Find the Blue Stick

Grégory Mermoud, Marc A. Schaub, and Grégory Théoduloz

School of Computer and Communication Sciences,
Ecole Polytechnique Fédérale de Lausanne (EPFL),
1015 Lausanne, Switzerland

Abstract. We propose a modular, reactive controller for locating a target in a maze while maintaining sufficient battery power. We have adapted our solution of take into account the limitations of the simulation environment and the high level of noise. The behavior of our robot proved to be sensible though being by far less efficient than mapping-based solutions.

1 Introduction

The aim of this project is to design a controller which allows a robot to explore a given world looking for a coloured object. Since the initial battery level is not sufficient to achieve this task in a single run, each robot has to look for chargers which are spread around the world.

In order to test the proposed controller, we use *Webots* [1], a mobile robotics simulation software which provides a rapid prototyping environment for modelling, programming and simulating mobile robots.

We use a 2-wheeled *Khepera* with 8 infrared distance sensors, one odometric sensor per wheel and a color camera.

To solve this problem, we prefer a reactive approach to one consisting of building a map. This alternative would have rised several complex issues specially due to the presence of noise in sensor readings and is therefore out of the scope of a short-time project. Consequently, we use a behavior-based controller inspired by the subsumption architecture presented by Brooks [2].

Section 2 presents the subsumption architecture which fulfills the whole task. In Section 3 we consider synchronization issues arising in some versions of *Webots*. Section 4 details the performance of our controller when run alone and during the competition with the other teams.

2 Subsumption Architecture

The key idea beyond this approach is to break down the desired behavior into a set of simpler basic coordinated behaviors. Each task achieving a behavior is represented as a separate layer. The layers are implemented incrementally: the simplest behaviors reside at the lowest level and the others are added upon one at a time. All the layers work concurrently and asynchronously with the condition that lower in the architecture are not aware of higher levels. Layers may contain several modules working together to complete the desired task. The basic coordination mechanisms in the original model are the inhibition which prevents a signal transmission to the actuators, and the suppression which prevents a signal transmission to a behavioral module and replaces that signal by the suppressing message.

This approach offers many advantages like providing a basis for incremental design or fitting well to team work.

2.1 Braitenberg Modules

Our subsumption architecture is composed of modules which we want to be lightweight and responsive to sensor changes. For that purpose, we use the Braitenberg approach [3] which tightly couples sensors to actuators to provide a simple controller architecture.

In this approach, the value of each actuator is a linear combination of the sensor values. The weights of the linear combinations need to be fine-tuned.

2.2 Odometry

To estimate the current position of the robot, we first calculate the difference on the odometric encoders with respect to the value from the last iteration. We then calculate which distance each wheel of the robot has run since the last time step. We use this to compute the position of the center of gravity of the robot, that is actually the arithmetic average of the distance ran by the two wheels, and the bearing. Finally, we update the position variables of the robot.

These values are approximative because of the wheel slip and the encoderx discretization. We can efficiently limit this error by bounding the robot speed to a low value.

2.3 Our Architecture

The behavior we want our robot to exhibit: it has to (1) explore the world while (2) sustaining a sufficient battery level to get back to the last seen charger (3) until it finds the blue stick, Moreover, the robot must be able to (4) avoid obstacles. We implement each basic behavior as a layer in our architecture:

1. The first layer is dedicated to exploration of the world. We separate the angle of view of the camera into slices for which we approximate the distance to

the closest obstacle by using the amount of white at the bottom of the image. Each slice provides an input to the Braitenberg controller which makes the robot move toward open areas. This is the default behavior of the robot.

2. The *Battery Level Guard* layer focuses on sustaining a battery level which enables the robot to return to last seen charger without exhausting its energy. To achieve that, it relies on the *Position Updating Module* which keeps the current robot position within an internal axis system. The *Battery Charger Detector* stores this position in memory when the robot sees a charger. The *Battery Charger Finding* module makes the robot move either toward the position stored in memory if no charger is seen at the moment or toward the charger it sees using a Braitenberg approach.
3. The *Blue Stick Finder* layer relies on a Braitenberg controller to go toward any blue object. When active, this behavior suppresses all the other lower levels since it is the primary goal of the robot.
4. The purpose of the top layer is to deal with obstacle avoidance. This is a Braitenberg controller which relies on infrared sensors and therefore deals only with very close obstacles. If an obstacle is detected, it suppresses the other behaviors.

