School of Electrical and Computer Engineering
Embedded Microprocessors Systems (HPY411)

Autonomous car

Final report

Apostolopoulos Theodoros
Dialektakis George

Content Table

Introduction

This semester, for the needs of the “embedded systems” course we have decided to construct an autonomous car, using the Lego Mindstorms technology, that is able to move freely through space, successfully avoiding any occurring obstacles and that is able to automatically park in any custom parking slot. Our goal was to create a versatile program that will allow our car to execute its goals successfully in any custom environment and in the most non-trivial way possible. The project was completed in three stages and during each one of them we added features to our robot, we extended its abilities and we corrected any constructional faults in its body.

Materials

First and foremost, we had to construct our vehicle. We had been given an already made vehicle from the projects available in the lab. However, we decided that we should create one on our own with the specifications that we wanted, as we already had many parts from an older version of NXT. First, we created the main body using examples and instructions we found on the official site of NXT and on YouTube. After many tests with our robot we made appropriate corrections to the main body, as our main problem was the lack of accuracy on the turning of the front wheels. As a result we succeeded in minimizing the error in the turning of the wheels by choosing our final design that utilized a certain pair of small gears.

Material list

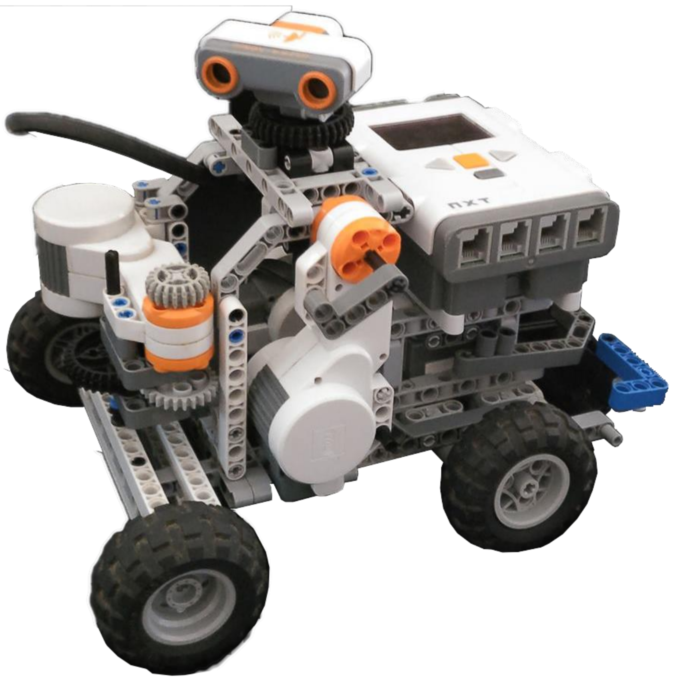
Our robot consists of the following parts:

Sensors:

- Ultrasonic sensor (Port A)

Motors:

- Motor A: movement(rear wheels)
- Motor B: ultrasonic sensor rotation
- Motor C: steering (front wheels)



Moreover, we have the following measured data:

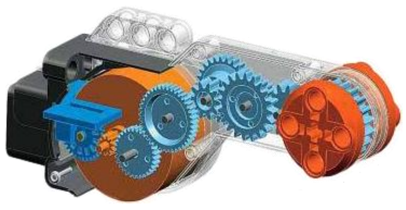
- Wheel radius R=2.7cm
- Car length 21cm
- Car width 17 cm
- Motor B gear ratio 3.2
- Motor C gear ratio 1.5

Technology

As it was mentioned before the technology we are using for this project is the NXT Lego Mindstorms. Although this technology has a plethora of components, such as motors and different kinds of sensors, the main component is a brick-shaped computer called the NXT brick. The brick is the main programming unit as it contains a **32-bit ARM7TDMI-core Atmel AT91SAM7S256** microcontroller with 256 KB of FLASH memory and 64 KB of RAM and an **8-bit Atmel AVR ATmega48** microcontroller with Bluetooth support. In the ports of this main unit there are connected all the peripherals that the robot uses. The brick provides us with four input ports, where sensors that gather data from the environment of the robot are connected and three output ports where we connect the NXT motors that execute the orders given by the main unit.

As far as the peripherals are concerned, in this project we are using one ultrasonic sensor and three motors to execute the processes of obstacle avoidance and automatic parking. To begin with, the ultrasonic sensor that we placed at the top of our vehicle, can measure the distance from the sensor to something that is facing and detect movement in a straight line within range from 0 to 253cm with 3cm accuracy. By placing it into a rotating gear and connecting it to a motor we were able to scan the whole area around the robot without having to disturb its course. Moreover, it is important to mention that the main unit receives data from the ultrasonic sensor only through hard polling. This process may be demanding for the processor; however he is strong enough to withstand it as long as the memory doesn't fill.

