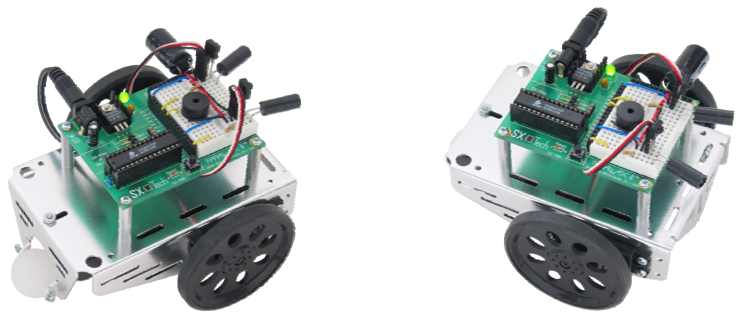
4.16 Robotics - Controlling the Parallax SX Tech Bot

4.16.1 Introduction

The Parallax SX Tech Bot shown below is a small battery-powered autonomous robot with two drive wheels in front and a 1″ polyethylene ball at its tail. This design is based on the popular Parallax Boe-Bot™ robot, which uses the BASIC Stamp® 2 module for its programmable controller on the Board of Education® prototyping platform. The SX Tech Bot uses the SX microcontroller for its programmable brain, and the SX Tech board for its prototyping platform. If you are interested in experimenting with this robot, it consists of two parts kits, the SX Tech Tool Kit (Part #45180) and the Boe-Bot Parts Kit (Part #28124).



The SX Tech board comes with all the components to run an SX-28 controller, which is inserted into the board’s LIF (low insertion force) socket. The board has a 5V voltage regulator, header sockets for all SX I/O pins plus some other signals, and a small breadboard prototyping area. We will use this prototyping area to build and test sensors and indicators for the SX Tech Bot.

The chassis also has a battery holder installed for four 1.5 V AA batteries, so you can run the robot without an external power source. Simply plug the power connector leading from the battery pack into the SX-Tech Board's 6-9 VDC power jack. Nevertheless, while doing the first tests, it might be a good idea to use an external power supply. A DC supply rated for an output of 7.5 V, 1000 mA with a center-positive, 2.1 mm plug is recommended. NOTE: The supply's output rating can be from 6 to 9 VDC with a capacity of 600 mA or more.

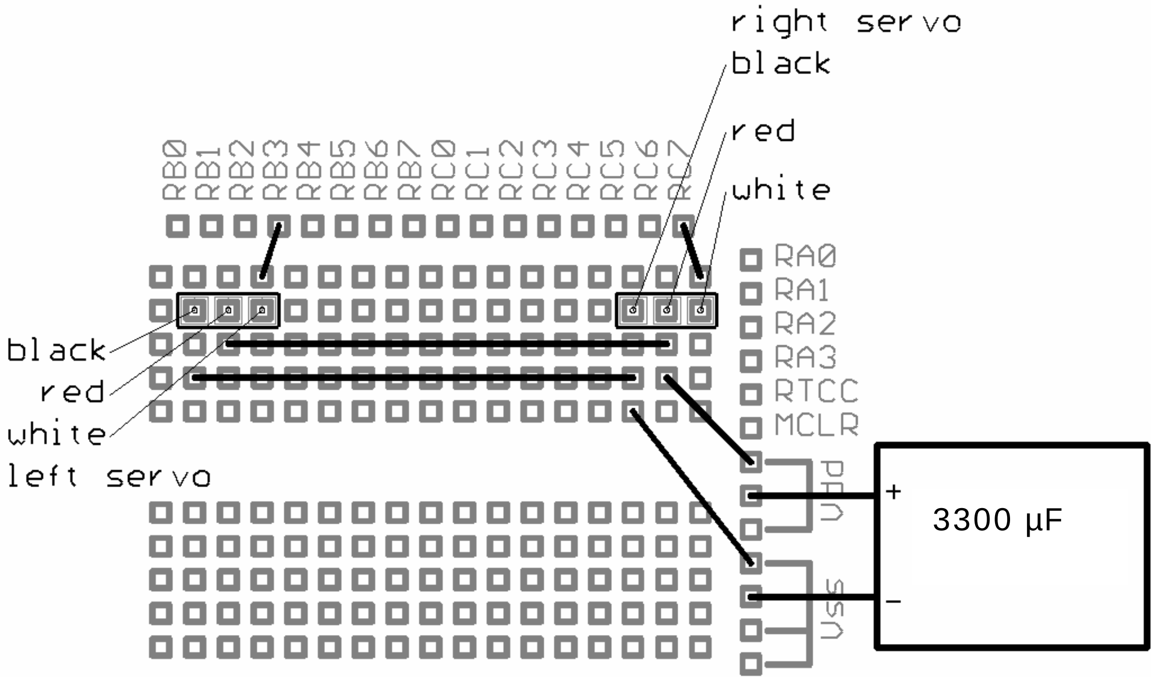
The SX Tech Bot's two front wheels are driven by two premodified modified RC hobby servos, called Parallax Continuous Rotation servos. Both Standard and Continuous Rotation servos have an output shaft which is controlled by the circuitry inside the servo. A Standard servo is designed to rotate its output shaft to particular position and hold that position. The position is dictated by the control signal it receives. In contrast, a Parallax Continuous Rotation servo makes its output shaft rotate at a particular speed in a particular direction. The speed and direction is dictated by the same type of signal that is used to control the standard servo's position.

All previous code examples in this book were designed for SX controllers clocked at 50 MHz. The sample code in this chapter assumes an SX clocked at 4 MHz using an external, 4 MHz resonator. Since the SX Tech Bot relies on AA batteries for its power, the 4 MHz clock rate is a better choice since the SX draws significantly less current at lower clock rates. The drawback of lower clock rates, of course, is that the resolution of the incremental timing changes that can be made to output signals and sampling rates is much lower.

> A precise external clock, such as the 4 MHz (accurate to +/- 0.3 %) ceramic resonator included in the SX-Tech Toolkit, is recommended for autonomous SX Tech Bot applications. In contrast, the SX microcontroller's internal oscillator is not recommended for these applications. Although it can supply a clock rate of 4 MHz, the IRC calibration can only guaranty an error within +/- 8 % at a given temperature. Additional programming techniques can be used to reduce this variation to around 1 %. Even so, this variation will still be noticeable if you are attempting to recalibrate the servos without the aid of the SX-Key, and the differences will be accentuated by changes in temperature.

Before assembling your SX Tech Bot, you will probably need to perform some tests and mechanical adjustments on the servos. The servos are connected to the SX Tech Board and a test and adjustment program is run. After the mechanical adjustments and potentially some software adjustments are made, you can then assemble your SX Tech Bot without having to worry about having to disassemble it again. Follow the instructions through Section 4.16.3.1 first. After that, you can then move on to the mechanical assembly instructions available from the www.parallax.com web site.

The drawing, below, shows how the two servos can be connected to power, ground, and SX I/O pins for control signals. Place two three-pin headers into the second row of the breadboarding area as shown, and also place the jumper wires as shown, to connect the servo inputs to the SX port pins RB3, and RC7, and to the power supply lines Vdd (+5V) and Vss (Ground). Use only insulated jumper wires, and as always, only make changes to circuits when power is disconnected. Also, make sure to correctly follow the color coding indications in the figure when connecting your servos to the three-pin headers.