Figure 1 depcits the subsumption architecture we use.

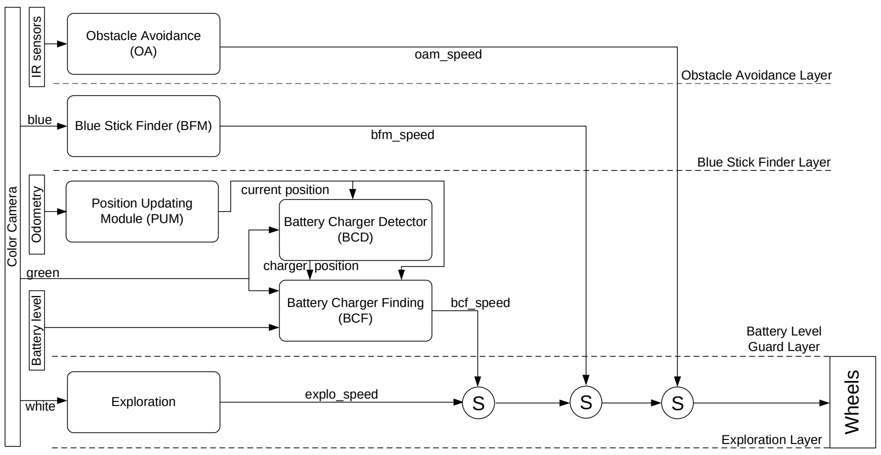

Fig. 1. Our subsumption architecture.

3 Synchronization

Webots provides two modes for the interaction between the physical simulation and the controller: a *synchronous mode* and an *asynchronous mode.* In synchronous mode, Webots stops the simulation while the controller is executed and waits for a notification from the controller to advance the simulation time. In *asynchronous mode*, which, surprisingly, is the recommmended default for robots competitions[1], the simulation runs concurrently with the controller, and the frequency with which the controller is called might thus be significantly lower than the expected one.

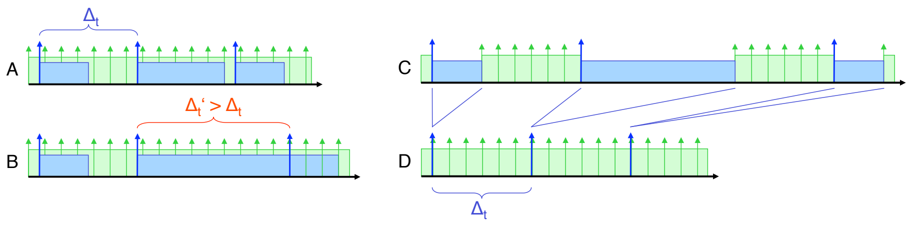

Fig. 2. Comparison of scheduling behaviors between reality and different Webots modes. Executions of the controller are represented in blue and execution of the physical world in green. Arrows indicate when the controller, respectively the physical world update in Webots, are called. (A) represents the real behavior as well as the asynchronous behavior in Webots: controller and the world are runing concurrently. (B) represents a scenario where a controller call exceeds the delay between two controller calls; The time between two controller calls is no longer constant. In (A) and (B), real time and simulation time are considered equal. (C) represents the Webots synchronous mode in the same situation as (B); the simulated time is stopped during the execution of the controller. (D) represents the same execution as (C), but in simulated time instead of real time. Even though one execution of the controller took longer, the behavior in simulation time is the same as in situation (A).