What is more, the NXT motors we are using are three identical servo motors that have build in reduction-gear assemblies with internal optical **rotary encoders** that sense their rotations within one degree of accuracy. By having rotary encoders we are able to define the operation of the motor by the distance they have travelled instead of using their operating time as a mean of measuring, thus optimizing the performance of our system.



Programming environment

As far as the programming part of our project is concerned, we are using the ROBOTC platform in order to implement the code of the NXT brick instead of using the original platform that is provided by Lego on its website. More specifically, ROBOTC is a C-based programming language which provides the user with special instruction sets that correspond to the original brick shaped instructions that control the peripherals. Additionally, this platform provides the user with a number of sample projects that implement plenty of operations of the robot which guided us through the first steps of code-making. However, the most important feature of ROBOTC was the debugger window. This window, by using the function *writeDebugStreamLine()*, can depict the values that are received by the sensor into the screen of our computer. As a result, we were able to perform tests and trace any mistakes to our design by comparing the behavior of the robot instead of wasting time in imagining our mistakes by trial and error.

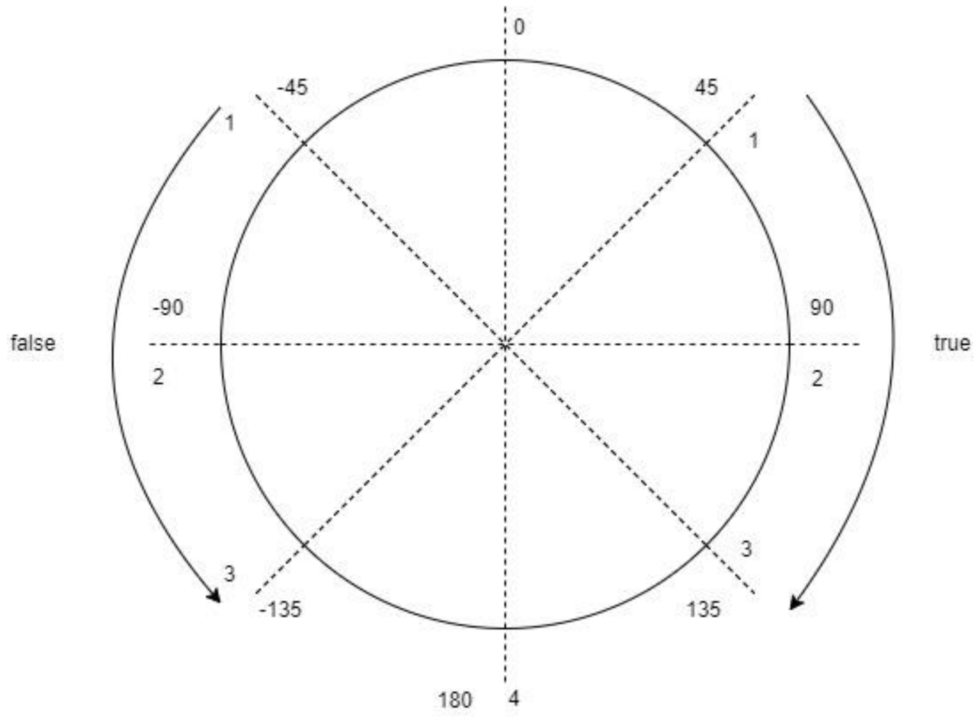
Program

As far as the programming part of our project is concerned, we benefited ourselves from the characteristics of Structured Programming and we decided to implement our basic operations in two separate parts.

