Lenguajes de programación de robots industriales

Una perspectiva histórica: del Control Numérico a los frameworks robóticos.

Versión 1.0. Diciembre 2018.

Fernando Comíns Tello

Índice

Versiones del documento

Versión	Fecha	Novedades
1.0	Dic-2018	Versión inicial: las tres etapas; taxonomía general, y otras clasificaciones. Lenguajes comentados: 172 Otros lenguajes mencionados: 20 Referencias bibliográficas: 110 Ejemplos de programas: 21

I. Introducción

La idea

Hace ya mucho tiempo, bastantes años atrás, sentí curiosidad por obtener información sobre la programación de robots industriales. Busqué en Internet y me sorprendió la escasa información elaborada que encontré; pocos textos repetidos una y otra vez. Algunos trabajos recopilatorios se publicaron en los años 80, pero nada por el estilo posteriormente.

Así que me autoimpuse la tarea de buscar información suelta, buscando libros, documentos, manuales, todo lo que pudiera encontrar; y elaborar un texto recopilatorio que contuviera:

- una taxonomía o clasificación de los lenguajes de programación de robots.
- una descripción de cada uno de los lenguajes
- una ubicación temporal de los lenguajes, de modo que pudiera construir una línea del tiempo de los lenguajes.

Y todo ello con una perspectiva histórica, desde los primeros lenguajes en la segunda mitad de los años 50 del siglo XX hasta la actualidad.

Otra idea clave en este trabajo fue que prácticamente todo lo que se plasmara en el documento estuviera extraído de las fuentes encontradas; así que el posterior texto está formado en su mayor parte por trozos de textos debidamente buscados, hallados, en algún caso traducidos, fusionados, ordenados,… excepto esta introducción y el punto II en el que, como idea propia, divido todo el alcance temporal en tres etapas o ciclos.

Y así empezó el proyecto y, poco a poco, año tras año, he ido construyendo este documento.

Sobre el autor

El autor realizó estudios universitarios de Informática en la Universidad Politécnica de Valencia entre los años 1987 y 1992, obteniendo la Licenciatura (título hoy equivalente al de Ingeniero en Informática) en 1994 tras la lectura y defensa del Trabajo Fin de Carrera.

Uso del texto

Referencias

Todas las referencias incluidas al final de este texto han sido localizadas y leídas (total o parcialmente) y de todas se ha extraído algo para este trabajo. No se ha "engordado" artificialmente la bibliografía. En todo caso puede que haya utilizado alguna parte de algún otro libro o documento o página web, y que olvidara anotarlo.

Todos los libros, textos y documentos referenciados, se pueden encontrar en Internet. De todas formas si alguien está interesado en alguno y no lo encuentra, podría facilitárselo si me lo pide al correo fcomins69@gmail.com.

Las referencias o bibliografía no siguen las directrices de la ISO 690, ni ningún estilo (Vancouver, APA, MLA, CSE, NLM, Harvard, IEEE, Chicago, etc.), sino como me ha parecido y como he ido anotando: Título de la obra, autor o autores, institución o empresa o universidad y año. En los casos de libros extraídos de Google Books, he añadido la url. Están ordenados por orden alfabético del título (sin tener en cuenta artículos -a, the, el, la, los,etc- iniciales).

Aportaciones

El autor agradecerá cualquier aportación para este documento: subsanación de errores (que seguro que los hay, y muchos), ampliación de algún texto, aportación de texto nuevo, etc. Eso sí, siguiendo el espíritu de lo ya plasmado, debería aportarse también la fuente de donde se ha extraído la información. Las aportaciones pueden remitirse al correo electrónico del autor (fcomins69@gmail.com)

El autor irá publicando nuevas versiones del texto donde se incluyan las aportaciones recibidas, así como las propias. En un apartado "aportaciones" se explicará lo aportado por otras personas, citándolas si así lo desean.

> Nota: una posible aportación, si alguien está dispuesto a ello, sería normalizar toda la bibliografía con algún estilo de la ISO 690.

Futura ampliación

En el punto II he dividido la historia de los lenguajes de programación de robots industriales en tres etapas: los lenguajes de control numérico, los lenguajes robóticos y los frameworks robóticos. El resto del texto, la taxonomía de los lenguajes y la recopilación de información de cada lenguaje en particular, está centrado en la segunda etapa, la de los lenguajes robóticos.

En una próxima versión se ampliará el documento para incluir un análisis de la tercera etapa, la de los frameworks robóticos.

También, posiblemente, se haga una ampliación temporal hacia atrás incluyendo un análisis de los diferentes lenguajes de control numérico.

II. Las tres etapas en la programación de robots industriales

Desde que a principios de los años 50 se empezó a utilizar computadoras para controlar máquinas-herramienta hasta la actualidad, la historia de los robots industriales y su programación ha ido evolucionando y cambiando con el tiempo. En un primer nivel de detalle, podemos distinguir tres etapas o ciclos diferentes.

II.1. Primera etapa: Lenguajes de CNC

Esta etapa vendría a ser como la prehistoria de los lenguajes de programación robótica. En estos lenguajes, todavía no estamos hablando de robots, sino de máquinas-herramienta, como tornos, taladros, fresadoras, pulidoras, etc., pero que cuentan con un sistema de automatización, es decir, son operadas mediante comandos que recibe de un sistema computerizado. Es la computadora la que controla la posición y velocidad de los motores que accionan los ejes de la máquina.

El Control Numérico Computerizado (CNC) tuvo su origen a principios de los años 50 en el Instituto Tecnológico de Massachusets (MIT), en donde se automatizó por primera vez una gran fresadora.

El lenguaje CNC más utilizado es el definido en los estándares 6983 de ISO (International Standarization Organization), RS274D de EIA (Electronic Industries Association) y 66024/66025 de DIN. Este estándar consiste en una serie de códigos definidos con una letra y dos números. Por ejemplo, los códigos G que representan movimientos o códigos M para funciones como arranque, parada, cambio de herramienta, etc. También se le conoce como G-Code o Código G.

No obstante, han surgido otros lenguajes de CNC como 2CL, AUTOSPOT, BOSS, CAMP, FMILL, GEPLAD, MILMAP, PROMO, STEP-NC, SYMPAC o ZAP, por poner algún ejemplo.

La frontera entre lenguaje CNC y lenguaje de programación robótica no es clara y bien definida. Por ejemplo, la familia de lenguajes APT (Automatically Programmed Tools) suelen ser considerados como lenguajes CNC en la mayor parte de la literatura, pese a ser un lenguaje bastante avanzado especialmente en la definición de la geometría (similar a un lenguaje de CAD). En este documento se han incluído como lenguajes de programación robótica. Por el contrario, algunos de los lenguajes más primitivos considerados como robóticos por la literatura van poco más allá de una lista de posiciones por las que debe ir pasando la máquina o robot. A veces la diferenciación parece más bien obedecer al dispositivo en el que se

ejecuta el programa: si es una máquina-herramienta (fresadora, torno, etc.) se considera un CNC, y si es un brazo con n ejes, se considera lenguaje robótico.

II.2. Segunda etapa: Lenguajes de programación de robots

A finales de los años 50 y comienzos de los 60 empezaron a desarrollarse lenguajes de programación de robots. Si exceptuamos los lenguajes de la familia APT[1] que, como se ha indicado, estaría a caballo entre este periodo y el anterior, el primer lenguaje de programación a nivel de robot fue el MHI (Mechanical Hand Interpreter) desarrollado en el MIT en 1960 para el robot MH-1 y que apenas tenía primitivas para mover el robot, testear los sensores y ramificar el programa según ciertas condiciones. Otros de los lenguajes más primitivos es WAVE desarrollado en 1970 en la Universidad de Standford y considerado por muchos como el primer verdadero lenguaje de programación de robots.

Durante cuatro décadas proliferó el desarrollo de estos lenguajes. En este documento se describen más de 20 lenguajes que vieron la luz en la década de los 60's, unos 30 lenguajes de la década de los 70's, unos 60 lenguajes de la década de los 80's y otros 30 lenguajes de la década de los 90's, además de otros 30 lenguajes que, siendo de esas mismas décadas, no se ha podido datar su año de inicio. A partir del año 2000 se reduce drásticamente la generación de nuevos lenguajes de programación de robots, entrando en el tercer ciclo temporal.

Durante esas décadas, los lenguajes fueron diseñados tanto por universidades e institutos de investigación como por empresas del sector de la robótica. El primer lenguaje comercial fue SIGLA (Sigma Language), desarrollado por Olivetti en 1974.

Lógicamente, a lo largo de ese tiempo, los lenguajes fueron evolucionando. Al principio eran básicamente una lista de puntos, de coordenadas, por las que va pasando el brazo robótico. Poco a poco se van incorporando funcionalidades para controlar el flujo del programa (bucles, ramificaciones, etc.), para recibir información de sensores y de visión, para tomar decisiones, para emitir órdenes a diferentes efectores del robot, y para sincronizar la acción de varios brazos o robots. En buena medida la evolución de los lenguajes ha sido consecuencia de la evolución de los robots: los robots dejaron de ser simples brazos, y los lenguajes fueron incorporando mecanismos para manejar las nuevas funcionalidades que posibilitaban los robots, tales como sensorización, visión, robots móviles, etc.

Otra línea de evolución de los lenguajes de programación de robots fue la propia de todos los lenguajes de programación, siendo el hito más destacable la introducción de la programación estructurada a finales de los años 70. Con este paradigma se permite la definición de estructuras de

1 En este documento se comentan más de una treintena de lenguajes de la familia APT

datos y de control más complejas, transformaciones de coordenadas, etc. y además aumenta la comprensión del programa, se reduce el tiempo de edición y se simplifican las acciones encaminadas a la consecución de tareas.

De forma contraria a lo ocurrido con la programación estructurada, la llegada del paradigma orientado a objetos a principios de los años 90, tuvo escasa incidencia (por no decir nula) en los lenguajes de programación robótica.

Un hecho destacable es la larga duración de los lenguajes, especialmente los desarrollados por las empresas fabricantes de robots. Así, los robots de ABB siguen utilizando el lenguaje RAPID que se originó en 1994; robots de Adept y Stäubli utilizan el lengaje V+ que data de 1989. E igualmente los de Mitsubishi con MBA (Melfa Basic) o los de Kuka con KRL. Los desarrolladores van emitiendo nuevas versiones de los lenguajes para actualizarlos, pero siguen utilizándose. Por ejemplo, SPEL (Suwa seiko Production Equipment Language), de Seiko-Epson tuvo su primera versión en 1984, y treinta años después, en 2014, se publicaba la versión 7.1.

II.3. Tercera etapa: Frameworks robóticos

A partir del año 2000, la programación en el ámbito robótico vive la misma situación que en los otros ámbitos: dejan de publicarse nuevos lenguajes de programación y empiezan a proliferar los "frameworks", plataformas o paquetes de software para el desarrollo de programas para robots.

Estos frameworks suelen permitir diferentes lenguajes de programación, proveen abstracción para diferentes robots, proporcionan entornos de desarrollo y entornos de simulación, librerías y componentes, etc.

Seguramente los más conocidos son OROCOS (Open Robot Control Software) y ROS (Robot Operating System), pero hay muchos más, como Player & Stage, Carmen, Microsoft Robotics Studio, Robot Framework, RoboComp, Gobot, etc.

II.4. Exclusiones

En el análisis realizado en este documento no se han considerado ciertos sistemas que no son en sí lenguajes de programación de robots, sino sistemas de simulación, generalmente gráficos, del comportamiento del robot, como por ejemplo:

- GRASP (Graphical Robot Applications Simulations Package), de BYG System (no hay que confundirlo con el lenguaje de programación de robots con el mismo acrónimo GRASP, General Robot Arm Simulation Program)

- IGRIP (Interactive Graphics Robot Instruction Program), desarrollado por la Universidad de Cornell
- GRIPPS, desarrollado por la Universidad Tecnológica de Michigan
- PLACE, ANIMATE, COMMAND, ADJUST y BUILD, desarrollados por McDonell Douglas Automation
- ROBOT-SIM de GE/Calma.
- AutoBots, de Autosimulations Inc. (Utah);
- CATIA (Computer Aided Three Dimensional Interactive Graphics), de Dassault Systems (Francia)
- KARMA (Knowledge Based Robot Manipulation System) de la Universidad de California
- MnCell de la Universidad de Minnesota
- RCode de SRI
- RoboCam de Robotics Research and Development Ca.
- Robot-Sim, de General Electrics Calma Co.
- RoboTeach de General Motors
- RoboGraphics, de ComputerVision Corporation
- TIPS/GS (TIPS/Geometris Simulator), de TIPS Research Assocciation

Tampoco se han tenido en cuenta lenguajes de programación de robots exclusivos de otros ámbitos de la robótica diferentes a la robótica industrial, como por ejemplo, los lenguajes dedicados a robots educativos que están proliferando en los últimos años.

III. Clasificación de los lenguajes de programación de robots

Se pueden hacer diferentes tipos de clasificación de los lenguajes de programación de robots, teniendo en cuenta diferentes factores.

III.1. Taxonomía general.

La taxonomía o clasificación de lenguajes más generalizada en la literatura existente sobre la materia es la siguiente:

1. Programación gestual punto a punto, programación por enseñanza o programación por guiado
   - 1.1. Guiado pasivo
     - 1.1.1. Guiado pasivo directo
     - 1.1.2. Guiado pasivo por maniquí
   - 1.2. Guiado activo
     - 1.2.1. Guiado básico

1.2.2. Guiado extendido

2. Programación textual
   2.1. Programación textual explícita
      2.1.1. Programación textual explícita a nivel de movimientos elementales
      2.1.2. Programación textual explícita estructurada
   2.2. Programación textual especificativa
      2.2.1. Programación textual especificativa a nivel de tarea
      2.2.2. Programación textual especificativa a nivel de objetivo

III.1.1. Programación gestual punto a punto, programación por enseñanza o programación por guiado

La programación se realiza on-line, o sea, con el robot "in situ". No requiere la escritura de un programa, sino que se basa en la enseñanza directa de la máquina por el usuario. El método es similar a las normas de funcionamiento de una grabadora, ya que disponen de unas instrucciones similares: PLAY (reproducir), RECORD (grabar), FF (adelantar), FR (atrasar), PAUSE, STOP, etc. Además, puede disponer de instrucciones auxiliares, como INSERT (insertar un punto o una operación de trabajo) y DELETE (borrar).

La programación se realiza punto a punto, posicionando el elemento terminal del robot en los puntos precisos para ejecutar la tarea. Esos puntos se almacenan en el sistema de control para poder repetir posteriormente los movimientos

Los movimientos pueden tener lugar en sistemas de coordenadas cartesianas, cilíndricas o de unión, siendo posible insertar y borrar las instrucciones que se desee. Es posible, también, implementar funciones relacionadas con sensores externos, así como revisar el programa paso a paso, hacia delante y hacia atrás.

El programa generado puede almacenarse para no tener que repetir la enseñanza en el futuro.

Otro forma con la que se ha denominado a estos lenguajes es "lenguajes a nivel de servo". Un programa consiste en una serie de puntos finales, velocidades y comandos de entrada/salida. Cada punto se representa como un grupo de coordinadas de ejes, de forma que un robot de 6 ejes representa una posición con 6 valores. Los programas son específicos del robot. Los comandos de entrada/salida normalmente consisten en leer el estado de un conmutador (entrada) o accionar un relé (salida), después de que se haya ejecutado un movimiento. El camino entre puntos finales es generado por el controlador del robot calculando una serie de puntos intermedios entre ambos. Entonces el servocontrol maneja cada eje para dirigirse a la posición objetivo, que será el primer punto intermedio. Una

vez alcanzado, el robot dirigirá hacia el siguiente punto intermedio, y así hasta alcanzar el punto final.

También se les conoce como lenguajes de programación por guiado, porque consiste en hacer realizar al robot, o a una maqueta del mismo, la tarea, registrando las configuraciones adoptadas para su posterior repetición en forma automática. Para guiar al robot por los puntos deseados se utilizan distintas soluciones:

III.1.1.1. Guiado Pasivo: Si los actuadores del robot están desconectados y el programador aporta en forma directa la energía para mover el robot, se habla de un guiado pasivo. A su vez, puede ser de dos tipos:

III.1.1.1.1 Guiado Pasivo Directo: En este caso, el programador puede tomar el extremo del robot y llevarlo hasta los puntos deseados a través de las trayectorias más adecuadas. La unidad de control del robot registra de manera automática la señal de los sensores de posición de las articulaciones en todos los puntos recorridos. Un ejemplo es el caso de los robots de pintura de la firma Gaiotto, los cuales fueron programados con este procedimiento.

III.1.1.1.2 Guiado Pasivo por Maniquí: La dificultad física de mover toda la estructura del robot se resuelve a través de este procedimiento. En este caso se dispone de un doble del robot, mientras que éste permanece fuera de línea. El doble posee una configuración idéntica que el robot real, pero es mucho más ligero y fácil de mover. La programación se realiza llevando de la mano a este doble, mientras que la unidad de control muestrea y almacena con cierta frecuencia los valores que toman los sensores de posición de las articulaciones, para su posterior repetición por el robot. Un ejemplo es el caso de los robots de pintura fabricados por Nordson, los cuales son programados utilizando este procedimiento.

III.1.1.2. Guiado activo: Esta posibilidad permite emplear el propio sistema de accionamiento del robot, controlado desde una botonera o bastón de mando (conocido como joystick) para que sea éste el que mueva sus articulaciones. Ejemplos de este tipo se encuentran en los robots de ABB (ARLA) o en los Cincinnati Milacron (T3). Atendiendo a la potencia del sistema, se habla de guiado básico y guiado extendido.

III.1.1.2.1 Guiado Básico: El robot es guiado por los puntos por los cuales se desea que pase durante la fase de ejecución automática del programa. Durante ésta, la unidad de control interpola dichos puntos según determinadas trayectorias. Muchas veces no es posible incluir ningún tipo de estructuras de control dentro del programa, por lo que los puntos son recorridos siempre secuencialmente, en el mismo orden que se programaron. Un ejemplo de este tipo de programación es la utilizada en casi todos lo robots de pintura, donde la unidad de control muestrea automáticamente los puntos recorridos por el robot con una frecuencia muy alta.

III.1.1.2.2 Guiado Extendido: Permite especificar, junto a los puntos por los que deberá pasar el robot, datos relativos a la velocidad, tipo de trayectoria, precisión con la que se quiere alcanzar los puntos, control del flujo del programa, atención a entradas/salidas binarias, etc. En este caso, el método guiado de utilizado es el de la botonera o joystick. El guiado extendido aumenta la potencia del sistema de programación.

Los métodos por guiado son muy útiles y presentan ventajas, como que son fáciles de aprender y requieren de un espacio de memoria relativamente pequeño para almacenar la información. Pueden ser programados por operadores familiarizados con la aplicación, pero sin ser programadores informáticos. También es relativamente fácil programar una trayectoria en una situación geométrica compleja con muchos obstáculos.

Sin embargo, también presenta inconvenientes como la necesidad de utilizar al propio robot y su entorno para realizar la programación, lo que obliga a sacar al robot de la línea de producción e interrumpir ésta, por lo que el sistema se vuelve improductivo durante la programación. También está la inexistencia de una documentación del programa y la dificultad de realizar modificaciones en el mismo. Para realizar nuevas tareas, la programación no puede contar con procedimientos de propósito general desarrollados previamente, como librerías de subrutinas. No es posible un desarrollo modular del programa. La programación puede ser peligrosa ya que el operador está cerca del robot durante el proceso de aprendizaje y debug del programa. Por último, la programación por aprendizaje puede no ser apta en un entorno altamente automatizado y flexible. Algunas de estas desventajas se reducen, incluso se eliminan, con algún tipo concreto de guiado, como el guiado pasivo por maniquí o el guiado activo.

Estos lenguajes no son lenguajes en el sentido de "leguajes de ordenador" pero son medios de programación a nivel de servo. Alguno de os lenguajes de guiado son: T3, Funky y AR-SMART.

III.1.2. Programación textual

El programa queda constituido por un texto de instrucciones o sentencias, cuya confección no requiere de la intervención del robot; es decir, se efectúan "off-line". Con este tipo de programación, el operador no define, prácticamente, las acciones del brazo manipulado, sino que se calculan, en el programa, mediante el empleo de las instrucciones textuales adecuadas.

En una aplicación tal como el ensamblaje de piezas, en la que se requiere una gran precisión, los posicionamientos seleccionados mediante la programación gestual no son suficientes, debiendo ser sustituidos por cálculos más perfectos y por una comunicación con el entorno que rodea al sistema.

En la programación textual, la posibilidad de edición es total. El robot debe

intervenir, sólo, en la puesta a punto final.

Según las características del lenguaje, pueden confeccionarse programas de trabajo complejos, con inclusión de saltos condicionales, empleo de bases de datos, posibilidad de creación de módulos operativos intercambiables, capacidad de adaptación a las condiciones del mundo exterior, etc.

En líneas generales, las ventajas de un lenguaje de programación off-lline frente a la programación on-line son:

- el robot no se vuelve improductivo durante la programación
- es más fácil la incorporación de sensores de visión y de fuerza
- permite la sincronización de equipamiento externo, para manejarlo eficientemente.
- Las construcciones de bifurcación y bucle permiten un manejo de errores personalizado (en algunos lenguajes on-line, también se puede).
- Tareas repetitivas (como paletización) pueden programarse con relativa facilidad, usando herramientas como las macro o las subrutinas (algunos lenguajes de guiado también tienen esta posibilidad).
- Se puede desarrollar una librería de subrutinas para utilizar en futuras tareas de programación; soluciones programadas previamente se reutilizan para nuevos programas.
- Varios programadores pueden trabajar simultáneamente en partes diferentes de un programa grande, y luego se juntan los módulos individuales.
- La programación off-line separa el entorno de programación del entorno de operación. Se pueden utilizar sofisticadas herramientas de programación para ayudar en el desarrollo del programa sin sobrecargar las capacidades computacionales del controlador.

No obstante, también existe alguna desventaja:

- Es muy dificultoso visualizar la trayectoria de un robot en el espacio de tres dimensiones. Tratar de determinar la accesibilidad, las orientaciones adecuadas y las trayectorias sin colisiones, plantea problemas importantes. Es más facil hacerlo con la programación mediante el 'teach pendant', donde la trayectoria del robot se crea a medida que se escribe el programa.
- Las limitaciones de la precisión del robot pueden dar lugar a errores de ejecución. La precisión de un robot es la capacidad de posicionar el efector final en un punto objetivo especificado, en relación con algún marco de referencia externo absoluto. La desviación entre el punto al que se mueve y el punto realmente deseado es una medida de la precisión. Las imprecisiones son resultado de las condiciones de carga y de la configuración del manipulador en el área de trabajo. Las tolerancias de posicionamiento absoluta ajustadas pueden no ser alcanzables con un programa generado estrictamente fuera de línea,

a menos que se utilicen sensores para posicionar el robot con "guiado terminal".

- El programador no puede anticipar todas las posiciones y orientaciones exactas del equipo en el espacio de trabajo. Por tanto, todos los puntos finales programados tienen probabilidad de tener cierto error. El programa generado off-line puede necesitar ser "afinado" en el taller antes de la producción, por ejemplo usando un teach pendant. La alernativa de una integración masiva de sensores puede no ser económicamente justificable.
- Modulos programados que funcionan de forma aislada, pueden no funcionar cuando están todos juntos ejecutándose en el robot.
- El trayecto exacto que tomará el robot, a menudo no es conocido en tiempo de programación. Esto es debido a que el controlador del robot genera sus propios puntos intermedios en función de la orienteación inicial, la orientación final, la carga, la velocidad, y los sensores de entrada. Además, el robot puede parar a mitad ejecución de un movimiento, reorientarse, y continuar el movimiento. Esto es una función del software interno del robot que trata de evitar que éste exceda los límites de recorrido, o estire el cable umbilical.
- La sintaxis completa de un lenguaje puede ser compleja de aprender si no se es un técnico informático.

Estos inconvenientes hicieron que surgieran lenguajes que combinaban el modo textual de programación con las sesiones de 'teach pendant' para producir los programas, dando como resultado un enfoque económico y flexible que alguno denominaron "augmented teach" (aprendizaje aumentado). En estos casos se genera un programa textual donde las posiciones del robot en el espacio de trabajo se representan simbólicamente, no con valores numéricos de las coordenadas. Después se obtiene un fichero con los valores de esos puntos, moviendo manualmente el controlador a los puntos del espacio que corresponden a esas posiciones, y almacenando los ángulos o coordenadas de los ejes del manipulador. Un compilador o cargador, junta ambos ficheros (el del programa textual y el de coordenadas) incorporando los valores a los puntos simbólicos del programa textual y genera un programa ejecutable por el robot.

Este sistema mixto, o "augmented teach" presenta dos ventajas. La primera que el programa textual del movimiento puede ser desarrollado off-line y simulado hasta cierto punto. Segunda, que el programa puede ser reutilizado con diferentes conjuntos de datos de coordenadas, correspondientes a diferentes partes geométricas.

Implementaciones de algunos lenguajes como VAL o CLIMPER requieren de una sesión de aprendizaje con el robot para establecer el conjunto de datos. Otros sistemas, como ASEA Off-line Programming System permite generar el fichero de coordenadas sin la interacción del robot, usando valores de coordenadas de un sistema CAD, con lo que toda la programación es off-line.

La clave del método de augmented teach reside en la separación de la estructura del programa y los datos geométricos.

III.1.2.1. Programación textual explícita

En la programación textual explícita, el programa consta de una secuencia de órdenes o instrucciones concretas, que van definiendo con rigor las operaciones necesarias para llevar a cabo una tarea. Se puede decir que la programación explícita engloba a los lenguajes que definen los movimientos punto por punto, similares a los de la programación gestual, pero bajo la forma de un lenguaje formal. Con este tipo de programación, la labor del tratamiento de las situaciones anormales, colisiones, etc., queda a cargo del programador.

III.1.2.1.1. Programación textual explícita a nivel de moviemientos elementales

Pueden describirse como "movimiento punto a punto en forma de lenguaje". El lenguaje describe los movimientos primitivos que realizará el robot. Los puntos y movimientos pueden definirse por guiado. Algunas características adicionales que proporciona el lenguaje son: estructuras de control, saltos condicionales, subrutinas (con pase de parámetros), capacidades mejoradas de sensores, ejecución paralela y definiciones de 'frames'.

Se pueden distinguir dos tipos de lenguajes:

- Articular, cuando el lenguaje se dirige al control de los movimientos de las diversas articulaciones del brazo.
- Cartesiano, cuando el lenguaje define los movimientos relacionados con el sistema de manufactura, es decir, los del punto final del trabajo (TCP)

Los lenguajes del tipo cartesiano utilizan transformaciones homogéneas. Este hecho confiere "popularidad" al programa, independizando a la programación del modelo particular del robot, puesto que un programa confeccionado para uno, en coordenadas cartesianas, puede utilizarse en otro, con diferentes coordenadas, mediante el sistema de transformación correspondiente. Son lenguajes que se parecen al BASIC, sin poseer una unidad formal y careciendo de estructuras a nivel de datos y de control.

Por el contrario, los lenguajes del tipo articular indican los incrementos angulares de las articulaciones. Aunque esta acción es bastante simple para motores de paso a paso y corriente continua, al no tener una referencia general de la posición de las articulaciones con relación al entorno, es difícil relacionar al sistema con piezas móviles, obstáculos, cámaras de TV, etc.

Los lenguajes correspondientes al nivel de movimientos elementales aventaja, principalmente, a los de punto a punto, en la posibilidad de

realizar bifurcaciones simples y saltos a subrutinas, así como de tratar informaciones sensoriales.

Algunos de los lenguajes de este tipo son: APT, ANORAD, EMILY, MAL, RCL, RPL, VAL, SIGLA, WAVE, PLAW.

III.1.2.1.2. Programación explícita estructurada

Este tipo de lenguajes permite la definición de estructuras de datos y de control complejas, incorporando elementos de programación estructurada. Se pueden definir puntos, líneas, planos, etc. así como transformaciones de coordenadas y capacidad sensorial para interactuar con el entorno.

Intenta introducir relaciones entre el objeto y el sistema del robot, para que los lenguajes se desarrollen sobre una estructura formal. Describen objetos y transformaciones con objetos, disponiendo, muchos de ellos, de una estructura de datos arborescente.

El uso de lenguajes con programación explícita estructurada aumenta la comprensión del programa, reduce el tiempo de edición y simplifica las acciones encaminadas a la consecución de tareas determinadas.

En los lenguajes estructurados, es típico el empleo de las transformaciones de coordenadas, que exigen un cierto nivel de conocimientos.

También se les denomina lenguajes de segunda generación, para diferenciarlos de los anteriores que conformarían la primera generación. Algunas de sus caractarísticas son:

- Capacidad de control del movimiento, como en los de la primera generación
- Capacidad de incorporar datos de sensores avanzados tanto como las señales simples on/off de los dispositivos binarios.
- Capacidad de procesar información recibida del entorno y modificar el comportamiento del sistema
- Capacidad de interactuar con ordenadores para mantener un registro de los datos, generar informes, etc.
- Extensibilidad del lenguaje para manejar requerimientos futuros (nuevos aspectos de los futuros robotos), mediante el desarrollo de comandos, subrutinas y macros.

Por ejemplo, utilizando un lenguaje de segunda generación, un robot es capaz de controlar la fuerza de la pinza sobre un objeto basándose en datos analógicos de un sensor de fuerza, lo que no sería posible en un lenguaje de la primera generación, ya que esos solo permitían sensores binarios on/off. En caso de producirse un error o fallo en el robot, el robot de primera generación seguramente pararía, mientras que un robot programado con un lenguaje de segunda generación puede recuperarse utilizando un algoritmo

inteligente programado en él.

Algunos de los lenguajes de este tipo son: AL, HELP, MAPLE, PAL, MCL, MAL Extendido, AML, VAL-II, RAIL

No siempre es fácil encuadrar un cierto lenguaje de programación explícita de robots en uno de los dos grupos (movimientos elementales o estructurado), pues no son dos conjuntos claramente separados, sino más bien una evolución, por lo que algunos están "a caballo" entre ambos tipos.

III.1.2.2. Programación textual implícita o especificativa

Se trata de una programación del tipo no procesal, en la que el usuario describe las especificaciones de los productos mediante una modelización, al igual que las tareas que hay que realizar sobre ellos.

