```
\ ---------------------
\ robot.fs
\ ---------------------
\ source code for the brains of TDR, my robot

\ ---------------------
\ I/O definitions
\ ---------------------

2 pin-d motor-reset    \ port d bit 2, pin 21
0 pin-a LED-a          \ port a bit 0, the on-board LED
0 pin-b left-bump      \ port b bit 0, left bump sensor
1 pin-b middle-bump    \ port b bit 1, middle bump sensor
2 pin-b right-bump     \ port b bit 2, right bump sensor


\ ---------------------
\ Variable definitions
\ ---------------------

variable my-count      \ counter location
variable cur-direction \ current motor direction
variable prev-direction \ previous motor direction
variable next-direction \ next direction to go
variable left-eye      \ temp spot to hold left IR sensor
variable right-eye     \ temp spot to hold right IR sensor
variable detect-no-h   \ high byte of detection freq
variable detect-no-l   \ low byte
variable detect-yes-h  \ high byte of detect metal freq
variable detect-yes-l  \ low byte
variable detect-center-h \ high byte of detect metal freq
variable detect-center-l \ low byte
variable detect-test-h \ high byte of detection test
variable detect-test-l \ low byte
variable detect-half-h \ high byte of test variable
variable detect-half-l \ low byte
variable pulse-loop    \ the counter to hold pulse loops

\ ---------------------
\ Constant definitions
\ ---------------------

$05 constant command-repeat \ how many times to repeat commands
\ motor definitions:
$01 constant left-forward
$00 constant left-backward
$03 constant right-forward
$02 constant right-backward
\ robot direction definitions:
$00 constant no-dir
$01 constant ahead-dir
$02 constant veer-left-dir
$04 constant veer-right-dir
$08 constant behind-dir
$10 constant turn-left-dir
$20 constant turn-right-dir
$40 constant spiral-dir
\ Speed constants:
$3F constant fast-speed
$1F constant slow-speed
$07 constant spiral-speed
\ IR sensor thresholds
$50 constant ir-near  \ about 2V - used to be $66
$33 constant ir-far   \ about 1V
```

```
\ how close something is to the IR sensors
$0A constant pulse-times \ how many times to repeat pulse
$45 constant detect-max

\ ---------------------
\ Word definitions
\ ---------------------
$06 org \ skip boot loader and leave space for ISR vectors

include libnibble.fs

\ rand
\ ---------------------
variable rand-test
macro
: rand ( -- n )
  rand-test @
  1 +
  rand-test !
;
target                 \ macro for speed since this is simple

\ ticks uses a negative tick count.
\ if the clock were 4 MHz and prescalar 64,
\ then each tick would be 64 microseconds
\ clock is actually 20 MHz.

\ ticks
\ ---------------------
: ticks ( -n -- )
  tmr0 !               \ store -n into timer 0
  t0if bit-clr         \ clear overflow bit
  begin
    clrwdt t0if bit-set?
  until                \ wait for timer overflow
;

variable delay-ms-count
\ delay-ms
\ ---------------------
: delay-ms ( n -- )
  delay-ms-count v-for
    -$4e ticks          \ wait for 1 ms ($9c is 10 ms for 4 MHz)
  v-next
;

variable delay-ts-count
\ delay-ts
\ ---------------------
: delay-ts ( n -- )
  delay-ts-count v-for
    $64 delay-ms       \ wait for 100 ms
  v-next
;

macro
\ analog0
\ ---------------------
: analog0 ( -- a0 )
  $81 adcon0 !         \ see init section, turn on channel 0
  go//done bit-set     \ start the A/D
  nop
  begin
    go//done bit-clr?  \ go//done should be 2
  until                \ wait for the /done bit to clear
  adresh @             \ read the analog port and put on stack
;
target

\ analog1
\ ---------------------
: analog1 ( -- a1 )
  $89 adcon0 !         \ see init section, turn on channel 1
  go//done bit-set     \ start the A/D
  nop
  begin
    go//done bit-clr?  \ go//done should be 2
  until                \ wait for the /done bit to clear
  adresh @             \ read the analog port and put on stack
;

\ analog2
\ ---------------------
: analog2 ( -- a2 )
  $91 adcon0 !         \ see init section, turn on channel 2
  go//done bit-set     \ start the A/D
  nop
  begin
    go//done bit-clr?  \ go//done should be 2
  until                \ wait for the /done bit to clear
  adresh @             \ read the analog port and put on stack
;

\ send-byte
\ ---------------------
: send-byte ( char -- )
  begin
    txif bit-set?
