ROBÓTICA
TP_Final Compilador

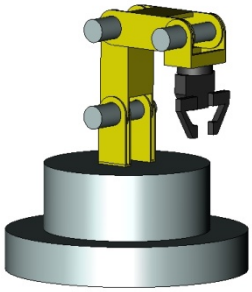
Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

GRUPO 1

Alumnos:
Hernán Muñoz
Fabián Zanella

PROFESOR:
Ing. GIANNETTA, Hernán

JTP:
Ing. GRANZELLA, Damián



FECHA DE ENTREGA:
16/07/2011

Índice

1. **Introducción……3**
2. **Marco teórico ……4**
3. **Características del compilador……..……..…12**
4. **Ejemplo código generado en matlab por compilador.....16**
5. **Conclusiones……18**

Introducción

A través de este trabajo expondremos el desarrollo de un compilador el cual es complemento para el trabajo realizado por los tps anteriores. Tanto la parte cinemático, como la dinámica de nuestro robot m5 el cual va a generar un código para su visualización en matlab paso por paso atraves del tool de robot de matlab.

En este trabajo se desarrolla programación en el generador antlr a través del cual creamos nuestro propio compilador y es visualizado el resultado a través de matlab.

Marco teórico

1. Introducción a la cinemática de un robot:

La cinemática del robot estudia el movimiento del mismo con respecto a un sistema de referencia. Así, la cinemática se interesa por la descripción analítica del movimiento espacial del robot como una función del tiempo, y en particular por las relaciones entre la posición y la orientación del extremo final del robot con los valores que toman sus coordenadas articulares. Existen dos problemas fundamentales para resolver la cinemática del robot, el primero de ellos se conoce como el problema cinemático directo, y consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot, el segundo denominado problema cinemático inverso resuelve la configuración que debe adoptar el robot para una posición y orientación del extremo conocidas.

Denavit y Hartenberg propusieron un método sistemático para descubrir y representar la geometría espacial de los elementos de una cadena cinemática, y en particular de un robot, con respecto a un sistema de referencia fijo. Este método utiliza una matriz de transformación homogénea para descubrir la relación espacial entre dos elementos rígidos adyacentes, reduciéndose el problema cinematico directo a encontrar una matriz de transformación homogénea 4 X 4 que relacione la localización espacial del robot con respecto al sistema de coordenadas de su base.

Diagrama entre cinematica directa e inversa.

	Cinemática directa ->->	
Valor de las coordenadas Articulares (q0, q1, ... qn)		posición y orientación del extremo del robot (x, y, z, α, ß, γ)
	<-<- Cinemática inversa	

Con base a estos ejes de referencia y los datos obtenidos de las medidas del robot m6 concluimos los parámetros de Denavit-Hartenberg:

articulación	θ	D	a mueve en x	α
	(Angulo de xi-1 a xi)	(distancia de xi-1 a xi)	(distancia de zt a zt+1)	(Angulo de zt a zt+1)
1	θ1	20,33	0	90
1_2	–90	0	0	-90
2	θ2+90	12.7	0	90
3	θ3–90	0	0	-90
4	θ4+90	0	12.7	90
4_5	90	0	0	90
5	θ5	14	0	0

Obtenida la tabla podemos deducir las matrices de cada movimiento para hallar la ecuación cinemática final:

a01 =

[cos(q1), 0, sin(q1), 0]
[sin(q1), 0, -cos(q1), 0]
[0, 1, 0, 2033/100]
[0, 0, 0, 1]

A12 =

[cos(q1 - 90), 0, -sin(q1 - 90), 0]
[sin(q1 - 90), 0, cos(q1 - 90), 0]
[0, -1, 0, 0]
[0, 0, 0, 1]

A23=

[cos(q2), 0, sin(q2), 0]
[sin(q2), 0, -cos(q2), 0]
[0, 1, 0, 127/10]
[0, 0, 0, 1]

A34=

[cos(q3 - 90), 0, -sin(q3 - 90), 0]
[sin(q3 - 90), 0, cos(q3 - 90), 0]
[0, -1, 0, 0]
[0, 0, 0, 1]

A45=

[cos(q4 + 90), 0, sin(q4 + 90), (127*cos(q4 + 90))/10]
[sin(q4 + 90), 0, -cos(q4 + 90), (127*sin(q4 + 90))/10]
[0, 1, 0, 0]
[0, 0, 0, 1]

