by Guido Ottaviani

Robot Navigation and Control (Part 2)

Software Development

Guido built a navigation and control subsystem for an autonomous differential steering explorer robot. Here he describes the software development phase of the project.

In the first part of this article series, I described how to build a robotic platform with Microchip Technology dsPIC controllers. Now I will describe the software loaded on the board that manages wheel speed, closed-loop control with a PID algorithm, dead-reckoning by odometry (in both theoretical and practical forms), field mapping, navigation, motor control (MC), and more. The software is modular, so all the pieces can be examined as stand-alone black boxes. I'll focus on the Microchip dsPIC30F board so you can better understand every block. You'll find the detailed comments in the code to be extremely helpful.

FIRMWARE

The philosophies of MC and supervisor programs are similar. Both involve the recycling of numerous portions of the code. The programs are described step by step in the code. The name of the MC's DSC program is dsPID. The program in the supervisor is dsODO.

The source code, MPLAB project, and detailed flowcharts are posted on the *Circuit Cellar* FTP site. Both programs (dsPID and dsODO) are fully interrupt-driven. At start-up, after the initialization of the supervisor and MCs, the programs enter a simple main loop, acting as a state machine. In the main loop, the program checks flags enabled by external events and enters in the relative state (see Figure 1). Because it's a kind of simple cooperative real-time operative system (RTOS), each routine has to be executed in the shortest possible amount of time to free up the system to handle frequent tasks. There are no delays in the code. Interrupts are used whenever possible, particularly for slow operations like the transmission or reception of strings of characters.

MCs use the C30's PID library to control the speed and position of the wheels. The feedback from the encoders on the motors' axes

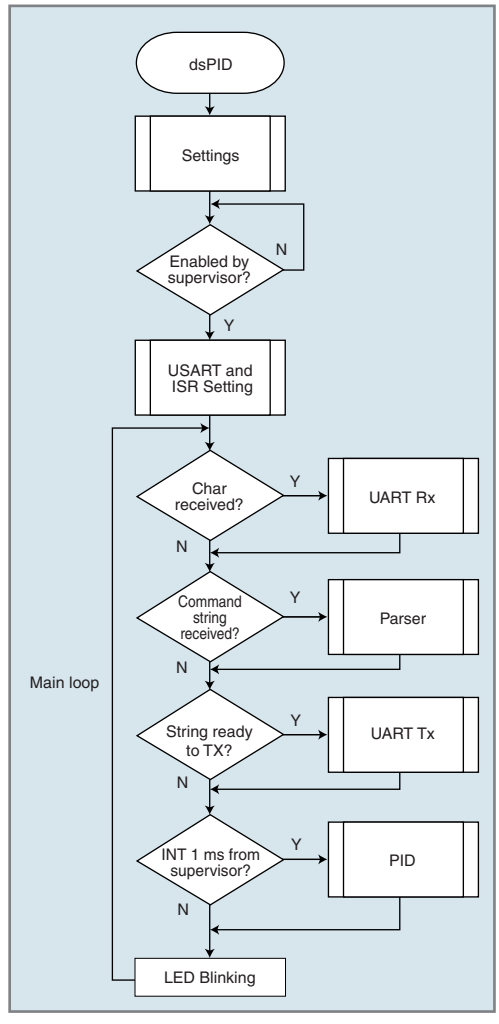


Figure 1—A dsPID main loop is simple because most of the program is interrupt-driven.

Usage	Pin name	Pin number		Pin name	Usage
	MCLR	1	15	INT1	TX Enable
ADC Reference	VREF+	2	16	INT0	Timer 1 ms from supervisor
Motor current reading	AN1	3	17	PGD/EMUD	
		4	18	PGC/EMUC	
Chip select from supervisor	CN5	5	19	VSS	
Quadrature encoder	QEA	6	20	VDD	
Quadrature encoder	QEB	7	21		
	VSS	8	22	RE4	LED 2
	OSC1	9	23	RE3	LED 1
		10	24	RE2	H-bridge Enable
Serial TX	U1ATX	11	25	PWM1H	PWM
Serial RX	U1ARX	12	26	PWM1L	PWM
	VDD	13	27	AVSS	
Velocity measurement	IC2	14	28	AVDD	

Table 1—These are the pins used on the Microchip Technology dsPIC30F4012.

enables this (see Table 1). Peripherals on the MCs include QEI to calculate the covered space, input capture (IC2) to calculate speed, an ADC to read motor current, enhanced PWM to drive the motors, and a UART to communicate with the supervisor.

dsPID

The same program (dsPID) is loaded in both of the MCs, and the supervisor assigns them a different ID at initialization (to address each one later). Speed and position measurements are executed simultaneously by both MCs when an external interrupt occurs from the general timing signal provided by the supervisor.

