

Autonomous Programming

Updated: December 16, 2004

Introduction

This handout provides an introduction to the 'art' and 'science' of programming robots to operate in the autonomous mode. In typical competitions of the past, the robot was required to operate without the influence of the human drivers for the first 15 seconds in a two-minute contest. It is anticipated that this will be true again for the upcoming competition.

There are literally hundreds and hundreds of configurations and programming styles to accomplish this 15 second operation. This handout will present what I hope to be several possibilities for the novice programming teams. Options are considered and presented in a simple to more challenging configuration.

Blind autonomous, memorized movements and line tracking will be considered. Use of encoders, inertial instruments for dead reckoning the robots location or IR technology for triangulating will not be considered in this handout.

Basic Robot

Figure 1 illustrates a simple chain drive system on a chassis. The signal pwm01 controls the left side of the robot and pwm02 controls the right side of the robot. It is assumed that a pwm value of 254 will cause all wheels to move in the forward direction and that a value of 0 will result in all wheels turning backwards. A pwm value of 127 will result in no movement.

Three light sensors for detecting tape are labeled s_left, s_middle and s_right.

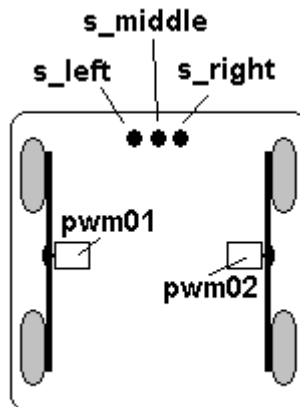


Figure 1 – Simple Robot

Autonomous Code in Default Program

The default autonomous code is listed below in Listing 1. This function User_Autonomous_Code() will be called every 26.2 mSec. Any code written should be placed in the space between the two lines that read GetData(&rxdata) and PutData(&txdata). (BOLD Print)

NOTE: The Operator Interface panel should always be placed into 'Disabled' mode and then the Robot Controller should be reset each time the following auto modes are tested. This results in critical variables being initialized. Don't forget this or you may get some 'whacky' results.

```
void User_Autonomous_Code(void)
{
  while (autonomous_mode) /* DO NOT CHANGE! */
  {
    if (statusflag.NEW_SPI_DATA) /* 26.2ms loop area */
    {
      Getdata(&rxdata); /* DO NOT DELETE, or you will be stuck
here forever! */

      /* Add your own autonomous code here. */

      Putdata(&txdata); /* DO NOT DELETE, or you will get no PWM
outputs! */
    }
  }
}
```

Program Listing 1 – Default Autonomous Mode

The code in Listing 1 is a valid autonomous mode without any additional code. This auto mode does absolutely nothing. Granted, it is not as exciting to watch, but on occasions there times when the best thing to do may be to do absolutely nothing for 15 seconds.

Driving Straight

Remember, any code listed in any further listings should be placed between the GetData() and the PutData() lines in the autonomous function listed in Listing 1.

```
pwm01 = 160;
pwm02 = 160;
```

Program Listing 2 – Driving Straight

During the entire 15 second duration, the code in Listing 2 will drive the robot straight ahead. This assumes all things being equal such as motor characteristics, drive system, etc.

The problem with this mode of operation is that should the robot collide with an obstacle or another robot, the controller will continue to drive the motors for 15 seconds. However, if there are no obstacles, then this robot should keep on driving.

Increasing or Decreasing the Distance Traveled in a Straight Line

If the robot goes too far using the code in Listing 2, then reduce the pwm values from 160 to something smaller such as 150. It will be necessary to experiment with these values.

If the robot does not go far enough then the pwm values can be increased to 170. Again experimentation with different values are key.

The driving surface must also be taken into consideration. Programming and testing on a hard surface will usually produce different results than driving on carpet. It may be necessary to increase the magnitude of the pwm values for operation on a carpet.

More Precise Timing

The auto code function is called by the controller every 26.2 mS. That is 1 / .0262 seconds or 38.17 times each second. For the purposes of this handout, it will be said that the auto function is called 38 times each second. It will be called 19 times in one-half second. With this in mind, it is possible to write the following code to allow the robot to drive straight for 3 seconds and then stop and wait for manual operation to begin. NOTE: 3 seconds x 38 loops/second = 114 loops.

