Exploring a Bounded Environment with Obstacles and Chasing a Target using e-puck2 (ACS6501)

Flavin Lee John and Ankur Singh Gulia

***Abstract*— An attempt at creating self-awareness for an e-puck2 robot with the help of two tasks, The first was to precisely perform domain exploration while taking into account the barriers in the way and the bounds of the domain or the available space so that the robot never collides. The second task entails that the robot should follow the object when it is presented itself in its proximity unless the robot is not too close to the object. In that case, the robot must reverse from its actual position to obtain a position that is at a boundary from the colliding position. These tasks were carried out with the aid of robot programming, which was done in accordance with the later described methodologies and program implementation.**

I. STRATEGIES

Below is an overview of the strategies used to complete exploration and chasing tasks.

A. *Explore a bounded environment with obstacles*

The e-puck2 robot's proximity sensors were employed to move forward while avoiding collisions. The proximity to objects in the course of motion was determined using the signals from the four front sensors. As illustrated in Fig. 3, two threshold boundaries were established using the proximity sensors at the front of the robot.

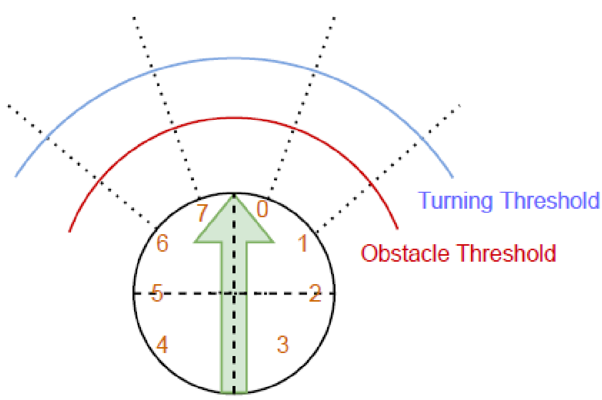

Fig. 1. The position of the eight proximity sensors on the e-puck2 robot and the two threshold boundaries defined using the four proximity sensors at the front.

The obstacle threshold determines the distance at which the robot should regard the object in front as an obstruction. As the robot moves forward, the sensor readings are continuously compared to this threshold to detect obstacles. When an obstruction is recognised, as shown in Fig. 2 - the finite-state machine diagram, the robot changes its state and begins to turn. The robot turns to the side with more open space based on the readings from the front proximity sensors. The turning threshold establishes the point at which the robot can cease turning and begin to advance. The turning threshold was introduced in addition to the obstacle threshold so that the robot turns until the obstacle is farther away than the obstacle threshold. This enables the robot to make a distinct forward motion before switching into the turn state.

Gulia and John are with the Department of Automatic Control and Systems Engineering, The University of Sheffield, UK" {asgulia1, fljohn1}@sheffield.ac.uk

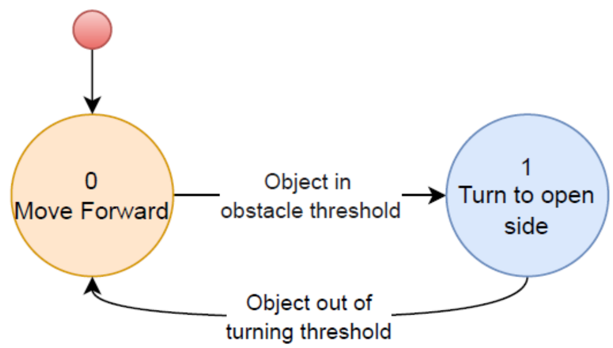

Fig. 2. Finite-state machine diagram for exploring the environment with obstacle avoidance

B. *Chase an object while avoiding collision with it*

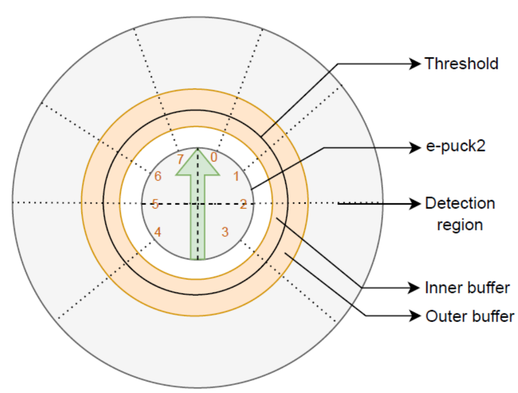

Fig. 3. Representation of the e-puck2 robot with the defined control zones.

