One-Armed Bandit:
A retrieval robot

By

Thomas R. Harrington

Table of Contents

Abstract

Bandit, a beacon retrieving robot, is made up of a TALRIK platform, EVBU M68HC11 with 32k of expanded memory, and a robotic arm manufactured by Lynxmotion, Inc. The sensors used on this robot are front/rear bump sensors, a collimated IR detector, and four general IR sensors. The behaviors are obstacle avoidance, collision avoidance, beacon acquire, and beacon capture. The bump sensors are used for collision detection in order to implement the collision avoidance behavior. The four IR sensors are involved in the obstacle avoidance and beacon acquire behaviors--the dual role of these sensors provides for less hardware and more complicated software. The collimated IR detector allows the robot to position itself in order to carry out the beacon capture behavior. All of these behaviors were implemented through *C* code compiled by *Interactive C*. Finally, *Bandit* uses two gear head motors (hacked servos) and wheels for propulsion.

Executive Summary

The robot *One-Armed Bandit* is a beacon-retrieving robot. It is a simple robot that satisfies the requirements of the Intelligent Machine Design Lab. These requirements include four sensors, as well as four integrated behaviors. The complete system designed and built allows the robot to wander around in its environment (searching for a beacon signal) and interact with the objects it encounters.

For this course, the four sensors were collision-avoidance bump sensors, obstacle-avoidance/beacon-acquire IR sensors, a beacon-capture collimated IR detector, and a beacon-capture robotic arm. The four behaviors are collision avoidance, obstacle avoidance, beacon acquire, and beacon capture. There are four IR sensors mounted on the mobile platform along with 10 micro-switches in order to implement the obstacle-avoidance/beacon-acquire and collision-avoidance behaviors. The robotic arm is mounted on a platform above the mobile platform and the collimated IR detector is mounted on the robotic arm's gripper.

Introduction

The objective of this project was to design a robot that would be able to retrieve a beacon. This report starts with a general overview of the robot's functions. It then proceeds to discuss the physical systems of the robot including the robotic arm used to pick up the beacon. From there, the sensors used on the robot are discussed and explained. Finally, the behaviors that the robot exhibits and the experiments performed are discussed.

Integrated System

The robot consists of two basic subsystems tied together by the M68HC11 microcontroller. These are the mobile platform and the robotic arm. The mobile platform consists of a TALRIK body with four IR detectors mounted on the top around ther perimeter in the front, back, left, and right positions. Underneath, there are a total eight IR LED's also mounted on the perimeter--two LED's are positioned below each IR detector. Also mounted on the underside is the battery pack, two casters (front and back), and two "hacked" servomotors each connected to one wheel. Mounted around the edge of the mobile platform is an acrylic bumper and 10 bump switches—five in front and five in back. The bump switches are use to detect collisions while the IR sensors are used to detect obstacles and the beacon.

The robotic arm functions as the retrieval system for this project by reaching out and grasping the beacon once the mobile platform is in range of the beacon. The arm consists of five servomotors, the Mini SSC II PCB servomotor controller, and a physical structure made of plastic PVC tubing and plastic foam board. The following list denotes the servomotor distribution to the joints of the arm:

- Shoulder—2 servo's
- Elbow—1 servo
- Wrist—1 servo
- Gripper—1 servo with pushrod

Figure 1 shows the arm mounted to a base. *Bandit's* arm is not mounted to this base, but rather to the mobile platform. The Mini SSC PCB is visible below the wrist joint and next to the shoulder joint. The pushrod is the cable seen connected to the gripper and disappearing behind the elbow joint.

Figure 1

The arm is designed and sold by Lynxmotion, Inc.

Actuation

Wheel Motors

Two motors were used to drive the robot. The motors are hacked MS410 Servos. The servos were hacked by removing the circuit board in them and cutting the stop on one of the gears. They were then mounted to the underside of the TALRIK board.

One problem encountered with running these motors at the same speed was that the robot would travel in a severe arc. After experimenting with different speeds, I found that the robot traveled close to straight by running both the motors at 30%.

