Universitat d'Alacant
Universidad de Alicante

Comparativa de los distintos Keypoints y Descirptores en la detección de objetos 3D

Perception Systems

Robotics Engineering

Saúl Cova

rscr1@alu.ua.es

27 de mayo de 2018

Resumen

En este trabajo voy a hacer una comparativa de los distintos extractores de puntos característicos y descriptores que ofrece la librería openPCL. Para ello voy a implementar el *pipeline* local de reconocimiento de objetos 3D. Una vez implementado, crearé un un *ground truth* del objeto en la escena y compararé la distancia del objeto al *ground truth*. Este error entre el *groun truth* y el objeto será el que cuantificará como de buenos o malos son los *keypoints* y los descriptores. Finalmente muestro como *Intrinsic Shape Signatures* y Color SHOT con unos radios de búsqueda determinados, son los que mejor me han funcionado.

Índice

1. Introducción

Vivimos en un espacio tridimensional, donde los objetos tienen volumen, textura, resistencia mecánica, etc. Es por eso, que hay aplicaciones para las cuales no podemos fiarnos solo de imágenes en bidimensionales.

Con el auge de la robótica y la visión artificial, los robots cada vez deben de ser más capaces desenvolverse en entornos estocásticos con multitud de objetos diferentes. Es aquí donde entra en juego la percepción tridimensional del mundo, donde podemos saber la profundidad a la que se halla un objeto. Esta información es tremendamente útil cuando el robot tiene que manipular objetos o moverse en el espacio.

Por tanto, debemos desarrollar métodos para que el robot sea capaz de reconocer, de la forma más precisa posible, en una escena tridimensional objetos guardados previamente en una base de datos.

Es en ese ámbito en el cual se desarrolla este trabajo. Basándome en el *pipeline* local de reconocimiento de objetos tridimensionales, he realizado una comparativa de los diferentes extractores de características y descriptores para encontrar que combinación ofrece mejores resultados.

El *pipeline* seguido ha sido el siguiente:

1. Eliminar los planos predominantes de la escena.
2. Extraer puntos característicos.
3. Calculo de descriptores.
4. Calculo de emparejamientos.
5. Filtrado de los emparejamientos incorrectos.
6. Calculo de la transformación entre la escena y el objeto.
7. Refinar el resultado con ICP (Opcional).

2. Estado del arte

El estudio de los diferentes extractores de *keypoints* y descriptores ya ha sido realizado por varios autores [4, 2]. En ellos se expone como ISS y SIFT son los mejores keypoints y Shot Color el mejor descriptor.

De hecho podemos encontrar implementaciones del *pipeline* local muy eficientes [5] que hacen usos de estos detectores de *keypoints*.

3. Enfoque propuesto

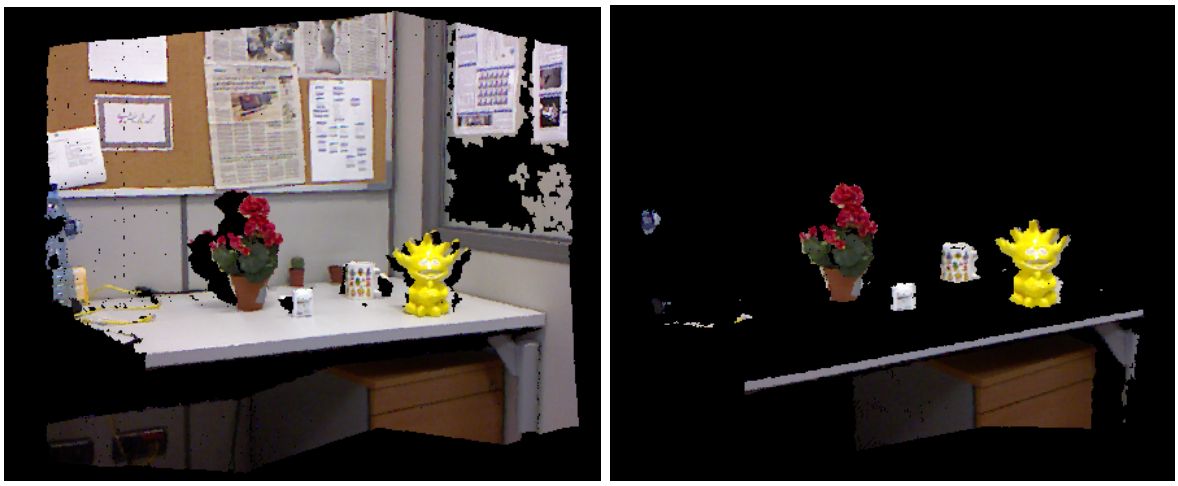
Debido a la naturaleza de este estudio, el enfoque que he propuesto ha sido la creación de un *script* automático de evaluación de las nubes de puntos. Aunque al principio comencé creando códigos separados, la gran cantidad de pruebas debido a los rangos de los parámetros de los extractores de puntos característicos, los descriptores y el *outlier rejection*, me hicieron darme cuenta que iba a necesitar condensarlo todo en un solo código. Para ello he hecho uso de las herramientas que ofrece el lenguaje C++. Al ser un Lenguaje Orientado a Objetos, he podido crear una clase que obtuviese los puntos característicos y otra que calculase descriptores, emparejamientos, y los demás pasos del *pipeline*. Además todo el código esta libre para la comunidad en un repositorio de GitHub [3].