A QEI module determines the wheels' distance and direction. This value is algebraically cumulated in a variable every 1 ms and sent to the supervisor at its request. After the value is sent, the variable is reset.

Speed is measured at every encoder's pulse. Every 1 ms, it calculates the mean speed by averaging samples, executes a PID algorithm, and corrects the motor speed according to its result, changing the PWM duty cycle (see Photo 1). For a detailed description of the C30 PID library application, refer to the following Microchip code example: "CE019: Proportional Integral Derivative (PID) Controllers & Closed-Loop Control."[1] A link is provided in the References section at the end of this article.

Speed variations of the motors are executed smoothly, accelerating or decelerating with a rising or falling slanted ramp to avoid heavy mechanical strain and wheel slippage that could cause errors in odometry. Deceleration is faster than acceleration to avoid bumps with obstacles during braking (see Photo 2).

IC2, input capture, is used to measure the time elapsed between two pulses generated by the encoder (i.e., when the wheel moves a fixed distance). Connected in parallel to QEA, it captures elapsed time on the rising edge of the encoder's signal. TIMER2 is used in free-running mode. At each IC2 interrupt, TMR2's current value is stored and its previous value is subtracted from it. This is the pulse period. The current value then becomes the previous value, awaiting the next interrupt. TMR2's flag must be checked to determine if an overflow occurred in the 16-bit register. If one occurred, the difference between 0xFFFF and the previous sample has to be added to the current value. Samples are algebraically added in the `IcPeriod` variable. The `_UPDN` bit of the QEI register is set or reset if the wheel is rotating forward or backward. The value of each sample is algebraically cumulated, so it's added if the bit is set, or subtracted if reset, to measure the actual space covered. This is one of the suggested methods in Microchip's application note AN545.[2]

The ADC continuously measures motor current, storing values in its 16-position ADCBUF buffer. When the buffer is full, an interrupt occurs and a mean value is calculated. This happens approximately every 1 ms.

The UART receives commands from the supervisor and sends it the results of the measurements. The communication portion of the program runs as a state machine. Status variables are used to execute actions in sequence. Simple and fast interrupt service routines (ISRs) get or put every single byte from or to a buffer and set the right flags to let the proper function be executed.

TX I/O is disabled at initialization. If an I/O pin is set as an input pin, it enters into a "three-state" mode, meaning a high-impedance mode, which enables you to use parallel pins. This is the default configuration. This setup enables you to connect both MCs' TX ports together. They will be enabled one at a time by the supervisor with INT1.

The same program is in both MCs. Each MC is identified by an ID code to enable the supervisor to send commands to the proper motor. At start-up, the program loops before the "main" idle loop, waiting for a supervisor's enable signal through CN5 I/O port. After that, the correct ID is assigned. The start-up ID is 9 for both MCs.

Photo 1—This test set verifies H-bridge and PID parameters. The motor under test is mechanically joined with a similar motor. This one is loaded on a power variable resistor to easily simulate a variation in mechanical load for the first motor.

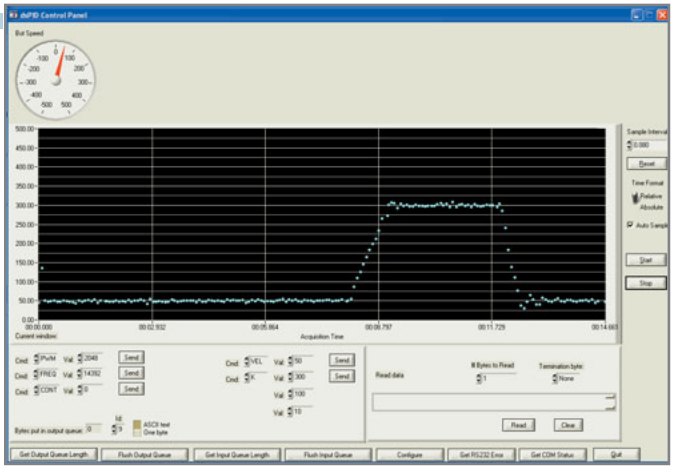


Photo 2—This is one of the first tests during the calibration of PID parameters. It shows the measured speed after a remote request to switch from 50 to 300 cm/s and back to 50 cm/s. Note the rising ramp with less slope than the falling one.