As the servos consume quite an amount of starting current, it is important to connect an electrolytic capacitor (3300 µF, 6 V or higher) across Vdd and Vss, to avoid that the SX resets due to supply-voltage drops.

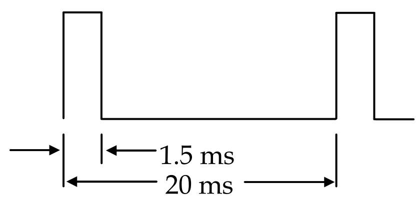
> WARNING: When connected properly, these capacitors store the additional charge required by the servo motors during starts and sudden direction chances. However, when connected incorrectly, in reverse polarity, these capacitors can rupture or even explode. So, follow these connection instructions carefully. The capacitor's positive lead is denoted by a longer lead, and the negative lead is denoted by a stripe on the metal canister with negative signs. Make sure to verify that capacitor’s positive lead is connected to Vdd, and that the negative lead is connected to Vss before connecting power to the system..

Finally, plug the servo connectors on to the two three-pin headers, and make sure that the orientation of the connectors is correct, i.e. that the color order of the wires follows the one shown in the drawing. The white wire (input) from the left servo should be connected to RC7, and the one from the right servo should go to RB3.

In the next section, we will address how the servos are controlled and discuss some general concepts for a basic SX Tech Bot application. We will then discuss an introductory SX program that takes care of the functions, necessary to control the SX Tech Bot, and we'll use this code as a "skeleton" for more examples in following sections.

4.16.2 Controlling the SX Tech Bot Servos

The servos that are used as the SX Tech Bot's "motors" are quite similar to the servos commonly used for RC models. In an RC model, such servos – for example – are used to move a rudder to any position between two left and right end positions. The servos have an input where they expect a PWM signal with a certain high time to control the servo position. Here is the typical timing diagram for a servo signal that instructs the servo to hold it's "center" position:



As you can see, the time between two pulses is 20 ms. This duration is not critical for servo control, and any time between 5 and 45 ms will suffice. The pulse width is the signal that must be precise for accurate servo control.

A pulse width of 1.5 ms means that a Standard servo moves to, and holds at its center (or zero) position. This is the mid-point position in a servo's range of motion. When the pulse width becomes larger than 1.5 ms, the servo's output shaft rotates to a position counterclockwise of center, and vice versa, when the pulse width is less than 1.5 ms, the servo's output shaft rotates to a position clockwise of the center.

Instead of holding a particular position, a Parallax Continuous Rotation servo responds by rotating its output shaft counterclockwise when it receives pulses that last longer than 1.5 ms. Likewise, its output shaft rotates clockwise when it receives pulses less than 1.5 ms. The speed of rotation depends on the difference between the current pulse-width, and the 1.5 ms "stop value". The greater the difference, the faster the rotation speed will be. At pulse-widths of 1.7 ms, or 1.3 ms, the Parallax Continuous Rotation servos rotate at their maximum counterclockwise or clockwise speeds. So, it does not make sense to send PWM signals to the servo inputs with pulse widths above, or below these values.

> Parallax Continuous Rotation servos are actually Standard servos that have been modified. The reason a continuous rotation servo's output shaft turns instead of holding a particular position is because the link between the its output shaft and the feedback potentiometer its circuitry uses to determine the output shaft's position has been severed. When a continuous rotation servo receives a signal that would tell a standard servo to rotate to and hold a particular position, the missing link between the continuous rotation servo's output shaft and feedback potentiometer fools its circuitry into thinking that it never arrives at the proper position. Thus, the servo's circuitry continues to drive its built-in DC motor in an attempt to reach a position it never gets to. The end result is "continuous rotation".

4.16.3 The Basic Control Program

The program listing, below, shows the basic servo control program:

```
; ================================================================
; Programming the SX Microcontroller
; APP028.SRC
; ================================================================
device  SX28L, oscxt2, turbo, stackx
freq    4_000_000
IRC_CAL IRC_FAST


reset   Main

LServo       = RC.7            ; Output to servo  - Left.
RServo       = RB.3            ; Output to servo  - Right.

LStop        = 115             ; Adjust values so that the servos don't move
RStop        = 115             ;  when Speed = 0 and Turn = 0

             org 8
Timer20L     ds 1              ; Counters for
Timer20H     ds 1              ;  20 ms timer
Ltimer       ds 1              ; Counter for left servo timer
Rtimer       ds 1              ; Counter for right servo timer
LSpeed       ds 1              ; Left servo speed
RSpeed       ds 1              ; Right servo speed
Speed        ds 1              ; The "Bot's" speed
Turn         ds 1              ; The "Bot's" turn factor

        org    0
ISR
        sb     LServo          ; Is left servo still on?
         jmp   :Right          ;  no - handle right servo
        dec    LTimer          ;  yes - count down
        sz                     ; Left timeout?
         jmp   :Right          ;  no - handle right servo
        clrb   LServo          ;  yes - left servo off
        mov    LTimer, LSpeed  ; Init left timer for next pulse
```

```
:Right
        sb     RServo          ; Is right servo still on?
        jmp    :Timer20        ;  no - handle 20 ms timer
        dec    Rtimer          ;  yes - count down
        sz                     ; Right timeout?
         jmp   :Timer20        ;  no - handle 20 ms timer
        clrb   RServo          ;  yes - right servo off
        mov    RTimer, Rspeed  ; Init right timer for next pulse
:Timer20                       ; Handle the 20 ms timer
        dec    Timer20L        ; Count down low order byte
        sz                     ; Is it zero?
         jmp   :ExitISR        ;  no - exit
        mov    Timer20L, #171  ;  yes, initialize and
        dec    Timer20H        ;  count down high order byte
        sz                     ; Is it zero?
         jmp   :ExitISR        ;  no - exit
        mov    Timer20H, #9    ;  yes, initialize and
        setb   LServo          ;  turn the servos
        setb   RServo          ;  on again
:ExitISR
        mov    w, #-52         ; ISR is invoked every 13 µs at
        retiw                  ;  4 MHz system clock

; Subroutine calculates the required values for LSpeed and RSpeed based upon
; the calibration factors, and the Speed and Turn parameters.
;
; Note: The routine does not check if the resulting values for LSpeed and
;       RSpeed are out of limits (100...130).
;
CalcValues
        mov    LSpeed, #LStop  ; Initialize left speed to stop
        mov    RSpeed, #RStop  ; Initialize right speed to stop

        add    LSpeed, Speed   ; Add the Speed value
        sub    RSpeed, Speed   ; Subtract the Speed value

        add    LSpeed, Turn    ; Add the turn value
        add    RSpeed, Turn    ;  to both speeds
        ret

Main
        clrb   LServo          ; Clear servo outputs
        clrb   RServo          ;
        mov    !rb, #%11110111 ; RB.3 is output for left servo
        mov    !rc, #%01111111 ; RC.7 is output for right servo

        mov    !option, #%00001000 ; Enable interrupts

        mov    Speed, #0
        mov    Turn, #0
        call   CalcValues

        jmp    $               ; Main program loops forever
```

As we will have to generate precisely timed PWM signals, it is obvious that we make use of an ISR that is periodically invoked on RTCC overflows. Therefore, some calculations are in order first:

The SX is clocked with 4 MHz here, and so the clock period is 250 ns. When we return from the ISR with the RETIW instruction, we have loaded –52 into w before, i.e. the ISR will be called every 13 µs. You may wonder why we use such an "odd" timing here. This will become obvious later, when we discuss a SX Tech Bot with infrared obstacle detection, for now, just accept this value.

