Robotics Challenge 2015

Fabienne Gürtler
IN.2022 Robotics, BSc Course, 2nd Sem.
University of Fribourg
fabienne.guertler@unifr.ch

May 28, 2015

Introduction

This project requires multiple robots to work together to solve a problem. They communicate through events which they send each other.

1 Problem statement

There are 3 e-pucks in an arena. In this arena there are 2 blue doors and 4 red switches. To open a door a switch must be pressed, meaning to pass through a door two e-pucks have to work together. One finds either the door or the switch and waits in front of it while the other searches the missing component. Once both the door and the switch have been found the door opens and the e-puck can pass. Another requirement for the challenge is that all e-pucks must have the same code, except for calibration values. Before I begin presenting one of the many possible solution strategies I'll present the e-puck and the arena in detail.

E-puck The Ecole Polytechnique Fédérale de Lausanne started the e-puck project with the main goal to develop a miniature mobile robot for educational purposes. The design of the e-puck (Figure 1) is based on desktop size and flexibility. By default it comes with sound sensors, a 3D accelerometer, 8 proximity sensors and a camera. It is possible to extend the e-puck with more hardware like a Color LED Communication Turret, ground sensors, magnetic wheels and more. For this project we used e-pucks with ground sensors. The e-pucks communicate via Bluetooth. An e-puck is a event-based robot, it can both receive and emit events.

Figure 1: E-puck

The structure (Figure 2) is based on one single part to keep the desing simple and elegant. This basic part has a diameter of 7 cm and supports the motor, the circuit and the battery. The e-puck uses a miniature stepper motor with gear reduction and the wheels have a diameter of 4,1 cm.

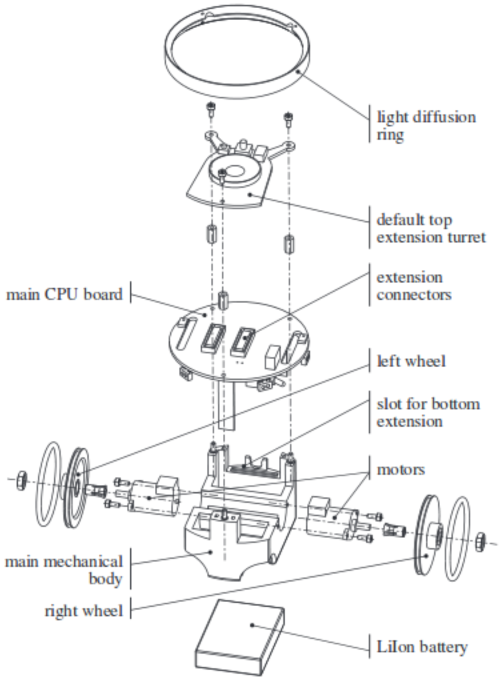

Figure 2: Mechanical structure

Arena The arena is a 80cm x 160cm zone divided into two subspaces of roughly the same area. Both the floor and the walls are white. In the area there are the following randomly placed objects:

- 2 blue doors, equipped with black stripes on the floor
- 4 red switches, two per subspace
- three epucks

Software For the completion of the project we used the help of two programs, Aseba Studio and Aseba Playground.
Aseba Playground simulates the arena, enabling us to test the robots behaviour virtually before trying to program the real robots. We did not use it very much because the color recognition works very differently in the simulation than in reality.
Aseba Studio is the programming environment used to program the e-pucks. It let's you load code onto one or more e-puck and execute it.

2 Solution Strategy

2.1 Basic Idea

The principle of our solution is to manage the e-puck's behaviour with the three variables *redFound, blueFound* and *color*. Figure 3 illustrates this idea.

redFound	blueFound	color	do	send
0	0		searchRB	-
0	1		searchR	-
0	1		wait	foundBlue
1	0		searchB	-
1	0		wait	foundRed
1	1		wait	-
1	1		wait	-
1	1		passDoor	passed

Figure 3: Coding for e-puck behaviour

While neither color has been found the e-puck searches for both. If the e-puck knows that one color was found but does not see it himself he searches for the missing color. If he is the one who found the color he waits in front of the object until released. It is easiest to use an example to illustrate:

redFound = True; blueFound = False; color = no color: The e-puck knows that the color red was found so he stops searching it and instead only looks for the color blue. Once he found red he sets *blueFound* to True and is in the constellation *redFound = True; blueFound = True; color = blue*. This means that both colors were found and he is standing in front of the door, which he can now begin to pass. As soon as that is done he emits the command *passed* which releases the variables, thus making the e-puck start searching again. The *do* column represents the states a e-puck can have and the send column are the events he sends to the other e-pucks.

