**Informe Final del proyecto: "Mobile-Robot-Trash-Can"**

Presentado por los estudiantes:

- Jhon Kevin Muñoz Peña.
- Juan Carlos Noguera Ramirez.
- Eduardo Osorio Garcia.

**Código del esquema de control basado en comportamientos. (Myversion2.java)**

Se tiene un esquema basado en comportamientos que maneja 3 comportamientos, los cuales se muestran a continuación, del de menor prioridad al de mayor prioridad:

- BehaviorTowardsPeople.
- BehaviorReachGoal.
- BehaviorEscape.

En el main del código se establece una velocidad crucero del robot de 100 grados por segundo, para minimizar el ruido de la resta de imágenes, y una variable BT, con la cual se crea el camino de comunicación con el bluetooth.

```
181  ////////////////////////////////////////////////////////////////
182  public class Myversion2
183  {
184
185  static final float WHEEL_DIAM=5.6f;
186  static final float TRACK_W=11.5f;
187  static BTConnection btc = Bluetooth.waitForConnection();
188
189  public static void main (String [] args)
190  {
191  UltrasonicSensor us1 = new UltrasonicSensor(SensorPort.S4);
192  Pilot robot = new Pilot(WHEEL_DIAM, TRACK_W, Motor.A, Motor.C);
193  robot.setSpeed(100);
194  Behavior b2= new BehaviorTowardsPeople(us1, robot, btc);
195  Behavior b3= new BehaviorReachGoal(us1, robot, btc);
196  Behavior b5 = new BehaviorEscape();
197  Behavior [] bArray={b2,b3,b5};
198  Arbitrator arby = new Arbitrator(bArray, false);
199  arby.start();
200  }
201  }
202
```

A continuación se anexan una breve descripción de lo que hace cada comportamiento, omitiendo el comportamiento escape, puesto que este solo es el código que permite salirnos del programa si se hunde el botón de exit.

**BehaviorTowardsPeople.**

Este comportamiento hace uso de una variable BT con la cual accede al canal de comunicación, el sensor de ultrasonido y los motores del robot. En este comportamiento se crean unas variables auxiliares para resolver unos problemas que se tuvieron al manejar los datos recibidos por bluetooth, x1 y y1 que corresponde al

vector objetivo, x2 y y2 que corresponden al vector obstáculo y grados que indica el ángulo de rotación del vector resultante final.

```
13  //////////////////////////////////////////////////////////////////////////
14  class BehaviorTowardsPeople implements Behavior
15  {
16  BTConnection btc;
17  UltrasonicSensor us1;
18  Pilot robot;
19  boolean girar=false;
20  double x1;
21  double y1;
22  double x2;
23  double y2;
24  double x;
25  double y;
26  int Aux1;
27  int Aux2;
28  int grados=0;
29
30
31  public BehaviorTowardsPeople(UltrasonicSensor us1, Pilot p, BTConnection btc)
32  {
33  this.us1=us1;
34  this.robot=p;
35  this.btc=btc;
36  }
37
```

El método takecontrol de este comportamiento retorna la variable booleana girar, la cual siempre estará en true, y el valor en grados que girara el robot en el método action. Primero se obtiene el valor del centroide que envía el celular, y usando el valor de la mitad del ancho de la pantalla del celular se obtiene un valor positivo o negativo de coordenada x para el vector objetivo y se da un valor establecido a la coordenada y.

```
37
38  public boolean takeControl()
39  {
40  btc.setIOMode(NXTConnection.RAW);
41  DataInputStream dis = btc.openDataInputStream();
42
43                  try {
44                      int n = dis.readInt();
45                      int n1 = dis.readInt();
46                      int n2 = dis.readInt();
47                      Aux1=n;
48                      Aux2=240-Aux1;
49                      x1=(double)(Aux2);
50                      y1=20;
51
```