The supervisor will assign the definitive ID, subsequently enabling each MC. In normal operation, both MCs simultaneously receive data sent by the supervisor, but only the addressed one (with the correct ID) decodes the message. A message with ID = 0 (broadcast) is decoded by both MCs. If an error occurs during reception (i.e., UART, checksum, parsing errors), the status variable is set to a negative number and the red LED illuminates to indicate the fault condition.

The supervisor drives both MCs through the UART1 communication port, sending commands and reading information (space, speed, and motor current). It estimates the robot's position using that information (dead-reckoning by odometry) and creates a map of the path, obstacles, and so on. This is done with the help of the dsPIC30F's trigonometric capabilities programmed with the C30 compiler.

The peripherals used on the supervisor include UART1 to communicate with the MCs, UART2 for telemetry with the remote PC, I²C to communicate with the main board, and OC simple PWM to generate the clock for both MCs (see Table 2).

dsODO

The peripherals UART1 through UART2 are used to communicate with the MCs and for telemetry with a remote PC, respectively. They are used the same way as the MCs: similar ISRs, similar functions. The protocol used for the handshake is also the same. The physical-layer-independent protocol is used with the I²C bus, as well as to communicate with the main board.

The dsPIC peripheral interface controls the first layer. Frame, or overrun errors (UART), or collisions (I²C) are detected by hardware, setting the appropriate flag. ISR routines handle the second layer. They fill the RX buffer with the bytes received from the interfaces. They also detect buffer overflow and command overrun. UartRx or UartRx2 functions manage the third layer. These routines act as a

Usage	Pin name	Pin number		Pin name	Usage
	MCLR	1	15	EMUC2	
		2	16		
Generic chip select 1	RB1	3	17	PGD/SCL	PGD also I²C clk
TX enable 1	RB2	4	18	PGC/SDA	PGC also I²C dat
Generic chip select 2	RB3	5	19	VSS	
TX enable 2	RB4	6	20	VDD	
1-ms Heartbeat	RB5	7	21	U2TX	Serial 2 TX
	VSS	8	22	U2RX	Serial 2 RX
	OSC1	9	23	RB9	LED 1
	OSC2	10	24	OC1	Clock out for Motor controllers
Serial 1 TX	U1ATX	11	25	EMUD2	EMUD2
Serial 1 RX	U1ARX	12	26	RB6	LED 2
	VDD	13	27	AVSS	
		14	28	AVDD	

Table 2—These are the pins used on the Microchip dsPIC30F3013.

state machine, getting bytes from the buffer and decoding the command string (see Table 3).

This layer controls timeout and checksum errors, as well as packet consistency (correct header, correct length). If everything is fine, it allows the Parser routine (fourth layer) to decode the message and to execute the required action. This routine sets the appropriate error flag if the message code received is unknown.

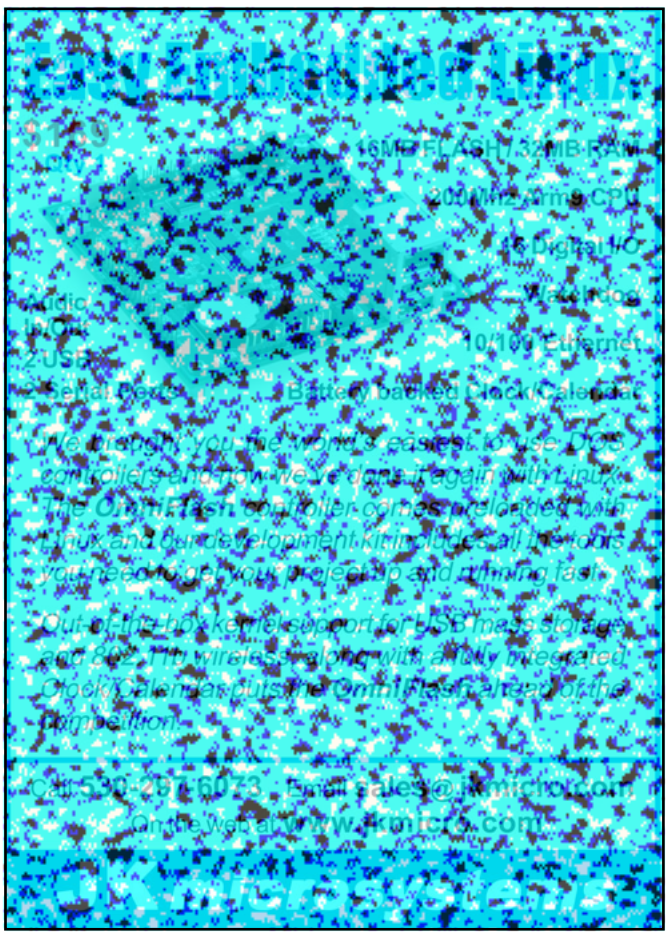
TMR1 generates a 1,000-Hz timing clock (the program's heartbeat). On each TMR1's interrupt, internal timers are updated, the watchdog is cleared, and a flag is set to enable the function that requests the MC's distance. Every 10 ms, an "All_Parameters_Ask" function (speed, position, and current) is enabled. The same clock is used, through a pulse on RB5, to synchronize MCs for PID and position elaboration.