El sistema informático para la programación textual especificativa ha de disponer del modelo del universo, o mundo donde se encuentra el robot. Este modelo será, normalmente, una base de datos más o menos compleja, según la clase de aplicación, pero que requiere, siempre, computadoras potentes para el procesado de una abundante información.

El trabajo de la programación consistirá, simplemente, en la descripción de las tareas a realizar, lo que supone poder llevar a cabo trabajos complicados.

Dentro de la programación textual especificativa, hay dos clases, según que la orientación del modelo se refiera a los objetos o a los objetivos.

Si el modelo se orienta al nivel de los objetos, el lenguaje trabaja con ellos y establece las relaciones entre ellos. La programación se realiza "off-line" y la conexión CAM es posible.

Dada la inevitable imprecisión de los cálculos del ordenador y de las medidas de las piezas, se precisa de una ejecución previa, para ajustar el programa al entorno del robot.

Los lenguajes con un modelo del universo orientado a los objetos son de alto nivel, permitiendo expresar las sentencias en un lenguaje similar al usado comúnmente. Por otra parte, cuando el modelo se orienta hacia los objetivos, se define el producto final.

La creación de lenguajes de muy alto nivel transferirá una gran parte del trabajo de programación, desde el usuario hasta el sistema informático; éste resolverá la mayoría de los problemas, combinando la Automática y la Inteligencia Artificial.

III.1.2.2.1. Programación implícita a nivel de objetivo

En estos lenguajes se define únicamente el producto final, siendo el sistema informático de control, el que define los estados intermedios.

III.1.2.2.2. Programación implícita a nivel de tarea

Se realiza una modelización del entorno para permitir especificar las acciones a realizar por el robot, indicando las tareas que se deben efectuar sobre los objetos. La programación es off-line, completamente independiente del robot en el que se ejecutará. Después se realiza automáticamente la generación de las acciones elementales que se ejecutarán en el robot.

No existe un verdadero lenguaje a nivel de tareas. Diferentes autores han mencionado diferentes lenguajes por tener ciertas capacidades de nivel de tarea. Alguno de ellos son AUTOPASS, LAMA, AL, RAPT y ROBEX, siendo RAPT y AUTOPASS los más mencionados. En algunos casos se definieron las instrucciones a nivel de tarea pero no se implementaron, o se implementaron parcialmente.

III.2. Clasificación en 5 niveles de Bonner y Shin

Una clasificación similar a la anterior es la denominada "en 5 niveles", o "de Bonner y Shin", establecida por Susan Bonner y Kang G. Shin a principios de la década de los 80. Los niveles de esta clasificación son:

- Nivel 1: Nivel de microcomputador. El usuario especifica las tareas del robot en un lenguaje de microcomputador, realizando explícitamente todos los cálculos de datos y conversiones necesarias para la tarea. Consiste en servocomandos e interfaces de sensores. No pueden ser considerados en sí "lenguajes de programación".
- Nivel 2: Nivel punto a punto. Equivalente a la programación por guiado en la taxonomía general
- Nivel 3: Nivel de movimientos primitivos: equivalente a la programción textual explicita a nivel de movimientos elementales en la taxonomía general.
- Nivel 4: Nivel de programación estructurada: con el mismo nombre en la taxonomía general
- Nivel 5: Nivel orientado a la tarea: que también tiene un equivalente con el mismo nombre en la taxonomía general

III.3. Clasificación por el nivel de abstracción

Diversos autores, como T. Lozano Pérez en 1983, o C.Blume y W. Jakob en 1986, o J. J. Craig en 1989 han clasificado los lenguajes de programación de robots según la forma de describir las operaciones físicas (como

mocimientos del robot) o la forma de referenciar a los objetos físicos, en estos tres tipos:

- Programación a nivel de tarea. Es el mayor nivel de abstracción, en el que las instrucciones especifican qué hacer sin especificar cómo debe ser realizado. La tarea puede ser una tarea completa o una subtarea
- Programación a nivel de objeto. Se utiliza un modelo del mundo formado por objetos, pero sin necesidad de un modelo completo (p.ej. con obstáculos) porque el sistema no planifica las trayectorias, sino que se dan, implícita o explícitamente, especificando relaciones entre los objetos. El sistema computa el movimiento necesario para conseguir la relación, y realiza el movimiento siguiendo una estrategia definida, como movimientos en línea recta en el espacio cartesiano.
- Programación a nivel de manipulador: se centra más en movimientos del manipulador más que en los objetos que son manipulados. Se utilizan movimientos en el espacio articular, así como parámetros específicos del manipulador como acelaración o velocidad como porcentaje sobre la velocidad máxima del manipulador.

III.4. Programción off-line y on-line

Una clasificación repetida en la literatura, basada en el uso del robot físico durante la programción, es la que divide los lenguajes en programación off-line y programación on-line.

- Programación off-line: significa que el robot mecánico y otro equipamiento de producción no está ocupado durante la programación, que tiene lugar en un ordenador. La programación off-line tiene la ventaja de que el entorno de producción puede ser diseñado, programado y simulado antes de que ese entorno se construya realmente. El resultado de esa simulación puede indicarnos qué robot utilizar y cómo organizar el entorno de producción. La desventaja de la programación off-line es que se utiliza una representación computerizada (aproximada) del robot y de su entorno, por lo que la programación es inadecuadamente abstracta y el programa puede ser inexacto.
- Programación on-line: significa que el robot físico está ocupado durante la programación. La programación on-line mantiene ocupado al equipo de producción, pero la ventaja es que es tangible, no necesitando un modelo abstracto del mundo ni simulación. Los movimientos programados serán precisos ya que las ubicaciones y marcos se pueden definir vía aprendizaje en referencia a los marcos reales. Sin embargo los movimientos complejos a través de trayectorias matemáticamente bien definidas pueden ser muy difíciles de programar por aprendizaje. Esos movimientos se describen mejor basados en datos de sistemas CAD.

III.5 Otras clasificaciones

En algunas obras también se clasifican los lenguajes de programación de robots, dentro de cada nivel o grupo, en dos conjuntos:

- los desarrollados por Universidades e Institutos de Investigación, generalmente no usados con fines comerciales, desarrollados con un objetivo de investigación particular en mente, y que han tenido influencia en el desarrollo de lenguajes comerciales.
- los desarrollados por empresas comerciales

Algunos autores, como John J. Craig, dividen los lenguajes de programación explícita en tres tipos:

- Lenguajes de manipulación especializados. Son lenguajes desarrollados completamente nuevos que, aunque se enfocan en la áreas específicas de los robots, pueden considerarse como un lenguaje general de programación. Ejemplos: VAL, AL o ARLA.
- Biblioteca robótica para un lenguaje existente. Son lenguajes que se han desarrollado a partir de un lenguaje de programación convencional (C, Pascal, Basic, etc) al que se le agrega una biblioteca de subrutinas específicas para robots. Ejemplos: AR-BASIC, JARS, RCCL o PASRO..
- Biblioteca robótica para un nuevo lenguaje de propósito general. Son lenguajes que se han desarrollado creando primer un nuevo lenguaje de propósito general como base de programación y después añadiéndole una biblioteca de subrutinas predefinidas específicas para robots. Ejemplos: RAPID, AML y KAREL.

IV. Lenguajes de programación de robots, de la A a la Z

ACL (Advanced Command Language)

ADAPT (Air Material Command Developed APT)

AL (Arm Language)

ALFA (1)(A Language For Automation)

ALFA (2)

AML (A Manufacturing Language)

AMPLE (Automated Manufacturing Programming Language Environment)

ANDROTEXT

ANORAD

APT (Automatically Programmed Tools)

APTLOFT

AR-BASIC (American Robot – Basic)

AR-SMART (American Robot - Smart)

ARCL (1)(A Robot Control Language)

ARCL (2)(Advanced Robot Control Library)

ARCLE (Assembly Robot Control Language)

ARL (Assembly Robot Language)

ARLA (ASEA Robot Language)

ARMBASIC

AS

ATP

AUTOAPT

AUTODRAFT

AUTOLOFT

AUTOMAP

AUTOPASS (Automatic Parts Assembly System)

AUTOPIT (Automatisch Programieren Inclusive Technologie)

AUTOPROMPT (Automated Programming of Machine Tools).

BASP

BEHAVIOR

CABSL(C-Based Agent Behavior Specification Language)

CADET

CAP1 (Conversational Auto Programming)

CIMPLER

COL Concurrency Oriented Language

COMPACT (COM-Share's Program for Automatically Controlled Tools)

CPL (CRAM Plan Language)

CRCL (Canonical Robot Command Language)

CURL (Cambridge University Robot Language)

DAMN (Dynamic Analysis of Mechanical Networks)

DARL

DATA-BEADS

DDL (Drawing Descriptive Language)

DIAL (Drapper Industrial Assembly Language)

EARLS-2

EMILY (ML Extended)

ESL (Execution Support Language)

EXAPT (Extended APT)

FA-BASIC (Factory Automation BASIC)

FDTL (Fuzzy Decision Tree Language)

FEL (Feature Extraction Language)

FROB (Functional Robotics)

FSTN (Finite-State Transducer Network)

FUNKY

GERCS (Generic Educational Robot Control System)

GOLOG (alGOL in LOGic)

GRASP (General Robot Arm Simulation Program)

GRL (Generic Robot Language)

HARL (Hitachi Assembly Robot Language)

HELP

HIGH

HILAIRE

HMDL (Humanoid Motion Description language)

HR-BASIC

HZAPT

IBL (Instruction Based Learning)

IFAPT

ILMR (Intermediate Language for Mobile Robots)

INFORM

IRL (1)(Intuitive Robot Language)

IRL (2)(Industrial Robot Language)

IRPASS (Interactive Robot Programming and Simulation System)

JARS

KAM (Kinematic Analysis Method)

KAREL

KRL (Kuka Robot Language)

K2

L-IRL (Lola Industrial Robot Language)

LAMA (Language for Automatic Mechanical Assembly)

LAMA-S

LENNY

LERNA

LINGRAPHICA

LM (Language de Manipulation)

LM-GEO

LMAC (Modular Command Language for Industrial Robots)

LPR (Language de Programmation de Robot)

LRP (Live Robot Programming)

MAL (Multipurpose Assembly Language)

MAPLE

MAPS

MAPT (Micro APT)

MCL (Manufacturing Control Language)

MELFA BASIC (MBA)

MHI (Mechanical Hand Interpreter)

MicroPLANNER

MINI

MINIAPT

ML (Manipulator Language)

MML (Model-base Mobile robot Language)

MRL (1)(Multiagent Robot Language)

MRL (2)(Movemaster Command Language)

MYBASIC

NC/360

NELAPT (National Engineering Laratory APT)

NUCOL (Numerical Control Language).

PADL (Part and Assembly Description Language)

PAL (1)(Portable AL)

PAL (2)

PAM

PASLA (Programmable Assembly robot Language

PASRO (Pascal for Robots)

PDL2

PILOT (Programming and Interpreted Language Of Actions for Telerobotics)

PLAW (Programming Language for Arc Welding)

POINTY

PRS (Procedural Reasoning System)

RAIL

RAPID (Robotics Application Programming Interactive Dialogue)

RAPT (Robot Automatically Programmed Tool)

RCCL. (Robot Control C Library)

RCL (Robot Command Language) (1)

RCL (Robot Command Language) (2)

RCS (Real-time control system)

REMAPT

RIPL (Robot Independent Programming Language)

RISE

RLC

ROBEX (Roboter Exapt)

ROBOCAM

ROBOFOTH

ROBOML

ROBOS

RoboTalk de Rhino

RoboTalk de Standord

RobotScript

ROCOL

ROL (Robot Language)

ROLL (Robot Learning Language)

ROMPS

ROPL (Robot Programming Laguage)

ROPS (Robot Off-line Porgramming System)

RPL (Robot Programming Language)

RPS (Robot Programming System)

RSS (Robot Servo System)

SERF (Sankyo Esasy Robot Formula)

SCOL (Symbolic Code Language for robot)

SCORBASE

SIGLA (Sigma Language)

SPEL (Suwa Seiko Production Equipement Language)

SPLAT (Simple Provisional Language for Actions and Tasks)

SPLIT (Sundstrand Processing Language Internally Translated)

SRCL (Siemens Robot Control Language)

SRIL-90

SRL (Structural Robot Language)

STRIPS (Standford Research Institute Problem Solver)

SWYM

T3

TDL (a Task Description Language for robot control)

TEACH

TPP (Teach Pendant Programming)

UNIAPT

URBISCRIPT

URSCRIPT

VAL (Vicarm Assembly Language) (Versatile Assembly Language)

VML (Virtual Machine Language)

V+

WAVE

XABSL (Extensible Agent Behavior Specification Language)

XPROBE (Experimental System for Programming Robots by Example)

YALTA (Yet Another Language for Telerobotics Application)

ZDRL (Zhe Da Robot Language)

ACL (Advanced Command Language)

ACL es un lenguaje de robot orientado al movimiento diseñado en Japón en 1990. que utiliza un entorno de comandos conversacionales amigables. Se ha utilizado en robots Yaskaua y Eshed Robotec. Permite operar al robot en dos sistemas de coordenadas: ejes (joints) y catesianas. Las posiciones, además de poderse indicar en los dos sistemas de coordenadas, pueden indicarse de forma absoluta, de forma relativa a otra posición, o de forma relativa a la posición actual.

ADAPT (Air Material Command Developed APT)

Air Material Command Developed APT (o Adaptation APT), es un subconjunto de APT, de 1961, patrocinado por la Fuerza Aérea de EEUU para extender la estadarización con un sistema que pudiera ejecutarse en computadores de media escala. Cualquier programa ADAPT puede ejecutarse en un sistema APT, pero no al revés. Requiere solo un computador pequeño con 32K de memoria, y está limitado a programación de caminos continuos en tres ejes.

AL (Arm Language)

Es un sucesor de Wave, desarrollado en el Laboratorio de Inteligencia Artificial de la Universidad de Stanford en 1974, siendo uno de los lenguajes originales de los brazos de Standford. Basado en el lenguaje ALGOL, lenguaje en el que fue escrito, con alguna característica de PASCAL concurrente, fue originalmente compilado en un PDP 10 y ejecutado en el DEC 11/45 (la versión comercial está basada completamente en el 11/45). AL fue el primero que tuvo tanto un lenguaje de control de robot como un avanzado lenguaje de computador. Ha servido de base para otros lenguajes posteriores.

AL es particularmente bueno para controlar tareas de manipulación, como en aplicaciones de ensamblaje. Fue diseñado para controlar 4 brazos al tiempo, dos brazos Stanford y dos robots PUMA. Los sensores de fuerza se pueden implementar a través de sentencias condicionales y subrutinas apropiadas. Los comandos SIGNAL y WAIT permiten la sincronización de los diferentes pasos del proceso de ensamblaje. Esos comandos, junto a COBEGIN y COEND, permiten controlar el movimiento de múltiples brazos.

Utiliza vectores, posiciones y transformaciones. Tiene comandos para el control de la sensibilidad del tacto de los dedos (fuerza, movimiento, proximidad, etc.).

Unos años después, se introdujo una nueva versión interactiva de AL, realizada entermente en Pascal OMSI en un PDP-11/45. Esa versión, particularmente útil para controlar robots PUMA, fue una versión comercial. AL salió de EEUU: una versión de la Universidad de Karlsruhe y otra versión de la Universidad de Tokyo.

ALFA (1)(A Langugaje For Automation)

Es un lenguaje de control de robot, de General Telephone Electronics (USA), de alto nivel orientado a problema que permite al operador programar las actividades del robot incluyendo el manejo de interrupciones de tiempo y proceso, de manera efectiva sin recurrir a lenguajes de nivel inferior. Fue publicado en 1976 por Shyh J. Wang en IEEE Transactions on Systems, Man and Cybernetics.

ALFA (2)

Publicado por Eran Gat, del Jet Propulsion Lab (California Inst. of Technology) en 1991 en IEEE International Conference on Robotics and Atomation. Es un lenguaje de programación de mecanismos de control para robots móviles autónomos. Los programas Alfa consisten en redes de módulos computacionales conectados por canales de comunicación.

AML (A Manufacturing Language)

Lenguaje desarrollado por IBM en 1977, aunque fue publicado en 1982 para el robot cartesiano hidráulico RS-1 (o 7565) también de IBM, y el robot eléctrico 7535 producido por Sankyo en Japón. Tanto el lenguaje AML como el robot RS-1 fueron el resultado de importantes trabajos de investigación llevados a cabo en el laboratorio Yorktown Heights a mediados de los años 70. Ahí se desarrollaron otros lenguajes como Maple, Emily y Autopass. El RS-1 fue el primer sistema equipado con medidores de deformación en los dedos de la pinza. El AML del robot de Sankyo es un subconjunto del AML.

El desarrollo de AML se basó en los lenguajes ALGOL, APL y PL/I. AML fue diseñado para ser un lenguaje general para el control de equipos de fabricación, pero su uso se centro en la robótica. La filosofía del diseño de este lenguaje altamente estructurado es proporcionar primitivas potentes de bajo nivel con las que el usuario construye bibliotecas de rutinas. Ello generó un lenguaje potente y flexible, pero que requiere de ciertas habilidades de programación. El lenguaje proporcional multitarea, comunicaciones del sistema host, marcos de referencia definidos por el usuario, funciones de control de programa para programación estructurada y una capacidad avanzada de interfaz sensorial.

Algunas versiones posteriores de AML fueron:

- AML/E (entry), de 1982, usado en los robots 7535 y 7540. Es un subconjunto de AML, con visualización gráfica de la posición del robot
- AML/V (vision), de 1982, con un paquete de subrutinas que proporcionan capacidades de visión.
- AML/X (eXperimental), de 1986, con sofisticadas capacidades de abstracción de datos, con inspiraciones de loa lenguajes SmallTalk y LISP. No era compatible con el código existente. Además de los

aspectos de control, había incorporado funciones de planificación, inventario y cálculo de costes.

- AML/2, de 1986, basado en AML/X, usado en los robots 7575 y 7576 de IBM

AMPLE (Automated Manufacturing Programming Language Environment)

Desarrollado en el Center for Manufacturing Engineering del National Bureau of Standars, del Departamento de Comercio de Estados Unidos. El proyecto AMPLE fue capitaneado por J. C. Boudreaux. Se emprendió este proyecto por dos motivos: para proporcionar un mecanismo preciso de construcción de interfaces de control para procesos de fabricación industrial; y para proporcionar un sistema integrado de herramientas software para trasladar el diseño del producto y la planificación de procesos en programas verificados de control. El proyecto se inició en 1984 y se publicó en 1987. Fue desarrollado en FranzLISP, un dialecto de LISP

AMPLE es una colección extensible de módulos software, distribuidos alrededor de un núcleo central llamado AMPLE/Core. El conjunto del resto de módulos es llamado AMPLE/mod. La principal función de Ample/Core es mantener la representación formal precisa de todos los objetos físicos y procesos del dominio de fabricación: partes (atributos geométricos, topológicos, datos de tolerancia, etc), dispositivos, sensores, y procesos.

ANDROTEXT

Diseñado en 1982 por Robotronic Corporation para el robot Hero-1 (Heathkit Educational Robot, de Heathkit Company), es un lenguaje de alto nivel, pensado para ser comprendido con poco conocimiento técnico, para que sirviera para otros tipos de robots con mínimas modificaciones, y para que fuera útil con futuras generaciones de robots. El compilador que traduce de Androtext al lenguaje de máquina del robot, es propiedad de Heathkit Company.

Además de las características comunes de los lenguajes de programación, agrega dos clases de instrucciones aplicables a robots:

- una para disponer y leer sensores. Se leen tanto sensores pasivos, como sensores externos, como visión o sistemas táctiles.

- otra para operar los efectores, tanto los efectores pasivos (como luces o sonido), como los efectores que mueven al robot.

ANORAD

El lenguaje ANORAD fue creado en los años 70, por Anorad Corporation para su robot Anomatic, un único brazo con 4 grados de libertad, sobre un procesador Motorola 6800. Más que un lenguaje de programación en sí, es

un control numérico mejorado. Es interpretado no transportable. La programación en Anorad proporciona ramificación condicional e incondicional. Los movimientos se pueden indicar en ángulo o en coordenadas cartesianas; también movimientos absolutos. Se puede llamar a subrutinas, pero sin pasar parámetros. Permite incluir ficheros como código ejecutable, que se incluyen como subrutinas. Está provisto de comandos simples de sensores táctiles binarios, pero no comandos de visión.

APT (Automatically Programmed Tools)

Desarrollado al final de la década de los 50 (desde la formulación inicial en 1956 hasta su presentación en 1959) por el Grupo de Aplicaciones de Computador del MIT (según algunas fuentes perteneciente al Laboratorio de Sistemas Electrónicos, y según otras al Laboratorio de Servomecanismos), patrocinado por las Fuerzas Aéreas de USA, aunque acabó conformándose un grupo formado por varias universidades, agencias gubernamentales y 14 compañías, denominado Aircraft Industries Association (posteriormente Aerospace Industries Association). Como fue creado antes de que estuvieran disponibles los interfaces gráficos, se utiliza el texto para indicar la geometría y las rutas de la herramienta. Se desarrolló incluso antes de disponer de FORTRAN, y fue uno de los primeros estándares de ANSI. Posteriormente se reescribieron nuevas versiones de APT en FORTRAN.

Fue el primer lenguaje en constituir un standar ANSI: ANSI X3.37. Podríamos situarlo a mitad camino entre un lenguaje de control numérico y un lenguaje de programación de robots industriales. Realmente, un programa APT antes de ser ejecutado en la máquina, se transforma en un programa CNC (Computer Numerical Control).

APT se usa para programar herramientas de máquinas CNC para crear piezas complejas usando una herramienta de corte que se mueve en el espacio. Se usa para calcular la ruta que debe seguir la herramienta para generar la forma deseada. APT es el predecesor de los posteriores sistemas de CAM (Computar-aided manufacturing; fabricación asistida por ordenador). APT puede generar geometría compleja en 5 ejes simultáneamente.

Un programa APT tiene una sección de geometría (con sentencias de geometría) y una sección de movimientos (con sentencias de movimientos). En la primera, el programador define una parte de la figura, y en la segunda describe los movimientos de la herramienta para producir esa parte. En la sección de movimiento se incluyen también comandos para encender la herramienta, definir su velocidad, encender el refrigerante, activar la compensación de la fresa, etc.

En la sección de geometría todos los comandos tienen la forma

<nombre> = comando / <definición>

El nombre, de hasta 6 caracteres, da nombre a la forma que se define.

Puede ser cualquier nombre, pero lo habitual es que ese nombre empiece por P si se define un punto, C para círculos, L para líneas, PL para planos. Los comandos son: POINT, CIRCLE, LINE, PLANE. Detrás de la barra se define la geometría. Ej:

P1=POINT/6,6,6 (punto definido en coordenadas cartesianas)

P2=POINT/INTOF,L1,L2 (punto definido por la intersección de dos líneas)

P3=POINT/CENTER,C1 (punto definido por el centro de un círculo)

L4=LINE/P1,P2 (línea desde el punto P1 hasta el P2)

L5=LINE/P2,PARLEL,L1 (línea paralela a la línea L1 a través del punto P2)

L6=LINE/P1,LEFT,TANTO,C1 (línea desde P1, tangente a C1 por la izquierda)

C1=CIRCLE/CENTER,P1,RADIUS,1.0 (círculo con centro en P1 y radio 1)

C2=CIRCLE/CENTER,P5,L1 (círculo con centro en P5, tangente a L1)

PL1=PLANE/P1,P2,P3 (plano conteniendo los puntos P1, P2 y P3)

PL2=PLANE/XYPLAN,1.0 (plano paralelo al plano XY a distancia de 1)

En la sección de movimientos todos los comandos tienen la forma:

comando / [descriptor] , [valor_numérico]

El comando indica la acción a realizar (SPINDL, GOTO, CYCLE, etc). Los descriptores pueden ser: IPM (inch per minute), IPR (inch per revolution), RPM (revolutions per minute), ON, OFF, etc. No todos los comandos tienen descriptores y valores.

Además de los comandos de geometría y de movimiento, hay sentencias de postprocesado para control de la herramienta, como por ejemplo fijar velocidades, establecer el valor de tolerancia para la interpolación circular, establecer unidades de medida.

Por último, hay una serie de sentencias auxiliares. Por ejemplo, todo programa APT comienza con la sentencia PARTNO <nombre> y finaliza con la sentencia FINI. Con REMARK se introduce un comentario.

La forma de trabajar, como se ha mencionado, es escribir el programa en APT y luego traducirlo a CNC (códigos ISO para control numérico).

APT ha conformado una familia de lenguajes, pues son múltiples las evoluciones, adaptaciones y extensiones desarrolladas, :

- AUTOPROMT (Automated Programming of Machine Tools). 1957.
- APT II. Evolución de APT de 1958 para el IBM 704
- AUTOAPT. 1960.
- APT III. Evolución de APT II, de 1961 para el IBM 7090

- ADAPT. Subconjunto de APT, de 1961.
- SPLIT (Sundstrand Processing Language Internally Translated). 1963.
- APT IV. Diseñado en 1964 por APTLRP (APT Long Range Program), un programa establecido por la Asociación de Industrias Aerospaciales, aunque la primera versión fue implementada en un IBM 7094 en 1967. La versión para sistemas 360 fue en 1968. Se incluyeron nuevos módulos, como un módulo conversacional y un módulo tecnológico.
- Data-beads. Extensión de APT de 1965.
- NC/360. Basado en APT III. De IBM. 1965
- UNIVAC APT III. Implementación de APT III para el Sperry Univac 1108 en 1966
- Chingari Conversational APT. Extensión de APT de 1966
- IFAPT. Extensión de APT, de 1966, en Canadá.
- APTLOFT. Extensión de APT III de 1967
- EXAPT. Extended APT. 1967. En el mismo año evolucionó a EXAPT-2 y al año siguiente a EXAPT-3.
- COMPACT. 1967. Evolucionó en 1969 a COMPACT II
- NUCOL. Extensión de APT III de 1967.
- MINIAPT. Versión alemana de APT de 1968.
- NELAPT. Dialecto de APT diseñado en 1968.
- REMAPT. Versión de ADAPT de General Electric. 1968.
- UNIAPT. Implementación de APT III de 1969
- SWYM. Exensión de APT III de 1969
- ATP. Extensión de APT de 1969.
- CADET (Computer Aided Design Experimental Transator), extensión de APT de 1969, enfocada al CAD.
- APT/70 Desarrollado por Symbolic Control Inc. en 1969, aunque la primera versión fue liberada en 1970 y para uso en producción en enero de 1971. Se basa en un rediseño extendido de APT III y APT IV. Es un completo procesador de 5 ejes capaz de procesar programas estándar de APT, pero también es compatible con APT IV, UCC APT III, UNIVAC APT III, ADAPT, etc.
- APT 77. Extensión de APT III, de 1977
- HZAPT. Versión china de APT 77, diseñada en 1985. En 1987 se presentó una nueva versión, HZAPT-2.

- RAPT. Extensión de APT diseñada en 1977 y publicada en 1978 en Reino Unido.
- Otras versiones, sin fecha disponible:
  - MAPT (MICRO-APT)
  - SEAPT
  - DNIAPT
  - AUTOMAP
  - MINIFAPT

APTLOFT.

Extensión de APT III de 1967

AR-BASIC (American Robot - Basic)

Publicado en 1984 por American Robot para uso con sus robots. Está basado en el lenguaje Basic, y soporta entrada/salida discreta, capacidades matemáticas, y control del flujo del programa. Sustituía al anterior lenguaje de esta empresa, el AR-SMART. Publicado por A. Gilbert y otros en "AR-BASIC: an advanced and user friendly programming system for robotics", una publicación de la propia American Robot Corp.

AR-SMART (American Robot – Smart)

En 1982 se fundó American Robot Corporation. En 1987 el nombre cambió a American Cimflex Corporation, para reflejar la diversificación de la empresa en otras áreas de la automatización. En 1990 se fusiona con Teknowledge Inc. Convirtiéndose en Cimflex Teknowledge Corporation. Más tarde en ese mismo año, la división de robots se separa convirtiéndose otra vez en American Robots Corporation. En su primera etapa, en 1982, equipaba a sus robots con el lenguaje de guiado AR-SMART. Posteriormente, en1984 cambió a AR-BASIC. El lenguaje AR-SMART permitía al usuario enseñar al robot puntos del brazo y mover de punto a punto en un orden definido. Era utilizado fundamentalmente para operaciones de "pick and place".

ARCL (1)(A Robot Control Language)

Lenguaje publicado por A.S. Elmaghraby en 1988 en la Conferencia Sudeste de IEEE de ese año, está basado en la sintaxis de Pascal. Es compilado y necesita tres pasadas de compilador antes de poder ser ejecutado por un robot. Tiene comandos de movimiento y comandos de control de sensores. Se implementó en el robot Hero-1.

Un ejemplo de comando es MOVA(GRIP,HI,CONT,MED) que abre la pinza del robot. ARCL hace más hincapié en la programación basada en sensores que en la planificación de la trayectoria del movimiento. No es posible la ejecución concurrente.

ARCL (2)(Advanced Robot Control Library)

Es una librería de software potente, general y portable que proporciona capacidades de control de robot en lenguaje C. Fue desarrollado por Manufacturing Technology e introducido por Peter Ian Corke y Robin Kirkham en 1995 en "The ARCL Robot Programming System"

Soporta control de movimiento basado en sensores. El modelo de programación se basa en la representación de coordenadas cartesianas por transformaciones homogéneas. ARCL proporciona al lenguanje C nuevos tipos de datos y operaciones específicos de robot. No fue una idea nueva, sino que siguió los conceptos de RCCL.

Es modular y portable a diferentes plataformas, sistemas operativos y manipuladores de robots.

ARCLE (Assembly Robot Control Language)

Es al parecer un lenguaje utilizado por Nissan, mencionado por H.Suda en el Journal of the robotics society of Japan en el año 1984, una mención posteriormente repetida en diversas publicaciones que sitúan a ARCLE como uno de los lenguajes de robot a nivel de primitivas existentes en Japón.

ARL (Assembly Robot Language)

Basado en Pascal, fue desarrollado en 1983 en Japón por Hitachi para sus robots. Se presentó, como SPEL y ARCLE en el Journal of the Robotics Society of Japan en diciembre de 1984, por S. Mohri y otros autores.

