Robotics With the XBC Controller

Session 8

Instructor: David Culp

Email: culpd@cfbisd.edu

Learning Goals

- The student will learn advanced techniques for working with the XBC camera including:
  - Bounding box data.
  - Filtering blob sizes.
  - Displaying live video.
  - Changing the camera display and config settings.
  - Getting and setting the white balance of the XBC.
  - Getting and setting the color models on the XBC.

Getting Bounding Box Data

- track_bbox_left(int ch, int i);
  - gets the pixel x coordinate of the leftmost pixel in the blob
- track_bbox_right(int ch, int i);
  - gets the pixel x coordinate of the rightmost pixel in the blob
- track_bbox_top(int ch, int i);
  - gets the pixel y coordinate of the topmost pixel in the blob
- track_bbox_bottom(int ch, int i);
  - gets the pixel y coordinate of the bottommost pixel in the blob
- track_bbox_width(int ch, int i);
  - gets the pixel x width of the bounding box of the blob. This is equivalent to track_bbox_right - track_bbox_left
- track_bbox_height(int ch, int i);
  - gets the pixel y height of the bounding box of the blob. This is equivalent to track_bbox_bottom - track_bbox_top

Camera API Review

- Remember to #use "xbccamlib.ic."
- Always call track_update() to get new tracking data!
- `int track_count(int ch);`
  - Returns the number of blobs on color channel ch the camera is currently tracking.
- `int track_size(int ch, int i);`
  - Returns the size, in pixels, of blob number i on channel ch.
- `int track_x(int ch, int i);`
  - Returns the x coordinate of the center of blob number i on channel ch.
- `int track_y(int ch, int i);`
  - Returns the y coordinate of the center of blob number i on channel ch.
- `int track_confidence(int ch, int i);`
  - Returns the confidence value (0-100) that blob i on channel ch is the correct color.
    - Higher numbers mean better confidence.

Why Use Bounding Box Data?

- Allows more precise control when positioning our robot in relation to an object.
- Think of our challenge.
  - We need to drive towards an orange object.
  - We must stop a certain distance from it.
  - We must then position ourselves in such a way that the robot can grasp the ball.
  - We can proportionally move the robot until the edges of the object are exactly where we need it and the bounding box width and height are where we need them.

Printing Bounding Box Data

```
#use "xbccamlib.ic"

void main()
{
   while(!b_button())
     {
       track_update();  // We must always call track update to get new
    tracking data!
       display_clear();
        printf("track_bbox_left: %d\n", track_bbox_left(0,0));
       printf("track_bbox_right: %d\n", track_bbox_right(0,0));
       printf("track_bbox_top: %d\n", track_bbox_top(0,0));
       printf("track_bbox_bottom: %d\n", track_bbox_bottom(0,0));
       printf("track_bbox_width: %d\n", track_bbox_width(0,0));
       printf("track_bbox_height: %d\n", track_bbox_height(0,0));
       sleep(0.2);
   }
}
```

Filtering Out Small Blobs

- void track_set_minarea(int minarea);
  - Sets the minimum area for a blob to be tracked. Blobs below this size will be ignored.
  - Default area is 100.
  - Can be set interactively (see on screen demo).
- int track_get_minarea();
  - Returns the current minimum area of a blob to be considered valid.

Showing Live Video on the XBC.

- void track_show_display(int show_processed, int frameskip, int channel_mask);
  - show_processed controls what type of video is displayed.
    - 0 = raw video.
    - Non zero = processed video.
  - frameskip = # of frames skipped between updates.
    - Lower numbers = smoother video but more processing time.
  - channel_mask = determines which channels (0-2) are tracked.
    - A three bit binary number. The LSB = channel 0, the middle bit controls channel 1 and the MSB is channel 2.
    - Examples.
      - 0b111 = Track all three channels.
      - 0b101 = Track channels 0 and 2.
      - 0b001 = Only track channel 2.

Example of Live Video

```
#use "xbccamlib.ic"

void main()
{
    while(!b_button())
      {
        //Show processed video with 5 frames
   skipped and only show tracking data for
   channels 0 and 2
        track_show_display(1, 5, 0b101);
    }
}
```

Setting Camera Display Options

- See onscreen demo to learn how to set the XBC camera display options interactively.

White Balance

- Different light sources contain differing amounts of red and blue.
  - The sun and incandescent lights are much redder.
  - Fluorescent lights are much bluer.
  - Our eyes “auto correct” to different lights.
  - Cameras cannot.
    - Objects will “appear” as different colors under different lighting.
    - Big problem in Botball!
  - The XBC defaults to auto setting the white balance.
  - We can interactively adjust the white balance of the camera.
  - Use the Vision/Camera Config option from the XBC menus to set the white balance.
  - See on screen demonstration.

Setting White Balance Programmatically

- We can get and set white balance information via IC.
  - This rarely needs to be done.
- int camera_get_awb();
  - Returns a 1 if AWB is on, otherwise 0.
- int camera_set_awb(int enable);
  - Sets the AWB mode of the camera.
    - 1 = turn AWB on.
    - 0 = turn AWB off.

More on Setting the White Balance.

