

E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Percepción mediante radar para seguridad proactiva en máquinas y vehículos industriales



Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Autor: José Antonio García Martínez

Director: Ignacio Del Villar Fernández

Pamplona, 2 de Junio de 2023

upna

Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

Resumen:

La seguridad en entornos logísticos e industriales es de interés primordial ya que los atropellos en naves industriales pueden tener consecuencias de gravedad. Se hará un resumen del estado del arte de la tecnología *Millimeter Wave* (mmwave) y Ultra Wide Band (UWB) y se aplicará la sensórica radar de tecnología mmwave para la detección de todo tipo de objetos, que, junto con *Robot Operating System* (ROS) se visualizarán en un entorno 3D. Se probará el radar tanto en estático como en movimiento, donde se verán diferencias significativas en su comportamiento. También se estudiarán los parámetros del radar y se elegirá la configuración adecuada para cada caso, teniendo en cuenta las limitaciones de este. Además, se añadirá una cámara estereoscópica con una red neuronal implementada que permite la detección de múltiples objetos.

Abstract:

Safety in logistics and industrial environments is of paramount interest since accidents in industrial buildings can have serious consequences. A summary of the state of the art of Millimeter Wave (mmwave) and Ultra Wide Band (UWB) technology will be made and the mmwave technology radar sensor will be used to detect all types of objects, which, alongside Robot Operating System (ROS), will be visualized on a 3D environment. The radar will be tested both static and in motion, and there will be significant differences in its behavior. The radar parameters will also be studied and the appropriate configuration will be chosen for each case, taking into account its limitations. In addition, a stereoscopic camera will be added with an implemented neural network that allows the detection and distinction of multiple objects.

Palabras clave:

Mmwave – UWB – Chirp – Nube de puntos – 3D_People_Counting – Mobile_tracker – Targets – CFAR – RCS – TLV – ROS – Cámara Estereoscópica – C++

ÍNDICE

1	Introducción/Antecedentes	1
2	Objetivos	1
3	Desarrollo teórico de tecnologías radar	2
3.1	Tecnología mmwave	2
3.1.1	Fundamentos de la tecnología mmwave	2
3.1.2	Diferencias entre frecuencias de trabajo (24/60/77 GHz)	7
3.1.3	Interferencias entre radares mmwave.....	8
3.1.4	Aplicaciones de la tecnología mmWave.....	14
3.2	Tecnología Ultra-Wide Band (UWB).....	15
3.2.1	Fundamentos de la tecnología UWB.....	15
3.2.2	Aplicaciones de la tecnología UWB	18
3.3	Comparación teórica entre mmwave y UWB.....	20
4	Radares mmwave	21
4.1	Radares mmwave de Texas Instruments	21
4.1.1	Gama de radares	21
4.1.2	Diferencias entre radares industriales y de automoción (IWR e AWR)	23
4.1.3	Diferencias entre AoP e ISK.....	23
4.2	Radares mmwave de Seeed Studio	23
4.3	Comparativa teórica y elección del fabricante de radar	25
4.4	Radares mmWave elegidos	25
4.4.1	Radar uRad Industrial v1.0 de Anteral	25
4.4.2	Radar IWR6843ISK de Texas Instruments	26
5	Herramientas y configuración de los radares	27
5.1	Uniflash	27
5.1.1	Demo Out_of_the_box (OOB).....	31
5.1.2	Demo 3D_People_Counting	31
5.1.3	Demo Mobile_Tracker.....	31
5.2	Mmwave Demo Visualizer.....	32
5.3	Comandos de configuración de la demo Out_of_the_box	34
5.4	Efecto de los .cfg en los chirps y los parámetros de entorno	36
5.4.1	Radar Cross-Section (RCS).....	37
5.4.2	Signal to Noise Ratio (SNR).....	43
5.4.3	Noise Figure (NF).....	44

5.4.4	Cálculo analítico de parámetros.....	44
5.5	<i>Constant false alarm rate (CFAR)</i>	49
5.5.1	CFAR_CA.....	50
5.5.2	CFAR_CAGO y CFAR_CASO.....	51
5.5.3	Configuración de CFAR en los radares de Texas Instruments (OOB).....	51
5.6	Formato de los datos de salida: Type-Lenght-Value (TLV).....	52
6	Desarrollo práctico	57
6.1	Descripción e instalación de hardware y software	57
6.1.1	Nvidia Jetson AGX Orin Developer Kit.....	57
6.1.2	Nvidia Jetson AGX Xavier Developer Kit.....	59
6.1.3	USB Hub.....	59
6.1.4	Kazam	60
6.1.5	Visual Studio Code.....	60
6.1.6	Robot operating system (ROS)	60
6.1.7	Rviz	61
6.1.8	Cámara estereoscópica ZED 2i	61
6.1.9	Configuración de la cámara ZED 2i y mmwave con ROS y Rviz.....	62
6.1.10	Rover Aion Robotic R1.....	70
6.2	Demo 3D_People_Counting	70
6.2.1	Comandos de configuración de la demo 3D_People_Counting.....	70
6.2.2	TLV de la demo 3D_People_Counting.....	77
6.2.3	3D_People_Counting con ROS	78
6.2.4	Creación de un nuevo nodo de ROS: ti_mmwave_trackarray_to_pcl.....	78
6.2.5	3D_People_Counting y ZED 2i	106
6.3	Mobile_Tracker	110
6.3.1	Comandos de configuración de la demo Mobile_Tracker	111
6.3.2	TLV de la demo Mobile_Tracker.....	111
6.3.3	Adaptación del paquete de ROS para Mobile_Tracker	111
6.3.4	Mobile_Tracker con ZED 2i	115
7	Ensayos en exteriores	117
7.1	Validación experimentalmente de las ecuaciones usando OOB.....	117
7.1.1	Configuración de corto alcance de <i>Mmwave Sensing Estimator</i>	117
7.1.2	Aumento del rango máximo por encima de los 10 metros:.....	122
7.1.3	Aumentar el rango máximo por encima de los 14 metros	126
7.1.4	Resolución de 5 cm	127

7.1.5	Velocidad máxima de 5.5 Km/h o 1.5m/s	129
7.1.6	Resolución de velocidad.....	134
7.1.7	Comprobación del ángulo máximo	136
7.2	Montaje de los componentes en el Rover	137
7.3	Demo 3D_People_Counting.....	140
7.4	Demo Mobile_Tracker.....	142
8	Conclusiones sobre las demos.....	144
9	Investigación y líneas futuras.....	144
10	Bibliografía.....	145
11	Ecuaciones para el cálculo del RCS de objetos simples	152
11.1	Esfera.....	152
11.2	Elipsoide	152
11.3	Plato circular plano.....	153
11.4	Cono truncado (Fustrum).....	154
11.5	Cilindro	156
11.6	Plato rectangular plano.....	157
11.7	Plato triangular plano.....	158
12	Tablas de contenido de los diferentes TLV	159

ÍNDICE DE FIGURAS

Figura 1. Chirp, con la amplitud en función del tiempo [1].	2
Figura 2. Chirp, con la frecuencia en función del tiempo [1].	2
Figura 3. Diagrama simplificado de bloques de un sistema FMCW [1].	3
Figura 4. Obtención de la señal de frecuencia constante [1].	3
Figura 5. Señales de frecuencia constante para múltiples objetos [1].	4
Figura 6. Medida de velocidad con 2 chirps [1].	4
Figura 7. Frame [1].	5
Figura 8. FFT de los chirps reflejados [1].	5
Figura 9. Separación de los objetos mediante la transformada Doppler [1].	5
Figura 10. Angle of arrival [1].	6
Figura 11. Configuración para la detección del ángulo del objeto, donde d es la distancia al objeto (R) [1].	6
Figura 12. Bandas de frecuencia de 24 y 64 GHz [2].	7
Figura 13. Bandas de frecuencia de 24 y 77 GHz [3].	7
Figura 14. Diferencias de tamaños de las antenas dependiendo de la frecuencia [2] [3].	8
Figura 15. Una frecuencia mayor permite un rayo más preciso [3].	8
Figura 16. Interferencia de cruce y consecuencias en la onda resultante [4].	9
Figura 17. Señal con glitch (arriba) y señal sin glitch (abajo).	10
Figura 18. Interferencia del tipo paralelo y consecuencias en la onda resultante [4].	10
Figura 19. Chirps ocupando una fracción del tiempo [4].	11
Figura 20. Utilización de diferentes espacios de tiempo y frecuencia para evitar interferencias [4].	11
Figura 21. Separación entre chirps para evitar las interferencias [4].	12
Figura 22. Sensing and avoidance [4].	12
Figura 23. Localización de interferencias mediante la localización de valores atípicos en las muestras [4].	13
Figura 24. Mitigación de interferencias [4].	13
Figura 25. Un objeto fantasma debido a una interferencia de tipo paralelo es expandido debido a la aleatoriedad de la fase del chirp y difuminada debido a la aleatoriedad del idle time [4].	14
Figura 26. Jaula de seguridad del robot [6].	14
Figura 27. Densidad espectral de una onda en función de la frecuencia [8].	15
Figura 28. Anchos de banda para ser considerado UWB [8].	16
Figura 29. Comparación espectral entre UWB y otras tecnologías [9].	16
Figura 30. Máscara espectral para el UWB en Europa para aplicaciones en interiores [10].	16
Figura 31. Principio básico del radar UWB [11].	17
Figura 32. Pulso Gaussiano [9].	17
Figura 33. Monopulso Gaussiano (línea continua) y doble pulso Gaussiano (línea discontinua [10].	17
Figura 34. Componente frecuencial del doble pulso Gaussiano [12].	18
Figura 35. Monitorización mediante UWB [13].	18
Figura 36. Señal recibida de un paciente [13].	19
Figura 37. Gráfico de la distancia de la persona detectada en función del tiempo [14].	19
Figura 38. Señal de la respiración tras eliminar los movimientos de la persona [14].	19
Figura 39. Componentes de los radares FMCW de Texas Instruments.	21

Figura 40. Gama de radares de Texas Instruments [16].	22
Figura 41. Conexión del sensor al módulo UART to USB [22].	24
Figura 42. Software de Seeed Studio [22].	24
Figura 43. Gráfico de velocidades [22].	25
Figura 44. Gráfico de distancias [22].	25
Figura 45. Acabado del radar [24].	26
Figura 46. Radar IWR6843ISK de Texas Instruments [27].	26
Figura 47. Puertos a los que se conecta el radar IWR6843AoP.	27
Figura 48. Puertos a los que se conecta el radar IWR6843ISK.	27
Figura 49. Posición de los jumpers para los diferentes modos de funcionamiento del radar IWR6843AoP [25].	28
Figura 50. Posición de los jumpers para los diferentes modos de funcionamiento del radar IWR6843ISK [26].	28
Figura 51. Jumpers del radar IWR6843ISK en posición de flashing [26].	28
Figura 52. Selección de dispositivo en Uniflash [28].	29
Figura 53. Dispositivo seleccionado en Uniflash [28].	29
Figura 54. Configuración del puerto en Uniflash [28].	29
Figura 55. Apartado para la selección de binario [28].	30
Figura 56. Binario seleccionado [28].	30
Figura 57. Binario cargado correctamente [28].	31
Figura 58. Ajustes seleccionables antes de iniciar el radar IWR6843AoP [35].	32
Figura 59. Setup Details para el radar IWR6843ISK [35].	32
Figura 60. Algunos gráficos disponibles en mmwave Demo Visualizer [35].	33
Figura 61. Peak grouping activado (izquierda) y desactivado (derecha) [35].	33
Figura 62. Parámetros de un chirp [36].	34
Figura 63. El radar emite ondas que rebotan en el objetivo y vuelven a él [39].	37
Figura 64. Esquema de una cámara anecoica para la medida de RCS [41].	39
Figura 65. Medida del RCS rotando alrededor de un eje [38].	40
Figura 66. Medida del RCS rotando alrededor de dos ejes [40].	40
Figura 67. Ancho de haz a media potencia [42].	41
Figura 68. Tipos de respuestas [8].	42
Figura 69. Signal to Noise Ratio (SNR) [44].	44
Figura 70. Ganancia de la antena respecto al ángulo (en la horizontal) [47].	47
Figura 71. Cubo de datos [31].	48
Figura 72. Diagrama de bloques de CFAR_CA [49].	50
Figura 73. Diferentes umbrales de detección de CFAR_CA [49].	51
Figura 74. Ejemplo de funcionamiento del CFAR_CASO [31].	51
Figura 75. Esquema de un TLV.	53
Figura 76. Vistas laterales de Nvidia AGX Orin Developer Kit [53].	57
Figura 77. Vista inferior de Nvidia AGX Orin Developer Kit [53].	58
Figura 78. Vista superior de Nvidia AGX Orin Developer Kit (oculta bajo el módulo) [53].	58
Figura 79 Nvidia Jetson AGX Xavier Developer Kit [55].	59
Figura 80. Hub USB con alimentación externa.	59
Figura 81. Ejemplo de Rviz [62].	61
Figura 82. Cámara ZED 2i.	61
Figura 83. Ejemplo de la nube de puntos generada por la cámara ZED 2i [63].	62
Figura 84. Ejemplo de la detección de objetos y personas de la cámara ZED 2i [63].	62

Figura 85. Visualización de la nube de puntos proporcionada por la cámara ZED 2i en Rviz.....	63
Figura 86. Detección de personas de la cámara ZED 2i.	63
Figura 87. Detección multiclase de la cámara ZED 2i, en este caso detectando un animal.	64
Figura 88. Archivo 6843AOP_multi_3d_0.launch por defecto.	65
Figura 89. Archivo 6843AOP_multi_3d_0.launch con las correcciones.....	66
Figura 90. Nube de puntos del radar (OOB) con Rviz mientras detecta a una persona y un objeto a la izquierda.....	66
Figura 91. Funciona el radar, pero falla la cámara ZED.....	67
Figura 92. Funciona la cámara ZED, pero falla el radar.....	67
Figura 93. Esquema del ejemplo de la web oficial de ROS [69].	68
Figura 94. Datos de base_laser en base_link en el ejemplo de la web oficial de ROS [69].	68
Figura 95. Archivo 6843AOP_multi_3d_0.launch después de realizar los cambios.	69
Figura 96. Cámara ZED 2i y radar (Out_of_the_box) en Rviz.....	69
Figura 97. Rover Aion Robotic R1 [70].	70
Figura 98. Antenas y canales de alimentación de los radares XWR6843AoP [31].....	73
Figura 99. Colocación de las antenas del radar.....	73
Figura 100. Patrón de antenas virtuales para el IWR6843AoP [31].	74
Figura 101. Antenas y canales de alimentación de los radares XWR6843ISK [31].	74
Figura 102. Patrón de antenas virtuales para el IWR6843ISK [31].	75
Figura 103. Antenas transmisoras del IWR6843AoP.....	75
Figura 104. Antenas receptoras del IWR6843AoP.	76
Figura 105. Nube de puntos de la detección de una persona.	78
Figura 106. Archivo CMakeLists.bit del nodo ti_mmwave_trackarray_to_pcl.	79
Figura 107. Archivo ti_mmwave_trackarray_to_pcl.launch del nodo ti_mmwave_trackarray_to_pcl.	79
Figura 108. Archivo LICENSE del nodo ti_mmwave_trackarray_to_pcl.....	79
Figura 109. Archivo package.xml del nodo ti_mmwave_trackarray_to_pcl.....	80
Figura 110. Archivo setup.py del nodo ti_mmwave_trackarray_to_pcl.....	80
Figura 111. Compilación mediante catkin build.....	80
Figura 112. Programa para leer el topic /mystring.	81
Figura 113. Comando que genera el topic mystring que publica el mensaje "Hello".	81
Figura 114. Comando para comprobar que el topic mystring publica correctamente.....	82
Figura 115. Nodo ti_mmwave_trackarray_to_pcl publicando el mensaje "Hello".	82
Figura 116. Datos que publica el topic ti_mmwave/radar_scan_trackarray.....	82
Figura 117. Contenido del mensaje RadarTrackArray.msg.	83
Figura 118. Contenido del mensaje RadarTrackContents.msg.	83
Figura 119. #include necesarios para leer el topic.....	83
Figura 120. Función TrackarrayCallback.	84
Figura 121. Cambio en la función del suscriptor.....	84
Figura 122. Diagrama de flujo del programa.	84
Figura 123. Diagrama de flujo del bucle main.....	85
Figura 124. Topic ti_mmwave_trackarray_to_pcl publicando los datos de posición del objeto.	86
Figura 125. #Include y definición del publicador simple.....	86
Figura 126. Bucle TrackarrayCallback para añadir el publicador simple.....	87
Figura 127. Advertencia del publicador simple.....	87
Figura 128. Recepción de datos del topic deseado.....	87