ARLA (ASEA Robot Language o ABB Robot Language)

Es el lenguaje lanzado en 1982 por la empresa sueca ASEA (Allmänna Svenska Elektrska Aktiebolaget; posteriormente ABB) para sus robots, concretamente para el controlador de la serie S3. Está disponible para programación on-line con teach pendant o programación off-line para coputadores VAX o IBM. El ASEA Off-Line Programming System permite la creación de programas a nivel de manipulador en ARLA, que es un lenguaje no estructurado con construcciones para movimiento, interacción con los sensores, comunicación, funciones matemáticas y lógicas, y aplicaciones específicas para soldadura y visión.

El conjunto de datos de coordinadas puede ser generado off-line con un editor o on-line durante una sesión de aprendizaje.

En 1988 ASEA se fusiona con la empresa suiza BBD (Brown, Boveri & cie) fromando ABB (Asea Brown Boveri), y en 1994 el lenguaje ARLA fue sustituido por su sucesor RAPID.

ARMBASIC

Diseñado en 1982 como una extensión de BASIC para permitir la manipulación del robot educativo Microbot Mino-Mover 5, a través de la microcomputadora TRS-80 de Radioshak. Fue publicado en 1983 por C.J. Shore y J. Huddleston en "A facility for the evaluation of the perfomance of small robotics arms, ana a comparision of the Colne Armdroid anda Microbot Minimover-5"

AS

Lenguaje de Kawasaki para sus robots. Tiene comandos de control del movimiento (MOVE, APPRO, DEPART, HOME, DRAW, ALIGN, DELAY, STABLE), comandos de precisión y velocidad (SPEED, ACCURACY, ACCEL, DECEL, BREAK, BRAKE), control de la pinza (OPEN, CLOSE, RELAX, GUNONTIMMER, GUNOFFTIMER, GUNON, GUNOFF), así como instrucciones de configuración (RIGHTY, LEFTY, ABOVE, BELOW, UWRIST, DWRIST) y de control del programa (GOTO, IF, CALL, RETURN, WAIT, PAUSE, HALT, STOP, LOCK, ONE, REPCYCLE), estructuras de control de flujo (IF..THEN..ELSE..END, WHILE..DO..END, DO...UNTIL, FOR..TO..STEP..END, CASE..OF..VALUE..ANY..END), comandos de control de señales binarias (RESET, SIGNAL, PULSE, DLYSIG, RUNMASK, BTIS, EXTCALL, ON, IGNORE), de entrada y salida (TYPE, PRINT, PROMPT), de definición de variables (HERE, POINT, DECOMPOSE, BASE, TOOL, LIMIT, TIMER, WEIGHT), y de manejo del programa y datos (DELETE, LOAD, TRACE).

No hemos podido encontrar referencias históricas sobre su diseño y/o publicación, siendo la referencia más antigua encontrada de 1999.

ATP

Extensión de APT de 1969.

AUTOAPT

Implementación de APT de 1960. [Ver APT]

AUTODRAFT

Evolución de DDL, publicada en 1965 por O.D. Smith y H.R. Harris, del North American Aviation. Publicado en la ACM IEEE Design Automation Conference, bajo el título "Autodraft: a language and processor for design and drafting".

AUTOLOFT

Diseñado en 1962, con influencia de APT y Sketchpad, por O.D. Smith del Numerical Group, North American Aircraft. Permite la creación y almacenamiento de paquetes estándares de rutinas.

AUTOMAP

Versión limitada de APT para programación en dos dimensiones de líneas y círculos. Consiste en 50 palabras y es fácil de aprender.

AUTOPASS (Automatic Parts Assembly System)

Parcialmente implementado por IBM en 1974 y publicado en 1975 por L. Liberman y M.A. Wesley ("Autopass, a very high level programming language for mechanical assembler system). Autopass, que viene a ser una extensión del lenguaje PL/I de IBM, pone el foco a nivel de objeto, describiéndose el movimiento a alto nivel. Es similar a RAPT en cuanto que los programas parecen Hojas de Instrucciones de Ensamblaje. Tuvo influencia en la definición de lenguajes posteriores como AML o LAMA.

La mayor parte del esfuerzo de implementación se enfocó a un método de planificación de la ruta libre de colisiones para robots cartesianos entre obstáculos poliédricos.

AUTOPIT (Automatisch Programieren Inclusive Technologie)

Diseñado por A.G. Pittler en Alemania en 1966, en cooperación con IBM para controlar el torno PINUMAT, corriendo uen un IBM 1620. Para su diseño se basó en EXAPT y AUTOPROMT. Un año después, en 1967, diseñó la segunda versión AUTOPIT II; y en 1974 la tercera versión AUTOPIT III

AUTOPROMPT (Automated Programming of Machine Tools)

Extensión de APT diseñada en 1957 por Sam Matsa de IBM, en cooperación con United Aircraft. Es un derivado tridimensional de APT. [Ver APT]

BAPS

Es un lenguage de la empresa Bosch. En la literatura aparecen diferentes denominaciones para estas siglas:

- Bosch Automatisierungs Programmier Sprache (Lenguaje de Programación de Automatizaciones de Bosch). Así se le menciona en la publicación de la Universidad de Münster *Robot-Programmiersprachen*, de Christian Hermans y Klaus Shaefers refiriéndose a él como un lenguaje similar a IRL o KRL.
- Bewegunds und Abaluf Programmierspache (Lenguaje de programación de secuencias y movimientos). Así se le menciona en Lexikon Automatisierung der Arbeitssysteme, de Stefan Hesse, de 1994 en el que se le define como un lenguaje de Bosch orientado a problema para la programación textual de robots industriales, especialmente para montaje, con comandos simples pero potentes. También se le da este nombre en Human-Robot Interactions de Mansour Rahimi y Waldemar Karwowski, de 1992. Y también en Einführung in die Roboter-programmierung, de Bruno Heck y Thomas Epting, de 1992.
- Bewegungs und Ablauforientierte Programmierspache (Lenguaje de programación orientado al movimiento y al flujo). Así se le menciona en diversas webs alemanas.
- Bosch Application and Programming System. Así se le meciona en Sensor Integration in Esprit, de N. Ghani, publicado en la IFAC Robot Control de Karlsuhe de 1988, definiéndolo como un lenguaje para programar el controlador del robot de Bosch rho2.
- Bosch Advanced Programming System. Así aparece en una serie de webs húngaras.

BEHAVIOR

Behavior es un lenguaje de programación de robots en tiempo real basado en reglas, pubicado por Rodney A. Brooks del Laboratorio de Inteligencia Artificial del MIT en 1990. Está basado en las ideas del propio Brooks, de Jonathan H. Connell y de Pattie Maes expresadas en diferentes publicaciones de 1986 a 1989.

Behavior contaba con backends para diferentes procesadores como Motorola 68000, 68HC11, Hitachi 6301. Los comportamientos son grupos de reglas que se activan por un número de diferentes esquemas. No hay estructuras de datos compartidas, sino que todas las comunicaciones se realizan mediante mensajes explícitos. Todas la reglas se ejecutan de forma paralela y asíncrona. Incluye las nociones de inhibición y supresión, y una serie de mecanismos para la propagación de la activación. El lenguaje Behavior es similar al Common Lisp.

CABSL(C-Based Agent Behavior Specification Language)

Permite especificar el comportamiento de un robot o un agente software en C++. Semánticamente sigue las ideas de XABSL pero su integración con C++ requiere menos sobrecarga de programación. Para más información, véase XABSL.

CADET (Computer Aided Design Experimental Translator),

Extensión de APT de 1969, enfocada al CAD.

CAP1 (Conversational Auto Programming)

Es un lenguaje utilizado por algunos controladores de FANUC, Inc. Como el 32-18-T, y es más bien un lenguaje de CNC.

CIMPLER

Es un lenguaje a nivel de manipulador, del año 1985, que se ejecuta en el controlador CIMROC de la empresa estadounidense GCA Corporation. Tiene prestaciones sofisticadas para el control del movimiento, ejecución paralela y entrada/salida, puesto que los desarrolladores tenían un enfoque claramente de manipulación industrial. Es un lenguaje estructurado. Proporciona operadores relacionales, aritméticos, lógicos y booleanos. Los tipos de datos soportardos son enteros, arrays de enteros, datos de posición, y datos de entrada/salida. Las sentencias de movimiento permiten a la herramienta moverse a la posición deseada relativa desde la posición actual; también es posible moverse una distancia incremental específica relativa a la posición actual. Las sentencias de control del movimiento proporcionan una trayectoria específica definida a través de una secuencia de puntos. El lenguaje permite controlar y variar la velocidad y la aceleración del movimiento del manipulador. También se proporciona comunicación con controladores y computadores externos. Puede interacturar con sensores externos y sistemas de visión.

COL Concurrency Oriented Language

Unversidad de Tokyo 1985. Publicado ese año por M. Mitsuishi y otros bajo el título *Development of Concurrency Oriented Language 'COL'*. Y dos años después en el volumen 2 de Advanced Robotics con el título *Development of the concurrent process oriented language 'COL'*.

COMPACT (Com-Share Program for Automatically Controlled Tools)

Subconjunto extendido de APT desarrollado por Bendix en 1967. A partir de él, MDSI (Manufacturing Data System Inc.), de Ann Arbor (Michigan) desarrolló en 1969 COMPACT-II. Es una variedad punto-a-punto de 3 ejes

con capacidades para trabajo de contorneo de dos ejes y medio. Está disponible una rutina de rotación de cuarto eje para trabajos complejos.

CPL (CRAM Plan Language)

CRAM (Cognitive Robot Abstract Machine) es un paquete de software para el diseño, la implementación y el despliegue de robots autónomos cognition-enabled que realizan actividades cotidianas de manipulación. CRAM equipa a los robots autónomos con mecanismos de razonamiento livianos que pueden inferir decisiones de control en lugar de requerir que las decisiones estén preprogramadas. De esta manera, los robots autónomos programados por CRAM son mucho más flexibles, confiables y generales que los programas de control que carecen de tales capacidades cognitivas.

CRAM fue presentado por Michael Beetz, Lorenz Mösenlechner y Moritz Tenorth, los tres pertenecientes al Grupo de Sistemas Autónomos Inteligentes del Departamento de Informática de la Technische Universität Müchen de Alemania, en la 2010 IEEE/RSJ Conferencia Internacional de Sistemas y Robots Inteligentes, celebrada en Taipei, China.

El lenguaje que incorpora CRAM para programar los robots es CPL, CRAM Plan Language. CPLes un lenguaje de especificación de comportamiento muy expresivo para robots autónomos que les permite, no solo ejecutar sus programas de control, sino también razonar y manipularlos, incluso durante su ejecución. Es decir los diferentes aspectos (especificaciones de objetos, descripciones de fallos, decisiones que puede tomar el robot, etc) no solo existen como piezas compiladas de código, sino que existen como objetos que se pueden consultar y razonar.

CPL es una evolución de su predeceosr RPL, Reactive Plan Language (de 1992). La mayor parte de los conceptos de CPL ya existían en RPL, pero CPL se ha diseñado particularmente para controlar robots físicos reales.

Otra característica clave es el acoplamiento apretado entre CPL y las estructuras de datos generadas, actualizadas y utilizadas por los módulos de control de nivel inferior. Eso significa que, a diferencia de las arquitecturas en capas que necesitan abstraerse de estas estructuras de datos de bajo nivel, en la CPL todavía están disponibles para el programa completo y se pueden usar para la toma de decisiones. De esta manera, el controlador basado en el plan puede utilizar modelos de control de robot mucho más detallados, precisos y realistas donde sea necesario.

CPL está implementado en Common Lisp y su sintaxis es similar a la de éste, o sea, programación funcional. Proporciona estructuras para evaluación de expresiones tanto de forma secuencial como paralela. Posee unos objetos, llamados fluents, que contienen un valor y proporciona acceso sincronizado. También dispone de manejo de fallos con creación de clases de excepciones, lanzamiento de excepciones, y ejecución de expresiones incluso en excepciones.

CRCL (Canonical Robot Command Language)

Desarrollado en 2015 en el NIST (National Institute of Standards and Technology) del Departamento de Comercio de EEUU, publicado un año después por Frederick M. Proctor, Stephen B. Balakirsky, Zeid Kootbally, Thomas R. Kramer, Craig I. Schlenoff, y William P. Shackleford.

CRCL es un modelo de información que proporciona una descripción de alto nivel de las tareas del roboty la información asociada de control y estado. Los autores partieron de las ideas de dos proyectos: JAUS (Joint Architecture from Unmaned Systems, un sistema de mensajes para vehículos no tripulados, inicialmente patrocinado por el Departamento de Defensa de EEUU en 2005 y estandarizado por la Sociedad de Ingenieros de Automoción, así como su implementación de código abierto, OpenJAUS) y MTConnect (un estándar basado en Internet para comunicación con equipos de fabricación patrocinado por la Asociación para la Tecnología de Fabricación en la Universidad de Berkeley, California). De ellos tomaron el enfoque de especificación de mensajes, pero centrándose en el dominio de la robótica industrial como caso especial de la ontología general de la automatización definida por IEEE.

CRCL proporciona contenido y sintaxis de mensaje para un protocolo de estado-comando de robot de bajo nivel. Es decir, CRCL es un lenguaje de mensajes de bajo nivel para enviar comandos al robot y recibir estados del robot. El contenido de los mensajes son, entre otros:

- establecimiento de los parámetros independientes del robot (unidades, velocidades, aceleraciones, tolerancias)

- establecimiento de los parámetros dependientes del robot

- realizar movimientos cartesianos (independientes del robot)

- realizar movimientos a nivel de eje (dependientes del robot)

- operar un efector, pinza o mano

- establecer parámetros de los efectores o específicos del hardware

- cambio de efectores

- configurar informes de estado

- leer un informe de estado

- enviar mensaje al usuario humano

- iniciar o finalizar una sesión

- ejecutar un programa de robot en un cierto lenguage

- detener un movimiento (con ciertos grados de urgencia).

CRCL se divide en 5 modelos de información separados. Tres de estos modelos especifican mensajes: mensajes al robot conteniendo un comando simple, mensajes al robot conteniendo un programa entero, y mensajes de

retorno de robot. Los otros dos modelos definen la estructura general y los elementos comunes de los mensajes de comando y los tipos de datos subyacentes. Los mensajes de estado de CRCL pueden tener: el estado del comando más reciente recibido, informe de estado de ejes, informe de estado de un efector, etc.

Los autores validaron la especificación de CRCL en dos proyectos, uno en el Georgia Tech Research Institute (GTRI), y otro en el propio NIST. En el GTRI se implementó CRCL en un Fanuc LR Mate 200iD como parte de una cadena de montaje. En el NIST se desarrollaron interfaces CRCL para dos sistemas: un brazo robótico Kuka LWR 4 de siete ejes y una mano de tres dedos Robotiq.

CURL (Cambridge University Robot Language)

Diseñado en 1994 por J.L. Dallaway y R.D. Jackson, publicado con el título "The user interface for interactive robotic workstations" en la International Conference on Intelligent Robots and Systems de IEEE/RSJ de 1994

DAMN (Dynamic Analysis of Mechanical Networks)

Diseñado en 1970 en EEUU por Milton A. Chace, publicado en la 7ª "Annual ACM IEEE Design Automation Workshop" celebrada en San Francisco, Californa, bajo el título "DAMN – a prototype program for the Dynamic Analysis of Mechanical Networks". Está basado en KAM.

DARL / DARL-II

Un lenguaje de Seiko basado en BASIC, con íneas de código numeradas. Además de comandos típicos de Basic (Goto, Print, Input, If, For, etc) tiene comandos para controlar el robot: Speed, Move, Output. Las dimensiones se dan en milímetros. Los puntos pueden definirse en el programa o aprenderse moviendo el robot y almacenanado la posición.

DATA-BEADS

Extensión de APT de 1965, incorporando algunas características de WISP, como sistema de representación y almacenamiento CAD 3D. Fue desarrollado por C.E. Robinson en United Aircrat Corp, tras dos años de estudios de forma conjunta entre los laboratorios de investigación de esta empresa e IBM, durante los que se analizó la aplicación del procesamiento de datos gráficos a la manipulación geométrica tridimensional. Por ejemplo, se investigaron técnicas para almacenar en la computadora descripciones tridimensionales de objetos.

DDL (Drawing Descriptive Language)

Diseñado en 1964 en North American Aircraft, basado en APT. Un año después evolucionó a AutoDraft.

DIAL (Draper Industrial Assembly Language)

Desarrollado en el Charles Stark Drapur Laboratory, utiliza una retroalimentación de fuerza electrónica para simular el sentido humano del tacto en componentes de ensamblaje.

EARLS-2

Es un lenguaje de programación de robot para programar un sistema diseñado por la universidad de Tokyo que simula un robot Sankyo de 4 ejes. La referencia más antigua encontrada es un comentario de T. Arai en "A robot language system with a colour graphic simulator" en 1983.

EMILY (ML Extended)

El lenguaje EMILY, basado en ML, fue creado en 1975 por IBM para sus brazos, pudiendo controlar dos brazos. La programación en Emily, basada en FORTRAN, proporciona ramificación condicional e incondicional, y bucles do..loop. Los movimientos se pueden indicar en ángulo o en coordenadas cartesianas; también movimientos absolutos. Se puede llamar a subrutinas, pasando parámetros. Permite incluir ficheros como código ejecutable, a través de una sentencia *include* que causa la ejecución inmediata del fichero deseado. Está provisto de diferentes formas de sensores táctiles, incluyendo sensores táctiles en la mano, un sensor en un dedo tipo 'bigotes', un sensor de proximidad, y un emisor-receptor de infrarrojos que detecta la presencia de un objeto entre los dedos. Emily monitoriza estos sensores de forma on/ off para ayudar en el proceso de ensamblaje, pero no tiene capacidades de visión.

Tiene un procesamiento paralelo simple, en forma de operaciones de los brazos con exclusión mutua, con límites y puntos de convergencia para asegurar que no ocurren colisiones. Proporciona el comando SYNC para indicar puntos de convergencia para programas que están ejecutándose simultáneamente en diferentes brazos.

ESL (Execution Support Language)

Es un lenguaje para codificar el conocimiento de ejecución en agentes autónomos integrados. Diseñado por Erann Gat en 1991, y publicado en la IEEE Aerospace Conference de 1997, bajo el título "ESL: A Language for Supporting Robust Plan Execution in Embedded Auotonomous Agents".

Tiene características de otros lenguajes como RAP y RPL. ESL es una herramienta potente, flexible y fácil de usar y es una extensión de Common Lisp. El manejo de contigencias tiene un enfoque "consciente de fallos". Este enfoque presume que muchos posibles resultados de las acciones pueden ser categorizados fácilmente como éxitos o fallos, y cuando ocurre el fallo, el sistema puede responder apropiadamente. La construcción FAIL señala que ha ocurrido un error y WHIT-RECOVERY-PROCEDURES establece procedimientos de recuperación de fallos. Cuando ningún procedimiento puede tratar la situación, la construcción WITH-CLEANUP-PROCEDURE llama a un procedimiento de limpieza.

Las construcciones ACHIEVE y TO-ACHIEVE se utilizan para desvincular las condiciones de logro y los métodos para conseguir esas condiciones. Se puede usar un evento para sincronizar múltiples tareas concurrentes en ESL, de forma que las tareas se envían señales entre ellas. Una tarea puede esperar a varios eventos simultáneamente. Cuando uno de esos eventos produce una señal, la taera permanecerá bloqueada. Múltiples tareas pueden estar esperando simultáneamente el mismo evento. Las construcciones para la sincronización son WAIT-FOR-EVENTS y SIGNAL. También hay construcciones para checkpoints, redes de tareas, guardianes, bloqueos de propiedades.

ESL se utilizó para construir un componente ejecutivo de una arquitectura de control para una astronave autónoma.

Tuvo influencia en la definición del lenguaje TDL.

EXAPT. (Extended APT)

Desarrollado en Alemania en 1967, con un esfuerzo conjunto de las universidades de Berlín y Aachen, más la empresas AEG y Siemens, dirigido por los profesores H.Opitz, W. Simon, G. Spur y G. Stute. Después fue mantenido por al EXAPT Association formada por la industria. Exapt-1 se utilizó para máquinas de posicionamiento, como un taladro. Se calcula automáticamente el ciclo de trabajo con selección automática de herramienta. Por ejemplo, si se requiere un agujero con una tolerancia que requiera escariado, todas la operaciones previas, como el centrado del taladro, el taladrado previo y el escariado se procesan automáticamente.

En el mismo año evolucionó a EXAPT-2, que se utliza para centros de torneado. El sistema calcula los cortes de desbaste y acabado, si se describen las dimensiones del material y la configuración de las partes. El propio sistema determina la alimentación, la velocidad de corte y la selección de herramienta.

Al año siguiente en Stuttgart evolucionó a EXAPT-3, utilizado para trabajos de fresado con control de trayectorias continuas en línea recta en 2.5 ejes.

FA-BASIC (Factory Automation BASIC)

Factory Automation BASIC es un lenguaje para control de robot, sistema de visión y control programable, cuyas principales escresiones, funciones, estructuras y sentencias están basadas en el lenguaje BASIC. FA-BASIC tiene tres subsistemas: FA-BASIC/R para el control del robot, FA-BASIC/V para control de sistemas de visión y FA-BASIC/C para PCs. La intención es tener un método común que interconecte los robots, los PCs y los sistemas de visión.

FDTL (Fuzzy Decision Tree Language)

FDTL es un lenguaje basado en un modelo computacional que combina el control difuso (fuzzy) basado en reglas con la naturaleza jerárquica de los árboles de decisión. Fue presentado por C. Voudouris, P. Chernett, C.J. Wang y V.L. Callaghan en 1995, en la 2nd IFAC Conference on Intelligent Autonomous Vehicles, celebrada en Espoo, Finlandia, en la ponencia "Hierarchical behavioural control for autonomous vehicles".

FDTL contiene un compilador que produce código C que puede ser ejecutado en múltiples plataformas, bien embebido o bien como programa independiente.

El lenguaje fue testado con simuladores y un controlador llamado GC (Garbage Collection) para un vehículo de test autónomo que conduce utilizando velocidades diferenciales en sus dos ruedas motrices y que está dotado de sensores ultrasónicos de proximidad junto a otros sensores que dan el ángulo y la distancia a diferentes metas. La tarea del vehículo es recoger y eliminar la basura, mientras evita obstáculos y se mueve al punto de recarga si la batería está baja o al punto de servicio si se detecta una avería. La arquitectura de árbol de decisión difusa es un sistema que se retroalimenta: la información de entrada de los sensores, junto a variables internas (posiciones meta, nivel de batería, detección de avería, variables de estado, etc.) alimentan el árbol. El árbol propaga la activación desde la raíz a los nodos hoja. Todos los nodos hoja tienen solo dos salidas: las velocidades de cada rueda.

FEL (Feature Extraction Language)

Desarrollado en 1982 por el Instituto de Robótica de la Carnegie Mellon University. Comentado por D. Bourne y P. Fussell en una publicación de dicho Instituto en abril de 1982 bajo el título "Designing Programming Languaes for Manufacturing Cells", así como en otro publicación de D. Bourne en octubre del mismo año, con el título "A numberless, tensed language for action oriented tasks". Ocho años después el mismo D. Bourne publicó en la misma publicación del Instituto de Robótica el artículo "Using the Feature Extraction Language in the next generation controller". Asímismo D.Bourne publicó junto a J.Baird, P.Erion y D.Williams en marzo 1990 el artículo "The Operational Feature Extraction Language"

FROB (Functional Robotics)

Diseñado en 1999 y presentado por Joh Peterson, Gregory D. Hager y Paul Hudack bajo el título "A Language for Declarative Robotic Programming". Está basado en FRP (Functional Reactive Programming). Frob soporta un estilo de programación que separa limpiamente el qué del cómo en la programación del robot. El "qué" es una simple definición de la estrategia de control usando grupos de ecuaciones y primitivas que combinan conjuntos de esas ecuaciones en un sistema complejo. El "cómo" se dirige a los aspectos de detalle como el método para realizar las ecuaciones, la conexión entre las ecuaciones de control y los sensores y efectores del robot, y la comunicación con otros elementos del sistema. Frob soporta prototipado rápido de nuevas estrategias de control, y permite la reutilización del software.

FSTN (Finite-State Transducer Network)

Descrito por P. Drews y P. Fromm en la 23rd International Conference on Industrial Electronics, Control and Instrumentation (IECON 97), realizada en noviembre de 1997, bajo el título "A Natural Language Processing Approach for Mobile Service Robot Control". Es un enfoque en lenguaje natural para el control de robots de servicio móviles.

Una unidad procresadora del lenguaje se realizó e instaló en una silla de ruedas, equipada con un sistema de control de la navegación y del entorno. Para el reconocimiento de las palabras habladas se usó un software comercial (Dragon Tools), y la secuencia muestreada de palabra se transmitía por vía inalámbrica a un PC conectado al módulo de control del entorno. Un control de alta prioridad para la silla de ruedas como "Stop" lo interceptaba y procesaba directamente la CPU de la silla de ruedas. La frase es parseada por una red de transductores de estados finito (FSTN), que testea la estructura gramatical y el vocabulario. La salida del FSTN contiene una versión formateada de la frase.

La base de datos en FSTN contiene clusters de palabras gramaticales como "nombre", "verbo", "negación", etc. En la siguiente fase, la frase formateada se procesa por el módulo de transformación. La frase se divide en componentes básicos "instrucción", "intervención" y "pregunta", y la división se basa en el reconocimiento de patrones gramaticales. Estos patrones están en la base de datos de transformación, y se pueden adaptar al usuario individual.

La base de datos del entorno describe todos los objetos, las posibles accioens relacionadas con los objetos, e información del estado. La base de datos semántica contiene los términos que describen los objetos individuales. Por ejemplo, el término "Apagar" implica que las variables que contienen velocidad, luz, dirección, etc. se establecen a cero. Cuando al

procesamiento le falta información, como en la frase "no, el otro", el sistema busca en la memoria de contexto la información que falta. La memoria de contexto se base en una esctructura LIFO "último que entra, primero que sale". En caso de no tener respuesta, se inicia el módulo de pregunta/respuesta y se genera una pregunta. Finalmente se genera una secuencia de control y se transmite al controlador.

El sistema tiene un vocabulario aproximado de 500 palabras y permite navegación en el entorno doméstico. Trabaja bien con frases sencillas, pero hay problemas cuando se formulan frases más complejas.

Este lenguaje es un buen ejemplo de las soluciones que existirán en un futuro. Con este sistemal incluso los no iniciados pueden controlar un robot móvil en un entorno doméstico.

Funky

Creado por IBM en 1977 para sus robots con procesador IBM System-7. Usa un joystick para el control de los movimientos y un comando especial para centrar la pinza sobre el objeto. Es un lenguaje interpretado y transportable, escrito punto por punto, que permite el movimiento de un solo brazo. Tras la sesión de enseñanza, se genera un texto en un lenguaje de programación, que puede editarse modificándolo o ampliándolo, introduciendo saltos condicionales dependiendo de información sensorial, o realizando cálculos para obtener una posición de forma precisa.

GERCS (Generic Educational Robot Control System)

Presentado en una Tesis de Robert E. Fletcher en 1991 en la Universidad de Lehigh (Pensilvania, EEUU), para obtener el Grado de Máster en Computer Science. Es una integración entre un intérprete de lenguaje C, y una librería de control de robot en lenguaje C. Esta librería consiste en funciones que proporcionan comunicación, transformación de coordenadas, y control del movimiento. La programación de robots consiste en escribir programas en C que llaman a la librería GERCS para controlar el robot.

Se clasifica como un lenguaje textual estructurado.

GOLOG (alGOL in LOGic)

Un lenguaje de alto nivel desarrollado por el grupo Cognitive Robotics de la Universidad de Toronto, bajo la dirección de Héctor Levesque y Ray Rieter. La propuesta inicial fue publicada en 1997. Es un lenguaje de programación lógica para dominios dinámicos, basado en el lenguaje de Prolog. Se basa en un intérprete que mantiene automáticamente una representación explícita del mundo dinámico que está siendo modelado, en base a los axiomas suministrados por el usuario sobre las precondiciones y los efectos de las acciones sobre el estado inicial del mundo. Esto permite a los programas razonar sobre el estado del mundo y considerar los efectos de

las posibles líneas de acción posibles antes de decantarse por un comportamiento concreto. Los programas se escriben con un alto nivel de abstracción. El lenguaje es adecuado para aplicaciones en el control de alto nivel de robots y procesos industriales. Se basa en una teoría formal, una especificación de una versión extendida del *cálculo de las situaciones* (un formalismo lógico diseñado para representar y razonar sobre dominios dinámicos).

El Golog original tenía varias carencias. Para solucionarlas, se presentaron unas extensiones en el año 2000 por H. Grosskreutz y G. Lakemeyer, en el libro "Towards more realistic logic-based robot controllers in the GOLOG framework". Estas extensiones, cc-Golog y pGOLOG, permitieron aplicar este sistema también a los robots móviles.

Otras variantes de GOLOG que ha surgido son: el ConGolog, o Golog concurrente; IndiGolog, ConGolog determinístico incremental; LeGolog, ConGolog para Lego Midstorm

GRASP (General Robot Arm Simulation Program)

Es tanto un sistema de programación off-line como un sistema de simulación gráfico. Fue desarrollado por J. Derby en el Renssealer Polytechnic Institute. Tiene un postprocesador para trasladar el programa al lenguaje del robot destino. Nota: aunque tienen el mismo acrónimo, GRASP, no hay que confundirlo con Graphical Robot Application Simulation Package, desarrollado en la Universidad de Nottingham (Reino Unido)

GRL (Generic Robot Language)

Lengaje desarrollado en 1998 y publicado por Ian Douglas Horswill, de la Northwestern University (EEUU) en el CIRA de 1999 bajo el título "Functional programming of behaviour-based systems". Comunmente pronunciado como "girl", extiende las técnicas tradicionales de programación funcional a los sistemas basados en comportamiento. Es un lenguage embebido en Scheme (un dialecto de Lisp). El programa escrito en GRL se pasa después por un compilador simple que lo convierte en código en otro lenguaje como Scheme, C, C++, Basic, etc. Así se consigue una manera mucho más modular de programar, y se consigue un código C más rápido de ejecutar que el código C escrito directamente a mano.

GRL realmente es solo para sistemas basados en comportamiento (behaviour-based), pero como GBBL tenía mala pronunciación, se optó por GRL. Las arquitecturas basadas en comportamiento consisten en un conjunto independiente de bucles de control ejecutándose en paralelo, generalmente con algún mecanismo de arbitraje para combinar sus salidas. La meta de GRL es tener un lenguaje con las facilidades de abstracción de LISP, y la eficiencia en tiempo de ejecución de un lenguaje compilado. En última instancia, los programas GRL consisten en redes de señales que se computan en paralelo y se actualizan continuamente.