Remember to put this code between the PutData() and GetData(); Also, any other code added from a previous listing in this handout should be deleted so it will not interfere.

```
//place this line before while (autonomous_mode).
static unsigned int t;

//place this after Getdata(&rxdata);
t++;

if (t < 114)
{
    pwm01 = 160;
    pwm02 = 160;
}
else
{
    pwm01 = 127;
    pwm02 = 127;
}
```

Program Listing 3 – Drive Straight for 3 Seconds and Stop

A static variable is created named 't'. Each time the function is called (every 26.2 mS) t is incremented by a value of 1. As long as the value is less than 114 (corresponds to 3 seconds) values of 160 will be sent to both variable speed controllers.

When the value of 't' is 114 or greater, the motors both stop.

Remember, the keyword 'static' is provided so that the function will not forget the previous 't' value. Without the keyword 'static' the robot would drive forward for the entire time in autonomous.

Go Forward, Stop, Go Backward, Stop

Imagine it is necessary to go forward for 3 seconds, stop for 9 seconds and go in reverse for an additional 3 seconds.

First, let's figure out how many loops correspond to these time marks. See Table 1.

Time	Variable 't'
0 seconds	0
3 seconds	114 (3 seconds x 38 loops/second)
12 seconds (3 seconds + 9 seconds)	456 (12 seconds x 38 loops/second)
15 seconds (12 seconds + 3 seconds)	570 (15 seconds x 38 loops/second)

Table 1 – Variable Values

```

//place this line before while (autonomous_mode).
static unsigned int t;

//place this after Getdata(&rxdata);
t++;

if (t < 114)    // between 0 and 3 seconds, go forward
{
    pwm01 = 160;
    pwm02 = 160;
}
else if (t < 456) //between 114 and 456, stop
{
    pwm01 = 127;
    pwm02 = 127;
}
else if (t < 570) //between 456 and 570, go backwards
{
    pwm01 = 94;
    pwm02 = 94;
}
else           //570 or greater, stop
{
    pwm01 = 127;
    pwm02 = 127;
}

```

Program Listing 4 – Drive Straight, Stop and Go Backwards

As long as the value of 't' is less than 114, the motors will drive forward. After the 't' value is 114 to less than 456, the motors will stop. When the 't' value is 456 to less than 570 the motors will drive backwards, After this 15 seconds, the motors will stop.

Now technically, the code should never be executed in the else loop. However, for safety and completeness, I am including it. Also, the if and else if code above could be rewritten to include AND '&&'. Remember, there are lots of ways to do this. I am trying to provide a nice clean layout for newbies.

Executing a Left Turn

Imagine the robot must perform a sharp left turn from a stopped position. Through experimentation, you determine that the left motor will be driven backwards using a pwm value of 95 and that the right motor will be driven forward using a pwm value of 150. Additionally, it is determine that it takes 2.5 seconds to turn exactly 90 degrees. After the robot completes the turn, the robot will stop and wait the remaining time.

First, let's figure out how many loops correspond to these time marks.

Time	Variable 't'
0 seconds	0
2.5 seconds	95 (2.5 seconds x 38 loops/second)
15 seconds (2.5 seconds + 12.5 seconds)	570 (15 seconds x 38 loops/second)

Table 2 – Variable Values

```
//place this line before while (autonomous_mode).
static unsigned int t;

//place this after Getdata(&rxdata);
t++;

if (t < 95) // between 0 and 2.5 seconds, go forward
{
    pwm01 = 95; //left motor goes in reverse
    pwm02 = 150; //right motor goes forward
}
else if (t < 570)//between 95 and 570, stop
{
    pwm01 = 127;
    pwm02 = 127;
}
else // 570 or greater, stop
{
    pwm01 = 127;
    pwm02 = 127;
}
}
```

Program Listing 5 – Turning Left from a Stopped Position

In Listing 5, the ‘else if (t < 570)’ code is not necessary. However it is included here for completeness.

Completing a Complex Manuever

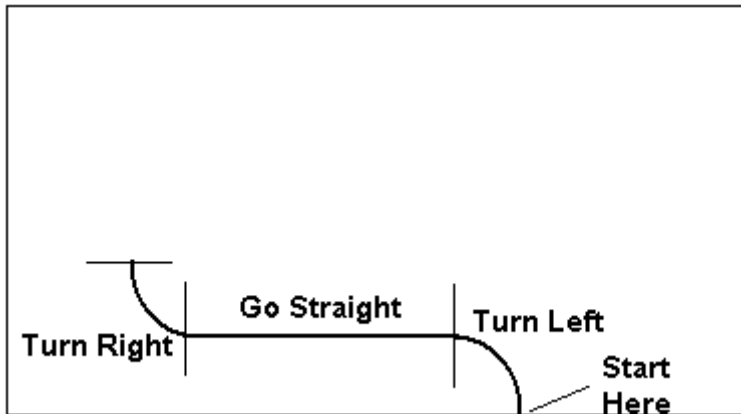


Figure 2 – Drive Path

Refer to Figure 2. This drawing represents a drive path that the robot must execute in autonomous mode. The bold black lines represents reflective tape on the course. However the tape will not be used in this example. It simply indicates the desired robot path.