A56=

[cos(q5 + 180), 0, sin(q5 + 180), 0]
[sin(q5 + 180), 0, -cos(q5 + 180), 0]
[0, 1, 0, 0]
[0, 0, 0, 1]

A67=

[cos(q1), -sin(q1), 0, 0]
[sin(q1), cos(q1), 0, 0]
[0, 0, 1, l5]
[0, 0, 0, 1]

atotals = A01*A12*A23*A34*A45*A56*A67=

atotals =

[cos(q1)*(sin(q5 + 180)*(sin(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) - cos(q3 - 90)*sin(q1 - 90)*cos(q1)) - cos(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)) + sin(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2)))) - sin(q1)*(sin(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)) - cos(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2))), - cos(q1)*(sin(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)) - cos(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2))) - sin(q1)*(sin(q5 + 180)*(sin(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) - cos(q3 - 90)*sin(q1 - 90)*cos(q1)) - cos(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)) + sin(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2)))), - cos(q5 + 180)*(sin(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) - cos(q3 - 90)*sin(q1 - 90)*cos(q1)) - sin(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)) + sin(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2)))), - (127*cos(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)))/10 - (127*sin(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2)))/10 - (127*sin(q1 - 90)*cos(q1))/10 - l5*(cos(q5 + 180)*(sin(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) - cos(q3 - 90)*sin(q1 - 90)*cos(q1)) + sin(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(sin(q1)*sin(q2) - cos(q1 - 90)*cos(q1)*cos(q2)) + sin(q1 - 90)*sin(q3 - 90)*cos(q1)) + sin(q4 + 90)*(cos(q2)*sin(q1) + cos(q1 - 90)*cos(q1)*sin(q2))))]
[sin(q1)*(sin(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)) - cos(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2))) - cos(q1)*(sin(q5 + 180)*(sin(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) + cos(q3 - 90)*sin(q1 - 90)*sin(q1)) - cos(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)) + sin(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2)))), cos(q1)*(sin(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)) - cos(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2))) + sin(q1)*(sin(q5 + 180)*(sin(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) + cos(q3 - 90)*sin(q1 - 90)*sin(q1)) - cos(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)) + sin(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2)))), cos(q5 + 180)*(sin(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) + cos(q3 - 90)*sin(q1 - 90)*sin(q1)) + sin(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)) + sin(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2))), (127*cos(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)))/10 + (127*sin(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2)))/10 - (127*sin(q1 - 90)*sin(q1))/10 + l5*(cos(q5 + 180)*(sin(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) + cos(q3 - 90)*sin(q1 - 90)*sin(q1)) + sin(q5 + 180)*(cos(q4 + 90)*(cos(q3 - 90)*(cos(q1)*sin(q2) + cos(q1 - 90)*cos(q2)*sin(q1)) - sin(q1 - 90)*sin(q3 - 90)*sin(q1)) + sin(q4 + 90)*(cos(q1)*cos(q2) - cos(q1 - 90)*sin(q1)*sin(q2))))]
[
cos(q1)*(cos(q5 + 180)*(cos(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)) - sin(q1 - 90)*sin(q4 + 90)*sin(q2)) + sin(q5 + 180)*(cos(q1 - 90)*cos(q3 - 90) - sin(q1 - 90)*sin(q3 - 90)*cos(q2))) + sin(q1)*(sin(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)) + cos(q4 + 90)*sin(q1 - 90)*sin(q2)),
cos(q1)*(sin(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)) + cos(q4 + 90)*sin(q1 - 90)*sin(q2)) - sin(q1)*(cos(q5 + 180)*(cos(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)) - sin(q1 - 90)*sin(q4 + 90)*sin(q2)) + sin(q5 + 180)*(cos(q1