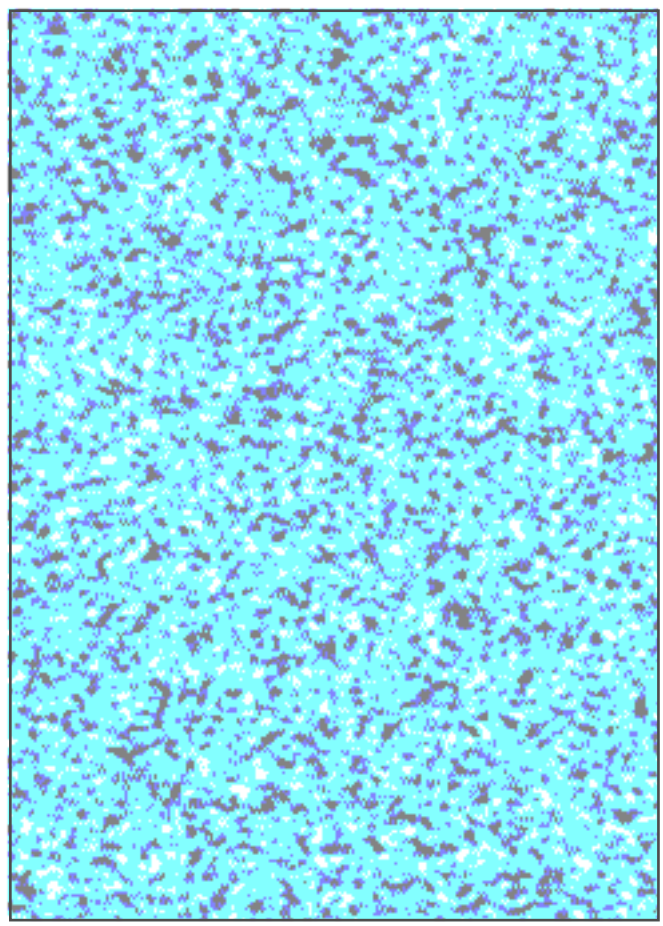
PWM (output Compare 1) is used to obtain the oscillator frequency for the MCs. The OC simple PWM I/O peripheral is set to have a PWM at 50% duty cycle with a 7.3728-MHz frequency (the same as the supervisor crystal):

```
PWM_period=(PRx+1)•4•TOSC•(TMRx_prescale_value)
```

With `Prx` = 3, `prescale` = 1 7.3728 MHz is obtained again at output. With this output, both MCs can be driven in EC 16xPLL mode. This way, all three DSCs have exactly the same clock and some components are saved on the board.

With data coming from the MCs, the supervisor performs field-mapping. For more information about the topic of dead-reckoning by odometry, refer to the following works: "Where Am I?: Sensors And Methods For Mobile Robot Positioning," by Johann Borenstein[3]; "Implementing Dead Reckoning by Odometry on a Robot with R/C Servo Differential Drive," by Dafydd Walters[4]; and "A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators," by G. W. Lucas.[5] Simplified algorithms are also in these





Sequence	Name	Range	Note
1	Header	@	
2	ID	0–9	ASCII
3	Cmd	A–Z	ASCII
4	CmdLen	1-MAX_RX_BUFF	Number of bytes following (checksum included)
5	Data	...	
..		...	
n	Checksum	0–255	Obtained by simply adding up in an 8-bit variable, all bytes composing the message (checksum itself excluded).

Table 3—This is the structure for the command packets. Each one contains all the bytes shown.

documents. You can find the correct compromise between precision and computing speed by using the trigonometric capability of the dsPIC30F series.

