Compilador en ANTLR para un robot Hexápodo

Trabajo Práctico Final

Emiliano Javier Borghi Orué

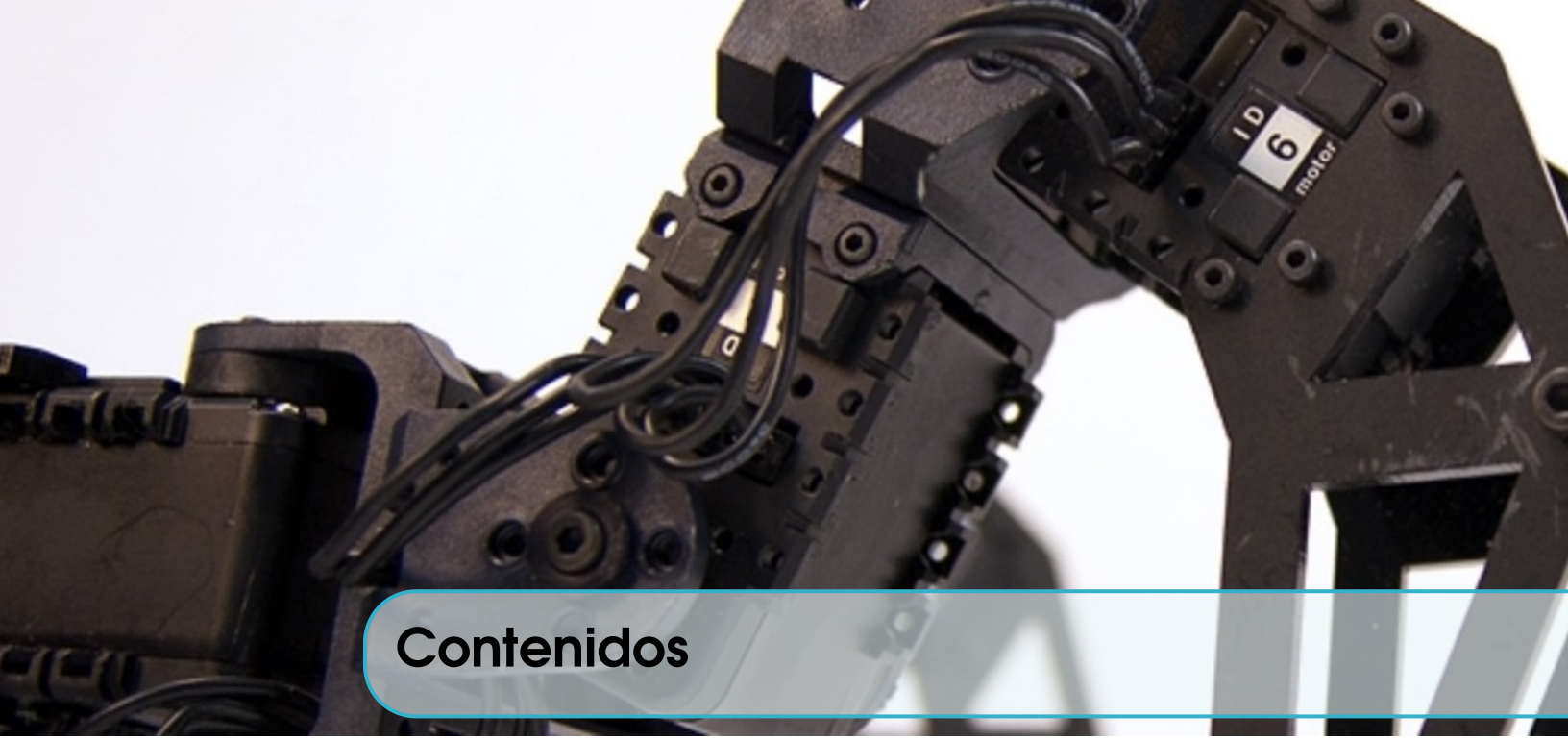
Universidad Tecnológica Nacional

Facultad Regional Buenos Aires

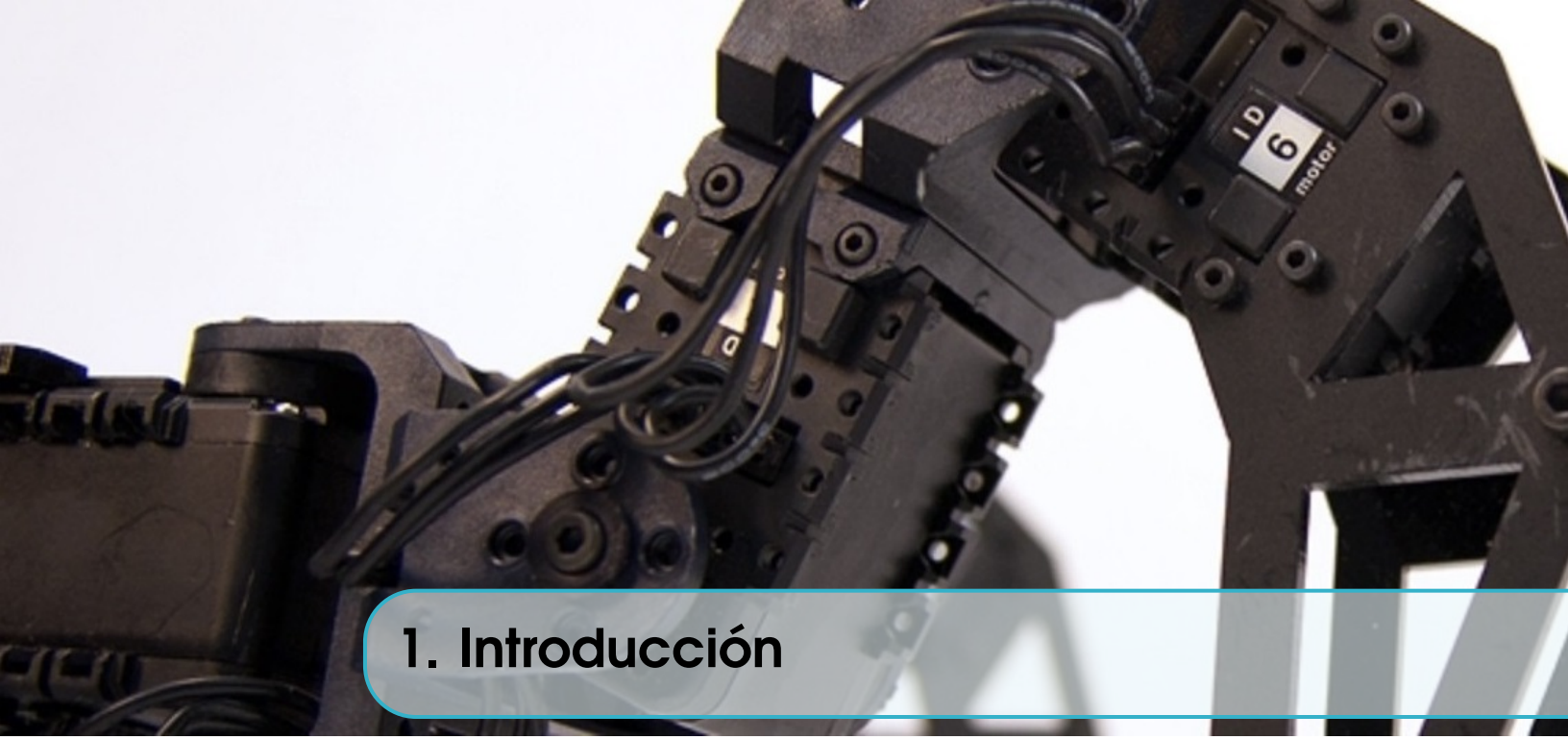
Departamento de Ingeniería electrónica, Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Trabajo práctico final de Robótica (R6055). Agosto 2015.

Documento desarrollado con LaTeX.



