**INFORME AVANCE NAVAR QA**

1. **Objetivos principales:**

* Depuración de Código de navar QA removiendo las funciones relacionadas con el rastreo de la broca y función *Rasterize*.
* Verificación de funcionamiento después de la depuración.
* Optimización de correspondencia de áreas y reorden (basado en matlab).
* Reconocimiento simultaneo Joskstra
* Verificación de reconocimiento de estrellas.

1. **Descripción del proceso**
2. **Depuración de código removiendo funciones relacionadas con el rastreo de broca y verificación de funcionamiento después de la depuración.**

**Objetivo:**

Remover funciones de la broca que no están siendo usadas y función de Rasterize() para mejorar rendimiento.

**Procedimiento:**

* Remover funcion Rasterize en GUIupdater.cpp que permite ver las imágenes de las cámaras por cada frame pero que no es necesaria dicha visualización.
* Remover booleanos cómo detect\_broca en detectbr.h que no se usan
* Remover matrices inicializadas para llenarlas con datos de broca en GUIUpdater

**Evaluación:**

Se requiere evaluar el correcto funcionamiento del sistema una vez se realizan estas dos modificaciones teniendo en cuenta las funciones más importantes y el valor de las variables que deben retornar, considerando los siguientes parámetros de evaluación:

1. *Función:* Nombre de las funciones a evaluar.
2. *Variable:* Variables de las funciones a evaluar.
   1. PP1: Matriz en pixeles de cámara derecha.
   2. PP2: Matriz en pixeles de cámara izquierda.
   3. P1: Matriz en pixeles sin esfera pequeña de la cámara derecha.
   4. P2: Matriz en pixeles sin esfera pequeña de la cámara izquierda.
   5. XL: Matriz de posiciones 3D triangulando con las dos cámaras.
   6. bRigid.Name: Nombre del Objeto rígido detectado.
3. *Retorno*: Retorno que realiza el sistema de las variables.
4. *Evaluación:* verificación de retorno esperado o inesperado.

**Resultados:**

1. Evaluación de verificación de correcto funcionamiento reconociendo un objeto rígido:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Función** | ***GetRigidsData*** | ***GetRigidsData*** | ***Correspondence*** | ***Correspondence*** | ***Stereo\_triangulation*** | ***Joskstra*** |
| **Variable** | PP1 | PP2 | P1 | P2 | XL | bRigid.Name |
| **Retorno** | (2,5) | (2,5) | (2,4) | (2,4) | (3,4) | POINTER |
| **Evaluación** | Checkmark | Checkmark | Checkmark | Checkmark | Checkmark | Checkmark |

1. Evaluación de verificación de correcto funcionamiento cambiando el objeto:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Función** | ***GetRigidsData*** | ***GetRigidsData*** | ***Correspondence*** | ***Correspondence*** | ***Stereo\_triangulation*** | ***Joskstra*** |
| **Variable** | PP1 | PP2 | P1 | P2 | XL | bRigid.Name |
| **Retorno** | (2,5) | (2,5) | (2,4) | (2,4) | (3,4) | FEMUR |
| **Evaluación** | Checkmark | Checkmark | Checkmark | Checkmark | Checkmark | Checkmark |

**Conclusión:**

Remover la función Rasterize() y funciones relacionadas con el reconocimiento de la broca no afectó el correcto reconocimiento de objetos y la ejecución del sistema.

1. **Optimización de función de áreas.**

**Objetivo:**

Agregar funcionalidades no consideradas en la version anterior.

**Procedimiento:**

Se añaden funcionalidades no consideradas en una versión anterior pero que se necesitan cómo:

* Verificar si el el contador de objetos retorna un multiplo de 5 para que solo ingresen 5 esferas.
* Empaquetar y ordenar los datos de menor a mayor.
* Obtener las esferas pequeñas que se deben obtener según el numero de estrellas.
* Dividir datos entre esferas pequeñas y esferas grandes.
* Evaluar la posición de las esferas pequeñas encontradas en el paquete de datos original.
* Enviar posiciones de esferas pequeñas.