  until                \ wait until TXIF bit is set
  txreg !              \ write char to txreg to be transmitted
  nop
;

\ send-hex-byte
\ ---------------------
: send-hex-byte ( hex -- )
  dup 4 rshift nibble>hex send-byte
  $0f   and    nibble>hex send-byte
;

\ update-motors
\ ---------------------
: update-motors ( speed motor/dir -- )
  $80 send-byte
  $00 send-byte
  send-byte            \ motor/dir
  send-byte            \ speed
;

variable pause-count
\ pause-motors
\ ---------------------
: pause-motors ( -- )
  command-repeat pause-count v-for
    $00 right-forward update-motors \ right motor off
    $00 left-forward  update-motors \ left motor off
  v-next
  $64 delay-ms         \ pause for 100 ms
```

```
;

macro
\ init-analog
\ ---------------------
: init-analog ( -- )
  $02 adcon1 !        \ A/D control reg 1
      \ 7:   0 - left justified
      \ 6:   0 - ADCS2 0, oscillator/32
      \ 5-4: 0 - unused
      \ 3-0: 2 - 5 analog inputs, VDD - VSS range
  $c1 adcon0 !        \ A/D control reg 0
      \ 7-6: 3 - use internal AD clock
      \ 5-3: 0 - A/D channel 0
      \ 2:   0 - A/D start bit - do not start yet
      \ 1:   0 - unused
      \ 0:   1 - A/D power - turn on A/D system
;
target

macro
\ init-serial
\ ---------------------
: init-serial ( -- )
  $bf trisc !         \ Port C6 is Tx
  $40 spbrg !         \ set baud rate to 19,200
  $24 txsta !         \ transmit status reg
      \ 7: 0 - clock source - dont care
      \ 6: 0 - 8-bit transmission
      \ 5: 1 - transmit enable
      \ 4: 0 - asynchronous mode
      \ 3: 0 - unimplemented
      \ 2: 1 - high speed baud rate
      \ 1: 0 - transmit register empty
      \ 0: 0 - 9th transmit bit
  $90 rcsta !         \ receive status reg
      \ 7: 1 - enable serial port
      \ 6: 0 - 8-bit reception
      \ 5: 0 - dont care
      \ 4: 1 - enable continuous receive
      \ 3: 0 - disable address detection
      \ 2: 0 - no framing error
      \ 1: 0 - no overrun error
      \ 0: 0 - 9th receive bit
  \ one last thing:
  \ pulse the motor controller reset for a short time
  motor-reset bit-clr
  $01 delay-ts        \ wait a short while for reset
  motor-reset bit-set
;
target

\ init
\ ---------------------
: init ( -- )
  $fe trisa !         \ set port A<7..1> to inputs
  $0f trisb !         \ set port B<7..4> out, B<3..0> in
  $fb trisd !         \ set port D bit 2 output - motor-reset
  $05 option_reg !    \ set prescalar to 64 for timer0
  \ turn on all port b pull-ups - option_reg bit 7, /rbpu bit-clr
  \ now turn on timer 1 for random numbers
  $07 t1con !         \ enable timer1 as asynchronous counter
  init-analog
  init-serial
  $64 delay-ms         \ a short delay before leaving init
;

macro
\ start-async-counter
\ ---------------------
: start-async-counter ( -- )
  $00 tmr1l !
  $00 tmr1h !          \ clear timer register
  $07 t1con !          \ turn the counter on
;

\ stop-async-counter
\ ---------------------
: stop-async-counter ( -- )
  $06 t1con !          \ turn the counter off
;

\ restart-async-counter
\ ---------------------
: restart-async-counter ( -- )
  $07 t1con !          \ turn the counter on
;
target

variable high-temp
\ detect-metal
\ ---------------------
: detect-metal ( -- )
  \ saves count to test variables
  start-async-counter
  $64 delay-ms         \ search for a 1/20s
  tmr1h @ detect-test-h !
  tmr1l @ detect-test-l !
  tmr1h @ detect-test-h @ = if
    nop
  else \ high byte rolled over
    tmr1h @ detect-test-h !
    tmr1l @ detect-test-l !
  then
;

\ save-detect-no
\ ---------------------
: save-detect-no ( -- )
  detect-test-l @ detect-no-l !
  detect-test-h @ detect-no-h !