HARL (Hitachi Assembly Robot Language)

Es una versión comercial de ARL especialmente adecuada para tareas de manipulación y ensamblaje punto a punto. Data del año 1983, el mismo año que el propio ARL. Hay 18 instrucciones HARL. Se programa el control del movimiento y después se enseñan los datos de posiciones utilizados en el programa.

HELP

Es un lenguaje comercial de programación de robots de la compañía General Electric para el control de sus brazos cartesianos en tareas de ensamblaje. Fue introducido en 1979 en Italia bajo licencia de Digital Electronic Automation (DEA) para los robots de la familia Pragma A3000. Es interpretado con una sintaxis basada en Pascal. El movimiento del robot se describe en términos de coordinadas rectangulares. Tiene una potente capacidad de entrada/salida y permite la definición y activación de múltiples tareas que se comunican con flags globales. Fue publicado en la First International Conference on Assembly Automation celebrada en 1980 en Brighton, Inglaterra, por G. Donato y A. Camera, con el título "A High Level Programming for a New Multi-Arm Assembly Robot"

HIGH

Lenguaje de 1983 de la Universidad de Tokyo, Japón, basado en Fortran

HILAIRE

Desarrolaldo en el Laboratorio de Automática y Análisis de Sistemas (LAAS) de Toulouse, para robots con procesadores IMS 80 C II. Está escrito en LISP, interpretado, transportable, y que permite la manipulación de varios brazos simultáneamente. El LAAS es parte del CNRS (Centre National de la Recherche Scientifique). En libro "Programación de robots industriales. Control remoto del robot ASEA IRB 2000" de Hilario López García y Rafael González Librán, editado por la Universidad de Oviedo, el lenguaje HILAIRE es mencionado como un lenguaje de programación implícito a nivel objetivo, pero no se encuentra más información, ni en la propia página web de LAAS.

HMDL (Humanoid Motion Description language)

Diseñado por Ben Choi y Yanbing Chen en 2002. Presentado en el Second International Workshop on Epigenetic Robotics, celebrando en Edinburgo, Escocia, en agosto de ese año. Se establecen cuatro niveles de descripción de movimientos:

- ángulo de ejes. Ej: eje rodilla 30, eje codo 45

- camino. Ej: pie (v1, v2)

- primitiva de movimiento: raise, lower, forward, backward

- secuencias de movimiento: walk, run, jump, turn

HR-BASIC

Lenguaje basado en Basic, es una versión compatible superior de HARL-III (Hirata Assembly Robot Language). Incluye sentencias para el control del robot, control de E/S, y control de tiempo. Puede ejecutar 32 tareas simultáneamente. Se pueden desarrollaR y depurar programas en un PC con HBDE, HrBasic Developing Environment.

Fue presentado en "Robotics and Computer-Integrated manufacturing" volumen 2, de 1985, por Shunji Mohri, Kenji Takeda, Seiji Hata Kichie Matsuzaki, y Yoshihiro Hyodo, del Production Engineering Research Laboratory de Hitachi en Japón.

HZAPT (Huazhong APT)

Versión china de APT 77, de la Universidad de Ciencia y Tecnología de Huazhong, diseñada en 1985. En 1987 se presentó una nueva versión, HZAPT-2.

IBL (Instruction Based Learning)

Publicado por S. Lauria, G. Bugmann, T. Kyriacou, J. Bos y E. Klein en el año 2001, bajo el título "Training Personal Robots Using Natural Language Instruction" en Intelligent Systems de IEEE, volumen 16.

Es un método para entrenar robots usando instrucciones en lenguaje natural. Un robot está equipado con un conjunto de procedimientos primitivos motores y sensoriales como "gira a la izquierda", o "sigue el camino" que puede verse como un lenguaje de comandos a nivel de ejecución. Las instrucciones verbales del usuario se convierten en un nuevo procedimiento, y éste se convierte en parte del conocimiento que puede usar el robot para aprender procedimientos cada vez más complejos. Con estos procedimientos el robot debe ser capaz de ejecutar tareas cada vez más complejas. Como siempre es posible un error en la comunicación hombre-máquina, IBL verifica si la subtarea aprendida es ejecutable. Si no lo es, se pregunta al usuario más información.

IBL fue evaluada en una ciudad de miniatura (170x120 cm) con un robot de miniatura (8x8 cm) equipado con cámara TV, transmisor VHF y radio FM, y el procesamiento de imágenes se realizó en un PC externo. Los datos lingüísticos y funcionales se recogieron usando 25 humanos dando

instrucciones para describir 6 rutas desde el mismo punto de partidas hasta seis destinos diferentes. Un operador humano en otra habitación guió al robot de miniatura según las instrucciones dadas, viendo solo la imagen de la cámara del robot. A partir de esos datos se concluyeron las primitivas necesarias, y cuando un humano especifica una acción sin terminar, tal como 'seguir adelante', ésta se clasifica como 'mover hacia adelante hasta'. Así se establecieron 14 primitivas: move forward until, take the, is located, go, go round roundabout, take the exit, follow known route to... until, take roadbend, stationary turn, cross road, take the road, go round..to, park at, exit.

Se diseñó un gestor de diálogos para actuar como interface entre el usuario y el manejador del roboto. El manejador del robot tiene tres procesos de ejecución disponibles: aprender, ejecutar y parar. Si hay un procedimiento correspondiente con el mensaje dado, se empieza un procedimiento de ejecución, sino se empieza un procedimiento de aprendizaje con instrucciones del gestor de diálogos. El manejador del robot se escribió usando C y Python.

IFAPT

Extensión de APT, de 1966, en Canadá.

ILMR (Intermediate Language for Mobile Robots)

Fruto de un proyecto llevado a cabo de 2001 a 2003 en el Technical University of Helsinki, financiado por el Technology Development Centre of Finland (Tekes) y el Technical Research Centre of Finland (VTT), aunque parte de los fondos fueron aportados por el Research Support Foundation of Helsinki University of Technology que, a su vez, había recibido una donación de Sandvik/Tamrock Ltd.

El trabajo partía de otro proyecto previo, de los años 2000-2001, también del VTT: el MCLMR (Motion Control Language for Mobile Robots). La tesis fue finalmente presentada por Ilkka Kauppi en octubre de 2003, bajo la dirección del profesor Aarne Halme.

ILMR actúa como un enlace intermedio desde el usuario, como un planificador inteligente o un interfaz hombre-robot para las acciones y el comportamiento del robot. Los principios para su desarrollo fueron la simplicidad y la facilidad de uso; que no hiciera falta un conocimiento previo de robótica ni buenas habilidades de programación. No obstante, ILMR ofrece todas las prestaciones necesarias para el control de robots especializados, incluyendo ejecución de tareas concurrentes y respuesta a excepciones.

Los usuarios de ILMR pueden dar directamente comandos o tareas a un robot, pero la idea es utilizarlo con lenguajes abstractos de alto nivel. Una acción en ILMR puede ser dada de forma abstracta o en detalle. No se

necesitan modelos complicados de robots ni del entorno. Solo se necesitan pocos parámetros para los robots y un mapa simple del entorno.

ILMR es una forma de traducir una descripción de tareas en alto nivel para un robot móvil en mediciones de sensores y controles sobre los actuadores en bajo nivel.

La arquitectura en que se base ILMR tiene tres niveles: el controlador de bajo nivel (LLC) maneja los sensores y actuadores y otros subsistemas del robot. El controlador de nivel medio (MLC) permite la ejecución de comportamientos, acciones primitivas y supervisión de los subsistemas del robot. Los algoritmos del ILMR corren en este nivel medio y se utiliza un proceso intermedio dependiente del robot para conectar ILMR a un entorno de robot determinado y al controlador de nivel alto (HLC). La terea de HLC es descomponer las intenciones del usuario en subtareas y monitorizar su ejecución.

ILMR tiene un conjunto de comandos elementales que pueden ser usados directamente para controlar el robot o pueden ser usados para construir nuevas prestaciones más inteligentes para el robot. Cuando se publicó IMLR, éste disponía de 33 comandos elementales, divididos en cuatro categorías: 11 de definición, 8 de movimiento, 5 de acción y 9 de control de flujo.

El modelo de robot que utiliza ILMR tiene dos secciones: Hardware y Software. La sección Hardware tiene 3 partes: Robot que contiene la información del robot, como dimensiones o velocidad máxima; Sensores que contiene la descripción de cada sensro; Actuadores que describe cada actuador. La sección Software tiene también tres partes: Objetos contiene los identificadores y definición de cada objeto reconocible; Identificadores contiene identificadores definidos por el usuario que pueden ser utilizados en comandos ILMR; Parámetros contiene valores por defectos definidos por el usuario para ciertos comportamientos de ILMR.

INFORM

Lenguaje desarrollado en 1985 por la empresa estadounidense Motoman Robotics Corporation (posteriormente Yaskawa Motoman, subsidiaria americana de la empresa japonesa Yaskawa Electric Corporation) para su controlador ERC. En 1990 evoluciona a INFORM II para el controlador MRC, con muchas diferencias respecto a la anterior versión, y no compatible con aquella. Posteriormente evolucionó a INFORM III para otros controladores como el DX100, NXC100 o FS100.

INFORM es un típico lenguaje con instrucciones del estilo MOVEJ P000 VJ=50.00. Hay instrucciones de entrada-salida, de control, de movimiento, de soldadura por arco, de soldadura de punto, etc

IRL (1)(Intuitive Robot Language)

Lenguaje desarrollado por la compañía suiza Microbo, asociada a las grandes compañías relojeras, por su amplia gama de robots de ensamblaje de alta precisión. En esa gama se incluye el modelo eléctrico Souris con 6 grados de libertad (precisión 1/100 mm), el modelo hidráulico Castor, y Ecureuil, un modelo introducido en 1983. Esos robots se utilizaron al principio exclusivamente en la industria relojera suiza, pero en 1983 Microbo decidió ponerlo disponible a todo el mercado. El trabajo de investigación y desarrollo para esta gama surgió parcialmente de los trabajos de investigación llevados a cabo en el Instituto Politécncio de Lausanne.

IRL (2)(Industrial Robot Language)

Introducida por la Organización Alemana para la Estandarización, DIN, descrita bajo el estándar DIN-66312. Los trabajos para su definición comenzaron en 1988 y se convirtió en estándar nacional de Alemania en 1992. Tuvo fuerte interés en Alemania en la industria del automóvil, fabricantes de robots e institutos de investigación

IRPASS (Interactive Robot Programming and Simulation System)

Desarrollado en 1983 por M. Katajamaki en la empresa finlandesa Nokia Robotics y publicado en el 14º Simposio Internacional de Robots Industriales de 1984 bajo el título "CAD/CAM, revolutionizing robot applications design". Es un sistema de programación off-line de robots y de simulación basado en un sistema CAD/CAM. Se establece una comunicación interactiva entre el sistema CAD/CAM y el procesador del robot. Tiene capacidades de operaciones de E/S y detección de colisiones.

JARS

Es un lenguaje de 1979 del Jet Propulsion Laboratory de la NASA, basado en Pascal, para el control de robots usados en el ensamblaje de paneles solares. Se usó en el robot PUMA 600. Publicado por J.J. Craig en 1980 bajo el título "JARS: JPL Automous Robot System"

KAM (Kinematic Analysis Method)

Diseñado en 1965 por IBM para analizar la cinemática de mecanismos como engranajes, levas y conexiones. Una descripción en lenguaje KAM utiliza una técnica de asignación de almacenamiento (descrita por Ross) que forma un modelo organizado en árbol en la memoria del computador.

KAREL

Introducido en 1985 por GMF Robotics, es un lenguaje basado en Pascal y Modula, que ofrece construcciones lógicas estructuradas, soporte a comunicaciones de alta velocidad, estructuras vector y array, capacidades de control de proceso, integración con sistemas de visión, y entrada/salida textual. El lenguaje soporta control del movimiento punto a punto e interpolación lineal y circular.

El programa Karel se puede escribir sobre el controlador, o bien off-line y luego descargarlo. Está provisto de un editor que conoce las construcciones permitidas de Karel, lo que mejora la creación y mantenimiento de programas. El código fuente es traducido a una forma de más bajo nivel, que se ejecuta en el intérprete del controlador. El traductor usa reglas de ámbito estático, lo que proporciona fiabilidad y facilidad de mantenimiento, mientras que el intérprete da flexibilidad en tiempo de ejecución. La programación modular es posible por la etapa de traducción y porque un módulo de programa puede tener referencias a procedimientos externos o archivos completos.

Karel es muy flexible porque incluye todos los elementos normalmente asociados a un lenguaje de computador y añadidos específicos para robotica y fabricación. A los procedimientos se les puede pasar parámetros por referencia y por valor, y pueden estar definidos localmente o accesibles en ficheros externos. Lo mismo ocurre con funciones. La recursión también es posible. Operaciones con vectores y el movimiento relativo también está soportado, así como el acceso a nivel de comando del controlador.

La combinación de traducción e interpretación usado para crear programas ejecutables Karel consigue que sea más fácil extender el lenguaje con la simple revisión del traductor. Karel suele tener una buena compatibilidad hacia arriba: cuando se emite un nuevo traductor para nuevas extensiones del lenguaje, los viejos programas Karel suelen seguir ejecutándose sin problemas.

El intérprete de bajo nivel solo está disponible para el controlador para el que se ha diseñado, pero el traductor puede servir para propósito general.

El paso de traducción asegura que un error en tiempo de ejecución no será un error sintáctico. Karel también tiene posibilidades de manejo de excepciones, que puede ser definido localmente o de forma global al controlador. Estos manejadores de excepción son monitores de condición que se chequean cada 32 milisegundos.

La abstracción de datos y la modularidad le dan a Karel un alto nivel de mantenibilidad.

KRL (Kuka Robot Language)

Es un lenguaje similar al Pascal utilizado para controlar los robots de Kuka. Un programa KRL consiste en dos ficheros diferentes: un fichero de datos

permanentes, con extensión .dat, y otro fichero de comandos de movimiento, con extensión .src

Los comandos de movimientos se pueden dar de diversas formas, siendo las más comunes las coordenadas articulares (un valor de rotación para cada eje), o las coordenadas cartesianas (se da la posición y orientación del efector en un sistema de coordenadas cartesianas definido.

El movimiento de un punto a otro puede ser de tres tipo: punto a punto (PTP), lineal (LIN) o circular (CIRC). Se pueden testear las entradas (comando $IN[nºentrada]), cambiar las salidas (comando $OUT[nºsal]=valor), o hacer salidas pulsadas, esto es, encender una salida durante un periodo corto y definido, con el comando PULSE($OUT[nºsal], valor, tiempo)

Para el control del flujo tiene comandos IF-THEN-ELSE-ENDIF, SWITCH...CASE...DEFAULT...ENDSWITCH, bucles FOR … ENDFOR, WHILE...ENDWHILE y REPEAT...UNTIL, así como LOOP...ENDLOOP.

Se puede definir variables de tipo entero (INT), real (REAL), booleano (BOOL), carácter (CHAR) y posición. Las variables de tipo posición se pueden definir de forma cartesiana (POS) con 6 valores que indican el punto cartesiano (3 valores, uno para cada eje) y la orientación del brazo en ese punto (otros 3 valores, uno para la rotación sobre cada eje); y también se puede definir de forma axial (AXIS) dando las rotaciones de los 6 ejes.

Hay una serie de variables predefinidas, como las intradas y salidas $IN[i], $OUT[i], o la velocidad máxima de los ejes $VEL_AXIS.

También tiene temporizadores, comandos de velocidad y comandos de espera.

Se pueden definir TRIGGERs que se disparen cuando una cierta entrada tome un cierto valor. Una vez definido un trigger, éste puede ser encendido o apagado.

K2

Es una implementación o clon de Karel desarrollada en 1995 en EEUU

L-IRL (Lola Industrial Robot Language)

Lenguaje estructurado de la clase de lenguajes orientados a problemas. Con una estructura de tipo Pascasl, y basado en IRL, fue diseñado por Lola Institute Ltd. propiedad de la República de Serbia. Contiene comandos para la programación de los movimientos del robot y comandos para el flujo del programa. Permite operaciones con variables, aritméticas, expresiones lógicas y geométricas y operaciones con señales del entorno, tanto analógicas como digitales. Permite controlar el trabajo cooperativo de varios robots

LAMA (Language for Automatic Mechanical Assembly)

Diseñado en 1973 en el MIT, y publicado en 1977 por Tomás Lozano-Pérez y Patrick Winston en "LAMA: Language for Automatic Mechanical Assembly" en la *5th International Joint Conference on Artificial Intelligence*, aunque ya había sido mencionado un año antes por Tomás Lozano Pérez en una publicación de MIT AI Technical con el título "The Design of a Mechanical Assembly System". LAMA está basado en Autopass y en LISP.

La meta de LAMA era crear un sistema que transformara una descripción de alto nivel de una operación de ensamblaje mecánica automática en un programa para ejecutarse en un manipulador controlado por ordenador. El sistema permite la descripción inicial del ensamblaje en términos de efectos deseados sobre las partes ensambladas.

La finalidad es determinar un plan de ensamblaje. Un plan es una secuencia de operaciones que tomarán las partes desde sus posiciones originales y, a través de unos estados intermedios, llegarán al ensamblaje final.

Una descripción de ensamblaje (Assembly Description) es una lista ordenada de pasos u operaciones de ensamblaje. Las operaciones son comandos de alto nivel como INSERT, PLACE, GRASP, UNGRAS. Las posiciones y orientacioens de las partes, deben ser especificadas simbólicamente. Por ejemplo "INSERT A INTO B" puede ser una operación de ensamblaje.

Una descripción de ensamblaje tiene estas flexibilidades:

- algunos parámetros de las operaciones de ensamblaje pueden no estar especificadas

- las operaciones de ensamblaje no necesitan tener sus prerrequisitos establecidos en los pasos precedentes

- los movimientos del manipulador no están explícitamente especificados.

Cuando una descripción de ensamblaje no tiene ninguna instancia de los dos primeros tipos de flexibilidades, se llama una descripción de ensamblaje completa o Plan de Ensamblaje. Cuando, además, incluye comandos de manipulador explícitos se denomina Programa Manipulador.

LAMA-S

Desarrollado dentro del proyecto Spartacus del IRIA de Francia, un proyecto destinado a desarrollar robots para ayudar a personas discapacitadas en sus tareas de la vida diaria. Publicado por D. Falek y M. Parent en 1979 bajo el título "LAMA-S: an evolutive language for an intelligent robot"

LAMA-S utiliza APL como lenguaje de implementación. Las funciones de nivel de usuario se traducen a un lenguaje de bajo nivel, PRIMA, y luego se ejecutan. Además de las instrucciones de movimiento basadas en el uso de

marcos, LAMA-S proporciona primitivas en tiempo real y ejecución paralela de tareas.

El lenguaje utiliza dos estructuras para definir el orden de ejecución: el bloque de secuencia, para indicar que todas las instrucciones internas se ejecutarán de forma secuencial, y el bloque paralelo, para indicar que todas las instrucciones internas se inician en paralelo (algo así como la estructura de cobegin-coend) . Se proporcionan otras estructuras de control estándar. El uso de APL demostró, según los autores, que APL es una buena herramienta de implementación porque permite la extensibilidad funcional del lenguaje. Por otro lado, no lo recomiendan para uso industrial debido a las siguientes deficiencias: necesita una máquina APL para ejecutarse, la sintaxis de APL no es conveniente, el análisis de sintaxis no es perfecto, es difícil implementar la programación interactiva utilizando APL.

LENNY

Desarrollado en 1982 por la Universidad de Genova, Italia. Fue usado para describir movimientos para un brazo antropomórfico emulado, con 7 grados de libertad. Su clave es su funcionalidad. La intención fue desarrollar un lenguaje que fuera comprensible por los humanos al mismo tiempo que lo suficientemente potente para expresar cadenas de acciones, procesos y computaciones concurrentes. No pueden hacerse referencias a una cantidad cinemática absoluta; las refencias siempre son al contexto mecánico actual. El marco de referencia del robot está fijado en el hombro, y los comandos como UP, DOWN, RIGHT, etc. refieren a este sistema de coordenadas. Lenny puede usar un nuevo procedimiento como parte de su lenguaje. Toma el nombre de una novela de Asimov, donde un robot llamado Lenny, se vuelve accidentalmente capaz de aprender.

LERNA

Lenguaje de 1984 del Institute de Microtechniques de l'ERFL, Suiza. Se trata de un lenguaje de programación a nivel de tarea.

LINGRAPHICA

Es un lenguaje visual utilizado para controlar robots en entornos no estructurados, que fue presentado por L. Leifer, M. Van der Loos y D. Lees en la V Conferencia Internacional de Robótica Avanzada (de IEEE Robotics y Automation Society), celebrada en Pisa (Italiza) en junio de 1991, con la ponencia "Visual Language Programming: for robot command control in usntructered environments".

Con este enfoque, el movimiento del robot y la planificación de tareas puede especificarse eficientemente y con mínimo entrenamiento usando primitivas texto-gráfico. Permite la conexión a una "galería de imágenes" con objetos gráficos organizados lingüísticamente. El tesauro se divide en

seis categorías: actores, acciones, ubicaciones, modificadores, cosas y otros. Las primitivas de movimiento en este sistema contienen acciones como rotación sobre un eje, balanceo (de una puerta), enroscado (perilla), movimiento controlado (insertar un objeto en una ranura) o movimiento en línea recta.

Con más de 2.000 pictogramas, incluso los no iniciados pueden definir una tarea para un robot. Sin embargo, puede ser difícil definiar los detalles de la tarea con imágenes o, al menos, puede durar largo tiempo encontrar las imágenes adecuadas.

LM (Language de Manipulation)

Desarrollado en el IMAG Robotics Laboratory de la Universidad de Grenoble en 1979/1980. En gran parte inspirado en AL, adoptando la mayor parte de sus conceptos, pero usado en un microcomputador (LSI-11/23 y 68000). Se firmó un acuerdo con Itmi, por el cual Itmi comercializa el lenguaje LM, que así puede ser adquirido por otros fabricantes de robots que no tienen que invertir en el desarrollo de métodos de control. Fue utilizado, por ejemplo, en el robot Renault TH8 y en robots Kemlin.

Presentado por J.C. Latombe y E. Mazer en el 11º Simposio Internacional de Robots Industriales, celebrado en Tokyo (Japón) en octubre de 1981, bajo el título "LM: A high-level language for controlling assembly robots".

LM-GEO

LM-GEO es una extensión de LM para incorporar especificación simbólica de destinos, intentando acercarse a un lenguaje a nivel de tarea. Trata de inferir las posiciones de los cuerpos a partir de las relaciones geométricas. Se parece en cierta medida a RAPT. Como LM, fue desarrollado por el IMAG Robotics Labotory de la Universidad de Granoble. En este caso en 1982. Fue publicado por E.Mazer en el Advanced Software in Robotics de Liege de 1983 bajo el título "LM-GEO: a geometric programming of assembly robots"

LMAC (Modular Command Language for Industrial Robots)

Desarrollado por la Universidad de Besancon, Francia, en 1984. Fue diseñado para garantizar un control seguro de dispositivos mecánicos en un entorno automatizado. Basado en Pascal, su clave es la modularidad, basada en la implementación de tipos abstractos de datos. Módulos de programa escritos en un variedad de lenguajes pueden referenciarse desde un programa LMAC, consiguiéndose una interfaz simple a programas existentes.

LPR (Language de Programmation pour Robots)

Desarrollado en 1978 como un proyecto conjunto de Renault y la Universidad de Montpellier, para el control de robots ACMA y Cybotech, producidos por Renault (también llamado control versión 5, o V5). Los sistemas de control precedentes usaban microcomputadores programados en ensamblador de forma específica para cada aplicación. Para responder a requerimientos internos más complejos, se desarrolló una nueva estructura de procesamiento basada en varios microprocesadores de 16-bits y procesadores bit-slice. Para facilitar el desarrollo de aplicaciones complejas se desarrolló LPR, diseñado usando el enfoque de Grafcet. Está basado en la definición de grafos de estados y transiciones entre estados. Los grafos están definidos como una jerarquía, de forma que todos los grafos del mismo nivel se ejecutan en paralelo. El lenguaje dispone de 24 puertos de entrada/salida para sensores y sincronización con equipamiento externo. En ese enfoque, el desarrollo se hace en máquinas potentes (minicomputadores VAX, Mitra o Philips) y luego el código se baja a los microprocesadores. LPR era considerado por Renault como una herramienta interna de trabajo no accesible por los usuarios.

Nota: otros autores lo datan en 1983

LRP (Live Robot Programming)

Desarrollado por Johan Fabry y Miguel Capusano en 2014, en el Departamento de Ciencias de la Computación de la Universidad de Chile y publicado en Advances in Artificial Intelligence IBERAMIA 2014, con el título de "Live Robot Programming".

La idea de Live Programming data de los años 90, encontrándose ya referenciado en trabajos de Tanimono, aunque tomó auge desde la charla de Bret Victor en la Canadian University Software Engineering Conference (CUSEC) de 2012. Así LRP es un lenguaje de programación de robots que impementa Live Programming, por lo que permite programar sobre un robot o simulador que se encuentra ejecutando el código mientras éste esta siendo escrito.

LRP se basa en máquinas de estado anidadas. El comportamiento del robot se modela mediante una máquina de estados, dentro de la cual cada estado representa una etapa de su comportamiento, y cada transición, un cambio de estado originado por un evento. Al tratarse de máquinas de estado anidadas, cada estado puede contener en su interior otra máquina de estados completa, y así sucesivamente. Así, cada etapa del comportamiento puede detallarse en tantos niveles de profundidad como se quiera.

En cada máquina se pueden definir acciones, que son trozos de código en lenguaje Smalltalk, que se ejecuta atómicamente. Las acciones pueden ser de tres tipos: onentry si se ejecutan cuando el estado pasa a ser el estado activo; running si se ejecuta cuando el estado está activo, una vez por ciclo de ejeución; onexit si se ejecuta cuando el estado deja de estar activo. La

máquina también puede tener eventos, que son acciones específicas que determinan condiciones para que ocurran las transiciones, o sea, los cambios de estado.

MAL (Multipurpose Assembly Language)

Lenguaje desarrollado en 1979 por el Instituto Politécnico de Milán para tareas de ensamblaje. Esta primera versión de MAL estaba basada en el lenguaje Basic y corría en un robot de investigación.

En 1983 se desarrolló una nueva versión basada en Fortran, ejecutándose en los robots Sigma e implementado sobre LS 11/02 y PDP II/34, requiriendo solo 20 kb de memoria. Posteriormente múltiples robots soportarían MAL.

Fue desarollado en Fortran IV excepto una pequeña interfaz en ensamblador. Es interpretado y transportable, y permite el movimiento simultáneo de dos brazos. Dispone de instrucciones de asignación, control, entrada/salida y operaciones del robot como MOVE, INCR, ACT, etc. MAL solo conoce posiciones articulares.

Permite programación independiente de diferentes tareas y proporciona semáforos para sincronización. Está implementado de tal manera que un cambio en el robot no requiera reescribir completamente el sistema. Por ejemplo, cambiar de un robot cartesiano a uno polar puede requerir la modificación de un solo módulo.

MAL se compone de dos partes, una dedicada a la compilación del lenguaje de entrada en una forma interna, y la otra dedicada a la ejecución del código intermedio. El módulo de compilación proporciona facilidades para crear, actualizar y mantener el código fuente. El sistema está orientado a la línea, en el sentido de que después de que se escribe cada línea, el compilador comprueba los errores sintácticos y, en su caso, muestra el mensaje de error correspondiente. El programa puede ser ejecutado parcialmente, parando en una cierta instrucción, y después se puede reanudar la ejecución.

MAL utiliza variable reales y enteras, y computa expresiones con operadores aritméticos y ciertas funciones definidas. El conjunto de instrucciones incluye bucles DO, test IF-THEN, GOTOs y llamada a subrutina con argumentos. Se pueden definir diferentes tareas y sincronizarlas con instrucciones SET-WAIT.

Permite movimientos absolutos y diferenciales para mover los seis ejes de dos brazos. Se pueden manejar sensores y actuadores, y se puede alertar al usuario con una campana y luces.

MAPLE

Lenguaje desarrollado en 1976 por IBM, basado en el PL/I, publicado por

Darringer, J.A. y Blasegn, M.W. en "MAPLE: A High Level Language for Research in Mechanical Assembly". Es un lenguaje tipo AL y no tuvo un uso significativo.

MAPS

Desarrollado en Turín (Italia), presentado en la 6ª COMPSAC (Computer Software and Applications Conference) de IEEE, de novimebre de 1982 en Chicago por G. Bruno bajo el título "On the use of abstractions in manipulator programming".

MAPT (MICRO-APT)

Subconjunto de APT para ejecutarse en microcomputadores

MCL (Manufacturing Control Language)

Desarrollado en 1979 EN el marco de los proyectos de ICAM (Integrated Computer-Aided Manufacturing), financiados por el Departamento de Defensa de USA, para resolver de forma unificada todos los problemas asociados a la programación de robots. La versión ICAM original usaba un IBM-370 en modo batch. Luego el proyecto se encomendó a McAuto (asociada a McDonell Douglas) que hizo una versión comercial interactiva (MCL/11) usando un PDP-11. Esta versión fue introducida en 1981.

MCL es una extensión de APT-III, y fue utilizado para especificar movimientos de robots, operaciones de sistemas de visión, modelado de imagen, coordinación de procesos, lógica condicional en tiempo real, múltiples sistemas de coordinación, macros, extensiones de lenguaje en tiempo de compilación. Usa una base de datos CAD como fuente de la información geométrica. Como ejemplo, se utilizó en los robots T3 de Cincinnati Milacrom y Allegro de Westinghouse.

MELFA BASIC (MBA)

Lenguaje desarrollado por Mitsubishi basado en Basic. A veces abreviado como MBA. Ha ido evolucionando en diferentes versiones. En 1992 surgió Melfa Basic 3 (MBA3) y existe hasta la versión 6. Se intercalan instrucciones propias de robot (como MOV para movimiento) entre líneas de Basic.