Every few milliseconds, after the current position elaboration, field mapping divides the unknown field in a 10 cm × 10 cm cell grid. Defining a maximum field dimension of 5 m × 5 m, you obtain a 2,500-cell matrix (50 × 50). Each cell is defined with a nibble, with a total memory occupation of 1,250 bytes. Sixteen different values can be assigned to each cell (e.g., n = 00 unknown cell, n = 01 – 10 cell visited n times, n = 11 obstacle found, n = 12 target of type A found, n = 13 target of type B found, and n = 14 target of type C found).

Listing 1—To create a 50 × 50 nibble matrix, you need to define a `struct`.

```
typedef struct
{
        unsigned char Low :4;
        unsigned char High :4;
}_Coordinate;

_Coordinate MapXY [25][50];
```

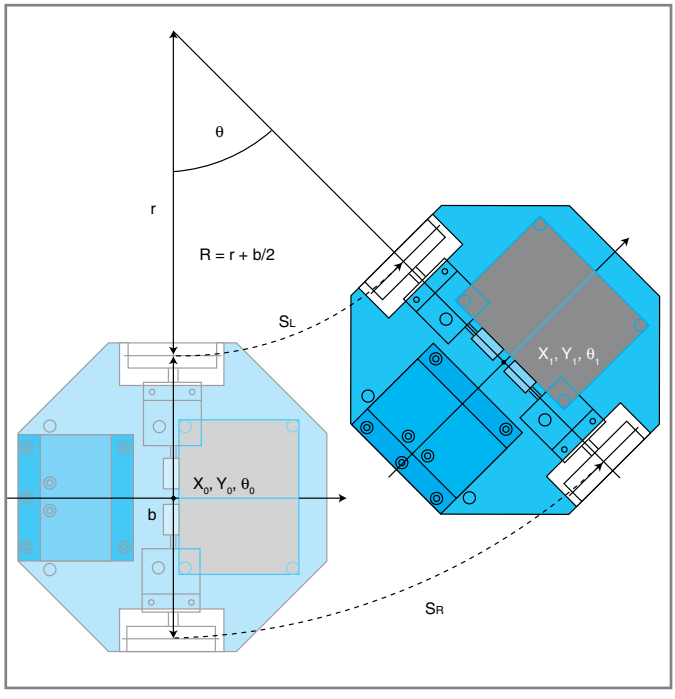


Figure 2—This is a definition of the terms used in the formulas for a turning platform.

The robot can start from any position in the field. Note that (0, 0) is the reference coordinate in its reference system. To translate robot reference system coordinates to a 50 × 50 matrix index pair, the values must be "normalized" in a 0 to 49 range: Xnorm = (Xrel + 50) mod 50 and Ynorm = (Yrel + 50) mod 50. Index is the remainder of division in a range of 0 to 49. A range check must be performed in advance to avoid overflow if the field is greater than 5 m × 5 m.

To create a 50 × 50 nibble matrix, you need to define a `struct` (see Listing 1). It fills up 1,250 bytes. Eliminating heap space (not needed if dynamic memory allocation or file I/O library functions are not used) leaves enough RAM to work with.

Field-mapping is useful for finding the best exploring strategy in an unknown field. The robot can direct itself to the less explored portion of the field (lower "n" value); it can save time by not stopping twice in an already discovered target; and it can find the best path to reach a given coordinate, and more.

DEAD RECKONING BY ODOMETRY

Let's consider the general dead-reckoning algorithm needed for a DSC- or microcontroller-based system. Once you have the information about the distance traveled by each wheel in a discrete time update (odometry), you can estimate the robot's position coordinates with the same periodicity without any external reference (dead reckoning). Refer to G.W. Lucas's aforementioned paper for information about the mathematics.[5] In the following equations, I used Lucas's symbols and terms:

$$\vartheta(t) = \frac{(v_R - v_L)t}{b} + \vartheta_0$$

$$x(t) = x_0 + \frac{b(v_R + v_L)}{2(v_R - v_L)}\left[\sin\left(\frac{(v_R - v_L)t}{b} + \vartheta_0\right) - \sin(\vartheta_0)\right]$$

$$y(t) = y_0 - \frac{b(v_R + v_L)}{2(v_R - v_L)}\left[\cos\left(\frac{(v_R - v_L)t}{b} + \vartheta_0\right) - \cos(\vartheta_0)\right]$$

Figure 2 shows the terms used in the formulas for a turning platform.

