Greater DC/Virginia Region

<u>**Period 2 Code Review Document**</u>

<u>**Introduction:**</u>

The following code was designed for our primary seeding robot, named Scorpio. Scorpio's purpose is to collect, sort, and place the red and green poms and yellow crates into their respective color zones. The code for Scorpio was written by Sam Du and Jacob Barats and then later reviewed by Anne Michelle. The review was performed on March 18th, 2018.

<u>**Best Practices Checklist:**</u>

✔ Code uses functions in order to organize itself
X Code includes comments that document purpose of functions
X Code includes comments of function arguments and return values
✔ Variable names are descriptive and represent their in code usage
✔ Code generally avoids un-named numeric constants other than 0,1, or 2
✔ Code is formatted to show the flow of control
✔ Comments that contain portions of code that are no longer in use have been removed

Because we use very serious and obvious names, it would be redundant to document their purpose extensively. For example, in the function "forward(int inches, int power)" it is quite clear that forward(...) will cause Scorpio to move forward. The same applies to the arguments and return variables. It is clear that the argument int inches will decide how many inches Scorpio will move forward and that int power will determine the speed at which Scorpio does so.

<u>**Reliability:**</u>

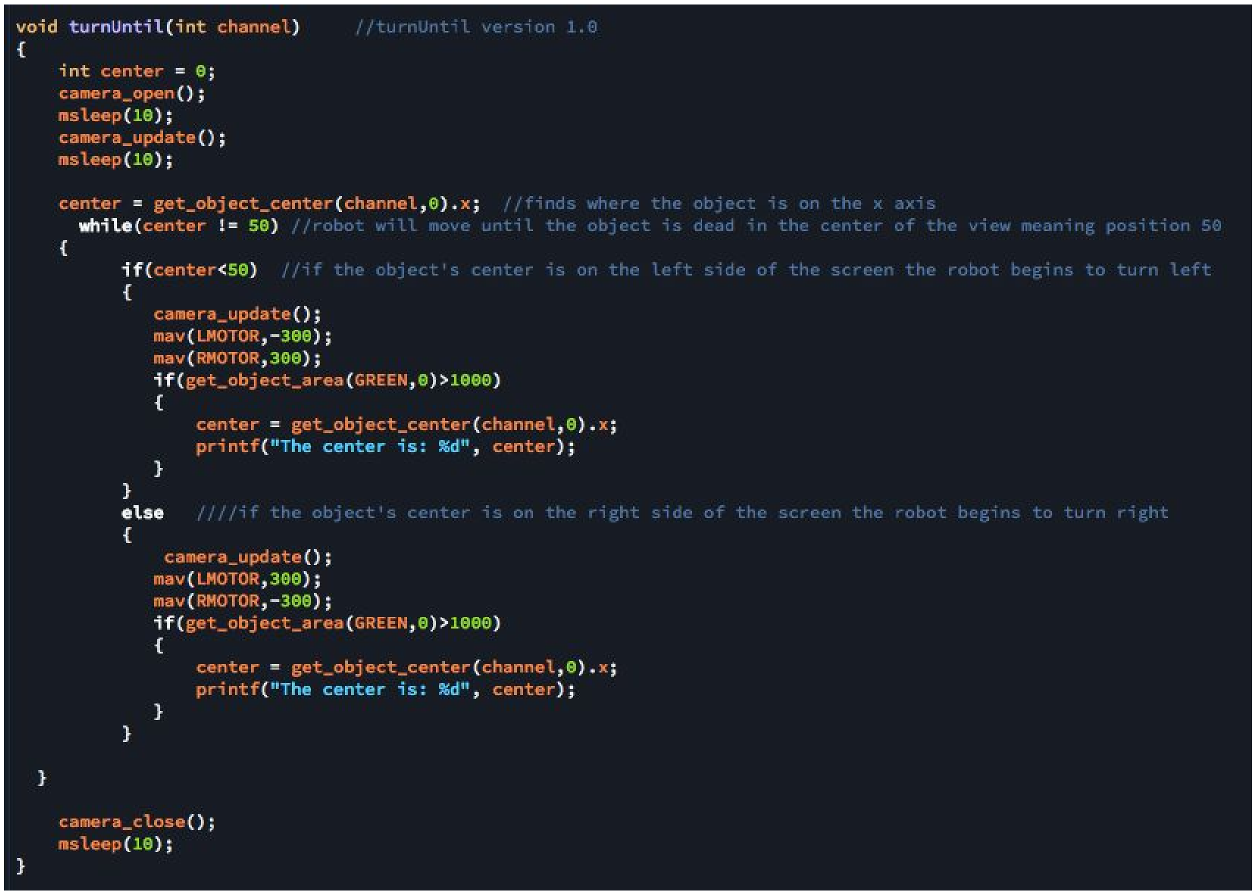
Because of the nature and unpredictability of our motors in precise movement, we have frequently enabled the camera for error detection and recovery. A common problem when turning toward the poms is that our robot would over or under turn because of battery charge levels. In order to solve this issue, once the turn is complete, we open our camera and use the "get_object_position().x" function to make sure that the pom pile is in a direct line of the robot. If the value that the function returns is less than 50 (the approximate center of our robot in the camera view) we activate the right motor with the "mav()" function and update the camera until the pile is at 50. If the value that the function returns is greater than 50, we activate the left motor and turn until the pile is at 50. One error that we sometimes encounter is that pom pile is at a position very close to 50 such as 49 or 51 and when the robot turns to correct itself it jumps over 50, resulting in an endless loop. In order to fix this issue, we should probably implement a range as opposed to a single value that would be acceptable for the x position of the pom piles.

