Qatar

Team Name: Doha College
Team Number: 0282

Period Two - Code Review

Introduction:

The code for review is written for the small robot. The aim is to take the medical supplies to the disaster relief zone and the injured/uninjured people to the non-burning hospital. The code for this robot is written by Ibrahim. Sara, Salma and Saif are performing the code review. This review was conducted on 1/17/2019.

Best Practices Checklist:

- ✓ The code uses functions to increase organization and efficiency
- ✓ Reviewed code includes comments documenting each function's purpose
- ✓ Reviewed code contains comments documenting each function's arguments and return values
- ✓ Variable names in the reviewed code are descriptive and convey their use in the program
- ✓ Reviewed code generally avoids the use of unnamed numeric constants other than 0, 1, or 2
- ✓ Reviewed code is appropriately formatted to show the flow of control
- ✓ Comments do not contain blocks of old code that is no longer in use

To reduce redundancy, we have created named constants to set up predefined values at the start of the code. Any unnamed constants have comments to explain what they do. We may need to add more comments to make the code more descriptive and easy to navigate.

General Code Analysis:

Reliability

The small robot must approach its target in a fast and efficient manner. To achieve this, we used line following - we decided to use PID (Proportional Integral Derivative) which is a type of line following involving a calculation which updates a variable called MotorSpeed. The variable is updated based on an error calculation taking a previous reading as input to reduce correction time and the margin of error as it follows the black line.

Another way to improve reliability is to assess the functionality for the claw mechanism. While the servos position counters are accurate to an extent, we wanted to have a system which tested the claw by achieving the maximum and minimum positions for the servos. The code below demonstrates how the servos will move the claw mechanism. First, the robot will move forward while opening the claw to its maximum, it does this for 1 second and then line follows until it reaches a second piece of black tape. It then closes the servo position to its minimum position, waits one second and the moves directly forward. This will allow us to test how well the robot can move while it has objects in its claws. It then opens its claws and the program ends.