- 90)*cos(q3 - 90) - sin(q1 - 90)*sin(q3 - 90)*cos(q2))),
sin(q5 + 180)*(cos(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)) - sin(q1 - 90)*sin(q4 + 90)*sin(q2)) - cos(q5 + 180)*(cos(q1 - 90)*cos(q3 - 90) - sin(q1 - 90)*sin(q3 - 90)*cos(q2)),
(127*cos(q1 - 90))/10 + (127*cos(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)))/10 + l5*(sin(q5 + 180)*(cos(q4 + 90)*(cos(q1 - 90)*sin(q3 - 90) + cos(q3 - 90)*sin(q1 - 90)*cos(q2)) - sin(q1 - 90)*sin(q4 + 90)*sin(q2)) - cos(q5 + 180)*(cos(q1 - 90)*cos(q3 - 90) - sin(q1 - 90)*sin(q3 - 90)*cos(q2))) - (127*sin(q1 - 90)*sin(q4 + 90)*sin(q2))/10 + 2033/100]
[0,0,0,1]

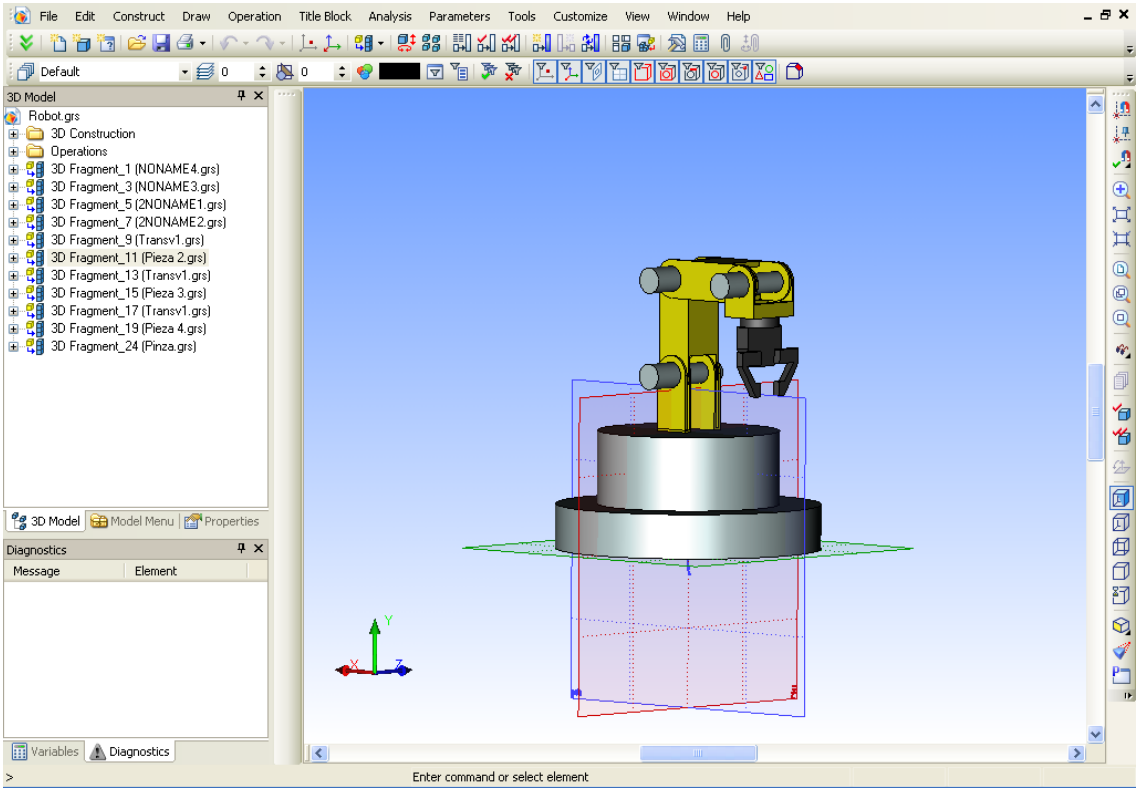
2. Introducción a la dinámica de un robot

