UNIVERSIDAD CARLOS III DE MADRID

TRABAJO FINAL

Departamento de Ingeniería de Sistemas y Automática.

Robótica II

Mª Pilar Molina Delgado NIA 100073815

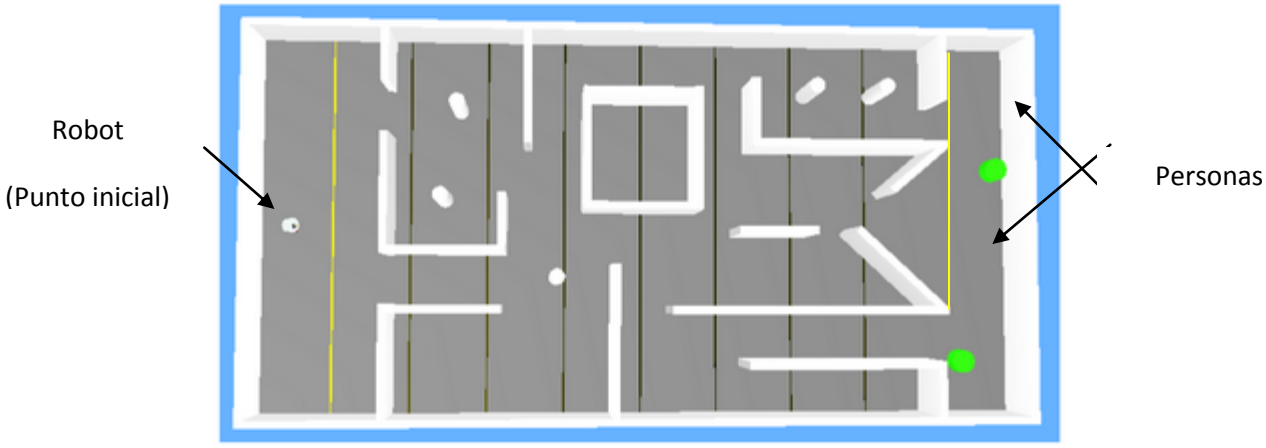
Luis Alejandro Martin Veloza NIA 100278361

Grupo 21 (4º) Curso 2014/2015

Índice

1. Descripción problema

El objetivo principal de este trabajo consiste en poder atravesar con un robot (Pioneer II) distintos entornos programados, en el programa Webots 7, pueden diferenciarse en: la distribución de obstáculos y paredes, iluminación, daños en el entorno y humo. Después de atravesar el entorno y cruzar la segunda línea amarilla el robot tiene que encontrar a dos personas (representadas en el entorno por dos cilindros verdes) y regresar a su punto inicial.



(Entorno/Mundo)

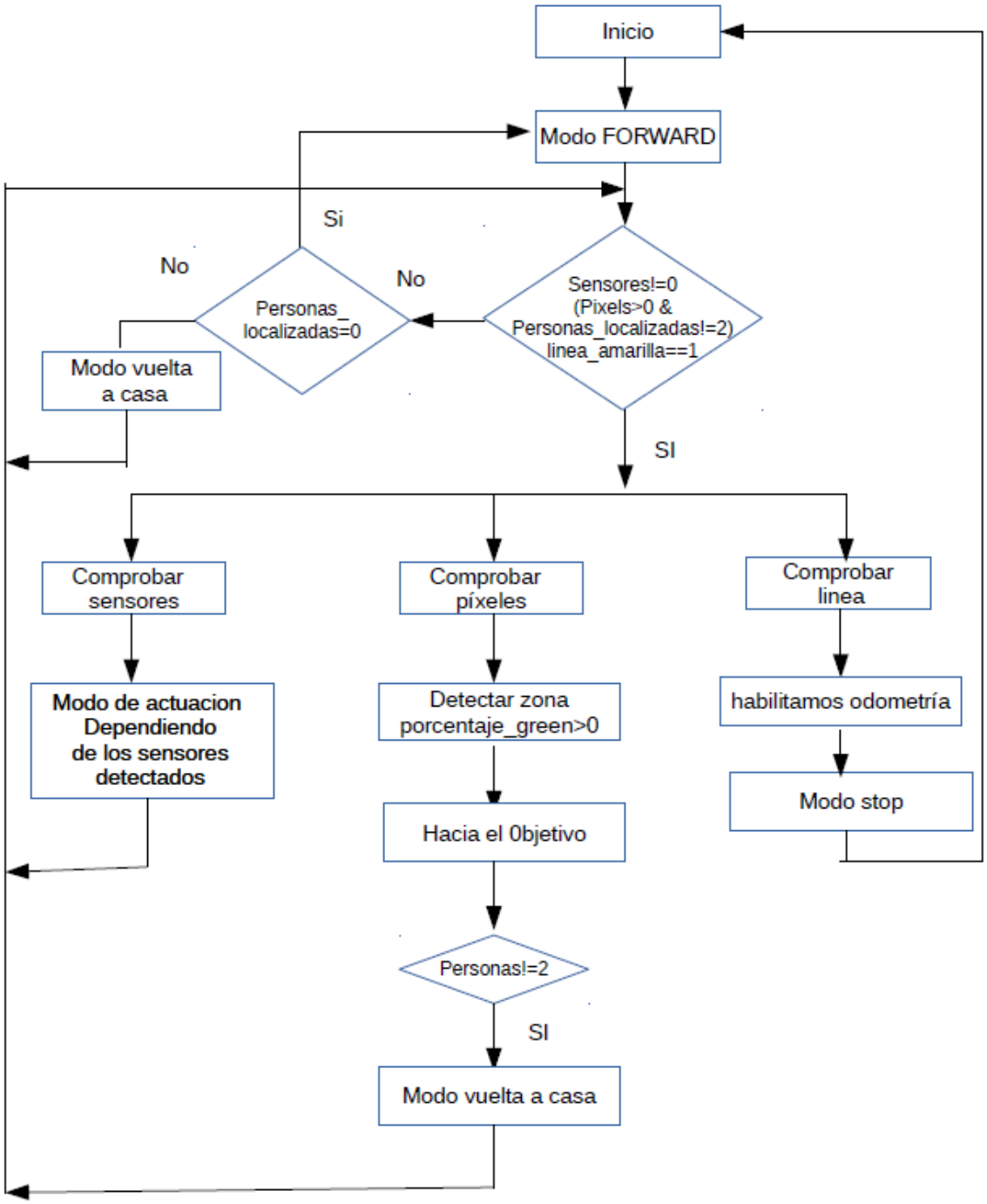
Los principales problemas a la hora de enfrentarse a este tipo de programación es claramente, la generalización de la solución. Es decir, que la solución lógica sea adecuada para todo tipo de situaciones y todo tipo de entornos diferentes.