Este estudio se ha realizado con una escena tomada por un sensor de luz estructurada, una Kinect, con una escena a 1 metro de distancia del sensor, dándonos una resolución de 0.002 metros. De hecho, si se comprueba experimentalmente calculando la resolución de la nube de puntos del objeto (que está segmentado de la nube de puntos original) da un valor de 0.00218916 metros, lo cual me confirma lo que intuía.

Esto implica una gran densidad en la nube de puntos, siendo muchos de estos puntos nada relevantes a la hora de buscar el mejor emparejamiento y haciendo que se ralenticen los cálculos. Es por ello por lo que se realiza un filtrado previo a la nube eliminando los planos predominantes con el algoritmo RANSAC. Como se puede observar en la Figura 1, con este método aligeramos enormemente la nube de puntos sin perder información de los objetos relevantes. En total hay tres planos principales que se corresponden con las dos paredes y la mesa. Para eliminarlas he utilizado tres umbrales deferentes. En primer lugar he eliminado la pared del fondo usado un umbral de 0.1 metros ya que quiero que elimine todo lo posible puesto que los objetos están mucho más adelantados. En segundo lugar he eliminado la pared de la derecha usando un umbral de 0.05 metros ya que los objetos también siguen estando alejados. En cambio, para la mesa, he usado un umbral de 0.01 metros para que elimine lo menos posible de los objetos. A partir de ahí, cualquier plano que eliminaba hacía que perdiera información de los objetos.

Debido a que este trabajo no trata de hacer una aproximación con cálculos en tiempo real, la eliminación de los planos predominantes se ha realizado uno a uno. Y una vez analizados los mejores extractores de características y descriptores, ya se podría ejecutar un algoritmo iterativo que fuera eliminando planos y calculando el mejor emparejamiento en tiempo real.

La librería openPCL ofrece una amplia variedad de extractores de puntos característicos en nubes de puntos. Entre ellos se han seleccionado los cinco: *Intrinsic Shape Signatures*, *Uniform Sampling*, *Susan*, *Harris3D*, *Harris6D* y *Sift*. Otros como *Narf* o *Agast*, fueron descartados ya que usaban nubes de puntos basadas en rangos. Cada uno de los extractores de *keypoints* tiene sus propios parámetros internos y diversas formas de funcionar, es por ello, que debido a la gran cantidad de parámetros he dejado por defecto los que venían en los *pipelines* y tutoriales de la web de OpenPCL [1] [6] y he variado el radio de búsqueda de los vecinos más cercanos. Por tanto, en mi evaluación he variado de forma iterativa tres parámetros básicos, estos son: los radios de búsqueda



(a) (b)

Figura 1: Nubes de puntos con (a) y sin (b) los planos predominantes de la escena