MHI (Mechanical Hand Interpreter)

Desarrollado en 1960 por el MIT. Posiblemente el primer lenguaje de programación a nivel de robot, desarrollado para uno de los primeros robots controlados por computador, el MH-1, que estaba equipado con varios sensores táctiles binarios en toda su mano, una serie de sensores de

presión entre los dedos y fotodiodos en la parte interior de los dedos. Toda esta capacidad de sensores condicionó el modo de programación desarrollado. MHI corría en un intérprete implementado en el computador TX-0. El estilo de programación se basa en movimientos protegidos, es decir, moviéndose hasta que se detecte cierta condición sensorial. Las primitivas del lenguaje son: move que indica un movimiento en una cierta dirección y con una cierta velocidad; until para testear sensores para ver si se cumplen ciertas condiciones; ifgoto y ifcontinue para ramificar el el programa según condiciones. No tenía otras posibilidades aritméticas ni otras estructuras de control más allá del testeo de sensores. Pasaron varios años hasta que se implementaron otros lenguajes más generales.

MicroPLANNER

Desarrollado en 1970 por G.J. Sussman, Terry Winograd y Eugene Charniak en el MIT. Es un subconjunto de PLANNER (un lenguaje de Cal Hewitt para escribir pruebas de teoremas, que nunca se implementó completament), implementado en Lisp.

MINI

Desarrollado en 1972 por el MIT. Consiste en un sistema LISP con extensiones de nuevas funciones. Su principal atractivo fue la disponibilidad de un brazo de 6 grados de libertad con sensor de fuerza, alcanzándose una sensibilidad del orden de una onza. Las funciones básicas eran el posicionamiento de cada eje (SETM), leer la posición y sensores de fuerza (GETM) y esperar alguna condición (WAIT). No tiene operaciones geométricas ni de control, pero son fáciles de implementar en procedimientos LISP.

ML (Manipulator Language)

Desarrollado en 1973 por IBM, nunca se lanzó comercialmente. Es un lenguaje de bajo nivel comparable al lenguaje ensamblador. El lenguaje proporcionaba posibilidad de movimientos protegidos mediante comandos SENSOR que permitían monitorizar sensores de forma que un cierto movimiento se interrumpa si el valor del sensor se sale de un cierto rango establecido. Permitía dos tareas de robot paralelas con sincronización entre ambas.

MML (Model-base Mobile robot Language)

Desarrollado en la Universidad de California (Santa Bárbara, USA) en 1989, publicado por Y. Kanayama en el International Workshop on Intelligent Robots and Systems de IEEE en ese mismo año, en la ponencia "Locomotion Functions for a a Robot Language".

En un lenguaje de programación off-line de altao nivel, que contiene funciones de sensor de alto nivel, descripción del modelo geométrico, planificación de trayectorias, etc. Tiene un importante concepto de funciones lentas y rápidas, esencial para el control en tiempo real de los robots. Una función lenta se ejecuta secuencialmente, y una función rápida se ejecuta inmediatamente. Otro concpeto importante es la separación entre postura actual y postura de referencia, lo que permite un control preciso y suave del movimiento, y una corrección dinámica.

MML fue un buen intento como propuesta de estándar de funciones de locomoción para robots móviles.

MRL (Multiagent Robot Language)

Es un lenguaje de especificación ejecutable para control de robots multiagente, presentado por H. Nishiyama, H. Ohwada y F. Mizoguchi en la International Conference on Multiagent System, realizada en julio de 1998, bajo el título "A Multiagent Robot Language for Communication an Concurrency Control".

Los robots físicos y los sensores son vistos en MRL como agentes inteligentes, y MRL se concentra en la comunicación a nivel semantico entre ellos. Cada agente robótico tiene su propia fuente de conocimiento de las reglas y procedimientos para realizar las tareas solicitadas por un agente externo. MRL tienen métodos para control de la concurrencia, manejo de eventos y negociación de los agentes. Un agente MRL puede contener subagentes, pero el superagente solo tiene dos canales de comunicación, uno para el superagente y otro para los subagentes. Un agente nunca envía un mensaje a un subagente concreto, pero los subagentes testean si hay mensaje disponible. El agente de más alto nivel, llamado agente raíz, gestiona indirectamente todos los agetnes.

El lenguaje MRL fue testeado con un sistema multirobot, que incluía 4 manipuladores, un sensor de visión y una cámara móvil. Estos objetos se conectaron a varias estaciones de trabajo vía interface serie. MRL también se aplicó satisfactoriamente a robots móviles.

Todos los programas MRL se compilan primero en lenguaje KL1, que es un lenguaje de programación lógica paralela diseñado en Japón en e Fifth Generation Computer System Project. El código KL1 se compila después en código C que ya puede ser ejecutado bien en sistemas UNIX paralelos, o en computadores basados en DOS.

El lenguaje permite control de concurrencia, control de prioridad y negociación pero su legibilidad es pobre y requiere tres fases de compilación hasta que pude ser ejecutado en robots.

MYBASIC

Lenguaje comercial desarrollado por Hobert

NC/360

Proporcionado por IBM en 1965, basado en APT III versión 7, preparado para procesamiento en IBM 360. Tiene un conjunto de características no definidas en el estándar APT, pero es básicamente compatible con los sistemas existentes APT III.

NELAPT (National Engineering Laratory APT)

Dialecto de APT diseñado en 1968 por el Laboratorio Nacional de Ingeniería del Reino Unido para el ICL 1900.

NUCOL (Numerical Control Language).

Extensión de APT III de 1967, y publicado un año después por S.C. Ambrosio en "NUCOL: Preliminary Report". Desarrollado en Italia tras 5 años de uso de APT en plantas de FIAT, e influenciado también por Algol 60.

PADL (Part and Assembly Description Language)

Desarrollado en 1975 por la University of Rochester, como una extensión de PDL (1973-74). Descrito ese año en "An introduction to PADL" en el Production Automation Technical Memorandum 22 de dicha universidad, es un lenguaje para definir completamente la geometría de las piezas. La geometría nominal (en términos generales, "la forma") de un objeto que se definirá en PADL se especifica a través de la Geometría Sólida Constructiva (CSG), es decir, como una composición de conjuntos teóricos regularizados de "bloques de construcción" sólidos primitivos.

En 1978 se presentó PADL-1 como una segunda versión de PADL desarrollada por la misma universidad. Esta versión atiende a objetos definidos como combinaciones de bloques y cilindros primitivos posicionados ortogonalmente. PADL-1 posee un rico conjunto de comandos de alto nivel para producir dibujos de ingeniería de los objetos definidos en una variedad de estilos: ortografías dimensionadas de una y tres vistas, perspectivas e isométricas, vistas de sección y dibujos de interferencia.

En 1982, en el IEEE Computer Graphics and Applicatons, C.Brown presenta PADL-2 como evolución de PADL-1, que no llegó a usarse como tal, aunque surgió UniSolid, una comercialización de PADL-2.

PAL (Portable AL)

Es una implementación del entorno de programación AL realizada en al Universidad de Karlsruhe sobre minicomputadores y microcomputadores. Incorpora un compilador AL, POINTY, y us sistema de depuración. Se desarrolló un sistema operativo dedicado para soportar multitarea y E/S. Corría sobre PDP 11/34 y LSI 11/2 para controlar el robot PUMA 500.

PAL

Desarrollado por la Universidad de Purdue (USA) en 1978, presentado por K. Takase, R.P. Paul y E.J. Berg en el IEEE COMPSAC celebrdo en Chicago en 1979 bajo el título "A structured approach to robot programming and teaching". En PAL las tareas se representan en términos de coordenadas cartesianas estructuradas. Es decir, un programa PAL consiste en una secuencia de ecuaciones de coordenadas homogéneas que referencian a las ubicaciones de los objetos y al efector del robot.

Cada declaración de movimiento es una solicitud para posicionar y orientar el manipulador de modo que se satisfaga una ecuación de posición, que representa una cadena cinemática cerrada de transformaciones homogéneas. En el proceso de enseñanza, la transformación que representa la posición del final del manipulador se define utilizando el propio manipulador, y se obtiene una ecuación que relaciona cualquier transformación indefinida. Al definir estas ecuaciones durante la interpretación del programa de tareas y resolverlas progresivamente, se enseña la tarea. El método es aplicable tanto al control de robots industriales prácticos como al control de robots inteligentes.

PAM

Lenguaje desarollado en 1982 por Remak para su robot.

PASLA (Programmable Assembly Robot Language.)

Lenguaje desarrollado por Nippon Electric Company, Ltd. (NEC) de Japón en 1983. Es un lenguaje dirigido al movimiento que consiste en 20 instrucciones básicas.

PASRO (Pascal for Robots)

Desarrollado en 1983 por Biomatik en Alemania Oriental. Es un lenguaje basado en Pascal con tipos de datos y procedimientos añadidos para tareas específicas de robot. Su desarrollo se basó en la experiencia con AL. Los procedimientos se proporcionan para dirigir el brazo punto a punto o a lo largo de un camino continuo. Publicado por C. Blume en "Pasro-Pascal for Robots" (Springer Vertag, 1985).

PDL2

Es un lenguage desarrollado por Comau Robotics, de tipo Pascal con características especiales para programar aplicaciones robóticas, lo que incluye: movimiento de los brazos del robot; enviar y recibir información; controlar el orden en que se ejecutan las instrucciones; testear errores y otras condiciones especiales.

PILOT (Programming and Interpreted Language Of Actions for Telerobotics)

Es un lenguaje visual, imperativo e interpretado para programación en telerobótica. Fue presentado por J.L. Fleureu, E. Le Rest y L. Marcé en la 2nd IFAC Conference on Inteligent Autonomous Vehicles, realizada en junio de 1995 es Espoo (Finlandia), bajo la ponencia "PILOT: A Language for Planning Mission".

En PILOT las primitivas son cajas gráficas con símbolos que describen sus efectos. Hay 7 primitivas disponibles:

- sequentially: significa que la siguiente acción empeiza cuando trermina la anterior

- conditional: para hacer una elección de acuerdo a variables booleanas o ecuaciones

- iterative: genera un blucle mientras una variable booleana se evalúa a falsa

- parallelism: empieza n acciones en paralelo, y termina cuando todas las acciones acaban

- pre-emptive: se usa para parar un paralelismo cuando una acción finaliza

- reactivity: para la misión y espera un cierto evento del operador o de un sensor

- alternative: permite al operador hacer una elección entre dos acciones

En PILOT se modelan estas acciones: Move(V), Turn(a), Detect L1, Initialize, Siren, Detec line, Line-guiding(V), y Approach line (a, V), donde V es una velocidad, y "a" es un ángulo.

PILOT deriva de YALTA y se usa para planificar misiones y modificarlas durante la ejecución. Permite la ejecución de acciones de forma secuencial y concurrente, y respuesta a eventos sensoriales. El conjunto de comandos es bastante limitado, pero su legibilidad es buena.

PLAW (Programming Language for Arc Welding)

Desarrolado en 1980 en el Electrotechnical Research Center de la empresa

Komatsu Ltd, para sus robots de la serie RW Cartesian. Es un lenguaje tipo BASIC específico para soldaduras al arco de CO2. Permite la definición de tres tipos de coordenadas y control adaptativo. Es un lenguaje adecuado para soldadura "inteligente", o sea, soldadura que implica el uso de sensores (por ejemplo, el seguimiento del cordón de soldadura). Lo presentó R. Abe en el 11º ISIR celebrado en Tokyo en 1981.

POINTY

Desarrolldo en 1975 sobre un PDP-11 en el Laboratorio de Inteligencia Artificial de Standford. Es similar a AL, e intenta ofrecer una versión más interactiva en términos de programación. El modelo de objetos se especifica y se construye incrementalmente con un lenguaje interactivo de alto nivel, mientras el manipulador se emplea como herramienta de medición para definir puntos de interés en las partes del ensamblaje. Una de las primeras publicaciones encontradas fue la realizadas por G. Gini y M. Gini, del Instituto Politécnico de Electrotécnica y Electrónica de Milán (Italia) en el 2º IFAC/IFIP Symposium on Information Control, bajo el título "Pointy. A philosophy in Robotic Programming". Y en el mismo año, los mismos autores también lo presentan en el LMPRI bajo el título "Pointy: a system for developing manipulator programs"

PRS (Procedural Reasoning System)

Es un lenguaje de alto nivel de control y supervisión para robots móviles autónomos, presentado por F.F. Ingrand, R. Chatila, R. Alami y R. Robert en la Conferencia Internacional de Robótica y Automatización de IEEE celebrada en Minnapolis en 1996, bajo el título "PRS: A High Level Supervision and Control Language for Autonomous Mobile Robots".

Ejecuta procedimientos, planes y scripts en entornos dimámicos. El núcleo de PRS se compone de tres elementos principales: una base de datos, una librería de planes y un gráfico de tareas.

La base de datos contiene hechos que representan la visión del sistema del mundo, que se actualiza automáticamente cuando aparecen nuevos eventos. La información de la base de datos puede ser tal como la posición del robot, el contenedor que transporta, punteros a las trayectorias producidas por el planificador de movimiento, los recursos actualemente en uso, etc. La base de datos también puede contener predicados que activan algún código C interno para recuperar su valor.

Cada plan en la librería describe una secuencia particular de acciones y tests que pueden realizarse para alcanzar metas, o reaccionar ante ciertas situaciones. PRS no planifica combinando acciones, sino escogiendo planes alternativos o caminos de ejecución en un plan que se está ejecutando. Por tanto, la librería debe contener todos los planes, procedimientos y scripts necesarios para realizar las tareas del robot.

El gráfico de tareas es un conjunto dinámico de tareas que están ejecutándose. En un roboto móvil, el gráfico de tareas puede contener las tareas correspondientes a varias actividades que está realizando el robot (una para refinar la misión actual, otra para monitorizar mensajes de entrada de uns estación central que da órdenes, otra para el menejo de comunicaciones , etc).

Un intérprete, recibe nuevos evento y metas internas, y chequea condiciones, y en base a ello, selecciona los planes (procedimientos) apropiados, pone los procedimientos seleccionados en el gráfico de tareas, escoge una tarea entre las raíces del gráfico y finalmente ejecuta un paso del procedimiento activo en la tarea seleccionada. Ello resultará en una acción primitiva o el establecimiento de una nueva meta.

En PRS, las metas (goals) son descripciones del estado deseado asociado al comportamiento para alcanzar ese estado. La meta para posiciones el robot se escribe:

(achive (position-robot 20 10 45))

Este meta se sataisface si el robot se encuentra es esa posición. La meta para testear si el robot está en una posición concreta sería:

(test (position-robot 20 10 45))

Para averiguar la posición del robot pueden usarse variables (test(position-robot $x $y $theta)). Ello implica que esos valores están en la base de datos o que existe un procedimiento para avergiar la posición.

La meta para esperar hasta que el robot esté preparado para moverse:

(wait(robot-status ready-for-displacement))

En este otro ejemplo asegura que el roboto no esté en estado de emergencia durante el movimiento a una posición.

(&(achive (position-robot 20 10 45))(preserve(~(robot-status-emergency))))

La meta para mantener la batería al menos al 20% mientras se ejecuta una treayectoria sería:

(&(achive(execute-trajectry @traj))(maintain(battery-leve 0,20000)))

Este lenguaje permite ejecutar múltiples tareas en tiempo real. Se puede condensar mucha información en una simple línea de comando, pero su legibilidad es pobre.

RAIL

Desarrollado por Automatix, una empresa formada en 1978 por el fundador de Computervision y un número de investigadores del MIT y Stanford, incluyendo a Victor Scheinman, que había estado en Unimation y Vicarm. RAIL surgió en 1981 para controlar los robots AID 800 de Automatix para inspección, soldadura por arco y ensamblaje. Fue el primer lenguage de

robots que se podía aplicar tanto a problemas de manipulación como a problemas de visión. Está muy basado en PASCAL, y se programa y ejecuta en una máquina desarrollada por Automatix usando un microprocesador 6800. Algunas de las aplicaciones robóticas son la soldadura con control de camino continuo (robot de Hitachi) con la posibilidad de usar sensores para el seguimiento del cordón de soldadura y el ensamblaje (robot DEA). En 1981 se usaba en los sistemas de Cybervision, Autovision y Robovision de Automatix. Los tipos de movimiento soportado son en línea recta, eje de coordenadas y trayectorias oscilantes (para soldadura).

RAPID (Robotics Application Programming Interactive Dialogue)

Es una evolución de ARLA, y fue introducido en 1994 en los sistemas S4 de ABB. Un programa RAPID se estructura en módulos (Module … End Module). Cada módulo tiene un conjunto de datos, una rutina principal (main) y otras subrutinas. Cada rutina puede ser un procedimiento (Proc...End Proc), una función (Func...End Func) o una interrupción (Trap...End Trap). Los datos pueden ser constantes (Cons), variables (Var) o persistentes (Pers). Hay datos atómicos (num, bool, string) y registros (pos, orient, pose, confdata, loaddata, tooldata, robtarget, spedddata, zonedata). Cuenta con instrucciones de movimiento (MoveJ, MoveL, MoveC), de entrada/salida (set, reset, setdo, dinput), de espera (waitdi, waittime, whileuntil), así como las habituales de control de flujo (if, for, while, test, goto). Se puede programar en el Pendant o con programas de ordenador, como el Rapid SyntaxChecker (analizador sintáctico), el ABB Deskware, el RobotStudio, o el Festo Cosimir.

RAPT (Robot Automatically Programmed Tool)

Desarrollado en 1977 por la Universidad de Edimburgo (Escocia, Reino Unido), y publicado en 1978. Es un lenguaje basado en APT, en el que las tareas se describen en términos de objetos, relaciones entre objetos y movimiento de objetos. Un programa RAPT consiste en la descripción de las partes involucradas, el robot, el área de trabajo, y el plan de ensamblaje. El plan es una lista de relaciones geométricas que expresan lo que debe realizarse en cada paso. El lenguaje es independiente del tipo de robot, y un postprocesador genera programas VAL a partir del código fuente RAPT.

RCCL. (Robot Control C Library)

Desarrollado en 1983 por la Universidad de Purdue (USA). El enfoque consiste en embeber comandos de robot en una librería de rutinas escritas en C, haciendo uso del sistema operativo Unix. Los objetivos del lenguaje eran proporcionar comandos potentes de control del manipulador, procesamiento flexible de datos, y una potencial estandarización. Se ha utilizado en robots PUMA.

RCL (Robot Command Language) (1)

El lenguaje RCL fue diseñado en 1983 por Rensselaer Polytetechnic Institute (RPI) de Nueva York (USA) para los brazos PACS de Bendix, pudiendo controlar un solo brazo. Está escrito en ensamblador, utilizando como CPU un PDP 11/03. La programación en RCL proporciona ramificación condicional e incondicional. Los movimientos se pueden indicar en ángulo o en coordenadas cartesianas; también movimientos absolutos. RCL proporciona además movimientos relativos y en línea recta. No dispone de subrutinas. Tampoco permite incluir ficheros como código ejecutable. Está provisto de comandos simples de sensores táctiles binarios, pero no comandos de visión. Tienen capacidades limitadas en cuanto a definición y provisión de capacidades de transformación de coordenadas.

RCL (Robot Command Language) (2)

Expuesto por R.J.C. Fraser y C.J. Harris en el Coloquio sobre Control Inteligente de IEEE de 1991, bajo el título "Infraestructura para control robótico en tiempo real: aspectos del Robot Command Language". Está basado en la arquitectura NASREM (National Standard Reference Model), de J.S. Albus, H.G. McCain y R. Lumia presentada en 1987 en la Goddard Conference on Space Applications of AI and Robotics, de la NASA. NASREM es una arquitectrura jerárquica y colateral, con diferentes niveles de abstracción. Los niveles más altos representan comandos de sistema simbólicos e inteligentes, mientras que en los niveles más bajos el modelo es el control inteligente geométrico y dinámico. La sintaxis de RCL es jerárquica y los elementos del lenguaje se definen sobre otros elementos RCL: el telerobot existe en el nivel 4, brazos y cámaras en el nivel 3, etc.

<Telerobot#1> ::= <Arm#1> <Arm#2> <Platform> <TVCamera#1>

<Arm#1> ::= <Joint#1> <Joint#2> <Hand#1>

Ejemplos de comandos RCL:

- position_trajectory(tiem1, x1, y1, time2, x2, y2,...)
- tur_on_place(time1, direction1, time2, direction2,...)

Los comandos se interpretan para producir comandos de actuador.

RCS (Real-time control system)

Diseñado en 1984 por el NBS (National Bureau of Standards), por Anthony J. Barbera, M.L. Fitzgerald, James S. Albus y Leonard S. Haynes. En RCS se definen metas de alto nivel y se van descomponiendo a través de una sucesión de niveles en el que cada uno produce comandos más simples para el siguiente nivel. El nivel más bajo genera las señales que dirigen al robot, pinza y otros actuadores. Cada nivel es un proceso separado con un alcance limitado, independiente de los detalles de otros niveles. Para especificar las tareas y su descomposición en procesos, se cuenta con un lenguaje de

programación y un entorno de desarrollo. Los programas en cada nivel de control se expresan como tablas de estados, y el entorno de programación permite la generación, edición, emulación y evaluación de esas tablas de estados.

Para control de robots, se definieron tres niveles: TASK, E-MOVE y PRIMITIVE.

El nivel Task (tarea) desarrolla una serie de puntos objetivo usando nombre de objetos y ubicaciones. Incluye comandos como Acquire, Move, Release, Transfer, o Clear.

El nivel E-Move desarrolla una trayectoria desde el último punto ordenado en el nivel Task hasta el punto de meta actual. Una trayectoria puede ser una simple línea recta entre los dos puntos o más compleja, implicando trayectorias de salida, intermedias y de aproximación. Algunos comandos son: Locate, Pick-up, Move-to, Move-to-obj, Release, Pause, Flood-flash, Line-flash

El nivel Primitive es el que interactúa con el robot y la pinza. Genera puntos intermedios a lo largo de la trayectoria definida por el nivel E-Move y pasa esos puntos al controlador del robot. Algunos comandos de este nivel son: Goto, Gothru, Approach-position-fingers, Departure-position-fingers, Immed-grasp, Pause.

Más abajo del nivel Primitive estaría el interface con el robot (las posiciones formadas por nueve números) y el interface con la pinza (comandos Position y Grasp)

REMAPT

Es una versión de ADAPT modificada en 1968 por General Electric Corporation para usarlo en sus sistema de tiempo compartido Mark II. En resumen, REMAPT es un lenguaje de procesamiento para 2 ejes con facilidades para 3 ejes y disponible en una base de tiempo compartido.

RIPL (Robot Independent Programming Language)

Lenguaje basado en RIPE (Robot Independent Programming Environment), un entorno de programación de robots orientado a objetos. Fue creado por Miller y Lennox en 1990. La arquitectura RIPE consiste en un enfoque multiprocesador jerárquico que utiliza tanto procesadores generales distribuidos como procesadores de propósito específico. La arquitectura permite el control en tiempo real de diversos subsistemas complejos, posibilitando la comunicación y coordinación entre ellos. Soporta la programación automatizada basada en modelos de dispositivos robóticos y máquinas, control basado en sensores en tiempo real, manejo de errores, comunicaciones robustas e interfaces gráficas para el control del robot. Los objetos incluyen robots, sensores, efectores, máquinas CN, y otros dispositivos. RIPL consiste en un conjunto de clases genéricas para

representar estos objetos y sus interfaces.

RISE

Lenguaje comercial desarrollado por Prab (USA) para sus robots G-Series.

RLC

Desarrollado por RPI para el robot PACS, está escrito en ensamblador y es interpretado, utilizando como CPU un PDP 11/03

ROBEX (Roboter Exapt)

Desarrollado en 1980 por la Universidad de Aachen, está basado en EXAPT (Extended APT, una extensión de APT) y en FORTRAN IV, tiene un limitado nivel de servocontrol y una entrada gráfica. Se utilizó en varios robots de Sieman. Presentado por W. Eversheim en Annals of the CIRP (International Academy for Production Engineering) vol. 30, de 1981 bajo el título "Off-line programming of numerically controlled industrial robots using de Robex programing system)

ROBOCAM

Desarrollado en 1985 por Prab (USA) para sus robots de Prab G-Series. Cuenta con una interface CAD.

ROBOFORTH

Desarrolado en 1982 por la empresa Intelligent Artefacts de Cambridge (Reino Unido), basado en el lenguaje Forth y presentado por David Sands, el creador de la empresa, en el volumen 7 de Microporcessors and Microsystems, de 1983, bajo el título "Using Forth to control a robot arm". Ese mismo año de 1982 la empresa cambió a Cyber Robotics, que acabó cerrando por falta de ventas. David Sands fundó en 1986 Sands Technology, que e 1991 se expandió a América, y desde 1997 se conoce como ST Robotics.

Para escribir programas en RoboFoth, se utiliza una aplicación windows llamada RobWin, pero el programa se ejecuta en el controlador del robot. RoboForth dispone de unas 500 palabras disponibles. Las posiciones del brazo se especifican en coordinadas relativas respecto a una posición central conocida como HOME.

ROBOML

Es un lenguaje basado en XML diseñado para servir como lenguaje común para programación de robots, comunicación de agentes y representación del conocimiento. Fue descrito por M. Makatchev y S.K. Tso en la IEEE International Workshop on Robot and Human Interactive Comunication celabrada en Osaka (Japón) en septiembre de 2000 (ROMAN 2000), bajo el título "Human-Robot Interface Using Agents Communicating in a XML-Based Markup Language.

Los elementos para la comunicación de agentes en RoboML son *set*, *get* y *subscribe*, que pueden tener como atributos opcionales *sender*, *receiver* y *ontology*. En RoboML se definen estos elementos contenedores para el hardware: robot, wheel, motor, sensor, controller, etc. Estos elementos pueden er especificados por el nombre y los atributos.

Para programación de robots, RoboML solo da el marco de referencia. Los lenguajes disponibles deben ser traducidos a RoboML.

RoboML es potente con interfaces hombre-robot basados en agentes vía Internet, pues XML es soportado por navegadores.

ROBOS (Robot Operating System)

Desarrollado por Philips en 1983, presentado por P. Saraga en el Colloquium on Robot Operating Systems de IEEE celebrado ese año en Londres, bajo el título "ROBOS, towards a general purpose robot operating system". Es concevido, no como un lenguaje, sino como un sistema operativo de robot (por analogía a los sistemas operativos de computadores), en el sentido de que contiene una serie de facilidades que son requeridas normalmente en el desarrollo de sistemas robóticos controlados por sensores. ROBOS tambiém proporciona un framework y una disciplina para guiar en las tareas de producción de software.

RoboTalk de Rhino

Desarrollado por Rhino Robotics Ltd., como un lenguaje de control robótico para sus robots de la serie XR, robots SCARA, o los controladores Mark. El conjunto de comandos es pequeño, potente y fácil de aprender.

La referencia más antigua encontrada es en "Robotalk: a new language to control the Rhino robot" de H.S. Sandhu y Herbert Schildt, de 1985.

RoboTalk de Standford

Desarrollado en 1992 en Stanford, es un lenguaje de control y ensamblaje basado en Forth, con extensiones de bajo nivel como las interrupciones y registros de propósito especial, así como trazas de C, Pascal y calculadora HP48.

Fue usado en el juego RoboWar de Davis Harris, implementado sobre Macintosh.

RobotScript

Diseñado en 1998 como un lenguaje de scripting universal para el URC (controlador robótico universal). Mencionado por John Lapham en el Industrial Robot Volume 26 de 1999 en el artículo "The introduction of a Universal Robot Programming Language"

ROCOL (Robots Control Language)

Desarrollado en 1976 por el Instituto Politécnico de Leningrado (URSS). Publicado por S.I. Novatchenko, V.V. Nickiporov y V.A Paulov en Robotechnica 1976 bajo el título "ROCOL, language for the robot control" Se utilizó en el robot expermimental LPI-2.

ROCOL incluye tres tipos de operadores: básicos, de edición y de control.

Los operadores básicos sirven para describir la lógica del programa de funcionamiento del robot. En este lenguaje, el elemento básico de significado es la Directiva, un conjunto de operadores básicos que especifican una ejecución de una acción. A una Directiva se le da un nombre, y luego pueden hacerse llamadas a ellas usando ese nombre. El nombre de una Directiva puede utilizarse en la definición de otra Directiva. Los operadores básicos de ROCOL se dividen en tres grupos:

- los que realizan operaciones lógicas y aritméticas con ficheros de datos, y operaciones de salto condicional e incondicional en el programa.

- los que aseguran la ejecución de varias acciones específicas del robot. Por ejemplo, abrir y cerrar los dedos de la mano.

- las que aseguran la comunicación entre hombre y robot.

Cada operación básica tiene una serie de parámetros que se establecen al introducir la directiva en el campo de información del operador.

Las operaciones de edición permiten transformar el código objeto ROCOL en la memoria del computador. Las operaciones de control se usan para especificar el régimen de trabajo del robot y del computador de control. Con estas operaciones se puede interactuar con el robot durante su funcionamiento.

ROL (Robot Language)

Se desarrolló, junto al lenguaje LAMA-S, en el proyecto Spartacus entre 1976 y 1979, con el objetivo de desarrollar un sistema comercial completo para control computerizado (incluyendo software y hardware) adaptable a cualquier robot. La novedad vino por el hecho de que GIXI, una casa de

software asociada a French CEA, no tenía una política propia de venta de robots. Su mercado, sin embargo, eran los fabricantes de robots (y máquinas técnicamente similares) y firmas de ingeniería. El sistema se hizo disponible al mercado por GIXI en 1983. Es altamente modular, y puede ser utilizado por sistemas muy simples (p.ej de dos ejes) así como por configuraciones con varios robots, procesadores y sensores sofisticados. La programación se lleva a cabo en un PC.

ROLL (Robot Learning Language)

Al principio se llamó RPL-Learn, y fue presentado por Michael Beetz, Alexandra Kirsch y Armin Müller en 2004 en la 3rd International Joint Conference on Autonomous Agents & Multi Agent Systems (AAMAS), ,bajo el título "RPL-LEARN: Extending an autonomous robot control language to perform experience-based learning".

Una año después, en 2005, en la publicación de Alexandra Kirsch, Michael Sxhweitzer y Michael Beetz, Making Robot Learning Controllable: a case study in robot navigation, ya se le da el nombre de RoLL al lenguaje.

Con este lenguaje se pretende extender un lenguaje de control de robots con constructores para especificar experiencias, problemas de aprendizaje, exploración de estrategias, etc. Usando estos constructores, los problemas de aprendizaje pueden ser representados de forma explícita y transparente y convertirse en ejecutables.

Es una extensión de CPL (o CRAM-PL, CRAM Plan Language) para integrar funcionalidades de aprendizaje continuo en los programas de control de robots. Por tanto, como aquel, tiene una sintaxis de tipo Lisp.

