**Combined Code**

```
#include <Servo.h>                      // Include servo library


Servo servoLeftPin;
Servo servoRightPin;
Servo servoTurret;
const int servoTurretPin = 11;
const int pingPin = 10;


const int msPerTurnDegree = 6;              // For maneuvers
const int tooCloseCm = 20;                  // For distance decisions
const int bumpedCm = 6;
int ccwLim = 1400;                          // For turret servo control
int rtAngle = 900;
const int usTocm = 29;                      // Ping))) conversion constant

int sequence[] = {0, 2, 4, 6, 8, 10, 9, 7, 5, 3, 1};
const int elements = sizeof(sequence);
int cm[elements];


const int degreesTurret = 180/(sizeof(sequence)/sizeof(int)-1);

int i = -1;                             // Global index
int sign = 1;                           // Sign of increments
int theta = -degreesTurret;                 // Set up initial turret angle


void setup(){
  Serial.begin(9600);
  servoLeftPin.attach(13);
  servoRightPin.attach(12);
  setupPing();
  setupQTI();
 }

void loop(){
   loopPing();
   loopQTI();
 }

void setupPing()
```

```
{
  servoTurret.attach(servoTurretPin);
  maneuver(0, 0, 1000);                    // Stay still for 1 second
  turret(0);                               // Set turret to 0 degrees
}

void loopPing()                            // Main loop auto-repeats
{
  maneuver(200, 200);                      // Go full speed forward UFN
  i++;                                     // Increment turret index

  // Advance turret servo to next position in sequence and wait for it to get there.
  theta = sequence[i] * degreesTurret;
  turret(theta);
   // delay(100);

  cm[i] = cmDistance();                    // Measure cm from this turret angle

  // If object in +/- 36 degrees from center is within tooCloseCm threshold...
  if ((sequence[i]>=3) && (sequence[i]<=7) && (cm[i] < tooCloseCm))
  {
    maneuver(0, 0);                        // Stop moving
    int theta = findOpening();             // Get opening (in terms of sequence element
    theta *= degreesTurret;                // Convert sequence element to degree angle

    // Convert degree angle to time the BOE Shield-Bot wheels will have to turn to face
    // direction of turret.
    int turnAngleTime = (90 - theta) * msPerTurnDegree;

    if(turnAngleTime < 0)                  // If negative turning angle,
    {

      maneuver(-200, 200, -turnAngleTime);       // then rotate CCW for turningAngleTime ms
    }
    else                                   // If positive turning angle,
    {

      maneuver(200, -200, turnAngleTime);        // then rotate CW for turningAngleTime ms
    }

    maneuver(200, 200);                    // Start going forward again
  }

  if(i == 10)                              // If turret at max, go back to zero
```

```
 {
   i = -1;
 }


}



/*
 * Control BOE Shield-Bot servo direction, speed, set and forget version.
 * Parameters: speedLeft - left servo speed
 *             speedRight - right servo speed
 *             Backward  Linear  Stop  Linear   Forward
 *             -200      -100.....0.....100      200
 */
void maneuver(int speedLeft, int speedRight)
{
  // Call maneuver with just 1 ms blocking; servos will keep going indefinitely.
  maneuver(speedLeft, speedRight, 1);
}

/*
 * Control BOE Shield-Bot servo direction, speed and maneuver duration.
 * Parameters: speedLeft - left servo speed
 *             speedRight - right servo speed
 *             Backward  Linear  Stop  Linear   Forward
 *             -200      -100.....0.....100      200
 *             msTime - time to block code execution before another maneuver
 * Source:     http://learn.parallax.com/ManeuverFunction
 */
void maneuver(int speedLeft, int speedRight, int msTime)
{
  servoLeftPin.writeMicroseconds(1500 + speedLeft);   // Set Left servo speed
  servoRightPin.writeMicroseconds(1500 - speedRight); // Set right servo speed
  if(msTime==-1)                          // if msTime = -1
  {
    servoLeftPin.detach();                      // Stop servo signals
    servoRightPin.detach();
  }
  delay(msTime);                          // Delay for msTime
}

/*
```

```
 * Position the horn of a Parallax Standard Servo
 * Parameter: degreeVal in a range from 90 to -90 degrees.
