

ENSEM Mobile Robots Testbed User's Guide

Lucas Campos Pires

Janvier 2010

1 Preface

This document is a quick user guide for experiments using ENSEM testbed. Basically, it shows how to set up an environment and program a controller using the libraries developed, here denoted **testbed middleware**. The middleware was developed in C++ language, under the object oriented paradigm, therefore a minimal knowledge in these items is required. All middleware sources are provided and its modification to fit specific necessities is highly encouraged for more experienced users. For more information about the project and specification of the middleware, which may be helpful in the modifications, please see the final project document.

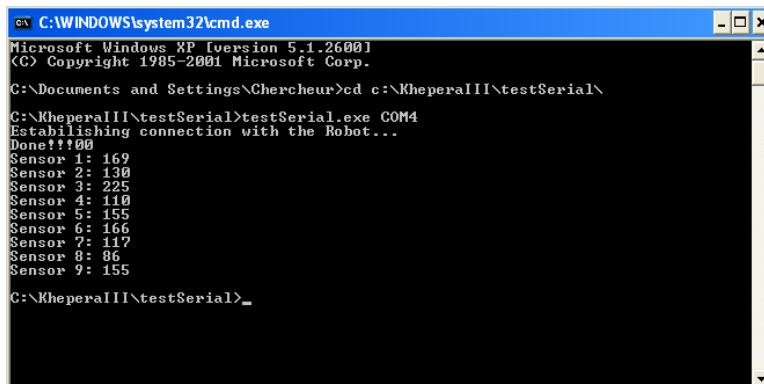
2 Getting Started with KheperaIII

The steps done in this section, except from sub-section 2.3, must be performed just in the first time you configure a computer to work with KheperaIII. If you plan to work only with matlab to program the controllers, perform just section 2.1. In other cases, you must do sections 2.1, 2.2 and 2.3.

2.1 Configuring Bluetooth

The first step to establish a connection with the robot is to configure the bluetooth link. In the end, we will have a serial port over bluetooth.

- Activate your Bluetooth service;
- Turn on ONE Khepera Robot, and see a green led active. If a red led is also active, you should first recharge the battery;
- Start a device search in your PC;
- Your search should encounter the robot with the following id: KHIII XXXXX, where XXXXX is a serial number that identifies each Khepera robot. The serial number is also placed in the bottom of the robot;



