# The RRC Project
# Manual del Sistema

Arakyd Sofware

2 de junio de 2014

# Índice

# 1. Introducción

El objetivo del proyecto es un sistema distribuido para la implementación y prueba de algoritmos de control sobre instancias físicas y simuladas del robot Khepera III. En el Host Controlador se ejecutará un conjunto de programas que permitirá al usuario ejecutar sus algoritmos de manera remota en los Khepera físicos o el Host Simulador, que simula las instancias virtuales del robot. El Software y entorno de simulación empleado será V-REP.

Este documento pretende servir de guía para el mantenimiento o modificación del código del software, proporcionando información acerca de los ítems que contiene el programa y una breve explicación del funcionamiento de cada una de las partes que lo componen.

# 2. KheperaSimGui_fuente

La carpeta KheperaSimGui_fuente contiene el código fuente que conforma el programa desarrollado por el equipo de ingeniería Arakyd Software. Las siguientes subsecciones tratarán, con cierto detalle, cada uno de los archivos que encontramos en ésta carpeta.

## 2.1. KheperaSimGui.pro

Proyecto de Qt que engloba todo lo contenido en esta carpeta, las clases, código fuente, la GUI, así como los headers y código de la API remota y V-rep, es el archivo de entrada a qmake y responsable de generar el makefile.

## 2.2. Carpeta: include

Contiene headers del entorno de simulación V-rep, así como de la API remota, para más información ver la documentación asociada.

- `http://coppeliarobotics.com/helpFiles/index.html`
- `http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctions.htm`

Los archivos que tenemos son:

- extApiCustomConst.h
- v_repConst.h
- v_repLib.h
- v_repTypes.h

## 2.3. Carpeta: remoteApi

API remota para V-rep, contiene las funciones que emplearemos para interactuar con los robots.

Archivos:

- extApi.c
- extApi.h
- extApiCustom.c
- extApiCustom.h
- extApiInternal.h
- extApiPlatform.c
- extApiPlatform.h

## 2.4. Clase I_control

La clase abstracta I_control, debe ser implementada por la clase del código cliente. Ésta consta de 5 métodos virtuales puros que se sobrecargarán con el control.cpp o demo.cpp.

```
1  //"Interfaz" (clase abstracta) que debe implementar la clase de
       código cliente
2  //Ver clase ejemplo en Demo.
3  #ifndef I_CONTROL_H
4  #define I_CONTROL_H
5
6
7
8
9  class I_Control
10 {
11     public:
12         virtual int control(int clientID) = 0; //Método virtual
               puro, debe sobrecargarse con el contenido de Control.
13         virtual int interrupt_1(int clientID) = 0; //Métodos
               virtuales puros, debe sobrecargarse con el contenido de
               interrupciones.
14         virtual int interrupt_2(int clientID) = 0; //(normalmente
               adelante,atras,izq y der, respectivamente).
15         virtual int interrupt_3(int clientID) = 0;
16         virtual int interrupt_4(int clientID) = 0;
17 };
18
19 #endif
```

## 2.5. Clase Demo

La clase de ejemplo que implementa la clase abstracta I_control, se emplea en el modo de demostración. Ésta define las 2 funciones que permitirán controlar qué robot seleccionar y la velocidad de los motores, además de algunas variables empleadas por demo.cpp.

```
1  //Definicion Clase Ejemplo de Código Usuario.
2  #ifndef DEMO_H
3  #define DEMO_H
4  #include <i_control.h>
5
6  //La clase implementa la "interfaz" (clase abstracta) i_control
7  class Demo: public I_Control
8  {
9      public:
10         int control(int clientID);
11         int interrupt_1(int clientID);
12         int interrupt_2(int clientID);
13         int interrupt_3(int clientID);
14         int interrupt_4(int clientID);
15         //establece el nombre del robot a controlar con las
               interrupciones demo.
16         void setRobot (int indice_robot);
17         //establece la velocidad de los motores en las
               interrupciones demo.
18         void setVelocidad (int multiplicador);
19     private:
20         std::string leftmotor = "K3_leftWheelMotor#";
21         std::string rightmotor = "K3_rightWheelMotor#";
22         const float pi=3.141592;
23         float velocidad = pi;
24 };
25
26 #endif // DEMO_H
```

## 2.6. demo.cpp

Se programan las cinco funciones que mapean los métodos de la clase abstracta I_control y las dos encargadas de la selección del robot y la velocidad de los motores.

