Code Convention and version control help us to success in Botball
Zebu Lan
USTB Robot Society Team 1

Code Convention and version control help us to success in Botball

1 Introduction

Clean and well-structured code boosts the collaboration of a team. As we all know, team work is the most common and effective way to achieve an objective. In the Botball project, the robot programming work is commonly a team work. Writing clean and well-structured code and posing proper and essential comments help prevent the potential bugs that might occur in the program and testing stage. Well-structured code helps other people to quickly understand the code as well, which also help to debug. Although messy codes do not absolutely equal to more bugs and long debugging time, clean and well-structured code provided a good foundation for subsequent program optimization.

Based on clean and well-structured code, the maintenance cost can be reduced dramatically. Robot design is a process where robot mechanisms and structures would be adjusted or even rebuilt frequently. In this process, the computer program has to follow the changes as well. If the code is not under a clean and well-structured situation, the changing, code refactoring and corresponding test work will become a nightmare for everyone.

2 Code convention our team follows

A set of clear and simple code conventions can help a team to reduce most of the misunderstanding when team members are working together, and help to increase the maintainability and reliability of the code which is essential to the success in Botball. We simply list 5 rules need to be followed, which can prevent most of the common errors from happening.

2.1 Naming

Naming is critical to a program. A reasonable naming convention can help us to save the time which is spent on understanding the logic of the program. It can also effectively help us to make changes to the code.

For example, the small top-hat sensors were used in our robot to track the black lines. In our program, the variable which is the left sensor is named LeftSensorValue and right sensor is named RightSensorValue. Code blocks with good naming and bad naming are shown in Fig 2-1 and Fig 2-2. Although the code block in Fig 2-2 looks longer than the corresponding one in Fig 2-1, but it reduces most of the “magic values” which makes the code more understandable.

```
while(analog(0) <= 3800 && analog(1) <= 3800)
{
  mav(0, 500);
  mav(3, 500);
  msleep(20);
}
```

Fig 2-1 Code without good naming

```
const int THRESHOLD = 3800;
const int LEFT_MOTOR_PORT = 0;
const int RIGHT_MOTOR_PORT = 3;
int LeftSensorValue = analog(0);
int RightSensorValue = analog(1);
while(LeftSensorValue <= THRESHOLD && RightSensorValue <= THRESHOLD)
{
  mav(LEFT_MOTOR_PORT, 500);
  mav(RIGHT_MOTOR_PORT, 500);
  msleep(20);
  LeftSensorValue = analog(0);
  RightSensorValue = analog(1);
}
```

Fig 2-2 Code with good naming

In this way, people can clearly understand the meaning of the function and the debug process becomes relatively easier. Moreover, when there is any change of the hardware happening, i.e. the motor port is changed, programmers don't need to go through the whole program to change the port value, but only need to change a constant value, which helps to increase the maintainability of the code.

2.2 Indent

In order to improve the readability of the code, we have an indent convention in our team. Please refer to Fig 2-3 as an example.

It is quite clear that the code in this program has a good indent style. Code in main function and while loop indent to different level, marked with a pair of curly braces. In this way, people can easily understand under which code block a statement is, which helps our team member to read the code especially when there is nesting loops in our code.

```
const int LEFT_MOTOR_PORT = 0;
const int RIGHT_MOTOR_PORT = 3;
void drive_and_turn();
int main()
{
  int counter = 0 ;
  while(counter < 4)
  {
    drive_and_turn(500);
    counter = counter + 1;
  }
  return 0;
}
void drive_and_turn(int speed)
{
  mav(LEFT_MOTOR_PORT, speed);
  mav(RIGHT_MOTOR_PORT, speed);
  msleep(2500);
  mav(LEFT_MOTOR_PORT, -speed);
  mav(RIGHT_MOTOR_PORT, speed);
  msleep(2500);
}
```

Fig 2-3 Code with good indent style

2.3 Case

In our program we have a lot of different symbols, i.e. local variables, global variable, functions, constant values. To easily distinguish them from each other, we designed the following case convention in our team. Please refer to Table 2-1 for the detail.

