ACS6501 Foundation of Robotics
E-PUCK2 Obstacle Avoidance and Tracking Strategies (ACS6501)

Saba Ansaria[1] and Chaoyu Song[2]

***Abstract*— In this era, autonomous robots are synonymous with every possible task, such as production lines, rescue missions, or health sector, as a standalone or swarm mobile device. E-PUCK2 is a miniature-sized mobile robot that can be used as a test case for research and development and two tasks have been given to explore this. In task1, the robot needs to explore the closed environment while avoiding collisions with surrounding objects. Task2 is to track an object in an open environment and avoid collision with that object. The algorithms and experiments are discussed for completing these tasks.**

I. STRATEGIES

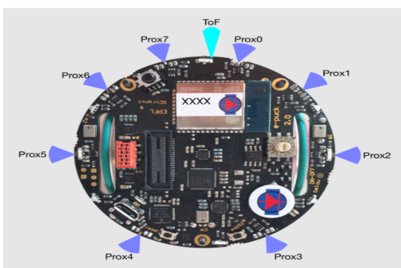

Fig. 1. E-PUCK2 IR proximity sensor locations (The image is reprinted from [1]).

The main idea of Task1 is to use the IR sensor reading value to determine whether there is an obstacle. The flowchart is presented in Figure 2. First, record the reading value of the infrared sensor under ambient light for the obstacle in front of the object. The characteristics of the infrared sensor suggest it is within a certain range of values. The closer the object is to the robot, the larger is the threshold value obtained. According to this characteristic, the E-PUCK robot can judge whether an object is in front of the current infrared sensor. E-PUCK2 has 8 infrared sensors, of which sensor 0 is located near the right side of the head of the E-PUCK2, and sensors 0-7 are arranged in a clockwise direction (shown in Figure 1). When the value of sensors numbers 0 and 1 is greater than the value of the set obstacle, it means that there is an obstacle on the right side of the E-PUCK2, so the function is called to make it turn left. Similarly, for sensor numbers 6 and 7, when it is detected, that is when it is greater than the set value, it means there is an obstacle on the left side, and it turns right. A problem arises when sensor numbers 1 and 6 detect obstacles simultaneously, and it signifies that the E-PUCK2 has entered a narrow area, and there are obstacles on both sides. At this time, a function is called to make the E-PUCK2 rotate 180 degrees approximately to turn backward and then move forward in that direction opposite to the previous moving direction (Figure 2). While during other situations, the E-PUCK2 executes the forward command. Furthermore, one crucial thing is the speed of the E-PUCK2, and the speed needs to be set in a way that the distance covered in 1 second is less than the threshold distance that has been set. It will be discussed in Section II. Task 2 uses the infrared sensors

