A pattern language for the ET robot contest:

On embedded software engineering

MASASHI KADOYA†, TOSHIYUKI NAKANO†, TAKANORI OZAWA†, MASAHIKO WADA†,
HIROKI ITOH†, HIRONORI WASHIZAKI†, YOSIAKI FUKAZAWA†

We extracted a pattern language for the participants of the contest called the Embedded Technology (ET) robot contest. The ET robot contest is a software design competition and provides young engineers or students with an opportunity to learn embedded software engineering. Therefore, most participants are beginners, and they have difficulty designing the robot. In order to help the participants, we have published the extracted pattern language on a website. This pattern language, named ***ET Robocon Strategy***, consists of 40 patterns extracted from our technical knowledge and contest experience. In this paper, two patterns that an engineer can apply to developing a robot with wheels are introduced.

1. Introduction

The Embedded Technology (ET) robot contest [4] is a national software design competition and is hosted by the experts who gathered voluntarily. The purpose of the competition is to provide young engineer or students with opportunity to learn embedded software design.

The contest has two divisions: a racing division and a model division. In the racing division, participants design a two-wheeled robot to complete a predetermined course as fast as possible. The robot can stay on the course by using a light sensor to trace a black line on the course. However, the line may not be solid black throughout the course, or it may even be missing in parts of the course. Therefore, participants need to account for that when designing the robot. Two patterns introduced in this paper are relevant to this situation.

In the model division, the documentation and software design (specification document, class diagram, development process, etc.) are evaluated. [3]

The course consists of two parts: the *Basic Stage* and the *Bonus Stage*. The *Basic Stage* tests the robot's speed, while the *Bonus Stage* tests the robot's ability to successfully navigate obstacles, such as a seesaws, stairs, etc. In the DRIFT zone, which is part of the *Bonus Stage*, there is no a black line for guidance, and the robot must pass in between two plastic bottles. The two patterns introduced in this paper are especially important here.

The course changes from year to year. A portion of the course changed mainly is the Bonus Stage. However, there are common difficult points which mean the pattern can be adapted to multiple points. The examples are as follows.

- A portion of the course lacks a black line.
- There are bumps about 1 centimeter on the course.
- In specific contest sites, there is a strong ambient light.

In this paper, we introduce the patterns which are related to lacking a black line.

The robot is composed of only LEGO MINDSTORMS which is used for software engineering education. However, the kit of LEGO MINDSTORMS has some difficult points. For instance, a motor provided by LEGO MINDSTORM is not good performance. That problem has a deep relationship with the pattern shown later.

Figures 1 and 2 depict the 2012 course and a sample robot, respectively.

