presión están bloqueadas?

- apropiate_line_gas_pressure : ¿Es correcta la presión de la línea de gas?
- gas_leakage_at_joint : ¿Hay una fuga de gas en la unión (realizar prueba con jabón)?
- leakage_fixed_with_the_wrench_at_joints : ¿Está fijada la fuga de gas en la unión, con una llave?

Comentarios y propuestas

A modo de futuras mejoras (que no pudimos desarrollar por errores de configuración dentro del editor, en este caso Visual Studio Code), se plantea obtener el estado de los componentes del sistema de forma externa al programa. Por ejemplo, podríamos extraer los datos de un informe en formato ".txt", en cada ejecución, que actualice los estados de la central. Otra opción posible sería obtener estos datos directamente de un sensado en los distintos componentes, integrando el programa con otros (sensores en Arduino, Python en Raspberry Pi, o de ser posible obtener el sensado directamente de un PLC industrial) que se encarguen de la parte física del sensado.

Sobre todo proponemos extraer el "Plan de Mantenimiento" en el mismo formato, en un archivo externo que pueda ser distribuido, impreso, etc. Para mayor facilidad del personal. Así como también llevar un registro de los mantenimientos realizados.

Ejercicio 2 – Planning:

En este ejercicio, se utilizó el planificador Fast Downward para modelar, en PDDL, el dominio de transporte aéreo de cargas y un sistema de armado de paquetes.

Sistema de transporte aéreo descargas

Para el caso del sistema de transporte, tomamos como base el archivo de ejemplo y desarrollamos algunas modificaciones para así ir probando distintas funcionalidades del lenguaje y del código implementado.

El dominio del problema es similar al planteado originalmente, pero con algunas diferencias.

El cambio se puede ver en el problema concretamente, en el cual, primero optamos por realizarlo con 30 cargas, 6 aeropuertos y 3 aviones, pero al correrlo resultaba que el costo computacional era muy alto y no se lograba obtener una solución. Dado esto decidimos disminuir la cantidad de aviones a uno solo, para así corroborar que el dominio y el problema funcionen correctamente y luego poder hacer las modificaciones deseadas para exigir un poco más al código.

A continuación, adjuntamos los resultados obtenido con un solo avión:

(cargar carga29 gol01 sp)
(cargar carga28 gol01 sp)
(cargar carga27 gol01 sp)
(cargar carga26 gol01 sp)
(volar gol01 sp fra)
(cargar carga25 gol01 fra)
(cargar carga24 gol01 fra)
(cargar carga23 gol01 fra)
(cargar carga22 gol01 fra)
(cargar carga21 gol01 fra)
(volar gol01 fra eze)
(descargar carga29 gol01 eze)
(descargar carga25 gol01 eze)
(cargar carga12 gol01 eze)
(cargar carga11 gol01 eze)
(cargar carga10 gol01 eze)
(volar gol01 eze hnd)
(descargar carga27 gol01 hnd)
(descargar carga22 gol01 hnd)
(descargar carga11 gol01 hnd)
(cargar carga07 gol01 hnd)
(cargar carga06 gol01 hnd)
(cargar carga05 gol01 hnd)
(volar gol01 hnd lys)
(descargar carga26 gol01 lys)
(descargar carga23 gol01 lys)
(cargar carga16 gol01 lys)
(cargar carga15 gol01 lys)
(cargar carga14 gol01 lys)
(descargar carga10 gol01 lys)
(descargar carga07 gol01 lys)
(volar gol01 lys nrt)
(descargar carga28 gol01 nrt)
(descargar carga21 gol01 nrt)
(descargar carga16 gol01 nrt)
(descargar carga12 gol01 nrt)
(cargar carga04 gol01 nrt)
(cargar carga02 gol01 nrt)
(cargar carga01 gol01 nrt)
(volar gol01 nrt eze)
(descargar carga15 gol01 eze)
(descargar carga04 gol01 eze)
(volar gol01 eze fra)
(descargar carga14 gol01 fra)
(descargar carga06 gol01 fra)
(descargar carga02 gol01 fra)
(volar gol01 fra sp)
(descargar carga24 gol01 sp)
(descargar carga05 gol01 sp)
(descargar carga01 gol01 sp)
; cost = 50 (unit cost)

