

COS 495 – Autonomous Robot Navigation

Lab 4

Introduction to the X80 Robot

INTRODUCTION

The purpose of this lab is to introduce students to the X80 and the associated C++ base code that will be programmed within Visual Studio. By the end of the lab, students should understand the system architecture, and how to program the robots to conduct future lab experiments.

BACKGROUND

The X80 uses a host computer to command and control the robot. Command signals, are accepted from a host computer application then sent over wireless communication to an embedded microcontroller on the robot. Sensor signals are sent from the embedded microcontroller back through the wireless communication to the host computer application. See Figure 1 below.

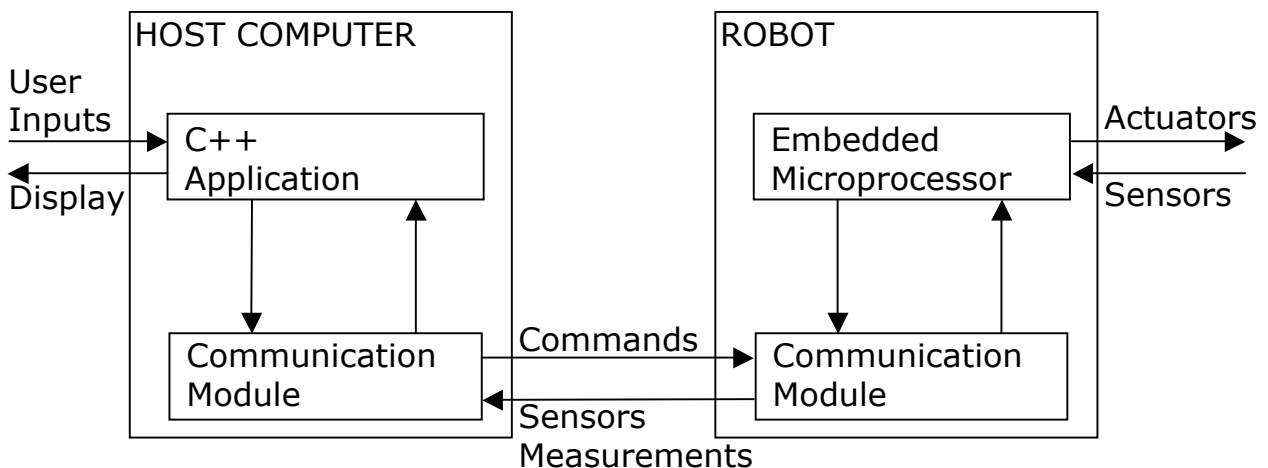


Figure 1: System Architecture

In this course, we will only be concerned with programming the C++ Application on the host computer. The communication modules, embedded microprocessor, actuators and sensors will not be reprogrammed or reconfigured.

For detailed description of the X80, see the manual that can be downloaded from the course website's lab page.

EXPERIMENTS

1. Locate the parts

Familiarize yourself with the robot. Be careful when lifting the robot off the ground. Note it is unstable without the batteries. Find the following components:

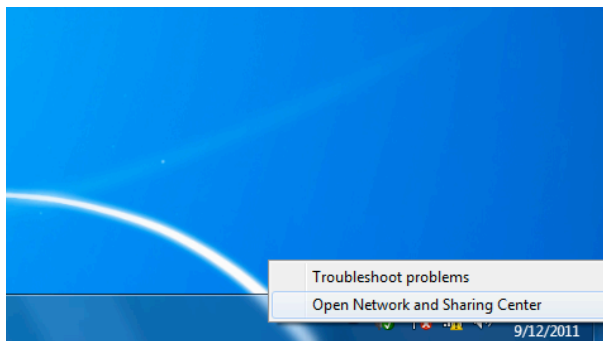
- Battery pack
- Infra red range sensors
- Ultrasound range sensors
- Wheel encoders
- Video camera
- Pan/tilt servos for the video camera
- Microprocessor board
- Wireless Ethernet antenna
- Power button

Try connecting the battery pack to the robot. There is a plug at the back, and a Velcro seat for the pack to be placed. Turn the robot on.