 */
void turret(int degreeVal)
{
  servoTurret.writeMicroseconds(ccwLim - rtAngle + (degreeVal * 10));

}

/*
 * Get cm distance measurment from Ping Ultrasonic Distance Sensor
 * Returns: distance measurement in cm.
 */
int cmDistance()
{
  int distance = 0;                       // Initialize distance to zero
  do                                      // Loop in case of zero measurement
  {
    int us = ping(pingPin);                 // Get Ping))) microsecond measurement
    distance = convert(us, usTocm);         // Convert to cm measurement
    delay(3);                             // Pause before retry (if needed)
  }
  while(distance == 0);
  return distance;                        // Return distance measurement
}

/*
 * Converts microsecond Ping))) round trip measurement to a useful value.
 * Parameters: us - microsecond value from Ping))) echo time measurement.
 *           scalar - 29 for us to cm, or 74 for us to in.
 * Returns:    distance measurement dictated by the scalar.
 */
int convert(int us, int scalar)
{
   return us / scalar / 2;                // Echo round trip time -> cm
}

/*
 * Initiate and capture Ping))) Ultrasonic Distance Sensor's round trip echo time.
 * Parameter: pin - Digital I/O pin connected to Ping)))
 * Returns:   duration - The echo time in microseconds
 * Source:    Ping by David A. Mellis, located in File -> Examples -> Sensors
 * To-Do:     Double check timing against datasheet
 */
```

```
long ping(int pin)
{
  long duration;                          // Variables for calculating distance
  pinMode(pin, OUTPUT);                     // I/O pin -> output
  digitalWrite(pin, LOW);                  // Start low
  delayMicroseconds(2);                     // Stay low for 2 us
  digitalWrite(pin, HIGH);                 // Send 5 us high pulse
  delayMicroseconds(5);
  digitalWrite(pin, LOW);
  pinMode(pin, INPUT);                      // Set I/O pin to input
  duration = pulseIn(pin, HIGH, 25000);          // Measure echo time pulse from Ping)))
  return duration;                         // Return pulse duration
}

/*
 * Find an opening in system's 180-degree field of distance detection.
 * Returns:    Distance measurement dictated by the scalar
 * To-do:      Clean up and modularize
 *           Incorporate constants
 */
int findOpening()
{

  int Ai;                               // Initial turret angle
  int Af;                               // Final turret angle
  int k = sequence[i];                      // Copy sequence[i] to local variable
  int ki = k;                           // Second copy of current sequence[i]
  int inc;                              // Increment/decrement variable
  int dt;                               // Time increment
  int repcnt = 0;                       // Repetitions count
  int sMin;                             // Minimum distance measurement

  // Increment or decrement depending on where turret is pointing
  if(k <= 5)
  {
    inc = 1;
  }
  else
  {
    inc = -1;
  }

  // Rotate turret until an opening becomes visible.  If it reaches servo limit,
  // turn back to first angle of detection and try scanning toward other servo limit.
```

```
// If still no opening, rotate robot 90 degrees and try again.