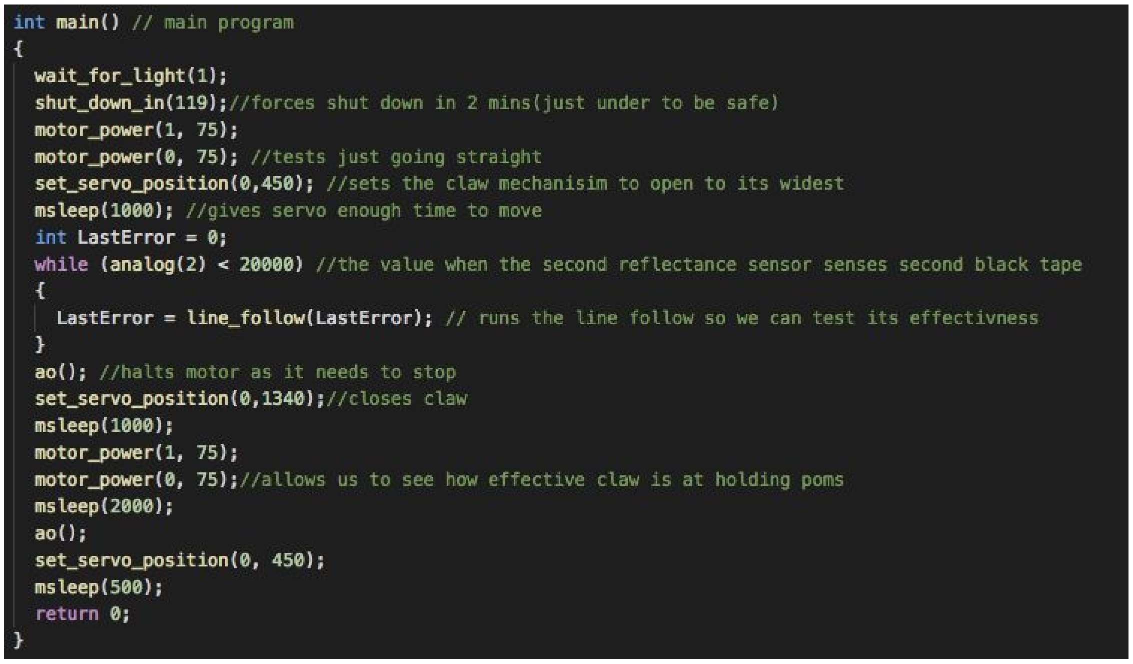
Maintainability

As a team, we are all required to read the code, hence, the code must include understandable statements. To achieve this, we need to include more comments and follow the linux kernel C coding style (https://www.kernel.org/doc/html/v4.10/process/coding-style.html) which was unanimously agreed on by the team. We are also running git version control on our school file system for version control. We used functional programming so that we could maximise code reuse. For example, the code used for the claw may also be used for the gripping mechanism of the iCreate bot. To improve maintainability, we will be using an editor called Atom, where the team can collaboratively edit the code and which will be available to us at all times.

Effectiveness

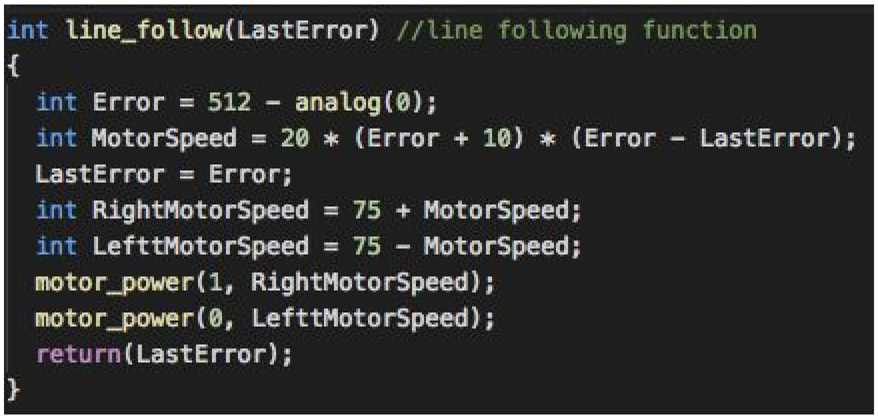
Our code correctly performs the intended task. To effectively complete this task, we think using compact functions will help improve the functionality and efficiency of the code.

The following example shows Ibrahim writing the code without subdividing the different tasks such as turning and going straight. This makes the code for PID difficult to follow and understand even with descriptive comments.



