UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES
Departamento de Electrónica

Materia: Robótica

Trabajo Práctico 3

Docente: Ing. Hernán Gianetta

Ayudante de TP: Ing. Damián Granzella

Grupo N°:

Alumnos :

	Apellido y Nombre	**Legajo**
1	Resquín, Fernando Emilio	112316-6

Tabla de contenido

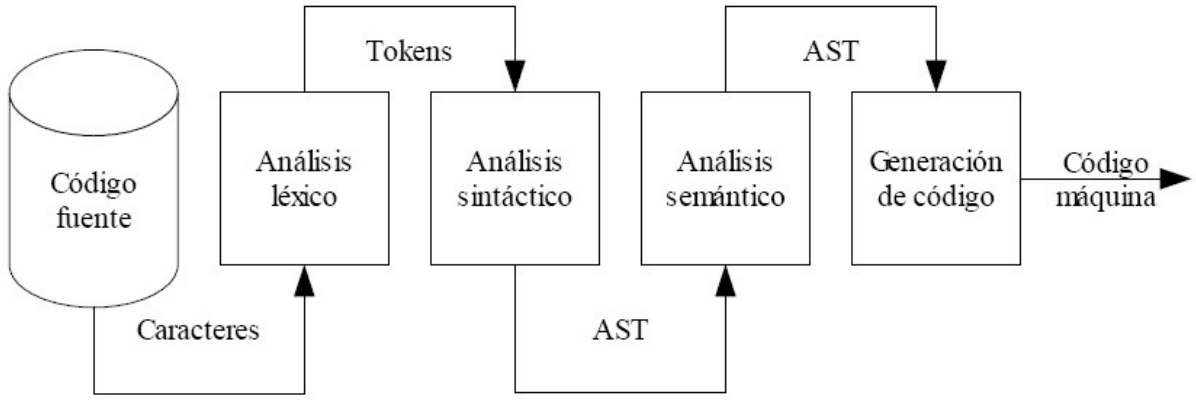
1. Introducción Teórica

1.1 Compiladores – Definición:

Podría definirse como un programa que lee un programa escrito en un lenguaje de programación y lo traduce a un programa equivalente en otro lenguaje de programación.
Desde otro punto de vista, un compilador es un programa que traduce un programa hecho en un lenguaje de alto nivel en un programa funcionalmente equivalente con un lenguaje de bajo nivel.
Un compilador es, a muy a groso modo "un programa que sirve para hacer otros programas, mediante el uso de un lenguaje de programación".

1.2 Fases del proceso de compilación:

Lo primero que debe hacer un compilador es comprobar que la información que se le suministra pertenece a su lenguaje (no hay errores léxicos, sintácticos ni semánticos). Si es así, el intérprete debe representar de alguna manera la información que se le suministro para poder trabajar con ella, y finalmente traducir dicha información a código máquina.



Estructura básica de un compilador

1.2.1 Definiciones de las distintas etapas de compilación:

Código Fuente: Es información almacenada en la memoria de un ordenador. Suele tratarse de uno o varios ficheros de texto normalmente en el disco duro de la máquina. En estos ficheros hay cierta información cuyo fin es provocar ciertas acciones en una maquina objetivo (que puede no ser la que está interpretándolos"). Para ello, los ficheros son leídos del disco y pasados a la memoria, conformando el flujo denominado "Caracteres".

Scanner (Análisis léxico): Su tarea es verificar si todas las cadenas pertenecen o no al lenguaje. Para ello analiza símbolo por símbolo indicando el token por cada uno de los elementos reconocidos o el error en caso de no reconocerlos. Este análisis no logra detectar muchos errores por su característica.
El programa fuente se trata inicialmente con el analizador léxico o scanner con el propósito de agrupar el texto en grupos de caracteres con entidad propia (tokens, unidades sintácticas o componentes léxicos) tales como constantes, identificadores (de variables, de funciones, de procedimientos, de tipos, de clases), palabras reservadas y operadores. A cada token se le asocia uno o más atributos, representados internamente por un código numérico o por un tipo enumerado.

Parser (Análisis sintáctico): Implica agrupar los componentes léxicos del programa fuente en frases gramaticales que el compilador utiliza para sintetizar la salida. El análisis sintáctico o parser es un análisis a nivel de sentencias y es más complejo que el análisis léxico.
Su función es tomar el programa fuente en forma de tokens que recibe del analizador léxico y determinar la estructura de las sentencias del programa. El análisis sintáctico agrupa a los tokens en clases sintácticas como expresiones y procedimientos, y obtiene un árbol sintáctico en el cual las hojas son los tokens y los nodos representan un tipo de clase sintáctica.

