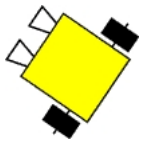
cis1.5/robotics-spring2008-azhar
robotics.edu—agents lab
Brooklyn College, CUNY



cis1.5-spring2008-azhar, lab IV, part 2

instructions

- This is the second part of the lab/homework assignment for unit IV.
- The entire assignment will be worth 9 points: the first part is worth 4 points and the second part is worth 4 points. **Both parts together are due on Thursday April 17** and must be submitted by email, as below:
  1. Create a mail message addressed to *mqazhar@sci.brooklyn.cuny.edu* with the subject line **cis1.5 hw4**.
  2. Attach ONLY the **.cpp** files for each part, as outlined below.
  3. Failure to follow these instructions will result in points being taken away from your grade. The number of points will be in proportion to the extent to which you did not follow instructions.

remote controller: remote controlling a robot to find a ball

For this assignment, your robot will find a ball. Your program will take the user's commands that will direct your robot to the ball. Robot will then take a picture of the ball!

Your program will save the users' commands in a character array(e.g., directions) and write that array to a file. Your program will then read the characters back into another array and then send those commands represented by the characters in the array to the robot one at a time (remember to close and open the file). The Robot will take a picture using *takePhoto()* method and *savePhoto("nameofyourpicture.jpg")* method of *Surveyor* [1] class and when it reached the destination.

You can make your own software design decisions about whether you want to write separate functions to do each step of if you want to do all the steps inside the `main()`.

Name your program: **findball.cpp** and submit ONLY this file.

programming assignment (TO BE DONE WITH A PARTNER)

- MY PARTNER'S NAME IS: _____

[1]

- After you get the program to work, write the code in the boxes provided. Partial credit will be given!
- **Demonstrate each working program for your instructor.**

vocabulary

- Teleoperated robots
- Surveyor Camera
- Array (e.g.,character array)
- File input/output
- class (e.g., Surveyor)
- object (e.g., robot)
- member function (e.g., takephoto(), savephoto("myphoto.jpg")
- calling member function (e.g., robot.takephoto())

materials

Make sure you have all of the following materials before you start the lab:

- Surveyor robot
- XBee USB dongle
- c++ compiler, such as mingw (Windows) or g++ (Mac)
- codeblock, text editor, such as Notepad (Windows) or TextEdit (Mac) or pico/VI/nano (Unix)
- you must be in
  - `\dev\examples` directory in Windows
  - `/dev/examples` directory In Mac or Unix or Linux computer
  - Please note that Windows computers use *back slash* (\) to separate directory (e.g., `\dev\examples`) while all other operating systems (e.g., Linux, Unix and Mac) use *forward slash* (/) to separate directory (e.g., `/dev/examples`)
  - use **cd** command to change directory

- **create your first program**
  - Enter the following program using a text editor:

```
#include "SVR.h"
#include <iostream>
#include <assert.h>
using namespace std;

/* make a global Surveyor object called robot */
Surveyor robot(ADDRESS);


int main()
{
  char buffer[256];

  robot.getVersion(buffer);
  cout << "SRV-1 version " << buffer << endl;

  /*take a picture of what robot is looking at*/
  robot.takePhoto();
  robot.savePhoto("whatrobotsees.jpg");

  return 0;
}
```

    In your editor, make sure you save the file as **findball.cpp** in `\dev\examples\` directory!

- **dissecting the code**
  - `#include "SVR.h";`
    This line always needs to be included at the top of all programs writen for the Surveyor. Whenever you use a command in a program like `robot.drive(50,50,100);`, the Surveyor has to know what that means. The `includet...` line tells the program where to find a "library" of all the surveyor specific terms. Please note that this header is not part of standard C++ library. SVR.h is specially written for Surveyor robot.

  - `#include <iostream>;`
    This header is an object-oriented library that provides input and output functionality using streams.

  - `#include <assert.h>;`
    This header defines one macro that can be used as a standard debugging tool.

  - `using namespace std;`
    This line lets you use cout and cin without additional syntax (e.g., std::cout).
  - `Surveyor robot (ADDRESS);`
    - **Creating class object:** This line creates an **object** of class type `Surveyor` called `robot`. You may use this object (e.g., **robot**) to send commands (e.g., **(**drive(50, 50, 100)) to your robot.

