E-Puck Behaviour Implementation

Robotics and Intelligent Systems

COP518

Authors:
Ross McQuillan
Amr Yousef
Xinyu Chen

Supervisor:
Dr. Qinggang Meng

February 18, 2016

Contents

1 Introduction

Within the Robotics and Intelligence systems module groups worked together to implement different behaviours with an E-Puck. Many different behaviours were available for implementation, including attraction, hate and aggression.

This report will explain, in detail, the tasks set out before each group. It will progress by explaining the design established to tackle these objectives, the implementation of the design, testing and results before concluding the coursework and evaluating the groups achievements.

The tasks given to each group were split into two separate parts. The first was to develop a behaviour on a single E-Puck. Such initial behaviours proposed by Dr. Meng, included love where the E-Puck would follow close to, but without touching, an object or, obstacle avoidance, where the E-Puck would try to avoid objects. More of the behaviours will be spoken about throughout this report in greater detail.

The second task included using two E-Pucks. One of the E-Pucks was tasked with following a human hand but not other objects, whilst the other was to follow the first E-Puck closely without touching it. This second E-Puck would imitate the love behaviour spoken about previously.

2 Background

The robotics industries association[1] defines a robot "a re-programmable, multi-functional, manipulator designed to move material, parts, tools or specialised devices through variable programmed motions for the performance of a variety of task"[3]. According to that definition any device that is programmed to preform variety of tasks is classified as a robot. Most robots consist of several hardware components such as:

- Physical body
- Actuators
- Sensors
- Controller
- Processor
- Software

These components are commonly used when designing and developing a robot. A good design is considered to be one which best utilizes these components to reach it's goals effectively and efficiently.

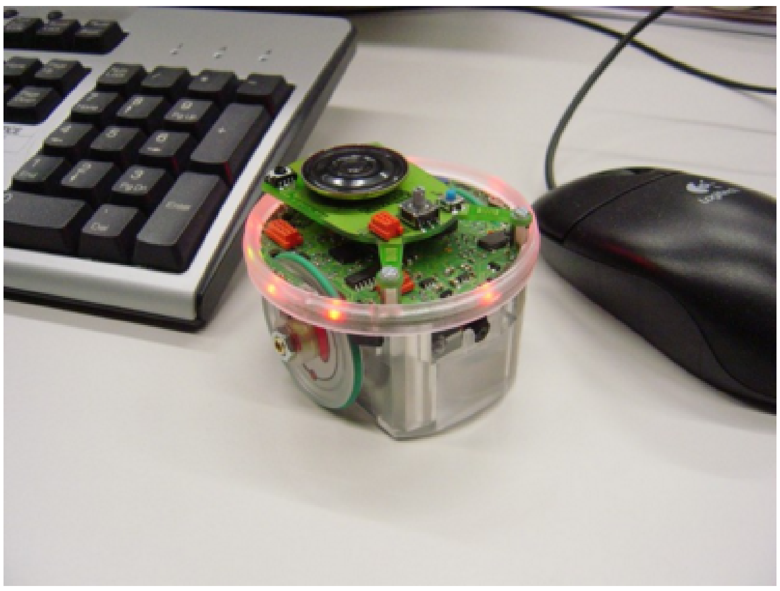


Figure 1

2.1 Behaviours

Many different behaviours can be seen in day to day life. If one was trying to describe a behaviour as simply as possible, one might say a behaviour is a "mapping of sensory inputs to a pattern of motor actions that are used to achieve a task"[4]. The behaviours robots wish to imitate include all of the common human behaviours. Getting the robots to show a very basic version of this behaviour, such as love, would require the robot to follow nearby object in a show of attraction. Naturally, the main goal in cutting edge research is to develop these skills such that they are indistinguishable to the fully fletched human behaviours.

2.2 E-Puck

An E-Puck[5] is a small sized, educational robot which is designed to allow developers to code on and learn about robotics. The robot is very inexpensive, user-friendly and interactive. It was designed by Martin Stefanec from the artificial life lab of University of Graz who made all the software developed for the E-Puck open source.

It includes a wide range of sensors and actuators. Sensors include infrared-sensors which can be used to detect objects and ambient light, motors to control the movement of the robot and for communication the robot is has 8 LEDs, a speaker and bluetooth.

2.3 HSV

HSV stands for Hue, Saturation and value. It's a cylindrical representation of colours, starting at red in 0 degrees, green at 120 degrees and blue at 240 degrees.

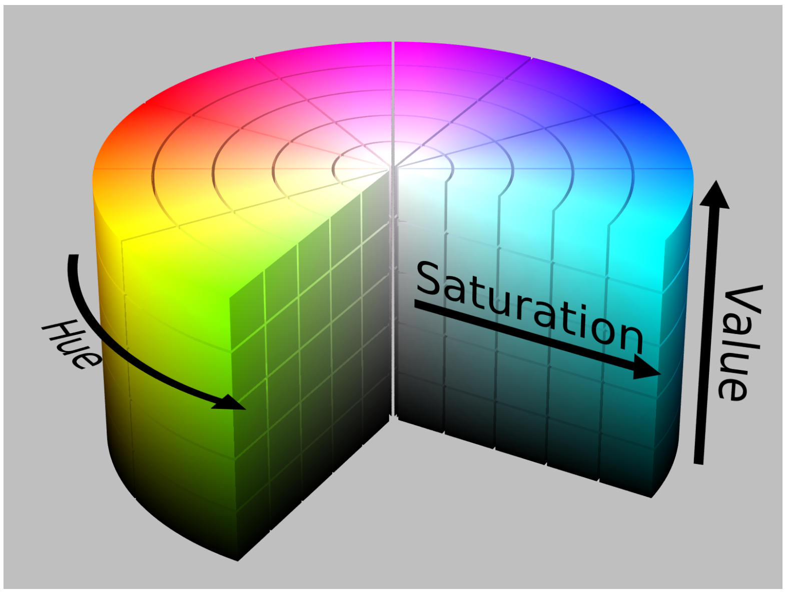


Figure 2: HSV Colour Space

Using figure 2 one can recognise that the colour space's saturation is most intense in the centre of the cylinder. The saturation decides how washed out the colours appear. The final component value, which increases the higher up the cylinder the pixel value is, decides how much colour or how vibrant each colour appears.

