EL7008 - Procesamiento Avanzado de Imágenes

Segmentación de objetos usando atención visual basada en profundidad y movimiento

Profesor: Javier Ruiz del Solar.
Auxiliar: Patricio Loncomilla Z.

Camilo Jara Do Nascimento

Introducción

En el presente informe se presentan las bases para la definición del proyecto de segmentación de objetos usando atención visual basada en profundidad y movimiento.

Este proyecto consiste en implementar un sistema que permita segmentar objetos al robot Pepper con el objetivo de entrenar posteriormente detectores de objetos. Debido a que Pepper tiene dos cámaras RGB es posible obtener la profundidad de cada pixel en la imagen. Por esto, los enfoques del proyecto se basan en aprovechar la información que cada una entrega tal como el movimiento y la distancia o profundidad entre objeto-cámara.

Para lo anterior, primeramente se realiza una breve explicación del uso del *disparity map* a fin de encontrar la distancia objeto-cámara utilizando ambas cámaras y posterior segmentación [2]. Luego es posible además segmentar sólo utilizando una, para ello el método propuesto utiliza un análisis de movimiento con Optical Flow Tracking (KLT) [1].

Las siguientes secciones corresponden al desarrollo de los método utilizando como principal librería Opencv, resultados obtenidos y conclusiones obtenidas al respecto.

Objetivo

El objetivo principal es una segmentación de imágenes robusta que permita obtener un dataset que posteriormente pueda ser entrenado para clasificación de objetos, para ello se busca encontrar un *blob* dada una secuencia de *frames* de un objeto en movimiento mostrado al robot Pepper por un “profesor” ubicado atrás del objeto. La gran dificultad de esto es diferenciar el objeto en movimiento y que el blob no tome en cuenta al “profesor”.

Marco Teórico

Seguimiento por profundidad

Cuando se toman imágenes utilizando una cámara estenopeica (pin-hole camera) se pierde la información de la profundidad de la imagen debido a la conversión de 3D a 2D. A partir de esto surge la duda de cómo mantener esta dimensión utilizando este tipo de cámaras, la respuesta es utilizar 2 cámaras (stereo vision), ver Figura 1.

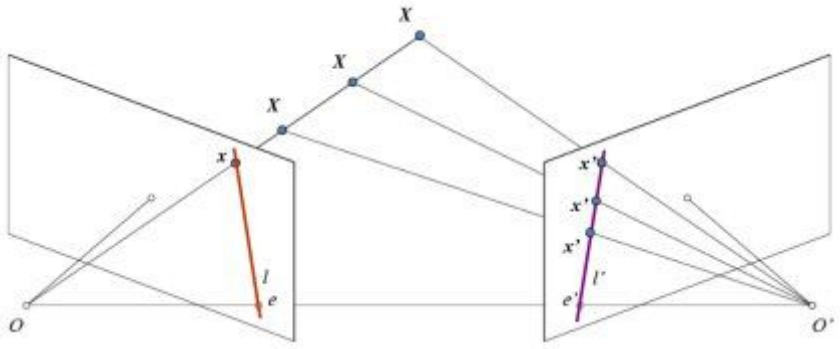

Figura 1. Representación de dos cámaras y sus respectivas proyecciones sobre un punto en el espacio.

Primero es necesario entender conceptos básicos de la geometría multivisión, en Figura 2 se observan triángulos semejantes dado el problema de las 2 cámaras. Luego sea x y x’ las distancia entre los puntos en el plano de la imagen correspondiente al punto 3D de la escena y su centro de la cámara, B la distancia entre las cámaras y f la distancia focal de cada una, por tanto, la disparidad se puede calcular como sigue:

$$\text{disparity} = x - x' = \frac{Bf}{Z}$$

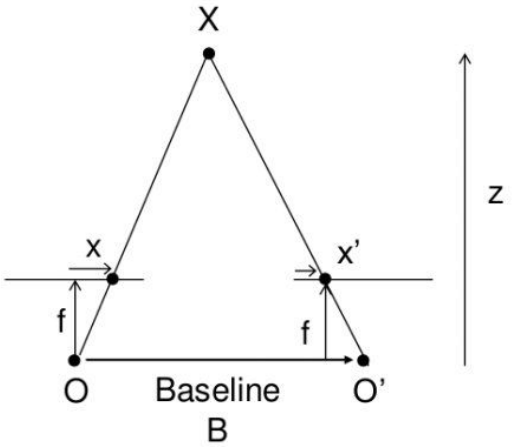

Figura 2. Representación de dos cámaras con triángulos formados para el cálculo de la disparidad.

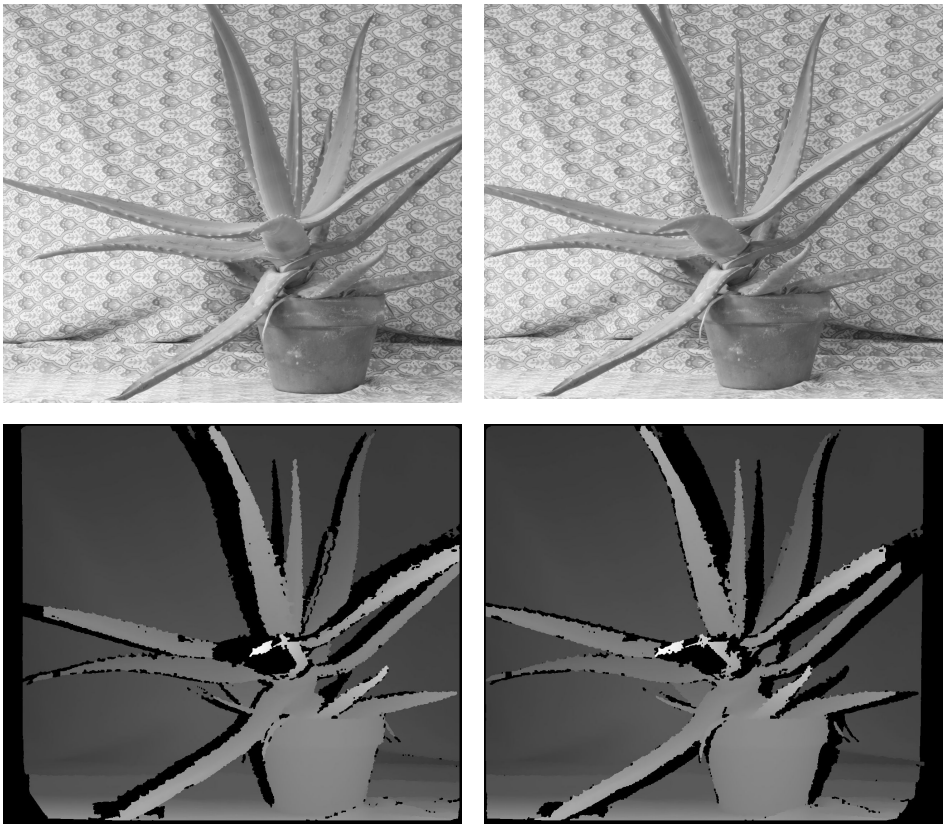

Figura 3. Cálculo de *disparity map* dada una imagen tomada desde la izquierda y otra desde la derecha.