Figura 129. Comprobación de que el publicador simple publica correctamente.....	88
Figura 130. Mensaje de nube de puntos del topic ti_mmwave/radar_scan_pcl_1.....	88
Figura 131. Mensaje sensor_msgs/PointCloud2 [74].	89
Figura 132. Mensaje std_msgs/Header [76].	89
Figura 133. Mensaje sensor_msgs/PointField [75].	90
Figura 134. Archivo CMakeLists.bit del nodo ti_mmwave_trackarray_to_pcl para publicar una nube de puntos.	90
Figura 135. Archivo ti_mmwave_tracker_to_pcl.h del nodo ti_mmwave_trackarray_to_pcl para publicar una nube de puntos.	90
Figura 136. #include para publicar un punto.	91
Figura 137. Código necesario para publicar un punto.	91
Figura 138. Rellenar mensaje std_msgs/Header para publicar un punto.....	91
Figura 139. Rellenar dos de los parámetros del sensor_msgs/PointCloud2.....	92
Figura 140. Rellenar mensaje sensor_msgs/PointField para publicar un punto.	92
Figura 141. Obtención de los datos para completar el vector de datos de una coordenada.....	93
Figura 142. Definición de la macro al principio del código para reinterpretar un punto como int.	94
Figura 143. Rellenar el resto de los valores del sensor_msgs/PointCloud2 y publicar el mensaje.	94
Figura 144. Cambio en la línea 118 para poder publicar correctamente	94
Figura 145. Comprobación de la publicación de un punto de una coordenada como nube de puntos.	95
Figura 146. Obtención de los datos para completar el vector de datos de un punto.	95
Figura 147. Comprobación de la publicación de un punto completo como nube de puntos.....	95
Figura 148. Visualización target y nube de puntos en Rviz.....	96
Figura 149. Error al detectar 2 objetos a la vez.....	96
Figura 150. Toma de datos y construcción del vector de datos.....	97
Figura 151. Ajustar los valores del mensaje para publicar todos los puntos.....	97
Figura 152. Target siguiendo la sube de puntos.	98
Figura 153. Dos targets siguiendo las nubes de puntos.....	98
Figura 154. Varios targets siguiendo las nubes de puntos (con ejes de coordenadas).	99
Figura 155. Mensaje visualization_msgs/Marker [78].	100
Figura 156. Mensaje geometry_msgs/Pose [79].	100
Figura 157. Mensaje geometry_msgs/Point [80].	100
Figura 158. Mensaje geometry_msgs/Quaternion [81].	100
Figura 159. Mensaje geometry_msgs/Vector3 [82].	101
Figura 160. Mensaje std_msgs/ColorRGBA [83].	101
Figura 161. #include necesarios para publicar un marker de texto.....	101
Figura 162. Definición del nuevo publicador de markers.	101
Figura 163. Código para rellenar el mensaje visualization_msgs/Marker.	102
Figura 164. Rellenar algunos valores del mensaje visualization_msgs/Marker.	102
Figura 165. Definición de la posición para un texto fijo en el origen.....	103
Figura 166. Escala y color de un texto fijo en el origen.	103
Figura 167. Formato que se le da al texto fijo que se verá en Rviz.....	103
Figura 168. Publicador del marker texto para un texto fijo.	104
Figura 169. Advertencia de publicación de un marker.	104
Figura 170. Formato del mensaje del marker.	104

Figura 171. Visualización en Rviz de un mensaje fijo.....	105
Figura 172. Obtención y cálculo de la velocidad de cada objeto.....	105
Figura 173. Cambios para publicar la velocidad por cada punto.....	106
Figura 174. Visualización en Rviz de 2 puntos con su velocidad.....	106
Figura 175. Agregación de velocidad e identificador en el vector de datos de la nube de puntos.	107
Figura 176. Sensor_msgs/PointField para rellenar los campos de coordenadas, velocidad y nube de puntos.	107
Figura 177. Definición de variables para el color de las etiquetas.....	107
Figura 178. Código para distinguir las velocidades no nulas.	107
Figura 179. Valores del mensaje std_msgs/ColorRGBA para diferentes colores.....	108
Figura 180. Etiquetas para definir tipo y acción.....	108
Figura 181. Modificación de la posición de la etiqueta de velocidad.	108
Figura 182. Diagrama de flujo del bucle TrackArrayCallback.....	109
Figura 183. Cambios en el formato del target.	110
Figura 184. Resultado final para la demo 3D_People_Counting.	110
Figura 185. Archivo mmWave.h modificado.....	112
Figura 186. Definición de los nuevos estados SorterState.....	112
Figura 187. Esquema de interpretación de los TLV.....	113
Figura 188. Definición de lo TLV dentro de CHECK_TLV_TYPE.....	113
Figura 189. Código del TLV de identificador 7.	114
Figura 190. Necesidad de tener al menos 1 punto para que pueda mostrarse en Rviz.	114
Figura 191. Código para el TLV con identificador 1000 añadiéndole un punto fijo.....	115
Figura 192. Demo Mobile_Tracker con Rviz.....	115
Figura 193. Resultado final para la demo Mobile_tracker.....	115
Figura 194. Chirp de la configuración por defecto de mmwave Demo Visualizer.	118
Figura 195. Chirp de la configuración por defecto de mmwave Demo Visualizer con $N_{ADC} = 128$	119
Figura 196. Detección de un objeto a 5,5 metros.....	119
Figura 197. Chirp de la configuración por defecto de mmwave Demo Visualizer con $S = 35$...	120
Figura 198. Detección de una persona tras reducir la pendiente de la recta.	121
Figura 199. Chirp de la configuración por defecto de mmwave Demo Visualizer con $T_{ramp} = 28$	121
Figura 200. Detección de una persona tras reducir el tiempo del chirp.....	122
Figura 201. Chirp para aumentar el rango por encima de los 7.5 metros.	122
Figura 202. Resultado después de bajar 7 dB los umbrales del CFAR.	123
Figura 203. Umbrales del CFAR demasiado bajos.....	124
Figura 204. Resultado con los umbrales finales para el CFAR.....	124
Figura 205. Medida de distancia de 14 metros.....	125
Figura 206. Tracking de una persona con la configuración de 14 metros.	125
Figura 207. Chirp para conseguir un rango máximo de 30 m.	127
Figura 208. Chirp para obtener un rango máximo de 30 m y una resolución de rango de 5 cm.	129
Figura 209. Valores recomendables del idle time para ciertos anchos de banda [47].	130
Figura 210. Chirp para obtener un rango máximo de 30 m, una resolución de rango de 5 cm y velocidad máxima de 5.5 Km/h.....	131
Figura 211. Gráfica de las velocidades del Rover.....	132

Figura 212. Posición del radar para el ensayo de velocidad.	133
Figura 213. Detección del Rover hasta que su velocidad sobrepasó el límite del radar.	133
Figura 214. Detección del Rover moviéndose hacia el radar.	134
Figura 215. Detección del Rover moviéndose en perpendicular a la dirección del radar.	134
Figura 216. Chirp para obtener un rango máximo de 30 m, una resolución de rango de 5 cm, velocidad máxima de 7 Km/h y resolución de velocidad de 1 Km/h.	136
Figura 217. Medición del ángulo máximo del radar.	136
Figura 218. Módulo WIFI añadido en la parte inferior de la placa de desarrollo.	137
Figura 219. Antenas WIFI de la placa Jetson AGX Xavier Developer Kit.	138
Figura 220. Batería y regulador.	138
Figura 221. Parte inferior del Rover.	139
Figura 222. Estructura metálica atornillada al Rover con los elementos colocados.	139
Figura 223. Objetos detectados por el radar y la cámara durante un giro brusco del Rover. ...	140
Figura 224. Detección de un objeto y una persona por el radar.	141
Figura 225. Detección de la persona cuando ya ha entrado en el rango de la cámara.	141
Figura 226. Detección de 3 objetos con Mobile_Tracker.	142
Figura 227. Detección errónea de una persona.	142
Figura 228. Detección errónea de un objeto.	143
Figura 229. Esfera [38].	152
Figura 230. Elipsoide [38].	153
Figura 231. Plato circular plano. [38]	154
Figura 232. Definición del ángulo de un cono. [38]	155
Figura 233. Cono truncado o fustrum [38].	155
Figura 234. Cilindro de sección elíptica (izquierda) y cilindro de sección circular (derecha). [38]	157
Figura 235. Plato rectangular plano. [38].	158
Figura 236. Plato triangular plano. [38]	158

ÍNDICE DE TABLAS

Tabla 1. Parámetros del comando channelCfg.	35
Tabla 2. Parámetros del comando adcCfg.	35
Tabla 3. Parámetros del comando profileCfg.....	36
Tabla 4. Parámetros del comando frameCfg.....	36
Tabla 5. Parámetros del comando cfarCfg.....	52
Tabla 6. Contenido de la cabecera del frame.....	53
Tabla 7. Contenido de la cabecera del TLV.	54
Tabla 8. Identificadores de los diferentes TLV.	55
Tabla 9. Descripción de los diferentes TLV.....	56
Tabla 10. Parámetros del servicio de ROS Object detection.....	65
Tabla 11. Parámetros de static_transform_publisher.	68
Tabla 12. Parámetros del comando dynamicRACfarCfg.	71
Tabla 13. Parámetros del comando staticRACfarCfg.	72
Tabla 14. Parámetros del comando antGeometry0.....	72
Tabla 15. Parámetros del comando antGeometry1.....	72
Tabla 16. Parámetros del comando antPhaseRot.....	75
Tabla 17. Parámetros del comando boundaryBox.....	76
Tabla 18. Parámetros del comando sensorPosition.....	77
Tabla 19. Parámetros del comando stateParam.....	77
Tabla 20. Parámetros de configuración por defecto mmwave Sensing Estimator.....	118
Tabla 21. Parámetros de entorno de la configuración por defecto mmwave Sensing Estimator.	118
Tabla 22. Parámetros de configuración para obtener un rango máximo de más de 10 m.	126
Tabla 23. Parámetros de entorno de la configuración para obtener un rango máximo de más de 10 m.....	126
Tabla 24. Parámetros de configuración para obtener un rango máximo de 30 m.....	127
Tabla 25. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m.	127
Tabla 26. Parámetros de configuración para obtener un rango máximo de 30 m y una resolución de rango de 5 cm.	129
Tabla 27. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m y una resolución de rango de 5 cm.	129
Tabla 28. Parámetros de configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm y velocidad máxima de 5.5 Km/h.	131
Tabla 29. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm y velocidad máxima de 5.5 Km/h.	131
Tabla 30. Velocidades del Rover.	132
Tabla 31. Parámetros de configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm, velocidad máxima de 7 Km/h y resolución de velocidad de 1 Km/h.	135
Tabla 32. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm, velocidad máxima de 7 Km/h y resolución de velocidad de 1 Km/h.....	135
Tabla 33. Estructura de datos del TLV Detected points.	159

Tabla 34. Estructura de datos del TLV Statics para XWR6843, XWR1843, XWR1642 y XWR1443.	159
Tabla 35. Estructura de datos del TLV Statics para XWRL6432.....	159
Tabla 36. Estructura de datos del TLV Side info for detected points.....	159
Tabla 37. Estructura de datos del TLV Temperature statics.	160
Tabla 38. Estructura de datos del TLV Shperical coordinates.	160
Tabla 39. Estructura de datos del TLV 3D Target List.....	160
Tabla 40. Estructura de datos del TLV Target index.....	161
Tabla 41. Estructura de datos del TLV Target height.	161
Tabla 42. Estructura de datos del TLV 3D Spherical Compressed Point Cloud: Estructura de la nube de puntos.	161
Tabla 43. Estructura de datos del TLV 3D Spherical Compressed Point Cloud: Estructura de los puntos.	161

1 Introducción/Antecedentes

La seguridad en entornos logísticos e industriales es de interés primordial para estos sectores dentro de la **factoría del futuro y la Industria 4.0**.

Los atropellos en naves industriales y de logística con alta movilidad de vehículos (*Forklifts* y *AGVs*), aun no siendo frecuentes, pueden tener **consecuencias de gravedad** para las personas trabajadoras. Asimismo, la seguridad en maquinaria con riesgo de atrapamiento, riesgo eléctrico y zonas de acceso restringido es de vital importancia para **reducir la siniestralidad laboral y minimizar sus consecuencias**.

Este trabajo se ha realizado en colaboración con la empresa **Fundación I+D Automoción y Mecatrónica**, también conocida como **NAITEC**. NAITEC es un Centro Tecnológico especializado en movilidad y mecatrónica, cuya sede se encuentra en Navarra. Esta empresa tiene el propósito de generar oportunidades de negocio aportando **soluciones tecnológicas** para mejorar la eficiencia, funcionalidad y sostenibilidad de los productos y procesos.

Uno de los **grandes retos** de esta empresa en cuanto al sector de la movilidad es crear productos y soluciones innovadoras, eficientes y sostenibles, de tal forma que el consumo energético se reduzca para avanzar a una movilidad más segura, confortable y respetuosa con el medio ambiente. NAITEC lleva trabajando en el mercado de la movilidad desde **2003**, y desde entonces, ha ido adquiriendo experiencia, conocimiento y equipamiento, lo que les **permite afrontar los nuevos retos** a los que se enfrenta este sector.

Este trabajo se ha desarrollado en su edificio ubicado en **Noain**, en el Polígono Mocholí, Plaza Ceín,
4.

2 Objetivos

El objetivo es **aplicar la sensórica de Radar y técnicas de percepción** para sistemas de seguridad proactiva de las personas trabajadoras. Esto es extrapolable a la detección de cualquier objeto o animal que se encuentre en el rango de acción del vehículo, permitiendo su detección. Se **implementarán algoritmos de procesamiento, tratamiento de datos y señal** para **detectar, principalmente, la presencia de personas** en situaciones de riesgo mediante Radar, que, junto con una cámara estereoscópica equipada con una red neuronal, permitirá la distinción entre personas, animales u objetos.

3 Desarrollo teórico de tecnologías radar

En esta parte se compararán dos tecnologías de radar, **millimeter wave**, también conocida como mmwave y **Ultra-Wide Band** o **UWB**.

3.1 Tecnología mmwave

Mmwave es una tecnología radar que usa **ondas electromagnéticas de longitud de onda pequeña** (en el rango de los **milímetros**) [1]. El sistema radar transmite ondas que son **reflejadas por los objetos** que se encuentran en su camino. Capturando esta señal reflejada, se puede determinar la distancia, rango y ángulo del objeto. Un sistema mmwave incluye antenas transmisoras (TX) y receptoras (RX) de radiofrecuencia (RF), componentes analógicos como un reloj u oscilador y conversores analógico-digital.

3.1.1 Fundamentos de la tecnología mmwave

En los dispositivos *Frequency-Modulated Continuous Wave* (FMCW) **la frecuencia de la onda electromagnética crece linealmente con el tiempo**. Este tipo de señal es conocida como *chirp* (Figura 1), cuya ventaja principal es que permite calcular distancias y velocidades de una forma sencilla.

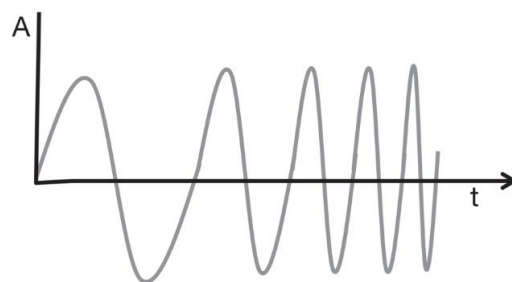


Figura 1. Chirp, con la amplitud en función del tiempo [1].

La Figura 2 muestra exactamente la misma señal representando la frecuencia en función del tiempo. Las magnitudes características del *chirp* son la **frecuencia de inicio**, f_0 (aunque en la Figura 2 se le llama f_c), el **ancho de banda** (B), la **duración** (T_c) y la **pendiente de frecuencia** (S). Esta pendiente recoge la variación de la frecuencia de la señal de la Figura 1.

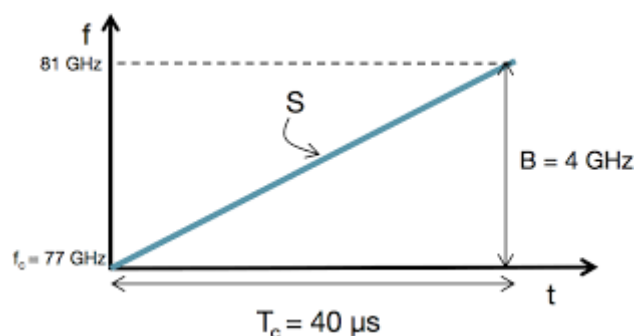


Figura 2. Chirp, con la frecuencia en función del tiempo [1].