Robotic Arm Controller

The robotic arm uses the Mini SSC II PCB servomotor controller to control its motion. Figure 2 depicts the Mini SSC II with the important components identified. An important modification was made to the controller by cutting the trace depicted. Since this application does not use a base servomotor, channel #0 (a spare channel) trace was cut and then tied to channel #1. This action alleviated the necessity of splicing together the two servos moving the shoulder joint. This combination of channels for the shoulder joint is illustrated along with the other joint control channels in Fig. 2.

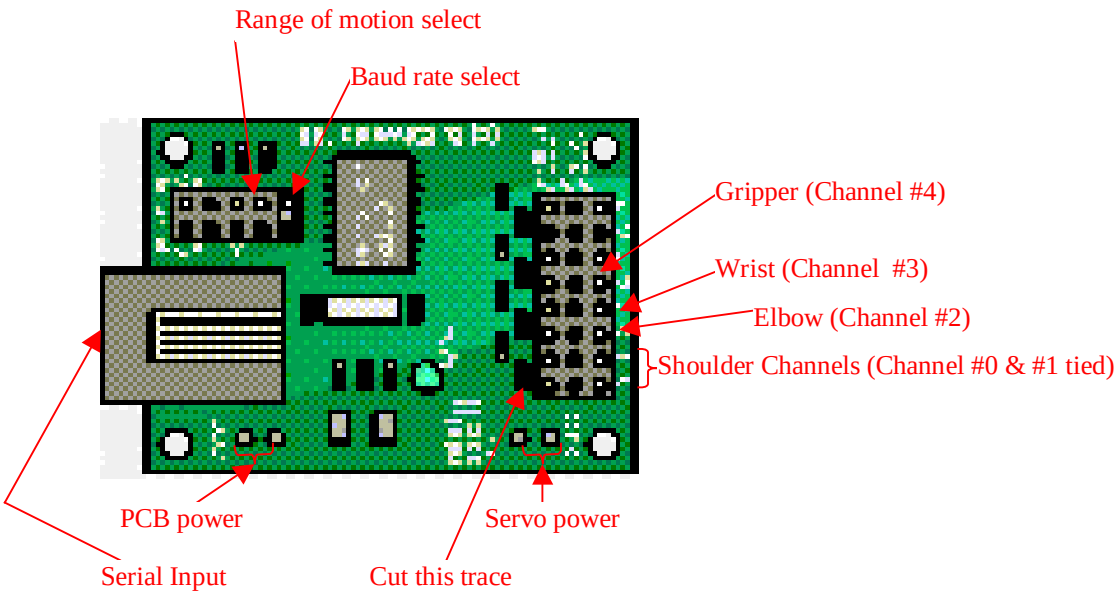

Figure 2

The following list defines the other functions depicted in Fig. 2:

- Range of motion select—Jumper on selects 90º ROM for servos (jumper off 180º)
- Baud rate select—Jumper on selects 9600 (*Bandit's* setting), jumper off selects 2400
- PCB power—9 V_{DC}

- Servo power— 4.8-6 V_{DC}
- Serial input—Mini SSC II accepts only input (Rx line).

The serial input accepts servo-position-control data in three bytes of data with the following format:

<sync. byte> <servo #> <position>

The <sync. byte> equals 255 decimal, the <servo #> equals the servo number to be moved, and the <position> equals the position where the servo is to be moved.

NOTE: these numbers are not in ASCII format, but byte format.

Sensors

Overview

The sensors on the robot consist of four IR detectors, eight IR LED's, 10 microswitches, and a collimated IR detector. The obstacle sensor is comprised of the four IR detectors along with the eight IR LED's. The collision sensor is made up of the 10 microswitches and the acrylic bumper. The acquire sensor consists of the same four IR detectors used in the obstacle sensor. The collimated IR detector along with the robotic arm constitutes the capture sensor.

Obstacle Sensor

On the robot there are four Sharp IR detectors facing outward. They are mounted on the TALRIK platform in the front, rear, left, and right. These sensors are hacked so that an analog reading can be taken from them. On the underside of the robot, two IR LED's are attached underneath each IR detector and are pointing 45 degrees from the center lines of the robot. The idea behind the sensor is that if there is an obstacle the IR from the LED's would be reflected off the object and into the IR detector, signaling that

there is an obstacle. These IR detectors were connected to the following A/D pins on the EVBU:

- Front—PE4
- Back—PE5
- Left—PE6
- Right—PE7

The LED's were connected to the ME11 digital output port.