do
{
  repcnt++;                                  // Increment repetition count
  if(repcnt > ((sizeof(sequence) / sizeof(int)))*2)// If no opening after two scans
  {
   maneuver(-200, -200, 100);                // Back up, turn, and stop to try again
    maneuver(-200, 200, 90*6);
    maneuver(0, 0, 1);
  }
  k += inc;                                  // Increment/decrement k
  if(k == -1)                                // Change inc/dec value if limit reached
  {
    k = ki;
    inc = -inc;
    dt = 250;                                // Turret will take longer to get there

  }
  if(k == 11)
  {
    k = ki;
    inc = -inc;
    dt = 250;

  }
//repcnt = 0;
  // Look for index of next turret position
  i = findIn(k, sequence, sizeof(sequence)/sizeof(int));

  // Set turret to next position
  int theta = sequence[i] * degreesTurret;
  turret(theta);                             // Position turret
  delay(dt);                                 // Wait for it to get there
  dt = 100;                                  // Reset for small increment turret movement

  cm[i] = cmDistance();                      // Take Ping))) measurement


}
while(cm[i] < 20);                           // Keep checking to edge of obstacle

sMin = 1000;                                 // Initialize minimum distance to impossibly large value
for(int t = 0; t <= 10; t++)
{
```

```
  if(sMin > cm[t]) sMin = cm[t];               // Use sMin to track smallest distance
 }
 if(sMin < 6)                                 // If less than 6 cm, back up a little
 {
  maneuver(-200, -200, 350);
  k = -1;                                     // Get turret ready to start over
 }
maneuver(0,0);

 // Keep rotating turret until another obstacle that's under 30 cm is detected or the turret
 // reaches the servo limit.  Keep track of the maximum distance and the angle when this
portion
 // of the scan started and stopped.

 Ai = sequence[i];                            // Save initial angle when obstacle disappeared from view

 k = sequence[i];                             // Make a copy of the turret position again

 int aMax = -2;                               // Initialize maximum distance measurements to impossibly
small values
 int cmMax = -2;

 do                                           // Loop for scan
 {
  k += inc;                                   // Inc/dec turret position
  // Look up index for turret position
  i = findIn(k, sequence, sizeof(sequence)/sizeof(int));
  int theta = sequence[i] * degreesTurret;        // Position turret
  turret(theta);
  delay(100);

  cm[i] = cmDistance();                       // Measure distance

  if(cm[i]>cmMax)                             // Keep track of max distance and angle(max distance)
  {
   cmMax = cm[i];
   aMax = sequence[i];
  }
 }
 while((cm[i] > 20)&&(sequence[i]!=0)&&(sequence[i]!=10));
 // Keep repeating while the distance measurement > 30 cm, and the turret is not near its
 // mechanical limits.

 Af = sequence[i];                            // Record final turret position
```

```
  int A = Ai + ((Af-Ai)/2);                    // Calculate middle of opening

  // Set turret index for straight ahead in prep for turn.
  i = findIn(5, sequence, sizeof(sequence)/sizeof(int));
  int theta = sequence[i] * degreesTurret;
  turret(theta);

  if(sMin < 7)                                 // Turn further if too close
  {
    if (A < aMax) return aMax; else return A;
  }
  else
  {
    if (A < aMax) return A; else return aMax;
  }
}

/*
 * Finds the first instance of a value in an array.
 * Parameters: value to search for
 *           array[] to search in
 *           elements - number of elements to search
 * Returns:    index where the matching element was found
 */
int findIn(int value, int array[], int elements)
{
  for(int i = 0; i < elements; i++)
  {
    if(value == array[i]) return i;
  }
  return -1;
}

//This part of the code essentially controls the line follower, based on input from QTI sensors.