The e-puck2 robot's proximity sensors were all used to establish the detection region shown in Fig. 3. When the object is detected, the robot engages sensors 0 and 7 and turns in its direction. As depicted in Fig. 4, the finite-state

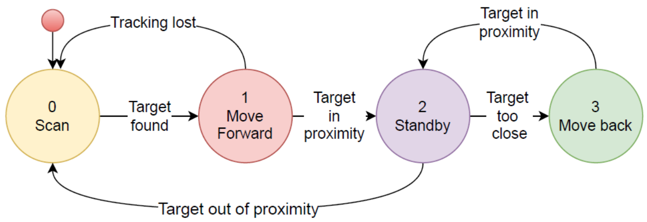

Fig. 4. Finite-state machine diagram for chasing an object

machine diagram, The robot changes states and moves to the direction of the target once sensors 0 and 7 have both detected the target object. Until the object is moved either away from or toward the robot, it remains in standby mode. It is programmed to keep a set distance from the object while allowing for a specific amount of tolerance, which is represented as the buffer region. The buffer region prevents the robot from rapidly switching states due to the noise in the proximity sensor readings. From the standby mode, the robot moves backwards to maintain distance if the object is moved past the inner buffer zone, whereas it switches to scan mode if the object is moved out of the buffer zone.

II. Implementation

The specifics on how the strategies were implemented are detailed below.

A. *Explore a bounded environment with obstacles*

Initialized with motor speeds for turning and forward motion equaling 1000, the e-puck2 robot's obstacle threshold was set at a proximity sensor reading of 350. Line 189 of Appendix I's definition of the function to identify obstacles uses data from proximity sensors 0, 1, 6, and 7. The turning threshold is set to 70% of the obstacle threshold value and is utilised in the function to determine whether the path is clear, and the robot can begin moving ahead. This function is defined in line 214 of Appendix I and uses the same sensors as the previous function.

B. *Chase an object while avoiding collision with it*

The target proximity threshold was set at a reading of 400, while the detection range was determined by the proximity sensor's reading of 60. 200% and 40% of the intended proximity threshold were chosen as the borders of the buffer region. The robot's state is controlled by the switch conditions specified in lines 131 through 165 of Appendix II. The function to set turn direction, defined in line 186, Appendix II, fetches readings of all eight sensors and sums the values of sensors on the left and right separately to decide which side to turn. The various function executed at the buffer zone depending on the movement and position of the object is illustrated in Fig. 5. The function to check if the target is in the proximity of the robot, defined in line 313, Appendix II, gave output considering the present state of the robot. This was done to reuse the function while the robot moved forward or backwards.

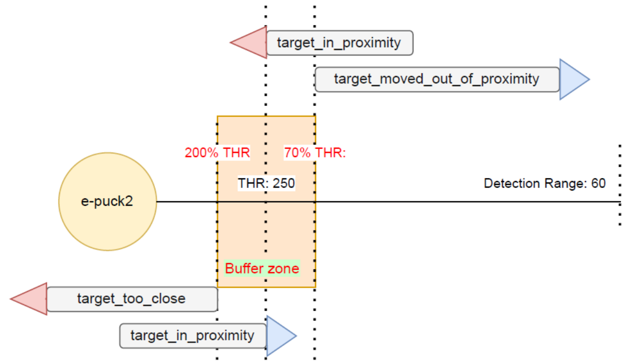

Fig. 5. Function calls corresponding to the movement and position of the object in the buffer zone

III. Discussions and Result

Various inbuilt libraries and algorithms based on finite-state machine concepts were used to execute both tasks successfully.