Note: the acquire sensor uses these same IR detectors to detect a beacon signal and will be discussed in the Behaviors section.

Collision Sensor

This sensor consisted of a front bump sensor (PE0) and a back bump sensor (PE1)—each using five switches. Both of the sensors were constructed identically, so only one will be described in detail. The front bump sensor used a resistor ladder network of 10K, 50K, 100K, 150K, and 200K resistors. These resistors made a voltage divider circuit in which the output was connected to the EVBU's A/D. With each different combination of closing switches came different output voltages. With this knowledge the robot could tell which side of it collided with an object and react accordingly.

Capture Sensor

The collimated IR detector (PE3) along with the robotic arm constitutes this sensor. The IR detector is collimated with a 0.5" tube to give it a narrow range of vision. This detector is scanned horizontal and vertically to give until the beacon is right in front of it—thereby positioning the robotic arm precisely from picking up the beacon.

Behaviors

Obstacle Avoidance/Beacon Acquisition

The robot takes 35 samples per IRD to detect if there is an obstacle or a beacon signal. The reason for taking 35 samples is because the IR beacon signal is 8 Hz (period of approx. 120ms) and in order to ensure that an entire period is sampled, a 16 Hz sampling rate is required (approx. 35 samples). With these samples, the robot takes the maximum values and subtracts the minimum values—called delta.

The robot will avoid an obstacle if delta is below threshold1 and the max value of the IRD is above thershold2. Threshold1 would be zero ideally, but ranges between 0 and 3 in reality. Threshold2 is the value of an IRD with no beacon signal present and an obstacle placed 5" in front of the IRD. These threshold values were obtained experimentally.

The robot will follow the beacon if the delta is above threshold2. The robot zigzags to the beacon until an IRD registers at a saturation value, then the Beacon Capture behavior takes over.

Beacon Capture

Once the beacon is in range of the robot, the robot will spin in place scanning the collimated IR detector horizontally until the beacon is directly in front of it. Next, the robotic arm will scan the collimated IR detector vertically until it is at the same level as the beacon. Once positioned precisely the arm will implemented a pre-arranged set of motions to pick up the beacon.

Collision Avoidance

If the robot bumps into an object, the bump sensor will register a value telling the robot where it collided. Knowing this information, the robot will execute the best reaction to the collision event in order skirt the object.

Experimental Results

The servomotors in the joints of the robotic arm do not behave identically; furthermore, they all have different characteristic values. Table 1 shows the maximum and minimum position values for 90º and 180º range of motion.

Table 1

Joint	Range of Motion	Maximum Position	Minimum Position
Shoulder	90º	225	25
Elbow	90º	225	20
Wrist	90º	254	1
Gripper	90º	180	40
Shoulder	180º	180	40
Elbow	180º	175	70
Wrist	180º	250	25
Gripper	180º	170	90

These values are illustrated in the following bar graph (Figure 3). The robot only uses a 90-degree range of motion.