**Resultados:**

1. **Version anterior:**
2. /\*\*
3. \*       Compara las áreas y detecta la posición de las esferas más pequeñas (esferas de referencia en cada cuerpo rígido) en el conjunto de esferas.
4. \*       @param A referencia al conjunto de áreas.
5. \*       @param areas\_men matriz que será llenada con las ubicaciones de las áreas más pequeñas encontradas
6. \*       @return entero que indica el estado de la búsqueda de éstas esferas de referencia.
7. \*/
8. int areas(Mat\_<double> &A, Mat\_<int> &areas\_men) {
9. //              cout << A;
10. bool state =  **false**;
11. **if** (!detect\_broca && A.rows % 5 == 0 && A.rows <= MAX\_ESFERA) {
12. int x=0;
13. **for** (int i = 0; i < A.rows; i++) {
14. **if** (A(i, 0) < AREA) {
15. **if** (x < A.rows / 5) {
16. areas\_men.push\_back(i);         // guardar posición de las áreas más pequeñas.
17. state = **true**;
18. }
19. **else** {
20. **cout** << "ERROR: Encontradas más esferas pequeñas >> correspondence.h" << endl;
21. state = **false**;
22. }
23. x++;
24. }
25. }
26. }
27. **else** **if** (detect\_broca) {
28. int x = 0;
29. **for** (int i = 0; i < A.rows; i++) {
30. **if** (A(i, 0) < AREA) {
31. **if** (x < A.rows / 5 + 1) {               // más una esfera, la del taladro
32. areas\_men.push\_back(i);         // guardar posición de las áreas más pequeñas.
33. state = **true**;
34. **cout** << A.row(i) << endl;
35. }
36. **else** {
37. **cout** << "ERROR: No se logra detectar el indicador de la broca >> correspondence.h" << endl;
38. state = **false**;
39. }
40. x++;
41. }
42. }
43. }
44. **return** state;
45. }
47. **Función modificada:**
48. /\*\*
49. \*       Compara las áreas y detecta la posición de las esferas más pequeñas (esferas de referencia en cada cuerpo rígido) en el conjunto de esferas.
50. \*       @param A referencia al conjunto de áreas.
51. \*       @param areas\_men matriz que será llenada con las ubicaciones de las áreas más pequeñas encontradas
52. \*       @return entero que indica el estado de la búsqueda de estas esferas de referencia.
53. \*/
54. int areas(Mat\_<double> &A, Mat\_<int> &areas\_men)
55. {
56. Mat\_<double> Arreglo;
57. Mat\_<double> Evaluate;
58. Mat\_<double> Promedio,Menores,rnd,Menores2,Mayores;
59. int flag = 0;
60. bool state =  **false**;
61. ofstream Areas;
63. **if** (A.rows <= MAX\_ESFERA) {
64. int s = A.rows; //Extrae numero de objetos
65. Arreglo = A;
66. **if** (s % 5 == 0)    //Verifica que sí es multiplo de 5 que corresponde a una estrella
67. {
68. cv::sort(Arreglo, Evaluate, 1); //Ordenamiento de areas de menor a mayor
70. **for** (int i = 0; i < s; i++) {   //Evaluar hasta el número de objetos
71. **if** (s / 5 == i) {  // Evaluar la cantidad de esferas pequeñas que deberian haber
72. Evaluate(Range(0, i), Range(0, Evaluate.cols)).copyTo(Menores); // Sacar areas menores
73. Evaluate(Range(i, Evaluate.rows), Range(0, Evaluate.cols)).copyTo(Mayores); // Sacar areas Mayores
75. **for** (int j = 0; j < Arreglo.rows; j++) {//Busqueda en el arreglo original las i esferas.
76. **for** (int y = 0; y < i; y++) {
77. **if** (Arreglo(j, 0) == Evaluate(y, 0)) { //Busqueda de esferas pequeñas
79. **if** (flag <= i && areas\_men.rows < i) {
80. areas\_men.push\_back(j); //Envío de posición de esfera pequeña
81. state = **true**; //Envío de estado
82. flag++;
83. }
84. }
85. }
86. }
87. }
88. }
89. }
90. **else** {
91. **cout** << "ERROR: Encontradas mas esferas pequeñas >> correspondence.h" << endl;
92. **system**("PAUSE");
93. state = **false**;
94. }
95. }
96. **return** state;
97. }
99. **Optimización de funcion de Reorden.**

**Objetivo:**

Mejorar el rendimiento de la ejecución reemplazando busqueda de elementos por posiciones a ángulos.