El modelo dinámico de un robot nos permite conocer la relación existente entre el movimiento y las fuerzas actuantes que provocan dicho movimiento.
A diferencia del modelo cinemático que solo representa el movimiento con respecto a un sistema de referencia, el modelo dinámico tiene en cuenta las fuerzas y pares aplicadas en las articulaciones y además los parámetros dimensionales del robot como la longitud, masa e inercia de sus elementos.
Desde un punto de vista matemático, se relaciona la locación del robot (que está definida por sus variables articulares o por las coordenadas de localización de su extremo), y sus derivadas: velocidad y aceleración.
Un punto a tener en cuenta, es que la complejidad del modelo dinámico se acrecienta en gran medida con el aumento de grados de libertad (GDL).
El estudio dinámico del robot tiene como resultado un conjunto de ecuaciones matemáticas que describen la conducta dinámica del manipulador.
Tales ecuaciones de movimiento son útiles para simulación por PC, diseño de ecuaciones de control apropiadas para el robot y la evaluación del diseño y estructura cinemática del robot.
Debido al ya comentado aumento de complejidad a medida que crece el número de GDL no siempre es posible hallar una solución cerrada, es decir, un conjunto de ecuaciones diferenciales que al ser integradas nos den como resultado las fuerzas y torques a ser aplicadas para obtener un determinado desplazamiento con una cierta velocidad y aceleración, es decir, un único resultado.
Así, el modelo dinámico se debe resolver mediante cálculo numérico e iterativo (por Ej. Uso de series convergentes).
Cada eslabón de la cadena se trata como rígido y de masa concentrada, se usa el centro de gravedad.
Existen 2 formas básicas de encarar el problema del modelo dinámico:

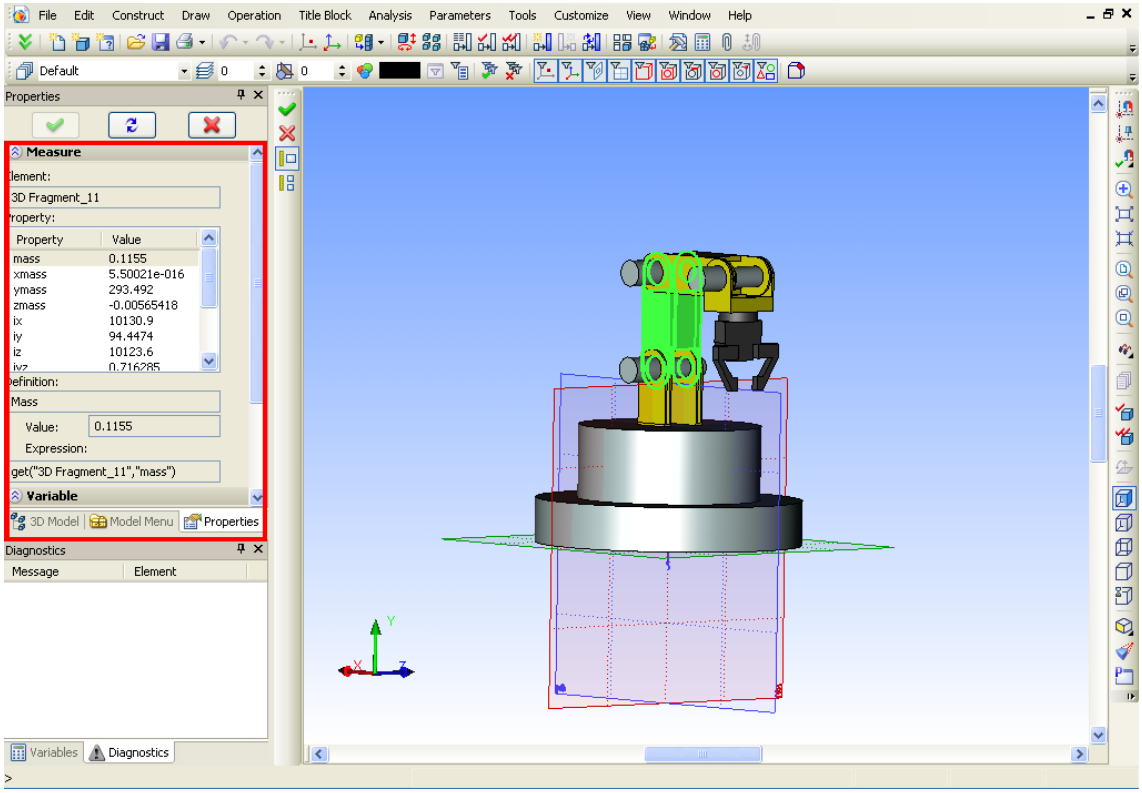
Modelo Dinámico Directo: Expresa la evolución temporal de las coordenadas articulares del robot en función de las fuerzas y pares que intervienen.