Junto con lo anterior se propone seguir el enfoque de [2] que a partir del cálculo de los *disparity map* de las imágenes, ver figura 3, se segmenta el objeto filtrando la imagen y obteniendo el blob a partir del píxel con mayor brillo e ir aumentando su tamaño a partir de su vecindad. Notar que el píxel con más brillo representa aquel que está más cercano en la imagen ya que la disparidad es inversamente proporcional a la distancia[1].

[1] Recordar que el objeto en cuestión será el objeto más cercano al robot Pepper.

Dado que se desea probar el funcionamiento del sistema antes de implementarlo en el robot Pepper se propone utilizar el dataset de iCub World [3] que tiene las imágenes requeridas verificando rotaciones 2D y 3D del objeto, escala o acercamiento y traslación de este.

Seguimiento por movimiento

Como se ha podido ver, el anterior método necesita de dos cámaras permitiendo obtener la profundidad de cada píxel, sin embargo, no siempre se tiene esta disponibilidad.

Otro enfoque es realizar el seguimiento a partir del movimiento de la imagen, para ello primeramente se realiza una búsqueda de N puntos de interés $\{x_i(t) = u_i(t), v_i(t)\}^N_{i=1}$ para cada frame, luego se calcula el flujo óptico (Optical Flow) entre el punto $x_i(t)$ y $x_i(t-1)$. Luego para cada frame I_{t-1} e I_t se obtiene un conjunto de vectores de flujo óptico $v_1(t),...,v_n(t)$ donde $v_i(t) = x_i(t) - x_i(t-1)$, con esto es posible calcular la esperanza $\mu(t)$ y matriz de covarianza $\Sigma(t)$ del conjunto. Posteriormente estás estadísticas del conjuntos serán entrenadas en 5 regresores ν-SVM permitiendo predecir la esperanza futura y matriz de covarianza futura[2], ver Figura 6.

Finalmente dado un conjunto de prueba se utiliza el clasificador pre-entrando calculando las matrices de esperanza y covarianza para posteriormente obtener la probabilidad condicional $P(v_i(t) \mid \mu', \Sigma')$ que mientras mayor sea su valor mejor es la confidencia, una forma de eliminar puntos con baja confidencia (baja probabilidad) es utilizar un umbral $P(v_i(t) \mid \mu', \Sigma') > \theta$ para seleccionar solo los mejores, por otra parte, el criterio adoptado en [1] es usar la distancia Mahalanobis $\mathbb{M}(v_i(t), \mu', \Sigma') = ((v_i(t)-\mu')^T \Sigma'^{-1}(v_i(t)-\mu'))^{1/2}$ y aplicar el umbral $\mathbb{M}(v_i(t), \mu', \Sigma') > \theta$ para seleccionar puntos.

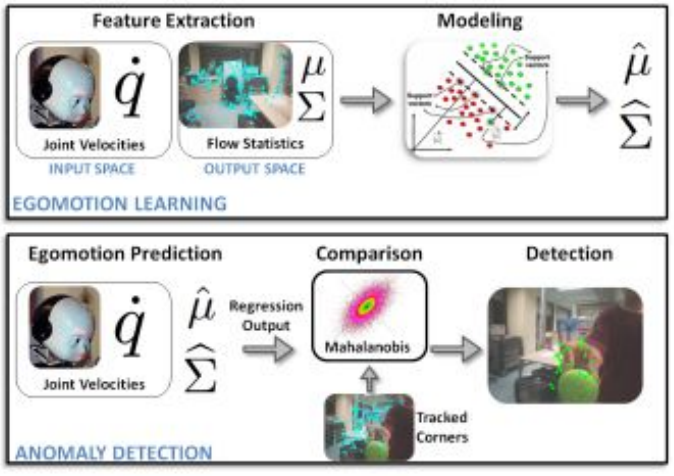

Figura 6. Esquemática para segmentación de objetos usando seguimiento por movimiento [1].

[2] Notar que es posible prescindir de los regresores

Desarrollo

Primero es necesario la obtención de los datos correspondientes a la base de iCub World [3].

Seguimiento por profundidad

Para el cálculo del disparity map se utiliza la librería LIBELAS [4] que permite la eficiente y veloz obtención de éstas dada una imagen izquierda y una derecha. Notar que se modifica el main.cpp de la librería debido que ésta realiza el cálculo dadas dos imágenes y es necesario realizarlo para todo el conjunto de imágenes.

La base de iCub World contiene un archivo .txt donde indica la correspondencia entre la imagen izquierda y la derecha, por lo tanto, es necesario utilizar este para una correcta elección al momento de calcular los disparity maps.

Por otra parte, la librería LIBELAS obtiene el disparity map a partir de imágenes en .pgm por lo tanto primero es necesario realizar la transformación correspondiente. A partir de esto es posible comenzar la segmentación del objeto, en figura 7 se observa el diagrama completo del procedimiento.

