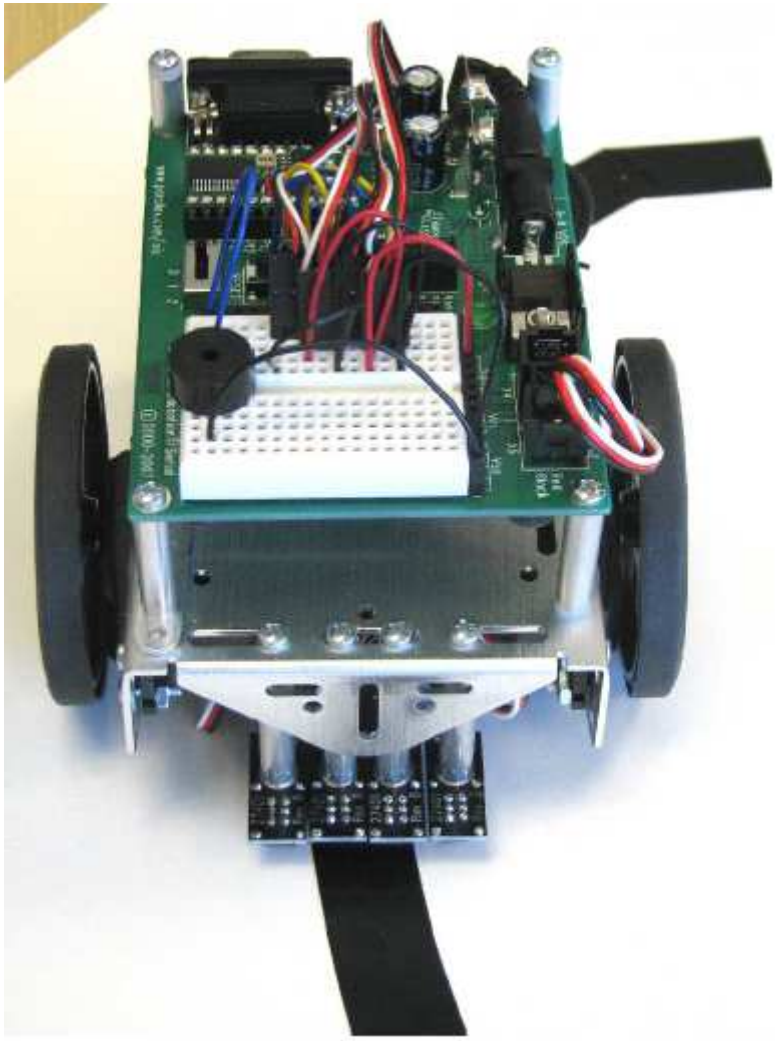
Line Following with the QTI Sensors



Start with the sample program given in the Qti manual:

Test this on the loop track to make sure your bot follows the line.

```
' LineFollowWithCheckQtis.bs2
' Navigates based on values acquired with the Check_Qtis subroutine.

' {$STAMP BS2}
' {$PBASIC 2.5}

qtis VAR Nib                    ' qti black/white states

OUTB = %1111                    ' Set OUTB bits to 1

DO                              ' Main DO...LOOP

 GOSUB Check_Qtis               ' Get QTI states

 SELECT qtis                    ' Control servo speeds/directions
  CASE %1000                    ' Rotate right
   PULSOUT 13, 650
   PULSOUT 12, 650
  CASE %1100                    ' Pivot right
   PULSOUT 13, 750
   PULSOUT 12, 650
  CASE %0100                    ' Curve right
   PULSOUT 13, 800
   PULSOUT 12, 650
  CASE %0110                    ' Straight ahead
   PULSOUT 13, 850
   PULSOUT 12, 650
  CASE %0010                    ' Curve left
   PULSOUT 13, 850
   PULSOUT 12, 700
  CASE %0011                    ' Pivot left
   PULSOUT 13, 850
   PULSOUT 12, 750
  CASE %0001                    ' Rotate left
   PULSOUT 13, 850
   PULSOUT 12, 850
  CASE ELSE                     ' Do nothing
   PAUSE 3
  ENDSELECT

LOOP

Check_Qtis:

 ' Result -> qtis variable.  0 means white surface, 1 means black surface.

 DIRB = %1111                   ' P7..P4 -> output
 PAUSE 0                        ' Delay = 230 us
 DIRB = %0000                   ' P7..P4  -> input
 PAUSE 0                        ' Delay = 230 us
 ' PULSOUT UnusedPin, 0         ' Delays = 208 + (Duration*2) us
 qtis = INB                     ' Store QTI outputs in INB

 RETURN
```

We will add LED indicators to show which of the Qti sensors are detecting the line:

In the program, it is achieved by adding just two lines of code:

