

# Auto-Gnome, an autonomous can searcher

Erwin Jansen, Gert Jan Verhoog, Patric de Groot, Michiel Hermesen

December 16, 1999

## Abstract

In this short paper we shall give an outline how we constructed a lego robot that is capable of searching and retrieving white cans in an arena filled with obstacles. We'll point out that a behaviour based approach is the only feasible solution for the challenge that has been presented.

## 1 Introduction

During the course Robotica given at the University of Utrecht we had an assignment to construct a robot, using the LEGO Mindstorms kit [1] and some additional sensors, that is capable of finding some white painted cans and returning them to a home base. For an exact description look at the homepage [2].

We had the use of a standard mindstorms kit with some additional motors and sensors. To sum up:

- several pieces of lego.
- 3 motors.
- 2 rotation sensors.
- 2 touch sensors.
- 2 light sensors.

Using only this and some inventive programming we constructed a robot that was able to fulfill the challenge.

## 2 AutoGnome, from bricks to a bot

In order to be able to achieve some interesting can collecting behaviour we had to consider the general requirements. Our robot should at least be able to do the following things:

- Grab a can, and lift it up. After figuring out where a can is, it must be picked up in order to bring it home.
- Sense if it crossed the barrier that denotes his home. Our robot needs to find a way to come home. We can use a light-sensor to sense wheter we're home or not.
- Sense if it can see a can. One light sensor could be used to recognize a can from a certain distance.
- Feel if it touches a wall, or an obstacle. Using "feelers" we should be able to sense wheter or not we bumped into an obstacle.
- Sense it's rotation, so it is able to derive it's angle. Using this we could calculate the position of the robot. This position can be used for smart retrieval technique's.

### 2.1 Combining sensors

The RCX-brick that we use to control the motors and sensors has some severe limitations. First of all we're only able to use 3 input ports, and we can use only 3 output ports. Since we have only 3 motors, the amount of output ports is not a real problem. The input ports however lead us to some problems. We want to use more sensory information than the amount of input ports we have.

We can work around this by combining different sensors on one port. We can combine touch sensors with light sensors for example, and even better we can combine a touch, rotation and light sensor all on one port. We have found no literature on this, so it seems that we're the first to make use of such a combination of sensors. We only made us of the combination of light and rotation.

The RCX reads it's port out using a D/A converter. The RCX reads out value's in the range from 0 - 1024. We can tell the position of a rotation sensor by looking at the value's it generates. For example the value 1000 would denote rotation position 0. If we combine the light sensor with the

Figure 1: A picture of our Auto-Gnome

rotation sensor we get a continuous read out within a given boundary. For example:

Range	State
680 - 1024	0
430 - 680	1
0 - 320	2

Due to an overlapping between state 2 and state 3 we can only sense 3 rotation states if we combine a rotation sensor with a light sensor. We also lose out a lot of accuracy in our light sensor, this is not a big problem since we only want to use the light sensor to check whether or not we crossed a black line. This will not pose a serious problem for our robot.

## 2.2 Building the robot

Using all the bricks we've constructed a robot that looks like picture 1. Our robot, called auto-gnome, is very robust. We can pick it up at any brick. It will not fall apart, it can crash into walls without losing parts of its body.

Our auto-gnome consists of several components:

- A gripper, that is capable of picking-up cans.
- A bumper, capable of sensing left or right bumps.
- A chassis, housing the RCX and the wheels used for movement.

Everything has been connected to the chassis. Auto-gnome is easy to decompose and constructed in a modular fashion.

## 3 Adding some intelligence

Our robot needs the ability to roam around the arena in search for cans. Therefore we need some intelligence that contributes to finding the cans in an efficient way. We can express the behaviour of AutoGnome in the language called NQC [3]. This language enables us to control the robot, to define tasks, read sensors, act upon them and do some simple arithmetic.

AutoGnome can construct its view on the world by making use of its sensors. Unfortunately there are some severe problems with the sensors and the way we read them and the brains of our robot.

Figure 2: A finite statemachine for rotation

1. The light sensor is very imprecise. Our robot cannot “see” a can that is further away then 5 cm.
2. It’s brains are very slow. NQC is not fast enough to read out our sensory data of the rotation sensors.
3. Rotation is very imprecise. If our robot tries to make an angle of 90 degrees, it is sometimes 10 degrees, the other time 120 degrees. The imprecision is so large we cannot
4. We only have 32 variables, meaning that AutoGnome cannot really built a history of what happened.

We have created a work around for the slowness of the reading of the rotation sensors. Using a finite state machine as in figure 2 it is possible to calculate which position the wheels are in. Using the direction of the motor and the new state of our rotation sensor we can calculate the new position.

## 4 No way out, behaviour based robotics

So we faced the dire problems of a robot that is extremely stupid, has got a slow brain and is very imprecise with its actions. Luckily we’re not the first who encountered this problem. As a matter of fact some even consider these problems rather like a blessing. In [4] the foundation is layed for a new

view on robotics. Instead of building complex models of the world and using mathematical analyses of the surroundings we should just couple input to output. These ideas stem from biology and give a new refreshing look at problems with robotics.

We adopted this view and have put a subsumption architecture in our autognome. We've implemented the following behaviours.

- A zigzagging behaviour to walk around the arena.
- A bumper behaviour, that pulls back the robot when it bumps into a wall.
- A light optimising behaviour, autognome will prefer to go to lighter places. Cans look lighter so we should find them using this technique.

## 5 Conclusion

Although we faced some major difficulties, we managed to create a robust robot with some

## References

- [1] Lego Mindstorms, <http://www.legomindstorms.com/>
- [2] Challenge Robotics, <http://www.cs.uu.nl/markov/lego/challenge/description.htm>
- [3] Not Quite C a language to control the RCX-brick, <http://www.enteract.com/dbaum/nqc/index.html>
- [4] Brooks, R. A. (1986), A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation
- [5] Constructing autognome