Como se ha introducido anteriormente, hay varias fases de ejecución del problema.

- En primer lugar, atravesar el entorno, y cruzar la línea amarilla que indica el final del mundo. Para ello será necesario salvar todos los posibles obstáculos encontrados.
- En segundo lugar, proceder al rescate de los objetivos localizados.
- En tercer y último lugar, conseguir volver al inicio del escenario en un tiempo determinado (menos de 4:00 minutos)

2. Diagrama de estados

A continuación se muestra la máquina de estados básica diseñada para el controlar el robot de manera adecuada. Esta máquina de estados es una simplificación del proceso real de control del robot, ya que en ese diagrama no se especifican todos los modos de funcionamiento concretos así como todas las características a tener en elementos tomados en cuenta para tomar la decisión. Sí que sirve como guía básica para entender la lógica principal que tomará el robot.



Como se puede observar, el robot siempre tendrá un modo de funcionamiento principal a seguir. Este modo será el modo recto (Forward). Este modo recto, será hacia un objetivo (fin del mundo o inicio del mundo) según lo especifique su lógica de control. Mientras que mantiene este modo principal, siempre estará atento a las posibles condiciones que pueden hacer cambiar su funcionamiento. Esto es, la entrada al estado de toma de decisión según las diferentes respuestas ante las siguientes preguntas:

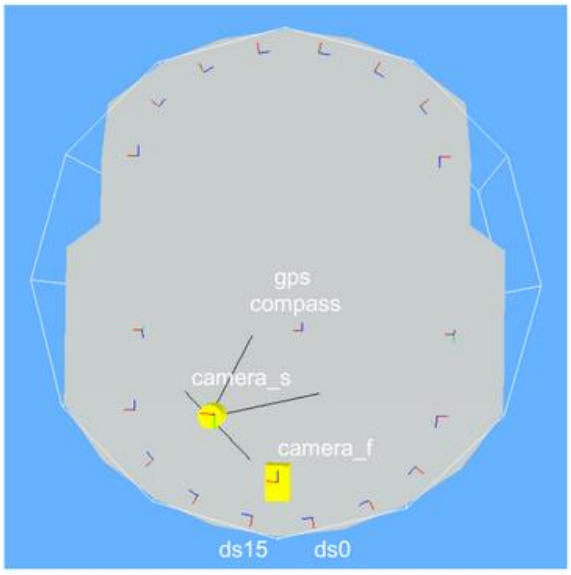
- ¿Los sensores de distancia detectan algo?
- ¿La cámara esférica ha detectado algún objetivo
- ¿Se ha detectado la línea amarilla en la cámara frontal?

Según las posibles respuestas a estas preguntas, se procederá a entrar en una lógica diferente.
Para el primer caso, en el que se detecta algo en los sensores de distancia, la máquina de estados inducirá al robot a moverse en función de las lecturas de los sensores, para posteriormente, al dejar de detectar, volver al modo inicial "Forward".
Para el segundo caso, en el que se detecta un objetivo (persona) a rescatar, la máquina de estados le dirá al robot que tiene que moverse en función de esas lecturas, para intentar mantener la máxima lectura de verde en su parte frontal.
Para el tercer caso, en caso de detección de línea amarilla, que será detectada por la cámara frontal; se inducirá al robot a activar el cálculo por odometría de distancia, para estimar los metros recorridos para finalmente concluir la misión.

Esta máquina de estados simple, será la que gobierne en todo momento el correcto funcionamiento del robot.

3. Proceso de diseño

Como se ha descrito en el apartado uno el robot se va a enfrentar a distintos casos.
Para conseguir enfrentarnos de manera eficaz a diferentes y cambiantes entornos, utilizararemos como elementos principales de percepción del robot los 16 sensores (ds) que el robot lleva instalado a lo largo de su perímetro, una cámara frontal que se encargara de visualizar todo aquello que este en línea de visualización directa con el robot, una cámara esférica que servirá como un vigilante de todo aquello que puede ser visible en un radio amplio del robot y una brújula (compass) que servirá de guía principal para orientarnos dentro del escenario.



Como se ha comentado, para diseñar de manera correcta un programa que funcione en todos los entornos posibles, hemos utilizado los dispositivos antes nombrados. A continuación se realiza un comentario más detallado de cada uno de ellos:

- 3.1 Los sensores de distancia

Los sensores los utilizamos para poder dos causas principales. Por un lado y como es obvio, detectar cualquier elemento en el camino que pueda considerarse como un obstáculo y de esa manera, reaccionar ante él. Por otro lado, gracias a los sensores de distancia conseguimos implementar una especie de comportamiento de seguir los perímetros de los obstáculos y las paredes y que así el robot no se choque y pueda llegar al final, independientemente de los diferentes entornos meteorológicos o externos que puedan hacer cambiar elementos básicos como, la luz disminuya o el nivel de claridad (niebla)

- 3.2 La brújula

.