While exploring a bounded environment with obstacles, the e-puck2 identified open space by comparing the proximity sensor values and turned to the open side, thus exploring the entire area. Adding a second threshold for turning enabled the robot to manoeuvre smoothly. However, the robot was unable to enter the narrow passage. Instead of using the same threshold value for all four proximity sensor readings, using a higher distance tolerance for the sensors on either side, sensors 6 and 2, might help to overcome this issue.

To chase an object, the robot could register the object during the scan and then move towards the object without colliding. It halted at a safe distance on reaching the object's proximity and moved backwards when the object was moved towards it. The robot entered scan mode and repeated the process when the object was moved away from it. The buffer zone implementation prevented the robot from vibrating when it was stopped. However, the robot couldn't always detect the object on its left as the algorithm decided the turn direction at the first iteration of the while loop in scan mode. Since the loop was run at a frequency of $20Hz$, the logic to decide the direction was executed in a $50ms$ timeframe. The robot spun right by default if no object was registered during this $50ms$ time period. Repeating the decision logic until an object is registered could mitigate this problem.

The performance of the robot could be increased by incorporating these minor fixes. Further, the capabilities could be enhanced by using the distance sensor and the IMU.

APPENDIX I

```c

/************** ********* ********* ********* ********* ********* ********* ********* ********* *****************/
/*                          TASK 1: Explore a bounded environment with obstacles                                */
/************** ***************** ********* ********* *********  ********* ********* ********* *****************/



/* Explore Arena v1.4.0
 *
 * Authors:
 * Flavin Lee John
 * Ankur Singh Gulia
 *
 * Date: 1 Nov 2022
 *
 * Algorithm:
 * > Move forward
 * > If obstacle Detected, turn
 * > If obstacle clear, move forward
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "ch.h"
#include "hal.h"
#include "memory_protection.h"
#include <main.h>

// header files for UART
#include "epuck1x/uart/e_uart_char.h"
#include "stdio.h"
#include "serial_comm.h"

#include "sensors/proximity.h"
#include "motors.h"

// Define inter process communication bus
messagebus_t bus;
MUTEX_DECL(bus_lock);
CONDVAR_DECL(bus_condvar);

// Function Declarations
void move_forward(void);
int obstacle_ahead(void);
int path_is_clear(void);
void turn(void);
void turn_left(void);
void turn_right(void);
void set_turn_direction(void);
void should_stop_move_forward(void);
void should_stop_turn(void);

/******* Global Variables ********/

/* Bot State:
 * 0: move forward mode
 * 1: turn mode */
int bot_state = 0;

// Speed
const int TURN_SPEED = 1000;
const int MOVE_SPEED = 1000;

// Obstacle threshold value : IR reading above this val => obstacle nearby
const int OBS_THR = 350;

/* Turn Direction
 * 1 : Turn Right
 * 0 : Turn Left;
 */
int turn_direction = 1;
/******* ***** ********** ********/

int main(void)
{
    halInit();
    chSysInit();
    mpu_init();

