Project Description:
This is the code used for the demonstration with all the original code maintained, but with some extra docs.

PROBLEM:
To tag as many lights as possible while avoiding tall obstacles. If a tall obstacle is touched by the robot all points are lost; whereas, a short obstacle contact has no penalty.

SOLUTION:
Our team's robot had two rear driving wheels, one front steering wheel, and a rotating range sensor and light sensor providing sensory coverage for the front half of the robot. The back half of the robot had no sensory coverage. We called the rotating range and light sensors the radar (because it rotates). Other sensors include: one fixed range sensor in front, two front-bumper button sensors, and two wheel guard lever sensors (one for each wheel).

The robot was purely reactive to its environment, and did not have any memory of the past. Memory of the past and other modifications to this code were made, but there was not enough time to tweak the code for the robot before the demo to fully incorporate the new improvements. It has something to say about the tweaking of the code: it should probably be done by a simple type of AI rather than a human programmer. For robot programming, a higher level of programming may be better.

Anyways, the code just makes the robot turn the radar back and forth and go towards the brightest light found, avoiding any obstacles detected. There is not very much code and it is not complicated, due to the amount of tweaking that has to be done to make the robot work. But I guess my code is always a little bit sloppy...
* define the ports used
 */
#define STEER_SERVO servo5
#define RADAR_SERVO servo0

#define LEFT_MOTOR_PORT 3
#define RIGHT_MOTOR_PORT 1

#define FIXED_RANGESENSOR_PORT 17
#define RADAR_RANGE_SENSOR_PORT 16
#define RADAR_LIGHT_SENSOR_PORT 2

#define FRONT_RIGHT_TOUCH_PORT 7
#define FRONT_LEFT_TOUCH_PORT 9

#define LEFT_WHEEL_GUARD_LEVER 12
#define RIGHT_WHEEL_GUARD_LEVER 11

/*
 * The maximum ET value, above which there is
 * to be considered an obstacle in front of the sensor.
 */
#define RADAR_RANGE_AVOID_LEVEL 80
#define FIXED_RANGE_AVOID_LEVEL 50

/*
 * The light level at which the robot considers itself
 * close enough to the light to not even worry about
 * obstacles, and make a charge at the light.
 */
#define LIGHT_PROXIMITY_LEVEL 3

/*
 * The motor power to use when the maximum light level
 * detected is less that 'LIGHT_PROXIMITY_LEVEL'. When
 * that level of light is reached, the robot makes a
 * charge towards the light, disregarding all obstacles.
 */
#define CHARGE_LIGHT_MOTOR_POWER 50

/*
 * The maximum motor power to use when in normal operation.
 */
#define MAX_MOTOR_POWER 60

/*
 * The minimum slices free of obstacles needed
 * for passing between two or more obstacles.
 */
#define MIN_PASS_SLICE_COUNT 12

/*
 * steering servo values
 * servo range: 100 - 3900
 */
#define STEER_CENTER 2150
#define STEER_FULL_RIGHT 1000
#define STEER_FULL_LEFT 3200
#define STEER_FULL DEGREES 80.0

/*
 * radar servo values
 */
#define RADAR CENTER 1900
#define RADAR Full RIGHT 3300
#define RADAR Full LEFT 800
#define RADAR Full DEGREES 90.0

/*
 * define sensor input slices
 * (80.0 * 2 / 10) = 16
 * (RADAR Full DEGREES * 2 / RADAR SCAN SLICE ANGLE) =
 * RADAR SCAN SLICE COUNT,
 * angle = (slice - RADAR SCAN SLICE COUNT/2) * RADAR SCAN SLICE ANGLE
 */
#define RADAR SCAN SLICE COUNT 18
#define RADAR SCAN SLICE ANGLE 10

/*
 * define how many times the radar sensors
 * are polled while waiting for the radar
 * servo to move the radar to the next slice.
 */
#define RADAR SCAN DELAY BY POLLING COUNT 5

/*
 * define the maximum turning angle of the
 * steering and radar from straight forward.
 */
#define RADAR TURN DEGREES 90
#define STEER TURN DEGREES 55