Maintainability:

Because we have a very large team this year and multiple people working on Scorpio bot, maintainability of the code was very important to us. All of our functions are stored in a header file and this has a number of advantages. For instance, it makes the main code much easier to read, as one does not have to scroll all the way down to see the code that needs to be edited. Another advantage is that if multiple programs are being run on the robot, it is easy to make sure that they have access to the same functions, as they can all be linked up to the header file. To keep the code itself simple, we use a system of comments and variable names. For every few chunks of code in our main, we write a comment that describes what the robot would be doing at that sequence in reality. In addition to comments, we try to avoid directly using numbers as much as possible and instead focus on using variables. This way, for instance, when a servo position value needs to be changed, it can be changed in one place, the variable, as opposed to every instance of that servo position. Our code is not perfect, though, and one issue comes from the different skill levels of coders who work on Scorpio. Some of our most advanced coders have used syntax such as “?” instead of traditional for loops, something that our newer members are not familiar with (we do plan to eventually teach them this). Although this syntax may be more efficient, it is canceled out by the fact that not everyone on the team is able to edit it, and thus we should revert to more basic syntax.

Effectiveness:

As stated previously, the purpose of Scorpio bot is to obtain three colored game pieces: red and green poms, as well as yellow crates and then sort them respectively into their sorting zones. By using a combination of movement based on camera functions, such as “get_object_area()” and “get_object_center(),” Scorpio has been highly successful in obtaining the red and green poms in particular. The usage of the camera to get to these locations allows Scorpio to adapt even when knocked off track, as the movement is not based on dead-reckoning. To get to different areas of the board that are significant distances away, we incorporate the top hat sensors to follow black lines, alternating between different motor powers based on the readings of the analog() tophats. One major area of difficulty so far coding wise has been obtaining the yellow crates. Because of the positioning of the yellow-crate claw in respect to the rest of Scorpio, we are unable to use the camera in order to align the claw to the crates. To get these crates, we currently have been dead-reckoning, using the motor tick functions “gmpc()” and “cmpc()” combined with motor functions such as “mav()” to align. In order to make the crate grab be more consistent, we could probably utilize the depth sensors run the motors in a while loop until we are at a proper reading or distance.