    * **Global declaration:** This is a **global** declaration and can be used by main and as well as your defined function in this program.

  - `int main ( )`
    This line needs to be in every program. This denotes the **main function**. This is what will be run when the program starts.

  - `{`
    This is a beginning **curly bracket**. Everything between the beginning ({) and ending (}) curly brackets is part of `main`. Curly brackets are used to delineate not only the main but also the function and blocks within a program.

  - `char buffer [256];`
    * declares a character array of size 256.
  - `robot.getVersion(buffer);`
    * gets the firmware version of the robot and store in the *buffer* array.
  - `cout<< "‘SRV-1 version "‘ << buffer << endl;`
    * prints SRV-1 version that was stored in the buffer array. It is important to note that in case of communication failure between your computer and robot you will not see the version. If you don't see the version number, try one of the following:
      · open up a new console and try running the program again
      · turn on and off the robot
      · connect your USB dongle to a different USB port
      · restart your computer
    * `robot.takePhoto();`
      · take the picture of what robot is looking at
    * `robot.savePhoto("whatrobotsees.jpg");`
    * save picture as "whatrobotsees.jpg". You may give a different name if you like. Please note that both *takePhoto()* and *savePhoto("whatrobotsees.jpg")* are member functions of Surveyor robot class. You are calling using *Surveyor class* object called *robot*.

• compile your program

  - From the Windows command window or the Mac terminal window, at the command-line prompt (>), type:
    `> make findball`
    This will compile and link the program.
  - If there are errors in your program code, fix them and then try compiling and linking again—until the program compiles and links without any errors.

• run your program

  - Make sure the **XBee dongle** is be same channel as your robot and is connected to your computer's USB port.
  - Connect it to your computer's USB port (ask if you need help finding it).
  - The Surveyor must be turned on.

– On Windows, at the command-line prompt (>), type:
`> findball`
or on the Mac (or Linux or Unix) terminal window, at the command-line prompt (>), type:
`> ./findball`
This will run your program and send commands to the robot.

– What did your robot do? Is that what you expected? Write your answer below.

– **troubleshooting**

* If you have trouble, check these things:
  1. The Surveyor must be turned on during communication.
  2. LEDs on the robot and dongle should be on.
  3. the `make` command should be in your **path** (ask if you need help).
  4. you should be in the right directory:
     · On Windows:

     `\dev\examples` directory in Windows.
     · On Mac:
     `/dev/examples` directory In Mac or Unix or Linux computer.
  5. Make sure the **XBee dongle** is be same **channel** as your robot and is connected to your computer's USB port.

• modify the program

– Define a character array to store 25 values. *(0.5 point)*

– Refer back to your **roomba** programs, and have your program prompt the user for input as follows:

* enter F to make the robot go forward
* enter B to make the robot go backward
* enter R to make the robot turn right
* enter L to make the robot turn left
* enter Q to quit the program

The program should let the user enter as many commands as s/he wants to (F, B, R, or L), but no more than 25.
Each time the user enters a command, you should store that value in the character array. *(0.5 point)*

– When the user is finished entering commands (i.e., s/he enters Q), then your program should open a file for writing, write all the values you stored in your character array to the file, and then close the file. *(0.5 point)*

– Then your program should open that file for reading, read in all the commands to a character array, and then close the file. *(1 point)*

– Your program should then read each command from the character array and send the command to the robot. By calling an appropriate function, we will assume that each command will execute for one second. *(1 point)*

– When finished, your robot will take a picture after executing all commands. *(0.25 point)*

– Test your program by giving your neighbor group a chance to try out your program. *(0.25 point)*

- OPTIONAL
  - Write a program that will prompt the user for input as follows:
    - enter F to make the robot go forward
    - enter B to make the robot go backward
    - enter R to make the robot turn right
    - enter L to make the robot turn left
    - enter S to save all the commands in a file called "mycommands.txt" and quit the program

    The program should let the user enter as many commands as s/he wants to (F, B, R, or L), but no more than 25.
    Each time the user enters a command, you should store that value in the character array.
    (0.5 point)
  - Write another program that will open the "mycommand.txt" file for reading, read in all the commands to a character array, and then close the file. Your program should then read each command from the character array and send the command to the robot. By calling an appropriate function, we will assume that each command will execute for one second. Save this program as **findball.cpp**. *(1 point)*