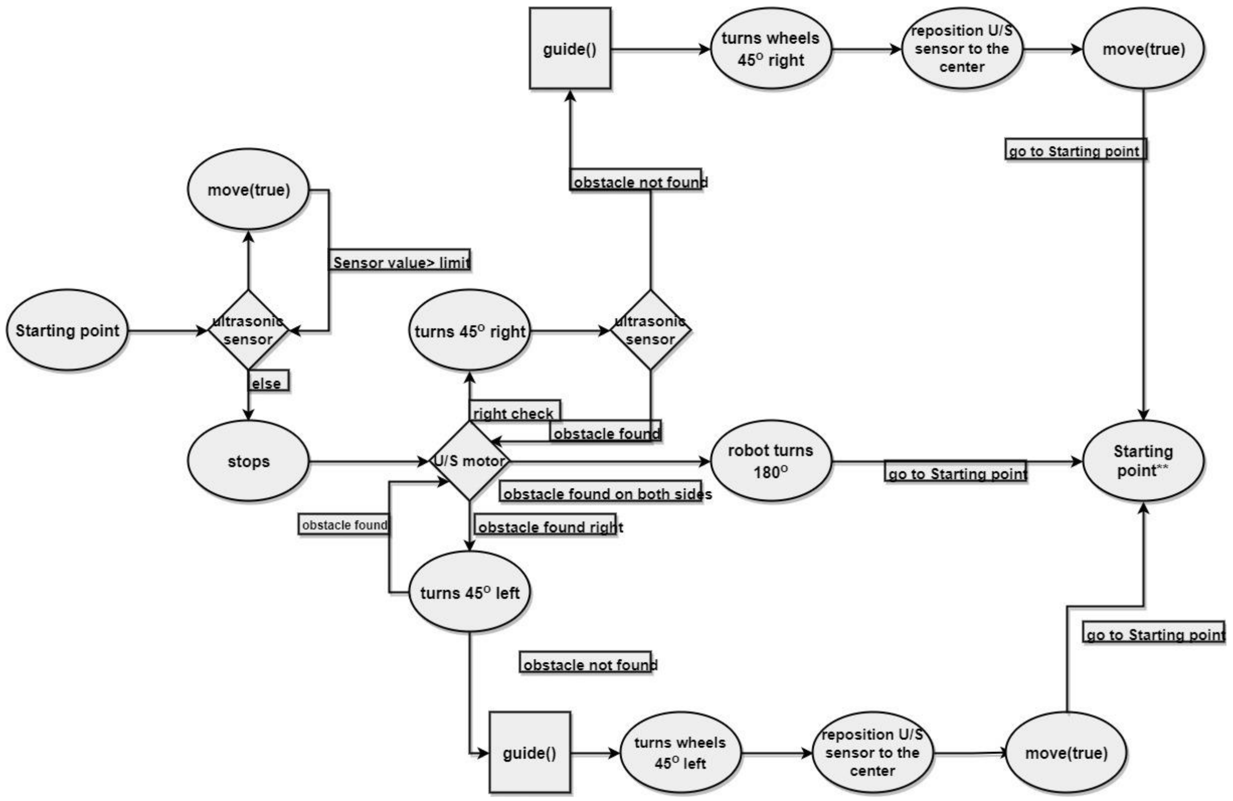
Obstacle avoidance

For the first part we set the foundations of our project in five major functions ("**move**", "**rotateSonar**", "**rotatewheels**", "**check**" and "**guide**"). The first three functions control the movement, the sensor-rotation and the wheels-rotation motors and they implement basic operations using classic NXT programming logic that has been tailored to our needs.

The most important functions were "**check**" and "**guide**" due to the fact that they are the ones responsible for the decision-making of the robot. After many changes in the way we interpreted our problem of avoiding the obstacles we decided that the sonar sensor would perform a check in a 90 degrees radius in front of it after its movement was interrupted by the detection of an object. If the sensor doesn't detect something within one meter in either side it moves towards the direction of the first available free space. Otherwise, if the whole area in front of it is blocked, it turns around. The “check” function is responsible to return the result of the sonar sensor when it reaches 45 and -45 degrees angle and according to its result the “guide” function guides the robot towards the correct direction according to a mapping system we created and that is listed below.