La brújula la usamos para preestablecer la dirección y orientación del robot y que este siempre intente ir en una dirección que nosotros le marquemos en el mundo. De esta manera y aunque encuentre obstáculos y, por consiguiente tenga que reaccionar ante ellos, tenga la posibilidad reorientarse y proseguir el avance hacia el objetivo principal que se le haya encomendado.

- 3.3 La Cámara Esférica

La cámara esférica será utilizada principalmente como "gran ojo" del robot. Es decir, debido a su gran capacidad de visualización de objetos en 360º, será el elemento encargado de rastrar el entorno en busca de objetivos a localizar y rescatar.

- 3.4 La Camera Frontal

La cámara frontal, será utilizada principalmente para la detección de las líneas amarillas, ya estas siempre serán vistas de manera directa por el robot. También será utilizada como alternativa a la cámara esférica debido a su menor peso de procesamiento.

De esta manera y juntando todo los elementos descritos, ahora podemos definir de una manera más detalla el proceso de control lógico del robot.

Así pues el proceso de diseño se basa en cinco principales fases:

- Primera fase: orientación del robot de manera adecuada para que el objetivo en primer momento sea conseguir llegar al final del mundo.
- Segunda fase: reaccionar de manera adecuada al entorno que le rodea (diferentes obstáculos del mundo)
- Tercera fase: cambiar el objetivo principal a ser el rescate de personas. Para ello cambiar a un modo de búsqueda de objetos verdes.
- Cuarta fase: señalizar correctamente el salvamento de los objetivos.
- Quinta fase: cambiara a objetivo principal, volver al inicio del mundo para completar la misión.

Todas estas fases deberán ser implementadas de tal manera que el robot no pueda entrar en conflicto, y de tal manera que siempre tenga presente el objetivo claro a conseguir.

Esta distinción de casos, así como evitar que entre en conflicto, se implementara mediante bucles de condición principalmente tipo if y else, quienes a través de sus condiciones para la entrada a esos bucles, serán quienes discriminen ente el caso a aplicar.

- 3.5 Alternativas de proceso diseño:

Realmente hay que señalar que las posibles alternativas planteadas no vienen dadas de manera general, sino más bien centradas en algunas partes concretas del programa.

- En nuestro caso, una de las mayores partes donde hemos encontrado más problemas y por tanto, más alternativas a proceder, viene dada en la parte de rescate de objetivos. Se planteó la posibilidad de proceder al rescate de una persona, su

señalización y vuelta al inicio, para posteriormente, proseguir en la búsqueda y rescate de la segunda persona.

Se desechó esta idea debido a la gran cantidad de tiempo que requería realizar dos veces el mundo. Por tanto, se optó por la recogida y señalización de las dos personas y de su vuelta a casa tras los dos rescates.

- Otra parte principal de planteamiento de diferentes alternativas, fue la parte del guardado de posiciones para no volver a rescatar dos veces al mismo objetivo.
  Se plantearon tres formas:
  Una primera forma que se basaría en el guardado de la posición mediante GPS, para posteriormente no volver a dicha posición para rescatar.
  Una segunda forma, basada en odometría, para conseguir calcular de manera matemática la posición en todo momento del objetivo ya rescatado.
  Por último, una tercera vía, basada en los ángulos. Determinar en todo momento el ángulo en el que estaba la persona ya rescatada, para no volver a rescatarla.
  Por desgracia, no tuvimos éxito a la hora de aplicar las diferentes soluciones planteadas.

- 3.6 Justificación del modelo de diseño:

  Tras varias, pruebas con diferentes modelos de comportamiento, decidimos centrarnos en el modelo anteriormente comentado. Un modelo que principalmente está basado en la reacción gracias a las lecturas de los sensores, hasta el momento de detectar verde en la cámara esférica. Tras esto, el modo prioritario será el de encontrar y señalizara a las dos personas, para posteriormente, poner rumbo a la vuelta a casa. Además, realizamos la activación y desactivación de diferentes elementos para mejorar la carga de procesamiento en la simulación. Si desde el primer momento se hubiera optado por la activación de todos los elementos, la carga de simulación hubiera sido enorme.

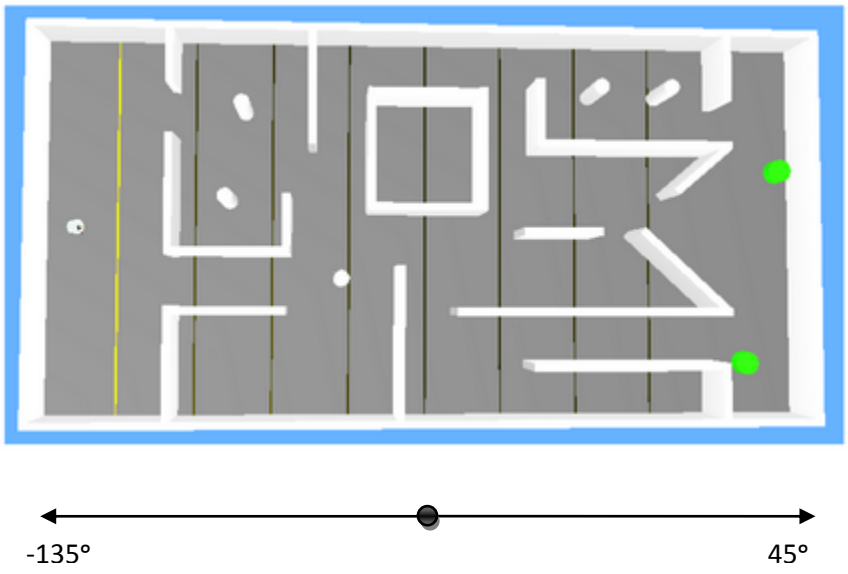
4. Descripción de la solución.

Como se ha comentado antes, principalmente se actuara por medio de diferentes elementos. El peso y funcionamiento de esos elementos, vienen comentados a continuación.