Éste modo de ejemplo emplea, principalmente, funciones de la Api Remota (`http://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctions.htm`)

```
1  //Clase Ejemplo de Código Usuario. Ver Documentación de Remote
       Api para entender funciones.
2  #include <demo.h>
3  #include <sstream>
```

```
4  #include <math.h>
5  extern "C" {
6  #include "extApi.h"
7  }
8
9  using namespace std;
10
11 int Demo::control(int clientID) {
12     cout << "Demo: Inicio de Control\n";
13     int leftMotorHandle;
14     int rightMotorHandle;
15     int leftMotorHandle0;
16     int rightMotorHandle0;
17     int error[4]={0,0,0,0};
18     int i=0;
19
20     //genera un handle para los motores de khepera y khepera#0 y
          de los objetos robot en sí.
21     do {
22             error[0]= simxGetObjectHandle(clientID,"
                  K3_rightWheelMotor#",&rightMotorHandle,
                  simx_opmode_oneshot_wait);
23             error[1]= simxGetObjectHandle(clientID,"
                  K3_leftWheelMotor#",&leftMotorHandle,
                  simx_opmode_oneshot_wait);
24             error[2]= simxGetObjectHandle(clientID,"
                  K3_rightWheelMotor#0",&rightMotorHandle0,
                  simx_opmode_oneshot_wait);
25             error[3]= simxGetObjectHandle(clientID,"
                  K3_leftWheelMotor#0",&leftMotorHandle0,
                  simx_opmode_oneshot_wait);
26         i++;
27         cout << "Código de error: " << error[0] << error[1] <<
              error[2] << error[3] << "\n";
28     } while ((error[0]!=0 || error[1]!=0 || error[2]!=0 || error
          [3]!=0) && i < 5);
29
30
31
32     while (simxGetConnectionId(clientID)!=-1)
33     {
34         //mover khepera
35         simxSetJointTargetVelocity(clientID,leftMotorHandle,2*pi,
              simx_opmode_oneshot);
36         simxSetJointTargetVelocity(clientID,rightMotorHandle,2*pi
              ,simx_opmode_oneshot);
37
38         //mover khepera#0
```

```
39          simxSetJointTargetVelocity(clientID,leftMotorHandle0,pi,
               simx_opmode_oneshot);
40          simxSetJointTargetVelocity(clientID,rightMotorHandle0,pi,
               simx_opmode_oneshot);
41
42      }
43      cout << "Demo: Control abortado\n";
44      return 0;
45
46  }
47  //adelante
48  int Demo::interrupt_1(int clientID){
49      int leftMotorHandle;
50      int rightMotorHandle;
51      int error[2] = {0,0};
52      int i =0;
53      float velocidad = this->velocidad;
54
55      cout << "Demo: Adelante v: " << velocidad << "\n";
56      do {
57          simxGetObjectHandle(clientID,rightmotor.c_str(),&
               rightMotorHandle,simx_opmode_oneshot_wait);
58          simxGetObjectHandle(clientID,leftmotor.c_str(),&
               leftMotorHandle,simx_opmode_oneshot_wait);
59          i++;
60      } while ((error[0]!=0 || error[1]!=0) && i < 5);
61
62      while (simxGetConnectionId(clientID)!=-1) {
63          simxSetJointTargetVelocity(clientID,leftMotorHandle,
               velocidad,simx_opmode_oneshot);
64          simxSetJointTargetVelocity(clientID,rightMotorHandle,
               velocidad,simx_opmode_oneshot);
65      }
66      return 0;
67  }
68  //marcha atrás
69  int Demo::interrupt_2(int clientID){
70      int leftMotorHandle;
71      int rightMotorHandle;
72      int error[2] = {0,0};
73      int i =0;
74      float velocidad = this->velocidad;
75
76      cout << "Demo: Marcha atrás v: " << velocidad << "\n";
77      do {
78          simxGetObjectHandle(clientID,rightmotor.c_str(),&
               rightMotorHandle,simx_opmode_oneshot_wait);
79          simxGetObjectHandle(clientID,leftmotor.c_str(),&
```

```
            leftMotorHandle,simx_opmode_oneshot_wait);
80          i++;
81      } while ((error[0]!=0 || error[1]!=0) && i < 5);
82
83      while (simxGetConnectionId(clientID)!=-1) {
84          simxSetJointTargetVelocity(clientID,leftMotorHandle,-
                velocidad,simx_opmode_oneshot);
85          simxSetJointTargetVelocity(clientID,rightMotorHandle,-
                velocidad,simx_opmode_oneshot);
86      }
87      return 0;
88  }
89  //giro izquierda
90  int Demo::interrupt_3(int clientID){
91      int i=0;
92      int leftMotorHandle;
93      int rightMotorHandle;
94      float angulos[3]={0,0,0};
95      float alfa =0;
96      float anterior=0;
97      float angulo=0;
98      float velocidad = this->velocidad;
99      int error[2] = {0,0};
100
101     cout << "Demo: Giro Izq. v: " << velocidad << "\n";
102     do {
103         simxGetObjectHandle(clientID,rightmotor.c_str(),&