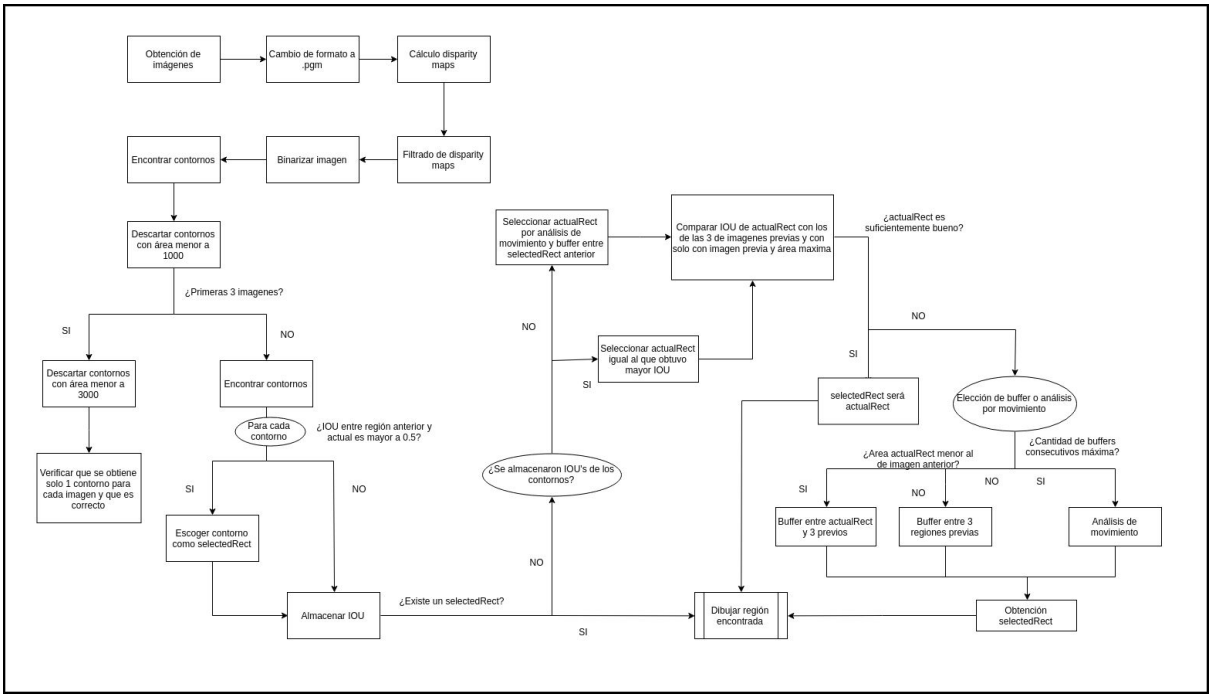

Figura 7. Diagrama de segmentación por profundidad.

En figura 8 se encuentra el código que permite el primer paso del método, es decir, filtrar el disparity map obtenido [2].

```
void filterDisparity(Mat& img){
   double thresh = 50;
   double maxValue = 255;
   threshold(img,img, thresh, maxValue, THRESH_TOZERO);
   GaussianBlur(img, img, Size(5,5), 1.5, 1.5);
   for(int i=0; i<4; i++){
       dilate(img, img, Mat());
       GaussianBlur(img, img, Size(5,5), 2, 2);
   }
   for(int i=0; i<2; i++){
       erode(img,img, Mat());
       GaussianBlur(img, img, Size(5,5), 2, 2);
   }
}
```

Figura 8. Filtrado de disparity map.

Acto seguido se procede a seleccionar el blob perteneciente al objeto, el proceso se divide en 2 partes: encontrar contornos en imagen binarizada dado un umbral y verificar para cada contorno su compromiso con el objeto que se desea segmentar.

Para la verificación se opta por comparar la región de interés actual con respecto a las anteriores. Para esto se utiliza el enfoque *intersection over union* (IOU), ver figura 9, y las áreas de cada región[3]. En caso de no encontrar una región lo suficientemente buena se utilizan buffers entre 3 imágenes previas y la actual o sólo entre las 3 imágenes previas permitiendo calcular el promedio ponderado entre éstas y por ende obtener una región que aproxima el lugar del objeto dependiendo de las regiones previas. A fin de solventar el uso de buffers y dada su propagación de errores se propone utilizar análisis por movimiento en caso de haber n buffers consecutivos. En figura 10 se observan los primeros filtros de contornos permitiendo descartar aquellos que tengan un IOU baja. Luego en figura 11 se observa la elección de actualRect y selectedRect.

```
double IOU2Rects(Rect iboundingRectActual, Rect boundingRectPrev){
    Rect interRect = iboundingRectActual & boundingRectPrev;
    Rect unionRect = iboundingRectActual | boundingRectPrev;
    double iou = double(interRect.area()) / unionRect.area();
    return iou;
}
```

Figura 9. Cálculo IOU entre una de las regiones encontradas y la región seleccionada en la imagen anterior.

[3] Aquella región que tenga una mayor IOU con respecto a las anteriores y un área razonable será la que tenga un mejor compromiso.

```
for(size_t i = 0; i< contours.size(); i++){
       color = Scalar(rng.uniform(0, 256), rng.uniform(0,256),
rng.uniform(0,256));
       int boundArea = boundRect[i].area();
       int realArea = img_real.rows * img_real.cols;

       //First Filter of bounds
       if(boundArea != realArea && boundArea >= 1000){
           //Second Filter of bounds for i_img < 3
           if(i_img < 3 && boundArea >= 3000){
               enterBuffer = 0;
               selectedRect = boundRect[i];
               buffer_rect.push_back(selectedRect);
               rectangle(img_real, selectedRect.tl(), selectedRect.br(),color,
2);
           }
           else if (i_img >= 3){
               if(IOU2Rects(boundRect[i],buffer_rect[i_img-1]) > 0.5){
                   enterBuffer = 0;
                   selectedRect = boundRect[i];
               }
               IOUs.push_back(IOU2Rects(boundRect[i],buffer_rect[i_img-1]));
               posBound.push_back(i);
           }
       }
   }
```

Figura 10. Almacenamiento de IOU para cada región encontrada en imagen actual.

```
if(selectedRect.area() == 0){
   Point2d meanPtTl;
   Point2d meanPtBr;
   Rect actualRect =
selectActualRect(selectedRect,boundRect,buffer_rect,IOUs,posBound,i_img,pointsMah
alanobis,selByPrev,testMode);
   double minArea = abs(buffer_rect[i_img-1].area() - actualRect.area());
   //elegir como condicion comparacion ious entre promedio 3 anteriores
   double compareIOU4Rects = IOU4Rects(actualRect,buffer_rect,i_img);
   double compareIOU2Rects = IOU2Rects(actualRect,buffer_rect[i_img-1]);
   selectedRect =
selectMethod(actualRect,buffer_rect,i_img,meanPtTl,meanPtBr,compareIOU4Rects,comp
areIOU2Rects,minArea,enterBuffer,pointsMahalanobis,threshActualRectArea,n_buffers
,threshCompIOU2Rects,threshCompIOU4Rects,testMode);
}
```

Figura 11. Elección de actualRect y selectedRect a partir de contornos que tengan mayor IOU.

En figura 12 se observa la elección de actualRect (selectActualRect), en caso de que se hayan encontrado contornos (IOUs.size() != 0) se escogerá aquel que tenga mayor IOU, caso contrario se elige a partir del análisis en movimiento del objeto y un buffer con el selectedRect anterior.

```
Rect selectActualRect(Rect selectedRect, vector<Rect> boundRect,vector<Rect>&
buffer_rect, vector<double> IOUs,vector<int> posBound, int i_img, vector<Point2f>
pointsMahalanobis,int& selByPrev, bool testMode){
   Point2d meanPtTl;
   Point2d meanPtBr;
   Rect actualRect;
   if(IOUs.size() != 0){ // and maxIOU
       int posMaxIOU = std::distance(IOUs.begin(), std::max_element(IOUs.begin(),
IOUs.end()));
       double maxIOU = *max_element(IOUs.begin(), IOUs.end());
       actualRect = boundRect[posBound[posMaxIOU]];
   }
   else{
       //actualRect = buffer_rect[i_img-1]; // VERIFICAR
       selByPrev = 1;
       Rect pointsRect = boundingRect(pointsMahalanobis);
       Point2d mPtTl;
       Point2d mPtBr;
       buffer2imgs(pointsRect,buffer_rect,i_img,mPtTl,mPtBr);
       actualRect = Rect(mPtTl,mPtBr);
   }
   return actualRect;
}
```

Figura 12. Elección de actualRect verificando si se almacenaron IOU's de los contornos.