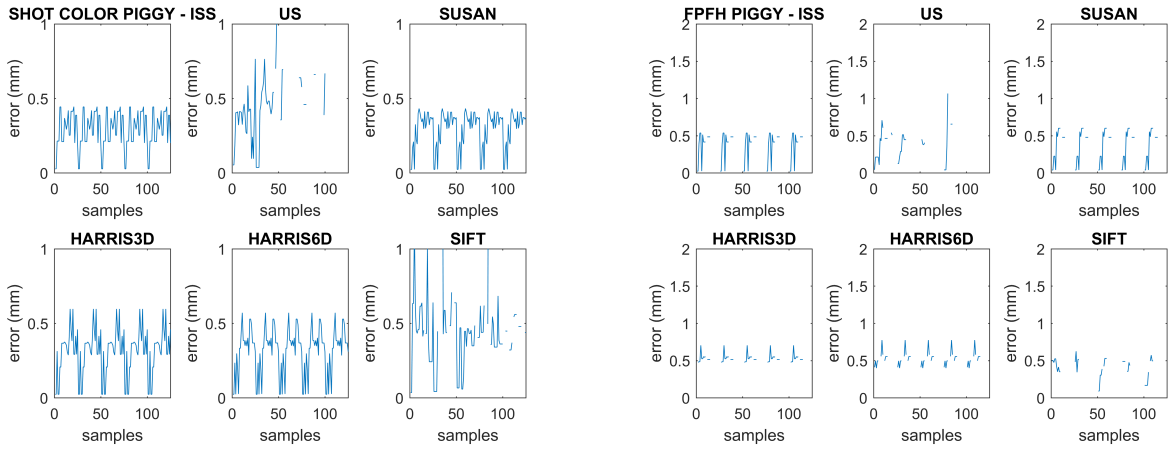
de puntos característicos y de los descriptores y el umbral para hacer el *outlier rejection*.

Los descriptores que he utilizado para analizar las nubes de puntos han sido: *SHOT*, *SHOT Color*, *Shape Context 3D*, *Fast Point Feature Histogram* y *Point Feature Histogram*. Todos estos descriptores son locales, de hecho, hay otros descriptores como *VFH*, pero estos son globales y para poder usarlos debería haber dividido la nube de puntos en *clusters*, y sobre cada *cluster* haber calculado el descriptor lo cual cambiaría mi *pipeline*. Esto es porque los descriptores globales te dan un único vector, a diferencia de los locales. En cuanto a porqué usar FPFH y PFH, es porque FPFH es una versión que parareliza los cálculos de PFH y hacerlo más rápido hacen aproximaciones numéricas, por tanto, PFH es más robusto. Por eso he querido probar ambos para ver si la variación es significativa.

Una vez obtenidas las correspondencias de los descriptores, para filtrar las malas he utilizado RANSAC.

4. Resultados experimentales

Cabe destacar que para aligerar los experimentos he usado implementaciones que usen OpenMP, y la librería de la STL, thread para paralelizar todos los cálculos posibles. Por desgracia, no he podido hacer una implementación con la GPU ya que la PCL es una librería joven y todavía no tiene un buen soporte para trabajar con CUDA.



(a) (b)

Figura 2: Gráficas experimentales usando el mismo objeto con dos descriptores: SHOT(a) y FPFH(b) usando todos los extractores de *keypoints*.

4.1. Experimentos iniciales

Los primeros test se han realizado variando los tres parámetros esenciales (radio de búsqueda de *keypoints*, de descriptores y de *outlier rejection*) de 0.02 hasta 0.1 con un muestreo de 0.02. Esto me ha llevado a que se realicen 750 pruebas por cada objeto por cada descriptor, lo cual nos lleva a que para todos los objetos se hallan realizado más de 15000 pruebas en total, sin contar las pruebas erróneas en las cuales han habido errores a la hora de calcular ciertos parámetros, o en los descriptores, *keytpoints*, etc.