Como se ha explicado anteriormente, **el radar transmite un chirp y captura la señal reflejada por el objeto**. La Figura 3 representa un diagrama de bloques simplificado de los componentes principales.

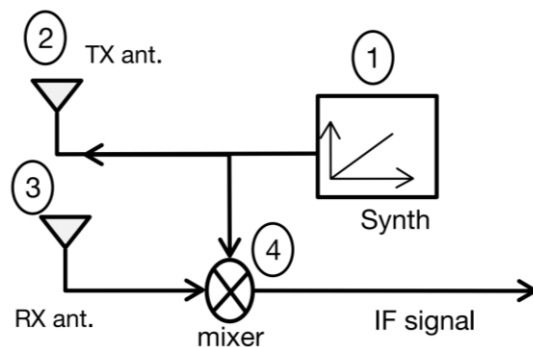


Figura 3. Diagrama simplificado de bloques de un sistema FMCW [1].

Un sintetizador (Synth) genera un *chirp* que es transmitido por la antena transmisora (TX ant.) y la reflexión de ese *chirp* es recogida por la antena receptora (RX ant.). Un mezclador (mixer) combina las señales de las antenas TX y RX para generar una señal de frecuencia intermedia. **La señal reflejada es exactamente la señal transmitida retrasada en el tiempo.** Ese tiempo de retraso (τ) puede calcularse matemáticamente mediante la siguiente ecuación, donde R es la distancia al objeto detectado y c es la velocidad de la luz. Es importante destacar que la señal resultante, de frecuencia S_τ , solo es **válida en el intervalo de tiempo donde coexisten el *chirp* emitido y el *chirp* recibido.**

$$\tau = \frac{2R}{c}$$

Ecuación 1

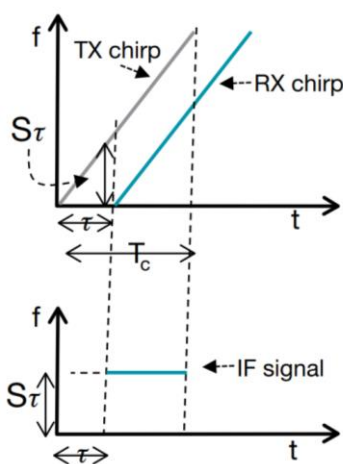


Figura 4. Obtención de la señal de frecuencia constante [1].

La fase inicial de la señal IF (ϕ_0) es la diferencia entre la fase del chirp transmitido y la fase del chirp recibido en el momento del comienzo de la señal de frecuencia constante, es decir, en el tiempo τ .

$$\phi_0 = 2\pi f_c \tau$$

Ecuación 2

Matemáticamente, la Ecuación 2 se puede transformar en la siguiente ecuación:

$$\phi_0 = \frac{4\pi R}{\lambda}$$

Ecuación 3

Resumiendo lo anterior, para un objeto a una distancia d del radar, la señal senoidal de frecuencia constante o señal IF es:

$$A \sin(2\pi f_0 t + \phi_0)$$

Ecuación 4

donde $f_0 = \frac{2SR}{c}$.

Cuando hay **varios objetos** detectados, se reciben **diferentes chirps**, cada uno retrasado un cierto tiempo dependiendo la distancia a la que se encuentra el objeto. Estos *chirps* se traducen en **múltiples señales de frecuencia constante**, como se puede ver en la Figura 5.

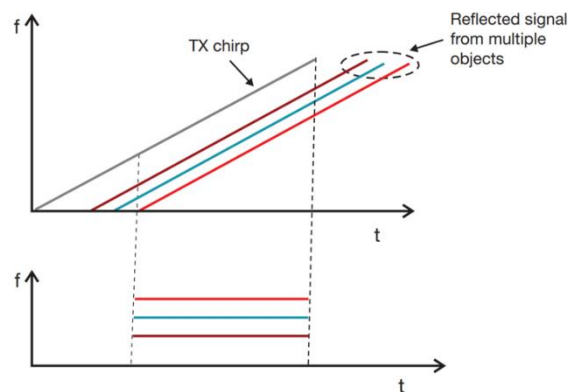


Figura 5. Señales de frecuencia constante para múltiples objetos [1].

Esta señal compuesta de múltiples tonos debe ser procesada usando la **transformada de Fourier (FFT)** para separar los diferentes tonos. La señal procesada dará lugar a **diferentes picos de frecuencia** por cada tono, que se corresponderá con cada distancia del objeto.

Para el cálculo de la velocidad se usan los fasores para denotar los números complejos. A la hora de medir velocidad, los radares emiten **2 chirps separados por T_c** . Estos *chirps* se procesarán aplicando la transformada de Fourier y el resultado será **2 chirps** que tendrán **picos de frecuencia en la misma localización, pero con diferente fase**. Esa diferencia en la fase corresponde al movimiento de los objetos.

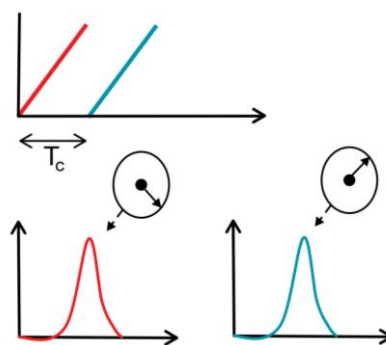


Figura 6. Medida de velocidad con 2 chirps [1].

La diferencia en las fases se deriva de la Ecuación 3, dando lugar a la siguiente expresión.

$$\Delta\phi = \frac{4\pi v T_c}{\lambda}$$

De esta ecuación, se puede despejar la velocidad:

$$v = \frac{\Delta\phi\lambda}{4\pi T_c}$$

Debido a que la medida de velocidad está basada en una diferencia de fases, habrá una **ambigüedad**. Esta ambigüedad **se resuelve para $|\Delta\phi| < \pi$** , es decir, el ángulo se medirá **desde $-\pi$ hasta π** . Por ejemplo, si el ángulo es $270^\circ = \frac{3\pi}{2} \text{ rad}$ se expresaría como $-90^\circ = -\frac{\pi}{2} \text{ rad}$.

La forma anterior de medir la velocidad **no sirve cuando hay múltiples objetos** que se mueven a distintas velocidades a la misma distancia del radar ya que, al estar a la misma distancia, los *chirps* reflejados generarán picos de la misma frecuencia. Para conseguir medir la velocidad, el **radar debe emitir más de 2 chirps**. Se transmiten N *chirps* equiespaciados, que forman lo que se conoce como **frame** (Figura 6).

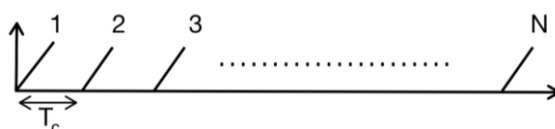


Figura 7. Frame [1].

Al igual que antes, tras el procesado FFT aparecerán N **picos de frecuencia iguales pero cada uno con una fase diferente**.

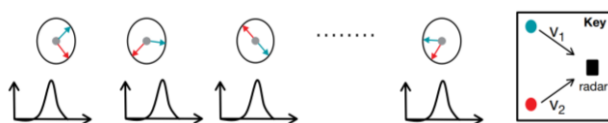


Figura 8. FFT de los chirps reflejados [1].

Una segunda transformada llamada **transformada Doppler** (Doppler-FFT) se aplica en los N fasores para diferenciar los objetos, como se ve en la Figura 9, donde ω_1 y ω_2 se corresponden con la diferencia de fase entre *chirps* consecutivos de cada uno de los objetos.

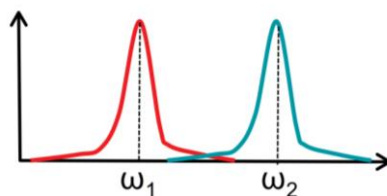


Figura 9. Separación de los objetos mediante la transformada Doppler [1].

Los radares FMCW también pueden estimar el **ángulo en el plano horizontal** de la señal reflejada (Figura 10). Este ángulo se le llama *Angle of Arrival* (AoA).

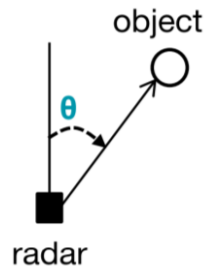


Figura 10. Angle of arrival [1].

La estimación del ángulo se basa en la observación de una pequeña variación en la distancia del objeto, que resulta en un **pequeño cambio en el pico de frecuencia** de en la FFT o Doppler-FFT. Para lograr esto, es necesario utilizar **al menos 2 antenas receptoras** (Figura 11).

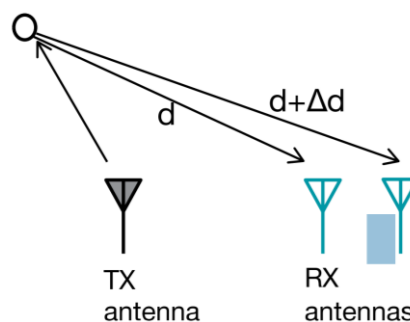


Figura 11. Configuración para la detección del ángulo del objeto, donde d es la distancia al objeto (R) [1].

La pequeña separación en distancia del objeto a cada una de las antenas hace que se pueda dar esa **pequeña variación en el pico de frecuencia**, permitiendo así estimar el ángulo. Con esta configuración, la variación de la fase se puede conocer con la siguiente expresión matemática.

$$\Delta\phi = \frac{4\pi\Delta R}{\lambda}$$

Ecuación 7

Haciendo la suposición de un frente de ondas plano, por geometría se obtiene que $\Delta R = l \sin \theta$, donde l es la separación entre antenas receptoras. De esta forma, el ángulo se puede estimar mediante la siguiente expresión, donde para medir ángulos sin **ambigüedad es necesario que $|\Delta\phi| < \pi$** .

$$\theta = \sin^{-1}\left(\frac{\lambda\Delta\phi}{2\pi l}\right)$$

Ecuación 8

De la Ecuación 7 se puede ver que $\Delta\phi$ depende de $\sin \theta$, que es una dependencia no lineal. Cuando el **ángulo θ tiene un valor pequeño**, $\sin \theta$ se puede aproximar como θ . Como resultado, **la estimación resulta más exacta cuando el ángulo tiene un valor pequeño**.

3.1.2 Diferencias entre frecuencias de trabajo (24/60/77 GHz)

Los radares mmwave utilizan **bandas de frecuencias diferentes, 24-24.25 GHz, 60-64 GHz y 77-81 GHz** [2] [3]. La banda de 77 GHz es la más utilizada en aplicaciones de automoción, pero se ha restringido en algunos países para otros usos.

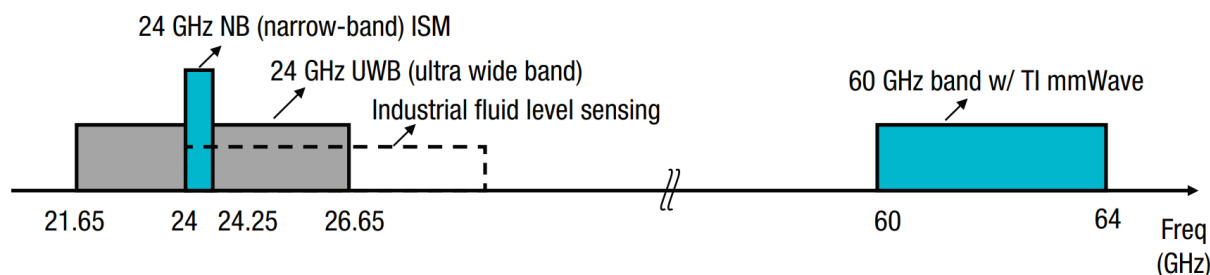


Figura 12. Bandas de frecuencia de 24 y 64 GHz [2].

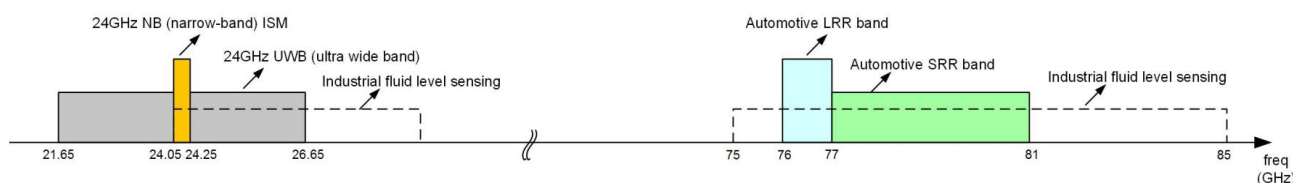


Figura 13. Bandas de frecuencia de 24 y 77 GHz [3].

Como se puede observar en la Figura 12 y Figura 13, el rango de frecuencias para el 24 GHz UWB va desde 21.65 GHz a 26.65 GHz, ofreciendo un ancho de banda de 5 GHz. Esta banda está **prohibida desde septiembre de 2018** y para el 2022 todos los dispositivos que trabajen en ese rango de frecuencias fueron eliminados. La otra componente de la banda de frecuencia de 24GHz es la **narrowband, que ofrece un ancho de banda 250MHz** para uso industrial, médico y científico. En comparación, tanto la banda de frecuencia de 60 GHz como la de 77 GHz ofrecen un **ancho de banda de 4 GHz**, que son 16 veces más del que ofrece la banda de 24 GHz teniendo en cuenta las regulaciones.

Una vez descartada la banda *narrowband* de 24 GHz, el uso de la banda de 60 GHz, que actualmente no está limitada por ninguna regulación, permite **obtener una nube de puntos con gran precisión**, ya que la resolución depende directamente del ancho de banda, por lo que usar sensores de 60 GHz es una buena alternativa. En cuanto al ancho de banda de 77 GHz (Figura 13), existe un rango de frecuencias de **76-77 GHz disponible para radares en vehículos de largo alcance**. Esta banda está disponible en Europa para infraestructura de transporte o sistemas de radares para otras aplicaciones como contador de vehículos, atascos, detección de accidentes o medida de la velocidad. El radar de corto alcance de 77-81 GHz, al igual que el de la banda de 60 GHz, ofrece un ancho de banda de 4 GHz, lo hace **una buena opción para aplicaciones donde se requiera buena resolución**.

Como se ha comentado anteriormente, la resolución **depende directamente del ancho de banda de la señal del radar**. Los radares de 60 y 77 GHz, que ofrecen un ancho de banda de 4 GHz, tienen una **resolución máxima de 3.75cm**. Para el caso de 24 GHz, la mejor resolución posible es 60cm, es decir, se consigue una resolución 20 veces mejor con los radares de 60 y 77 GHz que con los de 24 GHz.

La resolución de velocidad depende de varios parámetros, pero esta **se escala proporcionalmente con la frecuencia central**. Por lo tanto, **cuanto mayor sea la frecuencia central, mejor será la resolución de velocidad**. Los radares de 60 GHz mejoran esta resolución en 2.5 veces y los de 77GHz la mejoran 3 veces.

El tamaño del paquete o **encapsulado** (*package size*) es de vital importancia para los fabricantes. Una de las **ventajas de utilizar una mayor frecuencia es que el tamaño de las antenas puede ser menor**.

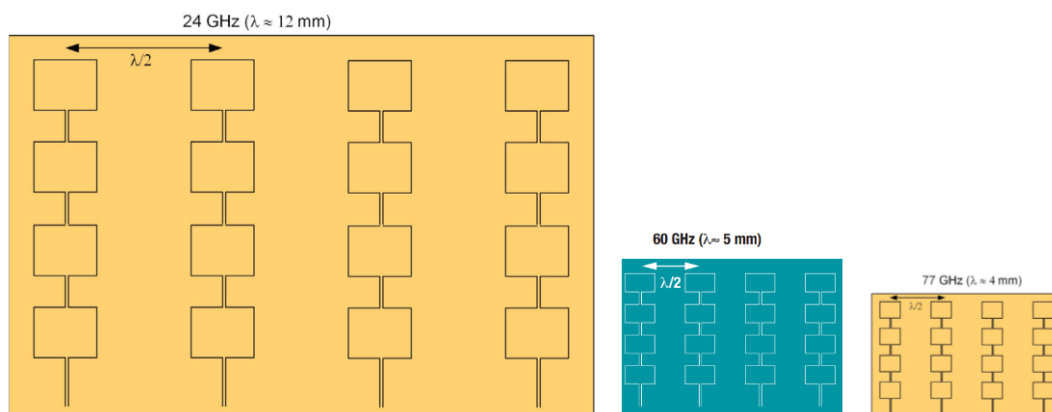


Figura 14. Diferencias de tamaños de las antenas dependiendo de la frecuencia [2] [3].