For each discrete time interval, the system measures the number of pulses generated by the encoders. Knowing the distance represented by a single encoder tick, you can calculate the distance traveled by the wheels (S_R, S_L) in time t. Note that velocity = distance/time:

$$v_R = \frac{S_R}{t}$$

$$v_L = \frac{S_L}{t}$$

According to Lucas:

$$R = \frac{b(v_R + v_L)}{2(v_R - v_L)} = \frac{b(S_R + S_L)}{2(S_R - S_L)}$$

You can calculate:

$$x(t) = x_0 + R\left[\sin\left(\frac{(S_R - S_L)}{b} + \vartheta_0\right) - \sin(\vartheta_0)\right]$$

$$y(t) = y_0 - R\left[\cos\left(\frac{(S_R - S_L)}{b} + \vartheta_0\right) - \cos(\vartheta_0)\right]$$

Note that at time t_i, the differences with the coordinates at t_{i-1} are:

$$\Delta\vartheta = \frac{(S_R - S_L)}{b}$$

$$\Delta x = R\left[\sin(\Delta\vartheta + \vartheta_{i-1}) - \sin(\vartheta_{i-1})\right]$$

$$\Delta y = R\left[\cos(\vartheta_{i-1}) - \cos(\Delta\vartheta + \vartheta_{i-1})\right]$$

By performing a summation of each delta x in x variable and each delta y in y variable, you know the current coordinates (position and orientation) of the platform.