As you know, the PWM signal for each servo should have a positive edge every 20 ms, and a negative edge 1.3 to 1.7 ms later, depending on the desired servo speeds and directions.

For a delay of 20 ms, approximately 1,539 ISR calls (20 ms/13 µs) are required. A division factor of 1,539 can be achieved by two nested decrementing counters, where one is initialized to 9, and the other to 171 (171 * 9 = 1,539).

A delay of 1.5 ms (this is the pulse width for a centered servo) requires approximately 115 ISR calls (1.5 ms/13µs = 115.38).

At the beginning of the program, we have defined some variables:

```
Timer20L     ds 1              ; Counters for
Timer20H     ds 1              ;  20 ms timer
Ltimer       ds 1              ; Counter for left servo timer
Rtimer       ds 1              ; Counter for right servo timer
LSpeed       ds 1              ; Left servo speed
RSpeed       ds 1              ; Right servo speed
```

Two variables, `Timer20L`, and `Timer20H` are the low and high counters for the 20 ms timing, `Ltimer`, and `Rtimer` are the counters for the pulse widths for the left and right servos. `Lspeed` and `Rspeed` contain the current speed values for the two servos. Let's assume for now, that they are both are initialized to 115. `LServo` and `RServo` are symbolic names for RB.3 and RC.7, the SX output pins, where the two servo inputs are connected.

The first instructions in the ISR code

```
ISR
        sb     LServo          ; Is left servo still on?
         jmp   :Right          ;  no - handle right servo
        dec    LTimer          ;  yes - count down
        sz                     ; Left timeout?
         jmp   :Right          ;  no - handle right servo
        clrb   LServo          ;  yes - left servo off
        mov    LTimer, LSpeed  ; Init left timer for next pulse
```

handles the pulse for the left servo. If the servo output is still high, `LTimer` is decremented each time the ISR is invoked until `LTimer` becomes zero. In this case (after 1.5 ms when `LTimer` was initialized to 115 before), the servo output is set to low, and **LTimer** is re-initialized with the contents of **LSpeed** (115 for now). In case the servo output is already low, execution continues at:

```
:Right
        sb     RServo          ; Is right servo still on?
        jmp    :Timer20        ;  no - handle 20 ms timer
        dec    Rtimer          ;  yes - count down
        sz                     ; Right timeout?
         jmp   :Timer20        ;  no - handle 20 ms timer
        clrb   RServo          ;  yes - right servo off
        mov    RTimer, Rspeed  ; Init right timer for next pulse
```

This code performs the similar actions for the right servo, as described for the left servo, before. When this code is done, or when the right servo output is already low, execution continues with:

```
:Timer20                       ; Handle the 20 ms timer
        dec    Timer20L        ; Count down low order byte
        sz                     ; Is it zero?
         jmp   :ExitISR        ;  no - exit
        mov    Timer20L, #171  ;  yes, initialize and
        dec    Timer20H        ;  count down high order byte
        sz                     ; Is it zero?
         jmp   :ExitISR        ;  no - exit
        mov    Timer20H, #9    ;  yes, initialize and
        setb   LServo          ;  turn the servos
        setb   RServo          ;  on again
:ExitISR
        mov    w, #-52         ; ISR is invoked every 13 µs at
        retiw                  ;  4 MHz system clock
```

Here, the low-order counter of the 20 ms timer is decremented first. When it is not yet zero, the ISR is terminated. In case it is zero, it is re-initialized to 171, and the high-order counter is decremented. When this one becomes zero, it will be re-initialized to 9, and both servo inputs are set to high level then. This happens every 20 ms.

We will discuss the remaining parts of this program later. Let’s first use the program ”as is” to calibrate the servos.

Enter the program code using the SX-Key Editor, or open a copy from the Parallax CD, and assemble it. For now, you should use the SX-Key debugger to load and run the code on the SX. Calibration is easier when there is only one servo connected, so leave the connected to RC7, but disconnect the other servo from RB3.

4.16.3.1 Calibrating the Servos

If no errors are reported by the assembler, start the debugger, and run the program at full speed. The servo is likely to respond one way if it is labeled Parallax Continuous Rotation and another way if it is labeled Parallax PM (pre-modified).

If the servo is labeled Parallax Continuous Rotation, it will most likely rotate in an arbitrary direction because it is not yet been manually calibrated. With this newer type of Parallax servo, calibration is quite easy: Locate the small hole at the side of the servo housing near where the cable comes out. Behind this hole is a trim potentiometer. Insert a small Philips screwdriver, and slowly turn the potentiometer in one direction. If the servo rotates faster, turn the potentiometer in the other direction until you find the setting where the servo completely stops, and no longer produces a humming sound. This setting is quite critical, so turn the potentiometer very slowly and in small increments.

If the servo is labeled Parallax, and the letters PM are highlighted, the internal potentiometer is pre calibrated. The servo should either stay still, or rotate very slowly. If it rotates slowly, it's usually easier to make a small adjustment to the program to make the servo stay still. This is a more attractive option than disassembling the servo to correct the small adjustment error in its potentiometer.

Let's assume that the PM servo rotates slowly clockwise. The program can compensate for the small potentiometer offset by sending slightly wider pulses. Therefore, stop the debugger, and increase the initial value for `RStop` from 115 to 116. Re-assemble the program, and run it again. Should the servo still turn right, but at a slower speed, you need to further increase the initial value of `RStop`. If the servo starts turning in the opposite direction, the offset is too large, so decrease the value. If you are lucky, you will find the correct initial value for a complete stop after a while. Depending on the tolerances of the servo, and the relatively coarse timing of our program (we will discuss this later), you might not be able to exactly match that value, so at least find a value that slows down the servo as much as possible.

Next, disconnect the servo from RC7, and connect the other servo to RB3. Repeat the same calibration procedure just discussed for the second servo.

After you have calibrated your servos, you can construct your SX Tech Bot by following the instructions in Robotics with the Boe-Bot, available for free Download from www.parallax.com. While assembling your SX Tech Bot, there are two differences to keep in mind. First, when attaching the standoffs to the chassis, use the four holes that have the same pattern and dimensions as the hole pattern on the SX-Tech Board. Second, the SX Tech Bot's left servo should be connected to RC7, and its right servo should be connected to RB3.

4.16.3.2 More Parts of the Control Program

For the SX Tech Bot to operate autonomously, the 4 MHz ceramic resonator supplied with the SX Tech Toolkit should be inserted into the 3-socket header on the SX-Tech Board. The SX should then be programmed (CTRL-P), and finally, the SX-Key should be disconnected from the SX-Tech board. Since the resonator is supplying the clock signal, the FREQ 4_000_000 for the SX-Key is no longer in effect. Instead, the **OSCXT2** directive sets the appropriate feedback and drive settings for the SX Tech Toolkit's ceramic resonator.