Después se revisa el estado del sensor de ultrasonido para saber si hay un obstáculo, si no hay obstáculo, se establece un vector de cero grados con respecto al eje y, y si hay obstáculo se establece un vector de alrededor de 89 grados con respecto al eje y.

```
51
52                  if(((us1.getDistance()>20))){
53                      girar=true;
54                      x2=0;
55                      y2=1;
56                      }
57
58                  else{
59                      girar=true;
60                      x2=1000;
61                      y2=10;
62                      }
```

Por último se hace una suma vectorial, se calcula el ángulo con respecto al eje y del vector resultante y se retorna el valor de girar.

```
                    x=x1+x2;
                    y=y1+y2;
                    grados=(int)((Math.atan(x/y))*(180/Math.PI));
                    }
                catch (IOException e) {
                    System.exit(0);
                    }
return girar;
}
```

En el método de action lo que se hace es detener el robot, girar el ángulo que se calculó y seguir adelante.

```
public void action ()
{
robot.stop();
robot.rotate(grados);
robot.forward();
}
```

**BehaviorReachGoal.**

En este comportamiento se hace uso de la variable BT, para tener acceso al canal de comunicación con el bluetooth, del sensor de ultrasonido y de los motores del robot. Además se crea una variable booleana girar, y dos enteros parar y grados, parar corresponde al valor de la variable num_pixeles que envía el celular, la cual indica la cantidad de puntos negros en la pantalla, y grados que es el grado a girar de activarse este comportamiento.

```
class BehaviorReachGoal implements Behavior
{
UltrasonicSensor us1;
Pilot robot;
BTConnection btc;
int parar;
int grados=0;
boolean girar=false;

public BehaviorReachGoal(UltrasonicSensor us1, Pilot p, BTConnection btc)
{
this.us1=us1;
this.robot=p;
this.btc=btc;
}
```

En el método takecontrol se revisa si el valor de parar supera un umbral definido, el cual indica que hay un movimiento muy cerca de la pantalla, y para asegurarnos de que ese movimiento es debido a una persona y

no a ruido en la resta debido al delay de la cámara, se revisa también la información que obtiene el ultrasonido, si se cumplen la condición de parar y la del rango del ultrasonido se establece un ángulo de giro definido de 180 y se da un valor de true a la variable booleana.

```
102
103  public boolean takeControl()
104  {
105  btc.setIOMode(NXTConnection.RAW);
106  DataInputStream dis = btc.openDataInputStream();
107
108                  try {
109                      int n = dis.readInt();
110                      int n1 = dis.readInt();
111                      int n2 = dis.readInt();
112                      parar=n2;
113
114                      if(parar>160000 && us1.getDistance()<35){
115                          girar=true;
116                          grados=180;
117                          }
118
119                      else{
120                          girar=false;
121                          }
122                      }
123                  catch (IOException e) {
124                      System.exit(0);
125                      }
126  return girar;
127  }
```

Si la variable girar es true, se entra al método action, donde se detiene el robot por 5 segundos, de tal manera que la persona tenga tiempo para colocar la basura, también se había agregado el código del ejemplo tune.java para que el robot sonara, pero como el robot no suena se eliminó esa parte de código. Una vez han pasado los 5 segundos, el robot gira los 180 grados y sigue con el algoritmo.

```
128
129  public void action ()
130  {
131  robot.stop();
132  try {
133      Thread.sleep(5000);                 //1000 milliseconds is one second.
134      }
135  catch(InterruptedException ex) {
136      Thread.currentThread().interrupt();
137      }
138  robot.rotate(grados);
139  robot.forward();
140  }
141
142  public void suppress()
143  {
144  robot.stop();
145  }
146  }
```

**Código de adquisición y procesamiento de imágenes. (MainActivity.java)**

El algoritmo de procesamiento y adquisición está divido en la toma de fotografías, en la aplicación de las resta básica de imágenes, en el cálculo del centroide y en la comunicación bluetooth entre el teléfono móvil y el Lego NXT. Lo primero es habilitar los permisos de cámara y comunicación bluetooth.