2.4 Skin Colour

Skin detection in HSV tends to be more accurate than RGB. As the colour our Hue in HSV trends to be affected much less be external parameters such as ambient light or direct sunlight.

After careful research a paper found on human skin colour[2], in the HSV colour plane, found that the skin colour of the majority of people across the globe lies within the range 6^o - 38^o. The paper when on to discuss how to continue to improve the reliability of finding skin tones within an image although this is beyond the scope of this project as it would make the functions to be described much more computationally expensive.

3 Design

This section will explain how both tasks A and B will be completed, giving diagrams and flowcharts of how the E-Pucks will carry out a particular task.

3.1 Task A

To begin designing a solution to the tasks set out in section 1, one must first decide upon the different behaviours to be implemented for the tasks. A relatively simple task to be implemented, which would be beneficial to both the first and second task, would be to implement a love behaviour.

Section 2 discussed how each behaviour would perform, illustrating that the love behaviour would follow but not touch the leading object. A second behaviour to be implemented would be an adaption of the aforementioned love behaviour, although instead of the E-Puck following an object it would instead follow a bright light placed before it. The E-Puck would use both the camera to discover any spots of light that have a much higher intensity than its surroundings.

One third and final possible behaviour to implement, providing time permits the group to do so, would be to to implement a obstacle avoidance behaviour. The behaviour which attempts to avoid any nearby objects whilst moving in a particular direction. When it approaches an object it changes its direction to move away from the object.

The first and final behaviours rely heavily on the infrared senses situated on around the edge of the E-Puck. The E-Puck will read in from the infrared sensors, each returning a voltage reading to the chip. This reading can then be used within a formula to calculate the distances to a nearby object. When an object is discovered nearby, the E-Puck must either turn to face it or alter it's direction to move away from it, for the love and obstacle avoidance behaviours respectively. The E-Puck is then required to move forward at a fixed speed until either the object is very close, but not touching for the love behaviour or no longer detected for the hate behaviour.

To create the second behaviour of an attraction to light the E-Puck must be implemented in a similar way to the previously mentioned behaviours, although instead of reading the infrared sensor value for distance one will measure the amount of ambient light falling onto the device.

3.2 Task B

Task B requires the group to implement 2 different E-Pucks and cause each to do their own tasks. One E-Puck will be required to follow a hand whilst the other E-Puck will be required to follow the first E-Puck. This may seem similar after implementing the behaviours from task A, although many issues could arise with this task. These issues will be discussed toward the end of this section.

Since the love behaviour will have already been implemented once the implementation of task B begins it would be logical to make use of these behaviours to save time. Therefore to obtain the the desired output of the second E-Puck (the following of the first E-Puck), one may use this behaviour and achieve the required result.

For the first E-Puck, the E-Puck which will be following a human hand, the group will have to use both the camera and infrared sensors on the E-Puck and consult the research performed within section 2 surrounding the colour of human skin.

The E-Puck will begin by discovering any near-by objects using the infra-red sensors before turning the front camera towards any discovered objects. The front

Selector	Function
0	run_breitenberg_follower
1	finding_light
2	avoid_light
3	run_breitenberg_shocker
4	followHand

Table 1: A table to show what function is ran for each selector.

camera will then read in the image of the object. Removing a few rows of pixels from the image, for fast processing, the pixels will then be processed to find any required colour that the E-Puck is looking for, the colour of a hand. This processing will be done in the HSV colour plane as to reduce the impact lighting has on the colour of an image.

3.2.1 Flowchart

4 Implementation

This section will describe how each of the tasks mentioned in section 3 have been implemented, explaining the functions within the code, and any discrepancies between the design and final implementation.

The design was gradually implemented in stages, building up to the final completed solution to each task. To make the project easier to implement the demo project provided on the E-Puck[5] website was used as a template.

4.1 Main File

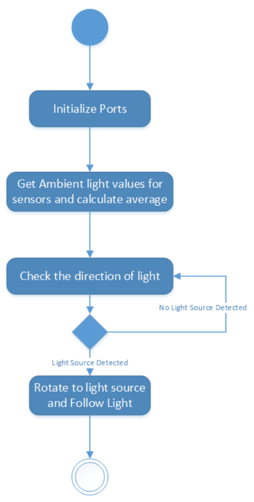
Within the demo project one can find the main file. This main file starts by finding the position of the selection, a single hexadecimal digit. This selector then decides which function will be called. This becomes very useful to the groups project as then each behaviour needed to be implemented can be ran when a unique selector is selected. Table 1 clearly shows the possible choices to the user with the code being displayed in the code snippet 1.

The functions shown in table 1 will each be discussed in detail throughout this section.

