VISION COMPUTACIONAL E INTELIGENCIA ARTIFICIAL APLICADAS A LA SOLUCION DE UN PROBLEMA DE ROBOTICA INDUSTRIAL

Luis Henriquez Valle Yanes (201210060115)
lyanes21@yahoo.es

RESUMEN: *Se utilizó el programa de computo MATLAB para dar solución a un problema de paletizado en robótica industrial, usando las toolbox de Visión, Redes neuronales y visualization Kawasaki para demostrar cómo se puede dotar de un grado de racionalidad a un manipulador industrial .*

1 INTRODUCCIÓN

Este documento pretende explicar paso por paso la lógica y programación utilizada para dar solución a nuestro problema, no se ahondara en nociones teóricas y nos limitaremos a definir los términos que vayan surgiendo en nuestra programación.

.

2 OBJETIVOS

Estos son algunos objetivos generales que se pretenden

- Solucionar el problema de paletizado propuesto.
- Hacer uso de un sistema de redes neuronales para dotar de racionalidad al manipulador.
- Hacer uso de un sistema de Visión para dotar de flexibilidad a nuestro sistema.

3 MATERIALES

Estos son algunos materiales que se utilizaron

- Cámara exterior USB Gigaware 25-157.
- Manipulador industrial Kawasaki FS003N
- MatlabR2015a [1].
- Image Processing Toolbox para Matlab [2].
- Neural Network Toolbox para Matlab [3].
- MatlabKK-robotic Toolbox [4].

4 PROBLEMA PLANTEADO

El problema de paletizado consta de dos almacenes, uno para las piezas del tipo 1 (color azul) y otro para las piezas del tipo 2 (color rojo), posee un surtidor de piezas en donde solamente podrá haber una pieza (ya sea del tipo 1 o del tipo 2).
El sistema deberá ser capaz de:

- Detectar si hay pieza en el surtidor.
- Detectar que tipo de pieza hay en el surtidor.
- Detectar el estado de los almacenes.

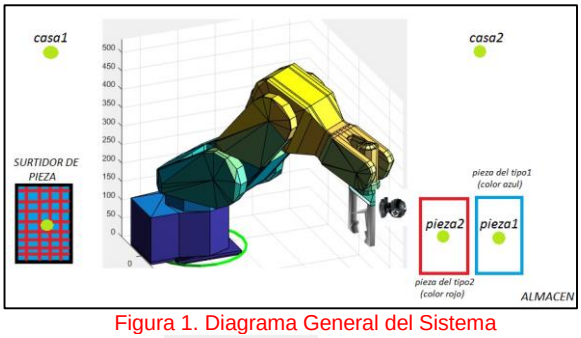

Figura 1. Diagrama General del Sistema

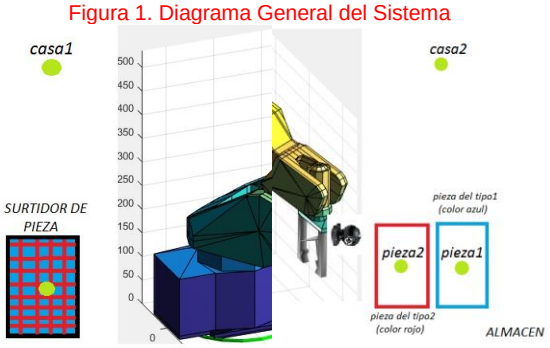

Figura 2.Zoom

Partiendo de la detección anterior se desarrollara la siguiente lógica:

- Si los almacenes están vacíos y hay una pieza en el surtidor, entonces tomara la pieza y la moverá al espacio que corresponda en el almacén.
- Si los almacenes están llenos y hay o no pieza en el surtidor, entonces no hará nada y esperara a que las condiciones cambien.
- Si los almacenes y el surtidor están vacíos, entonces no hará nada y esperara a que las condiciones cambien.
- Si un almacén esta vacío y en el surtidor hay una pieza del mismo tipo que el almacén, entonces tomara la pieza y la moverá al espacio que corresponda en el almacén, caso contrario no hará nada.

Para una mejor referencia observar la Fig.1 en donde se muestra el sistema en general.

