

Práctica 2

1) Adquisición, sistema de visión

a) Primero hago algunos cálculos (y suposiciones) preliminares:

Supongo que cada botella tiene un radio de unos 5cm. Entonces la distancia entre los centros de las botellas sería 20cm. En ese caso, a una velocidad de 1m/s, pasarían “5 distancias” por segundo, y por lo tanto 6 botellas por segundo. Utilizando un criterio de 10x, se requerirían unas 60fps como mínimo, por lo que 124fps es más que suficiente.

Se deben detectar dos objetos: Tapa y nivel de líquido.

Para calcular el FOV, se deben tener en cuenta ambos por separado, pero también los dos en conjunto (ya que se usa la misma imagen para detectar los dos).

Vertical:

Tapa) $D_p = 20 \text{ mm}$ (medido de una tapa típica)
 $L_v = 0 \text{ mm}$ (se asume que las variaciones de la posición vertical serán irrelevantes)

Nivel de Líquido) $D_p = 2\text{mm}$ (se asume que con esa precisión se quiere medir el nivel, para determinar el estado de “llena” o “no llena”)
 $L_v = 20\text{mm}$ (por debajo de los 2mm ya se consideraría “no llena”, pero es útil buscar el nivel hasta 10 veces más abajo, para dar más oportunidades de encontrar un nivel)

Distancia Líquido-Tapa) A todo esto hay que agregarle la distancia entre el líquido de “botella llena”, y la tapa, debido a que ambas se van a detectar en la misma imagen. Asumo que eso agrega 20mm más (una tapa hacia abajo).

Total) Finalmente, se concluye que en realidad se quiere detectar un conjunto “tapa-líquido”, y por lo tanto se toma:

$D_p = 20\text{mm(tapa)} + 20\text{mm(distancia)} + 20\text{mm}(L_v \text{ líquido}) + 2\text{mm}(D_p \text{ líquido}) = 62\text{mm}$
 $L_v = 0 \text{ mm}$ (es la precisión vertical de la tapa; tomo como que “el objeto” tiene el punto superior bien determinado, y “crece” hacia abajo, hasta 62mm)

Con $P_a = 10\%$, entonces:

FOV(vertical) = 68,2 mm

Horizontal:

Tapa) $D_p = 30\text{mm}$ (medido de una tapa típica)
 $L_v = 20\text{mm}$ (asumo un 10% de error en los 200mm de separación entre botellas)
 $\text{FOV(tapa_horiz)} = (30\text{mm} + 20\text{mm}) * 1,1 = 55\text{mm}$

Nivel de líquido)

La botella tiene 10cm de diámetro, sin embargo no es necesario ver todo el diámetro para conocer el nivel del líquido. Tomo un 60% del diámetro.

$D_p = 60\text{mm}$

$$L_v = 20\text{mm} \text{ (nuevamente el 10% de 200mm; se podría hacer con menos, si se cambian las hipótesis pero no es necesario)}$$

$$\text{FOV(líq_horiz)} = (60\text{mm} + 20\text{mm}) * 1,1 = 88\text{mm}$$

Total) Tomo el máximo:

$$\text{FOV(horizontal)} = 88\text{mm}$$

La resolución de imagen es: 640 x 480 píxeles

Por lo tanto la resolución espacial es:

$$R_s(\text{vert}) = 68,2\text{mm} / 480\text{pixels} = 0,142\text{mm}$$

$$R_s(\text{horiz}) = 88\text{mm} / 640\text{pixels} = 0,138\text{mm}$$

Tomo la peor $R_s = 0,142\text{mm}$

El elemento que determina la resolución de “objeto” necesaria (R_f), es el nivel de líquido, que se quiere con una precisión de 2mm. Entonces $R_f = 2\text{mm}$ (se podría decir que “en la vertical”, pero puede estar un poco inclinada, de todas formas se tomó la peor R_s , que era la vertical).

Si se quiere que se tengan por lo menos 4 píxeles por “objeto”, como es habitual en casos de iluminación y contraste buenos (3 o 4 recomienda la literatura), se tendría que la cámara puede detectar con resolución $R_f = 4\text{pixels} * 0,142\text{mm/pixel} = 0,57\text{ mm}$, que es menor a los 2mm requeridos.

Tasa de procesamiento: $R_i_{\text{horiz}} * R_i_{\text{vert}} / T_i = 640 * 480 / (1/6) = 1.840.000 \text{ pixels/s}$

Esa cantidad es menor que 10.000.000 pixels/s, por lo que se estima que se podrá utilizar un PC común para realizar el procesamiento (según la literatura del curso).

Se asume que las botellas tienen una altura de 35cm, y no se quiere que golpeen el lente del sistema de visión si se caen. Por lo tanto la distancia hasta el lente se estima en 60cm. El sensor tiene un tamaño de 8,5mm (máximo horizontal) y se quiere un rango de visión de 88mm.

Por lo tanto:

$$M_i = 8,5\text{mm} / 88\text{mm} = 0,0966$$

$$F = 600\text{mm} * 0,0966 / (1+0,0966) = 52,8 \text{ mm}$$

Por lo tanto se puede utilizar (luego de ver algunos catálogos) un lente de 50mm de distancia focal. En ese caso:

$$D_o = 50\text{mm} * (1+0,0966) / 0,0966 = 567,6 \text{ mm}$$

$$D_i = 567,6\text{mm} * 0,0966 = 55\text{mm} \text{ (unos 5,5cm de distancia entre el sensor y el lente [un tamaño razonable para el equipo])}$$

Por último, se nota que con un tiempo de exposición de 3us, la línea se habría movido $1\text{m/s} * 3\text{us} = 3\text{um} = 0,003\text{mm}$. Eso es bastante menor que la resolución espacial, por lo tanto no afectaría la imagen. Podría ser conveniente utilizar un tiempo de exposición mayor incluso, para evitar tener que aumentar el nivel de iluminación innecesariamente.

En conclusión, sí es posible desarrollar el sistema con la cámara propuesta.

b) Si bien la cámara anterior se podría utilizar, se intentará buscar una que no exceda los requerimientos innecesariamente, y que además exista en el mercado. Varios de los cálculos de la parte "a" van a ser útiles.

Se quiere que la cámara sea capaz de tomar imágenes a, por lo menos 60 fps.

FOV(vertical) = 68,2mm

FOV(horizontal) = 88mm

Rf = 2mm (vertical; la horizontal no limita)

Se quiere Fp = 4 pixels, entonces Rs = 2mm / 4pixels = 0,5 mm/pixel

Por lo tanto Ri(vertical) = 68,2mm / (0,5mm/pixel) = 136,4 pixels

Se va a buscar una cámara color (porque puede ayudar a la detección del líquido), de área, con una resolución baja pero una tasa de adquisición no tan baja (>60fps). Filtrando esos requerimientos, se tomó la más barata que se pudo encontrar, cuyas características son como sigue:

Tipo de sensor: CCD

Resolución: 648 x 488

Tamaño del sensor: 1/3"

Frame rate: 84 fps

Color

Tiempo de exposición: 23us hasta 32s

Mecanismo de transferencia de datos: USB 3.0

Para determinar el tiempo de exposición, voy a tomar el criterio de que, en el mismo, la botella no se mueva más que un 10% de la resolución espacial (Rs).

Por lo tanto Dx = 0,05mm, v = 1m/s, entonces el tiempo de exposición sería:

$$t_e = 0,05\text{mm} / (1000\text{mm/s}) = 50\text{us}$$

Dado que el tamaño del sensor es el mismo que en la parte a, y los datos del problema también, la distancia de observación y la distancia focal del lente serán las mismas:

Distancia de observación = 56,8 cm

Distancia del lente al sensor = 55 mm

Distancia focal = 50 mm

c) Se propone la utilización de luz de fondo "desde atrás" (backlight), debido a la existencia de objetos transparentes y/o translúcidos (botella y líquido). La luz se colocará detrás de las botellas y apuntando hacia la cámara. Se propone utilizar luz de un color que realce el contraste entre el líquido y la botella transparente. Dado que en este caso la luz atravesará el líquido, lo que se busca es utilizar luz de un color parecido al mismo, de forma que esta sea reflejada hacia atrás y lo muestre opaco (se busca detectar la luz que atraviesa el líquido; si se buscara la luz reflejada y se quisiera un contraste con un fondo blanco, por ejemplo, esto sería al revés: se utilizaría un color complementario al del líquido).

En conclusión, la detección, tanto de la tapa como del líquido, se espera hacer intentando mostrarlos como zonas opacas, mientras que la parte de la botella sin líquido se mostraría con un brillo de una intensidad considerable. Se asume que no habrá una frontera "tapa-líquido". Si esa hipótesis no se cumpliera, y el líquido se estuviera mostrando indistinguible de la tapa, se podría variar el color hasta conseguir distinguirlos, o aumentar el brillo.

2) Detección de color de piel

a)

Para implementar un sistema de detección de piel se deben tener en cuenta dos aspectos:

- El espacio de colores a utilizar
- El modelo de distribución del “color piel” (clasificador)

Dependiendo de la elección de cada una de esas partes, será el detector que se obtenga. Por otro lado, los detectores pueden dividirse en “basados en píxeles” o “basados en regiones”. Los últimos logran mejoras respecto de los primeros, pero son más complejos. En lo que sigue se hace un resumen de los métodos basados en píxeles.

