main ()

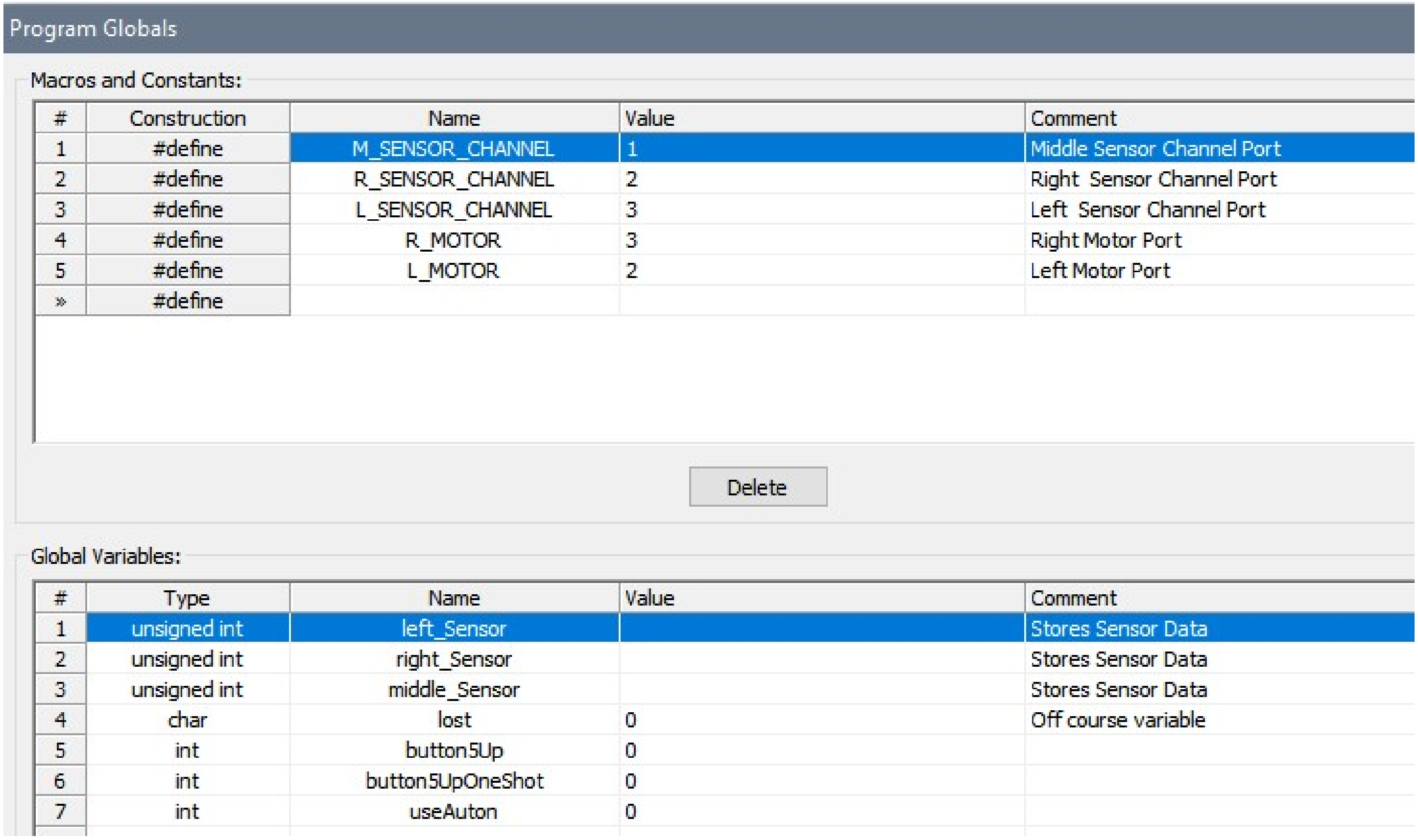
Inside the Main function for the program one typicallys sets up the infinite loop – while(1)

Check to see if driver wants to invoke autonomous – review the toggle autonomous function later in this document which sets the global variable useAuton to a 0 or a 1.

The condintional test if (useAuton) ensures the robot is only being controlled either by the autonomous routine or the driver i.e. arcade2 in this case. Any Arm / Claw manipulator code would need to followa similar pattern. In this simple example the only motors are the drive motors

This program also relies on four user defined functions (or five if you include the ability to use vex sensors which operate differently than Best sensors These will be described below.

