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†

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

1. Introduction

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

The contest has two divisions: a model division and a racing division. In the model division, documentation and software design (specification document, class diagram, development process, etc.) are evaluated. [3] In the racing division, participants design a two-wheeled robot, which must complete a predetermined course as fast as possible using a light sensor to trace a black line on the course. However, the line is not always solid and is even missing in parts of the course. Participants must account for these discrepancies in their robot design.

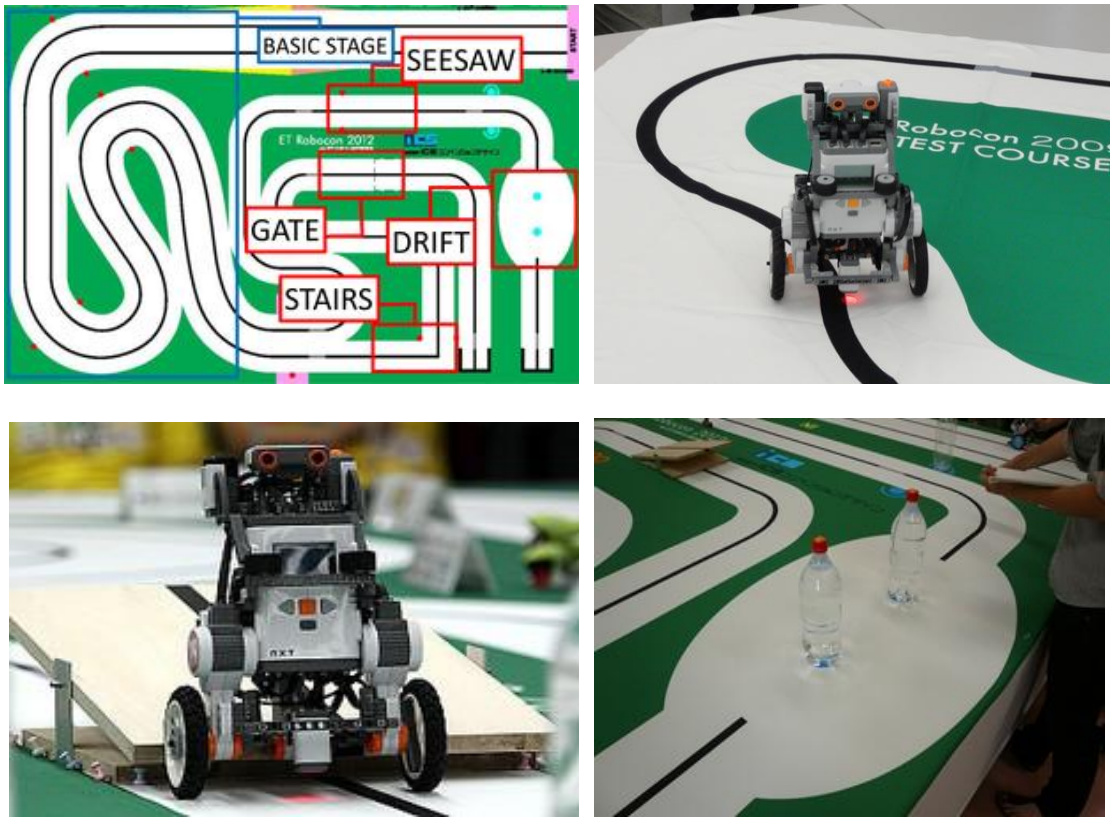
The robot must be composed of only LEGO MINDSTORMS, which is used for software engineering education. However, the LEGO MINDSTORMS kit has some disadvantages. For instance, the performance of the motor provided by LEGO MINDSTORM is not ideal for the contest, and greatly influences robot performance.

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 (e.g., seesaws, stairs, etc.). As part of the *Bonus Stage*, the DRIFT zone lacks the black guidance line and the robot must pass between two plastic bottles.

The course differs every year, but most changes are in the *Bonus Stage*. However, there are common difficult points, which means that patterns must be adaptable. Examples of difficulties include:

- A portion of the course lacks a black line.
- There are ~1-cm bumps on the course.
- At certain sites, the ambient light is very strong.