```

```c
    motors_init();

  // Initiate inter-process communication bus
  messagebus_init(&bus, &bus_lock, &bus_condvar);

    // Start & Calibrate the proximity sensor
    proximity_start();
    calibrate_ir();

    // initialize UART1 channel
    serial_start();

    /* Infinite loop. */
    while (1) {
      // delay in milliseconds. 20Hz
        chThdSleepMilliseconds(50);

        switch(bot_state) {
          // moving forward mode
        case 0:
          move_forward();
          should_stop_move_forward();
          break;

        // Turn mode
        case 1:
          turn();
          should_stop_turn();
          break;
        }
    }
}

#define STACK_CHK_GUARD 0xe2dee396
uintptr_t __stack_chk_guard = STACK_CHK_GUARD;

void __stack_chk_fail(void)
{
    chSysHalt("Stack smashing detected");
}

/*************** Helper Functions *************************/

// Move the bot forward;
void move_forward(void) {
  right_motor_set_speed(MOVE_SPEED);
  left_motor_set_speed(MOVE_SPEED);
}

/* Switch bot state to turning mode if obstacle ahead*/
void should_stop_move_forward(void) {
  if(obstacle_ahead()) {
    bot_state = 1;
    set_turn_direction();
  }
}

/*************** Turn *****************/

/* Turn the bot */
void turn(void) {
  if(turn_direction) {
    turn_right();
  } else {
    turn_left();
  }
}

// turn the bot clockwise
void turn_right(void) {
  right_motor_set_speed(-1 * TURN_SPEED);
  left_motor_set_speed(TURN_SPEED);
}

// turn the bot counter clockwise
void turn_left(void) {
  right_motor_set_speed(TURN_SPEED);
  left_motor_set_speed(-1 * TURN_SPEED);
}

/* Identify which direction to turn.
 * 1: Right
 * 0: Left
 *
 * Compare values of sensors at the edge (1 & 6). Turn towards
```

```c
 * the lowest sensor value (Reflection angle)
 */
void set_turn_direction(void) {
  turn_direction = get_prox(1) < get_prox(6);
}

/* If path is clear, set bot_state = 0
 * i.e stop turning & start moving forward */
void should_stop_turn(void) {
  if(path_is_clear()) {
    bot_state = 0;
  }
}

/*********** Obstacle Detection *********************/

/* return:
 *  1 : obstacle ahead
 *  0 : no obstacle ahead
 */
int obstacle_ahead(void) {
  int obstacle_detected = 0;

  // The 4 front sensors
  int sensors[4] = {0, 1, 6, 7};

  for(int i=0; i<4; i++) {
    if(get_prox(sensors[i]) > OBS_THR) {
      obstacle_detected = 1;
    }
  }

  return obstacle_detected;
}


/* Check if the path ahead is clear. The threshold value is set
 * lower than the threshold value of obstacle_ahead function so
 * that the bot turns further towards the free space. This is to
 * avoid vibration of the bot while traveling along the THR border.
 *
 * return:
 *  1 : path clear
 *  0 : path not clear
 */
int path_is_clear(void) {
  int path_is_not_clear = 0;

  // The 4 front sensors
  int sensors[4] = {0, 1, 6, 7};


  for(int i=0; i<4; i++) {
    // Compared to 70% of Obstacle Threshold value
    if(get_prox(sensors[i]) > (0.7 * OBS_THR)) {
      path_is_not_clear =  1;
    }
  }

  return !path_is_not_clear;
}

/************** THE END ;D ******************/
```