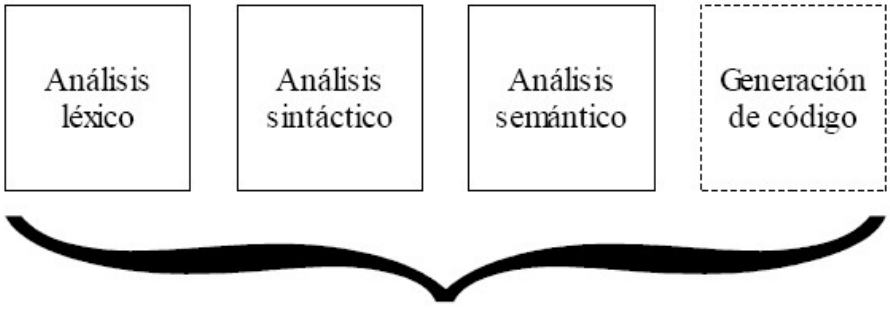
Análisis semántico: Se encarga de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico. Suele trabajar en simultáneo y en estrecha cooperación con el analizador sintáctico. La semántica es el conjunto de reglas que especifican el significado de cualquier sentencia sintácticamente correcta y escrita en un determinado lenguaje.
Cuando el analizador sintáctico reconoce un operador llama a una rutina semántica que especifica la acción que puede llevar a cabo, comprobando también que los operandos hayan sido previamente declarados, sean del tipo correspondiente y si se les ha asignado algún valor. En este paso se evalúa el rango de valores de cada parámetro y que el significado final de la instrucción tenga sentido para la máquina que la va a recibir, asegurándose que la instrucción pueda ser ejecutada exitosamente.

Generación de código: En esta fase se utiliza el AST enriquecido, producto del proceso de análisis semántico, para generar código máquina. El compilador es un “puente” entre el operario y el robot, que permite simplificar el uso del mismo.

2 Compilador ANTLR:

Es una herramienta desarrollada con el objetivo de programar o "construir" compiladores. Su nombre viene del acrónimo de "Another Tool for Language Recognition". En el transcurso de este trabajo se lo utilizo a través del IDE ANTLRWorks, el cual simplifica el proceso de creación al integrar herramientas, como la representación gráfica de los flujos de tokens y un debugger para probar el compilador creado.

Podemos presentar al ANTLR como una herramienta completa que agrupa todos los niveles de un compilador. ANTLR es capaz de actuar a tres niveles a la vez (cuatro si tenemos en cuenta la generación de código):



Comparación de las herramientas

El uso de una sola herramienta para todos los niveles tiene varias ventajas. La más importante es la "estandarización": con ANTLR basta con comprender el paradigma de análisis una vez para poder implementar todas las fases de análisis.

Tanto ANTLR (línea de comandos) como ANTLRWorks son programas escritos en java, por lo que se necesita alguna máquina virtual de java para poder ejecutarlo. El software ANTLRWorks ya incluye ANTLR dentro del paquete.

El programa escrito en ANTLR sigue el siguiente esquema:

- grammar JarvisCommand.g: Cabecera del fichero
- tokens: Declaración de tokens
- @header: Código de inicialización
- @members: Contiene el código en JAVA del programa
- Parser rules : Reglas del analizador sintáctico
- Lexer rules: Reglas del analizador léxico

Para este proyecto se utilizó el ANTLRworks-1.5.2, que incluye ANTLR v3. El programa ANTLRworks provee una GUI (Interfaz Gráfica de Usuario) para evitar trabajar por línea de comandos, además de agregar otras funcionalidades. Dicho software está presentado como archivo .JAR y para poder ejecutarlo hay que tener instalado el Java SE Development Kit. Para este trabajo se utilizó la Versión 6, update 65.

3 Compilador para el robot Jarvis:

Como criterio de diseño, se consideró que el compilador cumplirá la función de generar la trayectoria que seguirá la plataforma móvil para moverse por el terreno. En definitiva, se definen una serie de instrucciones secuenciales que guiarán al movimiento del robot, de manera similar a como se hace con un manipulador, por los que dichos movimientos serán programados por el usuario antes de colocar el robot en el terreno. El presente trabajo no abarca el diseño de las instrucciones para el manipulador robótico, las cuales pueden obtenerse de otros, para luego definir si se procesarán secuencialmente con las de la plataforma o en paralelo para ejecutarse con la plataforma en movimiento.

Tampoco se considerará la posibilidad de incluir inteligencia artificial al robot, con lo cual la naturaleza de las instrucciones cambiaría totalmente.
El compilador generará un archivo "Trayectoria1.m" que en realidad es un script de Matlab, el cual es parte del generador de trayectorias utilizado en TP1 y TP2.

3.1 Funciones implementadas:

Doblar_Izq(double r, int g, int acc, int desc, double vel1, double vel2)

Esta función traza una trayectoria de arco circular para doblar a la izquierda, según el radio de giro especificado. A su vez define el perfil de velocidades para recorrer dicha trayectoria. La trayectoria se divide en 100 puntos, de los cuales se puede definir por separado el porcentaje dedicado a acelerar y para frenar. Además se puede establecer la velocidad de la zona plana del perfil y también la velocidad en la que termina. La aceleración y la desaceleración se realiza mediante una curva polinómica de séptimo orden, para obtener continuidad en la posición, la velocidad, la aceleración y el jerk de la plataforma del robot.

r: Radio de giro en metros. Deberá expresarse siempre con una parte entera y una parte decimal, aunque esta sea cero. La separación debe hacerse usando "."

g: Ángulo de giro. Si g=1, la trayectoria dará 1/4 de giro. Si g=2 dará medio giro.

acc: Porcentaje de los puntos del perfil dedicados a acelerar.

desc: Porcentaje de los puntos del perfil dedicados a desacelerar.

vel1: Velocidad constante de desplazamiento, o velocidad de la zona plana del perfil. Se mide en m/s. Deberá expresarse siempre con una parte entera y una parte decimal, aunque esta sea cero. La separación debe hacerse usando "."

vel2: Ídem anterior, salvo que es la velocidad en la que finaliza el perfil de movimiento.