Contenidos

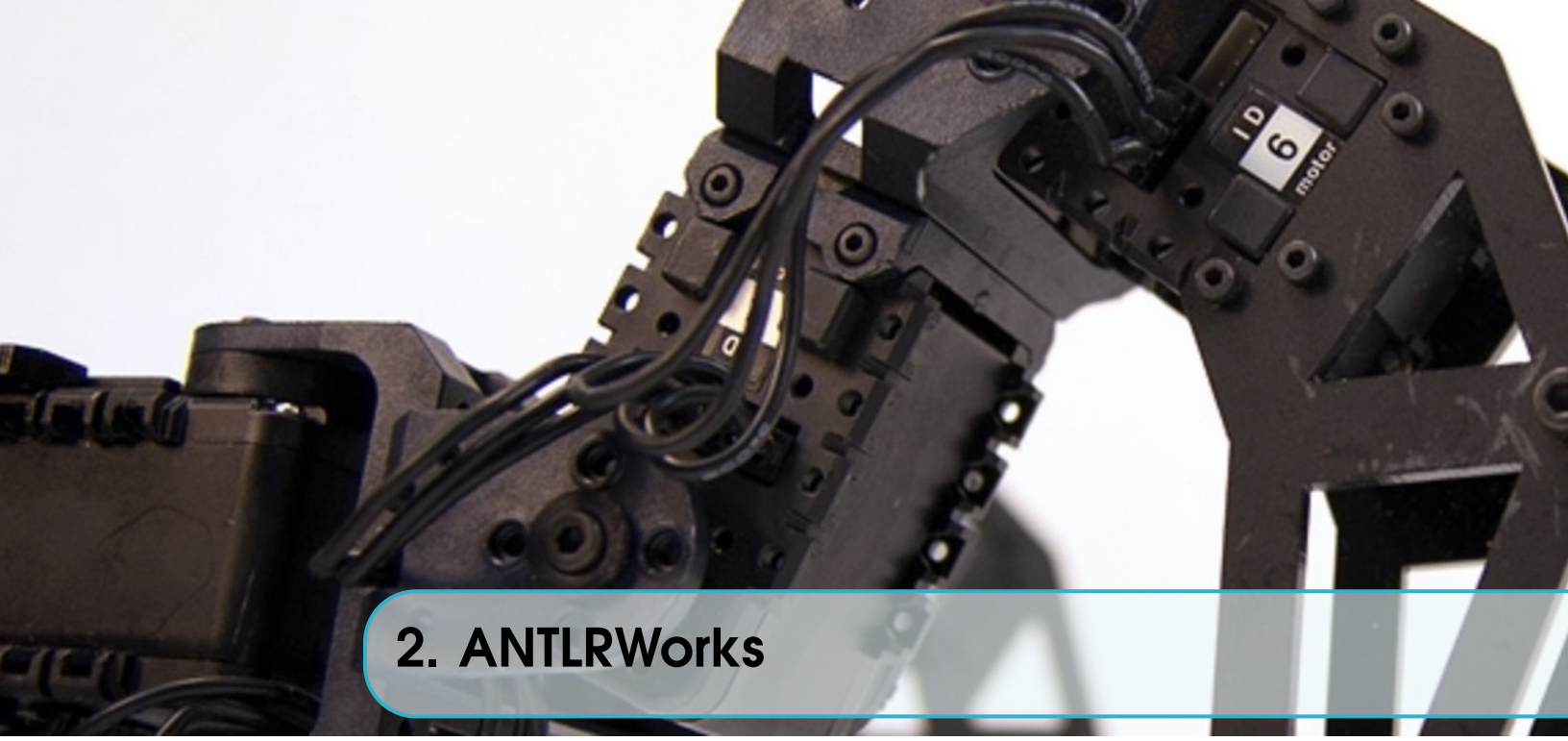


1. Introducción

Un robot hexápodo es un vehículo móvil de seis patas. Debido a que un robot puede ser estáticamente estable con tres o más patas, este tipo de robot posee una alta flexibilidad en cuanto a sus movimientos. Si una pata es deshabilitada, el robot es aún capaz de caminar [Wik15].

En los trabajos anteriores se describió el robot que se iba a desarrollar y la razón de la elección, y se procedió a realizar y mostrar los procedimientos para llevar a cabo el mismo. Se desarrolló en ellos el modelo cinemático y dinámico del robot utilizando distintas herramientas matemáticas y computacionales, y se planteó una forma de controlar los motores del robot en forma simultánea para que éste pueda realizar las acciones de movimiento que se requieran de él, lo que se logró mediante VHDL apuntado a usar como target una FPGA [BSa14].

El objetivo de este informe es desarrollar un lenguaje de ejecución para el análisis cinemático de un robot Hexápodo a través de un compilador de instrucciones desarrollado por Terence Parr y que se llama ANTLR.



2. ANTLRWorks

ANTLRWorks en un entorno de desarrollo con interfaz de usuario gráfica para el desarrollo de aplicaciones de ANTLR. Esto consiste en desarrollar, por cada instrucción, un analizador léxico y sintáctico. Para este trabajo se utilizó el ***ANTLRWorks 1.5rc2*** [Par14].

2.1 ANTLR

ANTLR es un programa cuya función es traducir un programa escrito en un lenguaje de programación (normalmente de alto nivel o de usuario) en otro lenguaje (de bajo nivel o de programación del robot)[MSo14].

2.1.1 Scanner(Análisis léxico)

La tarea del Scanner es únicamente verificar si todas las palabras y símbolos contenidos en el archivo fuente pertenecen al lenguaje definido. Para ello, analiza símbolo por símbolo indicando el token por cada uno de los elementos reconocidos, o error en caso contrario. 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 (*variables, funciones, procedimientos, tipos, 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 para poder trabajarlos más fácilmente.

2.1.2 Parser(Análisis sintáctico)

El Parser se encarga de 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.

De este modo, verifica que las sucesiones de tokens provenientes de la etapa anterior tengan sentido en lo que quieren expresar, ya que no todas las agrupaciones de palabras válidas de un lenguaje conforman una sentencia que tenga sentido y exprese algo coherente. En otras palabras, el análisis sintáctico verifica que las cadenas de tokens cumplan con las especificaciones de las reglas sintácticas.

2.1.3 Analizador 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 Parser. 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. A este nivel se lo conoce como *Tree Parsers*.

2.2 Funcionamiento general

El compilador desarrollado en este trabajo se encargará de traducir instrucciones sencillas introducidas por el usuario en instrucciones o códigos más complejos, que en este caso será un archivo llamado de ahora en más ***Output File*** **(OF)** y será leído por un script de MATLAB.

El programa escrito en ANTLR sigue el siguiente esquema:

- grammar $*.g$: cabecera del fichero
- tokens: declaración de los 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

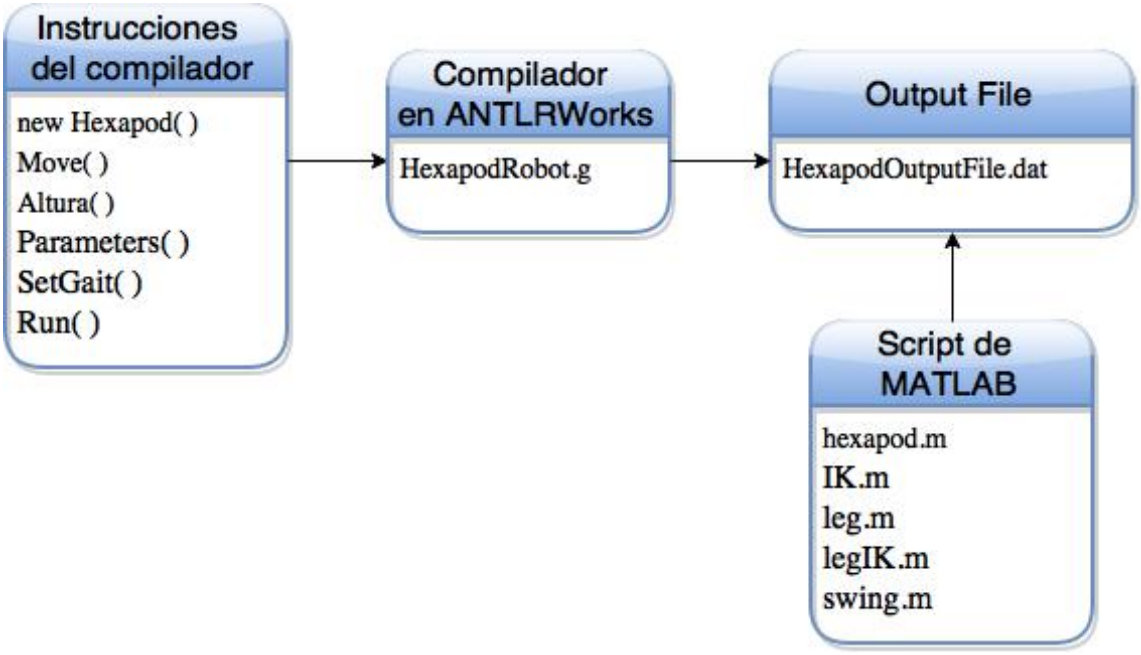
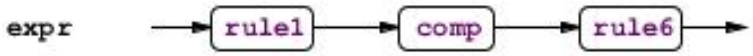


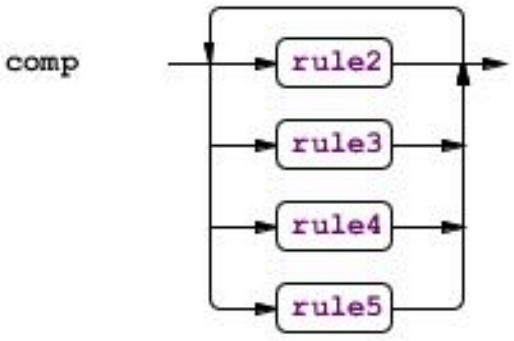
Figure 2.1: Estructura general del compilador en ANTLR y su interacción con MATLAB

2.3 Reglas léxicas y sintácticas

El programa es estricto en cuanto a las instrucciones inicial y final únicamente. A continuación se muestra que las reglas 1 y 6 son las indicadas.