Comparando las antenas de 77 y 24 GHz, el tamaño de las últimas puede ser **hasta 3 veces más grandes** que el de las antenas de 77 GHz. Las antenas pequeñas, además, pueden mitigar más reflexiones no deseadas del entorno que las antenas grandes, creando un haz más preciso (Figura 15).

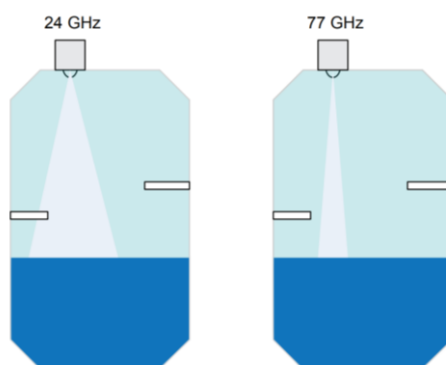


Figura 15. Una frecuencia mayor permite un rayo más preciso [3].

3.1.3 Interferencias entre radares mmwave

Las **interferencias mutuas** entre radares son un obstáculo para el amplio despliegue de radares [4]. Debido a ellas, se pueden dar **fallos en la detección**, objetos fantasmas (detecta objetos donde no los hay), etc. Utilizando diferentes técnicas tanto para **evitarlas** como para **mitigarlas** se pueden **controlar esas interferencias** y hacer que el radar sea un **dispositivo robusto**.

En primer lugar, se definen 2 términos: víctima y agresor. El **radar víctima** es el radar cuya **antena receptora se ve afectada por las interferencias**. El **agresor es un radar cuyas ondas transmitidas afectan al receptor** de la víctima. La intensidad de la interferencia ($P_{interference}$) se puede calcular con la siguiente ecuación (despreciando cualquier pérdida de espacio libre) [4], [5]:

$$P_{interference} = P_{Tx} + txAntGain + rxAntGain - 10 \log \left(\frac{4\pi R}{\lambda} \right)^2$$

donde P_{Tx} es la potencia de la antena transmisora, $txAntGain$ es la ganancia de la antena transmisora del radar agresor, $rxAntGain$ la ganancia de la antena receptora del radar víctima, R es la distancia entre los dos radares y λ es la longitud de onda media.

La ecuación de la **potencia recibida por el radar para objetos** es la siguiente, donde P_r es la potencia recibida y el RCS es el *Radar Cross-Section*, que será explicado más adelante, en el apartado 5.4.1:

$$P_r = P_{Tx} + txAntGain + rxAntGain + RCS - 10 \log \left(\frac{(4\pi)^3 R^4}{\lambda^2} \right)^2$$

Ecuación 10

Comparando el último término (término de pérdidas) de la Ecuación 9 y la Ecuación 10, se ve que el efecto de la distancia (R) es mucho menor en el de las interferencias, lo que hace que **la interferencia domine sobre la potencia recibida**, aunque el radar agresor se encuentre muy alejado.

Por ejemplo, para una longitud de onda de 5 mm y una distancia de 20 m, las pérdidas para la interferencia serán 47dBm (Ecuación 9) y para la potencia recibida (Ecuación 10) 191 dBm, es decir, se pierde mucho más para el caso de la potencia recibida.

3.1.3.a Interferencia de cruce o *crossing interference*

Si el radar agresor y víctima tienen **diferente pendiente**, los dos *chirps* se **crizan entre ellos**, como se puede ver en la gráfica de la parte superior de la Figura 16. Cuando esto ocurre, la señal transmitida del agresor se mezcla con la señal transmitida de la víctima, y se produce la **interferencia de cruce o *crossing interference***.

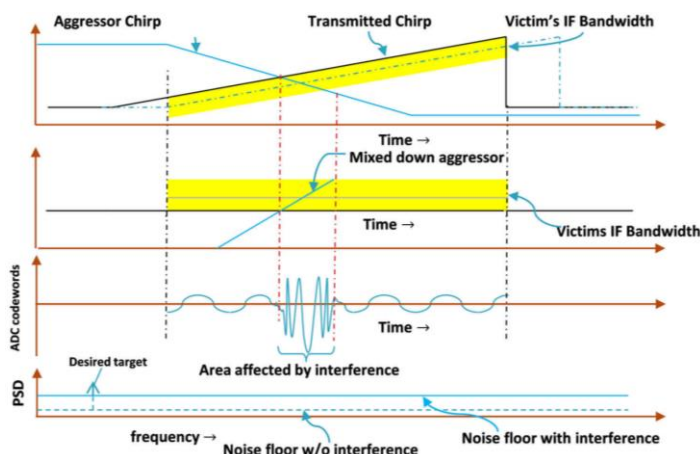


Figura 16. Interferencia de cruce y consecuencias en la onda resultante [4].

La energía del agresor solo puede ser vista si la diferencia de frecuencia está **dentro del ancho de banda válido (IF Bandwidth) de la víctima**, como se puede ver en las dos gráficas centrales de la Figura 16. Esta señal proveniente del agresor puede ser constante, ascendente, o decreciente. En la región afectada se produce un **glitch**, es decir, una región donde la onda resultante de la resta entre la transmitida y la recibida deja de ser de frecuencia constante.

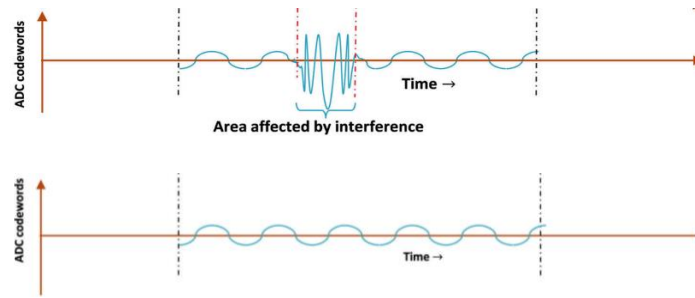


Figura 17. Señal con glitch (arriba) y señal sin glitch (abajo).

Finalmente, una vez aplicada la FFT en las muestras, típicamente el nivel de ruido o *noise floor* asciende (gráfica inferior de la Figura 16) y **reduce el *Signal to Noise Ratio (SNR)*** de objetivos fuertes, **atenuando objetos débiles**, y no permitiendo su detección. El tiempo de duración del *glitch* se puede calcular con la siguiente ecuación:

$$\tau_{Glitch} = \frac{IF \text{ Bandwidth}}{|slope_{aggressor} - slope_{victim}|}$$

Ecuación 11

La **duración del *glitch*** es típicamente **pequeña**, siguiendo la Ecuación 11, por ejemplo, para un ancho de banda efectivo de 12MHz y una diferencia de rampas de 40 MHz/μs, es aproximadamente 0.3μs.

3.1.3.b Interferencia de tipo paralelo o *parallel interference*

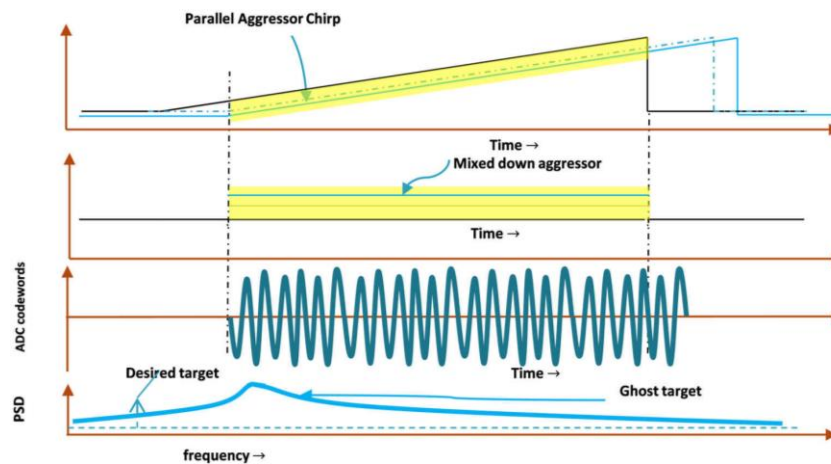


Figura 18. Interferencia del tipo paralelo y consecuencias en la onda resultante [4].

Cuando el *chirp* del agresor y la víctima tienen la **misma pendiente**, la interferencia solo ocurre cuando los tiempos de inicio de los *chirps* entre la víctima y el agresor están tan cerca que hace que el ***chirp* del agresor esté dentro del ancho de banda efectivo del *chirp* de la víctima**. Cuando se mezclan, la interferencia paralela se transforma en un **tono de frecuencia constante** y, después de la FFT, **se convierte en un objeto fantasma**, que se comporta como un objeto a una distancia y con una velocidad aleatorias. Cuando esto pasa, **la región de interferencia es casi todo el *chirp***, aunque la **probabilidad de que ocurra sea muy pequeña**.

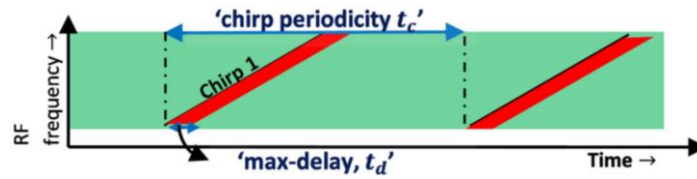


Figura 19. Chirps ocupando una fracción del tiempo [4].

La **probabilidad de interferencia** P_{intf} se calcula usando el tiempo de retraso máximo t_d (max-delay), la periodicidad de repetición del chirp t_c y el número de radares presentes N_r según la siguiente ecuación:

$$P_{intf} = 1 - \left(1 - \frac{t_d}{t_c}\right)^{N_r}$$

Ecuación 12

Cuando la víctima y el agresor tienen un **oscilador local independiente**, es **difícil saber la frecuencia exacta de la rampa**, a pesar de haber usado los mismos parámetros de configuración para ambos. En ese caso, la señal del **objeto fantasma no aparecería como una señal limpia**, si no que aparecería con un poco de ruido, lo que se puede usar para **detectar este tipo de interferencias**.

3.1.3.c Evitar las interferencias

A la hora de evitar las interferencias, el primer método es la **estandarización**, que trata de tener en cuenta la frecuencia y el **diseño de los chirps**. La **división del ancho de banda**, basado en la resolución necesaria, permite que distintos radares coexistan. Por ejemplo, para la familia de 60 GHz, los 4 GHz de ancho de banda se podrían dividir en 2 rangos de 2 GHz (60-62 y 62-64) aunque se suele dejar un espacio entre medio de los 2 rangos. Si el ciclo de trabajo es 10%, en teoría se podrían usar 10 radares diferentes separados en tiempo. La separación por ancho de banda es **muy sencilla de implementar**, mientras que para la **separación por tiempo** todos los radares deben tener un **reloj o timing source común** para lograr la correcta sincronización.

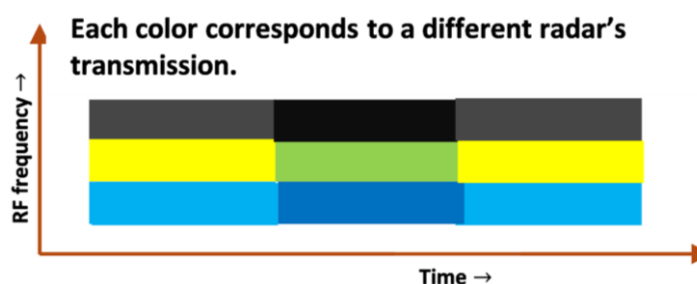


Figura 20. Utilización de diferentes espacios de tiempo y frecuencia para evitar interferencias [4].

En la Figura 20 se puede observar que se ha **dividido el ancho de banda en 3 rangos** (negro, amarillo y azul) y se transmiten 3 **chirps** por cada rango separados por el tiempo. En este caso podrían coexistir **9 radares**.

Si un solo fabricante está fabricando todos los radares, puede hacer que se sincronicen con el **mismo reloj**. Si todos los radares están configurados con los mismos parámetros del **chirp y frame**, se pueden evitar las interferencias paralelas si **cada frame del radar tiene un cierto offset**. De esta forma muchos radares pueden coexistir en espacios reducidos en el mismo ancho de banda.

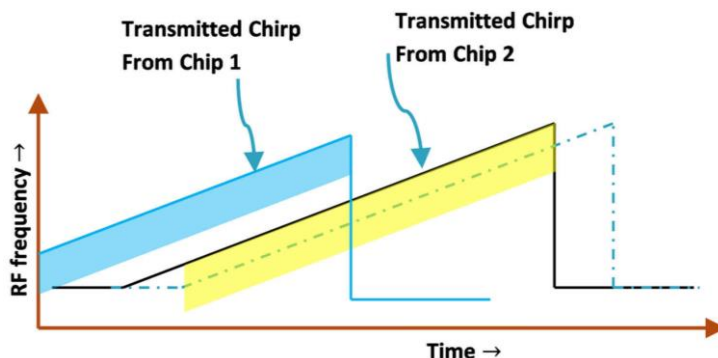


Figura 21. Separación entre chirps para evitar las interferencias [4].

Un método simple de conseguir la sincronización entre radares es con el modo **maestro/esclavo**. Al transmitir un *frame*, el radar maestro genera **triggers** a los esclavos (la señal recibida por los esclavos les indica cuándo deben transmitir) y con esto, se puede conseguir un **retraso bastante exacto** antes de emitir sus *frames*.

3.1.3.d Identificación de las interferencias

En **ausencia de alguna sincronización**, se puede aplicar el siguiente método. En el esquema de la Figura 22, antes de que un radar empiece su transmisión, **se mide el espectro** encendiendo los receptores y apagando los transmisores. Si no hay otras transmisiones de otros radares el **espectro es silencioso**, es decir, los datos solo deben mostrar ruido ambiente. Por otro lado, si hay transmisiones, se producen **picos en las muestras en las zonas donde se produce la interferencia**. Si el periodo es lo suficientemente largo para cubrir múltiples *frames*, se puede llegar a **estimar el número de radares agresores**. También se puede estimar la periodicidad del *frame* y encontrar **huecos o franjas temporales** donde no se perciban radares agresores y se pueda transmitir sin interferencia. A este método se le conoce como **sensing and avoidance**.

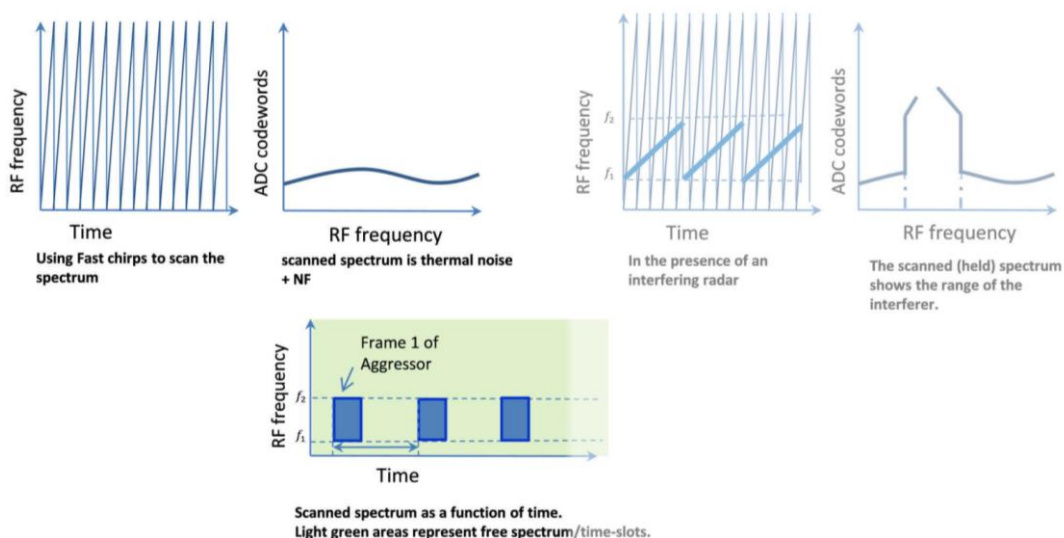


Figura 22. Sensing and avoidance [4].

En muchos casos, es posible **evitar las interferencias de cruce**, que pueden aumentar el *noise floor* y atenuar objetivos débiles. Este tipo de interferencia se puede localizar observando **valores atípicos en las muestras**, ya que fuertes interferencias aparecen como largos *glitches*. En la Figura 23 se puede ver la representación de la cada muestra en un *chirp* a lo largo del tiempo. En el punto donde se produce esta

interferencia hay un incremento en su energía. Se puede eliminar usando un **umbral** que haga que cualquier valor que lo supere cierta energía sea considerado como interferencia. Cuando se muestra el valor absoluto de las muestras, el *glitch* es bastante visible, aunque no muy distinguible de la señal. Sin embargo, si esas señales de baja frecuencia son eliminadas, el *glitch* sobresale todavía más.