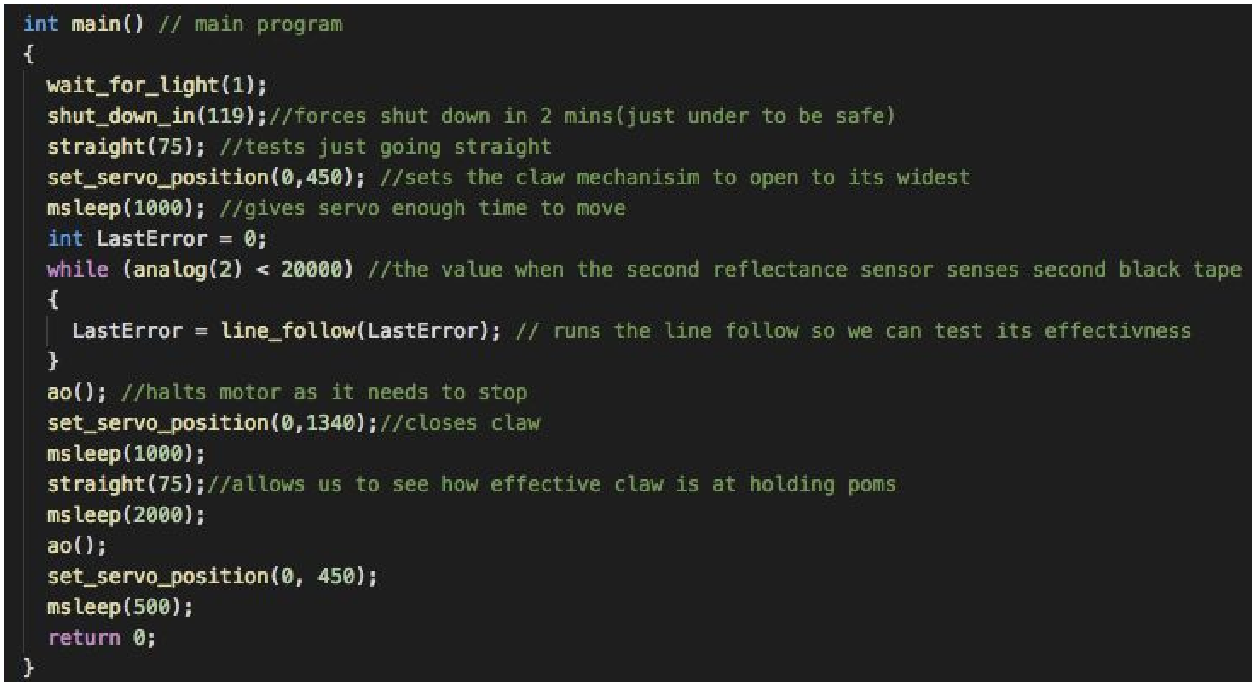
```
int main() // main program
{
  wait_for_light(1);
  shut_down_in(119);//forces shut down in 2 mins(just under to be safe)
  motor_power(1, 75);
  motor_power(0, 75); //tests just going straight
  set_servo_position(0,450); //sets the claw mechanisim to open to its widest
  msleep(1000); //gives servo enough time to move
  int LastError = 0;
  while (analog(2) < 20000) //the value when the second reflectance sensor senses second black tape
  {
    LastError = line_follow(LastError); // runs the line follow so we can test its effectivness
  }
  ao(); //halts motor as it needs to stop
  set_servo_position(0,1340);//closes claw
  msleep(1000);
  motor_power(1, 75);
  motor_power(0, 75);//allows us to see how effective claw is at holding poms
  msleep(2000);
  ao();
  set_servo_position(0, 450);
  msleep(500);
  return 0;
}
```