3.1 Issues Arising in Synchronous Mode

Webots does not simulate the embedded processing units of the robots but relies on the local machine on which the simulation is run to directly execute the controller code. The execution time of the controller code is therefore heavily dependent on the architecture of the machine, its performance and current load,

[1] according to Section 4.4.3 of the Webots User Guide [4]

compiler optimizations and programming language choices. These factors are likely to be extremely different on the actual robot. A controller which uses more ressources than available on the actual robot might work fine in Webots while it cannot work in reality. The only solution to this problem would be to simulate the whole hardware of a robot, which is practically untractable. This problem can however be avoided by careful code design and verification of the timing constraints.

3.2 Issues Arising in Asynchronous Mode

Specific function calls take longer to execute in *Webots* than on a real robot and have a variable delay depending on the machine on which the simulation is run. Camera calls for example cause an important delay in the execution of the controller code. In asynchronous mode, this delay causes a jitter of up to 0.8 seconds between calls of the controller. Such an imprecise timing leads to the incapacity for the robot to perform precise odometry. Taking rational decisions about which action to perform next becomes difficult since it is possible that the robot will perform this action for almost a second instead of the 64ms it is expected to. Furthermore, the information read on the sensor is outdated. Jitter variability due to factors like graphics hardware performance result in very different behaviors when runing the exact same setting on different machines. It is therefore necessary to run our competition in synchronous mode in order to maintain a realistic, platform-independent behavior.

3.3 Implication of Synchronization Issues

We noticed that the synchronous mode in version 5.0.3 of *Webots* does not work and behaves similarly to the asynchronous mode. The developpers confirmed that the synchronous mode is broken in version 5.0.3 due to a bug.

Given the asynchronicity, we did not make use of the odometry information at all. In order to avoid that our robots runs for a long time without controller input while the controller is fetching the camera image, we first stop the robot, then fetch the image and process it and finally let the robot move again. In synchronous mode, the stopping would not cause any difference in motion since the robot starts moving again when it has obtained the camera image, whereas in asynchronous mode, this makes the robot significantly slower since it is stopped for up to 0.8 seconds every time the camera image is read.

4 Results

Using our simple subsumption-based algorithm and given the synchronization issues mentioned in the previous section, the results we obtain are fairly satisfactory. In this section, we briefly discuss how well the controller behaves.

Overall, the objectives are met in most cases: obstacles are successfully avoided and when the target or a charger is seen the robot goes toward it.

Nevertheless, due to the aforementioned problems, our competition robot has no way of going back to a previously seen charger. Moreover, due to the lack of time for extensive parameter tuning, our controller cannot handle some special situations well enough. For instance, the robot can get stuck when going toward the target or a charger should it be blocked by some wall. Similarly, it can be stopped touching the charger block whilst not being charging because it is not in front of the charger. Finally, our simple exploration strategy tends to lead the robot into the corners of the map but thanks to the obstacle avoidance, the robot can get out of them afterward.

It is obvious to us that doing significantly better would have required to build some map as done by the competition winner, extensively based on the *ALife* contest winner *Piglet*.

5 Conclusion

We have designed a subsumption-based controller using reasonably simple algorithms for each module. Considering the problems encountered with the simulation environment, out controller uses approaches that have proved to be fairly robust.

Even though our controller is much less efficient than sophisticated, map-based controllers like *Piglet*, it has a sensible behavior. We decided to build a realistic solution from scratch within the three-week time frame. Therefore, we intentionally focus on simple, reactive solutions that can cope with high level of noise. Thanks to our modular design, it is possible to replace our simple reactive solutions by more sophisticated ones, for instance by using a map for the exploration. Finally, a dominant part of this project has been to identify and deal with *Webots* limitations and bugs.

References

1. O. Michel, "Cyberbotics Ltd. Webots$^{\text{TM}}$: Professional mobile robot simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, pp. 39–45, 2004.
2. R. A. Brooks, "Intelligence without representation," No. 47 in Artificial Intelligence, pp. 139–159, 1991.
3. V. Braitenberg, *Vehicles: Experiments in Synthetic Psychology.* The MIT Press, 1986.
4. Cyberbotics, *Webots User Guide, release 5.0.3.* Cyberbotics Ltd., 2005.