ROMPS

Lenguaje a nivel de tarea desarrollado en 1984 por CAM-1 de USA y JAIS de Japón.

ROPL (Robot Programming Laguage)

Lenguaje desarrollado por Henderson Industries para sus robots. Podía ser utilizado por programadores inexpertos, y permititía controlar cualquier brazo robótico. Aparece mencionado en el artículo "B.E. Company of The Year" de la publicación "Black Enterprise" de junio de 1984.

ROPS (Robot Off-Line Programming System)

Desarrollado en 1985 por Cincinnati Milacron para sus robots T7000S y T8000S. Permitía la programción off-line con un PC (bajo MS-DOS) o un DEC VAX (bajo VMS). Con un módulo de comunicaciones se pasan los

programas del ordenador al robot y viceversa. Acepta datos de sistemas CAD/CAM. Es compatible con T3, un lenguaje anterior de Cincinnati Milacron

RPL (Robot Programming Language)

El lenguaje RPL fue diseñado en 1980 por SRI internacional (Stanfod Research Institute) para los robots PUMA de Unimation, pudiendo controlar un solo brazo. Está escrito en Fortran sobre un procesador LSI-11, y es un lenguaje no transportable, interpretado y compilado. La programación en RPL proporciona ramificación condicional e incondicional, y bucles do...loop. Los movimientos se pueden indicar en ángulo o en coordenadas cartesianas; también movimientos absolutos. RPL no proporciona movimientos relativos y en línea recta. Permite la llamada a de subrutinas, con pase de parámetros. No permite incluir ficheros como código ejecutable. RPL tiene los comandos más simples de sensores táctiles, y un sistema de visión complejo capaz de tomar una imagen (picture), determinados aspectos de los objetos (getfea), y reconocer objetos como uno de aquellos que tiene en su base de datos (recogn). Tiene capacidades limitadas en cuanto a definición y provisión de capacidades de transformación de coordenadas. Proporciona comandos para definir 'frames', transformaciones inversas y matrices múltiples.

Un programa RPL tiene un bloque común opcional, un bloque de programa obligatorio, y bloques opcionales de subrutinas. Proporciona herramientas de debugging, como mensajes de error exhaustivos, interrupción manual, facilidades de trazado, paso a paso, breakpoints, y almacenamiento y carga de procedimientos.

Ha sido usado por el brazo Unimation Puma 550 y el sistema de visión Machine Intelligence, los brazos hidráulicos Unimate 2000A y 2000B y un sistema de visión de SRI.

RPS (Robot Programming System)

Desarrollado por Stanford Research Institute (SRI), publicado por W.T. Park en 1981 en "The SRI Robot Programming System (RPS): an executive summary".

RSS (Robot Servo System)

Desarrollado por la Universidad de Illinois (USA) en 1983

SERF (Sankyo Esasy Robotic Formula)

Desarrollado en 1978 por Sankyo para sus robots de ensamblaje, en la investigación universitaria para el proyecto Scara dirigido por el profesor Makino. El robot de Sankyo está orientado a trabajos de ensamblaje de

pequeña escala, donde la paletización juega un papel importante. Su lenguaje textual fue el primero en introducirse en Japón. El lenguaje tiene un estilo similar al de los lenguajes de control numérico, pero permite el uso de bucles y tests. La programación se lleva a cabo utilizando una consola especializada (on-line o off-line) que puede ser abandonada cuando el programa se ha completado. El control está basado en un simple microprocesador Z80.

SCOL (Symbolic Code Language for robot)

Un lenguaje desarrollado por Toshiba para sus robots, como las series System Robot Serie (SR-654H y SR-606V), Assembly Robots (SR-414H) o Material Handling Robots (SR-2006V, SR-1806V y SR-2206V).

Los comandos de SCOL se pueden clasificar en varios grupos:

a) control del movimiento: movimiento del robot (MOVEx, READY), parar temporalmente el robot (DELAY), mover la mano del robot (OPENxn, CLOSExn), operaciones de interrupción y restablecer (BREAK, RESUME, PAUSE).

b) control de programa: monitorizar señales externas, teporizadores, etc (ON DO, IGNORE, IF THEN ELSE, WAIT, TIMER), controlar la ejecución (PROGRAM, END, GOTO, RCYCLE, RETURN, FOR TO NEXT, STOP, TASK, KILL, SWITCH, TID, MAXTASK), introducir comentarios (REMARK)

c) control de entrada/salida: entrada y salida de señales externas (DIN, DOUT, PULOUT, RESET, BCDIN, BCDOUT), entrada y salida de datos de comunicación (PRINT, INPUT)

d) condiciones de movimiento: especificar condiciones para controlar el movimiento del robot (CONFIG, ACCUR, ACCEL, DECEL, SPEED, PASS, TORQUE, GAIN, ENABLE, SETGAIN, DISABLE, NOWAIT, PAYLOAD, FREELOAD, SWITCH, MOVESYNC)

e) paletización: cargar una librería (LOADLIB), iniciar una paleta (INITPLT), mover una paleta a una posición (MOVEPLT)

f) funciones de cálculos: realizar cálculos de números reales (SIN, COS, TAN, ASIN, ACOS, ATAN, ATAN2, SQRT, ABS, SGN, INT, REAL, LN, MOD, LOGIO, EXP, AND, OR, NOT), realizar cálculos que implican posició y datos de coordenadas (HERE, DEST, POINT, TRANS), usar matrices (DIM, AS)

g) referencia de movimiento: testear el movimiento del robot (MOTION, MOTIONT, REMAIN, RAMINT), testear movimiento del sistema (MODE, CONT, CYCLE, SEGMENT), asignar un sistema de coordenadas (TOOL, BASE, WORK)

h) Otros: definir variables (GLOBAL, DATA, END), restaurar un valor (RESTORE), guardar datos (SAVEEND)

Se permite la llamada a subrutinas con pase de parámetros . El comando de movimiento (MOVE) puede ser tal cual, para un movimiento síncrono, o seguido de una letra para indicar el tipo de movimiento: MOVES movimiento de interpolación lineal, MOVEC movimiento por interpolación circular, MOVEA movimiento simple absoluto de los ejes, MOVEI movimiento simple relativo de los ejes, MOVEJ movimiento en arco. Los comandos OPEN y CLOSE abren y cierran la mano cuando se ha completado el movimiento, mientras que OPENI y CLOSEI lo hacen inmediatamente.

SCORBASE

Es el lenguaje incluido en los sistemas robóticos de Eshed Scorebot, una empresa ubicada en Tel Aviv (Israel), que comercializa el robot Scorebot, de 5 ejes. El robot tiene ocho entradas y ocho salidas digitales que pueden ser usadas para sincronizar los movimientos del robot con dispositivos externos. Por ello, tienen tanto comandos de movimiento, como comandos de entrada/salida. El lenguaje tiene varios niveles: scorbase1, scorbase2, scorbase3, scorbase pro, scorbase 4 y scorbase5. La diferencia es que en cada uno se introducen nuevos comandos y nuevas prestaciones.

Con el entorno de programación de scorebase3 se utiliza un PC como si fuera el "teach pendant" del robot, y se puede aprender y memorizar posiciones, así como reproducirlas. Una vez memorizadas las posiciones, pueden referenciarse por un número.

El entorno de programación es un entorno basado en menús. El usuario selecciona el tipo de comando que desea y entonces se le preguntan los parámetros. Es una funcionalidad similar a las calculadoras programables. El usuario dispone de 6 contadores numérico de 1 a 6, que pueden ser usados para bucles o contaje de eventos. Para almacenamiento general, están disponibles 16 posiciones de memoria para almacenar números de dos dígitos. Las salidas lógicas pueden encenderse o apagarse, o testearse su estado. Se permite la ramificación del programa en base al resultado del testeo de una entrada o salida, o por el valor de un contador. Se pueden definir hasta 16 subrutinas, numeradas del 1 al 16, que pueden ser llamadas por su número.

Como requiere el aprendizaje previo de las posiciones, este lenguaje estaría a mitad camino entre un lenguaje gestual punto a punto y un primitivo lenguaje textual ya que dispone de subrutinas, sensores binarios de entrada y salida y capacidades de bucles. Aunque no tiene las mas básicas capacidades de transformación de coordenadas que normalmente tienen los primitivos lenguajes de movimiento. Y tampoco proporciona tipos de datos simples como entero o carácter.

SIGLA (Sigma Language)

Fue el primer lenguaje comercial disponible para usar con robots industriales. Fue desarrollado en 1974 por Olivetti para sus robots Cartesian Sigma. Muy influenciado por los lenguajes de control numérico, y basado en el lenguaje AL, permite el control de varios brazos (hasta 4), con bucles, testeo de sensores (señales binarias), instrucciones de movimiento de ejes, etc. Soporta ejecución pseudoparalela de múltiples tareas y cierto control de fuerza simple.. Los robots Sigma se utilizaban para diversas aplicaciones: ensamblaje, taladrado, remachado, fresado y soldadura.

Durante años fue utilizado en la planta de Olivetti en Crema (Italia). Posteriormente los robots Sigma fueron fabricados en USA por Westinghouse junto con los robotos Puma. SIGLA compitió con VAL.

SIGLA se escribió en ensamblador, y empleaba un miniordenador con 8 kb de memoria. Es interpretado y transportable. Incluye: un supervisor, que interpreta el lenguaje de control del trabajo; un módulo de aprendizaje, que permite realizar aprendizaje por guiado; un módulo de ejecución, edición y salvado de programas y datos. Es por tanto, no solo un lenguaje dr programación, sino un entorno de programación

Aunque SIGLA surgió en 1974, su uso se extiende en 1978 tras su presentación en el 8º Simposio Internacional de Robots Industriales, celebrado en Alemania con el título "SIGLA: The Olivetti SIGMA robot programming language".

SPEL, Suwa seiko Production Equipment Language,

SPEL, de Seiko Epson, y su evolución SPEL+ es el lenguaje para el entorno de desarrollo Epson RC+ (Epson Robot Control) con la que se programan y simulan los robots de Epson. La primera versión data de 1984, cuando K. Saito la presentó en el Journal of the Robotics Society of Japan. La versión 3 es de 2003; la versión 4 de 2005; la versión 5 de 2006; la vesión 6 de 2009; la versión 7 de 2012; la última publicada, la 7.1.0 es de julio de 2014. Para uso con Epson RC controladores.

Tienen comandos de administración del sistema, de control del robot, de entrada/salida, de manejo de puntos, de cambio de coordenadas, de control del programa, de ejecución del programa, pseudo sentencias, de manejo de ficheros, de valor numérico, de cadena, operadores lógicos, de variables, de seguridad, de sensores de fuerza, de seguimiento del transportador, de base de datos, de detección de colisiones, etc.

SPLAT (Simple Provisional Language for Actions and Tasks)

Desarrollado en 1997 por Bill Gribble, dentro del grupo de investigación Qualitative Reasoning de la Universidad de Texas, en Austin. Es una librería para escribir reglas y planes de control de robots. Proporciona tanto un lenguaje (implementado en macros de Scheme) para especificar esas reglas

de control, como un entorno para ejecutarlas y monitorizarlas. Se desarrolló SPLAT como back-end de ejecución del plan y control de robot para el sistema de visión en tiempo real ARGUS.

SPLIT (Sundstrand Processing Language Internally Translated)

Desarrollado en 1963 por Sundstrand Machine Tool Company, operativo en los IBM 7090, 1620, 620 y 360. Estaba formado por 16 a 18 sentencias mayores y otras 40 sentencias menores.

Es un sistema de programación de posicionamiento multi-eje (de dos a cinco ejes) con capacidades de contorneado

SRCL (Siemens Robot Control Language)

Un lenguaje orientado a problemas para robots industriales de Siemens. Está compuesto de instrucciones. Todos los comandos son nemotécnicos, por ejemplo DEF para la declaración de definición. El lenguaje incluye más de 80 comandos.

SRIL-90

De Imperial, para correr en un micro 68000, y para sus robots.

SRL (Structural Robot Language)

Desarrollado en 1983 por la Universidad de Karlsruhe (Alemania) como parte de un proyecto de estandarización. Está basado en AL y Pascal, y el programa de usuario se traduce a un código independiente del robot denominado IRDATA. Como tipos de datos incluye todos los de Pascal y tipos abstactos de datos de inteligencia artificial. Las instrucciones se pueden ejecutar secuencialmente, en paralelo, de forma cíclica, o de forma retardada.

SRPL (Simple Robot Programming Language)

Un programa SRPL consiste en múltiples pasos, y cada paso puede ser: una posición del robot; un tipo de movimiento para mover a posición; varias funciones como anulación, suspensión, llamada a subrutinas, etc. Los programas SRPL se generan eficientemente en un entorno orientado a menú, las operaciones de sistemas son fáciles de aprender incluso para personas no expertas en robótica, y la funcionalidad cubre la mayoría de problemas en las áreas de aplicación. Se presenta en la IFAC (International Federation of Automatic Control) de Viena (Austria) en 1992, por E.Freund, H.J. Buxbaum y U. van der Valk, de la Universidad de Dortmund, en el artículo titulado "PC-Based hierarchical manufacturing cell control".

STRIPS (Standford Research Institute Problem Solver)

Diseñado en 1969, es un lenguaje formal inventado por Richard Fikes y Nils Nilsson para declarar instancias de problemas de planificación automatizada en inteligencia artificial. Una instancia Strips se compone de :

- un estado inicial

- la especificación de los estados meta, o sea, situaciones que el planificador tratará de alcanzar.

- un conjunto de acciones. Para cada acción se indica:

    - precondiciones (qué debe estar establecido antes de que la acción se realice)

    - postcondiciones (que se establece cuando se realiza la acción)

Fue publicado por sus autores en 1971, tanto en el volumen 2 de Artificial Intelligence, como en la Second International Joint Conference on Artificial Inteligence celebrada en Londres, en ambos casos bajo el título "STRIPS: a nes approach to the application of theorem proving to problem solving"

Es una evolución de QA4, un lenguaje de pregunta-respuesta desarrollado en SRI en 1967, y que es un cálculo procedural para razonamiento intuitivo. Algunas de sus características fueron incluídas posteriormente en HEOPS (Heuristic Oriented Language for Purposeful Systems Simulation), desarrollado en 1970 en la Federación Rusa, como un híbrido de STRIPS y Simula, propuesto como un lenguaje de resolución de problemas.

STRIPS fue utilizado en el robot Shakey, desarrollado entre 1966 y 1972 en el Centro de Inteligencia Artificial de SRI y financiado por DARPA, que fue el primer robot móvil de propósito general capaz de razonar sobre sus propias acciones

SWYM

Exensión de APT III de 1969 de W.J. Hansen que, a su vez, también extiende Automated Engineering Design (AED, un sistema de propósito general basado en Algol 60 diseñado en el MIT en 1963, y publicado en 1965).

T3

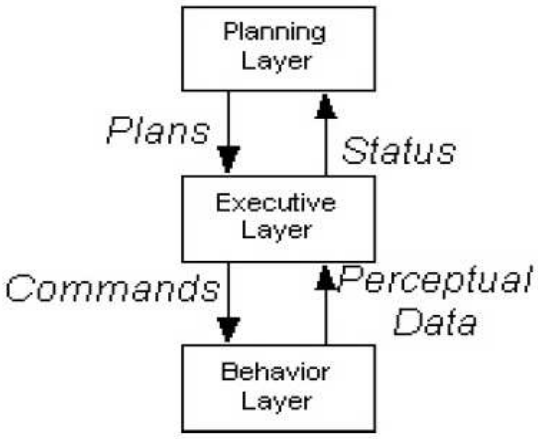
En 1974 Cincinnati Milacrom introdujo el robot T3 con control por computadora. En 1978 el robot se adaptó y programó para realizar operaciones de taladro y circulación de materiales en componentes de aviones, bajo el patrocinio de Air Force ICAM (Integrated Computer – Aided Manufacturing). El robot se programaba mediante un lenguaje gestual punto a punto, denominado también T3. El processador usado es el AMD

2900 (“bit slice”) y se dispone de un dispositivo de enseñanza denominado “teach pendant”. El lenguaje T3 permite el movimiento de un solo brazo, es transportable, y puede ser interpretado, compilado y ensamblado.

TDL (a Task Description Language for robot control)

Es un lenguaje que es una extensión de C++ que contiene medios para crear, sincronizar y manipular árboles de tareas. Presentado por R. Simmons y D. Apfelbaum en la Conference on Intelligent Robotics and Systems, celebrada en Vancouver en octubre de 1998.

Se basa en una arquitectura de control en tres niveles (Three-Tiered Control Architecture), común en los robots modernos de esa época. Hay una capa de comportamiento que controla los sensores y actuadores. Hay una capa ejecutiva que transforma los planes abstractos en comandos de bajo nivel, los ejecuta y maneja las excepciones. La capa de planificación especifica la misión a nivel abstracto.



La representación básica usada en TDL es un árbol de tareas y un programa de control opera creando y ejecutando árboles de tareas. Cada nodo en un árbol de tareas tiene una acción asociada. La acción puede realizar computaciones, añadir dinámicamente nodos hijos al árbol de tareas, o realizar alguna acción física en el mundo. Las acciones pueden incluir códgo condicional, iterativo o recursivo. Las acciones asociadas a los nodos pueden usar datos de los sensores para tomar decisiones sobre qué nodos añadir al árbol y cómo parametrizar sus acciones. Así, el mismo programa de control de tareas puede generar diferentes árboles de tareas de una ejecución a otra.

TDL contiene un compilador que transfroma código TDL en código C++ eficiente e independiente de la plataforma, que invoca una libraría Task Control Management (TCM) para manejar los aspectos de control de tareas del robot.

Los programadores también tienen disponible una herramienta de diseño visual para diseñar tareas TDL. El uso en tiempo real de TDL está limitado por el hecho de que requiere dos fases de compilación.

TEACH

Desarrollado en 1975 como parte del sistema PACS de Bendix Corporation. El sistema PACS se centraba en dos problemas importantes de la programación de robots: el problema de la ejecución paralela de diferentes tareas con múltiples dispositivos, incluyendo una variedad de sensores; y el problema de la definición de programas independientes del robot. En base a eso, TEACH introducía varias innovaciones clave:

- los programas se componen de secuencias de sentencias parcialmente ordenadas que pueden ser ejecutadas en paralelo o secuencialmente.

- el sistema permite un mapeo muy flexible entre los dispositivos lógicos especificados en el programa y los dispositivos físicos que los llevan a cabo.

- todos los movimientos se especifican en relación con marcos de referencia locales, lo que permite la reubicación simple de la secuencia de movimiento.

TPP Tecah Pendant Programming

Desarrollado en 1995 por Fanuc. Es interpretado y tiene instrucciones de movimiento, de registro, de entrada/salida, de salto, de espera,…

UNIAPT.

Implementación de APT III de 1969, desarrollada por la United Computing Corporation of California tanto en un DEC PDP8 como en un IBM 1130. Requería una memoria de solo 12K, en lugar de los 256K requeridos en los típicos sistemas APT. Pretendía tener capacidades para programas de 2, 3, 4, y 5 ejes, pero al no tener capacidades completas para 5 ejes se comercializó como sistema de contorneado de 3 ejes. Su principal ventaja fue el bajo coste de hardware y software respecto a otro sistemas.

URBISCRIPT

Lenguaje creado en 2003 por Jean-Christophe Baillie en el Laboratorio de Robótica Cognitiva en el ENSTA (École nationale supérieure de techniques avancées) del Paris Tech (Instituto Tecnológico de París). Se desarrolló en la industria a través de la empresa Gostai creada en 2006, pero sigue siendo un proyecto de código abierto. Puede describirse mejor como un lenguaje script de orquestación: se puede usar para unir componentes C++ en un comportamiento funcional. Como lenguaje de orquestación proporciona abstracciones útiles para el programador, teniendo paralelismo y programación basada en eventos como parte de la semántica del lenguaje. Otras características del lenguaje son: programación basada en prototipado, sintaxis tipo C++, arquitectura de componentes basada en

Java y C++, arquitectura cliente-servidor, multiplataforma, etc.

El nombre le viene de URBI, un software multiplataforma de código abierto utilizado para desarrollar aplicaciones para robótica y sistemas complejos.

URScript

Un lenguaje de Universal Robots para programar sus robots. Como cualquier lenguaje, tiene variables, tipos, sentencias de control de flujo, funciones, etc, además de tener funciones y variables incorporadas para monitorizar y controlar la entrada/salida y los movimientos del robot. Los programas escritos en URScript se ejecutan en tiempo real en el URControl, el controlador del robot de bajo nivel.

VAL (Vicarm Assembly Language) (Versatile Assembly Language)

Originalmente desarrollado por Bruce Shimano de Vicarm en 1975, basándose en WAVE, usado para su manipulador de robot, creado por Victor Scheinman[2]. En esos momentos, VAL significaba Vicarm Assembly Language, o símplemente VicArm Language.

Esta firma fue comprada por Unimation en 1977 (Vicarm se convirtió en la División de la Costa Oeste de Unimation) y el lenguaje se desarrolló para los robots PUMA (Programmable Universal Machine for Assembly) por Scheinman y Brian Carlisle. Así, VAL fue lanzado en 1979 como el primer lenguaje de robot disponible comercialmente. Fue escrito en ensamblador, y es un lenguaje interpretado no transportable. El significado de las siglas cambió, pasando a ser Versatile Assembly Language. En 1982 tuvo una revisión completa, formando el VAL-II, publicado en 1984.

VAL fue una extensión del BASIC para correr en microcomputadores DEC LSI-11. Fue usado para controlar Motorolas 6502, los microprocesadores que controlan los seis movimientos del brazo. VAL es capaz de manejar múltiples brazos. Las primeras versiones no aceptaban información de sensores, y no tenían posibilidades de herramientas intercambiables, ni interacción de visión, pero fueron incorporadas en versiones posteriores.

Alguna de las características de VAL/VAL-II son:

a) Definición de posiciones. Se pueden utilizar diferentes métodos como la posición actual del robot (HERE), a través del panel de aprendizaje ("touch pendant") con el comando (TEACH), o con el comando POINT.

   Las posiciones ('locations') que se introducen mediante el panel de aprendizaje (estilo lenguaje gestual), se pueden almacenar de dos formas:

   - en términos de las posiciones de articulaciones individuales del robot. En este caso las posiciones se llaman 'precision point'.

2 Según alguna fuente, el VAL original lo habrían desarollado para Vicarm en 1973 algunos estudiantes de la Universidad de Stanford.

- en términos de coordenadas cartesianas (x, y, z) y ángulos de orientación de la herramienta del robot respecto a un marco de referencia fijo en la base del robot. Estas posiciones son llamadas 'transformations' y son unan forma más intuitiva para representar posiciones que los 'precision point'.

  VAL también permite 'transformations' compuestas, que definen una posición relativa a otras posiciones y son escritas como cadenas de nombres de 'transformations' separadas por comas.

b) Edición y control de programas. Para la creación y edición de programas se utilizan diferentes comandos como EDIT <nombre> para empezar la edición de un programa con cierto nombre; EXIT salde del modo edición al modo monitor; SPEED indica la velocidad del manipulador, en una escala de 0,39 hasta 12800, donde 100 es la velocidad "normal"; EXECUTE <nombre>,n ejecuta un cierto programa n veces. VAL-II posee otros comandos como STORE, COPY, LOAD, FLIST, RENAME, DELETE, etc.

c) Comandos de movimiento. VAL usa dos métodos diferentes de controlar la tayectoria del robot desde una posición a otra:

- interpolando entre la posición inicial y final de cada articulación, produciendo una complicada curva de la herramienta en el espacio ('joint interpolated motion')
- moviendo el robot a través de un camino en línea recta ('straight line motion')

Por eso, hay pares de comandos para indicar el tipo de movimiento como DEPART/DEPARTS, MOVE/MOVES, APPRO/APPROS. La 'S' indica movimiento straight-line ("en línea recta"). MOVE/MOVES mueve la herramienta hasta un cierto punto, mientras que APPRO/APPROS y DEPART/DEPARTS se acerca o se aleja de una posición con una cierta distancia en el eje Z. Otros comandos son: SPEED, DRIVE y ALIGN. El comando DRIVE permite mover una única articulación (ej: DRIVE 5, 55, 60 mueve la articulación 5 en 55º en dirección positiva al 60% de la velocidad de monitor).

d) Control de la mano. La pinza de la mano se controla con los comandos OPEN/OPENI, CLOSE/CLOSEI y GRASP. La letra I en los comandos Open y Close significan que la apertura o cierre de la pinza debe ser inmediato, si no hay I, el cierre o apertura se produce cuando finaliza el movimiento del brazo.

e) Comandos de configuración, control e interbloqueo. Los comandos de configuración sirven para asegurar que el manipulador está espacialmente configurado para abordar su tarea como la muñeca, codo y hombro del brazo humano. Los comandos RIGHTY y LEFTY significan que el codo apunta al lado derecho o al izquierdo. Otros comandos son ABOVE, BELOW, etc.

Los comandos de interbloqueo son utilizados para comunicar con las señales de entrada/salida. RESET apaga todas las señales de salida, y es utilizado típicamente en la inicialización. El comando SIGNAL enciende o apaga señales de salida. El comando REACT interrumpe el curso normal del programa debido a la señal de un sensor externo (p.ej REACT Var2, Subr5 causa la monitorización permanente de la señal binaria externa especificada como variable Var2, y cuando ocurre esa señal, se llama a la subrutina Subr5). Otros comandos de entrada/salida son PROMPT y TYPE. También hay instrucciones de control del programa como IF...THEN...ELSE, WHILE...DO, DO...UNTIL, etc.

En sus inicios VAL fue débil en sus capacidades de procesamiento de datos, por lo que VAL-II ofrece constructores estructurados, funciones aritméticas, modificaciones de trayecto externas y soporte a las comunicaciones. Una tarea de control del proceso se ejecuta concurrentemente en 'background' durante la ejecución del movimiento.

VML (Virtual Machine Language)

Desarrollado en el Politécnico de Milán (Italia) en 1980 en cooperación con el CNR Ladseb de Padova (Italia). Estaba destinado a ser un lenguaje intermedio entre los sistemas de inteligencia artificial y el robot. Fue utilizado para transformar puntos en el espacio cartesiano a puntos en el espacio articular (de nivel de manipulador a nivel servo). También maneja definición de tareas y sincronización

V+

Desarrollado en 1989 por Adept Technology, como una ampliación de VAL-II, por lo que también ha sido denominado VAL-III. Se ha utilizado en robots de Adept y también en algunos de Stäubli.

Los programas se desarrollan en un PC y luego se envían al controlador del robot. Ofrece más adaptabilidad, fiabilidad y transportabilidad que sus precedentes. Es interpretado, estructurado y multitarea.

WAVE

Quizá el primer verdadero lenguaje de programación de robots, desarrollado en 1970 en la Universidad de Stanford[3] (Artificial Intelligence Laboratory), como un lenguaje experimental de programación de robots. Corría en un computador DEC PDP-8. Como este ordenador era lento y los algoritmos complejos, no se desarrolló el control en tiempo real.

WAVE fue pionero en el concepto de representar una posición a través de coordenadas cartesianas y tres ángulos de Euler, así como la coordinación

3 Según fuentes lo asocian al Artificial Intelligence Laboratory o al Stanford Resarch Institute. Las fechas también varían, pues alguna fuente lo data en 1973.

del movimiento de varios ejes. El lenguaje requiere movimientos preplanificados y solo se permiten pequeñas desviaciones durante el tiempo de ejecución. Puede aceptar datos de sensores de contacto para parar el movimiento. También puede interactuar con un sistema de visión para mostrar la coordinación ojo-mano.

WAVE sirvió como una guía para posteriores lenguajes, como AL.

XABSL (Extensible Agent Behavior Specification Language)

Es un enfoque pragmático para la ingeniería del comportamiento de agentes autónomos en entornos complejos y dinámicos. Se basa en jerarquías de máquinas de estados finitos para la selección de acciones, dando soporte al diseño de procesos de decisión deliberativos a largo plazo, así como comportamientos reactivos a corto plazo. Se ha aplicado satisfactoriamente en varias plataformas robóticas, destacando los robots futbolistas de RoboCup. De hecho, la primera versión fue desarrollada en 2001 por el GermanTeam, un conjunto de investigadores alemanes que competían en la RoboCup Four-Legged League, una liga de futbol con robots de 4 patas. Lo que fue narrado por M. Lötzsch, J. Bach, H.D. Burkhard, y M. Jüngel en 2004 en la publicación "Designing agent behavior with the extensible agent behavior specification language XABSL", en "RoboCup 2003: Robot Soccer World Cup VII".

Una posterior publicación, más formal, fue la de Marint Loetzsch, Max Risler, Matthias Jungel en la International Conference on Intelligent Robots and Systems de IEEE en 2006, bajo el título "XABSL, a pragmatic approach to behavior engineering"

XPROBE (Experimental System for Programming Robots by Example)

Es un sistema de programación de robots mediante ejemplo, de IBM, presentado en 1981 por Philip D. Summers y David G. Grossman en el Research Report IBM RC 9082 bajo el título "XPROBE: An experimental system for programming robots by example)

Xprobe extiende el método de programación de robots de "enseñanza mediante demostración" a enseñanza de estrategias de sensores. El sistemas Xprobe tiene un diálogo con el usuario durante el cual éste tiene que guiar al robot a través de un ejemplo de la tarea deseada. Luego, Xprobe escribe automáticamente un programa AML que incluye la toma de decisiones sensoriales como una generalización del ejemplo enseñado.

YALTA (Yet Another Language for Telerobotics Application)

Lenguaje para telerobótica presentado por J.C. Paoletti y J. Marce en el First Internationsl Symposium on Measurement and Controlling Robotics, realizado por la NASA en Texas en el año 1990, mediante la ponencia "A

monitoring language for telerobotics applications". Un año después Paoletti presentó su tesis para el doctorado en Informática, bajo la dirección de Marce, titulada "Conception d'un language de controle d'excecution de plans d'actions pour la telerobotique" en la que definía un sistema sobre el lengaje Yalta.

Es un lenguaje imperativo y gráfico que permite diseñar una misión con la ayuda de primitivas gráficas. La ejecución es interpretada, dando la posibilidad al operador de intervenir en línea sobre la estructura de la misión.

ZDRL (Zhe Da Robot Language)

Lenguaje de programación de robot orientado a movimiento, publicado en la Conferencia Internacional de IEEE de 1988 sobre "Systems, Man and Cybernetics", celebrada en Beijing, China, por Baokang Chen, de la Universidad de Zhejiang, y Vuliang Xu.