;

\ lower-detect-no
\ ---------------------
: lower-detect-no ( -- )
  detect-test-h @ detect-no-h @ <
  detect-test-h @ detect-no-h @ =
  detect-test-l @ detect-no-l @ <
  and or if \ if test-h < no-h or test-h = no-h & test-l < no-l
    detect-test-h @ detect-no-h !
    detect-test-l @ detect-no-l !
  then
;

\ save-detect-yes
\ ---------------------
: save-detect-yes ( -- )
```

```
  detect-test-l @ detect-yes-l !
  detect-test-h @ detect-yes-h !
;

\ higher-detect-yes
\ --------------------
: higher-detect-yes ( -- )
  detect-test-h @ detect-yes-h @ >
  detect-test-h @ detect-yes-h @ =
  detect-test-l @ detect-yes-l @ >
  and or if \ if test-h > yes-h or test-h = yes-h & test-l > yes-l
    detect-test-h @ detect-yes-h !
    detect-test-l @ detect-yes-l !
  then
;

\ center-detect
\ --------------------
: center-detect ( -- )
  detect-yes-l @ rrf-tos $7f and \ divide by two
  detect-no-l @ rrf-tos $7f and \ pull saved and divide by two
  + detect-center-l !  \ add, this can not overflow
  detect-no-l @ $80 and if \ is high bit set
    detect-yes-h @ $01  and
    detect-no-h  @ $01 and xor if \ only one shift in bit
      \ clear high bit
      detect-center-l @ $7f and detect-center-l !
    then
    detect-yes-h @ $01 and
    detect-no-h  @ $01 and or if \ any shift in bits
      detect-no-h @ $02 + detect-no-h !
    then              \ add 2 because rshift before add
  else                \ high bit not set
    detect-yes-h @ $01 and
    detect-no-h @ $01 and xor if \ only one shift in bit
      \ set high bit
      detect-center-l @ $80 or detect-center-l !
    then
    detect-yes-h @ $01 and
    detect-no-h  @ $01 and and if \ carry-out
      detect-no-h @ $02 + detect-no-h !
    then              \ add 2 because rshift before add
  then
  \ whew!  low 8 bits done, now for the high 8 bits
  detect-yes-h @ rrf-tos $7f and
  detect-no-h  @ rrf-tos $7f and
  + detect-center-h !
  \ done - easy now, right?
;

\ detect-result
\ --------------------
: detect-result ( -- result )
  \ returns 1 if metal is detected
  \ otherwise, returns 0
  \ first check to see if we ignore this due to
  \ bogus readings
  detect-half-h @ detect-max > if
    0
  else
    detect-half-h @ detect-center-h @ > if
      1 \ return true
    else
      detect-half-h @ detect-center-h @ = if
```

```
        detect-half-l @ detect-center-l @ > if
          1
        else
          0
        then
      else
        0 \ return false
      then
    then
  then
;

\ move-backward
\ ---------------------
: move-backward ( -- )
  command-repeat pause-count v-for
    fast-speed left-backward  update-motors
    fast-speed right-backward update-motors
  v-next
;

\ move-forward
\ ---------------------
: move-forward ( -- )
  command-repeat pause-count v-for
    fast-speed left-forward  update-motors
    fast-speed right-forward update-motors
  v-next
;

\ veer-left
\ ---------------------
: veer-left ( -- )
  command-repeat pause-count v-for
    slow-speed left-forward  update-motors
    fast-speed right-forward update-motors
  v-next
;

\ veer-right
\ ---------------------
: veer-right ( -- )
  command-repeat pause-count v-for
    fast-speed left-forward  update-motors
    slow-speed right-forward update-motors
  v-next
;

\ turn-left
\ ---------------------
: turn-left ( -- )
  command-repeat pause-count v-for
    slow-speed left-backward update-motors
    slow-speed right-forward update-motors
  v-next
;

\ turn-right
\ ---------------------
: turn-right ( -- )
  command-repeat pause-count v-for
    slow-speed left-forward   update-motors
    slow-speed right-backward update-motors
  v-next
```

```
;


\ spiral
\ ---------------------
: spiral ( -- )
  command-repeat pause-count v-for
    spiral-speed left-forward update-motors
    slow-speed right-forward  update-motors
  v-next
;


\ find-direction
\ ---------------------
\ writes to cur-direction and clears next-direction
\ saves cur-direction to prev-direction
: find-direction ( -- )
  rand                 \ get a random number
  $03 and              \ limit to low two bits
  dup $02 = if         \ veer right
    veer-right-dir
    cur-direction !    \ write to current direction
  then
  dup $01 = if         \ veer left, consume stack
    veer-left-dir
    cur-direction !    \ write to current direction
  then
  dup $03 = if         \ spiral
    spiral-dir
    cur-direction !    \ write to current direction
  then
  $00 = if             \ go straight
    ahead-dir
    cur-direction !    \ write to current direction
  then
  no-dir
  next-direction !     \ clear the next direction
;

\ select-direction
\ ---------------------
\ writes to cur-direction and clears next-direction
\ saves cur-direction to prev-direction
: select-direction ( -- )
  next-direction @     \ get the new direction
  cur-direction !      \ write selected direction
  no-dir
  next-direction !     \ clear the next direction
;

\ motor-time
\ ---------------------
\ writes a random time to my-count
: motor-time ( -- )
  rand                 \ get a random number
  $1f and              \ clip to 3.1 sec
  $0a +                \ make it last at least a second
  my-count !           \ write to my-count
  cur-direction @
  spiral-dir = if      \ if spiraling
    my-count @
    $32 +              \ spiral for 5 more seconds
    my-count !