Modelo Dinámico Inverso: Expresa las fuerzas y pares que intervienen en función de la evolución de las coordenadas articulares y sus derivadas.
Al hacer el análisis, se debe tener en cuenta las siguientes fuerzas intervinientes:
Inercia, gravedad, coriolis y centrípeta.
Existen distintos métodos para resolver el problema:
- Lagrange-Euler
- Newton-Euler
- Variables de estado
- Kane

Se realizó el modelo del robot M5 en el software T-Flex:



Para ahorrar el cálculo de los centros de masa, se determinará que los mismos se encuentran en las articulaciones de las piezas del robot. De la misma manera, no se tendrán en cuenta los momentos de inercia. El procedimiento para determinar los parámetros correspondientes a la masa de las articulaciones del robot es el siguiente: Una vez realizado el modelado del robot con el software T-Flex, se procederá a hacer click derecho sobre el nombre de la pieza a medir en la ventana “3D Model” a la izquierda de la pantalla y se seleccionará la opción “Measure”. La misma desplegará una barra en donde figuran la masa, centro de masa, momentos de inercia, etc.



De esta manera, se obtienen las masas de las articulaciones:

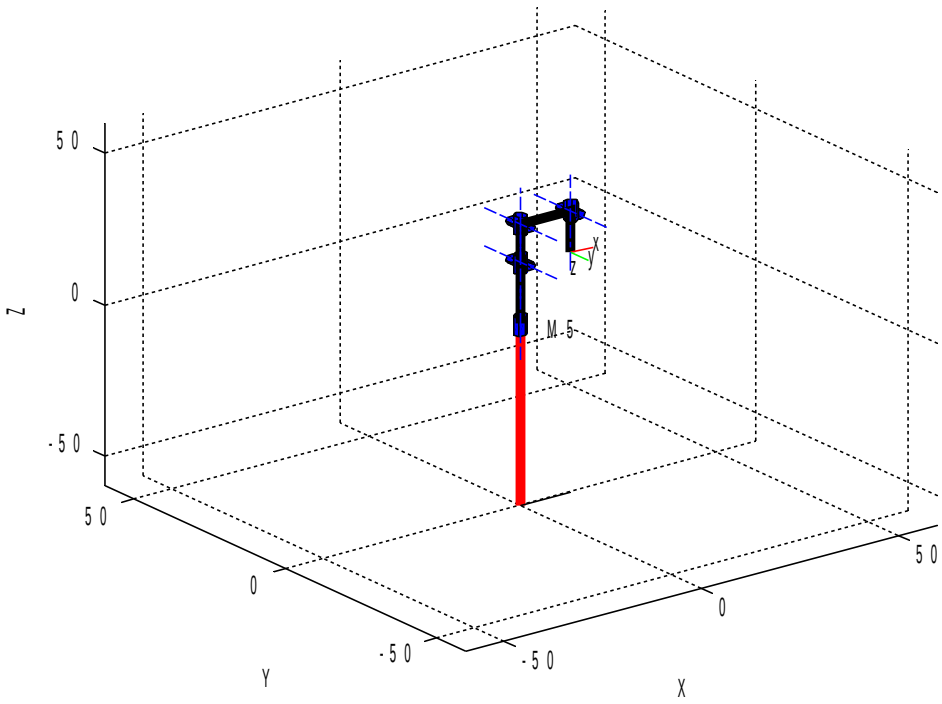
m1 = 0.0689 Kg

m2 = 0.1155 Kg

m3 = 0.1221 Kg

m4 = 0.1840 Kg

Recordando que sólo se tendrán en cuenta las articulaciones que aportan a la posición del muñón, por lo que se dejará afuera la articulación de rotación de la pinza.



3. INTRODUCCION A COMPILADORES (ANTLR):

ANTLR es un programa que está escrito en java, por lo que se necesita alguna máquina virtual de java para poder ejecutarlo. Es software libre, obtendremos tanto los ficheros compilados *.class como el código fuente en forma de ficheros *.java.

ANTLR es un generador de analizadores. Mucha gente llama a estas herramientas compiladores de compiladores, dado que ayudar a implementar compiladores es su uso más popular.

ANTLR es capaz de generar un analizador léxico, sintáctico o semántico en varios lenguajes (java, C++ y C# en su versión 2.7.6) a partir de unos ficheros escritos en un lenguaje propio. Dicho lenguaje es básicamente una serie de reglas EBNF y un conjunto de construcciones auxiliares.