- 4.1 Recorrido del mundo por medio de los sensores:

Los sensores serán utilizados durante gran parte de la simulación. Solo habrá un caso concreto en los que no participen en la toma de decisiones. Este caso será cuando encontremos a un objetivo (persona). En ese momento, los sensores de distancia serán desactivados para dar un papel principal a otros elementos que tomaran el control principal del robot. Toda la parte del recorrido del mundo por medio de los citados sensores se encuentra la programación pura, más concretamente en la parte de código dentro del archivo MyRobot.cpp indicada como: *logic control área of distance sensors*.

Para conseguir el objetivo principal de recorrer el mundo y que el robot siempre intente dirigirse hacia el final del mundo, el ángulo establecido para la variable DESIRED_ANGLE debe ser de 45 grados.

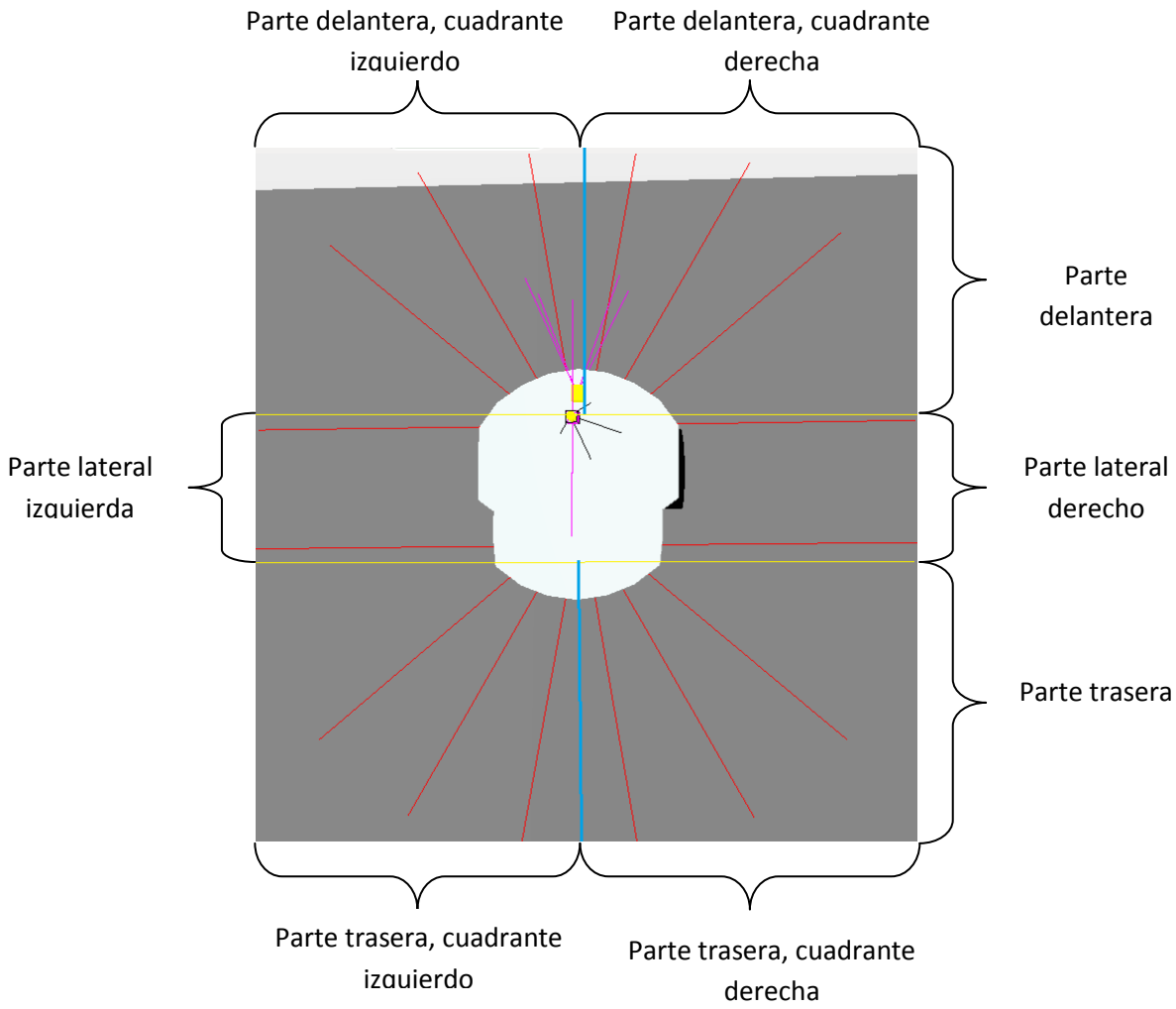


Estos 45º grados, serán los encargados de reorientar al robot en caso de haber tenido que reaccionar ante diferentes situaciones. De esta manera conseguimos que el robot no pierda el objetivo principal de avanzar hacia el final del escenario.

Volviéndonos a centrar en la parte de los sensores, se definieron varios sectores principales de agrupamiento de sensores según la zona de ubicación.

A continuación, se muestra un gráfico del robot, donde se detallan la definición de esas diferentes zonas en función de posición.

Partes del Robot:



Como se puede observar, existen varias partes. Lo principal será siempre comprobar toda la parte frontal del robot, debido a que la lógica nos dice que el robot se encontrara los obstáculos principalmente de frente, ya nuestra intención es que se dirija siempre de manera recta para poder avanzar hasta parte final del mundo. Es por ello que hemos intentado programar lo mejor posible y con más cuidado en esta parte concreta del robot, para poder analizar mas detenidamente los movimientos del robot dependiendo de que objetos y de cómo los haya detectado.
Ahora, dentro de esta frontal de sensores dividiremos todo en tres posibles casos:

- El primero es el control de los sensores en el cuadrante izquierdo dentro de los frontales, es decir el ds0, ds1 y ds2.
- El segundo es el control de los sensores centrales dentro de los frontales es decir el ds0, ds15, ds14 y ds1.
- El tercero es el control de los sensores en el cuadrante derecho de los frontales, es decir ds13, ds14 y ds15.

