Paritosh Chokandorkar
Amandeep Gill
Joshua Shuller
CS 4970, Spring 2003
Project 1 - Robot Code Documentation

Robot Code Documentation – Team 2

Code Overview:

The robot used no data structures. There were two simple algorithms, or functions: `goStraight()` and `align2()`. There was one helper function, `shouldStop()`, and the main function.

- The `goStraight()` function made the robot go straight by checking the encoders.
- The `align2()` function used reflectivity sensors to align the robot with a strip of black tape.
- The `shouldStop()` helper function returned whether the stop button had been pressed or not.
- The `main()` function is the main, high-level procedure for achieving the project 1 task.

Code Detail:

The code evolved from a simple procedural outline for testing the basic tasks of the robot (i.e. turning, alignment, and going straight) to a full fledged procedure for handling the goal functionality of project 1. So the code used a bottom-up design approach, which is rarely successful for larger projects. Even though this is just the first project and this project is supposedly simple, the time needed to connect the robot to the download cable, download the code, reset the robot, and move the robot back into position was really the biggest problem with debugging the hardware and software both.

At any rate, the code is detailed as follows:

- The `shouldStop()` helper function returned whether the stop button had been pressed or not. This function used a global variable, `stopPressed`, to record if the stop button had been pressed or not. The function first checked if the stop button is currently being pressed with the `stop_button()` function. If the stop button was currently being pressed, it would then set the `stopPressed` global variable to `1`, or `true`. The function then read and returned the value contained by the global `stopPressed` variable. This function was used in both the `main()` function and in the `goStraight()` function. It was not used in the `align2()` function because the `align2()` function was added to the project Friday morning, the same day of the demonstration. For this reason, when the robot is going straight, it can be stopped by pressing the stop button at any time except while it is trying to align itself with black tape.
- The `align2()` function used reflectivity sensors to align the robot with a strip of black tape. The `align2()` function can be described as two separate behaviors. Each sensor would have its own behavior in which it tells the wheel its same side

of the robot to go forward until it has reached a black line. When it reaches a black line then that side of the robot goes backwards rather than forwards. These two behaviors only stop when both the left and right reflectivity sensors switch from light to dark on the same iteration of the loop. Furthermore, for each time a reflectivity sensor remains in either light or dark, it multiplies the number of times it has remained in either light or dark by it's motor power to increase the power to the motors to help it switch to the opposite shading again. The implementation was neither graceful nor efficient, but it worked.

- The `goStraight()` function made the robot go straight by checking the encoders. The function used the `enable_encoder()`, `read_encoder()`, and `disable_encoder()` functions to count how many times the state of the encoder sensors changed. This function was programmed along with the first prototype robot. To get a better view of how this function transcends exact robot designs, it was used with a second hardware design, and then with a third and final hardware design with no changes at all from the first robot design. In specific, this function took care of slowly accelerating the wheels of the robot to avoid sudden slippage from jerk of acceleration. This function also counted how many clicks each encoder counted from both the left and right wheels. To compensate for lost clicks, it would give the wheel with the lost ticks the number of lost ticks times the `correctionAccell` parameter more power to that motor. To compensate for extra ticks, it would give the wheel with the extra ticks the number of extra ticks times the `correctionAccell` parameter less power to that motor. In short, it one wheel was falling behind, it would receive more power; if a wheel was going to fast, it would receive less power.
- The `main()` function is the main, high-level procedure for achieving the project 1 task. The `main()` function counted the number of times that the robot had to align itself and go straight, and also handled all of the different cases for hitting the black tape square from different angles. The different cases handled were:
    1. Head on contact. Head on contact happens when both the left and right sensors notice the black tape at the same time. In this case, the robot just goes forward for a period of time and then turns 90 degrees left, and starts the align process upon the next iteration of the main loop.
    2. Right sensor reads first. In this case, the right reflectivity sensor is the first to notice the presence of the black tape. Here, it can be concluded that the robot was traveling left of straight. In this case it would be easiest to simply back up, rotate a little bit right, and go straight again to reduce coding. It would execute this procedure repeatedly until it encountered one of the other 3 cases listed here.
    3. Left sensor reads right side of square. In this case, it just did the same thing as in case 1, except that after the robot turns 90 degrees to the left, it goes forward for a certain time period to avoid detecting the black tape on the next alignment.
    4. Neither sensor reads black tape, encoder count down ends. In this case, it simply turns 90 degrees left and aligns itself twice – once with the outer black tape line, and then again with the inner black tape line.

Our software approach was straightforward and the design was bottom-up. In hindsight, it may be wise to design a robot first by emulating or designing its functionality it in software, and then in constructing its mechanical body. This would fit in well with the top-down design approach.