ANTLR genera analizadores pred-LL(k), y él mismo utiliza un analizador pred-LL(k) para leer los ficheros en los que están escritas las reglas EBNF.

ANTLR admite acciones en sus reglas, además de otras prestaciones como paso de parámetros, devolución de valores o herencia de gramáticas. ANTLR genera analizadores pred-LL(k), y él mismo utiliza un analizador pred-LL(k) para leer los ficheros en los que están escritas las reglas EBNF.

ANTLR admite acciones en sus reglas, además de otras prestaciones como paso de parámetros, devolución de valores o herencia de gramáticas.

Los inicios del proyecto se remontan a otoño de 1988, cuando el profesor de la Universidad de Purdue, Hank Dietz, inicia el proyecto PCCTS (Purdue Compiler Construction Tool Set) como un generador de

parsers, para su utilización en un curso de graduación. Bajo la tutela de Dietz, Terence Parr realiza su tesis para el grado de Máster, creando ANTLR (originalmente llamado YUCC).Esta versión alfa, fue totalmente re-diseñada y escrita, para dar paso a la versión 1.00B. Dicha versión fue la primera release distribuida a través de Internet por un grupo de noticias (comp.compilers), en Febrero de 1990, obteniendo un buen recibimiento y comunidad de seguidores. Esta versión solo generaba parsers LL(1) y no manejaba parámetros de retorno, pero permitía expresar de manera conjunta la descripción del análisis léxico y sintáctico a realizarse.

Conforme Parr iniciaba su Ph.D. en Purdue en otoño de 1990, comenzó la segunda completa revisión y re-escritura de ANTLR. A partir de la misma, así como de la incorporación de nuevas técnicas de analizar gramáticas descubiertas en esos momentos, es que surge la versión 1.00, publicada vía un artículo en Noticias SIGPLAN (periódico mensual de la ACM). Dicha versión incluía métodos para generar parsers LL(k), para la creación y manipulación de AST's, clases léxicas, clases para manejo de errores, y clases para recuperación automática de fallas. Esta versión fue extensamente testeada por profesionales de Micro Data Base System así como candidatos a Ph.D de laUniversidad de Minnesota, mostrando ser una sólida base para la evolución que surgiría a partir de la misma, y continuaría hasta hoy en día.

Características del compilador

El programa realizado por nosotros en antlr consiste en varias funciones que generan un código de comprobación y a continuación vamos a ver el desarrollo de estas funciones en el programador de programas:

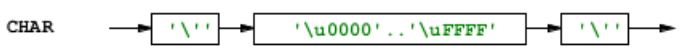
1) tokens {
   PLUS = '+' ;
   MINUS = '-' ;
   MULT = '*' ;
   DIV = '/' ;
   }

   En esta sección del compilador es donde asignamos símbolos para operaciones básicas a través de la consola y como los interpreta.

2) @header {
   import java.util.HashMap;
   import java.io.*;
   }
   @members {
   HashMap variables = new HashMap();
   public static void main(String[] args) throws Exception {
   SimpleCalcLexer lex = new SimpleCalcLexer(new
   ANTLRFileStream(args[0]));
   CommonTokenStream tokens = new CommonTokenStream(lex);
   SimpleCalcParser parser = new SimpleCalcParser(tokens);
   try {
   parser.expr();
   } catch (RecognitionException e) {
   e.printStackTrace();
   }
   }
   }
   En esta sección lo que configuramos son librerías para el desarrollo de letras y caracteres atraves del compilador.

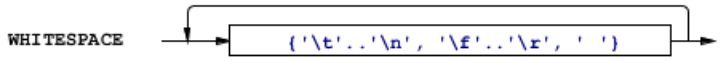
3) Reglas para funciones:

   a) CHAR:



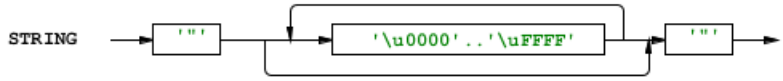
   Esta función determina que la entrada es un carácter.

   b) Whitespace:



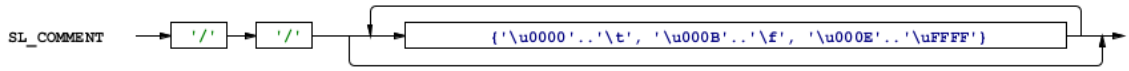
En esta regla lo que determinamos es como se interpretan los espacios, con que caracteres.