4.2 Breitenburg Follower

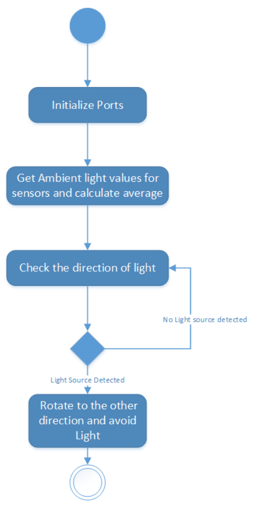
The 'run_breitenburg_follower' function is a function that was taken from the demo project mentioned previously. The function can be found in the 'runBreitenberg_adv.c' file along with it's dependencies.

The function uses the front 4 infrared sensors to attempt to follow an object, reading in the distance to the object and calculating the linear and angle speeds using the functions shown in code snippets 3 and 2.



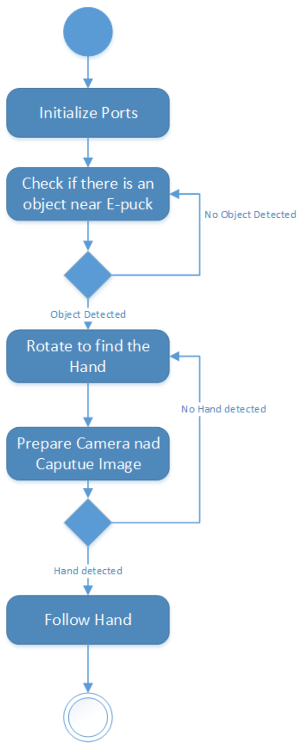
Initialize Ports
Get Ambient light values for sensors and calculate average
Check the direction of light
No Light Source Detected
Light Source Detected
Rotate to light source and Follow Light

(a) Flow chart describing the follow light function



Initialize Ports
Get Ambient light values for sensors and calculate average
Check the direction of light
No Light source detected
Light Source Detected
Rotate to the other direction and avoid Light

(b) Flow chart describing the avoid light function



(c) Flow chart describing the follow hand function

Figure 3: Flow charts describing 3 of the different behaviours implemented within the E-Puck

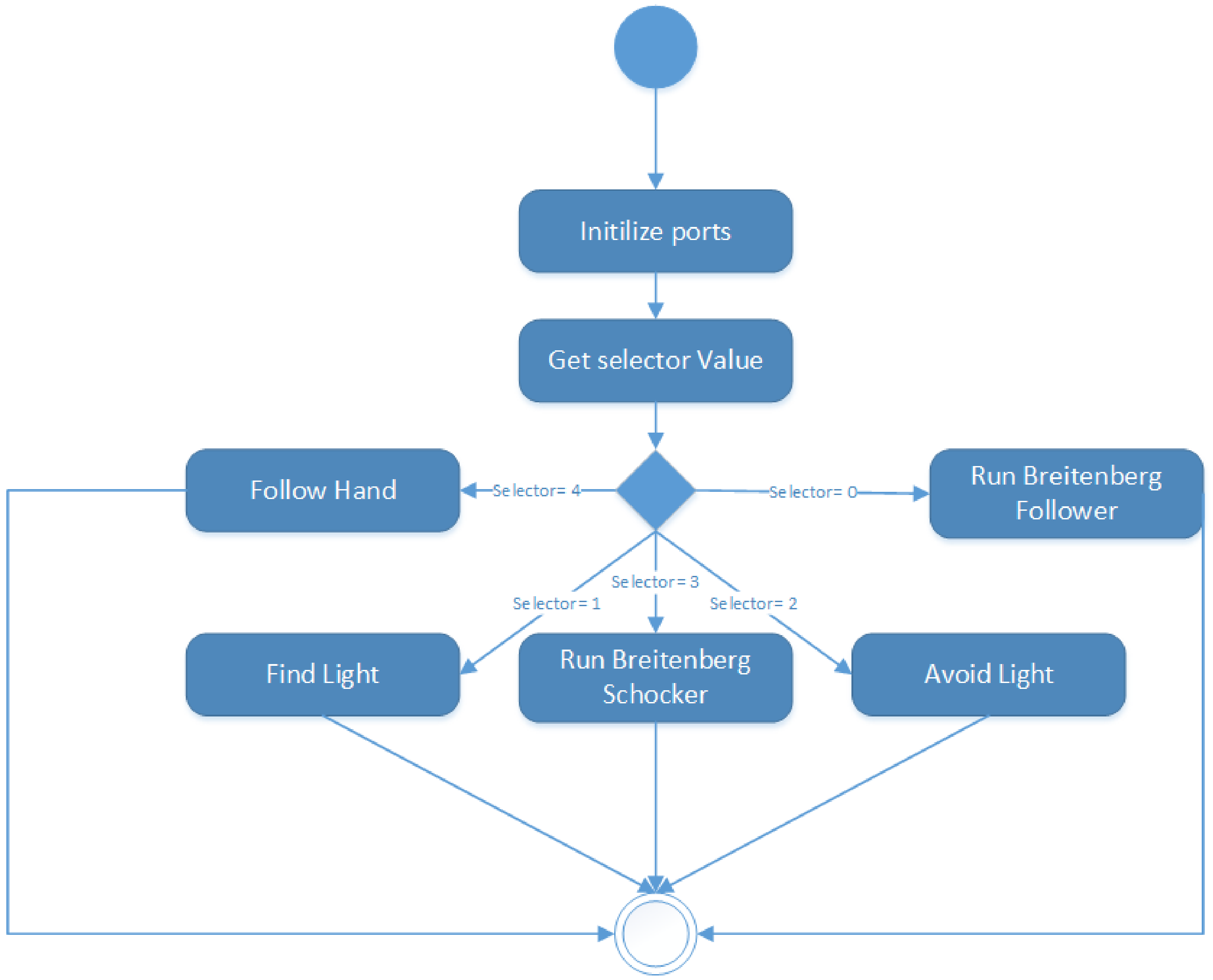


Figure 4: Flow chart describing how the main function will be implemented

```
        selector=getselector();
        if (selector==0) {
                run_breitenberg_follower();
        } else if (selector==1) {
                finding_light();
        } else if (selector==2) {
                avoid_light();
        } else if (selector==3) {
                run_breitenberg_shocker();
        } else if (selector==4) {
                followHand();
        } else{
        }
```

