The Object Search Library
Ethan Y. Myers
Wesley Y. Myers
Cedar Brook Academy

The Object Search Library

1. Introduction

Object_search_lib is one of our many libraries that we have developed since we started botball. It holds the functions that use the XBC camera[1] to search for objects such as in this year's game[2] the tribbles and bins. These functions include ones that move the robot towards a colored object or target. Targets are objects with two colors. Other functions are used to center the robot on an object and to scan the table for objects of interest.

2. Find_Closest_Object

The function that supports most of the functions in *object_search_lib* is *find_closest_object* (FCO). FCO scans the list of objects returned by the XBC camera functions to determine the closest object as opposed to the largest object. During the list scan, FCO applies a variety of filters to find a suitable object. Three of these filters can be used to restrict where an object can be searched for in the camera's view and the remainder filter the object's characteristics. This function finds the object that fits a certain criteria that are set by parameters every time the function is called.

FCO has seven parameters. One parameter is for the color model of interest and the other six control the filters. The six filter parameters control four filter functions, three of which deal with regions in the camera's vision and the fourth deals with the object's size.

The first parameter is *color_model*. This tells the function which color model the caller is interested in. The third parameter is *min_size*. This check is an addition to the minimum size set by *track_set_min_area()* and allows us to use a size filter for each color model. The camera sees many blobs in its vision and this function enables the program to ignore all objects with an area smaller than the *min_size*.

The remaining parameters dictate which area of the camera's vision is to be ignored. The filters are based on the camera's coordinate system as shown in Figure 1. The camera uses an (x,y) coordinate system. The coordinate system origin (0,0) is located on the top left of the screen. The positive y-values go downward, which is different than a standard (x,y) coordinate system. The greatest x-value is 357 and the greatest y-value is 293. This makes the bottom right corner of the camera's vision on the coordinate plane (357,293).

The second parameter is *min_y*. *Min_y* tells the function to ignore everything below that value. However, since when going down the coordinate plane, the y-value increases, FCO is really ignoring everything near the top of the camera's vision.