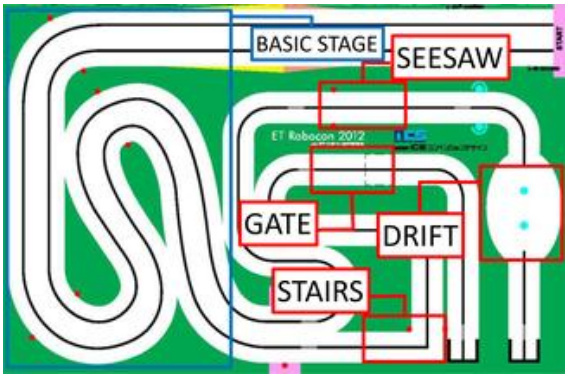


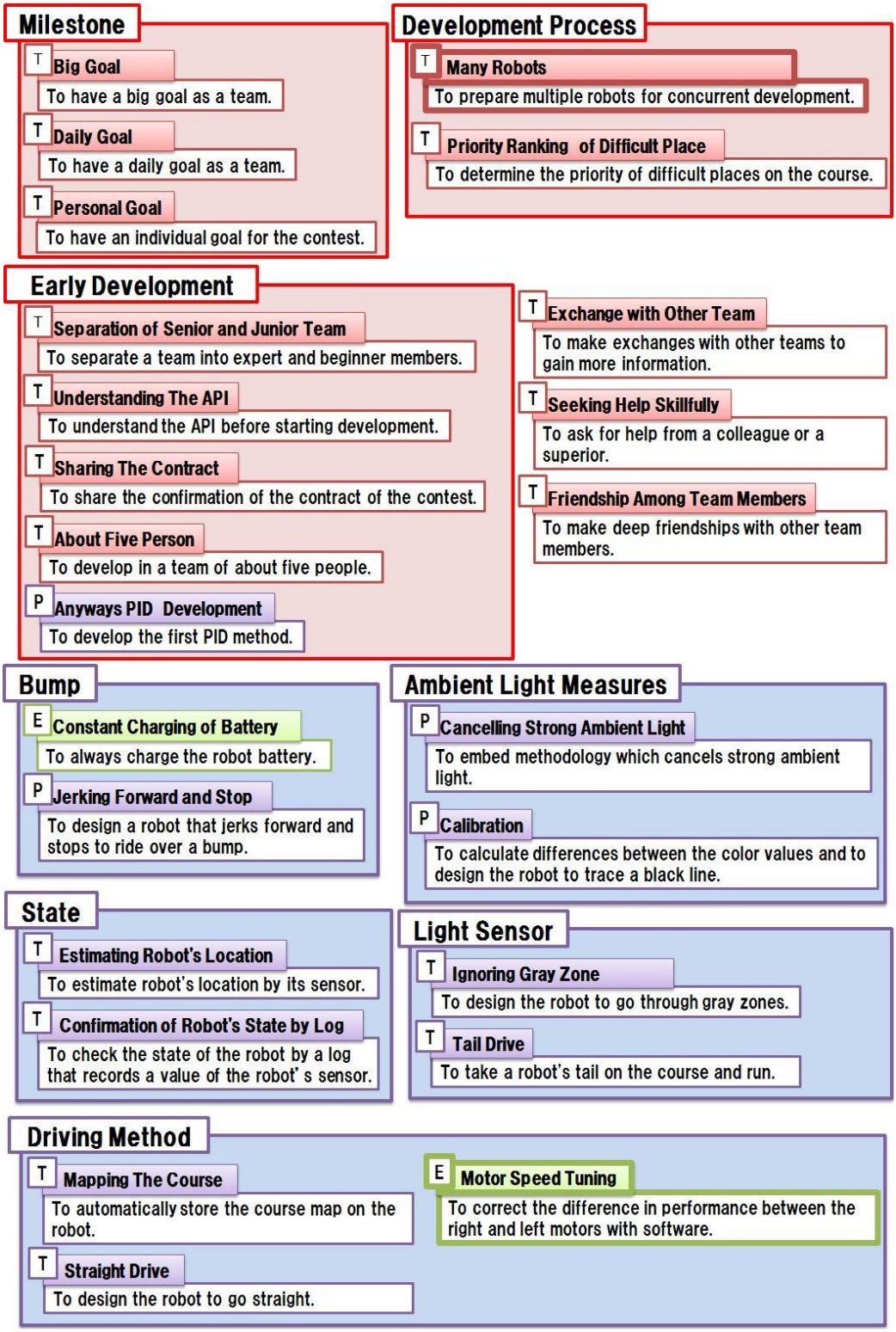
Figure 1 The 2012 course

Figure 2 A robot used in the 2012 contest.

2. The pattern language

From October 2011 to June 2012, a pattern language named ***ET Robocon Strategy*** was extracted from members of the Washizaki lab who participated in the ET robot contest via workshops and interviews. ***ET Robocon Strategy*** consists of 40 patterns. Each pattern is composed of Name, Picture, Context, Problem, Force, and Solution. If further explanation is necessary, we add related patterns and cases.

ET Robocon Strategy can be largely classified into four divisions: *Environment*, *Team*, *Model*, and *Programming*. *Environment* relates to failures of robots and the course setup. *Team* refers to issues of team members or the entire team. *Model* is the specific know-how to improve own model. *Programming* refers to the methodology to navigate the robot through the course. Herein, how ***ET Robocon Strategy*** functions to resolve these problems is presented. The map of ***ET Robocon Strategy*** is shown in Figure 3. The four divisions are represented by the first letter of each division's name. For example, "E" represents Environment. For better understandability, the map is divided into various groups related to each pattern.: ‘Milestone', ‘Develop Process', ‘Early Development', ‘Model', 'Bump', ‘Ambient Light Measures', ‘State', ‘Light Sensor', ‘A Way of Drive', ‘Ambient Light', ‘Course', and ‘Performance'. The two patterns introduced in this paper are indicated by heavy boxes.