To avoid computational errors (divide by zero) and wasted

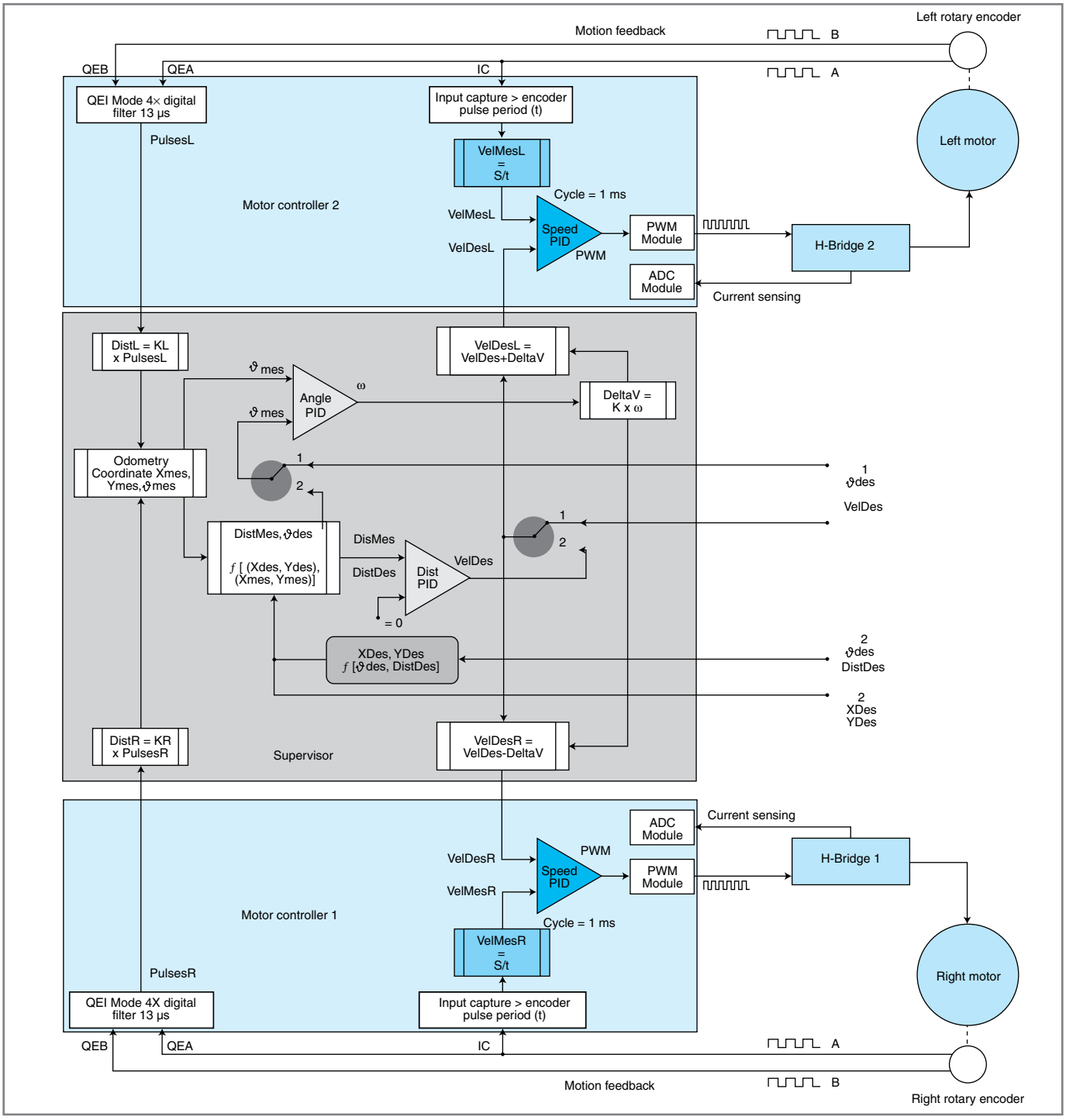


Figure 3—These software logical blocks govern the robot's navigation.

controller time, both the S_R and S_L variables must be checked in advance. Defining a quasi-zero value S_{min}, which takes care of minimal mechanical and computational approximations, you get the following. If $|S_R - S_L| < S_{min}$, the platform is traveling in a nearly straight line and you can use the method from Lucas's paper without approximations:[5]

$$\Delta S = \frac{(S_R + S_L)}{2} \cong S_R \cong S_L$$
$$\Delta\vartheta = \frac{(S_R - S_L)}{b} = 0$$
$$\Delta x = \Delta S \cos(\vartheta_{i-1})$$
$$\Delta y = \Delta S \sin(\vartheta_{i-1})$$

If $|S_R + S_L| < S_{min}$, the platform is pivoting around its own vertical axis without moving. Thus:

$$\Delta\vartheta = \frac{(S_R - S_L)}{b}$$
$$\Delta x = \Delta y = 0$$

SOFTWARE ARCHITECTURE

Figure 3 shows the overall software architecture for the dsNavCon board's control procedures and navigation strategies. The most important logical blocks are the four PID controls. They are shown in a three-level nested control loop. Starting from the top level, the Distance PID controls the robot's mean speed toward the target every 50 ms. The Angle PID corrects the orientation to point the target every 10 ms by adding or subtracting a DeltaV to the mean speed to make the vehicle spin around its vertical axis. By combining the outputs of the Angle and Distance PIDs, you can determine the setpoint for the most internal level, the Speed PIDs. Each of the PIDs controls the speed of its wheel every 1 ms to maintain the value set by outer loops (see Figure 3). By combining the output of Angle and Distance PIDs, you can obtain the setpoint for the Speed PIDs (see Figure 3). The three levels are nested. But, fortunately, the different PIDs (speed, orientation, and distance) are independent of each other, simplifying the K parameter's calibration procedure. They can be set one at a time starting from the bottom.

The motor controllers appear as dark boxes that take care of the wheels' speed. The supervisor sends them the reference speed (VelDesX: desired velocity) and the input capture modules of the microcontrollers get pulses from the encoders connected to the motor axis and derive the rotational speed of the motors (VelMesX: measured velocity). By combining the values in the PID control Speed PID every 1 ms, you can obtain the necessary PWM value in that condition to keep the desired speed of each wheel. In PID terminology, VelDesX is usually called the setpoint or control reference. VelMesX is the measured output or process variable. PWM is the control output, manipulated variable, or simply output.[6]

The Quadrature Encoder Interface (QEI) modules get both the A and B pulses from the encoders. They receive the traveling direction and the number of pulses in 4× mode (counting the rising and falling edges of signal A and signal B: 2 × 2 = 4) to the supervisor.

Multiplying the number of pulses by K—which indicates the distance traveled for each encoder pulse—you can determine the distances traveled by right and left wheels every 10 ms. The supervisor combines this traveling information and applies the dead-reckoning procedure to determine the robot's position coordinates: Xmes, Ymes, and θMes (orientation angle).

The supervisor receives an external navigation command via the serial interface (telemetry) or via the I²C interface (main board). Different strategies can be applied: A—Free running is movement at a given speed in a given direction (VelDes, θDes). B—Cartesian is movement toward a given coordinate (XDes, YDes). C—Polar is movement for a given distance in a set direction (DistDes, θDes).