This is the basic version of the project and it uses only two colors. It can already be used for three e-pucks as we defined a state where both colors were found but the e-puck sees no color. In this first version the third e-puck simply waits for the other to finish their routine. We later expanded the third e-puck's behaviour. Ideas for expansions with a third color are discussed later in this article. With this strategy it is relatively easy to expand the code for more colors.

2.2 Final strategy

Our first logic worked fairly well so we kept most of it. In fact, we only added some additional cases to better handle a third e-puck. The final logic is shown in Figure 4
We changed the behaviour for the third e-puck for when the other two have found both colors and started passing the door. Now the third e-puck no longer just waits until they are done, but now already starts looking for both colors again. If he finds a color he waits until they completed their procedure and then emits the event that he found a color. This speeds up the whole process because the third e-puck may already begin to search while the other are in the passing respectively the pressing the button state thus increasing the speed with which the objects are found. The process shown here is infinite so we also introduced a point system to define a end for the game. For successfully passing a door

redFound	blueFound	color	do	send	Description
0	0		searchRB	-	Nothing found
0	1		searchR	-	Other e-puck found blue
0	1		wait	foundBlue	This e-puck found blue
1	0		searchB	-	Other e-puck found red
1	0		wait	foundRed	This e-puck found red
1	1		searchRB	-	Third e-puck while other pass
1	1		wait	-	Press button
1	1		passDoor	passed	Pass door

Figure 4: Logic table

the e-puck gets 2 points and for pressing a switch he gets 1 point. We decided to give points to both participants but to give more points to the one passing the door because it is harder than pressing a switch. The e-puck that first reaches 7 points is the winner and emits the victory event upon which all e-pucks start "dancing" with the winner blinking all his leds.

2.3 State machine

From the logic table in Figure 4 we came up with our state machine (Figure 5).

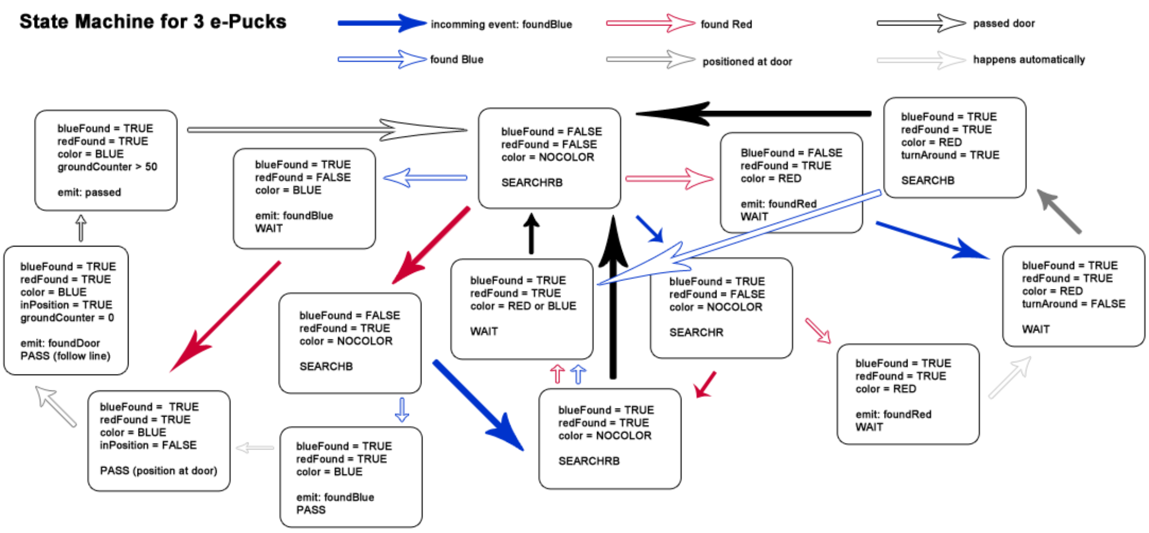

Figure 5: State Machine

States Our state machine has the following states:

- SEARCHRB: nothing found yet → search both colors
- SEARCHR: Someone else found blue → search red

- SEARCHB: Analogue searchR but with red
- PASS: Both colors found. In front of blue → pass the door
- WAIT: Either found object or press button → wait
- VICTORY: E-puck got at least 7 points → dance

Events

- foundRed: Camera recognized red and approached it → set redFound = True
- foundBlue: analogue foundRed
- passed: Passed the door → reset controlling variables to False
- victory: Emitter won → dance

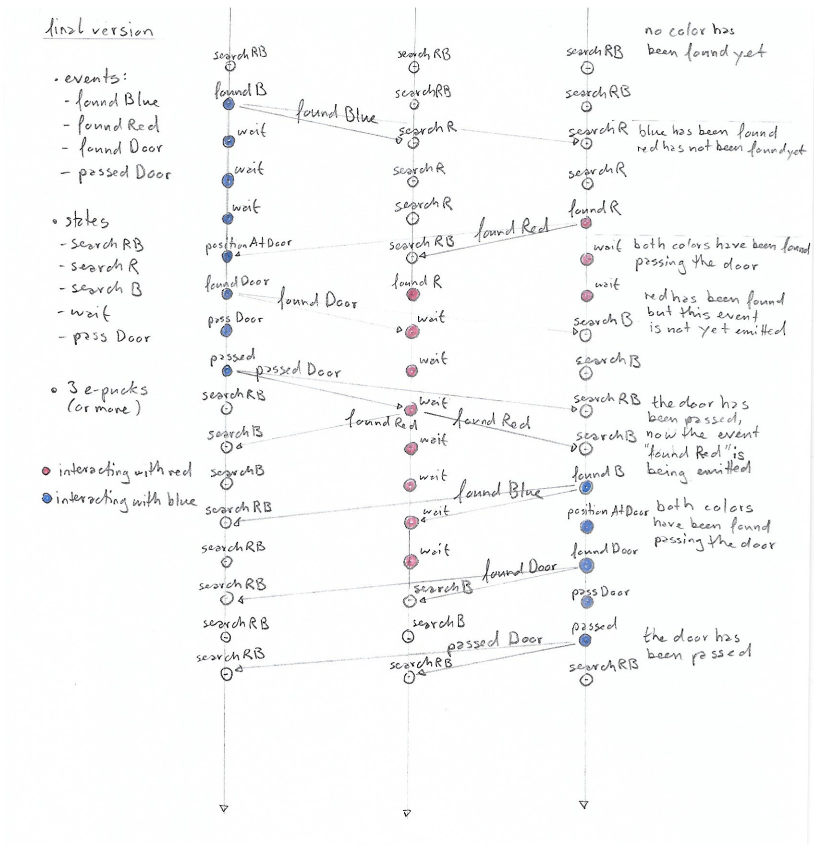

Figure 6: Event diagram

3 Implementation

The implementation is pretty straight forward from the state machine. In fact we had bigger problems to implement the behaviours themselves than to implement the main logic connecting them. Once we managed to get all subroutines to work we could see that our logic worked as intended.

Control variables The main controllers of our project are

- redFound: True if red was found by either himself or another e-puck. False otherwise
- blueFound: like redFound but for blue
- color: stores color he found. Used to check whether he is the finder of the object

Taking a look at the implementation, many commonalities to the logic table can be noted.

```
onevent ir_sensors
        [...]
        # both colors were found -> pass door
        elseif redFound == TRUE and blueFound == TRUE and
                color == BLUE then
                leds[0] = 1
                leds[2] = 0
                leds[4] = 0
                leds[6] = 0
                state = PASS
```

Secondary variables In addition to these main controller variables we introduced some helper variables. They are not an essential part of the logic but were needed to implement the behaviours.