[t=48.0563s, 42832 KB] Plan length: 50 step(s).
[t=48.0563s, 42832 KB] Plan cost: 50
[t=48.0563s, 42832 KB] Expanded 104236 state(s).
[t=48.0563s, 42832 KB] Reopened 0 state(s).
[t=48.0563s, 42832 KB] Evaluated 701675 state(s).
[t=48.0563s, 42832 KB] Evaluations: 701675
[t=48.0563s, 42832 KB] Generated 1320029 state(s).
[t=48.0563s, 42832 KB] Dead ends: 0 state(s).
[t=48.0563s, 42832 KB] Expanded until last jump: 104192 state(s).
[t=48.0563s, 42832 KB] Reopened until last jump: 0 state(s).
[t=48.0563s, 42832 KB] Evaluated until last jump: 701120 state(s).
[t=48.0563s, 42832 KB] Generated until last jump: 1319360 state(s).
[t=48.0563s, 42832 KB] Number of registered states: 701675
[t=48.0563s, 42832 KB] Int hash set load factor: 701675/1048576 = 0.669169
[t=48.0563s, 42832 KB] Int hash set resizes: 20
[t=48.0563s, 42832 KB] Search time: 48.0538s
[t=48.0563s, 42832 KB] Total time: 48.0563s
Solution found.
Peak memory: 42832 KB

Al obtener buenos resultados computacionales y a nivel de código, decidimos incrementar la cantidad de aviones a 2, y obtuvimos los siguientes resultados:

(cargar carga29 gol01 sp)
(cargar carga28 gol01 sp)
(cargar carga27 gol01 sp)
(cargar carga26 gol01 sp)
(volar gol01 sp fra)
(cargar carga25 gol01 fra)
(cargar carga24 gol01 fra)
(cargar carga23 gol01 fra)
(cargar carga22 gol01 fra)
(cargar carga21 gol01 fra)
(volar gol01 fra eze)
(descargar carga29 gol01 eze)
(descargar carga25 gol01 eze)
(cargar carga12 gol01 eze)
(cargar carga11 gol01 eze)
(cargar carga10 gol01 eze)
(volar gol01 eze hnd)
(descargar carga27 gol01 hnd)
(descargar carga22 gol01 hnd)
(descargar carga11 gol01 hnd)
(cargar carga07 gol01 hnd)
(cargar carga06 gol01 hnd)
(cargar carga05 gol01 hnd)
(volar gol01 hnd lys)
(descargar carga26 gol01 lys)
(descargar carga23 gol01 lys)
(cargar carga16 gol01 lys)
(cargar carga15 gol01 lys)
(cargar carga14 gol01 lys)
(descargar carga10 gol01 lys)
(descargar carga07 gol01 lys)
(volar gol01 lys nrt)
(descargar carga28 gol01 nrt)
(descargar carga21 gol01 nrt)
(descargar carga16 gol01 nrt)
(descargar carga12 gol01 nrt)
(cargar carga04 gol01 nrt)
(cargar carga02 gol01 nrt)
(cargar carga01 gol01 nrt)
(volar gol01 nrt eze)
(descargar carga15 gol01 eze)
(descargar carga04 gol01 eze)
(volar gol01 eze fra)
(descargar carga14 gol01 fra)
(descargar carga06 gol01 fra)
(descargar carga02 gol01 fra)
(volar gol01 fra sp)
(descargar carga24 gol01 sp)
(descargar carga05 gol01 sp)
(descargar carga01 gol01 sp)
; cost = 50 (unit cost)

[t=48.0563s, 42832 KB] Plan length: 50 step(s).
[t=48.0563s, 42832 KB] Plan cost: 50
[t=48.0563s, 42832 KB] Expanded 104236 state(s).
[t=48.0563s, 42832 KB] Reopened 0 state(s).
[t=48.0563s, 42832 KB] Evaluated 701675 state(s).
[t=48.0563s, 42832 KB] Evaluations: 701675
[t=48.0563s, 42832 KB] Generated 1320029 state(s).
[t=48.0563s, 42832 KB] Dead ends: 0 state(s).
[t=48.0563s, 42832 KB] Expanded until last jump: 104192 state(s).
[t=48.0563s, 42832 KB] Reopened until last jump: 0 state(s).
[t=48.0563s, 42832 KB] Evaluated until last jump: 701120 state(s).
[t=48.0563s, 42832 KB] Generated until last jump: 1319360 state(s).
[t=48.0563s, 42832 KB] Number of registered states: 701675
[t=48.0563s, 42832 KB] Int hash set load factor: 701675/1048576 = 0.669169
[t=48.0563s, 42832 KB] Int hash set resizes: 20
[t=48.0563s, 42832 KB] Search time: 48.0538s
[t=48.0563s, 42832 KB] Total time: 48.0563s
Solution found.
Peak memory: 42832 KB

En ambos casos, cada “step” tiene un valor unitario, por lo tanto, el costo total será igual a la cantidad de “steps” total realizados por el algoritmo para lograr los objetivos planteados.