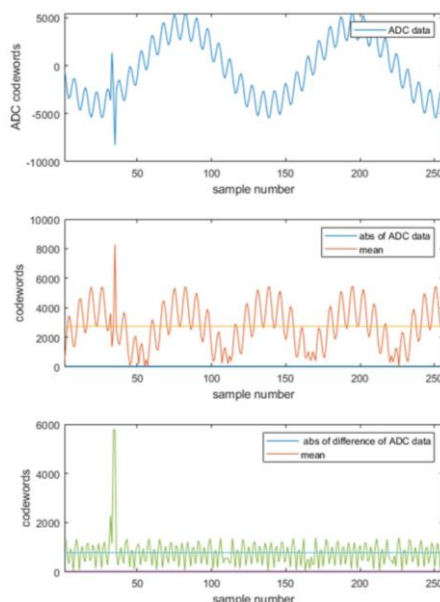


Figura 23. Localización de interferencias mediante la localización de valores atípicos en las muestras [4].

3.1.3.e Mitigación de las interferencias

Ahora que se han encontrado las interferencias, se busca **mitigarlas**. El método más simple de mitigación es **sustituir las interferencias por ceros** (Figura 24.1, a la izquierda las muestras tomadas y a la izquierda, la salida después de realizar la FFT). Un mejor enfoque es hacerlo en una ventana móvil, que hace que se dé una mejor detectabilidad de objetivos suaves. (Figura 24.2). Otro enfoque todavía mejor es llevar a cabo una **interpolación lineal** en la zona donde se detectó la interferencia, usando la última muestra buena antes y después del periodo de interferencia (Figura 24.3).

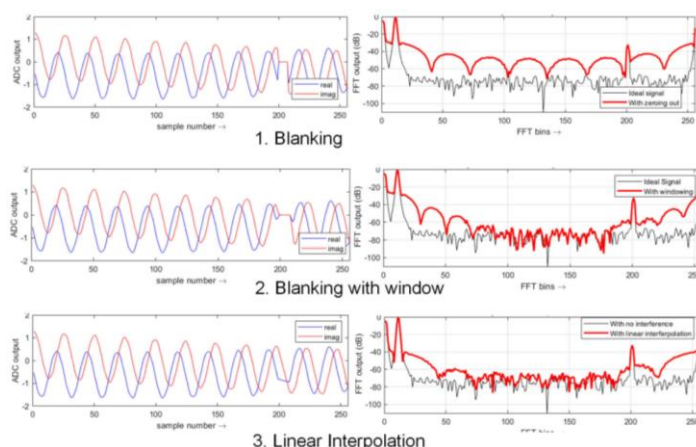


Figura 24. Mitigación de interferencias [4].

Cuando ocurre la interferencia de tipo paralelo, **todo o casi todo el chirp se ve afectado** y es difícil corregirlo. Sin embargo, el método de localización y mitigación (el caso anterior) no es muy útil para este tipo

de interferencias. Este tipo de interferencias se pueden atenuar mediante un proceso llamado **aleatorización o difuminación del chirp** (*chirp dithering or chirp randomization*).

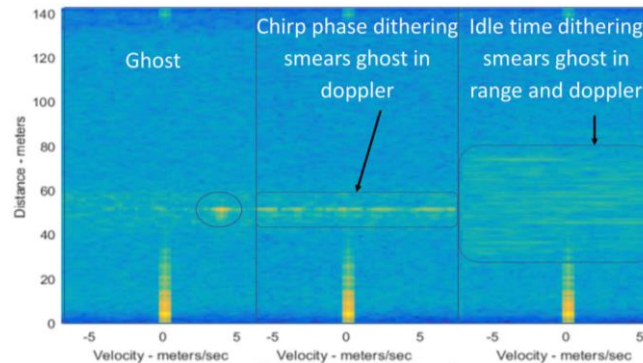


Figura 25. Un objeto fantasma debido a una interferencia de tipo paralelo es expandido debido a la aleatoriedad de la fase del chirp y difuminada debido a la aleatoriedad del idle time [4].

En este proceso, **ciertos parámetros del chirp se hacen aleatorios** (como, por ejemplo, la fase inicial). Hay otros múltiples parámetros del *chirp* que se pueden aleatorizar como la rampa de frecuencia, la frecuencia de inicio y el idle time. La Figura 25 muestra un objeto fantasma debido a una interferencia de tipo paralelo que se ha **difuminado** en el diagrama Doppler. Si no se hace ninguna randomización, estas interferencias aparecen como objetos fantasmas. Con la randomización, el pico de la interferencia es destruido por la difuminación y se pueden **eliminar usando algoritmos CFAR** (que se explicarán en el apartado 5.5).

Este método introduce **mayor complejidad** durante el procesado Doppler debido a que se le tiene que **aplicar una corrección**. Por ejemplo, la aleatorización de la fase inicial se puede corregir aplicando una fase opuesta justo antes del procesado Doppler. Algunos parámetros que se pueden modificar, como el idle time pueden introducir un incremento del **noise floor** en el rango Doppler.

3.1.4 Aplicaciones de la tecnología mmWave

Los radares mmwave tienen múltiples aplicaciones, pero una de las más importantes es su **uso en la robótica** [6]. Una de las ventajas sobre otros sensores de medida (Como un LIDAR o una cámara RGB) es que **puede trabajar en ambientes con lluvia, polvo, humo e incluso sin visibilidad**. Cada vez los robots interactúan más con los humanos y es muy importante que no causen daños a las personas que trabajan a su alrededor. El método común es crear una zona o **jaula de seguridad** (Figura 26) alrededor del robot para aislar al robot y asegurar la separación.



Figura 26. Jaula de seguridad del robot [6].

Los radares (en general, los sensores) hacen posible la **creación de una zona de seguridad virtual** para **separar al robot del humano**, incluso **evitar que dos robots colisionen**. Sensores como la cámara requieren un entorno visible y un mantenimiento en el caso de que haya suciedad o polvo. Estos radares trabajan bien en esas condiciones, por lo que **cada vez se usan más para este tipo de aplicaciones**. El amplio rango (de distancia y angular) y la habilidad de detectar más de un objeto (o humano en este caso) los hace una **opción robusta y barata**.

3.2 Tecnología Ultra-Wide Band (UWB)

A continuación, se describirá la otra tecnología radar, **Ultra-Wide Band** o UWB. Al igual que los mmwave, el radar transmite ondas que son **reflejadas por los objetos** que se encuentran en su camino.

3.2.1 Fundamentos de la tecnología UWB

Los radares UWB producen unos **pulsos muy cortos** de radiofrecuencia en el dominio del tiempo, en el rango del nanosegundo o picosegundo.

Un radar de esta tecnología consiste en una **antena transmisora** con un **generador de pulsos** (debe ser capaz de generar pulsos del orden de los nanosegundos), una **antena receptora** (detector de pulsos) y una **unidad de procesado** [7].

El ancho de banda de la señal se define como $\Delta f = f_h - f_l$, donde al límite inferior se le llama f_l y al límite superior f_h .

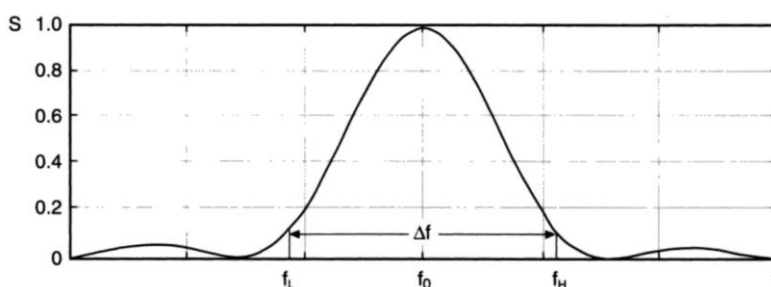


Figura 27. Densidad espectral de una onda en función de la frecuencia [8].

Para que se considere radar *Ultra-Wide band*, el **ancho de banda fraccional debe superar 0.2**, según *Institute of Electrical and Electronics Engineers (IEEE)* y *American Federal Communications Commission (American FCC)*, y **0.25 según U.S. Defense Advanced Research Projects Agency's (DARPA)** [8].

Terms	DARPA 1991	IEEE STD 1672	American FCC	European Union
Bandwidth power limits		-10 dB	-10 dB	Based on spectrum and radiated power
Band lower frequency	f_l	f_l	f_l	
Band high frequency	f_h	f_h	f_h	
Bandwidth	$f_h - f_l$	$f_h - f_l$	$f_h - f_l$	
Center frequency	$f_c = \frac{(f_h + f_l)}{2}$	$f_c = \frac{(f_h + f_l)}{2}$	$f_c = \frac{(f_h + f_l)}{2}$	
Fractional bandwidth	$\frac{2(f_h - f_l)}{f_h + f_l} \leq 0.25$	$\frac{2(f_h - f_l)}{f_h + f_l} \leq 0.20$	$\frac{2(f_h - f_l)}{f_h + f_l} \leq 0.20$	
Absolute bandwidth		500 MHz	500 MHz	>50 MHz

Figura 28. Anchos de banda para ser considerado UWB [8].

La norma también define el ancho de banda que debe tener una señal UWB el cual debe ser **mayor a 500 MHz** (o tener un ancho de banda fraccional mayor al 20%) [9]. Se le asigna también una porción de espectro para tecnología, que en este caso está entre **3.1 GHz y 10.6 GHz** (ancho de banda de 7.5 GHz). La densidad espectral de potencia no debe superar los **-41.3 dBm/MHz** (Según la American FCC) para **garantizar la coexistencia con otras tecnologías inalámbricas**. De esta forma, el UWB se queda por debajo del *noise floor* o nivel de ruido y **las otras tecnologías lo detectan como ruido y lo ignoran**.

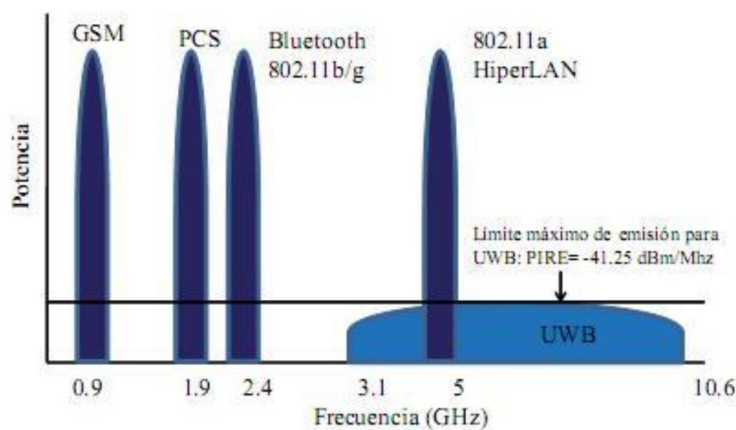


Figura 29. Comparación espectral entre UWB y otras tecnologías [9].

En Europa, se puede utilizar la banda desde los 1.6 GHz [10], pero **no se permite utilizarlo a la máxima potencia (-41.3 dBm/MHz) hasta los 3.6 GHz**, como se puede ver en la Figura 30.

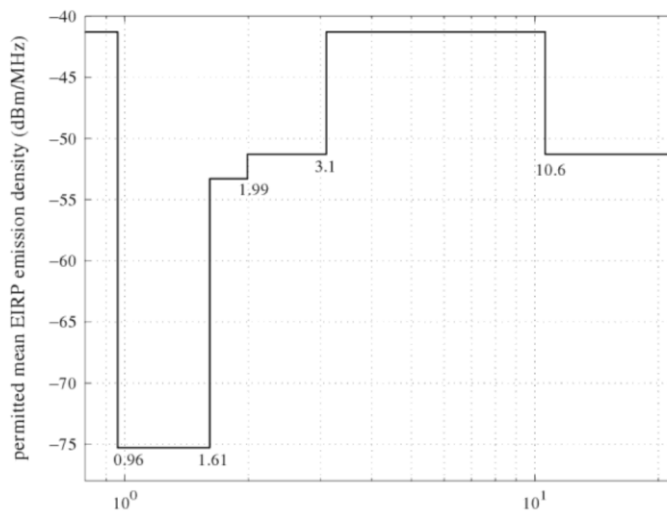


Figura 30. Máscara espectral para el UWB en Europa para aplicaciones en interiores [10].

El principio de funcionamiento del radar UWB es que **se genera un pulso por el terminal de transmisión TX**, la señal se propaga hasta que se **encuentra con un objeto**, donde **parte de la energía electromagnética se refleja** y se **propaga de vuelta a la antena receptora RX** [11]. El tiempo entre las dos ondas representa la **distancia** a la que se encuentra el objeto.

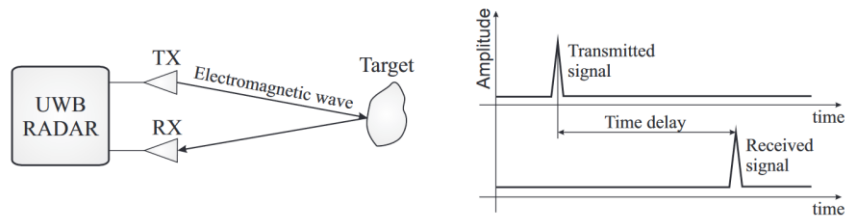


Figura 31. Principio básico del radar UWB [11].

El **monopulso gaussiano** y el **pulso doble gaussiano** son los más típicos y corresponden matemáticamente a la **primera y segunda derivada del pulso gaussiano** respectivamente. Una de las principales razones por las que se usa es su facilidad de generación.

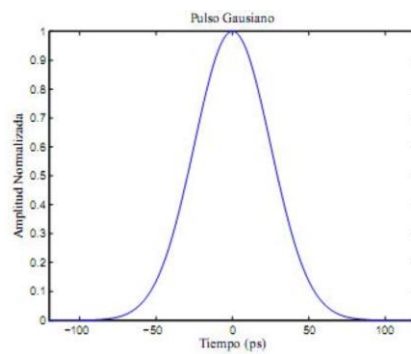


Figura 32. Pulso Gaussiano [9].

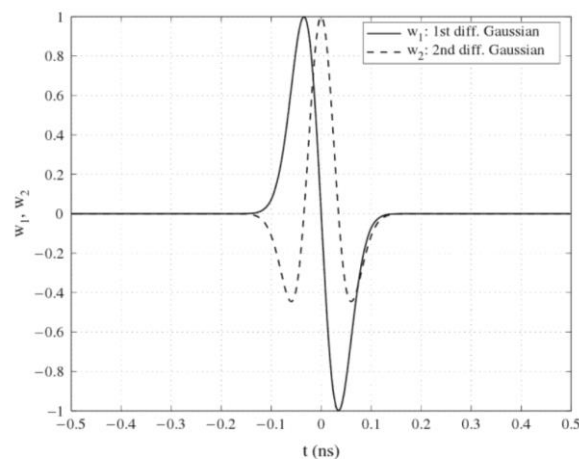


Figura 33. Monopulso Gaussiano (línea continua) y doble pulso Gaussiano (línea discontinua) [10].

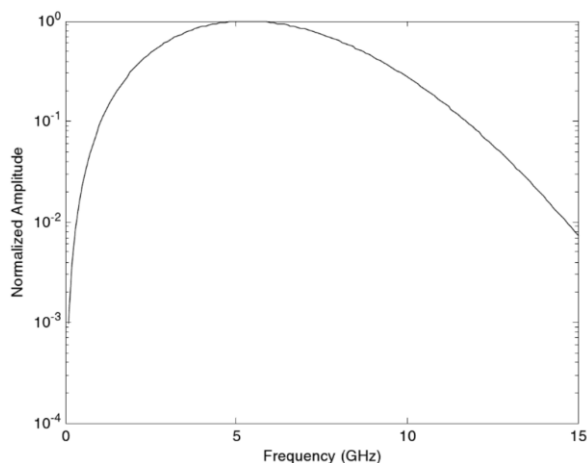


Figura 34. Componente frecuencial del doble pulso Gaussiano [12].

3.2.2 Aplicaciones de la tecnología UWB

Una de las aplicaciones más comunes de los radares UWB es su **uso en medicina**, para la monitorización de pacientes en una distancia corta [13]. Esta monitorización se puede aplicar en unidades de cuidados intensivos, operaciones de rescate, pediatría, etc., para medir **respiración y pulsaciones**. En la Figura 35 se muestra cómo se monitorizan las constantes vitales de un paciente, con la ventaja de que no hay cables de por medio.



Figura 35. Monitorización mediante UWB [13].