- camColor: Color camera sees at this moment
- gameOver: True if one e-puck reached 7 points, false otherwise
- turnAround: helper variable used after press button to make e-puck turn.
- inPosition: True if e-puck found black stripe and is ready to pass the door

We had to differentiate between camColor and color because for example while passing the door the camera no longer sees the color blue once the door was taken away. To ensure it still stays in the pass state defined in the logic table color needs to stay blue. So most of the time camColor and color are one and the same but not always. It also helps with fluctuation when searching.
Once the door was passed all main controller variables are reset meaning all e-pucks start searching red and blue. Of course this means that the e-puck that pressed the button would immediately find red again since it is standing in front of it. We found that a bit boring so we used the turnAround variable to make the e-puck turn around once the door is open. To implement this we made the e-puck at the door emit an event foundDoor once he found the black strip (inPosition = True) and is ready to follow the line.

```
if not(v[0] < −50 and v[0] > −75) or
   not(v[2] < −50 and v[2] > −75) then
         [...]
         emit foundDoor
```

If the receiver is in front of the switch he sets turnAround to true, otherwise he doesn't react.

```
onevent foundDoor
         if color == RED then
                  [...]
                  turnAround = TRUE
         end
```

All the variable turnAround does is force the e-puck to search for blue while it is true. So from the moment the e-puck at the door sends that he is on the line until he sends the passed event the e-puck at the door searches for blue making him turn away from the switch. Of course this implies that the switch doesn't need to keep being pressed for the door to stay open.

3.1 Code structure

The main subroutines are searchRB, searchR, searchB, passDoor and wait. They represent the states. The code structure is shown in Figure 7 The only really new concepts are

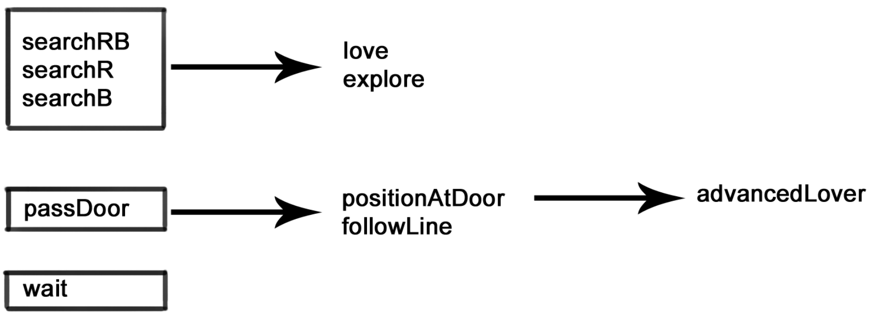

Figure 7: Subroutines

the positionAtDoor and the followLine subroutines, everything else is taken either from the sample solution of series 3 or is code form series 6. Which is why I will not discuss those in much detail.

3.2 Search functions

For obvious reasons the search algorithms are essential for the project. The best logic or follow line implementation is useless when the e-puck does not find the objects. In other words how good or bad the color recognition works plays a big role in the success or failure of the whole project. This means every time we wanted to work with the e-pucks in the arena, we had to recalibrate the color recognition beforehand. To do so used the color recognition of an earlier series and adjusted the calibration values. The search functions are quite simple to implement and with a good calibration they worked reliably.

```
sub searchRB
        if camColor == RED then
                callsub love

        elseif camColor == BLUE then
                callsub love

        else
                callsub explore
        end
```

The implementations for searchR and searchB work analogue. Once he spots the desired color he approaches it with the love behaviour, which makes him approach the object up to a certain distance. He then goes to the wait state and emits the corresponding found event if it has not been found already.

```
sub love
        [...] calculate proximity and delta-speed
        # approach object, stop when in certain distance
        if proxTotal < P_THRESH_LOVE then
        speed.right = SPEED_NORM - ds_left
        speed.left = SPEED_NORM - ds_right

        # only emit found when object wasn't found already
        else
                if camColor == BLUE and blueFound == FALSE then
                        color = BLUE
                        blueFound = TRUE
                        emit foundBlue
                [...]
                end
                callsub wait
        end
```

3.3 Pass Door

The passDoor subroutine is called when the e-puck is waiting in front of the door and the switch has been found. Since the e-puck doesn't necessarily stand on the line it first calls a getInPosition subroutine which is tries to move him onto the black strip. Once it is in position it calls the followLine subroutine.

