IA158 Real Time Systems, Assignment 3 Documentation

Cupak Miroslav,
Niznan Juraj,
Nemecek David,
Streck Adam

April 30, 2012

1 Introduction

We have decided to create an interpret - robot capable of reading graphical instructions and later executing them. As a hardware, we have picked four-wheel car with two light sensor, one that reads movement line and ensures that robot stays on its path and second one for instructions. Using our instruction set, one can make the car move in different directions or make sounds. Each instruction corresponds to a four bit number (we call such number byte), that is encoded using four colored stripes, each representing on or off bit.

2 Construction

We used a rear-wheel-driven car with steering in the front. As an engine, we use two simultaneously working motors, while steering is assured by a single motor with gear box.

There are two build-on light sensors. At first we put them on the back (one in the middle, the other one on the right side), but soon we have found out that this constellation of sensors does not allow behaviour that we wanted it to express - one sensor read slightly darker values than the other one and also this placement problematically coped with our method of steering - track sensor detected that the car was off-track later than we would want.

Currently we have light sensors on the sides of the car, having the same distance and direction from the core of the car, giving us requested precision of the reading.

3 Movement calibration

To ensure continuous reading of the instructions without possible errors caused by incorrect starting direction, we used a trace - two lines, one with grey and one with black colour. While on the trace he continues forward. When the white color is reached, he returns on the trace depending on the last color he saw before white - if it was gray, he turns right, if black, he turns left.

Instructions reading is ended by a short timeout within which no new bit is read. To ensure that movement calibration does not cause this timeout to expire, line following thread pauses the data reading thread for the time in which the car stops, steers and starts moving again.

Robot does not start reading instructions until it finds an initial line. If he is not able to find the trace or if he loses it during the computation, an exception is thrown and robot stops moving.

4 Reading data

Although in the beginning it seemed as a simple task, reading the data from the data-line turned out to be quite tricky. One would expect that when the light sensor moves from a white region onto a black region, it would result in a few readings of WHITE and then BLACK values. Instead it results in WHITE - GREY - DARK GREY - BLACK values.

4.1 Data encoding

Because the light sensor is basically an analog input device, the outgoing values have to be treated that way - as an analog signal. And as the device is not that sensitive, we decided (for the sake of reliability) to reduce the number of recognized colors to three: WHITE, GREY, BLACK.

This gives us the option to encode binary strings into a 3-level signal. WHITE being the *no-data*, *synchronization* level and GREY and BLACK being the *data* levels, 0 and 1 respectively. Every signal interval between two WHITE regions is considered to be one bit and based on the maximal value (darkness of the color) we recognize its value.

This is a ”peak value encoding” with automatic synchronization. The encoding therefore delivers 1 bit of information for every two signal elements (changes).

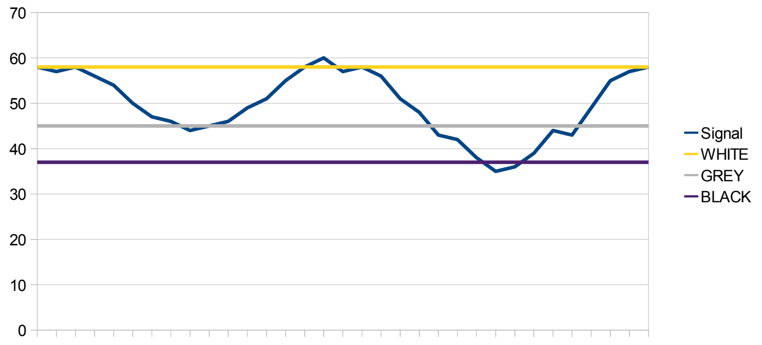

Figure 1: Light sensor readings

4.2 Various details

After testing a few layouts and configurations we found that the best results are achieved with *data* stripes 2 cm thick and *synchronization* spaces about 3 cm thick. Thin stripes tend to be overlooked by the robot, extra thick *data* stripes could be erroneously interpreted as multiple bits. Larger areas of white do not matter much, although there is always the possibility of picking up ”noise” bits (e.g. dirt marks).

The cooperation of the line-following (main) and data-reading (slave) threads proved to be essential. When the robot leaves the guiding line, it needs to stop, adjust its direction and then continue again. This often results in erroneously read or re-read bits, because the data-line is not moving strictly in one direction. The master thread therefore signals the slave thread when it is safe to read data values (i.e. the robot is moving forward).

4.3 Error handling

To make the encoding as simple as possible, we implemented a timeout-based reading termination to avoid control sequences in the data flow. Every successfully read bit resets the timer, otherwise after a specified timeout the reading process is terminated. This signals the main line-following thread that it should terminate as well.

In the end the resulting array of bits is converted into an array of 2-bit *bytes*. This implies an even number of bits, which is the only error-checking performed by the decoder. Bit values are not checked in any way as it would require more bits to be read.

5 Instructions execution

The final phase of the robot's program aims to interpret the newly decoded binary data. This works as expected: the instruction interpreter sets its Program Counter (PC) to the first data *byte* (i.e. 2-bit value) which determines what should be executed and then repeats the whole process until it moves out of the array of instructions. Every instruction moves the PC one *byte* further, instructions have access to the PC and can (theoretically) alter it to perform loops.

5.1 Instruction set

Although it would be possible to prepare a "Turing-complete" set of instructions, our goal was to showcase that the robot does what it is told using only a few bytes. The same goes for the number of instructions - we used 2-*byte* instructions (first its function, then its parameter), but it is possible to have longer / shorter / variable length instructions.

Byte 1	**Byte 2**	**Description**
0	0	Go forward
	1	Go forward left
	2	Go forward right
	3	Repeat LAI 2x
1	0	Go back
	1	Go back left
	2	Go back right
	3	Repeat LAI 5x
2	X	Play tone (X+1) * 100 Hz
3	X	Delay time (X+1) * 1000 ms

Every standard instruction is executed for 1000 ms. Sound playing instructions are executed in parallel and therefore take 0 ms to execute.

The *Repeat LAI Xx* performs the *Last Action Instruction X*-times. This servers as a simple way to perform movement instructions multiple times using fewer data *bytes*.

Because every possible 2-*byte* value has a meaning assigned, it is not possible for the user program to cause an error along the execution based on a wrong *byte* value. However if the number of *bytes* is not even, error occurs on the very last instruction.

6 Folder content

- Documentation - source for this documentation.
- Foto - brief fotodocumentation.
- Instructions - complete set of instructions for printing.
- Presentation - source for class presentation.
- Program - source code.