Espacios de colores

RGB) Es de los más utilizados. Se guarda la información del nivel de rojo, verde y azul. No es muy bueno debido a la alta correlación entre los canales, la “no uniformidad perceptiva” (se explica más abajo) y la mezcla de crominancia y luminancia.

RGB normalizado) Reduce la dimensión del espacio a 2. La transformación desde RGB es muy simple. Este espacio tiene la propiedad de ser invariante frente a reorientaciones de la fuente luminosa, para objetos opacos.

HSI, HSV, HSL) “Hue, Saturation, <Intensity/Value/Lightness>”. Describen el color con valores intuitivos: “tinte”, “saturación”, “tono”. Discrimina explícitamente entre crominancia y luminancia. Presenta varias propiedades de invariancia frente a cambios de iluminación y orientación de la misma pero también discontinuidades en el matiz de color (“hue”). No se adecúa bien a la detección por modelos paramétricos.

T, S, L) “Tint”, “Saturation”, “Lightness”. Se obtiene mediante una transformación del espacio RGB normalizado, pensada para que las componentes resultantes se aproximen al valor intuitivo de “matiz de color”, “saturación”, y “brillo”. Puede ser especialmente bueno para modelos de piel con distribución conjunta gaussiana unimodal.

YCrCb) Las dimensiones son la “luminancia” y las diferencias del nivel de color rojo y azul respecto a la luminancia. Separa explícitamente luminancia de crominancia.

Sistemas de color “perceptualmente uniformes”) Se basan en intentar que pequeñas desviaciones en los valores sean igualmente percibidas por una persona en todo el rango del espacio de colores. Ejemplos de estos sistemas son CEILAB, CIELUV. Requieren transformaciones no lineales del espacio RGB, que pueden ser bastante complejas.

Cocientes de canales RGB) Debido al alto componente de rojo en la piel, se ha utilizado el cociente R/G como característica (también hay estudios con R/B y G/B).

Otras varias transformaciones lineales del espacio RGB) YES, YUV, YIQ, CIE-xyz

Modelo de distribución del “color piel”

Luego de elegido el espacio de colores se procede a la clasificación de los píxeles mediante diferentes métodos:

- Definición explícita de una región: El clasificador es simple y rápido pero no es sencillo elegir las fronteras de forma de lograr una buena performance (en los últimos tiempos se han utilizado algoritmos de “machine learning” con buenos resultados).
- Modelos no paramétricos: Se intenta estimar la distribución del color de la piel sin un modelo explícito. Como ejemplos:
 - “Lookup table” normalizada
 - Clasificador Bayesiano
 - Mapa auto-organizado (SOM), con redes neuronales

Estos métodos son independientes de las distribución de los píxeles de piel, pero requieren mucha memoria y no son capaces de interpolar o generalizar el conjunto de entrenamiento.

- Modelos paramétricos:
 - Gaussiano simple

- Mezcla de gaussianas
- Clústers de gaussianas
- Modelo de frontera elíptica

Todos los métodos paramétricos operan en el plano de crominancia. Surge el problema de la validación del modelo teórico (comprobación directa de las hipótesis en los datos, de ser posible, además de las medidas de performance habituales).

- Modelos dinámicos: Son utilizados para el seguimiento de una cara específica en, por ejemplo, un video. Se ajustan a las características de esa cara en particular, y por lo tanto se obtienen mejores resultados que con modelos (estáticos) más generales.

Referencia: “A Survey on Pixel-Based Skin Color Detection Techniques” - V. Vezhnevets, V. Sazonov, A. Andreeva

b) Se utilizan, para esta parte, el script “analizarBD.m” y “ej2.m”.

El script “analizarBD.m” toma las imágenes de la base de datos de MUCT, selecciona las regiones que fueron manualmente marcadas como “cara” (en esa BD), y guarda tres vectores (correspondientes a los colores r, g, b) con valores de color para los puntos de las caras.

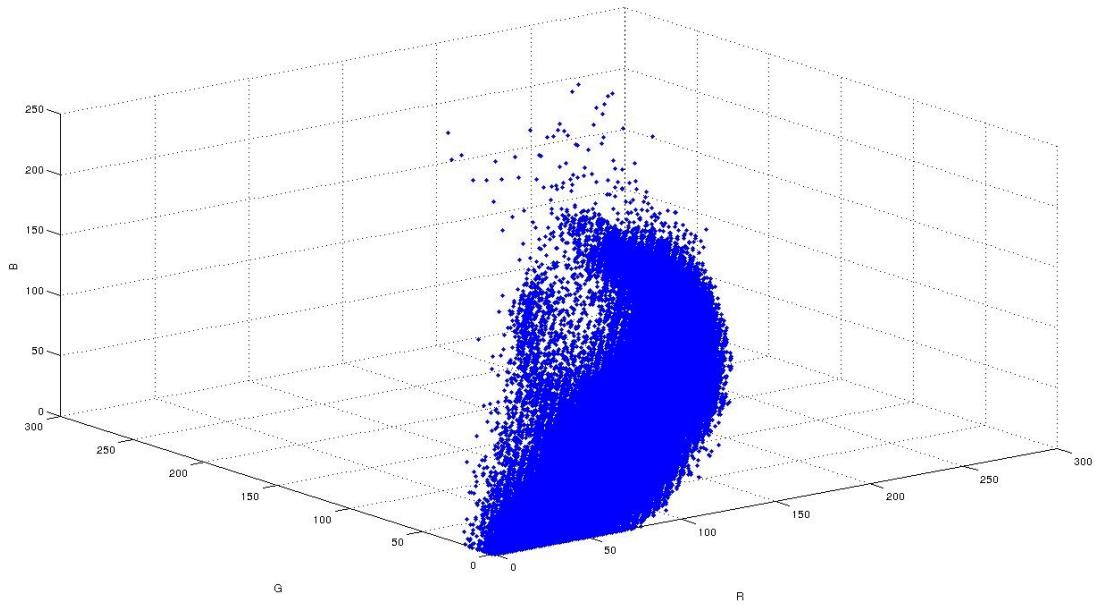
También muestra un ejemplo de la región de “cara” guardada en la BD para la primer imagen y realiza la gráfica en 3D de los puntos de color piel, en el espacio RGB.

Este script se debe ejecutar al menos una vez antes de ej2.m, si no está generado “pielFinal.mat”.

Cara según datos previos)



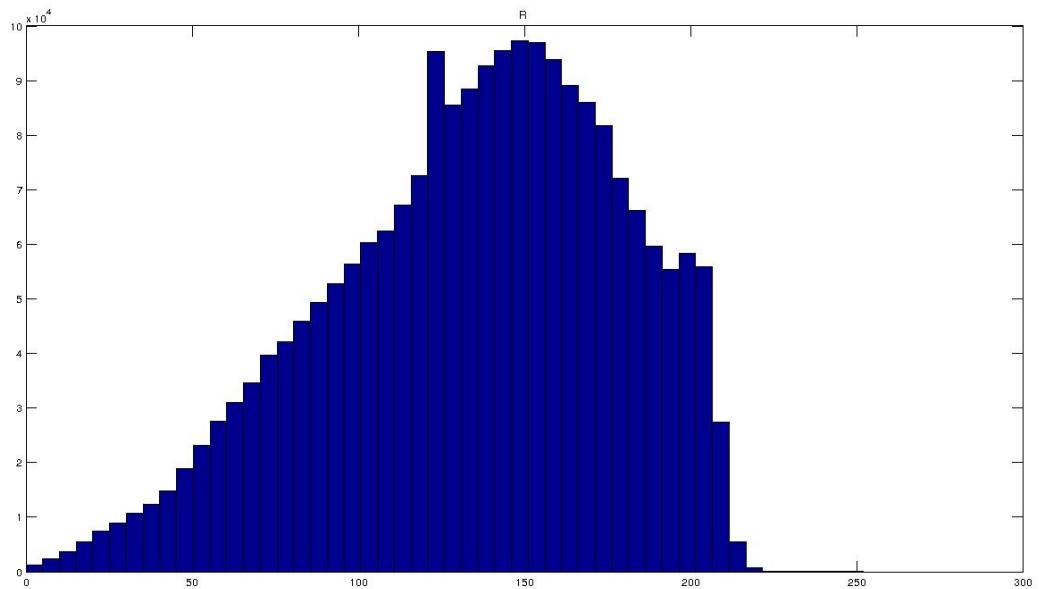
Gráfica del color piel en el espacio RGB)