Finalmente en figura 13 se encuentra la implementación de selectMethod que permite identificar si es necesario utilizar buffers o análisis por movimiento para el selectedRect final, en caso que se usen más de n_buffers consecutivos se usará éste ultimo.

```
Rect selectMethod(Rect actualRect,vector<Rect>& buffer_rect, int i_img,Point2d
meanPtTl, Point2d meanPtBr, double compareIOU4Rects,double compareIOU2Rects,
double minArea, int& enterBuffer, vector<Point2f> pointsMahalanobis,int
threshActualRectArea,int n_buffers,double threshCompIOU2Rects,double
threshCompIOU4Rects,bool testMode){
   if((compareIOU4Rects >= 0.19 || compareIOU2Rects >= 0.23) && minArea < 18500
&& actualRect.area() > 8000){ //and areas minArea > 8000
       enterBuffer = 0;
       return actualRect;
   }
   else if((compareIOU4Rects >= threshCompIOU4Rects || compareIOU2Rects >=
threshCompIOU2Rects) && minArea < 30000 && actualRect.area() >
threshActualRectArea){
       enterBuffer = 0;
       return actualRect;
   }
   else{
       Rect selectedRect;
       if(buffer_rect[i_img-1].area() > actualRect.area()  && enterBuffer <
n_buffers){
           enterBuffer += 1;
           buffer4imgs(actualRect,buffer_rect,i_img,meanPtTl,meanPtBr);
           selectedRect = Rect(meanPtTl,meanPtBr);
       }
       else if (buffer_rect[i_img-1].area() <= actualRect.area() && enterBuffer <
n_buffers){
           enterBuffer += 1;
           buffer3imgs(buffer_rect,i_img,meanPtTl,meanPtBr);
           selectedRect = Rect(meanPtTl,meanPtBr);
       }
       else{
           enterBuffer = 0;
           selectedRect = boundingRect(pointsMahalanobis);
       }
       return selectedRect;
   }
}
```

Figura 13. Elección de método que escoge selectedRect.

Seguimiento por movimiento

Se utilizó la misma base de datos que la parte de seguimiento por profundidad, iCub World [3], usando solo las de la cámara izquierda.

En figura 14 se observa el diagrama para la segmentación por movimiento. Para la detección de puntos de interés realizaron pruebas con un detector Shi-Tomasi. El cálculo de flujo óptico se realiza mediante el método disponible en la librería de Opencv calcOpticalFloPyrLK, igualmente los regresores usados son obtenidos de la misma. El cálculo de las estadísticas se utiliza el método calcCovarMatrix y a partir de ellas se calcula la distancia Mahalanobis. Finalmente para encontrar la región de interés se utiliza el método boundingRect dado los puntos seleccionados.

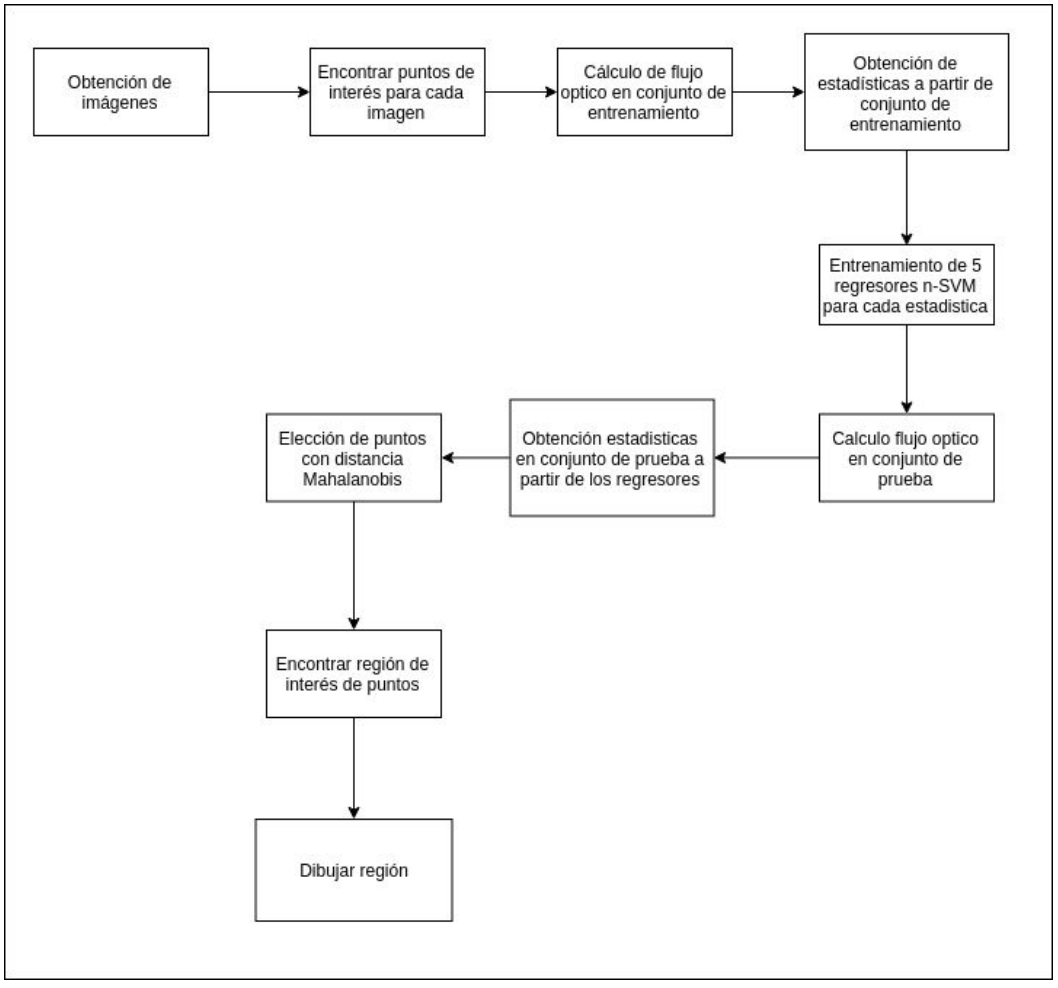

Figura 14. Diagrama de segmentación por profundidad.

