**16C001 Robotics Coursework**

**Group 2**

Acourt, Abigail Ellen (B323253)

Bhudia, Ushma (B422415)

Driscoll, Joshua (B416688)

Jackson, Alyssia Fay (B420801)

Simpkins, Christopher (B423288)

Toward, Andrew (B419754)

Yip, Simon Yan Lung (B424047)

**Introduction**

For this coursework we were given three tasks. Since we were aware of the amount of marks assigned to each task we split up our group’s effort accordingly. The table below shows how we split up the group:

|  |  |
| --- | --- |
| Group Member | Contributions To Tasks |
| Abby | Task “a” programming, report, testing |
| Ushma | Task “a” programming, report, testing |
| Andy | Task “b” programming, algorithm design and report, testing |
| Alyssia | Task “b” algorithm design, testing and report. Demonstration environment creation |
| Chris | Task “c” programming, report, testing |
| Josh | Task “c” programming, report, testing |
| Simon | Task “c” programming, report, testing |

All team members contributed the same amount.

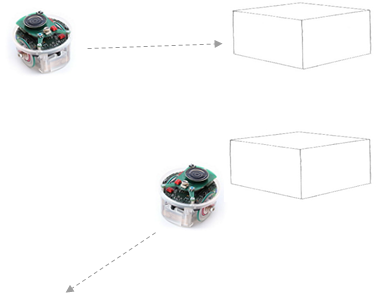
In order to manage our work we held weekly meetings during the labs as a full group and then worked independently in our teams for the rest of the week.

**Part “A” - Braitenberg Behaviours**

# **Designs**

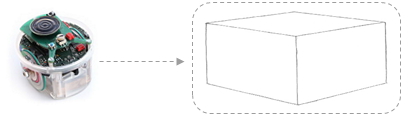
## **Fear**

The way we chose to represent fear was by programming the robot to walk around and if it comes into contact with another object, it will turn and run away from that object. The way this works is the robot will move forward, and when it detects something in its front sensors it will move away from the object it sensed.



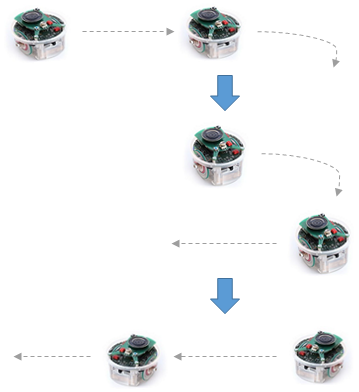
## **Curiosity**

The way we chose to represent curiosity was by programming the robot to walk around and if it comes into contact with another object, to circle it and explore around the entire object. So for example it comes in contact with a box it’ll follow the edges of the box and explore around it. The way it does this is by walking around and when the sensors detect an object it’ll turn so the object is on its side, and maintain a certain distance around all edges of the object it continues to circle.



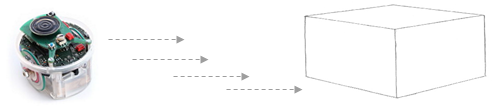
## **Love**

The way we chose to represent love was by programming the robot to walk around and if it comes into contact with another object or robot, it will stick to it and follow it. This best works when used with another robot demonstrating fear, so that robot is moving around avoiding objects, and the robot demonstrating love will follow it, no matter which direction it turns and moves (as shown in the diagram below). This works by it moving until its sensors detect something and then uses this information to maintain the distance between the sensor and the object, no matter where the other object moves.



## **Aggression**

The way we chose to represent aggression was by programming the robot to walk around and if it comes into contact with another object, it’ll begin to aggressively intimidate it. It does this by making aggressive jab motions at the object, it’ll do this 8 times, then it’ll let out an aggressive burst and ‘attack’ the object (do a long dart forward into the object), then it turns and walks away. The way this works is by walking along until the sensors detect an object, as long as the object remains within a certain distance of the sensors, the robot will be more agitated until it finally attacks the object.