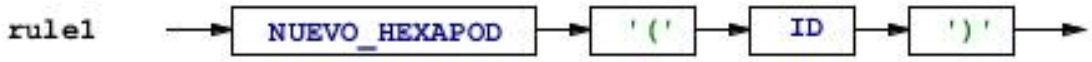
Las reglas *expr* y *comp* solo hacen referencia a otras reglas para que *new Hexapod()* y *Run()* se puedan ejecutar sólo una vez, y todas las reglas intermedias puedan ser ejecutadas cuantas veces se desee y en el orden que se desee. Estas reglas optativas son cuatro (*Move*, *Altura*, *Parameters* y *SetGait*).



A continuación se explican más en detalle las instrucciones.

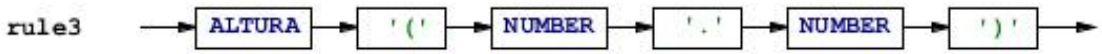
La primer regla es una función que setea los parámetros físicos del robot a simular, o sea, la longitud de las patas, la distancia entre ellas, el tamaño del cuerpo, etc. La instrucción se carga como: ***new Hexapod( )***, y como argumento puede recibir un *string* (sin comillas):

- Espacio en blanco: Carga PhantomX por defecto
- "*PhantomX*"
- "*Mark II*": Ídem PhantomX
- "*Phoenix*"



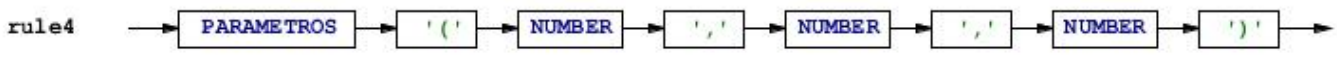
La regla *Move(x,y)* especifica la distancia que se moverá el robot en cada cálculo.

Altura( ) únicamente define la distancia respecto al piso a la que se moverá el hexápodo. Este número posee parte entera y decimal.



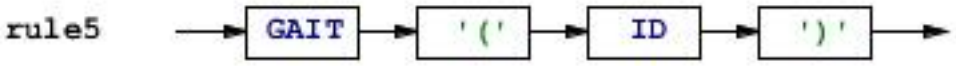
Parameters( ) recibe tres argumentos, que especifican propiedades de ejecución del simulador:

- *dt*
- *tick*
- *StepCount*

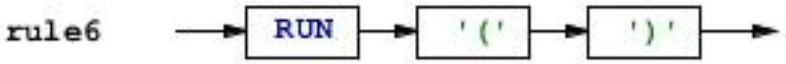


SetGait( ) determina el tipo de marcha que realizará el robot. Para especificarlo, se debe pasar como argumento alguno de las siguientes palabras reservadas:

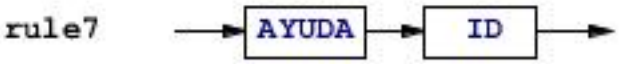
- "*Tripod*": Trípode (3+3)
- "*Ripple*": Riple (4+2)
- "*Wave*": Metacrónico (5+1)



La última palabra que debe colocarse para que compile correctamente es *Run( )*. Esta función comprueba si alguna de las funciones no se utilizó, y carga los valores por defecto en el OF.

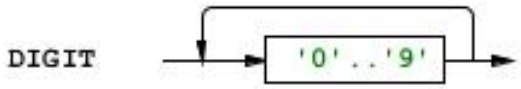


Se ha realizado un manual de ayuda con todas las funciones creadas, pero no se ha podido lograr que funcione correctamente. Para poder utilizarlo se debe escribir *help FUNCIÓN*, donde *FUNCIÓN* debe reemplazarse por algún nombre reservado.

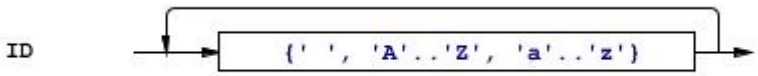


Las reglas léxicas utilizadas son las siguientes:

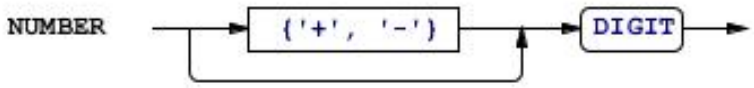
DIGIT se utiliza para especificar números.



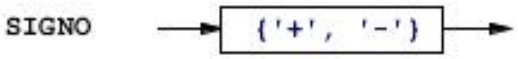
La regla *ID* posee incorporado un espacio en blanco para reconocer varias palabras.



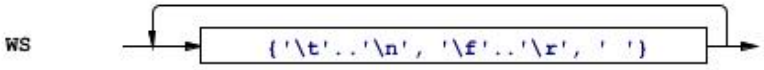
En *NUMBER* pueden, o no, indicarse los signos de un número.



SIGNO es utilizado en *NUMBER* para detectar si los números son negativos.



WS no es explícitamente utilizado, pero se incorporó para futuras implementaciones.



2.4 Instrucciónes de uso

Para utilizar de manera correcta el compilador, a continuación se mostrará un ejemplo.

```
new Hexapod( )
Move(-1,-1)
SetGait(Tripod)
Run()
```

Como puede verse, el orden de la primera y última instrucción son necesarios para que el programa funcione correctamente. Además, se han agregado dos reglas para luego observar cómo es el AST. Además, no se utilizan *;* ni otra simbología especial para detectar el final de una línea.

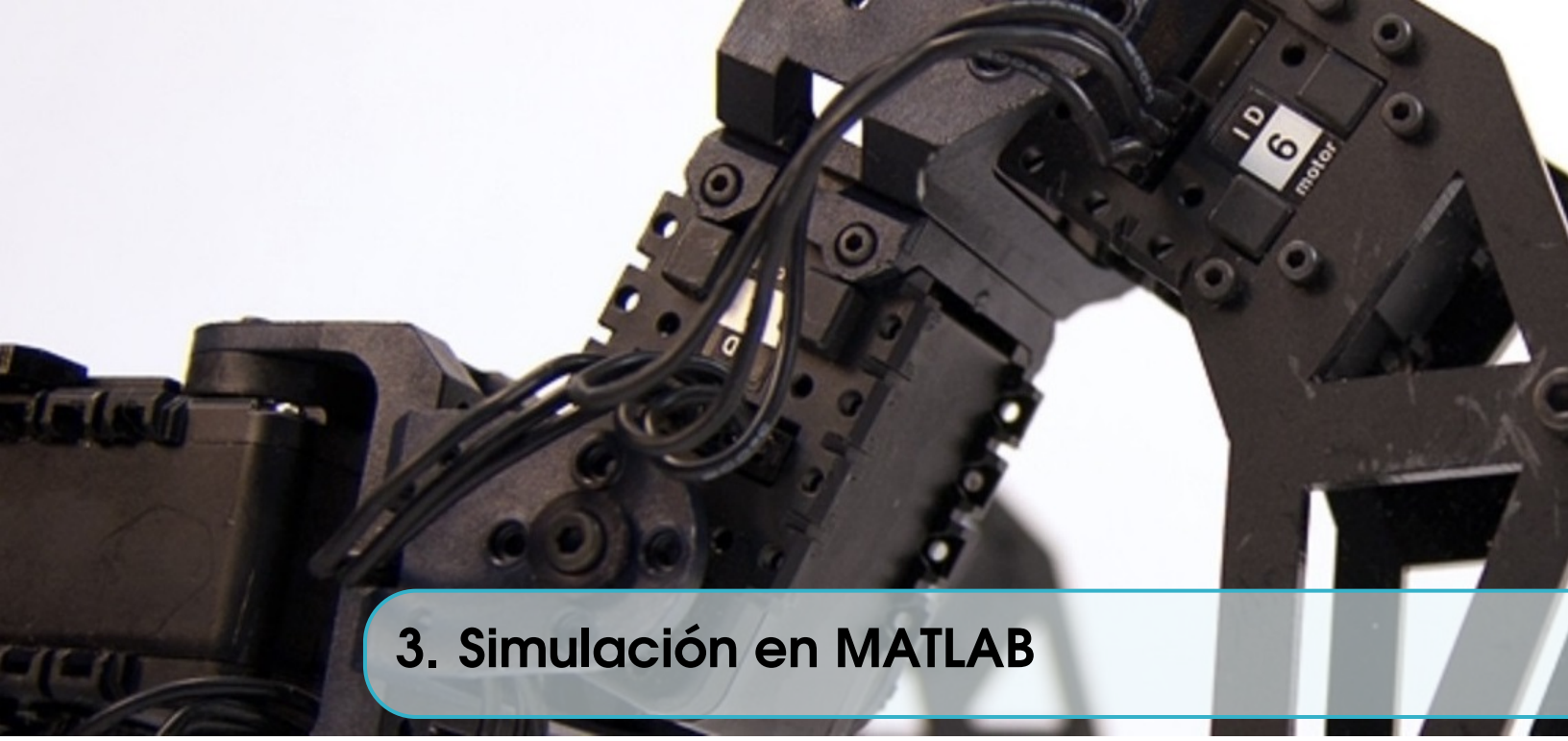
2.4.1 Árbol sintáctico

El árbol sintáctico generado en el ejemplo se muestra en la figura 2.2.