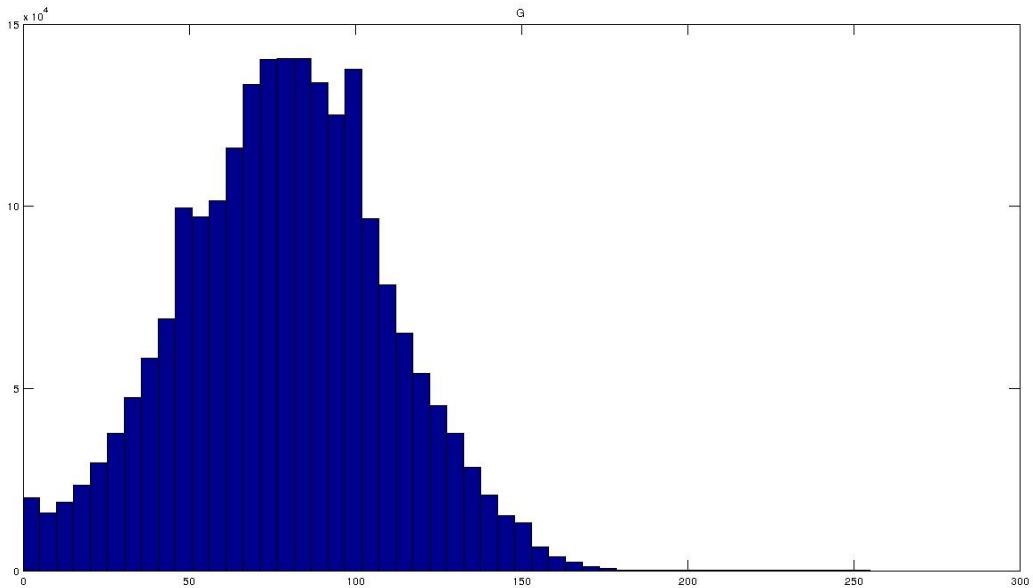
Luego el script “ej2.m” realiza los histogramas (en 1D), establece una región del espacio RGB como “color piel”, e intenta clasificar como “piel” ciertos píxeles de imágenes que no estaban en la BD de entrenamiento (conseguidas en www.freedigitalphotos.net).

Los histogramas pueden verse abajo.

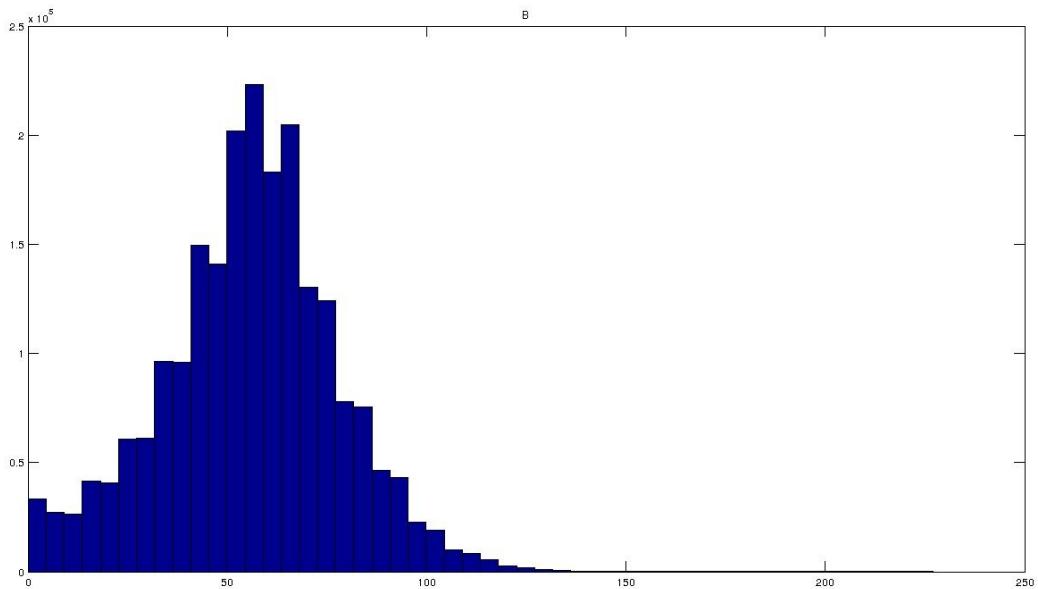
Histograma R)



Histograma G)



Histograma B)

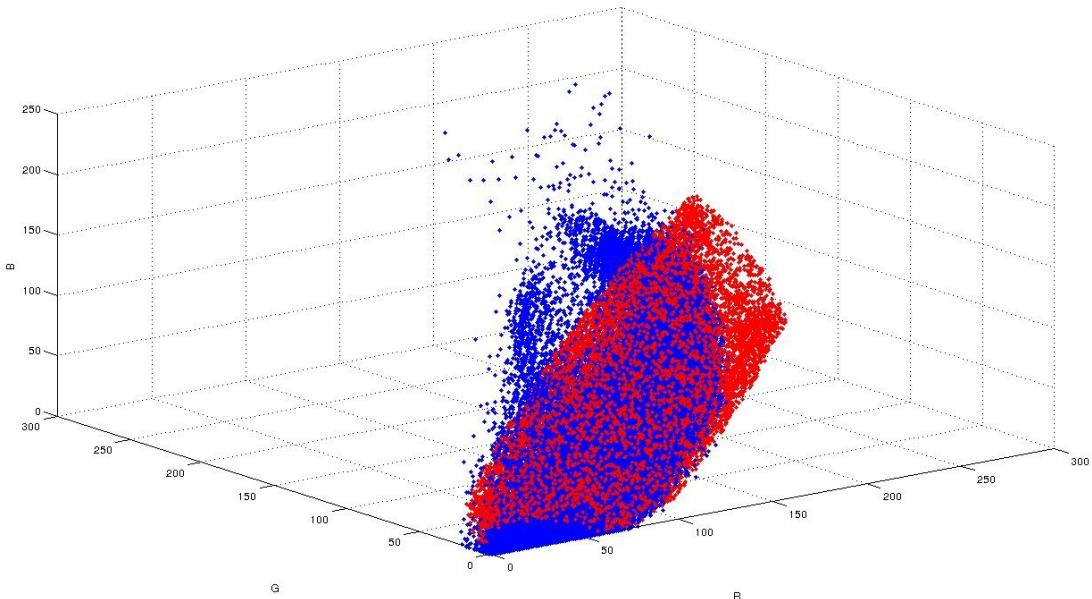


Se comprobó qué porcentaje de los datos entraban en una región que se utiliza típicamente en la literatura como “región fija” para “color piel”. El resultado fue que 81.6288% de los puntos tomados en la muestra de entrenamiento están en la región.

El criterio utilizado para definir la zona del espacio RGB del “color piel” fue el siguiente:

- Un píxel se clasifica como color piel si está a una distancia de Mahalanobis menor que un cierto umbral “L”.
- Para definir L se hace lo siguiente. Se define un porcentaje aceptable de “falsos negativos” en el conjunto de entrenamiento (es decir, píxeles que se clasificarían como “no piel” siendo en realidad “piel”). Luego se establece L para que la cantidad de “falsos negativos” al clasificar el conjunto de entrenamiento coincida con ese porcentaje.

Abajo puede verse la región de “color piel”, en puntos rojos, superpuesta a los puntos de entrenamiento, en azul.



(Notar que el corte “recto” se debe a que los valores van hasta 255)

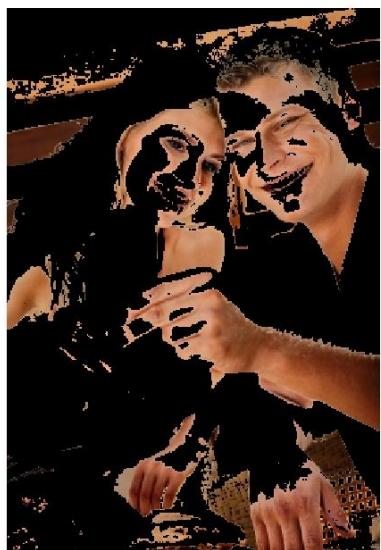
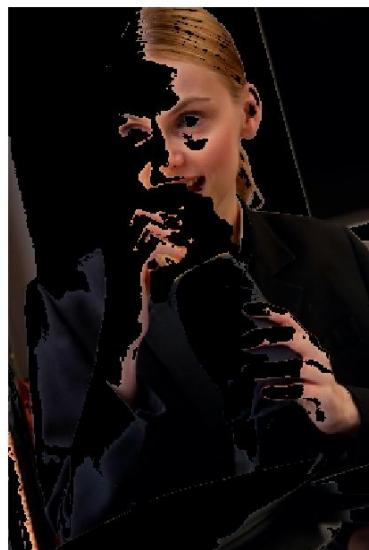
Los resultados al clasificar las imágenes de la base de datos de entrenamiento son bastante buenos (ajustando P (“falso negativo”) en cada caso, pero más o menos aceptable incluso con uno fijo). En el caso de la clasificación de imágenes que no estaban en el conjunto de entrenamiento (que es lo que se pedía en el ejercicio), el resultado es peor, y bastante variable además. Los dos principales problemas que se notan:

- Cuando las caras están muy iluminadas (más que las del conjunto de entrenamiento que tenía la iluminación “q”), las mismas se clasifican como “no piel”.
- El pelo castaño se confunde con piel

Los resultados se pueden ver abajo.