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Chercheur>cd c:\KheperaIII\testSerial\
C:\KheperaIII\testSerial>testSerial.exe COM4
Establishing connection with the Robot...
Done!!00
Sensor 1: 169
Sensor 2: 130
Sensor 3: 225
Sensor 4: 110
Sensor 5: 155
Sensor 6: 166
Sensor 7: 117
Sensor 8: 86
Sensor 9: 155
C:\KheperaIII\testSerial>_
```

Figure 1: Command line running testSerial.exe

- Create a connection with the dispositive using "0000" as security code.
- Verify if your system has assigned a serial port to the robot. In case of two ports assigned, use the first one.

- The serial parameters are software configurable as: 115200bps, 8 Data bits, 1 stop bit, no parity, no hardware control.
- Run the provided test program "testSerial.exe" in the command line¹ indicating the serial port², as presented in figure 1, to evaluate the connection. You should see the infrared sensors output printed in the command line.
- Put some reflective object close to some of the robot sensors and run the program again. You will see as the measures change.

In case of any problems, try reading KheperaIII user Manual for more details.

2.2 Installing Development Tools

The softwares used here are highly encouraged for development, specially for users that are not too experienced with c++. It is important also to take care with the softwares versions, which may not be compatible with the rest of the project.

2.2.1 Installing Visual C++

The development environment and compiler can be chosen by the user. Yet, it is recommended Visual C++, specially for users not experienced with linking issues, since it is easier to integrate with additional libraries.

- Download Visual C++ 2008 Express edition from:
<http://www.microsoft.com/express/Downloads/#webInstall>
- Run the installer and follow the instructions.

2.2.2 Installing Boost Libraries

The installation of boost libraries is MANDATORY. These libraries are used in the communication modules of the middleware.

- Get boost installer from: <http://www.boostpro.com/download>. The version used in this project was: 1.40.0;
- Run the installer and select beyond the default ones, the following libraries: **DateTime**, **Regex**, **System** and **Thread**, as presented in figures 2 and 3.

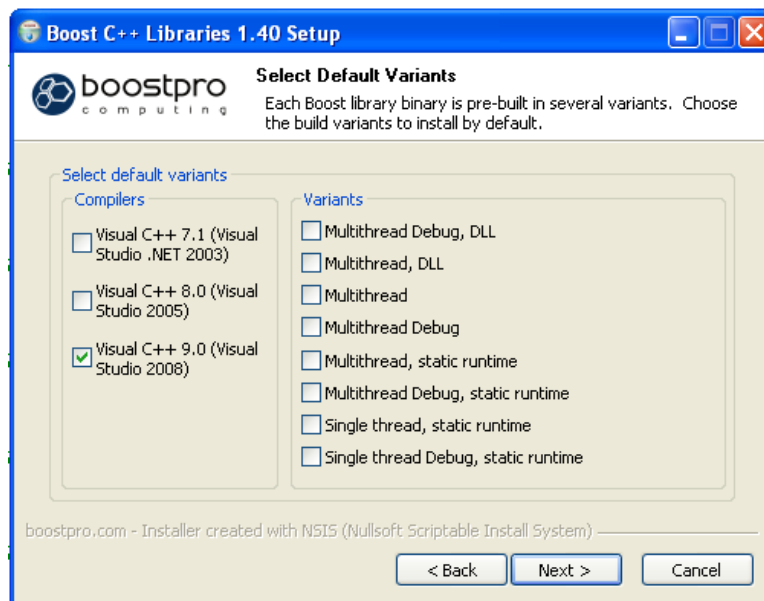


Figure 2: Boost installation; compilers.

- Select just VC9.0 to reduce the download size.
- Follow the instructions.

¹Windows: Start ->Execute ->'cmd' ->Ok

²...\KheperaIII_user_package\testSerial\testSerial.exe <PORT>

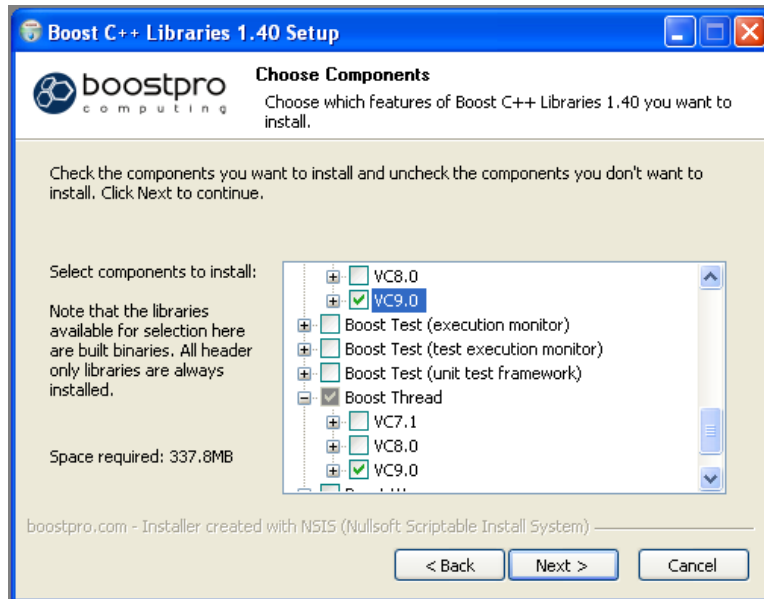


Figure 3: Boost installation; libraries.

2.2.3 Integrating Boost and Visual C++

This proceeding is necessary to develop your controller from visual c++.

- Locate boost installation directory. Something like:
C:\Program Files\boost\boost_1.40. We are going to refer it as BOOST.ROOT.
- Start Visual C++.
- From **Tools** menu, open **Options** window.
- In the left, select: **Projects and Solutions: VC++ Directories**.
- Below **Show directories for:**, select in the combo box: **Include files**.
- Click in the icon **New line** and enter the **BOOST_ROOT** directory in the new field, as in figure 4.

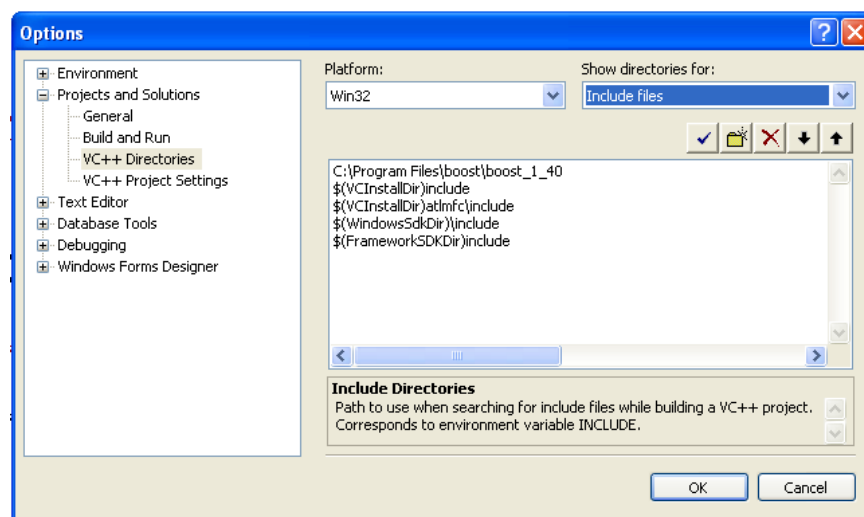


Figure 4: Visual c++ and boost integration.

- Now, select in the combo box **Library files**.

- As done before, include a new entry with **BOOST_ROOT\lib**
- That's it, we are ready to program our first controller.

2.3 Programming a Controller

Here we start to program a new controller using testbed middleware in Visual C++.

- Start Visual C++ and create a new project in **File ->New ->Project...**
- Select **Win32 ->Win32 Console Application**. Choose a name and a directory for the project. Also, clear **Create directory for solution** check box, and click **Ok**.
- In the following window press **Next**, check **Empty project** and press **Finish**.
- Copy the middleware source folder **KheperaIII_src** to the project root directory.
- Inside Visual C++, in the project tree, add a new filter inside **Source** and name it **KheperaIII_src**.
- Add to the created filter **ALL** the files inside **KheperaIII_src** folder, as shown in figure 5. If a pop-up asks to create a custom rule to build..., click **NO**.
- Left-Click in **Source** and add a new item. Select **.cpp** file and name it **main.cpp**.
- Type the code above, replacing (COMX) with the port assigned to the robot.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

#include "KheperaIII_src\KheperaIII.h"

int main(int argc, char **argv)
{
    KheperaIII *robot = new KheperaIII("COMX");
    robot->setVelocity(0,2);
    Sleep(2000);
    robot->setVelocity(0,0);
    printf("Press to continue...");
    _getch();
}
```

- In solution configuration combo box, located in the standard visual c++ toolbar, change **Debug** by **Release**.
- Compile and run.