Puesto que son una cantidad de datos inmensa, he utilizado Matlab para extraer los datos de los ficheros donde había guardado la información de cada experimento y los he representado gráficamente como se puede ver en la Figura 2. En esta figura vemos enfrentado el error en milímetros y cada prueba con las nubes de puntos, donde la primera prueba se corresponde con un valor de 0.02 de radio de búsqueda en los tres parámetros y se va incrementando de forma secuencial en tres bucles hasta llegar a 0.1 en los tres radios de búsqueda. Este error, medido como la distancia entre el objeto y la nube de puntos se ha calculado teniendo en cuenta los puntos de la escena que están a 1.75 de distancia del objeto como mucho.

De estas gráficas se puede observar que hay ciertas discontinuares a la hora de calcular la distancia, estas discontinuidades se deben a que no ha sido capaz de calcular bien la trasformación entre ambas nubes de puntos y, por tanto, no hay datos. Esto depende del extractor de *keypoints*, pero también del descriptor que usemos, como se puede ver al comparar los resultados de SHOT Color y FPFH. Por otro lado, es muy significativo que vemos patrones claros, que si nos fijamos, corresponden a los cálculos de los *keypoints*, pero también con los demás radios. Si bien es cierto que en este proceso estamos haciendo búsquedas con radios mucho mayores a la resolución de la nube.

Por tanto, de estos experimentos iniciales me lleva descartar los descriptores y extrac-

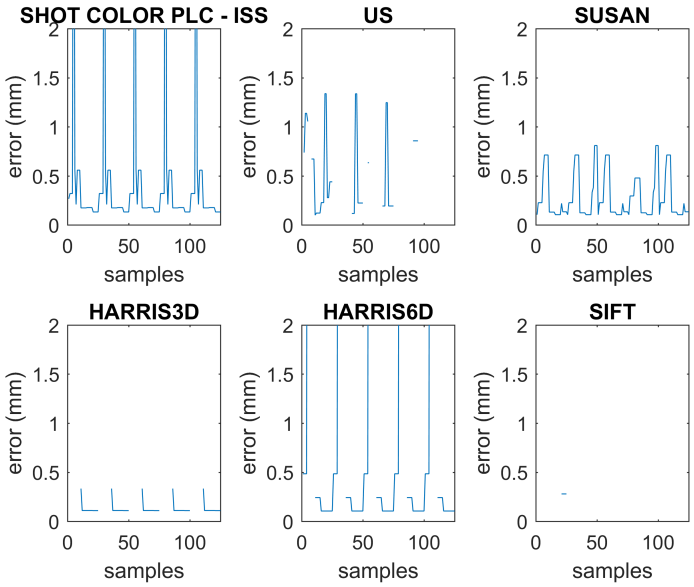


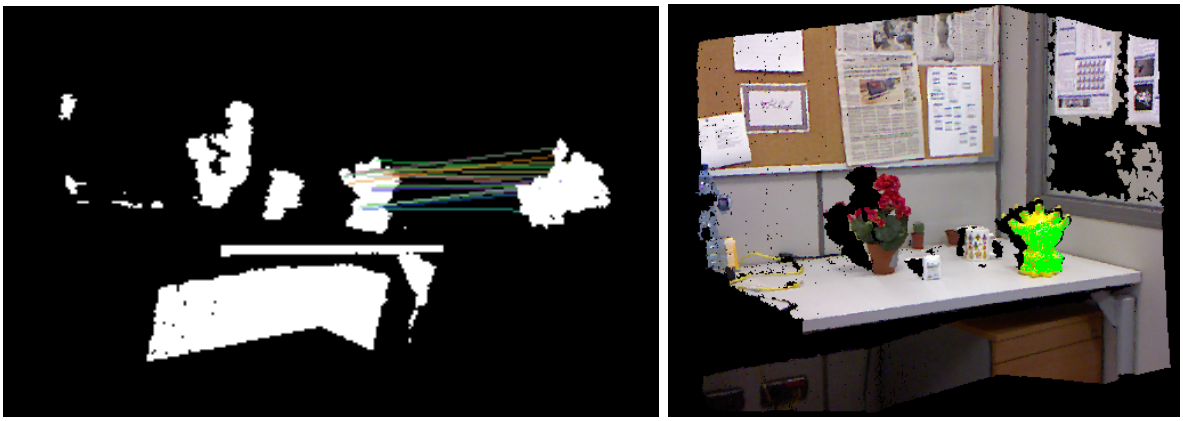
Figura 3: Comparación de los diferentes extractores de *keypoints* para el plc usando SHOT.