3) Restauración

a) Correr los scripts: ej3.m, tablaEj3.m

b) Las imágenes resultantes de aplicar el ruido y los filtros pueden verse abajo. En las columnas se aplican distintos tipos de ruido. La primer fila tiene la imagen con ruido sin filtrar. Las siguientes filas tienen las imágenes ruidosas filtradas con distintos filtros.

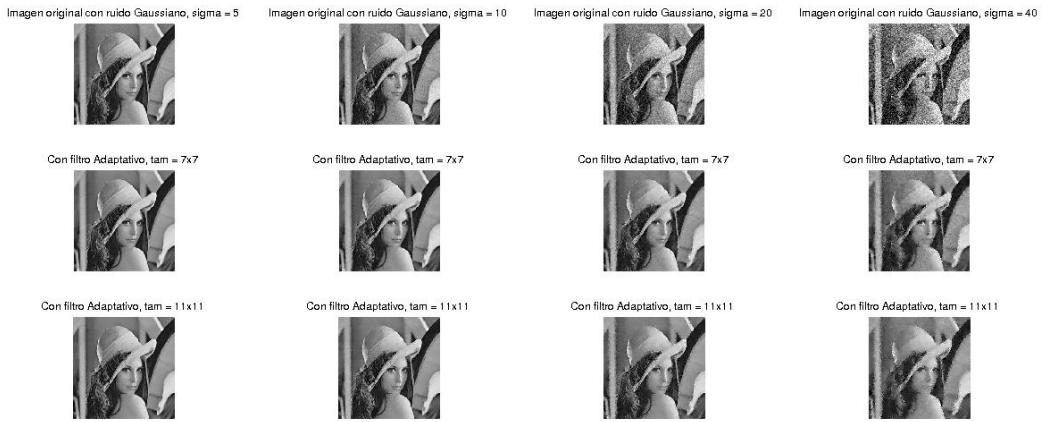
Ruido Gaussiano, filtro de Media)



Ruido Gaussiano, filtro de Mediana)



Ruido Gaussiano, filtro Adaptativo)



Ruido gaussiano, filtro NLM

Imagen original con ruido Gaussiano, sigma = 5 Imagen original con ruido Gaussiano, sigma = 10 Imagen original con ruido Gaussiano, sigma = 20 Imagen original con ruido Gaussiano, sigma = 40



Con filtro NLM



Con filtro NLM



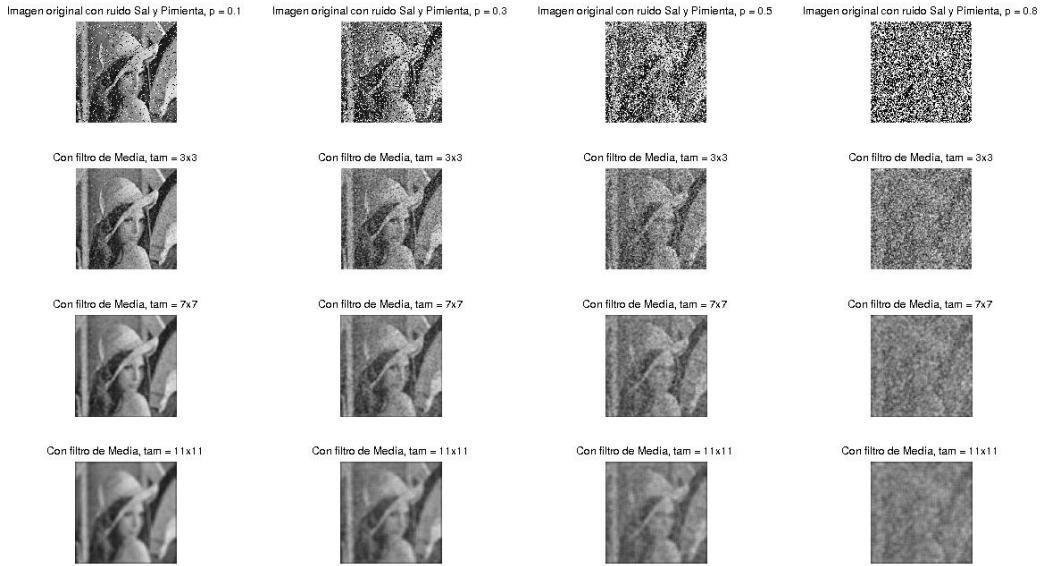
Con filtro NLM



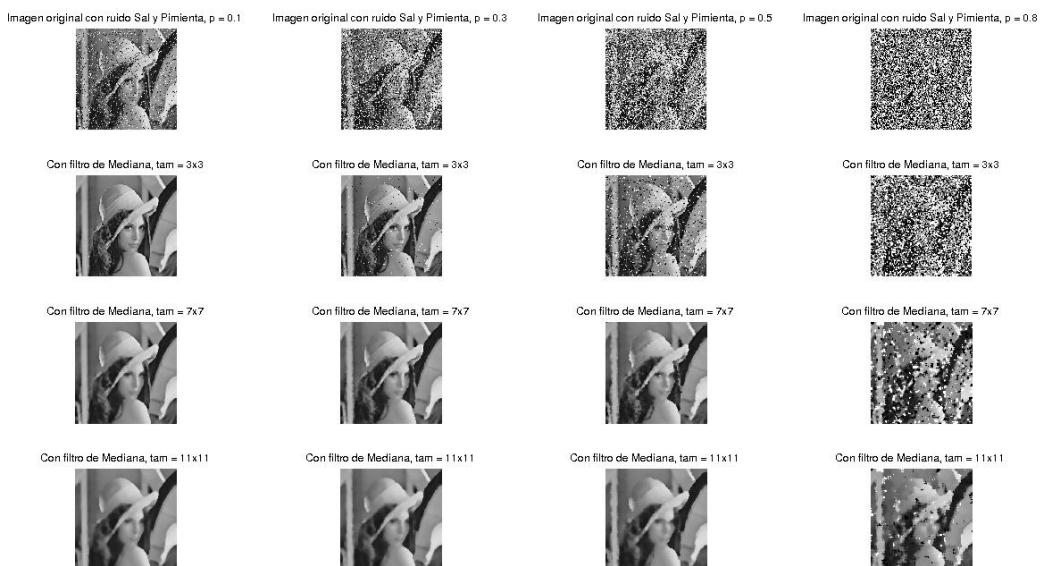
Con filtro NLM



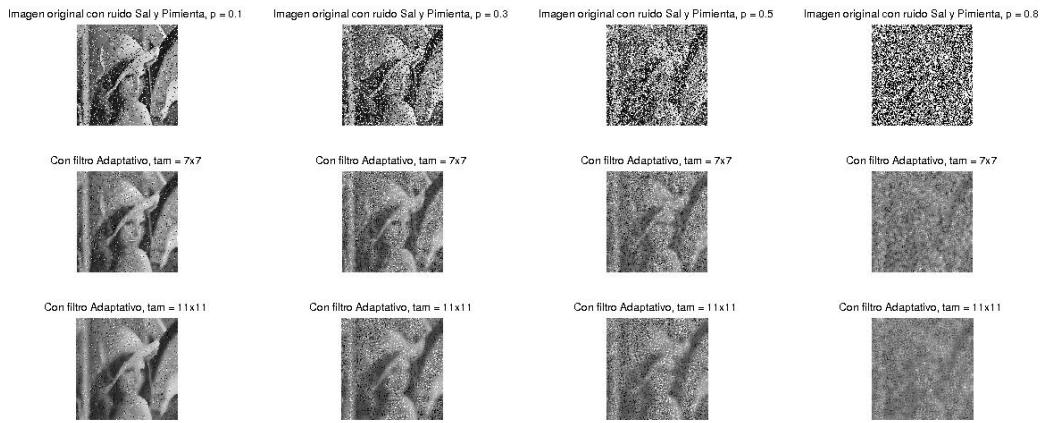
Ruido Sal y Pimienta, filtro de Media)



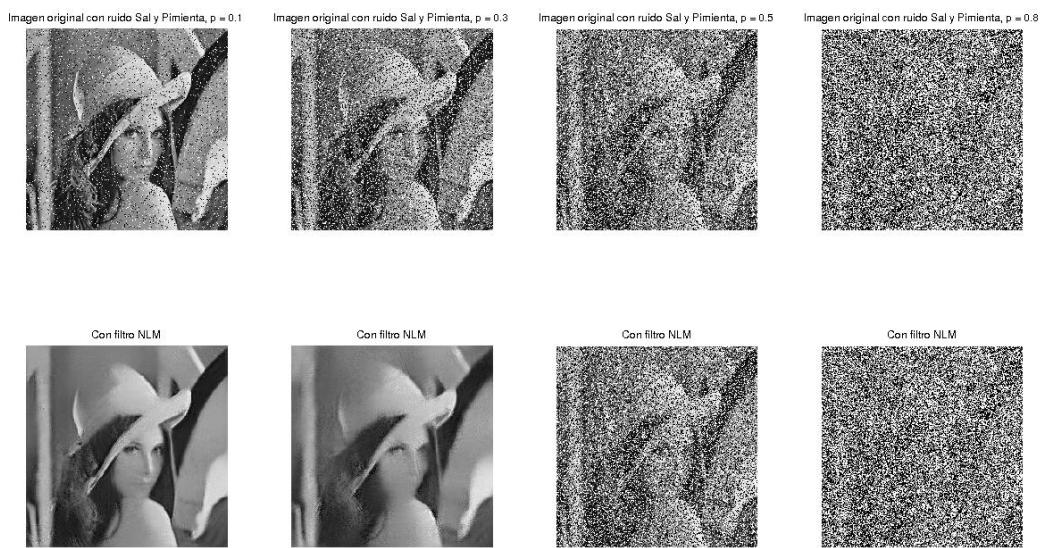
Ruido Sal y Pimienta, filtro de Mediana)



Ruido Sal y Pimienta, filtro Adaptativo)



Ruido Sal y Pimienta, filtro NLM)



c) Para el caso de ruido gaussiano los filtros de media o de mediana parecen tener una performance parecida. Los tamaños muy grandes de la base del filtro tienden a “borronear” la imagen. El filtro adaptativo y NLM parecen dar mejores resultados (sobretodo este último).