For this example, the robot will follow the time and pwm values listed in Table 3.

Time (Seconds)	Variable ‘loops’	pwm01 (left motor)	pwm02 (right motor)
0 (turn right)	0	95	150
2.5 (go straight)	95	150	150
7.5 (turn right 2.5 + 5)	285	150	90

10 (stop 7.5 + 2.5)	380	127	127
---------------------	-----	-----	-----

Table 3 – Drive Path Values

Important!!! The following code requires the motors to shift directions immediately. This is generally bad on the system as it will generate all sorts of spikes on the electrical system. It is recommended to slow the motors for a moment and then shift directions.

```
//place this line before while (autonomous_mode).
static unsigned int t;

//place this after Getdata(&rxdata);
t++;

if (t < 95) //turn left for 2.5 seconds
{
    pwm01 = 95; //left motor goes in reverse
    pwm02 = 150; //right motor goes forward
}
else if (t < 285) //go straight for 5 seconds
{
    pwm01 = 150;
    pwm02 = 150;
}
else if (t < 380) //turn right for 2.5 seconds
{
    pwm01 = 150;
    pwm02 = 95;
}
else //stop after 10 seconds
{
    pwm01 = 127;
    pwm02 = 127;
}
}
```

Program Listing 6 – Turning Left, Go Straight, Turn Right and Stop

Line Tracking Problem

For this example, ensure three light sensors are connected to the following digital inputs in Table 4.

s_left	rc_dig_in01
s_middle	rc_dig_in02
s_right	rc_dig_in03

Table 4 – Light Sensor Connection to RC Digital Inputs

Three sensors are mounted next to each other and perpendicular to the line path. The sensors are assumed to start on a straight line and the robot starts by moving forward. (Refer to Figure 1 for sensor placement). It is also assumed that the sensors are close enough that the reflective

tape will not be hidden between sensors. **One sensor must be made at all times. For example: if the tape is 2" wide, then the center of the sensor**

Figure 3 will be used for explaining some basic line following concepts. The robot will start at the position indicated by Position A. In this position, the robot is straddling the line and only the middle sensor is on. The outer sensors are off.

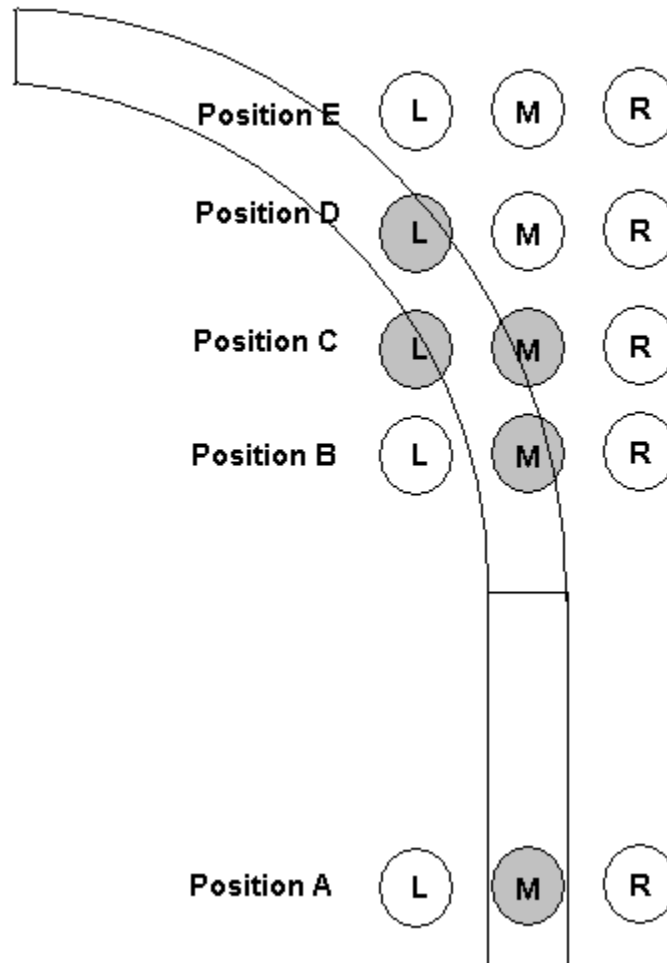


Figure 3 – Line Sensors

Starting at position A, imagine the robot is moving straight up. In positions A and B only the middle sensor is being made. At position C, both the left and middle sensors are being made. The robot continues to move straight to position D in which only the left sensor is made. Eventually, as the robot travels, none of the sensors will be made.