c) STRING

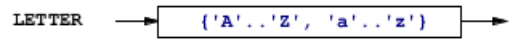


En esta regla simplemente relacionamos varios caracteres atraves de una cadena de caracteres, decimos que esta compuesta por varios valores.

d) SL_COMMENT



e) LETTER

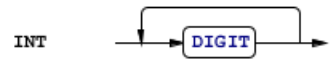


En esta regla le decimos al compilador que caracteres son letras de entrada al sistema para ser interpretadas mas adelante en otra función.

f) DIGIT

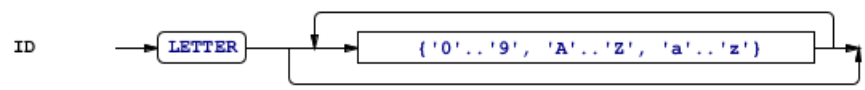
En esta regla interpretamos que entradas representan números y cuales para el compilador.

g) INT



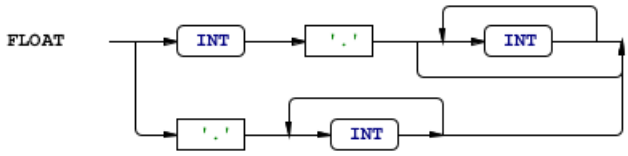
Acá se relaciona tanto los números como un digito para el compilador.

h) ID



Acá relacionamos después de una letra tanto una letra como una letra.

i) FLOAT



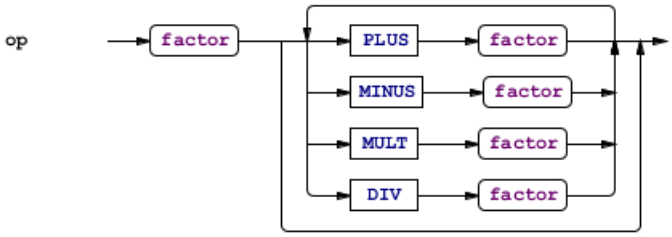
Esta refla es para generar números con enteros de parte temporal para ser usados o se puede salir si no tiene solo valores después de coma.

4) Funciones: en esta parte ya mostramos cómo funcionan las funciones que realizamos para este compilador con ayuda de las reglas para el desarrollo del mismo:

a) Pause

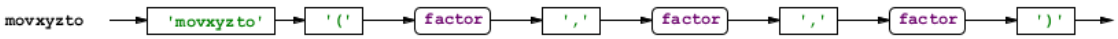
En esta función introducimos un tiempo de retardo antes de poder analizar un factor entre paréntesis.

b) Op



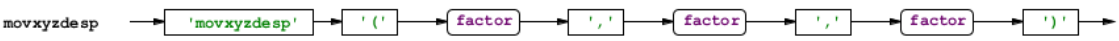
Acá realizamos operaciones dependiendo del carácter que reconoce tanto suma, resta, multiplicación, división o continuar. Esto lo realiza después de entrar un factor, hace este análisis para después introducir otro factor.

c) Movxyzto



En esta función reconocemos los valores para que genere un código a la salida en el cual va a verse un movimiento en la consola de matlab.

d) Movxyzdesp



esta función genera un código para el desplazamiento según el valor introducido y genera un nuevo código para este movimiento.

e) Movangto

En esta función realizamos algo muy similar a las funciones anteriores con la diferencia de que los parámetros de entrada a esta función no son posiciones si no que son ángulos para articulación y genera la programacion para mover en esa cantidad dichas articulaciones.

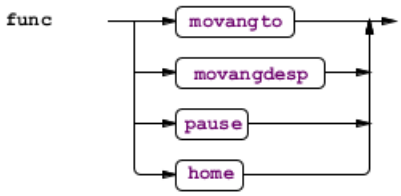
f) Movangdesp

En estafuncion imprimimos un archivo el cual contiene un desplazamiento por angulos pero de manera lenta paso por paso es muy similar a la anterior.

g) Inic

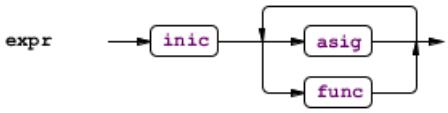
En esta función simplemente cargamos datos al texto de salida que siempre tienen que ser cargados tales como configuración y limpieza del espacio de trabajo. Inicialización del matlab.

h) Func



En esta función correlacionamos cada acción a realizar según los datos que introducimos al sistema, tiene 4 opciones para desarrollar. Elige el scrip a desarrollar según parámetros asignados por el usuario.

i) Expr



Este es por decir así como el main principal del sistema en el cual llama las siguientes funciones a realizar: operación matemática o función de implementar código.