/*
 * define radar modes.
 */
#define RADAR OFF 0
#define RADAR SCAN NORMAL 1
#define RADAR SCAN LEFT TO RIGHT 2
#define RADAR SCAN RIGHT TO LEFT 3

/*
 * The current radar mode...
 */
int radarMode = RADAR OFF;

/*
 * Used to cause the radar to make a full sweep of
 * the area covered. When this flag is set, the
 * radar process then sets it to false after the radar
 * has gone at least one full rotation from either
 * left-to-right or right-to-left.
 */
int radarFullSweepFlag = 0;
/*
 * The slices used for radar input.
 */
int rangeSlices[RADAR_SCAN_SLICE_COUNT];
int lightSlices[RADAR_SCAN_SLICE_COUNT];

/*/ 
* Causes the steering servo to go towards the specified
* angle, where -90 degrees would be all left, 0 degrees
* would be center, and +90 degrees would be all right.
*/
void steerTo(int degrees) {
  if (degrees == 0) {
    STEER_SERVO = STEER_CENTER;
  } else if (degrees < 0) {
    STEER_SERVO = STEER_CENTER + (int)( ((float)(STEER_FULL_LEFT - STEER_CENTER)) * ((float)(-degrees)) / STEER_FULL_DEGREES));
  } else {
    STEER_SERVO = STEER_CENTER + (int)( ((float)(STEER_FULL_RIGHT - STEER_CENTER)) * ((float)(degrees)) / STEER_FULL_DEGREES));
  }
}

/*/ 
* Causes the radar servo to go towards the specified
* angle, where -90 degrees would be all left, 0 degrees
* would be center, and +90 degrees would be all right.
*/
void radarTo(int degrees) {
  if (degrees == 0) {
    RADAR_SERVO = RADAR_CENTER;
  } else if (degrees < 0) {
    RADAR_SERVO = RADAR_CENTER + (int)( ((float)(RADAR_FULL_LEFT - RADAR_CENTER)) * ((float)(-degrees)) / RADAR_FULL_DEGREES));
  } else {
    RADAR_SERVO = RADAR_CENTER + (int)( ((float)(RADAR_FULL_RIGHT - RADAR_CENTER)) * ((float)(degrees)) / RADAR_FULL_DEGREES));
  }
}

/*/ 
* Returns the value of the range sensor on the radar. This
* function is here because originally both range sensors were
* on the radar, rather than having one fixed in front as now.
*/
int getRadarRange() {
  return analog(RADAR_RANGE_SENSOR_PORT);
}

/*/ 
* Returns the value of the light sensor on the radar. This
* function is here because originally there were going to be
* more than one light sensor on the radar, but in the end only
* one was used.
*/
int getRadarLight() {
  return analog(RADAR_LIGHT_SENSOR_PORT);
*/
* Function defining the radar process, or thread,
* or whatever they wanna call it.
*/
void radar(void) {
  while (1) {

    // the current degrees at which the radar is turned... supposedly...
    int degrees;
    while (radarMode == (int)RADAR_SCAN_NORMAL || radarMode ==
          (int)RADAR_SCAN_LEFT_TO_RIGHT || radarMode ==
          (int)RADAR_SCAN_RIGHT_TO_LEFT) {

      int slice = 0;
      int pollcount;
      int maxrange;
      int maxlight;
      int foo;
      int setRadarFullSweepFlagToFalse = radarFullSweepFlag;

      degrees = -(int)RADAR_TURN_DEGREES;

      setRadarFullSweepFlagToFalse = radarFullSweepFlag;

      if (radarMode == RADAR_SCAN_LEFT_TO_RIGHT) {
        radarTo(degrees);
        sleep(0.5);
      }

      while (degrees < (int)RADAR_TURN_DEGREES && (radarMode ==
                     (int)RADAR_SCAN_NORMAL || radarMode == (int)RADAR_SCAN_LEFT_TO_RIGHT)) {