IMPORTANT: A common mistake is to unplug the SX-Key and wonder why the SX Tech Bot is not functioning. The SX Tech Bot will not function until the resonator is plugged-in. Also, when you are using the external resonator, always remember to remove before using the SX-Key for debugging. You can program the SX chip while the resonator is plugged in, but the Debugging tools will not work until the resonator is unplugged.

The main program code looks like this:

```
Main
        clrb   LServo         ; Clear servo outputs
        clrb   RServo         ;
        mov    !rb, #%11110111 ; RB.3 is output for left servo
        mov    !rc, #%01111111 ; RC.7 is output for right servo

        mov    !option, #%00001000 ; Enable interrupts

        mov    Speed, #0
        mov    Turn, #0
        call   CalcValues

        jmp    $              ; Main program loops forever
```

At the very beginning, the servo output bits are cleared to avoid any “glitches” at startup, and then, the two port pins RB.3 and RC.7 are configured as outputs to control the two servo inputs.

Next, RTCC interrupts are enabled, and the two variables, **Speed** and **Turn** are both initialized to zero. The idea here is, to make the interface to the servo control code in the ISR as simple as possible. Instead of defining the initial values for the two variables **LSpeed** and **RSpeed** to control the pulse widths of the PWM signals for various SX Tech Bot moves and turns, we use **Speed** to control the forward/backward speed, and **Turn** to control the left/right turn rate.

When **Speed** is 0, the SX Tech Bot shall stop. For **Speed** > 0, the SX Tech Bot should move forward, and for **Speed** < 0, the SX Tech Bot should move backwards. When **Turn** is 0, the SX Tech Bot shall not turn at all; it will either go straight forward or straight backward, or stop, depending on the value of speed. When is **Turn** > 0, it should turn right, and when **Turn** is < 0, it should turn left. The greater the absolute values of **Speed** and **Turn** are, the faster the SX Tech Bot moves or turns in the specified directions. For now, the Main routine initializes both, **Speed** and **Turn** to 0, i.e. the SX Tech Bot should not move or turn at all. So in this mode, we can calibrate the servos.

Following the initialization of **Speed** and **Turn**, **CalcValues** is called. This subroutine performs the necessary calculations to convert **Speed** and **Turn** into the initialization values that are stored in **LSpeed** and **RSpeed**. We’ll discuss this subroutine in a moment. Finally, the program enters into an endless look because the remaining tasks are handled by the ISR for now.

Before discussing the **`CalcValues`** routine, let's discuss some general considerations on the values for **`Speed`** and **`Turn`**, and the resulting settings of **`LSpeed`** and **`RSpeed`**. As mentioned before, both servos stop when **`LSpeed`** and **`RSpeed`** both contain 115 (or the values you have determined during "software servo calibration"). When the values are below 115, the servos turn clockwise, and on values above 115, they turn counterclockwise.

In order to have the SX Tech Bot move straight forward, the left servo must turn right at a certain counterclockwise speed, and the right servo must turn clockwise at the same speed. This means that **`LSpeed`** must be 115+v, and **`RSpeed`** must be 115-v. For a straight backward direction, the left servo must turn clockwise, and the right servo must turn counterclockwise. Therefore, L**`Speed`** must be 115-v, and for **`RSpeed`** it is 115+v.

The SX-Tech Bot can also rotate in place to perform turns. When viewed from above, the SX-Tech Bot must rotate counterclockwise to perform a left turn, and clockwise to perform a right turn. For the SX-Tech Bot to perform a left turn, both its left and right wheels must rotate clockwise. Thus, **`RSpeed`** and **`LSpeed`** both are 115-t . For the SX-Tech bot to turn right, both its wheels must turn counterclockwise, so **`RSpeed`** and **`LSpeed`** should both be 115+t. The table, below, summarizes the various combinations, where v and t are now replaced by **`Speed`** and **`Turn`**, where both can be positive or negative:

Movement	**Speed**	**Turn**	**Lspeed**	**RSpeed**
Stop	0	0	115 + Speed + Turn	115 - Speed + Turn
Straight forward	> 0	0	115 + Speed + Turn	115 - Speed + Turn
Straight backward	< 0	0	115 + Speed + Turn	115 - Speed + Turn
Turn right	0	> 0	115 + Speed + Turn	115 - Speed + Turn
Turn left	0	< 0	115 + Speed + Turn	115 - Speed + Turn
Right and curve fwd	> 0	> 0	115 + Speed + Turn	115 - Speed + Turn
Left and curve fwd	> 0	< 0	115 + Speed + Turn	115 - Speed + Turn
Curve backward and right	< 0	< 0	115 + Speed + Turn	115 - Speed + Turn
Curve backward and left	< 0	> 0	115 + Speed + Turn	115 - Speed + Turn

The code for the **`CalcValues`** subroutine converts these terms (115, Speed and Turn) into the equivalent SX instructions:

```
CalcValues
        mov    LSpeed, #LStop  ; Initialize left speed to stop
```

```
        mov    RSpeed, #RStop  ; Initialize right speed to stop

        add    LSpeed, Speed   ; Add the Speed value
        sub    RSpeed, Speed   ; Subtract the Speed value

        add    LSpeed, Turn    ; Add the turn value
        add    RSpeed, Turn    ;  to both speeds
        ret
```

LSpeed and **RSpeed** first are initialized with the two stop constants (usually, 115), then **Speed** is added to **LSpeed**, and subtracted from **RSpeed**, and finally **Turn** is added to both, **LSpeed**, and **RSpeed**.

4.16.4 Some Timing Considerations

As already mentioned, the Parallax servos make maximum speeds at pulse widths of 1.3 ms (clockwise), and 1.7 ms (counterclockwise). This means that the values of **LSpeed** and **RSpeed** should not go above 130 (for 1.7 ms) or below 100 (for 1.3 ms). As **CalcValues** does not check if the resulting values are out of limits you will need to take care of that when assigning values to **Speed** and **Turn**.

With the timing provided by the ISR, there are only 15 increments for **LSpeed** and **RSpeed** to control the servo speed into each direction from 0% to 100%. In other words, each increment corresponds to about 6.7%. This is a relatively coarse resolution, but for the next experiments, this is fine enough.

If you are using the older Parallax PM servos together with the “software calibration” you should now understand why you possibly could not find a value for **LStop** or **RStop** to completely stop the servos.

In order to increase the resolution, you might consider invoking the ISR more often, e.g. every 7.5µs by replacing the line

```
        mov    w, #-52         ; ISR is invoked every 13 µs
```

with

```
        mov    w, #-26         ; ISR is invoked every 7.5 µs
```

Besides adjusting the 20 ms timer (which is quite easy – simply initialize **Timer20H** with 18), this also means that the possible values for **LSpeed**, and **RSpeed** would range from 200 to 260 then. As an 8-bit counter can only handle a maximum divide-by 256 factor, this means that you would have to use two-byte counters for **LTimer** and **RTimer**, so more code in the ISR would be required to handle them. On the other hand, invoking the ISR every 26th clock cycle does not allow for total ISR execution times above 26 clock cycles. We are already close to that value, so no more code in the ISR would be possible (which we plan to add later). Besides this, the **Main** code would not have much time to execute its own instructions because most of the time, it would be interrupted to service the ISR. For now, this is not a problem because `Main` just performs an endless loop, but we plan to add more instructions there later.