  then
;

\ move-robot
\ ---------------------
: move-robot ( -- )
  cur-direction @      \ read current direction
  prev-direction @     \ read previous direction
  = if                 \ if last two directions were equal
    nop
  else
    pause-motors       \ change in direction
    cur-direction @
    prev-direction !   \ write cur direction to prev direction
  then

  cur-direction @ ahead-dir = if
    move-forward
  else
  cur-direction @ veer-left-dir = if
    veer-left
  else
  cur-direction @ veer-right-dir = if
    veer-right
  else
  cur-direction @ behind-dir = if
    move-backward
  else
  cur-direction @ turn-left-dir = if
    turn-left
  else
  cur-direction @ turn-right-dir = if
    turn-right
  else
  cur-direction @ spiral-dir = if
    spiral
  else
    pause-motors \ if no direction, stop
  then then then then then then then

  my-count @ 1- my-count !
  \ reduce count by one
;


variable celebrate-count
\ celebrate
\ ---------------------
: celebrate ( -- )
  pause-motors
  turn-left
  led-a bit-set
  $05 delay-ts

  4 celebrate-count v-for
    pause-motors
    turn-right
    led-a bit-clr
    $0a delay-ts

    pause-motors
    turn-left
    led-a bit-set
    $0a delay-ts
  v-next

  pause-motors
  turn-right
  led-a bit-clr
  $05 delay-ts

  \ clear detection registers
  0 detect-half-h !
  0 detect-half-l !
;


\ pulse
\ ---------------------
: pulse ( -- )
  \ check sensors first
  \ IR sensors are checked first
  analog1 right-eye !
  analog2 left-eye !

  left-eye @ ir-far >
  if
    turn-right-dir cur-direction !
    $02 my-count !     \ just turn until IR clears
  then

  right-eye @ ir-far >
  if
    turn-left-dir cur-direction !
    $02 my-count !     \ just turn until IR clears
  then
  left-eye @ ir-far >
  if
    right-eye @ ir-far >
    if                 \ if both eyes see something...
      behind-dir cur-direction !
      $0a my-count !     \ go backwards for 1 second
      rand $01 and if    \ even or odd
        veer-left-dir next-direction !
      else               \ turn left or right randomly
        veer-right-dir next-direction !
      then
    then
  then
  \ this should help when TDR approaches a wall head on

  \ bump sensors
  left-bump bit-clr?   \ hit something on the left?
  if
    behind-dir cur-direction !
    $0a my-count !     \ go backwards for 1 second
    veer-right-dir next-direction !
    \ set up veering to the right next
  then
  right-bump bit-clr?  \ hit something on the right?
  if
    behind-dir cur-direction !
    $0a my-count !     \ go backwards for 1 second
    veer-left-dir next-direction !
    \ set up veering to the left next
  then
  middle-bump bit-clr? \ hit something in the middle?
  if
```

```
    behind-dir cur-direction !
    $0a my-count !     \ go backwards for 1 second
    rand $01 and if    \ even or odd
      veer-left-dir next-direction !
    else               \ turn left or right randomly
      veer-right-dir next-direction !
    then
  then

  \ sensors updated, now move the robot
  move-robot

  \ check to see if count has expired
  my-count @ 0 = if    \ is count 0?
    next-direction @   \ yes, count is 0
    0 = if             \ do we have a next dir?