OUTC = %1111

-placed here

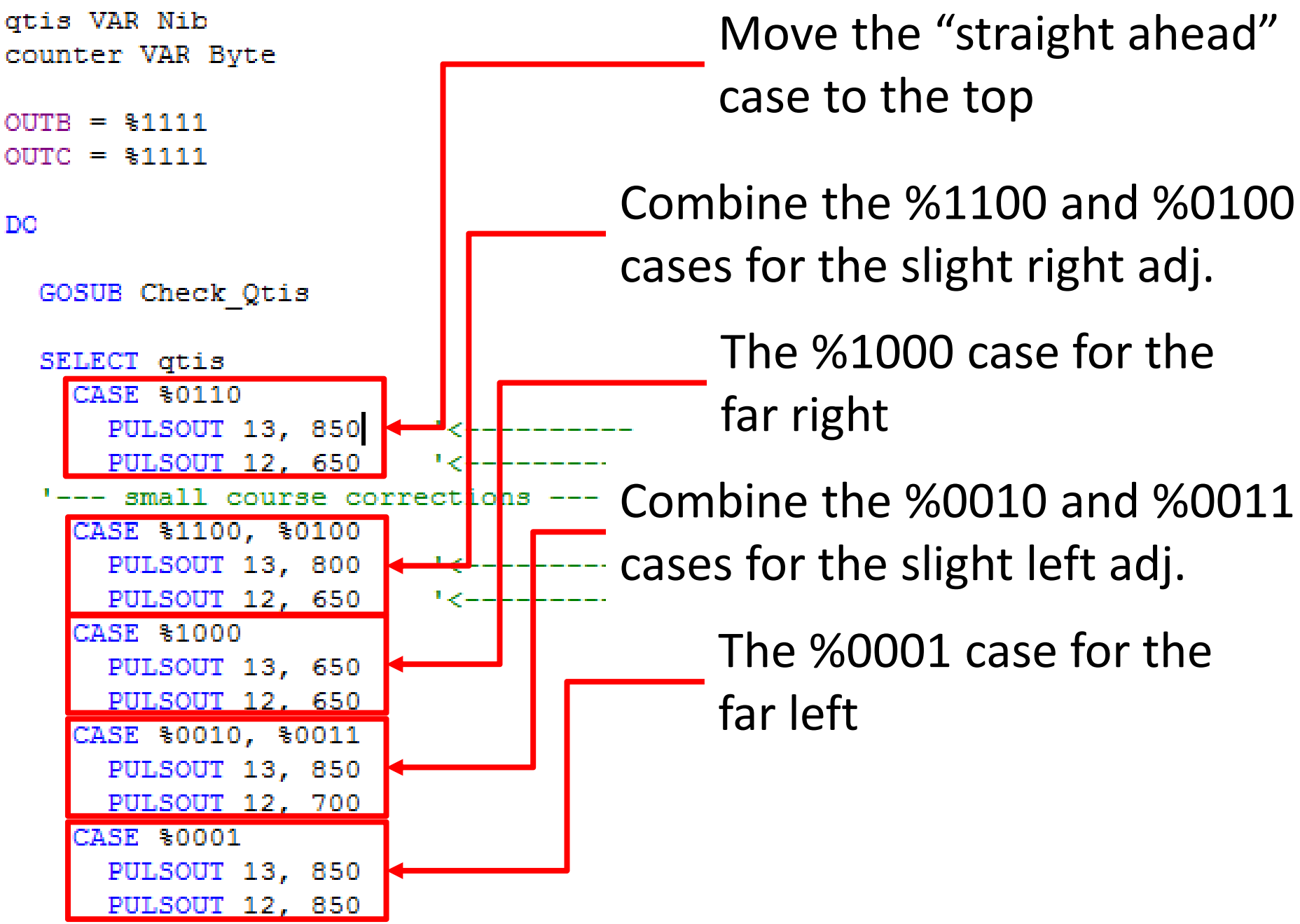
DIRC = qtis

-placed here

```
' LineFollowWithCheckQtis.bs2
' Navigates based on values acquired with the Check_Qtis subroutine.

' {$STAMP BS2}
' {$PBASIC 2.5}

qtis VAR Nib                    ' qti black/white states

OUTB = %1111                    ' Set OUTB bits to 1
OUTC = %1111
DO                              ' Main DO...LOOP

 GOSUB Check_Qtis               ' Get QTI states

 SELECT qtis                    ' Control servo speeds/directions
  CASE %1000                    ' Rotate right
   PULSOUT 13, 650
   PULSOUT 12, 650
  CASE %1100                    ' Pivot right
   PULSOUT 13, 750
   PULSOUT 12, 650
  CASE %0100                    ' Curve right
   PULSOUT 13, 800
   PULSOUT 12, 650
  CASE %0110                    ' Straight ahead
   PULSOUT 13, 850
   PULSOUT 12, 650
  CASE %0010                    ' Curve left
   PULSOUT 13, 850
   PULSOUT 12, 700
  CASE %0011                    ' Pivot left
   PULSOUT 13, 850
   PULSOUT 12, 750
  CASE %0001                    ' Rotate left
   PULSOUT 13, 850
   PULSOUT 12, 850
  CASE ELSE                     ' Do nothing
   PAUSE 3
  ENDSELECT

LOOP

Check_Qtis:

 ' Result -> qtis variable.  0 means white surface, 1 means black surface.

 DIRB = %1111                   ' P7..P4 -> output
 PAUSE 0                        ' Delay = 230 us
 DIRB = %0000                   ' P7..P4  -> input
 PAUSE 0                        ' Delay = 230 us
 ' PULSOUT UnusedPin, 0         ' Delays = 208 + (Duration*2) us
 qtis = INB                     ' Store QTI outputs in INB
 DIRC = qtis
 RETURN
```

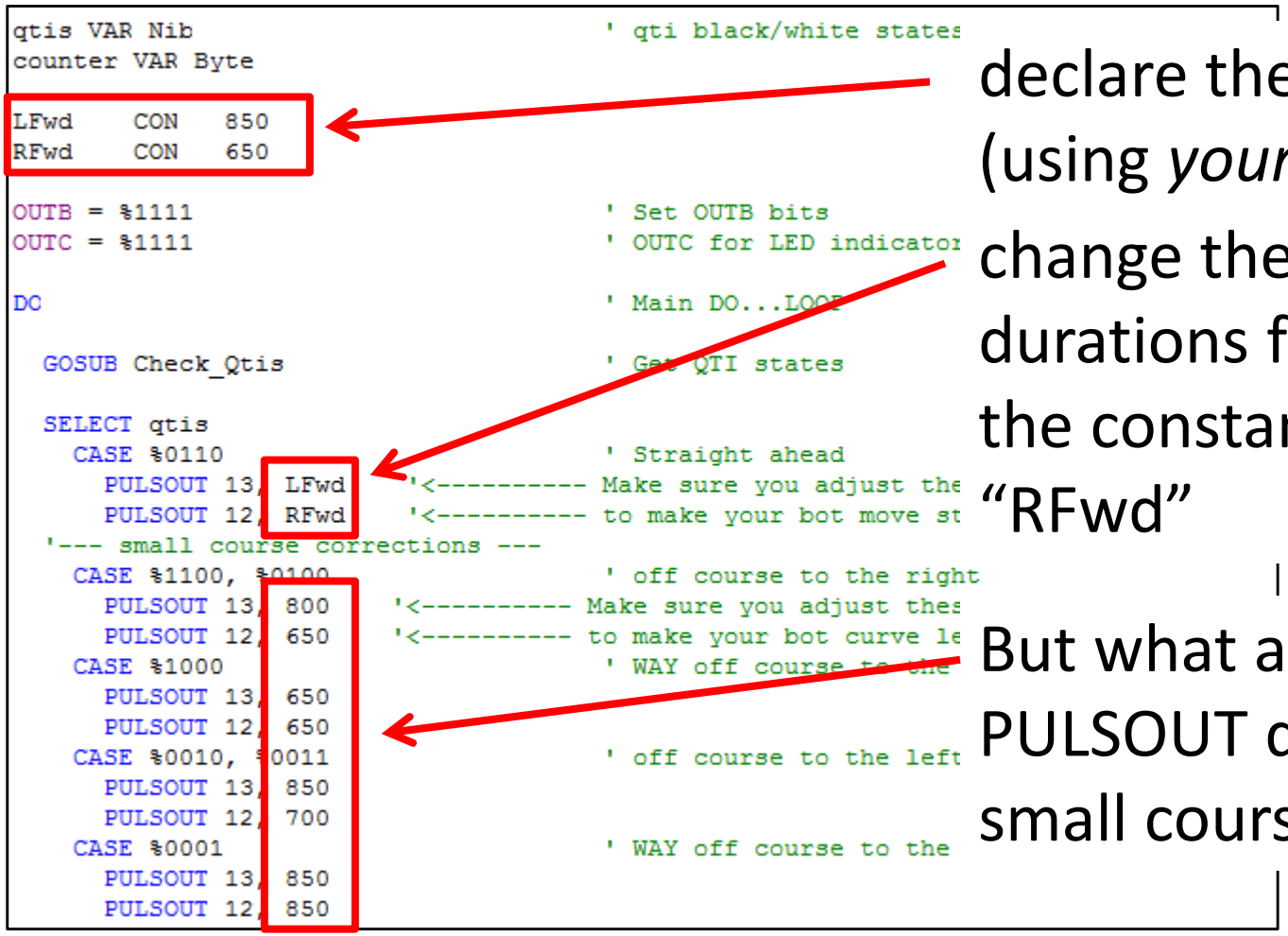
Rename your program.
(something like “QtiLineMaze.bs2”)

and SAVE IT!!!


qtis VAR Nib
counter VAR Byte
OUTB = %1111
OUTC = %1111
DO
GOSUB Check_Qtis
SELECT qtis
CASE %0110
PULSOUT 13, 850
PULSOUT 12, 650
'--- small course corrections ---
CASE %1100, %0100
PULSOUT 13, 800
PULSOUT 12, 650
CASE %1000
PULSOUT 13, 650
PULSOUT 12, 650
CASE %0010, %0011
PULSOUT 13, 850
PULSOUT 12, 700
CASE %0001
PULSOUT 13, 850
PULSOUT 12, 850
Move the “straight ahead” case to the top
Combine the %1100 and %0100 cases for the slight right adj.
The %1000 case for the far right
Combine the %0010 and %0011 cases for the slight left adj.
The %0001 case for the far left

```
qtis VAR Nib                                    '
counter VAR Byte

OUTB = %1111                                    '
OUTC = %1111                                    '

DO                                              '

  GOSUB Check_Qtis                              '

  SELECT qtis
    CASE %0110                                  ' Straight ahead
      PULSOUT 13, 850    '<---------- Make sure you adjust these values
      PULSOUT 12, 650    '<---------- to make your bot move straight
  '--- small course corrections ---
    CASE %1100, %0100                           ' off course to the right
      PULSOUT 13, 800    '<---------- Make sure you adjust these values
      PULSOUT 12, 650    '<---------- to make your bot curve left
    CASE %1000                                  ' WAY off course to the right
      PULSOUT 13, 650
      PULSOUT 12, 650
    CASE %0010, %0011                           ' off course to the left
      PULSOUT 13, 850
      PULSOUT 12, 700
    CASE %0001                                  ' WAY off course to the left
      PULSOUT 13, 850
      PULSOUT 12, 850
```

PULSOUT occurs very often, so we will use constants instead of having to write the numbers every time.

