

Customizing the Isaac 16

Beyond the Basics

Authored by:

Branden Gunn

Plymouth North High School Robotics Program, 1996 – 1999

Stamp Applications Intern, Parallax Inc., Summer 2000

ElectroMechanical Intern, IRobot Corp. Summer 2001

ElectroMechanical Engineering Student, Wentworth Inst. Of Tech., 1999 – 2004

Edited for BattleBots IQ By Michael Bastoni

Reviewing the Basics

Let's review the basics of the control system. Each joystick has two axes and a thumb wheel. In the program, the X axis is given a name like p1_x or p2_x. The number represents which port the joystick is plugged into. For example: p3_x would be the X axis on the joystick plugged into Port 3. Similarly, the Y axis is given the name of p1_y or p2_y. Since there are 4 ports, the variables p3_x, p3_y, p4_x, and p4_y are declared. The position of the stick on each axis is given as a number between 0 and 255. When joystick 1 is centered, the value of p1_x is 127, and the value of p1_y is also 127, or about half of 255. You can adjust the value of the centered sticks by adjusting the trim wheel for each axis.

Changing the Default Code

To allow us to change the outputs from the control system, we need to make a few initial modifications to the default program. At the very end of the program is the line:

```
Serout USERCPU, OUTBAUD,  
[255,255,p1_y,relayA,p2_y,relayB,p3_y,p4_y,  
p1_x,p2_x,p3_x,p4_x,p1_wheel,p2_wheel,p3_wheel,p4_wheel,127,  
127,127,127]
```

This line actually fits on one line in the program, and is not split up as it is shown here. In order to change the outputs additional variables must be added. For now, we only need to change a few of the first ones so that it reads:

```
Serout USERCPU, OUTBAUD,  
[255,255,PWM1,relayA,PWM2,relayB,p3_y,p4_y,
```

```
p1_x,p2_x,p3_x,p4_xp1_wheel,p2_wheel,p3_wheel,p4_wheel,127,  
127,127,127]
```

Comments, the Road Map to Remembering

You can change as many of these as you want, but remember to follow the template given at the end of the code. Whenever you make a change to your program, remember to write a comment explaining what you changed. All comments start with an apostrophe, and can be as long as you like on the same line. Writing comments will save you time later when you try and read your changes and can't remember how they work.

I Do Declare....

To use the two variables we've added, we need to *declare* them, or, tell the program that we will be using them. The beginning of the program is the best place for variable declarations, so we will add them to the existing list of variables. Add these two lines near the beginning:

```
PWM1      var  byte      `this will be for the left motor  
PWM2      var  byte      `this will be for the right motor
```

Check Your Memory

Each variable takes up a byte of memory, and bytes are limited. In order to see how many bytes of space (memory) you have left, do this; press the F8 key from within the program editor. This will give you the Memory map screen. The RAM map on the right shows you how much variable space you have used. The default program has seven free bytes for new variables.

It's the Joystick....

Moving the stick forward increases the value of `p1_y`. Moving the stick to the right increases the value of `p1_x`. Refer to the graphics in this lesson in order to better understand this concept. Ideally, the left drive motor would turn in one direction for values of PWM1 between 128 and 255, and it would turn in the opposite direction for values of PWM1 between 126 and 0. In actuality, the motor does not respond until the value of PWM1 is at least 5 away from the center value of 127.

What Goes in Must Come Out...Sometimes

We can alter the output going to a motor by writing a line of code to change the value based on a mathematical formula. For example, to simply copy the input to the output, the program needs two lines:

```
PWM1 = p1_y  
PWM2 = p2_y
```

These are the lines that copy the inputs (joystick) to the outputs (motors). If we wanted to divide every input value by 2 before sending it to the motors, we could re-write those two lines to read:

```
PWM1 = p1_y / 2 'divide the inputs by two, and send them to the outputs  
PWM2 = p2_y / 2
```

“There is No Spoon”, and no Remainder Either

An important item to note here is that the programming language (called PBasic) does not calculate the fractional part or remainder from division. That is to say that $7 / 2$ is 3, instead of 3.5. PBasic also does not round numbers up. It simply cuts off any number after the decimal point.

Robot Control is a Key to Winning

The default control style is called Tank style, or two-stick control. This means that the 1st (left, connected to port 1) stick controls the left motor, and the 2nd (right, connected to port 2) stick controls the right motor.

Why would you want to change how the controls work? There are many features you can add to the control system to make your robot easier to control. One feature is the ability to drive using only one joystick, usually connected to port 1. Some people are better at driving when they only need one hand on a joystick, and can have the other hand free to do something else.

To change your system to use only one joystick, we need to choose what you would like the single stick to do. In this example, we will make the Y axis control the speed of the robot, and the X axis will control how fast the robot is turning. When the X axis is centered on 127, the robot will only travel forward or backwards, based on the input from the Y axis. One method of figuring out which way the motors will be going is to use

```
PWM1 = p1_y - p1_x + 127
PWM2 = p1_y + p1_x - 127
```

Zen and the Art of Pbasic

PBasic math can't understand negative numbers, so we need to be careful that the program will never use any. A safe way to do this is to add a large value to the beginning of each calculation, and subtract the same value at the end of the calculation. A good number for this must be in the range of 0 to 65535 (the largest number the program can handle). 2000 is an arbitrary choice that will do what we want. Since our output must be within the range of 0 to 255, we can use the Min and Max modifiers to make sure the answer is within that range. The value will never be allowed to go above the max value, or below the min value.

'the next two lines change the control from two sticks to one stick.

```
PWM1 = (((2000 + p1_y - p1_x + 127) Min 2000 Max 2254) -
2000)
PWM2 = (((2000 + p1_y + p1_x - 127) Min 2000 Max 2254) -
2000)
```

This algebraic expression insures that the values of PWM1 and PWM2 remain between 0 and 254.