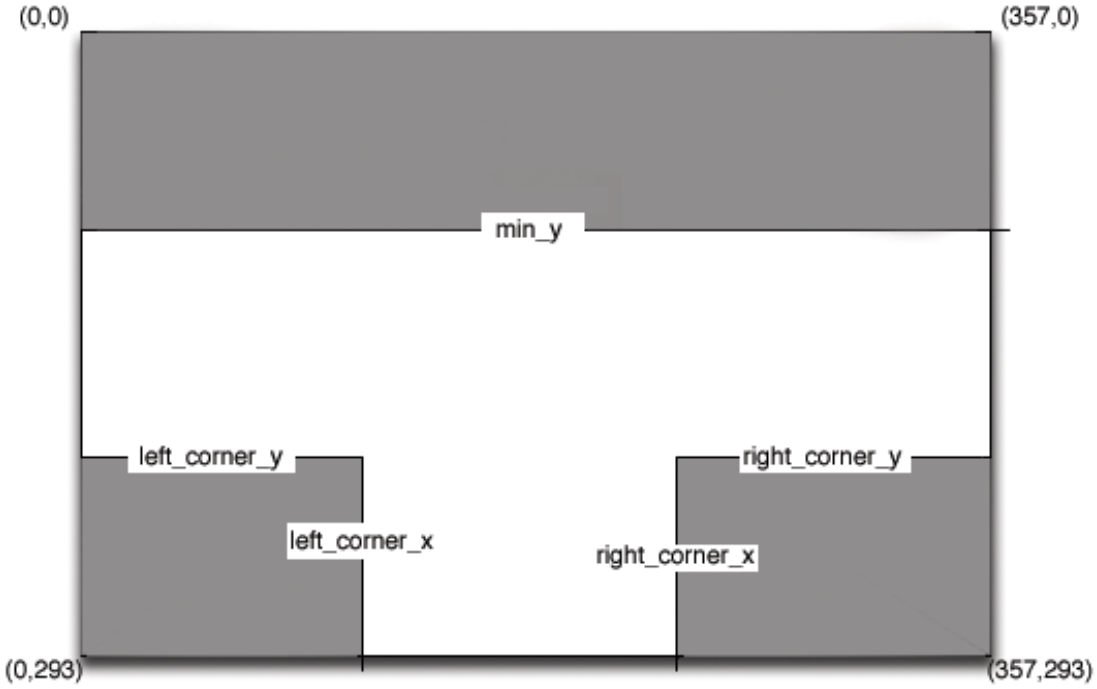

Figure 1

Similarly, two separate filters restrict the bottom left and bottom right of the camera's vision. The right corner filter is controlled by the fourth and fifth parameters, *right_corner_x* and *right_corner_y*. These two numbers form a box in the lower right corner of the camera. With these two variables, the function ignores anything inside that box. Similarly, the left corner is controlled by the parameters *left_corner_x* and *left_corner_y*. They have the same properties as *right_corner_x* and *right_corner_y* except that the box is now in the left corner. The filter relationships are shown in Figure 1.

The two bottom filters were added this year because one of our robots had two baskets to put tribbles in. If enough tribbles filled a basket, the camera would start to see them. Since there were two baskets of different heights and the desired object was to be located at the edge of one basket, we needed two different filters to cover these cases and still allow the camera to see the desired objects.

2.1 FCO Program

After all the parameters have been set, the function gets a blob from the list provided by the XBC camera functions. The filters mentioned above are applied to the object.

After these three filter tests, the program then checks if the object is "strong enough." These filters make sure that the object is not color noise caused by reflections on the table. It also checks to see whether it is the closest object by using the top y of the object and seeing it is closer than any before it. It also checks to see if the blob is a part of a target or marker and rejects it if it is. This year's game did not include targets or markers.

The program then checks to see whether the object is of interest by checking the object's width and height. If the object passes the width and height checks, the function then checks to see whether the ratio of the width and height are okay. These checks are used to determine if this object is noise. The program does this because sometimes the PVC pipe reflects one of the colors, so we get a long but thin object. A long thin object could possibly pass both the size and confidence tests. The ratio height and width test eliminates these types of reflections.

If the object does not pass all of these checks, that object is discarded and the next object is then tested. The process continues till all the objects have been checked. The function then returns the blob number of the closest object. Along with these checks, the program can provide printouts on the screen to explain why an object was not selected.

3. Go_To_Object

Go_to_object (GTO) is one of the main functions used with the camera. GTO uses FCO to locate an object and the GTO moves the robot towards that object. Like FCO, GTO has many parameters, but unlike FCO, many of these parameters are coordinate parameters. Its parameters move the robot so that the object becomes located in a very specific area of the camera's vision. The relationships of the coordinate parameters to each other and the object are shown in Figure 2 and are explained in the x-axis and y-axis parameter sections. FCO is used by GTO to find the object. Some of GTO's parameters are passed directly to FCO. The other parameters are used to control the movement of the robot to that object.

3.1 Filter Parameters

The function consists of six filter parameters. The ones described in this paragraph are passed to FCO for object searching. *Min_y* holds the value for which the camera ignores everything below that y-coordinate, which is the top of the camera's vision. *Min_size* gives the minimum area of the objects that the camera looks at. All objects below this size are ignored. The parameters *right_corner_x* and *right_corner_y* form a box on the right corner of the camera's vision, where objects are ignored. Similarly, the parameters *left_corner_x* and *left_corner_y* form a box in the left corner of the camera's vision, where any object in that box is ignored.

3.2 Y-Axis Parameters

GTO has two parameters that control the y-coordinate of the object on the camera screen. The parameter *distance* determines how close the robot gets to the object. In the camera's vision, the objects on the top are the more distant objects and the objects on the bottom are the closer objects. As the robot moves closer to an object, the object moves down the screen. The *distance* value is used on the y-axis. When the blob gets to that number, the robot stops. The *y_stop* parameter is used with the *distance* parameter. *Y_stop* is also a y-axis value. It determines which part of the object to be compared with the *distance* parameter. The *y_stop* parameter directs GTO to use either the top of the object, the center of the object, or the bottom of the object to determine the distance from

the top. This location is compared with *distance* until it is greater, at which point GTO is done.