5 DESARROLLO

Para comprender mejor la solución propuesta al problema planteado dividiremos el desarrollo en tres etapas:

- Detección de Piezas y Estados.
- Selección de Movimientos.
- Ejecución de movimientos.

En cada etapa explicaremos las herramientas y algoritmos utilizados.

5.1 DETECCION DE PIEZAS Y ESTADOS

A continuación usaremos varios comandos de la Image Processing Toolbox de Matlab [2] para capturar y procesar fotogramas provenientes de la cámara.

El algoritmo en general funciona de la siguiente manera:

5.1.1 Configuración y apertura del adaptador (cámara)

Tomando como referencia el código de ejemplo que se puede encontrar en los foros de Matlab [5] empezaremos configurando nuestra cámara, para ello usaremos el comando imaqhwinfo el cual retornara todos los adaptadores instalados en nuestra pc. fig.3

```
ans =

    InstalledAdaptors: {'winvideo'}
        MATLABVersion: '8.5 (R2015a)'
          ToolboxName: 'Image Acquisition Toolbox'
       ToolboxVersion: '4.9 (R2015a)'
```

Figura 3.

Para la configuración y apertura del adaptador usaremos el comando Videoinput el cual establece la configuración dentro de una variable, donde se establece el controlador ´winvideo´, el dispositivo ´1´ y la resolución de la imagen ´MJPG_640x480´:

```
vid = videoinput('winvideo', 1,'MJPG_640x480');
```

5.1.2 Captura de fotograma.

Una vez establecida la configuración procedemos a tomar la captura de un fotograma con el comando getsnapshot, el único parámetro que recibe (vid) corresponde a la configuración del adaptador a utilizar, el fotograma es guardado en la variable data la cual es un arreglo del tipo:

Nótese que el tamaño del arreglo corresponde al tamaño de la resolución antes definida, a su vez es multiplicada por tres, debido a los 3 valores que representan las componentes de color RGB. El uint8 representa los valores para cada casilla de los arreglos (valores comprendidos entre el 0 y el 255).

```
data = getsnapshot(vid);
```

Con el comando imshow mostraremos el fotograma (ver fig.4)

```
figure,imshow(data)
```

Figura 4.

5.1.3 Extracción de la componente de color deseada.

Con el fotograma guardado en la variable data extraemos la componente de color deseada para ellos usaremos el comando imsubtract.

```
diff_im = imsubtract(data(:,:,3), rgb2gray(data));
```

Donde data(: , : ,3) corresponde a la selección de la componente Azul, el resultado se almacenara en escala de grises. La visualización de diff_im se debe ver como en la figura 5.

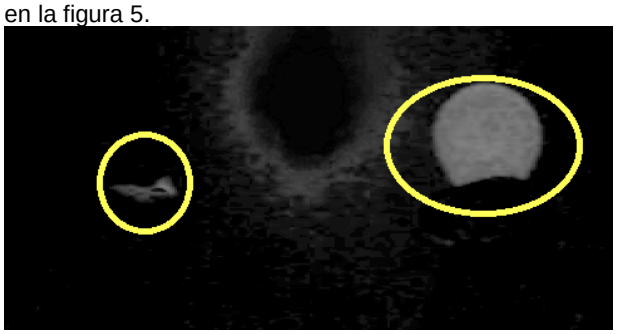

Figura 5.

5.1.4 Binarizacion de la imagen antes obtenida

Esta es una de las partes fundamentales del algoritmo, la binarizacion de la imagen nos permite ahorrar tiempo de procesado. El comando im2bw permite binarizar una imagen partiendo de un umbral definido. Los parámetros de entrada son diff_im que corresponde a la imagen que será binarizada y 0.18 que corresponde al umbral.

```
diff_im = im2bw(diff_im,0.18);
```

La visualización de diff_im en esta etapa se debe ver como en la figura 6.

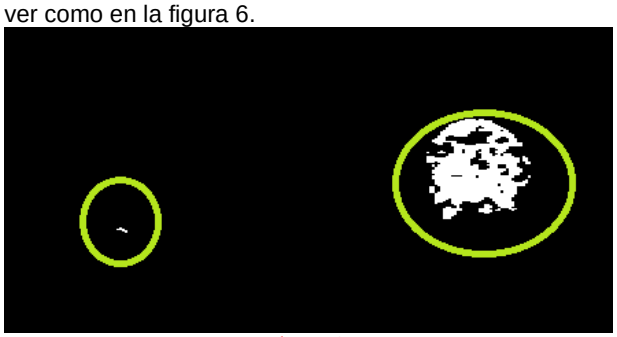

Figura 6.