        radarTo(degrees);
        slice = (int)(((float)(degrees +
                           (int)RADAR_FULL DEGREES))/((float)RADAR_SCAN_SLICE_ANGLE));

        maxrange = 0;
        maxlight = 255;
        pollcount = 0;
        while (pollcount < RADAR_SCAN_DELAY_BY_POLLING_COUNT) {
          foo = getRadarRange();
          if (foo > maxrange) {
            maxrange = foo;
          }
          foo = getRadarLight();
          if (foo < maxlight) {
            maxlight = foo;
          }
          pollcount++;
        }

        rangeSlices[slice] = maxrange;
        lightSlices[slice] = maxlight;
        degrees += RADAR_SCAN_SLICE ANGLE;
    }
  }
}
while (degrees < (int)RADAR_TURN_DEGREES) {
    degrees += RADAR_SCAN_SLICE_ANGLE;
}

if (radarMode == RADAR_SCAN_RIGHT_TO_LEFT) {
    radarTo(degrees);
    sleep(0.5);
}

if (setRadarFullSweepFlagToFalse) {
    radarFullSweepFlag = 0;
}

sleep(0.01);

setRadarFullSweepFlagToFalse = radarFullSweepFlag;

while (degrees > -(int)RADAR_TURN_DEGREES && (radarMode == (int)RADAR_SCAN_NORMAL || radarMode == (int)RADAR_SCAN_RIGHT_TO_LEFT)) {
    degrees -= RADAR_SCAN_SLICE_ANGLE;
    radarTo(degrees);
    slice = (int)((float)(degrees + (int)RADAR_FULL_DEGREES))/(float)RADAR_SCAN_SLICE_ANGLE);
    maxrange = 0;
    maxlight = 255;
    pollcount = 0;
    while (pollcount < RADAR_SCAN_DELAY_BY_POLLING_COUNT) {
        foo = getRadarRange();
        if (foo > maxrange) {
            maxrange = foo;
        }
        foo = getRadarLight();
        if (foo < maxlight) {
            maxlight = foo;
        }
        pollcount++;
    }
    rangeSlices[slice] = maxrange;
    lightSlices[slice] = maxlight;
}
if (setRadarFullSweepFlagToFalse) {
    radarFullSweepFlag = 0;
}

}/*
 * Function that makes the radar do a full sweep
 * before returning.
 */
void doFullRadarScan() { // set flag for radar process to look at...
    radarFullSweepFlag = 1;
}
/ wait for radarFullSweepFlag to be set
 / to zero by the radar process...
 while (radarFullSweepFlag);
 }

 /*
 * Returns whether there is an obstacle at, or close to, the
 * specified slice. This function searches within
 MIN_PASS_SLICE_COUNT/2
 * slices of the specified slice to see if there is an obstacle,
 * and returns true if there is an obstacle within that range,
 * false otherwise......
 */
 int isObstacleAt(int slice) {
     int start = slice - MIN_PASS_SLICE_COUNT/2;
     int end = start + MIN_PASS_SLICE_COUNT;
     int i;

     if (start < 0) {
         start = 0;
     }
     if (end > RADAR_SCAN_SLICE_COUNT) {
         end = RADAR_SCAN_SLICE_COUNT;
     }

     i = start;
     while (i < end) {
         if (rangeSlices[i] > RADAR_RANGE_AVOID_LEVEL) {  
             return 1;
         }
         i++;
     }
     return 0;
 }

 /*
 * Returns the closest slice that is free of obstacles
 * from the specified target slice.
 */
 int findObstacleFreeSlice(int targetSlice) {
     int diff = 0;
     int slice;
     while (diff < RADAR_SCAN_SLICE_COUNT) {
         // check left of target...
         slice = targetSlice - diff;
         if (slice > 0) {
             // check if no obstacles detected...
             if (!isObstacleAt(slice)) {
                 break;
             }
         }
     }
     // check right of light...
     slice = targetSlice + diff;
     if (slice < RADAR_SCAN_SLICE_COUNT) {
         // check if no obstacles detected...
if (!isObstacleAt(slice)) {
    break;
}