- int camera_get_wb_color_temp(int color[]);
  - Returns a 2 element array corresponding to the red and blue levels.
- int camera_set_wb_color_temp(int color[]);
  - color[] is a two element array which is the red and blue levels to use.
  - color[0] = red.
  - color[1] = blue.
    - 8 bit numbers (0-255).
    - Lower numbers filter out more of that color.
  - Returns a 0 for success and a –1 for failure.

```
// This program shows how to read and set the WB levels on the XBC

#use "xbccamlib.ic"

void main()
{
   int color[2]; // This two element array will hold the red and blue color levels

   display_clear();
   printf("press B to get the current WB componets\n");
   while(!b_button()){ }; // wait for b button
   beep();
   sleep(1.0);
   camera_get_wb_color_temp(color);  // Fills two element array with the current WB levels
   printf("Red=%d, Blue=%d\n", color[0], color[1]);// Print the current WB config levels

   printf("press B to set a new WB level\n");
   while(!b_button()){ }; // wait for b button
   beep();
   sleep(1.0);

   color[0] = 200;  // This is the red level, we will see the image with LOTS of red in it.
   color[1] = 0;    //This is the blue level, filter out all blue!
   camera_set_wb_color_temp(color);  // Set the levels
   printf("Red=%d, Blue=%d\n", color[0], color[1]);  // Print them again to show they have
     changed!
}
```

Color Model API's

- We can dynamically change the color models stored inside the XBC through IC.
- The XBC uses an HSV color model.
  - Hue = "color."
    - Red ~= 0, Green ~= 100, Blue ~= 240.
  - Saturation (range 0 - 223) is how pure and intense the hue is.
    - 0 = totally unsaturated, such as black, white, or gray; 223 = totally saturated, such as neon orange, fire-engine red.
    - Color distinction is more robust, for pixels with high Saturation.
  - Value (range 0-223) is how dark or bright the pixel is: 0 = black, 223 = bright.
    - color distinction is more robust, for pixels with high Value.

We Can See HSV Values Dynamically on the XBC

- See on screen demonstration of displaying HSV values for color models.

The Color Model array

- Four element array:
  - model[0] = hMin
  - model[1] = hMax
  - model[2] = sMin
  - model[3] = vMin

Setting and Retrieving Color Model Data.

- int color_get_model(int model_num, int model[]);
  - int model_num = color model number (0-2)
  - int model[] = Four element array to hold model data
- int color_set_model(int model_num, int model[]);
  - int model_num = color model number (0-2)
  - int model[] = Four element array to hold model data

```
//An example of dynamically reading and setting color models in IC on the XBC

#use "xbccamlib.ic"

void main()
{
   int model[4]; //Holds our color models
   color_get_model(0, model); //Fill our array with the current model data!
   /*
model[0] = hMin
model[1] = hMax
model[2] = sMin
model[3] = vMin
*/
  display_clear();

   //Print out color values!
   printf("H=(%d->%d) \nS>=%d\nV>=%d\n", model[0], model[1], model[2], model[3]);

   //set out color model array to a "blue" color
   model[0] = 201;
   model[1] = 256;
   model[2] = 161;
   model[3] = 100;

   printf("Changing color model!\n");
   color_set_model(0,model);  //Send the changes to the XBC
   printf("New color model:\n");
   printf("H=(%d->%d) \nS>=%d\nV>=%d\n", model[0], model[1], model[2], model[3]);
}
```

A Way to Do It Without Arrays!

- int color_get_ram_hmin(int model_num);
- int color_get_ram_hmax(int model_num);
- int color_get_ram_smin(int model_num);
- int color_get_ram_smax(int model_num);
- int color_get_ram_vmin(int model_num);
- int color_get_ram_vmax(int model_num);
- int color_set_ram_model(int model_num, int hmin, int hmax, int smin, int vmin);

```
//An example of dynamically reading and setting color models in IC on the XBC

#use "xbccamlib.ic"

void main()
{
   int model[4]; //Holds our color models
   color_get_model(0, model); //Fill our array with the current model data!
   /*
model[0] = hMin
model[1] = hMax
model[2] = sMin
model[3] = vMin
*/

   display_clear();

   //Print out color values!
   printf("H=(%d->%d) \nS>=%d\nV>=%d\n", model[0], model[1], model[2], model[3]);

   printf("Changing color model!\n");

   //send a new color model to the XBC!
   color_set_ram_model(0, 201, 256, 161, 100);
   printf("New color model:\n");
   printf("H=(%d->%d) \nS>=%d\nV>=%d\n", model[0], model[1], model[2], model[3]);
}
```

Tonight's Challenge (Continued From Session 7)

1. You should have the arm built.
2. Using what you know about IC, simple XBC vision, servos and motor control write a program that will:
    1. Seek out and find an orange ball.
    2. Grasp and pick up the orange ball.
3. The solution to last weeks challenge will be VERY helpful.
4. This is a big challenge, use incremental design!

Possible Sub-problems to Solve

1. Go out a fixed distance turn around and return [measure the repeatability by measuring the end points after careful positioning of the starting point and direction.]
2. Go out to a ball/tribble at fixed position, about 3 feet away, and grab it; return to starting point and drop it. [note that both grabbing and lifting is needed to return reliably with the object.]
3. Use vision to guide robot to a ball/tribble, about 3 feet away within the camera FOV, and grab it; return to starting point and drop it. [Set a color model to respond only to the target object; use the vision guidance function from the 6th class to direct the robot. Note the relation between the y track of a blob and how close it is.]