5.1.5 Eliminación de ruido.

Con el fin de eliminar algunos falsos positivos (ruido) que se pueden filtrar al sistema debido a las condiciones de luz u otros factores, usaremos el comando bwareaopen el cual eliminara aquellos objetos que estén por debajo de un umbral definido.

```
diff_im = bwareaopen(diff_im,300);
```

La imagen resultante deberá ser similar a la mostrada por la figura 7. Nótese como la sección blanca más pequeña, de la figura 6, ha desaparecido.

Figura 7.

5.1.6 Etiquetar los objetos de la imagen binaria.

Dos comandos útiles para detectar las regiones blancas de una imagen binaria son bwlabel y regionprops. Es en esta sección del código donde haremos la detección de las piezas y los estados de los almacenes. La propiedad Object del comando regionprops nos devuelve la cantidad de objetos encontrados, si esta es igual a 1 indica que encontró una pieza del color analizado, si es igual a 0 indica que no encontró nada. Con esta propiedad desarrollamos nuestra lógica de estados, las mismas son almacenadas en variables distintas que serán de gran utilidad para la RNA.

```
if object == 1
A1=1;
else
A1=0;
end
if object2 == 1
A2=1;
else
A2=0;
end
```

```
if object == 1
P1=1;
else
P1=0;
end
if object2 == 1
P2=1;
else
P2=0;
end
```

Figura 8.

5.1.7 Mostrar los resultados enmarcados.

El último paso es mostrar la imagen procesada (ver figura 9), para obtener esta imagen se requiere el uso del comando rectangle, el cual nos dibuja un rectángulo en el centro de masa del objeto detectado.

```
rectangle('Position',bb,'EdgeColor','b','LineWidth',2)
```

Observe como el objeto más pequeño no es ¨detectado¨.

Con este sencillo código se garantiza la detección de una o varias piezas.

Figura 9.

5.2 SELECCIÓN DE MOVIMIENTOS

La segunda parte de nuestro sistema consiste en la creación de una red neuronal entrenada, esta decidirá los movimientos en función de sus parámetros de entrada (ver figura 8).

Para la creación de nuestra RNA se hizo uso de la herramienta nntool del neural network toolbox [3]. A continuación podrá observar los pasos que se siguieron para su entrenamiento:

5.2.1 Entradas y Salidas esperadas

Las entradas de nuestra red siguen una lógica binaria (ver tabla 1), todos los posibles valores que pueden tomar los datos de entrada son considerados.

La entrada se representa por 4 valores, dos para el tipo de pieza y dos para los almacenes.

Representación:

- [0 0] en la casilla Pieza, indica que no hay pieza.
- [0 1] en la casilla Pieza, indica que hay pieza del tipo 1 (azul).
- [1 0] en la casilla Pieza, indica que hay pieza del tipo 2 (rojo).
- [0 0] en la casillas almacén, indica que no hay piezas en el almacén.
- [0 1] en la casillas almacén, indica que hay una pieza del tipo 2 en el almacén.
- [1 0] en la casillas almacén, indica que hay una pieza del tipo 1 en el almacén.
- [1 1] en la casillas almacén, indica que el almacén está lleno.

Siguiendo la lógica anterior hay que establecer variables de salidas en función a los movimientos que deseemos. (Ver tabla 1).

Representación:

- [0 0 0] en la casilla Salida, indica que el robot tendrá que moverse a la posición casa1.
- [0 0 1] en la casilla Salida, indica que el robot tendrá que moverse a la posición casa2.
- [0 1 0] en la casilla Salida, indica que el robot tendrá que moverse a la posición surtidor de pieza.
- [0 1 1] en la casilla Salida, indica que el robot tendrá que moverse a la posición pieza1.
- [1 0 0] en la casilla Salida, indica que el robot tendrá que moverse a la posición pieza2.

Para referenciar mejor las posiciones observe la figura1, ocupamos un modelo que se ajuste a este comportamiento y es aquí donde entran las redes neuronales.