En caso de no producirse lecturas en estas zonas, el robot se procederá a establecerse en modo "Forward", con lo que seguirá una trayectoria recta.

Centrándonos ahora en otra parte importante, los giros, para poder controlarlos de manera correcta, así como los objetos cuando se tienen en alguna o ambas partes laterales, o también en la parte trasera, se crearon diferentes funciones a las cuales se les introduce los valores de los sensores.
Dichas funciones son:

- *detection_and_turn_right(ir3_val,ir4_val,compass_angle,α):*
  Esta función la usamos cuando detectamos algo solo en el cuadrante izquierdo de la parte frontal, puesto que puede darse el caso que haya una pared o un obstáculo que también puedan detectar los sensores laterales. Por ello pasamos los valores de los sensores 3 y 4 y modificamos el ángulo que tiene el robot predeterminando (ángulos α) para que realice los giros más exactos.
  Dependiendo de como detectemos el objeto y que sensor marque, mas α va a será introducido.

- *detection_and_turn_left(ir3_val,ir4_val,compass_angle,α):*
  Esta función tiene un objetivo muy similar a *detection_and_turn_right* . La diferencia es que se realizan los giros hacia la izquierda, ya que esta función será utilizada cuando detectamos algo en el cuadrante derecho de la parte frontal y en los sensores laterales izquierdos.

- *obstacles_to_the_sides_look(ir12_val,ir11_val,ir3_val,ir4_val,ir7_val,ir8_val, compass_angle, α, β):*
  Esta función es más especifica que las explicadas anteriormente, ya que se activara cuando tengamos alguno de los sensores centrales frontales activados, sin discriminar por cuadrantes y además, le introduciremos los valores de los sensores laterales izquierdos, derechos y traseros para considerarlos en su movimiento posterior.
  Además, se le introducirán dos ángulos diferentes, para poder cambiar el ángulo de giro dependiendo de si están los sensores laterales izquierdos o derechos realizando lecturas.

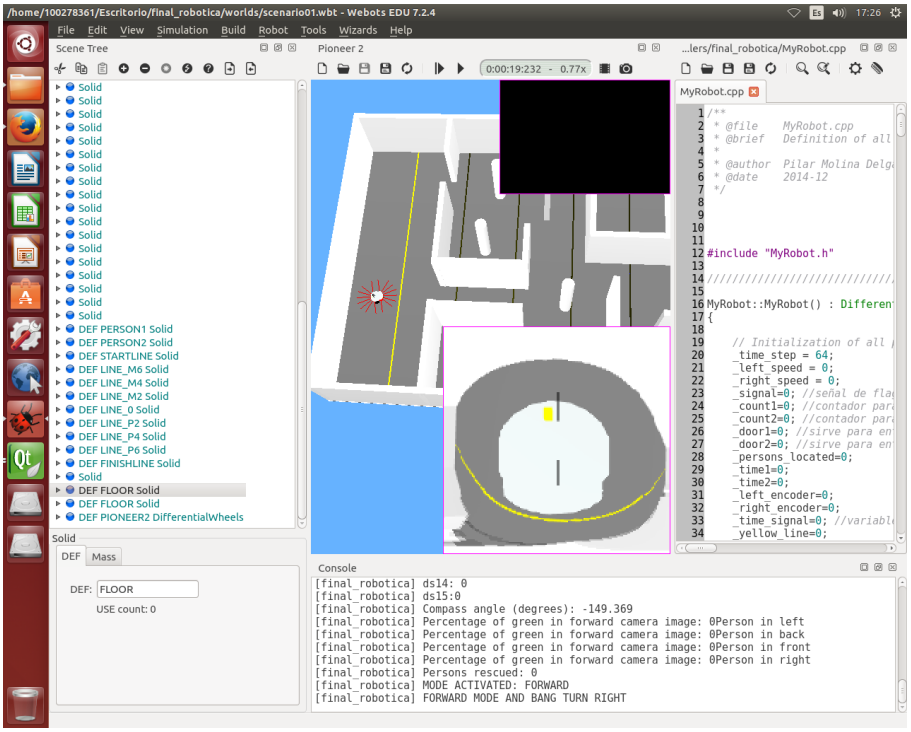
- 4.2 Programación y utilización de la brújula:

La brújula es una parte fundamental en este proyecto, puesto que gracias a ella conseguimos dar la orientación de robot. No solo eso, sino que además podremos indicarle que mire hacia un lado o a hacia otro, modificando el ángulo que nos da la brújula. Es decir, la brújula no solo sirve para llegar al fin del mundo, o en el caso contrario, para volver. Sino que también sirve para conseguir que siga un ángulo específico en el caso que se haya detectado un objetivo.

Para conseguir los grados de la brújula utilizamos la función *convert_bearing_to_degrees()*.

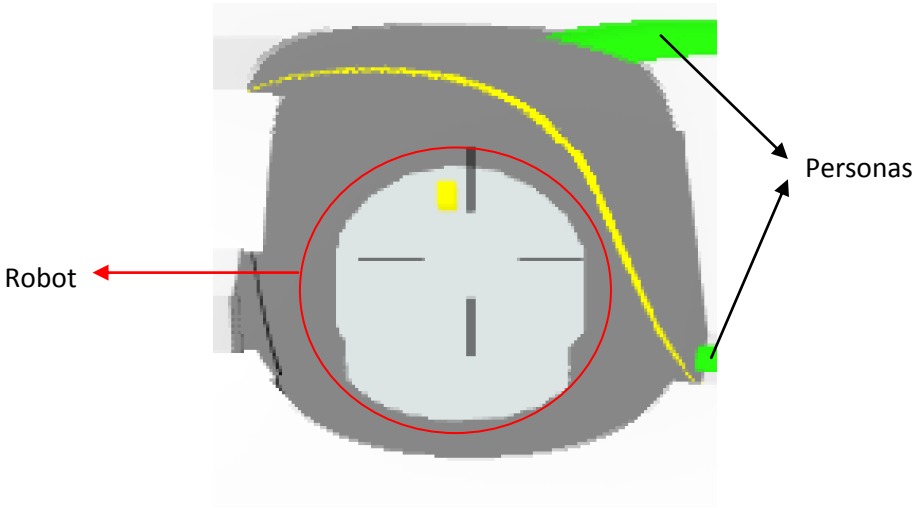
- 4.3 Programación de la cámara frontal:

Una vez que hemos encontrado a las personas (será necesario que hayan sido localizadas 2 personas) el robot entrará en el modo "Vuelta a casa". Para ello cambiamos el DESIRED_ANGLE (que en el caso inicial de ida hacia el fin del mundo, el ángulo definido es 45 grados) a un ángulo definido como 45 -90 para que el robot vaya al sentido contrario y así poder recorrer el mundo en el sentido contrario. Es decir, un ángulo de -135º proporcionara al robot el objetivo de dirigirse hacia el inicio del mundo.
En este modo, desactivamos la cámara esférica para no cargar de procesamiento innecesario la simulación y activamos la cámara frontal para poder ver la línea amarilla cuando nos acercamos al inicio del mundo. Una vez detectada la línea amarilla, usaremos la edometría para poder contabilizar los metros recorridos del robot. Esto lo hacemos gracias a que la cama frontal deja de ver la línea amarilla, cuando aun quedan unos metros hasta que el robot la sobrepase, podemos medir los metros que recorre el robot y así, que siga avanzando los metros que le faltan hasta la línea amarilla y se posteriormente se detenga tras haberla dejado atrás.



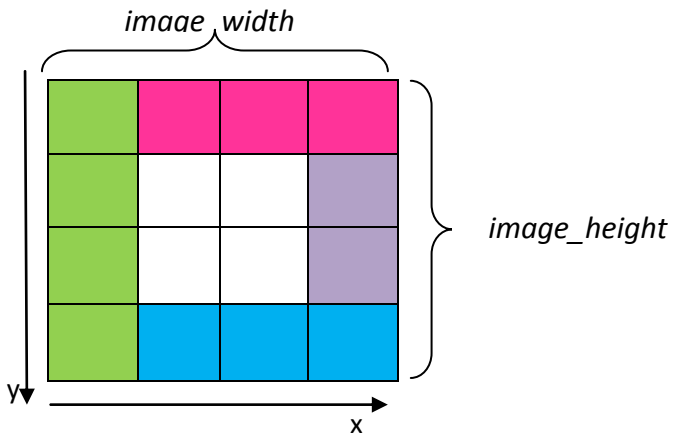
- 4.4 Programación de la cámara esférica:

El objetivo principal para la visualización de las personas, será el uso de la cámara esférica del siguiente modo:
Para hacer más sencillo el recorrido de la lectura de los pixeles de la imagen, solo vamos a recorrer el marco exterior de la imagen puesto que, en el centro de ésta, lo que se visualiza es el propio robot; una información que no nos interesa obtener ni procesar.



Dividimos la imagen en 8 cuadrantes y por medio de 4 bucles *for,* solo realizamos la lectura de los marcos exteriores, poniendo como limites los que se citan a continuación. El total de la imagen está predeterminado por las variables que nos da la cámara que son:
Ancho: *image_width* (w)
Alto: *image_height* (h)

Al utilizar estas variables como límites, nos permite que si en algún momento se amplía la resolución de la cámara, esta lectura se pueda amoldar a la nueva resolución.



Parte izquierda (color verde):
Inicio eje x: x=0; hasta que x <=w/4.
Inicio eje y: y=0; hasta que y <=h.

Parte de abajo (color azul):
Inicio eje x: x=w/4; hasta que x <=w.
Inicio eje y: y=(3h)/4; hasta que y <=h.

Parte de arriba (color rosa):
Inicio eje x: x=w/4; hasta que x <=w.
Inicio eje y: y=0; hasta que y <=y/4.

Parte izquierda (color morado):
Inicio eje x: x=(3w)/4; hasta que x <=w.
Inicio eje y: y=y/4; hasta que y <=(3h)/4.

Cuando nuestro robot ve a una persona con la cámara esférica y esta a mayor distancia de un metro, comparamos el porcentaje de pixeles verdes que tenemos en cada porción del marco de la imagen. Una vez comparado, se procede a asignar el ángulo concreto que le corresponde a esa posición del marco, es decir, si es la parte izquierda del marco la marcada como objetivo, al ángulo que en ese momento nos marca el robot con la brújula, le restamos -90 grados.
De manera similar, si es la parte trasera del robot o parte de abajo del marco, entonces se le suman +90 grados.
Po último, si es en la parte frontal dejamos el ángulo como esta.
Esto se hace para que el robot se fije en la persona y así podamos dirigirnos a ella. Una vez que localizamos a la persona y estamos a menos de un metro (esto nos lo indican los sensores puesto que empiezan a detectar objetos cuando estos están a un metro o menos) deshabilitamos los sensores y entramos en el modo de SIGNALING_PERSON donde procederemos a esperar 2 segundos. Pasados estos 2 segundos giramos el robot sobre sí mismo para señalizar el rescate de la persona.

- 4.5 Alternativas:

Una de las alternativas posibles seria la utilización del GPS para ubicar a la persona en el mundo y así no volver a ella, en este caso no se ha utilizado por que el error cometido es de 3 a 10m con lo que podía sitúanos a la persona fuera del mundo y eso no nos serviría.

Por lo demás hemos intentado ver todas las posibilidades que tiene el robot para realizar todas las tareas y las utilizadas son para nuestro criterio las mas fáciles y útiles, pero no absentas de mejoras en la programación.