diff++;
return slice;

void steerToSlice(int slice) {
    int turnAngle = (slice * RADAR_SCAN_SLICE_ANGLE -
    (int)RADAR_FULL_DEGREES);
    if (turnAngle < -STEER_TURN_DEGREES) {
        turnAngle = -STEER_TURN_DEGREES;
    }
    if (turnAngle > STEER_TURN_DEGREES) {
        turnAngle = STEER_TURN_DEGREES;
    }
    steerTo(turnAngle);
}

void main(void) {
    // useless var...
    int foo;

    int frontRange = 0;
    int middleSlice = RADAR_SCAN_SLICE_COUNT/2 - 1;

    // initialize stuff...
    STEER_SERVO = STEER_CENTER;
    RADAR_SERVO = RADAR_CENTER;

    for (foo = 0; foo < RADAR_SCAN_SLICE_COUNT; foo++) {
        rangeSlices[foo] = 255;
        lightSlices[foo] = 255;
    }

    init_expbd_servos(1);

    // servoTest();
    // etTest();
    // touchTest();
    // return;

    start_process(radar());
printf("Press Start\n");
while (!start_button());

radarMode = RADAR_SCAN_NORMAL;

// do full sweep of radar to initialize stuff...
do();
doFullRadarScan();

// do main program feedback loop...
while (!stop_button()) {

    int slice = 0;
    int maxLight = 255;
    int maxLightSlice = 0;
    int turnAngle;

    // check touch sensors and backup if touching anything...
    if (digital(FRONT_LEFT_TOUCH_PORT) == 1 ||
        digital(FRONT_RIGHT_TOUCH_PORT) == 1 ||
        digital(LEFT_WHEEL_GUARD_LEVER) == 1 ||
        digital(RIGHT_WHEEL_GUARD_LEVER) == 1) {
        steerTo(0);
        motor(LEFT_MOTOR_PORT, -20);
        motor(RIGHT_MOTOR_PORT, -20);
        sleep(0.5);
        ao();
        radarMode = RADAR_SCAN_NORMAL;
doFullRadarScan();
    }

    // find the slice with the maximum light value...
    while (slice < (int)RADAR_SCAN_SLICE_COUNT) {
        if (lightSlices[slice] < maxLight) {
            maxLight = lightSlices[slice];
            maxLightSlice = slice;
        }
        slice++;
    }

    // this sets the maxLightSlice to straight ahead... for testing...
    //maxLightSlice = RADAR_SCAN_SLICE_COUNT/2 - 1;

    // check if light is close enough to make a charge at it...
    if (maxLight < LIGHT_PROXIMITY_LEVEL) {
        // charge the light bulb...
        steerToSlice(maxLightSlice);
        motor(LEFT_MOTOR_PORT, CHARGE_LIGHT_MOTOR_POWER);
        motor(RIGHT_MOTOR_PORT, CHARGE_LIGHT_MOTOR_POWER);
    } else {
        int shouldBackup = 0;

        // read the fixed et sensor...
        frontRange = analog(FIXED_RANGE_SENSOR_PORT);
if (frontRange > FIXED_RANGE_AVOID_LEVEL) {
    // wait for confirmation of obstacle from radar...
    ao();
    doFullRadarScan();

    // check radar scan slices...
    if (rangeSlices[middleSlice] > RADAR_RANGE_AVOID_LEVEL
        || rangeSlices[middleSlice + 1] > RADAR_RANGE_AVOID_LEVEL) {
        shouldBackup = 1;
    }
}

if (shouldBackup) {
    // obstacle close by
    // stop...
    ao();
    // now find best obstacle free window...
    slice = findObstacleFreeSlice(maxLightSlice);
    turnAngle = (slice * RADAR_SCAN_SLICE_ANGLE -
                 (int)RADAR_FULL_DEGREES);