In this paper, we introduce 40 patterns that are part of a pattern language called ***ET Robocon Strategy.*** This paper will outline the pattern language and then will give the details of 5 of these patterns related to the absence of black line and navigating bumps: Replication of Course, Course Rental, Drive Straight, Mapping the Course, and Ignoring the Gray Zone. Two patterns introduced in this paper (*Drive Straight* and *Estimating Robot's Location*) are especially important for a portion of the course lacking a black line.



Figures 1　the 2012 course and a sample robot.

2. ET Robocon Strategy Pattern Language

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

The ***ET Robocon Strategy*** can be classified into four categories: *Environment*, *Team*, *Model*, and *Programming*. *Environment* relates to failures of the robots and course setup. *Team* refers to issues of team members or the entire team. *Model* is the specific know-how to improve one's own model. *Programming* refers to the methodology to navigate the robot through the course. Herein how the ***ET Robocon Strategy*** functions to resolve these problems is presented. Table 1 is the summary of the ***ET Robocon Strategy***. Each category is represented by the first letter of its name (e.g., "E" represents Environment). Each category is sub-divided into various situations related to the patterns: 'Milestone', 'Develop Process', 'Early Development', 'Model', 'Bump', 'Ambient Light Measures', 'State', 'Light Sensor', 'A Way of Drive', 'Ambient Light', 'Course', and 'Performance'. These situations help to make sure what problems can happen. Of these 40 patterns, the bold five are discussed in this paper.

Table 1 Summary of Environment Category of the pattern language

Pattern Name	Category	Situation	Contents
Always Charging	E	Bump	To always charge the robot battery.
A Difference of A Motor's Performance	E	A Way of Drive	To correct the difference in performance between the right and left motors with software.
Torment by Light	E	Ambient Light	To shine a flashlight on the course to confirm whether the robot can deal with ambient light.
Check Contest Hall	E	Ambient Light	To check the contest hall before starting the contest to confirm that there is ambient light.
Two-Stage Preparation	E	Ambient Light	To adopt both *Cancel Strong Ambient Light* and *Calibration*.
Replication of Course	**E**	**Course**	**To buy a replica of the actual course.**
Course Rental	**E**	**Course**	**To borrow another team's course replica if *Replication of Course* is not adopted.**
Tune Up in Contest Hall	E	Performance	To confirm the state of the robot before starting the run.
Starting Through Measures	E	Performance	To make a list of what the team should confirm before the contest.
Searching Cause of Problem	E	none	To check for any uncertainties in this pattern language.

Table 2 Summary of Programming Category of the pattern language

Pattern Name	Category	Situation	Contents
Anyways PID	P	Early Development	To develop the first PID method.
Jerked Forward and Stop	P	Bump	To design a robot that jerks forward and stops to ride over a bump.
Cancel Strong Ambient Light	P	Ambient Light Measures	To embed methodology which cancels strong ambient light.
Calibration	P	Ambient Light Measures	To calculate differences between the color values and to design the robot to trace a black line.
Estimating Robot's Place	P	State	To estimate robot's location by its sensor.
Confirmation of	P	State	To check the state of the robot by a log that records a value

Robot's State by Log			of the robot's sensor.
Ignoring Gray Zone Drive	**P**	**Light Sensor**	**To design the robot to go through gray zones.**
Tail Drive	P	Light Sensor	To take a robot's tail on the course and run.
Mapping Drive	**P**	**A Way of Drive**	**To automatically store the course map on the robot.**
Straight Drive	**P**	**A Way of Drive**	**To design the robot to go straight.**
Anyways PID	P	Early Development	To develop the first PID method.
Jerked Forward and Stop	P	Bump	To design a robot that jerks forward and stops to ride over a bump.