Main is dependent upon the global variable useAuton which is an integer and would be set to a 0 for manual drive or a 1 for autonomous or line following mode



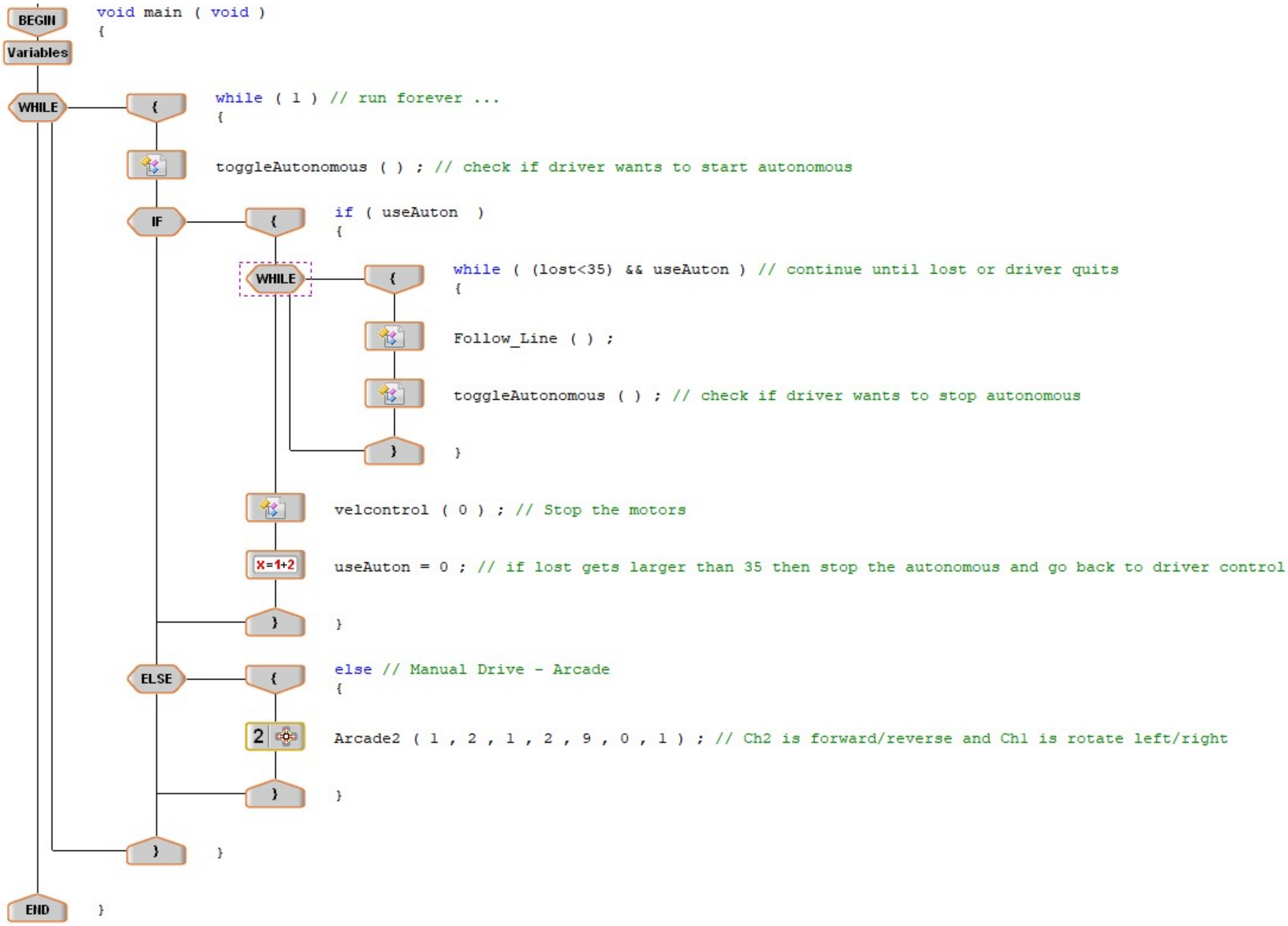
Macros and Constants:

#	Construction	Name	Value	Comment
1	#define	M_SENSOR_CHANNEL	1	Middle Sensor Channel Port
2	#define	R_SENSOR_CHANNEL	2	Right Sensor Channel Port
3	#define	L_SENSOR_CHANNEL	3	Left Sensor Channel Port
4	#define	R_MOTOR	3	Right Motor Port
5	#define	L_MOTOR	2	Left Motor Port
»	#define			

Global Variables:

#	Type	Name	Value	Comment
1	unsigned int	left_Sensor		Stores Sensor Data
2	unsigned int	right_Sensor		Stores Sensor Data
3	unsigned int	middle_Sensor		Stores Sensor Data
4	char	lost	0	Off course variable
5	int	button5Up	0	
6	int	button5UpOneShot	0	
7	int	useAuton	0	

Code image for main follows:

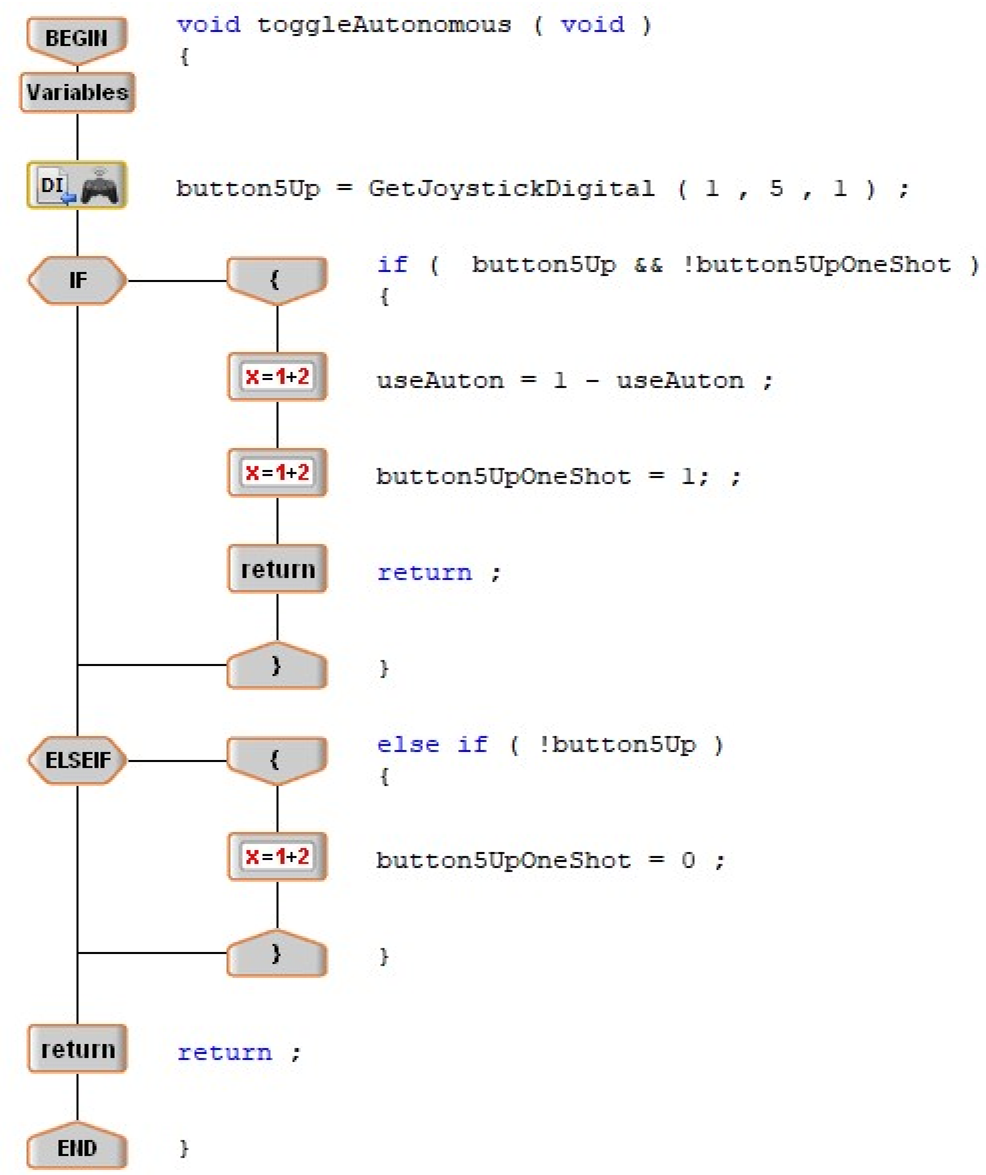


```
void main ( void )
{
     while ( 1 ) // run forever ...
     {
          toggleAutonomous ( ) ; // check if driver wants to start autonomous
          if ( useAuton  )
          {
               while ( (lost<35) && useAuton ) // continue until lost or driver quits
               {
                    Follow_Line ( ) ;
                    toggleAutonomous ( ) ; // check if driver wants to stop autonomous
               }
               velcontrol ( 0 ) ; // Stop the motors
               useAuton = 0 ; // if lost gets larger than 35 then stop the autonomous and go back to driver control
          }
          else // Manual Drive - Arcade
          {
               Arcade2 ( 1 , 2 , 1 , 2 , 9 , 0 , 1 ) ; // Ch2 is forward/reverse and Ch1 is rotate left/right
          }
     }
}
```

toggleAutonomous ()

This function changes the global variable useAuton from zero to one , or one to zero each time button5Up is pressed on the controller. The button must be released and pressed again to toggle the global variable. This is enabled by the following logic and use of another global variable called button5UpOneShot which is also an integer and set to either a one or zero.