- 4.6 Diferentes funciones utilizadas:
  Las siguiente funciones se programaron para reducir considerablemente el código, pero debido a algún tipo de problema (no daba warning, ni error), a veces el controlador entraba en "crash". Así que, por precaución, no se han añadido al código aunque si se probaron de manera individual, siendo correcto su funcionamiento.

  - 4.6.1 Funciones para obtener el verde de la cámara esférica:

```
int MyRobot::obtain_green_scamera_pix1(int image_width_s, int
image_height_s)
{
int pix=0;
unsigned char green = 0, red = 0, blue = 0;

// Get current image from forward camera
const unsigned char *image_s = _spherical_camera->getImage();

    for (int x = 0; x < (image_width_s/4); x++)
    {
        for (int y = 0; y < image_height_s; y++)
        {
            // Values RGB
            green = _spherical_camera->imageGetGreen(image_s,
image_width_s, x, y);
            red = _spherical_camera->imageGetRed(image_s,
image_width_s, x, y);
            blue = _spherical_camera->imageGetBlue(image_s,
image_width_s, x, y);

            // To detect the green RGB values
            if ((green >75) && ((green-75)/red >= 2) && ((green-
75)/blue >= 2))
            {
                // Having green pixel in that area
                pix = pix + 1;
            }
        }
    }
    return pix;
}

int MyRobot::obtain_green_scamera_pix2(int image_width_s, int
image_height_s)
{
int pix=0;
unsigned char green = 0, red = 0, blue = 0;

// Get current image from forward camera
```

```
const unsigned char *image_s = _spherical_camera->getImage();

for (int x =(image_width_s/4); x < image_width_s; x++)
{
    for (int y = (3*(image_height_s/4)); y < image_height_s; y++)
    {
        // Values RGB
        green = _spherical_camera->imageGetGreen(image_s,
image_width_s, x, y);
        red = _spherical_camera->imageGetRed(image_s, image_width_s,
x, y);
        blue = _spherical_camera->imageGetBlue(image_s,
image_width_s, x, y);

        // To detect the green RGB values
        if ((green >75) && ((green-75)/red >= 2) && ((green-75)/blue
>= 2))
        {
            // Having green pixel in that area
            pix = pix + 1;
        }
    }
}
return pix;

}

int MyRobot::obtain_green_scamera_pix3(int image_width_s, int
image_height_s)
{
    int pix=0;
    unsigned char green = 0, red = 0, blue = 0;

    // Get current image from forward camera
    const unsigned char *image_s = _spherical_camera->getImage();

    for (int x =(image_width_s/4); x < image_width_s; x++)
    {
        for (int y = 0; y < (image_height_s/4); y++)
        {
            // Values RGB
            green = _spherical_camera->imageGetGreen(image_s,
image_width_s, x, y);
            red = _spherical_camera->imageGetRed(image_s,
image_width_s, x, y);
            blue = _spherical_camera->imageGetBlue(image_s,
image_width_s, x, y);

            // To detect the green RGB values
            if ((green >75) && ((green-75)/red >= 2) && ((green-
75)/blue >= 2))
            {
                // Having green pixel in that area
                pix = pix + 1;
            }
        }
```

```
    }
    return pix;
}


int MyRobot::obtain_green_scamera_pix4(int image_width_s, int
image_height_s)
{
    int pix=0;
    unsigned char green = 0, red = 0, blue = 0;

    // Get current image from forward camera
    const unsigned char *image_s = _spherical_camera->getImage();

    for (int x =(3*(image_width_s/4)); x < image_width_s; x++)
    {
        for (int y = (image_height_s/4); y < (3*(image_height_s/4));
y++)
        {
            // Values RGB
            green = _spherical_camera->imageGetGreen(image_s,
image_width_s, x, y);
            red = _spherical_camera->imageGetRed(image_s,
image_width_s, x, y);
            blue = _spherical_camera->imageGetBlue(image_s,
image_width_s, x, y);

            // To detect the green RGB values
            if ((green >75) && ((green-75)/red >= 2) && ((green-
75)/blue >= 2))
            {
                // Having green pixel in that area
                pix = pix + 1;

            }

        }
    }
    return pix;
}
```

- 4.6.2 Función para mostrar el porcentaje de verde:

```
void MyRobot::display_green_in_scamera(double percentage_green_s1,
double percentage_green_s2, double percentage_green_s3, double
percentage_green_s4)
{

    cout << "Percentage of green in forward camera image: " <<
percentage_green_s1<<"Person in left"<< endl;
    cout << "Percentage of green in forward camera image: " <<
percentage_green_s2<<"Person in back"<< endl;
    cout << "Percentage of green in forward camera image: " <<
percentage_green_s3<<"Person in front"<< endl;
```

```
    cout << "Percentage of green in forward camera image: " <<
percentage_green_s4<<"Person in right"<< endl;
}
```

- 4.6.3 Función para obtener la lectura de amarillo de la cámara frontal:

```
int MyRobot::obtain_yellow_fcamera(int image_width_f, int
image_height_f)
{
     unsigned char green_f = 0, red_f = 0, blue_f = 0;
     int local_sum = 0;

    // Get current image from forward camera
    const unsigned char *image_f = _forward_camera->getImage();

    // Count number of pixels that are yellow in the captured image
    for (int x = 0; x < image_width_f; x++)
    {
        for (int y = 0; y < image_height_f; y++)
        {
            // Obtaining different RGB values
            green_f = _forward_camera->imageGetGreen(image_f,
image_width_f, x, y);
            red_f = _forward_camera->imageGetRed(image_f,
image_width_f, x, y);
            blue_f = _forward_camera->imageGetBlue(image_f,
image_width_f, x, y);

            // Logical reading to find out if there is a yellow line
in front
            if ((blue_f<=20) && ((red_f/green_f)>=0.8) &&
((red_f/green_f)<=1.1) && (red_f>=20) && (green_f>=20))
            {
                local_sum = local_sum + 1;
            }
        }
    }
   return local_sum;
}
```