Milestone
T
Big Goal
To have a big goal as a team.
T
Daily Goal
To have a daily goal as a team.
T
Personal Goal
To have an individual goal for the contest.
Development Process
T
Many Robots
To prepare multiple robots for concurrent development.
T
Priority Ranking of Difficult Place
To determine the priority of difficult places on the course.
Early Development
T
Separation of Senior and Junior Team
To separate a team into expert and beginner members.
T
Understanding The API
To understand the API before starting development.
T
Sharing The Contract
To share the confirmation of the contract of the contest.
T
About Five Person
To develop in a team of about five people.
P
Anyways PID Development
To develop the first PID method.
T
Exchange with Other Team
To make exchanges with other teams to gain more information.
T
Seeking Help Skillfully
To ask for help from a colleague or a superior.
T
Friendship Among Team Members
To make deep friendships with other team members.
Bump
E
Constant Charging of Battery
To always charge the robot battery.
P
Jerking Forward and Stop
To design a robot that jerks forward and stops to ride over a bump.
Ambient Light Measures
P
Cancelling Strong Ambient Light
To embed methodology which cancels strong ambient light.
P
Calibration
To calculate differences between the color values and to design the robot to trace a black line.
State
T
Estimating Robot's Location
To estimate robot's location by its sensor.
T
Confirmation of Robot's State by Log
To check the state of the robot by a log that records a value of the robot' s sensor.
Light Sensor
T
Ignoring Gray Zone
To design the robot to go through gray zones.
T
Tail Drive
To take a robot's tail on the course and run.
Driving Method
T
Mapping The Course
To automatically store the course map on the robot.
T
Straight Drive
To design the robot to go straight.
E
Motor Speed Tuning
To correct the difference in performance between the right and left motors with software.