The idea for the getInPosition function was to let the e-puck drive along the wall with the advanced lover until he stands on the line but it does not quite work as we

wanted. The problem is that because of the advanced lover it may get on the strip vertically. We could have let the e-puck make a 90 degree turn towards the door to fix this but unfortunately I did not think of this when we implemented it. This is probably the weakest part of our code and if we had more time I would definitely work on improving it.
The followLine subroutine works better than the positioning so if the e-puck stands in a good angle to the door it manages to pass the door. Like with the search algorithms a good calibration is very important to make a good line follower. We also used a short program to get the ground values and used that to find the range for black respectively white. Also the actual line following subroutine is only called once he is in position, meaning he has at least one wheel on the line. The line following itself is simple: if the ground sensor on the left (v[0]) leaves the stripe the e-puck turns right and if the sensor on the right leaves black it turns left. If both sensors leave the stripe it has passed the door. We used a variable groundCounter as a buffer because if both sensors are half on the stripe and half on the table it sometimes thinks both are on the table. Without the buffer it would then immediately send the passed event. Now every time both sensors are on white the groundCounter is incremented and once this counter reaches 50 it emits passed and starts searching again.

```
sub followLine
        call ground.get_values(v)
        # 0 & 2 not on line
        if (v[0] < -55 and v[0] > -70) and
           (v[2] < -55 and v[2] > -70) then
                groundCounter++

        # 0 not on line
        elseif (v[0] < -55 and v[0] > -70) then
                speed.left = 200
                speed.right = 70
                groundCounter = 0

        # 2 not on line
        elseif [...]

        else # both on line
                speed.left = 200
                speed.right = 200
        end

        if groundCounter > 50 then
                [...]
                emit passed
                end
        end
```

4 Validation

Our implementation more or less manages to solve the basic problem. A demonstration can be seen in the video we made. I'm generally happy with our solution because the state machine we designed in the beginning worked immediately. It's the implementation of the subroutines, especially positionAtDoor, that I'm not too pleased with. Our code works for 3 e-pucks, it might even work with more but we did not test it. As mentioned before we noticed that the correct calibration of the e-pucks is about the most important factor in the success of the problem. The best implementation is useless if the e-puck does not properly recognize colors or the black strip. We had the biggest problems with the passDoor subroutine and the ground sensors. We would have liked to add a third color but unfortunately we spent so much time on the implementation and recalibration process that we simply did not have enough time left. If we wanted to add a third color we probably would have needed to shorten our code, since we already use over 90% of the space available.

Conclusion

Our project can find both the door and the switch and pass it. It can handle multiple e-pucks without problems. It solves the basic problem with two colors. We had an idea for an additional color which I'll present as a thought experiment.

Third color We thought of adding some green switches to the arena that work like this: As soon as one object was found, if an e-puck sees green he pushes that switch which forces the other e-pucks to go search for a green switch and resets all the variables. Additionally if he manages to interrupt an e-puck while he is passing a door he gets all the points the others would have gotten, so he gets 3 points in that case. While the others are searching for the green switch, he can already go search for a door or a switch, increasing his chances to find a object before the others.

Personal Comments

I really enjoyed this course and liked that it is very project oriented and that we were able to experiment with the e-pucks. It is a nice addition to the mostly theoretical lectures I took this semester.

References

[1] Mondada F., Bonani M. i.a. *The e-puck, a Robot Designed for Education in Engineering*, 2009

[2] *Aseba User Manual*. https://aseba.wikidot.com/en:asebausermanual. Last visited: 20.05.15.