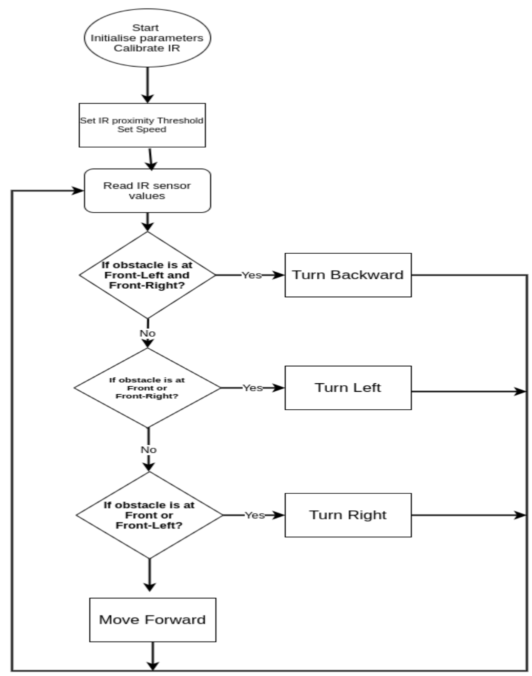

Fig. 2. Task 1: E-PUCK2 Navigation Flowchart

in a similar way for reading values to chase the object. The flowchart is presented in Figure 3. Firstly, the values of all the sensors are first compared to decide if the object is at the front or front-left, or front-right. If none of them, that means the object is backward, so the E-PUCK2 rotates. The idea is to make the E-PUCK2 front-facing to the object. If the object is front-right, the E-PUCK2 turns right. If the object is front-left, the E-PUCK2 turns left. Then the E-PUCK2 moves forward until it reaches the threshold distance. If the object comes forward to the E-PUCK2 and crosses the threshold boundary, the E-PUCK2 moves backward until the distance between them is more than the threshold distance. The given

[1]Ansaria is with the Department of Automatic Control and Systems Engineering. The University of Sheffield, UK.
[2]Song is with the Department of Automatic Control and Systems Engineering. The University of Sheffield, UK.

environment is obstacle-free, so it is assumed that there will be no other object other than the tracking object. Similarly, the forward speed and the turning speed are very crucial for these maneuvers. It is discussed in Section II.

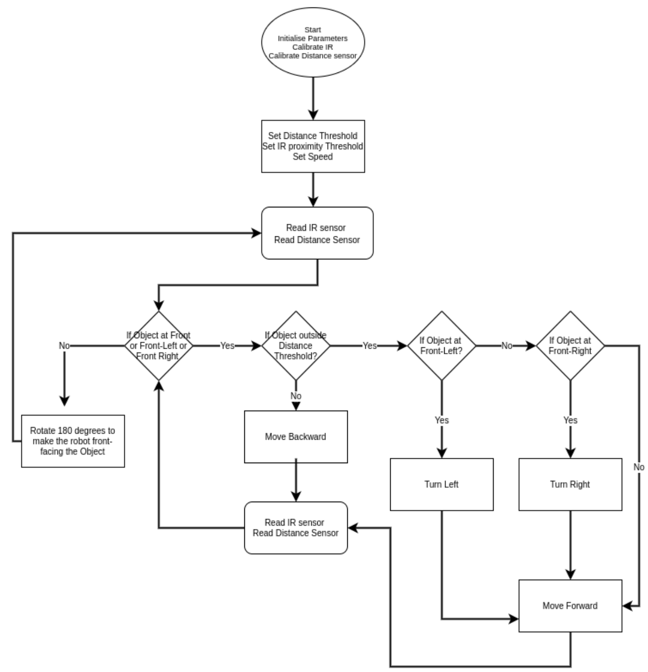

Fig. 3. Task 2: E-PUCK2 Object Track Flowchart

II. Implementation

As mentioned in Section I, the speed of E-PUCK2 is very crucial for the experiment. The IR sensor inputs are normally taken in a delay. Hence, the speed of the E-PUCK2 should be less than the threshold distance covered in a second. Otherwise, the robot will collide with an obstacle before it gets the next IR reading. Therefore, the experiment is run in two stages. First, the sensor values are calibrated and set the forward, backward, rotation speeds according to that, considering it goes not too fast or too slow. The rotation speed is also crucial. The rotation speed is set in a way that it rotates approximately 45 degrees on each function call. So that, if the robot needs to rotate 180 degrees, the function will be called four times. The rotation speed can be increased, but the number of function calling should be adjusted in that case. Next, the second stage is to run the given tasks.

Three threshold zones such as start, cutoff and neutral zone have been set for the second task. The robot starts moving forward when the object is away from the start zone and it stops until it reaches the cutoff zone. The neutral zone has been put for safety reason if there is any sensor delay or noise and the robot moves for more time than it suppose to move.

The main program is written in the C programming language using the standard E-PUCK2 system call interface. The inbuilt LED's were used to show the direction and the status of the E-PUCK2, which makes the decision of the mobile robot more interpretable.

