TEAM 8

PROJECT 1
SENSING AND MOVEMENT

ROBOT CODE DOCUMENTATION.

Josh Guice
Neelam Chauhan
Ramakrishna Pantangi
Vitaliy Marin

Date: Feb 17, 2003.

Objective

To explain the various data structures and algorithm used in our software code.

Description

Our basic strategy for this project was to make robot travel straight and whenever it goes beyond *f* (=6) feet and it has detected the black tape it will make approximately 90^o turn to its left side. The robot has to repeat this for *n* (=12,) times, since we had to reach each square 3 times and we had 4 such squares at 6 feet from each other in square (Figure 1). We have used for-loop to accomplish this, i.e. the robot goes straight for *f* feet and when it detects black tape it would make left turn, goes back for certain distance and then aligns itself to the black tape at either front side or back side of the square; repeating these sequences of action for *n* times we accomplish the objective of the project. We have functions namely, *go_straight()* and *leftt_turn(), go_back() and align()* which performs the above mentioned actions.

- ✓ *go_straight()* – The robot is made to go straight using *motor (int port, int power),* an in-built function which makes the motor connected to *motor ports* to run at *power*. This function also makes sure that the robot stops when it has reached either *f* feet or after going little but forward after detecting the black tape to make approximately 90^o turn.

  The algortithm:

  This module basically keeps track of the encoder counts of both left and right break-beam sensors and depending on the difference in those encoder counts it powers the two motors. Suppose, the left wheel is moving faster than right wheel, and then we have to slow it down and make the right wheel faster to make both wheels go at approximately the same speed to eventually keep the robot straight. So, left wheel is to be given less power and right wheel more. The same strategy is followed when right wheel moves faster than left wheel. This is what done in the two lines of code…

*motor (L_MOTOR, max(0,(100 - 0 * (max(0,L_Enc - R_Enc)))));*
*motor (R_MOTOR, max(0,(100 - 50 * (max(0,R_Enc - L_Enc)))));*

We will also change the distance to be traveled by the robot, depending on whether it has aligned itself to the front side or back side tape of the square.

✓ *left_turn()* – This function is inspired by the *dmiller-super-demo.ic* file from the Interactive C 4 reference. This code turns the robot by approximately 90^{o} based on the encoder counts, while moving right motor in forward direction and the left one in backward direction. The same methodology is used for any kind of turning in our program with required amount of power to both motors.

Pictorial representation of the program execution:

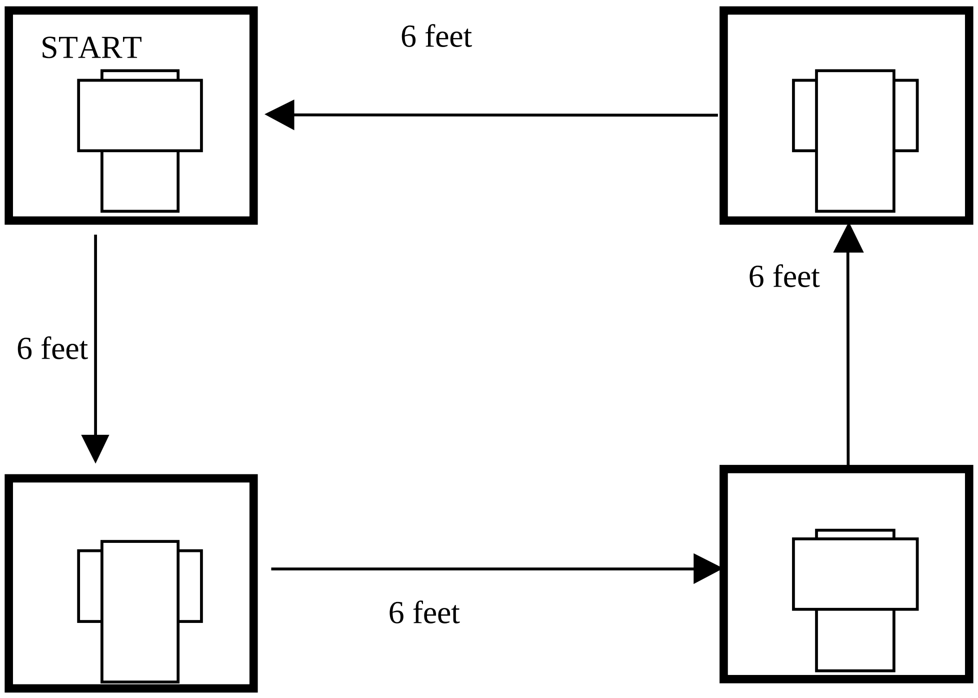

Figure : 1

Once the robot has entered the black square and before it starts another straight journey, we align the front wheels of robot to the straight black tape using user-defined *align( )* function.

- ✓ *align()* – If only the left sensor has detected the black tape then we move the robot forward from right side (i.e. turning robot left), similarly if only the right sensor has detected the black tape then we move the robot forward from left side (i.e. turning robot left). And if both sensors have not detected black tape then just move robot forward slowly; lastly, if both detected the black tape then we say that the robot has been aligned properly and it can start its journey again.

The above-mentioned are the major functions used to control the movement of robot. We have other various small but crucial functions like *mild_forward( ), mild_left_turn( ), mild_right_turn( ), stop_wheels( ), stop_wheels_for_align( ), go_back( ), special_align( ) and max( ).* The names mentioned suggest the function they perform using other in-built functions. The other important module of the program is *monitor_sensors().*

- ✓ *monitor_sensors()* – This module will be running almost through out program as a separate process, but started and killed some times during the program execution. This module will be basically monitoring all the sensors and encoders used in the design to get their latest readings to be used in the program.

In this project we have not used any high-level data structures like arrays or structures etc., but just variables and global constants. In particular global constants, which defines various important values and/or readings to check certain important conditions. For example,

R_MOTOR - a global constant indicating the port number where the right motor is connected
L_MOTOR - a global constant indicating the port number where the left motor is connected

SENSOR_VALUE - another global constant that defines the value of sensor reading when the robot detect the black tape.

R_ENC – indicates the port number to which Right encoder is connected
L_ENC – indicates the port number to which Left encoder is connected

R_SENSOR – indicates the port number to which Right sensor is connected
L_SENSOR – indicates the port number to which Left sensor is connected

In the same way, some more global constants were used to indicate the enocoder count used for turning or for going staright for 6-7 feet, powers to the motors etc., which are well documented in the source code.

Final word

We have made the code as general as possible so as to accomplish similar task for different distances and number of cycles to be completed with little or no modifications.