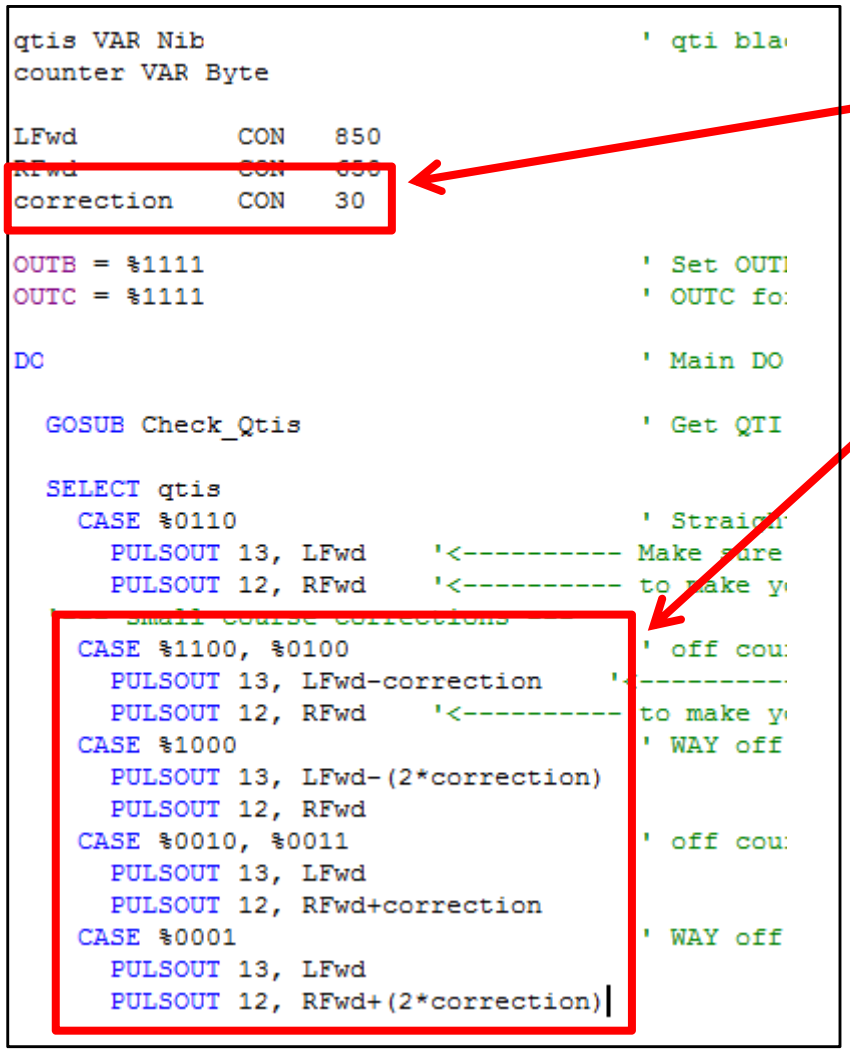


declare the constants here (using *your* values)

change the PULSOUT durations from numbers to the constants “LFwd” and “RFwd”

But what about the PULSOUT durations for the small course corrections?

These are just the “LFwd” and “RFwd” constants plus or minus another constant



declare the “correction” constant here

enter the “LFwd,” “RFwd,” and “correction” constants like this

```
CASE %1100, %0100
  PULSOUT 13, LFwd-correction
  PULSOUT 12, RFwd
CASE %1000
  PULSOUT 13, LFwd-(2*correction)
  PULSOUT 12, RFwd
CASE %0010, %0011
  PULSOUT 13, LFwd
  PULSOUT 12, RFwd+correction
CASE %0001
  PULSOUT 13, LFwd
  PULSOUT 12, RFwd+(2*correction)
```

adjust the constants until your robot makes smooth corrections back onto the line

- Before moving forward, watch the Qti video
  (go to the South Robotics website, then click on the "Course Materials" Link, then the video link)

  We will not use the program developed in this video, but it is important to understand some of the concepts presented.

- Next, click on the "Line-Maze Algorithm" file on the same page above. Read the document up to page 36.

- Our robot must decide what to do at an "intersection"

```
'---Small course corrections ---
    CASE %0100, %1100
      PULSOUT 13, LFwd-correction
      PULSOUT 12, Rfwd
    CASE %1000
      PULSOUT 13, LFwd-(2*correction)
      PULSOUT 12, RFwd
    CASE %0010, %0011
      PULSOUT 13, LFwd
      PULSOUT 12, RFwd+correction
    CASE %0001
      PULSOUT 13, LFwd
      PULSOUT 12, RFwd+(2*correction)

'---Intersection Decisions ---
    CASE %1110
```

Start a section under the small course corrections called “Intersection Decisions”

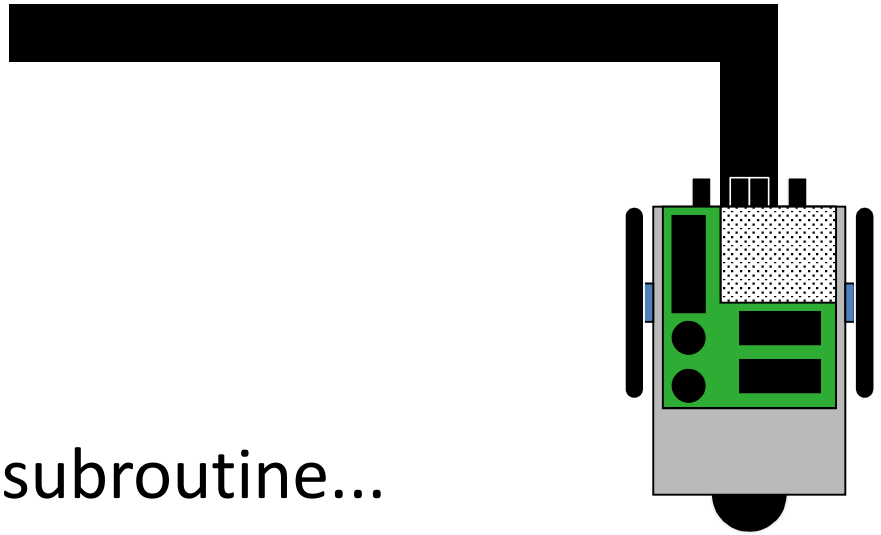
```
'---Intersection Decisions ---
   CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
```

The first decision will be what to do with a left turn or a left-T

The sensors will go from a “0110” pattern to a “1110” pattern

After moving forward an “Inch” the Qtis are checked again.

The “Inch” subroutine...



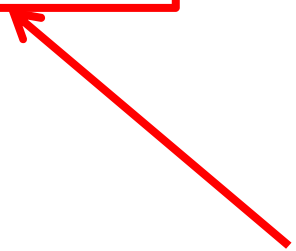
```
Inch:
  FOR counter = 1 TO 20
    PULSOUT 13, LFwd
    PULSOUT 12, RFwd
    PAUSE 20
  NEXT
  PULSOUT 13, 750
  PULSOUT 12, 750
  STOP
  RETURN
```

Makes the robot move forward a certain distance

Makes the robot stop, instead of drifting

The STOP command stops the program at this point, so we can calibrate the “Inch” subroutine. We will later make this a comment so the program ignores it.

```
Inch:
  FOR counter = 1 TO 20
    PULSOUT 13, LFwdSlow
    PULSOUT 12, RFwdSlow
    PAUSE 20
  NEXT
  PULSOUT 13, 750
  PULSOUT 12, 750
  STOP
  RETURN
```



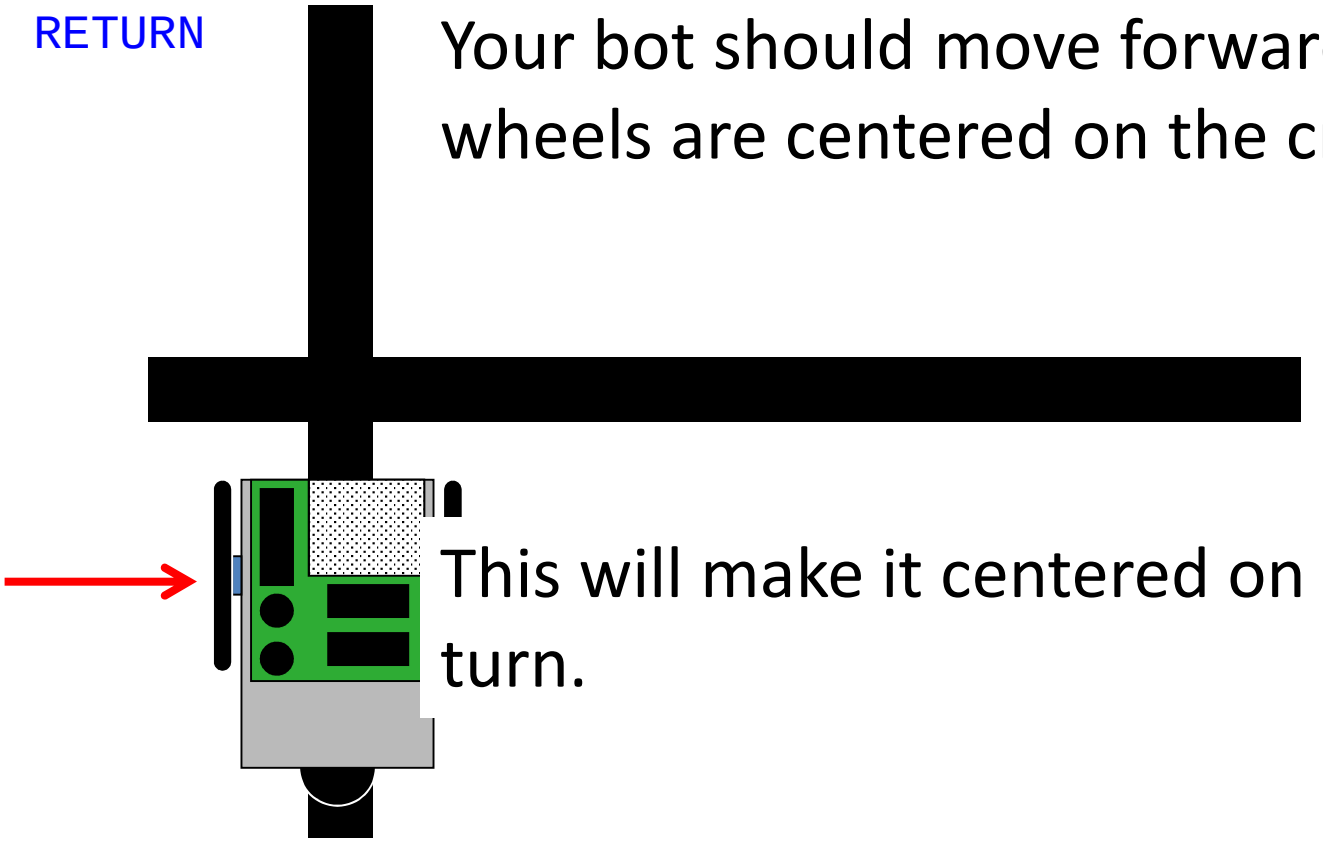
The robot may move too fast during the “Inch” subroutine, so we can use a slower forward pulse width