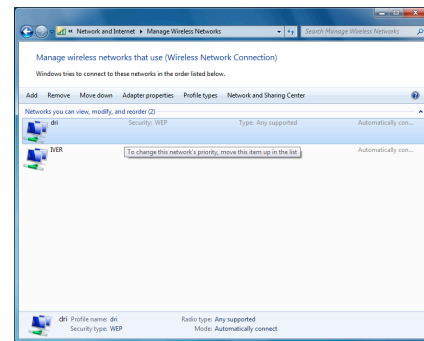
2. Connect to the robot

To make a wireless communication connection with the robot, you must first establish a connection between your host PC and one of the wireless routers in the lab. Each robot is configured to work with a different wireless router, so make sure you connect to the correct router.

To connect to a router, plug in your NetGear USB wireless adapter. Right-click on the wireless connection icon located on the bottom right corner of your system tray, and select *Open Network and Sharing Center* (see Fig. 2a).



(a)



(b)

Figure 2: Connecting to the wireless router

On the pop up menu, select *Manage Wireless Networks*. This will open a new window listing the available wireless connections, (Fig. 2b). Disconnect from puwireless if your computer is connected to it.

Double click on the appropriate wireless router, (see table below). Enter the routers Web key: 112233445566778899AABBCCDD. You need to enter it twice. Wait for the connection to be made. Note you will not be able to get internet access from Mustang Wireless and be connected to the robot simultaneously.

Identification	Color	IP	SSID
dri	NA	192.168.0.200	dri
robot1	Light blue	192.168.0.201	dri
robot2	Light green	192.168.0.202	dri
robot3	Black	192.168.0.203	dri
robot4	Red	192.168.0.204	dri
robot5	Dark green	192.168.0.205	dri
robot6	Dark yellow	192.168.0.206	dri2
robot7	Dark blue	192.168.0.207	dri2
robot8	Bright yellow	192.168.0.208	dri2
robot9	Silver	192.168.0.209	dri2
robot10	Orange	192.168.0.210	dri2
dri2	NA	192.168.0.211	dri2

Table 1: Network Configuration

Now that you are connected to the wireless router, double click on the *WiRobotGateforWiFi* icon on your desktop. Make sure you don't use the one from the start menu. A window should pop up that looks like:

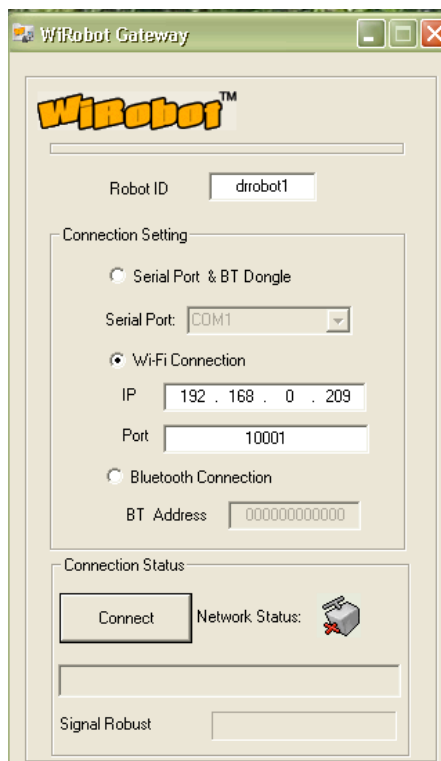


Figure 3: WiFi gateway utility window

Set the *Robot ID* to “drrobot1” as shown in Fig. 3. This will link your C++ code to the wireless connection. If you want to use a different name other than drrobot1, you must also change this in your C++ code.

Click on the *Wi-Fi Connection* radio button, and enter the IP of your robot, (shown on bottom of robot or in Table 1). Enter the *Port* as “10001” as shown in Fig. 3. Then press *Connect*. If the connection is made, the WiFi gateway window will minimize itself.

3. Locate the code

In this class, we will not be touching the embedded code on the robot’s microprocessor. Instead, we will program everything on a stand alone PC. This code will run on the PC, using the WiFi connection to get closed loop control.

First, let’s get the code base on to the PC. From the website, download the zipped folder “COS495-Lab4_BaseCode.zip” and extract it to your *Desktop*. You may want to rename the extracted folder “COS495-Lab4_BaseCode_X”, where X is your group name.

Open the folder and double click on the file “RoboticsLab.sln”. A visual studio window should open, with the base code solution, (see Fig. 4).