Table 3 Summary of Model Category of the pattern language

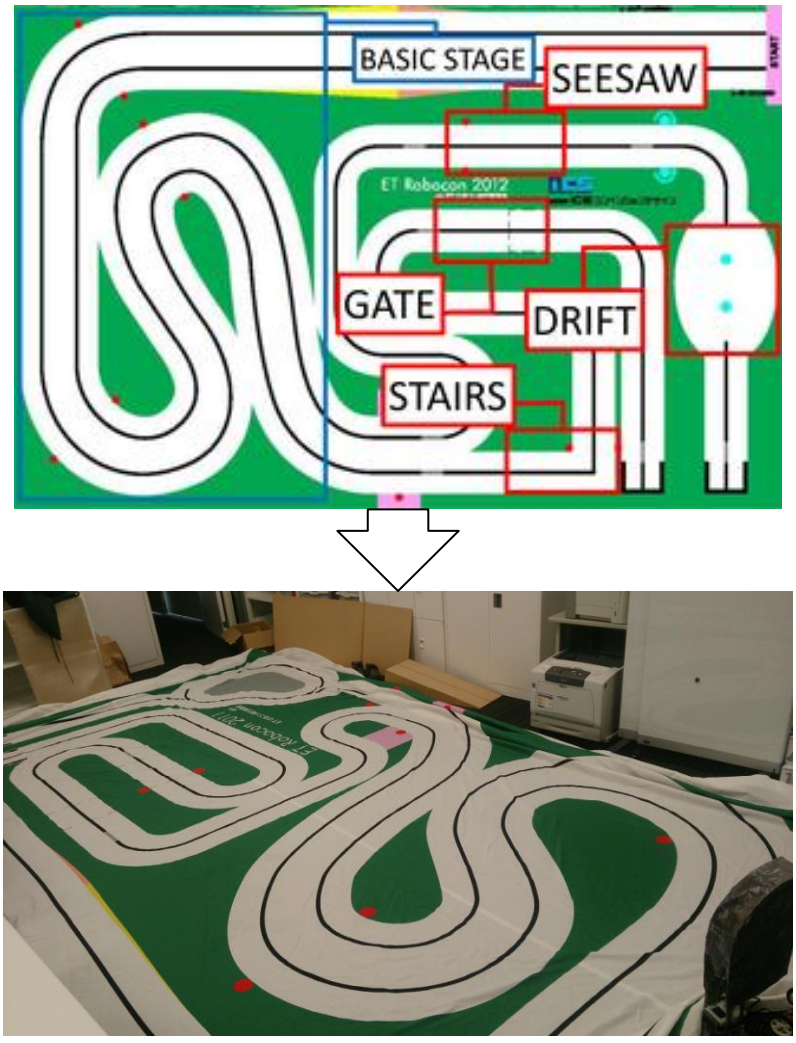
Pattern Name	Category	Situation	Contents
Winning Model	M	Model	To fulfill each element of a model better than the other teams.
The Third Eyes	M	Model	To get advice from other people.
Why Trace	M	Model	To continue to question why the current model has been made.
Championship Model	M	Model	To make the championship model differ clearly from previous models.
Judges Liking	M	Model	To research the judges' preferences.
To Study is To Imitate	M	Model	To imitate a model that was highly rated.
Plain Model	M	Model	To decorate a model for understandability.

Table 4 Summary of Team Category of the pattern language

Pattern Name	Category	Situation	Contents
Big Goal	T	Milestone	To have a big goal as a team.
Daily Goal	T	Milestone	To have a daily goal as a team.
Personal Goal	T	Milestone	To have an individual goal for the contest.
Concurrent Development with Many Robots	T	Develop Process	To prepare multiple robots for concurrent development.
Priority Ranking of Difficult Place	T	Develop Process	To determine the priority of difficult places on the course.
Separation of Senior and Junior Team	T	Early Development	To separate a team into expert and beginner members.
Understand The API	T	Early Development	To understand the API before starting development.
Share The Contract	T	Early Development	To share the confirmation of the contract of the contest.
About Five Person	T	Early Development	To develop in a team of about five people.
Exchange with Other Team	T	none	To make exchanges with other teams to gain more information.
Skillful Begging	T	none	To ask for help from a colleague or a superior.
Friendship of a Team	T	none	To make deep friendships with other team members.

2.1. Replication of Course

This pattern is concerned with creating a practice course that is nearly identical to the one used in the contest.



Context

The team is developing a robot to run on a specific course.