void setupQTI()
{

}

void loopQTI()
{
  DDRD |= B11110000;                    // Set direction of Arduino pins D4-D7 as OUTPUT
  PORTD |= B11110000;                    // Set level of Arduino pins D4-D7 to HIGH
```

```
delayMicroseconds(230);               // Short delay to allow capacitor charge in QTI module
DDRD &= B00001111;                    // Set direction of pins D4-D7 as INPUT
PORTD &= B00001111;                   // Set level of pins D4-D7 to LOW
delayMicroseconds(230);               // Short delay
int pins = PIND;                    // Get values of pins D0-D7
pins >>= 4;                         // Drop off first four bits of the port; keep only pins D4-D7
int starttime ;
int endtime ;
int loopcount ;



int vR, vL;
switch(pins)
{
  case B1110:
    vR = -50;                       // -100 to 100 indicate course correction values
    vL = 100;                       // -100: full reverse; 0=stopped; 100=full forward
    break;                          // right one sees black , turns clockwise
 case B1111:
    starttime = millis();
    endtime = starttime;
    while ((endtime - starttime) <=900) // do this loop for up to 1000mS
      {
        vR = 100;
        vL  = 100;
       // back up for a few seconds
      loopcount = loopcount+1;
      endtime = millis();
      }
    vL = 100;
    vR = -50;
      break;
  case B0110:

   vR = -100;                       // when they all see black
   vL = -100;

      break;
  case B0111:
    vR = 100;
    vL = -50;      // when the left one sees black and the right one sees white , turns ccw
    break;
}
```

```
  servoLeftPin.writeMicroseconds(1500 + vR);     // Steer robot to recenter it over the line
  servoRightPin.writeMicroseconds(1500 - vL);

  delay(200);                          // Delay for 50 milliseconds (1/20 second)
}
```

**Sonar Code (separate)**

```
#include <Servo.h>                      // Include servo library

Servo servoLeft;                       // Servo object instances
Servo servoRight;
Servo servoTurret;

const int servoLeftPin = 12;                  // I/O Pin constants
const int servoRightPin = 13;
const int servoTurretPin = 11;
const int pingPin = 10;


const int msPerTurnDegree = 6;                // For maneuvers
const int tooCloseCm = 20;                   // For distance decisions
const int bumpedCm = 6;
int ccwLim = 1400;                          // For turret servo control
int rtAngle = 900;
const int usTocm = 29;                       // Ping))) conversion constant

// Sequence of turret positions.
int sequence[] = {0, 2, 4, 6, 8, 10, 9, 7, 5, 3, 1};

// Declare array to store that many cm distance elements using pre-calculated
// number of elements in array.
const int elements = sizeof(sequence);
int cm[elements];

// Pre-calculate degrees per adjustment of turret servo.
const int degreesTurret = 180/(sizeof(sequence)/sizeof(int)-1);

int i = -1;                            // Global index
int sign = 1;                            // Sign of increments
int theta = -degreesTurret;                  // Set up initial turret angle
```

```
void setup()                          // Built-in initialization block
{


  Serial.begin(9600);                 // Open serial connection

  servoLeft.attach(servoLeftPin);     // Attach left signal to pin 13
  servoRight.attach(servoRightPin);   // Attach right signal to pin 12
  servoTurret.attach(servoTurretPin); // Attach turret signal to pin 12
  maneuver(0, 0, 1000);               // Stay still for 1 second
  turret(0);                          // Set turret to 0 degrees
}

void loop()                           // Main loop auto-repeats
{
  maneuver(200, 200);                 // Go full speed forward UFN
  i++;                                // Increment turret index

  // Advance turret servo to next position in sequence and wait for it to get there.
  theta = sequence[i] * degreesTurret;
  turret(theta);
  delay(100);

  cm[i] = cmDistance();               // Measure cm from this turret angle

  // If object in +/- 36 degrees from center is within tooCloseCm threshold...
  if ((sequence[i]>=3) && (sequence[i]<=7) && (cm[i] < tooCloseCm))
  {
    maneuver(0, 0);                   // Stop moving
    int theta = findOpening();        // Get opening (in terms of sequence element
    theta *= degreesTurret;           // Convert sequence element to degree angle

    // Convert degree angle to time the BOE Shield-Bot wheels will have to turn to face
    // direction of turret.
    int turnAngleTime = (90 - theta) * msPerTurnDegree;

    if(turnAngleTime < 0)             // If negative turning angle,
    {
      maneuver(-200, 200, -turnAngleTime);  // then rotate CCW for turningAngleTime ms
    }
    else                              // If positive turning angle,
    {
      maneuver(200, -200, turnAngleTime);   // then rotate CW for turningAngleTime ms
    }
```

```
   maneuver(200, 200);                    // Start going forward again
  }

  if(i == 10)                            // If turret at max, go back to zero
  {
    i = -1;
  }
}

/*
 * Control BOE Shield-Bot servo direction, speed, set and forget version.