Qatar



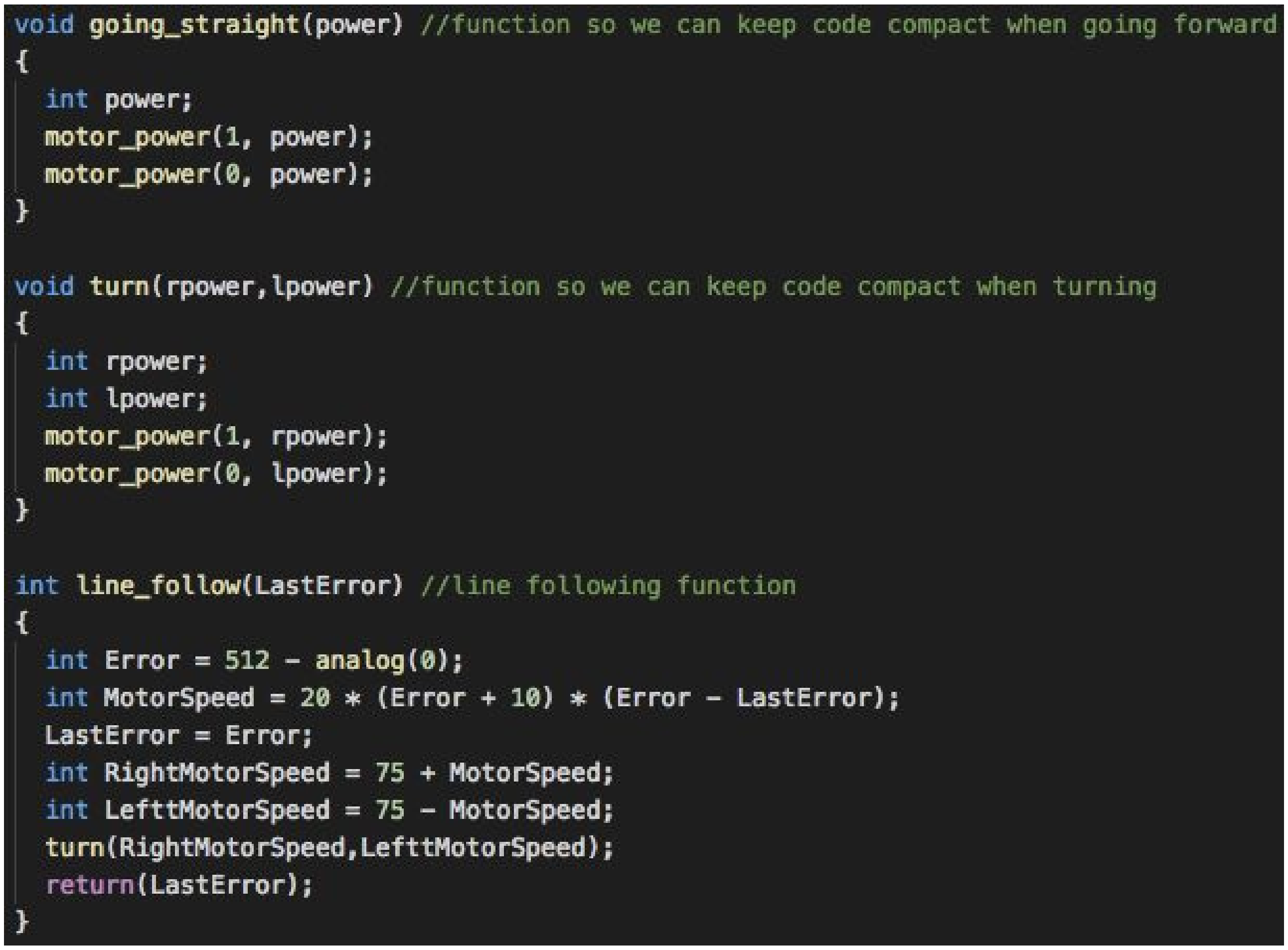
```
int line_follow(LastError) //line following function
{
  int Error = 512 - analog(0);
  int MotorSpeed = 20 * (Error + 10) * (Error - LastError);
  LastError = Error;
  int RightMotorSpeed = 75 + MotorSpeed;
  int LefttMotorSpeed = 75 - MotorSpeed;
  motor_power(1, RightMotorSpeed);
  motor_power(0, LefttMotorSpeed);
  return(LastError);
}
```

By reducing the amount of repeated code we are decreasing redundancy while also decreasing our risk of making mistakes. Furthermore it will make troubleshooting easier as we can adjust fewer lines of code and see the same differences. The more efficient and compacted code follows directly below:



```
int main() // main program
{
  wait_for_light(1);
  shut_down_in(119);//forces shut down in 2 mins(just under to be safe)
  straight(75); //tests just going straight
  set_servo_position(0,450); //sets the claw mechanisim to open to its widest
  msleep(1000); //gives servo enough time to move
  int LastError = 0;
  while (analog(2) < 20000) //the value when the second reflectance sensor senses second black tape
  {
    LastError = line_follow(LastError); // runs the line follow so we can test its effectivness
  }
  ao(); //halts motor as it needs to stop
  set_servo_position(0,1340);//closes claw
  msleep(1000);
  straight(75);//allows us to see how effective claw is at holding poms
  msleep(2000);
  ao();
  set_servo_position(0, 450);
  msleep(500);
  return 0;
}
```

Qatar



```
void going_straight(power) //function so we can keep code compact when going forward
{
  int power;
  motor_power(1, power);
  motor_power(0, power);
}

void turn(rpower,lpower) //function so we can keep code compact when turning
{
  int rpower;
  int lpower;
  motor_power(1, rpower);
  motor_power(0, lpower);
}

int line_follow(LastError) //line following function
{
  int Error = 512 - analog(0);
  int MotorSpeed = 20 * (Error + 10) * (Error - LastError);
  LastError = Error;
  int RightMotorSpeed = 75 + MotorSpeed;
  int LefttMotorSpeed = 75 - MotorSpeed;
  turn(RightMotorSpeed,LefttMotorSpeed);
  return(LastError);
}
```