Pieza		almacen 1	almacen 2	Segunda Salida		
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	1
0	1	0	0	0	1	1
0	1	0	1	0	1	1
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	0	0	1
1	0	1	0	1	0	0
1	0	1	1	0	0	1

Tabla 1.

5.2.2 Creación de la Red

Nntool es una herramienta que permite la creación y el entrenamiento de una RNA de manera fácil y rápida.

Procederemos a crear nuestra red ejecutando desde la consola el comando nntool (ver figura11).

En la casilla Input Data introduciremos los valores de entrada de la Tabla 1 (Pieza, almacen1, almacen2).

En la casilla Target Data introduciremos los valores de una columna de Salida de la Tabla 1, ya que solo se puede introducir una columna por red, será necesario crear tres redes neuronales.

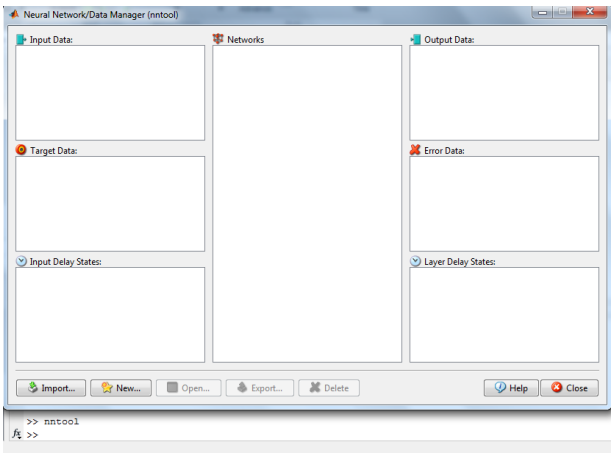

Figura 11.

Una vez introducido los datos, daremos clic al botón New para crear nuestra red. Podrás variar una serie de parámetros que harán que tu red se ajuste a tu modelo de entrenamiento (ver figura12), uno de los modelos de redes más potentes es el Backpropagation, pero, ya que nuestro sistema es binario el modelo que mejor se ajusta es el de perceptrón simple.

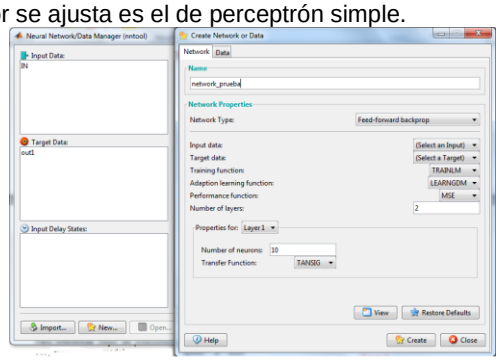

Figura 12.

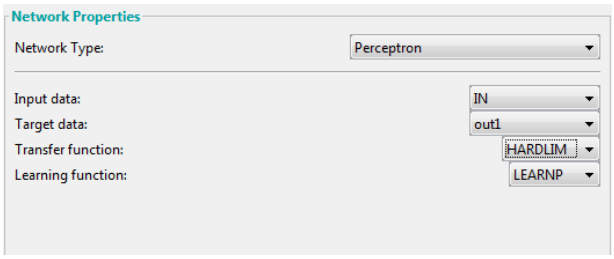

Figura 13.

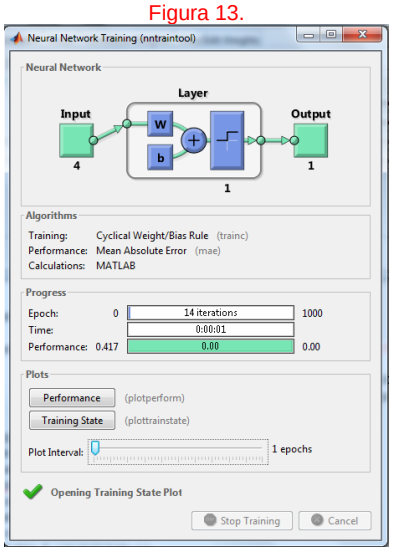

Figura 14.

Ajustamos la configuración tal y como se muestra en la figura13, esta configuración es la que mejor resultados nos dio.