Figure 2.2: Estructura del árbol sintáctico para el ejemplo mostrado

2.4.2 Ejemplo de Output File

```
% HEXAPOD OUTPUT FILE
% Cargando valores por defecto
l1 3.9
l2 6.5
l3 14.2
half_length 14
half_width1 4.45
half_width2 7.2
legdist 12.5
h 2
X 1
Y 0
Z 10.0
dt 10
tick 1
stepCount 5
gait 1
% END OF HEXAPOD OUTPUT FILE
```



3. Simulación en MATLAB

3.1 Scripts

El programa de MATLAB encargado de leer el *Output File* consiste en una serie de Scripts que se listan a continuación:

- *hexapod.m*: Script principal. Lee el OF y asigna los parámetros a variables.
- *IK.m*: calcula la cinemática inversa de una pata.
- *leg.m*: calcula la cinemática directa.
- *legIK.m*: maneja los scripts de IK y FK.
- *swing.m*: genera la trayectoria cicloidea para las patas.

3.2 Animaciones obtenidas

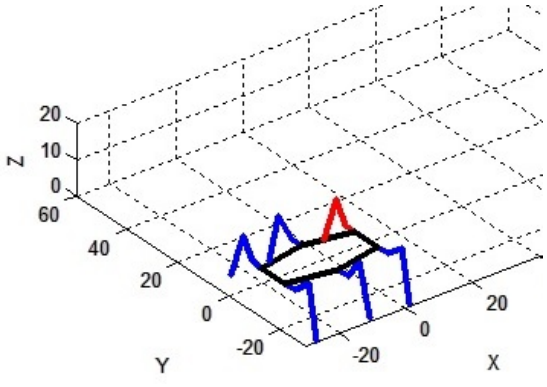


Figure 3.1: PhantomX Hexapod Mark II

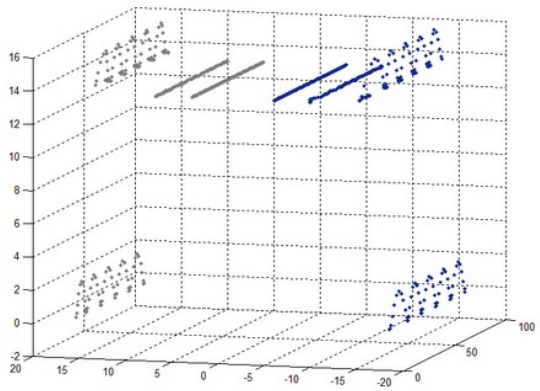
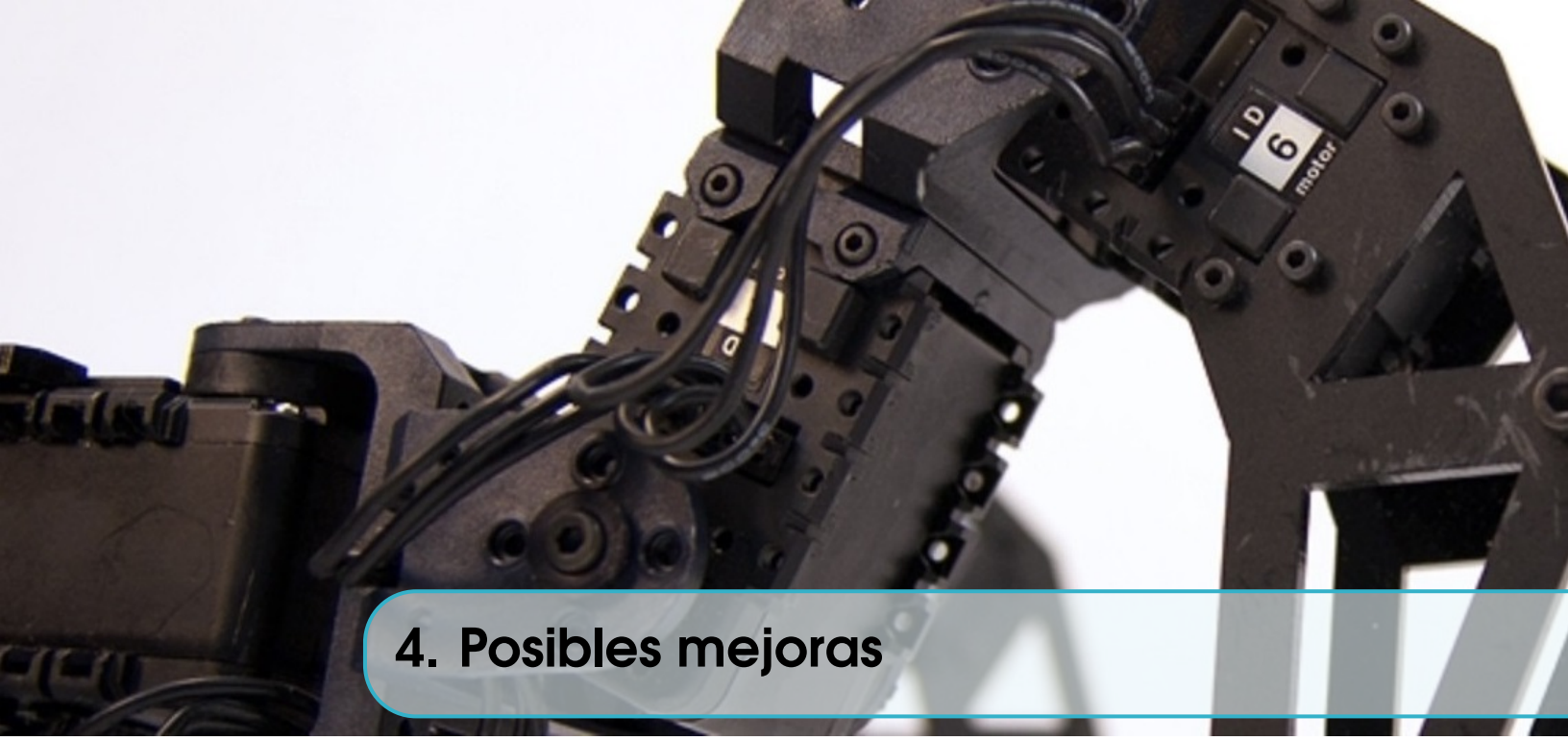