While increasing the SX clock frequency, say to 10 MHz, or even more, would allow for finer timing resolution, it would also cause the SX to consume more power. If you are interested in experimenting with higher clock rates, consider also replacing the four 1.5 V AA batteries with five or six 1.2 V AA rechargeable batteries, or a 7.5 V rechargeable battery pack. For now, however, let's be happy with the 4 MHz clock, and keep in mind that precision is not a major feature here.

4.16.5 The SX Tech Bot's First Walk (in the Park)

At this point, you should have completed the necessary servo calibrations and mechanical assembly of the SX Tech Bot, so now the time has come to send out the SX Tech Bot to make its first steps.

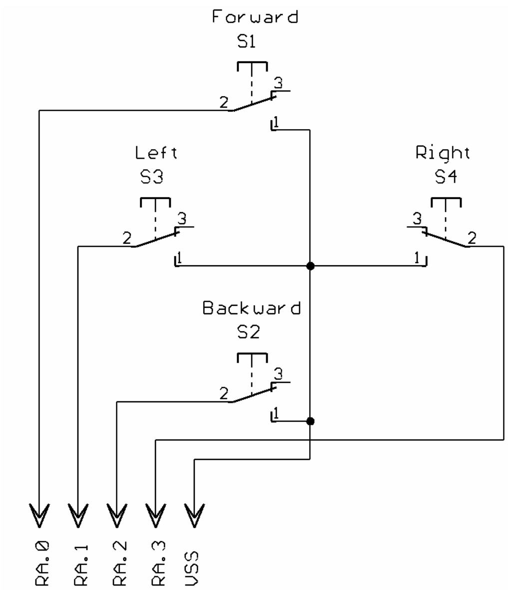
In the `Main` section of our program, simply assign values other than 0 to `Speed` to make the SX Tech Bot walk. Remember that positive values result in a forward movement, where negative values make it back up. You may also assign values other than 0 to `Turn` in order to let the SX Tech Bot turn around, or perform curves when both, `Speed` and `Turn` are other than 0. Feel free to experiment with various combinations of values for `Speed` and `Turn` (positive, or negative), but keep an eye on the resulting values for `LSpeed` and `RSpeed` – they should not exceed the limit from 100 to 130.

Keep in mind, after each modification to `Speed` or `Turn` , it will be necessary to re-load the program into the SX. This will not be necessary when the SX is making decisions based on sensor inputs.

4.16.6 Adding a "Joystick" to the SX Tech Bot

You may find it annoying to re-program the SX each time you want to make changes to the `Speed` and `Turn` assignments in the Main program, so here comes an improvement:

If you have one of the "antiquarian" joysticks on hand that came with four micro switches for "forward", "backward", "left", and "right", grab it from your junk box, and connect it to the SX Tech Bot. As an alternative, you could install four pushbuttons on a breadboard, according to the schematic, below:



Connect the "Joystick Assembly" via a cable (the length depends on how much "freedom" you want to allow for the SX Tech Bot) to the four header sockets on the SX Tech board marked RA0 through RA3, and to one of the free Vss header sockets. Maybe, it is a good idea to solder the four leads going to RA.0 through RA.4 to a four-pin header, and the fifth lead going to Vss to a single header pin before plugging them in.

Change the **Main** section in our basic program to look like this:

```
Main
        mode   $0e                   ; Select PLP
        mov    !ra, #%11110000       ; Activate pull-ups on port A
        clrb   LServo
        clrb   RServo
        mode   $0f                   ; Select TRIS
        mov    !rb, #%11110111       ; RB.3 is output for left servo
        mov    !rc, #%01111111       ; RC.7 is output for right servo
        mov    !option, #%00001000   ; Enable interrupts
CheckSwitches
        cjne   Timer20H, #9, $        ; Wait until after servo pulses have
        cjne   Timer20L, #9, $    ; been delivered to check buttons.
        mov    Speed, #0
        mov    Turn, #0
        snb    ra.0               ; Forward button pressed?
          jmp  :TestBack
        mov    Speed, #7          ; Positive speed = forward
        jmp    :TestLeft
:TestBack
```

```
        snb    ra.2                ; Backward button pressed?
         jmp   :TestLeft
        mov    Speed, #-7          ; Negative speed = backward
:TestLeft
        snb    ra.1                ; Left button pressed?
         jmp   :TestRight
        mov     Turn, #-7          ; Negative turn = left
        jmp    :TestEnd
:TestRight
        snb    ra.3                ; Right button pressed?
         jmp    :TestEnd
        mov    Turn, #7            ; Positive turn = right
:TestEnd
        call   CalcValues
        jmp    CheckSwitches
```

In the beginning of this program version, we activate the internal pull-up resistors for all port A pins, so that we don't need to connect external pull-up resistors to the pushbuttons. Instead of an endless "do-nothing" loop, the main program executes code that checks to find out which pushbutton or pair of pushbuttons are pressed, and sets the values for **`Speed`** and **`Turn`** accordingly. You can press single buttons, such as forward, backward, left or right. You can also press and hold combinations of buttons such as forward and right, backward and left, etc.

Please note the order of how the buttons are checked. This makes it impossible to let the SX Tech Bot go "crazy" in case you push two opposite buttons, like Forward and Backward at the same time. When the Forward button is pressed, no check for the Backward button will be performed, and so it does not matter if you have pressed it, or not. The same is true for the Left and Right buttons.

For **`Speed`**, we use a value of 7 here, and 7 for **`Turn`** as well. In the event that one speed and one turn button are pressed at the same time, it causes the SX-Tech Bot to perform a pivot-turn, where one wheel stays still while the other turns the bot. This is different from a rotating turn, where both wheels turn in the same directions at the same speeds. This gives you eight possible maneuvers with four buttons.

4.16.7 The SX Tech Bot "Learns" to Detect Obstacles

So far, we did not really build a robot, but just a very simple "toy". Usually, robots are supposed to have some kind of "brainpower", and this is what we are going to add now.