Doblar_Der(double r, int g, int acc, int desc, double vel1, double vel2)

Idem a "Doblar_Izq" pero permite girar a la derecha.

Adelante(double len, int acc , int desc, double vel1, double vel2)

Esta función agrega un tramo rectilíneo a la trayectoria, según la longitud especificada. A su vez define el perfil de velocidades para recorrer dicha trayectoria. La trayectoria se divide en 100 puntos, de los cuales se puede definir por separado el porcentaje dedicado a acelerar y para frenar. Además se puede establecer la velocidad de la zona plana del perfil y también la velocidad en la que termina. La aceleración y la desaceleración se realizan mediante una curva polinómica de séptimo orden, para obtener continuidad en la posición, la velocidad, la aceleración y el jerk de la plataforma del robot.

len: Longitud de desplazamiento en metros. Deberá expresarse siempre con una parte entera y una parte decimal, aunque esta sea cero. La separación debe hacerse usando "."

acc: Porcentaje de los puntos del perfil dedicados a acelerar.

desc: Porcentaje de los puntos del perfil dedicados a desacelerar.

vel1: Velocidad constante de desplazamiento, o velocidad de la zona plana del perfil. Se mide en m/s. Deberá expresarse siempre con una parte entera y una parte decimal, aunque esta sea cero. La separación debe hacerse usando "."

vel2: Ídem anterior, salvo que es la velocidad en la que finaliza el perfil de movimiento.

Iniciar()

Se ocupa de generar el encabezado del código intermedio (Matlab). Esta instrucción deberá ejecutarse antes que cualquier otra. En caso de no encontrarse en el programa como primera instrucción todas las demás instrucciones provocarán un error de compilación.

Terminar()

Se ocupa de agregar las instrucciones de cierre al código intermedio (Matlab). Esta instrucción deberá ejecutarse al final de todas las instrucciones del programa. En caso de no estar los scripts de Matlab no se ejecutarán.

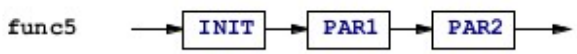
3.2 Representaciones gráficas del compilador creado:

El IDE ANTLRWorks permite representar gráficamente las reglas del Parser, las reglas del Lexer, y el árbol AST, las cuales se muestran a continuación:

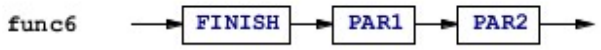
Reglas del Parser

Variables

Func5 es Iniciar()

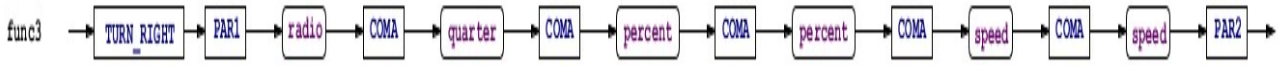


Func6 es Terminar()

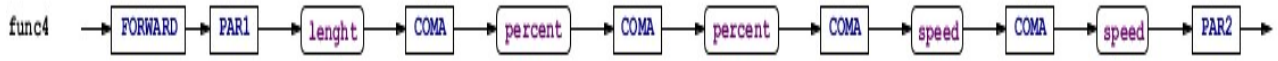


Func2 es Doblar_Izq()

Func3 es Doblar_Der()

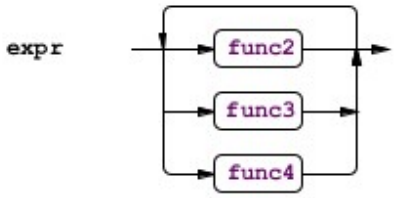


Func4 es Adelante()

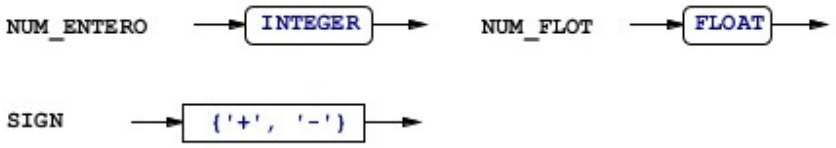


Con esta regla se obliga a que primero se invoque a la instrucción Iniciar(), luego las instrucciones generales y al final la instrucción Terminar()

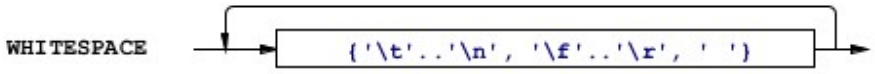
Esta regla se invoca recursivamente las tres funciones generales del lenguaje para ir compilando cada instrucción a medida que aparecen en el archivo fuente



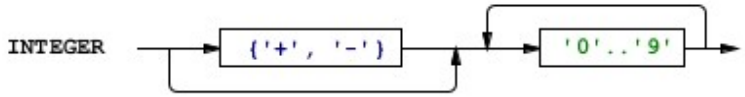
Reglas del Lexer



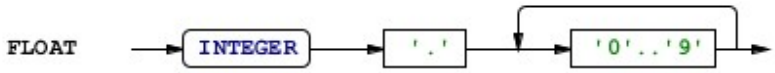
Definición del espacio en blanco



Tipo de datos Integer



Tipo de datos Float



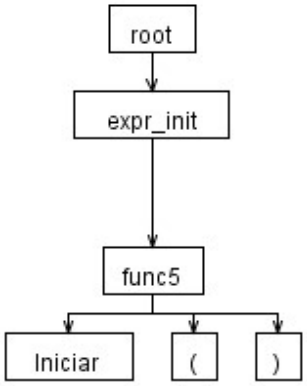
3.3 Código fuente:

Se transcribe a continuación el código fuente que está en el archivo .PRG que luego será compilado y ejecutado en Matlab.