You should see the robot turn for 2 seconds and than stop. In case of any problems, try to redo the previous steps or see the documentation of the used tools. If your linker complains about a previous definition of "WinSock.h", include <winsock2.h> before the other libraries.

Try to access other khepera methods that don't use communication and localization modules which will be explained later. Try, for instance: **int* getIrOutput()** and **int* getEncodersValue()**.

3 Getting Started with Motion Analysis

Here, we will present all the steps to create a project in Cortex³, calibrate it, define a robot and retrieve the information in other computer.

3.1 Setting up a Project

- Open cortex software.
- Create a new folder to save your project.
- Load a previous project and "save as" in your newly created folder.
- In **Layouts** menu, choose **2 panes**.
- Click in the upper plane and pres **F3**, for the 3-D view, which should be default.
- In the lower plane, press **F2** and click in **All On** at the left bottom of the software. These are the 2-D view of each camera.
- Click in **Connect to Cameras**. A pop-up will present the number of cameras found. Your program should look like figure 6.

³Motion Analysis software

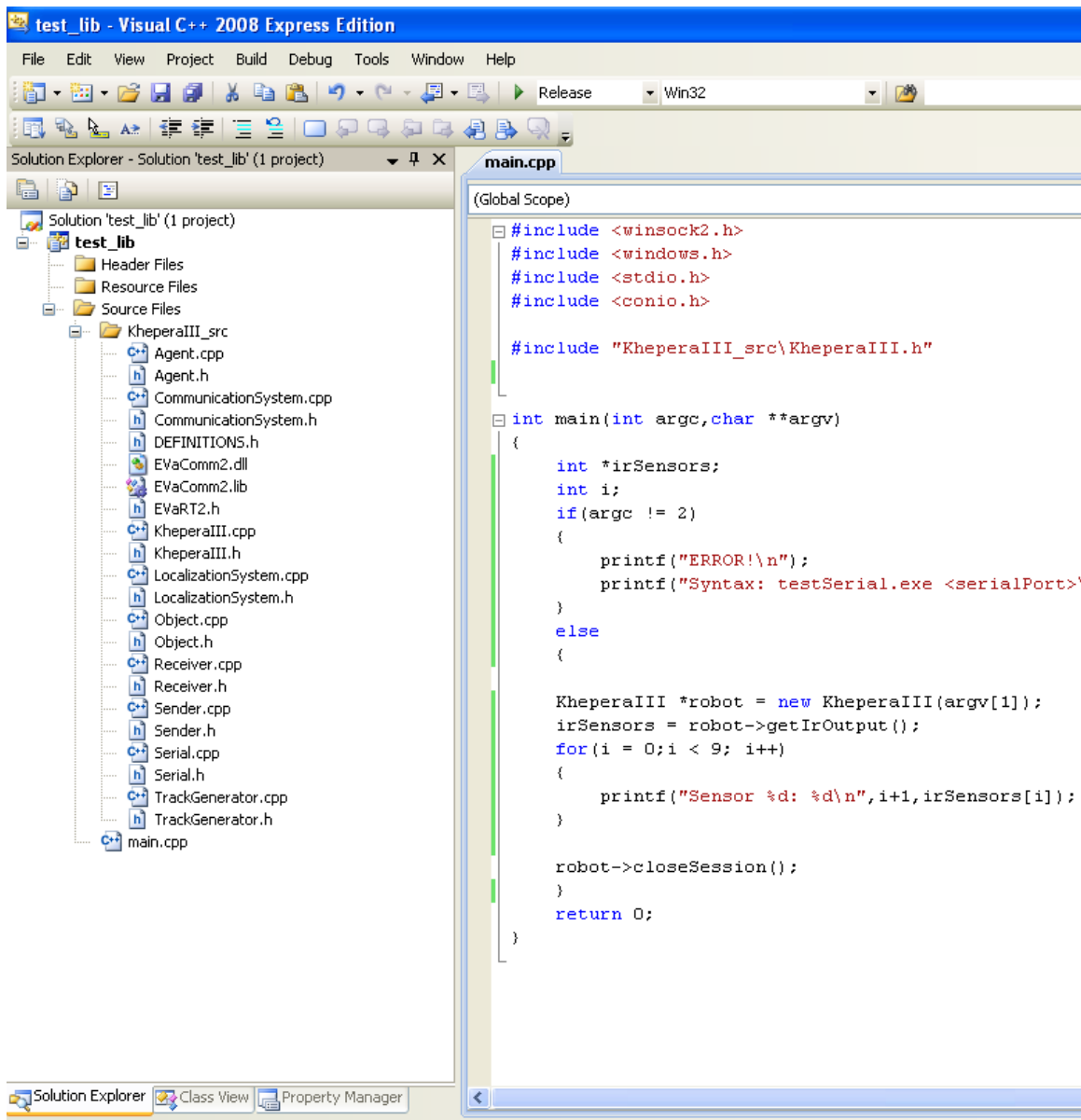


Figure 5: Visual c++ project tree.

The next steps are dedicated to calibrate the system.

- Place the "L" on the floor. This "L" defines the origin of the coordinate system. The side with three markers is the "X" axis and the one with two markers is "Y" axis.
- Click in the **Calibration** tab and mark **Camera Aiming** under **Calibrate** at the right top.
- Press the **Run** button in the right bottom of the software.
- Notice the four markers in the displays. Notice also that all camera numbers that are at the bottom of the software turn to yellow.
- If more than four markers appears in any 2-D display or some camera does not see the four markers try changing the position of the cameras or their configuration of **bright** and **threshold** under **Settings**....
- If any extra marker still appears in any 2-D display, you can mask it by clicking **Pause** and with the middle button of the mouse drag a box over it.
- All set, click in **Collect and Calibrate** in **Calibration with L-Frame** box.
- Now, it is time to optimize the camera positioning and define our working area.