Se declaran las variables necesarias, en este caso se tienen DONE, NEXT y PERIOD para operar el hilo que permite tomar fotografías continuamente. La variable camera para acceder a la cámara, cameraID que es la dirección para acceder a la cámara trasera del teléfono. Además se tiene las variables para poder enviar datos mediante el bluetooth, la dirección MAC del Lego y demás variables necesarias para la comunicación.

```
33 public class MainActivity extends Activity {
34       private Camera camera;
35       public static final int DONE=1;
36       public static final int NEXT=2;
37       public static final int PERIOD=1;
38       private Timer timer;
39       private int cameraId = 0;
40     // Punto de salida del canal de comunicación
41     // Se usa para enviar hacia el robot
42     private DataOutputStream DataOut = null;
43
44      //Target NXTs for communication
45              String nxt1 = "00:16:53:05:C6:8E";
46
47              BluetoothAdapter localAdapter;
48              BluetoothSocket socket_nxt1;
49              boolean success=false;
50        //Layout donde se verá la imagen de la camara
51      Bitmap Imag, Imag1, ImagM; ImageView view, view1; Button But; TextView text,text1,text2;
52      int picw, pich; int pix[],cnt=0, ci, cj ;
53      byte[] tempdata;
54
```

Las variables tipo Bitmap permiten guardar en memoria imágenes en un mapa de bits. Las variables del tipo ImageView, Button y TextView son las usadas para acceder a los elementos del entorno gráfico. Las variables picw, pich almacenan el ancho y alto de la imagen. En la siguiente imagen se tiene todo lo necesario para determinar si existe cámara en el dispositivo y si se accedió a la cámara trasera. Además se habilita el bluetooth del dispositivo.

```
56⊖    @Override
57     protected void onCreate(Bundle savedInstanceState) {
58         super.onCreate(savedInstanceState);
59         setContentView(R.layout.activity_main);
60          // do we have a camera?
61         if (!getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)) {
62          Toast.makeText(this, "No camera on this device", Toast.LENGTH_LONG)
63            .show();
64         } else {
65          cameraId = findFrontFacingCamera();
66          if (cameraId < 0) {
67           Toast.makeText(this, "No back facing camera found.",
68             Toast.LENGTH_LONG).show();
69          } else {
70              safeCameraOpen(cameraId);
71          }
72         }
73         SurfaceView fake = new SurfaceView(this);
74         try {
75                 camera.setPreviewDisplay(fake.getHolder());
76            } catch (IOException e) {
77                 // TODO Auto-generated catch block
78                 e.printStackTrace();
79            }
80
```

```
        camera.startPreview();
        Camera.Parameters params = camera.getParameters();
        params.setJpegQuality(100);
        camera.setParameters(params);
        view = (ImageView)findViewById(R.id.Image);
        view1 = (ImageView)findViewById(R.id.Image1);
        But = (Button)findViewById(R.id.Butt);
        text = (TextView)findViewById(R.id.text);
        text1 = (TextView)findViewById(R.id.text1);
        text2 = (TextView)findViewById(R.id.text2);
        But.setOnClickListener(buttonListener);
        enableBT();
    }
```

El hilo permite la captura consecutiva de imágenes, cada vez que pasa por el case DONE se toma una fotografía que se accede mediante myJpeg, y se guarda en memoria dicha imagen mediante la variable Imag1 (original). Cuando se accede a NEXT es cuando termina la toma de la foto, y nuevamente se ejecuta el hilo guardando la imagen inmediatamente anterior en la variable Imag (background).

```
    private Handler threadHandler = new Handler() {
        public void handleMessage(android.os.Message msg) {
              switch(msg.what){
              case DONE:
                  // Trigger camera callback to take pic
                   camera.takePicture(null, null, myJpeg);
                   Imag1=ImagM;
                   break;
              case NEXT:

                   timer=new Timer(getApplicationContext(),threadHandler);
                   timer.execute();
                   Imag=ImagM;
                   pix = new int[picw*pich];
                   cnt++;
                   if(cnt>3){
                       Operation();
                   }
                   break;
              }
              }
         };