Para el caso de ruido Sal y Pimienta se ve claramente que los filtros de media y adaptativo son muy malos. Se “borronea” la imagen incluso para valores chicos de p. El filtro de mediana es el que consigue los mejores resultados.

El filtro NLM, consigue resultados aceptables para $p = 0.1, 0.3$, aunque peores que el filtro de mediana. Los casos $p = 0.5, 0.8$ no se probaron con NLM debido a que el algoritmo implementado no acepta $\sigma > 100$ (en todos los casos que fuera necesario [Adaptativo y NLM] se calculó la desviación estándar según la fórmula teórica para el ruido S y P, en función de p).

d)

PSNR

	Ninguno	FMedia3	FMedia7	FMedia11	FMediana3	FMediana7	FMediana11	FAdapt7	FAdapt11	FNLM
RGauss5	42.5843	41.4444	40.0404	39.5037	42.3416	40.7708	40.1333	43.5247	43.2425	44.2225
RGauss10	40.0027	41.1176	40.0032	39.4862	41.3617	40.5219	39.9498	41.4326	41.0879	42.2291
RGauss20	38.6902	40.3299	39.9519	39.4846	40.0281	40.1445	39.7132	40.2224	39.9979	41.0002
RGauss40	38.0842	39.0934	39.6548	39.4057	38.8254	39.4888	39.3706	39.3171	39.349	39.9107
RSyP0.1	42.6486	39.145	39.2515	39.0702	42.4983	40.8219	40.1888	38.4097	38.4838	39.5996
RSyP0.3	40.1856	38.1886	38.4172	38.4517	41.4787	40.5999	40.0761	37.8819	37.9274	38.9214
RSyP0.5	39.0702	37.9042	38.1111	38.1807	40.341	40.4081	39.9856	37.7206	37.7245	39.0702
RSyP0.8	38.0487	37.7162	37.8811	37.9555	38.5135	39.2644	39.4532	37.6884	37.6907	38.0487

RMSE

	Ninguno	FMedia3	FMedia7	FMedia11	FMediana3	FMediana7	FMediana11	FAdapt7	FAdapt11	FNLM
RGauss5	3.5583	4.6262	6.3919	7.2326	3.7627	5.4024	6.2567	2.8655	3.0579	2.4402
RGauss10	6.4476	4.9878	6.4469	7.2619	4.7151	5.7211	6.5266	4.6388	5.022	3.8615
RGauss20	8.7226	5.9796	6.5235	7.2646	6.41	6.2404	6.8921	6.1296	6.4548	5.1244
RGauss40	10.0287	7.9493	6.9854	7.3977	8.4552	7.2574	7.4577	7.5502	7.4949	6.5856
RSyP0.1	3.506	7.8554	7.665	7.9919	3.6294	5.3392	6.1772	9.3046	9.1472	7.0746
RSyP0.3	6.1817	9.7906	9.2884	9.2149	4.5899	5.6192	6.3396	10.507	10.3975	8.2704
RSyP0.5	7.9919	10.4531	9.9669	9.8085	5.9644	5.873	6.473	10.9046	10.8947	7.9919
RSyP0.8	10.111	10.9156	10.509	10.3303	9.0849	7.6423	7.3172	10.9856	10.98	10.111

-- |

e) Con los datos cuantitativos, se puede confirmar lo anterior y además agregar algunos detalles. Mirando la tabla de RMSE, se puede ver que los filtros de Media y Mediana con base más grande (11x11) en general empeoran la imagen, cuando el ruido es chico, y son mejores cuando el ruido es grande.

En el caso de ruido gaussiano, se ve que el filtro de media y el de mediana son similares, y que cuanto mayor el ruido, conviene utilizar una base más grande. El filtro adaptativo funciona mejor que cualquiera de los anteriores y no tiene tantas variaciones al variar el nivel de ruido y/o la base del filtro. El filtro de “Non Local Means” está dando la mejor performance para todos los casos de ruido gaussiano.

En el caso de ruido Sal y Pimienta, es muy claro que el filtro que tiene mejor desempeño es el de Mediana. Para un nivel de ruido bajo, la base de 3x3 es la que da mejor resultado, pero para ruidos medios y altos las bases de 7x7 y 11x11 pasan a ser las mejores. El filtrado con NLM da resultados bastante peores para $p= 0.1, 0.3$, aunque sin embargo son mejores que los del filtro de media o el adaptivo (que da los peores resultados debido a que asume explícitamente un ruido aditivo). No se pudo probar el filtro NLM para $p= 0.5, 0.8$ debido a que la desviación estándar de ruido en esos casos es mayor que 100, y el algoritmo utilizado no trabaja con esos valores (se copió la imagen ruidosa en esos casos).

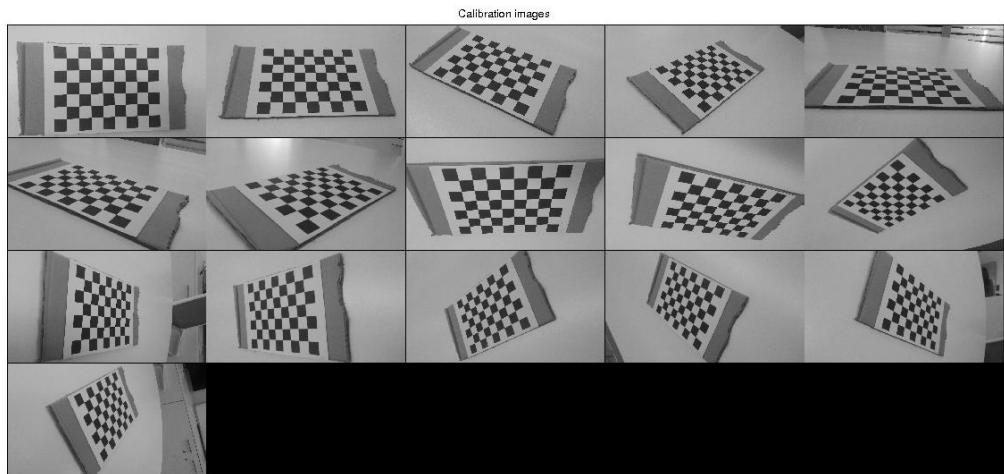
4) Cámaras)

Se tomaron 16 imágenes de un patrón de “tablero de damas”.

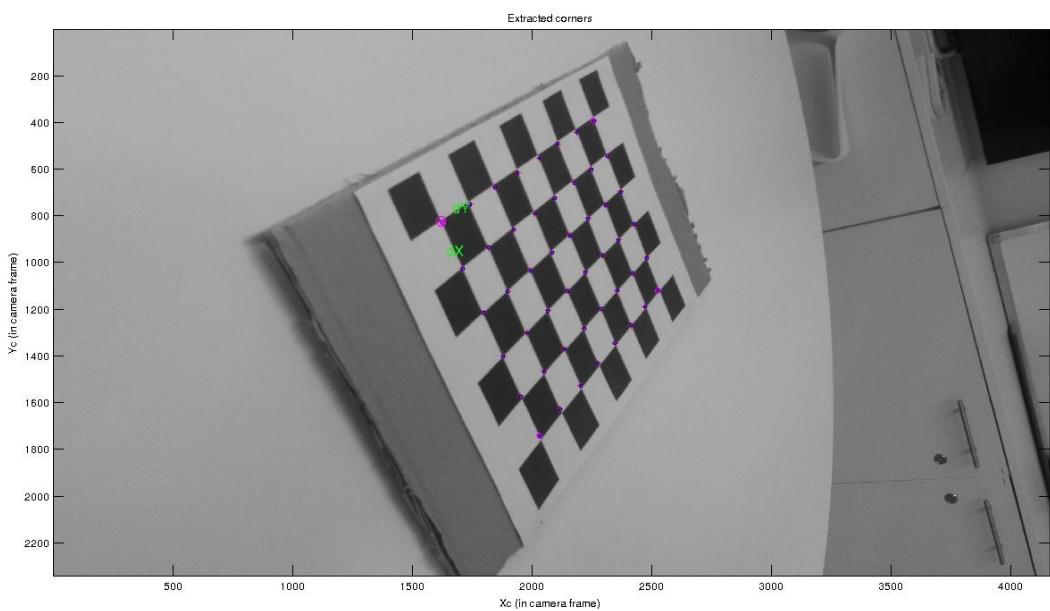
Primero se utilizó la herramienta de calibración de Caltech para Matlab.