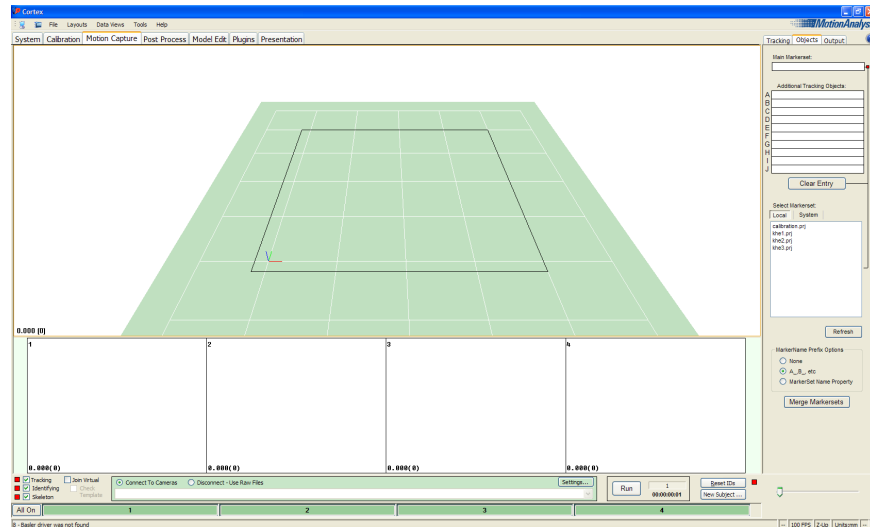


Figure 6: Cortex with 2 pane view.

- Right-click in the 3-D view and mark **Cam Field-of-View** and **Volume**.
- Define the work volume in **Tools->Settings->Calibration->Capture Volume**
- Verify if ALL cameras cover the whole working volume. If not, relocate the cameras to do so, as shown in figure 7.

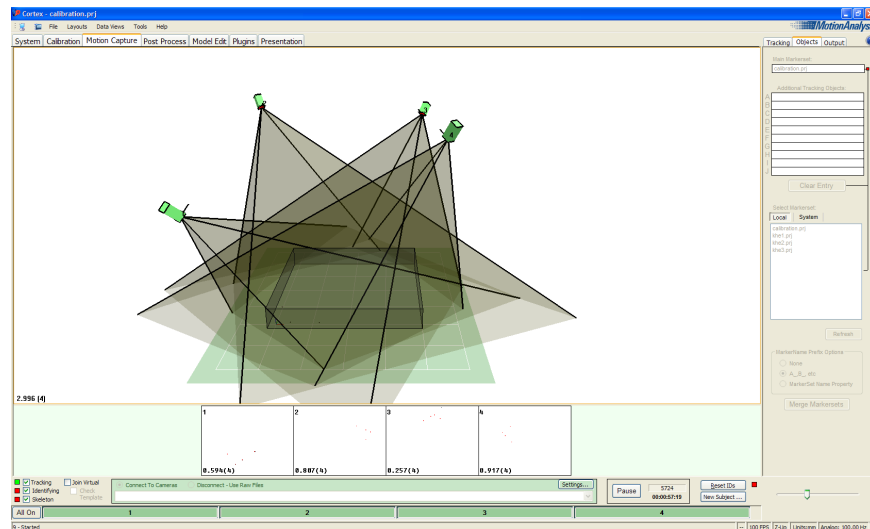


Figure 7: Cortex camera field-of-view.

- After that, press once again **Collect and Calibrate** in **Calibration with L-Frame** box. **Overwrite** it, if necessary.
- Next step: calibration with wand.
- First define the wand size in **Calibration with Wand** to 200 or 500 depending the wand you are using. Also set the duration in 100 seconds.
- Remove the "L" of the floor, and hide ALL markers in the room.
- Click in **Collect and Calibrate** in **Calibration with Wand** box and wave the wand in the whole working volume. Keep doing it during the whole capture time.
- In the end of the capture time a window will appear saying if extra markers were seen by one of the cameras during the capture.

- The next window will show the results of the calibration. If the parameters are acceptable click in **Accept**.
- If, for example, the wand length is totally wrong, probably there were more than three markers during the capture. See the 2-D display, mask any extra marker and do the process again.
- SAVE YOUR PROJECT!!!

3.2 Placing markers in the Robot

Continuing, now it is time to set up an object: Your Robot! Primarily you must position the markers in the Robot.

- You **MUST** use **FOUR** markers. Use the bigger ones.
- Place one in the front.
- Place one in the middle.
- Place the other two wherever you want.
- Try to avoid singularities, i.e., equidistant markers.
- You can use little blocks to higher your markers also. This may help the software.

3.3 Robot in Cortex

Identifying your robot in Cortex.

- First, to not loose your project settings, save your project with the name of the Object you want to define. Ex: Khe_1.prj.
- Position the robot in the floor and select **Motion Capture** tab. Press **Run** and verify if all markers are visible.
- At right, select **Output** tab and mark **Raw video** and **Tracked binary**.
- Give a name to it, set a duration of 5 seconds and click in **RECORD**.
- Change to **Model Edit** tab.
- Under **File->Load Tracks File...** and select the record you have just done. You will notice a change in the screen.
- In the right, define four markers names **EXACTLY** as follows, respecting even the order. Only the names of the offset markers can be modified.
 - Middle
 - Offset1
 - Front
 - Offset2
- Click in **Quick ID** and correctly identify the markers in the graphic.
- You can create links between the markers by clicking in **Create Linkages for Template** and drawing a straight between two markers. In the end, your model will look like figure 8.
- Click in **Template Create**, define the name (Ex: Khe_1) and click in **Create Template**.
- Save your project.