Problem

Without an accurate course to practice running the robot, a team cannot fully understand the course environment used in the contest and cannot develop good tests for the robot to prepare for the course**. So how can the team create the best environment to make sure they develop the robot the best and are prepared for the contest?**

Forces

- It is important that the team has enough space to prepare a practice course.
- A team has sufficient funding. ➔ Creating a separate course can be expensive.
- Not fully understanding the surface of the course causes issues with the robot's mobility because the course surface influences the robot's performance. Then, the robot will veer off course. If the surface of the course of the contest is harder than that of practice, the robot may act more than you can image. ➔ Although it is a serious matter that the team has enough space, setting up the ground corresponded to the course is very hard.
- Not fully understanding the course layout, it is difficult to accurately position the robot in the course. An

inability to recognize the robot's position in each *Bonus Stage* leads to mistakes. In particular, when the course lacks a black line, it is extremely difficult to apply the *Mapping the Course* pattern.

Solution

After ensuring that the team has sufficient space (5460mm×3640mm), **purchase and setup a replica course.** This should be as close to the competition course as possible. If the team doesn't have the space or money, then create a smaller course to at least partially replicate the course for testing.

Also, an important part of the solution is that the robot must move on a hard surface (e.g., a table) because surface hardness affects the robot's movements. Practicing on a surface covered with a soft material (e.g., carpet) is not recommended. Ideally, the replica course should be built on Styrofoam, which is the actual contest surface. Furthermore, it is important to smooth the creases in the course as much as possible.

As an example of this pattern, for our contest, we purchased for 140,000 yen (about $1,400 US dollars) the course shown in Figure 7 from Afrel (URL: http://www.afrel.co.jp/et2013_course.html). This course was setup and made available for the following period (e.g., 2013 years):

- First: from May 7 to May 14
- Second: from June 3 to June 10

PUT A Real PICTURE HERE

Figure 2

Results

For the 2012 contest, our lab purchased a replica course, allowing the robot to be accurately positioned on the course. However, we did not verify that there was sufficient space to build the course, and had to practice tuning the robot on carpet. Consequently, the *Mapping the Course* pattern was not accurately embedded, and the robot veered off

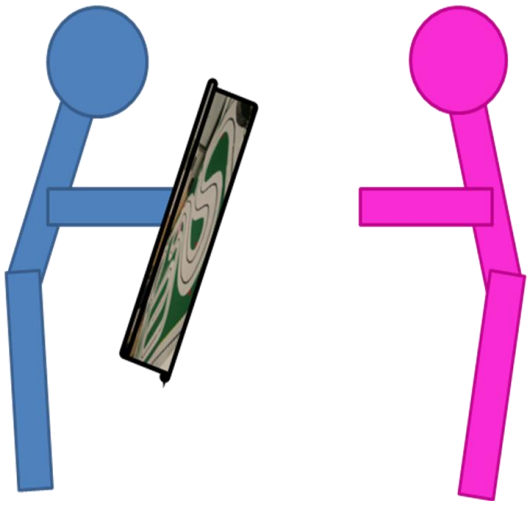
course during the contest.

Related Pattern

If a team lacks funding or the proper space to replicate the course, another option is *Course Rental*.

2.2. Course Rental

If it is infeasible to adopt *Replication of Course*, another option is *Course Rental* in which a practice course is rented from another team.



Context

A team does not have sufficient funds and/or space to replicate the course.

Problem

Similar to *Replication of Course*, the team must be able to get a course that they can build and test the robot on. However the team is lacking the resources or money to replicate or build their own course. **So how can the team get a course that can help them build and test their robot with?**

Forces

- A team can have a close relationship with other teams, and can exchange and/or share resources.
- Teams might be rivals and competing. ➔ Observing how other team works out their strategy leads to invent new idea.
- A team might not have time to build a replicated course. ➔ But a team can't understand accurate position of *Bonus Stage*.
- Replicating a course can be very expensive. ➔ If the team creates newly replicating course, the robot veers in the actual contest because of difference of reflection of light
- A course can take up a lot of space and the team might have limited space available. ➔ The team can rent space like conference room. But it still more costs money.

Solution

If two teams have a relationship, one team can rent the course from the other. This one-sided rental differs from an *Exchange with Other Teams*. The forces in the *Exchange with Other Teams* pattern include a give-and-take relationship in regards to resource sharing. Although the *Exchange with Other Teams* involves cooperation, the teams are rivals during the contest, and the course lending is a mock competition to simulate the actual contest.

Another thing that be considered is that the two teams can share the cost of replicating the course. So if one team builds it and rents to the other team, it can save the team that built it money.

