

## **Making the Color Camera Effective**

### **1 Introduction**

Have you ever been stopped by a traffic signal in a slow part of town, no one around, but still the light stays red? We all like technology to be smarter, faster and more efficient, and in our growing economy, robots are becoming a larger part of everyday life. In Botball, my goal has always been to create a consistent and high scoring robot that depends heavily on the use of sensors. Many times I see other teams at Botball competitions whose robots do certain tasks by using only motor commands. My experience with robots and motor commands has been mainly frustration, low reliability and inefficiency, so I strive to take building and programming to a higher level by incorporating more complex programs that allow the robot to use the camera and many other analog and digital sensors. Even though the camera and other sensors might be harder to learn how to use at first and many other students would rather just use motor commands, I feel as though the effort it takes to learn how a sensor works with the robot and how it is programmed reflects the true spirit of Botball. Botball isn't about just how to build a robot that moves, but is about going deeper into robotics to explore how robots can become tuned to take in information from their surroundings and use that information in order to accomplish a certain goal or to solve a complex problem.

### **2 Why use Sensors like the Camera?**

In the 2014 Botball game, we designed a robot to locate BotGuy with the color camera, and also the Blue Block in order to maximize points. Although the robot itself has a simple design, with a camera mount pointed forward next to a servo that controls an arm with a claw for grabbing BotGuy, the camera increases the effectiveness and speed of the robot greatly. Most teams adopt a similar strategy to start the match with, which is race for BotGuy who is located in the middle of the board. However, who gets there first is not always the winner, for robots that use simple motor commands or just a touch sensor to find BotGuy might miss him because of a sensor failure, a simple misplacement of the robot in the starting box, or a slip in the motors to skew the path of the robot. Any of these might result in the failure to reach BotGuy, the robot going clear over to the other side of the board, or getting caught on the PVC piping that create a box around BotGuy. The usage of the color camera decreases the chance of these problems from happening, and when used properly, it is the most effective way to score with BotGuy

### **3 Programming the Camera**

In order to achieve our goal of capturing BotGuy, we started by creating a scan function that has the robot check for a blob and if it were to find none, move to the left a little and check again. The first iteration of our scan function, which was executed until a red blob was detected, appears below:

```

void scan_r ()
{
    motor (R, -40);
    motor (L,40);
    msleep (60);
    camera_update ();
    motor (R, 0);
    motor (L, 0);
    msleep (200);
    printf ("Botguy not detected\n");
}

```

Once a blob is detected, the program exits *scan* and goes to a second function called *track*. We put *track* inside a touch sensor-controlled loop where the touch sensor was activated by contact with BotGuy. *Track* consists of three IF functions within the while loop, one to turn right, one straight and one to turn left. These if functions use the camera's X Y coordinate plain to position the center of the blob or to see if the center is left or right of the center of view of the camera. The robot positions the center of the blob every time it checks the camera and sees if it needs to turn left, right or neither in order to keep the blob in the center.

```

void track_r ()
{
    printf("scanning...\n");
    camera_update ();
    if(get_object_center_column(0,0)>80)
    {
        motor(L,35);
        motor(R,10);
        camera_update ();
    }
    if(get_object_center_column(0,0)<65)
    {
        motor(L,10);
        motor(R,35);
        camera_update ();
    }
    if(get_object_center_column(0,0)>=65 && get_object_center_column(0,0)<=80)
    {
        motor(L,100);
        motor(R,100);
        camera_update ();
    }
}
}

```

In theory, these two functions together would work to find and move towards Botguy, but when they are put to the test, the robot performance is not quite adequate.

#### 4 The Camera Problem and Finding the Solution

When the robot went into the scan function, it failed to find Botguy when in scanned by pivoting to the left and checking the camera. When it didn't see red where it should have, the robot would continue pivoting, missing Botguy entirely. At first, once the robot missed Botguy, we would stop the program right away, thinking that it might be a problem with the color calibration in the CBC, or some simple error in the program. Once we were sure there were no problems in the program or calibration, we went on to figure out that if the robot is left to run the scan program through for nine or ten loops after scanning past Botguy, it would jump into the track program as if it had seen Botguy, making the robot overshoot the target outcome every time.



We began to isolate the problem by ruling out as many variables as we could. The programming was errorless, the connection between the camera and the CBC was good and the

color channels in the CBC were calibrated properly, and there were no red objects behind Botguy that might throw the scan off. At this point we had no explanation on why the robot exhibited such erratic performance. We concluded that when the camera updated, the CBC didn't process the data right away, but had a large delay that threw off the robot and made it miss its target. The next step for solving the problem then, was to find where in the program the fix needs to go or what line needs to be changed. We also wanted to make sure our solution was an innovative solution that solves the problem but also does not limit the function of the Camera or any other part of the robot.

## 5 The Solution

Our solution was to add a simple, yet effective line of code that makes sure the robot's delay doesn't interfere with its response time or any other function. At first we tried to add in multiple camera updates within the scan function, but this required the robot to take multiple pauses that, although increasing reaction time, it slowed the speed of the entire program by creating multiple pauses within the loop. With speed and efficiency in mind, this fix was not the optimum solution. However, when we put multiple camera updates together simultaneously, the robot can run through the loop, check the camera, and get good data without risk of any delay that might mess with the data coming from the camera. We created a new function that accomplishes this fix, and called it update camera multiple. This function is inserted wherever a normal camera update is needed.

```
void update_camera_multiple(int n)
{
    int i;
    for( i = 0; i < n; i++ )
    {
        camera_update();
        printf("Camera update %d of %d\n", i, n);
    }
}
```

The integer n is defined above the program, which gives the programmer quick control of how many camera updates the robot will run through

It is a simple fix, but effective, very efficient and fast. Rather than checking the camera once and then a delay might throw off the whole program, the robot runs through ten camera updates in less than a second, to ensure the robot will see what is in front of the camera.