Analizando los valores obtenidos en cada solución, podemos notar que, con 2 aviones, los nodos expandidos y el tiempo total, incrementan su valor significativamente en comparación con los valores de 1 avión. Dado esto, podemos entender porque al realizarlo con 3 aviones, se hace difícil poder lograr los objetivos en poco tiempo y porque aumentan significativamente los recursos computacionales.

En conclusión, es posible afirmar que el código planteado funciona correctamente con una gran cantidad de cargas y varios aeropuertos, ya que no necesita reabrir nodos, no encuentra caminos sin salida y el tiempo empleado en lograr los objetivos es relativamente bajo.

Armado automatizado de pedidos

En este inciso, optamos por la opción de realizar un dominio a elección nuestra, ya que resultaba un desafío pensar una práctica útil en la cual implementar este tipo de soluciones.

La elección del dominio está basada en un brazo robótico que está situado en la línea de despacho de pedidos de una empresa que fabrica y distribuye cremas directamente al consumidor, por lo tanto, dicho robot es el encargado de armar los paquetes con los pedidos realizados por los clientes, para que luego sean embalados correctamente y pasar a reparto. Por ahora, la empresa vende 4 tipos de cremas: crema corporal, crema de día, crema de noche y crema exfoliante; las cuales se encuentran situadas en diferentes estantes de acuerdo con el tipo de crema al que corresponden, es decir, estante de cremas corporales, estante de cremas de día, etc.

La tarea específica del robot es armar el pedido buscando las cremas que corresponden a ese pedido en los estantes correspondientes y colocarlas en una caja. Por ejemplo, el pedido 1 consta de: 1 crema corporal, 1 crema de día y 1 crema de noche. En la práctica el robot arma varios pedidos a la vez, por lo tanto, no es necesario que termine un pedido para comenzar el otro, lo que hace es ir armando los pedidos solicitados, en el menor tiempo posible y con el menor costo posible.

El dominio del problema cuenta con 2 acciones, la primera es “agarrar” las cremas del estante adecuado, y la segunda “colocar” las cremas en el paquete o caja conforme al número de pedido. Para la acción de “agarrar”, colocamos como efecto que la crema buscada se encuentre en la pinza o mano del robot, para así determinar que ya no se encuentra en el

estante. Y luego para la acción de “colocar”, continuando con la acción anterior, ingresamos como precondición que la crema se encuentra en la pinza, para así estar seguros de que fue recogida del estante.

En el problema implementado el robot debe armar 5 pedidos en 5 paquetes distintos:

Paquete 1:

- 1 crema corporal
- 1 crema de día
- 1 crema exfoliante

Paquete 2:

- 1 crema de día
- 1 crema de noche
- 1 crema exfoliante

Paquete 3:

- 1 crema corporal
- 1 crema de día
- 1 crema de noche
- 1 crema exfoliante

Paquete 4:

- 1 crema de día
- 1 crema de noche

Paquete 5:

- 1 crema exfoliante

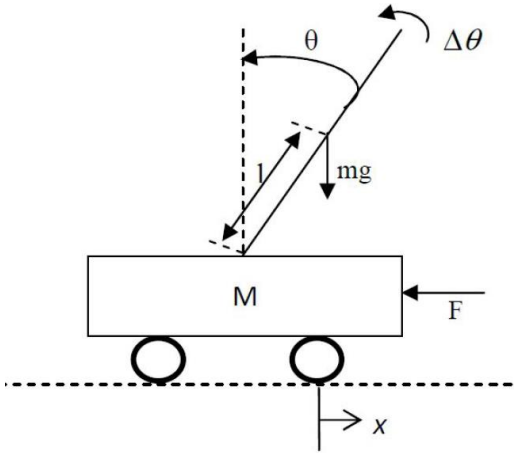
Luego de ejecutar el programa, obtuvimos estos resultados:

```
Nodes generated during search: 170
Nodes expanded during search: 26
Plan found with cost: 26
BFS search completed
0.00100: (agarrar crema_dia estante_dia ok)
0.00200: (colocar crema_dia paquete1 ok)
0.00300: (agarrar crema_exfoliante estante_exfoliante ok)
0.00400: (colocar crema_exfoliante paquete1 ok)
0.00500: (agarrar crema_dia estante_dia ok)
0.00600: (colocar crema_dia paquete2 ok)
0.00700: (agarrar crema_noche estante_noche ok)
0.00800: (colocar crema_noche paquete2 ok)
0.00900: (agarrar crema_exfoliante estante_exfoliante ok)
0.01000: (colocar crema_exfoliante paquete2 ok)
```