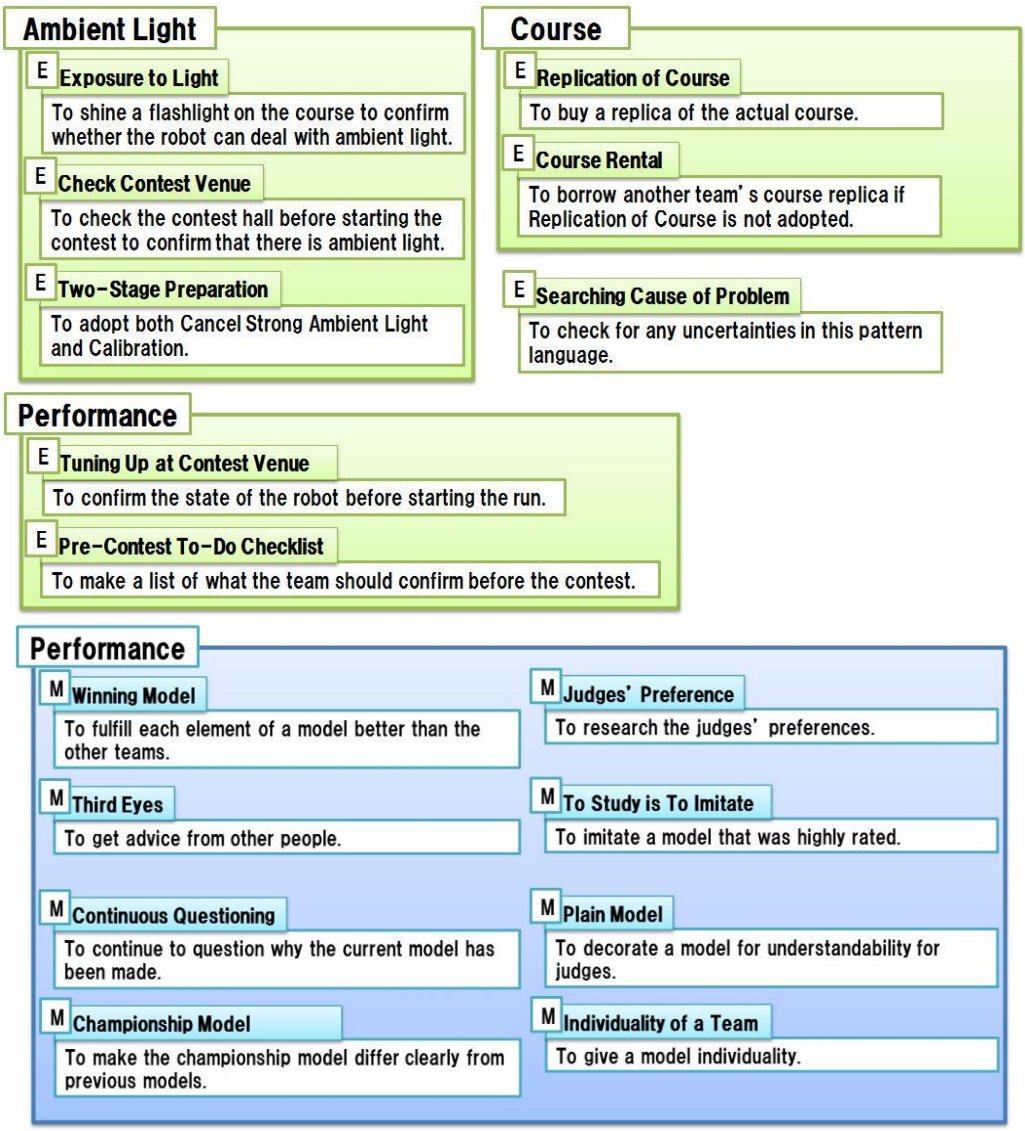
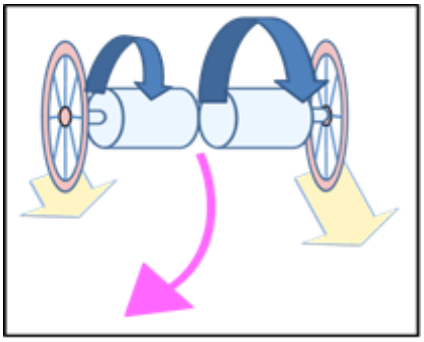


Figure 3 Map of ET Robocon Strategy

2.1. Motor Speed Tuning

This pattern is concerned with correcting the performance difference between the right and left motors using software.



Context

When going through a portion of the course lacking a black line, the robot must be designed so that it can get on the next black line without using its light sensor. In this situation, most developers would set the rotational frequency to be equal for the right and left motors to make the robot go straight. But a developer should tune each robot because manufacturing standards of LEGO MINDSTORM is low and all robots have different individual qualities.

Problem

Although the right and left motors are supposed to be turning equally according to the program, the robot does not go straight and pivots against the developer's expectations. This is because individual differences exist between motors.

Forces

- The performance difference between different motors is small. Although the developer tunes the robot properly、 the small difference will cause the robot to pivot slightly.
- Software can be edited, but Hardware can't be changed.

Solution

One solution is for the developer to correct the performance difference between the right and left motors using software. The most intuitive way may be to look at the motion of the robot and to adjust the rotational frequency of the motors by using software through trial and error. A more methodical approach that we recommend is to collect a log of the rotational frequency of the tires, and to adjust the robot settings based on this log. Figure 4 shows sample code.

```
float StraightDriver::steer(float velocity) {
        //get a value of a right motor.
        int rightWheelCount = getRightMotor().getCount() - initialRightWheelCount;
        //get a value of a left motor.
        int leftWheelCount = getLeftMotor().getCount() - initialLeftWheelCount;
        //compare a value of right and left motor.
        int countDiff = getRightMotor().getCount() - leftWheelCount;
        /*
        "degree" is positive value: rotate to the left for numerical value of "degree".
        "degree" is negative value: rotate to the right for numerical value of "degree".
        */
        if(countDiff > 0){
                degree = 5;
        }else if(countDiff < 0){
                degree = -5;
        }else{
                degree = 1;
        }
        return degree;
}
```

Figure 4 Sample code

Aside from using software, the developer can start by selecting a robot whose right and left motors have only a small performance difference.