Be sure to declare these new constants at the beginning of your program

```
Inch:
  FOR counter = 1 TO 20
     PULSOUT 13, LFwdSlow
     PULSOUT 12, RFwdSlow
     PAUSE 20
  NEXT
  PULSOUT 13, 750
  PULSOUT 12, 750
  STOP
  RETURN
```

Next, we will adjust the *endvalue* of the counter so that the robot moves forward just enough to make the next turn.

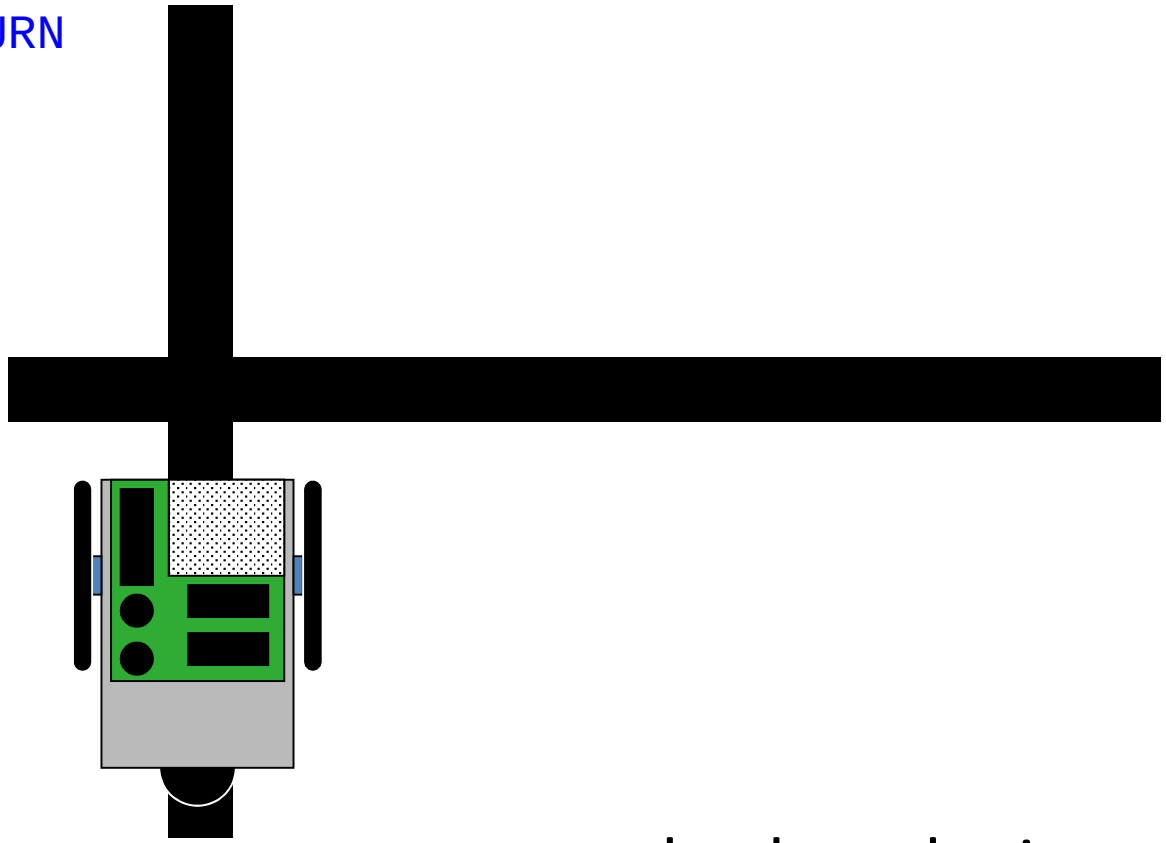
Your bot should move forward until the wheels are centered on the crossing line.



This will make it centered on the path after a turn.

```
Inch:
  FOR counter = 1 TO 20
    PULSOUT 13, LFwdSlow
    PULSOUT 12, RFwdSlow
    PAUSE 20
  NEXT
  PULSOUT 13, 750
  PULSOUT 12, 750
  'STOP
  RETURN
```

Finally, make the STOP command a comment when the “Inch” subroutine is calibrated



… back to the intersection decisions

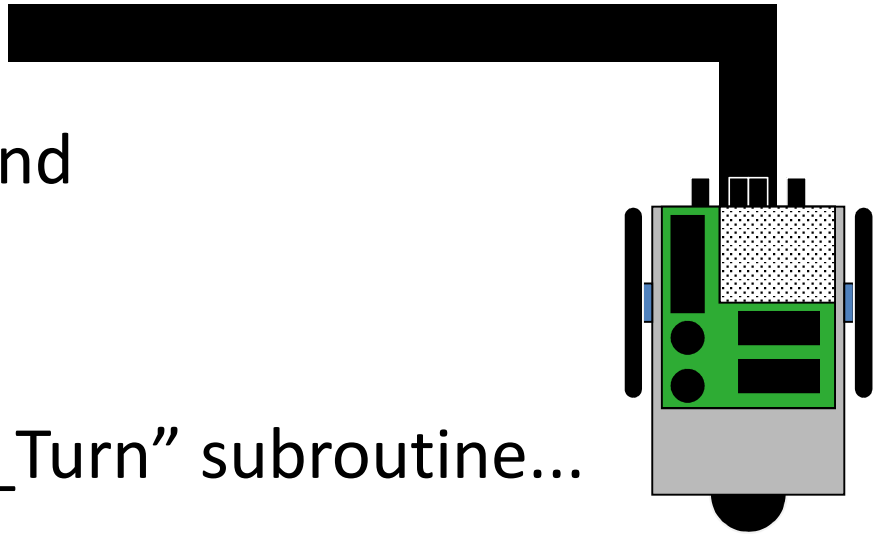
```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
      ENDSELECT
```

So… the sensors will go from a “0110” pattern to a “1110” pattern…

and after moving forward an “Inch” the Qtis are checked again.

If the pattern becomes “0000” then the turn is a simple left, and the “Left_Turn” subroutine is called.

The “Left_Turn” subroutine...