0.01100: (agarrar crema_corporal estante_corporal ok)
0.01200: (colocar crema_corporal paquete1 ok)
0.01300: (agarrar crema_dia estante_dia ok)
0.01400: (colocar crema_dia paquete3 ok)
0.01500: (agarrar crema_noche estante_noche ok)
0.01600: (colocar crema_noche paquete3 ok)
0.01700: (agarrar crema_exfoliante estante_exfoliante ok)
0.01800: (colocar crema_exfoliante paquete3 ok)
0.01900: (agarrar crema_corporal estante_corporal ok)
0.02000: (colocar crema_corporal paquete3 ok)
0.02100: (agarrar crema_dia estante_dia ok)
0.02200: (colocar crema_dia paquete4 ok)
0.02300: (agarrar crema_noche estante_noche ok)
0.02400: (colocar crema_noche paquete4 ok)
0.02500: (agarrar crema_exfoliante estante_exfoliante ok)
0.02600: (colocar crema_exfoliante paquete5 ok)
Planner found 1 plan(s) in 1.359secs.

Ejercicio 3 – Lógica Difusa

Se implementó un sistema de inferencia difusa para controlar un péndulo invertido montado sobre un carro que se mueve en una sola dirección; considerando que el mismo no tiene un espacio restringido de movimiento, y que responde al siguiente modelo planteado:



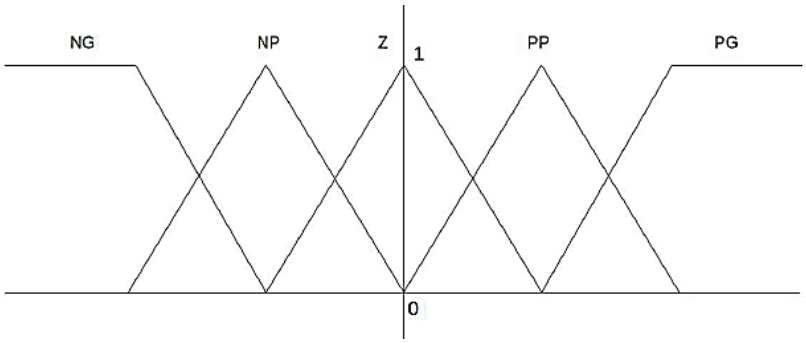
$$\theta' = \theta' + \theta''\Delta t$$

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

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

Las variables de entrada para nuestro problema son la posición (theta) y velocidad angular (theta prima). Nuestra salida es la Fuerza que hay que aplicar para alterar la posición del péndulo.

Se adoptaron para cada uno de los dominios 5 conjuntos borrosos NG, NP, Z, PP, PG. Las funciones de pertenencia adoptadas fueron en todos los casos triangulares y con hombro derecho e izquierdo en los extremos. Se decidió que el solapamiento de cada conjunto borroso fuera el máximo, es decir del 50%. El borrosificador adoptado es del tipo "singleton".



Habiendo definido como se vería nuestro sistema difuso procedimos a armar la FAM del mismo:

Variables lingüísticas de entrada:

theta={NG,NP,Z,PP,PG}

theta_prima={NG,NP,Z,PP,PG}

Variable lingüística de salida:

fuerza={NG,NP,Z,PP,PG}

THETA'\THETA	NG	NP	Z	PP	PG
NG	PG	PG	PG	PP	Z
NP	PG	PG	PG	Z	NG
Z	PG	PP	Z	NP	NG
PP	PP	Z	NP	NG	NG
PG	Z	PG	NG	NG	NG

Dado que para el armado de la FAM se necesitaría el conocimiento experto nosotros actuamos en ese rol usando nuestro criterio y ayudándonos de información obtenida de distintos trabajos[1] que existen en internet sobre lógica difusa y péndulo invertido.