Una vez creada, entrenaremos la red usando nntraintool la cual ajustara los pesos según las épocas y encontrara la que mejor se ajuste a los Target introducidos (ver figura14). Observe que la red creada solo es de una capa, posee cuatro entradas y una salida. Como resultado la Red retornara 2 valores de salida (Output data y Error data)

5.2.3 Resultado

Output data y Error data (ver figura15) estos dos valores de retornos son importantes, pues nos indican si la red logro ajustarse a los parámetros de entrenamiento. Observemos que en la pantalla de Error_data todos los valores aparecen en 0, esto indica que la red logro ajustarse en un 100% a los parámetros introducidos.

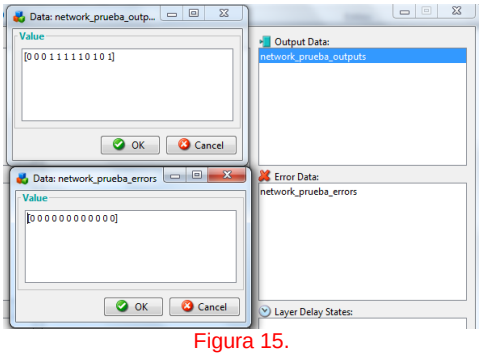

Figura 15.

Con los pasos anteriores logramos entrenar una red neuronal que se ajustó a nuestro modelo, para ejecutar esta red solo se necesita hacer el llamado, como si de una función nueva se tratase, el comando sim servirá para simular nuestra red.

```
z = sim(network1,[0;1;0;1])
y = sim(network2,[0;1;0;1])
x = sim(network3,[0;1;0;1])
```

Observemos que los valores de salida para estos datos de entrada son:

```
union =

    100
```

Mismos que figuran en la Tabla1, demostrando la funcionalidad de nuestra red neuronal.

5.3 EJECUCION DE MOVIMIENTOS

La Tercera parte de nuestro sistema consiste en el uso de la toolbox KUKA-KAWASAKI-Robotic desarrollada por el departamento de ingeniería en computación y automática de la Universidad de Ciencias Aplicadas. Wismar-Alemania [4] .Esta toolbox nos permite comunicar Matlab con el robot Kawasaki FS003 mediante el protocolo TCPIP o mediante el puerto serial, además, nos permite simular el robot en casi toda su totalidad dentro de Matlab.

Para el desarrollo y demostración de nuestra aplicación usaremos la simulación del robot, a continuación mostraremos algunos de los comandos o las instancias para la configuración del entorno:

El comando rtb desplegara la ayuda de la toolbox (ver figura16), para el uso de la toolbox de simulación del robot es necesario tener dos consolas de Matlab abiertas, una será para el robot simulado, y la otra será para los comandos (interprete) del manipulador (ver figura17).

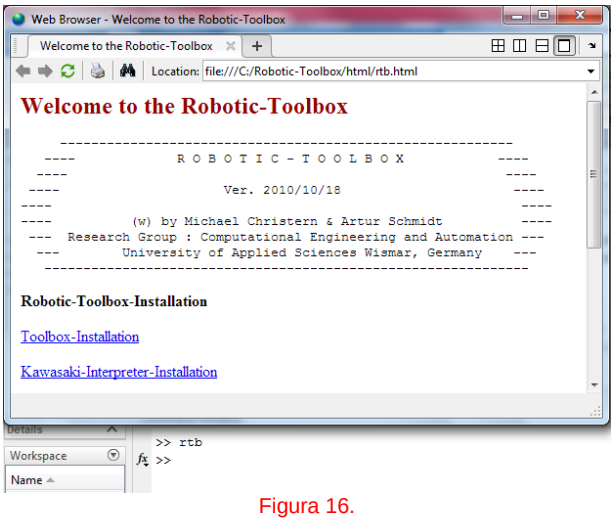

Figura 16.