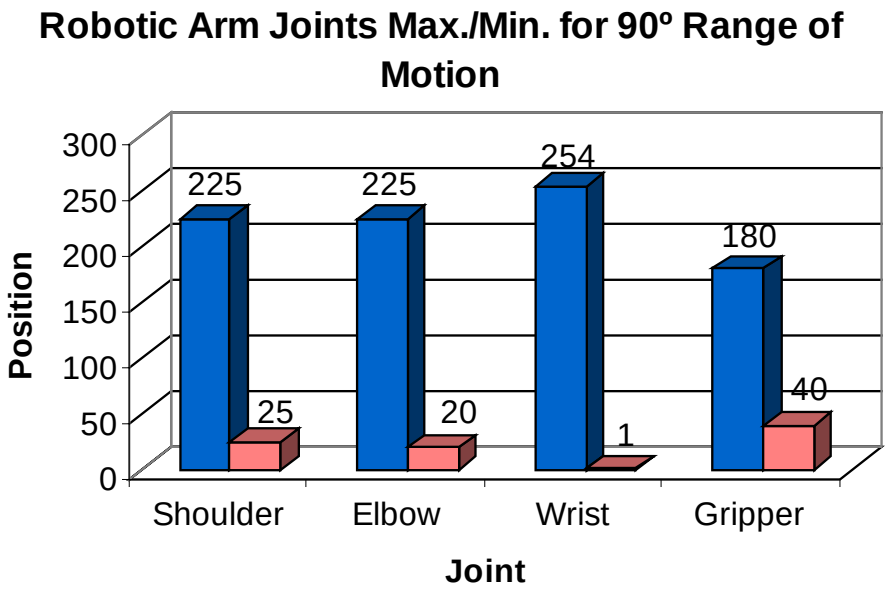

Figure 3

The purpose in using this data is to supply the M68HC11 with maximum and minimum values for joint positions in order not to hurt the arm.

Conclusion

This robot performed collision avoidance and obstacle avoidance well, but did not behave very well when beacon acquisition was introduced. There was irreconcilable conflict between obstacle avoidance and acquisition. If I had known earlier that there was 32 kHz IRD available, I would have used these for beacon acquisition in order to alleviate the previously mentioned conflict. Beacon Capture never came to fruition because the Beacon Acquire never worked well enough.