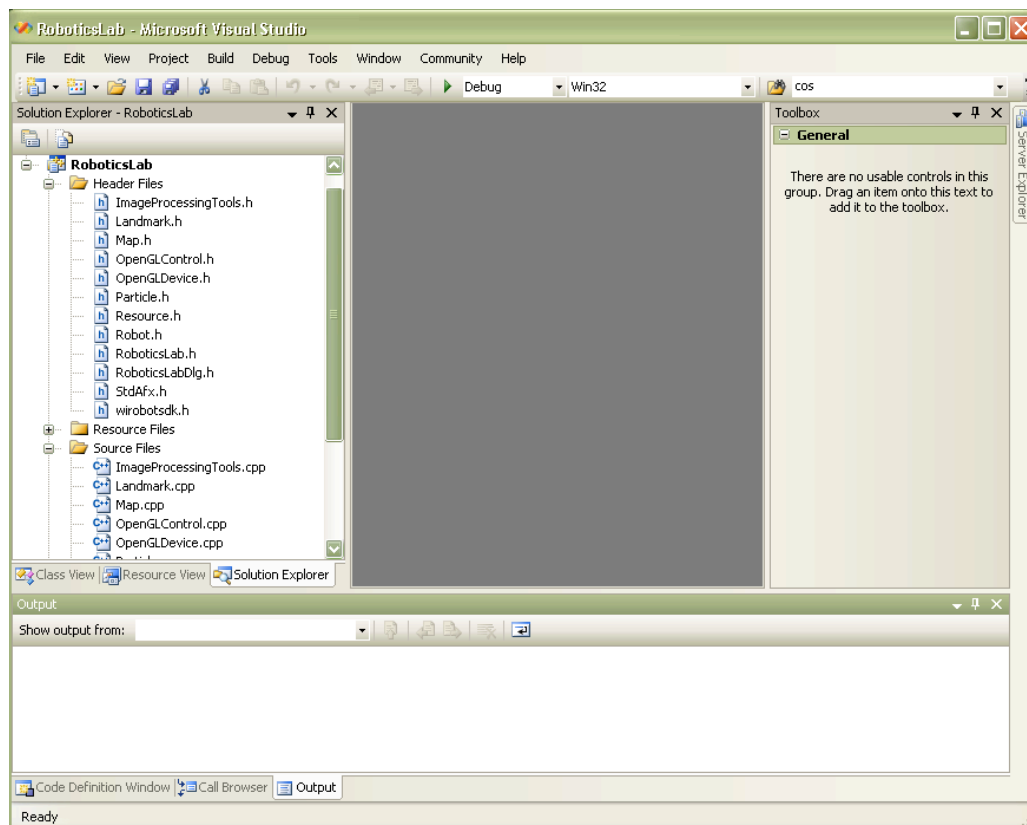


Figure 4: Robotics Lab Solution opened in Microsoft Visual Studio

Now you can use the list of files in the *Solution Explorer* (left pane) to navigate the code. You can double click on the files to open them for viewing or editing. Make sure you can find the files listed below.

3.1 Files needed for this lab

The following files are those that you will edit in subsequent steps of this lab.

- Robot.*
- RoboticsLabDlg.*

3.2 Files needed for future labs

The following files are those that you will edit in future labs. These files will be used for localization and image tracking labs.

- ImageProcessingTools.*
- Landmark.*
- Map.*
- Particle.*

3.3 Files that shouldn't be edited

You shouldn't have to touch the files listed below. The OpenGL files enable the 3D viewing of the robot. The wirobotsdk files provide the interface with the actual robot (or simulator as you will see). The RoboticsLab.* files are the application file with instantiates the dialog box (RoboticsLabDlg).