APPENDIX II

```c

/************** ********* ********* ********* ********* ********* ********* ********* ********* *****************/
/*                           TASK 2: Chase an object while avoiding collision with it                             */
/************** ********* ********* ********* ********* ********* ********* ********* ********* *****************/

/* Chase Target v1.8.0
 *
 * Authors:
 * Flavin Lee John
 * Ankur Gulia
 *
 * Date: 1 Nov 2022
 *
 * Algorithm:
* Scan
  if s0 && s7 > range_thr     : Target detected
    move forward
* Move forward
  if s0 || s7 > prox_thr      : Target reached
    go to standby
  if s0 || s7 < range_thr     : Tracking lost
    start scan
* Standby
  if s0 && s7 < outer_prox_thr  : Target moved out
    start scan
  if s0 || s7 > inner_prox_thr  : Target too close
    move back
* Move back
  if s0 && s7 < prox_thr      : Safe distance
    go to standby
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>

#include "ch.h"
#include "hal.h"
#include "memory_protection.h"
#include <main.h>

// header files for UART
#include "epuck1x/uart/e_uart_char.h"
#include "stdio.h"
#include "serial_comm.h"

#include "sensors/proximity.h"
#include "motors.h"

/********* CONSTANTS ***********/

// Speed
#define TURN_SPEED 400
#define MOVE_SPEED 800

// Target Proximity threshold value : IR reading above this val => don't get closer
#define TARGET_PROX_THR 400
#define INNER_PROX_THR (2 * TARGET_PROX_THR)
#define OUTER_PROX_THR (0.4 * TARGET_PROX_THR)

// Range to detect target
#define RANGE_THR 60

/**********************************/

// Define inter process communication bus
messagebus_t bus;
MUTEX_DECL(bus_lock);
CONDVAR_DECL(bus_condvar);

// Function Declarations
void move_forward(void);
void move_back(void);
int target_detected(void);
int target_in_proximity(void);
void turn(void);
void should_stop_turn(void);
void should_stop_move_forward(void);
void should_stop_move_back(void);
int tracking_lost(void);
void should_stop_standby(void);
```

```c
int target_moved_out_of_proximity(void);
void standby(void);
int target_too_close(void);
void set_turn_direction(void);
void print_state_error(void);

/******* Global Variables ********/

/* Bot State:
 * 0: scan mode
 * 1: move forward mode
 * 2: standy mode
 * 3: move back mode*/
int bot_state = 0;

int scan_state = 0;

int turn_direction = -1;

/******* ***** ********** ********/

int main(void)
{
    halInit();
    chSysInit();
    mpu_init();

    motors_init();

  // Initiate inter-process communication bus
  messagebus_init(&bus, &bus_lock, &bus_condvar);

    // Start & Calibrate the proximity sensor
    proximity_start();
    calibrate_ir();

    // initialize UART1 channel
    serial_start();

    /* Infinite loop. */
    while (1) {
      // delay in milliseconds. 20Hz
        chThdSleepMilliseconds(50)

        switch(bot_state) {
      // Scan mode
      case 0:
        switch(scan_state) {
          case 0:
            // identify direction
            set_turn_direction();
            scan_state = 1;
            break;
          case 1:
            // turn
            turn();
            should_stop_turn();
            break;
        }
        break;

          // moving forward mode
        case 1:
          move_forward();
          should_stop_move_forward();
          break;

        // standby mode
        case 2:
          standby();
          should_stop_standby();
          break;

      // moving back mode
      case 3:
        move_back();
        should_stop_move_back();
        break;
        }
    }
}

#define STACK_CHK_GUARD 0xe2dee396
uintptr_t __stack_chk_guard = STACK_CHK_GUARD;
```

```c
void __stack_chk_fail(void)
{
    chSysHalt("Stack smashing detected");
}

/*************** Helper Functions *************************/

/**************** Turn *****************/

/* If any object detected on right side, set 1. else -1
 *
 * 1: clockwise
 * -1: counter clockwise
 */
void set_turn_direction(void) {
  int left_sum = 0;
  int right_sum = 0;

  for(int i=0; i<4; i++) {
      right_sum += get_prox(i);
  }

  for(int i=4; i<8; i++) {
    left_sum += get_prox(i);
  }

  int direction = right_sum > left_sum ? 1 : -1;