**Procedimiento:**

Realizar la conversión o traducción de funciones de codigo en matlab a C++ y Opencv.

1. Código antiguo:
2. /\*\*
3. \* Reordena un conjunto de esferas de tal manera que la correspondencia se cumpla. Las ordena siguiendo sentido horario y partiendo de la esfera más pequeña
4. \* que tienen los marcadores.
5. \* @param S una referencia al conjunto de esferas que van a ser ordenadas.
6. \*/
7. void reorden(Mat\_<double> &S) {
8. Point2d ref = S.col(0);
9. Mat\_<double> N = S.colRange(1, S.cols), tmp, L, R, U, D;                                        // sacar la 1era columna
10. double minVal;
11. double maxVal;
12. Point minLoc;
13. Point maxLoc;
14. int up = 0, down = 0, left = 0, right = 0;// = 0;
15. int x = 0;
17. tmp = S.col(0);                                 // alamcenar la 1ra esfera identificada
19. minMaxLoc(N.row(1), &minVal, &maxVal, &minLoc, &maxLoc);
21. **if** (minVal < ref.y)     { // hay arriba?
22. up = 1;
23. x = 0;
24. **for** (int i = 0; i < N.cols; i++) { // contar cuántas esfera hay arriba
25. **if** (ref.y > N(1, i)) {
26. **if** (x == 0)
27. N.col(i).copyTo(U);
28. **else** hconcat(U, N.col(i), U);
29. ++x;
30. }
31. }
32. //                      cout << "hay arriba: " << x << endl;// << U << endl;
34. **if** (maxVal > ref.y) { // hay abajo?
35. down = 2;
36. x = 0;
37. **for** (int i = 0; i < N.cols; i++) { // contar cuántas esfera hay arriba
38. **if** (ref.y < N(1, i)) {
39. **if** (x == 0)
40. N.col(i).copyTo(D);
41. **else** hconcat(D, N.col(i), D);
42. ++x;
43. }
44. }
45. //                              cout << "hay abajo: " << x << endl;// << D << endl;
46. }
47. }
48. **else** {
49. down = 2;
50. x = 0;
51. **for** (int i = 0; i < N.cols; i++) { // contar cuántas esfera hay arriba
52. **if** (ref.y < N(1, i)) {
53. **if** (x == 0)
54. N.col(i).copyTo(D);
55. **else** hconcat(D, N.col(i), D);
56. ++x;
57. }
58. }
59. //                      cout << "hay abajo: " << x << endl;// << D << endl;
60. }
62. minMaxLoc(N.row(0), &minVal, &maxVal, &minLoc, &maxLoc);
64. **if** (maxVal > ref.x) { // hay a la derecha?
65. right = 3;
66. x = 0;
67. **for** (int i = 0; i < N.cols; i++) { // contar cuántas esfera hay a la derecha
68. **if** (ref.x < N(0, i)) {
69. **if** (x == 0)
70. N.col(i).copyTo(R);
71. **else** hconcat(R, N.col(i), R);
72. ++x;
73. }
74. }
75. //                      cout << "hay derecha: " << x << endl;// << R << endl;
77. **if** (minVal < ref.x) { // hay a la izquierda?
78. left = 4;
79. x = 0;
80. **for** (int i = 0; i < N.cols; i++) { // contar cuántas esfera hay a la izquierda
81. **if** (ref.x > N(0, i)) {
82. **if** (x == 0)
83. N.col(i).copyTo(L);
84. **else** hconcat(L, N.col(i), L);
85. ++x;
86. }
87. }
88. //                              cout << "hay izquierda: " << x << endl;// << L << endl;
89. }
90. }
91. **else** {
92. left = 4;
93. x = 0;
94. **for** (int i = 0; i < N.cols; i++) { // contar cuántas esfera hay a la izquierda
95. **if** (ref.x > N(0, i)) {
96. **if** (x == 0)
97. N.col(i).copyTo(L);
98. **else** hconcat(L, N.col(i), L);
99. ++x;
100. }
101. }
102. //                      cout << "hay izquierda: " << x << endl;// << L << endl;
103. }
104. //              cout << endl << up << down << right << left << endl;
106. //              cout << endl << "\*\*\*\*\*\*\*\*\*\*\*" << endl;