# **Testing**

|  |  |  |  |
| --- | --- | --- | --- |
| Area Testing | Expected Outcome | Actual Outcome | Outcome as Expected? |
| Fear – Object to avoid on the left | Robot will veer to the right to avoid the object. | Robot turned right to avoid a small object to its left. | Yes |
| Fear – Object to avoid on the right | Robot will veer to the left to avoid the object. | Robot turned left to avoid a small object to its right. | Yes |
| Fear – Object to avoid directly in front | Robot will move either left or right depending on which sensor the item is closer too. | Robot moved away from any small object placed directly in its past. | Yes |
| Fear – Object distinctly larger than robot to avoid. | Robot will turn away from the object until nothing is picked up from the sensors. | Robot did more of a sharp turn in order to completely turn away from the large object | Yes |
| Fear – Many objects to avoid | Robot should turn away from any objects in its past, no matter which direction. | Robot consistently turned away from all objects in different directions. | Yes |
| Love – Small object to follow | Robot should find the small object and follow it wherever it goes. | Robot finds small object and continues to follow it until it finds a larger object | Yes |
| Love – Large object to follow | Robot should find the large object and follow it wherever it goes. | Robot finds the large object and follows it no matter which way | Yes |
| Love – Object moving quickly around the robot to follow | Robot should be able to keep track of the object and follow it in a 360 degree motion. | Robot keeps track of where the object is and is able to spin to continue follow it | Yes |
| Love – Other robot to follow | Robot should stay following the other robot whichever direction it goes. | Robot finds other robot and follows it, even if it changes direction rapidly. | Yes |
| Curiosity – Large object to explore (circle) | Robot should find the large object to curiously follow the entire object round, exploring all sides. | Robot finds large object and circles around it, following its circular edge. | Yes |
| Curiosity – Small object to explore (circle) | Robot should find the small object to curiously follow the entire object round, exploring all sides. | Robot finds small object and manages to circle around it, following its circular edge. | Yes |
| Curiosity – Rectangular object to explore | Robot should find the rectangle object to curiously follow the entire object round, exploring all sides. | Robot finds the rectangular object and manages to follow round all its edges. | Yes |
| Curiosity – Triangular object to explore | Robot should find the triangular object to curiously follow the entire object round, exploring all sides. | Robot finds triangular object and providing it is large enough it manages to follow around the entire object. | Yes – however if the object is too small the sharp change of the directions of the corners can confuse the robot |
| Aggression – Robot finds an object to act aggressively to. | Robot finds an object and begins its intimidating jabs with progressive lighting up LED’s , then makes its final attack before turning away. | Robot finds object and then begins its intimidation, until all LED’s are lit, which is when it attacks. It then moves away from the object. | Yes |
| Aggression- Object moves away during the intimidation part. | Robot begins intimidating object but the object moves away. | The robot stops it’s intimidating and carries on moving forward until it finds another object to act aggressively to. | Yes |

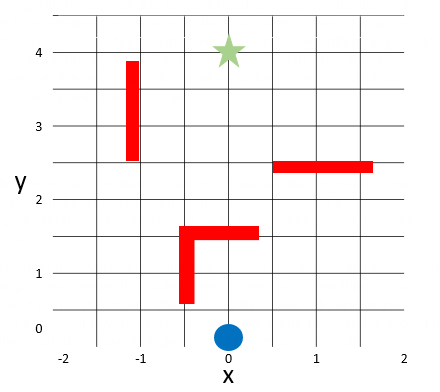
**Part “B” - Goal Seeking and Obstacle Avoidance**

**Starting the task**

We decided that there were two key areas of work within part B. Firstly we had to develop the algorithm that we wanted to use to perform the goal seeking and obstacle avoidance. Secondly there was the was the actual hands on programming part of the work consisting of controlling the robot and running our algorithm.