tores de *keypoints* con grandes discontinuidades. De hecho, ese objeto es relativamente simple y vemos que la mayor parte de los *keypoints* usados no dan saltos con SHOT Color. Pero si analizamos un objeto como el PLC, que tiene una muy baja concentración de *keypoints*, nos damos cuenta de que solo ISS y Susan son capaces de mantener el cálculo como se aprecia en la Figura 3.

En la Figura 4, muestro gráficamente los resultados obtenidos con uno de los objetos, obteniendo una distancia media entre ambas nubes de puntos de 26e-5 metros usando ICP y de 27e-5 sin ICP. Esto me lleva a una colocación perfecta, sin siquiera tener que usar ICP. Debido a las pruebas que había realizado. Pero, al llevar esto a varios objetos, tuve que añadir ICP si quería que los encontrara bien.

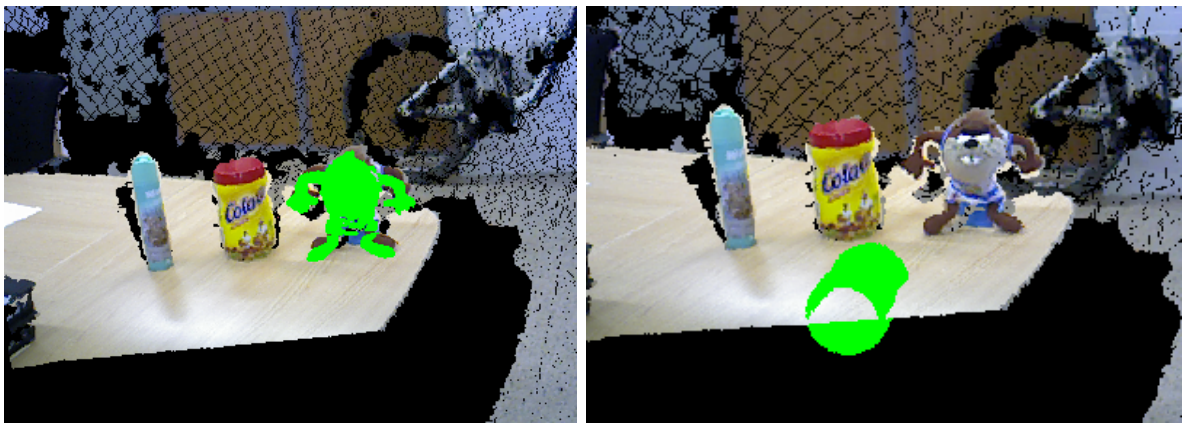
Una cuestión muy importante en este trabajo ha sido la caracterización del error. Cuando ves las gráficas del error, parece que uses el valor que uses en los radios las correspondencias van a ser buenas ya que el error suele ser bajo, entorno a 1 mm o menos. Este error, como he dicho puede parecer bajo, pero es tremendamente alto cuando visualizamos las nubes de puntos. Una de las conclusiones a las que he llegado es que los buenos emparejamientos se alcanzan con el error es del orden de magnitud de centésimas de milímetro, como ocurren en el caso anterior. Cuando estamos en décimas de milímetro, el emparejamiento es deficiente, no llegando a encajar bien e incluso puede llegar a estar del revés el objeto. Por encima de esos valor de error, los emparejamientos son totalmente inaceptables.

Probando este *pipeline* con los valores de 0.02 por cada radio de búsqueda y usando ICP he conseguido alinear todos los objetos con un error pequeño. Todos estos resultados los expondré en tablas en el siguiente apartado.



(a) (b)

Figura 4: Representación de las correspondencias filtradas(a) y de la transformación completa(b) en color verde.