Figure 3.2: Posiciones de las articulaciones

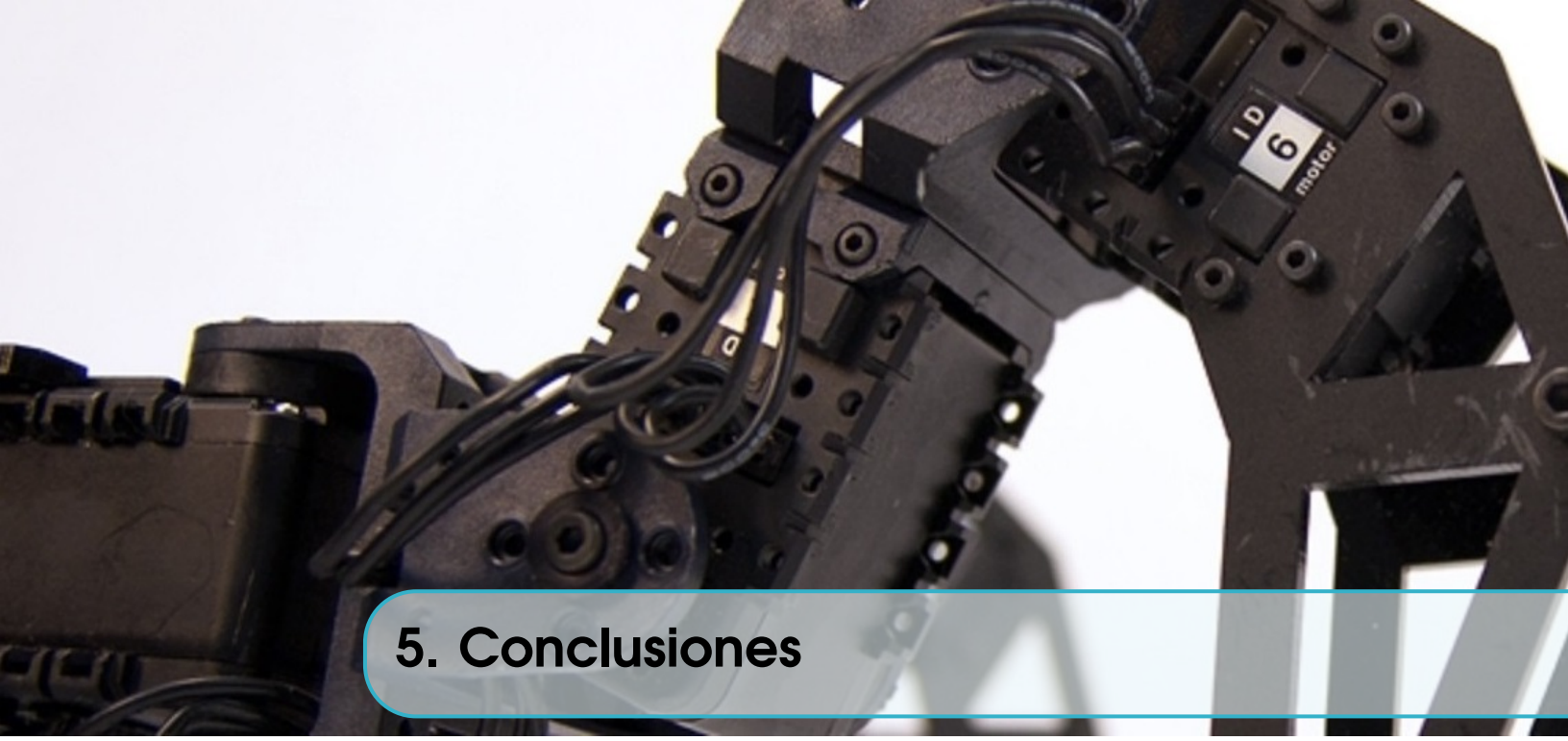


4. Posibles mejoras

A continuación se listarán algunas de las posibles mejoras que podrían realizarse al trabajo realizado.

Respecto al compilador en ANTLR, la primera mejora a implementar sería la de resolver el inconveniente de incorporar la instrucción de ayuda (*help*). Para mejorar aún más la personalización de la simulación de MATLAB podrían agregarse más funciones, pero para ello, también es necesario realizar cambios en los scripts de MATLAB.

En cuanto a este último programa, convendría agregarle la simulación dinámica del robot (contemplar los torques que se ejercen sobre los servomotores modelados en el TP2).



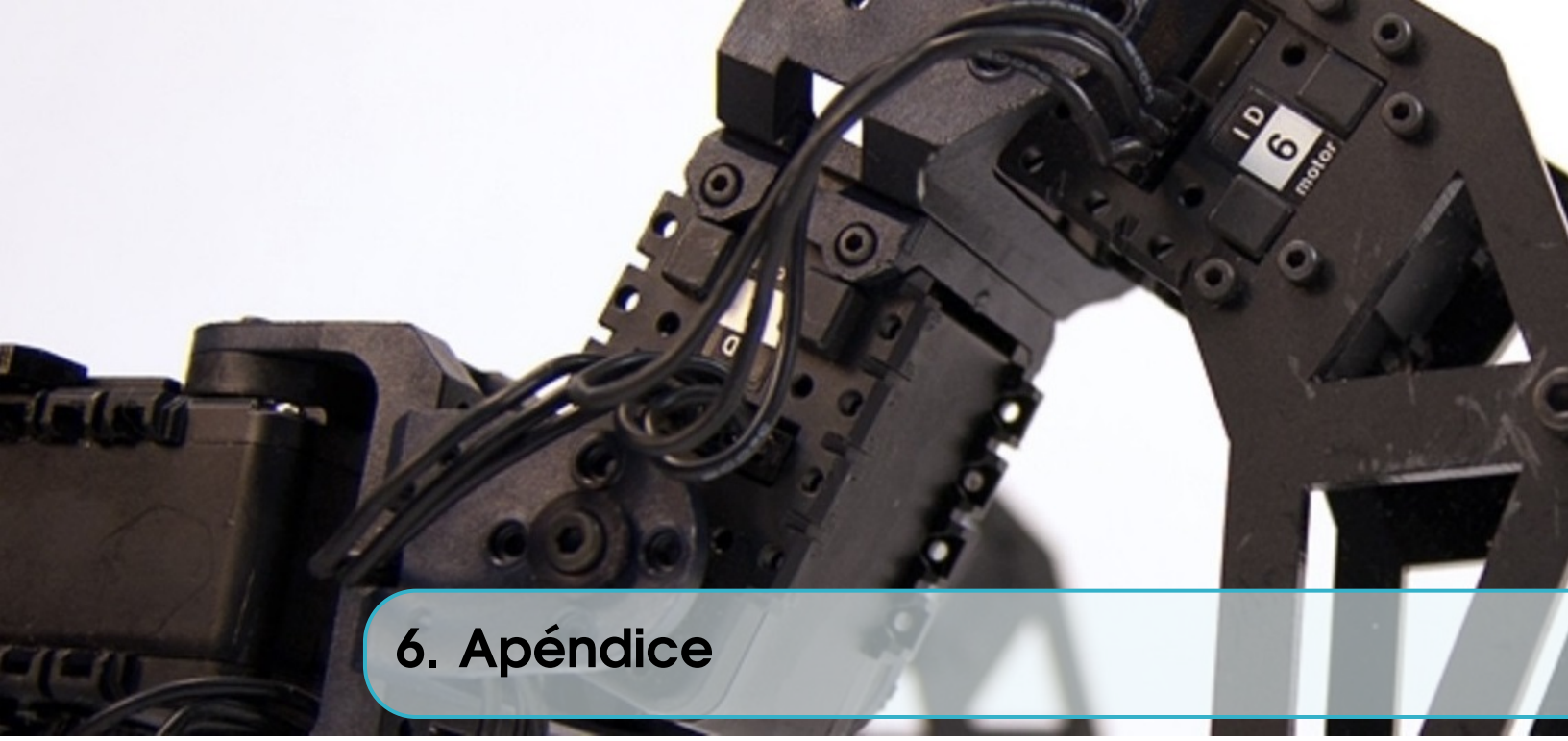
5. Conclusiones

Se trabajó siguiendo la filosofía de la cátedra según la cual el esfuerzo debe ser acumulativo y no repetitivo.

Se ha logrado obtener un compilador simple pero adecuado a las necesidades del robot utilizado. Se realizaron varias instrucciones básicas de movimiento del robot y se verificó el funcionamiento de las mismas a través de la simulación. Si se quisiera llevar el robot a la práctica ya se cuenta con la cinemática, la dinámica y el compilador realizados en los informes de la cursada, por lo que solo resta realizarlo y afrontar los inconvenientes que pueden surgir en la realización del mismo.

Cabe destacar que se trabajó con una herramienta muy potente, ya que permite poner una capa de abstracción muy importante y permitiendo hacer las cosas más sencillas para el usuario. Por lo tanto lo tomamos como algo más para hacer más fácil la realización de los proyectos.

Se pudo analizar de manera satisfactoria la cinemática simplificada de un robot hexápodo.