La señal emitida se refleja y cuando el paciente se mueve, **las señales recibidas por el sensor variarán**. En la Figura 36 se puede observar unos picos de la señal recibida, cuanto más alta sea, más movimiento se habrá detectado.

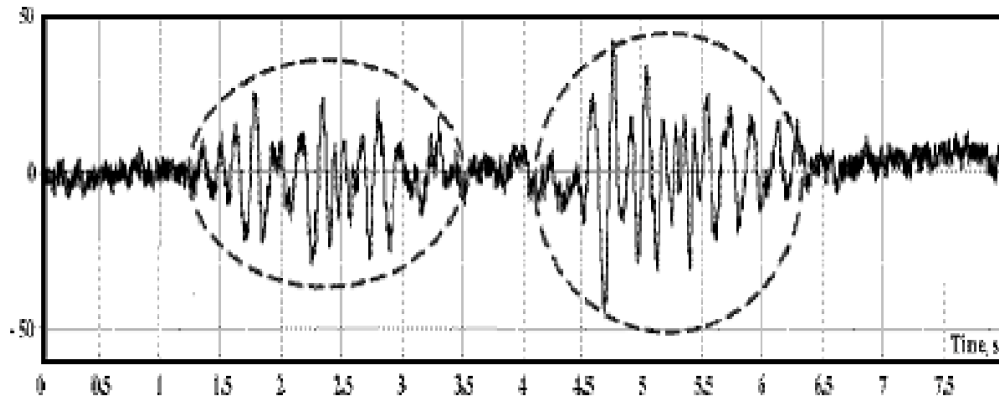


Figura 36. Señal recibida de un paciente [13].

Cuando una persona camina, los movimientos de torso, cabeza, brazos, piernas, etc., generan **variaciones de frecuencia** en el rango Doppler [14]. Esos cambios producidos por el movimiento se pueden **eliminar usando un algoritmo muy complejo**, de tal forma que se obtiene la respiración cancelando los movimientos del cuerpo.

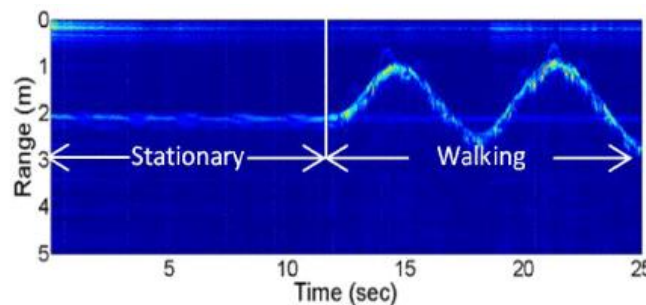


Figura 37. Gráfico de la distancia de la persona detectada en función del tiempo [14].

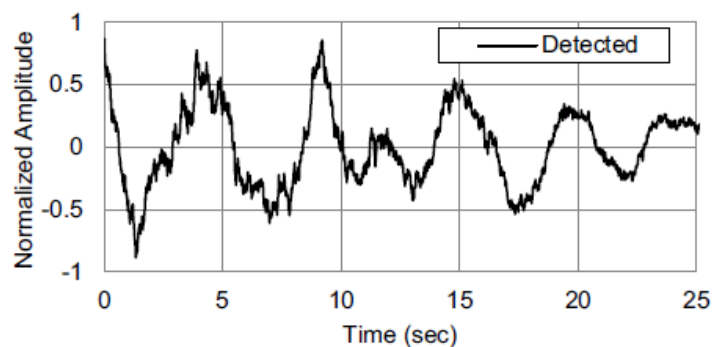


Figura 38. Señal de la respiración tras eliminar los movimientos de la persona [14].

En este ejemplo, una persona permanece quieta durante 12 segundos y luego anda durante 13 segundos (Figura 37). La señal de respiración corresponde a ambos periodos de medida, en estático y en movimiento (Figura 38). El algoritmo **puede detectar la respiración** incluso cuando hay otro tipo de movimientos de gran amplitud.

3.3 Comparación teórica entre mmwave y UWB

Una de las principales ventajas de los radares mmwave es que **pueden trabajar perfectamente condiciones adversas** (baja visibilidad como niebla, humo, polvo, etc.) y se pueden recubrir con una cobertura de plástico sin perjudicar su trabajo. Para el cálculo de posiciones es necesario realizar FFT o Doppler-FFT con los radares mmwave, lo que conlleva un **procesado de datos adicional que UWB no necesita** y, por lo tanto, **necesidad de microprocesadores más potentes o elementos hardware** que sean capaz de realizar esos cálculos en un tiempo óptimo. En la tecnología UWB, al emitir pulsos más rápidos (del orden de los picosegundos) y no requerir procesado extra, se pueden **tomar bastantes más datos por segundo** que con radares mmwave (ondas del orden de los microsegundos)

Los radares mmwave son inmunes al denominado *ground clutter* (muestras reflejadas por el suelo), lo cual le permite **observar elementos de forma cercana y superficies** con poca reflectividad. Además, debido al rango de frecuencias de trabajo (60-64 GHz y 77-81 GHz) hace que el **tamaño de la antena de estos radares sea más pequeño** que los de UWB (un tamaño común de antena UWB suele ser alrededor de 12x12 mm y de mmwave alrededor de 5x5 mm).

El radar que se va a utilizar va a trabajar en **ambientes industriales sucios**, con humo, polvo, etc., por lo que este es uno de los principales motivos por el que se ha decidido trabajar con la **tecnología mmwave**, a pesar de que sea necesario un procesado extra de los datos con transformadas de Fourier. Además, estos radares pueden trabajar bajo la incidencia de luces brillantes, poca luz o incluso sin luz. Al trabajar con frecuencias más elevadas, **el tamaño de las antenas será menor** que los UWB, por lo que son una buena opción en aplicaciones donde el tamaño disponible sea reducido. Al tener unas regulaciones tan estrictas en cuanto a potencia, **el rango máximo de los radares UWB será menor** que los de los radares mmwave.

4 Radares mmwave

En este apartado se realizará la **comparativa** entre los radares de la tecnología mmwave para la detección de objetos y personas de dos fabricantes, **Texas Instruments y Seeed Studio**. Se describirán las características de los diferentes radares y se **elegirá el fabricante** que más se adapte a los requerimientos del estudio.

4.1 Radares mmwave de Texas Instruments

Texas Instruments (TI), es una empresa tecnológica estadounidense con sede en Dallas, Texas, que **diseña, manufactura, prueba y vende dispositivos semiconductores y tecnología embebida**, para crear diferentes aplicaciones en el marco industrial, automoción, personal, etc. [15].

Los radares de Texas Instruments **integran en un único chip Radio Frequency Complementary Metal-Oxide Semiconductor (RFCMOS) un Digital Signal Processor (DSP), MCU y diversos componentes analógicos** como un **convertor analógico-digital, un reloj y un acelerador hardware (HWA)** [1].

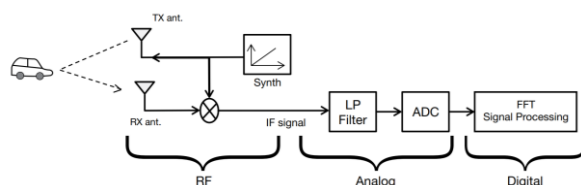


Figura 39. Componentes de los radares FMCW de Texas Instruments.

Los radares RFCMOS se diferencian de las soluciones anteriores de SiGe permitiendo **flexibilidad en la programabilidad y en el procesamiento de información**. Los radares de SiGe pueden almacenar un número limitado de *chirps* y requieren de una intervención a tiempo real para actualizar los *chirps* durante un *frame*. Sin embargo, **los radares de Texas Instruments RFCMOS pueden almacenar hasta 512 chirps antes del inicio del frame**, lo que ayuda a su mejor configuración para maximizar las prestaciones del radar.

4.1.1 Gama de radares

Texas Instruments ofrece una **amplia gama de radares** mmwave, cada uno con diferentes características [16]. Todos estos radares se pueden ver en la Figura 40. Existen tipos de radares para **diferentes frecuencias** (60 GHz y 77 GHz) y con **distintos componentes**.

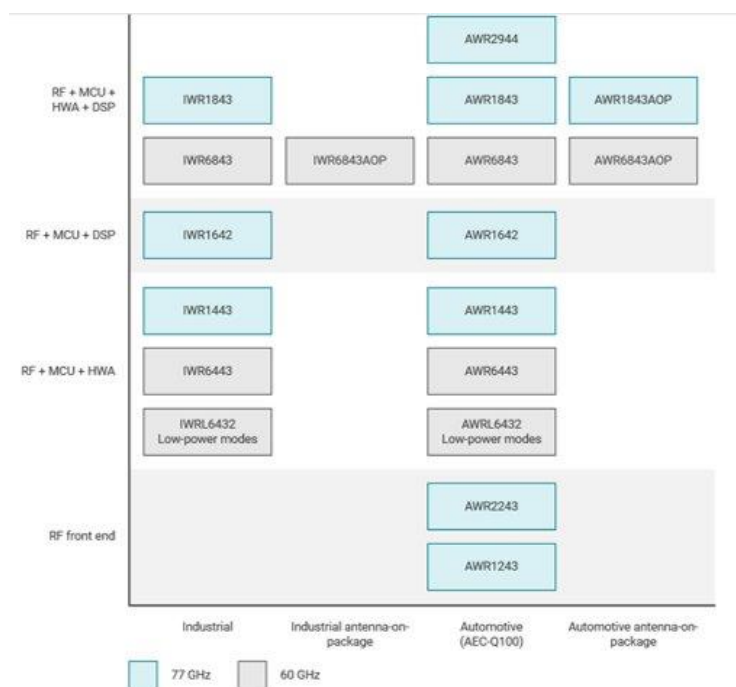


Figura 40. Gama de radares de Texas Instruments [16].

Los radares más básicos son los de la primera franja (empezando a contar de abajo a arriba) de la Figura 40, que solo disponen del circuito generador, receptor y mezclador, es decir, dispondrían de un diagrama de bloques como el de la Figura 3. Son una buena opción para tenerlos en ambientes controlados para una medición precisa.

El resto de los radares posee, como mínimo, un microcontrolador (MCU). Los radares de la segunda franja tienen además un acelerador hardware (HWA) [17]. El acelerador hardware puede realizar algoritmos típicos de los radares FCMW como FFT, CFAR, etc., liberando de carga al microcontrolador y al Digital Signal Processor (DSP) en el caso de que tenga. Los radares de la tercera franja tienen en vez del HWA un DSP. Un DSP es un sistema basado en un microprocesador que posee un conjunto de instrucciones, hardware y un software optimizado para realizar operaciones numéricas a muy alta velocidad. Los radares de la franja superior tienen todos los componentes anteriores. Al ser los radares más completos, son los radares que interesan.

Dentro de los radares, se usarán los radares industriales de 60 GHz, ya que la banda de 77 GHz se suele emplear para automoción. Por lo tanto, de todos los radares interesa usar los de la familia 6843, que son el IWR6843 y el IWR6843AOP.

Todos los radares de este fabricante son altamente configurables y muestran los objetos detectados en forma de nube de puntos, es decir, muestran todos los puntos que detectan en 3 dimensiones. Estos puntos se pueden ver en el visualizador en Python gratuito que Texas Instruments proporciona. Devuelven los datos en forma de TLV y no requieren un dispositivo adicional para la visualización de datos, solo requieren unos drivers gratuitos. Además, un solo radar puede llevar a cabo múltiples tareas, lo que se conoce como demos. Una demo es un programa que se carga en el firmware del radar y, devolverá unos datos propios de la demo que podrán ser tratados.

4.1.2 Diferencias entre radares industriales y de automoción (IWR e AWR)

Ambos radares son iguales solo que los AWR son capaces de trabajar a un **mayor rango de temperatura** y que están "**AECQ100 qualified**" y son "**ASIL-B capable**" [18] [19]. También puede haber diferencias en los puertos de comunicación, por ejemplo, para el 1642 el AWR tiene 2 interfaces CAN y el IWR tiene 1. Además, para el caso del 6843, el AWR tiene un ARM CPU ARM-Cortex R4F @200MHz y los IWR ARM R4F @200MHz.

- ASIL-B: **Automotive Safety Integrity Level**. Para cada componente electrónico se miden 3 cosas. **Severidad** (tipo de lesiones provocadas a los pasajeros), **exposición** (Cuánto tiempo está expuesto el vehículo a ese riesgo) y **controlabilidad** (cuánto puede hacer el piloto para evitar la lesión). Cada riesgo se divide en varias subclases. Cuando se clasifican las variables se combinan para dar el **rango de ASIL, siendo A el más leve y el D el más severo**. Los radares AWR deben cumplir con la normativa **ASIL-B**.
- AECQ100: Es un fallo mecánico basado en el **estrés de circuitos integrados**. Pasar esta prueba significa que el dispositivo aguanta los niveles de estrés a los que ha sido expuesto y tiene cierto nivel de fiabilidad.

4.1.3 Diferencias entre AoP e ISK

El término **ISK proviene de Industrial Starter Kit**, creado para la familia de IWR6843. El término **AoP proviene de Antenna on Package**, es decir, la diferencia con los radares ISK, las antenas se encuentran empaquetadas en vez de en el exterior.

4.2 Radares mmwave de Seeed Studio

Seeed Studio es un **fabricante de módulos electrónicos**, con sede en Shenzhen, Guangdong, que forma parte de la tendencia cada vez más popular de los **proyectos de open source o de código abierto** [20]. Los productos de esta empresa son perfectos para los fanáticos de Arduino y las computadoras de una sola placa. El fabricante, en su página web, además de la **documentación** técnica, proporciona una gran cantidad de **guías y ejemplos** de proyectos.

Seeed Studio dispone de **5 radares diferentes especializados** para realizar una tarea concreta. Los radares que dispone son:

- **24GHz Human Static Presence Lite**
- 24GHz Human Stationary Sensor
- 24GHz Respiratory Sleep Sensor
- 60GHz Respiratory Heartbeat Sensor
- 60GHz Fall Detection Pro Sensor

Para esta aplicación, se necesitará el **24GHz Human Static Presence Lite (MR24HPC1)**, para que detecte personas u objetos [21]. Se trata de un sensor FMCW, que posee herramientas para el **debugging y configuración del radar**. Además, es **compatible con Arduino**, lo que hace que sea posible su utilización en múltiples aplicaciones para la detección de personas.

Para conectar el radar al ordenador, hace falta conectarlo con un **dispositivo UART to USB** y hacer las conexiones como se indica en la Figura 41 [22].

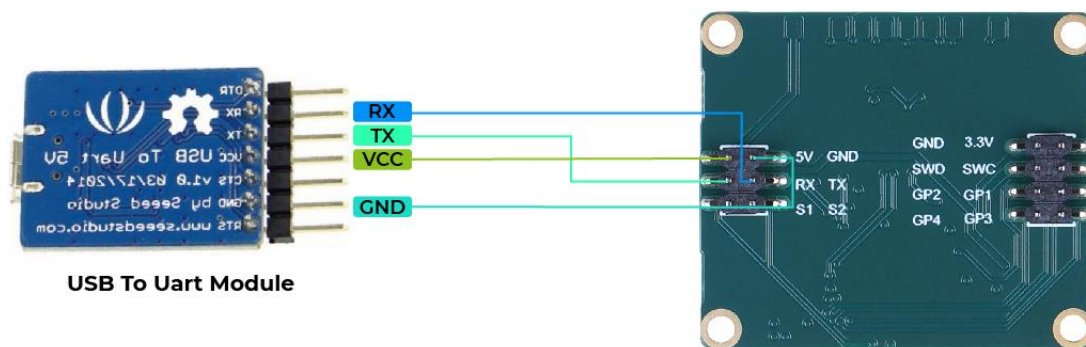


Figura 41. Conexión del sensor al módulo UART to USB [22].

Además de poder utilizar Arduino, se puede emplear el **upper computer software** que proporciona el fabricante. Se trata de un software en el que se pueden configurar algunos parámetros.