108. // si la refrencia esta toalmente a la izq o a la derecha entonces simpre hay esferas a arriba o abajo
109. **if** (up == 0 || down == 0) { // la referencia esta totalmente arriba o abajo
110. **if** (up == 0) { // la referencia totalmente esta arriba
111. **if** (R.cols < 3 && R.cols != 0 && L.cols != 0) {
112. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la der está más abajo?
113. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
114. hconcat(tmp, R.col(minLoc.x), tmp);
115. **if** (R.cols > 1)
116. hconcat(tmp, R.col(maxLoc.x), tmp);
117. // encontramos las otras posiciones que faltan.
118. minMaxLoc(L.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la izq está más abajo?
119. //                                      cout << "El siguiente es: " << endl << L.col(maxLoc.x) << endl;
120. hconcat(tmp, L.col(maxLoc.x), tmp);
121. **if** (L.cols > 1)
122. hconcat(tmp, L.col(minLoc.x), tmp);
123. tmp.copyTo(S);
124. }
125. **else** **if** (R.cols == 3) {
126. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la izq está más abajo?
127. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
128. hconcat(tmp, R.col(minLoc.x), tmp);
129. R.row(1).at<double>(minLoc) = 10000;            // cambiar el valor de la coord que indica el corrimiento a la izq
130. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de arriba está más a la izq?
131. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
132. hconcat(tmp, R.col(minLoc.x), tmp);
133. R.row(1).at<double>(minLoc) = 15000;            // cambiar el valor de la coord que indica el corrimiento a la izq
134. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál es el menor de todos?
135. hconcat(tmp, R.col(minLoc.x), tmp);
136. tmp.copyTo(S);
137. //                                      cout << tmp << endl;
138. }
140. }
141. **else** **if** (down == 0) { // la referencia esta totalmente abajo
142. **if** (L.cols < 3 && L.cols != 0 && R.cols != 0) {
143. // encontramos las otras posiciones que faltan.
144. minMaxLoc(L.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la izq está más abajo?
145. //                                      cout << "El siguiente es: " << endl << L.col(maxLoc.x) << endl;
146. hconcat(tmp, L.col(maxLoc.x), tmp);
147. **if** (L.cols > 1)
148. hconcat(tmp, L.col(minLoc.x), tmp);
149. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la der está más abajo?
150. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
151. hconcat(tmp, R.col(minLoc.x), tmp);
152. **if** (R.cols > 1)
153. hconcat(tmp, R.col(maxLoc.x), tmp);
154. tmp.copyTo(S);
155. }
156. **else** **if** (L.cols == 3) {
157. minMaxLoc(L.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la izq está más abajo?
158. //                                      cout << "El siguiente es: " << endl << L.col(maxLoc.x) << endl;
159. hconcat(tmp, L.col(maxLoc.x), tmp);
160. L.row(0).at<double>(maxLoc) = 10000;            // cambiar el valor de la coord que indica el corrimiento a la izq, se tira a la izq
161. minMaxLoc(L.row(0), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de arriba está más a la izq?
162. //                                      cout << "El siguiente es: " << endl << L.col(minLoc.x) << endl;
163. hconcat(tmp, L.col(minLoc.x), tmp);
164. L.row(0).at<double>(minLoc) = 1500;             // cambiar el valor de la coord que indica el corrimiento a la izq