Code Snippet 1: Processes for selecting which function to run

The angular speed is calculated by passing the function the difference in the readings between the 2 front left infrared sensors and the 2 front right infrared sensors. The magnitude of this value then decides how quickly the robot must turn left or right, whilst whether the value is positive or negative decides which way the robot should turn.

The linear speed calculator works in a similar way where the average of the front 2 middle infrared sensors are passed into the function and dependent upon the magnitude of this value the linear speed can then be calculated.

These values are then added together for the speed of the right wheel and the angular speed is subtracted from the linear speed for the left wheel causing the E-Puck to move in the desired way.

4.3 Following Light

The 'finding_light' function finds light using the infrared sensors once again, although instead of detected the emitted infrared light the sensors detect the ambient light.

One anticipated obstacle for this function would be for different environments the E-Puck would behave very differently. For example, in very dark environments the E-Puck will find it very easy to find a light source. Although, in brighter environments the E-Puck will find it much more difficult to find the light source being shone onto it. To attempt to overcome this problem a baseline reading of the environment is taken before the E-Puck begins to look for any light source. From here the E-Puck can compare any future readings with the baseline readings and thus find any increases in light, being any newly introduced light sources.

```
int angle_speed_calc(int pos, int gain)
{
        int consigne = 0;
        int angle_speed = 0;
        int ecart = consigne - pos;

        angle_speed = ecart*gain;

        if(angle_speed >= 1000)
                angle_speed = 999;
        else if(angle_speed <= -1000)
                angle_speed = -999;

        return angle_speed;
}
```

Code Snippet 2: Functions for calculating the angular speed of the E-Puck

4.4 Avoiding Light

The 'avoid_light' function works in a very similar way to the 'finding_light' function although instead of being attracted to the light source it moves in the opposite direction.

4.5 Breitenburg Shocker

Once again, the 'run_breitenberg_shocker' function runs in a very similar way to the 'run_breitenberg_follower' but instead of moving towards an objected when one is discovered on the front infrared sensors the E-Puck turns and moves away from it.

4.6 Follow Hand

As one may realise the 'followhand' function is more complex and therefore contains more problems to solve. The idea behind the function is to check whether there is object nearby the front 2 infrared sensors, if so the camera takes an image and checks whether or no the image is a hand. The camera does this by checking the colour of the image and recognising whether it is the range of the hue of a hand.

The first stage of this function is very simple and can be done easily, in the same way as previously seen functions by just reading the distance from an infrared sensor on the E-Puck.

These sensors then trigger whether an image should be taken. If the image is to be taken then it is done so by calling the 'hgetImage' function which has been taken from the aforementioned demo project and renamed. The following 'hImage'

```c
int lin_speed_calc(int distance, int gain)
{
        int consigne = 100;
        float h = 0.05;
        int ti = 3;
        int ecart = consigne-distance;
        int lin_speed;

        ui_lin = ui_lin + h * ecart / ti;
        lin_speed = (ecart + ui_lin) * gain;

        if(lin_speed >= 1000)
        {
                ui_lin = 999/gain - ecart;
                if(ui_lin > 60)
                        ui_lin = 60.0;
                lin_speed = 999;
        }
        else if(lin_speed <= -1000)
        {
                ui_lin = -999/gain + ecart;
                if(ui_lin < -10)
                        ui_lin = -10.0;
                lin_speed = -999;
        }
        return lin_speed;
}
```

Code Snippet 3: Functions for calculating the linear speed of the E-Puck

```
//RGB turned into an integer value for comparison
red = (hbuffer[2*i] & 0xF8);
green = (((hbuffer[2*i] & 0x07) << 5) | ((hbuffer[2*i+1]
    & 0xE0) >> 3));
blue = ((hbuffer[2*i+1] & 0x1F) << 3);
// convert to RGB to Hue
double hue = RGB2Hue(red, green, blue);
if(hue> 6 && hue<38){
        hnumbuffer[i] = 1;
        vis +=1;
}else{
        hnumbuffer[i] = 0;
}
```

Code Snippet 4: Converting RGB pixels to HSV

function then begins to process the image, firstly converting the RGB image to HSV. The code and algorithm for this can be seen in code snippet 4. One may notice that only the hue has been calculated and this is because only the hue is needed as the saturation and value intensity are both environment dependent. Therefore finding a hand in a brightly light room or a poorly light room should obtain similar results.

Once the hue has been calculated the function proceeds to categorise each pixel as a 1 if the hue lies within the range of a human hand or a 0 otherwise. The range the hue must lie within starts at 6^o through to 38^o. These values have been calculated after careful research was carried out on human skin colour, referred to in section 2.

The final stage of the function is to check the amount of times this hue has been seen and calculate a threshold where the E-Puck decides whether human skin is visible or not. This threshold was discovered after testing various threshold values which will be analysed in more detail throughout section 5.

5 Testing & Results

This section will first look at the testing that was completed throughout the project to ensure all the processes the were implemented within the E-Puck function as expected. Secondly this section will analyse the results and outcome obtained from running the completed project on the E-Pucks.