This toggle logic can be very useful if the programmer would like to enable and disable capability with just one button on the controller.



```
void toggleAutonomous ( void )
{

button5Up = GetJoystickDigital ( 1 , 5 , 1 ) ;

    if (  button5Up && !button5UpOneShot )
    {

    useAuton = 1 - useAuton ;

    button5UpOneShot = 1; ;

    return ;

    }

    else if ( !button5Up )
    {

    button5UpOneShot = 0 ;

    }

return ;

}
```

Follow_Line()

This is basically the same function as provided in the example that comes with easy C however the global variable useAuton has been added to each of the while loop tests to allow for user to break out of the loop manually if needed.

Follow Line calls the function velControl to do one of the following:

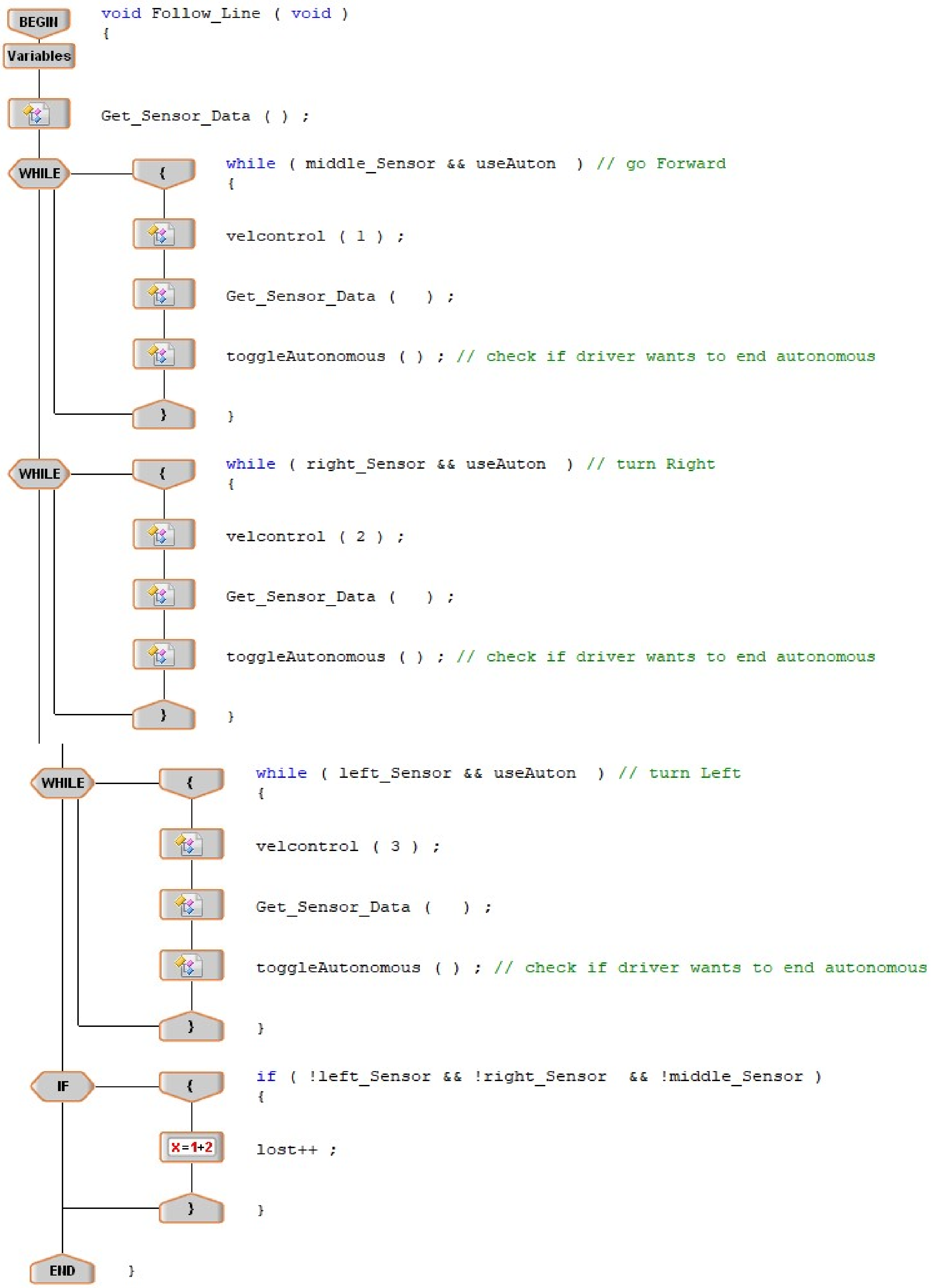
Drive Straight;

Turn Right;

Turn Left;

Stop;

Note the picture was pasted in two parts so there is an unavoidable break in the main function line which should be ignored.