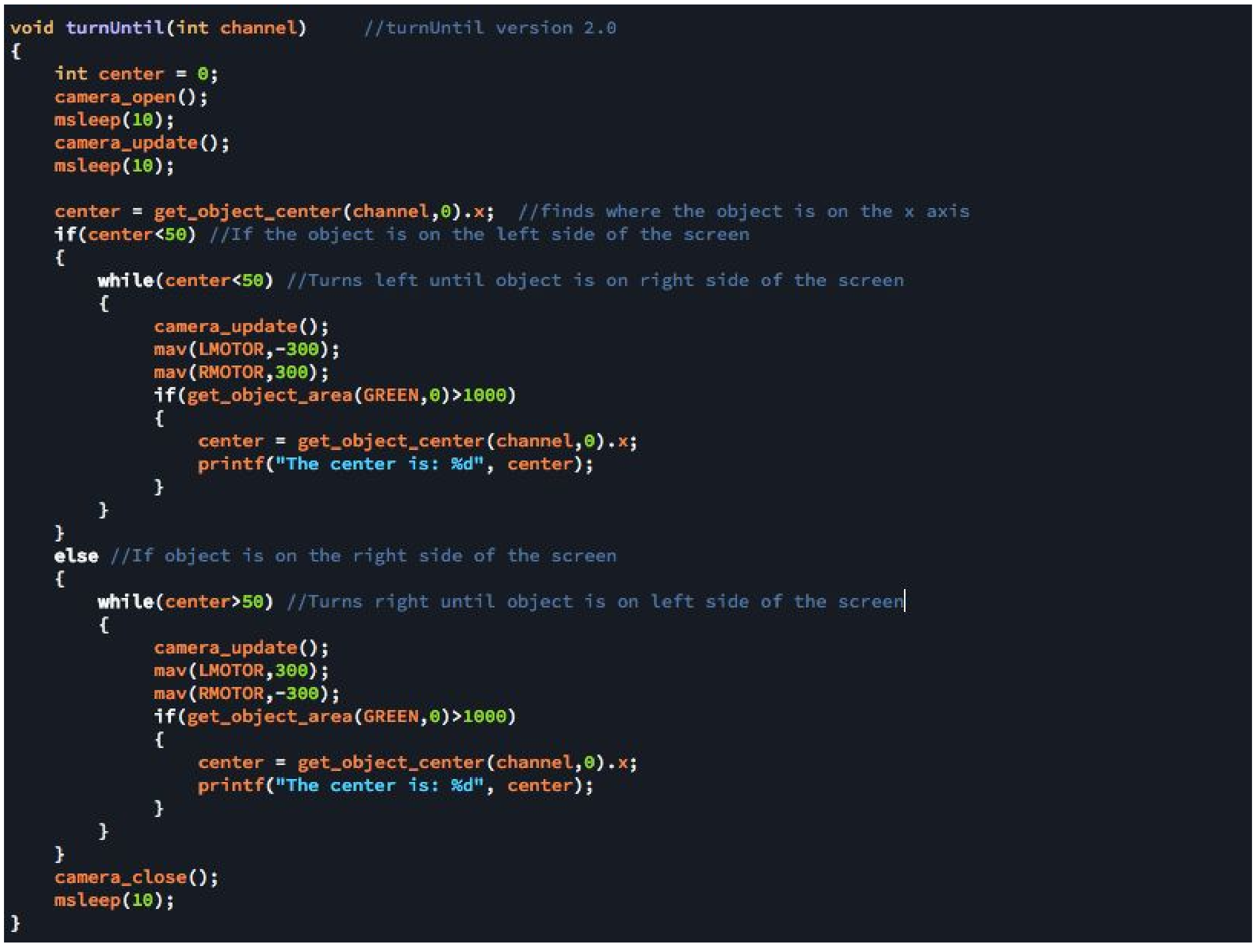


```
void turnUntil(int channel)     //turnUntil version 1.0
{
    int center = 0;
    camera_open();
    msleep(10);
    camera_update();
    msleep(10);

    center = get_object_center(channel,0).x;  //finds where the object is on the x axis
      while(center != 50) //robot will move until the object is dead in the center of the view meaning position 50
    {
          if(center<50)  //if the object's center is on the left side of the screen the robot begins to turn left
          {
             camera_update();
             mav(LMOTOR,-300);
             mav(RMOTOR,300);
             if(get_object_area(GREEN,0)>1000)
             {
                 center = get_object_center(channel,0).x;
                 printf("The center is: %d", center);
             }
          }
          else   ////if the object's center is on the right side of the screen the robot begins to turn right
          {
              camera_update();
             mav(LMOTOR,300);
             mav(RMOTOR,-300);
             if(get_object_area(GREEN,0)>1000)
             {
                 center = get_object_center(channel,0).x;
                 printf("The center is: %d", center);
             }
          }

  }

    camera_close();
    msleep(10);
}
```

Above is the inefficient code

(Scorpio will turn until the pom pile is in the dead center of the screen which almost never happens, causing the function to be more or less a never ending loop)

Below is the efficient code



```
void turnUntil(int channel)     //turnUntil version 2.0
{
    int center = 0;
    camera_open();
    msleep(10);
    camera_update();
    msleep(10);

    center = get_object_center(channel,0).x;  //finds where the object is on the x axis
    if(center<50) //If the object is on the left side of the screen
    {
        while(center<50) //Turns left until object is on right side of the screen
        {
            camera_update();
            mav(LMOTOR,-300);
            mav(RMOTOR,300);
            if(get_object_area(GREEN,0)>1000)
            {
                center = get_object_center(channel,0).x;
                printf("The center is: %d", center);
            }
        }
    }
    else //If object is on the right side of the screen
    {
        while(center>50) //Turns right until object is on left side of the screen
        {
            camera_update();
            mav(LMOTOR,300);
            mav(RMOTOR,-300);
            if(get_object_area(GREEN,0)>1000)
            {
                center = get_object_center(channel,0).x;
                printf("The center is: %d", center);
            }
        }
    }
    camera_close();
    msleep(10);
}
```

(Instead of turning until the pom pile is dead center, if the pom pile begins on the left side of the screen, Scorpio will turn left until the pile ends on the right side. If the pile begins on the right side of the screen the opposite movement will occur. This is much more effective as the pile has the entire range of half a screen to be in as opposed to a single value or few pixels.)