Era un lenguaje interpretado provisto de un entorno de programación "user-friendly", con edición, depuración y ejecución de programas. Tiene 32 comandos de sistema y 37 instrucciones de programa. Los comandos se usan para preparar el sistema para la ejecución de programas. Como ventajas sobre otros lenguajes de la época se destacaba sus más bajos requierimientos de computador, las abundantes estructuras de control del lenguaje, y las facilidades de depuración. Estaba implementado sobre un IBM-PC para controlar el robot Rhino XR-1.

Otros lenguajes

Otros lenguajes de programación de robots mencionados en alguna publicación, de los que no he encontrado información para documentarlos (en algún caso no he podido ni corroborar su existencia), son:

- CARL, de Lund University
- CASOR, de Stuttgart
- HAL, de M.Shahid Mujtaba y William D. Fisher
- EYE, de Fanuc
- FAPT, Fanuc APT, una versión de APT de Fanuc
- GPM, del Instituto Verkstad, Forskning, Suecia
- INDA, Philips
- LUNA, Languaje for Users Needs ans Aims, de Sony
- MAPS, de Turín (Italia) presentado por G.Bruno en la 6ª COMPSAC de IEEE, de noviembre de 1982 en Chicago.

- PARL-1, de Panasonic
- RCCL, Robot Control Command Language, diseñado en Canadá en 1984, mencionado ese año en un libro de V. Partington.
- ROBOTLAN, de Kawasaki Heavy Industries
- ROBOT-STAR, de REIS para conroladores G-70, LR-30, L4-70, V-15, H-15 y H-30
- ROLF, Robot Language Formula
- SAIL, Stanford Artificial Intelligence Language, de 1968
- SIL, de Silma Inc.
- SIRCH, de Nottingham University
- STAR, de la Universidad de Wisconsin
- TL, Toyota Language
- URI, de Rhode Island University

BIBLIOGRAFÍA

- ✔ ACL: Lenguaje de Control Avanzado. Versión 1.43, F.44
  Guía de Referencia para Controlador-A
  Eshed Robotec

- ✔ Advances in manufacturing technology II
  Proceedings of the Third National Conference on Production Research
  Edited by P F McGoldrick
  University of Nottingham, 1987
  https://books.google.es/books?id=CVPxBwAAQBAJ

- ✔ The AMPLE Project
  J. C. Boudreaux
  National Bureau of Standards. U.S. Department of Commerce. Marzo 1987

- ✔ The APT programming language for the numerical control of machine tools
  J. Vlietstra
  Philips Technical Review, volume 28, 1967

- ✔ The ARCL Robot Programming System
  Peter Ian Corke, Robin Kirkha
  CSIRO Division of Manufacturing Technology.
  International Conference of the Australian Robot Association and the International Federation of Robotics. Robots for Competitive Industries. Julio 1993

- ✔ AS Language Reference Manual
  Kawasaki Robot Controller D
  Kawasaki Heavy Industries, Ltd.

- ✔ The Behavior Language; User's Guide
  Rodney A. Brooks, abril 1990
  Massachusetts Institute of Technology, Artificial Intelligence Laboratory

- ✔ COLBERT: A language for reactive control in Saphira.
  Konolige, Kurt. (1997).
  SRI International.

- ✔ A comparative study of robot languages
  Susan Bonner, Kang G. Shin, 1982
  Center for Robotics and Integrated Manufacturing
  Robot systems division. College of Engineering.
  University of Michigan

✔ CAD/CAM Theory and Practice
Ibrahim Zeid, R Sivasubramanian
Mc Graw Hill, 1991
https://books.google.es/books?id=DJeRtGVAr6MC

✔ Computer Aided Design and Manufacturing
K. Lalit Narayan, K. Mallikarjuna Rao, M.M.M. Sarcar
Prentice-Hall of India Private Limited
Nueva Delhi, 2008

✔ Computer Aides Manufacturing
P.N. Rao, N.K. Tewari, T.K. Kundra
Indian Institute of Technology
Mc Graw Hill. Nueva Delhi, 1993
https://books.google.es/books?id=kgkEDj5TRh8C

✔ Computer Assisted Part-Programming Facilities for Numerically Controlled Machine Tools. Erik V. Klaassen. Mc Master University, noviembre 1971

✔ Computer Integrated Manufacturing. Current Status and Challenges
NAT ASI Series.
I. Burhan Turksen, 1987
https://books.google.es/books?id=58ioCAAAQBAJ

✔ Computer Graphics in the Dynamic Analisys of Mechanical Networks
Milton A. Chace, Michael E. Korybalski
University of Michigan
CONCOMP : Research in Conversational Use of Computers.
Technical Report 26. Febrero de 1970,

✔ Computer Numercial Control. Concepts & Programming
Warren S. Seames
Delmar, division of Thomson Learning Inc, 2002
https://books.google.es/books?id=6M1E8ydzAgkC

✔ Conversational Automatic Programming. Function II for Lathe Operator's Manual

✔ CRAM – a Cognitive Robot Abstract Machine for everyday manipulation in human environments
Michael Beetz, Lorenz Mösenlechner, Moritz Tenorth
Departmen of Informatics, Technische Universität München
2010 IEEE/RSJ International Conference on Intelligent Robots and Systems

✔ The Design of a Mechanical Assembly System
Tomás Lozano Pérez. MIT.
Diciembre 1976

✔ Development of an interactive graphical simulator for the IBM 7545 robot.
Velluva P. Mohandas
Blacksburg, Virgina, junio 1987

✔ Developments in Computer-Integrated Manufacturing
D.Kochan, 1986
https://books.google.es/books?id=iwarCAAAQBAJ

✔ Diseño Software de una Arquitectura de Control de Robots Autónomos Inteligentes. Aplicación a un Robot Social.
Rafael Rivas Estrada, 2010. Tesis doctoral.
Departamento de Ingeniería de Sistemas y Automática
Universidad Carlos III de Madrid

✔ Encyclopedia of computer science and technology, volume 21
Universtiy of Pittsburgh (Pennsylvania), 1990
Allen Kents, James G. Williams
https://books.google.es/books?id=TZrxHta_0qYC

✔ ESL: A Language for Supporting Robust Plan Execution in Embedded Autonomous Agents
Erann Gat. Jet Propulsion Laboratory. California Institute of Technology.
IEEE Aerospace Conference, Snowmass at Aspen, CO, USA, 1997, pp. 319-324 vol.1

✔ A feature based approach to the integration of design, manufacturing and process planning.
Richard Mark Schulte.
Iowa State University, 1990.

✔ FROB: A tranformational approach to the design of robot software
Gregory D. Hager, John Peterson
International Symposium on Robotics Research, 1999

✔ FROB: Functional Reactive Programmaing Applid to Robotics
Gregroy Hager, John Peterson, Henrik Nilsson

✔ Functional Programming of Behaviour-Based Systems
Ian Douglas Horswil
Computational Intelligence in Robotics and Automation (CIRA), IEEE 1999

✔ Fundamentals of robotics
David D. Ardayfio
Maercel Dekker, Inc.1987
https://books.google.es/books?id=zIPr3b_SK9QC

✔ Fundamentals of Robotics Engineering
Harry H. Poole, 1989
https://books.google.es/books?id=AmQPCQAAQBAJ

✔ Generic educational robot control System: GERCS
Robert E. Fletcher, 1991
Lehigh University

✔ GOLOG: a logic programming language for dynamic domains
Hector J. Levesque, Raymond Reiter, Yves Lespérance, Fangzhen Lin y Richard B. Scherl
The Journal of Logic Programming, Elsevier Science, Inc., 1997

✔ HARL-U1, Ver. 2. Instruction Manual
Hirata Corporation, 1992.

✔ High-level Object-Oriented Program Language for Mobile Microrobot Control
Farenc Vajda, Tamás Urbancsek, octubre 2003

✔ High-Performance Visual Closed-Loop Robot Control
Peter Ian Corke, 1994
Department of Mechanical and Manufacturing Engineering.
University of Melbourne

✔ Higher order languages for robots
James R. Blaha, John P. Lamoureux, Keith E. McKee,
Manufacturing Technology Information Analysis Center (Chicago, USA), 1986

✔ History of Programming Languages
Richard L. Wexeblat
Academic Press, 1981

✔ HrBasic Developing Environment, Operation Manual, ver. 2.10
Hirata Corporation. Japón, 1999-2003

✔ Human-machine Interfaces in Industrial Robotics
H. McIlvaine Parsons, Anne S. Mavor.
Essex Corporation, Septiembre 1988.
U.S. Army Human Engineering Laboratory, Maryland.

✔ Human-robot Interaction
Mansour Rahimi, Waldemar Karwowski
Taylor&Francis, 1992
https://books.google.es/books?id=DroEY14-5lIC

✔ Humanoid Motion Description Language
Ben Choi, Yanbing Chen

✔ Humanoid Robot Interactions by Motion Description Language and Dialogue
Kankana Shukla y Ben Choi
International Journal of Electrical Energy, Vol. 1, No. 2, June 2013

✔ *Humanoid Robotic Language and Virtual Reality Simulation*
Ben Choi, 2017

✔ Industrial Automation and Robotics
A.K.Gupta, S.K.Arora, 2013 (primera edición, 2007)
https://books.google.es/books?id=Y7rgCP7iC18C

✔ INFORM manual. NX100.
Yaskawa Motoman Robotics

✔ INFORM II, User's Manual
Yaskawa Motoman, 2007

✔ Integration of programming and learning in a control language for autonomous robots perfroming everyday activities
Alexandra Kirsch. 2007.

✔ Intermediate Language for Mobile Robots
A link between the high-level planner and low-leve services in robots
Ilkka Kauppi, 2003
VTT Industrial Systems
Helsinki University of Technology, Espoo, Finlandia

✔ Introducción a la programación MELFA BASIC IV
Mitsubishi Electric

✔ Introducción a RAPID, manual del operador. RobotWare 5.0
ABB Robotics Products. Suecia, 2007.

✔ Instructions for INFORM language. DX100 Options.
Yaskawa Motoman Robotics

✔ Investigations in Computer-Aided Design fro Numerically Controlled Production
D.T. Ross, J.E. Ward. MIT, 1967

✔ Kawasaki Robot AS Language Referencia Manual
Kawasaki Heavy Industries, Ltd., 2007

✔ KR C1 / KR C2 / KR C3 Reference Guida, release 4.1
Kuka Roboter

✔ LAMA: A language for automatic Mechanical Assembly
Tomas Lozano-Pérez, Patrick H. Winston
Proceedings of the 5th international joint conference on Artificial intelligence - Vol 2. Enero 1977

✔ Learning ROS for Robotics Programming
A practical, instructive, and comprehensive guide to introduce yourself to ROS, the top-notch, leading robotics framework
Aaron Martinez, Enrique Fernández
Packt Publishing, 2013
https://books.google.es/books?id=2ZL9AAAAQBAJ

✔ Lenguaje y programación de robots. Monografía presentada al Curso de Metodología del Trabajo Universitario.
Vera Zurita, Victor Jhonatan.
Escuela Profesional de Ingeniería de Sistemas. Universidad de Lambayeque. Chiclayo, agosto de 2010.
http://lenguaje-programacion-robots.blogspot.com.es/

✔ Lenguajes de programación de los robots
Samuel Candelas Rodríguez
UNAM Campus "Aragón", Ingeniería en Computación
http://www.monografias.com/trabajos3/progrob/progrob.shtml

✔ El lenguaje V+. Miniproyecto de robótica.
Jaume Yebra Pérez, Núria Lagos Fernández, diciembre 2002
Escuela Universitaria Politécnica de Vilanova i la Geltrú

✔ El libro blanco de la robótica. De la investigación al desarrollo tecnológico y futuras aplicaciones.
Comité Español de Automática (CEA), Grupo Temático de Robótica (GTRob), con subvención del Ministerio de Educación y Ciencia. 2008.

✔ Live Robot Programming: the language, its implementation, and robot API independece
Miguel Campusano, Johan Fabry, marzo 2'16

✔ A low level robot interface: The high speed host interface
Marilyn Mashman
National Bureau of Standards. U.S. Departement of Commerce. Junio 1986

✔ Machine tool design and numerical control
N K Mehta
https://books.google.es/books?id=_wWET38FZqsC

✔ Making Robot Learning Controllable: a case study in robot navigation
Alexandra Kirsch, Michael Schweitzer, Michael Beetz. 2018.

✔ Manufacturing, Automation Systema and CIM Factories
K.Asai y S.Takashima
Chapman&Hall, 1994
https://books.google.es/books?id=B-iWG5J14LIC

✔ Micro-Planner reference manual.
Gerald Jay Sussman y Terry Winograd.
MIT Artificial Intelligence, memo n.º 203, julio 1970.

✔ Modelado, programación y simulación del robot IRB 120 de ABB con RobotStudio
Proyecto fin de máster en Automática, Robótica y Telemática
Beatriz Matos Agudo. Universidad de Sevilla, 2017

✔ Numerical Control Programming Languages for Lathes
Gerald C. Dunsford
McMaster University. Septiembre 1972.

✔ Open-Source Robotics and Process Control Cookbook. Designing and Building Robust, Dependable Real-Time Systems
Lewin A.R.W. Edwards, 2005

✔ On reverse-engineering the KUKA Robot Language
Henrik Mühe, Andreas Angerer, Alwin Hoffmann y Wofgang Reig, 2010

✔ An Overview of off-line robot programming systems.
R. Hocken y G. Morris.
Annals of the CIRP vol. 35. Año 1986.

✔ Part and Assembly Description Languages II
W.B Fisher, A. A. G. Requicha, N. M. Samuel, H. B. Voelcker
Production Automation Project. Technical Memorandum 20b
University of Rochester, 1978

✔ PC-Based hierarchical manufacturing cell control
E.Freund, H.J. Buxbaum y U. van der Valk. Universidad de Dortmund.
IFAC (International Federation of Automatic Control) de Viena (Austria) en 1992

✔ Perspective on Standarization in Mobile Robot Programming: the Canegie Mellon Navigation (CARMEN) Toolkit.
Michael Montemerlo, Nicholas Roy, Sebastian Thrun, 2003.

✔ Plataforma de comunicación entre Live Robot Programming y el robot Ar.Drone 2.0
Carolina Massiel Hernández Philips
Universidad de Chile, 2016

✔ Pointy. A philosophy in robot programmming
G. Gini, M. Gini
IFAC Proceedings Volumes, volume 12, issue 10, pags. 173-181.
Septiembre 1979

✔ Programacion de robots industriales. Control remoto del Robot ASEA IRB2000
Hilario López García, Rafael González Librán
Universidad de Oviedo, Servicio de Publicaciones
https://books.google.es/books?id=py8Raj7FxgMC

✔ Programming Robots with ROS
A Practical introduction to the robot operating system
Morgan Quigley, Brian Gerkey, William D. Smart, 2015
https://books.google.es/books?id=G3v5CgAAQBAJ

✔ RAPID Instructions, Functions and Data Types. Technical reference manual.
ABB Robotics Products, 2004-2010.

✔ RAPID Reference On-line Manual
ABB Flexible Automation AB

✔ ROBOBASIC Command Instruction Manual v.2.10
www.hitecrobotics.com

✔ Robot Fanuc LR Mate 200iC ROBOGUIDE. Langage KAREL, guide de l'étudiant.

✔ Robot language from the standpoint of FA system development – An outline of FA-Basic
Shunji Mohri, Kenji Takeda, Seiji Hata Kichie Matsuzaki y Yoshihiro Hyodo
Production Engineering Research Laboratory, Hitachi Ltd. Japón, 1985

✔ Robot Languages in the Eighties
Guiseppina Gini y Maria Gini, 1985.

✔ Robot Programming
Tomás Lozano Pérez
Proceedings of the IEEE vol 71, julio 1983

✔ Robot Programming Languages—A State of the Art Survey
K. Srihari, M. P. Deisenroth
http://link.springer.com/chapter/10.1007%2F978-3-642-73890-6_76

✔ Robot Programming system based on L-IRL programming language
Maja Lutovac, Goran Ferenc, Vladimir Kvrgic, Jelena Vidakovic, Zoran Dimic
Lola Institute, Belgrado (Serbia)
Acta Technica Corviniensis. Bulletin of Engieneering tome V, fascicule 2. Abril-junio 2012.

- ✔ Robot Technology, volume 5
  Logic and programming
  Michel Parent and Claude Laurgeau, 1983
  https://books.google.es/books?id=wbreBwAAQBAJ

- ✔ Robótica.
  John J. Craig
  https://books.google.es/books?id=hRzOp_qdxG8C

- ✔ Robótica: Control, Detección, Visión e Inteligencia
  K.S. FU, R.C González, C.S.G. LEE
  McGraw Hill

- ✔ Robótica. Manipuladores y robots móviles.
  Aníbal Ollero Baturone, 2001.
  Marcombo Boixareu Editores
  https://books.google.es/books?id=TtMfuy6FNCcC

- ✔ Robótica Practica Tecnología y Aplicaciones
  José Ma. Angulo
  Ed. Paraninfo, 1999

- ✔ Robotics
  Appuu Kuttan K.K.
  IK Intenational Publishing House Pvt. Ltd.
  Nueva Delhi (India), 2007
  https://books.google.es/books?id=5N7NY_YVufkC

- ✔ The Robotics Primer
  Maja J. Mataric
  MIT, 2007

- ✔ Robotics Technology and Flexible Automation
  S.R. Deb, Sankha Deb
  McGraw Hill, 2010
  https://books.google.es/books?id=0gAiBAAAQBAJ

- ✔ Robot Learning Language (RoLL). Reference Manual
  Alexandra Kirsch. 2011.

- ✔ Robots, Communication, and Language: an overview of the Lingodroid Project
  Ruth Schulz, Arran Glover, Gordon Wyeth y Janet Wiles, 2010
  Australasian Conference on Robotics and Automation (ACRA), Brisbane (Australia)

- ✔ Scorbase para Windows, nivel 1, nivel 3 y Scrobase Pro. ScorBot-ER 4 PC. Manual de usuario. Eshed Robotec, 1999.

- ✔ Seminario de Robótica 2004. Universidad de Atacama
  http://www.industriaynegocios.cl/Academicos/AlexanderBorger/Docts%20Docencia/Seminario%20de%20Aut/trabajos/2004/Rob%C3%B3tica/seminario%202004%20robotica/Seminario_Robotica/Documentos/PROGRAMACI%C3%93N%20DE%20ROBOTS.htm

- ✔ Simulation, Modeling, and Programming for Autonomous Robots
  Second International Conference, SIMPAR 2010
  Darmstadt, Germany, noviembre de 2010
  https://books.google.es/books?id=8USi-anN1_MC

- ✔ SPEL+ Languaje Reference, rev.5. Epson RC+ 7.0, ver.7.1
  Seiko Epson Corporation, 2012-2015

- ✔ SPLAT A Simple Provisional Language for Actions and Tasks
  http://www.cs.utexas.edu/users/qr/robotics/splat/docs/
  UT Intelligent Robotics Research. Qualitative Reasoning Research Group.
  University of Texas at Austin. Computer Science

- ✔ Success Story. Looking back at ABB's contribution to industrial robotics
  David Marshall, Christina Bredin

- ✔ Symbolic Computation. Computer-aided Design and Manufacturing. Methods and tools
  U.Rembold, R.Dillmann
  Springer-Verlag Berlin, 1986
  https://books.google.es/books?id=WcSqCAAAQBAJ

- ✔ A structured approach to robot programming and teaching
  Kunikatsu Takase, Richard P. Paul, E.J. Serg
  IEEE, 1979

- ✔ Technical Report TR83730
  ISO Technical Comittee, 1988, subcomittee 2

- ✔ Theory of Automatic Robot Assembly and Programming
  Bartholomew O. Nnaji
  Springer Science+Business Media Dordrecht, 1993

- ✔ Towards advanced robot programming. Journal of the robotics society in Japan
  Inoue, H., 1984

- ✔ Towards an Adaptable Robot Language
  Holly Yanco
  AAAI Technical Report FS-92-02, 1992.

- ✔ Towards high-peformance robot plans with grounded action models: integrating learning mechanisms into robot control languages
  Alexandra Kirsch. Enero 2005.

- ✔ TS3000 series Robot Controller. Robot Language Manual.
  Toshiba Machine Co., Ltd., 2009

- ✔ VAL2 Guía. Manual nº1.
  Stäubli SA, 1992

- ✔ VAL3 Referencie Manual, vesion 5.3
  Stäubli Faverges, 2006

- ✔ What every engineer should know about
  Computer-Aided Design and Computer-Aided Manufacturing
  The CAD/CAM Revolution
  John K. Krouse
  Marcel Dekker, INC., 1982
  https://books.google.es/books?id=EyvUwkF96PAC

ANEXO 1.- EJEMPLOS DE PROGRAMAS PARA ROBOTS INDUSTRIALES

Ejemplo 1

Lenguaje: **FUNKY**
Tarea: Paletizado

```
ABSOLUTE  <starting-point>
STORE 1
ABSOLUTE  <conveyor-approach-point>
STORE 2
ABSOLUTE  <convert-pickup-point>
STORE 3
CENTER 8
RECALL 2
ABSOLUTE  <first-pallet-position>
STORE 4
ABSOLUTE  <first-drop-position>
RELEASE 8
RECALL 4
HALT 0
RECALL 2
RECALL 3
CENTER 8
RECALL 2
ABSOLUTE  <second-pallet-position>
STORE 4
ABSOLUTE  <second-drop-position>
RELEASE 8
RECALL 4
HALT 0
RECALL 2

{…}

HALT 0
RECALL 2
RECALL 3
CENTER 8
RECALL 2
ABSOLUTE <ninth-pallet-position>
STORE 4
ABSOLUTE  <ninth-drop-position>
RELEASE 8
RECALL 4
HALT 1
```

Explicación

Los parámetros <nombre> representan posiciones previamente definidas mediante métodos de aprendizaje. El comando ABSOLUTE referencia una posición predefinida. El comando STORE almacena la posición de un comando previo en un stack del 1 al 9. El comando RECALL recoge una posición del stack y mueve el robot a esa posición. Los comandos CENTER y RELEASE posicionan y activan la pinza para coger o dejar un objeto. El comando CENTER usa touch-sensors para centrar la pinza sobre el objeto. En este ejemplo el comando HALT se ha usado para simular comandos de Start y Stop del transportador.

Ejemplo 2

Lenguaje: **VAL**
Tarea: paletizado

```
1.          SETI PX = 1
2.          SETI PY = 1
3.    10    GOSUB 100
4.          IF PX = 3 THEN 20
5.          SHIFT PALLET BY 100.0, 0, 0
6.          GOTO 10
7.    20    IF PY = 3 THEN 40
8.          SETI PX = 1
9.          SETI PY = PY + 1
10.         SHIFT PALLET BY -300.0, 100.0, 0
11.         GOTO 10
12.   100   APPRO CON,50
13.         WAIT CONRDY
14.         MOVES CON
15.         GRASP 25
16.         DEPART 50
17.         MOVE PALLET:APP
18.         MOVES PALLET
19.         OPENI
20.         DEPART 50
21.         SIGNAL GOCON
22.         SETI PX = PX + 1
23.         RETURN
24    40    STOP
```

Explicación

Las líneas 1 y 2 inicializan variables enteras que se usan para contar las partes que han sido cargadas en el pallet, en ambas direcciones, x e y.

El GOSUB de la línea 3 llama a una subrutina que empieza en la etiqueta "100" (líneas 12 a 23). Esta subrutina descarga una pieza del transportador y la carga en el pallet.

El comando SHIFT PALLET no causa ningún movimiento físico del pallet, sino que redefine na coordenada llamada PALLET, de forma que la próxima

vez que el robot se mueva a esa posición PALLET, irá a una nueva posición. SHIFT implementa una traslación en los 3 ejes. En el caso 100,0,0 significa una traslación de 100mm en el eje X, y 0 en los ejes Y y Z.

Dentro de la subrutina, el robot primero se aproxima (APPRO) a la posición CON, es este caso a 50mm de distancia en el eje Z.

El WAIT de la línea 13 hace que el robot espere una señal de una fuente externa, en este caso un indicador de que el transportador está preparado. En ese momento, el robot se mueve a la posición CON (MOVES significa movimiento en línea recta -S de straight-), y cierra la pinza (comando GRASP). Si la pinza se cierra menos que la distancia mínima indicada en el parámetro (en este caso 25mm), ocurre un error y el programa para.

Con el comando DEPART se separa una distancia en el eje Z, en el ejemplo 50mm.

El comando de la línea 17 (MOVE PALLET:APP) es equivalente a "APPRO PALLET, 50" pero escrito de una forma alternativa. MOVE indica movimiento interpolado (a diferencia de MOVES que es en línea recta) a un cierto punto. Los dos puntos indican que el argumento es una transformación que es el producto de dos transformaciones, la del Pallet y la del desplazamiento de 50mm en el eje Z.

Por último, un programa VAL puede enviar señales de salida a dispositivos externos, como la línea 21 (SIGNAL GOCON), con la que inicia movimiento en el transportador.

Ejemplo 3

Lenguaje: VAL-II
Tarea: paletizado

```
1. FOR PX = 1 TO 3
2.     CALL load.row( 3 )
3.     SHIFT pallet BY 100.0, 0, 0
4. END
5.
6.
7. SUBROUTINE load.Row( Length)
8.     FOR PY = 1 to Length
9.             APPRO conveyor, 50
10.            WAIT conveyor.ready 50
11.            MOVES conveyor
12.            GRASP 25
13.            DEPART 50
14.            MOVE pallet:App
15.            MOVES pallet
16.            OPENI
```

```
17.         SIGNAL go.conveyor
18.         SHIFT pallet BY 0, 100, 0
19.   END
20.   SHIFT pallet BY 0, -300, 0
21. END SUBROUTINE
```

Explicación
Es el mismo caso que el ejemplo 2, pero con VAL-II que es una versión estructurada de VAL, con lo que se eliminan las sentencias GOTO, y se usan bucles y subrutinas con nombre y pase de parámetros.

Ejemplo 4

Lenguaje: **AUTOPASS**
Tarea: paletizado

```
WHILE emptypallette DO
      BEGIN
            WHILE holeinpalletfree DO
                  BEGIN
                        INSERT block IN holeinpallet;
                  END;
      END;
```

Explicación
En AUTOPASS no necesitamos definir puntos de acercamiento y recogida. La localización de los objetos y la trayectoria requerida para coger los objetos son derivados por el sistema. La posición del pallet así como el estado del palet (que huecos están libre y cuales ocupados) también es derivado por el sistema.

Ejemplo 5

Lenguaje: **ScorBase3** (de Eshed Robotec)
Tarea: Paletizado

```
1)
2)
3) CALL SUBROUTINE #2       Go pickup object from conveyor
4) GO POSITION 3 * FAST     Above Pallet's first position
5) GO POSITION 4 * SLOW     Put down point
6) OPEN GRIPPER
7) GO POSITION 3 * SLOW     Above Pallet's first position
8)
9) CALL SUBROUTINE #2       Go pickup object from conveyor
10) GO POSITION 5 * FAST    Above Pallet's Second position
11) GO POSITION 6 * SLOW    Put down point
12) OPEN GRIPPER
```

```
13) GO POSITION 5 * SLOW      Above Pallet's second position

60) CALL SUBROUTINE #2        Go pickup object from conveyor
61) GO POSITION 19 * FAST     Above Pallet's ninth position
62) GO POSITION 20 * SLOW     Put down point
63) OPEN GRIPPER
64) GO POSITION 20 * SLOW       Above Pallet's ninth position
65) JUMP TO 73

66) SET SUBROUTINE #2           Subroutine to Pickup object from conveyor
67) IF INPUT #5 OFF             Wait for part
    JUMP TO 67
68) GO POSITION 1 * FAST        Move to the approach point
69) OPEN GRIPPER
70) GO POSITION 2 * SLOW        Move slow to the grasp point
71) CLOSE GRIPPER
72) GO POSITION 1 * FAST        Lift object from conveyor
73) RETURN FROM SUBROUTINE  Return to caller
74) END
```

Explicación

Se supone que previamente están memorizadas las posiciones (con aprendizaje gestual), cada una con un número. En el ejemplo, las posiciones 1 y 2 son las de recoger (pickup) en el transportador (conveyor), una para acercarse y la otra la exacta, y luego otros 9 pares de posiciones (3 y 4, 5 y 6,..., 20 y 21) para las 9 posiciones de paletización (de cada par, la primera posición para acercarse aprisa, y la segunda la posición concreta a la que se va con velocidad lenta).

Se ha creado una subrutina con número 2, para coger una pieza de la cinta transportadora.

Nota: Los comentarios y las líneas en blanco no forman parte del programa real, pues no los permite Scorbase3.

Ejemplo 6

Lenguaje: **GERCS**

Tarea: Paletizado

```
1) #include <gercs.h>
2) #include <points.h>
3)
4) main()
5) {
6)       int row, col;
7)       Robot R;
8)
9)       R = Standard_Rhino;
11)
12)      for(col=l; col<3; col++)
```

```
13)     {
14)           for(row=l; row<=3; row++)
15)           {
16)                 Fetch From Conveyor (&R, Conveyor)
17)                 Place-In Pallet (&R, Pallet Point);
18)                 Shift(Pallet, 100.0, 0.0 ,-0.0);
19)                 Signal (Conveyor_Port, Go_Conveyor);
20)           }
21)           Shift (Pallet, -300.0, 100.0, 0.0);
22)     }
23)
24)
25)     Fetch From Conveyor ( Robot * R, Joint_Type Conveyor_Location)
26)     {
27)           Wait (Conveyor Port, Conveyor Ready);
28)           Appro(R, Conveyor Location, 50);
29)           Sp(R, LOW); -
30)           Move(R, Conveyor Location, 50);
31)            Sp(R, HIGH) -
32)           Grasp(R);
33)           Depart(R,50);
34)     }
35)
36)     Place In Pallet (Robot *R, Joint_Type Pallet_Point)
37)     {
38)           Sp(R, HIGH)
39)           Appro (R, Pallet Location, 50);
40)           Sp(R, LOW); -
41)           Move(R, Pallet Location, 50);
42)     }
```

Descripción

Con el Include de la línea 1 cargamos la definición de tipos de GERCS. Tiene diferentes declaraciones de tipos de datos, como el tipo Robot de la línea7 o el tipo Joint_Type de las líenas 25 y 36.

El Include de la línea 2 carga el fichero de definición de puntos, que es creado por el teach pendant y contiene definiciones en C para los puntos definidos, en este ejemplo Conveyor y Pallet.