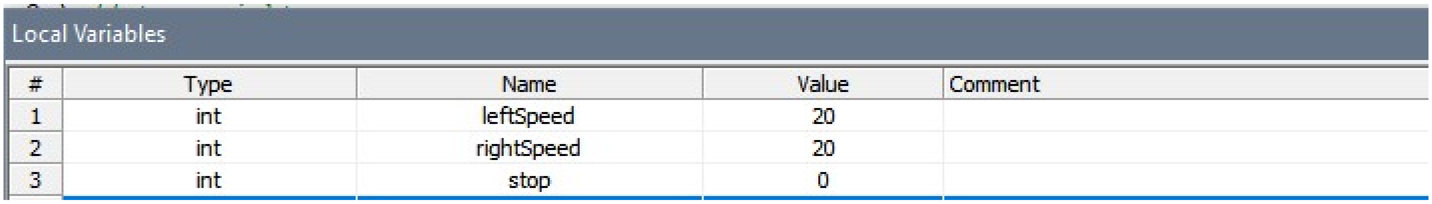


```
void Follow_Line ( void )
{

Get_Sensor_Data ( ) ;

    while ( middle_Sensor && useAuton  ) // go Forward
    {

    velcontrol ( 1 ) ;

    Get_Sensor_Data (   ) ;

    toggleAutonomous ( ) ; // check if driver wants to end autonomous

    }

    while ( right_Sensor && useAuton  ) // turn Right
    {

    velcontrol ( 2 ) ;

    Get_Sensor_Data (   ) ;

    toggleAutonomous ( ) ; // check if driver wants to end autonomous

    }

    while ( left_Sensor && useAuton  ) // turn Left
    {

    velcontrol ( 3 ) ;

    Get_Sensor_Data (   ) ;

    toggleAutonomous ( ) ; // check if driver wants to end autonomous

    }

    if ( !left_Sensor && !right_Sensor  && !middle_Sensor )
    {

    lost++ ;

    }

}
```

velControl ()

VelControl, or Velocity Control, is the main motive output for the line following routine

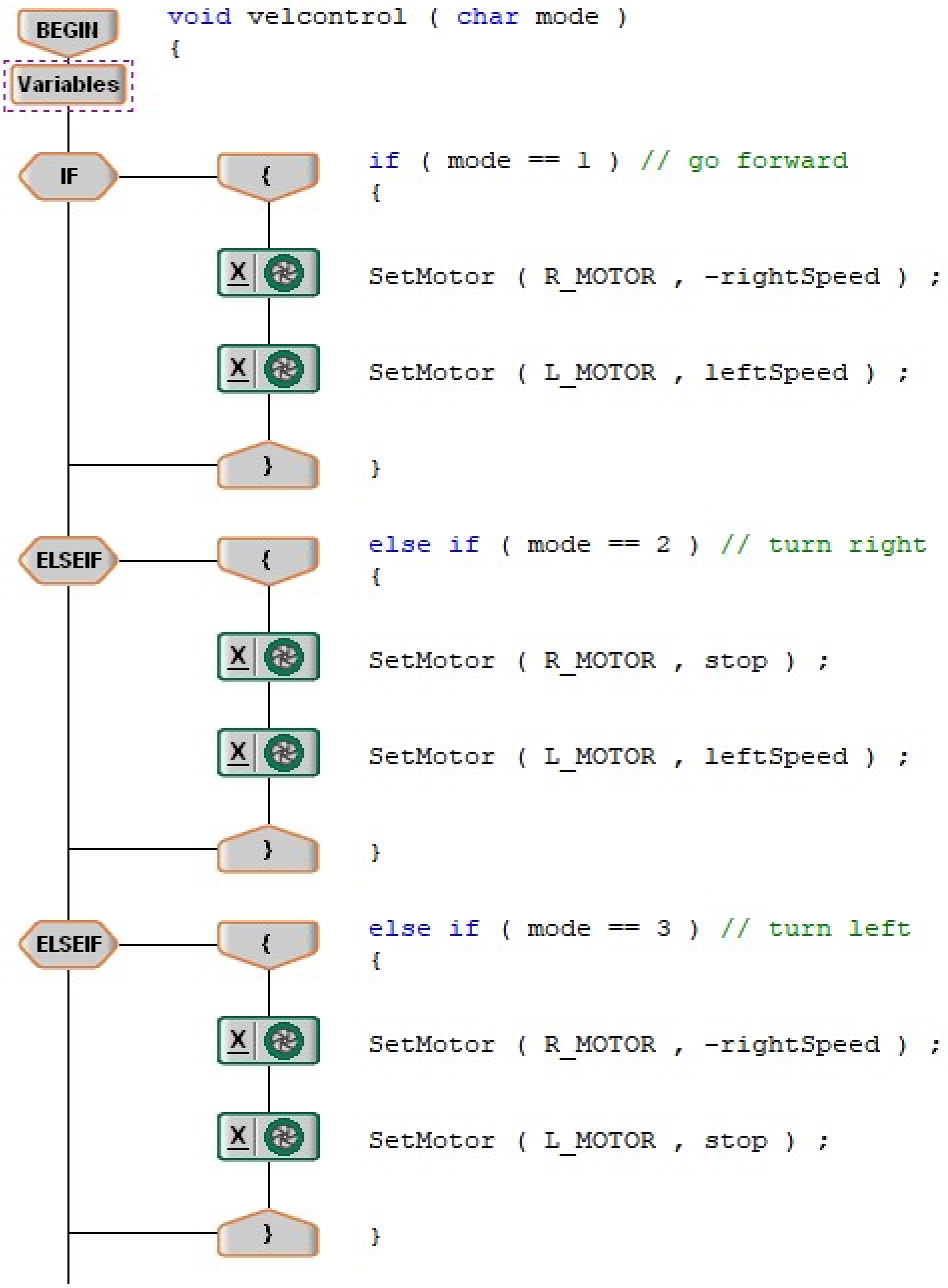
There are local variables for leftSpeed,rightSpeed and stop which are for tuning the max speed of the drive motors in line follow mode