Item	Case
Local variable	First word in lowercase, other words first letter capitalized
Global variable	First letter capitalized
Function	Lowercase words connected by underlines
Constant value	Capitalized words connected by underlines

Table 2-1 Case convention

Below are some examples from our code:

- Local variable: counter
- Constant value: LEFT_MOTOR_PORT
- Function: drive_turn()

By strictly follow the above rules, our programs are easier to read.

2.4 Function

When you find there are some pieces of code appear more than one time in your code, it indicates that you may need to encapsulate those code into a function. Using functions can make the code more reusable and easy to maintain.

When we use functions, we separate the declaration and the implementation of the function into two parts. The declarations of functions are placed before the main() function, while the implementations are put behind the main function. Fig 2-4 is an example of our function.

```
const int LEFT_MOTOR_PORT = 0;
const int RIGHT_MOTOR_PORT = 3;
void drive_and_turn();
int main()
{
  int counter = 0 ;
  while(counter < 4)
  {
    drive_and_turn(500);
    counter = counter + 1;
  }
  return 0;
}
void drive_and_turn(int speed)
{
  mav(LEFT_MOTOR_PORT, speed);
  mav(RIGHT_MOTOR_PORT, speed);
  msleep(2500);
  mav(LEFT_MOTOR_PORT, -speed);
  mav(RIGHT_MOTOR_PORT, speed);
  msleep(2500);
}
```

Fig 2-4 Example of user function

This program is used to make the car go as a square route, where the function “drive forward” was used only one time, however it may be used many times in other tasks. For this problem, using encapsulation can solve it perfectly. The design method is to subdivide the system functions into simple functional module, so that function statements generally are no more than fifty lines. From this program we can draw a conclusion that we should use modularization to encapsulate the function module as general functions where declarations are put in the file header and the implementation codes are put behind the main function, if it will be called more than two times.

2.5 Comments

Comments are useful when sharing work with teammate, and it also remind the programmer himself how code is originally designed to work. We normally put a short sentence at the beginning of a program to indicate the function of this program.

And comments are placed at main statements, i.e. loop statements, conditional statements. Fig 2-5 is an example of the comments we add to our code.

```
//Function prototypes for turn_left and turn_right
const int LEFT_MOTOR_PORT = 0;
const int RIGHT_MOTOR_PORT = 3;
const int TOUCH_SENSOR = 3;
const int TOP_HAT = 3;

void bot_line_following(int threshold)
{
  while (digital(TOUCH_SENSOR) != 1)//Loop: Is not touched?
  {
    if (analog(TOP_HAT) > threshold)//If: Is dark detected?
    {
      // Turnleft
      mav(LEFT_MOTOR_PORT, 200);
      mav(RIGHT_MOTOR_PORT, 1000);
    }
    else
    {
      // Turn right
      mav(LEFT_MOTOR_PORT, 1000);
      mav(RIGHT_MOTOR_PORT, 200);
    }
  }
  printf("Bot Line Following");
}
```

Fig 2-5 Example code of comments

2.6 Version Control

Although version control which refers to more sophisticated technologies and methodologies is normally not a code convention issue, we have our own simple but effective version control method in our team, which helped us to well manage our code during several months' Botball season. As the program in a Botball team is relatively simply, and the team is a small team compared with most software companies. We found it is not necessary to use CVS, SVN or Git to manage our code. Instead, we follow a naming convention of the C program files, to make the version management work easy to apply.

Our programs are named as the following style: Function_Author_Date_Version.c. For example, we meet up on 16th May and wrote a program to control the robot to walk up the slope and sweep the balls, we had 3 major updates and several minor updates during the day. The file name will be Walk_Sweep_Zebu_5_16_V3.1.c. In this manner we achieved well management of our code.

3 Conclusion

Follow several simple rules and write clear and consistent code is essential to Botball teams. It can help our team to really work in an engineer's way, which is the most meaningful experience we got during the whole process. No matter what programming language we use and how sophisticated our future task will be, following a proper designed code convention will largely facilitate team work and make the code maintainable and reliable.