5.1 Testing

Testing took place throughout the entirety of the implementation stage, before more in-depth testing took place once the implementation had been completed.

This testing could be completed in such a way because of the ‘AGILE’ work flow which took place, where the code development would be completed in stages and gradually built up. Therefore each stage can be partially tested after it had been completed.

Firstly, once the main file had been completed, instead of calling the required function, the function name was printed to the terminal. This allowed the group to test if the selector is working properly on the device.

The next heavily tested stage took place when implementing the follow and avoid light functions. This function required careful altering of the variables which set the threshold for finding a nearby light source. This threshold was found using trial and error changing the variable to fit better in all environments as opposed to optimising it for a single environment.

The final function to be tested throughout testing, which also required careful tweaking to complete, was the ‘followHand’ function. Within this function the threshold to evaluate whether the object in front of the camera is a hand or not must be carefully assessed. This was done once again by trial and error, obtaining a the value by improving the last after each test.

One major improvement made because of these tests was that because the image is only taken when an object is found in front of the camera using the infrared sensor much more of the image could be processed and thus finding a hand becomes much more accurate.

Throughout the final testing of the E-Puck, the E-Puck was ran in a range of scenarios and expected to complete the tasks set out in the introduction of this report. One of the major problems found whilst testing was that the E-Puck would find the hand in front of the camera but then immediately lose track of it again and become stationary. Whilst lowering the threshold for this task solved the problem it also introduced a new one where the E-Puck would follow much more devices not consisting of the colour of human skin

After many different attempts to solve the problem a simple yet effective solution was put into place where once a hand is detected in front of the camera, the E-Puck will then continue to follow the object whilst the object is in range of the infrared sensors. This stops the E-Puck repeatedly taking pictures with the camera and performing costly algorithms on them. This method also greatly reduces battery power and is much more computationally inexpensive for the E-Puck to run.

5.2 Results

Throughout the testing of the software implemented within the E-Puck, analysis of the performance of both E-Pucks were also taken. Each behaviour was found to perform particularly well when running individually. With the E-Pucks performing exactly as expected.

One strange anomaly found whilst testing the E-Puck is that the infrared sensors do not pick up the ambient light produced from a single LED, the flash on a smart phone for example, very well. Whilst this was an issue with the E-Puck itself there was not much the group could do to improve the results in this situation. For all other sources of light, providing they were not too dim, the E-Puck responded very well turning and moving towards the light source.

One other small issue found whilst testing the E-Pucks was with the second E-Puck that was to follow the first. The E-Puck would very often lose track of the first E-Puck, which is possibly down to the speed at which they are traveling with respect to each other. Although, the maneuver where the first E-Puck turns fairly quickly increases the probability of the second E-Puck losing track of it. To try to reduce the frequency that this error occurred the speeds were altered to find an optimum speed at which the 2 E-Pucks operated best together.

Overall, all of the features preformed very well on the majority of the tests, preforming much better than expected at finding hands with different skin tones and a light source in different lighting environments.

6 Conclusion

The conclusion will begin by discussing any improvements that could be made to the projects or anything that the group would change whilst completing the project. It will follow by stating what was done well by the group and what could have been improved to achieve a better result.

6.1 Improvements

Whilst the group worked to the best of their ability to produce the work described throughout this report there are a couple of improvements that the group would have liked to implement given more time. One of these improvements would be to correct the issue described towards the end of section 5.2. The issue was that the second E-Puck struggled to follow the first E-puck consistently.

To attempt to reduce the effect of turning the first E-Puck too quickly and the second E-Puck losing track of it, one may wish to incorporate more than just the front 4 infrared sensors on the second E-Puck. This would allow the second E-Puck to detect if the first E-Puck has turned significantly enough to pass the side of the second E-Puck and thus allowing better detection. Whilst this seems relatively simple, because of the complex formula used to move the E-Puck in the demo ‘run_breitenburg_follower’ function, it becomes much more complicated to achieve the ideal situation.

One other improvement the group could have made collectively throughout the entire project was to ensure better communication was established between each group member to allow everybody to know what was being developed at each stage. This would lead to faster cooperation on tasks and a greater understanding by all.

Overall this project has been a success in achieving what it had been set out to achieve, with each group member contributing towards the end target goal. Each task has been successfully solved with only small errors being produced in some of the multiple solutions.

A Code

A.1 Main

A.1.1 C File

```c
#include "p30f6014A.h"
#include "stdio.h"
#include "string.h"

#include "uart/e_uart_char.h"
#include "motor_led/e_init_port.h"
#include "motor_led/e_epuck_ports.h"
#include "motor_led/advance_one_timer/e_motors.h"
#include "motor_led/advance_one_timer/e_agenda.h"
#include "finding_Light.h"
#include "avoidlight.h"
#include "imageCapture.h"
#include "findRed.h"
#include "followGreen.h"
#include "followHand.h"
#include "runBreitenberg_adv.h"

int getselector() {
        return SELECTOR0 + 2*SELECTOR1 + 4*SELECTOR2 +
           8*SELECTOR3;
}

int main() {
//      char bufferDebug[80];
        int selector;
//init
        e_init_port();
        e_init_uart1();
        e_init_motors();

        selector=getselector();

        if (selector==0) {
                run_breitenberg_follower();
        } else if (selector==1) {
                finding_light();
        } else if (selector==2) {
                avoid_light();
        } else if (selector==3) {
                run_breitenberg_shocker();
        } else if (selector==4) {
                followHand();
        }else{
        }


        while(1);
}
```