- 4.6.4 Función para el control lógico según el nivel de verde leído:

```
void MyRobot::logical_control_pixels(double
percentage_green_s3,double ir0_val,double ir1_val,double ir15_val,
double ir14_val, double compass_angle )
{

    cout<<"Come to the target"<<endl;
    _mode=TURN;

    if (DESIRED_ANGLE==compass_angle)
        _mode=FORWARD;
```

```
    // If we have to target near (we set a flag of person found
_signal)
    if(percentage_green_s3>=8)
    {
        _signal=1;
    }

    // If we close and touch the target with sensors ( 1 meter )
    if(_signal==1 && ((ir0_val>0 && ir0_val<DISTANCE_LIMIT)
||(ir1_val>0 && ir1_val<DISTANCE_LIMIT) ||(ir15_val>0 &&
ir15_val<DISTANCE_LIMIT) ||(ir14_val>0 && ir14_val<DISTANCE_LIMIT)))
    {

        // Disable the distance sensors
        for (int i=0; i<NUM_DISTANCE_SENSOR; i++)
        {
            _distance_sensor[i]->disable();
        }

        // We proceed to signal the rescue
        _mode=SIGNALING_PERSON;
    }
}
```

5. Descripción de los resultados.

Escenario 1:

Llegada al final del mundo en aprox 2:00 min (según varias simulaciones).

Vuelta a casa correcta.

Problemas:

- obstáculo tipo muro en V. No consigue salir.
- Si en la detección de la persona, hay lectura en algún sensor lateral; entrada en bucle infinito.

Escenario 2:

Llegada al final del mundo en aprox 1:30 min.

Vuelta a casa por lo general correcta, aunque según el ángulo de detección de paredes, puedo no salir.

Problemas:

- obstáculo tipo muro en V. No consigue salir.
- En ciertas ocasiones, en la ida al final del mundo, no consigue salir de algún obstáculo.
- No consigue atacar con el ángulo correcto a las personas, y se queda dando vueltas alrededor de ellas.

Escenario 3:

Llegada muy cerca de final del mundo, pero siempre se encuentra con un obstáculo del que no sale.

Vuelta a casa bastante correcta.

Problemas:

- Obstáculo tipo mixto con muro y cajas en forma de L inversa. No se atasca, pero siempre se queda en esa zona del mundo.
- Problemas detección del verde por sombras.

Escenario 4:

Problemas en general para llegar al final del mundo (según varias simulaciones). Casi siempre se queda en una zona concreta del mundo intentando buscar salida

Escenario 5:

Problemas en general para llegar al final del mundo. Al igual que el escenario 5, casi siempre se queda en una zona concreta del mundo intentando buscar salida.

Escenario 6:

Problemas en general para llegar al final del mundo. Al igual que el escenario 4 y 5, casi siempre se queda en una zona concreta del mundo intentando buscar salida.

Escenario 7:

Problemas en general para llegar al final del mundo. Al igual que el escenario 4,5 y 6, casi siempre se queda en una zona concreta del mundo intentando buscar salida.

Nota: Al tener una morfología de obstáculos parecidos los mundos 4, 5,6 y 7, entra siempre en el mismo problema.

Escenario 8:

Llegada al final del mundo en aproximadamente 1:30 min.

Vuelta bastante buena.

Problemas:

- Debido a la disposición de las personas según las entradas a la zona final, no ataca con el ángulo correcto, y se queda dando vueltas alrededor de las personas intentado detectarlas con los sensores delanteros.

Escenario 9:

Importante problema nada más empezar. Debido a que nuestra programación impedía que el robot estuviera en ningún caso tan cerca de la pared, el giro para evitar la pared no es suficiente para poder salir.

Si obviamos eso, y ataca con un ángulo correcto las paredes, llegada al fin del mundo en aproximadamente 2:30 minutos.
Los objetivos son atacados y detectados correctamente.
La vuelta es correcta.

Escenario 10:

Al igual que en el escenario 9, debido a que nuestra programación impedía que el robot estuviera en ningún caso tan cerca de la pared, el giro para evitar la pared no es suficiente para poder salir.
Además, no consigue llegar a la zona de detección de personas ya que siempre entra en una zona de obstáculo que le hacer retroceder.
Los objetivos se detectan correctamente si acercamos el robot a la zona de detección.
Vuelta con correcta pero tarda bastante

6. Conclusiones y futuras mejoras.

Como conclusión se podría mejorar la programación del robot intentando hacerla mas escueta y directa, pero por falta de tiempo puesto que no disponemos del programa Webots en nuestros ordenadores personales nos ha sido mejorar este aspecto.
Por otro lado, nuestro robot detecta dos veces a la misma persona, por tanto eso es una mejora más que necesaria para el correcto funcionamiento.
En la programación de sensores, cuantos más casos tenidos en cuenta, mejor será la respuesta, por tanto es otro aspecto importante a tener en cuenta.

Como conclusión final, podemos decir que, nos hemos dado cuenta de la gran cantidad de trabajo que conlleva trabajar y programar robots. Que no es tan fácil como parece cuando uno ve un robot que funciona a la perfección. Ahora nos damos cuenta del trabajo tan importante, así como la cantidad de horas y el potencial humano necesario para conseguir un correcto funcionamiento de un robot simple.
Por otro lado, gracias a este curso más que practico, hemos ha aprendido a programar siguiendo estándares, a conocer los diferentes elementos que componen el robot, y sobre todo a ponernos en la piel de un programador de robots. Aunque el trabajo ha sido muy complejo, incluso a veces desesperante, ha sido más que enriquecedor poder trabajar en grupo para después poder exponer los diferentes planteamientos utilizados por cada grupo, y así poder ver la cantidad de soluciones diferentes ante un mismo problema.