Resultados

Seguimiento por profundidad

Como se menciona anteriormente se utiliza la librería LIBELAS para el cálculo de los disparity maps, el primer paso es cambiar el formato de la base de imágenes desde .jpg a .pgm dado que LIBELAS utiliza este último[4], lo que da paso al primer entregable.

A partir de la modificación de formato es posible calcular los disparity maps, en figura 15 se muestra un ejemplo de ello donde es posible verificar su correcto funcionamiento, es decir, los objetos más cercanos a la cámara son aquellos píxeles en blanco (libro). Las pruebas se realizaron para libros, celulares y estuches, en caso de querer utilizar el método para otro objeto basta modificar los parámetros que se mencionan más adelante y entrar al modo de prueba (testMode).

Figura 15. Cálculo de disparity map. A la izquierda imagen real y a la derecha su respectivo disparity map.

Luego es necesario filtrar la imagen permitiendo eliminar las irregularidades que ésta posea, para ello se utiliza un umbral dejando en 0 aquellos valores menores a 50 y manteniendo el valor de los otros. Luego se utiliza un filtro gaussiano de tamaño 5x5 con σ igual a 1.5 seguido de dilataciones y erosiones. En total 4 dilataciones y 2 erosiones intercaladas por filtros Gaussianos de tamaño 5x5 con σ igual a 2, en figura 16 se observa el resultado final de estos filtros donde se verifica la eliminación de “huecos” en la imagen y su suavizado.

[4] Notar que la compresión .pgm tiene mejor calidad que la .jpg.

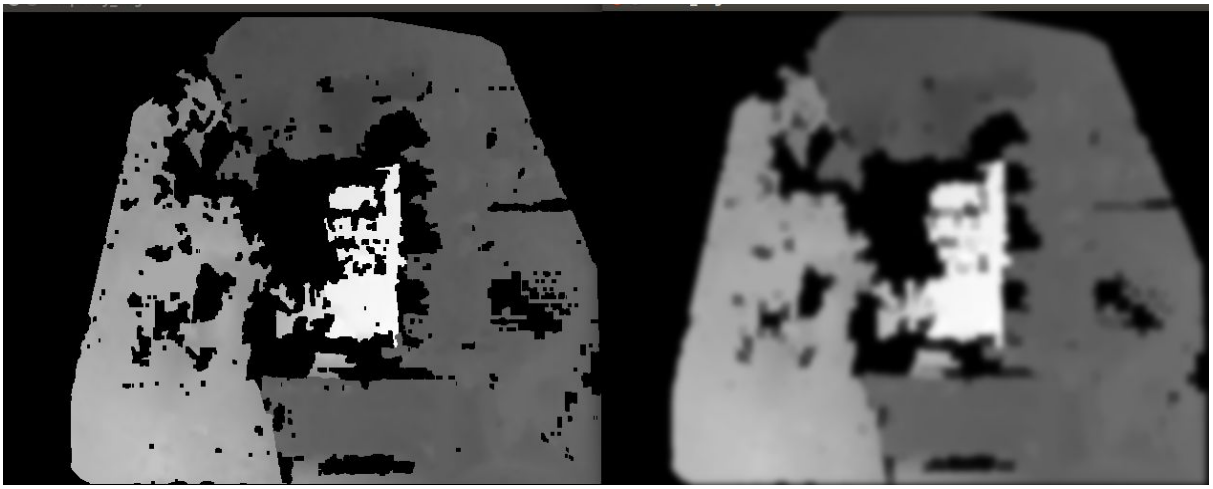

Figura 16. Disparity map luego de aplicar filtros. A la izquierda imagen sin filtros y a la derecha imagen con filtros.

Con la imagen ya filtrada es posible encontrar el blob que pertenece al objeto en cuestión para ello primero se utiliza umbral binario para acentuar la región del objeto con valor en 200. Luego se buscan los contornos de la imagen usando el método findContours de Opencv en modo RETR_TREE que entrega un vector con cada contorno encontrado, véase [5] para más detalle del algoritmo. A partir de los contornos encontrados es posible encontrar una región de interés para cada uno, sin embargo, es necesario descartar aquellos que no corresponden al objeto a detectar.

Para lo anterior se propone verificar que el área de la región es lo suficientemente grande con un umbral de 1000[5] que representa el primer filtro a los contornos obtenidos. Luego se utiliza un segundo filtro para las 3 primeras imágenes para áreas mayores de 3000, es importante este punto ya que algunos disparity maps no son lo suficientemente concluyentes para encontrar el objeto en cuestión, a modo de solución se propone utilizar un buffer comparando la imagen actual y las 3 previas[6] que permita predecir el movimiento del objeto pudiendo prescindir del buen funcionamiento del disparity map, además si se ocurren muchos buffers consecutivos se opta por usar análisis de movimiento. Por lo dicho, una restricción del método es que las primeras 3 imágenes tengan una buena elección del contorno, en caso de que el disparity map de una o varias imágenes de las 3 primeras sea mala basta comenzar el algoritmo de las imágenes posteriores[7].

Una vez escogidas las primeras 3 imágenes se procede a verificar cada uno de los contornos escogidos por las siguientes, para ello se compara el IOU (intersection over union) entre el contorno actual y el anterior[8]. Si el IOU entre la región actual y la anterior es mayor a cierto umbral (0.5) se escoge ese contorno (selectedRect) y se almacena su IOU en un vector, caso contrario solo se almacenará su IOU.

---

[5] Es decir se descartan las regiones menores a 30x30 píxeles aproximadamente.
[6] Posteriormente se explicará el uso de dos buffers, uno con respecto a la imagen actual y las tres previas, mientras que el otro solo con respecto a las tres anteriores.
[7] Si el disparity map de alguna de las primeras 3 imágenes es malo el error se propagará en los buffers no permitiendo una buena caracterización del objeto.
[8] Como primer filtro compara solo con el anterior, más adelante será necesario comparar con los 3 anteriores.

En caso que aún no se encuentre selectedRect se procede a escoger entre todos los contornos aquel que presenta mayor IOU (actualRect), si el método no encontró ningún contorno actualRect será el mismo de la imagen anterior (selectedRect anterior).

Dado un actualRect se propone un segundo filtro con respecto a las IOUs entre los 4 contornos previos o con al anterior y que contenga un área máxima[9], en caso de que el contorno con mayor IOU pase este filtro actualRect será el nuevo selectedRect, caso contrario se utilizarán buffers[10] dependientes de la imagen actual y los tres anteriores o solo de los tres previos que permiten encontrar el selectedRect actual. Si el área de actualRect es menor al selectedRect de la imagen anterior y mayor a threshMinArea se utiliza el buffer de 4 imágenes[11], caso contrario se utiliza un buffer sólo con las imágenes previas[12]. Por otra parte, como se menciona anteriormente se usa análisis de movimiento en caso de muchos buffers consecutivos.