                rightMotorHandle,simx_opmode_oneshot_wait);
104         simxGetObjectHandle(clientID,leftmotor.c_str(),&
                leftMotorHandle,simx_opmode_oneshot_wait);
105         i++;
106     } while ((error[0]!=0 || error[1]!=0) && i < 5);
107     i=0;
108     //para girar 90°, obtiene la orientación (angulos de euler
            con respecto a los ejes absolutos).
109     simxGetObjectOrientation(clientID,leftMotorHandle,-1,angulos,
            simx_opmode_oneshot_wait);
110     //alfa es el angulo beta inicial.
111     alfa=angulos[1];
112     //anterior es necesario para calcular el incremento de angulo
            (debido al sistema angular de vrep: 0->90->0->-90->0)
113     anterior=alfa;
114     //angulo es el angulo actual en coordenadas radiales.
115     angulo=alfa;
116     while (simxGetConnectionId(clientID)!=-1) {
117         //calculo angulo nuevo con la diferencia del dado por
                vrep y el anterior dado por vrep (incremento).
118         simxGetObjectOrientation(clientID,leftMotorHandle,-1,
```

```
            angulos,simx_opmode_oneshot);
119         angulo= angulo+fabs(angulos[1]-anterior);
120         //si el angulo actual está mas allá de 90° del inicial,
               giro completo. Si no, sigue girando.
121         if ( angulo<alfa+pi/2 ) {
122             simxSetJointTargetVelocity(clientID,leftMotorHandle
                   ,0,simx_opmode_oneshot);
123             simxSetJointTargetVelocity(clientID,rightMotorHandle,
                   velocidad,simx_opmode_oneshot);
124
125         } else {
126             simxSetJointTargetVelocity(clientID,leftMotorHandle,
                   velocidad,simx_opmode_oneshot);
127             simxSetJointTargetVelocity(clientID,rightMotorHandle,
                   velocidad,simx_opmode_oneshot);
128         }
129         anterior=angulos[1];
130     }
131     return 0;
132 }
133 //giro derecha (ídem giro izquierda).
134 int Demo::interrupt_4(int clientID){
135     int i=0;
136     int leftMotorHandle;
137     int rightMotorHandle;
138     int error[2] = {0,0};
139     float angulos[3]={0,0,0};
140     float alfa =0;
141     float anterior=0;
142     float angulo=0;
143     float velocidad = this->velocidad;
144
145     cout << "Demo: Giro Der. v: " << velocidad << "\n";
146     do {
147         simxGetObjectHandle(clientID,rightmotor.c_str(),&
               rightMotorHandle,simx_opmode_oneshot_wait);
148         simxGetObjectHandle(clientID,leftmotor.c_str(),&
               leftMotorHandle,simx_opmode_oneshot_wait);
149         i++;
150     } while ((error[0]!=0 || error[1]!=0) && i < 5);
151     //para girar a la
152     simxGetObjectOrientation(clientID,rightMotorHandle,-1,angulos
           ,simx_opmode_oneshot_wait);
153     alfa=angulos[1];
154     anterior=alfa;
155     angulo=alfa;
156     while (simxGetConnectionId(clientID)!=-1) {
157         simxGetObjectOrientation(clientID,leftMotorHandle,-1,
```

```
            angulos,simx_opmode_oneshot);
158         angulo= angulo-fabs(angulos[1]-anterior);
159         if (angulo>alfa-pi/2) {
160             simxSetJointTargetVelocity(clientID,leftMotorHandle,
                    velocidad,simx_opmode_oneshot);
161             simxSetJointTargetVelocity(clientID,rightMotorHandle
                    ,0,simx_opmode_oneshot);
162
163         } else {
164             simxSetJointTargetVelocity(clientID,leftMotorHandle,
                    velocidad,simx_opmode_oneshot);
165             simxSetJointTargetVelocity(clientID,rightMotorHandle,
                    velocidad,simx_opmode_oneshot);
166         }
167         anterior=angulos[1];
168     }
169     return 0;
170 }
171 //adjunta el índice del robot al label de vrep, para identificar
       que robot "interrumpir".
172 //permitir robots con nombre personalizado es posible, pero
       resulta mas trabajoso de lo que reporta al usuario, por tanto
       no se incluye en la demo.
173 void Demo::setRobot(int indice_robot) {
174
175     if (indice_robot>0) {
176         std::ostringstream sstr;
177         sstr << "K3_leftWheelMotor#" << indice_robot -1;
178         leftmotor = sstr.str();
179         sstr.str("");
180         sstr.clear();
181         sstr << "K3_rightWheelMotor#" << indice_robot -1;
182         rightmotor = sstr.str();
183     } else {
184         leftmotor = "K3_leftWheelMotor#";
185         rightmotor = "K3_rightWheelMotor#";
186     }
187
188 }
189 //establece la velocidad (multiplo de pi) con el argumento dado.
190 void Demo::setVelocidad(int multiplicador) {
191     velocidad=multiplicador*pi;
192 }
```