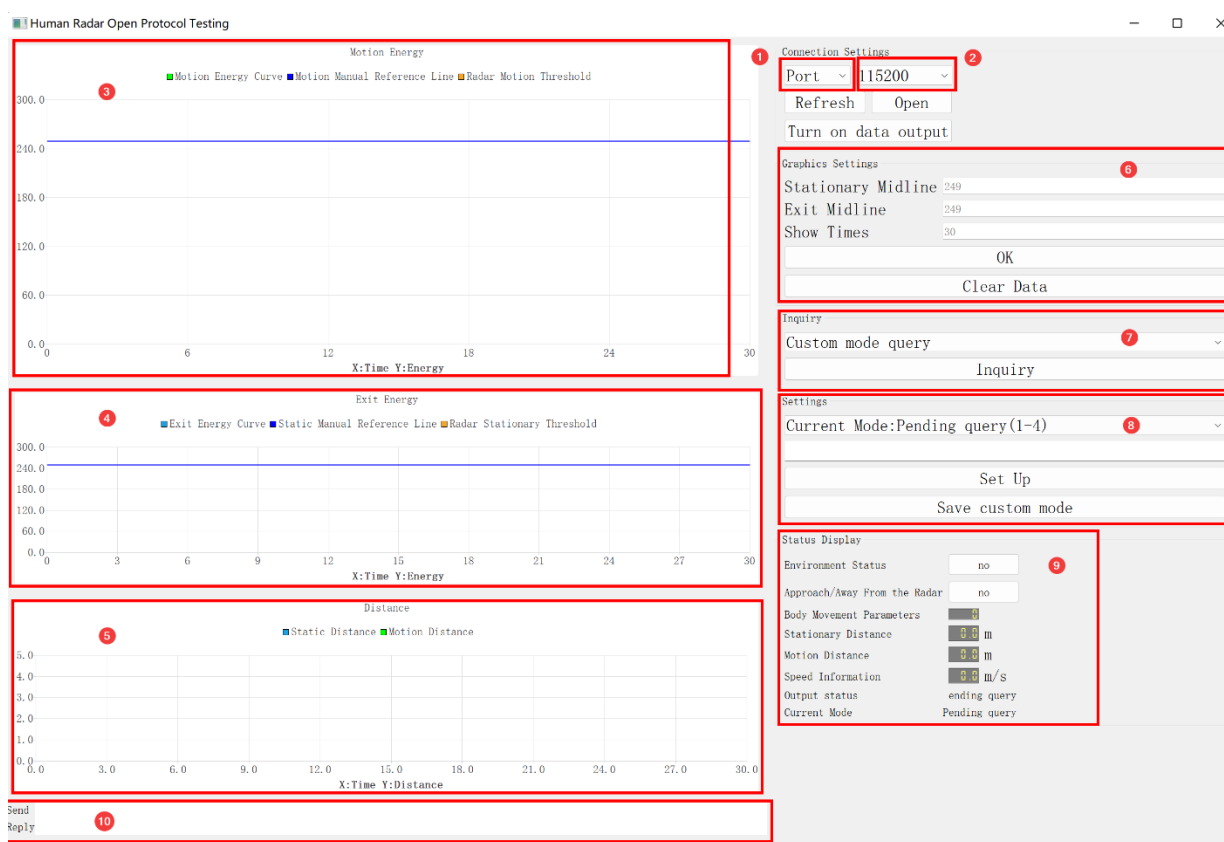


Figura 42. Software de Seeed Studio [22].

Las partes del software más importante son la 3 y la 5, que son los **gráficos de velocidades y distancias** respectivamente. Cuando la velocidad sobrepase un **umbral** configurable, significará que una persona u objeto se está moviendo (Figura 43). En el gráfico de distancia, aparece la **distancia en línea recta del objeto más cercano** al radar (Figura 44).

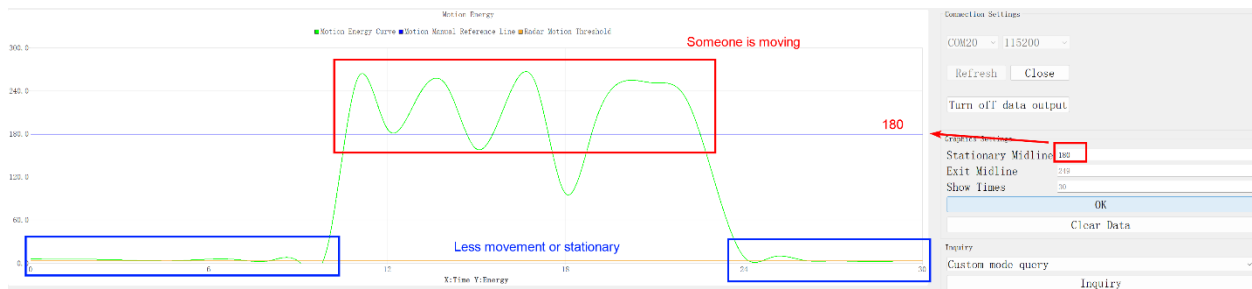


Figura 43. Gráfico de velocidades [22].

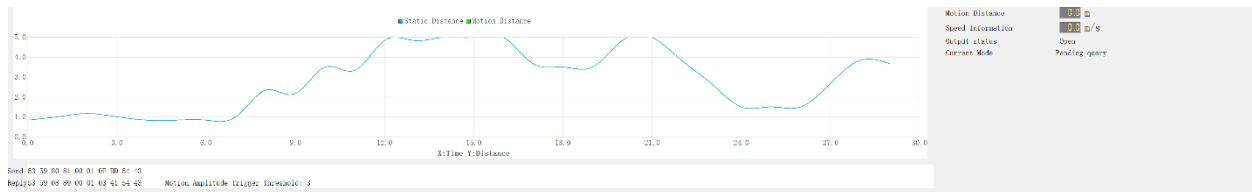


Figura 44. Gráfico de distancias [22].

4.3 Comparativa teórica y elección del fabricante de radar

Para la realización del trabajo, se han elegido los radares del **fabricante Texas Instruments**. El motivo principal por el que se han elegido es porque estos radares presentan un **algoritmo de detección más complejo**, ya que muestra una nube de puntos en el lugar donde se encuentra un objeto, en comparación con los de Seeed Studio, que muestran la distancia del objeto más cercano y utilizan un límite para detectar objetos en movimiento.

Otro de los motivos es la **gran cantidad de software y soporte que proporciona Texas Instruments**, por lo que se hace mucho más fácil desarrollar con radares de este fabricante. Además, un solo radar puede realizar **múltiples tareas** dependiendo de la demo que tenga cargada en el firmware, mientras que los de Seeed Studio están diseñados para una aplicación concreta. Por ejemplo, si se quiere medir distancia y constantes vitales, con un radar de Texas Instruments sería suficiente, simplemente habría que cargar la demo correspondiente en el firmware, mientras que los de Seeed Studio harían falta dos radares diferentes. Estos últimos serían una opción muy barata para aplicaciones concretas (como la aplicación para la salud o la detección de algún objeto por debajo de una cierta distancia o velocidad).

A pesar de que el protocolo de comunicación de ambos sea el **protocolo UART**, Los radares de Seeed Studio tienen también el inconveniente de **necesitar un módulo USB to UART**, en comparación con los radares de TI, que no requiere elementos hardware extras.

4.4 Radares mmWave elegidos

Se van a emplear **dos modelos diferentes de radar**, dependiendo del uso o de la **demo** que se vaya a cargar en el firmware.

4.4.1 Radar uRad Industrial v1.0 de Anteral

Anteral, con sede en Pamplona, Navarra **desarrolla antenas innovadoras, pasivas y tecnología radar** que cumple los exigentes requerimiento de aplicaciones en el sector aeroespacial, industria, telecomunicaciones y ciudades inteligentes, entre otras [23]. El objetivo principal es impulsar el desarrollo tecnológico a la vez que satisfacer las necesidades de la población.

El modelo de radar que se va a utilizar va a ser el **uRad Industrial v1.0 de Anteral** (Figura 45), **basado en el radar IWR6843AoP de Texas Instruments** [24] [25]. A pesar de que sea el radar uRad Industrial v1.0, **en este documento se le llamará radar IWR6843AoP**, ya que es el radar en el que está basado. La banda de frecuencia en la que trabaja es la de **60 a 64 GHz**.



Vista superior



Vista inferior



Vista lateral

Figura 45. Acabado del radar [24].

4.4.2 Radar IWR6843ISK de Texas Instruments

El otro radar que se va a utilizar es el **IWR6843ISK**, en concreto se va a utilizar su **placa de evaluación** [26]. Al igual que el otro modelo, trabaja en la **banda de frecuencia de 60-64 GHz**.

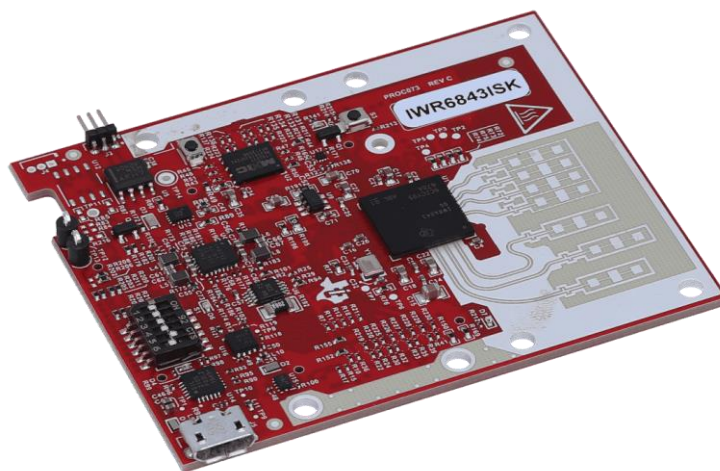


Figura 46. Radar IWR6843ISK de Texas Instruments [27].

5 Herramientas y configuración de los radares

Como ambos radares están basados en un modelo de Texas Instruments todo el software y documentación que proporciona el fabricante se debe usar a la hora de trabajar con cualquiera de ellos.

5.1 Uniflash

Uniflash es la herramienta que proporciona Texas Instruments para cargar una demo al firmware del radar. Existe tanto la versión de escritorio como la versión online [28]. De todos las demos que proporciona el fabricante se utilizarán 3 de ellas, **Out_of_the_box (OOB)**, **3D_People_Counting** y **Mobile_Tracker**. Tanto las demos como la documentación necesaria están disponibles en la sección **Resource explorer** de Texas Instruments.

Para que herramienta *Uniflash* funcione correctamente, hay que descargar el instalar los **drivers de Silicon Labs**. Para el radar hay que descargar los drivers universales disponibles en su página web [29], con el nombre "CP210x_Universal_Windows_Driver".

Tras la instalación de los drivers, ya se puede cargar un programa en la memoria flash o firmware del radar, lo que se conoce como flashear (*flashing*). El radar, al conectarse al ordenador, se conecta por **dos puertos COM**, uno de ellos sirve para **mandarle datos al radar** (Cfg_port) y el otro para **recibir los datos del radar** (DATA_port). El puerto **Enhanced es el Cfg_port**, y sirve para flashear el radar y enviarle los archivos de configuración, es decir, para enviarle datos al radar y el **puerto Standard sirve para recibir los datos** que genera el radar. El radar IWR6843AoP, se ha conectado a los puertos COM de la Figura 47 y el IWR6843ISK a los de la Figura 48 (varía dependiendo del equipo al que se conecten).

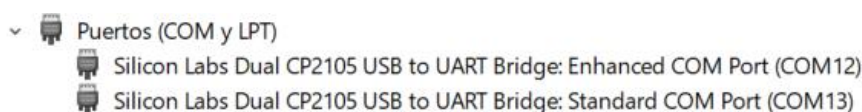


Figura 47. Puertos a los que se conecta el radar IWR6843AoP.

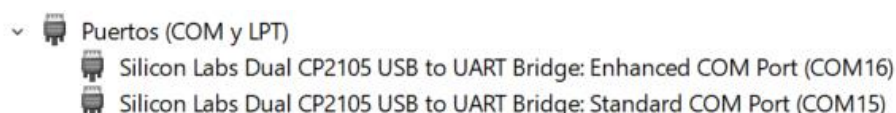


Figura 48. Puertos a los que se conecta el radar IWR6843ISK.

El radar **IWR6843AoP** tiene **4 jumpers** que se usan para seleccionar los **modos de funcionamiento** del radar, modo *flashing* y modo funcional. Para pasar de un modo a otro, basta con mover el **jumper número 1** arriba (on) para flashear y abajo (off) para el modo funcional, como se ve en la Figura 49. El radar **IWR6843ISK** tiene **6 jumpers** en la parte inferior (Figura 51). Para pasar de un modo a otro hay que colocar los jumpers como muestra la Figura 50.

SW number	SOP	Functional	Flashing
1	SOP2	OFF (low)	ON (high)
2	SOP1	OFF (low)	OFF (low)
3	SOP0	ON (high)	ON (high)
4		UNUSED	

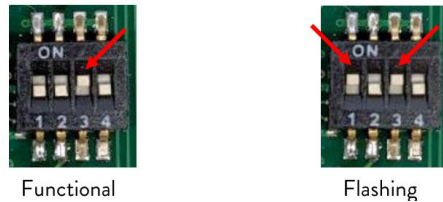


Figura 49. Posición de los jumpers para los diferentes modos de funcionamiento del radar IWR6843AoP [25].

Table 3-3. S1 Config Flashing and Functional Mode

	s1.1	s1.2	s1.3	s1.4	s1.5	s1.6
Flashing	On	Off	On	On	Off	-
Functional	Off	Off	On	On	Off	-

Figura 50. Posición de los jumpers para los diferentes modos de funcionamiento del radar IWR6843ISK [26].

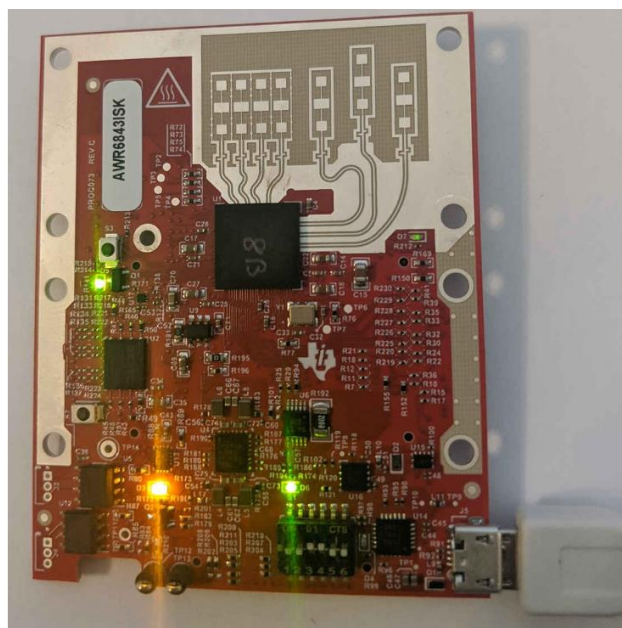


Figura 51. Jumpers del radar IWR6843ISK en posición de flashing [26].

El primer paso para utilizar *Uniflash* es **seleccionar el radar**. El programa puede detectarlo automáticamente, o se puede introducir el modelo del radar a mano. En este caso se va a hacer con el modelo **IWR6843AoP**, ya que el **procedimiento para el IWR6843ISK es idéntico**.

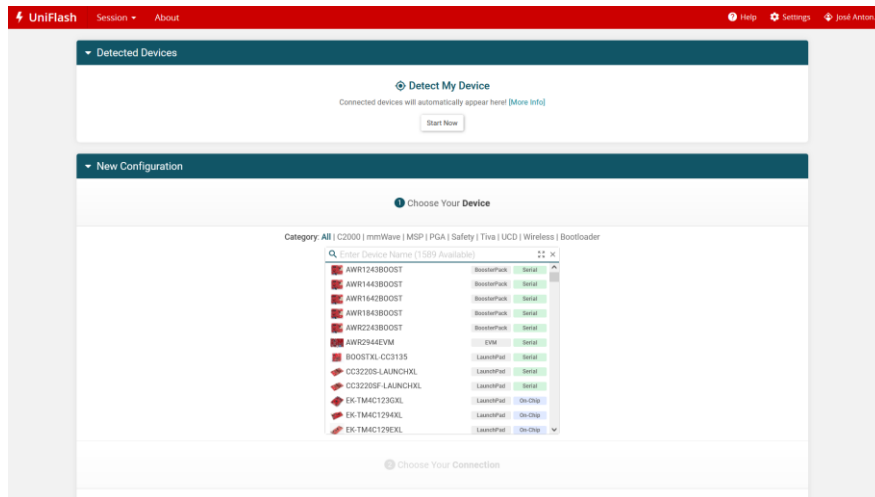


Figura 52. Selección de dispositivo en Uniflash [28].