Resulting Context

In the 2011 and 2012 contests, we rented our course to a team from another university. In these cases, course rental was one-sided as we provided the practice course and information. However, in 2013, we did not have a practice course, but due to building relationships in previous years, we were able to easily rent a practice course.

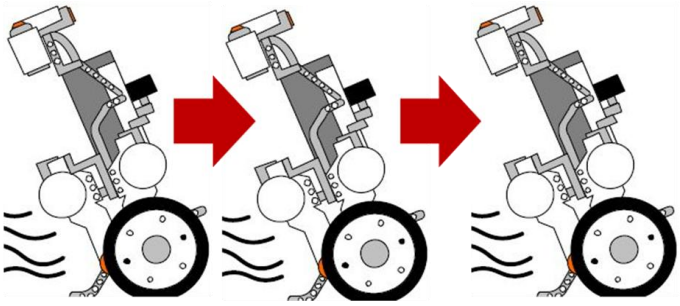
---

* Exchange with other teams:
Teams have close relationships, and exchange or share resources.

Forces:
- It is possible to interact with other teams physically or virtually via the Internet.
- Although each team provides useful information, the teams are rivals in the actual competition.
- A team can provide other teams resources such as past design models or a course replica.

2.3. Drive Straight

The robot must go straight automatically to easily navigate the *Bonus Stage*.



Context

The robot must have a driving system that is independent of the robot's light sensor in order to traverse the *Bonus Stage* without a black line or with a gray line, which may not be detected by the light sensor. Additionally, the driving system must possess speed as well as climb stairs, seesaws, etc.

A team should resolve these problems of *Motor Speed Tuning* and *Course Replication*.

Problem

First, a system to automatically navigate the course regardless if there is a black line to follow must be embedded. However, development of an automatic system is costly. It is very difficult to gain speed when the robot follows a line because the robot shakes to the right and left. **So how can the team make it such that the robot can go straight even with a line and when it climbs?**

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.]
- If *Drive Straight* is applied toward wrong direction, the robot can't come back on black line..
- The team determines accurately the position where the robot should pause because the robot should come back on black line. So applying frequently *Drive Straight* is a high risk.

Solution

The team must develop a system that allows the robot to travel straight. One way is shown in *Motor Speed Tuning*. It is to correct the performance difference between the right and left motors using software. The way to solve this problem is below.

STEP1: Get the initial value of rotations of right and left motor in a place where the robot should travel straight.
STEP2: Compare difference between the current value and the initial value by 4mm sec when the robot moves a little forward.
STEP3: Move one of motor which rotates less than another of that.

The source codes developed actually are shown below.

```
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 = rightWheelCount - 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;
}
```

Straight drive is applicable to various scenes.

- The team wants to embed *Mapping the Course*.

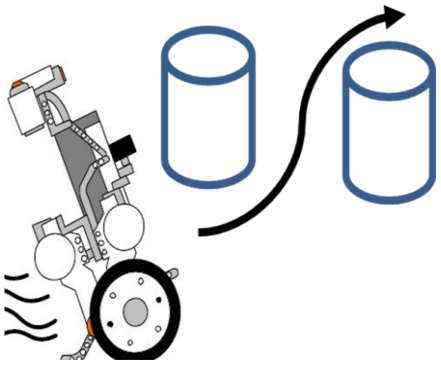
  By combining *Straight Drive*, the robot can move with complete freedom on most courses, but the robot must also be able to turn around on a spot.
- The robot must have speed and a system for climbing.
- As the robot gains speed, it ignores the line and goes straight. For more speed, refer to *Jerking Forward and Stopping.*
- If the robot does not approach the bumps straight-on, one of its wheels may fall off.

Resulting Context

In the actual contest, our robot could traverse the seesaw and stairs. In addition, the robot could cancel the line trace as it went through the gray zone.

2.4. Mapping the Course

The robot is programmed to move automatically through the course without relying on the light sensor.



Context

The team adopted *Replication of Course* or *Course Rental*, and the team makes sure that the environment for practice is well-regulated. Moreover, it is guaranteed that the robot goes straight along smoothly.