**Algorithm Design**

We decided to use a bug algorithm for this task. This was because they are relatively simple and so should be quicker to program. At first we considered bug 0. This algorithm is very easy to understand. However as we started to perform tests of this algorithm we realised that it would have certain issues due to the precision needed in remembering the angle to the goal. The next algorithm that we looked at was bug 1. We have sample code to circumnavigate obstacles and so we thought that it would definitely be achievable to use this algorithm. However, we could see that a difficulty that we were going to come up against when implementing this algorithm was that due to the limited sensing capabilities of the E-Puck we would struggle to remember the distance to the goal when circumnavigating obstacles. We therefore decided to use bug 2. After writing all three algorithms in pseudocode we realised that this algorithm would be the easiest to work with since the most important thing is to remember the robot's position relative to the M line, and that was something we could store easily within our program.

We therefore decided to implement a coordinate system into our program. A bird's eye view of this system is shown in the below diagram. The distances and orientations are relative to the robot’s starting orientation.

As can be seen in the diagram, the robot starts at coordinate (0,0) and is aware of the distance to the goal and also the direction it is in. It executes a bug 2 algorithm, following the M line until it encounters an obstacle. It will then circumnavigate the obstacle and return to the M line as soon as possible before then continuing towards the goal. The robot remembers it’s Y coordinate, the distance from the start point, and also the goal’s Y coordinate. It is also aware of its horizontal displacement from the M line, which on this diagram runs from (0,0) to (0,4).

**Development and testing**

As soon as we started coding we realised that working with the E-Pucks made development much more difficult than we had previously experienced with computer-only systems. This is because the hardware element was prone to behaving unpredictably, for example in different lighting conditions or with varying battery levels. It was also much more difficult to debug our work since we could not step through code and trace what was going on with our programs since the E-Puck needed to run at real time speed. In order to deal with these difficulties we decided to develop this part of the coursework with a unit based approach. To do this we broke our algorithm down into extremely simple parts and then tested them individually. For example we made sure we could make the robot perform simple, accurate movements and sensing before then combining it all together into our algorithm.

In order to test our full program we created practice courses for the robot to run through. To do this we used toy building blocks as they meant we could easily alter the environment in order to test as many different scenarios as possible.

Furthermore, we also noticed that after our initial program was working that the robot could have taken a much quicker route to the goal state than the one it took. This is due to the fact the if the robot meets an obstacle it always turns to the right to go around that way. However, if the robot was to turn left on some of these objects it would have made the journey a lot shorter. The main aim then was to try and make the robot turn in a direction so that it wasn't going too far away from the M line.

|  |  |  |  |
| --- | --- | --- | --- |
| Area Testing | Expected Outcome | Actual Outcome | Outcome as Expected? |
| Distance to objective, no obstacles | Move forward until at end goal position | Robot reached end position and stopped | Yes |
| Obstacle in front | Stop before obstacle. Turn right, move around obstacle and return to M line once past obstacle | Moved around obstacle and back to M line | Yes |
| Obstacle in front and to right | Stop before obstacle. Turn left, move around obstacle and return to M line once past obstacle | Stop before obstacle. Turn left, move around obstacle and return to M line once past obstacle | Yes |
| Longer obstacle in front | Stop before obstacle. Turn right, move around obstacle and return to M line once past obstacle | Moved around obstacle and back to M line | Yes |
| Longer obstacle in front and to right | Stop before obstacle. Turn left, move around obstacle and return to M line once past obstacle | Stop before obstacle. Turn left, move around obstacle and return to M line once past obstacle | Yes |
| While avoiding to right encounter a second obstacle | Avoid both obstacles, return to M line | Avoided second obstacle and returned to M line | Yes |
| While avoiding to left encounter a second obstacle | Avoid both obstacles, return to M line | Avoided second obstacle and returned to M line | Yes |
| While avoiding second obstacle to right encounter a third obstacle | Avoid all obstacles, return to M line | Avoided all obstacles and returned to M line | Yes |
| While avoiding second obstacle to left encounter a third obstacle | Avoid all obstacles, return to M line | Avoided all obstacles and returned to M line | Yes |

**Part “C” - Our Own High Level Behaviour**

For this task, we have assigned three members of our group to complete this task, Simon Yip, Joshua Driscoll and Chris Simpkins. To demonstrate high level behaviour of the robots, we came up with the following scenario. The scenario is a search and rescue operation involving two robots. One will be the rescuer and one will be the survivor.

