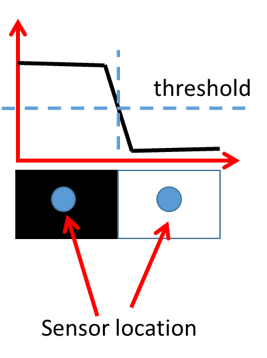
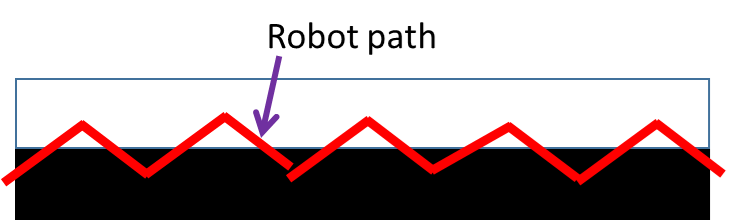
1. Q1. Line following Robot I (binary state solution).

The idea of the line following programs is to sense whether the light sensor either sees black or white (i.e., dark or light) or adjusts the motion accordingly (Figure 1). The robot should see both the states (light and dark) to maintain the forward movement along the track edge. Otherwise, it would either be venturing into a dark or light area.

*Figure 2: Robot traveling while changing the direction.*

*Figure 1: Sensor value change*

As the programmer, you need to decide if you wish to follow the left or the right edge of the line and write the code accordingly. The robot will oscillate back and forth over the line continuously checking for either black or white values (Figure 2); (for the reasons mentioned above) even if the line is straight, the robot continues to turn left and right searching for the track. Depending on the left or edge you are following, the robot will either go to the right or the left.

page1image63879616

The key to a good program is to control the motion appropriately. You can use setMotorSync() and control the turning with nTurnRatio and speed with nSignedPower parameters. The more the robot oscillates, the slower it will perform. The main objective is to follow the line as smoothly and as fast as possible. Develop a RobotC program to read the reflected light intensity from color sensor using getColorReflected() and control the motion to follow a line.

* 1. a)  We will be using the reflected light intensity of the color sensor to follow the line. You will be tracking a dark line on a white surface. To measure the reflected values sensed by the color sensor, first place the robot in the starting position D (light surface color) in the Color Sensor Table under the Utility Tables in the Robot Virtual Worlds (Figure 3). Record this value. Similarly, with the robot on the C starting position (dark color) and record the value. Calculate the threshold value by taking the mean of these two values.
  2. b)  Develop the RobotC program. Test your program using the Line Tracking Challenge (Figure 4) under the Program Flow. Test your program with different forward motion speeds and comment on the performance of the algorithm.
  3. c)  (**CSC 475 Students**) There is another function getColorName() that gives one of the seven colors as a reading of the Color Sensor. Can you give a pseudocode of the program using that function? How effective would that one be in comparison to your present code?

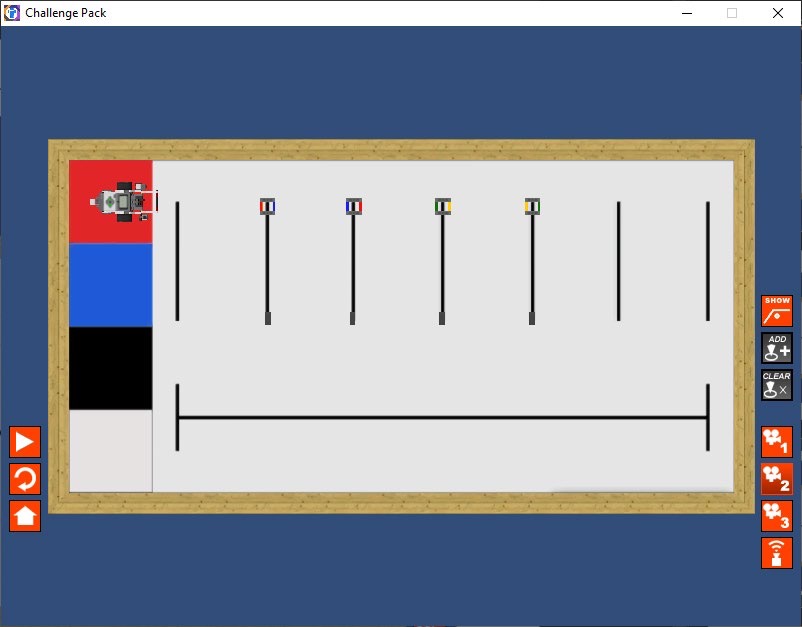
**Deliverables:** A word document containing the discussion, and ROBOTC file for the code. Discussion should cover how the value of TurnRatio and Power affected the Line Following. Pseudocode for the part c and Discussion.

*Figure 3: Color Sensor Table*

Q2. Line following Robot II (proportional solution).