Ejemplo código generado en matlab por compilador

Para el compilador anterior vamos a generar una trayectoria y verificaremos a través de matlab si realmente hace lo que tiene que hacer la cual es:

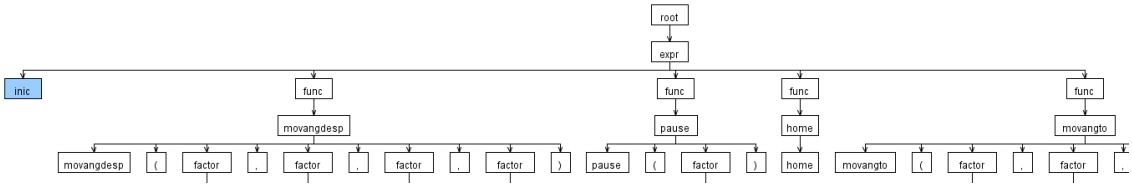
movangdesp(40,0,0,90)
pause(5)
home
movangto(50,27,64,20)

Generando la siguiente programación:

```
clear all
close all
clc
addpath C:\Compilador_Robotica\toolbox_Corke_robot_v8 -begin
l1 = link([pi/2 0 pi/2 22.53 0 0]);
l2 = link([0 12.7 pi/2 0 0 pi/2]);
l3 = link([0 12.7 -pi/2 0 0 -pi/2]);
l4 = link([pi/2 0 0 0 0 0]);
l5 = link([0 0 0 14 0 0]);
RobotM = robot({l1 l2 l3 l4 l5},'M5' );
q1a=0; q2a=0; q3a=0; q4a=0; q5a=0;
Q=jtraj([q1a q2a q3a q4a q5a],[q1a+40.0*pi/180 q2a+0.0*pi/180
q3a+0.0*pi/180 q4a+90.0*pi/180 q5a],100);
q1a=q1a+40.0*pi/180; q2a=q2a+0.0*pi/180; q3a=q3a+0.0*pi/180;
q4a=q4a+90.0*pi/180;
if(q1a<-2.1014||q1a>2.1586||q2a<-0.9336||q2a>0.5002||q3a<-0.4869||
q3a>2.042||q4a<-0.3176||q4a>3.3807)
error('Movimiento fuera de los límites de las articulaciones');
else
plot(RobotM,Q)
hold on
end
pause(5.0);
Q=jtraj([q1a q2a q3a q4a q5a],[0 0 0 0 0],100);
q1a=0;q2a=0;q3a=0;q4a=0;q5a=0;
plot(RobotM,Q)
hold on
Q=jtraj([q1a q2a q3a q4a q5a],[50.0*pi/180 27.0*pi/180 64.0*pi/180
20.0*pi/180 q5a],100);
q1a=50.0*pi/180; q2a=27.0*pi/180; q3a=64.0*pi/180; q4a=20.0*pi/180;
if(q1a<-2.1014||q1a>2.1586||q2a<-0.9336||q2a>0.5002||q3a<-0.4869||
q3a>2.042||q4a<-0.3176||q4a>3.3807)
error('Movimiento fuera de los límites de las articulaciones');
else
plot(RobotM,Q)
hold on
end
```

la cual al ser implementada pudimos ver que esta funcionando perfectamente nuestro compilador.

Compilador robot Hernán Muñoz Fabián Zanella



Conclusiones

En el anterior trabajo pudimos analizar tanto desde el desarrollo cinemático, dinámico para lograr finalmente un propio compilador a través del cual controlamos nuestro robot basándonos en todas estas normas.
Desarrollamos este análisis completo para un m5 aun que puede llegar a ser base para el desarrollo de otro tipos de modelos y amplio desarrollo en cuanto a la propia programacion de control del mismo.