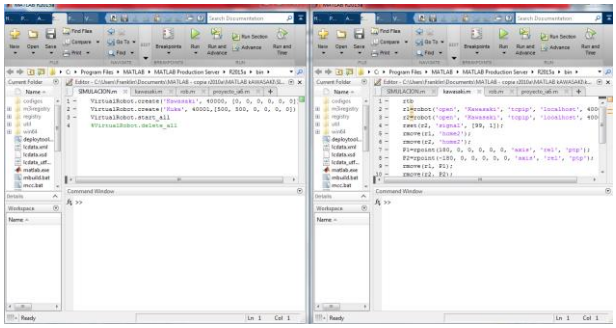

Figura 17.

step	Matlab instance **Visualization PC** with MatlabKK-Robotic-Visualization Tbx	Matlab instance **Control PC** with MatlabKK-Robotic Tbx
1	VirtualRobot.create('Kawasaki', 40000, [0, 0, 0, 0, 0, 0])	
2		r1=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
3	VirtualRobot.create('Kuka', 40001,[500, 500, 0, 0, 0, 0])	
4		r2=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40001)
5	VirtualRobot.start_all	

6		rset(r2, 'signal', [99, 1]);
		rmove(r1, 'home2');
		rmove(r2, 'home2');
		P1=rpoint(180, 0, 0, 0, 0, 0, 'axis', 'rel', 'ptp');
		P2=rpoint(-180, 0, 0, 0, 0, 0, 'axis', 'rel', 'ptp');
		rmove(r1, P1);
		rmove(r2, P2);

7		robot(r1, r2, 'close')
8	VirtualRobot.delete_all	

Tabla2.

Los pasos generales para la configuración del entorno se podrán observar en la Tabla2.

En la consola de visualización, el comando Virtualrobot.create crea un robot con los parámetros de entrada, VirtualRobot.star_all inicia la simulación del robot, VirtualRobot.delete_all detiene la simulación y elimina las variables de robot creadas.

En la consola de control el Comando robot sirve para crear y establecer la configuración de conexión de esta manera:

```
r1=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
```

Los parámetros de entrada 'open' sirve para abrir la comunicación, 'kawasaki' indica el tipo de robot con el que se espera conectar, 'tcpip' indica el protocolo por el cual se establecerá la comunicación, 'localhost' indica que será un robot simulada (en caso de ser un robot físico, este parámetro se sustituye por la dirección ip del robot).

```
rmove(r1, 'home2');
```

Este comando ejecuta un movimiento en el robot r1, en este caso el robot se moverá de su posición inicial hasta home2 (ver figura18).

```
casa1=rpoint(421, 282, 14, -84, 179, 92, 'pose', 'abs', 'lin','speed',20);
```

Otro comando utilizado es rpoint, con el definiremos posiciones y propiedades de movimiento del robot, los primeros 6 valores indican las posiciones (ya sea por transformadas o por juntas), los otros 5 valores indican propiedades del movimiento, 'pose' indica que los valores están por transformadas, 'abs' indica que el movimiento será absoluto al sistema de referencia, 'lin' indica la interpolación usada, en este caso sera lineal, 'speed' define la velocidad del movimiento, aquí mismo se pueden definir otras propiedades como ser la precisión accuracy o signal para activar o desactivar señales al alcanzar su posición.

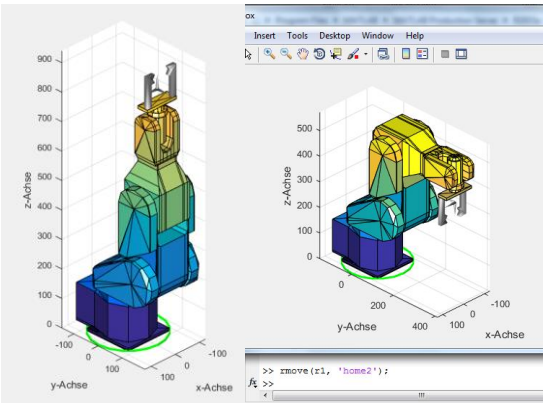

Figura 18.

