

e-puck Leader/Follower and Obstacle Avoidance

Teli Yin
Bo Tao
Max Sobell
CPE 555

Background and Introduction:

We worked with the e-puck educational robot. It is a desktop sized robot with a wide range of sensors including sound sensors, an accelerometer, proximity sensors and a camera. It can communicate with a computer via Bluetooth¹.

Approach:

We used the MPLAB environment to program the e-puck robot and the C 30 compiler to compile the program for the microprocessor onboard the e-puck. We used tinybldWin to flash the compiled program onto the e-puck using Bluetooth.

We implemented a “follower” program, where a “follower” follows a “leader”. When the leading robot meets an obstacle, the robot will dynamically avoid the obstacle (turn until the IR sensors are below a certain threshold). The follower will follow the lead robot as it avoids the obstacle. When there is a loud noise (over a certain threshold), the robots will switch roles. First, they will turn 180 degrees, then the leader will become the follower and the follower will become the leader.

Experimental Results:

We were able to implement our design successfully. Initially we had planned to have the robots communicate via Bluetooth when switching roles. However, this proved impractical as the e-puck’s Bluetooth module is not well documented. We chose, instead, to have both robots react to a common sound, such as a loud clap. In general, this method worked fairly well. There were some instances, however, when the noise level of the clap was over the threshold of one robot but not both. It was also necessary to have the threshold sufficiently high as to not have the robots react to background noise in a room.

We implemented two separate obstacle avoidance algorithms. In the first, when the leader robot meets a rectangular obstacle, it follows the border of the obstacle through two turns and continues in its original direction. The second is dynamic obstacle avoidance. When the lead

¹ http://www.e-puck.org/index.php?option=com_content&task=view&id=5&Itemid=27

robot meets an obstacle, it turns away while continuing to move forwards until the object no longer registers on its front sensors. This method is much faster and more practical for the leader/follower scenario.

Conclusion and Future Work:

We were able to meet our proposed goals: leader/follower program, role switching, and dynamic obstacle avoidance. We ran into several issues along the way, mainly regarding obstacle avoidance and inter-robot communication, which we worked through.

Future work could involve a more robust obstacle avoidance algorithm. The lead robot could use its side sensors as well as its front sensors to make sure it is properly avoiding the object and hasn't caught its axle on the object. We could also implement inter-robot communication using Bluetooth. This would be very time consuming, however, until the proper documentation is published.

We could extend our algorithm to multiple robots using a single leader. Using the current implementation, this would not be too difficult. We could flash all the robots with the follower program, change one to leader, and run them all in a line. The current follower program has three speeds: FAST, SLOW, and STOP. This enables a follower to catch up with a leader; in our extended case, this speed model would enable a follower to catch up with another follower.

Appendix

Code:

```
#include <p30f6014A.h>
#include <e_epuck_ports.h>
#include <e_init_port.h>
#include <e_micro.h>
#include <e_motors.h>
#include <e_agenda.h>
#include <e_prox.h>

#define FAST          700 //speed
#define SLOW         200 //speed
#define SLOW_RANGE   500 //sensor value
#define STOP         700 //sensor value, ~= SLOW_RANGE + 200
#define TURN         350 //turn value
#define SOUND_T     1500 //sound threshold

int follow = 1;
int swapping = 0;

/*
 * follow_pos: returns a weighted average of the sums
 * where the sensors to the left of the middle carry a
 * negative weight and the sensors to the right carry a
 * positive weight. We use sensors 2,1 and 7,6. The
 * outermost sensors are weighted more heavily.
 *
 * sensor value higher if closer
 */

int follow_pos( int *values ) {
    int i, val;
    int weight = 12;
    for( i = 0; i < 8; i++ ) {
        if( values[i] > 500 )
            values[i] = 500;
        values[i] /= 10;
    }
    val = -weight*1.5*values[1]-
weight*values[0]+weight*values[7]+weight*1.5*values[6];
    if( val > -100 && val < 100 ) val = 0;
    return val;
}

void swap() {
    follow = !follow; //switch!
    swapping = 1; //switching!
}

int main(void)
{
    int values[7]; // 0 and 7 front sensors
```

```

e_init_prox();
e_init_port();
e_init_micro();
e_init_motors();
// sensor most effective 300 to 1000
int speed_value, turn_value = 0;
int i, k;
long j;

//for mic:
int m1, m2, m3;

while ( follow ) {
    LED1 = LED2 = LED3 = LED4 = LED5 = LED6 = 0;
    for( i = 0; i < 8; i++ ) {
        values[i] = e_get_prox(i);
    }
    if ( values[0] > SLOW_RANGE || values[7] > SLOW_RANGE ) {
        // front sensors -- slow down
        speed_value = SLOW;
    } else {
        // speed up!
        speed_value = FAST;
    }

    // just stop!
    if ( values[0] > STOP || values[1] > STOP || values[7] > STOP ||
values[6] > STOP ) {
        speed_value = 0;
    }

    // now decide which direction to turn
    if ( speed_value > 0 )
        turn_value = follow_pos( values );
    else
        turn_value = 0;

    e_get_micro(&m1, &m2, &m3);
    if(m1 > SOUND1T && m2 > SOUND1T && m3 > SOUND1T) {
        LED0 = 1;
        swap();
        e_set_speed_left( 400 );
        e_set_speed_right( -400 );
    } else {
        LED0 = 0;
        e_set_speed( speed_value, turn_value );
    }

    e_start_agendas_processing();

    // delay
    for( k = 0; k < 3000; k++ ) {
        asm("nop");
    }
    if( swapping ) {
        for( j = 0; j < 2000000; j++ ) {
            asm("nop");
        }
    }
}

```

```

    }
    e_set_speed( 0, 0 );
    for( j = 0; j < 2000000; j++ ) {
        asm("nop");
    }
    swapping = 0;
}
}
while ( !follow ) {
    LED1 = LED2 = LED3 = LED4 = LED5 = LED6 = 1;
    for( i = 0; i < 8; i++ ) {
        values[i] = e_get_prox(i);
    }

    speed_value = FAST - 50;

    if ( values[0] > 500 || values[7] > 500 || values[1] > 500 ||
values[6] > 500 ) {
        // decide which way to turn to avoid the obstacle
        turn_value = TURN * (values[7] > values[0] ? -1 : 1);
    }

    e_get_micro(&m1, &m2, &m3);
    if(m1 > SOUND1T && m2 > SOUND1T && m3 > SOUND1T) {
        LED0 = 1;
        swap();
        e_set_speed_left( 400 );
        e_set_speed_right( -400 );
    } else {
        LED0 = 0;
        e_set_speed( speed_value, turn_value );
    }

    e_start_agendas_processing();

    // delay
    for( k = 0; k < 3000; k++ ) {
        asm("nop");
    }
    // reset turning issue
    turn_value = 0;
    if( swapping ) {
        for( j = 0; j < 2000000; j++ ) {
            asm("nop");
        }
        e_set_speed( 0, 0 );
        for( j = 0; j < 1000000; j++ ) {
            asm("nop");
        }
        swapping = 0;
    }
}
}
}

```