6. Apéndice

6.1 Código de ANTLR

```
grammar HexapodRobot;

////////////////////////////////////////////////////////////////////////
// Simbolos reconocidos por el compilador
//

tokens {
  NUEVO_HEXAPOD = 'new Hexapod';
  MOVE = 'Move';
  ALTURA = 'Altura';
  PARAMETROS = 'Parameters';
  GAIT = 'SetGait';
  RUN = 'Run';
  AYUDA = 'help';
}

////////////////////////////////////////////////////////////////////////
// Librerias y clases utilizadas dentro del compilador
//

@header {
import java.util.HashMap;
import java.io.*;

}

@members {
// Variables externas
    public static String path = "C:\\Users\\Emiliano Borghi\\Desktop\\
    Robotica TP Final\\";
    public static File file = new File(path + "HexapodOutputFile.dat");
```

```
31      // http://www.mkyong.com/java/how-to-write-to-file-in-java-bufferedwriter
        -example/
32      public FileWriter fw;
33      public BufferedWriter bw;
34
35      public int I; // Gait variable
36
37      // Flags para saber si se ejecuto la regla
38      public boolean fRule1=false, fRule2=false, fRule3=false, fRule4=false,
        fRule5=false;
39
40      HashMap variables = new HashMap();
41
42      public static void main(String[] args) throws Exception {
43          HexapodRobotLexer lex = new HexapodRobotLexer(new ANTLRFileStream(
        args[0]));
44          CommonTokenStream tokens = new CommonTokenStream(lex);
45
46          HexapodRobotParser parser = new HexapodRobotParser(tokens);
47
48          try {
49              parser.expr();
50          } catch (RecognitionException e)  {
51              e.printStackTrace();
52          }
53      }
54  }
55
56  ////////////////////////////////////////////////////////////////////
57  // LEXER RULES
58  //
59
60  expr   : rule1 comp rule6;
61  // TODO: Agregar regla de ayuda (help).
62
63  comp   : ( rule2 | rule3 | rule4 | rule5)+;
64
65  rule1
66    :   NUEVO_HEXAPOD '('  ID ')' { // | NUEVO_HEXAPOD '(' ')' { // ID ')' {
67
68      System.out.println("NUEVO HEXAPODO CREADO");
69
70    try {
71        // Si el archivo no existe, lo crea
72        if (!file.exists())    file.createNewFile();
73        fw = new FileWriter(file.getAbsoluteFile());
74        bw = new BufferedWriter(fw);
75        bw.write("\% HEXAPOD OUTPUT FILE \n");
76
77        if( $ID.text.equals(" ") || $ID.text.equals("Phoenix") ){
78
79            if($ID.text.equals(" ")) bw.write("\% Cargando valores por defecto \
        n");
80            else bw.write("\% PhantomX Hexapod Mark II \n");
81
82            bw.write("l1 3.9 \n");
```

```
83            bw.write("l2 6.5 \n");
84            bw.write("l3 14.2 \n");
85            bw.write("half_length 14 \n");
86            bw.write("half_width1 4.45 \n");
87            bw.write("half_width2 7.2 \n");
88            bw.write("legdist 12.5 \n");
89            bw.write("h 2 \n");
90          }
91          else if( $ID.text.equals("Mark II") || $ID.text.equals("Phantom X") ||
        $ID.text.equals(-1) ){
92              System.out.println("Error Hexapodo no implementado aun.");
93          }
94          else System.out.println("Error pasando parametro a Hexapod()");
95
96        bw.flush();
97       } catch (IOException e) {
98            e.printStackTrace();
99        }
100     };
101
102   rule2
103     : MOVE '(' xPos=NUMBER ',' yPos=NUMBER ')' {
104       // MUEVE AL ROBOT EN X e Y
105       try{
106         bw.write( "X " + $xPos.int + '\n');
107         bw.write( "Y " + $yPos.int + '\n');
108         bw.flush();
109         fRule2=true;
110       }catch(IOException e){
111         e.printStackTrace();
112       }
113     };
114
115   rule3
116     : ALTURA '(' parteEntera = NUMBER '.' parteDecimal = NUMBER ')' {
117   // TODO: Considerar no agregar parte decimal
118       try{
119         bw.write("Z " + $parteEntera.int + '.' + $parteDecimal.int + '\n');
120         bw.flush();
121         fRule3=true;
122       }
123       catch (IOException e){
124         e.printStackTrace();
125       }
126     };
127
128   rule4
129     :  PARAMETROS '(' dt = NUMBER ',' tick = NUMBER ',' stepCount = NUMBER ')'
        {
130       try{
131         bw.write("dt " + $dt.int + '\n');
132         bw.write("tick " + $tick.int + '\n');
133         bw.write("stepCount " + $stepCount.int + '\n');
134         bw.flush();
135         fRule4=true;
136       }
```

```
137     catch (IOException e){
138       e.printStackTrace();
139     }
140   };
141
142 rule5
143   : GAIT '(' ID ')' {
144     try{
145       switch( $ID.text ){
146         case "Tripod":
147           I = 1;
148           break;
149         case "Ripple":
150           I = 2;
151           break;
152         case "Wave":
153           I = 3;
154           break;
155       }
156       bw.write("gait " + Integer.toString(I) + '\n');
157       bw.flush();
158       fRule5=true;
159     }
160     catch (IOException e){
161       e.printStackTrace();
162     }
163   };
164
165 rule6
166   : RUN '(' ')' {
167   // TODO: Implementar la comunicacion en MATLAB aca - NO CREO QUE PUEDA
      EJECUTARSE UN SCRIPT DESDE ACA
168     try{
169       if( fRule2 == false ){
170         bw.write("X 1 \n");
171         bw.write("Y 0 \n");
172       }
173       if( fRule3 == false ){
174         bw.write("Z 10.0 \n");
175       }
176       if( fRule4 == false ){
177         bw.write("dt 10 \n");
178         bw.write("tick 1 \n");
179         bw.write("stepCount 5 \n");
180       }
181       if( fRule5 == false ){
182         bw.write("gait 1 \n");
183       }
184       bw.write("\% END OF HEXAPOD OUTPUT FILE");
185       bw.flush();
186       bw.close();
187     }
188     catch (IOException e){
189       e.printStackTrace();
190     }
191   };
```

```
192
193 rule7
194   : AYUDA ID {
195   // Help Manual del programa
196     switch( $ID.text ){
197       case "new Hexapod":
198         System.out.println(
199           "\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\%\n" +
200           "==========New Hexapod==========\n" +
201           "Setea parametros fisicos del robot\n" +
202           "Modo de uso:\n" +
203           "new Hexapod(*) \n" +
204           " * Espacio en blanco" +
205           " * Nombres reservados:" +
206           "\t PhantomX" +
207           "\t Mark II"
208         );
209         break;
210       case "Move":
211         System.out.println("\nMOVE FUNCTION HELP\n");
212         break;
213       case "Altura":
214         break;
215       case "Parameters":
216         break;
217       case "SetGait":
218         break;
219       case "Run":
220         break;
221       default:
222         System.out.println("\nFuncion no encontrada\n");
223         break;
224     }
225   };
226
227 /*------------------------------------------------------------------
228  * LEXER RULES
229  *------------------------------------------------------------------*/
230
231 ID   :    ('a'..'z'|'A'..'Z'|' ')+ ;
232
233 NUMBER   : SIGNO? DIGIT ;
234
235 WS : ( '\t' | ' ' | '\r' | '\n'| '\u000C' )+  { $channel = HIDDEN; } ;   //
    WhiteSpace
236
237 fragment DIGIT   : ('0'..'9')+ ;
238
239 SIGNO    : '+' | '-';
```