(a) (b)

Figura 5: Representación de las transformaciones obtenidas (a con error de 23e-5 y b con 25e-5).

Una vez conseguida una buena aproximación, he probado mi *pipeline* con escenas reales, es decir, capturando una escena por un lado y con el modelo tridimensional del objeto por otro. Los resultados de estos experimentos(Figura 5) demuestran que la hipótesis de la caracterización del error es una medida engañosa. Esto es porque si el detector de correspondencias falla, y encuentra correspondencias en partes de la nube que no son al usar ICP y encajarlas sucede que la distancia entre las nubes es baja. Esto es cierto, ya que ha realizado una transformación y ha encajado el objeto en la escena, pero no ha encajado con el objeto que de la escena.

4.2. Experimentos finales

Debido a que la medida de la distancia no es fiable, he usado un *ground truth* para medir la distancia entre ambas nubes. De esta forma, se calcula la transformación entre el objeto y la escena y después se calcula la distancia entre el *ground truth* y el objeto transformado.

Este *ground truth* lo he conseguido de forma gráfica, mostrando en pantalla los resultados y buscando unos parámetros que ajusten bien el objeto en la escena y usando ICP. Una vez obtenidos, calculé la transformación del objeto en la escena y comparé la distancia que había al *ground truth*, obteniendo estos resultados para CSHOT.

En la Tabla 1 comparo los diferentes objetos, los extractores de puntos de interés, la distancia media obtenida, la distancia mínima y la distancia máxima al *ground truth*; y por último el número de valores para los cuales no se ha podido computar la transformación.

Uno de los principales problemas a la hora de calcular la transformación, es que ni siquiera sea capaz de calcularla. Por eso, he marcado en rojo las celdas que corresponden a los extractores de características que han presentado discontinuidades. Estos son SIFT y US, sobre todo en los casos más complicados, como son el plc y la taza. Por tanto, viendo los resultados del primer experimento y de este, puedo descartar el calculo de ambos extractores de puntos de interés de los experimentos.

Por otro lado, en naranja están los valores cuya distancia media supera el milímetro. Esto es porque al calcular la transformación, no lo hace correctamente sobre el *ground truth* como pasó en la Figura 5 con el objeto b. Por tanto, analizando los resultados del experimento anterior y viendo que HARRIS6D da valores altos de distancia en los objetos más complejos, voy a descartarlo también del cálculo.

Una vez hecho esto, quería analizar la influencia de ICP en todos los casos, por eso, calculé de nuevo con el mismo descriptor y todos los extractores de puntos de interés las nubes de puntos. Como se puede ver en la Figura 6, ICP mejora notablemente los resultados. Dependiendo de como se representen los valores la influencia de ICP se ve de una forma u otra siendo el resultado el mismo. agrupando los valores dispersos de distancia.

Por tanto, a la vista de los resultados, he podido descartar Uniform Sampling, Harris 3D y Sift.

COLOR SHOT					
Objet	Kpts	Midd dist(mm)	Min dist(mm)	Max dist(mm)	NaN
Piggy	ISS	0.1690	0	0.3980	0
	US	5.3936	0.0590	9.9030	0
	SUSAN	0.1888	0.0330	0.4150	0
	HARRIS3D	0.1994	0.0260	0.9090	0
	HARRIS6D	0.2012	0.0260	0.4030	0
	SIFT	1.7366	0.1970	5.8880	0
Plant	ISS	0	0	0	0
	US	2.3180	0.0470	8.2330	0
	SUSAN	0.0370	0.0370	0.0370	0
	HARRIS3D	0.0230	0.0230	0.0230	0
	HARRIS6D	0.0349	0.0170	0.2470	0
	SIFT	0.2087	0.1360	0.5800	0
Mug	ISS	0.2420	0	0.5630	0
	US	13.8562	0.3640	22.1120	0
	SUSAN	0.2818	0.1170	0.5500	0
	HARRIS3D	0.2711	0.0600	0.4680	0
	HARRIS6D	5.8158	0.1220	19.2180	0
	SIFT	11.5628	0.2730	15.3260	25
Plc	ISS	0.0352	0	0.1960	0
	US	10.3299	0.1850	19.8120	5
	SUSAN	0.2180	0.1740	0.3490	0
	HARRIS3D	6.9244	0.3460	16.7920	0
	HARRIS6D	3.0644	0.2510	14.1950	0
	SIFT	-	-	-	125

Cuadro 1: Comparativa usando CSHOT de los diferentes extractores de keypoints usando como radios 0.02 hasta 0.1 en incrementos de 0.02.