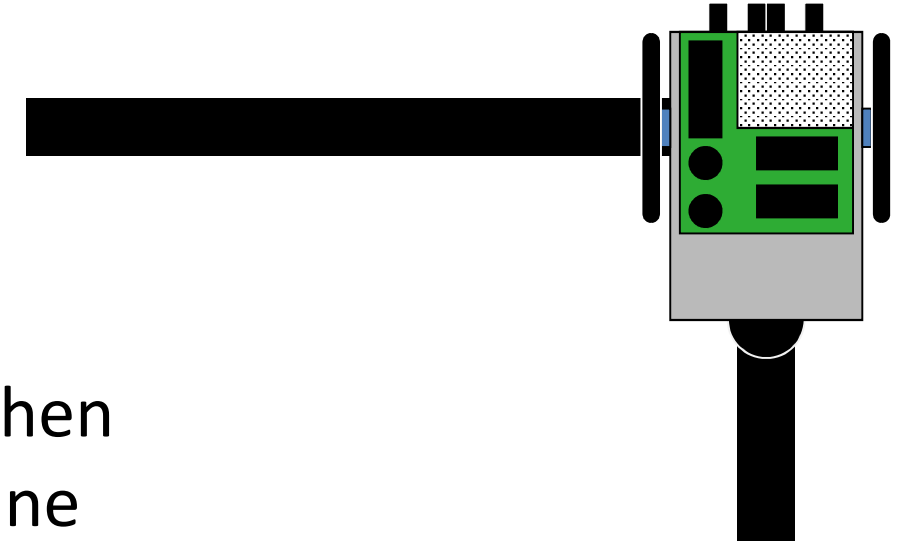
```
Turn_Left:
  FOR pulseCount = 0 TO 20
    PULSOUT 13, 650
    PULSOUT 12, 650
    PAUSE 20
  NEXT
  RETURN
```

A typical turn subroutine where the endvalue has to be adjusted to make it 90°

… will work fine, but there is a better way:

```
Left_Turn:
  DO
    GOSUB Check_Qtis
    PULSOUT 13, LRevSlow
    PULSOUT 12, RFwdSlow
    PAUSE 20
  LOOP UNTIL qtis = %0110
  PULSOUT 13, 750
  PULSOUT 12, 750
  RETURN
```

This subroutine constantly checks the qtis while turning



… and stops turning when it's centered on the line

```
Left_Turn:
  FOR counter = 1 TO 5
    PULSOUT 13, LRevSlow
    PULSOUT 12, RFwdSlow
    PAUSE 20
  NEXT
  DO
    GOSUB Check_Qtis
    PULSOUT 13, LRevSlow
    PULSOUT 12, RFwdSlow
    PAUSE 20
  LOOP UNTIL qtis = %0110
  PULSOUT 13, 750
  PULSOUT 12, 750
  RETURN
```

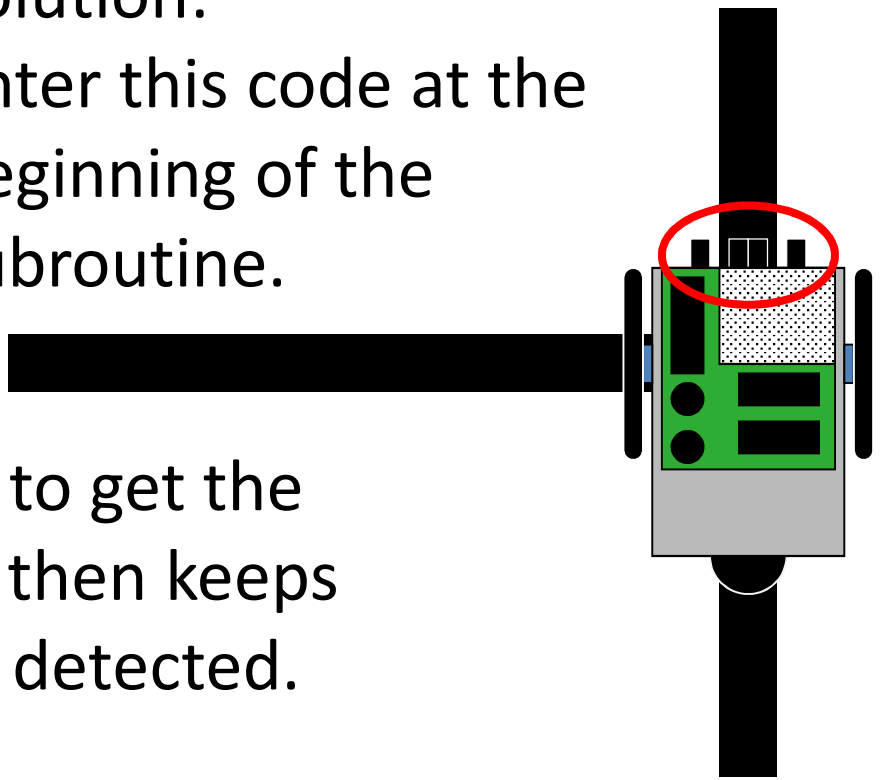
But what about a Left-T intersection?

Problem:

The qtis will sense the line and stop the turn before it even begins…

Solution:

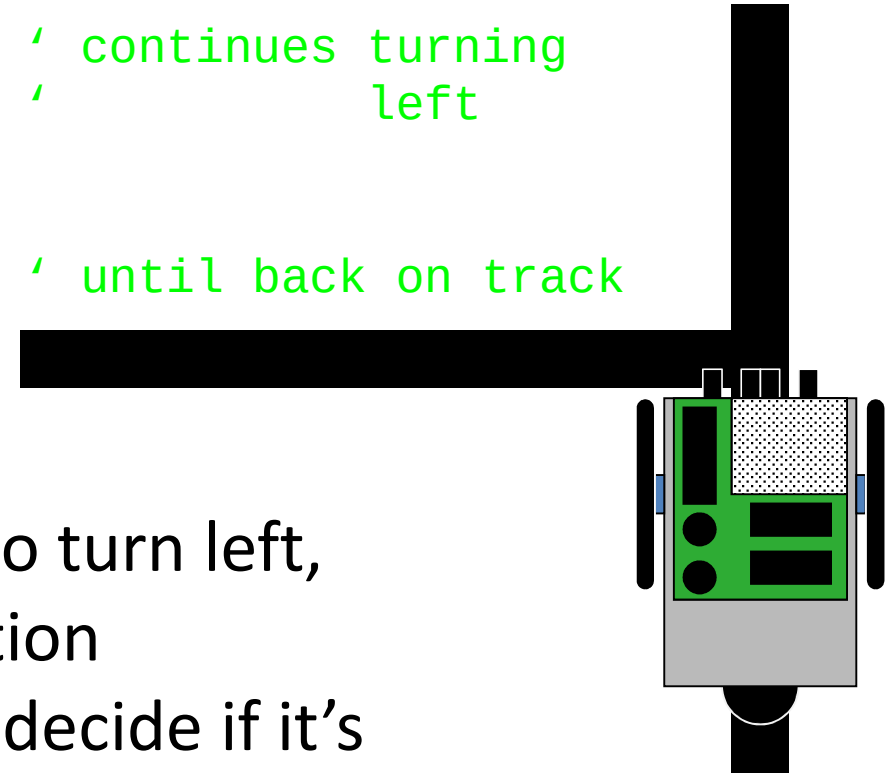
Enter this code at the beginning of the subroutine.



This turns the robot slightly to get the sensors away from the line, then keeps turning until the next line is detected.

The Left Turn Subroutine:

```
Left_Turn:
  FOR counter = 1 TO 5          ' turns slightly to
    PULSOUT 13, LRevSlow        '     get off track
    PULSOUT 12, RFwdSlow
    PAUSE 20
  NEXT
  DO
    GOSUB Check_Qtis            ' continues turning
    PULSOUT 13, LRevSlow        '            left
    PULSOUT 12, RFwdSlow
    PAUSE 20
  LOOP UNTIL qtis = %0110       ' until back on track
  PULSOUT 13, 750
  PULSOUT 12, 750
  RETURN
```



Now our robot knows how to turn left, let's go back to the intersection decisions to teach it how to decide if it's at a left turn or a left-T…

```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
```

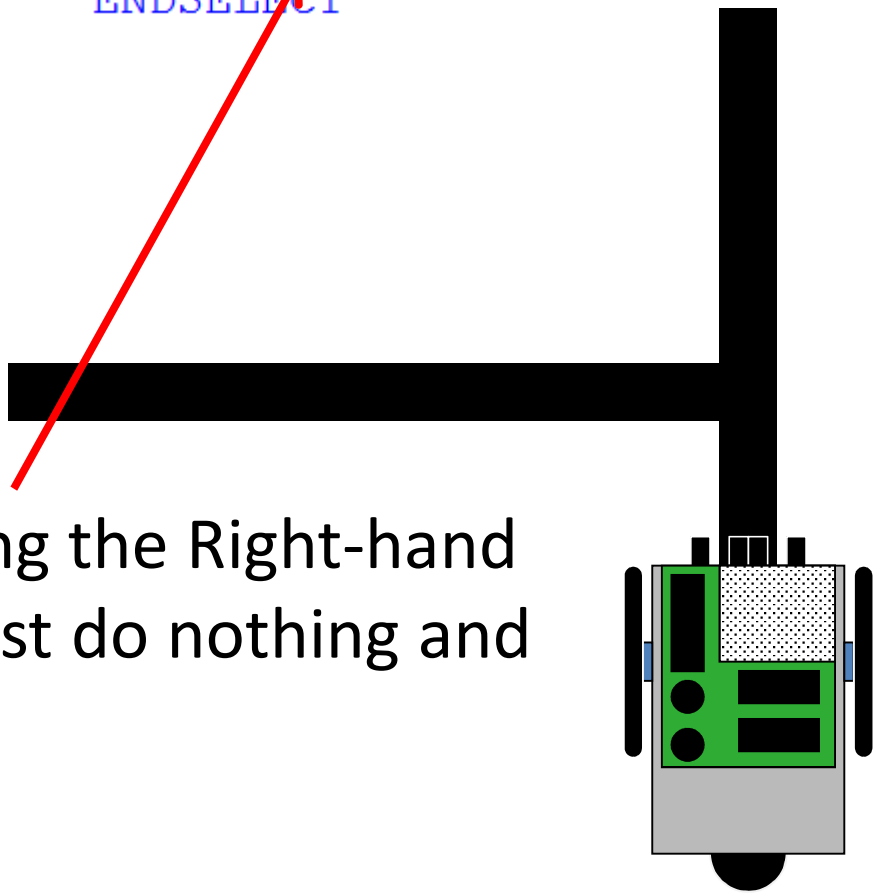
If any other pattern is detected after the “Inch,” then the turn is a Left-T