 * Parameters: speedLeft - left servo speed
 *             speedRight - right servo speed
 *             Backward  Linear  Stop  Linear   Forward
 *             -200      -100.....0.....100      200
 */
void maneuver(int speedLeft, int speedRight)
{
  // Call maneuver with just 1 ms blocking; servos will keep going indefinitely.
  maneuver(speedLeft, speedRight, 1);
}

/*
 * Control BOE Shield-Bot servo direction, speed and maneuver duration.
 * Parameters: speedLeft - left servo speed
 *             speedRight - right servo speed
 *             Backward  Linear  Stop  Linear   Forward
 *             -200      -100.....0.....100      200
 *             msTime - time to block code execution before another maneuver
 * Source:     http://learn.parallax.com/ManeuverFunction
 */
void maneuver(int speedLeft, int speedRight, int msTime)
{
  servoLeft.writeMicroseconds(1500 + speedLeft);   // Set Left servo speed
  servoRight.writeMicroseconds(1500 - speedRight); // Set right servo speed
  if(msTime==-1)                         // if msTime = -1
  {
    servoLeft.detach();                    // Stop servo signals
    servoRight.detach();
  }
  delay(msTime);                         // Delay for msTime
}

/*
```

```
 * Position the horn of a Parallax Standard Servo
 * Parameter: degreeVal in a range from 90 to -90 degrees.
 */
void turret(int degreeVal)
{
  servoTurret.writeMicroseconds(ccwLim - rtAngle + (degreeVal * 10));
}

/*
 * Get cm distance measurment from Ping Ultrasonic Distance Sensor
 * Returns: distance measurement in cm.
 */
int cmDistance()
{
  int distance = 0;                       // Initialize distance to zero
  do                                      // Loop in case of zero measurement
  {
    int us = ping(pingPin);               // Get Ping))) microsecond measurement
    distance = convert(us, usTocm);       // Convert to cm measurement
    delay(3);                             // Pause before retry (if needed)
  }
  while(distance == 0);
  return distance;                        // Return distance measurement
}

/*
 * Converts microsecond Ping))) round trip measurement to a useful value.
 * Parameters: us - microsecond value from Ping))) echo time measurement.
 *             scalar - 29 for us to cm, or 74 for us to in.
 * Returns:    distance measurement dictated by the scalar.
 */
int convert(int us, int scalar)
{
  return us / scalar / 2;                 // Echo round trip time -> cm
}

/*
 * Initiate and capture Ping))) Ultrasonic Distance Sensor's round trip echo time.
 * Parameter: pin - Digital I/O pin connected to Ping)))
 * Returns:   duration - The echo time in microseconds
 * Source:    Ping by David A. Mellis, located in File -> Examples -> Sensors
 * To-Do:     Double check timing against datasheet
 */
long ping(int pin)
```

```
{
  long duration;                          // Variables for calculating distance
  pinMode(pin, OUTPUT);                     // I/O pin -> output
  digitalWrite(pin, LOW);                  // Start low
  delayMicroseconds(2);                     // Stay low for 2 us
  digitalWrite(pin, HIGH);                 // Send 5 us high pulse
  delayMicroseconds(5);
  digitalWrite(pin, LOW);
  pinMode(pin, INPUT);                      // Set I/O pin to input
  duration = pulseIn(pin, HIGH, 25000);          // Measure echo time pulse from Ping)))
  return duration;                         // Return pulse duration
}

/*
 * Find an opening in system's 180-degree field of distance detection.
 * Returns:    Distance measurement dictated by the scalar
 * To-do:      Clean up and modularize
 *           Incorporate constants
 */
int findOpening()
{

  int Ai;                               // Initial turret angle
  int Af;                               // Final turret angle
  int k = sequence[i];                      // Copy sequence[i] to local variable
  int ki = k;                           // Second copy of current sequence[i]
  int inc;                              // Increment/decrement variable
  int dt;                               // Time increment
  int repcnt = 0;                       // Repetitions count
  int sMin;                             // Minimum distance measurement

  // Increment or decrement depending on where turret is pointing
  if(k <= 5)
  {
    inc = 1;
  }
  else
  {
    inc = -1;
  }

  // Rotate turret until an opening becomes visible.  If it reaches servo limit,
  // turn back to first angle of detection and try scanning toward other servo limit.