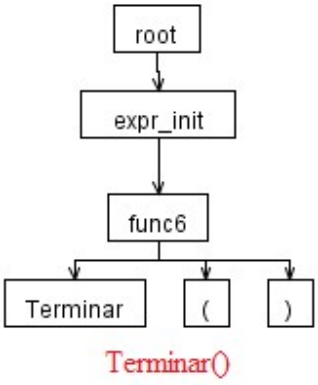
```
Iniciar()
Adelante(10.0,30,30,3.5,1.0)
Doblar_Der(3.0,1,10,10,1.0,1.0)
Adelante(7.0,30,30,2.5,1.5)
Doblar_Izq(4.0,1,10,10,1.5,1.5)
Adelante(13.0,30,30,3.5,2.5)
Doblar_Izq(5.0,1,10,10,2.5,2.5)
Adelante(29.0,30,30,5.0,3.0)
Doblar_Izq(8.0,2,10,10,3.0,3.0)
Doblar_Der(8.5,2,10,10,3.0,3.0)
Adelante(20.0,30,30,4.0,3.0)
Doblar_Izq(10.0,2,10,10,3.0,3.0)
Adelante(20.0,30,30,4.0,1.0)
Doblar_Der(3.0,1,10,10,1.0,1.0)
Adelante(9.0,30,30,3.0,2.5)
Doblar_Izq(5.0,1,10,10,2.5,2.5)
Adelante(42.0,30,30,5.0,3.5)
```

Doblar_Izq(10.0,2,10,10,3.5,3.5)
Doblar_Der(10.0,2,10,30,3.5,2.5)
Doblar_Izq(5.0,1,10,10,2.5,2.5)
Adelante(30.0,30,30,3.5,0.01)
Terminar()

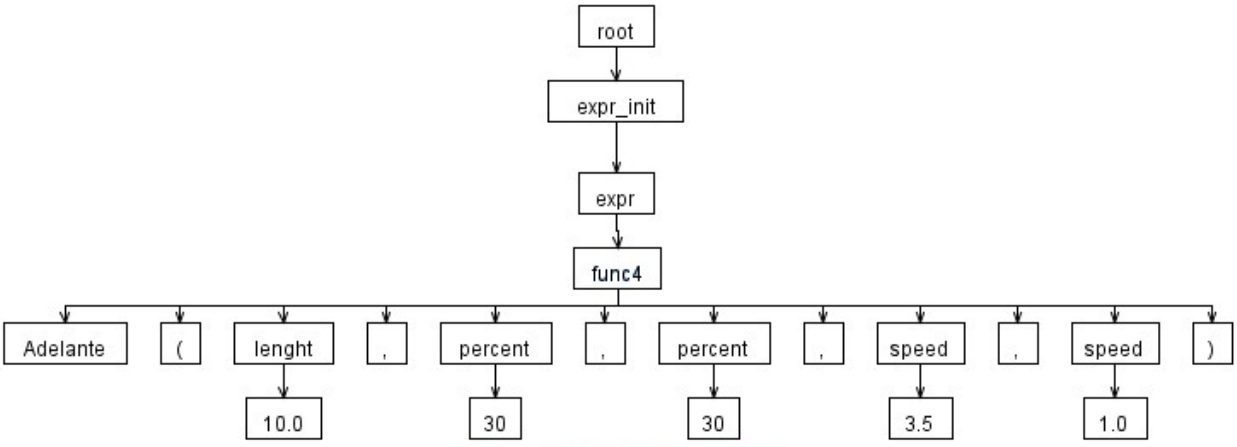
3.4 Ejemplos de árbol AST:



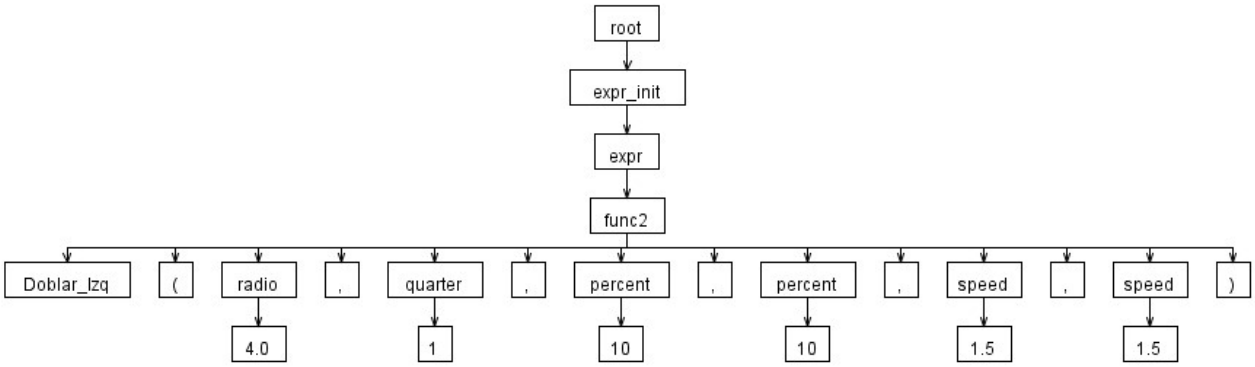
Iniciar()