165. minMaxLoc(L.row(0), &minVal, &maxVal, &minLoc, &maxLoc); // cuál es el menor de todos?
166. hconcat(tmp, L.col(minLoc.x), tmp);
167. tmp.copyTo(S);
168. //                                      cout << tmp << endl;
169. }
170. **else** **if** (R.cols == 3) {
171. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la der está más abajo?
172. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
173. hconcat(tmp, R.col(minLoc.x), tmp);
174. R.row(1).at<double>(maxLoc) = 10000;            // cambiar el valor de la coord que indica el corrimiento a la izq
175. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de de la der está más abajo?
176. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
177. hconcat(tmp, R.col(minLoc.x), tmp);
178. R.row(1).at<double>(minLoc) = 15000;            // cambiar el valor de la coord que indica el corrimiento a la izq
179. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál es el menor de todos?
180. hconcat(tmp, R.col(minLoc.x), tmp);
181. tmp.copyTo(S);
182. //                                      cout << tmp << endl;
183. }
184. }
185. }
186. **else** **if** (left == 0 || right == 0) { // la referencia esta totalmente a la izq o a la derecha
187. **if** (left == 0) { // la referencia totalmente esta a la izq
188. minMaxLoc(U.row(0), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de la arriba está más a la izq?
189. //                              cout << "El siguiente es: " << endl << U.col(minLoc.x) << endl;
190. hconcat(tmp, U.col(minLoc.x), tmp);
191. **if** (U.cols > 1)
192. hconcat(tmp, U.col(maxLoc.x), tmp);
193. // encontramos las otras posiciones que faltan.
194. minMaxLoc(D.row(0), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de abajo está más a la der?
195. //                              cout << "El siguiente es: " << endl << D.col(maxLoc.x) << endl;
196. hconcat(tmp, D.col(maxLoc.x), tmp);
197. **if** (D.cols > 1)
198. hconcat(tmp, D.col(minLoc.x), tmp);
199. tmp.copyTo(S);
200. }
201. **else** { // la referencia esta totalmente a la der
202. minMaxLoc(D.row(0), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de abajo está más a la der?
203. //                              cout << "El siguiente es: " << endl << D.col(maxLoc.x) << endl;
204. hconcat(tmp, D.col(maxLoc.x), tmp);
205. **if** (D.cols > 1)
206. hconcat(tmp, D.col(minLoc.x), tmp);
207. // encontramos las otras posiciones que faltan.
208. minMaxLoc(U.row(0), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de las esferas de arriba está más a la izq?
209. //                              cout << "El siguiente es: " << endl << U.col(minLoc.x) << endl;
210. hconcat(tmp, U.col(minLoc.x), tmp);
211. **if** (U.cols > 1)
212. hconcat(tmp, U.col(maxLoc.x), tmp);
213. tmp.copyTo(S);
214. **cout** << tmp << endl << endl << S << endl << endl;
216. }
217. }
218. **else** {
219. **if** (R.cols == 1) {
220. **if** (ref.y < R(1,0))     { // si la referencia esta más arriba que la que esta a la der
221. hconcat(tmp, R, tmp);
222. minMaxLoc(L.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de las esferas de la izq está más abajo?
223. //                                      cout << "El siguiente es: " << endl << L.col(maxLoc.x) << endl;
224. hconcat(tmp, L.col(maxLoc.x), tmp);