The implementation code is provided in the APPENDIX. It has been found that the calibrated system call for IR input has an approximate 12-15ms delay. Also, the maneuver procedure has some delays. For the second task, this is crucial. The distance sensor is additionally used to add an extra threshold function. However, in our case, some inconsistencies in the sensory input have been found, which is discussed in Section III.

III. Results and Discussions

In the Task1 experiment, E-PUCK2 can successfully avoid collisions in the surrounding enclosed area. When there is an obstacle in the center area, E-PUCK2 can also successfully avoid collisions. When E-PUCK2 enters a narrow area, E-PUCK2 has obstacle avoidance behavior, but it is easy to fall into an endless loop in a narrow area. To avoid the endless loop, the 180 degrees of rotation is applied as discussed in Section I, and this strategy successfully avoids this scenario. In Task2, the robot can rotate itself and find the target object even if the target object is behind the E-PUCK2. E-PUCK2 realizes the sensor to get the forward direction and can follow the object to move forward and backward. The collision can be avoided when the distance is very close. Nevertheless, it cannot follow the target object precisely to turn left and right quickly. This problem is due to the inconsistent IR reading in the given E-PUCK2 robot, which is found during printing the IR values. These noisy IR inputs caused the forward movement to be bumpy. The IR0, IR1 were used to determine the right and IR6,IR7 were used to determine the left position. However, the noise caused this method significant damage.

For task1, the place that can be improved is adding the judgment of the dead zone loop, such as making the E-PUCK2 execute the instruction of a certain rotation angle at a specific interval. Furthermore, the speed of the E-PUCK2 is fixed here. The robot would navigate more smoothly if the speed is adaptive with the cluster of obstacles at a given period. Task2, the improvement, is the judgment of the turning logic. Due to the large size of the object, if the number difference among the front sensors is small, the effect may not be obvious. Probably comparing the sensors pairwise with all combinations may alleviate the situation. Considering distance sensor with paired IR sensor may also help to determine whether E-PUCK2 needs to turn or not.

To further improve the autonomy, a risk management method should be appointed to handle sensor failures by detecting sensor misreadings using the relative comparison between sensors in a robot cluster over time [2]. Finally, the camera image can be used for path detection which may compensate the IR sensor noises [3].

References

[1] E-PUCK2. e-puck2 GCtronic Wiki, https://www.gctronic.com/doc/index.php/e-puck2.
[2] Gauci, M., Chen, J., Li, W., Dodd, T.J., Groß, R. (2014). Clustering objects with robots that do not compute. AAMAS.
[3] Sumbal, M. S., Carreras, M. (2015). Environment Detection and Path Planning using the e-puck Robot.