6.2 Código de MATLAB

```
1 % Variables globales
2 % Se usan a traves de todos los archivos
```

```
3
4  clear all
5
6  global grados;
7
8  global l1;
9  global l2;
10 global l3;
11 global pivot;
12 global half_length;
13 global half_width1;
14 global half_width2;
15
16 global t1;
17 global t2;
18 global t3;
19 global t4;
20 global t5;
21 global t6;
22
23 global swing1;
24 global swing2;
25 global swing3;
26 global swing4;
27 global swing5;
28 global swing6;
29
30 global dt;
31 grados = [];
32
33 %%
34 % LECTURA DEL HEXAPOD OUTPUT FILE Y ASIGNACION DE VARIABLES
35
36 archivo = fopen('HexapodOutputFile.dat');
37
38 % http://www.mathworks.com/help/matlab/ref/textscan.html
39 A = textscan( archivo, '%s %f' , 'CommentStyle', '%');%, 'Delimiter', '\n' );
40
41 for i=1:length(A{:,1})
42     S = char(A{1,1}(i));
43     switch S
44         case 'l1'
45             l1 = A{1,2}(i);
46         case 'l2'
47             l2 = A{1,2}(i);
48         case 'l3'
49             l3 = A{1,2}(i);
50         case 'half_length'
51             half_length = A{1,2}(i);
52         case 'half_width1'
53             half_width1 = A{1,2}(i);
54         case 'half_width2'
55             half_width2 = A{1,2}(i);
56         case 'legdist'
57             legdist = A{1,2}(i);
58         case 'h'
```

```
59              h = A{1,2}(i);  % Altura que se eleva la pasa para dar un paso
60          case 'X'
61              control(1) = A{1,2}(i);
62          case 'Y'
63              control(2) = A{1,2}(i);
64          case 'Z'
65              control(3) = A{1,2}(i);
66          case 'dt'
67              dt = A{1,2}(i);
68          case 'tick'
69              tick = A{1,2}(i); % dt/tick calculations per swing
70          case 'stepCount'
71              stepcount = A{1,2}(i);
72          case 'gait'
73              gait = A{1,2}(i); % 1: Tripod Gait
74                                % 2: Ripple Gait
75                                % 3: Metachronal Wave Gait
76          otherwise
77              disp('Error de asignacion');
78      end
79  end
80
81  % Cierra el archivo abierto
82  fclose(archivo);
83
84  %%
85  % Vector de control de la marcha del Hexapodo
86  control(4) = 3.5;% Distancia del paso [cm]
87
88  pivot = [0 ; 0 ; control(3)];   % Posicion inicial del robot
89
90  time = 0;
91
92  Npos = [0 ; legdist ; 0];
93
94  % Inicializo variables en cero
95  x1 = 0; x2 = 0; x3 = 0; x4 = 0; x5 = 0; x6 = 0;
96  y1 = 0; y2 = 0; y3 = 0; y4 = 0; y5 = 0; y6 = 0;
97  z1 = 0; z2 = 0; z3 = 0; z4 = 0; z5 = 0; z6 = 0;
98
99  swing1 = 0; swing2 = 0; swing3 = 0; swing4 = 0; swing5 = 0; swing6 = 0;
100
101 oldgait = gait;
102
103 switch gait
104     case 1          % Tripod Gait
105         period = 2; % Cantidad de movimientos distintos
106         t1 = 0;
107         t2 = dt;
108         t3 = 0;
109         t4 = dt;
110         t5 = 0;
111         t6 = dt;
112     case 2
113         period = 3;
114         t1 = 0;
```

```
115         t2 = 2*dt;
116         t3 = dt;
117         t4 = 0.5*dt;
118         t5 = 2.5*dt;
119         t6 = 1.5*dt;
120     case 3
121         period = 6;
122         t1 = 0;
123         t2 = 5*dt;
124         t3 = 4*dt;
125         t4 = 3*dt;
126         t5 = 2*dt;
127         t6 = dt;
128 end
129
130 change = 0;
131
132 while time <= period*dt*stepcount
133     % Calcula el objetivo proximo (X,Y) normalizado.
134     norm1 = sqrt(control(1)^2 + control(2)^2);
135     X = control(1)/norm1;
136     Y = control(2)/norm1;
137     % Distancia del paso [cm]
138     s = control(4);
139
140     if(gait - oldgait) ~= 0
141         switch gait
142             case 1
143                 period = 2;
144                 t1 = 0;
145                 t2 = dt;
146                 t3 = 0;
147                 t4 = dt;
148                 t5 = 0;
149                 t6 = dt;
150             case 2
151                 period = 3;
152                 t1 = 0;
153                 t2 = 2*dt;
154                 t3 = dt;
155                 t4 = 0.5*dt;
156                 t5 = 2.5*dt;
157                 t6 = 1.5*dt;
158             case 3
159                 period = 6;
160                 t1 = 0;
161                 t2 = 5*dt;
162                 t3 = 4*dt;
163                 t4 = 3*dt;
164                 t5 = 2*dt;
165                 t6 = dt;
166         end
167
168         swing1 = 0;
169         swing2 = 0;
170         swing3 = 0;
```

```
171         swing4 = 0;
172         swing5 = 0;
173         swing6 = 0;
174     end
175
176     oldgait = gait;
177
178     %% step calculations
179     tickstep = tick/(dt*(period-1)); % change per tick
180
181     if t1 < dt
182         if(swing1 == 0)
183             Pold1 = [x1 ; y1 ; z1];
184             Pnew1 = s * [X ; Y ; 0];
185             swing1 = 1;
186         end
187         [x1, y1, z1] = swing(t1, dt, Pold1, Pnew1, h);
188     else
189         x1 = x1 - tickstep*2*s*X;
190         y1 = y1 - tickstep*2*s*Y;
191         z1 = 0;
192         swing1 = 0;
193     end
194
195     if t2 < dt
196         if(swing2 == 0)
197             Pold2 = [x2 ; y2 ; z2];
198             Pnew2 = s * [X ; Y ; 0];
199             swing2 = 1;
200         end
201         [x2, y2, z2] = swing(t2, dt, Pold2, Pnew2, h);
202     else
203         x2 = x2 - tickstep*2*s*X;
204         y2 = y2 - tickstep*2*s*Y;
205         z2 = 0;
206         swing2 = 0;
207     end
208
209     if t3 < dt
210         if(swing3 == 0)
211             Pold3 = [x3 ; y3 ; z3];
212             Pnew3 = s * [X ; Y ; 0];
213             swing3 = 1;
214         end
215         [x3, y3, z3] = swing(t3, dt, Pold3, Pnew3, h);
216     else
217         x3 = x3 - tickstep*2*s*X;
218         y3 = y3 - tickstep*2*s*Y;
219         z3 = 0;
220         swing3 = 0;
221     end
222
223     if t4 < dt
224         if(swing4 == 0)
225             Pold4 = [x4 ; y4 ; z4];
226             Pnew4 = s * [X ; Y ; 0];
```

```
227             swing4 = 1;
228         end
229         [x4, y4, z4] = swing(t4, dt, Pold4, Pnew4, h);
230     else
231         x4 = x4 - tickstep*2*s*X;
232         y4 = y4 - tickstep*2*s*Y;
233         z4 = 0;
234         swing4 = 0;
235     end
236
237     if t5 < dt
238         if(swing5 == 0)
239             Pold5 = [x5 ; y5 ; z5];
240             Pnew5 = s * [X ; Y ; 0];
241             swing5 = 1;
242         end
243         [x5, y5, z5] = swing(t5, dt, Pold5, Pnew5, h);
244     else
245         x5 = x5 - tickstep*2*s*X;
246         y5 = y5 - tickstep*2*s*Y;
247         z5 = 0;
248         swing5 = 0;
249     end
250
251     if t6 < dt
252         if(swing6 == 0)
253             Pold6 = [x6 ; y6 ; z6];
254             Pnew6 = s * [X ; Y ; 0];
255             swing6 = 1;
256         end
257         [x6, y6, z6] = swing(t6, dt, Pold6, Pnew6, h);
258     else
259         x6 = x6 - tickstep*2*s*X;
260         y6 = y6 - tickstep*2*s*Y;
261         z6 = 0;
262         swing6 = 0;
263     end
264
265     pivot = pivot + [X ; Y ; 0] * tickstep*2*s;
266
267     %% time calculations
268     t1 = t1 + tick;
269     t2 = t2 + tick;
270     t3 = t3 + tick;
271     t4 = t4 + tick;
272     t5 = t5 + tick;
273     t6 = t6 + tick;
274     time = time + tick;
275
276     if(t1 >= period*dt)
277         t1 = t1 - period*dt;
278     end
279
280     if(t2 >= period*dt)
281         t2 = t2 - period*dt;
282     end
```

```
283
284     if(t3 >= period*dt)
285         t3 = t3 - period*dt;
286     end
287
288     if(t4 >= period*dt)
289         t4 = t4 - period*dt;
290     end
291
292     if(t5 >= period*dt)
293         t5 = t5 - period*dt;
294     end
295
296     if(t6 >= period*dt)
297         t6 = t6 - period*dt;
298     end
299
300     %% Cuerpo
301
302     B = [ half_length + pivot(1)   half_width1 + pivot(2)  pivot(3);...
303           pivot(1)                 half_width2 + pivot(2)  pivot(3);...
304          -half_length + pivot(1)   half_width1 + pivot(2)  pivot(3);...
305           half_length + pivot(1)  -half_width1 + pivot(2)  pivot(3);...
306           pivot(1)                -half_width2 + pivot(2)  pivot(3);...
307          -half_length + pivot(1)  -half_width1 + pivot(2)  pivot(3)];
308
309     body = [ B(1,:);... % second body definition to draw properly in the
    diagram
310              B(2,:);...
311              B(3,:);...
312              B(6,:);...
313              B(5,:);...
314              B(4,:);...
315              B(1,:)];
316
317     B1 = B(1,:)'; % body coxa joints
318     B2 = B(2,:)';
319     B3 = B(3,:)';
320     B4 = B(4,:)';
321     B5 = B(5,:)';
322     B6 = B(6,:)';
323
324     %% Patas
325     % Extremos del robot, relativos a cada articulacion del Coxis
326     P1 = [x1 ; y1 ; z1] + Npos - [0;0;pivot(3)];
327     P2 = [x2 ; y2 ; z2] + Npos - [0;0;pivot(3)];
328     P3 = [x3 ; y3 ; z3] + Npos - [0;0;pivot(3)];
329     P4 = [x4 ; -y4 ; z4] + Npos - [0;0;pivot(3)];
330     P5 = [x5 ; -y5 ; z5] + Npos - [0;0;pivot(3)];
331     P6 = [x6 ; -y6 ; z6] + Npos - [0;0;pivot(3)];
332
333     leg1 = legIK(P1+B1,B1);
334     leg2 = legIK(P2+B2,B2);
335     leg3 = legIK(P3+B3,B3);
336     leg4 = legIK(P4+B4,B4);
337     leg5 = legIK(P5+B5,B5);
```

```
338     leg6 = legIK(P6+B6,B6);
339
340     %% visualization
341
342     plot3(leg1(:,1),leg1(:,2),leg1(:,3),'- .r',...
343                 leg2(:,1),leg2(:,2),leg2(:,3),'- .b',...
344                 leg3(:,1),leg3(:,2),leg3(:,3),'- .b',...
345                 leg4(:,1),leg4(:,2),leg4(:,3),'- .b',...
346                 leg5(:,1),leg5(:,2),leg5(:,3),'- .b',...
347                 leg6(:,1),leg6(:,2),leg6(:,3),'- .b',...
348                 body(:,1),body(:,2),body(:,3),'- .k','LineWidth',3)
349
350     axis equal
351     grid on
352     xlabel('X')
353     ylabel('Y')
354     zlabel('Z')
355     xlim([-30 100])  % AUTOAJUSTAR!!
356     ylim([-30 60])
357     zlim([0 20])
358     title(time)
359
360     drawnow
361 end
```