## 2.7. Clase Control

Clase que implementa la clase abstracta I_control.

Se da como plantilla al usuario, éste podrá modificarla o emplear otra clase siempre que ésta implemente la clase abstracta I_control.

```
1  //Definición de Clase Plantilla de Código de Control (puede ser
       cualquiera con tal de que implemente I_control)
2  #ifndef CONTROL_H
3  #define CONTROL_H
4  #include <i_control.h>
5
6  //La clase implementa la "interfaz" (clase abstracta) i_control
7  class Control: public I_Control
8  {
9      public:
10        int control(int clientID);
11        int interrupt_1(int clientID);
12        int interrupt_2(int clientID);
13        int interrupt_3(int clientID);
14        int interrupt_4(int clientID);
15 };
16
17 #endif // CONTROL_H
```

## 2.8. control.cpp

Plantilla que se da al usuario para programar las 5 funciones definidas en control.h

```
1  #include <control.h>
2  extern "C" {
3  #include "extApi.h"
4  }
5
6
7
8
9  int Control::control(int clientID) {
10     //LLAMADA DESDE EJECUTAR (CONTROL PRINCIPAL)
11
12     return 0;
13 }
14
15 int Control::interrupt_1(int clientID){
16     //LLAMADA DESDE BOTON DE INTERRUPCION 1
17
18     return 0;
19 }
20
21 int Control::interrupt_2(int clientID){
```

```
22      //LLAMADA DESDE BOTON DE INTERRUPCION 2
23
24      return 0;
25  }
26
27  int Control::interrupt_3(int clientID){
28      //LLAMADA DESDE BOTON DE INTERRUPCION 3
29
30      return 0;
31  }
32
33  int Control::interrupt_4(int clientID){
34      //LLAMADA DESDE BOTON DE INTERRUPCION 4
35
36      return 0;
37  }
```

## 2.9. kheperasimgui.h

En el archivo kheperasimgui.h definiremos el nombre de los métodos, así como sus argumentos y el valor de retorno a implementar en el kheperasimgui.cpp. Mientras que en éste último daremos cuerpo a dichos métodos e implementaremos la funcionalidad de éstos.

La clase KEPHERASIMGUI es aquella que definirá la interfaz de usuario. En ella estarán contenidos todos los botones y elementos de interacción entre el usuario y la aplicación. Se implementarán las funciones que ejecutarán al interactuar con los botones de la interfaz. Se incluyen también las librerías que serán utilizadas en el programa .cpp:
QMainWindows: Nos proropociona las herramientas para generar la interfaz gráfica Qt-Creator.

El resto de includes nos permiten utilizar las clases definidas anteriormente.

```
1   //Definición de la clase de la Interfaz Gráfica.
2   #ifndef KHEPERASIMGUI_H
3   #define KHEPERASIMGUI_H
4   #include <QMainWindow>
5   #include <demo.h>
6   #include <control.h>
7   #include <i_control.h>
8   #include <stdio.h>
9
10
11  namespace Ui {
12  class KheperaSimGUI;
13  }
14
15  class KheperaSimGUI : public QMainWindow
```

```
16 {
17     Q_OBJECT
18
19 public:
20
21     explicit KheperaSimGUI(QWidget *parent = 0);
22     ~KheperaSimGUI();
23     typedef int (I_Control::* ptrfunc) (int);
24
25 private slots:
26     //inicia la simulación remotamente (IP:parámetro. Puerto:
          19997, predeterminado de V-REP)
27     void iniciar_sim(std::string ip);
28     //abre una conexión remota nueva en la ip y puerto dada
29     int conectar(std::string ip,int puerto);
30     //ejecuta la función hilo como un thread y lo desvincula del
          padre-> detach().
31     void codigo_en_hilo(KheperaSimGUI::ptrfunc ptrfuncion);
32     //muestra los datos de posicion y velocidad en la GUI.
33     void refrescar_datos();
34     //Slots/Métodos que se cargan en los eventos definidos por
          sus nombres.
35     void on_ejecutar_clicked();
36
37     void on_sim_clicked();
38
39     void on_parar_clicked();
40
41     void on_pausar_clicked();
42
43     void on_interrupt1_clicked();
44
45     void on_interrupt2_clicked();
46
47     void on_interrupt3_clicked();
48
49     void on_interrupt4_clicked();
50
51     void on_controlset_clicked();
52
53     void on_demoset_clicked();
54
55     void on_comboBox_2_activated(QString robottexto);
56
57     void on_comboBox_3_activated(const QString a);
58
59     void on_comboBox_activated(int index);
60
```

```
61      void on_actionSalir_triggered();
62
63      void on_actionMostrar_Output_toggled(bool arg1);
64
65      void closeEvent(QCloseEvent *bar);
66
67      void on_actionAyuda_triggered();
68
69  private:
70
71      Ui::KheperaSimGUI *ui;
72      //Handle/identificador usado para conexiones de señales de
           manejo de simulación (Iniciar, Parar, Pausar/Reanudar)
73      //Usa siempre el puerto 19997, inicializado por defecto en V-
           REP
74      int clientIDsim=-1;
75      //Handle/identificador usado para conexiones de señales de
           control/interrupcion.
76      //Usa el puerto definido por el usuario (debe abrirse
           manualmente con un Script Lua. Ver modelo Khepera incluido)
           .
77      int clientIDexe=-1;
78      //IP del Host Simulador. Localhost por defecto.
79      std::string ip="127.0.0.1";
80      //Puerto de escucha. Debe configurarse en un Script Lua. Ver
           modelo Khepera incluido.
81      int puerto=20001;
82      // buffer de coordenadas de posición.
83      float position[3];
84      // buffer de módulos de velocidad (lineal).
85      float velocidad[3];
86      //toggle de la advertencia de velocidad excesiva.
87      bool nomostrarwarning =false;
88
89      FILE * log;
90
91      Demo demo;
92
93      Control control;
94
95
96
97  };
98
99  #endif // KHEPERASIMGUI_H
```