- OpenGLControl.*
- OpenGLDevice.*
- StdAfx.*
- wirobotsdk.*
- RoboticsLab.*

The overall code architecture is shown in Fig. 5:

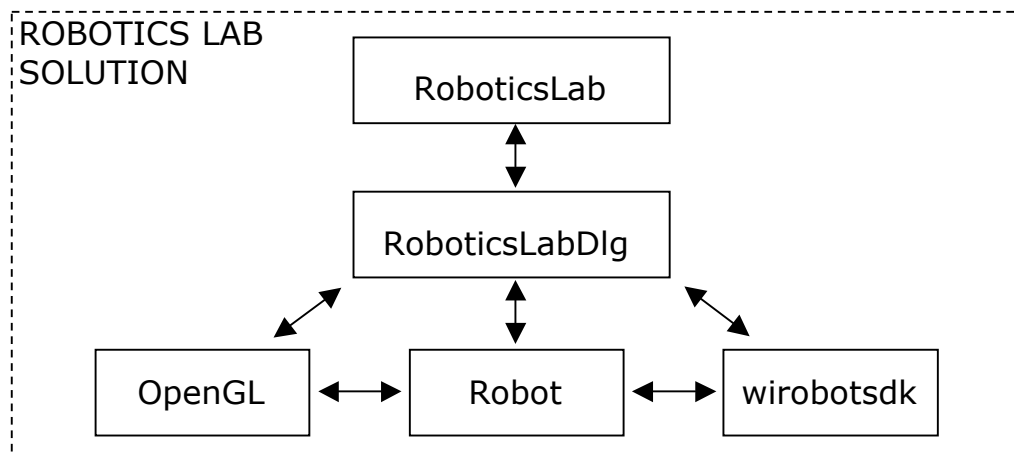


Figure 5: Overview of Class structure.

Note that the application file is *RoboticsLab*, which instantiates a dialogue window: *RoboticsLabDlg*. Within *RoboticsLabDlg*, a *Robot* is instantiated. This is where most of the programming is done for localization and control of the robot. The OpenGL classes are used for displaying robot states. The *wirobotsdk* class is used to interface with the real robot through WiFi.

4. Build and Run

From the *Build* drop down menu on Visual Studio, select *Build Solution*. From the *Debugging* drop down menu, select *Start Debugging*. A dialog box should appear as shown in Fig. 6.

The left panels display sensor readings. The top right section shows a 3D interpretation of the robot within its environment. The bottom right panels have control buttons and settings.

Of importance is the *Robot Type* radio button set. This allows users to switch between control of a simulated robot agent, or the actual robot itself. Students should conduct all debugging in *Simulator* mode.

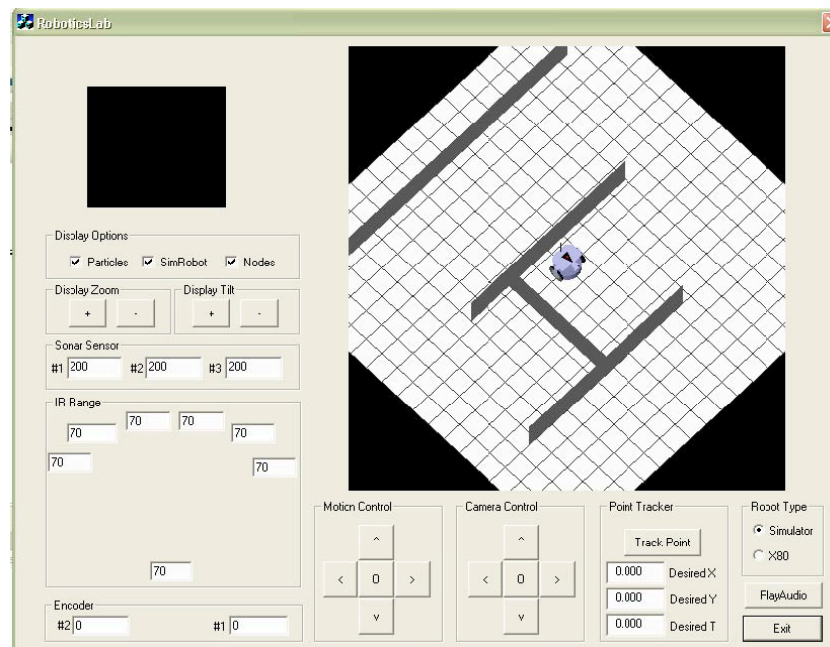


Figure 6: RoboticsLab Dialogue Window

5. Control the Robot Manually

Select the *X80* radio button for *Robot Type*. Experiment with the *Motion Control* and *Camera Control* buttons.

Repeat using the *Simulator*. Note at this point the robot will not move around in your display, we will code that up in the next lab!

