UNIVERSIDAD DE OVIEDO

Departamento de Ingeniería Eléctrica,
Electrónica, de Computadores y Sistemas

# Guía de Prácticas

| | | | |
|---|---|---|---|
| ASIGNATURA: | Informática Industrial y Comunicaciones | | |
| CENTRO: | Escuela Politécnica de Ingeniería de Gijón | | |
| ESTUDIOS: | Grado en Ingeniería Electrónica y Automática | | |
| CURSO: | 3º | CUATRIMESTRE: | 1 |
| CARÁCTER: | Obligatoria | CRÉDITOS ECTS: | 6 |
| PROFESORADO: | Ignacio Alvarez, José Mª Enguita, Angel Navarro, Mariam Saeed | | |

PRACTICA 10: E/S en archivo, comunicaciones

1. Añadir al comienzo de programa (después de inicializar el simulador y arrancar el temporizador de control, antes del bucle de solicitud por teclado) lectura de comandos del archivo de entrada "init.txt", que contendrá las líneas siguientes:

```
RZ = [0.13, -0.11] / [1 -0.43]
POS = 90
SLEEP = 2500
POS = -90
SLEEP = 5000
POS = POT
```

   Como resultado, el programa debe realizar al comienzo la asignación del RZ de posición y ejecutar los movimientos indicados.

2. Añadir a un archivo de texto "comandos.log" los diferentes comandos recibidos en el control del simulador, incluyendo la fecha y hora de cada uno. Utilizar funciones siguientes de <time.h> (ver ayuda con buscador Internet: "man nombre_de_funcion"):

   `time_t time(time_t* pt_time);` Obtiene la fecha y hora actual, como un entero que indica el tiempo en segundos que ha pasado desde 1/Ene/1970

   `struct tm* localtime(const time_t* pt_time);` Obtiene la fecha y hora en una estructura con campos año, mes, día, etc

   Ejemplo de uso:

```
#include <time.h>

time_t ahora_sec;
struct tm* ahora;
…
ahora_sec=time(NULL);
ahora=localtime(&ahora_sec);
printf("La hora es: %02d:%02d\n",ahora->tm_hour,ahora->tm_min);
```

   Ejemplo de salida deseada en el archivo comandos.log:

```
02/12/2018 - 10:27:19  >> Arrancado programa de control
02/12/2018 - 10:27:25  >> RZ = [0.13,-0.11] /[1 -0.43]
02/12/2018 - 10:27:29  >> POS = 90
02/12/2018 - 10:27:30  >> SLEEP =3000
02/12/2018 - 10:27:33  >> POS =POT
```

**AMPLIACIONES**

3. Añadir el servicio de los siguientes comandos a la función ProcesarComando( ):

| Comando | Objetivo |
|---|---|
| `CONNECT=dirección ip / puerto` | Conecta un socket cliente TCP con el servidor de la dirección y puerto indicados. Usar para pruebas:<br>- Dirección IP: 156.35.152.87<br>- Puerto: 52323 |
| `USER=UOxxxxxx` | Identifica al cliente para que el servidor le envíe comandos cada 5seg, y realice un "log" de la actividad. **Utilizar usuario UO de cada alumno** para que quede registrada su actividad. |
| `SEND=STOP ó RESTART` | Envía comando al servidor para que deje de enviar datos, o que vuelva a enviarlos. |
| `DISCONNECT` | Desconecta el cliente TCP previamente conectado |

Pasos en el programa:

a) Activar en archivo .pro la librería de Windows Sockets (ejecutar run qmake tras ello):

```
LIBS += -lws2_32
```

b) Añadir en .c la inclusión de los archivos de cabecera de la librería:

```
#include <winsock2.h>
```

c) Crear variable global tipo SOCKET:

```
SOCKET _sockCli=INVALID_SOCKET;
```

d) Al comienzo del programa (main, tras conexión con simulador):

```
_sockCli =socket(AF_INET,SOCK_STREAM,0);

struct sockaddr_in add_local;
add_local.sin_family=AF_INET;
add_local.sin_addr.S_un.S_addr=0;
add_local.sin_port=htons(ip_port);
memset(add_local.sin_zero,0,8);
bind(_sockCli,(struct sockaddr*) &add_local,
              sizeof(struct sockaddr_in));
```

e) A la recepción del comando CONNECT:

```
struct sockaddr_in add_remote;
rellenar campos de add_remote según IP y puerto del comando
int err=connect(_sockCli,(struct sockaddr*) & add_remote,
                         sizeof(struct sockaddr_in));
if (err==0) {
   char myUO="UOxxxxxx\n";
   send(_sockCli, myUO,strlen(myUO),0);
} else {
   mensaje de error en conexión
}
```

f) A la recepción de comando SEND

```
Enviar mediante send() la parte tras el '='
```

g) A la recepción de comando DISCONNECT

```
close(_sockCli);
_sockCli=INVALID_SOCKET;
```