  for(int i=0; i<4; i++) {
    if(get_prox(i) > RANGE_THR) {
      direction = 1;
    }
  }

  turn_direction = direction;
}

/* Turn the bot
 *
 * turn direction = 1: turn clockwise
 * turn direction = -1: turn counter clockwise
 */
void turn(void) {
  right_motor_set_speed((-1) * (turn_direction) * TURN_SPEED);
  left_motor_set_speed((turn_direction) * TURN_SPEED);
}

/* Switch bot state from scanning (turning) to:
 * - move forward (1) if target detected
 * i.e stop turning & start moving forward
 *
 * + Reset scan_state */
void should_stop_turn(void) {
  if(target_detected()) {
    bot_state = 1;
    scan_state = 0; // reset scan state;
  }
}

/**************** Move Forward/Backward *****************/

// Move the bot forward;
void move_forward(void) {
  // Add PID
  right_motor_set_speed(MOVE_SPEED);
  left_motor_set_speed(MOVE_SPEED);
}

// Move the bot back;
void move_back(void) {
  right_motor_set_speed(-1 * MOVE_SPEED);
  left_motor_set_speed(-1 * MOVE_SPEED);
}

/* Switch bot state from move forward to:
 * - to standby mode(2) if target reached
 * - to scan mode(0) if tracking lost */
void should_stop_move_forward(void) {
  if(target_in_proximity()) {
    bot_state = 2;
  }
  else if(tracking_lost()) {
    bot_state = 0;
  }
}
```

```
/* Switch bot state from move back to:
 * - to standby mode(2) if target back in proximity */
void should_stop_move_back(void) {
  if(target_in_proximity()) {
    bot_state = 2;
  }
}

/******************** Stand by ********************/

// Stand by: Stop the movement
void standby(void) {
  right_motor_set_speed(0);
  left_motor_set_speed(0);
}

/* Switch bot state from stand by:
 * - to scan mode if target moved out of proximity
 * - to move back mode if target too close
 */
void should_stop_standby(void) {
  if(target_moved_out_of_proximity()) {
    bot_state = 0;
  } else if (target_too_close()) {
    bot_state = 3;
  }
}

/*********** Target Detection *********************/

/* Check if the front sensors have any object in the range
 *
 * return:
 *  1 : target found
 *  0 : target not found
 */
int target_detected(void) {
  // Using the two front sensors. s0 & s7.
  // target found if both s0 & s7 registers object.
  int target_found = (get_prox(0)>RANGE_THR) && (get_prox(7)>RANGE_THR);

  return target_found;
}

/* Check if bot is in the proximity of the target
 *
 * while moving forward:
 *    prox values increase & crosses TARGET_PROX_THR (>)
 * while moving backward:
 *    prox values decrease & crosses TARGET_PROX_THR (<)
 *
 * return:
 *  1 : target in proximity
 *  0 : target not in proximity
 */
int target_in_proximity(void) {
  // Using the two front sensors. s0 & s7.
  int in_proximity;
  const int thr = TARGET_PROX_THR;  // target proximity threshold
  const int thr1 = 0.7*INNER_PROX_THR;

  if(bot_state == 1) {      // moving forward
    in_proximity = (get_prox(0)>thr) || (get_prox(7)>thr);
  } else {          // moving backward
    // ensure both sensors register target out of threshold
    in_proximity = (get_prox(0)<thr1) && (get_prox(7)<thr1);
  }

  return in_proximity;
}

/* Check if bot lost track of the target
 *
 * return:
 * 1: tracking lost     => target not detected ahead
 * 0: tracking not lost   => target detected ahead
 */
int tracking_lost(void) {
  return !target_detected();
}

/* Check if target moved out of proximity
 * Use OUTER_PROX_THR for comparison -> buffer (to prevent vibration)
 *
```

```
 * return:
 *  1 : target moved out of proximity
 *  0 : target didn't move out of proximity
 */
int target_moved_out_of_proximity(void) {
  // Using the two front sensors. s0 & s7.
  // target moved out of proximity if neither of the
  // sensors register target inside outer proximity threshold
  const int thr = OUTER_PROX_THR;
  int out_of_proximity = (get_prox(0)<thr) && (get_prox(7)<thr);

  return out_of_proximity;
}

/* Using the two front sensors. s0 & s7.
* target moved too close to the bot if either of the
* sensors register target at max INNER_PROX_THR
*/
int target_too_close(void) {
  const int thr = INNER_PROX_THR;
  int too_close = (get_prox(0)>thr) || (get_prox(7)>thr);

  return too_close;
}
/************** THE END ;D *****************/
```