Programmers Make Robot Drivers Look Good

A useful feature to aid in the control of your robot is to be able to adjust the sensitivity of the joystick. There are two easy ways to control the maximum speed of the robot. Both involve using another input to be the control. A handy control that is built right into the joystick is the thumb wheel. The first method is to use the value of the thumbwheel to clip the output so that it never passes a certain value. Moving the joystick any further would simply not do any more to move the robot. This isn't very efficient, but can be useful. A better way might be to use the thumbwheel to scale down all of the outputs. When we scale down all of the output values, we also scale down the center value of the output, so we need to re-center the output by adding 128 minus half of the scale value back to the output.

```
'these first two lines simply copy the inputs to the
outputs.
PWM1 = p1_y
PWM2 = p2_y
```

```
'the next 2 lines run the robot at a speed selected with
the p1_wheel
PWM1 = ((PWM1 * p1_wheel)/256) + (128 - (p1_wheel/2)) min 0
max 255
PWM2 = ((PWM2 * p1_wheel)/256) + (128 - (p1_wheel/2)) min 0
max 255
```

What Would You Want in an Ideal Control System?

For one, we would like to choose between using one joystick or two for controlling the robot. We want the system to default to one joystick if the second joystick isn't plugged in. This means that we need some way to detect if the second stick is plugged in. One simple way to do this is to use the thumb wheel as an indicator that the joystick is there. When the stick is not plugged in, the thumbwheel's value is defaulted to 127. We can take advantage of the fact that PBasic doesn't calculate decimal places by dividing p2_wheel by some value greater than 127. p2_wheel/200 is calculated as 1 whenever p2_wheel is over 199. Otherwise, p2_wheel/200 is a 0. if we move the thumbwheel to either of it's extremes, we have an input into the program that will be either a 1 or a 0, and will not change until you move the wheel.

Let's take this one line at a time:

```
PWM1 = (p1_y * (p2_wheel/200))
```

If p2_wheel/200 is 1, then multiplying it by the left joystick's y value will not change anything. If p2_wheel/200 is 0, then we are making the output 0. so if the thumbwheel is up (1) then we can use tank drive. Now we the same thing to the other output and the right joystick.

```
PWM2 = (p2_y * (p2_wheel/200))
```

Now we will add the option for one-stick control.

```
PWM1 = PWM1 + (((2000 + p1_y - p1_x + 127) Min 2000 Max 2254) - 2000)*(1-
(p2_wheel/200))
PWM2 = PWM2 + (((2000 + p1_y + p1_x - 127) Min 2000 Max 2254) - 2000)*(1-
(p2_wheel/200))
```

1-(p2_wheel/200) is always a 1 or a 0 as well, but it is the opposite of the value we used for the tank drive. When the value for tank drive is multiplied by 1, it doesn't change, but the value for one-stick is multiplied by 0, canceling it out. There will never be a value of p2_wheel that will activate both one-stick and two-stick control.

Then we would add the two lines for scaling speed, as shown above.

Finally, we can add some lines that will help you set the trim values on your joysticks.

```

debug 0          'clear the debug window
debug "X1: ",dec3 p1_x,      " Y1: ",dec3 p1_y ,cr 'show the
1st joystick's inputs
debug "W2: ",dec3 p2_wheel, " Y2: ",dec3 p2_y ,cr 'show the
2nd joystick's inputs
debug " L: ",dec3 PWM1,      " R: ",dec3 PWM2 ,cr 'show the
outputs to the left and right motors
debug "W1: ",dec3 p1_wheel, " %: ",dec3 100 * p1_wheel/256
'show what scale speed we are using.

```

The debug command opens up a window so the robot can send information back to your computer. The commands above send back the values of various inputs and the two outputs of the robot.

It is important to comment out the lines that send information back to your computer after you set the trim values. The debug command slows down the whole program slightly and should only be uncommented when you need to watch the values change.

Our final program additions were:

```

'the following 2 lines are the one-stick program, with
modifications
'as explained above.
PWM1 = (p1_y * (p2_wheel/200))
PWM1 = PWM1 + (((2000 + p1_y - p1_x + 127) Min 2000 Max
2254) - 2000)*(1-(p2_wheel/200))
PWM2 = (p2_y * (p2_wheel/200))
PWM2 = PWM2 + (((2000 + p1_y + p1_x - 127) Min 2000 Max
2254) - 2000)*(1-(p2_wheel/200))

'the next 2 lines run the robot at a speed selected with
the p1_wheel
PWM1 = ((PWM1 * p1_wheel)/256)+(128-(p1_wheel/2)) min 0 max
255
PWM2 = ((PWM2 * p1_wheel)/256)+(128-(p1_wheel/2)) min 0 max
255

'uncomment the next four debug lines to set trim on the
sticks
'and to watch the PWM outputs to the motors
debug 0          'clear the debug window
debug "X1: ",dec3 p1_x,      " Y1: ",dec3 p1_y ,cr 'show the
1st joystick's inputs
debug "W2: ",dec3 p2_wheel, " Y2: ",dec3 p2_y ,cr 'show the
2nd joystick's inputs

```

```
debug " L: ",dec3 PWM1,      " R: ",dec3 PWM2 ,cr 'show the
outputs to the left and right motors
debug "W1: ",dec3 p1_wheel, "  %: ",dec3 100 * p1_wheel/256
'show what scale speed we are using.
```

**Load it, debug it and run it...and ALWAYS EXERCISE PRUDENT
DECISION MAKING WHEN USING UNTRIED PROGRAMS!**

Good luck in your programming efforts, and remember...learning to code, like anything else is about DOING IT.... successful programmers do it over and over again.