3.4 Retrieving Information

Now, you must validate the data you defined and the communication with Cortex.

- Load the project after Calibration.
- Connect to cameras.
- Select **Motion Capture** tab and in the right, the **Objects** tab.
- Select one box bellow **Additional Tracking Objects** and choose the object you created in the list bellow **Select Markerset**.
- Click **Run** and, there it is, your robot is identified, like in figure 9.
- Run in a computer inside the LAN the give program **ClientTest.exe** <localIP> <CortexIP>, like in figure 10⁴.
- Tape "2" and "Q".
- In the program folder you will find a file "FrameOfData.txt", containing the name of the object you have defined and the position of the markers. In figure 11 there is an example of this file.
- After that, you are all set.

⁴type **ipconfig** in the command line to retrieve the machine IP

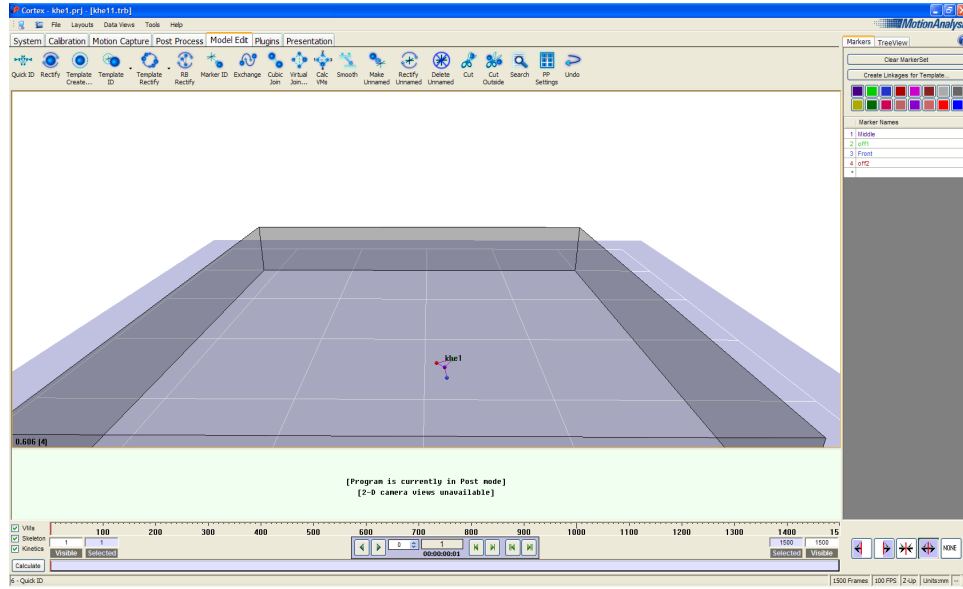


Figure 8: Model edit.

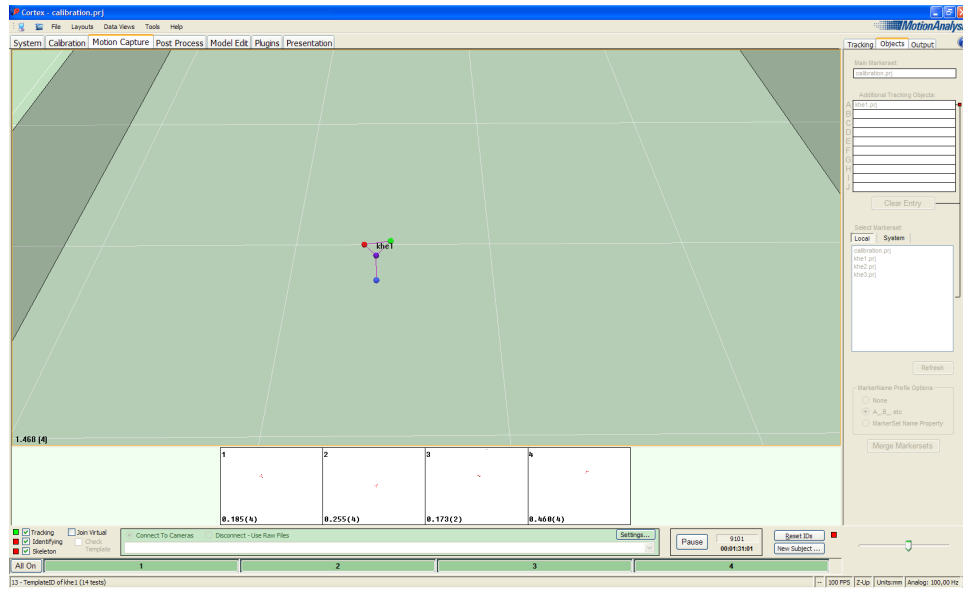


Figure 9: Robot identified in Cortex.