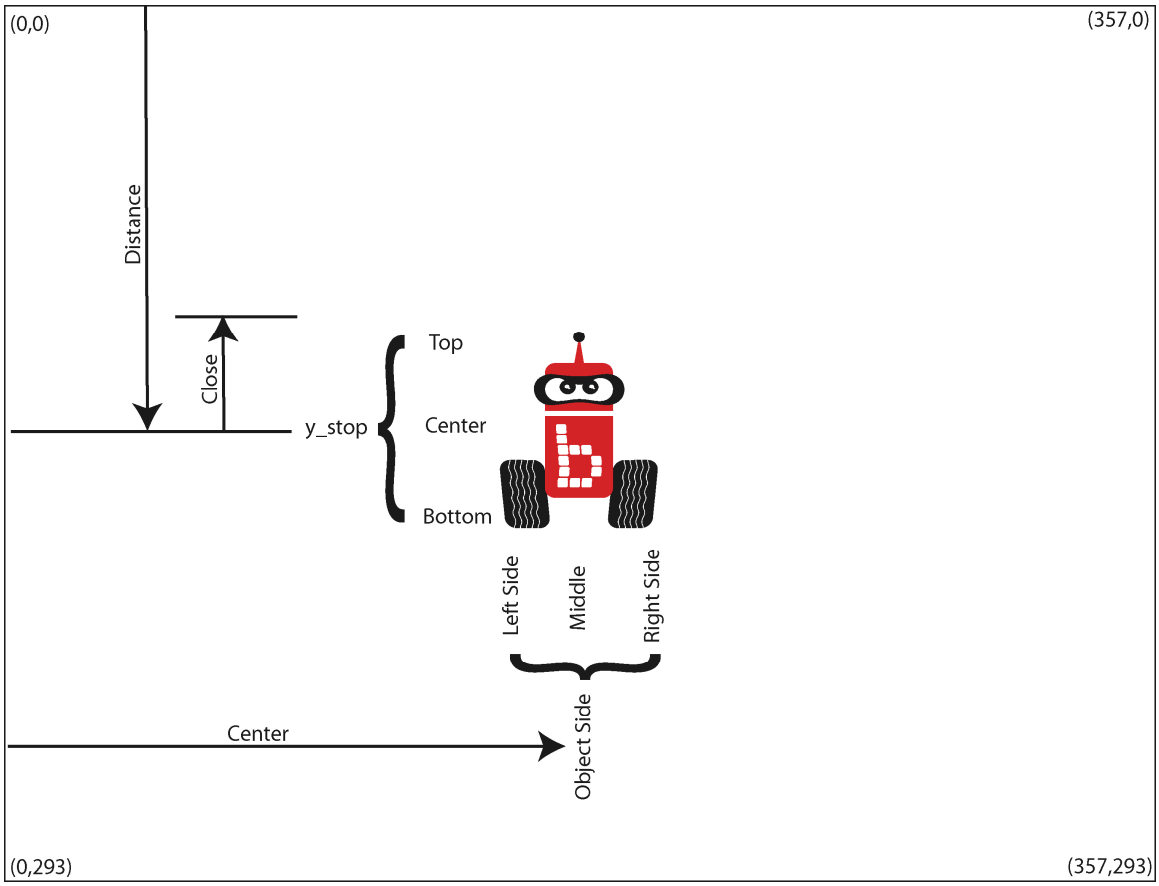

Figure 2

The next distance parameter is *close*. *Close* is very similar to *distance*. When the robot moves too fast towards an object, the robot might bump the object. *Close* is a relative value on the y-axis with respect to the object. When the object is within the *close* value, the robot then moves slower towards the object. This eliminates the possibility of hitting the object and causing it to move.