Regarding positions A and B, the robot believes it is still centered over the line. The robot should drive straight.

Position C suggests that the line is starting to move to the left because both sensors (L and M) are on. The response of the robot should be to steer to the right.

Position D suggests that the line has move somewhat significantly to the left because only the L sensor is on and th middle sensor is no longer sensing the line. This generally indicates that the

robot is even further off course. The correction to this condition should be a greater turn to the right.

Position E is really challenging. What should the robot do when no sensors are made? Should the robot back up until a line is detected? Should it drive in circles until a line is detected or should it simply stop to prevent a collision and risking possible damage to the robot.

If the history of the robot's movement along the path is known, it is possible to make a better decision. In the example above in Figure 3, the robot needs to turn to the left quite sharply until a sensor is made.

Program Listing 7 provides code for all positions, left and right side of the line with the exception of position E. This will be covered in subsequent listings.

The following pwm values in Table 5 will be used for each of the desired operations. Again, these values should be adjusted by experimenting with required speeds and floor surface for each robot. NOTE: Reversing motor direction when driving in a forward direction to minimize stress upon the motors and the drive system.

Operation	pwm01 – Left Motor	pwm02 – Right Motor
Go Straight	160	160
Turn Left	127	160
Turn Hard Left	127	180
Turn Right	160	127
Turn Hard Right	180	127
Stop	127	127

Table 5 – PWM Values for Various Operations

No timing variable will be used to count loops. This robot will respond only to sensors. If no sensors are made this robot will stop. '1' means the sensor is made. '0' means the sensor is off.

This code has worked very well for following straight lines. It can be tweaked (changing pwm value) to allow for smoother line following.

```
//place this line before while (autonomous_mode).
unsigned char s_left = rc_dig_in01;
unsigned char s_middle = rc_dig_in02;
unsigned char s_right = rc_dig_in03;

//place this after Getdata(&rxdata);
if (s_left == 0 && s_middle == 1 && s_right == 0) //middle
//sensor on, go straight
{
    pwm01=160; //left motor
    pwm02=160; //right motor
}
else if (s_left == 1 && s_middle == 1 && s_right == 0) // turn
//right
{
    pwm01=160; //left motor
    pwm02=127; //right motor
}
else if (s_left == 1 && s_middle == 0 && s_right == 0) // turn
//hard right
{
    pwm01=180; //left motor
```

```

        pwm02=127; //right motor
    }
    else if (s_left == 0 && s_middle == 1 && s_right == 1) // turn
    //left
    {
        pwm01=127; //left motor
        pwm02=160; //right motor
    }
    else if (s_left == 0 && s_middle == 0 && s_right == 1) // turn
    //hard left
    {
        pwm01=127; //left motor
        pwm02=180; //right motor
    }
    else //stop
    {
        pwm01=127; //left motor
        pwm02=127; //right motor
    }
}
    
```

Program Listing 7 - All Sensor Conditions Except All Off

Note, in actual tests with robots, a value of 30 had to be added to the the values 160 and 180 to yield 190 and 210 in order for this code to work.

Now there are numerous possible outcomes of using the above program in 1000 different robots. Robots will respond differently. Depending upon the motors used, the motors should be balanced, the amount of torque to get started is different then the amount of torque to continue moving. The above code is not guarantee to work as written. The values will most likely need correcting.

Line Tracking – No Sensors are Made

In order to figure out the correct action based upon no sensors being made it important to know what the last sensor condition was before they all went out.