En este punto ya se tiene un selectedRect para la imagen actual cubriendo todos los casos posibles por lo que se almacena en el vector de selectedRects y se gráfica su resultado segmentando la imagen del objeto, ver figura 17.

Figura 17. Segmentación de objetos.

[9] Esto debido a que existen contornos muy grandes que pueden pasar el filtro de IOUs.
[10] Los buffers principalmente consisten en un promedio ponderado entre imágenes dando un mayor peso al selectedRect previo v/s los anteriores.
[11] Este caso ocurre cuando actualRect es más pequeño y se encuentra contenido en el selectRect de la imagen anterior, por esto basta aumentar su tamaño usando un buffer de las 4 imágenes (3 anteriores y la actual.
[12] Este caso ocurre cuando actualRect es mucho más grande que el selectedRect anterior y posiblemente desplazado, por lo tanto, se considera como un contorno totalmente erróneo.

Pese a los buenos resultados mostrados aún existen ciertos casos donde el método falla, esto ocurre cuando varios disparity maps consecutivos son malos ya que la propagación de error debida a los buffers es muy grande por lo que no permite discernir de forma correcta la posición del objeto, ver figura 18. Sin embargo, es posible ver que el método vuelve a la normalidad cuando un nuevo buen disparity map aparece.

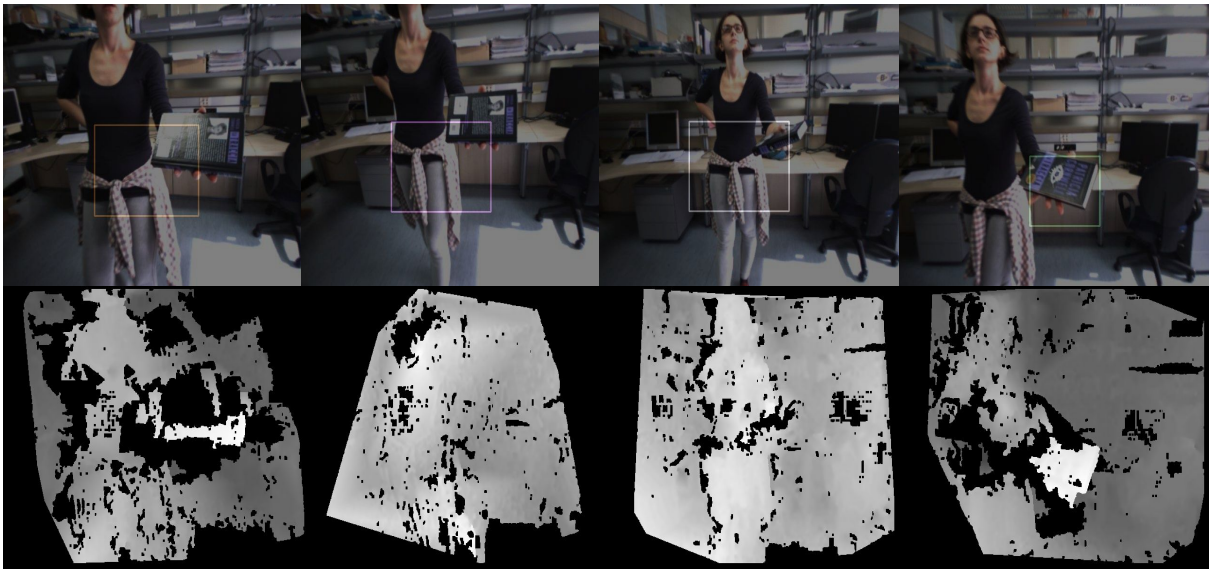

Figura 18. Malas detecciones debido a su respectivo disparity map. Primera fila muestra las detecciones y la segunda sus disparity maps.

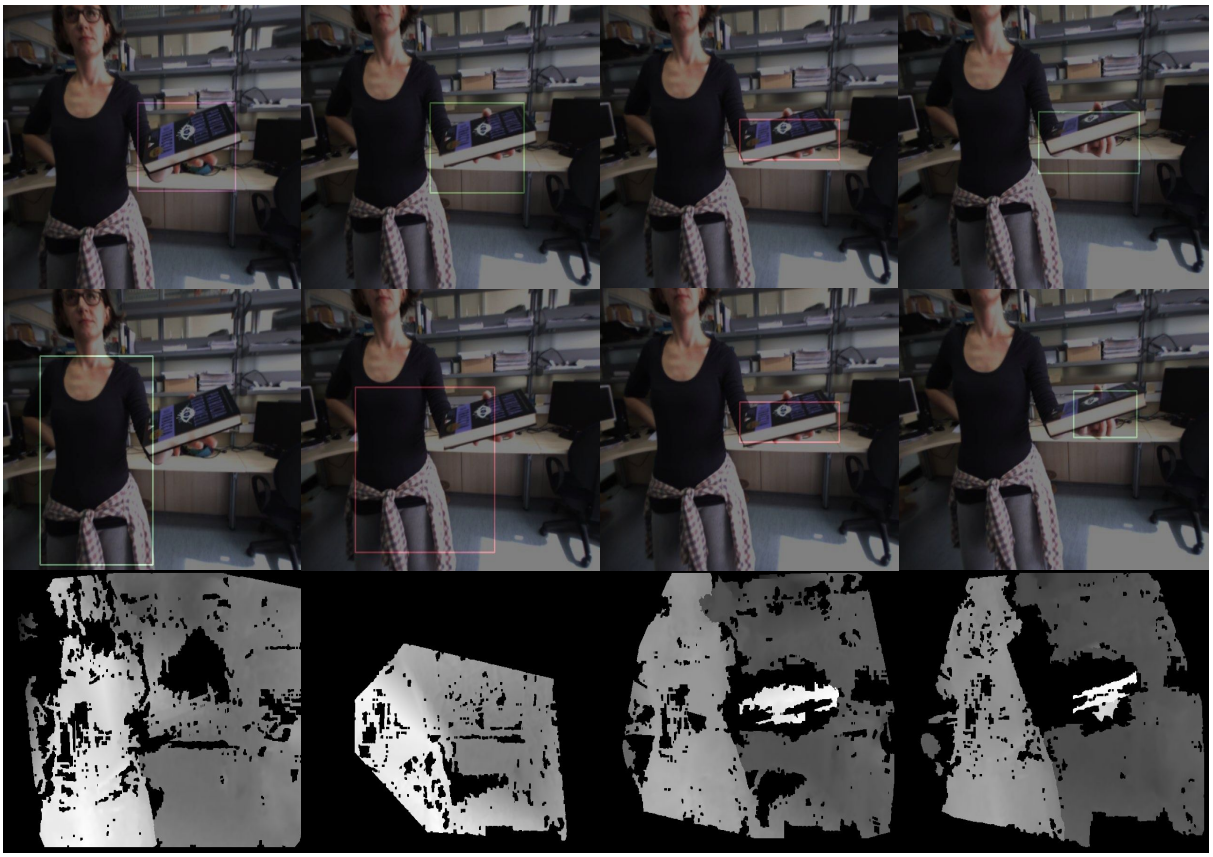

Figura 19. Funcionamiento de buffers. Primera fila con su aplicación, la segunda sin ésta y la tercera sus respectivos disparity maps.

