Robótica con ladrillos LEGO

José María Cañas Plaza

jmplaza@gsyc.escet.urjc.es

Abril 2003

Índice del curso

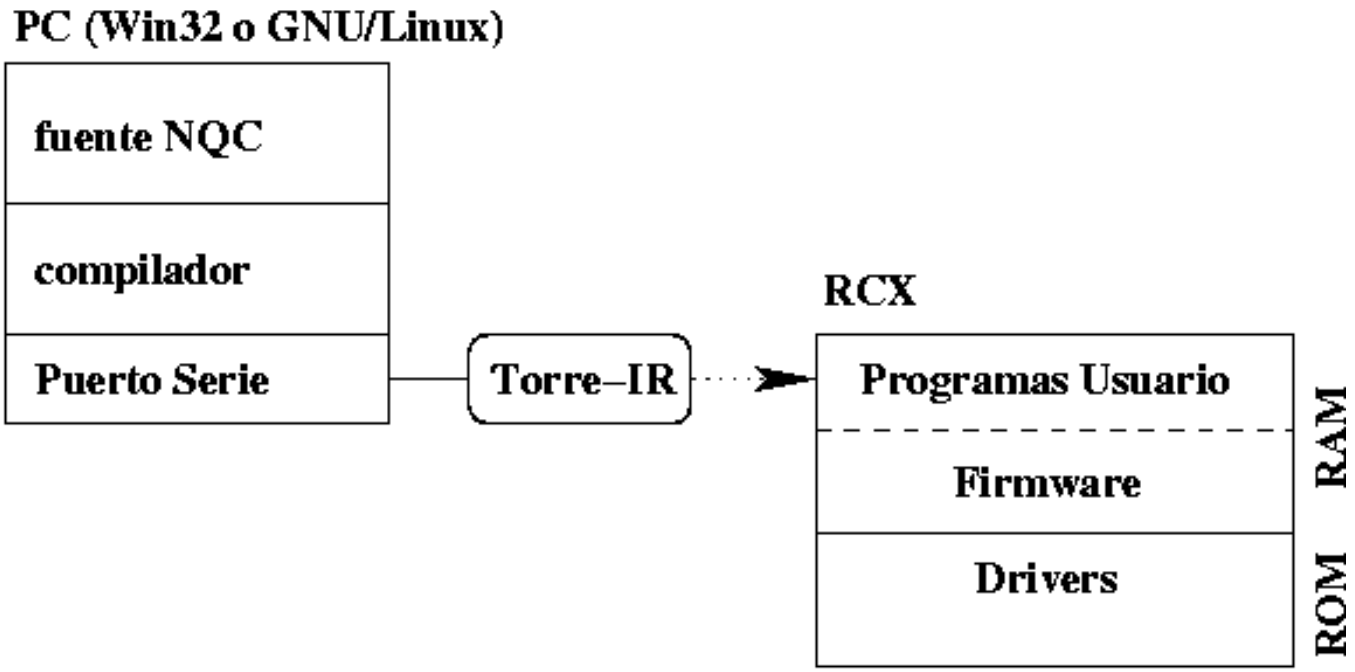
- Introducción
- Elementos del kit
- Programación con código RCX
- Programación con NQC
- Programación con C y BrickOS
- Programación con Java y LeJOS
- Conclusiones

Programación con NQC

NQC (Not Quite C)

- Sintaxis similar al lenguaje C
- Múltiples tareas, la principal es obligatoria (`main`)
- Manejo de variables
- Cuatro temporizadores: contadores de 100 ms.
- Generación de números aleatorios (`Random()`)
- Espera temporal (`Wait`) y por eventos (`until`)
- Manejo de sonido (`PlayTone()`)
- Comunicación IR (`SendMessage()`)

Arquitectura software usando NQC



Uso típico

- Escribir el código fuente `test.nqc`
- Compilador cruzado genera *código ejecutable*
- Es código intermedio `test.rcx`, la máquina virtual RCX lo interpreta
- Descarga del firmware
  `nqc -firmware firm0328.lgo`
- Generación y descarga del ejecutable
  `nqc -d -S/dev/ttyS0 test.nqc`

Actuadores

Programación de motores

- OUT_A, OUT_B, OUT_C
- Los motores tienen 3 modos: *on-off-floating*
  - `On(OUT_A+OUT_B)`
  - `Off(OUT_A)`
  - `Float(OUT_A)`
- Sentido giro
  - `Toggle(OUT_A),Fwd(OUT_A), Rev(OUT_A)`
  - No mueven al motor OJO
  - Se recuerdan hasta la siguiente activación

- Combinados
  - `OnFwd(OUT_A)`
  - `OnRev(OUT_A)`
  - `OnFor(OUT_B,200);` //centésimas de segundo
- Control velocidad
  - `SetPower(OUTPUT_B+OUTPUT_C,3)`
  - de 0 a 7, se visualiza 1 a 8

Programación sonido

- `PlayTone(440,25)`
  Frecuencia (herzios), duración (décimas)
- `PlaySound(`*número*`)`
  Sonidos pregrabados
  SOUND_CLICK, SOUND_DOUBLE_BEEP,
  SOUND_DOWN, SOUND_UP,
  SOUND_LOW_BEEP, SOUND_FAST_UP