Codigo/hexapod.m

```
1  function [ degs ] = IK(P,P0)
2      %% Cinematica Inversa - Inverse Kinematics (IK)
3
4      % Usa acosd() en vez de atan2(). Para ello, tiene en cuenta el signo de
5      % la articulacion 'q'.
6    global grados;
7      global l1;
8      global l2;
9      global l3;
10
11     p = P(2) - P0(2);        % Adyacente
12     q = P(1) - P0(1);
13     t = sqrt(p^2 + q^2);     % Hipotenusa
14
15     if t == 0
16         a1 = 0;
17     else
18         if q >= 0
19             a1 = acosd(p/t);     % cos() = Adj/Hyp
20         else
21             a1 = -acosd(p/t);
22         end
23     end
24
25     P1 = P0 + l1 * [sind(a1 ); cosd(a1) ; 0];
26
27     u = P - P1;
28     r = sqrt(u(1)^2 + u(2)^2);
29     s = sqrt(u(1)^2 + u(2)^2 + u(3)^2);
```

```
30
31      v = (l2^2 + s^2 - l3^2) / (2 * l2 * s);
32
33      if u(3) < 0
34          a2 = acosd(v) - acosd(r/s);
35      else
36          a2 = acosd(v) + acosd(r/s);
37      end
38
39      % Aplicando la Ley de los Cosenos
40      a3 = 180 - acosd((l2^2 + l3^2 - s^2) / (2 * l2 * l3));
41
42      degs = [a1 ; a2 ; a3];  % Vector retornado por IK()
43
44      grados = [grados;degs'];
45  end
```

Codigo/IK.m

```
1   function [ leg ] = leg( a1,a2,a3,P0 )
2
3       % Cinematica Directa - Forward Kinematics (FK)
4
5       global l1;
6       global l2;
7       global l3;
8       global pivot;
9
10      % sind() = sin() pero con el argumento en grados!
11
12      if P0(2) > pivot(2)
13
14          P1 = P0 + l1 * [sind(a1) ; cosd(a1) ; 0];
15          P2 = P1 + l2 * [sind(a1)*cosd(a2) ; cosd(a1)*cosd(a2) ; sind(a2)];
16          P3 = P2 + l3 * [sind(a1)*cosd(a2-a3) ; cosd(a1)*cosd(a2-a3) ; sind(a2
        -a3)];
17
18      else
19
20          P1 = P0 + l1 * [sind(a1) ; -cosd(a1) ; 0];
21          P2 = P1 + l2 * [sind(a1)*cosd(a2) ; -cosd(a1)*cosd(a2) ; sind(a2)];
22          P3 = P2 + l3 * [sind(a1)*cosd(a2-a3) ; -cosd(a1)*cosd(a2-a3) ; sind(
        a2-a3)];
23
24      end
25
26      leg = [P0(1) P0(2) P0(3);...    % Matriz retornada
27             P1(1) P1(2) P1(3);...
28             P2(1) P2(2) P2(3);...
29             P3(1) P3(2) P3(3)];
30
31  end
```

Codigo/leg.m

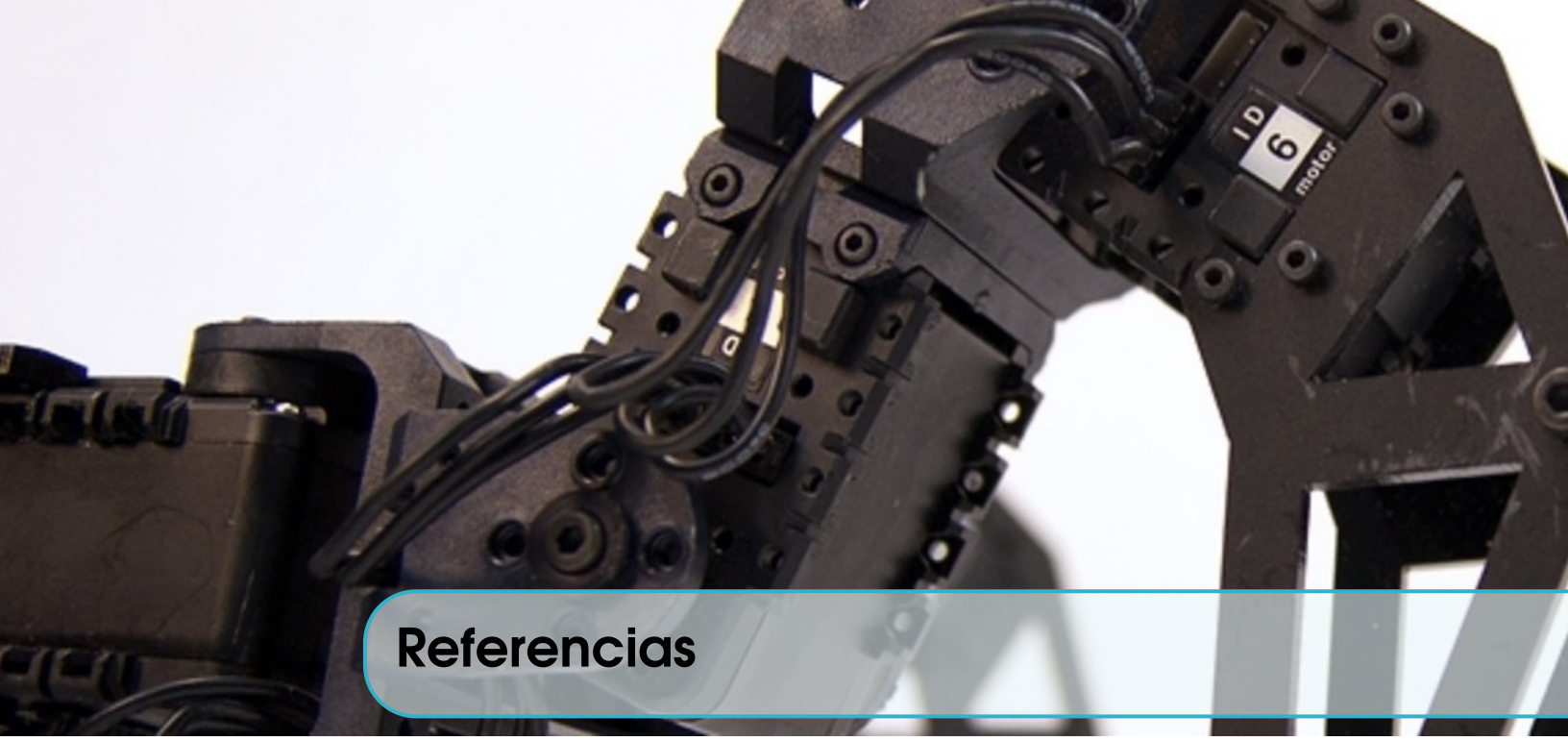
```
1   function [ legout ] = legIK(P,P0)
```

```
    %% Combina FK e IK para dibujar todo en el diagrama

    a = IK(P,P0);
    legout = leg(a(1),a(2),a(3),P0);
    % Retorna la matriz de FK
end
```

Codigo/legIK.m

```
function [ x, y, z ] = swing(t, dt, P1, P2, h)

    % Esta funcion realiza una trayectoria CICLOIDEA cuando gira.

    % X
    ax = (P2(1) - P1(1)) / 2;
    nx = (P2(1) + P1(1)) / 2;

    x = nx - ax * cos(t/dt * pi);

    % Y
    ay = (P2(2) - P1(2)) / 2;
    ny = (P2(2) + P1(2)) / 2;

    y = ny - ay * cos(t/dt * pi);

    % Z
    z = h/2 * (1 - cos(t/dt * 2*pi)) + (P2(3) - P1(3))*t/dt;
end
```

Codigo/swing.m



Referencias

[BSa14] M.Samez B.Samudio Cáceres. “Trabajo Práctico 3.B.” In: *UTN Bs.As.* (2014).

[MSo14] M.Focaraccio M.Sokolowicz. “Trabajo Práctico Final-Compilador”. In: *UTN Bs.As.* (2014).

[Par14] Terence Parr. *ANTLR*. 2014. URL: `http://www.antlr.org/`.

[Wik15] Wikipedia. *Hexapod (robotics)*. 2015. URL: `https://en.wikipedia.org/wiki/Hexapod_(robotics)`.