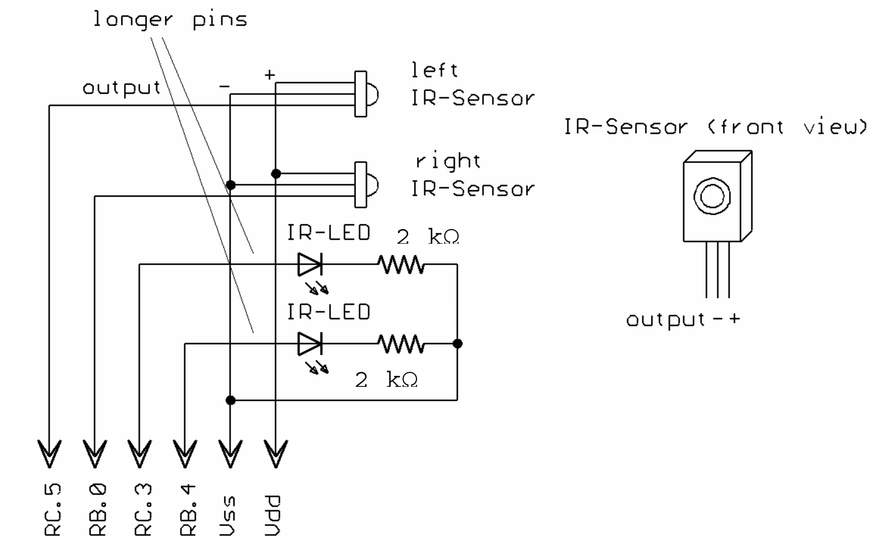
The SX Tech Bot Kit comes with two infrared (IR) LEDs, and two infrared sensors. In this experiment, we will attach these components plus two resistors to the breadboarding area on the SX Tech board. The two IR LEDs are used like headlights of a car to illuminate any obstacles that might occur along the SX Tech Bot's path. The two sensors are used to detect the infrared light that will be reflected from such obstacles, and cause a change in the SX Tech Bot's maneuver.

The infrared sensors used here, have a built-in filter that makes them only sensitive for infrared light that is pulsed with a frequency of approximately 38.5 kHz. That is, the sensors will only react on infrared light that is turned on and off 38.500 times per second. This prevents interference with other infrared sources, like the sunlight (which is turned on and off only once per day), and other light sources powered by mains power. Such lights usually flash at 100 or 120 Hz, depending on the country where you live (with 50 or 60 Hz mains frequency).

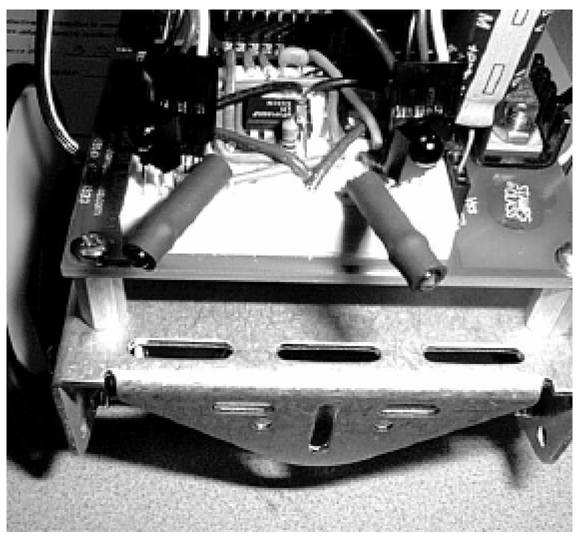
The schematic, below, shows the necessary wiring of the two infrared sensors, the infrared LEDs, and the two resistors that are necessary to limit the LEDs current. The photograph, further below, gives you an idea where, and how to position the IR sensors and LEDs on the breadboard. Adjust the left LED and sensor, to that thy are “looking” about 30° to the left, and let the right LED and sensor “look” about 30° to the right.

Be careful to correctly connect both, the LEDs and the sensors. The longer LED pins go to the SX ports, and the shorter pins to the resistors. Please refer to the front view drawing of the sensor to correctly identify the three pins. Make sure that the SX Tech Bot's left sensor output (shown at the right side of the picture) is connected to RC.5, and the right sensor output (shown on the left side of the picture) is connected to RB.0. The same applies to the IR LEDs. The one on the SX Tech Bot's left side should be connected to RC.3, and the one on its right side should be connected to RB.4.

Also be careful that there are no short circuits between the leads as the breadboarding area is much more “crowded” now. If necessary, cover the leads with insulating tube, or use isolated wires.



Wiring the IR Sensors and LEDs



Aligning the IR Sensors and LEDs

4.16.7.1 The Control Program for the Obstacle-Detecting SX Tech Bot

The program is an enhanced version of the basic version we have used to calibrate the servos. Here comes the program listing:

```
; ===================================================================
; Programming the SX Microcontroller
; APP029.SRC
; ===================================================================
device  SX28L, oscxt2, turbo, stackx
freq    4_000_000
IRC_CAL IRC_FAST

reset   Main

LServo       = RC.7            ; Output to servo  - left
RServo       = RB.3            ; Output to servo  - right
LSensor      = RC.5            ; Input for left IR sensor
RSensor      = RB.0            ; Input for right IR sensor
LLED         = RC.3            ; Output for left IR LED
RLED         = RB.4            ; Output for right IR LED
Calibrate    = RA.0            ; Input for calibrate jumper

LStop        = 115             ; Adjust values so that the servos don't move
RStop        = 115             ;  when Speed = 0 and Turn = 0

             org 8
Timer20L     ds 1              ; Counters for
Timer20H     ds 1              ;  20 ms timer
Ltimer       ds 1              ; Counter for left servo timer
Rtimer       ds 1              ; Counter for right servo timer
LSpeed       ds 1              ; Left servo speed
RSpeed       ds 1              ; Right servo speed
Speed        ds 1              ; The "Bot's" speed
Turn         ds 1              ; The "Bot's" turn factor

             org $30
Sensors      ds 1

        org    0
ISR
        sb     LServo          ; Is left servo still on?
         jmp   :Right          ;  no - handle right servo
        dec    LTimer          ;  yes - count down
        sz                     ; Left timeout?
         jmp   :Right          ;  no - handle right servo
        clrb   LServo          ;  yes - left servo off
        mov    LTimer, LSpeed  ; Init left timer for next pulse
:Right
        sb     RServo          ; Is right servo still on?
        jmp    :Timer20        ;  no - handle 20 ms timer
        dec    Rtimer          ;  yes - count down
        sz                     ; Right timeout?
         jmp   :Timer20        ;  no - handle 20 ms timer
        clrb   RServo          ;  yes - right servo off
```

```
        mov    RTimer, Rspeed  ; Init right timer for next pulse
:Timer20                       ; Handle the 20 ms timer
        dec    Timer20L        ; Count down low order byte
        sz                     ; Is it zero?
         jmp   :ExitISR        ;  no - exit
        mov    Timer20L, #171  ;  yes, initialize and
        dec    Timer20H        ;  count down high order byte
        sz                     ; Is it zero?
         jmp   :ExitISR        ;  no - exit
        mov    Timer20H, #9    ;  yes, initialize and
        setb   LServo          ;  turn the servos
        setb   RServo          ;  on again
:ExitISR
        movb   LLED, Timer20L.0        ; Toggle both
        movb   RLED, Timer20L.0        ;  IR LEDs
        mov    w, #-52                 ; ISR is invoked every 13 µs at
        retiw                          ;  4 MHz system clock

; Subroutine calculates the required values for LSpeed and RSpeed based upon
; the calibration factors, and the Speed and Turn parameters.
;
; Note: The routine does not check if the resulting values for LSpeed and
;       RSpeed are out of limits (100...130).
;
CalcValues
        mov    LSpeed, #LStop          ; Initialize left speed to stop
        mov    RSpeed, #RStop          ; Initialize right speed to stop

        add    LSpeed, Speed           ; Add the Speed value
        sub    RSpeed, Speed           ; Subtract the Speed value

        add    LSpeed, Turn            ;  no, add the turn value
        add    RSpeed, Turn            ;  to both speeds
        ret

Main
        mode   $0e                     ; Select PLP
        mov    !ra, #%11111110         ; Activate pull-up on pin 0
        clrb   LServo
        clrb   RServo
        mode   $0f                     ; Select TRIS
        mov    !rb, #%11110111         ; RB.3 is output for left servo,
        mov    !rc, #%01111111         ; RC.7 is output for right servo,
        mov    !option, #%00001000     ; Enable interrupts
        mov    Speed, #0
        mov    Turn, #0
        bank   Sensors
:Loop
        clr    Speed
        clr    Turn
        clr    Sensors
        sb     Calibrate               ; Do nothing when the Calibrate
          jmp  :Loop                   ;  jumper is in position
        cjne   Timer20H, #3, $         ; Wait for Timer20H = 3
```

```
        mov     !rb, #%11100111         ; Right IRLED to output
        cjne    Timer20H, #2, $         ; Wait for Timer20H = 2
        mov     !rb, #%11110111         ; Right IRLED to input
        movb    Sensors.0, RSensor      ; Copy right IR detect bit
        mov     !rc, #%01110111         ; Left IRLED to output
        cjne    Timer20H, #1, $         ; Wait for Timer20H = 1
        movb    Sensors.1, LSensor      ; Copy left IR detect bit
        mov     !rc, #%01111111         ; Left IRLED to input
        mov     w, Sensors              ; Depending on the sensor states,
        jmp     pc+w                    ;  jump to the state handler
        jmp     :Both
        jmp     :Right
        jmp     :Left
        jmp     :None
:Both                                   ; Both sensors detect, so
        mov     Speed, #-10             ;  back up
        jmp     :Done
:Right                                  ; Right sensor detects, so
        mov     Turn, #10               ;  turn left
        jmp     :Done
:Left                                   ; Left sensor detects, so
        mov     Turn, #-10              ;  turn right
        jmp     :Done
:None                                   ; No obstacles at all, so
        mov     Speed, #10              ;  go forward
:Done
        call    CalcValues              ; Calculate LSpeed and RSpeed
        jmp     :Loop                   ; Repeat it forever
```

