Explorer Post 1010
Botball Team 16-0162
Greater DC Region

Period 2 Software Design

Introduction:

The purpose of this code is to have the create obtain the potato bin, and then proceed to fill the bin with all of the water deposits and Martian dirt before bringing the potato bin back to the starting box. To accomplish this, we are using the create's built in sensors combined with an external camera to sense the Martian dirt and water deposits. The code and review were written by Jacob Barats on 3/25/16.

Best Practices Checklist:

The code...

- [yes] uses functions to organize code
- [yes]includes comments documenting each function's purpose
- [yes]includes comments documenting each function's arguments and return values
- [yes]uses descriptive variable names that covey their uses in the program
- [yes]generally avoids the use of unnamed numeric constants other than 0,1, or 2
- [yes]is appropriately formatted to show flow of control
- [no]does not contain comments with blocks of old code that are no longer in use

Currently, the program contains comments with blocks of old code that are no longer in use. The reason being is that there are multiple ways to obtain the water deposits and Martian dirt, and until our create is fully built, we will not know which method is the most effective. Currently we believe that a camera would be the most effective way to know when we reach the Martian dirt and water deposits, but in past iterations of the code, we planned to do this with a depth sensor. Once the robot is built, in the case that it becomes apparent that the depth sensor is a more viable solution for obtaining the water deposits and Martian dirt we will be able to uncomment that block of code and delete the camera code. If it turns out that the camera is more effective as we predicted, we will at that point delete the depth sensor code.

General Code Analysis:

Reliability:

The create uses a camera to detect once it is above a cluster of Martian dirt or water deposits. Once the camera sees the color configuration of Martian dirt or Water deposits, our robot stops and proceeds to scoop the cluster up. By using a camera to know when to stop the create, we do not have to time the exact point at which the create would reach the clusters, since the create would just keep going until it sees them. There are two main problems with using time to stop the create as opposed to the camera. Number one, the charge of the battery in the create could affect the speed the speed of the create, which in the end would change the time it would take to reach a cluster. Number two, the location of the clusters may vary slightly from board to board, which would slightly change the time it would take to reach clusters. In both of these scenarios, there would be no problem if we used a camera, since the create would just keep going until it saw the clusters.

Every now and then, the create responds a little too late after the Martian dirt or water deposits appear under it. To lower the chance of this occurring, we can increase the number of times the camera is updated while the create is driving. This will provide the create with a more current picture of what is occurring, allowing it to react faster.

Maintainability:

To make the code easy to understand, we have documentation for each function in the form of comments, which clearly explains the purpose of each function and the type of value it returns if any. For each complicated section of code in the main() function, we have comments that explain their purpose.

In the main() function, we try to rely on calling on functions as much as we can. This way if a portion of the code needs to be modified, it would only have to be modified in its function instead of many places in the main(). To keep our code reusable for future programs, we try to have each function run independently (meaning that it does not have to rely on other team made functions). That way if a future programmer on the team wants to include an older function from this program in their code, they only have to include one function.

One issue with our code is the fact that it may be hard to read and does not use good indentation and spacing. Since some of our functions do not have spaces between them, it can often be difficult to find the portion of code that you are looking for. To improve on this aspect of the code, spaces between the functions should be added, and the inner parts of the functions should be indented. This will add an overall flow to our programs making them easier to read and understand.

Effectiveness:

At the moment it seems that for the most part, the code will be able to perform the create's objective of collecting and placing the Martian Dirt and Water Deposits into the potato bin. Because the create robot itself is not fully built at the moment, we cannot have a definite answer on whether the code will work as a whole, but when run individually most of the functions and portions of the programs work. The camera is affectively able to detect when it has seen either Martian Dirt or a water deposit, and we are able to use degree tracking to move the create at very precise angles and directions.

The best way to make our code more effective would be to make it more sensor based as opposed to using time to determine when to make turns. Currently, whenever the create needs to make a turn we have code that makes it wait for a certain amount of time and then turn. To make the code more effective for un-ideal conditions, we are planning to make use of more sensors and landmarks as opposed to time. For example, if in the past we would have the create turn after 30 seconds, we would now have it turn when it would encounter a black line with the tophat sensor.

Specific Code Analysis:

```
// IMPLEMENTATION: turns to "exact" number of degrees specified. Parameter:
specified # of degrees sets normalized angle to 0. CCW increments this value
so spin until our normalized angle is greater than deg.
void goodTurnCCW(int degrees) // pretty sure this works {
     set_create_normalized_angle(0);
     while(get_create_normalized_angle() < degrees)        {
     create_spin_CCW(50); // change speed to turn faster
     msleep(10)         } }); // might be too short, check
     later
```

As you can see, this code is a mess. In the implementation, it is hard to see where a line begins or ends, because of the poor spacing and lining. In the body of the function, it is hard to see what is a comment and what is an actual line of code. Finally, the function typically turns the create twice the amount of degrees that was inputted.

Improved Code:

```
// IMPLEMENTATION: Turns to "exact" number of degrees specified.
// Parameter: specified # of degrees
// Sets normalized angle to 0. CCW increments this value so spin until our
//normalized angle is greater than deg.
// no value is returned

void goodTurnCCW(int degrees)
{
  set_create_normalized_angle(0);
  while(get_create_normalized_angle() < degrees/2.1)
  {
    create_spin_CCW(50); // change speed to turn faster
    msleep(10); // might be too short, check later
  }
}
```

As you can see, the code is now much cleaner and easier to read. It easy to differentiate from the comments and the main body of the function. Proper spacing has been applied to the inside of the function, to make it easier to understand what is part of the while loop. After some investigating, I discovered that the reason that the robot was overturning was because the degree counter built into the create was off by a factor of 2.1. To solve this issue, I divided the degree value by 2.1, making the turn of the create equal to the actual degree that was inputted.