3.3 X-Axis Parameters

There are three x-axis parameters and they determine where the x-coordinate of object will be on the camera's screen. The parameter *center* determines where the object will be on the x-axis. Thus we can have the object anywhere on the left side of the camera's vision or on the right side. The *object_side* parameter is used in conjunction with the *center* parameter. *Object_side* determines the part of the object the robot is to center. *Object_side* causes either the middle of the object, the left side of the object, or the right side of the object to be used to center the object. The function maintains the object at this location on the camera as the robot moves toward the object, by varying the power to the motors.

3.4 Object Tracking

GTO also makes sure that the robot stops when the object of interest is "lost." This can happen if the object becomes obscured by the robot's structure or an opponent knocks it away. In either of these instances, FCO will lock on a different object and the robot might go too far. In the current implantation, if the object's y-coordinate takes a sudden reduction in value, the object is assumed lost and GTO stops the robot and returns to the caller with an error code to indicate that the object was lost.

3.5 GTO Program

The main section of the program is a continuous loop. At the start of the loop, FCO is called. If FCO returns -1, this means that FCO did not find any objects matching the parameters. In this case, GTO breaks out of the loop, stops the motors and returns to the caller. Otherwise, FCO returns the blob number of the closest object. The program then goes through the parameters to determine how the object is supposed to be located on the screen. GTO takes the blob number and moves the robot towards the object and attempts to keep the object located at the desired location on the screen. When the distance conditions are satisfied, GTO stops the robot and returns to the caller.

4. Other Object_Search_Lib Functions

FCO and GTO are the two main programs used in this year's game. However, the *object_search_lib* has many other functions that are useful. One of the functions is *go_to_target* (GTT) and was originally created for the 2005 game[4], where it was used to locate two targets on the table that had an inner color and outer color. It was also used for the 2006 game[3]. GTT is similar to GTO in its operation in that it moves the robot towards the target while keeping the center color of the target at *center*, the x-coordinate, and stopping at *distance*, the y-coordinate. It has only four parameters: *inner_color*, *outer_color*, *distance* and *center*. GTT is similar to GTO in its operation in that it moves the robot towards the target while keeping the center color of the target at *center*, the x-coordinate, and stopping at *distance*, the y-coordinate. It uses *Check_for_target* (CFT) to lock on to a desired object. CFT performs similar functions to FCO, except that it locates targets and returns the blob number of the center color of the target. The *inner_color* and *outer_color* parameters allow CFT to select which target is desired.

In 2006, the game used three different colored balls[3]. Our strategy was to take the balls off of the shelf and drop them on our side. To get extra points, the balls needed to be placed in a specific area based on the color of the ball. However, since the balls were dropped they would roll at random. The function *seek_any_object* (SAO) was created to compensate for this. The function of SAO is to find objects in a wider range than a single camera view. This is done by causing the robot to rotate, thus scanning the board.

SAO takes three color parameters to tell the robot which colors to search for. The robot turns until it has found an object. The function has two variations. The first is to look at colors in priority order. This was used in the 2005 game[4] where it was important to find Botguy first. Thus it would scan an area of the table, checking a single color at a

time returning immediately when an object is found. The second is to find any color object, which was used for the 2006 game. Once it finds an object it returns the object that it found. At this point, GTO is used to move the robot to the object.

5. Summary

Our Object Search Library is an important part of our code library in that it provides us with the ability to search for objects and move the robot to the object. This library has evolved over the past couple of years in response to the dynamics of each particular game. This and other libraries have provided us with the ability to quickly develop strategies for Botball games without having to deal with too much low-level code.

References

[1] R. LeGrand, K. Machulis, D. Miller, R. Sargent and A.Wright, "The XBC: a Modern Low-Cost Mobile Robot Controller," *Proceeding of IROS 2005*, IEEE Press, 2005.

[2] Anonymous, "Botball 2007 Teacher's Workshop Game Review," KIPR, 2007

[3] Anonymous, "Botball 2006 Teacher's Workshop Game Review," KIPR, 2006

[4] Anonymous, "Botball 2005 Teacher's Workshop Game Review," KIPR, 2005