Compared to the servo calibration program, we have added some more definitions for the port I/O pins that are connected to the IR sensors and LEDs now, and for a "Calibrate" input. We also have introduced a new variable, **`Sensors`** in bank $30, two additional instructions in the ISR following the **`:ExitISR`** label, and added some code to the **`Main`** program.

As mentioned before, the IR sensors have built-in filters that let pass infrared light only that is pulsed at a frequency of approximately 38.5 kHz. This means that we need to turn the two IR LEDs on and off at that rate. This happens in the ISR due to the two new instructions:

```
:ExitISR
        movb   LLED, Timer20L.0  ; Toggle both
        movb   RLED, Timer20L.0  ;  IR LEDs
```

`Timer20L` is decremented on each ISR call, i.e. every 13 µs, so the LEDs are repeatedly turned on for 13 µs, and turned off for another 13 µs, or the on-off period is 26µs which is equivalent to a frequency of 38.46 kHz. This is close enough to 38.5 kHz. Now it becomes clear why we are using such an "odd" interrupt period of 13 µs; it makes it really easy to "flash" the LEDs.

At the beginning of the **`Main`** section, we configure a pull-up resistor on pin 0 of port A, two additional port pins as outputs for the LEDs (RB.4 and RC.3), and select the bank for the **`Sensors`** variable.

The SX Tech Bot's "brainpower" lies in the new **:Loop** code with the **Main** section:

```
:Loop
        clr    Speed
        clr    Turn
        clr    Sensors
        sb     Calibrate              ; Do nothing when the Calibrate
          jmp  :Loop                  ;  jumper is in position
        cjne   Timer20H, #3, $        ; Wait for Timer20H = 3
        mov    !rb, #%11100111        ; Right IRLED to output
        cjne   Timer20H, #2, $        ; Wiat for Timer20H = 2
        mov    !rb, #%11110111        ; Right IRLED to input
        movb   Sensors.0, RSensor     ; Copy right IR detect bit
        mov    !rc, #%01110111        ; Left IRLED to output
        cjne   Timer20H, #1, $        ; Wait for Timer20H = 1
        movb   Sensors.1, LSensor     ; Copy left IR detect bit
        mov    !rc, #%01111111        ; Left IRLED to input
        mov    w, Sensors             ; Depending on the sensor states,
        jmp    pc+w                   ;  jump to the state handler
        jmp    :Both
        jmp    :Right
        jmp    :Left
        jmp    :None
:Both                                 ; Both sensors detect, so
        mov    Speed, #-10            ;  back up
        jmp    :Done
:Right                                ; Right sensor detects, so
        mov    Turn, #10              ;  turn left
        jmp    :Done
:Left                                 ; Left sensor detects, so
        mov    Turn, #-10             ;  turn right
        jmp    :Done
:None                                 ; No obstacles at all, so
        mov    Speed, #10             ;  go forward
:Done
        call   CalcValues             ; Calculate LSpeed and RSpeed
        jmp    :Loop                  ; Repeat it forever
```

At each entry into **:Loop**, we clear **Speed**, **Turn**, and **Sensors** for a clean start. We then test the port bit (RA.0) that is assigned to **Calibrate**. When this bit is clear, RA.0 has been connected to Vss in order to activate the calibration mode. In this case, we do nothing else. It is a good idea to make this mode available because normally, the SX Tech Bot would always be in motion. It may, from time to time, be necessary to stop the servos in order to re-calibrate them, especially in situations where the vibration from prolonged operation causes the manually adjusted potentiometer calibration setting to drift.

When calibrate mode is inactive, the first step is to wait for a full cycle of servo pulses to complete. For the sake of navigation, the effective sampling rate of checking the detectors 40 to 50 times per second is ample. This also prevents sampling and adjustment while the actual pulse is delivered, which could cause instability in a given pulse width.

Although the IR LED I/O ports (RB4 and RC3) are toggled each time through the ISR, the IR LEDs do not flash on/off because the output bits for these ports are disabled since !RB.4 and !RC.3 are set to 1 (input). The program waits until the value of Timer20H has counted down to 3. At this point, the next step is to broadcast infrared to SX Tech Bot's right IR LED. This is accomplished by clearing !RB.4 (setting it to output). The signal is allowed until the ISR decrements the Timer20H variable, at which point the right IR detector's output is tested and stored in bit-0 of the `Sensors` variable. Then, !RB.4 is set, restoring RB4 to input and in turn stopping the right IR LED from broadcasting IR. This process is repeated for the SX Tech Bot's left IR LED and detector, and the result is stored in bit-1 of the sensors variable.