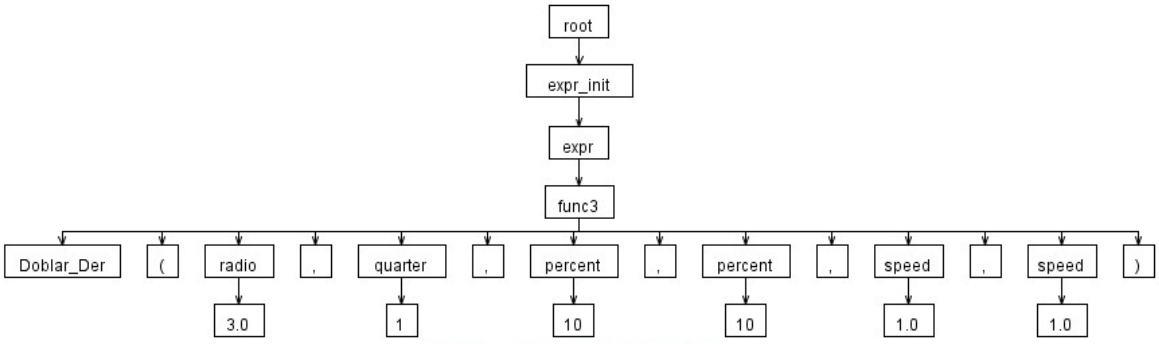
Terminar()



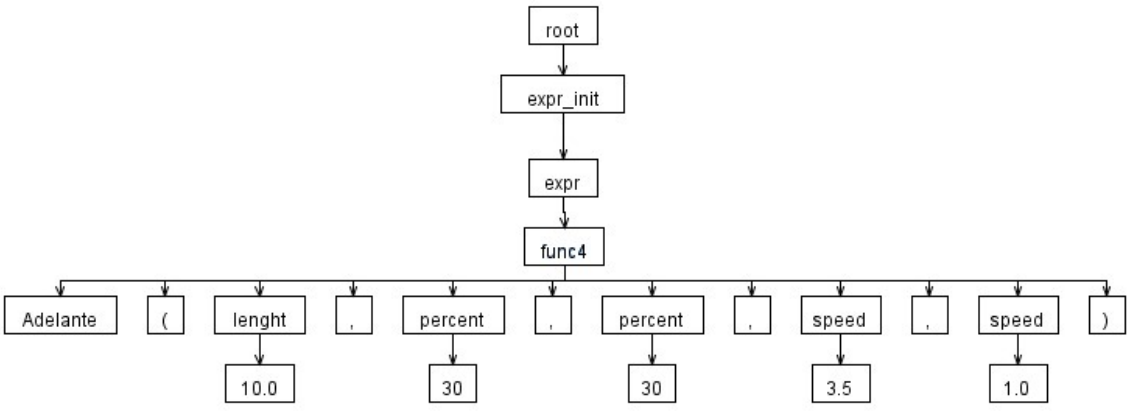
Adelante(10.0,30,30,3.5,1.0)



Doblar_Izq(4.0,1,10,10,1.5,1.5)



Doblar_Der(3.0,1,10,10,1.0,1.0)



Adelante(10.0,30,30,3.5,1.0)