En nuestro sistemas hemos creado 5 coordenadas por transformadas, estas coordenadas corresponden a las 5 posibles posiciones que puede tomar el robot (casa1, casa2, pieza, pieza1 y pieza2) (ver figura1).Para la creación de las coordenadas nos apoyamos de un software de simulación llamado PC-ROSET (distribuido por Kawasaki) [6], aunque, también era posible ejecutar el modo teach desde la toolbox de Matlab y luego guardar las posiciones con el comando rget.

```
casa1=rpoint(421, 282, 14, -84, 179, 92, 'pose', 'abs', 'lin','speed',20)
casa2=rpoint(-93, 481, 12, -86, 179, 92, 'pose', 'abs', 'lin','speed',20)
pieza1=rpoint(3, 488, -140, -90, 179, 87, 'pose', 'abs', 'lin','speed',20
pieza2=rpoint(-201, 474, -140, -84, 179, 94, 'pose', 'abs', 'lin','speed'
pieza=rpoint(421, 280, -140, -84, 179, 92, 'pose', 'abs', 'lin','speed',2
```

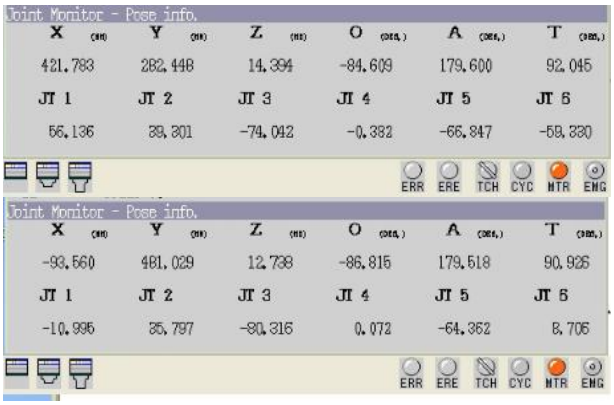

Figura 19.

6 SINTESIS

Ya hemos explicado las tres partes fundamentales del código por separado, ahora haremos una pequeña síntesis del funcionamiento en conjunto:

1. Se inicializa la comunicación con el robot.
2. Se inicializa la captura de video.
3. El robot se mueve a la posición casa2.
4. Se captura un fotograma, el robot se encuentra en la posición casa2, tomando una foto del almacén.
5. Se procesa el fotograma y se establece el estado del almacén.
6. El robot se mueve a la posición casa1.
7. Se captura un fotograma, el robot se encuentra en la posición casa1, tomando una foto del surtidor de pieza.
8. Se procesa el fotograma y se establece el estado del surtidor.
9. La red neuronal recibe los parámetros de estado del almacén y surtidor, en función a ellos decide qué movimiento ejecutara.
10. Se envían los movimientos al manipulador y este los ejecuta.
11. Al finalizar el movimiento vuelve a la posición casa2.
12. Se reinicia el Ciclo.

7 CONCLUSIONES

- El problema de paletizado se pudo solucionar exitosamente gracias a que la RNA pudo ajustarse al modelo de entrenamiento.
- Vemos como la aplicación de una RNA nos simplifico el trabajo, aclaramos que este problema se pudo haber solucionado con lógica digital, pero, una RNA dota de flexibilidad al sistema, agregar una variable más (por ejemplo otro almacén) solo requerirá de agregar sus condiciones al entrenamiento.
- Observamos como con una cámara puede dotar a un robot con la facultad de ver, los sistemas de visión en un futuro reemplazaran a muchos sensores, pero, de eso depende también la robustez de un sistema, por el momento nuestro sistema es robusto en condiciones óptimas de iluminación.

8 REFERENCIAS

[1] *Matlab*,(1994-2016).Recuperado de http://es.mathworks.com

[2] *Documentation Image processing toolbox*, (1994-2016). Recuperado de http://es.mathworks.com/help/images/index.html

[3] *Documentation neural network toolbox*, (1994-2016). Recuperado de http://es.mathworks.com/help/nnet/index.html

[4] M. Christern, A. Schmidt, T. Schwatinski, T. Pawletta, (2011). *KUKA-KAWASAKI-Robotic Toolbox for Matlab.* Recuperado de https://www.mb.hs-wismar.de/cea/KK_Robotic_Tbx/KK_Robotic_Tbx.html

[5] A. Bhargav Anand, 18 Sep 2010. *Tracking red color objects using Matlab*. Recuperado de http://www.mathworks.com/matlabcentral/fileexchange/28757-tracking-red-color-objects-using-matlab/content/redObjectTrack.m

[6] *Simulation Software PC-ROSET*, 2016.Recuperado de: http://kri-us.com/products/?page=otherPCROSET