6. Reverse directions of robot Motion

In this part of the lab, you will edit the RoboticsLabDlg.cpp file to make the motion control buttons work in reverse. The purpose of this step is to get acquainted with the code in two ways:

- Where command functions from the dialogue window are located.
- How the wirobotsdk is used to communicate with the robot

Find the function `CRoboticsLabDlg::OnForward()`. This function is called when the button with the upward facing arrow is pressed. The code in this function is:

```
// Set Control Mode to
robot_dlg->controllerType = CONTROLLER_TYPE_MANUALCONTROL;

// Set Desired Speed
double desiredWheelSpeed1 = -0.1*encoderResolution / (2*3.14*wheelRadius);
double desiredWheelSpeed2 = +0.1*encoderResolution / (2*3.14*wheelRadius);
m_MOTSDK_real.SetDcMotorControlMode (0,M_VELOCITY);
m_MOTSDK_real.SetDcMotorControlMode (1,M_VELOCITY);
m_MOTSDK_real.SetDcMotorVelocityControlPID (0, 30, 10, 0);
m_MOTSDK_real.SetDcMotorVelocityControlPID (1, 30, 10, 0);
m_MOTSDK->DcMotorVelocityNonTimeCtrAll((short)desiredWheelSpeed1,
(short)desiredWheelSpeed2,NO_CONTROL,NO_CONTROL,NO_CONTROL,NO_CONTROL);
```

The first line is used to set the robot control mode to manual (as opposed to autonomous mode used for point tracking, wall tracking etc.) Then, desired speeds for each wheel are set. Note that `encoderResolution` is the number of counts in one rotation. These commands are used to set the speed to 0.1 m/s. Confirm the math makes sense to you, making sure to understand the units. Such calculations are used often in mobile robots.

The next 2 lines of code are used to set the type of DC Motor control. They set the mode to `M_VELOCITY`, implying that the motors should track desired velocity. After these lines, there are two lines of code setting the controller gains for velocity control mode. These are PID gains.

Finally, there is a command that sends the desired speeds to the robot, so that they may be used for velocity control of the motors.

Note that two different instances of the wirobotsdk are used here: `m_MOTSDK_real` and a pointer `m_MOTSDK`. In general, the base code uses the pointer `m_MOTSDK` to point to either `m_MOTSDK_real` or `m_MOTSDK_simulator`, depending whether the dialogue window has *X80* or *simulator* selected under robot type. Since the simulator does not need motor parameters set, we only set PID gains on `m_MOTSDK_real`.

Now that you are somewhat familiar with the code, you can make some simple changes. See if you can edit the code to make the robot move backwards instead of forward when the button labeled with the up arrow is pressed. Don't forget to change this back for later.

7. Read the Sensors

Now, find the function `CRoboticsLabDlg::StartDlgUpdateThread`. This is one of a few threads that runs on its own loop. This particular thread is used to update the sensor measurements from the robot.

For example, the following code is used to grab the current encoder count from the robot.

```
// Update Encoders
ts->_this->m_Encoder1 = ts->_this->m_MOTSDK->GetEncoderPulse1();
ts->_this->m_Encoder2 = ts->_this->m_MOTSDK->GetEncoderPulse2();
```

Again we see that `m_MOTSDK` is used to access the robot or simulated robot, (depending on where it is pointing). See if you can make encoder1 display 99 on the dialogue window. Now see if you can switch sonar readouts 1 and 3 on the dialogue window. Don't forget to switch these back when you are done.

8. Closed loop control

The goal of this part of the lab is to make the robot drive to a wall, and position itself 50 cm from the wall. The robot should start facing the wall, and be located somewhere between 20 and 150 cm from the wall.

The closed loop control should work as follows. If the robot is farther than 50 cm from the wall, it should move forward at a SLOW constant speed. If the robot is closer than 50 cm to the wall, it should move backwards. If you are clever you can use a "Proportional Feedback Control" system. Ultrasound range sensors will provide the distance measurement.

The function to be edited is `Robot :: WallPositioning`, located in `Robot.cpp`.

Note how the function is called from `Robot :: RunControlLoop`, the main function called from the thread dedicated to robot control in `RoboticsLabDlg.cpp`. (Try to find these calls).

Within `Robot :: RunControlLoop`, most of your future labs will be based. It is from here that different localization and control algorithms will be called.