225. **if** (L.cols > 1)
226. hconcat(tmp, L.col(minLoc.x), tmp);
227. tmp.copyTo(S);
228. }
229. **else** **if** (ref.y > R(1, 0)) { // si la referencia esta más abajo que la que esta a la der
230. minMaxLoc(L.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de las esferas de la izq está más abajo?
231. //                                      cout << "El siguiente es: " << endl << L.col(maxLoc.x) << endl;
232. hconcat(tmp, L.col(maxLoc.x), tmp);
233. **if** (L.cols > 1)
234. hconcat(tmp, L.col(minLoc.x), tmp);
235. hconcat(tmp, R, tmp);
236. tmp.copyTo(S);
237. }
238. }
239. **else** **if** (L.cols == 1) {
240. **if** (ref.y < L(1, 0)) { // si la referencia esta más arriba que la que esta a la izq
241. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de las esferas de la der está más abajo?
242. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
243. hconcat(tmp, R.col(minLoc.x), tmp);
244. **if** (R.cols > 1)
245. hconcat(tmp, R.col(maxLoc.x), tmp);
246. hconcat(tmp, L, tmp);
247. tmp.copyTo(S);
248. }
249. **else** **if** (ref.y > L(1, 0)) { // si la referencia esta más abajo que la que esta a la izq
250. hconcat(tmp, L, tmp);
251. minMaxLoc(R.row(1), &minVal, &maxVal, &minLoc, &maxLoc); // cuál de las esferas de la der está más abajo?
252. //                                      cout << "El siguiente es: " << endl << R.col(minLoc.x) << endl;
253. hconcat(tmp, R.col(minLoc.x), tmp);
254. **if** (R.cols > 1)
255. hconcat(tmp, R.col(maxLoc.x), tmp);
256. tmp.copyTo(S);
257. }
258. }
259. }
261. //              cout << S << endl << endl << N << endl;
262. }
263. **Codigo en Matlab:**
264. **function** PP=Ordenamiento\_Pointer(centroids,[**Area**](https://www.mathworks.com/help/matlab/ref/area.html))
265. %FunciÃ³n que permite ordenar las esferas en sentido horario visto desde la
266. %cÃ¡mara.
267. % centroids (matriz con datos de coordenadas x,y de las esferas) : Nx2
268. %   columna 1: coordenads x
269. %   columna 2: coordenads y
270. % Area (vector con el Ã¡rea de las esferas) : Nx1
271. % SE ASUME N=5 (5 ESFERAS)
273. % La salida es una matriz PP de 4 filas 2 columnas. De acuerdo con NavarQA la Ãºltima esfera
274. % debe ser la esfera mÃ¡s cercana a la esfera pequeÃ±a.
276. %CÃ¡lculo del centroide
277. cm=[**mean**](https://www.mathworks.com/help/matlab/ref/mean.html)(centroids);
278. %indice de la esfera pequeÃ±a
279. [~,ind]=[**min**](https://www.mathworks.com/help/matlab/ref/min.html)(**[Area](https://www.mathworks.com/help/matlab/ref/area.html)**);
280. X=centroids;
281. %Distancia de las esferas a la esfera pequeÃ±a incluida la pequeÃ±a
282. d=[**sqrt**](https://www.mathworks.com/help/matlab/ref/sqrt.html)([**sum**](https://www.mathworks.com/help/matlab/ref/sum.html)((X-**[repmat](https://www.mathworks.com/help/matlab/ref/repmat.html)**(X(ind,:),5,1)).^2,2));
283. [~,val]=**[sort](https://www.mathworks.com/help/matlab/ref/sort.html)**(d);
284. %ExtracciÃ³n de las 4 esferas (se excluye la pequeÃ±a). En la nueva matriz de
285. %centros de esferas hay 4 esferas y la primera es la mÃ¡s cercana a la
286. %esfera pequeÃ±a.
287. X=X(val(2:**end**),:);
288. %Desplazamiento al centroide
289. DX=X-**[repmat](https://www.mathworks.com/help/matlab/ref/repmat.html)**(cm,4,1);
290. %Vectores unitarios a cada esfera