Las líneas 7 y 9 definen el tipo de robot que va a ser controlado.

Los comandos Shift, Appro, Move de las líneas 21, y 27 a 33 son funciones de la librería GERCS. Hacen lo que su equivalente VAL. Shift altera el valor de las coordenadas de un punto; Appro mueve a una cierta distancia aobre un punto; Move mueve a un cierto punto; Sp establece una velocidad.

Ejemplo 7

Lenguaje: **ILMR**
Tarea: el robot trae dos cajas desde otra habitación

```
Init()
Turns = 0
Obsavoid(on)
While(Turns<2)
        MyDoor=sensor(scanner,door)
        Wait(MyDoor)
        Speed(0.3)
        Gotoxy(MyDoor)
        Ok=Turn(right)
        If(Ok)
                Speed(0.6)
                Followwall(middle)
                Door=sensor(scanner,door,left)
                If(Door)
                        Break(Followwall)
                        Record(ToDoor)
                        Speed(0.3)
                        Turn(Door)
                        Turn(left)
                        Goahead(3)
                        Over=Takebox
                        Wait(Over)
                        Record(close)
                        Speed(-0.3)
                        Followpath(ToDoor,backwards)
                        Turnok=Turn(d=-90,c=1)
                        Wait(Turnok)
                        Speed(0.6)
                        Followwall(middle)
                        Door=sensor(scanner,door,left,forever)
                        If(Door == MyDoor)
                                Break(sensor)
                                Speed(0.3)
                                Turn(Door) // or Turn(MyDoor)
                                GoReady=Goahead(3)
                                Wait(GoReady)
                                Speed(-0.3)
                                Turn(d=180,c=1)
                                Turns = Turns + 1
                        EndIf
                EndIf // Door
        EndIf // Ok
EndWhile
Battery=sensor(battery)
If(Battery<LowBattery)
```

```
            Priority=Highest
            Go_Charger
      EndIf
```

Explicación

- Init inicializa el robot y todos sus subsitemas y sensores

- Turns es una variable. Incialmente se le establece valor cero.

- ObsAvoid(on) se activa el comportamiento del robot de evitar obstáculos

- While estable un bucle de repetición con una condición

- MyDoor=sensor(scanner,door) significa que el robot empieza a buscar una puerta usando un sensor (escáner). Tanto el escáner como la puerta están definidos en el modelo del robot. En este caso "scanner" es un escáner láser, y "door" significa una abertura entre 0.8 y 1.2 metros. Cuando se encuentra la puerta, sus parámetros se almacenan en la variable MyDoor.

- Wait(MyDoor) espera hasta que se encuentra la puerta.

- Speed(0.3) fija la velocidad del robot a 0,3 m/s.

- Gotoxy(MyDoor) mueve el robot a la puerta encontrada

- Ok=Turn(right) el robot se mueve a la derecha siguiendo un segmento de arco de un círculo por defecto, predeterminado por la curvatura máxima del robot. Se usa el escáner para comprobar que el giro es posible. Si no es posible, se determina un nuevo camino libre según la lectura del escáner. Cuando se ha realizado el giro, se establece a 1 la variable Ok.

- If(Ok) empieza un bloque condicional que finaliza con el correspondiente Endif. La condición va entre paréntesis.

- Followwall(middle) mueve el robot por el pasillo, por el medio.

- Door=sensor(scanner,door,left) empieza a buscar una puerta por el lado izquierdo del robot. Este comando está ejecutándose al mismo tiempo que el followwall anterior. Cuando encuentra la puerta, establece Door a 1

- If(Door) comienza otro bloque condicional cuando la variable Door es 1.

- Break(Followwall) termina el comando Followwall. Con las 4 últimas sentencias, camina por el pasillo mirando a la izquierda hasta encontrar una puerta, momento en el que deja de caminar.

- Rercord(ToDoor) Empieza a grabar la posición del robot en el fichero ToDoor

- Turn(Door) mueve el robot al medio de la puerta abierta

- Turn(left) como el anterior Turn(right)

- Goahead(3) mueve el robot 3 metros más adelante

- Over=Takebox no es un comando elemental, sino una tarea aprendida previamente. Cuando la subtarea "Takebox" termina, la variable Over se pone a 1.

- Wait(Over) espera hasta que Over se establece a 1, o sea, hasta que se

termina la tarea Takebox

- Record(close) para la grabación de posiciones

- Followpath(ToDoor,backwards) mueve el robot atrás por el camino grabado en ToDoor

- Turnok=Turn(d=-90,c=1) gira el robot a la derecha 90 grados usando un valor 1.0 para la curvatura. Al final la variable Turnok se establece a 1.

- Door=sensor(scanner,door,left,forever) es lo mismo que antes, pero "forever" indica que el comando no termine cuando se encuentre una puerta. Cuando se detecta una puerta, el comando sigue buscando otra puerta, y así para siempre

- If(Door==MyDoor) compara si la puerta encontrada es la misma que almacenó en MyDoor (por la que entró).

- Break(sensor) termina el comando sensor. Con estos tres últimos comandos, el robot va buscando puertas hasta que encuentra la puerta por la que entró. Entonces deja de buscar.

- GoReady=Goahead(3) mueve el robot 3 metros hacia adelante

- Wait(GoReady) espera hasta que haya avanzado esos 3 metros

- Turn(d=180,c=1) gira el robot 180 grados.

- Turns=turns+1 incrementa la variable Turns

- Battery=sensor(battery) lee el voltaje de la batería y lo almacena en la variable Batery

- if(Battery<LowBattery) compara el voltaje actual con un valor predefinido (LowBattery) en el modelo del robot.

- Priortiy=Highest significa que el siguiente comando se ejecuta inmediatamente y cualquier otro comando está prohibido

- Go_Charger es una habilidad aprendida, que conduce al robot a la estación de carga.

Ejemplo 8

Lenguaje: **MML**

```
user()
{
POSTURE p,a,b;
int i;
        def_posture(200.0,0.0,0.0,&a);
        def_posture(200.0,0.0,0.0,&a);
        set_rob(def_posture(-100.0,-200.0,0.0,&p));
        for(i=0;i<4;i++)
        {
                move(comp(&p,&a,&p));
                if(i<3)
```

```
                move(comp(&p,&b,&p));
            else
                stop(comp(&p,&b,&p));
        }
    }
```

Ejemplo 9

Lenguaje: **FDTL**

Tarea: definición de reglas para un GC (vehículo recolector de basura)

```
rules
  begin
    GC
      begin
        if dmin==NEAR then ObstacleAvoidance;
        if dmin==FAR then Goals;
          ObstacleAvoidance
            begin
              if d1==VN and d2==VN and d3==VN and d4==VN
                 then Vleft:=NS and Vright:=NS;
              if d1==VN and d2==NE and d3==NE and v4==VN
                 then Vleft:=PS and Vright:=PS;
            end
          Goals
            begin
              if energy<=20 and damage=='yes'
                 then go_to_charger;
              if energy>20 and damage=='yes'
                 then go_to_service;
              ...
              if energy>20 and damage=='no' then do_jobs;
                 go_to_charger
                   begin
                   ...
                   end
                 go_to_service
                   begin
                   ...
                   end
                 do_jobs
                   begin
                     if hold_rub=='yes' then go_to_bin;
                     if hold_rub=='no' then go_to_rub;
                       go_to_bin
                         begin
                         ...
                         end
```

```
                                    go_to_rub
                                        begin
                                        ...
                                        end
                                end // end of do jobs
                        end // end of Goals
                end // end of GC
        end; // end of rules
```

Ejemplo 10

Lenguaje: **PRS**

Tarea: plan de desplazamiento de largo alcance.

```
(defka |Long Range Displacement|
:invocation (achieve (position-robot $x $y $theta))
:context (and (test (position-robot @current-x @current-y @current-theta))
              (test (long-range-displacement $x $y $theta @current-x
              @current-y @current-theta)))
:body ((achieve (notify all-subsystems displacement))
(wait (V (robot-status ready-for-displacement) (elapsed-time (time) 60)))
(if (test (robot-status ready-for-displacement))
  (while (test (long-range-displacement $x $y $theta @current-x @current-y
              @current-theta))
   (achieve (analyze-terrain))
   (achieve (find-subgoal $x $y $theta @sub-x @sub-y @sub-theta)
   (achieve (find-trajectory $x $y $theta @sub-x @sub-y @sub-theta @traj)
   (& (achieve (execute-trajectory @traj))
      (maintain (battery-level 0.200000)))
   (test (position-robot @current-x @current-y @current-theta)))
   (achieve (position-robot @x @y @theta))
else
(achieved (failed)))))
```

Ejemplo 11

Lenguaje: **GOLOG**

Tarea: el robot entrega correo en las oficinas, dice "hello" cuando pasa cerca de una cierta habitación, e interrumpe su actual acción cuando el nivel de la batería baja de 46 voltios y entonces recarga la batería.

```
withPol(loop(waitFor(battLevel<=46,
 seq(grapWhls,chargeBatteries,releaseWhls)))),
 withPol(loop(waitFor(nearDoor6213,
  seq(say(hello),waitFor(nearDoor6213))))
  withCtrl(wheels,deliverMail)))
```

Explicación: La primera línea significa que el robot está esperando hasta que el nivel de la batería baje del nivel 46. En ese momento, la acción atómica grabWheels establece wheels a false y se bloquea el programa deliverMail. Cuando la tarea chargeBatteries se completa, la acción releaseWhls establece wheels a true y deliverMail retoma su ejecución.

La tarea deliverMail hace uso de las acciones startGo(x,y), waitFor(destino) y stop.

Ejemplo 12

Lenguaje: **ESL**

```
(defun recovery-demo-1 ()
   (with-recovery-procedures
    ( (:widget-broken
        (attempt-widget-fix :broken)
        (retry))
      (:widget-broken
        (attempt-widget-fix :severely-broken)
        (retry)
      (:widget-broken :retries 3
        (attempt-widget-fix :weird-state)
        (retry)) )
  (operate-widget)))
```

Ejemplo 13

Lenguaje: **MLR**
Tarea: programa cooperativo con dos robots

```
:- agent (robotic_agent).
new :- init,run.
init :- #robot:new(robot(left,.)),
  #robot:new(robot(right,.)).
run:-
   *do(left,pickup(.),[ok,C0]),
   *do(left,deliver(.),[C0,C1,C2,C3,C4]),
   *do(right,receive(.),[ok,C1,C2,C3,C5]),
   *do(right,release,[C5,C6]).
:- agent(robot).
new(robot(Robot,.)):-init(.),run(Robot).
Init(.):- #socket:new(.).
[…]
run(Robot):-
   ^do(Robot,deliver(.),[ok,C0,C1,C2,C3])|
   *do(mp(.),[ok,C0]), *do(go,[C1,C2]),
   *do(mp(.),[C2,C3]),
```

```
    run(Robot).
  run(Robot):-
    ^do(Robot,receive(.),[ok,C0,C1,C2,C3])|
    Y1 := Y-50,
    *do(mp(.),[ok,_]), *do(mp(.),[C0,C5]),
    *do(gc,[C45,C1], *do(mp(.),[C2,_]),
    *do(mp(.),[C2,C3]),
    run(Robot).
  Otherwise.
    run(Robot):- ^NonSense | run(Robot).
```

Ejemplo 14

Lenguaje: **RoboML**
Tarea: se establece un mensaje del agent AGV1 al interface de usuario MyInterface

```
<set sender='AGV1" receiver="MyInterface" ontology="Hardware">
    <robot name="AGV1">
        <wheel name="1">
            <motor name="steering motor">
                <position>2577</position>
            </motor>
        </wheel>
        <wheel name="2">
            <motor name="steering motor">
                <position>754</position>
            </motor>
        </wheel>
    </robot>
</set>
```

Ejemplo 15

Lenguaje: **DARL**
Tarea: pick and place

```
10 SPEED 20 'Set the speed to a slow value for checkout
20 T1 400 -70 -25 50 'Define the first point T1= 400 -70 -25 50
30 T2 290 -275 -50 90 'Define the second point T2= =290 -275 -50 90
40 T3 -13 400 -17 -45 'Define the third point T3= -13 400 -17 -45
50 MOVE T1 'Move to point T1
60 PRINT "LINE 60"
70 INPUT "ENTER Y TO CONTINUE" S
80 IF S = 89 THEN GOTO 110
90 '89 is the ASCI value for Y
```

```
100 GOTO 60 'Try again if S <> Y
110 MOVE T2 'move to point T2
120 OUTPUT +OG0 250
130 'This turns on the vacuum gripper
140 DELAY 2000 'Delay 2 seconds
150 MOVE T3 'Move to the third TRANSLATION POINT
160 OUTPUT +OG1 250 'Turn off the gripper
170 DELAY 200 'Wait for the gripper to operate
180 PRINT "DID I DO OK?"
190 'Another message for the operator
200 INPUT "ENTER Y OR N" S 'Provide an operator response
210 IF S = 89 THEN GOTO 50 'Test the INPUT response
220 PRINT "I NEED TO REST"
240 STOP 'Push START to demonstrate
250 'START after STOP
260 PRINT "GOT TO LINE 260"
270 PRINT "THE END IS NEAR" 'No operator response required
280 END
```

Ejemplo 16

Lenguaje: KRL, Kula Robot Language

```
DEF example()
   DECL INT i
   DECL POS cpos
   DECL AXIS jpos

   FOR i = 1 TO 6
      $VEL_AXIS[I] = 60
   ENDFOR

   jpos = {AXIS: A1 0, A2 -90, A3 90, A4 0, A5 0, A6 0}
   PTP jpos

   IF $IN[1] == TRUE THEN
      cpos =  {POS: X 300, Y -100, Z 1500, A 0, B 90, C 0}
   ELSE
      cpos =  {POS: X 250, Y -200, Z 1300, A 0, B 90, C 0}
   ENDIF

   INTERRUPT DECL 3 WHEN $IN[2]==FALSE DO BRAKE
   INTERRUPT ON 3
```

```
   TRIGGER WHEN DISTANCE=0 DELAY=20 DO $OUT[2]=TRUE
   LIN cpos
   LIN {POS: X 250, Y -100, Z 1400, A 0, B 90, C 0} C_DIS
   PTP jpos

   INTERRRUPT OFF 3
END
```

Ejemplo 17

Lenguaje: V+
Tarea: paletizado

```
.PROGRAM move.parts()
;DESCRIPCION: Este programa coge cajas en la localización "pick"
; y las deposita en "place", incrementando la z de place, y así apilando
; las cajas en el palet.
        parts = 6 ; nº de cajas a apilar
        height1 = 300 ; altura de "approach/depart" en "pick"
        height2 = 500 ; altura de "approach/depart" en "place"
        parameter HAND.TIME = 0.16 ; movimiento del brazo lento
        OPEN ; apertura de pinza
        RIGHTY ; seleccionamos configuración derecha
        MOVE start ; mover a la localización segura de inicio
        FOR i = 1 TO parts ; iniciar el apilado de cajas
                APPRO pick, height1 ; ir a "pick-up"
                MOVES pick ; mover hacia la caja
                CLOSEI ; cerrar la pinza
                DEPARTS height1 ; volver a la posición anterior
                APPRO place, height2 ; ir a "put-down"
                MOVES place ; mover a la localización de destino
                OPENI ; abrir la pinza
                DEPARTS height2 ; volver a la posición anterior
                SHIFT heicht2 BY 0.00, 0.00, 300
        END
        TYPE "Fin de tarea. ", /IO, parts, " cajas apliadas."
        RETURN ; fin del programa
.END
```

Ejemplo 18

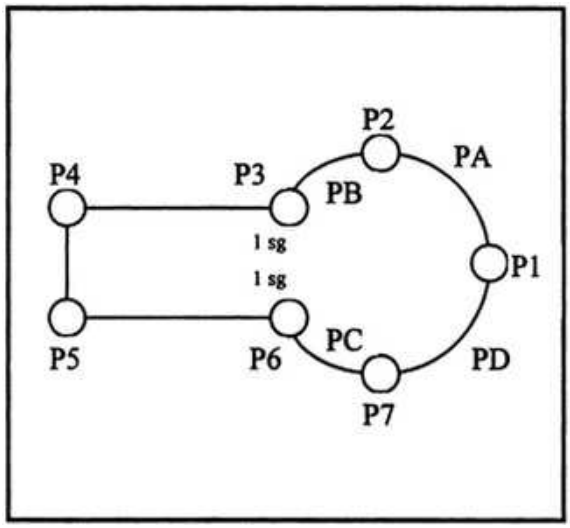
Lenguaje: TPP

```
SAMPLE1
      1:  J P[1]  100%  FINE
      2:  J P[2]  70%  CNT50
      3:  L P[3] 100cm/min CNT30
      4:  L P[4]  500mm/sec FINE
      6:  J P[5]  100% FINE
[End]
```

Ejemplo 19

Lenguaje: ARLA

```
        10 V=500mm/s    MAX=2000mm/s
        20 TCP 0
        30 COORD RECT
        40 BASCOOR 0
(P1)    50 POS V=100% FINA M
(PA)    60 POS V=50% CIRCULAR
(P2)    70 POS V=50% FINA M
(PB     80 POS V=50% CIRCULAR
(P3)    90 POS V=50% FINA M
       100 ESPERAR 1S
       110 COORD ROBOT
(P4)  120 POS V=100% FINA M
(P5)  130 POS=100% FINA M
(P6)  140 POS V=100% FINA M
       150 ESPERAR 1S
       160 COORD RECT
(PC)  170 POS V=50% CIRCULAR
(P7)  180 POS V=50% FINA M
(PD)  190 POS V=50% CIRCULAR
(P1)  200 V=50% FINA M
       210 RETORNO
```



Ejemplo 20

Lenguaje: LRP
Tarea: robot móvil que sigue una línea negra y rebota si choca

```
1 (var lightlim := [128]) (var maxlook := [100]) (var forward := [0.2])
2 (var search := [0.2]) (var back := [=0.2]) (var turn := [1])
3 (machine follower
4 (state moving (running [robot move: [ :msg | msg linear x: forward]]))
5 (on outofline moving => looking tlooking)
6 (on intheline looking => moving tmoving)
7 (event outofline [robot light data > lightlim + 10])
8 (event intheline [robot light data < lightlim = 10])
9 (state looking
10 (machine lookalgo
11 (var time := [ maxlook ])
12 (state lookleft
13 (running [robot move: [ :msg | msg angular z: search]]))
14 (state returnleft
15 (running [robot move: [ :msg | msg angular z: search * =1]]))
```

```
16 (state lookright
17 (running [robot move: [ :msg | msg angular z: search * =1]]))
18 (state returnright
19 (running [robot move: [ :msg | msg angular z: search]])
20 (onexit [time := time * 2 ]))
21 (ontime time lookleft => returnleft treturnleft)
22 (ontime time returnleft => lookright tlookright)
23 (ontime time lookright => returnright treturnright)
24 (ontime time returnright => lookleft tlookleft))
 25 (onentry (spawn lookalgo lookleft)))
 26 (var nobump := [true])
27 (event bumping [robot bumper data == 1 & nobump])
28 (event ending [robot bumper data == 1 & nobump not])
29 (on bumping *=> bumpback)
30 (on ending *=> end tend)
31 (state bumpback
32 (onentry [ nobump = false])
33 (running [ robot move: [ :msg | msg linear x: back]]))
34 (state bumpturn
35 (running [ robot move: [ :msg | msg angular z: search]]))
36 (ontime 1000 bumpback => bumpturn)
37 (ontime 3000 bumpturn => looking)
38 (state end)
39)
 40(spawn follower looking)
```

Las línas 1-2 definen variables globales. En la línea 3 empieza a definición de la máquina follower. La línea 4 define el estado moving con una acción “running” (que se ejecutará mientras el estado esté activo), que es un bloque de código Smalltalk. Las líneas 7-8 crean eventos. Las líneas 5-6 definen transiciones de un estado a otro cuando se produce uno de los eventos definidos. En las líneas 9-25 se define el estado compuesto looking, y dentro de este se define el estado lookalgo (líneas 10-24). En la línea 25 una acción onentry del estado looking genera la máquina lookalgo. A partir de la 26 está el código responsable del comportamiento de choque, retroceso y giro

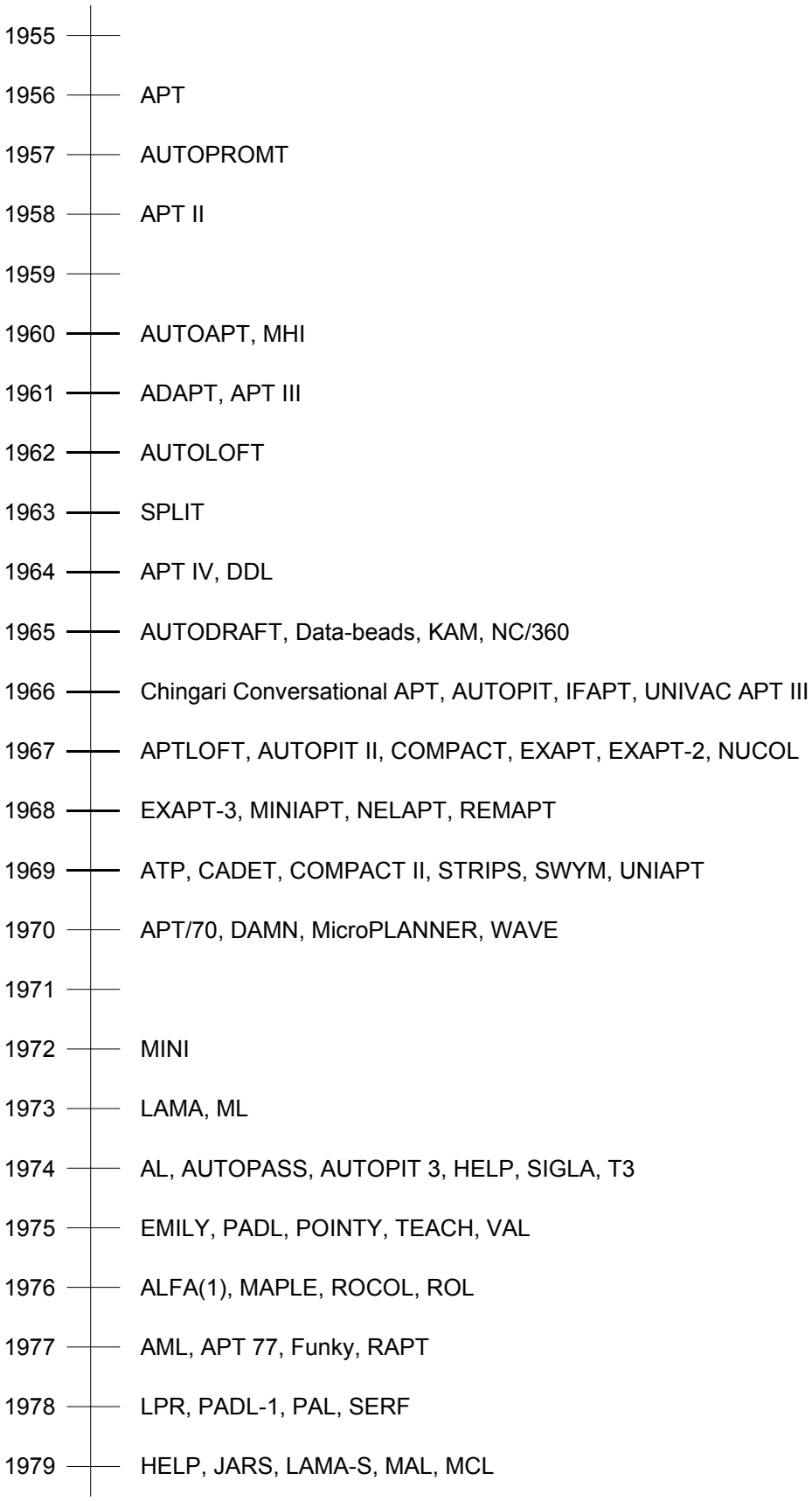
Ejemplo 21

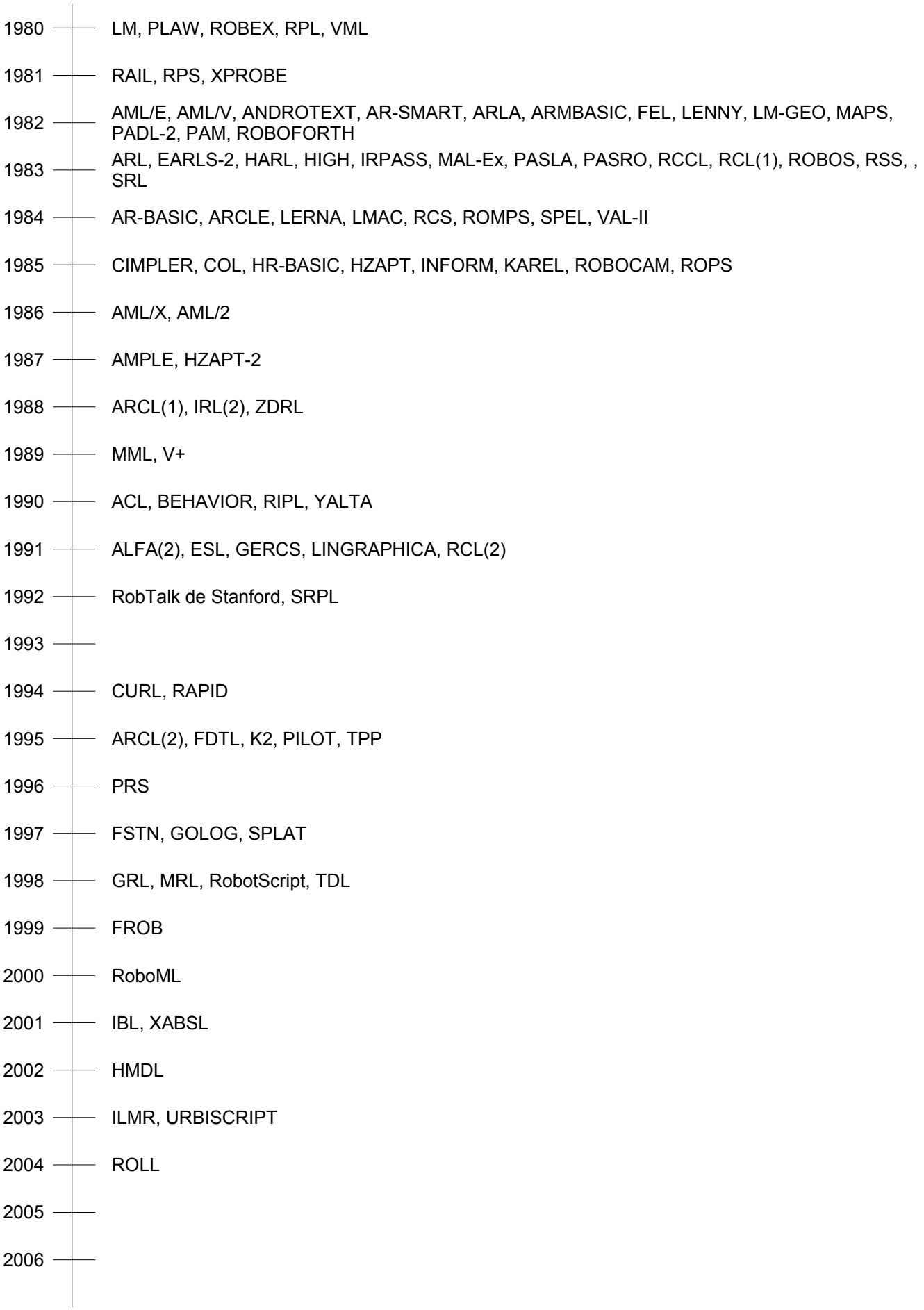
Lenguaje: IRL (Industrial Robot Language)

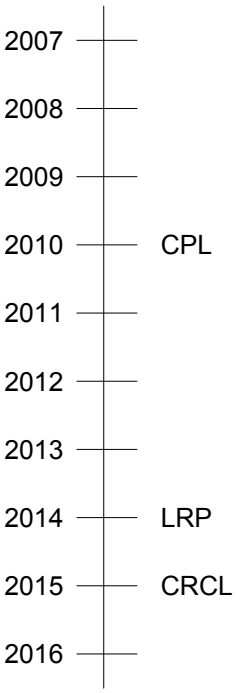
MOVE LIN p1 → instrucción de movimiento al punto p1
SPEEDOVERRIDE size → modifica parámetos para instrucciones de movimiento
CONTINUE; → permite continuar con las siguientes instrucciones sin esperar a que finalice el movimiento

MOVE DONE := FALSE;
REPEAT → bucle
size := 0.233 * analog3 +0.55; → cálculo con el valor de la entrada analógica 3
WAIT_SAMPLE(); → sincronización de la señal con el ratio de muestreo del controlador
UNTIL (MOVE_DONE); → el bucle se repite hasta que move_done sea true

Anexo II.- Línea del tiempo de los lenguajes de programación de robots industriales




1980
LM, PLAW, ROBEX, RPL, VML
1981
RAIL, RPS, XPROBE
1982
AML/E, AML/V, ANDROTEXT, AR-SMART, ARLA, ARMBASIC, FEL, LENNY, LM-GEO, MAPS, PADL-2, PAM, ROBOFORTH
1983
ARL, EARLS-2, HARL, HIGH, IRPASS, MAL-Ex, PASLA, PASRO, RCCL, RCL(1), ROBOS, RSS, , SRL
1984
AR-BASIC, ARCLE, LERNA, LMAC, RCS, ROMPS, SPEL, VAL-II
1985
CIMPLER, COL, HR-BASIC, HZAPT, INFORM, KAREL, ROBOCAM, ROPS
1986
AML/X, AML/2
1987
AMPLE, HZAPT-2
1988
ARCL(1), IRL(2), ZDRL
1989
MML, V+
1990
ACL, BEHAVIOR, RIPL, YALTA
1991
ALFA(2), ESL, GERCS, LINGRAPHICA, RCL(2)
1992
RobTalk de Stanford, SRPL
1993
1994
CURL, RAPID
1995
ARCL(2), FDTL, K2, PILOT, TPP
1996
PRS
1997
FSTN, GOLOG, SPLAT
1998
GRL, MRL, RobotScript, TDL
1999
FROB
2000
RoboML
2001
IBL, XABSL
2002
HMDL
2003
ILMR, URBISCRIPT
2004
ROLL
2005
2006


2007
2008
2009
2010
CPL
2011
2012
2013
2014
LRP
2015
CRCL
2016