      find-direction   \ no, pick a random one
    else
      select-direction \ yes, use the one we have chosen
    then
    motor-time         \ pick a new time to move
  then

  $01 delay-ts         \ loop at 10 Hz
;


\ new-detect
\ ---------------------
: new-detect ( -- )
  detect-metal
  detect-test-l @ rrf-tos $7f and \ divide by two
  detect-half-l @ rrf-tos $7f and \ pull saved and divide by two
  + detect-half-l !    \ add, this can not overflow
  detect-half-l @ $80 and if \ is high bit set
    detect-half-h @ $01 and
    detect-test-h @ $01 and xor if \ only one shift in bit
      \ clear high bit
      detect-half-l @ $7f and detect-half-l !
    then
    detect-half-h @ $01 and
    detect-test-h @ $01 and or if \ any shift in bits
      detect-test-h @ $02 + detect-test-h !
    then               \ add 2 because rshift before add
  else                 \ high bit not set
    detect-half-h @ $01 and
    detect-test-h @ $01 and xor if \ only one shift in bit
      \ set high bit
      detect-half-l @ $80 or detect-half-l !
    then
    detect-half-h @ $01 and
    detect-test-h @ $01 and and if \ carry-out
      detect-test-h @ $02 + detect-test-h !
    then               \ add 2 because rshift before add
  then
  \ whew!  low 8 bits done, now for the high 8 bits
  detect-test-h @ rrf-tos $7f and
  detect-half-h @ rrf-tos $7f and
  + detect-half-h !
  \ done - easy now, right?
;


\ mainloop
\ --------------------
: mainloop ( -- )
  ahead-dir cur-direction !
  $0a my-count !
  no-dir next-direction !
  no-dir prev-direction !
  led-a bit-set        \ turn on the LED
  motor-reset bit-clr  \ don't confuse the motor driver with
                       \ regular serial traffic
  begin
    left-bump bit-clr?
  until

  led-a bit-clr
  begin
    detect-metal       \ detect until under the clip point
    detect-test-h @ detect-max <
  until
  save-detect-no
  $05 delay-ts
  begin
    detect-metal
    detect-test-h @ detect-max <
  until
  lower-detect-no
  $05 delay-ts
  begin
    detect-metal
    detect-test-h @ detect-max <
  until
  lower-detect-no

  detect-no-h @ send-hex-byte
  detect-no-l @ send-hex-byte
  $20 send-byte        \ seperate with a space

  begin
    right-bump bit-clr?
  until

  led-a bit-set
  begin
    detect-metal
    detect-test-h @ detect-max <
  until
  save-detect-yes
  $05 delay-ts
  begin
    detect-metal
    detect-test-h @ detect-max <
  until
  higher-detect-yes
  $05 delay-ts
  begin
    detect-metal
    detect-test-h @ detect-max <
  until
  higher-detect-yes

  detect-yes-h @ send-hex-byte
  detect-yes-l @ send-hex-byte
  $20 send-byte        \ seperate with a space

  center-detect        \ find the spot in the middle
  detect-center-h @ send-hex-byte
  detect-center-l @ send-hex-byte
  $20 send-byte        \ seperate with a space

  begin
    new-detect         \ detect and report treasure
    detect-half-h @ send-hex-byte
    detect-half-l @ send-hex-byte
    $20 send-byte      \ seperate with a space
    detect-result      \ return 0 or 1 if it matches
    if                 \ metal has been detected
      led-a bit-set
      detect-center-h @ detect-half-h !
      detect-center-l @ detect-half-l !
      new-detect       \ check to see if it is still there
      detect-result
      if               \ really happy here
        led-a bit-clr
        led-a bit-set
        $41 send-byte  \ send an 'A'
        $20 send-byte  \ send a space
      then
    else
      led-a bit-clr
    then
    $05 delay-ts
    middle-bump bit-clr?
  until                \ wait until the front bumper is presed

  led-a bit-clr        \ turn off the LED
  motor-reset bit-set  \ now it is time to send motor commands again
  $0a delay-ts         \ wait a second, then go

  begin
    pulse-times pulse-loop v-for
      pulse            \ 10 pulses between every metal detection attempt
    v-next             \ pulse takes ~0.1s, metal detection every 1s
    new-detect         \ detect and report treasure
    detect-result      \ return 0 or 1 if it matches
    if                 \ metal has been detected
      led-a bit-set
      pause-motors
      detect-center-h @ detect-half-h !
      detect-center-l @ detect-half-l !
      new-detect       \ check to see if it is still there
      detect-result
      if               \ really happy here
        celebrate      \ do a happy dance
      then
    else
      led-a bit-clr
    then
  again
;

\  main program
\ ---------------------
main : main
  init
  mainloop
;


\ ----------------------
\ Configuration Options
\ ----------------------
fosc-hs set-fosc        \ high-speed oscillator
false set-wdte          \ do not use watchdog timer
```