3.5 Código del Compilador:

```
grammar JarvisCommand;

tokens{
	TURN_LEFT ='Doblar_Izq';
	TURN_RIGHT ='Doblar_Der';
	FORWARD ='Adelante';
	INIT ='Iniciar';
	FINISH ='Terminar';
	PAR1   = '('		;
	PAR2   = ')'		;
	COMA   = ','		;
}

@header{

	import java.util.ArrayList;
	import java.lang.Math;
	import java.io.PrintWriter;			// Para generar el script de salida
	import java.util.HashMap;			// Para almacenar variables
	import java.io.BufferedReader;			// Para leer la base del script a generar
	import java.io.File;
}

@members {
	PrintWriter writer;
	HashMap variables = new HashMap();
	HashMap valores = new HashMap();
	int flag = 0;
	public void main(String[] args) throws Exception {
		JarvisCommandLexer lex = new JarvisCommandLexer(new ANTLRFileStream(args[0]));
		CommonTokenStream tokens = new CommonTokenStream(lex);
		JarvisCommandParser parser = new JarvisCommandParser(tokens);
		try {
			parser.expr();
		}catch (RecognitionException e){
			e.printStackTrace();
		}

	}
}

/*------------------------------------------------------------------
 * PARSER RULES
 *------------------------------------------------------------------*/

 expr_init  :		(func5) & expr & (func6);
 expr	    :		((func2) | (func3) | (func4))+;


 quarter returns [int value]
	: NUM_ENTERO
	{
	$value = Integer.parseInt($NUM_ENTERO.text);
	if($value < 1 || $value > 2) {
		System.err.println("Error en el ángulo de giro");
		$value = 255;
	}
	}
;

 percent returns [int value]
	: NUM_ENTERO
	{
	$value = Integer.parseInt($NUM_ENTERO.text);
	if($value < 0 || $value > 100) {
		System.err.println("Error en el ángulo de giro");
		$value = 255;
	}
```

```
		}
;

lenght returns [Double value]
		:
		NUM_FLOT
		{
		$value = Double.parseDouble($NUM_FLOT.text);
		if($value <= 0.0){
			System.err.println("El largo de avance no debe ser negativo");
			$value = -1.0;
		}
		}
;

radio returns [Double value]
		:
		NUM_FLOT
		{
		$value = Double.parseDouble($NUM_FLOT.text);
		if($value <= 0.0){
			System.err.println("El radio de la curva no debe ser negativo");
			$value = -1.0;
		}
		}
;

speed returns [Double value]
		:
		NUM_FLOT
		{
		$value = Double.parseDouble($NUM_FLOT.text);
		if($value <= 0.0){
			System.err.println("La velocidad de movimiento no debe ser negativa");
			$value = -1.0;
		}
		}
;


func2		:
		TURN_LEFT PAR1 r=radio COMA g=quarter COMA acc=percent COMA desc=percent COMA vel1=speed COMA vel2=speed PAR2
		{
		if(flag == 0 || $r.value == -1.0 || $g.value == 255 || $acc.value == 255 || $vel1.value == -1.0 || $vel2.value == -1.0 || $desc.value == 255)
			System.err.println("Error de Compilación");
		else		{
				// GENERACION DE SCRIPT PARA MATLAB
				System.out.println("\%Giro a la izquierda");
				System.out.println("Last_Vel = Tray.v(Tray.ind);");
				System.out.println("Tray = Change_Vel(Last_Vel , "+$vel1.value+","+$acc.value+", Tray);\%velocidad de recorrido");
	System.out.println("Tray = Change_Vel(Last_Vel , "+$vel1.value+",100 - "+$acc.value+" - "+$desc.value+", Tray);\%velocidad de recorrido");
			System.out.println("Tray = Change_Vel("+$vel1.value+" , "+$vel2.value+","+$desc.value+", Tray);\%velocidad de recorrido");

				System.out.println("Last_X = Tray.x(Tray.ind);");
				System.out.println("Last_Y = Tray.y(Tray.ind);");

				System.out.println("Orient = Get_Orient(Tray);\%Orientación  a partir de los 2 últimos puntos");

				System.out.println("if(Orient == 1)");
				if($g.value == 2)
			System.out.println("Tray = plot_arc(-pi/2,pi/2,Last_X,Last_Y + "+$r.value+","+$r.value+", Tray);\%Trazo curva a la izquierda");
				else
			System.out.println("Tray = plot_arc(-pi/2,0,Last_X,Last_Y + "+$r.value+","+$r.value+", Tray);\%Trazo curva a la izquierda");
				System.out.println("end");

				System.out.println("if(Orient == 2)");
				if($g.value == 2)
			System.out.println("Tray = plot_arc(0,pi,Last_X - "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la izquierda");
				else
			System.out.println("Tray = plot_arc(0,pi/2,Last_X - "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la izquierda");
				System.out.println("end");
```

```
                        System.out.println("if(Orient == 3)");
                        if($g.value == 2)
                System.out.println("Tray = plot_arc(pi/2,1.5*pi,Last_X,Last_Y - "+$r.value+","+$r.value+", Tray);\%Trazo curva a la izquierda");
                        else
                System.out.println("Tray = plot_arc(pi/2,pi,Last_X,Last_Y - "+$r.value+","+$r.value+", Tray);\%Trazo curva a la izquierda");
                        System.out.println("end");

                        System.out.println("if(Orient == 4)");
                        if($g.value == 2)
                System.out.println("Tray = plot_arc(-pi,0,Last_X + "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la izquierda");
                        else
                System.out.println("Tray = plot_arc(-pi,-pi/2,Last_X + "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la izquierda");
                        System.out.println("end");
                                        }
        }
        ;

func3   :
        TURN_RIGHT PAR1 r=radio COMA g=quarter COMA acc=percent COMA desc=percent COMA vel1=speed COMA vel2=speed PAR2
        {
        if(flag == 0 || $r.value == -1.0 || $g.value == 255 || $acc.value == 255 || $vel1.value == -1.0 || $vel2.value == -1.0 || $desc.value == 255)
                System.err.println("Error de Compilación");
        else    {
                        // GENERACION DE SCRIPT PARA MATLAB
                        System.out.println("\%Giro a la derecha");
                        System.out.println("Last_Vel = Tray.v(Tray.ind);");
                        System.out.println("Tray = Change_Vel(Last_Vel , "+$vel1.value+","+$acc.value+", Tray);\%velocidad de recorrido");
        System.out.println("Tray = Change_Vel(Last_Vel , "+$vel1.value+",100 - "+$acc.value+" - "+$desc.value+", Tray);\%velocidad de recorrido");
                System.out.println("Tray = Change_Vel("+$vel1.value+" , "+$vel2.value+","+$desc.value+", Tray);\%velocidad de recorrido");

                        System.out.println("Last_X = Tray.x(Tray.ind);");
                        System.out.println("Last_Y = Tray.y(Tray.ind);");

                        System.out.println("Orient = Get_Orient(Tray);\%Orientación  a partir de los 2 últimos puntos");

                        System.out.println("if(Orient == 1)");
                        if($g.value == 2)
                System.out.println("Tray = plot_arc(pi/2,-pi/2,Last_X,Last_Y - "+$r.value+","+$r.value+", Tray);\%Trazo curva a la derecha");
                        else
                System.out.println("Tray = plot_arc(pi/2,0,Last_X,Last_Y - "+$r.value+","+$r.value+", Tray);\%Trazo curva a la derecha");
                        System.out.println("end");

                        System.out.println("if(Orient == 2)");
                        if($g.value == 2)
                System.out.println("Tray = plot_arc(pi,0,Last_X + "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la derecha");
                        else
                System.out.println("Tray = plot_arc(pi,pi/2,Last_X + "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la derecha");
                        System.out.println("end");

                        System.out.println("if(Orient == 3)");
                        if($g.value == 2)
                System.out.println("Tray = plot_arc(1.5*pi,pi/2,Last_X,Last_Y + "+$r.value+","+$r.value+", Tray);\%Trazo curva a la derecha");
                        else
                System.out.println("Tray = plot_arc(-pi/2,-pi,Last_X,Last_Y + "+$r.value+","+$r.value+", Tray);\%Trazo curva a la derecha");
                        System.out.println("end");

                        System.out.println("if(Orient == 4)");
                        if($g.value == 2)
                System.out.println("Tray = plot_arc(0,-pi,Last_X - "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la derecha");
                        else
                System.out.println("Tray = plot_arc(0,-pi/2,Last_X - "+$r.value+",Last_Y,"+$r.value+", Tray);\%Trazo curva a la derecha");
                        System.out.println("end");
                        }
        }
        ;

func4   :
        FORWARD PAR1 len=lenght COMA acc=percent COMA desc=percent COMA vel1=speed COMA vel2=speed PAR2
        {
                if(flag == 0 || $len.value == -1.0 || $acc.value == 255 || $vel1.value == -1.0 || $vel2.value == -1.0 || $desc.value == 255)
                        System.err.println("Error de Compilación");
```