El procedimiento se describe abajo, copiando los comandos utilizados y algunas imágenes y resultados:

- Image names



-Extract grid corners



- Calibration

Primera Calibración:

Calibration results after optimization (with uncertainties):

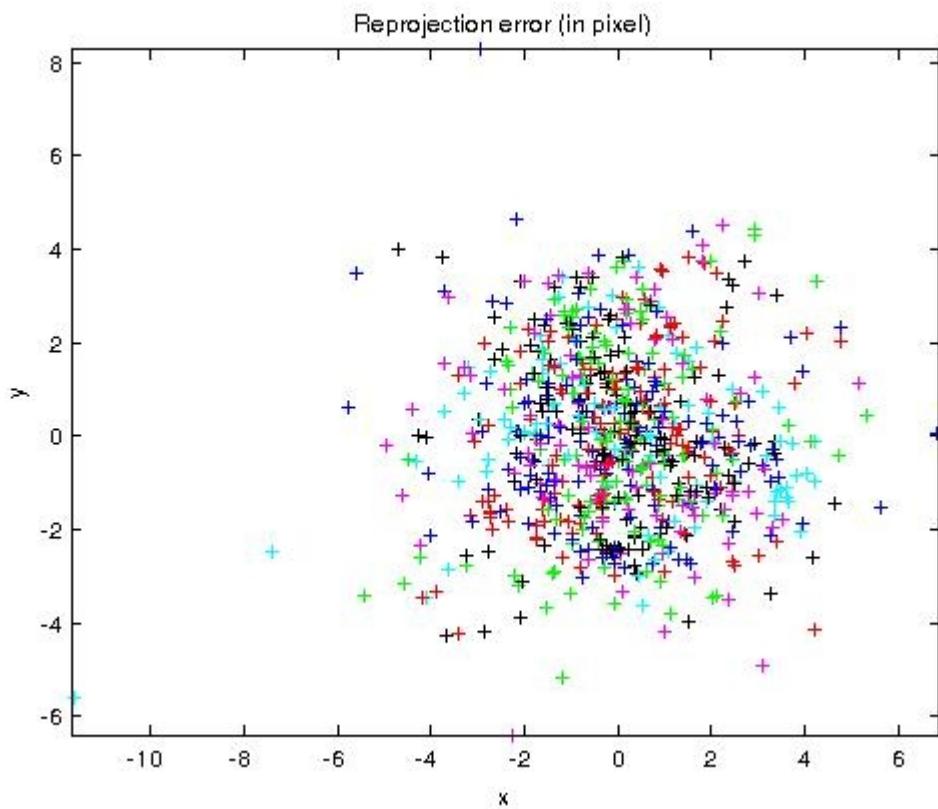
Focal Length: $fc = [3562.64630 \ 3538.73059] +/- [21.25863 \ 22.34652]$
Principal point: $cc = [2027.50333 \ 1068.22856] +/- [47.67460 \ 32.11571]$
Skew: $\alpha_c = [0.00000] +/- [0.00000] \Rightarrow \text{angle of pixel axes} = 90.00000 +/- 0.00000 \text{ degrees}$
Distortion: $kc = [0.00932 \ 0.22582 \ -0.00435 \ -0.00927 \ 0.00000] +/- [0.04228 \ 0.28622 \ 0.00360 \ 0.00542 \ 0.00000]$
Pixel error: $err = [1.95676 \ 1.87292]$

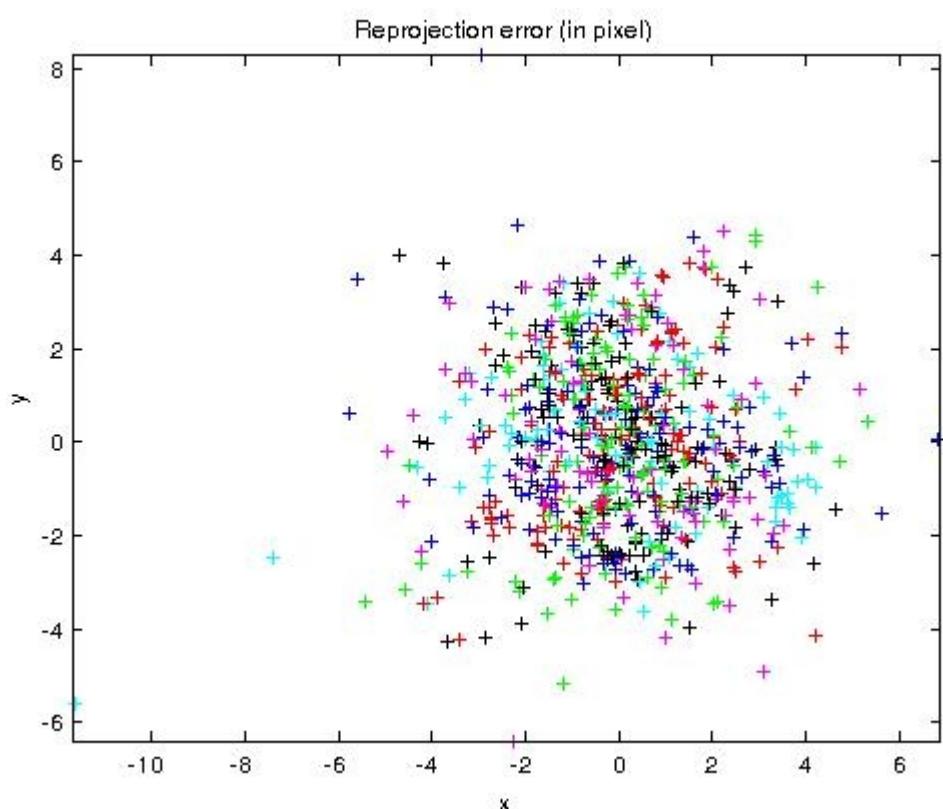
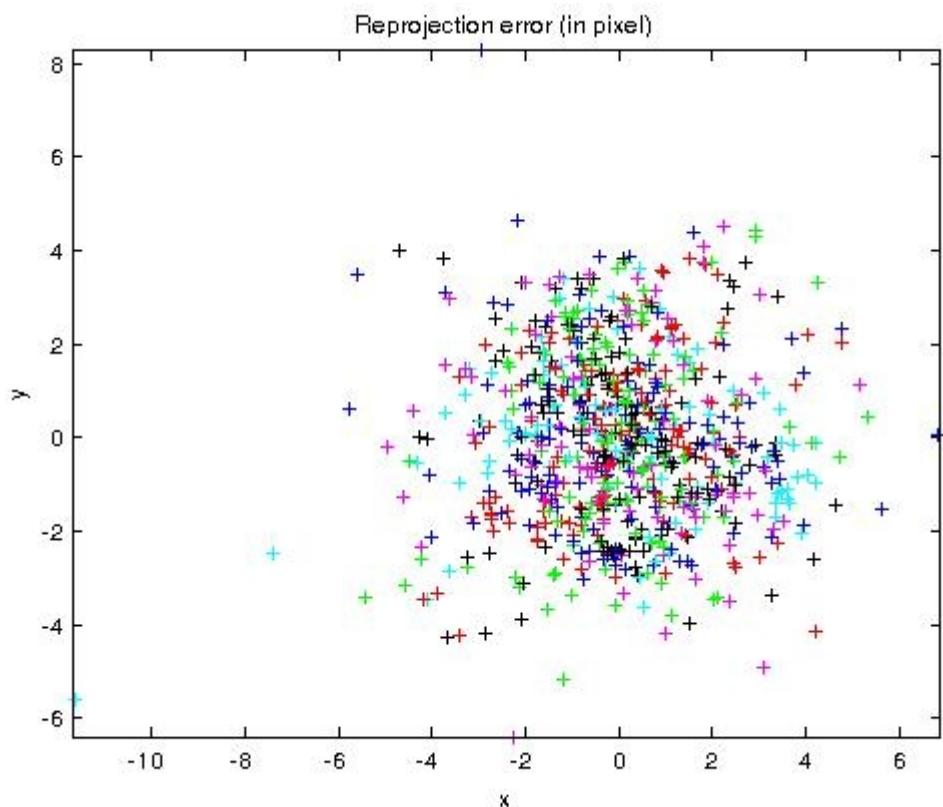
Note: The numerical errors are approximately three times the standard deviations (for reference).

Recommendation: Some distortion coefficients are found equal to zero (within their uncertainties).