Resulting

Our team was able to make a robot run in an almost straight line.

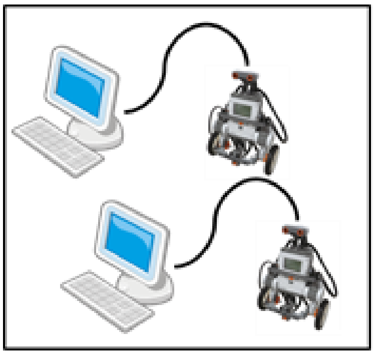
Applicability

This pattern can be applied to embedded development for operating a two-wheeled or four-wheeled robot.

◇◇◇

2.2. Many Robots

This pattern is concerned with preparing multiple robots for use in concurrent development when a team overcomes many difficult points which are such as seesaw, stairs and so on.



Context

A team possessing only one robot plans to conduct concurrent development.

Problem

If there is only one robot in a team, conflicts arise among team members when one member embeds a program in the robot. This is inefficient.

Forces

- Even if the development of one robot succeeds, the same program cannot be applied to other robots because there is a difference in motor performance.
- If a team does not plan to conduct concurrent development, preparing multiple robots is useless because almost no conflicts would arise among team members.
- Robots are not cheap. This pattern is only applicable to teams that have more than enough capital.

Solution

Having multiple robots in a team can make it easier to conduct concurrent development without causing conflict among team members. For instance, when a team starts a development of *Bonus Stage*, a team need to have a “seesaw” subteam and a “stairs” subteam. However, only one robot can run in the contest. The team needs to determine which robot is going to run in the contest early on in the development, or else it becomes difficult to decide which robot to finalize. Team members should recognize that the extra robots are to be used for testing purposes only, and that fine-tuning them is meaningless.

It is to learn something through experience that one robot is necessary for every 2-3 team members.

Resulting Context

Our team conducted concurrent development without any conflict using two robots.

Applicability

This pattern can be applied to concurrent embedded development.

3. Conclusion and Future Work

We have extracted a pattern language consisting of 40 patterns from our technical knowledge and contest experience. We applied two of these patterns (Motor Speed Tuning and Many Robots) to the 2012 ET robot contest. Our team, which consisted of 7 members, prepared two robots and was able to carry out the development without any conflicts among the members. If more robots were available, we may have been able to conduct the development more effectively. If our team consisted of 5 members, two robots are enough. We think that one robot is necessary for every 2-3 members. With respect to the difference in motor performance, we made an effort to eliminate it and the robot could run in an almost straight line. We believe a new method is needed to make the robot run perfectly straight.

This pattern language for the ET robot contest is still being developed. Currently, we are collecting input using Wiki and discussing a pattern language in workshops about the contest. In the future, we plan to extract a pattern language that can be applied to embedded development and to the development of the ET robot contest.

Reference

1) Gerard Meszaros, Jim Doble, “A Patten Language for Pattern Writing”, PLoP’97.
2) Kouichiro Eto, “Origin and Evolution of Wiki”, SIGHCI 2007.
3) Hironori Washizaki, Yasuhide Kobayashi, Hiroyuki Watanabe, Eiji Nakajima, Yuji Hagiwara, Kenji Hiranabe and Kazuya Fukuda, "Experiments on Quality Evaluation of Embedded Software in Japan Robot Software Design Contest," Proc. 28th International Conference on Software Engineering (ICSE 2006), May 2006.
4) “ET robot contest 2012”, http://www.etrobo.jp/2012/gaiyou/intro.php
5) “nxtOSEK”, http://lejos-osek.sourceforge.net/index.htm
6) Masashi Kadoya, Toshiyuki Nakano, Takanori Ozawa, Masahiko Wada, Hiroki Itoh, Hironori Washizaki, Yosiaki Fukazawa, “A pattern language for ET robot contest”, GuruPLoP, May 2013.