Guidance model



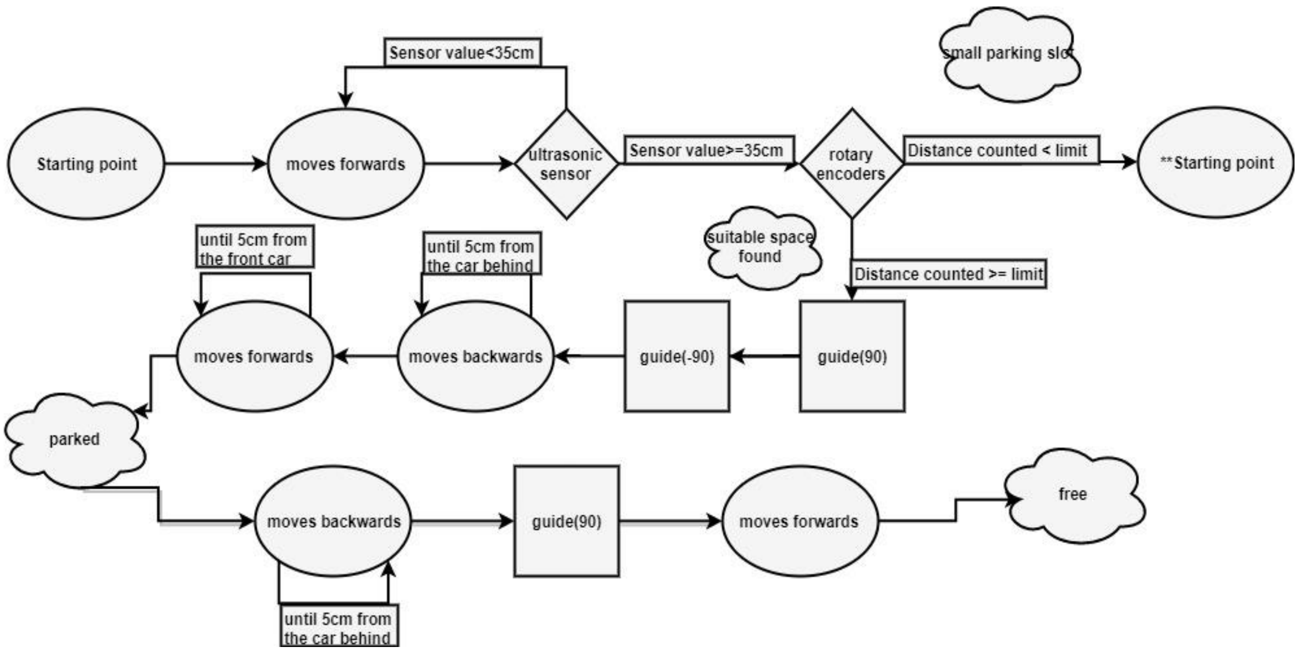
Obstacle Avoidance Diagram

Automatic parking

For the second part of our project we added to the code we previously described the **find_parking()** function**,** which searches for a suitable parking spot that is at least 1.5 times our car length and the **park()** function which gives orders to the motors in order for the car to park efficiently.

More specifically, in order for our car to park we measure the distance of our car from a parallel surface by turning the ultrasonic either left or right. Once a measurement that is larger than 35cm has been received, the robot considers this spot as the possible beginning of a parking slot. Immediately, the rotary encoders of the rear wheels motor are set to zero and they start counting degrees until the ultrasonic sensor stops measuring a distance larger than 35cm, which means that the candidate parking slot has ended. Afterwards, we convert the degrees of the encoders to travelled distance and we compare this value to the length of the car. If the space is less than 1.5 times the length of our car we reject this parking space, otherwise we initiate the parking procedure.

As far as the parking procedure is concerned, we take advantage of the mapping system and the guide() function we have previously created and we command the car to move backwards in a diagonal 90^o course, while at the same time we turn the ultrasonic sensor towards the direction we are moving to avoid collision. Thereinafter, we move the car forwards until the distance from the front car becomes approximately 5cm. Finally, the vehicle leaves the parking spot by following in reverse the previously mentioned steps.



Autonomous Parking Diagram

Problems and solutions

During our project we faced many difficulties mostly due to the fact that we had to get accommodated to a new environment which has both physical characteristics and new programming rules. At first we had to discover what the platform provided us with (Firmware, Debugger, Sample Projects) and afterwards we had to understand the way this C-like programming language controlled our robot. After we gained some experience, our biggest problem was the administration of time. When we started dividing our code into separate functions many of them where fully functional when we executed them separately, however when they were followed by other functions everything was breaking down. The cause of these malfunctions was the fact that at the end of some functions we didn't restrict the running time of the motors. As a result, when we executed a function alone the motor wouldn't continue working because the program ended thus adding difficulty to the debugging process.

What is more, as far as hardware is concerned, we had to overcome difficulties that were caused by the NXT technology itself. More specifically, although we changed the front wheels turning system many times, we had accuracy problems to the last minute. Additionally, the ultrasonic sensor that was attached on top of the robotbecause of its design was accurate when detecting objects from a long distance, however when it came to really close distance it misfired sometimes. Because of this unreliability, when the car was searching for a parking space and needed a continuous stream of accurate data, we had to increase its distance from the wall in order to secure our measurements and prevent the initiation of a function that would occupy our processor.

Finally, before starting using rechargeable batteries, this used to be a huge problem as the NXT has high power consumption and typical batteries die after a few days of tests thus costing us a lot of money and causing unexpected misses of the sensor when battery was almost over.

Bibliography

- Basic knowledge and information about ROBOTC
  http://help.robotc.net/WebHelpMindstorms/index.htm
- Information concerning problems we faced along the way
  http://www.robotc.net/forums/viewforum.php?f=1&sid=6decc57fb332bd1c7039d8a0840df4bb
- General information
  https://www.khanacademy.org/science/electrical-engineering/lego-robotics
- Download of original firmware and general information about NXT
  https://www.lego.com/en-us/mindstorms/downloads
- Calculating gears ratio
  http://gears.sariel.pl/