The rescuer will be searching for survivors within an obstacle filled environment and try to locate the survivor in the environment. Once it finds the survivor, it will return to the entrance together with the survivor.

The other robot would be the one waiting to be rescued by the robot. It will spin continuously until it is found by the rescuer. Once it is find by the rescuer, it follows the other robot to safety.

The environment will have a blue gate to start off with, the rescuer will be labelled as green and the survivor will be labelled as red. The purpose of the gate is to represent the end of the danger zone.

The functions of the rescuer would be as follows:

1. First it will go through the blue gate.

2. It will avoid any obstacle that stands in the way

3. While moving, it will continuously be scanning for any survivor (the colour)

4. Once it finds the survivor, it will than return to the blue gate.

The functions of the survivor would be as follows:

1. It will continue to spin if the rescuer hasn’t appeared.

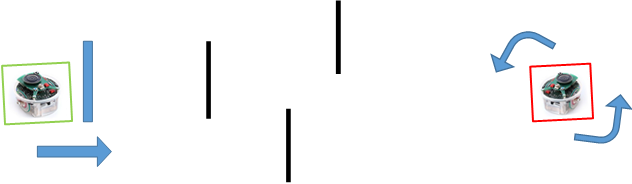
2. If the rescuer labelled the colour green appears, it will stop spinning.

3. The robot will follow the robot to the blue gate.

The key area of this task would be the operating the camera for colour detection and combining the code for obstacle avoidance with this task.

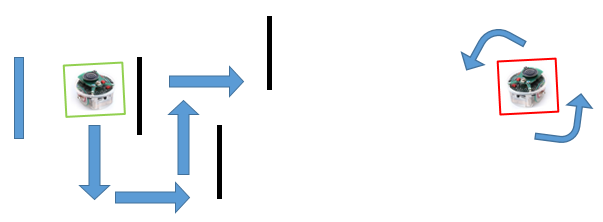
***Design***

*State 1*



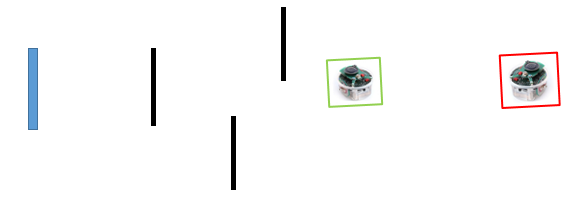
From the diagram above, we can see the rescue robot is moving past the blue gate and the survivor robot is spinning left trying to find the rescue robot.

State 2



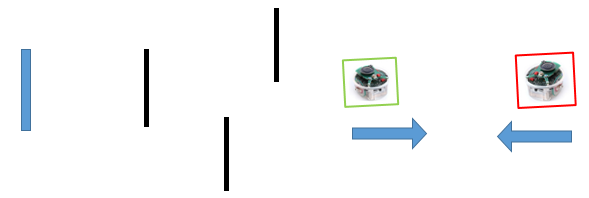
From the diagram above, we can see the rescue robot avoid obstacles. The survivor robot is still spinning left trying to find the rescue robot.

State 3



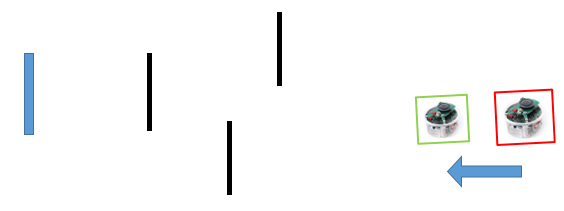
From the diagram above, we can see the rescue robot has find the survivor robot and likewise with the survivor robot. Both robots will wait for a few seconds trying to confirm the other robot. The robots show this by flashing its leds one by one.