To reject them from the optimization set $est_dist=[0;1;1;0]$ and run Calibration

- Reproject on Images (se muestra la gráfica del error)





- Show extrinsic (camera view y world-centered view; se muestran para otra calibración)
- Recompute corners (se recalculan las esquinas haciendo uso de los parámetros obtenidos)
- Calibration

Calibración después de recompute corners:

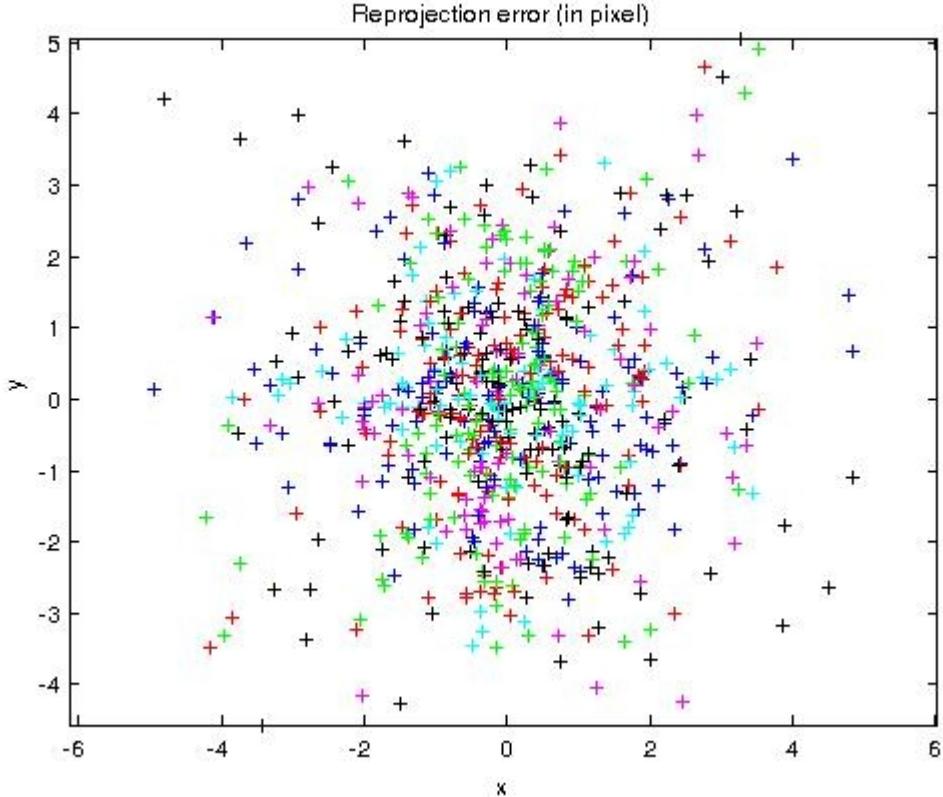
Calibration results after optimization (with uncertainties):

Focal Length: $fc = [3587.62999 \ 3564.24409] +/- [17.17871 \ 18.23374]$
Principal point: $cc = [2077.71140 \ 1073.36839] +/- [39.19302 \ 26.94416]$
Skew: $\alpha_c = [0.00000] +/- [0.00000] \Rightarrow \text{angle of pixel axes} = 90.00000 +/- 0.00000 \text{ degrees}$
Distortion: $kc = [0.05143 \ 0.12209 \ -0.00663 \ -0.00251 \ 0.00000] +/- [0.03487 \ 0.24368 \ 0.00320 \ 0.00473 \ 0.00000]$
Pixel error: $err = [1.51191 \ 1.58267]$

Note: The numerical errors are approximately three times the standard deviations (for reference).

- Save

- Reproject on Images (ver cambios en el error)



-Show extrinsic (ahora sí se muestran):

- Recompute corners; cambiando wx,wy a 9 para todas las imágenes (se podrían haber seleccionado las que tengan más error arriba...).

- Calibration

Calibration results after optimization (with uncertainties):

Focal Length: $fc = [3593.48874 \ 3590.47570] +/- [19.65611 \ 21.09791]$
Principal point: $cc = [2101.04539 \ 1085.50129] +/- [44.25466 \ 31.11432]$
Skew: $\alpha_c = [0.00000] +/- [0.00000] \Rightarrow \text{angle of pixel axes} = 90.00000 +/- 0.00000 \text{ degrees}$
Distortion: $kc = [0.12696 \ -0.19636 \ -0.00945 \ 0.00127 \ 0.00000] +/- [0.03923 \ 0.27627 \ 0.00396 \ 0.00568 \ 0.00000]$
Pixel error: $err = [1.70037 \ 1.82115]$

Note: The numerical errors are approximately three times the standard deviations (for reference).

El error fue peor, así que se cargan los valores viejos y se va a intentar cambiar las ventanas de los vértices con más error, sólamente. Para eso se utiliza "Analyze error". Se encuentra que hay puntos con mucho error en las imágenes: 10, 15, 14, 7. Se va a tomar una ventana de 8 para recalcular las

esquinas en esas imágenes.

- Recompute corners; cambiando wx,wy a 8 para las imágenes 10, 15, 14, 7.
- Calibration

Calibration results after optimization (with uncertainties):

```
Focal Length:    fc = [ 3582.61156 3561.16900 ] +/- [ 17.80970 18.95918 ]
Principal point: cc = [ 2095.12390 1078.04659 ] +/- [ 40.79508 28.19693 ]
Skew:           alpha_c = [ 0.00000 ] +/- [ 0.00000 ] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:     kc = [ 0.07322 0.05118 -0.00770 0.00012 0.00000 ] +/- [ 0.03570 0.24831 0.00344 0.00506 0.00000 ]
Pixel error:    err = [ 1.55842 1.66627 ]
```

Note: The numerical errors are approximately three times the standard deviations (for reference).

Recommendation: Some distortion coefficients are found equal to zero (within their uncertainties).
To reject them from the optimization set est_dist=[1;0;1;1;0] and run Calibration

Se puede ver que el error no mejoró, y tampoco lo hace al tomar est_dist = [1;0;1;1;0].

- Reproject on Images (no mejora el resultado)
- Extract grid corners (se va a repetir el proceso para las imágenes 10, 15, 14 y 7)

(se utilizó un valor “adivinado” para k_c de 0,5 en la imagen 10)

Luego de la nueva extracción, y calibrar el resultado es el siguiente:

Calibration results after optimization (with uncertainties):

```
Focal Length:    fc = [ 3582.89972 3563.11268 ] +/- [ 15.89499 17.07306 ]
Principal point: cc = [ 2070.80450 1080.44207 ] +/- [ 38.10813 26.05703 ]
Skew:           alpha_c = [ 0.00000 ] +/- [ 0.00000 ] => angle of pixel axes = 90.00000 +/- 0.00000 degrees
Distortion:     kc = [ 0.07847 0.00000 -0.00774 -0.00338 0.00000 ] +/- [ 0.01150 0.00000 0.00318 0.00469 0.00000 ]
Pixel error:    err = [ 1.42908 1.57469 ]
```

Note: The numerical errors are approximately three times the standard deviations (for reference).

Se ve que el resultado mejoró un poco.

Ahora voy a calibrar relajando la hipótesis de “ángulos rectos”, y con factor de distorsión hasta orden 6 (como se hace en el tutorial de Caltech).

```
est_alpha = 1
est_dist = [1;1;1;1;1]
```

- Calibration

Calibration results after optimization (with uncertainties):

```
Focal Length:    fc = [ 3581.55243 3561.83143 ] +/- [ 17.02623 18.03862 ]
Principal point: cc = [ 2087.06645 1087.04231 ] +/- [ 40.19584 26.25912 ]
Skew:           alpha_c = [ -0.00255 ] +/- [ 0.00186 ] => angle of pixel axes = 90.14628 +/- 0.10685 degrees
Distortion:     kc = [ 0.09876 -0.27201 -0.00723 -0.00115 0.87570 ] +/- [ 0.07241 1.11394 0.00319 0.00496 4.96715 ]
Pixel error:    err = [ 1.41964 1.56681 ]
```

Note: The numerical errors are approximately three times the standard deviations (for reference).

Recommendation: Some distortion coefficients are found equal to zero (within their uncertainties).
To reject them from the optimization set est_dist=[1;0;1;1;0] and run Calibration

Se ve que el resultado no mejora y, por otro lado, los coeficientes de distorsión 2 y 5 tienen incertidumbres más grandes que su valor (como dice la sugerencia). Se cambia est_dist = [1;0;1;1;0]. También se ve que alpha_c es muy cercano a 90°.