Acceso a sensores

- SENSOR_1, SENSOR_2, SENSOR_3
- `SetSensor(Sensor, const Configuración)`

Configuración	Tipo	Modo	`ClearSensor()`
`SENSOR_TOUCH`	Contacto	1 (presionado) 0 (no)	-
`SENSOR_LIGHT`	Luz	0 (oscuro) 100 (luz)	-
`SENSOR_ROTATION`	Rotación	16 por vuelta	Sí
`SENSOR_CELSIUS`	Temperatura	grados centígrados	-
`SENSOR_FARENHEIT`	Temperatura	grados Farenheit	-
`SENSOR_PULSE`	Contacto	Contador de pulsaciones	Sí
`SENSOR_EDGE`	Contacto	Contador de transiciones	Sí

- `ClearSensor(SENSOR_3)` para sensores acumulativos
- NO se pueden combinar con $+$
  `SetSensor(SENSOR_1+SENSOR_2,SENSOR_TOUCH)` genera error
- Tipo y modo se pueden configurar por separado:
  - `SetSensorType()`: características eléctricas
  - `SetSensorMode()`: cómo se interpretan los valores

Variables y expresiones

```
int mivariable;
task main()
{
int velocidad=1;
}
```

```
bar = 10;
bar += 7;
bar = SENSOR_1 + 7;
bar = Timer(0);
```

```
bar = bar & 0x7;
bar = abs(bar);
bar = sign(bar); // (-1,+1)
var = 7 << 2
```

- Variables enteras. Locales o globales
- Asignar valor: otra variable, un sensor, una expresion
- Operaciones aritméticas: +,-,*,/, %, ++, - -
- Precedencia de operadores. Usar paréntesis.
- Manejo de bits: AND, OR, complemento, desplazamiento

Estructuras de control

```
until(SENSOR_1 == 1)
until(SENSOR_1 != 30)
until(Timer(0) < 100)
until(SENSOR_1 > 10
&& SENSOR_1 < 20);
```

```
while(SENSOR_1 == 1)
{
PlayTone(440,20);
}
```

```
do
{
PlayTone(440,20);
} while(SENSOR_1 == 1)
```

- Condiciones
- Combinaciones lógicas: &&, ||, <=, >, ...
- `until(`*condición*`);`
- `while(`*condición*`){` *sentencia* `}`
- `do {` *sentencia*`}` `while (`*condición*`)`

```
repeat(10)
{
until(SENSOR_1 == 1)
PlayTone(440,10);
}
```

```
if (SENSOR_2 == 1)
On(OUT_A);
else
On(OUT_B);
```

```
if (SENSOR_2 == 1)
On(OUT_A);
On(OUT_B);
```

- `repeat(n){` *sentencia* `};`
- `if (`*condición*`)` *sentencia1*`;`
- `if (`*condición*`)` *sentencia1*`; else` *sentencia2*`;`

Temporizadores

- Se incrementan cada 100 ms.
- `Wait(`*centésimas segundo*`)`
- Temporizadores
  - Se tienen 4
  - Aparecen en las condiciones
  - `Timer(2)` para consultar
  - `ClearTimer(3)`, de 0 a 3

Tareas

```
task main()
{
SetSensor(SENSOR_1,
SENSOR_TOUCH);
SetSensor(SENSOR_3,
SENSOR_TOUCH);
start agudo;
start rota;
Wait(1000);
StopAllTasks;
}
```

```
task agudo()
{
while(true)
{ until(SENSOR_3 == 1);
PlayTone(440,10);
}}
```

```
task rota()
{
while(true)
{ Off(OUT_C);
until(SENSOR_1 == 1);
OnFor(OUT_C,100);
}}
```

- `start, stop`
- `StopAllTasks();`

Funciones

```
void pulso(int count, int duracion)
{
   repeat(count)
    {
     OnFor(OUT\_A,duracion);
     Wait(duracion);
    }
}
```

- Trozos de código que se van a invocar muchas veces
- No devuelven nada
- Admiten parámetros (paso valor y paso referencia)

Preprocesador

- comentarios
  - // ignora el resto de la línea
  - /* */
- #define RUEDA_IZQ OUT_A
- Permite poner nombres más significativos
- *Funciones* como macros
- #define espera(outs,secs) OnFor(outs,(secs)*100)

Varios

- `Random(n)` entrega un número entero aleatorio entre 0 y n.
- Comunicaciones
  - Message();
  - ClearMessage();

Ejemplos

Primer programa

```
task main()
{
SetSensor(SENSOR_1, SENSOR_TOUCH);
On(OUT_A);

while(true)
{
until(SENSOR_1 == 1);
Toggle(OUT_A);
until(SENSOR_1 == 0 );
}}
```

Ejemplo choca-gira con NQC

```
int tiempo, giro;

task main() {
  SetSensor(SENSOR_1,SENSOR_TOUCH);
  SetSensor(SENSOR_2,SENSOR_TOUCH);
  start avanzar;
  start evitar;
}

task avanzar(){
  while(true){
    OnFwd(OUT_A+OUT_C);
  }
}
```

```
task evitar(){
  while(true){
    if (SENSOR_1 == 1) {
      stop avanzar;
      onRev(OUT_C); Wait(50);
      start avanzar;
    }
    if (SENSOR_2 == 1) {
      stop avanzar;
      onRev(OUT_A); Wait(50);
      start avanzar;
    }
  }
}
```

Pros y contras de NQC

- Usa el *firmware* de LEGO
- Sólo se pueden usar 32 variables
- Es interpretado
- Las subrutinas no pueden devolver valores
- No existen estructuras de datos, sólo enteros
- Más ágil que *código RCX*
- Basado en texto

Bibliografía NQC

- The Unofficial Guide to LEGO MINDSTORMS Robots, Jonathan B. Knudsen, O'Reilly, 1999
- Dave Baum's definitive guide to LEGO MINDSTORMS, Dave Baum, Apress, 2000
- En la web existen muchos manuales