Documentation

J. L. Jones and A. M. Flynn, “Mobile Robots: Inspiration to implementation,” A. K. Peters, ltd., Wellesley Ma, 1993.

F. G. Martin, “The 6.270 Robots Builder’s Guide” , F. G. Martin, 1992.

Motorola, MC68HC11 EVBU User’s Manual, Motorola, Inc., 1992

Appendix

```
/*******************************/
/* COLLISION AVOIDANCE         */
/* by THOMAS R. HARRINGTON     */
/*                             */
/* INPUTS:  front_bmp, rear_bmp */
/* OUTPUTS: bmp_move           */
/* FLAGS:   bmp_flg            */
/*******************************/

/* GLOBAL VARIABLES FOR COLLISION AVOIDANCE */

  int bmp_move;
  int bmp_flg = 0;
  long bmp_dur;

/* COLLISION AVOIDANCE PROCESS */

  void avoid_collision() {
    while(1) {

      if(front_bmp != 0 || rear_bmp != 0) {

        if(front_bmp != 0) {
          if(front_bmp < 70 || front_bmp >= 85 && front_bmp < 100 ||
front_bmp > 130 && front_bmp < 145) {
            bmp_move = REV_RGT;
            bmp_dur = 2000L;
          }
          else {
            bmp_move = REV_LFT;
            bmp_dur = 2000L;
          }
        }

        else {
          if(rear_bmp < 70 || rear_bmp >= 85 && rear_bmp < 100 ||
rear_bmp > 130 && rear_bmp <145) {
            bmp_move = FOR_LFT;
            bmp_dur = 2000L;
          }
```

```
          else {
            bmp_move = FOR_RGT;
            bmp_dur = 2000L;
          }
        }

        bmp_flg = 1;
      }

      else {
        bmp_flg = 0;
      }

      defer();
    }
  }



/*******************************/
/* OBSTACLE AVOIDANCE          */
/* by THOMAS R. HARRINGTON     */
/*                             */
/* INPUTS:                     */
/* OUTPUTS: obst_move          */
/* FLAGS:   obst_flg           */
/*******************************/

/* GLOBAL VARIABLES FOR OBSTACLE AVOIDANCE */

  int obst_thr = 104;
  int obst_flg = 0;
  int obst_move;
  long obst_dur;

/* OBSTACLE AVOIDANCE PROCESS */

  void avoid_obstacle() {
    while(1) {

      if(max_front > obst_thr && delta_front < 3) {

        if(max_lft > obst_thr && delta_lft < 3 && max_rgt > obst_thr &&
delta_rgt < 3) {
          obst_move = TURN_RGT;
          obst_dur = 1000L;  /*180 deg.*/
        }

        else if(max_rgt > obst_thr && delta_rgt < 3) {
          obst_move = TURN_LFT;
          obst_dur = 600L;
        }

        else {
          obst_move = TURN_RGT;
          obst_dur = 600L;
        }
```

```
        obst_flg = 1;
      }

      else {
        obst_flg = 0;
      }

      defer();
    }
  }


/*******************************/
/* BEACON ACQUIRE               */
/* by THOMAS R. HARRINGTON      */
/*                              */
/* INPUTS:                      */
/* OUTPUTS: acq_move           */
/* FLAGS:   acq_flg            */
/*******************************/

/* GLOBAL VARIABLES FOR BEACON ACQUIRE */

  int acq_flg = 0;
  int acq_move, acq_max;
  long acq_dur;

/* BEACON ACQUIRE PROCESS */

  void bcon_acquire() {
    while(1) {

      if(delta_front > 4 || delta_rear > 4 || delta_lft > 4 ||
delta_rgt > 4) {

        acq_max = max_front;
        if(acq_max < max_rear) acq_max = max_rear;
        if(acq_max < max_lft) acq_max = max_lft;
        if(acq_max < max_rgt) acq_max = max_rgt;

        if(acq_max == max_front) {
          acq_move = FORWARD;
          acq_dur = 0L;
        }
        else if(acq_max == max_rear) {
          acq_move = TURN_RGT;
          acq_dur = 1000L;
        }
        else if(acq_max == max_lft) {
          acq_move = TURN_LFT;
          acq_dur = 600L;
        }
        else {
          acq_move = TURN_RGT;
          acq_dur = 600L;
        }
```

```
      acq_flg = 1;
    }

    else {
      acq_flg = 0;
    }

    defer();
  }
}

/*GLOBAL VARIABLES FOR LYNXARM*/

  int dest_pos[6];
  int act_pos[]={0,125,125,125,0,125};

  /*VARIABLES FOR 90ø RANGE OF ARM MOTION*/

  int home[]={0,225,255,140,0,180};
  int max[]={0,225,225,254,0,180};
  int min[]={0,25,20,1,0,40};

/*SUBROUTINES FOR LYNXARM CONTROL*/

  void bend_joint(int joint, int position) {
    int sync=255;

    if (position < min[joint]) position=min[joint];
    if (position > max[joint]) position=max[joint];

    put_char(sync);
    put_char(joint);
    put_char(position);
  }

  void bend_arm(int position[]) {
    int i;
    for (i=5; i>=1; i--) bend_joint(i, position[i]);
  }

  void init_serial(int baud) {
    bit_clear(0x1028, 0x20);
    poke(0x102D, 0x0C);
    if (baud==9600) poke(0x102B, 0x30);
    if (baud==2400) poke(0x102B, 0x32);
  }

  void put_char(int outchar) {
    int test = 0;
    while (test == 0) {
      test = peek(0x102E);
      test = test & 0x80;
    }
    poke(0x102F, outchar);
  }
```

```
/*******************************/
/* MAIN CONTROL PROGRAM        */