4 An Individual Robot Experiment

Here, we will extend the program presented in section 2.3 to include the Cortex functionality. Replace the code in "main.cpp" for the following one, changing the fields: "COMX", "localIP", "CortexIP" and "objectNameCortex". The last one is the template name you have chosen in Cortex. Copy EVaRT.dll to the Release folder of your project directory. Before running the program, be sure that Cortex is running and ALL the four markers of the robot are visible.

```
#include <windows.h>
#include <stdio.h>
#include <conio.h>

#include "KheperaIII_src\KheperaIII.h"
#include "KheperaIII_src\LocalizationSystem.h"

int main(int argc, char **argv)
```

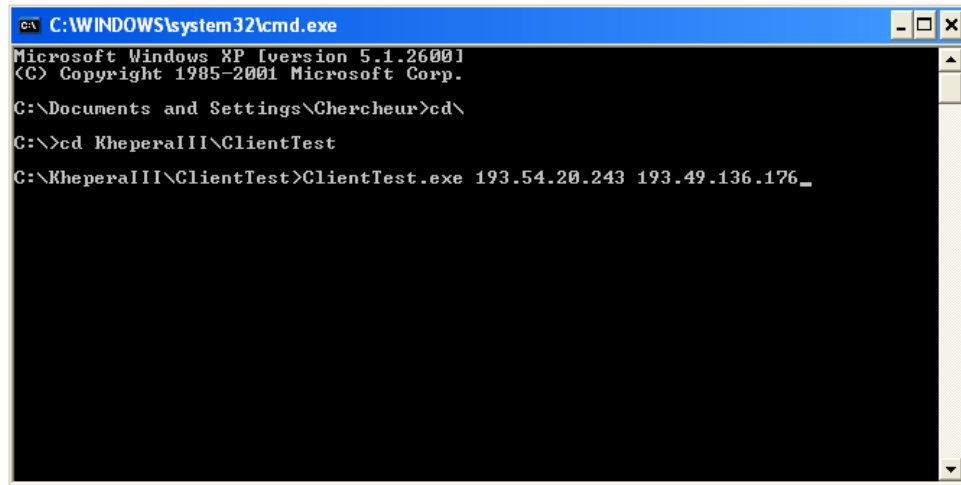



Figure 10: EVaRT test program.

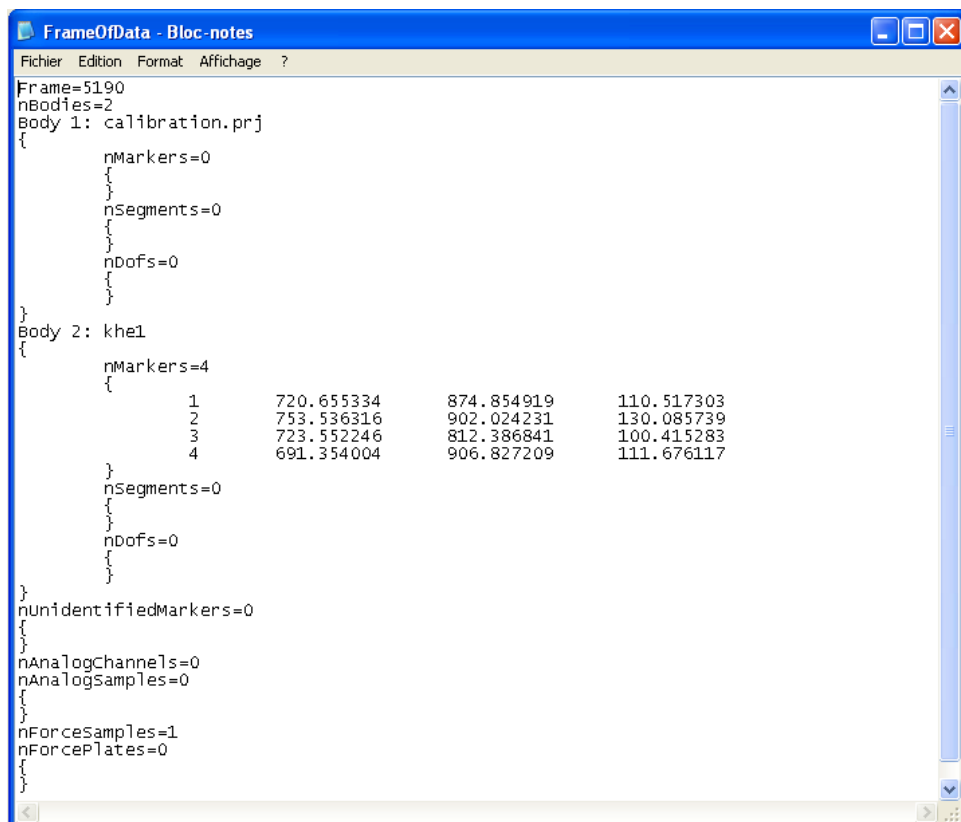


Figure 11: FrameOfData.txt example.

```
{
  int count = 0;

  KheperaIII *robot = new KheperaIII("COMX");
  printf("Initializatin local system...\n");
  robot->localizationSystem->init(2,1,"localIP",
  "CortexIP","objectNameCortex");

  while(count < 9)
  {
    robot->setVelocity(3,0);
  }
}
```

```

    printf(" Position X: %f Y: %f T: %f\n",
robot->getPosition()[0],robot->getPosition()[1],robot->getOrientation());
    robot->timeStep();
    Sleep(500);
    count++;
}

robot->setVelocity(0,0);
robot->closeSession();

return 0;
}

```

The robot will advance printing its position on the command line. Notice on the code, the initialization of the localization system and the method `timeStep()`. That method, among other functionalities, actualize the position of the robot. Try commenting that line and running the program again, you will see that only the initial position will be written in the console. Full details about the methods are available at section 8.