291. U=DX./**[repmat](https://www.mathworks.com/help/matlab/ref/repmat.html)**([**sqrt**](https://www.mathworks.com/help/matlab/ref/sqrt.html)([**sum**](https://www.mathworks.com/help/matlab/ref/sum.html)(DX.^2,2)),1,2);
292. %DefiniciÃ³n de un sistema coordenado x,y de tal mamnera que el vector
293. %unitario de la primera esfera es el eje x; el eje z es el perpendicular a
294. %las esferas (que solo tienen coordenadas x,y en pixeles) y el eje y es el
295. %producto cruz.
296. ux=U(1,:);
297. uy=**[cross](https://www.mathworks.com/help/matlab/ref/cross.html)**([0,0,1],[ux,0]);
298. uy=uy(1:2);
299. %CÃ¡lculo de las proyecciones a los vectores unitarios x,y,z
300. Compx=[**sum**](https://www.mathworks.com/help/matlab/ref/sum.html)(U.\***[repmat](https://www.mathworks.com/help/matlab/ref/repmat.html)**(ux,4,1),2);
301. Compy=[**sum**](https://www.mathworks.com/help/matlab/ref/sum.html)(U.\***[repmat](https://www.mathworks.com/help/matlab/ref/repmat.html)**(uy,4,1),2);
302. %CÃ¡lculo de los Ã¡ngulos con el eje x en el rango de -pi a pi
303. Theta=atan2d(Compy,Compx);
304. %correcciÃ³n de Ã¡ngulos negativos
305. Theta(Theta<0)=360+Theta(Theta<0);
306. %OrganizaciÃ³n en sentido horario de los angulos
307. [~,Ind]=**[sort](https://www.mathworks.com/help/matlab/ref/sort.html)**(Theta);
308. %ReasignaciÃ³n de centroides
309. PP=[X(Ind(2),:),X(Ind(3),:),X(Ind(4),:),X(Ind(1),:)];
310. **Código resultante:**
311. \* Reordena un conjunto de esferas de tal manera que la correspondencia se cumpla. Las ordena siguiendo sentido horario y partiendo de la esfera más pequeña
312. \* que tienen los marcadores.
313. \* @param S una referencia al conjunto de esferas que van a ser ordenadas.
314. \* @param cm representa el promedio de los centroides.
315. \*/
316. void reorden(Mat\_<double> &S, Mat\_<double> cm) {
318. Mat\_<double> N, DX1, sum, raiz, rep, Theta, ORDENADO,DX, U, ux, uy,us, Compx, Compy;
319. Mat testIdx;
320. cv::Mat\_<double> unit(1,3);
321. unit[0][0]=0; unit[0][1]=0; unit[0][2]=1;
322. N=S.t();
324. /\*Desplazamiento al centroide.\*/
325. subtract(N, repeat(cm, 4, 1), DX1);
327. /\*Vectores unitarios a cada esfera.\*/
328. reduce(DX1.mul(DX1, 1), sum, 1, CV\_REDUCE\_SUM);
329. cv::**sqrt**(sum, raiz);
330. rep=cv::repeat(raiz, 1, 2);
331. cv::divide(DX1, rep, U);
333. /\*Definición de un sistema coordenado x,y de tal manera que el vector unitario de la primera esfera es el eje x; el eje z es el perpendicular a las esferas (que solo tienen x,y en pixeles) y el eje y es el producto cruz.\*/
334. U(Range(0, U.rows-3), Range(0, U.cols)).copyTo(ux);
335. cv::hconcat(ux,0,us);
336. uy = cross\_product(unit,us);
337. uy(Range(0, uy.rows), Range(0, uy.cols - 1)).copyTo(uy);
339. /\*Calculo de proyecciones a los vectores unitarios x,y,z\*/
340. reduce(U.mul(cv::repeat(ux, 4, 1)),Compx,1,CV\_REDUCE\_SUM);
341. reduce(U.mul(cv::repeat(uy, 4, 1)), Compy, 1, CV\_REDUCE\_SUM);
343. /\*Calculo de los ángulos con el eje x en el rango de pi a -pi y corrección de ángulos negativos\*/
344. Theta = invTan2(Compy,Compx);
346. /\*Organización en sentido horario de los ángulos\*/
347. sortIdx(Theta, testIdx, CV\_SORT\_ASCENDING | CV\_SORT\_EVERY\_COLUMN);
349. /\*Reasignación de centroides.\*/
350. ORDENADO = OrderByIdx(N, testIdx, 4);   //Generar redoren by indexes
351. ORDENADO = ORDENADO.t();
352. ORDENADO.copyTo(S);
353. }