A.2 finding_Light

A.2.1 Header File

```
#ifndef _FLIGHT
#define _FLIGHT

void finding_light(void);

#endif
```

A.2.2 C File

```
#include <p30f6014A.h>
#include <stdlib.h>//for random numbers
#include <stdio.h>//for random numbers
#include "a_d/advance_ad_scan/e_prox.h"
#include "motor_led/e_init_port.h"
#include "motor_led/e_epuck_ports.h"
#include "motor_led/advance_one_timer/e_motors.h"
#include "motor_led/advance_one_timer/e_agenda.h"
#include "a_d/advance_ad_scan/e_ad_conv.h"
#include "runBreitenberg_adv.h"
#include "string.h"
#include "math.h"
#include <uart/e_uart_char.h>

void finding_light (void) {

        e_start_agendas_processing();

        int value0_A, value1_A, value2_A, value3_A,
           value4_A, value5_A, value6_A, value7_A,
           averageLight;
        long k;
```

```
//char bufferDebug[40];
e_init_port();
e_init_ad_scan(ALL_ADC);
e_init_motors();
LED7 = LED6 = LED0 = LED2 = LED4 = LED1 = LED3 =
    LED5 = 1;

value0_A = e_get_ambient_light(0);
value1_A = e_get_ambient_light(1);
value2_A = e_get_ambient_light(2);
value3_A = e_get_ambient_light(3);
value4_A = e_get_ambient_light(4);
value5_A = e_get_ambient_light(5);
value6_A = e_get_ambient_light(6);
value7_A = e_get_ambient_light(7);

averageLight = (value0_A+value1_A+value2_A+
   value3_A+value4_A+value5_A+value6_A+value7_A)
   /8-200;
for(k=0;k<20000;k++){}
while (1) {

        value0_A = e_get_ambient_light(0);
        value1_A = e_get_ambient_light(1);
        value2_A = e_get_ambient_light(2);
        value3_A = e_get_ambient_light(3);
        value4_A = e_get_ambient_light(4);
        value5_A = e_get_ambient_light(5);
        value6_A = e_get_ambient_light(6);
        value7_A = e_get_ambient_light(7);
        if ((value0_A+value7_A)/2 < averageLight
           -100) {
                LED0 = LED7 = 1;
                LED1 = LED2 = LED3 = LED4 = LED5
                    = LED6 = 0;
                e_set_speed_left(800);
                e_set_speed_right(800);
                for(k=0; k<100000; k++) {}
        }
        else if ((value0_A+value1_A)/2 <
           averageLight) {
                LED0 = LED1 = 1;
                LED1 = LED3 = LED4 = LED5 = LED6
                    = LED7 = 0;
                e_set_speed_left(800);
```

```
            e_set_speed_right(400);
            for(k=0; k<100000; k++) {}
    }
    else if ((value1_A+value2_A)/2 <
        averageLight) {
            LED1 = LED2 = 1;
            LED0 = LED3 = LED4 = LED5 = LED6
                = LED7 = 0;
            e_set_speed_left(400);
            e_set_speed_right(-400);
            for(k=0; k<100000; k++) {}
    }
    else if ((value2_A+value3_A)/2 <
        averageLight) {
            LED2 = LED3 = 1;
            LED0 = LED1 = LED4 = LED5 = LED6
                = LED7 = 0;
            e_set_speed_left(600);
            e_set_speed_right(-600);
            for(k=0; k<100000; k++) {}
    }
    else if ((value3_A+value4_A)/2 <
        averageLight) {
            LED3 = LED4 = 1;
            LED0 = LED1 = LED2 = LED5 = LED6
                = LED7 = 0;
            e_set_speed_left(800);
            e_set_speed_right(-800);
            for(k=0; k<100000; k++) {}
    }
    else if ((value7_A+value6_A)/2 <
        averageLight) {
            LED7 = LED6 = 1;
            LED0 = LED1 = LED2 = LED3 = LED4
                = LED5 = 0;
            e_set_speed_left(400);
            e_set_speed_right(800);
            for(k=0; k<100000; k++) {}
    }
    else if ((value6_A+value5_A)/2 <
        averageLight) {
            LED6 = LED5 = 1;
            LED0 = LED1 = LED2 = LED3 = LED4
                = LED7 = 0;
            e_set_speed_left(-400);
```

```
                                e_set_speed_right(400);
                                for(k=0; k<100000; k++) {}
                        }
                        else if ((value5_A+value4_A)/2 <
                           averageLight) {
                                LED5 = LED4 = 1;
                                LED0 = LED1 = LED2 = LED3 = LED6
                                    = LED7 = 0;
                                e_set_speed_left(-600);
                                e_set_speed_right(600);
                                for(k=0; k<100000; k++) {}
                        }
                        else {
                                LED7 = LED6 = LED0 = LED2 = LED4
                                    = LED1 = LED3 = LED5 = 1;
                                e_set_speed_left(0);
                                e_set_speed_right(0);
                                for(k=0; k<100000; k++) {}
                        }
        }
}
```

A.3 avoidlight

A.3.1 Header File

```
#ifndef _AVOID_LIGHT
#define _AVOID_LIGHT

void avoid_light(void);

#endif
```