```

La función Operation permite el cálculo de la resta de imágenes y el centroide, solo se accede a ella debido a que en las primeras imágenes no se tiene fotografía alguna o su valor es null. En onClick se abre el canal de comunicación y a su vez se ejecuta el hilo ya mencionado por primera vez, para allí en adelante quedarse ejecutando hasta cerrar el programa.

```
    private OnClickListener buttonListener = new OnClickListener() {

        public void onClick(View v) {

            connectToNXTs();
            timer=new Timer(getApplicationContext(),threadHandler);
            timer.execute();

      //      boolean s=connectToNXTs();
       //     text1.setText(String.valueOf(s));

        }
```

Si el dispositivo móvil tiene deshabilitado el bluetooth la función enableBT se encarga de habilitarlo.

```
136     //Enables Bluetooth if not enabled
137             public void enableBT(){
138                 localAdapter=BluetoothAdapter.getDefaultAdapter();
139                 //If Bluetooth not enable then do it
140                 if(localAdapter.isEnabled()==false){
141                     localAdapter.enable();
142                     while(!(localAdapter.isEnabled())){
143
144                     }
145                 }
146
147             }
```

La función connectToNXTs permite establecer la comunicación y abrir el canal para ser utilizado para escribir, solo está habilitado para enviar datos. Si existe un error try-catch genera un mensaje de error.

```
148     //connect to both NXTs
149             public  boolean connectToNXTs(){
150
151
152
153                 //get the BluetoothDevice of the NXT
154                 BluetoothDevice nxt_1 = localAdapter.getRemoteDevice(nxt1);
155                 //try to connect to the nxt
156                 try {
157                     socket_nxt1 = nxt_1.createRfcommSocketToServiceRecord
158                             (UUID.fromString("00001101-0000-1000-8000-00805F9B34FB"));
159                         socket_nxt1.connect();
160                     success = true;
161                     DataOut = new DataOutputStream(socket_nxt1.getOutputStream());
162                 //     DataIN = new DataInputStream(socket_nxt1.getInputStream());
163                 } catch (IOException e) {
164                     Log.d("Bluetooth","Err: Device not found or cannot connect");
165                     success=false;
166                 }
167
168
169
170
171                 return success;
172
173             }
```

La función findFrontFacinCamera permite determinar si la cámara trasera existe en el dispositivo, entrega la dirección del dispositivo.

```
174     private int findFrontFacingCamera() {
175         int cameraId = -1;
176         // Search for the front facing camera
177         int numberOfCameras = Camera.getNumberOfCameras();
178         for (int i = 0; i < numberOfCameras; i++) {
179              CameraInfo info = new CameraInfo();
180              Camera.getCameraInfo(i, info);
181              if (info.facing == CameraInfo.CAMERA_FACING_BACK) {
182                   Log.v("MyActivity", "Camera found");
183             cameraId = i;
184             break;
185            }
186           }
187           return cameraId;
188         }
```

La función onPause permite liberar a la cámara, sin esta función el programa se bloquea. Mediante la sentencia de Camera.open(id) se abre de nuevo la cámara.

```
189    @Override
190    protected void onPause() {
191         if (timer!=null){
192             timer.cancel(true);
193        }
194      releaseCamera();
195      super.onPause();
196     }
197    private boolean safeCameraOpen(int id) {
198        boolean qOpened = false;
199
200        try {
201          releaseCamera();
202          camera = Camera.open(id);
203          qOpened = (camera != null);
204        } catch (Exception e) {
205            Log.e(getString(R.string.app_name), "failed to open Camera");
206            e.printStackTrace();
207        }
208        return qOpened;
209      }
210     private void releaseCamera() {
211         if (camera != null) {
212              camera.stopPreview();
213            camera.release();
214            camera = null;
215          }
216       }
219    PictureCallback myJpeg = new PictureCallback() {
220        @Override
221        public void onPictureTaken(byte[] data, Camera myCamera) {
222           // TODO Auto-generated method
223             camera.startPreview();
224           tempdata=data;
225           ImagM= BitmapFactory.decodeByteArray(data, 0, data.length);
226    //     view.setImageBitmap(ImagM);
227           picw = ImagM.getWidth(); pich = ImagM.getHeight();
228
229           Message.obtain(threadHandler, MainActivity.NEXT, "").sendToTarget();
230
231
232
233
234        }
235
236        };
237
```