-OR-

```
CASE ELSE
  PAUSE 0
ENDSELECT
```

If you are following the Left-hand rule, your bot must turn left

If you are following the Right-hand rule, your bot must do nothing and continue forward

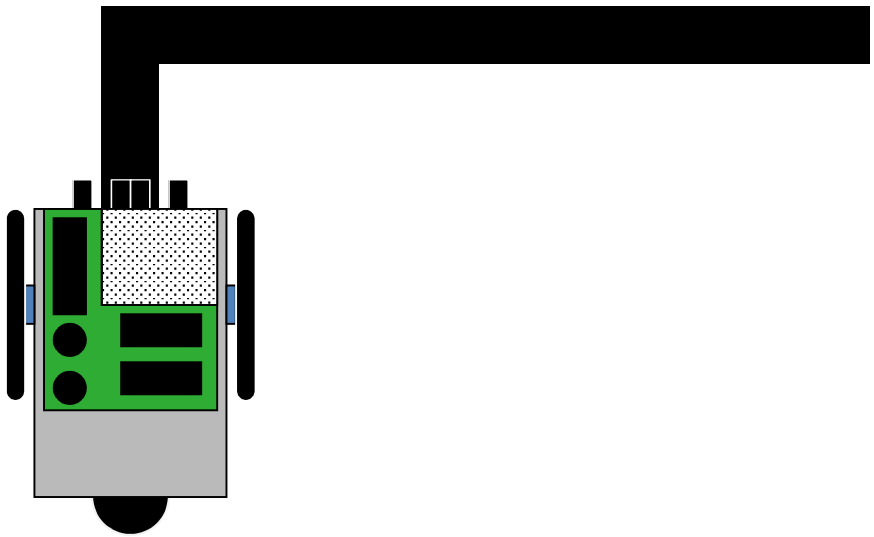


```
CASE %0111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %0000
      GOSUB Right Turn
  ENDSELECT
```

Next, your bot will decide what to do with a right turn or a right-T

The sensors will go from a “0110” pattern to a “0111” pattern

After moving forward in “Inch” the Qtis are checked again. If the pattern becomes “0000” then the turn is a simple right, and the “Right_Turn” subroutine is called.



The Right Turn Subroutine:

```
Right_Turn:
  FOR counter = 1 TO 5          ' turns slightly to
    PULSOUT 13, LFwdSlow        '     get off track
    PULSOUT 12, RRevSlow
    PAUSE 20
  NEXT
  DO
    GOSUB Check_Qtis            ' continues turning
    PULSOUT 13, LFwdSlow        '            right
    PULSOUT 12, RRevSlow
    PAUSE 20
  LOOP UNTIL qtis = %0110       ' until back on track
  PULSOUT 13, 750
  PULSOUT 12, 750
  RETURN
```

… is similar to the left turn, just with “LFwdSlow” and “RRevSlow”

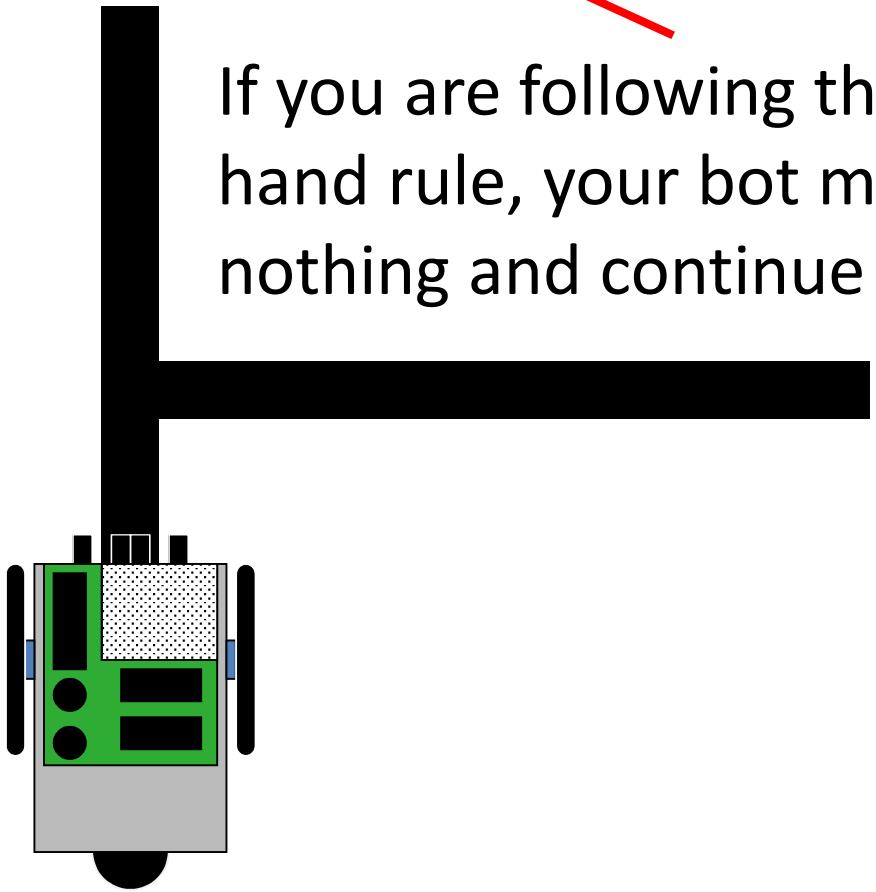
```
CASE %0111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %0000
      GOSUB Right_Turn
    CASE ELSE
      PAUSE 0
  ENDSELECT
```

If any other pattern is detected after the “Inch,” then the turn is a Right-T

-OR-

```
CASE ELSE
  GOSUB Right_Turn
ENDSELECT
```

If you are following the Left-hand rule, your bot must do nothing and continue straight



If you are following the Right-hand rule, your bot must turn right

Completed Turn Decisions

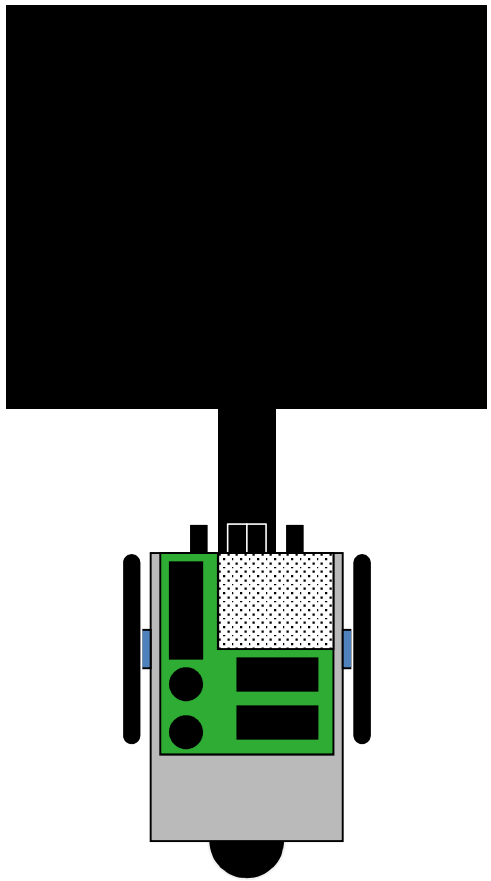
Left-Hand Rule

```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
        CASE ELSE
          GOSUB Left_Turn
      ENDSELECT

    CASE %0111
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Right_Turn
        CASE ELSE
          PAUSE 0
      ENDSELECT
```

Right-Hand Rule

```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
        CASE ELSE
          PAUSE 0
      ENDSELECT

    CASE %0111
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Right_Turn
        CASE ELSE
          GOSUB Right_Turn
      ENDSELECT
```

```
CASE %1111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %1111
      GOSUB Finish
  ENDSELECT
```



A 4-way, end-T, or the maze finish will cause a “1111” pattern

If the “1111” pattern is detected again after the “Inch” subroutine then the bot is at the finish

The “Finish” subroutine is called. We will create this subroutine later.

