

This example would score 100 points out of 100.

P2 Code Review Document

Introduction

The code we are reviewing was written for our main robot MONTE. MONTE is going to collect all of our opponent's poms using a 7 DOF arm and then bring them back to our side and deposit them in the scoring zone. MONTE's code was written by Howard, so Leonard and Rajesh are performing the code review. The review was conducted on 03/04/2011.

Best Practices Checklist

- ☒ Code uses functions for organization
- ☒ Code contains comments documenting each function's purpose
- ☒ Code contains comments documenting each function's arguments
- ☒ Code contains comments documenting each function's return values
- ☒ All variable names are descriptive and convey use in code
- ☐ No unnamed constants other than 0,1, or 2
- ☒ Code is formatted to show flow control
- ☒ Removed commented out code that is no longer in use

Our code has several constants that are called once, so we decided not to make up names for all of our constants. This keeps our code cleaner and easier to understand. All unnamed constants are described in comments. The constants that are used in more than one location are named and defined at the start of our code.

General Code Analysis

Reliability

While the motor position counters are accurate to some extent, we wanted to have a redundant system to externally measure the distance traveled. To accomplish this our robot has two perpendicular wheels with low friction. These wheels have slots in them that trigger our U shaped sensors, turning them into external encoders. In our code we are checking the motor position counters against the appropriate encoder to see that it is in an acceptable range, indicating that we have traveled a desired distance. If the encoders or the motor position counters return values that are not in the acceptable range of values the program runs a correction routine that does a weighted average to determine how to correct the position.

This code is currently pretty reliable, but the accuracy can be increased. The more we test the robot, the more we can fine tune the correction routine. Currently our accuracy is plus or minus 3 inches of the intended target, but we hope to lower that to 1 inch through future testing.

Maintainability

While Howard has written most of the code, our teacher requires all members of the team to read through the code once a week. We do this so that all the team members have at least a rough understanding of the code. Because we do this weekly, our code has to be easy to understand and modify. We like comments at the beginning of each function stating its task, and additional comments in the functions to clarify any unique or hard to understand lines. We also rely very heavily on versioning our code. Every change is documented at the top with a quick note about the change, author's name and date. Over the years we have built some functions we reuse, so it is important that we can reuse code in the future. We also keep backups of our code to prevent disasters, like what happened last year.

We think that we are already have really good code maintainability. From here our next step would be to an online repository, like github, that will manage the versioning better and still be available at all times.

Effectiveness

Our code correctly preforms our task. Where we struggle is in effectively completing tasks. Howard wants to show off his programming skills by using structs and arrays when variables will often work faster and better.

The following example shows Howard using arrays to hold the left and right motor assignments because he can:

Comment [A1]:

1. Clearly labeled Introduction

Comment [A2]:

2. Purpose of code

Comment [A3]:

3. Who wrote code and who reviewed

Comment [A4]:

4. Date of review

Comment [A5]:

5. Clearly labeled Best Practices Checklist

Comment [A6]:

6. Checklist item 1

Comment [A7]:

7. Checklist item 2

Comment [A8]:

8. Checklist item 3

Comment [A9]:

9. This and the previous item make up checklist item 4

Comment [A10]:

10. Checklist item 5

Comment [A11]:

11. Checklist item 6

Comment [A12]:

12. Checklist item 7

Comment [A13]:

13. Why checklist item 5 is not checked off

Comment [A14]:

14. Clearly labeled Reliability

Comment [A15]:

15. Discussion on error detection

Comment [A16]:

16. How reliability can be improved

Comment [A17]:

17. Clearly labeled Maintainability

Comment [A18]:

18. Is code easy to understand, modify and reuse

Comment [A19]:

19. How maintainability can be improved

Comment [A20]:

20. Clearly labeled Effectiveness

Comment [A21]:

21. How effective and correctly code preforms
23. Coding example for support of analysis

This example would score 100 points out of 100.

This example would score 100 points out of 100.

```
/*-----  
move_straight function by howard 1/12/12  
-----  
Function drives straight and checks encoder and  
BEMF. It uses the correction_distance function to  
calculate error to correct for. Uses distance()  
and encoder() functions to check distance traveled.  
Takes speed and distance arguments, and returns 1  
when complete.  
-----*/  
  
int move_straight(int speed,int inches)  
{  
    while(correction_distance()!=encoder() && correction_distance()!=distance() && correction_distance()>= 1)  
    {  
        printf("making correction");  
  
        if(encoder()<=correction_distance() && distance()<=correction_distance())  
        {  
            move_at_velocity(motor[left],speed);  
            move_at_velocity(motor[right],speed);  
            printf("driving straight\n");  
        }  
  
        else  
        {  
            printf("stopping\n");  
        }  
        off(left[motor]);  
        off(right[motor]);  
    }  
  
    printf("Reached Destination");  
    return(1);  
}
```

To increase our efficiency we just need to convince our programmer that he is good, but what we need is to have simpler more straight forward code that follows the KISS principle. In the Example below, we removed the array and replaced it with variables, making the code more effective and easier for the rest of us to understand and fix.

Comment [A22]:
22. Improving effectiveness of code
24. Example code showing improvements

```
/*-----  
move_straight function by howard 1/12/12  
-----  
Function drives straight and checks encoder and  
BEMF. It uses the correction_distance function to  
calculate error to correct for. Uses distance()  
and encoder() functions to check distance traveled.  
Takes speed and distance arguments, and returns 1  
when complete.  
-----*/  
  
int move_straight(int speed,int inches)  
{  
    while(correction_distance()!=encoder() && correction_distance()!=distance() && correction_distance()>= 1)  
    {  
        printf("making correction");  
  
        if(encoder()<=correction_distance() && distance()<=correction_distance())  
        {  
            move_at_velocity(left_motor,speed);  
            move_at_velocity(right_motor,speed);  
            printf("driving straight\n");  
        }  
  
        else  
        {  
            printf("stopping\n");  
        }  
        off(left_motor);  
        off(right_motor);  
    }  
  
    printf("Reached Destination");  
    return(1);  
}
```

This example would score 100 points out of 100.