State 4



Once the robots have found each other, they will move toward each other.

State 5



Once they are close enough with each other, the rescue robot will make an u-turn and move back towards the gate while the survivor robot will follow the rescue robot.

State 6



After going through all the obstacles and the robots have moved past the blue gate, the rescue robot will stop and the survivor robot will try to confirm the rescue robot has stopped by flashing its leds. Once it has confirmed the rescue robot has stopped, it will flash all its leds and play a sound.

**The code structure is as follows:**

For the rescuer robot, it will use the following functions:

- void push(void);

- int goalTest(long correct90, long correctGoalDistance, long correctSensor);

- void turn180(long spin);

- int approachRed(void);

- int approachBlue(void);

For the rescuer robot, it will use the following functions:

- int e\_start\_follow(void);

**See appendix A for more details.**

***Development and testing***

|  |  |  |  |
| --- | --- | --- | --- |
| **Testing scenario** | **Expected Outcome** | **Actual Outcome** | **Outcome as Expected?** |
| Founding the colour red using the robot’s camera | The leds will turn on one by one until all the leds are turned on. | As expected | Yes |
| Not founding the colour red using the robot’s camera | The robot will spin to the left continue to search for the colour red. | As expected | Yes |
| After it has find the colour red but lost sight of it. | The robot will spin to the left continue to search for the colour red and the led will have reset when the next red object is spotted | As expected | Yes |
| After finding the colour red, approach the object. | After all the leds have light up, the robot will move forward. | As expected | Yes |
| When the robot is very close to the red object, stop the robot. | When the robot is very close to the red object, it will light up its leds one by one until they are all turned on. Then all the leds will flashes afterward. | As expected | Yes |
| Founding the colour blue using the robot’s camera | The leds will turn on one by one until all the leds are turned on. | As expected | Yes |
| Not founding the colour blue using the robot’s camera | The robot will spin to the left continue to search for the colour blue. | As expected | Yes |
| After it has find the colour blue but lost sight of it. | The robot will spin to the left continue to search for the colour blue and the led will have reset when the next blue object is spotted | As expected | Yes |
| After finding the colour blue, approach the object. | After all the leds have light up, the robot will move forward. | As expected | Yes |
| When the robot is very close to the blue object, stop the robot. | When the robot is very close to the red object, it will light up it’s leds one by one until they are all turned on. Then all the leds will flashes afterward. | As expected | Yes |
| Founding the colour green using the robot’s camera | The leds will turn on one by one until all the leds are turned on. | As expected | Yes |
| Not finding the colour green using the robot’s camera | The robot will spin to the left continue to search for the colour green and will play a sound repeatedly | As expected | Yes |
| After it has find the colour green but lost sight of it. | The robot will spin to the left continue to search for the colour green and the led will have reset when the next green object is spotted | As expected | Yes |
| After finding the colour green, follow the green object | After all the leds are turned on which means the robot has confirmed the existence of the green object, all the leds will turn off except for led 0 and the robot will move toward the object. | As expected | Yes |
| While follow the green object, if lost sight of object, spin left | If the camera lost the green object, the robot will spin to the left and led number 1 and 7 will be turned on. | As expected | Yes |
| When the green object is not completely centred with the camera | If the green is not aligned to the centre of the camera, the robot will turn to the direction of green object so it is centred in the middle of the camera. | As expected | Yes |
| When the robot is very close to the green object, stop the robot. | If the robot is very close to the object, it will stop and start turning the leds on one by one. | As expected | Yes |
|  |  | As expected | Yes |
| The come close to the green object and the object shows no sign of moving. | From the previous test, if the robot turned all the leds on, then the all the leds will flashes and the robot will play a sound repeatedly. | As expected | Yes |
| Testing the turn180 function | Inputting values between 820000 and 1400000 will make the robot turn from 90 to 180 degree. | As expected | Yes |
| Testing the push function | Move forward for a few second then stop | As expected | Yes |
|  |  |  |  |