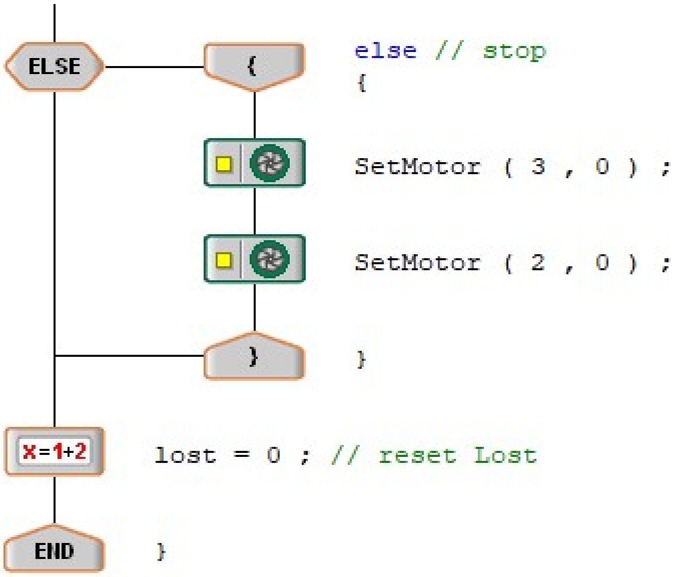
Local Variables

#	Type	Name	Value	Comment
1	int	leftSpeed	20	
2	int	rightSpeed	20	
3	int	stop	0	

The 'reset' of lost at the end is part of the overall line following example logic. I did not design or test with it in this location. It should be considered in understanding the overall effectiveness of this example.



```
void velcontrol ( char mode )
{

      if ( mode == 1 ) // go forward
      {

      SetMotor ( R_MOTOR , -rightSpeed ) ;

      SetMotor ( L_MOTOR , leftSpeed ) ;

      }

      else if ( mode == 2 ) // turn right
      {

      SetMotor ( R_MOTOR , stop ) ;

      SetMotor ( L_MOTOR , leftSpeed ) ;

      }

      else if ( mode == 3 ) // turn left
      {

      SetMotor ( R_MOTOR , -rightSpeed ) ;

      SetMotor ( L_MOTOR , stop ) ;

      }
```