Luego en figura 19 se observa el buen funcionamiento de los buffers, para las primeras dos imágenes se utiliza el buffer con las tres imágenes anteriores dado que el actualRect encontrado tiene un área mucho mayor con respecto a los anteriores, en la tercera imagen el método funciona correctamente gracias al buen disparity map mientras que en la última imagen se utiliza el buffer con la imagen actual y las tres anteriores debido a que su área es menor a la deseada permitiendo adecuar su tamaño.

A modo de descartar algunos disparity maps malos se calculará la desviación estándar para cada disparity map y se mantendrán aquellos bajo cierto umbral (este paso es opcional pero recomendable)[13]. La desviación estándar resulta un buen caracterizador de malos disparity maps, sin embargo, no es absoluta, ver figura 20.

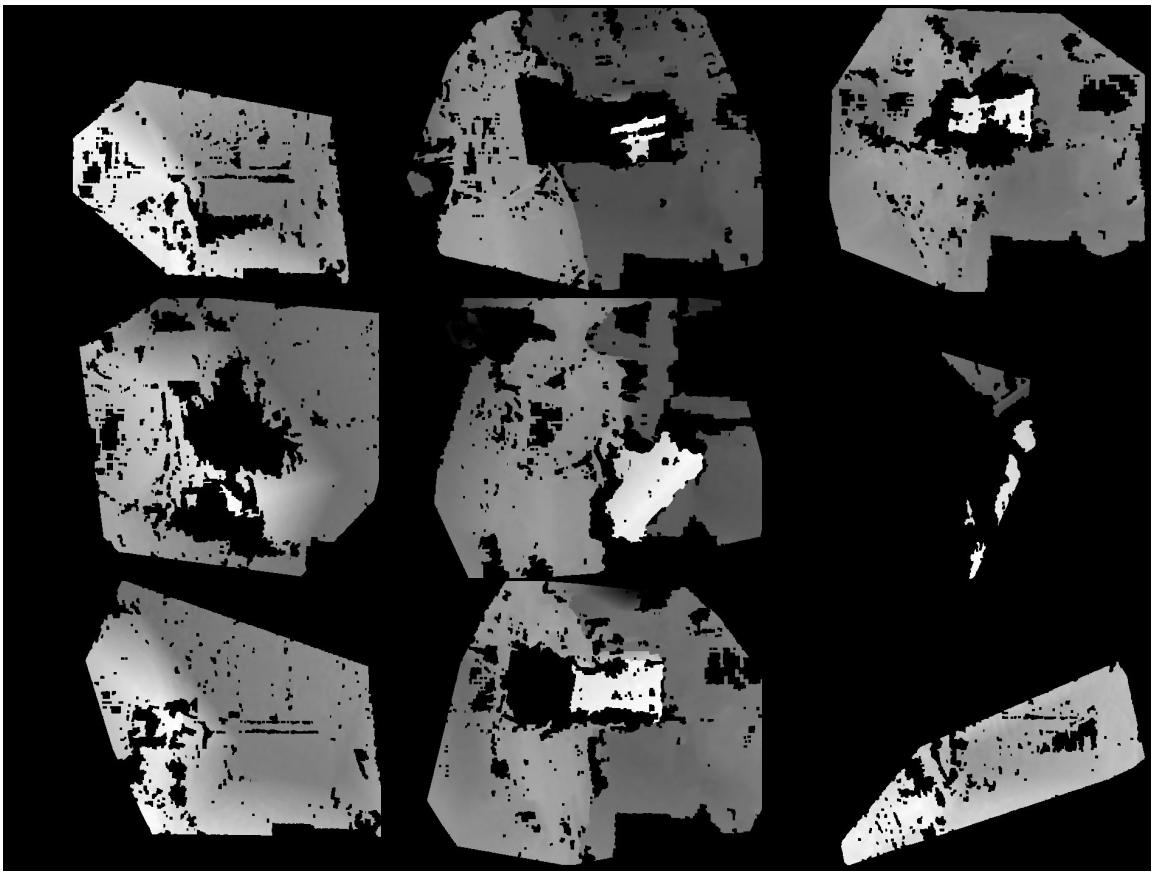

Figura 20. Algunos disparity maps eliminados tras aplicar umbral por desviación estándar.

Finalmente se añade el modo prueba (testMode) que permite visualizar de mejor manera lo que está sucediendo en cada frame a segmentar permitiendo la elección de parámetros para cada objeto[14]. En figura 21 se muestran todos los posibles contornos que pueden ser elegido en blanco y el selectedRect en color, además se visualizan los centroides (cruces) pertenecientes al selectedRect actual, previo y predicho usando flujo óptico, junto con esto se añaden círculos que indican el seguimiento utilizando un filtro de

[13] Véase en selectDisparitys.cpp

[14] Destacar que el tamaño del objeto es distinto entre pruebas por lo que modificar el número de buffers consecutivos, el umbral para seleccionar actualRect y los umbrales de comparación de IOU juegan un rol importantisimo a la hora de elección de selectedRect.

Kalman, por otra parte, en consola se pueden ver datos importantes que permiten la elección de parámetros. Junto con lo anterior también se visualizan los puntos de interés encontrados mediante el análisis de movimiento explicado más adelante.

Figura 21. Visualización en modo de prueba.

Por lo tanto, si existe una región en blanco que describa mejor el objeto que la región en color (selectedRect) modificar los umbrales. En cambio, si ninguna región es buena pero el análisis por movimiento tiene buenos resultados cambiar el número de buffers consecutivos. Por lo general, basta realizar este proceso para un objeto nuevo o que tenga distinto tamaño, una vez hecho esto se puede segmentar toda la base de imágenes de manera más robusta.

Seguimiento por movimiento

El primer paso es encontrar puntos de interés para cada frame, para ello se utiliza el detector Shi-Tomasi, luego a partir del segundo frame es posible calcular su flujo óptico entre el la imagen actual y la previa. Con esto es posible calcular la matriz de covarianza Σ y su esperanza media μ para cada flujo óptico repitiendo este procedimiento y guardando sus valores en todo el conjunto de entrenamiento.