  // If still no opening, rotate robot 90 degrees and try again.
```

```
do
{
  repcnt++;                              // Increment repetition count
  if(repcnt > ((sizeof(sequence) / sizeof(int)))*2)// If no opening after two scans
  {
    maneuver(-200, -200, 100);                // Back up, turn, and stop to try again
    maneuver(-200, 200, 90*6);
    maneuver(0, 0, 1);
  }
  k += inc;                             // Increment/decrement k
  if(k == -1)                           // Change inc/dec value if limit reached
  {
    k = ki;
    inc = -inc;
    dt = 250;                           // Turret will take longer to get there
  }
  if(k == 11)
  {
    k = ki;
    inc = -inc;
    dt = 250;
  }
  // Look for index of next turret position
  i = findIn(k, sequence, sizeof(sequence)/sizeof(int));

  // Set turret to next position
  int theta = sequence[i] * degreesTurret;
  turret(theta);                          // Position turret
  delay(dt);                            // Wait for it to get there
  dt = 100;                             // Reset for small increment turret movement

  cm[i] = cmDistance();                     // Take Ping))) measurement
}
while(cm[i] < 20);                        // Keep checking to edge of obstacle

sMin = 1000;                            // Initialize minimum distance to impossibly large value
for(int t = 0; t <= 10; t++)
{
  if(sMin > cm[t]) sMin = cm[t];                // Use sMin to track smallest distance
}
if(sMin < 6)                            // If less than 6 cm, back up a little
{
  maneuver(-200, -200, 350);
  k = -1;                               // Get turret ready to start over
```

```
 }
 maneuver(0, 0);                          // Stay still indefinitely

 // Keep rotating turret until another obstacle that's under 20 cm is detected or the turret
 // reaches the servo limit.  Keep track of the maximum distance and the angle when this
portion
 // of the scan started and stopped.

 Ai = sequence[i];                        // Save initial angle when obstacle disappeared from view

 k = sequence[i];                         // Make a copy of the turret position again

 int aMax = -2;                           // Initialize maximum distance measurements to impossibly
small values
 int cmMax = -2;

 do                                       // Loop for scan
 {
  k += inc;                               // Inc/dec turret position
  // Look up index for turret position
  i = findIn(k, sequence, sizeof(sequence)/sizeof(int));
  int theta = sequence[i] * degreesTurret;        // Position turret
  turret(theta);
  delay(100);

  cm[i] = cmDistance();                   // Measure distance

  if(cm[i]>cmMax)                         // Keep track of max distance and angle(max distance)
  {
   cmMax = cm[i];
   aMax = sequence[i];
  }
 }
 while((cm[i] > 20)&&(sequence[i]!=0)&&(sequence[i]!=10));
 // Keep repeating while the distance measurement > 20 cm, and the turret is not near its
 // mechanical limits.

 Af = sequence[i];                        // Record final turret position
 int A = Ai + ((Af-Ai)/2);                // Calculate middle of opening

 // Set turret index for straight ahead in prep for turn.
 i = findIn(5, sequence, sizeof(sequence)/sizeof(int));
 int theta = sequence[i] * degreesTurret;
 turret(theta);
```

```
  if(sMin < 7)                          // Turn further if too close
  {
    if (A < aMax) return aMax; else return A;
  }
  else
  {
    if (A < aMax) return A; else return aMax;
  }
}

/*
 * Finds the first instance of a value in an array.
 * Parameters: value to search for
 *           array[] to search in
 *           elements - number of elements to search
 * Returns:    index where the matching element was found
 */
int findIn(int value, int array[], int elements)
{
  for(int i = 0; i < elements; i++)
  {
    if(value == array[i]) return i;
  }
  return -1;
}
```