Appendix A Source Code

A.1 Advanced Love Behavior

The code below shows an e-puck implementing our solution for the project.

```
<!--node e-puck2-->
<node nodeId="3" name="e-puck2">
# color recognition
var camRed = 0
var camBlue = 0
var camGreen = 0

# counters
var i
var j = 0

#calibration
var r = 7
var b = 0
var g = 2
var w = -5

# proximity
var proxRight = 0
var proxLeft = 0
var proxTotal = 0
var ds
var ds_left
var ds_right

# ground sensor
var v[3]

# controllers
var redFound = FALSE
var blueFound = FALSE
var camColor = NOCOLOR
var color = NOCOLOR
var groundCounter = 0
var points = 0
var gameOver = FALSE
var turnAround = FALSE
var inPosition = FALSE
var state = WAIT

##############################################################
```

```
sub wait
    speed.left = 0
    speed.right = 0

sub advancedLover # code from sample solution
    # proximity
    proxRight = (prox[0] + 3*prox[1] + prox[2])/5
    proxLeft = (prox[7] + 3*prox[6] + prox[5])/5

    # check which side is closer to obstacle
    if proxRight > proxLeft then # turn left

        # delta speed ds
        call math.muldiv(ds, SPEED_NORM, proxRight, P_THRESH)

        # right and left speed
        speed.right = ds
        speed.left = SPEED_NORM - ds

    else # turn right

        # delta speed ds
        call math.muldiv(ds, SPEED_NORM, proxLeft, P_THRESH)

        # right and left speed
        speed.right = SPEED_NORM - ds
        speed.left = ds
    end

sub positionAtDoor
    call ground.get_values(v)

    # at least one of the sensors on line: e-puck in position
    if not(v[0] < -50 and v[0] > -75) or
       not(v[2] < -50 and v[2] > -75) then
        speed.left = 0
        speed.right = 0
        inPosition = TRUE
        emit foundDoor
    # follow wall
    else
        callsub advancedLover
    end

sub love # implemented according to Braitenberg
    # proximity
```

```
    proxLeft = (3*prox[0] + prox[1])/4
    proxRight = (3*prox[7] + prox[6])/4
    call math.add(proxTotal, proxLeft,proxRight)

    # delta speed ds
    call math.muldiv(ds_left,SPEED_NORM,proxLeft,P_THRESH)
    call math.muldiv(ds_right,SPEED_NORM,proxRight,P_THRESH)

    # approach object, stop when in certain distance
    if proxTotal < P_THRESH_LOVE then
    speed.right = SPEED_NORM - ds_left
    speed.left = SPEED_NORM - ds_right

    # only emit found when object wasn't found already
    else
        if camColor == BLUE and blueFound == FALSE then
            color = BLUE
            blueFound = TRUE
            emit foundBlue

        elseif camColor == RED and redFound == FALSE then
            color = RED
            redFound = TRUE
            emit foundRed
        end
        callsub wait
    end

sub explore # from sample solution
    # proximity
    proxRight = (4*prox[0] + 2*prox[1] + prox[2])/7
    proxLeft = (4*prox[7] + 2*prox[6] + prox[5])/7

    # check which side is closer to obstacle
    if proxRight > proxLeft then # turn left

        # delta speed ds
        call math.muldiv(ds, SPEED_NORM, proxRight, P_THRESH)

        # right and left speed
        speed.right = SPEED_NORM + ds
        speed.left = SPEED_NORM - ds

    else # turn right

        # delta speed ds
```

```
        call math.muldiv(ds, SPEED_NORM, proxLeft, P_THRESH)

        # right and left speed
        speed.right = SPEED_NORM - ds
        speed.left = SPEED_NORM + ds

    end

sub searchRB
    if camColor == RED then
        callsub love

    elseif camColor == BLUE then
        callsub love

    else
        callsub explore
    end

sub searchR
    if camColor == RED then
        callsub love

    else callsub explore
    end

sub searchB
    if camColor == BLUE then
        callsub love

    else callsub explore
    end

sub followLine
    call ground.get_values(v)
    # 0 & 2 not on line
    if (v[0] < -55 and v[0] > -70) and
       (v[2] < -55 and v[2] > -70) then
        groundCounter++

    # 0 not on line
    elseif (v[0] < -55 and v[0] > -70) then
        speed.left = 200
        speed.right = 70
        groundCounter = 0