Binary weighted values will be used to store the the entire sensor configuration into a single variable. The left sensor, if on, will be multiplied by 4. The middle snesor, if on, will be multiplied by 2 and the right sensor, if on, will be multiplied by 1.

s_last	s_left	Value	s_middle	Value	s_right	Value
0	0	0	0	0	0	0
1	0	0	0	0	1	1
2	0	0	1	2	0	0
3	0	0	1	2	1	1
4	1	4	0	0	0	0
5	1	4	0	0	1	1
6	1	4	1	2	0	0
7	1	4	1	2	1	1

Table 6 – Binary Weight Values for Sensors

The 's_last' values are from 0 to 7. They are calculated by adding the values in the respective row from each sensor. The code to accomplish this is written as follows.

```
s_last = (s_left * 4) + (s_middle * 2) + s_right;
```

The code below is designed to allow the robot to turn corners. Remember, the pwm values will most likely need changed to make this program work better.

Code typed in **BOLD** below in Listing 8 should be added to the code in Listing 7 as indicated.

```
//place this line before while (autonomous_mode).
static unsigned char s_last;
unsigned char s_left = rc_dig_in01;
unsigned char s_middle = rc_dig_in02;
unsigned char s_right = rc_dig_in03;

//place this after Getdata(&rxdata);
if (s_left == 1 || s_middle == 1 | s_right == 1)
{
    s_last = (s_left * 4) + (s_middle * 2) + s_right;
}

if (s_left == 0 && s_middle == 1 && s_right == 0) //middle
//sensor on, go straight
{
    pwm01=160; //left motor
    pwm02=160; //right motor
}
else if (s_left == 1 && s_middle == 1 && s_right == 0) // turn
//right
{
    pwm01=160; //left motor
    pwm02=127; //right motor
}
else if (s_left == 1 && s_middle == 0 && s_right == 0) // turn
//hard right
{
    pwm01=180; //left motor
    pwm02=127; //right motor
}
else if (s_left == 0 && s_middle == 1 && s_right == 1) // turn
//left
{
    pwm01=127; //left motor
    pwm02=160; //right motor
}
else if (s_left == 0 && s_middle == 0 && s_right == 1) // turn
//hard left
{
    pwm01=127; //left motor
    pwm02=180; //right motor
}

//no sensors made, react based upon last know value
//assume one of the outer sensors were made
else if(s_left == 0 && s_middle == 0 && s_right == 0 && s_last>0)
{
    if (s_last == 4 || s_last == 6) {
        pwm01=127; //left motor
        pwm02=190; //right motor
    }
}
```

```

    }
    else if (s_last == 1 || s_last == 3) {
        pwm01=190; //left motor
        pwm02=127; //right motor
    }
}
else //stop
{
    pwm01=127; //left motor
    pwm02=127; //right motor
}

```

Program Listing 8 - All Sensor Conditions

Autonomous Mode Using Predefined Values at 1 Second Intervals

First, add one line of code (**BOLD**) to the following in user_routines_fast.c.

```

void User_Autonomous_Code(void)
{
    while (autonomous_mode) /* DO NOT CHANGE! */
    {
        if (statusflag.NEW_SPI_DATA) /* 26.2ms loop area */
        {
            Getdata(&rxdata); /* DO NOT DELETE, or you will be stuck
here forever! */

            /* Add your own autonomous code here. */
            AutoDrive();
            Putdata(&txdata); /* DO NOT DELETE, or you will get no PWM
outputs! */
        }
    }
}

```

Second, create a file named "auto.h" and add it to the project. Add this code to the "auto.h" file.

```

#include "ifi_aliases.h"
#include "ifi_default.h"
#include "ifi_utilities.h"

void AutoDrive (void);

```

Third, create another file named "auto.c" and add it to the project. Add this code to the "auto.c" file.

```

#include "auto.h"

unsigned char pl[] = {127,150,150,150,150,150,150,150,150,127};

void AutoDrive (void)
{
    static unsigned char i;
    static unsigned char k;
    unsigned char t = 38;
    if (i < 10)
    {

```

```
    k++;  
    if(k>t)  
    {  
        pwm01 = p1[i];  
        k = 0;  
        i++;  
    }  
}  
else  
    pwm01 = 127;  
}
```

Program Listing 9 – Predefined Values

Explanation. The array 'p1' stores 10 values representing desired 'pwm01' commands. The function AutoDrive is called every 26.2 mSec. The variables 'i and k' are remembered because of the keyword 'static'. Every 38 times, 'k is > then t'. The 'pwm01' command is set to the value in the array at position 'i'. This repeats until the array is exhausted. At this time the pwm01 is set to 127. Modify the value of 't' to see the results.

The value of pwm01 will be updated every ($t \times 26.2$ mS). In this case $t = 38$. Time is 38×26.2 mSec or 0.99 seconds (approximately 1 second).

Conclusion

This handout hopefully has given you some ideas and possibly some help. Good luck!!