A.3.2 C File

```
#include <p30f6014A.h>
#include <stdlib.h>//for random numbers
#include <a_d/e_prox.h>
#include <motor_led/e_init_port.h>
#include <motor_led/e_motors.h>
#include <a_d/e_ad_conv.h>
#include "motor_led/advance_one_timer/e_agenda.h"
#include <stdlib.h>//for random numbers #include "stdio.
   h"
```

```
#include "string.h"
#include "math.h"
#include <motor_led/e_epuck_ports.h>
#include <motor_led/e_led.h>

#include <uart/e_uart_char.h>


void avoid_light (void) {

        int value0_A, value1_A, value2_A, value3_A,
           value4_A, value5_A, value6_A, value7_A,
                value0, value1, value2, value3, value4,
                   value5, value6, value7;
        long k;
        //char buffer[1];
        e_start_agendas_processing();

        while (1) {

                value0 = e_get_prox(0);
                value1 = e_get_prox(1);
                value2 = e_get_prox(2);
                value3 = e_get_prox(3);
                value4 = e_get_prox(4);
                value5 = e_get_prox(5);
                value6 = e_get_prox(6);
                value7 = e_get_prox(7);

                value0_A = e_get_ambient_light(0);
                value1_A = e_get_ambient_light(1);
                value2_A = e_get_ambient_light(2);
                value3_A = e_get_ambient_light(3);
                value4_A = e_get_ambient_light(4);
                value5_A = e_get_ambient_light(5);
                value6_A = e_get_ambient_light(6);
                value7_A = e_get_ambient_light(7);

                if (value6 > 1000 && value7 > 1000) {
                        e_set_speed_right(200);
                        e_set_speed_left(-200);
                        for(k=0; k<30000; k++) { asm("
                           nop"); }
                } else if (value2 > 1000 || value0 >
                   1000 || value1 > 1000 || value7 >
```

```
            1000) {
                    e_set_speed_right(-200);
                    e_set_speed_left(200);
                    for(k=0; k<30000; k++) { asm("
                        nop"); }
        } else if (value5 > 1000 || value6 >
            1000) {
                    e_set_speed_right(200);
                    e_set_speed_left(-200);
                    for(k=0; k<30000; k++) { asm("
                        nop"); }
        } else if (value2 < 1000 && value0 <
            1000 && value1 < 1000 && value7 <
            1000 && value5 < 1000 && value6 <
            1000) {
                    if ( ((value1_A < 4000) || (
                        value2_A < 4000)) && ((
                        value0_A < 4000) || (value7_A
                         < 4000)) ) {
                            LED0 = LED7 = 1;
                            LED1 = LED2 = LED3 =
                                LED4 = LED5 = LED6 =
                                0;
                            e_set_speed_left(-800);
                            e_set_speed_right(-800);
                            for(k=0; k<1000000; k++)
                                 { asm("nop"); }
                    } else if ( (value1_A < 4000) ||
                         (value2_A < 4000) ) {
                            LED0 = LED7 = LED3 =
                                LED4 = LED5 = LED6 =
                                0;
                            LED1 = LED2 = 1;
                            e_set_speed_left(-500);
                            e_set_speed_right(500);
                            for(k=0; k<1000000; k++)
                                 { asm("nop"); }
                    } else if ( (value3_A < 3000) ||
                         (value4_A < 3000) ) {
                            LED0 = LED7 = LED1 =
                                LED2 = LED5 = LED6 =
                                0;
                            LED3 = LED4 = 1;
                            e_set_speed_left(-500);
                            e_set_speed_right(500);
```

```
                                for(k=0; k<1000000; k++)
                                    { asm("nop"); }
                        } else if ((value5_A < 4000) ||
                            (value6_A < 4000) ) {
                                LED0 = LED7 = LED3 =
                                    LED4 = LED1 = LED2 =
                                    0;
                                LED5 = LED6 = 1;
                                e_set_speed_left(500);
                                e_set_speed_right(-500);
                                for(k=0; k<1000000; k++)
                                    { asm("nop"); }
                        } else if ( (value0_A < 4000) ||
                             (value7_A < 4000) ) {
                                LED0 = LED7 = 1;
                                LED1 = LED2 = LED3 =
                                    LED4 = LED5 = LED6 =
                                    0;
                                e_set_speed_left(-800);
                                e_set_speed_right(-800);
                                for(k=0; k<1000000; k++)
                                    { asm("nop"); }
                        } else if ( (value0_A < 6000) ||
                             (value1_A < 6000) || (
                            value2_A < 7000) || (value3_A
                             < 7000) || (value4_A < 7000)
                             || (value7_A < 7000) || (
                            value5_A < 7000) || (value6_A
                             < 7000)) {
                                LED7 = LED6 = LED0 =
                                    LED2 = LED4 = LED1 =
                                    LED3 = LED5 = 1;
                                e_set_speed_left(800);
                                e_set_speed_right(-800);
                                for(k=0; k<1000000; k++)
                                    { asm("nop"); }
                        } else {
                        }
                } else {
                                // for completion
                }

        }
}
```

A.4 followHand

A.4.1 Header File

```
#ifndef _FOLLOWHAND
#define _FOLLOWHAND

void followHand(void);

#endif
```