```

```
    # 2 not on line
    elseif (v[2] < -55 and v[2] > -70) then
        speed.left = 70
        speed.right = 200
        groundCounter = 0

    else # both on line
        speed.left = 200
        speed.right = 200
    end

    if groundCounter > 50 then
        blueFound = FALSE
        redFound = FALSE
        color = NOCOLOR
        j = 0
        inPosition = FALSE
        groundCounter = 0
        emit passed
        points+=2
        if points >= 7 then
            emit victory
        end
    end

sub passDoor
    if j > 100 then
        if inPosition == TRUE then
            callsub followLine
        else
            callsub positionAtDoor
        end
    else
        callsub wait
    end

sub dance
    # if winner -> blink
    if points >= 7 then
        if j%5 == 0 then
            for i in 0:7 do
                leds[i] = 1
            end
        else
            for i in 0:7 do
                leds[i] = 0
```

```
            end
        end
    end

    speed.left = SPEED_NORM
    speed.right = -SPEED_NORM
##############################################################
onevent camera
    # calculation of colors, only takes center pixels.
    for i in 0:5 do
        camRed += cam.red[i+28]
    end
    camRed /= 6

    for i in 0:5 do
        camBlue += cam.blue[i+28]
    end
    camBlue /= 6

    for i in 0:5 do
        camGreen += cam.green[i+28]
    end
    camGreen /= 6

    if camBlue > 30+w and camGreen > 30+w and
       camRed > 30+w then
        camColor = NOCOLOR

    elseif 2*(camBlue -r) <= camRed and
           2*(camGreen -r) <= camRed then
        camColor = RED

    elseif 7*camRed <= 6*(camBlue +b) and
           3*(camBlue -b) <= 9*camRed and
           camGreen -b <= camBlue and not(camBlue > 20 and
           camRed > 20 and camGreen > 20) then
        camColor = BLUE

    else
        camColor = NOCOLOR
    end

onevent foundDoor
    if color == RED then
        points++
        if points >= 7 then
```

```
                emit victory
            end
            turnAround = TRUE
        end

onevent foundBlue
    blueFound = TRUE

onevent foundRed
    redFound = TRUE

onevent passed
    redFound = FALSE
    blueFound = FALSE
    color = NOCOLOR
    turnAround = FALSE

onevent victory
    gameOver = TRUE
#############################################################
onevent ir_sensors

# nothing found yet
    if redFound == FALSE and blueFound == FALSE then
        leds[0] = 0
        leds[2] = 1
        leds[4] = 0
        leds[6] = 1

        state = SEARCHRB

# someone else found blue -> search red
    elseif redFound == FALSE and blueFound == TRUE and
        color == NOCOLOR then
            leds[0] = 0
            leds[2] = 1
        leds[4] = 0
        leds[6] = 0

        state = SEARCHR

    elseif turnAround == TRUE then
        state = SEARCHB

# found blue -> wait infront of it
    elseif redFound == FALSE and blueFound == TRUE and
```

```
        color == BLUE then
        leds[0] = 0
        leds[2] = 1
        leds[4] = 1
        leds[6] = 1

        state = WAIT

# someone else found red
    elseif redFound == TRUE and blueFound == FALSE and
        color == NOCOLOR then
        leds[0] = 0
        leds[2] = 0
        leds[4] = 0
        leds[6] = 1

        state = SEARCHB

# found red -> wait
    elseif redFound == TRUE and blueFound == FALSE and
        color == RED then
        leds[0] = 0
        leds[2] = 1
        leds[4] = 1
        leds[6] = 1
        state = WAIT

# both colors were found -> press button
    elseif redFound == TRUE and blueFound == TRUE and
        color == RED then

        leds[0] = 1
        leds[2] = 1
        leds[4] = 1
        leds[6] = 1
        state = WAIT

# both colors were found -> pass door
    elseif redFound == TRUE and blueFound == TRUE and
        color == BLUE then
        leds[0] = 1
        leds[2] = 0
        leds[4] = 0
        leds[6] = 0
        state = PASS

```

```
# both colors found by other e-pucks -> searchRB
    elseif redFound == TRUE and blueFound == TRUE and
           color == NOCOLOR then
        state = SEARCHRB

# should not happen, but to be sure:
    else
        for i in 0:7 do
            leds[i] = 1
        end
        state = WAIT
    end

# e-puck won -> make victory dance
    if points >= 7 then
        state = DANCE
    end

# other e-puck won -> gameOver dance
    if gameOver == TRUE then
        state = DANCE
    end

# call corresponding functions
    if state == SEARCHRB then
        callsub searchRB

    elseif  state == SEARCHR then
        callsub searchR

    elseif state == SEARCHB then
        callsub searchB

    elseif state == PASS then
        j++
        callsub passDoor

    elseif state == DANCE then
        j++
        callsub dance

    else
        callsub wait
    end</node>
```