Función para calculo de tangente inverso a ángulos positivos:

1. /\*\*
2. \*       Obtiene el tangente inverso entre dos Mat\_
3. \*       @param y Primer Mat\_
4. \*       @param x Segundo Mat\_
5. \*       @return Tangente inverso entre x e y
6. \*/
7. cv::Mat invTan2(cv::Mat\_<double> y, cv::Mat\_<double> x) {
8. cv::Mat\_<double> Ainv;
9. double result;
11. **for** (int i = 0; i < y.rows; i++)
12. {
13. **for** (int j = 0; j < x.cols; j++)
14. {
15. result = **atan2**(y[i][j], x[i][j]) \* 180 / PI;
16. **if** (result<0)
17. {
18. result = result + 360;
19. }
20. Ainv.push\_back(result);
21. }
22. }
23. **return** Ainv;
24. }

**Conclusiones:**

* Se logra disminuir el tiempo de ejecución de 0.43 milisegundos a 0.135 milisegundos una vez entre a la función.
* Se realizan calculo de proyecciones a través de vectores unitarios.

1. **Reconocimiento simultaneo de estrellas:**

**Objetivo:**

Modificar el sistema de tal manera que pueda obtener información de estrellas de manera simultánea.

**Procedimiento:**

1. Ubicar dos objetos frente a las cámaras.
2. Modificar el sistema de tal manera que se pueda empaquetar los datos en posibles objetos rígidos y enviarlos a reconocimiento Joskstra de objetos para evaluar si estos corresponden a alguno de los objetos predefinidos en el sistema, así:

**Resultado:**

Enviar paquetes a reconocimiento

Datos

Paquete objeto rígido 1

Paquete objeto rígido 2

Función Reconocimiento Joskstra

Objeto 1 reconocido

Objeto 2 reconocido

Dividir paquetes

Obtener resultado

El siguiente es la parte del código que realiza lo requerido dentro de una función llamada GetRigidsData():

1. **if** (wResult == WAIT\_OBJECT\_0) { //Comprobar hilo de ejecucion
3. int limitStart = 0; //Inicializar paquete de datos
4. int limitEnd = 4; //Inicializar limite de datos del paquete
6. **if** (limitEnd<9 && XL.cols == 8)
7. {
8. XL = XL.t();
9. **for** (int i = 0; i < 2; i++)
10. {
11. XL(Range(limitStart, limitEnd), Range(0, XL.cols)).copyTo(XLClone); // Sacar los menores
12. CustomCameraLibrary::rigid = **new** CustomCameraLibrary::BodyR[cdata::distances.rows + 1];
13. CustomCameraLibrary::nbr = CustomCameraLibrary::joskstra(XLClone, cdata::distances, CustomCameraLibrary::rigid);
14. limitStart = limitStart + 4;
15. limitEnd = limitEnd + 4;
16. }
17. }
18. **else**
19. {
20. CustomCameraLibrary::rigid = **new** CustomCameraLibrary::BodyR[cdata::distances.rows + 1];
21. CustomCameraLibrary::nbr = CustomCameraLibrary::joskstra(XLClone, cdata::distances, CustomCameraLibrary::rigid);
22. }

**Conclusiones:**

Se pudo dividir el paquete de datos de entrada para agruparlo por objeto y obtener los objetos detectados en escena.

1. **Verificación de reconocimiento de estrellas simultaneo.**

**Objetivo:**

Ejecutar el sistema y verificar su correcto funcionamiento e identificación de los objetos rígidos.

**Procedimiento:**

* Ubicar dos objetos rígidos en la escena
* Ejecutar el programa
* Evaluar el correcto funcionamiento del sistema teniendo en cuenta las funciones más importantes y el valor de las variables que deben retornar, considerando los siguientes parámetros de evaluación:

1. *Función:* Nombre de las funciones a evaluar.
2. *Variable:* Variables de las funciones a evaluar.
   1. PP1: Matriz en pixeles de cámara derecha.
   2. PP2: Matriz en pixeles de cámara izquierda.
   3. P1: Matriz en pixeles sin esfera pequeña de la cámara derecha.
   4. P2: Matriz en pixeles sin esfera pequeña de la cámara izquierda.
   5. XL: Matriz de posiciones 3D triangulando con las dos cámaras.
   6. bRigid.Name: Nombre del Objeto rígido detectado.
3. *Retorno*: Retorno que realiza el sistema de las variables.
4. *Evaluación:* verificación de retorno esperado o inesperado.

**Resultados:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Función** | ***GetRigidsData*** | ***GetRigidsData*** | ***Correspondence*** | ***Correspondence*** | ***Stereo\_triangulation*** | ***Joskstra*** |
| **Variable** | PP1 | PP2 | P1 | P2 | XL | bRigid.Name |
| **Retorno** | (2,10) | (2,10) | (2,8) | (2,8) | (3,8) | POINTER  FEMUR |
| **Evaluación** | Checkmark | Checkmark | Checkmark | Checkmark | Checkmark | Checkmark |

**Conclusiones**

Se observa que se obtienen los objetos puestos en escena de manera satisfactoria.