Con la FAM podemos elaborar 25 reglas de inferencia borrosa de Mamdani, las cuales son:

- R1: If theta NG is **and** theta_prima is NG THEN fuerza is PG
- R2: If theta NG is **and** theta_prima is NP THEN fuerza is PG
- R3: If theta NG is **and** theta_prima is Z THEN fuerza is PG
- R4: If theta NG is **and** theta_prima is PP THEN fuerza is PP
- R5: If theta NG is **and** theta_prima is PG THEN fuerza is Z

[1] *" Lógica Difusa Aplicada al Control Local del Péndulo Invertido con Rueda de Reacción"* (https://dialnet.unirioja.es/descarga/articulo/4606949.pdf)

- R6: If theta NP is **and** theta_prima is NG THEN fuerza is PG
- R7: If theta NP is **and** theta_prima is NP THEN fuerza is PG
- R8: If theta NP is **and** theta_prima is Z THEN fuerza is PP
- R9: If theta NP is **and** theta_prima is PP THEN fuerza is Z
- R10: If theta NP a is **and** theta_prima PG is THEN fuerza is PG

- R11: If theta Z is **and** theta_prima is NG THEN fuerza is PG
- R12: If theta Z is **and** theta_prima is NP THEN fuerza is PG
- R13: If theta Z is **and** theta_prima is Z THEN fuerza is Z
- R14: If theta Z is **and** theta_prima is PP THEN fuerza is NP
- R15: If theta Z is **and** theta_prima is PG THEN fuerza is NG

- R16: If theta PP is **and** theta_prima is NG THEN fuerza is PP
- R17: If theta PP is **and** theta_prima is NP THEN fuerza is Z
- R18: If theta PP is **and** theta_prima is Z THEN fuerza is NP
- R19: If theta PP is **and** theta_prima is PP THEN fuerza is NG
- R20: If theta PP is **and** theta_prima is PG THEN fuerza is NG

- R21: If theta PG is **and** theta_prima is NG THEN fuerza is Z
- R22: If theta PG is **and** theta_prima is NP THEN fuerza is NG
- R23: If theta PG is **and** theta_prima is Z THEN fuerza is NG
- R24: If theta PG is **and** theta_prima is PP THEN fuerza is NG
- R25: If theta PG is **and** theta_prima is PG THEN fuerza is NG

Por último se adoptó como método para desborrosificar la media por centro dado que es muy efectivo a un bajo costo computacional.

$$y = \frac{\sum_{l=1}^{l=M} \bar{y}^l \mu_{B'}(\bar{y}^l)}{\sum \mu_{B'}(\bar{y}^l)}$$

donde $\bar{y}^l$ es el punto máximo de B', y *M* es la cantidad de conjuntos borrosos de salida

Ya con toda esa información el procedimiento del algoritmo desarrollado es el siguiente.

Primero ingresar los valores de theta y theta prima para ser borrosificados y obtener su valor de pertenencia, en nuestro caso dado que teníamos un solapamiento del 50% en muchos de los casos las variables iban a tener valor de pertenencia para dos conjuntos difusos. Esto nos implica que cada vez que ingresáramos un valor de theta y theta_prima, íbamos a tener que aplicar de 1 a 4 de las 25 reglas. Un caso de una regla sería tener valor extremos en los hombros que nos darían valores de pertenencia a un solo conjunto borroso, tanto para theta y theta_prima.

En estas condiciones se puede dar que tuviéramos reglas con mismo consecuente o distinto consecuente.

En caso de que dos reglas tuvieran el mismo consecuente de fuerza se calcula el máximo valor de pertenencia entre ellas para obtener un solo consecuente. Si no tuviéramos más reglas que estas dos cuando calculemos la media de centros nuestro valor resultante al desborrosificar sería el correspondiente al centro de ese conjunto.

En caso de que tengamos solamente reglas con distinto consecuente lo que se hace es la disyunción entre ambos conjuntos borrosos para obtener un solo conjunto borroso con los máximos de pertenencia de cada uno para luego aplicar la media de centro.

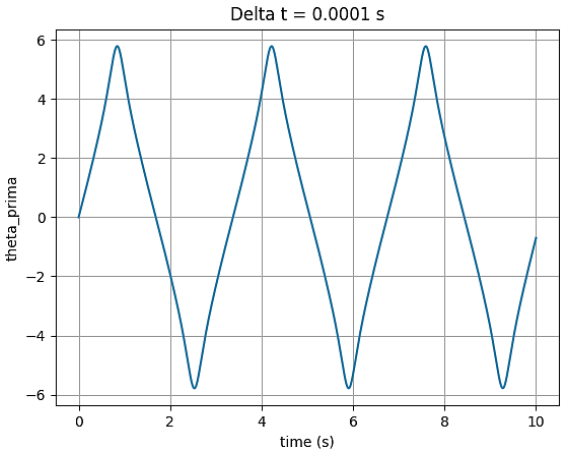
Estos dos casos nos indican que aplicando lógica a las 25 reglas se pueden agrupar todas las que tuvieran el mismo consecuente para obtener solo 5 reglas. En nuestro caso el algoritmo trabaja con las 25 reglas y a medida que se recorren se aplica la lógica del caso de que haya reglas con el mismo consecuente.

