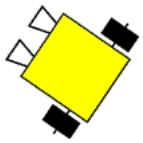
cis1.5/robotics-spring2008-azhar
robotics.edu—agents lab
Brooklyn College, CUNY



cis1.5-spring2008-azhar, lab IV, part 3

instructions

- This is the third part of the lab/homework assignment for unit IV.
- The third part is worth 4 points and **optional**. The **third part is due on Thursday April 17th** 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 hw4part3**.
  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.

vision processing: surveyor robot finds the ball *autonomously!*

A robot becomes a lot better when it has sensors, then it can make decisions about what to do based on its environment. The Surveyor has a color camera as its main sensor. For this assignment, your robot will find a ball by itself **autonomously** (i.e., without user's help) using its color camera. Robot will take the picture using *takePhoto()* method and *savePhoto("nameofyourpicture.jpg")* method of *Surveyor* [1] class. 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: ____
- 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.**

[1]

vocabulary

- autonomous
- vision processing
- image size (e.g.,64 rows by 80 colomns)
- pixel
- indexing through an array
- class (e.g., Surveyor, YUVRange) and object (e.g., robot, ballColor)
- member function (e.g., takephoto() and savephoto("myphoto.jpg"))
- calling member function (e.g., robot.getCountScan(1, ball))

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 (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:

```
/*findball
  a simple demonstration program for the Surveyor class

  usage: findball

  Description: robot will turn until it sees the ball
*/

#include "SVR.h"
/*the color of the ball is stored here*/
#include "color.h"

#include <stdio.h>
#include <stdlib.h>

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


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

/* make a global YUVRange object called ballColor
to set up the color of the ball*/
YUVRange ballColor(T_ORANGE_BALL);

int main()
{
    char buffer[256];
    int ball[80];
    robot.getVersion(buffer);
    cout << "SRV-1 version " << buffer << endl;

    /*robot will compare what it sees by using given ballColor object*/
    robot.setBin(1, ballColor);

    for (;;) /*this is a forever loop*/
    {
    /*checks to see if there is a color match */
    robot.getCountScan(1, ball);
    if (ball[40] > 0){
        break; /*robot thinks that it finds a match*/
    robot.drive(-40, 40, 3); /* turn a bit for .03 second*/
    }

   return 0;
}
```

- **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.

- **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.getCountScan(1,ball);`, 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 "color.h"` the color value (YUVRange) of the ball is stored here
  - `#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.
  - `YUVRange ballColor(T_ORANGE_BALL);`
    - we have a new object called *ballColor* which encapsulates the robots idea of the color of the ball. We included *color.h* to get the value T_ORANGE_BALL used to initialize the ballColor object. One of the nice things about C++ is that we can use objects without knowing everything about them. For the moment think of ballColor as specifying orange.
  - `robot.setBin(1, ballColor);`
    - tells the robot we are interested in orange things. The robot has ten "color bins" numbered from 0 to 9 (think: a color array) and we have set bin number 1 to orange.
  - `robot.getCountScan(1, ball);`
    - this line fill the array ball with values depending on where the robot sees the color orange.

- **vision processing**
  The Surveyor robot divides its field of vision into an 80 by 64 grid. 80 horizontal and 64 vertical (80 in the X direction, 64 in Y direction). Which the robot views as 80 columns, and the *getCountScan* method will load each element of the array with the number of orange pixels in the corresponding column.

  For example, if the robot is seeing the above picture, the *getCountScan(1, ball);* will fill the array ball with the following values

```
getCountScan(1, ...) =
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  4  4  5  3
 5  3  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

  The robot doesn't see any orange in colums 0 through 34 so the array elements 0 through 34 are loaded with zeros. In colums 35 to 41 the robot sees the orange ball and so array elements 35 to 41 are loaded with values greater than zero. The robot doesn't see any orange in columns 42 to 79 and so array elements 42 to 79 are loaded with zeros.

  Back to the program, inside the for loop we getCountScan, if the element 40 is zero or one we turn the robot a bit(note use of the duration parameter to get a small precise turn). When element 40 is two or greater we break out of the for loop, stop the robot and terminate the program.
  The effect of this is that the robot will turn until it sees the ball.

- **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**

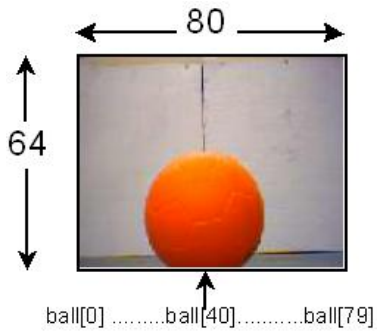
* If you have trouble, check these things:
  1. The Surveyor must be turned on during communication.
  2. LEDs on the robot and dongle should be on.
  3. the `make` command should be in your **path** (ask if you need help).
  4. you should be in the right directory:
     · On Windows:

       `\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**

– Display all the values of array **ball** in 4 lines. (Hint: you should enter a new line after displaying 20 consecutive integers.) Here is the entire array from the figure below:

```
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 3  9 10 11 12 13 11 13 14 14 15 16 18 18 19 19 19 19 19 19
18 19 18 17 17 17 15 15 18 20 21 21 18 14  5  0  0  0  0  0
 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```



– Please note that line 1 and line 2 (i.e., first two lines, ball[0] ... ball[39]) represent left side of the picture and line 3 and line 4 (last two lines, ball[40] ... ball[79]) represent right side of the picture.

– Modify the program to count the number of pixels that match the ball's color in *ball* array.
(Hint: `if (ball[i]>0) increment counter variable`).
Print out this number. What do you think is the significance of this number?

- Modify the program to add up the number of pixels that match the ball's color.
  (Hint: add up all the values in array **ball[80]**). Print out this number. What does this number represent?
- Display "Yeah, I think that I found the ball"
- Take and save a picture of what the robot sees. Do you see the ball in the picture? [Hint: your picture is saved in the same directory of your **findball.cpp** program]

- **more challenges**
  - Make your robot move around (pattern or random), checking regularly if it sees the ball. If the robot sees more pixels of the correct color than it has before, it will take and save a picture named *bestshot.jpg*. After the end of a test run display array **ball** and examine the *bestshot.jpg* picture.
  - Modify the program so the robot points to the the center of the ball (not the edge). Take a picture and display ball array. Is it the center?
  - Then modify the program so the robot moves towards the ball, taking pictures as it goes. Stop the robot when it has no longer "sees" the ball.

    moving towards the ball when it finds it. Robot will take pictures as it goes. It should stop taking pictures and come to a hault when there is no ball in front of it.
  - Modify the program to make the robot find the ball and push the ball.
  - Modify the program to make the robot track a moving ball.
  - Modify the program to make the robot play soccer. What would be a good strategy to avoid its own goals? What would you need for robots to cooperate?