Se crea un bitmap del tamaño de las imágenes tomadas, se ejecuta la función resta utilizando las dos imágenes tomadas en el hilo, por último el resultado en el Bitmap bm y por ultimo visualizada en pantalla.

```
239         public void Operation()
240         {
241
242
243
244
245             // open the camera and pass in the current view
246
247             Bitmap bm = Bitmap.createBitmap(picw, pich, Bitmap.Config.ARGB_4444);
248             pix = Resta(Imag,Imag1);
249             bm.setPixels(pix, 0, picw, 0, 0, picw, pich);
250             view.setImageBitmap(bm);
251
252
253         }
```

En Resta se declaran las variables a operar. Mediante la sentencia .getWidth() y .getHeight() se obtiene el ancho y alto de la imagen y mediante .getPixels() se pasa de un mapa de bits a un vector, esto para poder acceder a los pixeles de la imagen.

```
255     public int[] Resta(Bitmap mBitmap, Bitmap aBitmap)
256     {
257         int picw, pich;
258         int r31 = 0;
259         int g31 =0;
260         int b31=0;
261         picw = mBitmap.getWidth();
262         pich = mBitmap.getHeight();
263         int[] pix = new int[picw * pich];
264         mBitmap.getPixels(pix, 0, picw, 0, 0, picw, pich);
265         int[] pix1 = new int[picw * pich];
266         int[] dilate =new int[picw*pich];
267         int[] dilate1 =new int[picw*pich];
268         int[] dilate2 =new int[picw*pich];
269         int[] dilate3 =new int[picw*pich];
270         aBitmap.getPixels(pix1, 0, picw, 0, 0, picw, pich);
271         int[] pixt = new int[picw * pich];
```

El ciclo for es utilizado para poder operar cada pixel de la imagen, a través de desplazamientos se acceden a los colores que están en formato RGB. Ya tendiendo la imagen descompuesta en los colores rojo, verde y azul, se procede a hacer la resta, si el pixel resultante está en el umbral de ruido, se procede a determinar que no existe diferencia y por lo tanto se recupera la imagen original.

```
272         for (int y = 0; y < pich; y++)
273         for (int x = 0; x < picw; x++)
274         {
275             // Pixels are located sequentially inside a very long array
276             int pixelLocation = x + y*picw;
277             // The R,G,B values from our 2 images
278             float r1 = (pix[pixelLocation] >> 16) & 0xff;
279             float g1 = (pix[pixelLocation] >> 8) & 0xff;
280             float b1 = pix[pixelLocation] & 0xff;
281             float r2 = (pix1[pixelLocation] >> 16) & 0xff;
282             float g2 = (pix1[pixelLocation] >> 8) & 0xff;
283             float b2 = pix1[pixelLocation] & 0xff;
```

```
284             // We substract pixel by pixel (RGB values): 3rd image = 2nd - 1st
285             float r3 = Math.abs(r2 - r1);
286             float g3 = Math.abs(g2 - g1);
287             float b3 = Math.abs(b2 - b1);
288             if(r3<1 && g3<1 && b3<1){
289                 r31=(int)r1;
290                 g31=(int)g1;
291                 b31=(int)b1;
292             }
293
294             else{
295                 r31=(int)r3;
296                 g31=(int)g3;
297                 b31=(int)b3;
298             }
```