APPENDIX

A. Task 1 E-PUCK2 Navigation Code

```c

#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>

#include "sensors/proximity.h"
#include "sensors/VL53L0X/VL53L0X.h"
#include "chprintf.h"
#include "usbcfg.h"
#include "epuck1x/uart/e_uart_char.h"
#include "stdio.h"
#include "serial_comm.h"
#include "leds.h"
#include "spi_comm.h"


messagebus_t bus;
MUTEX_DECL(bus_lock);
CONDVAR_DECL(bus_condvar);
int proximity_sensor_values[8]; // this array will store proximity sensor values
int state[8];
int proximty_threshold = 480; // this is set as proximity threshold
uint16_t distance_threshold = 40;
int MAX_SPEED = 400;
int LOW_SPEED = 0;
int cl=200;


/* function to stop the motor (set motor velocity to zero) */
void motor_stop() {
  right_motor_set_speed(LOW_SPEED);
  left_motor_set_speed(LOW_SPEED);
}

/* function to set motor velocity to move forward */
void motor_move_forward() {
  right_motor_set_speed(MAX_SPEED);
  left_motor_set_speed(MAX_SPEED);
}

void motor_move_backward() {
  right_motor_set_speed(-MAX_SPEED);
  left_motor_set_speed(-MAX_SPEED);
}

/* function to set motor velocity to rotate right in place*/
void motor_rotate_right() {
  right_motor_set_speed(-MAX_SPEED);
  left_motor_set_speed(MAX_SPEED);
}

/* function to set motor velocity to rotate left in place*/
void motor_rotate_left() {
  right_motor_set_speed(MAX_SPEED);
  left_motor_set_speed(-MAX_SPEED);
}

//task2

int main(void)
{
    /* initialization */
    halInit();
    chSysInit();
    mpu_init();
    messagebus_init(&bus, &bus_lock, &bus_condvar);
```

```
  proximity_start();
  calibrate_ir();
  VL53L0X_start();
  usb_start();
  serial_start();
  motors_init();
  clear_leds();
  spi_comm_start();
  set_body_led(0);
  set_front_led(0);

  // set this value to 1 if you want to just test the sensor values and motor speed

  int check_parameters = 0;
  int rotate_test =0;
  int speed_test =0;
  int distance_sensor_test = 0;

  // set this value to 1 if you want to use distance sensor

  int use_distance=0;


  /* Infinite loop. */
  while (1) {
    proximity_sensor_values[0] = get_calibrated_prox(0);
    proximity_sensor_values[1] = get_calibrated_prox(1);
    proximity_sensor_values[2] = get_calibrated_prox(2);
    proximity_sensor_values[5] = get_calibrated_prox(5);
    proximity_sensor_values[6] = get_calibrated_prox(6);
    proximity_sensor_values[7] = get_calibrated_prox(7);


    if (check_parameters==1){
      motor_stop();
    // Skip printing if port not opened.
      if (SDU1.config->usbp->state == USB_ACTIVE) {
        chprintf((BaseSequentialStream *)&SDU1, "PROXIMITY\r\n");
        chprintf((BaseSequentialStream *)&SDU1, "%4d,%4d,%4d,%4d,%4d,%4d,%4d,%4d\r\n\n",
            proximity_sensor_values[0], proximity_sensor_values[1], proximity_sensor_values[2],
          proximity_sensor_values[3], proximity_sensor_values[4], proximity_sensor_values[5],
          proximity_sensor_values[6], proximity_sensor_values[7]);
      }

      if (speed_test==1){
        motor_move_forward();
        motor_stop();
       }

      if (rotate_test==1){
        motor_rotate_right();
        motor_rotate_right();
        motor_stop();
      }

      if (distance_sensor_test==1){
        // Read distance sensor.
        chprintf((BaseSequentialStream *)&SDU1, "DISTANCE SENSOR\r\n");
        chprintf((BaseSequentialStream *)&SDU1, "%d\r\n\n", VL53L0X_get_dist_mm());
      }
    }

    else{
    for(int i=0;i<8;i++){
          if(proximity_sensor_values[i]>proximty_threshold){
                state[i]=1;
          }
          else {
                state[i]=0;
          }
    }

    if (state[1]==1 || state[6]==1){
       /* test the number of rotate right and put that many number
        * of times to do 180 degree rotation. Here we put 2*motor_rotate_right */
```

```
        // body led indicates the robot is moving
            set_body_led(1);
            set_front_led(0);
            motor_rotate_right();
            motor_rotate_right();

        }
        else if (state[0]==1 || state[1]==1){
          // the obstacle is on right so rotate left
         set_body_led(1);
         set_front_led(0);
            motor_rotate_left();
        }
        else if (state[7]==1 || state[6]==1){
          // rotate right
         set_body_led(1);
         set_front_led(0);
              motor_rotate_right();
        }
        else{
          set_body_led(0);
         // front led indicates the robot is moving forward
          set_front_led(1);

          if (use_distance==1){
           if(VL53L0X_get_dist_mm()>distance_threshold){
              motor_move_forward();
           }
           else{
             right_motor_set_speed(100);
             left_motor_set_speed(100);
           }
          }
          else{
            motor_move_forward();
          }
        }
       }

       //waits 1 second
         chThdSleepMilliseconds(1000);
    } // end of while
} // end of main function

#define STACK_CHK_GUARD 0xe2dee396
uintptr_t __stack_chk_guard = STACK_CHK_GUARD;

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

Listing 1. Task 1 E-PUCK2 Navigation Code

B. Task 2 E-PUCK2 Object Tracking Code

