Code Review Period 2 – Team 12-0075

Introduction

The code we are reviewing was written for our robot Lydia. In this section of code, Lydia is going to use its color-sensing camera and 2 DOF claw to detect the color of blocks and stack them properly (in addition to its other task of collecting poms in an effector-mounted bucket, but that is not the focus of this section of code). Jake Tango and Juan Vistro wrote this code so Xuelong Mu, Dan Wang, and Nathan Wolfe will be reviewing it. This review was conducted on February 27-28, 2012.

Best Practices Checklist

☒ Code uses functions to organize code

☒ Code includes comments documenting the purpose of each function

☒ Code includes comments documenting the arguments and return values of each function

☒ Variable names are descriptive and convey their use in the program

☒ No unnamed numerical constants other than 0, 1, or 2

☒ Code is appropriately formatted to show flow of control

☒ Comments do not contain blocks of old code that is no longer in use.

There are currently no checklist criteria that our reviewed code fails to meet. Originally there was one criterion which was not met by our code, that it was not appropriately formatted to show flow of control. However, our programmers Juan and Jake managed to catch this grievous error by themselves several days before this code review.

General Code Analysis

Reliability

While our robot Lydia, for which code is being reviewed, is only going to stay on our half of the board and thus will likely not come into contact with an opposing robot, there is always the possibility that another robot may cross onto our side and collide with Lydia. As such, a prominent example of error detection and recovery logic in the code we have reviewed is a method that uses the accelerometer contained within the CBC to constantly check whether or not another robot has collided with our own. When this method is “tripped,” it halts the action of any currently running method, including movement, and keeps track of the direction and duration of the collision so that once the opposing robot moves away, Lydia can correct its motion to go back to where it was before the collision and continue on with the halted method.

This code is currently reliable the majority of the time, but its accuracy can still be increased. As we test our robot more, we can better determine the amount of correction that is needed to account for the change in direction caused by a collision with another robot. Presently our robot can figure out around 75% of the time how much it needs to move to return to its original location, but we hope to increase that figure to up to 90% through further testing and fine-tuning of the code.

Maintainability

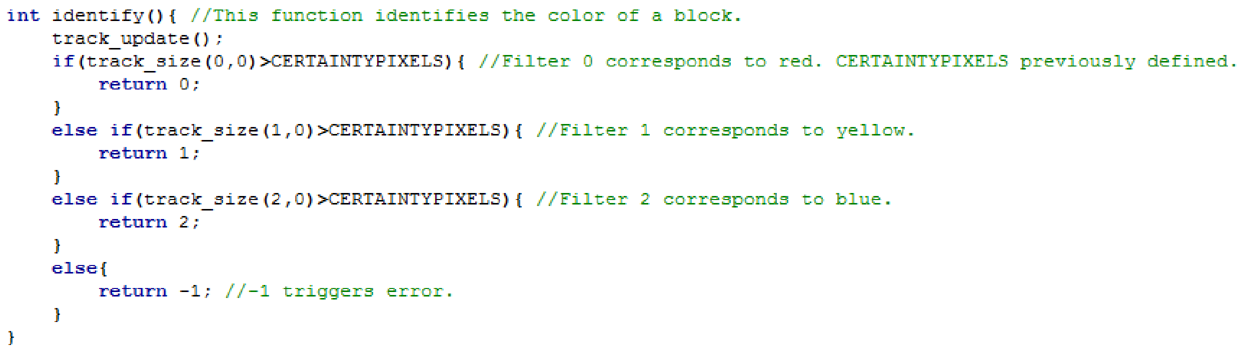
Jake and Juan have written most of the code for this robot (Lydia), but our team holds weekly meetings that bring the entire team together, both those directly involved in programming Lydia and those not, to go through the coding of the previous week and discuss how it relates to the rest of the program. In this fashion, all our team members are able to understand what is going on with Lydia's code. Since every week (typically on Mondays) people who do not necessarily actively program Lydia view its code, we have to ensure that it is easy enough to comprehend and modify as necessary. Our team captains require all our programmers to comment each function stating its purpose and execution, as well as additional comments concerning arguments and return types as applicable. As there may also be some lines of code that are particularly "wordy" or otherwise difficult to understand, we also encourage the use of in-line commenting (using the double slash the end of a line) in such cases. Thus, one programmer can carry on the work of another without too much confusion in the transition, which is greatly beneficial to the team as it allows people like Jake and Juan to share workload instead of having to write and tackle problems alone.