*Part C Test description*

**Result Analysis and Conclusion**

Overall, we have been able to create code that has enabled us to complete all three sections of the coursework. However, we did have some issues while working on these sections, and these issues are common to all three sections. One issue we encountered was that the E-Pucks can behave very unpredictably, especially when they start to run out of battery. A good example of this is that we found the proximity sensors need to be regularly calibrated in order to maintain consistency of sensing distances. Another example of how the E-Pucks provide difficulties is that there is no user interface. This provides difficulties when it comes to troubleshooting what part of the code needs to change in order to make the program work in the way it’s expected. Yet as a group we were able to overcome these issues by recalibration on a regular basis, and also using part of our code ‘spin right’ to see where the program has stopped, or what parts of the loops or statements it enters or not.

Yet, despite these issues, we have been able to work together as a group to produce a working project that includes code in order to complete parts A, B, and C.

**Appendix A**

**Part C function explanations**

**Function explanations**

***void push(void)***

This function will first destroy the ‘uTurn’ agenda if it is previously called and make the robot to move forward for a short period then stop.

***Int goalTest(long correct90, long correctGoalDistance, long correctSensor)***

This function is used to make the robot to reach a certain distance specify by the *long correctGoalDistance* while avoiding obstacles along the way. (See Part B report for detail information).

***Void turn180(long spin)***

This function will first destroy the ‘uTurn’ agenda if it is previously called then activate ‘uTurn’ agenda for a limited duration of time specify by the value ‘spin’ which make the robot to move forward for a short period then stop. The ‘uTurn’ function set the left and right motor speed to 500.

***Int approachRed(void)***

*This function search for the colour red with the camera of the robot and approach the source of the colour. The details of this function are as follows:*

*First, we initialise the basic setup for the camera. These functions are*

- *e\_poxxxx\_init\_cam();*

- *e\_poxxxx\_config\_cam(0, (ARRAY\_HEIGHT - 4) / 2, 640, 4, 8, 4, RGB\_565\_MODE);*

- *e\_poxxxx\_write\_cam\_registers();*

*These functions come from the library “camera/fast\_2\_timer/e\_poxxxx.h". We wouldn’t be going through the details of these function as it is much to do with the e-puck hardware.*

*Next we initialise the start of agenda processing by e\_start\_agendas\_processing() which come from the library "motor\_led/advance\_one\_timer/e\_agenda.h".*

*Now, we introduce two variables: int centreValue and int counter = 0 which we will use later and a while loop with condition ‘1’ which means it will run continuously until it reaches a break function. Within the while loop, we first have use the function rgetImage(); to load the camera image, rImage(); to process the image and e\_led\_clear(); to clear any led that was set before on the robot.*

*rImage() will read each bit of the image and separate them into red or green values. First, the RGB is turned into an integer value for comparison. If the image contains red colours, then green will be less then red and we set rnumbuffer[i] = 1 and increase the value of ‘vis’ which is a counter to see if red is visible in whole image. Otherwise we set rnumbuffer[i] = 0. rnumbuffer is an array which represent the redness of the image captured by the camera and it has the size of 80. We repeat this process 80 times until we finished with image. If the value of ‘vis’ is greater than 0, we set the variable risRedVisable = 1 which means red is visible in the image.*

*Next, we take the centre section of the image which is represented as rnumbuffer array numbered from 38 to 43, take the sum of these values and store them into the variable centreValue. As the array will only contains numbers either 0 or 1, the most possible value we can obtain would be 6 for the centreValue (43-38 = 6).*

*Now, we move onto the if statements. First, if the centreValue > 3, we will destroy the agenda ‘rturn’ if it is previously called, set the speed of the motors to 0 and increase the counter variable by 1. After the increase the counter variable, we switch on the led corresponding to the counter value and make the program wait for a second before continuing with the program.*