Problem

The light sensor does not work in the *Bonus Stage* where the line is missing. Additionally, because ambient light affects the light sensor, the robot may veer off course when using the light sensor. Although the *Canceling Ambient Light* pattern can resolve this problem, it is difficult to embed. **So how can the team make it such that the robot move automatically in a place where the line is missing?**

Forces

- Even if the team makes the robot move automatically, it is very difficult that the robot moves just as expected because there is difference of hardness of surface between the course of the contest and practice.
- Because the robot should enter straightly toward black line when the robot comes back on black line, the team should be careful of applying *Mapping the Course.*
- As the former forces are written, *Mapping the Course* has a high risk. But the robot can take a shortcut across the course having no black line if the robot succeeds *Mapping the Course*. It can reduce time a lot.

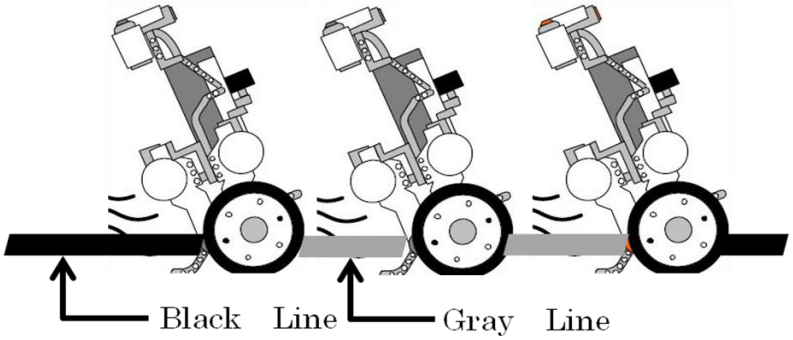
Solution

These problems can be solved by programming the robot to memorize the turning points on the course and to ignore the line. Adopting *Estimating Robot's Location* makes sure accurately the robot's location to use various sensors. For more detailed explanation, the way to estimate the robot's location is that the robot knows itself the value of angle, rotating of motors, and time. When the team confirms that the robot understands a place where it should start Mapping Drive, the team adds and combines the method of *Drive Straight* and rotating to the robot.

However, due to *Motor Speed Tuning* problems, it is difficult for the robot move on "memory", especially if the course is not exactly replicated.

2.5. Ignoring the Gray Zone

The robot can navigate the gray zone by combining *Straight Drive* and *Estimating Robot's Location.*



Context

The robot must drive on a gray line because gray line mixed black line. This reason is that gray zone indicates that there is *Bonus Stage* on the front of gray zone.

Straight Drive should be embedded into the robot. And the robot can determine accurately its location by adopting *Estimating Robot's Location*.

Problem

Detecting the gray line is extremely difficult because the robot's cords can interfere with the light sensor. Thus, it is difficult for the light sensor to detect the value of color, causing the robot to deviate off course. **So how does the robot pass through the gray line?**

Forces

- A light sensor occasionally recognizes the gray line as the black line. However it depends on the situation of the contest. If the robot can recognize the gray line as the black line, the team should not apply following the solution because it only increases a risk.
- If the robot detects gray zone, the robot can make sure its location. But applying this pattern leads to lose this possibility.
- If the team decides the distance of ignoring gray zone, the robot can't come back on black line.

Solution

The robot can recognize the gray zone by combining the patterns of *Straight Drive* and *Estimating Robot's Location*. First, the robot should "memorize" the start of the gray zone using the *Estimating Robot's Location* to measure the distance and angle of the robot. When the robot recognizes it has arrived at the front the gray zone, it can go forward a definite distance by changing to *Straight Drive* mode, allowing the gray zone to be ignored because the gray zone always is straight.

---

* Estimating Robot's Location:
To estimate the robot's location by various sensors.

Conclusion and Future Work

We have extracted a pattern language consisting of 40 patterns from our technical knowledge and contest experience. We applied five of these patterns to the 2012 ET robot contest. Although we were able to recreate a practice environment, the ***Mapping Drive*** pattern did not work very well and the robot moved liked it was on the practice course. This reason may be that the contest surface was harder than the practice surface or that the robot turned more than expected. Thus, ***Mapping Drive*** needs fine tune adjustments when creating a practice course.

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.

Acknowledgement

We thank for shepherd Joseph W. Yoder for very careful review that improve this paper.

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.