5 A Multiple Robots Experiment

To perform this experiment, which uses both the communication and localization system, you must perform the steps presented in sections 3.2 and 3.3 for a second robot. Therefore you will have a project in Cortex tracking two different robots. If you have controllers running in different computer, they must be located in the same LAN. Above is presented the code for `main.cpp`.

```

#include <winsock2.h>
#include <windows.h>
#include <stdio.h>
#include <conio.h>

#include "KheperaIII_src\KheperaIII.h"
#include "KheperaIII_src\LocalizationSystem.h"
#include "KheperaIII_src\CommunicationSystem.h"
#include "KheperaIII_src\Agent.h"

int main(int argc, char **argv)
{
    int count = 0;

    Agent neighbor;

    KheperaIII *robot = new KheperaIII(argv[2]);
    robot->setId(atoi(argv[1]));
    printf("Initializatin local system...\n");
    robot->localizationSystem->init(2,1,"193.54.20.158","193.49.136.176",
argv[3]);
    printf("Initializatin communication system...\n");
    robot->initComm("", "239.255.0.1",30001);

    while(count < 25)
    {
        if(robot->getnNeighbors() != 0)
        {
            neighbor = robot->getNeighbors()[0];
            printf("Neighbor %d X: %f Y: %f\n",neighbor.getId(),
neighbor.getPosition()[0],neighbor.getPosition()[1]);
        }
        robot->timeStep();
        Sleep(1000);
        count++;
    }

    robot->closeSession();

    return 0;
}

```

Notice now, that you must launch two programs, which require three parameters in the COMMAND LINE, so you cannot run directly from Visual C++. The syntax is: `programName.exe <id> <serialPort> <CortexIdentifier>`, where `id` is required to be an integer. The controller keeps the robot still, and prints the position of its neighbors, if they exist. After sometime the program will end, so you need to launch the second controller before it. When the second controller starts, both controllers will print the id and the position of the robot neighbor. Do not forget to ensure that both robots are TOTALLY visible at Cortex software.

The same method responsible to actualize the robot position, `robot->timeStep()`, is also responsible to send the robot information for its Neighbors. The communication system uses a broadcast protocol, i.e., one send, everyone

listen. Everyone that is on the same broadcast group, with same broadcast IP and port. If you modify the program to make the robots move, you will see that the position of the neighbors will change automatically.

6 Using Matlab to Run a Controller

In this section, we will demonstrate how to use simulink to run a controller for the khepera III. This is done with the provided simulink model.

- Open matlab and change the working directory to the one containing the matlab tools provided for khepera III.
- Launch simulink.
- Start a new model.
- Open KheperaIII.mdl and copy the block provided to your new created model. Do not modify the block in KheperaIII.mdl!
- Double click the block to fill the **Block parameters**.
 - Relative sample time : must be a multiple of the basic time step defined in simulink.
 - Serial Port : port to communicate with the robot, ex: 'COM2'. Strings use simple quotes in matlab.
 - Cortex Id: string identifier of the robot in the Cortex system.
 - Number of Neighbors: **WARNING!!! if you have a single robot experiment type 99**. In other case, type exactly the number of neighbors used in the experiment. When you run the simulation it will be blocked until ALL the neighbors connect to the robot. The system may stay unstable, during the wait time.
 - Id: integer to identify the robot.
 - Local IP: string of the local IP.
 - Cortex IP: string of Cortex Ip.

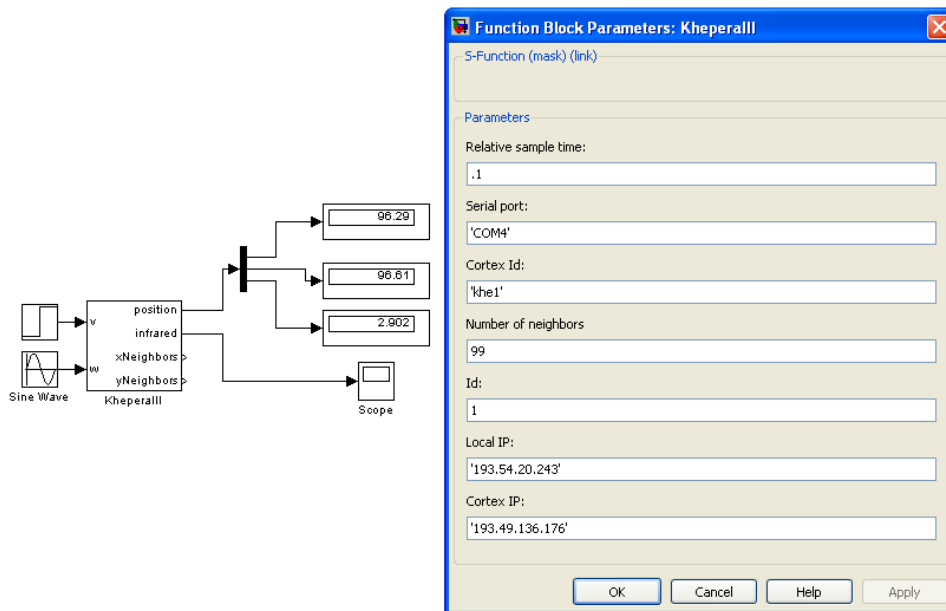


Figure 12: Simulink diagram and block parameters.

- After typing the arguments, click Ok. You will notice the block two inputs and four outputs.
- Inputs
 - v: linear speed in cm/s.
 - w: angular speed in rad/s

- Output
 - position: Array of length three. Robot position and orientation, X , Y , θ .
 - infrared: Array of length nine. Robot infrared sensors.
 - xNeighbors: Array of length $\langle \text{numberOfNeighbors} \rangle$. X position of the neighbors.
 - yNeighbors: Array of length $\langle \text{numberOfNeighbors} \rangle$. Y position of the neighbors.
- IN CASE OF CONTROL LAWS THAT USE A FEEDBACK OF SOME OUTPUT IN THE COMPUTE OF ANY INPUT, YOU MUST ADD A **UNIT DELAY** BLOCK ON THE INPUT.
- Now you can use your KheperaIII block as any other simulink block.