A.4.2 C File

```
#include "motor_led/e_init_port.h"
#include "motor_led/e_epuck_ports.h"
#include "a_d/advance_ad_scan/e_prox.h"
#include "motor_led/advance_one_timer/e_motors.h"
#include "motor_led/advance_one_timer/e_agenda.h"
#include "a_d/advance_ad_scan/e_ad_conv.h"
#include "uart/e_uart_char.h"
#include "camera/fast_2_timer/e_poxxxx.h"

#include "stdio.h"
#include "string.h"
#include "stdlib.h"

#include "./findRed.h"

#define MIN(X, Y) (((X) < (Y)) ? (X) : (Y))
#define MAX(X, Y) (((X) > (Y)) ? (X) : (Y))

char hbuffer[160];
int hnumbuffer[80];
char debugBuffer[80];
long isHandVisable;


//custom cam picture load
void hgetImage(){
        e_poxxxx_launch_capture((char *)hbuffer);
    while(!e_poxxxx_is_img_ready()){};
}

double RGB2Hue(int r, int g, int b){
        double red = ((double)r)/255, green = ((double)g
           )/255, blue = ((double)b)/255,
                maxv = MAX(MAX(red, green), blue),
                minv = MIN(MIN(red, green), blue),
                h = 0, d = maxv-minv;
        if(d!=0){
                if(maxv == red){
                        h = 60.0*((green - blue)/d + (g
                           < b ? 6 : 0));
                } else if(maxv == green){
                        h = 60.0*((blue - red)/d + 2);
                } else{
                        h = 60.0*((red - green)/d + 4);
                }
        }
        return h;
}
// Image processing removes useless information
void hImage(){
        long i;
        int green, red, blue, vis;
        for(i=0; i<80; i++){
                //RGB turned into an integer value for
                   comparison
                red = (hbuffer[2*i] & 0xF8);
                green = (((hbuffer[2*i] & 0x07) << 5) |
                   ((hbuffer[2*i+1] & 0xE0) >> 3));
                blue = ((hbuffer[2*i+1] & 0x1F) << 3);
                // convert to RGB to Hue
                double hue = RGB2Hue(red, green, blue);
                if(hue> 6 && hue<38){
                        hnumbuffer[i] = 1;
                        vis +=1;
                }else{
                        hnumbuffer[i] = 0;
                }
                if(vis>0){
                        isHandVisable = 1;
                }else{
                        isHandVisable = 0;
                }
        }
}
```

```
//Decide which way to turn based on the number of true
   pixels.
int hturnDirection(){
        int sumL = 0;
        int sumR = 0;
        int i;
        for(i=0;i<40;i++){
                sumL += hnumbuffer[i];
                sumR += hnumbuffer[i+40];
        }
        if(sumL<sumR){
                return 1;
        }else{
                return 0;
        }
}
//Function to deal with turning.
void hturn(void) {
        if(hturnDirection()){
                e_set_speed_left (500);
                e_set_speed_right(-500);
        }else{
                e_set_speed_left (-500);
                e_set_speed_right(500);
        }
}
void hforward(void){
        e_set_speed_left (600);
        e_set_speed_right(600);
}
//Main function of follower
void followHand(void){
        int value0, value1, value2, value3,
                value4, value5, value6, value7,
                   handFound;
        long i;
        handFound = 0;
        //basic set up for the camera and
        e_init_ad_scan(ALL_ADC);
        e_calibrate_ir();
        e_poxxxx_init_cam();
        e_poxxxx_config_cam(0,(ARRAY_HEIGHT - 4)
           /2,640,4,8,4,RGB_565_MODE);
        e_poxxxx_set_mirror(1,1);
        e_poxxxx_write_cam_registers();
        e_start_agendas_processing();
        int centreValue;

        while(1){
                value0 = e_get_calibrated_prox(0);
                value1 = e_get_calibrated_prox(1);
                value2 = e_get_calibrated_prox(2);
                value3 = e_get_calibrated_prox(3);
                value4 = e_get_calibrated_prox(4);
                value5 = e_get_calibrated_prox(5);
                value6 = e_get_calibrated_prox(6);
                value7 = e_get_calibrated_prox(7);
                if(value0>100 || value7>100){
                        if(!handFound){
                                hgetImage();
                                hImage();
                                //Take a section of the
                                   center, this means if
                                    there is an error
                                   with one it won't
                                   effect it as a whole.
                                centreValue = 0;
                                for(i=0; i<80; i++){
                                        centreValue +=
                                           hnumbuffer[i
                                           ];
                                }
                                if(centreValue > 15){
                                        handFound = 1;
                                }
                        }
                        //centreValue = hnumbuffer[38] +
                            hnumbuffer[39] + hnumbuffer
                           [40] + hnumbuffer[41] +
                           hnumbuffer[42] + hnumbuffer
                           [43]; // removes stray
                        if(handFound){
                                e_destroy_agenda(hturn);
                                if(value1>value6+50){
                                        e_set_speed_left
                                            (800);
                                        e_set_speed_right
                                           (400);
                                }
                                else if(value6>value1
                                   +50){
                                        e_set_speed_left
                                            (400);
                                        e_set_speed_right
                                           (800);
                                }
                                else{
                                        hforward();
                                }
                        }
                }
                /*else if(value2>100 || value3>100){
                        e_destroy_agenda(hturn);
                        e_set_speed_left (800);
                        e_set_speed_right(-800);
                }
                else if(value5>100 || value6>100){
                        e_destroy_agenda(hturn);
                        e_set_speed_left (-800);
                        e_set_speed_right(800);
                }*/
                else{
                        handFound = 0;
                        e_destroy_agenda(hturn);
                        e_set_speed_left(0);
                        e_set_speed_right(0);
                }
        }
}
```

References

[1] Robotics Industries Association. http://www.robotics.org/, 2016.

[2] V. A. OLIVEIRA, A. CONCI. Skin detection using hsv color space, 2008.

[3] INC. Robotics. http://www.inc.com/encyclopedia/robotics.html, 2016.

[4] Dr. Q. Meng. Behaviour based robotics lecture slides, 2016.

[5] Pugh J., Cianci C., Klaptocz A., Magnenat S., Zufferey J.-C., Floreano D. Mondada F., Bonani M., Raemy X. and A Martinoli. The e-puck, a robot designed for education in engineering. http://http://www.e-puck.org, 2016.