In the binary state example, the program only had two conditions to deal with: a sensed value either greater or less than the threshold. The problem with the binary approach is that the robot

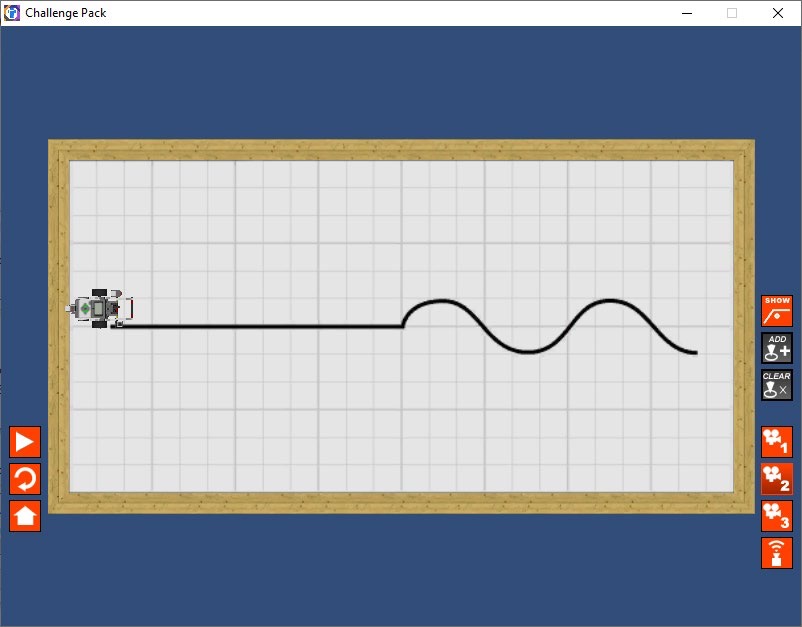
2



will tend to overcompensate for changes in the detected value with hard turns (only two turn values, +/-, used). For instance, imagine a car traveling down the road, and it starts to go off the roadway. Turning the car’s steering wheel drastically to the left will bring the vehicle back onto the road (like we did in the first question with just one nTurnRatio value) but could quite possibly cause the car to lose control. Instead, the driver will gradually turn the vehicle back toward the road, and while maintaining the control of the vehicle, the correction is proportional to the error.

*Figure 4: Line Tracking Challenge*

If you want to follow lines, especially of curved lines, even smoother, you need a proportional controller. A proportional controller calculates the amount of correction that is required to get the robot back on the track that it is following. Instead of using a set value to correct the direction of the robot, we calculate the direction change based on the value read by the color sensor. If the error value is small, the robot corrects very slightly, whereas a more substantial value results in a stronger correction.



Now you can relate this example to the proportional control that you have been using so far. We can extend the same concept for the line following scenario. Think about a value read from the color sensor; it will be between minValue (Line) and maxValue (Background surface). If the value is close to minValue, we are going to want to make an aggressive change of direction compared to a value around closer to the threshold, where we would only need a slight correction in direction. Define the error to be

𝑒𝑒𝑒𝑒𝑒𝑒𝑜𝑜𝑒𝑒 (𝑒𝑒) = 𝑐𝑐𝑐𝑐𝑒𝑒𝑒𝑒𝑒𝑒𝑐𝑐𝑐𝑐𝑐𝑐𝑒𝑒𝑐𝑐𝑐𝑐𝑜𝑜𝑒𝑒𝑐𝑐𝑐𝑐𝑐𝑐𝑐𝑐𝑒𝑒 – 𝑐𝑐h𝑒𝑒𝑒𝑒𝑐𝑐h𝑜𝑜𝑐𝑐𝑜𝑜𝑐𝑐𝑐𝑐𝑐𝑐𝑐𝑐𝑒𝑒

𝑜𝑜𝑑𝑑𝑒𝑒𝑒𝑒𝑐𝑐𝑐𝑐𝑑𝑑𝑜𝑜𝑐𝑐𝑑𝑑h𝑐𝑐𝑐𝑐𝑎𝑎𝑒𝑒 (𝑐𝑐𝑛𝑛𝑐𝑐𝑒𝑒𝑐𝑐𝑛𝑛𝑐𝑐𝑐𝑐𝑑𝑑𝑜𝑜) = 𝑘𝑘 × 𝑒𝑒

Then define the desired direction change as a function of error as

where 𝑘𝑘 is an appropriate gain, which determines how steep the turn when the robot goes off the track. You can experiment with different 𝑘𝑘 values and select the one that works best for the speed you are using.

3

Note that the sign of the error guides the turning direction. Develop a RobotC program to read the color sensor using getColorReflected() and control the motion to follow a line (dark tape on a light surface) using a proportional control scheme. Experiment how your algorithm performs with different motorPower levels and 𝑘𝑘 values. Similar to Q1, You can use setMotorSync() and control the turning with nTurnRatio and speed with nSignedPower parameters.

a) Complete part b from Q1.

**Hint:** Start with slow forward motion speed and the proportional gain (k). Increase the forward motion and change the proportional gain until you gain sufficient line tracking capability. The real test of the values comes in the second part of the Line Track. You can directly start from Point B when ‘Start Challenge’ pop up opens to find values best for turning.

b) Comment on the relationship between the forward motion speed and the proportional gain.

**Deliverables:** A word document containing the discussion, and ROBOTC file for the code. Discussion should cover how the value of TurnRatio and Power affected the Line Following. Pseudocode for the part d and Discussion.

Now that you have come up with the algorithm to follow a line, see if this solution translates to other environments too. For example, try the environment “Program Flow>Line Tracking Challenge”.

A slight hiccup here is a black box in the way of the robot in the start position. You can move it out of the way or ram into it so it is out of the way and continue with the line following challenge.