*If centreValue is less than 3 we move to the else if statement. Else if risRedVisable = 1, activate agenda ‘rturn’ and set set led 1 to be on and reset the counter to 0. ‘rturn’ uses the function ‘rturnDirection’ in its if statements. The function ‘rturnDirection’ will read the rnumbuffer array to see whether the colour is on the left or right of the image. If the colour is on the right side of the image, it will return 1 otherwise it will return 0. Back to the ‘rturn’ function, so if rturnDirection == 1, this means the colour is on the right side, so we make the robot spin to the right, otherwise spin to the left.*

*This means that if red isn’t in the centre but it is visible by the camera, we make the robot turn so that the robot is pointing at the centre of the red image/object.*

*If the previous is not true either, which means red isn’t visible so we make the robot turn spin to search for a red colour image/object.*

*Now, we check the if the counter reaches 7, if counter == 7 we break out of the while loop, if not we will do another iteration of the loop.*

*Once, we break out of the while loop we start new function called e\_start\_anotherfunction();.*

*This function will make the robot approach the red object and stop when it is close to it. First, we initialise the IR sensors using e\_init\_prox() which come from the library "a\_d/e\_prox.h".*

*Now, we create a while loop. Within the while loop, we have the if statement, if IR sensor number 1 and IR sensor number 7 has a value greater than 2000, this means the robot is close to the object. The function we then destroy the agenda ‘rforward’ if previously called and set the motor speed to 0 and break out of the while loop since the robot has reached the red object. If the if statement is not true, then we switch on led number 1 and 7 and activate the agenda ‘rforward’ which will make the robot move forward.*

*Once we break out of the while loop, we make all the led flashes and end the function.*

***Int approachBlue(void)***

*Int approachBlue(void) is the same as approachRed() but instead of red, it will look for blue when we process the image.*

***int e\_start\_follow(void);***

*All of the functionality needed for the survivor robot are contained within this function. To start the survivor robot program, only this function need to be called by the sector. This function will call other functions within this function.*

*The purpose of this function (for the survivor robot) would be to search for the colour green.*

*Once it finds the rescuer, it will follow the rescuer until the rescuer stops.*

*First, we initialise the basic setup for the camera. These functions are*

- *e\_poxxxx\_init\_cam();*

- *e\_poxxxx\_config\_cam(0, (ARRAY\_HEIGHT - 4) / 2, 640, 4, 8, 4, RGB\_565\_MODE);*

- *e\_poxxxx\_write\_cam\_registers();*

*These functions come from the library “camera/fast\_2\_timer/e\_poxxxx.h". We wouldn’t be going through the details of these function as it is much to do with the e-puck hardware.*

*Next, we introduce three variables: int centreValue, int counter = 0 and int robotfind = 0 which we will use later. We also initialise the IR sensors and Audio output using:*

- *e\_init\_prox();*

- *e\_init\_sound();*

*e\_init\_prox() which come from the library "a\_d/e\_prox.h". e\_init\_sound come from the library "codec/e\_sound.h". These functions are necessary to use the functions related to the IR sensors and Audio output in the library.*

*Now, we have our while loop. First in the while loop we have ggetImage(), gImage() and led\_clear(). ggetImage(); to load the camera image, gImage(); to process the image and e\_led\_clear(); to clear any led that was set before on the robot. gImage() and rImage are the same but instead of searching for red, gImage() searches for green in the image.*

*Next, we have two if statements which changes the indicators (the variables we declared earlier).*

*If counter == 7 and robotfind == 0 we set robotfind = 1 and counter = 0.*

*The purpose of this statement is to determine if the robot has encounter a green object (in other word, the rescue robot) and we change the robotfind indicator to 1 to let the program know that the robot has find a green object. The counter is to make sure that the robot has spotted a green object. We set the counter value to 0 so it can be reuse again later, you will find out more about how we change the counter value later.*