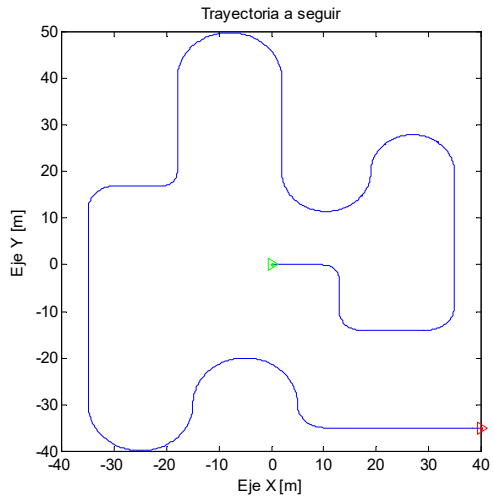
```
                else {
                        // GENERACION DE SCRIPT PARA MATLAB
                        System.out.println("\%Tramo rectilíneo");
                        System.out.println("Last_Vel = Tray.v(Tray.ind);");
                        System.out.println("Tray = Change_Vel(Last_Vel , "+$vel1.value+","+$acc.value+", Tray);\%velocidad de recorrido");
        System.out.println("Tray = Change_Vel(Last_Vel , "+$vel1.value+",100 - "+$acc.value+" - "+$desc.value+", Tray);\%velocidad de recorrido");
        System.out.println("Tray = Change_Vel("+$vel1.value+" , "+$vel2.value+","+$desc.value+", Tray);\%velocidad de recorrido");

                        System.out.println("Last_X = Tray.x(Tray.ind);\%Posición de último punto de la trayectoria");
                        System.out.println("Last_Y = Tray.y(Tray.ind);\%Posición de último punto de la trayectoria");
                        System.out.println("Orient = Get_Orient(Tray);\%Orientación  a partir de los 2 últimos puntos");
                        System.out.println("if(Orient == 1)");
                System.out.println("Tray = plot_line(Last_X,Last_Y,Last_X + "+$len.value+",Last_Y,Tray);\%Trazo el tramo de trayectoria");
                        System.out.println("end");
                        System.out.println("if(Orient == 2)");
                System.out.println("Tray = plot_line(Last_X,Last_Y,Last_X,Last_Y + "+$len.value+",Tray);\%Trazo el tramo de trayectoria");
                        System.out.println("end");
                        System.out.println("if(Orient == 3)");
                System.out.println("Tray = plot_line(Last_X,Last_Y,Last_X - "+$len.value+",Last_Y,Tray);\%Trazo el tramo de trayectoria");
                        System.out.println("end");
                        System.out.println("if(Orient == 4)");
                System.out.println("Tray = plot_line(Last_X,Last_Y,Last_X,Last_Y - "+$len.value+",Tray);\%Trazo el tramo de trayectoria");
                        System.out.println("end");
                        //System.out.println("Last_Vel = Tray.v(Tray.ind);");
                        //System.out.println("Tray = Change_Vel(Last_Vel, Last_Vel ,"+$len.value+", Tray);\%velocidad de recorrido");
                        }
        }
        ;


func5   :
        INIT PAR1 PAR2
        {
        flag = 1;
        System.out.println("function H = Trayectoria1(Tray)");
        }
        ;

func6   :
        FINISH PAR1 PAR2
        {
        if (flag == 0)
        return;

        System.out.println("\%Eliminar elementos repetidos. No debe haber 2 puntos consecutivos");
        System.out.println("\%del vector con las mismas coordenadas x, y.");
        System.out.println("i=2;");
        System.out.println("barrer = 0;");
        System.out.println("while (barrer == 0)");
        System.out.println("if(i ==  length(Tray.x))");
        System.out.println("barrer = 1;");
        System.out.println("else");
    System.out.println("if(Tray.x(i) == Tray.x(i-1))");
    System.out.println("Tray.x(i)=[];\%Igualando a '[]' elimino el elemento");
    System.out.println("Tray.y(i)=[];");
    System.out.println("Tray.v(i)=[];");
    System.out.println("Tray.ind = length(Tray.x);\%Corrijo el nro de posiciones de Tray");
    System.out.println("end");
        System.out.println("i = i + 1;");
        System.out.println("end");
        System.out.println("end");

        System.out.println("H.x = Tray.x;");
        System.out.println("H.y = Tray.y;");
        System.out.println("H.v = Tray.v;");
        System.out.println("H.ind = Tray.ind;");
        System.out.println("if ~nargout");
        System.out.println("display('Función Trayectoria() - Se requieren argumentos')");
        System.out.println("return;");
        System.out.println("end");

        // FIN DE GENERACION DE SCRIPT
```