In mode A with the logical control switches in position 1, only the PID control (Angle PID) is used on the supervisor (see Figure 3). This combines the desired angle θDes with the measured angle θMes computed by the odometry procedure—to obtain the value of the rotation angular speed ω of the vehicle around its vertical axis—needed to correct the orientation error.

The DeltaV value is proportional to ω. It's added to VelDes to obtain the left wheel's speed and subtracted from

VelDes—to obtain the right wheel's speed—in order to keep the heading corresponding to the θDes value, while the center of the robot is still moving at the VelDes speed.

In mode B, with the logical control switches in position 2, the desired speed VelDes is calculated by the PID control Dist PID, and it is used as in mode A. This means the mean speed decreases proportionally to the distance from the target. It becomes zero when the target is reached. The measured input for this PID (DistMes) is computed as a function of the current coordinates and the destination coordinates. The desired orientation angle θDes also comes from the same procedure and it's used as reference input for Angle PID. The reference input for Dist PID is 0, meaning that the destination is reached. With ω and VelDes available, the wheels' speed control runs as it does in the first mode.

In mode C, with the logical control switches in position 2, the destination coordinates (Xdes, Ydes) are computed once at the beginning as a function of input parameters (DistDes, θDes). After that, everything operates as it does in mode B.

A sequencer is also available to perform some specific paths for UMBmark (or something like the RTC competition I mentioned in the first part of this article series).[7] Like a washing machine timer, it schedules the robot's behavior by executing a series of primitives. The sequence is written in some arrays, and it is synchronized by external events. Some higher-priority events (e.g., obstacles found by external sensors) can override scheduling.

TIME TO GO ROBO

There are plenty of affordable robots on the market. Plus, the MPLAB development environment is free. You can design the schematic and PCB with the freeware version of CadSoft Computer's Eagle. These tools are versatile enough for a wide variety of applications. Affordable electronic and mechanical components are also widely available on the Internet. Do some reseach before shelling out a lot of cash for an expensive kit.

No more excuses. You are now ready to design, build, program, and test your own robot.

Guido Ottaviani (guido@guiott.com) has worked with electronics and ham radios for years. After working as an analog and digital developer for an Italian communications company for several years, Guido became a system integrator and then a technical manager for a company that develops and manages graphic, prepress, and press systems and technologies for a large Italian sports newspaper and magazine publisher. A few years ago, he dusted off his scope and soldering iron and started making autonomous robots. Guido is currently an active member in a few Italian robotics groups, where he shares his experiences with other electronics addicts and evangelizes amateur robotics.

PROJECT FILES

To download code, go to ftp://ftp.circuitcellar.com/pub/Circuit_Cellar/2009/225.

REFERENCES

[1] Microchip Technology, Inc., Microchip code examples, "CE019: Proportional Integral Derivative (PID) Controllers & Closed-Loop Control," 2005, www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2620.

[2] M. Palmer, "AN545: Using the Capture Module," DS00545D, Microchip Technology, Inc., 1997.

[3] J. Borenstein, H. R. Everett, and L. Feng, "Where Am I?: Sensors and Methods for Mobile Robot Positioning," University of Michigan, 1996, www-personal.umich.edu/~johannb/position.htm.

[4] D. Walters, "Implementing Dead Reckoning by Odometry on a Robot with R/C Servo Differential Drive," *Encoder*, 2000, www.seattlerobotics.org/encoder/200010/dead_reckoning_article.html.

[5] G. W. Lucas, "A Tutorial and Elementary Trajectory Model for the Differential Steering System of Robot Wheel Actuators," SourceForge, 2001, http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html.

[6] Wikipedia, "PID Controller," http://en.wikipedia.org/wiki/PID_controller.

[7] J. Borenstein and L. Feng, "UMBmark: A Method for Measuring, Comparing, and Correcting Odometry Errors in Mobile Robots," 1994, www-personal.umich.edu/~johannb/umbmark.htm.

RESOURCES

G. Ottaviani, www.guiott.com/Rino/index.html.

Roboteck Discussion Group, http://it.groups.yahoo.com/group/roboteck/ (Italian) or http://groups.yahoo.com/group/roboteck_int/ (English)

Robot Italy, www.robot-italy.com

SOURCES

Eagle Software
CadSoft Computer, Inc. | www.cadsoftusa.com

dsPIC30F3013 Digital signal controller, dsPIC30F4012 motor controller, and dsPIC33FJ64MC802 microcontroller
Microchip Technology, Inc. | www.microchip.com