```

#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>

#include "sensors/proximity.h"
#include "sensors/VL53L0X/VL53L0X.h"
#include "chprintf.h"
#include "usbcfg.h"
#include "epuck1x/uart/e_uart_char.h"
#include "stdio.h"
#include "serial_comm.h"
#include "leds.h"
```

```c
#include "spi_comm.h"


messagebus_t bus;
MUTEX_DECL(bus_lock);
CONDVAR_DECL(bus_condvar);
int proximity_sensor_values[8]; // this array will store proximity sensor values
int state[8];
int proximty_threshold_start = 50; // this is set as proximity threshold
int proximty_threshold_neutralzone = 450; // this is set as proximity threshold
int proximty_threshold_cutoff = 550; // this is set as proximity threshold
int FRONT_SPEED = 100;
int LOW_SPEED = 0;
int BACK_SPEED = 200;
int ROTATE_SPEED = 200;
int cl=200;
uint16_t distance_threshold = 15;

/* function to stop the motor (set motor velocity to zero) */
void motor_stop() {
  right_motor_set_speed(LOW_SPEED);
  left_motor_set_speed(LOW_SPEED);
}

/* function to set motor velocity to move forward */
void motor_move_forward() {
  right_motor_set_speed(FRONT_SPEED);
  left_motor_set_speed(FRONT_SPEED);
}

void motor_move_backward() {
  right_motor_set_speed(-BACK_SPEED);
  left_motor_set_speed(-BACK_SPEED);
}

/* function to set motor velocity to rotate right in place*/
void motor_rotate_right() {
  right_motor_set_speed(-ROTATE_SPEED);
  left_motor_set_speed(ROTATE_SPEED);
}

/* function to set motor velocity to rotate left in place*/
void motor_rotate_left() {
  right_motor_set_speed(ROTATE_SPEED);
  left_motor_set_speed(-ROTATE_SPEED);
}



int main(void)
{
    /* initialization */
    halInit();
    chSysInit();
    mpu_init();
    messagebus_init(&bus, &bus_lock, &bus_condvar);
    proximity_start();
    calibrate_ir();
    VL53L0X_start();
    usb_start();
    serial_start();
    motors_init();
    clear_leds();
    spi_comm_start();
    set_body_led(0);
    set_front_led(0);

    // set this value to 1 if you want to just test the sensor values and motor speed

    int check_parameters = 1;
    int speed_test =0;
    int rotate_test =0;

    int back_test = 0;
    int distance_sensor_test = 0;
```

```c

    // set this value to 1 if you want to use distance sensor

    int use_distance_sensor=0;



    /* Infinite loop. */
    while (1) {
      proximity_sensor_values[0] = get_calibrated_prox(0);
      proximity_sensor_values[1] = get_calibrated_prox(1);
      proximity_sensor_values[2] = get_calibrated_prox(2);
      proximity_sensor_values[5] = get_calibrated_prox(5);
      proximity_sensor_values[6] = get_calibrated_prox(6);
      proximity_sensor_values[7] = get_calibrated_prox(7);


      if (check_parameters==1){
        motor_stop();
      // Skip printing if port not opened.
        if (SDU1.config->usbp->state == USB_ACTIVE) {
          chprintf((BaseSequentialStream *)&SDU1, "PROXIMITY\r\n");
          chprintf((BaseSequentialStream *)&SDU1, "%4d,%4d,%4d,%4d,%4d,%4d,%4d,%4d\r\n\n",
              proximity_sensor_values[0], proximity_sensor_values[1], proximity_sensor_values[2],
            proximity_sensor_values[3], proximity_sensor_values[4], proximity_sensor_values[5],
            proximity_sensor_values[6], proximity_sensor_values[7]);
        }
    // testing each component one by one
        if (speed_test==1){
          motor_move_forward();
          motor_stop();
         }