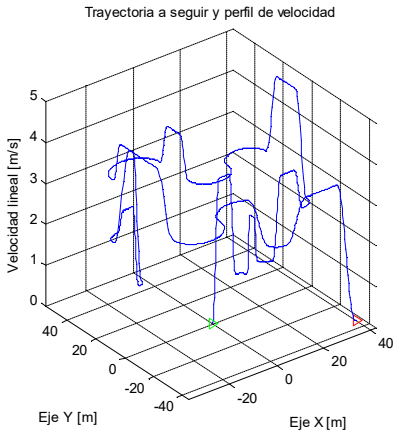
```
        System.out.flush();
        System.out.println("\%Compilación exitosa.");
        }
        ;


/*------------------------------------------------------------------
* LEXER RULES
*------------------------------------------------------------------*/
WHITESPACE : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+        { $channel = HIDDEN; } ;
NUM_ENTERO      :       INTEGER;
NUM_FLOT        :       FLOAT;
fragment SIGN   :       ('+'|'-');
fragment FLOAT  :   INTEGER '.' '0'..'9'+;
fragment INTEGER : (SIGN)? ('0'..'9')+;
```

4. Resultados de la Compilación

Como ya se había adelantado, se obtiene como "código de máquina" un script de Matlab que representa una función, la cual al ejecutarse genera en una estructura de datos en memoria la trayectoria deseada en forma de puntos en el plano. Además define la velocidad instantánea para cada punto de esa trayectoria.
Se creó un script de Matlab que grafica dicha trayectoria en el plano y además plotea un gráfico en 3D que agrega la velocidad instantánea en el eje "Z".

Estos serían los resultados:





5. Mejoras a futuro

El sistema de instrucciones puede ser ampliado para abarcar todas las maniobras posibles o movimientos que puede hacer el robot:

- Instrucciones para retroceder marcha atrás.
- Instrucciones para girar sobre su eje, verificando primero que el robot está detenido.
- Agregado de instrucciones para el movimiento del manipulador. Como se había dicho antes, hay que definir si la operación del brazo es en simultáneo al movimiento de la plataforma o si las operaciones se harán cuando la misma esté detenida, definiendo estaciones en la trayectoria sobre el plano.

Además habría que ampliar los script de Matlab para que abarque todas las funciones del robot y se pueda simular el robot completo. Luego, el paso siguiente sería implementar el sistema en un conjunto Microconrolador+DSP+FPGA adecuado a la complejidad y velocidad del robot.

Aún con realizando todo este trabajo el robot sólo será capaz de seguir trayectorias previamente definidas por el usuario, sin poder desviarse de la misma.
Para agregar autonomía al robot, hay que incorporar sensores, algoritmos de inteligencia artificial y de toma de decisiones, para encontrar caminos alternativos o incluso trazar sus propias trayectorias para realizar una tarea. Con este enfoque las instrucciones del compilador cambiarían totalmente, y tanto más cuanto mayor inteligencia tenga el sistema. Las instrucciones del lenguaje se transformarían en tareas a realizar.

6. Conclusiones

- El software ANTLRWorks y el núcleo ANTLR demostró ser muy práctico para generar compiladores, ya que tiene todas las fases de la compilación ya resueltas y toda la estructura necesaria implementada. En el presente trabajo se utilizaron ejemplos ya hechos de compiladores y se los modificó brevemente para implementar las 5 instrucciones del lenguaje "JarvisCommand". Sin embargo, las reglas del compilador en ANTLR se deben programar en Java, por lo que es muy conveniente tener al menos nociones básicas de programación en Java.

- Con las 5 instrucciones creadas se puede trazar las trayectorias mucho más fácilmente sin pensar en las coordenadas de inicio y fin de cada tramo, además de que el perfil de velocidad continúa donde terminó el del tramo anterior, previniendo discontinuidades bruscas.

7. Referencias:

- *Focaraccio Maximiliano, Sokolowicz Martín (2014). Trabajo Práctico Final - Compilador*
- *Abaca Daniel Alberto, Tunez Diego (2014). Tesis Final*
- *Sitio web ANTLR: http://www.antlr.org/*
- *Sitio web ANTLRWorks: http://www.antlr3.org/works/*