Unfortunately, If you want to perform a multiple robot experiment in matlab you need to use multiple instances of matlab. In other words, you can not put two kheperaIII blocks in the same simulation. This is due to a limitation of simulink that does not perform multi-thread in the simulations.

7 Troubleshooting

1. **When I run my program the system complains that EVaComm2.dll is not found.**
Just copy EVaComm2.dll and EVaComm2.lib, which are located inside KheperaIII.src, to the same folder where the executable is.
2. **When I compile my controller, the compiler says that WinSock.h has already been included**
Include $\langle \text{winsock2.h} \rangle$ before the other libraries in main.cpp.
3. **I run my program, it does not show any problem in the connection with the robot, but it looks blocked.**
Probably there was a previous communication that was interrupted before the robot received a "line feed", so the robot keeps waiting the end of the message. Just reset the robot or run the program more than one time to unblock the connection. If the problem persists, probably the battery has reached a low level. Recharge it.

8 Middleware Reference

Object.h

Method / Attribute	Description
Object()	Constructor.
void setId(int idArg);	Set the identifier of the object, which should be unique.
int getId();	Get the identifier of the object
void setPosition(double x,double y);	Set the xy position of the object in cm. Automatically set when using a localization system. Concurrent use not encouraged.
double* getPosition();	Get a size two array containing the xy position of the object {x,y} in cm
void setOrientation(double theta);	Set the orientation of the object in radians $[0;2*\pi]$. Automatically set when using a localization system. Concurrent use not encouraged.
double getOrientation();	Set the orientation of the object in radians $[0;2*\pi]$

Figure 13: Object.h reference.

Agent.h – generalization of Object.h

Method / Attribute	Description
Agent()	Constructor.
Agent* getNeighbors();	Return an agents array of size 'n' containing the agent neighbors.
Object* getObstacles();	Return an agents array of size 'n' containing the obstacles.
string getLocalAddress();	Return the ip address of the agent.
int getnNeighbors();	Return the number of neighbors.
void setnNeighbors(int n);	Set the number of neighbors. Automatically set and actualized when using communication system. Concurrent use not encouraged.
void setNeighbors(Agent* neighbors);	Set the agent neighbors. Automatically set and actualized when using communication system. Concurrent use not encouraged.
void setObstacles(Object* obstacles);	Set the agent array of obstacles.
void setLocalAddress(string host);	Set agent ip address.
TrackGenerator* trackGenerator	Point to the agent trackGenerator.

Figure 14: Agent.h reference.

KheperaIII.h – generalization of Agent.h

Method / Attribute	Description
KheperaIII(string serialPort)	Constructor. Argument is the serial port name ex: COM4
void initComm(string adLoc, string adMult, int porMult);	Initialize communication system. adLoc: local ip address adMult: multicast group address portMult: multicast group port
void timeStep();	Actualize, if enabled, the robot variables (position, neighbors and trajectory) and send its position to the others. Should be called in every control step.
void setVelocity(double vSpeed, double wSpeed);	Set robot speed. vSpeed: linear speed in cm/s wAngular: angular speed in rad/s
int* getIrOutput();	Return a size 9 array with the infrared sensors measures.
int* getEncodersValue();	Return a size 2 array with the encoders values. {leftEncoder, rightEncoder}
void setEncodersValue(int leftEncoder, int rightEncoder);	Set the value of the encoders. Not recommended when using the localization system
void closeSession();	Close sockets and links.

Figure 15: KheperaIII.h reference.

LocalizationSystem.h

Method / Attribute	Description
LocalizationSystem(KheperaIII* robot);	Constructor. When a KheperaIII is instantiated, it already assigns a localization system to it.
void init(int modeArg, int autoAtualize);	Initialize localization system in the purely estimated mode. modeArg = 1 autoAtualize: 1->true 0-> false. The initial position must be defined in DEFINITIONS.H as X0, Y0 and THETA0
void init(int modeArg, int autoAtualize, string myAddress, string hostAddress, string bodyName);	Initialize localization system in the mixed mode. modeArg = 2 autoAtualize: 1->true, 0-> false. myAddress: robot ip address hostAddress: localization system ip address hostAddress: EVaRT ip address bodyName: EVaRT identifier.
void atualizePosition();	Atualize the position in the robot variable. Automatically called by the robot time step if autoAtualize is enabled.
void Close();	Close connection with EVaRT.
double* getOwnPosition_EVaRT();	Returns the robot position from EVaRT. {ack,x,y,theta}. Ack: 1->measure OK, 2->bad measure.
int countT;	Total number of attempts to consult EVaRT. Only works in mixed mode and with autoEnable=true;
int countO;	Number of accomplished attempts to consult EVaRT. Only works in mixed mode and with autoEnable=true;

Figure 16: LocalizationSystem.h reference.

TrackGenerator.h

Method / Attribute	Description
void init(int mode);	Initialize the track generator: mode: 1->point, 2->straight. If mode= 2, the straight parameter should be defined in DEFINITIONS.h: KX, KY, OSX and OSY. The timer starts to count when this method is called.
void nextStep();	Recalculate the desired position. Automatically called by robot time step.
void setMode(int);	Set the function mode: mode: 1->point, 2->straight
void setDesiredPosition(double*);	Set the desired position {xd,yd}. Only use it in mode '1'.
int getMode();	Return the function mode
double* getDesiredPosition();	Return the desired position {xd,yd}.

Figure 17: TrackGenerator.h reference.