A key feature of this IR detection algorithm is that the detections are mutually exclusive. If both IR LEDs were to broadcast at the same time, both detectors might "see" an object that is really only on one side of the SX Tech Bot. So broadcasting and sampling on one side, then moving on to the other side ensures accurate detection of an object's position relative to the SX Tech Bot. Bit-1 of Sensors is 1 if an object on the SX Tech Bot's left is not detected, or 0 if an object is detected. Likewise, bit-0 of Sensors is 1 if an object on the right is not detected, or 1 if it is. As a result, the `Sensors` variable can only contain the values shown in the table, below:

Sensors	**Obstacle to the…**
0	left and right
1	right
2	left
3	no obstacles

After bits for both detectors are stored in the `Sensors` variable, navigation decisions can be made. We take this value as offset for a jump table that directs the program-flow to the right state handler. Depending on the current `Sensors` state, we set the according values for `Speed` and `Turn`, so that the SX Tech Bot changes its direction of movement, in order not to bump into the obstacle.

Please enter the program code into the SX-Key IDE, assemble, and transfer it into the SX Tech Bot’s SX for “stand-alone” execution, because this type of SX Tech Bot would not be too agile with the SX-Key “umbilical cord” still connected. Note that the sample code contains the device configuration `OSCXT2`, i.e. the external ceramic resonator must be used. Remember to insert the resonator into the SX Tech Board's 3-socket header so that the program executes after the SX-Key is unplugged from the SX Tech Board.

Should you prefer to use the Debugging environment, make sure to set the SX Tech Bot on something that will prevent its wheels from touching the ground. You can then run the program, place obstacles at various positions in front of the SX Tech Bot, and verify that the wheel rotations indicate it is performing the correct maneuver.

> IMPORTANT: This SX Tech Bot detection system performs well with reflective obstacles. For best results, use white cardboard boxes. Most colors of cardboard or paper boxes will work, as will your hand or foot so long as you are not wearing black shoes. Many black surfaces absorb infrared light so well that the SX Tech Bot will be blind to them. You can increase the SX Tech Bot's sensitivity to IR absorbing objects by reducing the value of the resistors in series with the IR LEDs. This also makes the SX Tech Bot more farsighted with reflective objects.

Keep in mind that you can test and recalibrate your SX Tech Bot's servos by connecting a jumper across the Vss and RA0 header sockets on the SX Tech board. When this connection is made, the servos should stay still. If they do rotate, or produce humming sounds, it is time for re-calibration. When you are done with it, remove the jumper, and off you go…the SX Tech Bot will autonomously roam and avoid obstacles it detects.

In the sample code, we have used a relatively high speed setting for the continuous rotation servos, with `Speed` and `Turn` ranging from -10 to 10 depending on each maneuver. Feel free to experiment with other values, and combination of values. You may find that slower settings perform better in crowded areas. This in combination with adjustments to the IR LED series resistors can prepare your SX Tech Bot for a variety of obstacle courses.

4.16.7.2 Some More Thoughts About the Obstacle-Detecting SX Tech Bot

Let’s think about what should happen (at least in theory) when the SX Tech Bot performs a forward move, perpendicular to a flat obstacle, like a wall:

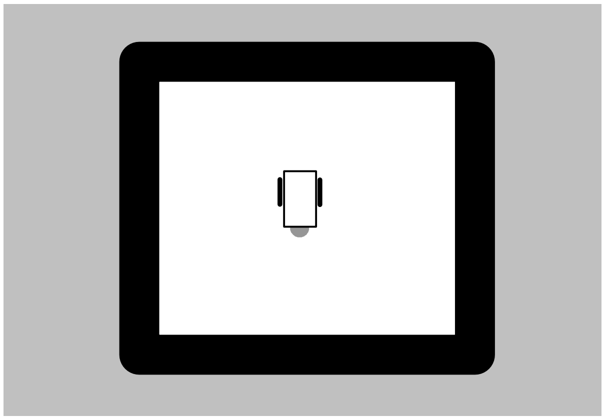
When the wall comes into “sight” of the sensors, both will report an obstacle, and according to the program logic, the SX Tech Bot would back up, until the sensors no longer report this obstacle. Then, it will again move forward closer to the wall until the sensors “see” the wall again, which will be the case after a short while. This means that the SX Tech Bot would move back and forth forever, until its batteries are dead.

But this is only theory. Due to the coarse resolution of the PWM signals, the servos will never run completely synchronized. This means that the SX Tech Bot will soon leave the straight perpendicular line to the wall, and one sensor will detect the wall earlier than the other, making the SX Tech Bot turn. In other words, the lack of precision is an advantage here, adding some “fuzziness” to our system.

Equipped with two infrared “headlights”, and “eyes”, the SX Tech Bot can easily be configured to react on “non-existing” obstacles. What does that mean at all?

You could align the IR LEDs and sensors to “look” at points on the surface, the SX Tech Bot is moving on. The `Main` code must be modified in a way that the SX Tech Bot moves straight forward as long as it “sees” an obstacle, that is the surface on which it is moving. In this case, it must react accordingly, when it detects an “abysm”, like the corner of the table it is moving on.

We leave it up to you, modifying the SX Tech Bot software to fulfill that task. When you try this, it is a good idea to first put a larger square of white cardboard, covered with black tape, some inches wide, on the floor, like this:



Here, the black tape acts as “abysm”, or “restricted area” without the danger that the SX Tech Bot drops down, in case it would ignore it. When you are sure that the program works as expected, and have verified that the SX Tech Bot does not “overshoot” the “restricted area”, you may actually try this experiment on a table. For safety reasons, it may be a good idea to run the SX Tech Bot at reduced speeds this time.

4.17 More Ideas for SX Tech Bot Applications

The examples shown here, are intended to give you a basic idea on how to control the SX Tech Bot servos using an SX controller, and how to “automate” the SX Tech Bot’s behavior.

You will certainly have other ideas in mind that could be realized, so the only limits are your imagination, and the precision of the SX Tech Bot.

Here are some tips for more experiments:

- Replace the IR components by two photo resistors (LDRs) that are "looking" at the floor, some inches in front of the SX Tech Bot at angles of about 30° to the right and to the left. You can then modify the SX Tech Bot program in a way that it follows a flashlight beam that you direct to the floor in front of the SX Tech Bot.
- Instead of optical devices, you could also attach some mechanical "whiskers" to the SX Tech Bot that pull two port pins low, when the right or left "whisker" touches an obstacle.
- Think of an SX Tech Bot with two sensors (IR sensors/LEDs, or mechanical "whiskers"), one "looking" to the right, and one "looking" forward. This could be used to help the SX Tech Bot find its way through a maze. Unfortunately, the SX does not have enough memory to store the shortest trace, but what about adding a serial EEPROM for additional storage capacity?
- As mentioned several times before, the resolution of the PWMs controlling the servos at 4 MHz system clock is quite coarse. Therefore, you may consider increasing the system clock, and modify the ISR code to handle shorter clock cycles. When designing a new timing concept, you should keep in mind to provide a "source" for the 38.5 kHz signal which is required to drive the LEDs for an infrared-based sensor system.

For more ideas on robotics, please visit the Parallax site at www.parallax.com, where you can find a lot of robotics-related material, including the text "*Robotics With the Boe-Bot, Student Guide*". Although this text is intended to be used with the BASIC Stamp-controlled version of the SX Tech Bot, it contains many hints, concepts and ideas that you may port to the "World of SX Robotics".