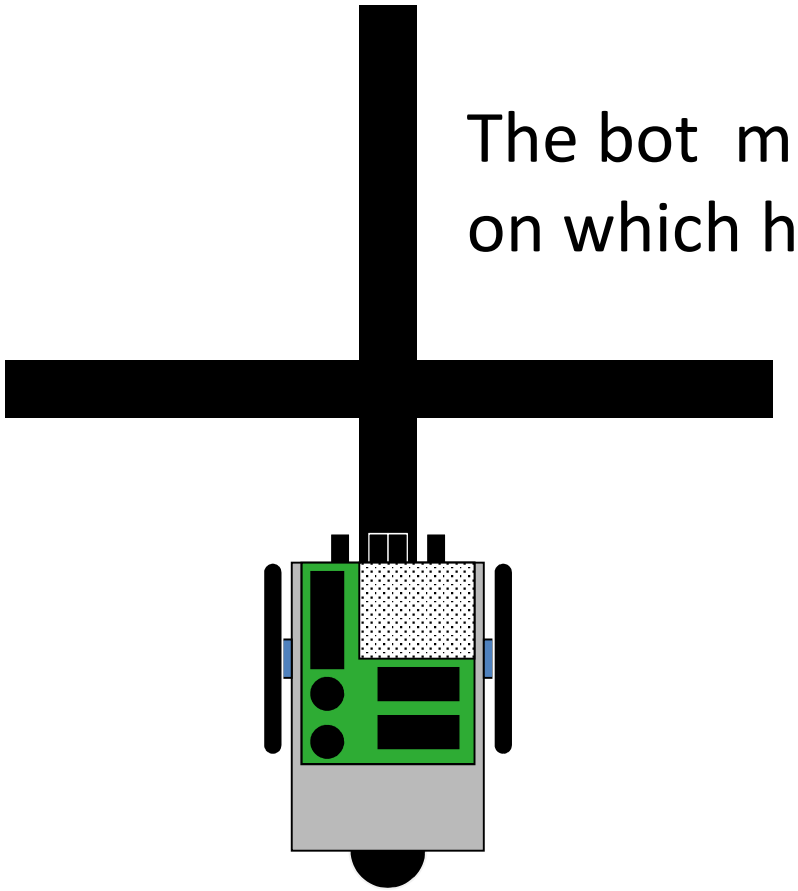
```
CASE %1111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %1111
      GOSUB Finish
    CASE ELSE
      GOSUB Left_Turn
  ENDSELECT
```

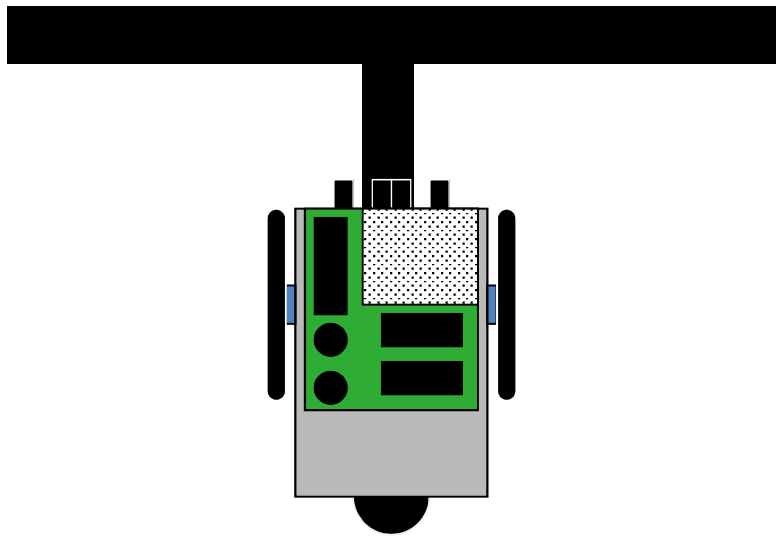
If any other pattern is detected after the “Inch”, then the bot has come to a 4-way or End-T.

-OR-

```
CASE ELSE
  GOSUB Right_Turn
```

The bot must turn left or right, depending on which hand-rule you are using

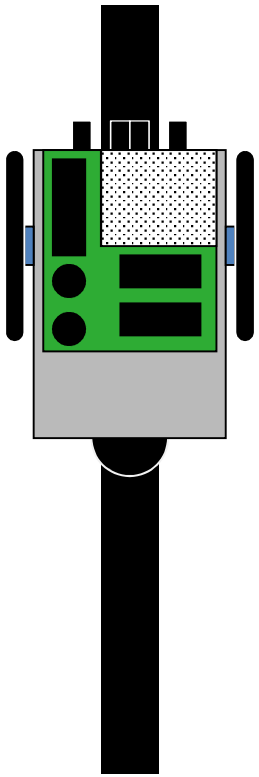




```
CASE %0000
   GOSUB Inch
   GOSUB Right_Turn
```

A dead end will cause a “0000” pattern

The bot will move forward, then turn around by calling the “Right_Turn” or “Left_Turn” subroutine



Remember, the turn subroutine will make it turn until it is back on track.

```
'---Intersection Decisions ---
CASE %1110
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %0000
      GOSUB Left_Turn
    CASE ELSE
      GOSUB Left_Turn
  ENDSELECT
CASE %0111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %0000
      GOSUB Right_Turn
    CASE ELSE
      PAUSE 0
  ENDSELECT
CASE %1111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %1111
      GOSUB Finish
    CASE ELSE
      GOSUB Left_Turn
  ENDSELECT
CASE %0000
  GOSUB Inch
  GOSUB Right_Turn
CASE ELSE
  PAUSE 3
ENDSELECT
```

Here is the full set of turn decisions

Left Turn or Left-T

Right Turn or Right-T

4-Way, End-T, or Finish

Dead-End

Any other case (robot does nothing and continues straight)

The next step will be to store the turns in memory, so that the bot can learn the maze and find the best route on the second try.

Before you go on, make sure you finish reading through the **ENTIRE** "Line Maze Algorithm" file

We will create two new variables: one to store the actual turns (called "turns") and one to act as a turn counter (called "tcount"):

```
' {$STAMP BS2}
' {$PBASIC 2.5}

qtis            VAR     Nib
counter         VAR     Byte
turns           VAR     Byte(15)
tcount          VAR     Byte

OUTB = %1111
OUTC = %1111
tcount = 0
```

Declare these variables after the "qtis" and "counter" variables…

… and set the counter to zero at the beginning of the program

```
' {$STAMP BS2}
' {$PBASIC 2.5}

qtis            VAR     Nib
counter         VAR     Byte
turns           VAR     Byte(15)
tcount          VAR     Byte
```

The (15) after “Byte” tells the program that the “turns” variable will have up to 15 characters, so we can store up to 15 turns.

“turns” will store letters that represent each turn

turns: **SULLULULLF**

tcount: **0123456789**

“tcount” will store a number that represents the position in the “turns” variable

```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
        CASE ELSE
          turns(tcount) = "L"
          tcount = tcount + 1
          GOSUB Left_Turn
      ENDSELECT
    CASE %0111
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Right_Turn
        CASE ELSE
          turns(tcount) = "S"
          tcount = tcount + 1
      ENDSELECT
```

Every time a turn decision is made, the bot must store this decision as a letter in the “turns” variable

A simple left turn is not a decision, so it is not stored

At a Left-T, the decision to turn left (according to the left-hand rule) is stored, and the turn counter is advanced by 1.

```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
        CASE ELSE
          turns(tcount) = "L"
          tcount = tcount + 1
          GOSUB Left_Turn
      ENDSELECT
    CASE %0111
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Right_Turn
        CASE ELSE
          turns(tcount) = "S"
          tcount = tcount + 1
      ENDSELECT
```

Every time a turn decision is made, the bot must store this decision as a letter in the “turns” variable

A simple right turn is not a decision, so it is not stored

At a Right-T, the decision stay straight (according to the left-hand rule) is stored, and the turn counter is advanced by 1.

```
CASE %1111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %1111
      turns(tcount) = "F"
      GOSUB Finish
    CASE ELSE
      turns(tcount) = "L"
      tcount = tcount + 1
      GOSUB Left_Turn
  ENDSELECT
CASE %0000
  turns(tcount) = "U"
  tcount = tcount + 1
  GOSUB Inch
  GOSUB Right_Turn
```

The maze finish is stored as an “F” (tcount does not need to be advanced)

At a 4-way or End-T, the decision to turn left (according to the left-hand rule) is stored, and the turn counter is advanced by 1.