El resultado no mejora, por lo que se vuelve a tomar α_c como 90° ($\text{est_alpha}=0$). Finalmente, el resultado es:

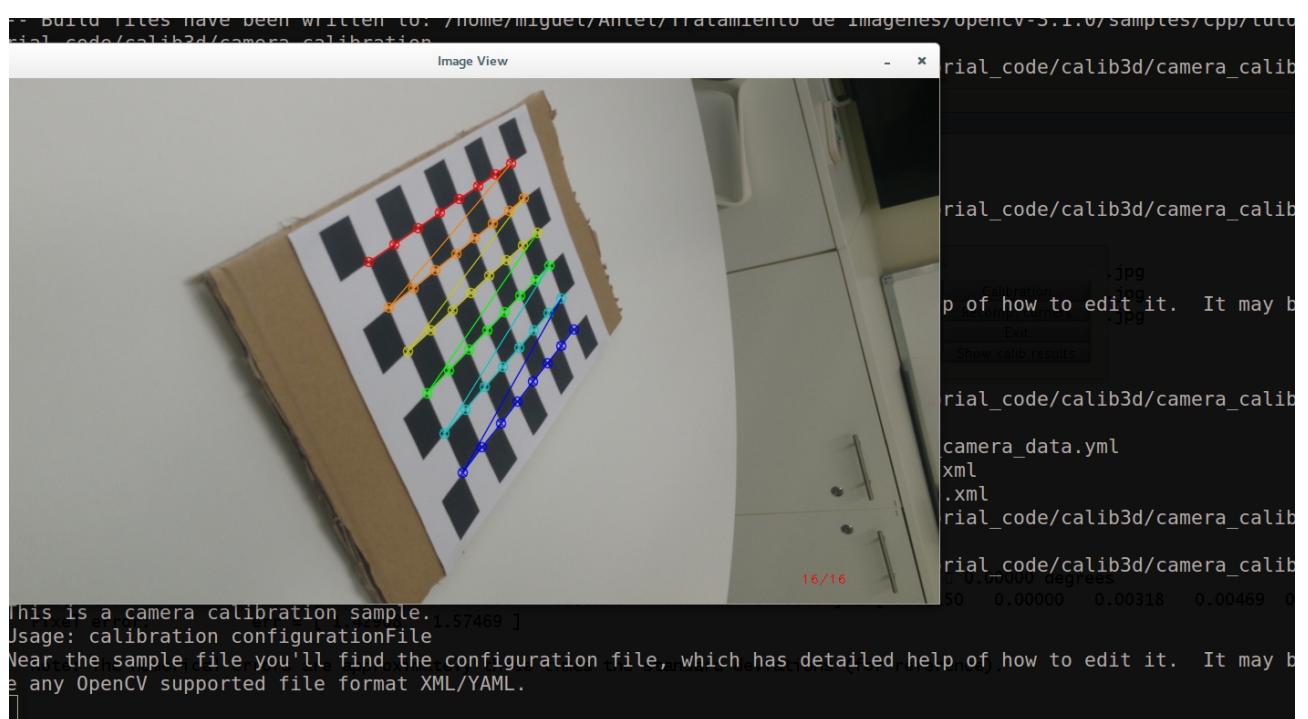
Calibration results after optimization (with uncertainties):

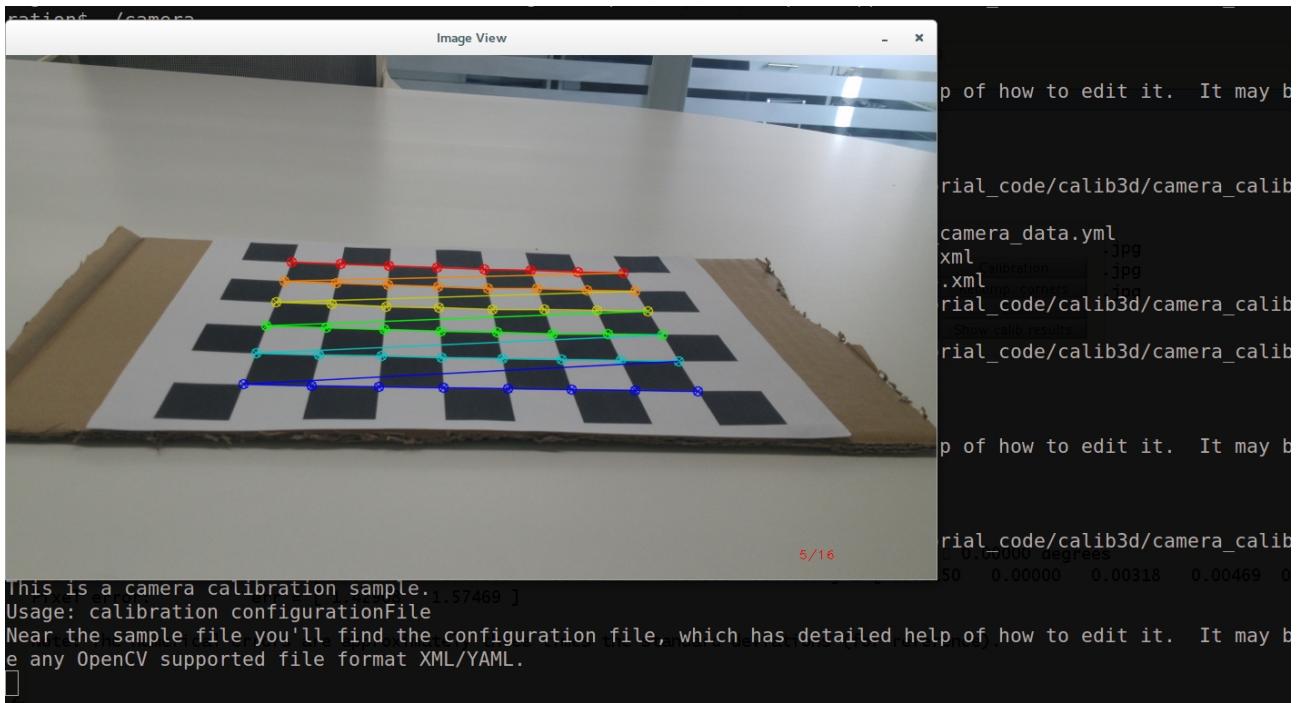
Focal Length: $\text{fc} = [3582.89972 \ 3563.11268] \pm [15.89499 \ 17.07306]$
 Principal point: $\text{cc} = [2070.80450 \ 1080.44207] \pm [38.10813 \ 26.05703]$
 Skew: $\text{alpha_c} = [0.00000] \pm [0.00000] \Rightarrow \text{angle of pixel axes} = 90.00000 \pm 0.00000 \text{ degrees}$
 Distortion: $\text{kc} = [0.07847 \ -0.00000 \ -0.00774 \ -0.00338 \ 0.00000] \pm [0.01150 \ 0.00000 \ 0.00318 \ 0.00469 \ 0.00000]$
 Pixel error: $\text{err} = [1.42908 \ 1.57469]$

Note: The numerical errors are approximately three times the standard deviations (for reference).

No se realizan más pruebas con este toolbox. Se pasará a utilizar el de OpenCV.

b) Debido a que las imágenes originales tenían un tamaño muy grande (4160x2340 píxeles y aprox. 2MB), y el algoritmo no terminaba nunca de procesarlas, se escalaron (proporcionalmente) con GIMP, hasta 1024x576 píxeles, lo que dio un tamaño de aproximadamente 160kB. Con esos cambios, se pudo calibrar sin problemas. Abajo se pueden ver dos imágenes del proceso de calibración:





Abajo se puede ver la parte más relevante del resultado (parte del archivo “out.xml”):

```

-<camera_matrix type_id="opencv-matrix">
<rows>3</rows>
<cols>3</cols>
<dt>d</dt>
-<data>
8.8210771682033499e+02 0. 5.115000000000000e+02 0. 8.8210771682033499e+02
2.875000000000000e+02 0. 0. 1.
</data>
</camera_matrix>
-<distortion_coefficients type_id="opencv-matrix">
<rows>5</rows>
<cols>1</cols>
<dt>d</dt>
-<data>
2.1625654147208581e-01 -1.2091019005178165e+00 0. 0. 3.0867664710135747e+00
</data>
</distortion_coefficients>
<avg_reprojection_error>4.5925440203439771e-01</avg_reprojection_error>

```

Estos datos se interpretan así:

$$M_{cámara} = \begin{bmatrix} 882,1 & 0 & 511,5 \\ 0 & 882,1 & 287,5 \\ 0 & 0 & 1 \end{bmatrix}$$

Por lo tanto:

$$f_x = f_y = 882,1$$

$$c_x = 511,5$$

$$c_y = 287,5$$

$$\text{Distorsión} = [0,216 \quad -1,21 \quad 0 \quad 0 \quad 3,09]$$

Por lo tanto: $k_1 = 0,216$; $k_2 = -1,21$; $p_1 = 0$; $p_2 = 0$; $k_3 = 3,09$

Se nota, como curiosidad, que no hay distorsión tangencial que el algoritmo haya detectado.

c) Debido a que se habían utilizado imágenes sin modificar en la parte a, los resultados son diferentes (en “píxeles”). Sin embargo se pueden reescalar para comparar.

El “factor de escala” (E) sería: $E = 4160 / 1024 = 4,0625$

Voy a expresar todos los valores en “píxeles de las imágenes originales” (es decir de 4160x2340):

OpenCV:

$$f_x = f_y = 3583,5$$

$$c_x = 2078,0$$

$$c_y = 1168,0$$

$$\text{Distorsión} = [0,8775 \quad -4,9156 \quad 0 \quad 0 \quad 12,5531]$$

$$\text{Error de reproyección} = 1,87$$

Matlab:

$$f_x = 3582,9$$

$$f_y = 3563,1$$

$$c_x = 2070,8$$

$$c_y = 1080,4$$

$$\text{Distorsión} = [0,0780 \quad 0 \quad -0,0077 \quad -0,0034 \quad 0]$$

$$\text{Error de píxel} = [1,43 \quad 1,57] \quad (\|err\| = 2,12)$$

Se puede ver que los valores de la “matriz de cámara” son muy similares en ambos casos. Sin embargo, los modelos para la distorsión conseguidos fueron bastante distintos.