        if (rotate_test==1){
          motor_rotate_right();
          motor_stop();
        }

        if (back_test==1){
          motor_move_backward();
            motor_stop();
        }

        if (distance_sensor_test==1){
          // Read distance sensor.
          chprintf((BaseSequentialStream *)&SDU1, "DISTANCE SENSOR\r\n");
          chprintf((BaseSequentialStream *)&SDU1, "%d\r\n\n", VL53L0X_get_dist_mm());
        }
      }

      else{
    // if IR0 or IR7 is greater than the proximty_threshold_start then the object is in fron of the robot and out of threshold zone
                // So the robot will go forward until it reaches to the cutoff zone (too close to the object) and the motor stops to avoid collison
    // for a failsafe option a cuttoff neutral threshold is added if the IR sensors have any delay or noise in the output.
        if (proximity_sensor_values[0]> proximty_threshold_start || proximity_sensor_values[7]> proximty_threshold_start)
        {
           while(get_calibrated_prox(0)<proximty_threshold_cutoff && get_calibrated_prox(7)< proximty_threshold_cutoff && get_calibrated_prox(6)<proximty_threshold_cutoff && get_calibrated_prox(1)<proximty_threshold_cutoff)
           {

          if (get_calibrated_prox(0)>proximty_threshold_neutralzone || get_calibrated_prox(7)> proximty_threshold_neutralzone){
              motor_stop();
              set_body_led(1);
              set_front_led(0);
              break;
          }
      set_body_led(0);
      set_front_led(1);
          motor_move_forward();
```

```
                chThdSleepMilliseconds(150);
            }
            if(get_calibrated_prox(1)>get_calibrated_prox(7)){
            motor_rotate_right();
            }
            else if (get_calibrated_prox(6)>get_calibrated_prox(0)){
            motor_rotate_left();
            }
        }
    // if IR0 or IR7 is not greater than the proximty_threshold_start but IR4 or IR3 is greater than the proximty_threshold_start then the robot
    // will rotate to face the object. Set the amout of rotation in the calibration stage 1 and set the number of function calling accordingly
    else{
       if( get_calibrated_prox(4)>proximty_threshold_start || proximity_sensor_values[3]>proximty_threshold_start){
        // motor_rotate_right() is called twice here. This is set depending on the rotation speed. Here the goal is to roatate 180 degrees
        // if the rotation degree is smaller in each function calling then we might need to call motor_rotate_right() more than twice.
                    motor_rotate_right();
        motor_rotate_right();
        }
    }

        if (get_calibrated_prox(0)>proximty_threshold_cutoff || get_calibrated_prox(7)>proximty_threshold_cutoff)
        {
      // if the object is moving closer and cross the threshold zone, the robot will go backward. The it will calibrate its position to left or right
      // to face the object at front.
          motor_move_backward();
      set_body_led(1);
      set_front_led(0);
          if(get_calibrated_prox(1)>get_calibrated_prox(7)){
              motor_rotate_right();
          }
          else if (get_calibrated_prox(6)>get_calibrated_prox(0)){
              motor_rotate_left();
          }
        }
    // using distance sensor for extra threshold checking
    if (use_distance_sensor==1)
    {
      if(VL53L0X_get_dist_mm()<distance_threshold)
      {
        set_body_led(1);
        set_front_led(0);
        motor_move_backward();
      }

 }


      } // end of else


      //waits 1 second
        chThdSleepMilliseconds(900);
    } // end of while
} // end of main function

#define STACK_CHK_GUARD 0xe2dee396
uintptr_t __stack_chk_guard = STACK_CHK_GUARD;

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

Listing 2. Task 2 E-PUCK2 Object Tracking Code