Ya sea cualquiera de los dos casos con el valor de fuerza obtenido por media de centros se ingresaba en la simulación y se repite el procedimiento con los nuevos valores obtenidos de theta y theta_prima.

Resultados de nuestro algoritmo

Para calcular el dominio de las velocidades se observó entre qué valores oscila el péndulo en vacío, dado estos valores se definió el dominio (-6,6).

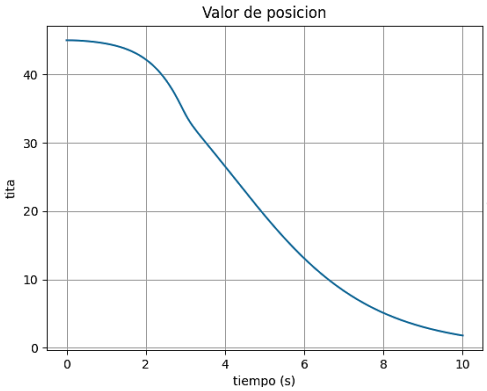
Para el caso de theta, siendo que nuestro ángulo 0º es la posición de equilibrio nuestro universo nos permite como máximo dar dominio entre (-180, 180), pero consideramos que el carro está apoyado en el suelo, pasar por debajo de la horizontal implicaría que la barra golpee el suelo. Por lo tanto, definimos el dominio de trabajo entre (90,90).

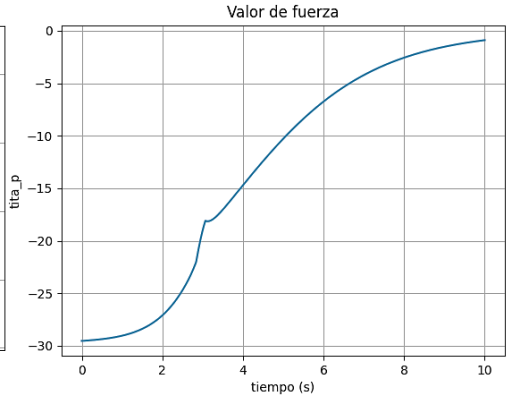


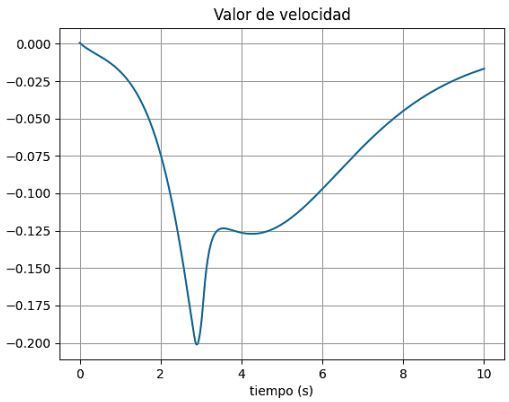
Ahora presentaremos múltiples casos para determinar el funcionamiento de nuestro sistema.

Primero se observaron los casos de soltar el péndulo desde la posición de 45° sin velocidad inicial y se fueron variando el dominio de las fuerzas para ver tiempos de demora para llegar al equilibrio.