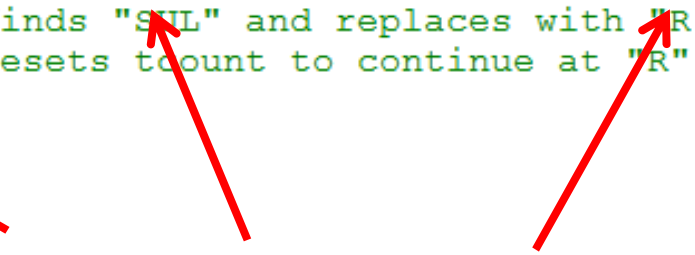
At a dead-end, a “U” is stored, and the turn counter is advanced by 1.

```
'---Intersection Decisions ---
    CASE %1110
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Left_Turn
        CASE ELSE
          turns(tcount) = "L"
          GOSUB Algorithm
          tcount = tcount + 1
          GOSUB Left_Turn
      ENDSELECT
    CASE %0111
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %0000
          GOSUB Right_Turn
        CASE ELSE
          turns(tcount) = "S"
          GOSUB Algorithm
          tcount = tcount + 1
      ENDSELECT
    CASE %1111
      GOSUB Inch
      GOSUB Check_Qtis
      SELECT qtis
        CASE %1111
          turns(tcount) = "F"
          GOSUB Algorithm
          GOSUB Finish
        CASE ELSE
          turns(tcount) = "L"
          GOSUB Algorithm
          tcount = tcount + 1
          GOSUB Left_Turn
      ENDSELECT
    CASE %0000
      turns(tcount) = "U"
      GOSUB Algorithm
      tcount = tcount + 1
      GOSUB Inch
      GOSUB Right_Turn
    CASE ELSE
```

To correct the turn for the second try, we will call a subroutine called “Algorithm”

The subroutine is called every time a letter is stored

```
Algorithm:   'this subroutine reads the u-turns and replaces with correct turns
  IF turns(tcount-2) = "S" AND turns(tcount-1) = "U" AND turns(tcount) = "L" THEN
    turns(tcount-2) = "R"                    'finds "SUL" and replaces with "R"
    tcount = tcount-2                        'resets tcount to continue at "R"
```



The algorithm looks for an “SUL” pattern in the turns variable…

… and replaces it with an “R”

Now, see if you can figure out the other replacement rules…

```
Algorithm:   'this subroutine reads the u-turns and replaces with correct turns
GOSUB Display
IF tcount > 1 THEN
  IF turns(tcount-2) = "S" AND turns(tcount-1) = "U" AND turns(tcount) = "L" THEN
    turns(tcount-2) = "R": turns(tcount-1) = " ": turns(tcount) = " " 'finds "SUL" an
    tcount = tcount-2                          'resets tcount to continue at "R"

  ELSEIF turns(tcount-2) = "L" AND turns(tcount-1) = "U" AND turns(tcount) = "L" THEN
    turns(tcount-2) = "S": turns(tcount-1) = " ": turns(tcount) = " " 'finds "LUL" an
    tcount = tcount-2                          'resets tcount to continue at "S"

  ELSEIF turns(tcount-2) = "L" AND turns(tcount-1) = "U" AND turns(tcount) = "S" THEN
    turns(tcount-2) = "R": turns(tcount-1) = " ": turns(tcount) = " " 'finds "LUS" an
    tcount = tcount-2                          'resets tcount to continue at "R"

  ELSEIF turns(tcount-2) = "R" AND turns(tcount-1) = "U" AND turns(tcount) = "L" THEN
    turns(tcount-2) = "U": turns(tcount-1) = " ": turns(tcount) = " " 'finds "RUL" an
    tcount = tcount-2                          'resets tcount to continue at "U"

  ENDIF
ENDIF
RETURN

Display:
  DEBUG CRSRXY, 0, 1, DEC ? tcount
  DEBUG CRSRXY, 0, 2, STR ? turns
  RETURN
```

To Add the Second run, we start by adding a variable:

```
' {$STAMP BS2}
' {$PBASIC 2.5}

qtis          VAR Nib
counter       VAR Byte
turns         VAR Byte(15)
tcount        VAR Byte
Trial         VAR Bit
```

The “Trial” variable will tell the robot which run it’s making:

Trial = 0 first run

Trial = 1 second run

```
' ---INITIALIZATION---
FREQOUT 15, 2000, 3000

DEBUG CLS
OUTB = %1111
OUTC = %1111
turns = 0
tcount = 0
trial = 0
GOSUB Display
```

Set the “Trial” variable to 0 to initialize it

Then we modify each intersection decision case:

```
CASE %0111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %0000
      GOSUB Right_Turn
    CASE ELSE
      IF trial = 0 THEN
        turns(tcount) = "S"
        GOSUB Algorithm
        tcount = tcount + 1
      ELSEIF trial = 1 THEN
        GOSUB SecondRun
      ENDIF
  ENDSELECT
```

If the robot is in the first run, it follows the original commands

If the robot is in the second run, it goes to the “SecondRun” subroutine

Then we modify each intersection decision case:

```
CASE %1110
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %0000
      GOSUB Left_Turn
    CASE ELSE
      IF trial = 0 THEN
        turns(tcount) = "L"
        GOSUB Algorithm
        tcount = tcount + 1
        GOSUB Left_Turn
      ELSEIF trial = 1 THEN
        GOSUB SecondRun
      ENDIF
  ENDSELECT
```

If the robot is in the first run, it follows the original commands

If the robot is in the second run, it goes to the “SecondRun” subroutine

Then we modify the %1111 case:

```
CASE %1111
  GOSUB Inch
  GOSUB Check_Qtis
  SELECT qtis
    CASE %1111
      IF trial = 0 THEN
        turns(tcount) = "F"
        GOSUB Algorithm
        tcount = tcount + 1
        GOSUB FirstFinish
      ELSEIF trial = 1 THEN
        GOSUB SecondFinish
      ENDIF
    CASE ELSE
      IF trial = 0 THEN
        turns(tcount) = "L"
        GOSUB Algorithm
        tcount = tcount + 1
        GOSUB Left_Turn
      ELSEIF trial = 1 THEN
        GOSUB SecondRun
      ENDIF
  ENDSELECT
```

If the robot reaches the finish in the first run, it stores the “F” and goes to the “FirstFinish” subroutine

If the robot reaches the finish in the second run, it goes to the “SecondFinish” subroutine

If the robot reaches a 4-way or End-T, if follows commands similar to the other turns.

The “SecondRun” subroutine:

```
SecondRun:
  GOSUB Display
  SELECT turns(tcount)
    CASE "R"
      GOSUB Right_Turn
      tcount = tcount + 1
    CASE "L"
      GOSUB Left_Turn
      tcount = tcount + 1
    CASE "S"
      tcount = tcount + 1
  ENDSELECT
  RETURN
```

Each time the bot comes to a turn in the second run, it reads the “turns” variable, follows the directions, and advances the tcount by one.

If the directions indicate “S” then no turn is made, but the tcount must be advanced.

The “FirstFinish” subroutine:

```
FirstFinish:
  PULSOUT 13, 750
  PULSOUT 12, 750
  PAUSE 20
  FOR counter = 0 TO 9
    DIRC = %0110
    FREQOUT 15, 100, 4000
    DIRC = %1001
    FREQOUT 15, 100, 5000
  NEXT
  DIRC = %0000
  '--2nd try starting routine:
  DO
    GOSUB Check_Qtis
  LOOP UNTIL qtis = %0110

  FOR counter = 1 TO 3
    DIRC = %1001
    FREQOUT 15, 500, 3000
    DIRC = %0110
    PAUSE 500
  NEXT
  DIRC = %0000
  FREQOUT 15, 1000, 6000
  tcount = 0
  trial = 1
  RETURN
```

The robot should stop … and can do a light and sound sequence (optional)

Most important part: The bot must constantly check the Qtis while waiting to be put back on the line.

A starting signal should be used. Any light/sound sequence can be used, but must be at least 2 seconds.

Finally, the tcount is reset and the trial is set to “1” (2^{nd} run)

The “SecondFinish” subroutine:

```
SecondFinish:
 PULSOUT 13, 750
 PULSOUT 12, 750
 PAUSE 20
 DO
'
'    __          __
'   |              |
'   | light/sound  |
'   | sequence     |
'   |__          __|
'
 LOOP

 END

 RETURN
```

The robot should stop

… and can do any light and sound sequence (optional)

This sequence should loop continuously…

… or an END command should be carried out to end the program.

You should now have a working robot that will go through the maze once, storing the turns and correcting the wrong turns; then it will go through the maze a second time, following the shortest route to the finish.