    // this helps debug the light seeking problem
    printf("L:%d,R:%d,A:%d,F:%d\n", maxLight, rangeSlices[maxLightSlice],
           (maxLightSlice * RADAR_SCAN_SLICE_ANGLE -
            (int)RADAR_FULL_DEGREES), frontRange);
    if (turnAngle < -STEER_TURN_DEGREES) {
        turnAngle = -STEER_TURN_DEGREES;
    }
    if (turnAngle > STEER_TURN_DEGREES) {
        turnAngle = STEER_TURN_DEGREES;
    }
    // backup away from target slice......
    steerTo(-turnAngle);
    motor(LEFT_MOTOR_PORT, -20);
    motor(RIGHT_MOTOR_PORT, -20);
    sleep(0.7);
    ao();
} else {
    // no obstacle close in front, so doesn't need ot backup...
    int leftMotorPower = 0;
    int rightMotorPower = 0;

    // nothing in front, just cruise towards most open window...
    slice = findObstacleFreeSlice(maxLightSlice);
    // at this point, slice is the place where we want to go...
    turnAngle = (slice * RADAR_SCAN_SLICE_ANGLE -
                 (int)RADAR_FULL_DEGREES);

    // this helps debug the light seeking problem
    printf("L:%d,R:%d,A:%d,F:%d\n", maxLight, rangeSlices[maxLightSlice],
           (maxLightSlice * RADAR_SCAN_SLICE_ANGLE -
            (int)RADAR_FULL_DEGREES), frontRange);
if (turnAngle < -STEER_TURN_DEGREES) {
    turnAngle = -STEER_TURN_DEGREES;
}
if (turnAngle > STEER_TURN_DEGREES) {
    turnAngle = STEER_TURN_DEGREES;
}
steerTo(turnAngle);

if (turnAngle > 35) {
    leftMotorPower = 40;
    rightMotorPower = (int)(40.0 * (float)(1.0 - (float)turnAngle/90.0));
    radarMode = RADAR_SCAN_RIGHT_TO_LEFT;
} else if (turnAngle < -35) {
    rightMotorPower = 40;
    leftMotorPower = (int)(40.0 * (float)(1.0 + (float)turnAngle/90.0));
    radarMode = RADAR_SCAN_LEFT_TO_RIGHT;
} else {
    leftMotorPower = rightMotorPower = 10;
    radarMode = RADAR_SCAN_NORMAL;
}

foo = random(80);
leftMotorPower += foo;
rightMotorPower += foo;
if (leftMotorPower > MAX_MOTOR_POWER) {
    leftMotorPower = MAX_MOTOR_POWER;
}
if (rightMotorPower > MAX_MOTOR_POWER) {
    rightMotorPower = MAX_MOTOR_POWER;
}
// turn motor on random power to overclimb rocks
// when tracking light and avoiding obstacles...
motor(LEFT_MOTOR_PORT, leftMotorPower);
motor(RIGHT_MOTOR_PORT, rightMotorPower);

// turn off motors after stop has been pressed...
motor(LEFT_MOTOR_PORT, 0);
motor(RIGHT_MOTOR_PORT, 0);
printf("Foo\n");
init_expbd_servos(0);

/*
 * Tests the servos...
*/
void servoTest() {
    steerTo(0);
    radarTo(0);
sleep(1.0);
steerTo(-45);
radarTo(-45);
sleep(1.0);
steerTo(45);
radarTo(45);
sleep(1.0);
init_expbd_servos(0);
}

/*
 * Tests the range sensors...
 */
void etTest() {
    while (!stop_button()) {
        printf("r:%d, f:%d\n", analog(RADAR_RANGE_SENSOR_PORT),
               analog(FIXED_RANGE_SENSOR_PORT));
        sleep(0.1);
    }
}

/*
 * Tests the touch sensors...
 */
void touchTest() {
    while (!stop_button()) {
        printf("fl:%d, fr: %d, wl:%d, wr:%d\n",
               digital(FRONT_LEFT_TOUCH_PORT), digital(FRONT_RIGHT_TOUCH_PORT),
               digital(LEFT_WHEEL_GUARD_LEVER), digital(RIGHT_WHEEL_GUARD_LEVER));
        sleep(0.1);
    }
}