- Primer caso

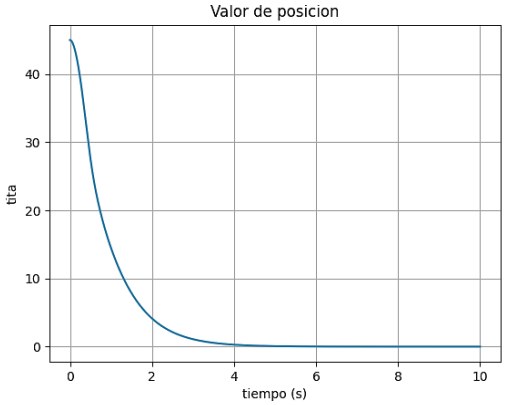


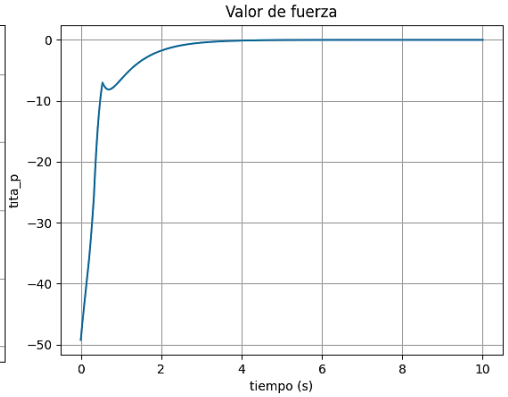


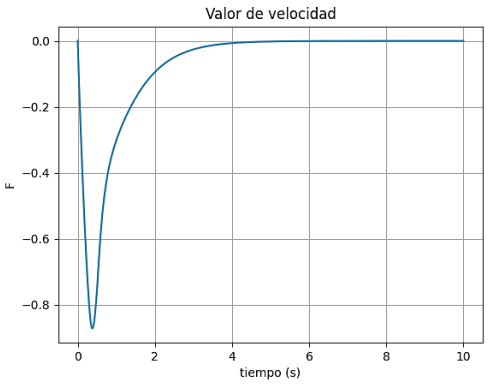


Comenzamos con un dominio de fuerza pequeño (-60,60). Podemos ver que el sistema parte de su posición de 45° y tarda un tiempo considerable en estabilizarse. La fuerza llega a un valor máximo de -30 y el péndulo no presenta una aceleración considerable.

- Segundo caso

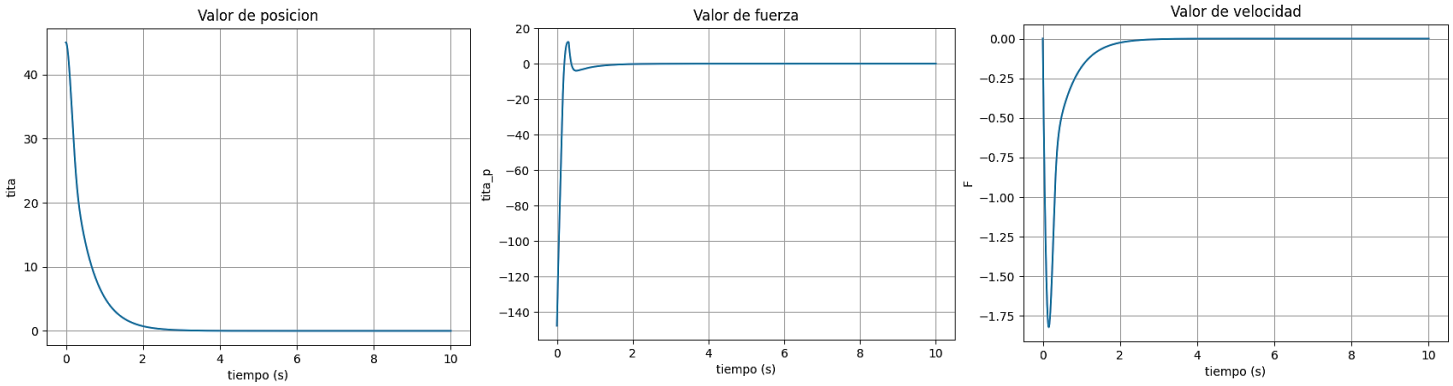






Para el caso de tener un dominio de fuerza (-100,100) el sistema se estabiliza después de 4 segundos y la fuerza aplicada máxima alcanza el valor de -50. En ningún momento el péndulo se aleja más de la posición de la cual fue soltado, por lo tanto, podemos aceptar que la solución es aplicable.