## 2.10. kheperasimgui.cpp

Como se dijo en el apartado anterior, aquí se dará cuerpo a los métodos y se implementará la funcionalidad de éstos:

*on_ejecutar_clicked*: Establece la conexión con el host simulador y ejecuta el código de control.
*on_sim_clicked*: Inicia la simulación remotamente.
*on_parar_clicked*: Para la simulación remotamente.
*on_pausar_clicked*: Pausa o reanuda la simulación remotamente.
*interrupt_X*: Ejecutan el código de interrupción correspondiente a su enumeración.
*on_comboBox_activated*: Es el menú que permite modificar la velocidad a la que se moverá el robot en las interrupciones.
*on_comboBox_(2/3)_activated*: Permiten añadir nombres nuevos predeterminados de robots a la lista desplegable de selección de robots a interrumpir o monitorizar. Tendremos también dentro de ese menú desplegable la posibilidad de introducir un nombre personalizado, en el cual al pulsar el ítem personalizado nos aparecerá un diálogo para introducir el nuevo nombre (solo en el caso de monitorización de velocidad y posición, no interrupción).
*refrescar_datos*: En la interfaz se pueden apreciar 6 LCD virtuales los cuales nos mostrarán la posición y velocidad del Khepera virtual. Éstos serán actualizados mediante ésta función (se ejecuta cada 150 ms debido a la llamada desde un timer).
*codigo_en_hilo*: Permite la ejecución concurrente de la GUI y los métodos de control, cargando dichos métodos como hilos (threads del stdC++11) independientes.
*conectar*: Establece una conexión con V-REP en el host simulador, localizado en la IP y puerto dados.
*iniciar_sim*: Establece una conexión con V-REP en el host simulador, localizado en la IP dada y el puerto 19997 (abierto por defecto por V-REP).
El resto de métodos son sencillos y están comentados en el código a continuación.

```
1  #include "kheperasimgui.h"
2  #include "ui_kheperasimgui.h"
3  #include <thread>
4  #include <QMessageBox>
5  #include <QTimer>
6  #include <QInputDialog>
7  #include <iostream>
8  #include <i_control.h>
9  #include "control.cpp"
10 #include "demo.cpp"
11 #include <QTextStream>
12 #include <QScrollBar>
13 #include <QUrl>
14 #include <QDesktopServices>
15 extern "C" {
16 #include "extApi.h"
17 }
18
```

```
19 using namespace std;
20 //Constructor del objeto GUI
21 KheperaSimGUI::KheperaSimGUI(QWidget *parent) :
22     QMainWindow(parent),
23     ui(new Ui::KheperaSimGUI)
24 {
25     ui->setupUi(this);
26     log = freopen ("log","w",stdout);
27 
28     //Define un temporizador que ejecutar refrescar_datos() cada
          150ms
29     QTimer *timer = new QTimer(this);
30          connect(timer, SIGNAL(timeout()), this, SLOT(
               refrescar_datos()));
31          //inicia el temporizador
32          timer->start(150);
33 
34 }
35 //Destructor del objeto GUI
36 KheperaSimGUI::~KheperaSimGUI()
37 {
38     delete ui;
39 }
40 
41 
42 //Abre una conexión con V-REP (puerto por defecto 19997), Inicia
       la simulacion y cierra la conexión.
43 void KheperaSimGUI::iniciar_sim(string ip){
44     //Comprueba que no haya una conexión ya abierta antes de
          conectar.
45     if (simxGetConnectionId(clientIDsim)==-1){
46          clientIDsim=simxStart((simxChar*) ip.c_str(),19997,true,
               true,2000,5);
47     }
48 
49     //Comprueba que se haya realizado la conexión con éxito. Si
          no, muestra error.
50     if (clientIDsim!=-1)
51     {
52          simxStartSimulation(clientIDsim,simx_opmode_oneshot_wait)
               ;
53          simxFinish(clientIDsim);
54          ui->pausar->setEnabled(true);
55          cout << "Gui: Simulación iniciada\n";
56 
57     } else {
58          cout << "Error: Imposible iniciar simulación remotamente.
               (Pulsar Play en V-REP y volver a intentar)\n";
```

```
59          QMessageBox error;
60          error.setText("Error: Imposible iniciar simulación
                remotamente. (Pulsar Play en V-REP y volver a intentar)
                ");
61          error.exec();
62      }
63  }
64  //Abre una conexión con la escena V-REP, en la IP y puerto dado.
65  int KheperaSimGUI::conectar(string ip,int puerto){
66
67      clientIDexe=simxStart((simxChar*) ip.c_str(),puerto,true,true
            ,2000,5);
68      //Comprueba la conexión, si ha fallado, muestra error.