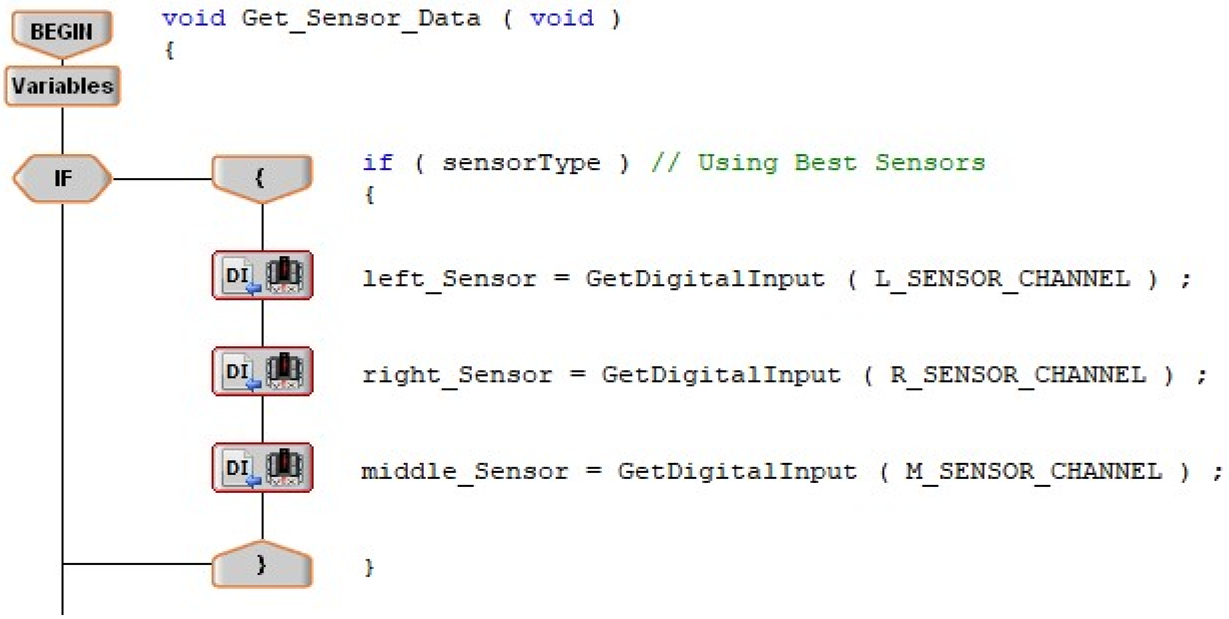


```
          else // stop
          {
          SetMotor ( 3 , 0 ) ;
          SetMotor ( 2 , 0 ) ;
          }
lost = 0 ; // reset Lost
}
```

Get_Sensor_Data ()

Get sensor data is more complicated that would be required for use in a BEST competition as it is currently set up to use either BEST sensors of VEX Sensors depending on the value of the variable sensorType. It was coded this way as the author does not have access to the BEST sensors, however wanted to set up the logic in a fashion that would allow VEX sensors to mimic a BEST sensor.

For BEST one would only need this part without the 'if'

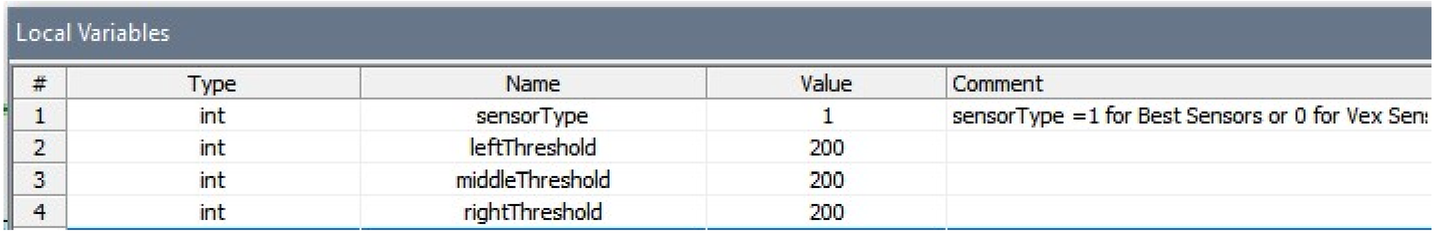


```
void Get_Sensor_Data ( void )
{

    if ( sensorType ) // Using Best Sensors
    {

    left_Sensor = GetDigitalInput ( L_SENSOR_CHANNEL ) ;

    right_Sensor = GetDigitalInput ( R_SENSOR_CHANNEL ) ;

    middle_Sensor = GetDigitalInput ( M_SENSOR_CHANNEL ) ;

    }
```

For completeness the remainder of the code is included below.