Una vez obtenido el conjunto de entrenamiento es posible entrenar los 5 regresores lo que permite calcular la matriz de covarianza Σ' y su esperanza media μ' predicha. Posteriormente para cada nuevo frame del conjunto de prueba se obtiene el flujo óptico y sus estadísticas que serán usadas como entrada en la predicción de los regresores permitiendo ajustar los puntos respecto del movimiento general. Dadas las predicciones de los regresores es posible calcular la distancia Mahalanobis para cada uno permitiendo descartar aquellos bajo cierto umbral[15], ver figura 21. A partir de lo anterior es posible dibujar la región de interés final, ver figura 22 y 23.

Figura 21. Selección de puntos de interés usando métrica Mahalanobis. En primera columna se observan todos los puntos encontrados y en segunda aquellos seleccionados usando la métrica.

[15] Es de suma importancia modificar este umbral para que el método se ajuste al tamaño del objeto, caso contrario el método seleccionará puntos lejanos a éste. Para objetos pequeños un umbral cercano a 0.5 funciona correctamente.

Figura 22. Segmentación de libro.

Figura 23. Segmentación de celular.

En figuras 22 y 23 se observa la segmentación de un libro y un celular donde el método funciona, sin embargo, su precisión no es excelente dado que el detector se confunde escogiendo puntos no sólo del objeto en movimiento, sino además del profesor. Por otra parte, el método falla ante movimientos muy bruscos del profesor, por lo que es altamente recomendado usar este método con movimientos en 2 y 3 dimensiones. En otras palabras, el método fallará si se intenta trabajar con movimiento de escala o translaciones (lo cual funciona perfectamente para la detección en profundidad) debido a la sensibilidad del movimiento.

Conclusiones

A lo largo del proyecto se pudo observar la segmentación de imágenes a partir de dos enfoques distintos, mediante análisis de profundidad y de movimiento.

El objetivo de la experiencia se logra tras segmentar correctamente las imágenes tomando las debidas restricciones. En el análisis profundidad es necesario que las primeras 3 imágenes sean lo suficientemente buenas y en RGB que el profesor tenga el menor movimiento con respecto del objeto. Sin embargo, los métodos no son absolutos dada la dificultad del problema, con respecto al seguimiento por profundidad se propone un modo de prueba a fin de ajustar debidamente los parámetros.

Se puede verificar que el método más robusto es utilizando la profundidad del objeto, sin embargo, utilizando sólo este enfoque los resultados no son lo suficientemente bueno por lo que se añadió el enfoque de movimiento a fin de corregir estos errores.

Junto a lo anterior cabe destacar el rápido procesamiento que estos métodos por lo que es viable realizarlo en tiempo real aunque dado que el dataset se necesita para un post-entrenamiento se recomienda realizar el proceso offline.

La gran dificultad del proyecto mencionada previamente (diferenciación entre profesor y objeto) se vio reflejada en los resultados, sin embargo, se lograron aplacar obteniendo regiones razonables.

A fin de mejorar el método de RGB se propone utilizar background deletion y comparar ambos métodos. Un enfoque totalmente distinto es a partir de imágenes RGB aproximar las distancias con un enfoque de sampling no paramétrico usando SIFT Flow [6] o a partir de una red convolucional calcular el *disparity map* [7] con lo que a partir de ellas es posible utilizar el mismo enfoque de profundidad.

Anexos

Se adjunta ENTREGABLES.zip donde observará una serie de Readmes.txt que explican cómo utilizar los paquetes. Para mayor información ver códigos.

Seguimiento por profundidad

Véase carpeta example para realizar proceso completo.

Important READMES:

1. ./ENTREGABLES/analisis profundidad/LIBELAS/libelas/README.TXT: use "cmake .", "make" and then the elas command (see 3)
2. ./ENTREGABLES/analisis profundidad/LIBELAS/libelas/README ELAS COMMAND.txt: the elas command
3. ./ENTREGABLES/analisis profundidad/LIBELAS/libelas/methods/README.txt: Explanation of the parameters you can change

Parameters that you need to know and maybe change (in methods folder):

imagesToPgm.cpp:

- path_in_left: the path of the left images in jpg
- path_in_right: the path of the right images in jpg
- path_out_left: the path to save the left images in pgm
- path_out_right: the path to save the right images in pgm

selectDisparitys.cpp:

- path_in: the path of the disparity images
- path_out: the path to save the filtered disparity images
- stdThresh: the threshold to select disparitys by standard desviation

segmentationImgMix.cpp:

- path_in: the path of the disparity images
- path_out: the path to save the segmentation images
- path_real: the path of jpg images
- testMode: if true enter in the test mode.
- threshMahalanobis: the threshold to discard some points in the motion analysis
- n_buffers: number of max consecutive buffers.
- threshActualRectArea: threshold in selectMethod used to set the min area of the actualRect
- threshCompIOU4Rects: threshold used to set the min of the comparison between actualRect and the three previous
- threshCompIOU2Rects: threshold used to set the min of the comparison between actualRect and the previous one

Seguimiento por movimiento

Parameters that you need to know and maybe change:

pointsExtraction2.cpp:

- path_in: the path of the images
- path_out: the path to save the segmentation images
- threshMahalanobis: the threshold to discard some points in the motion analysis
- numberTrainImages: the fists images used to train the 5 regressors

Referencias

[1] Fanello, S. R., Ciliberto, C., Natale, L., & Metta, G. (2013, May). Weakly supervised strategies for natural object recognition in robotics. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on* (pp. 4223-4229). IEEE.

[2] Pasquale, G., Mar, T., Ciliberto, C., Rosasco, L., & Natale, L. (2016). Enabling depth-driven visual attention on the iCub humanoid robot: instructions for use and new perspectives. *Frontiers in Robotics and AI*, *3*, 35.

[3] iCubWorld. (2018). Retrieved from https://robotology.github.io/iCubWorld/

[4] Autonomous Vision Group | MPI for Intelligent Systems. (2018). Retrieved from http://www.cvlibs.net/software/libelas/

[5] Satoshi Suzuki and others. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, 30(1):32–46, 1985.

[6] Karsch, K., Liu, C., & Kang, S. B. (2014). Depth transfer: Depth extraction from video using non-parametric sampling. *IEEE transactions on pattern analysis and machine intelligence*, *36*(11), 2144-2158.

[7] Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., & Navab, N. (2016, October). Deeper depth prediction with fully convolutional residual networks. In *3D Vision (3DV), 2016 Fourth International Conference on* (pp. 239-248). IEEE.