69      if (simxGetConnectionId(clientIDexe)==-1) {
70          cout << "Error: Imposible conectar con el servidor.
                Asegúrese de que el puerto es correcto (" << ui->
                puertotexto->text().toInt() <<")\n";
71          cout << "Debe ejecutar un proceso servidor en el Host
                Simulador, normalmente con una llamada a\n";
72          cout << "simExtRemoteApiStart(puerto) en un Child Script
                de Lua.\n";
73          cout << "Ver código de control demo y escena demo2khepera.
                ttt\n";
74          QMessageBox error;
75          error.setText("Error: Imposible conectar con el servidor")
                ;
76          error.exec();
77      }
78       return clientIDexe;
79  }
80
81  //crea y lanza un thread (hilo) con el metodo pedido, para
        permitir funcionamiento concurrente.
82
83  void KheperaSimGUI::codigo_en_hilo(ptrfunc ptrfuncion)
84  {
85      if (ui->demoset->isChecked()){
86          std::thread hilo(ptrfuncion,&demo,clientIDexe);
87          //el hilo continua ejecutándose hasta que retorna la
                función hilo (el hilo padre se "desprende" del hijo)
88          hilo.detach();
89      } else {
90          std::thread hilo(ptrfuncion,&control,clientIDexe);
91          hilo.detach();
92      }
93  }
94
95  //refresca los datos de posición y velocidad de la gui. La invoca
```

```
    el timer creado en el constructor de la GUI.
96  void KheperaSimGUI::refrescar_datos()
97  {       //Comprueba que haya una conexión. Si la hay, obtiene los
        datos. (Ver Doc. Remote API para entender funciones)
98          if (simxGetConnectionId(clientIDexe)!=-1) {
99              int robotHandle;
100             //indica de que robot se representarán los datos,
                    segun el combobox de selección.
101             string robot = ui->comboBox_2->currentText().toUtf8()
                    .constData();
102             simxGetObjectHandle(clientIDexe,robot.c_str(),&
                    robotHandle,simx_opmode_oneshot_wait);
103             simxGetObjectPosition(clientIDexe,robotHandle,-1,
                    position,simx_opmode_oneshot);
104             simxGetObjectVelocity(clientIDexe,robotHandle,
                    velocidad,NULL,simx_opmode_oneshot);
105         } else {
106             //Si no hay conexión, pone a 0 los valores.
107             for (int i = 0; i<3; i++){
108                 position[i]=0;
109                 velocidad[i]=0;
110             }
111         }
112         //actualiza los "LCD" de la GUI
113         ui->xlcd->display(position[0]);
114         ui->ylcd->display(position[1]);
115         ui->zlcd->display(position[2]);
116         ui->vxlcd->display(velocidad[0]);
117         ui->vylcd->display(velocidad[1]);
118         ui->vzlcd->display(velocidad[2]);
119         //Muestra el contenido del archivo log en el textEdit.
                Solo si "mostrar output" está selecc.
120         if (ui->actionMostrar_Output->isChecked()) {
121             fclose(log);
122             log = fopen("log","r");
123             QTextStream stream(log);
124             QString str = stream.readAll();
125             ui->textEdit->setText(str);
126             QScrollBar *sb = ui->textEdit->verticalScrollBar();
127             sb->setValue(sb->maximum());
128             fclose(log);
129             //redirecciona stdout al archivo log. ver página man
                    de freopen.
130             log = freopen ("log","a",stdout);
131         }
132
133
134 }
```

```
135
136 //código de evento: pulsar botón ejecutar.
137 void KheperaSimGUI::on_ejecutar_clicked()
138 {   //define la IP y puerto desde los textbox.
139     ip = ui->iptexto->text().toUtf8().constData();
140     puerto = ui->puertotexto->text().toInt();
141     //si está marcada la opción de Inicio remoto, inicia la sim.
          remotamente.
142     if(ui->inicioremotoset->isChecked()){
143         iniciar_sim(ip);
144     }
145     //cierra cualquier conexión de control/interrupcion que haya.
146     simxFinish(clientIDexe);
147     //conecta y ejecuta el código de control (Demo ó Control) en
          un hilo independiente.
148     if (conectar(ip,puerto) != -1) {
149         ptrfunc ptrfuncion = &I_Control::control;