Se pasa a escala de grises mediantes la ecuación que es almacenada en la variable R y luego se pasa a blanco y negro. Se aplica la operación morfológica de dilatación tantas veces hasta que el ruido disminuya. Se crea un mapa de bits con el vector de pixeles resultado. Log.d es utilizado para imprimir un mensaje en la herramienta de Debug de Android.

```
299             int R = (int)(0.299*r31 + 0.587*g31 + 0.114*b31);
300
301             pixt[pixelLocation] = 0xff000000 | (R << 16) | (R << 8) | R;
302
303
304             if(pixt[pixelLocation]>0xff000000){
305                 pixt[pixelLocation]=0xff000000;
306             }
307
308             else{
309                 pixt[pixelLocation]=0xffffffff;
310
311             }
312
313
314         }
315         dilate3=dilate(pixt);
316         dilate2=dilate(dilate3);
317         dilate1=dilate(dilate2);
318         dilate=dilate(dilate1);
319         dilate=centroid(dilate);
320         Bitmap bm = Bitmap.createBitmap(picw, pich, Bitmap.Config.ARGB_4444);
321         bm.setPixels(pixt, 0, picw, 0, 0, picw, pich);
322         //view1.setImageBitmap(bm);
323         text.setText(Integer.toString(ci));
324         text1.setText(Integer.toString(cj));
325         Log.d(Integer.toString(cj),Integer.toString(picw));
326
327         return dilate;
328     }
```

La función dilate() da blanco o el código FFFFFFFF cuando se cumple la condición de si hay color blanco en la vecindad del pixel de color negro o código FF000000.

```
329    public int[] dilate(int[] pixt){
330        int[] dilate =new int[picw*pich];
331        for (int y = 1; y < pich-1; y++)
332        for (int x = 1; x < picw-1; x++)
333                {
334                   int pixelLocation = x + y*picw;
335                   dilate[pixelLocation]=pixt[pixelLocation];
336
337                   if(pixt[pixelLocation]==0xff000000 &&
338                           (pixt[pixelLocation-1]==0xffffffff||pixt[pixelLocation+1]==0xffffffff
339                           ||pixt[pixelLocation-picw]==0xffffffff||pixt[pixelLocation+picw]==0xffffffff){
340
341                        dilate[pixelLocation]=0xffffffff;
342                   }
343
344                }
345        return dilate;
346    }
```

Utilizando el for se accede al pixel de la imagen al cual se quiere calcular el centroide, si se cumple que el pixel es negro se suma en 1 la variable num_pixeles, y se suman las coordenadas tanto x como y. Por último se divide la suma acumulada de coordenadas con la variable num_pixeles.

```
346    public int[] centroid(int[] pixt){
347
348        int num_pixeles=0, sum_i=0, sum_j=0,pixelLocation;
349        for (int y = 1; y < pich-1; y++)
350        for (int x = 1; x < picw-1; x++){
351                pixelLocation = x + y*picw;
352                if(pixt[pixelLocation]==0xff000000){
353                    num_pixeles= num_pixeles + 1;
354                    sum_i = sum_i + x;
355                    sum_j = sum_j + y;
356                }}
357        ci = sum_i/num_pixeles;
358        cj = sum_j/num_pixeles;
359        text2.setText(Integer.toString(num_pixeles));
360        pixelLocation = ci+ cj*picw;
```

Cuando se calcula lo anterior utilizando la sentencia DataOut.writeInt() se envía por bluetooth el dato entero cj, pich y num_pixeles.

```
367         try {
368             DataOut.writeInt(cj);
369             DataOut.writeInt(pich);
370             DataOut.writeInt(num_pixeles);
371             DataOut.flush();
372             try {
373                 Thread.sleep(1);
374             } catch (InterruptedException e) {
375                 // TODO Auto-generated catch block
376                 e.printStackTrace();
377             }
378         } catch (IOException e) {
379             // TODO Auto-generated catch block
380             e.printStackTrace();
381         }
```