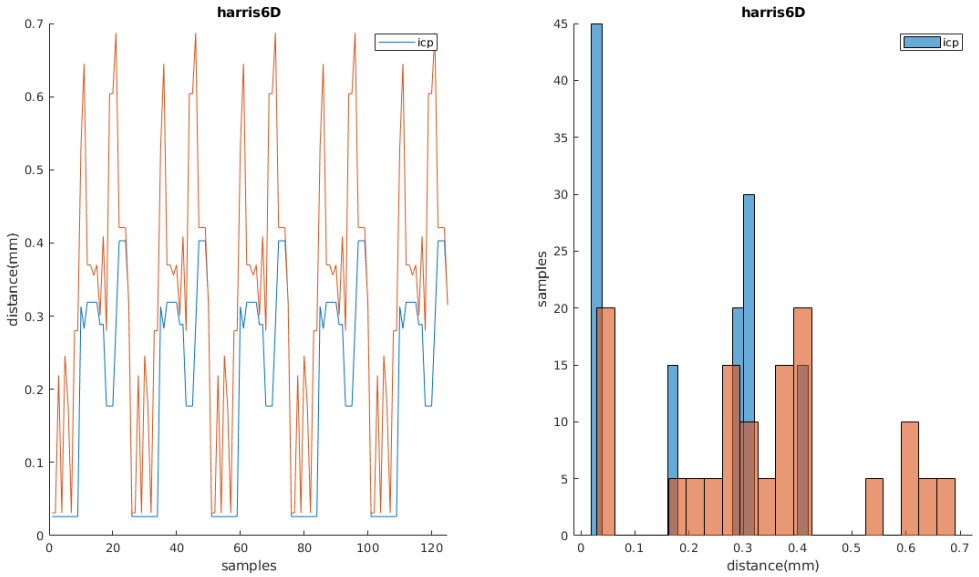


Figura 6: Analisis de la influencia de ICP en el calculo de la distancia al *groud truth*. (a) Representa como va cambiando la distancia al cambiar la muestras. (b) Representa el número de muestras por cada medida de distancia obtenida.

Puesto que los valores de los radios que había usado (de 0.02:0.02:0.1) habían sido elegidos en un principio sin criterio alguno y más bien por error, decidí cambiar los radios a múltiplos de la resolución de la escena. Para ello he evaluado ISS, Susan y Harris3D multiplicando el valor de resolución por 2,3,4,5 y 6. No he usado el valor de la resolución de la nube (es decir, por 1) ya que me parecía demasiado pequeño. Ya que en un principio estos valores me parecían que se comportarían mejor, al ser dependientes de la escena.

Como se puede apreciar en la Tabla 2, con objetos grandes se obtiene una ligera mejora, pero si vamos a objetos más complicados de detectar porque tienen menos puntos, los radios dependientes de la resolución no introducen ninguna mejora. Vistos los resultados obtenidos, también probé usando la resolución del propio objeto, pero esto tampoco mejoró el problema.

Como último experimento, he buscado cual de los 3 extractores de *Keypoints* funciona mejor. Para ello, he filtrado para ello he realizado un algoritmo, que busca la combinación que minimiza la distancia a la nubes de puntos para los 4 objetos. Para ello he ido calculando la distancia mínima media que se generaría con los cuatro objetos y guardando los radios para los cuales se ha alcanzado, quedándome al final con la mejor muestra.

Como se muestra en la Tabla 3, el mejor resultado se obtiene con ISS.