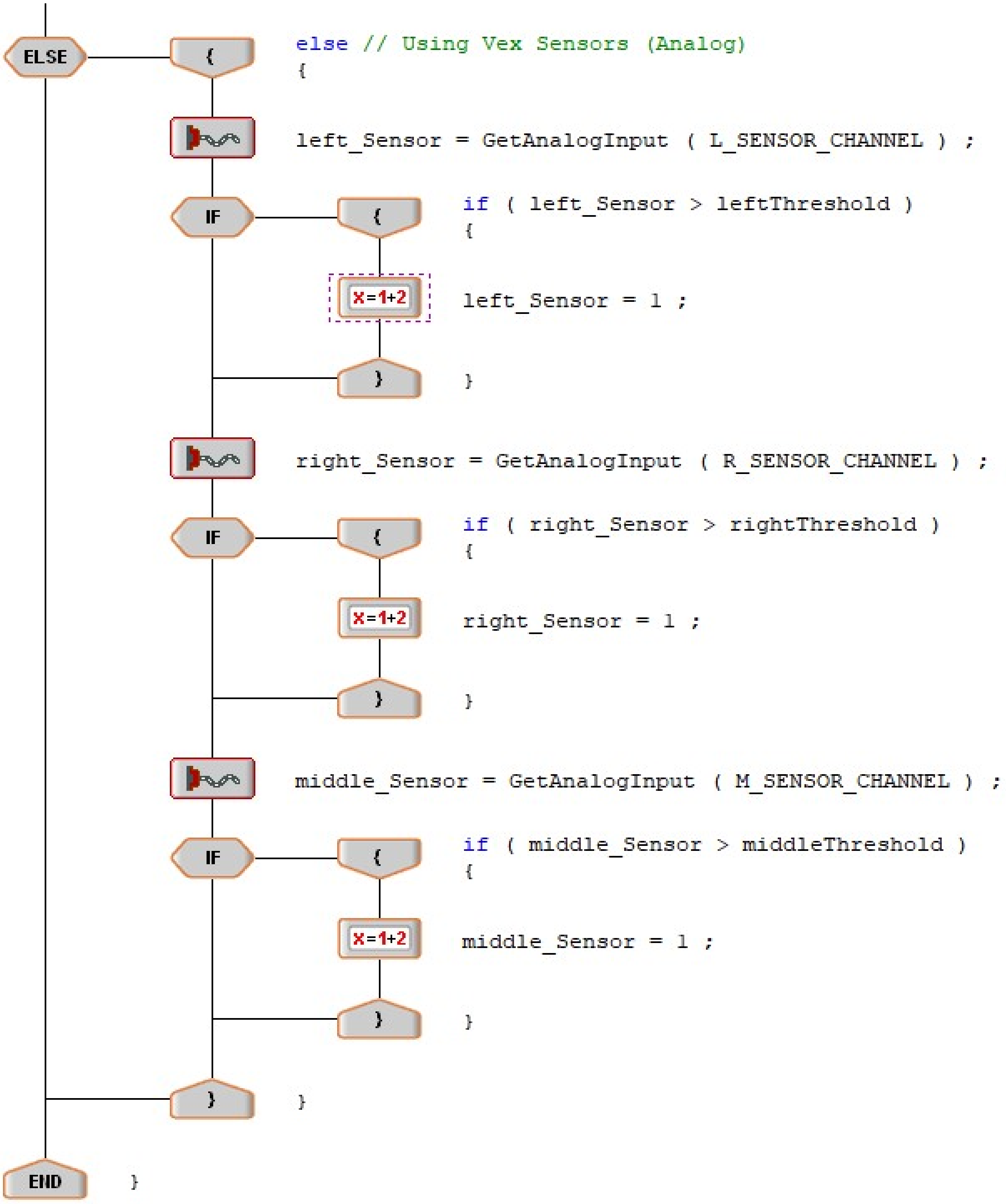
This module uses a local variable 'sensorType' set to and nonzero positive value for Best sensors and 0 for Vex Sensors

The other local variables are for tuning the detection threshold of the VEX sensors The example had them set to 200 but they would likely need to be adjusted for your specific VEX sensor configuration



Local Variables

#	Type	Name	Value	Comment
1	int	sensorType	1	sensorType =1 for Best Sensors or 0 for Vex Sen
2	int	leftThreshold	200	
3	int	middleThreshold	200	
4	int	rightThreshold	200	



```
else // Using Vex Sensors (Analog)
{

left_Sensor = GetAnalogInput ( L_SENSOR_CHANNEL ) ;

            if ( left_Sensor > leftThreshold )
            {

            left_Sensor = 1 ;

            }

right_Sensor = GetAnalogInput ( R_SENSOR_CHANNEL ) ;

            if ( right_Sensor > rightThreshold )
            {

            right_Sensor = 1 ;

            }

middle_Sensor = GetAnalogInput ( M_SENSOR_CHANNEL ) ;

            if ( middle_Sensor > middleThreshold )
            {

            middle_Sensor = 1 ;

            }

}

      }
```

Note for the Best competition students will still need to provide a mechanism to stop ‘autonomous’ one the destination is reached, and to manipulate a game piece during that phase as well.

https://www.youtube.com/watch?v=x4iBxsRjGYs&feature=youtu.be