150         codigo_en_hilo(ptrfuncion);
151         cout << "Gui: Control ejecutado\n";
152     }
153
154
155 }
156 //código de evento: pulsar botón iniciar simulación.
157 void KheperaSimGUI::on_sim_clicked()
158 {
159     iniciar_sim(ui->iptexto->text().toUtf8().constData());
160
161 }
162 //código de evento: pulsar botón parar simulación.
163 void KheperaSimGUI::on_parar_clicked()
164 {
165     //abre una conexión en el puerto por defecto, para la
          simulación y cierra la conexión.
166     clientIDsim=simxStart((simxChar*) ip.c_str(),19997,true,true
          ,2000,5);
167     simxStopSimulation(clientIDsim,simx_opmode_oneshot_wait);
168     simxFinish(clientIDsim);
169     cout << "Gui: Simulación parada\n";
170     ui->pausar->setText("Pausar Sim.");
171     ui->pausar->setEnabled(false);
172
173 }
174 //código de evento: pulsar boton pausar/reanudar sim.
175 void KheperaSimGUI::on_pausar_clicked()
176 {   //obtiene el estado actual del botón: Pausar/Reanudar.
177     string texto = ui->pausar->text().toUtf8().constData();
178     //abre una conexión en el puerto por defecto.
```

```
179     clientIDsim=simxStart((simxChar*) ip.c_str(),19997,true,true
            ,2000,5);
180     //si tiene que pausar, pausa, y se pone en estado reanudar.
            Ídem al contrario.
181     if (!texto.compare("Pausar␣Sim.")) {
182         simxPauseSimulation(clientIDsim,simx_opmode_oneshot_wait)
                ;
183         cout << "Gui:␣Simulación␣pausada\n";
184         ui->pausar->setText("Reanud␣Sim.");
185     } else {
186         simxStartSimulation(clientIDsim,simx_opmode_oneshot_wait)
                ;
187         cout << "Gui:␣Simulación␣reanudada\n";
188         ui->pausar->setText("Pausar␣Sim.");
189     }
190     //cierra la conexión en puerto por defecto.
191     simxFinish(clientIDsim);
192
193
194 }
195 //código de evento: pulsar botón de interrupt X.
196 void KheperaSimGUI::on_interrupt1_clicked()
197 {   //cierra cualquier conexión de control/interrupcion que
        exista,conecta
198     //y ejecuta el código de interrupción (Demo ó Control) en un
            hilo independiente.
199     simxFinish(clientIDexe);
200     if (conectar(ip,puerto) != -1) {
201         //crea un puntero a metodo miembro con el metodo
                seleccionado.
202         ptrfunc ptrfuncion = &I_Control::interrupt_1;
203         //ejecuta el metodo seleccionado en un hilo.
204         codigo_en_hilo(ptrfuncion);
205     }
206 }
207
208 void KheperaSimGUI::on_interrupt2_clicked()
209 {
210     simxFinish(clientIDexe);
211     if (conectar(ip,puerto) != -1) {
212         ptrfunc ptrfuncion = &I_Control::interrupt_2;
213         codigo_en_hilo(ptrfuncion);
214     }
215
216 }
217
218 void KheperaSimGUI::on_interrupt3_clicked()
219 {
```

```
220     simxFinish(clientIDexe);
221     if (conectar(ip,puerto) != -1) {
222         ptrfunc ptrfuncion = &I_Control::interrupt_3;
223         codigo_en_hilo(ptrfuncion);
224     }
225 }
226 void KheperaSimGUI::on_interrupt4_clicked()
227 {
228     simxFinish(clientIDexe);
229     if (conectar(ip,puerto) != -1) {
230         ptrfunc ptrfuncion = &I_Control::interrupt_4;
231         codigo_en_hilo(ptrfuncion);
232     }
233 }
234 //código de evento: selección de nombre de robot para mostrar sus
        datos.
235 void KheperaSimGUI::on_comboBox_2_activated(QString robottexto)
236 {   //Si se selecciona Personalizado, abre un dialogo para
        introducir un nuevo nombre.
237     //string texto = robottexto.toUtf8().constData();
238     if (!robottexto.compare("Personalizado...")){
239         QString nombrerobot = QInputDialog::getText(this,tr("
              Nombre␣Nuevo"),tr("Nombre␣robot:"));
240         //Comprueba que no se canceló el dialogo, añade la nueva
              opción a la lista y la selecciona.
241         if (nombrerobot!=NULL) {
242           ui->comboBox_2->insertItem(0,nombrerobot);
243           ui->comboBox_2->setCurrentIndex(0);
244         }
245     } else if (!robottexto.compare("Añadir")){
246         std::ostringstream sstr;
247         sstr << "K3_robot#" << ui->comboBox_2->count() -3;
248         QString str = QString::fromStdString(sstr.str());
249         ui->comboBox_2->insertItem(ui->comboBox_2->count()-2,str)
              ;