COLOR SHOT					
Objet	Kpts	Midd dist(mm)	Min dist(mm)	Max dist(mm)	NaN
Piggy	ISS	0	0	0	0
	SUSAN	0.0330	0.0330	0.0340	0
	HARRIS3D	0.0260	0.0260	0.0260	0
Plant	ISS	0	0	0	0
	SUSAN	0.0370	0.0370	0.0370	0
	HARRIS3D	0.0230	0.0230	0.0230	0
Mug	ISS	12.5454	0.0110	19.4220	0
	SUSAN	4.0926	0.1170	19.9940	0
	HARRIS3D	11.4436	0.0600	19.5710	0
Plc	ISS	11.5058	0.0050	14.3810	0
	SUSAN	8.3780	0.1740	14.8090	0
	HARRIS3D	16.7920	16.7920	16.7920	0

Cuadro 2: Comparativa usando CSHOT de los diferentes extractores de *keypoints* usando como radio múltiplos (2,3,4,5,6) de la resolución de la escena.

Kpts	Kpts rad(m)	Desc rad(m)	Inlier Threshold(m)	Middle dist(mm)
ISS	0.02	0.02	0.04	0
SUSAN	0.02	0.02	0.02	0.0902
HARRIS3D	0.02	0.1	0.02	0.1530

Cuadro 3: Búsqueda de la mejor combinación de valores que dan la mínima distancia media a los 4 objetos.

Figura 7: Transformación final con los mejores parámetros para CSHOT e ISS.

5. Conclusiones

En este trabajo, he implementado el *pipeline* local de detección de objetos para comparar los diferentes extractores de puntos característicos, descriptores. A partir de una escena y un objeto dados, he usado la técnica del calculo de la distancia para poder cuantificar los resultados. Esto me ha llevado a tener que encontrar un *ground truth* de los objetos que quería reconocer.

Una vez realizada la experimentación he podido descartar todos los descriptores, debido a que en el rango de estudio producen discontinuidades a la hora de calcular la transformación de todos los objetos en la escena. Estos son FPFH, PFH, SHOT y Shape Context. Aunque no puedo descartar totalmente que no se pudiera encontrar una buena transformación con esos descriptores, esas discontinuidades me han llevado a descartarlos, sobre todo si se usan radios de búsqueda dinámicos que varíen con la resolución de la nube de puntos.

Después de quedarme con CSHOT como el mejor, descarté los extractores de Keypoints Sift, Harris 6D y Uniform Sampling ya que, para los objetos complejos, producían también discontinuidades a la hora de calcular la transformación o daban un mal emparejamiento.

Por último, he evaluado ISS, Susan y Harris 3D obteniendo la mejor transformación de cada uno. De esto he sacado en conclusión que ISS es el que mejor funciona. Pero tampoco se puede descartar Susan, ya que mi *ground truth* no es perfecto y el valor de distancia es suficientemente pequeño como para considerarlo ruido. En cambio, Harris

3D aunque funciona bien, no es tan bueno como los otros dos.

Referencias

[1] *3D Object Recognition based on Correspondence Grouping*. URL: `http://pointclouds.org/documentation/tutorials/correspondence_grouping.php` (visitado 27-05-2018).

[2] Lu'is A. Alexandre. «3D Descriptors for Object and Category Recognition: a Comparative Evaluation». En: (ene. de 2012).

[3] Saúl Cova. *3D object detector repository*. 2018. URL: `https://github.com/rscova/3d_object_detection` (visitado 27-05-2018).

[4] Sílvio Filipe y Luís A. Alexandre. «A comparative evaluation of 3D keypoint detectors in a RGB-D Object Dataset». En: *International Conference on Computer Vision Theory and Applications (VISAPP)*. IEEE, 2014.

[5] Alberto Garcia-Garcia y col. «Interactive 3D object recognition pipeline on mobile GPGPU computing platforms using low-cost RGB-D sensors». En: *Journal of Real-Time Image Processing* (2016), págs. 1-20.

[6] Víctor Rodríguez. *PhD-3D-Object-Tracking*. 2018. URL: `http://robotica.unileon.es/index.php/PhD-3D-Object-Tracking` (visitado 27-05-2018).