/* by THOMAS R. HARRINGTON     */
/*                             */
/* INPUTS:                     */
/* OUTPUTS:                    */
/* FLAGS:                      */
/*******************************/

/* GLOBALS FOR MAIN PROGRAM */

  int move;
  long duration;

/* ARBITRATOR PROCESS FOR MAIN PROGRAM */

  void arbitrator() {
    while(1) {

      if(bmp_flg) {
        move = bmp_move;
        duration = bmp_dur;
      }
      else if(obst_flg) {
        move = obst_move;
        duration = obst_dur;
      }
      else if(acq_flg) {
        move = acq_move;
        duration = acq_dur;
      }
      else {
        move = FORWARD;
        duration = 0L;
      }

      steer();
      defer();
    }
  }

/* MAIN CONTROL PROGRAM */

  void main() {
    init_serial(9600);
    bend_arm(home);
    start_process(sensor_input(), 1000);
    start_process(avoid_collision(), 1000);
    start_process(avoid_obstacle(), 1000);
    start_process(bcon_acquire(), 1000);
    start_process(arbitrator(), 1000);
  }


/*******************************/
```

```
/* SENSOR INPUT                 */
/* by THOMAS R. HARRINGTON      */
/*                              */
/* INPUTS:  < None >            */
/* OUTPUTS: front_bmp, rear_bmp,*/
/*          max_front, max_rear,*/
/*          max_rgt, max_lft,   */
/*          arm_ird             */
/*                              */
/* FLAGS:   < None >            */
/********************************/

/* GLOBAL VARIABLES FOR SENSOR INPUTS */

  int i;
  int N = 34;
  int front_bmp, rear_bmp;
  int front_ird[35], rear_ird[35], lft_ird[35], rgt_ird[35];
  int max_front, max_rear, max_lft, max_rgt;
  int min_front, min_rear, min_lft, min_rgt;
  int delta_front, delta_rear, delta_lft, delta_rgt;

/* SUBROUTINES FOR SENSOR INPUT PROCESS */

  int maximum(int value[]) {
    int max = value[0];
    for(i=1; i<=N; i++) {
      if(max < value[i]) max = value[i];
    }
    return max;
  }

  int minimum(int value[]) {
    int min = value[0];
    for(i=1; i<=N; i++) {
      if(min > value[i]) min = value[i];
    }
    return min;
  }

/* SENSOR INPUT PROCESS */

  void sensor_input() {
    while(1) {

      poke(0x7000, 0xff);

      front_bmp = analog(0);
      rear_bmp = analog(1);

      if(front_bmp == 0 && rear_bmp == 0) {

        for(i=0; i<=N; i++) {
          front_ird[i] = analog(4);
          rear_ird[i] = analog(6);
          lft_ird[i] = analog(7);
          rgt_ird[i] = analog(5);
```

```
        }

        max_front = maximum(front_ird);
        max_rear = maximum(rear_ird);
        max_lft = maximum(lft_ird);
        max_rgt = maximum(rgt_ird);

        min_front = minimum(front_ird);
        min_rear = minimum(rear_ird);
        min_lft = minimum(lft_ird);
        min_rgt = minimum(rgt_ird);

        delta_front = max_front - min_front;
        delta_rear = max_rear - min_rear;
        delta_lft = max_lft - min_lft;
        delta_rgt = max_rgt - min_rgt;
      }

      defer();

    }
  }

/*******************************/
/* STEERING SUBROUTINES        */
/* by THOMAS R. HARRINGTON     */
/*                             */
/* INPUTS:                     */
/* OUTPUTS:                    */
/* FLAGS:                      */
/*******************************/

/* GLOBALS FOR STEERING SUBROUTINE */

  int left = 1;
  int right = 0;

  int STOP = 0;
  int FORWARD = 1;
  int FOR_RGT = 2;
  int FOR_LFT = 3;
  int REV_RGT = 4;
  int REV_LFT = 5;
  int TURN_RGT = 6;
  int TURN_LFT = 7;

  float lft_speed, rgt_speed;
/*  float act_lft_speed, act_rgt_speed;*/

/* STEERING SUBROUTINE */

  void steer() {

    if(move == STOP) {
      lft_speed = 0.0;
      rgt_speed = 0.0;
    }
```

```
    else if(move == FORWARD) {
      lft_speed = 30.0;
      rgt_speed = -30.0;
    }
    else if(move == FOR_RGT) {
      lft_speed = 30.0;
      rgt_speed = -10.0;
    }
    else if(move == FOR_LFT) {
      lft_speed = 10.0;
      rgt_speed = -30.0;
    }
    else if(move == REV_RGT) {
      lft_speed = -30.0;
      rgt_speed = 10.0;
    }
    else if(move == REV_LFT) {
      lft_speed = -10.0;
      rgt_speed = 30.0;
    }
    else if(move == TURN_RGT) {
      lft_speed = 30.0;
      rgt_speed = 30.0;
    }
    else if(move == TURN_LFT) {
      lft_speed = -30.0;
      rgt_speed = -30.0;
    }

    drive_motors();
    msleep(duration);
  }

/*  void drive_motors() {

    int k;
    float inc_lft, inc_rgt;

    inc_lft = (lft_speed - act_lft_speed) / 10;
    inc_rgt = (rgt_speed - act_rgt_speed) / 10;

    for(k=0; k < 10; k++) {
      act_lft_speed += inc_lft;
      act_rgt_speed += inc_rgt;
      motor(left, act_lft_speed);
      motor(right, act_rgt_speed);
    }
*/
  void drive_motors() {
    motor(left, lft_speed);
    motor(right, rgt_speed);
  }
```