250         ui->comboBox_2->setCurrentIndex(ui->comboBox_2->count()
              -3);
251     }
252 }
253 //cambia los nombres de los botones de interrupcion al modo
        Control.
254 void KheperaSimGUI::on_controlset_clicked()
255 {
256     ui->interrupt1->setText("Interrupt␣1");
257     ui->interrupt2->setText("Interrupt␣2");
258     ui->interrupt3->setText("Interrupt␣3");
259     ui->interrupt4->setText("Interrupt␣4");
260     ui->comboBox_3->setEnabled(false);
```

```
261     ui->label_13->setEnabled(false);
262     ui->comboBox->setEnabled(false);
263 }
264 //revierte los botones de interrupcion al modo demo.
265 void KheperaSimGUI::on_demoset_clicked()
266 {
267     ui->interrupt1->setText("Adelante");
268     ui->interrupt2->setText("Atrás");
269     ui->interrupt3->setText("Izquierda");
270     ui->interrupt4->setText("Derecha");
271     ui->comboBox_3->setEnabled(true);
272     ui->label_13->setEnabled(true);
273     ui->comboBox->setEnabled(true);
274 }
275 //permite cambiar el robot al que se envian las interrupciones.
    Solo modo demo.(En modo control el usuario puede elegir con su
    código cualquier robot/acción)
276 void KheperaSimGUI::on_comboBox_3_activated(const QString a)
277 {   string texto = a.toUtf8().constData();
278     if (!texto.compare("Añadir")){
279         std::ostringstream sstr;
280         sstr << "K3_robot#" << ui->comboBox_3->count() -2;
281         QString str = QString::fromStdString(sstr.str());
282         ui->comboBox_3->insertItem(ui->comboBox_3->count()-1,str)
            ;
283         ui->comboBox_3->setCurrentIndex(ui->comboBox_3->count()
            -2);
284     }
285     demo.setRobot(ui->comboBox_3->currentIndex());
286 }
287
288 //permite cambiar la velocidad de los motores en las
    interrupciones. Solo modo Demo. Advierte que el robot se
    desestabiliza para altas velocidades.
289 void KheperaSimGUI::on_comboBox_activated(int index)
290 {   //comprueba velocidad > 2x y que no se ha especificado que no
    se muestre la advertencia.
291     if (index>1 && !nomostrarwarning){
292         QMessageBox warning;
293         QCheckBox nomostrar;
294         nomostrar.setText("No␣volver␣a␣mostrar");
295         warning.setText("Para␣V␣>␣2x,␣el␣control␣no␣asegura␣la␣
            estabilidad␣del␣robot");
296         warning.setCheckBox(&nomostrar);
297         warning.exec();
298         if (warning.checkBox()->isChecked()){
299             nomostrarwarning = true;
300         }
```

```
301         }
302     demo.setVelocidad(index+1);
303 }
304 //codigo de cierre por menú
305 void KheperaSimGUI::on_actionSalir_triggered()
306 {
307     fclose(log);
308     exit(0);
309 }
310 //muestra el output (hace la ventana mas grande) o lo oculta (
    ventana mas pequeña)
311 void KheperaSimGUI::on_actionMostrar_Output_toggled(bool arg1)
312 {
313     if (arg1) {
314           setFixedSize(450,410);
315     } else {
316           setFixedSize(450,250);
317     }
318 }
319 //código de cierre por pulsar X en ventana.
320 void KheperaSimGUI::closeEvent(QCloseEvent *bar) {
321
322     fclose(log);
323
324 }
325
326 void KheperaSimGUI::on_actionAyuda_triggered()
327 {
328     QDesktopServices::openUrl(QUrl("http://xpeiro.github.io/
        compu3/"));
329 }
```

## 2.11. kheperasimgui.ui

Es la interfaz gráfica creada con QtCreator. Con ella se define el aspecto visual de kheperasimgui.cpp.

## 2.12. main.cpp

Es la función principal, en la cual se inicializa la librería Qt y se define la instancia de la clase kheperasimgui.h que tendrá implementada la interfaz gráfica con un tamaño prefijado de 425x275 px.

```
1 #include "kheperasimgui.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
```

```
5  {    //Instancia la GUI
6       QApplication a(argc, argv);
7       KheperaSimGUI w;
8       //muestra la GUI
9       w.show();
10      //Fija el tamaño de la ventana.
11      w.setFixedSize(450,275);
12
13      return a.exec();
14 }
```

# 3. Escena: demo2khepera.ttt

Escena de V-rep que contiene dos robots del modelo kh3_noplugin.ttm, ya configurada para probar el modo Demo.

# 4. Modelo: kh3_noplugin.ttm

Modelo del Khepera que se ha de cargar en el simulador, no se debe utilizar el modelo de Khepera por defecto pues las instrucciones del software están preparadas para éste en concreto.