We consider our current standards of code maintainability to be fairly good, especially relative to previous years when there would sometimes be great confusion as to, say, the purpose of a method or why it even exists. One further improvement we could make is to mandate version documenting of edited programs, essentially noting after each edit the name of the editor and date, as well as a brief explanation of what was changed and for what reason. This would improve the maintainability of our code even more as two or three joint authors of a program will no longer necessarily have to talk in person to have some degree of communication, so we may soon implement that as an additional rule.

Effectiveness

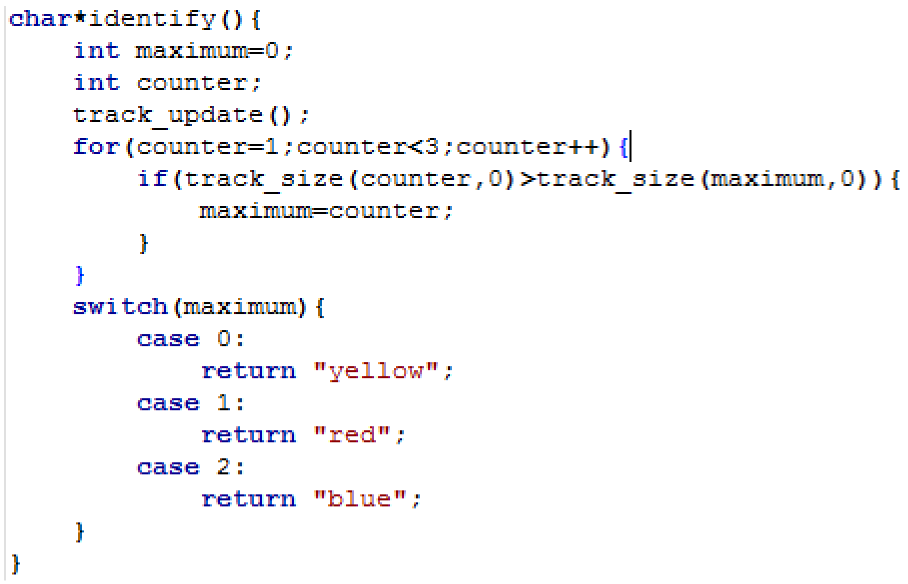
We've tested this code multiple times, and it consistently performs its task of stacking the blocks correctly. The only errors happen when the robot fails to identify the color of one of the blocks correctly. This code could be improved by expanding upon its color identification function to have better accuracy.

Here is an example as to how the color identification function can be improved to increase effectiveness. The current code:



```
int identify(){ //This function identifies the color of a block.
    track_update();
    if(track_size(0,0)>CERTAINTYPIXELS){ //Filter 0 corresponds to red. CERTAINTYPIXELS previously defined.
        return 0;
    }
    else if(track_size(1,0)>CERTAINTYPIXELS){ //Filter 1 corresponds to yellow.
        return 1;
    }
    else if(track_size(2,0)>CERTAINTYPIXELS){ //Filter 2 corresponds to blue.
        return 2;
    }
    else{
        return -1; //-1 triggers error.
    }
}
```

Unfortunately, the camera will not always pick up the same exact size of blob when it views the block, and thus produces a significant number of errors. This code produces none:



```
char*identify(){
    int maximum=0;
    int counter;
    track_update();
    for(counter=1;counter<3;counter++){
        if(track_size(counter,0)>track_size(maximum,0)){
            maximum=counter;
        }
    }
    switch(maximum){
        case 0:
            return "yellow";
        case 1:
            return "red";
        case 2:
            return "blue";
    }
}
```