*For the next if statement, if counter == 7 and robotfind == 1, we break out of the while loop. This statement is for the second part of the survivor robot, where it has find, followed and stopped behind the rescue robot. This means that the survivor robot has escaped outside of the danger zone and reached the goal with the rescue robot. Therefore, we break out of the current while loop and move on to the next function outside of the while loop.*

*With all the indicator sorted, we move on the analysing the image of the camera. First, we get the centreValue of the image. The explanation of how to get the centreValue came about can be seen above in the int approachRed(void) function. Now, we have a series of if statement inside an if statement.*

*First, if centreValue > 2, which means if there exist the colour green in the centre of the image, we first destroy gturn2 and stop1 which are the possible motors agenda could be called in this while loop (More about the agenda gturn2 and stop1 later).*

*Next, we have another if statement, if robotfind == 0 within the current if statement centreValue > 2. This means that if the robot has identified the colour green in the centre of the camera but haven’t confirmed that it is the rescue robot, we will run the following process:*

*It will destroy agenda flow\_led, if it is previously called, activate agenda stop1, increase counter by 1, switch the led on corresponding to the counter value and wait for a second before repeating the while loop again, long i; for (i = 0; i < 500000; i++) {asm("nop");} .*

*The process above has the following purpose: The robot has find a green object with the camera at that moment in time, it stops its motor which is what agenda stop1 does, increase it counter by 1 telling the program that it has find a green object in that direction and light up the led to show the users that the counter has increased by one.*

*If the green object continues to remain in that direction, the counter should continue to increase and the while loop repeats and eventually reaches the value 7. Once it reaches 7, the program can safely confirm that it is the rescue robot and we run the if statement that I have explained above.*

*If robotfind == 0 is not true, this means that the robot has indicated that it finds that rescue robot, it will switch on led 0 and run another if statement. If e\_get\_prox(0) > 200 or e\_get\_prox(7) > 200, we activate agenda stop1, increase counter by 1, switch the led on corresponding to the counter value and wait for a second before repeating the while loop again, long i; for (i = 0; i < 500000; i++) {asm("nop");} .*

*This means that if the colour green is identified in the centre of the image and the robotfind == 0 is not true and the IR sensor number 0 and number 7 detected an object and give a value greater than 200, this means that the rescue robot has stopped and the robot is right behind the rescue robot. Once, the counter reaches the value of 7 again, the program will confirm that the rescue robot has stopped which means it has reached the goal. It will run the if statement that I previously explained before at the beginning of the while loop and break out of the loop.*

*if the colour green is identified in the centre of the image and the robotfind == 0 is not true but the IR sensor didn’t detect an object in front of it, we will destroy agenda stop1, in case it is previously called, set the speed of the left and right motor to 500 which will make the robot move forward and reset the counter to 0 since the rescue robot is still moving.*

*Now, back to the beginning if statement. If centreValue > 2 is not true, we will first destroy agenda stop1 in case it if previously called and activate agenda gturn2. The function gturn2 will check the image to see if the colour green is on the left side or the right side since it doesn’t exist in the centre of the image. It will turn to the direction of the green colour so it the camera will point to the colour green in its centre. If green doesn’t exist in the image, it will continue to spin left until it finds the colour green.*

*Now, we encounter another if statement within the else statement of centreValue > 3. If robotfind == 0, activate agenda flow\_led, play sound and wait for a second before repeating the loop. This means that if the colour green is not in the centre of the image and the robot hasn’t find the rescue robot, the robot will spin, light up the leds and make the robot to play a sound which is the distress signal.*

*If robotfind == 0 is not true, this means the colour green is not in the centre of the image but the robot has find the rescue robot from before, the robot will spin and light up two of it leds and reset the counter to 0.*

*When the function breaks out of the while loop, it will run the function finish();*

*The finish function will have the following. First we initialise a while loop. Within the while loop, we activate agenda flow\_led and play a sound through the robot speaker and wait for a few second before repeating the loop again.*

**Appendix B**

**Flow Charts**

See following pages