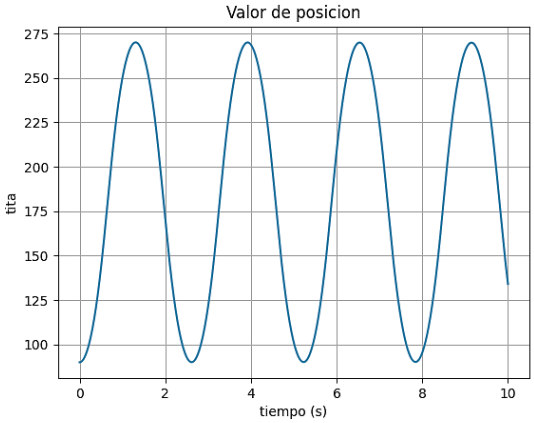
➢ Tercer caso



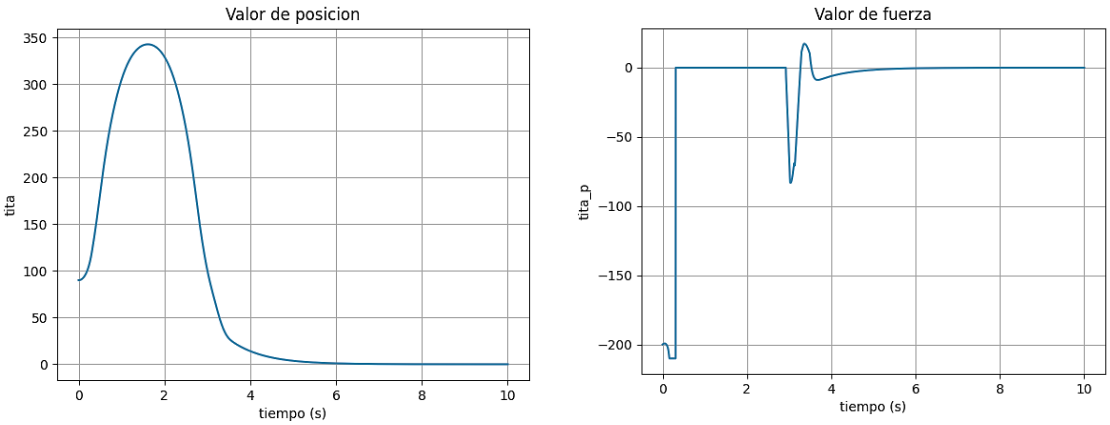
En este tenemos un dominio de fuerzas de (-300,300), la estabilización se logró en poco más de 2 segundos. Es un tiempo bastante bueno, pero estamos aplicando una fuerza de 150 y el péndulo experimenta una velocidad mayor.

➢ Ahora vamos a analizar qué pasa si soltamos el péndulo en el límite de nuestro dominio de posición, 90°:

Este comportamiento se debe a que al trabajar en el límite, nos salimos del dominio de trabajo de nuestro algoritmo de lógica difusa. Por lo cual es el equivalente a no haber hecho nada sobre el péndulo.



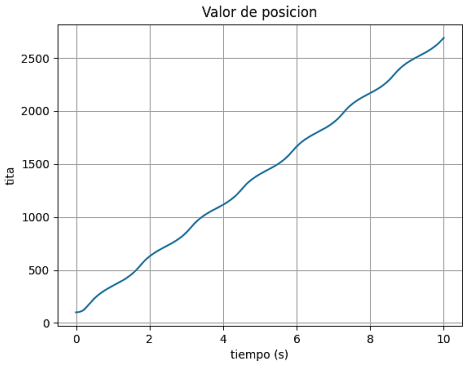
➢ Pero si al caso anterior le modificamos un poco el dominio de theta para que pase de (-90,90) a ser de (-120,120):



Como podemos ver al haber modificado un poco el dominio comenzamos con un pico de fuerza que no logra estabilizar el sistema y luego se tiene un momento en el cual no puede

actuar dado que solo trabaja en (-120,120), pero cuando la oscilación vuelve al dominio de trabajo el sistema logra llevarlo al valor de 0. En este caso la construcción física del péndulo tendría que permitir la oscilación por debajo de la línea horizontal.

- Caso en el cual el dominio es de (-160,160) y soltamos el péndulo en 100°:



Esta grafica nos indica que el sistema está en condiciones de trabajo que no puede actuar como deseamos. Se aplica una fuerza al comienzo intentando llevar al equilibrio pero se sale rápidamente del dominio y el péndulo comienza a girar en una sola dirección.

Para los casos en que damos una velocidad inicial distinta de 0 lo que se logra es que el sistema va a tener una perturbación extra, lo importante es que esta no saque al sistema fuera de los dominios de trabajo definidos de lo contrario no lograremos la convergencia del péndulo a 0.

Conclusiones

La aplicación de lógica difusa para este caso no implicó un gran desarrollo de programación hoy en día incluso se podría simplificar en gran medida la programación usando Matlab que cuenta con librerías de lógica difusa.

La verdadera dificultad se encuentra en lograr analizar el problema real, saber elegir qué criterios de borrosificación aplicar, generar reglas de Mamdani que reflejen nuestro problema, en este punto se nota la importancia de trabajar con experto o ver la necesidad de invertir un buen tiempo en la experimentación. Definir reglas adecuadas es la diferencia entre que nuestro problema se resuelva o no. Una vez definido todo es momento de elegir los dominios de cada una de nuestras variables, en este punto entra el juego las limitaciones de la vida real. Ver que costo implica cada resultado, los tiempos para estabilizar pueden que se necesiten en un cierto margen, por cuestiones constructivas a lo mejor no se pueden superar valores de velocidad para no comprometer la estructura. El método para aplicar la fuerza puede limitarnos a ciertos valores y también esa aplicación de fuerza puede comprometer la estructura. Nuestro criterio como ingenieros no solo implica reconocer las buenas aplicaciones de lógica difusa sino saber minimizar los tiempos de implementación con un buen armado de base de conocimiento, también saber maximizar los resultados adaptándose a la disponibilidad de recursos en la realidad.