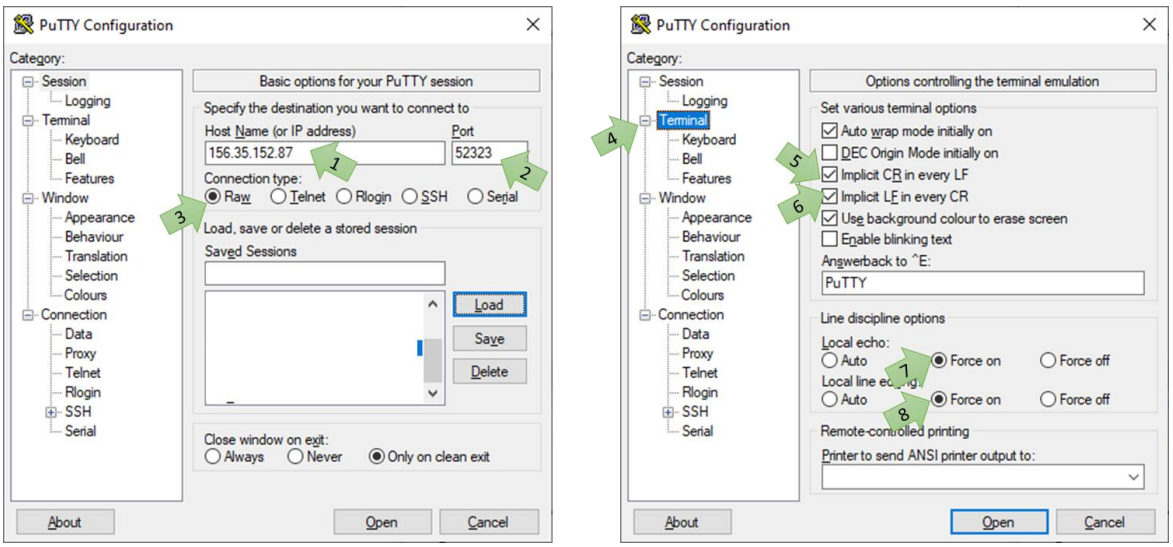
h) En la rutina de interrupción, comprobar la disponibilidad de datos en el socket, y leer y procesar si están disponibles:

```
if (_sockCli!=INVALID_SOCKET) {
   FD_SET readSet;
   FD_ZERO(&readSet);
   FD_SET(_sockCli, &readSet);
   struct timeval timeOut={0,0};
   select(0, &readSet, NULL, NULL, &timeOut);
   if (FD_ISSET(_sockCli,&readSet)) {
      char cmd[40];
      int n=recv(_sockCli,cmd,40,0);
      procesar cmd
   }
}
```

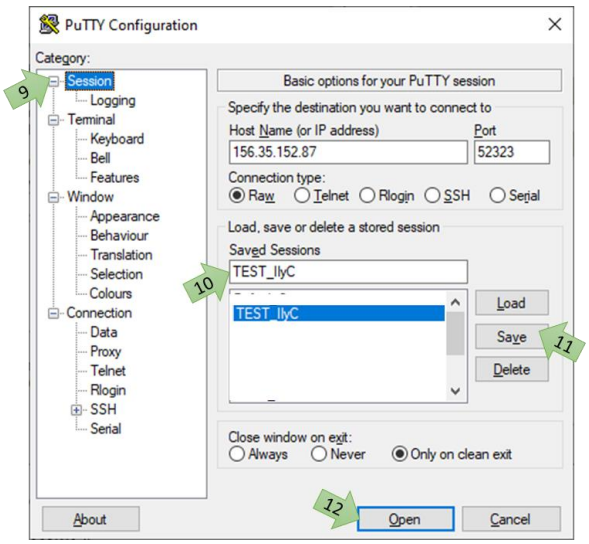
Se debe observar que, tras estos pasos, se producen las acciones esperadas cada 5 seg:

`POS=90 , POS=-90, VEL=20, vuelta a empezar`

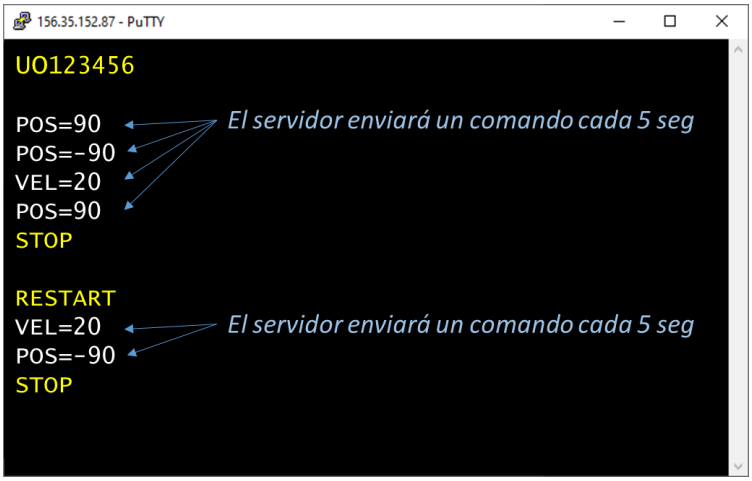
**Para pruebas iniciales del funcionamiento del servidor, se puede utilizar el programa putty**, descargable desde https://www.putty.org/. La configuración a establecer es:




Una vez establecida la configuración, se puede salvar para disponer de la misma conexión en posteriores ocasiones a través de Load:




El resultado a observar según la interacción del usuario (en amarillo) será algo como:


156.35.152.87 - PuTTY
UO123456
POS=90
POS=-90
VEL=20
POS=90
STOP
El servidor enviará un comando cada 5 seg
RESTART
VEL=20
POS=-90
STOP
El servidor enviará un comando cada 5 seg