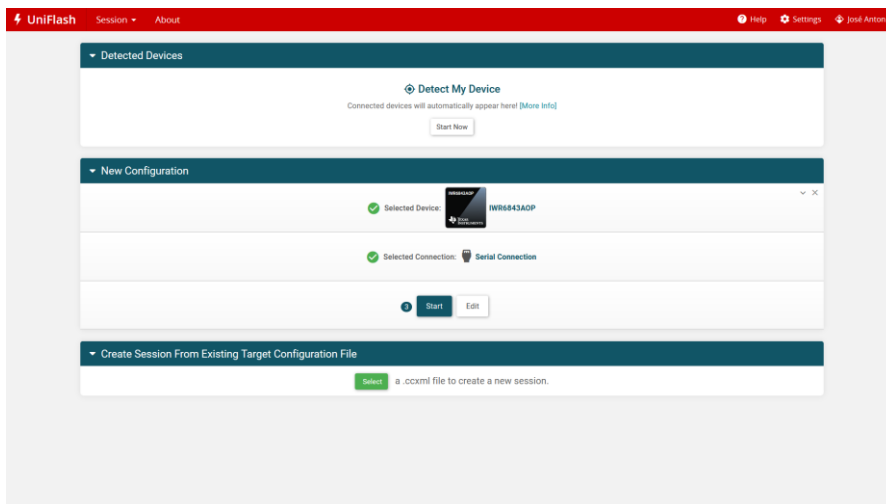


Figura 53. Dispositivo seleccionado en Uniflash [28].

Una vez seleccionado el radar, hay que **configurar los puertos** en *Uniflash* en el apartado “*Settings & Utilities*”. Hay que **conectarse al puerto Enhanced**, que en este caso es el puerto COM12 (Figura 47).

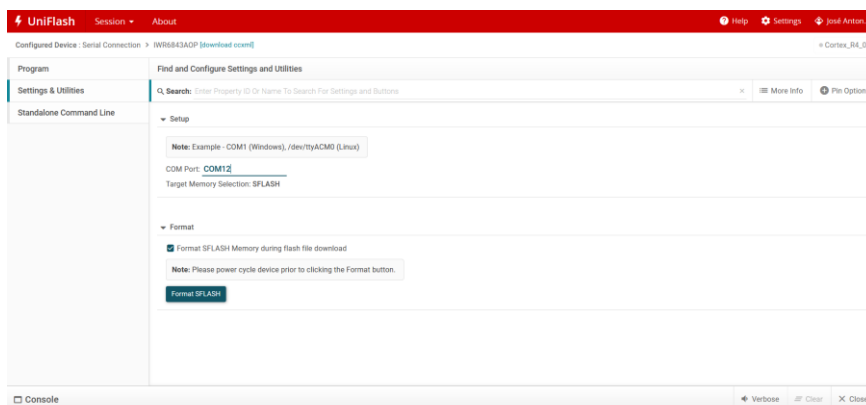


Figura 54. Configuración del puerto en Uniflash [28].

Después de esto, en el apartado “Program” hay que **cargar el binario** en “Meta Image 1” pinchando en el botón “k”.

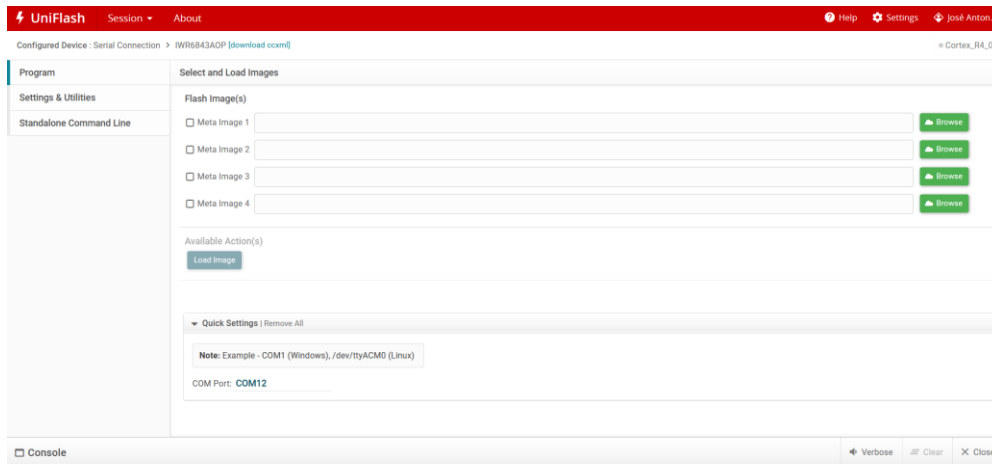


Figura 55. Apartado para la selección de binario [28].

Una vez seleccionado el binario, hay que darle a “**Load Image**” para que el programa actualice el firmware del radar.

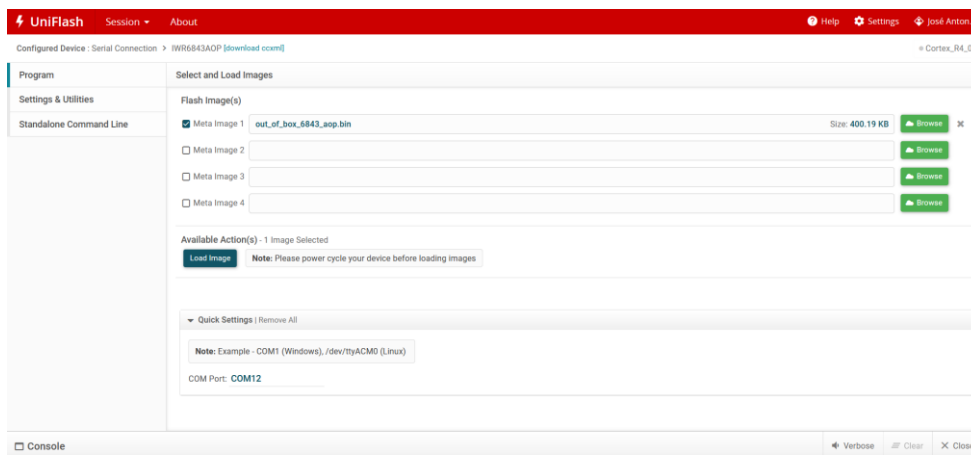


Figura 56. Binario seleccionado [28].

Si todo se ha hecho bien, aparecerá en la consola un mensaje diciendo que se ha **completado el proceso correctamente**.

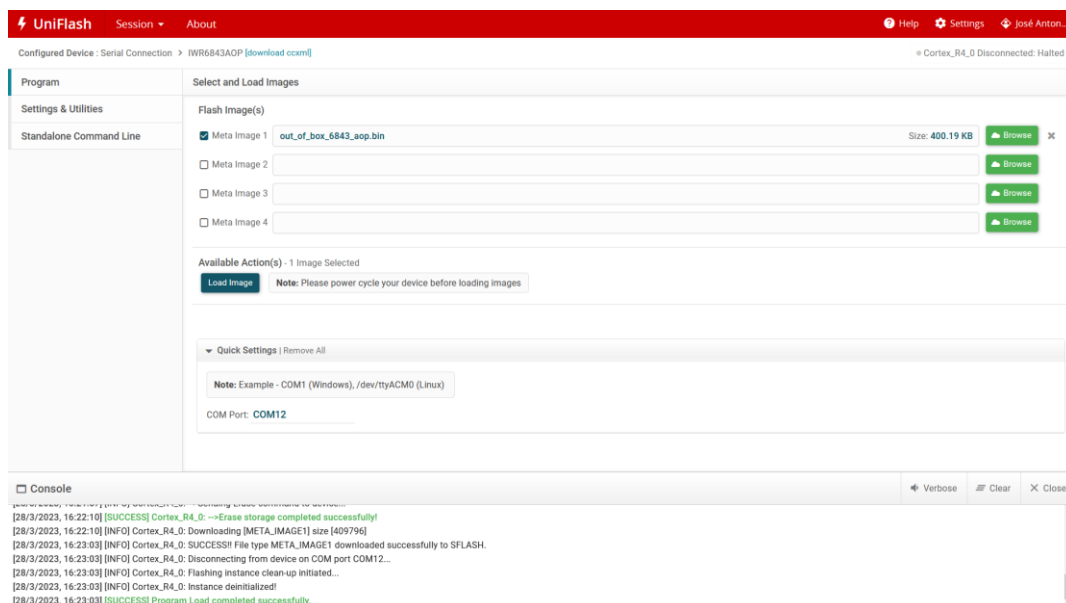


Figura 57. Binario cargado correctamente [28].

5.1.1 Demo Out_of_the_box (OOB)

Esta es la **demo más básica de todas** ya que únicamente **devuelve la nube de puntos que detecta**. Es la demo perfecta para comprender el funcionamiento completo del radar (Parámetros, configuraciones, chirp, etc.) antes de utilizar otra demo más compleja.

5.1.2 Demo 3D_People_Counting

Esta es una de las demos que **se utilizará para el desarrollo del trabajo**. En esta demo, el radar se coloca en una posición elevada y estática y, como su propio nombre indica, **se encarga de detectar, hacer tracking o seguimiento** para contar el número de personas que pasan por debajo. En este caso se utilizará esta demo para la **detección de cualquier tipo de objetos**, no solo de personas. Muestra una nube de puntos y, como se ha mencionado anteriormente, hace un seguimiento o tracking de objetos en movimiento.

La demo dispone de una **guía de usuario o guía de implementación** [30], una **guía para modificar los parámetros de la capa de detección** [31], una **guía para modificar los parámetros de la capa de tracking** [32] y un documento para el tracking de objetivos con múltiples puntos de reflexión (no se usará este documento) [33].

5.1.3 Demo Mobile_Tracker

Esta es la segunda de las demos que **se utilizará para el desarrollo del trabajo**. Esta demo está preparada para que el radar esté en movimiento, **detecte y haga tracking** de los objetos que se encuentra. Al igual que la demo 3D_People_Counting, muestra una nube de puntos y hace un seguimiento o tracking de objetos en movimiento.

La demo dispone de una **guía de usuario o guía de implementación** [34]. Los parámetros que utiliza son exactamente los mismos que los de 3D_People_Counting, por lo que para modificar los parámetros hay que seguir los documentos de esa demo.

5.2 Mmwave Demo Visualizer

Otra herramienta de Texas Instruments es Mmwave demo visualizer [35]. Esta herramienta sirve tanto para **visualizar toda la información que devuelve el radar** como modificar alguna configuración, en la mayor parte de los casos, mediante desplazables. Cabe destacar que esta herramienta **solo funciona para la demo Out_of_the_box**. Una vez configurados los puertos y seleccionado el dispositivo correcto, en el apartado configure, se pueden ajustar algunos valores de la escena y ajustes de los diagramas o gráficos, como se puede ver en la Figura 58. Al igual que para Uniflash, **se hará para el radar IWR6843AoP**. Para el otro modelo de radar únicamente hay que cambiar los *Setup Details*, como se muestra en la Figura 59.

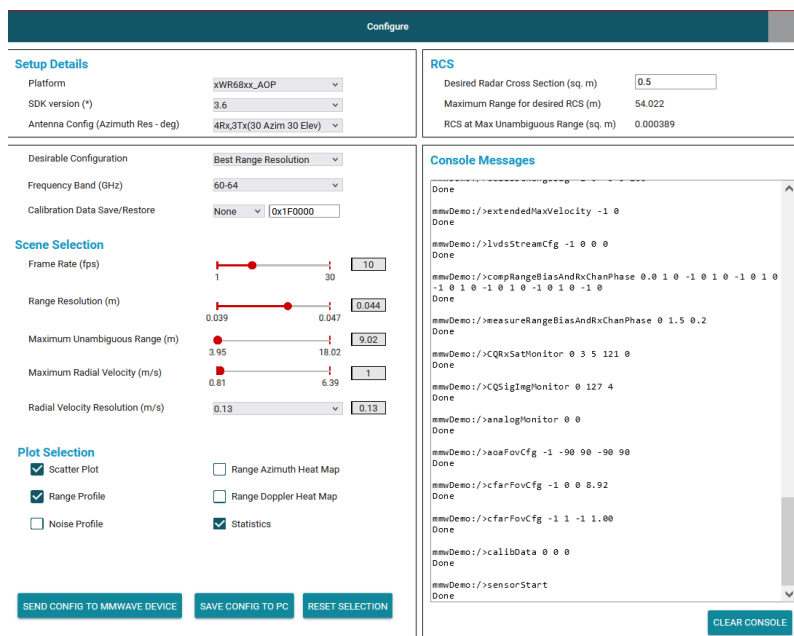


Figura 58. Ajustes seleccionables antes de iniciar el radar IWR6843AoP [35].

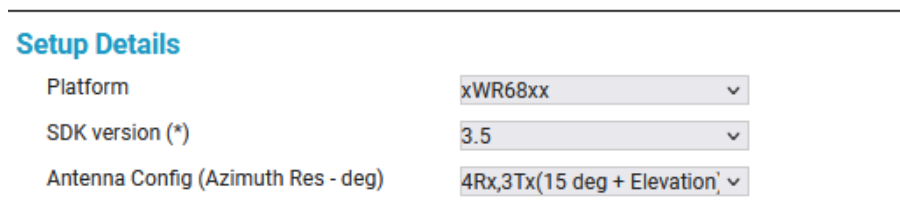


Figura 59. Setup Details para el radar IWR6843ISK [35].

Una vez seleccionados los ajustes, se le manda la configuración al radar para que se ponga en funcionamiento pinchando en el botón “Send config to mmwave device”. En el apartado *plot* se pueden observar **gráficos realizados a partir de los datos** que proporciona el radar. Se pueden visualizar varios tipos de gráficos (Figura 60), el más importante a la hora de entender el funcionamiento es el *3D Scatter Plot*, que muestra los puntos que el radar ha detectado en un sistema en 3 dimensiones. Se pueden ver otros gráficos no tan importantes como los de las velocidades, mapas de calor, etc. Algunos de ellos se pueden ver en la Figura 60.

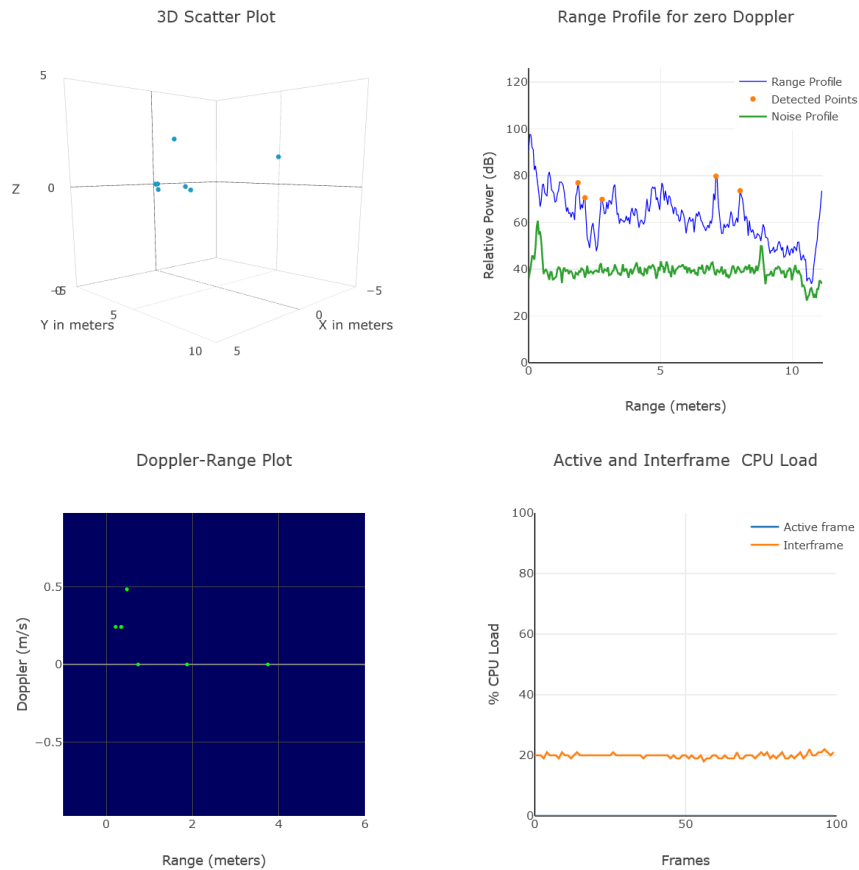


Figura 60. Algunos gráficos disponibles en mmwave Demo Visualizer [35].

En la parte derecha, se encuentran algunos valores estadísticos del radar, configuración del chirp y de la escena y algunas opciones y parámetros de configuración que pueden ser variados mientras el radar se encuentra en funcionamiento, como, por ejemplo, limitar el rango máximo del radar. Esto quiere decir que, aunque el rango máximo detectable sea, por ejemplo, 10 metros, se puede limitar la distancia máxima a la que se muestre un objeto mediante software. Otra de ellas es el agrupamiento o **peak grouping**. Cuando esta opción está activada, en vez de mostrar todos los puntos vecinos, se muestra únicamente uno de ellos, reduciendo así el número total de puntos (Figura 61).

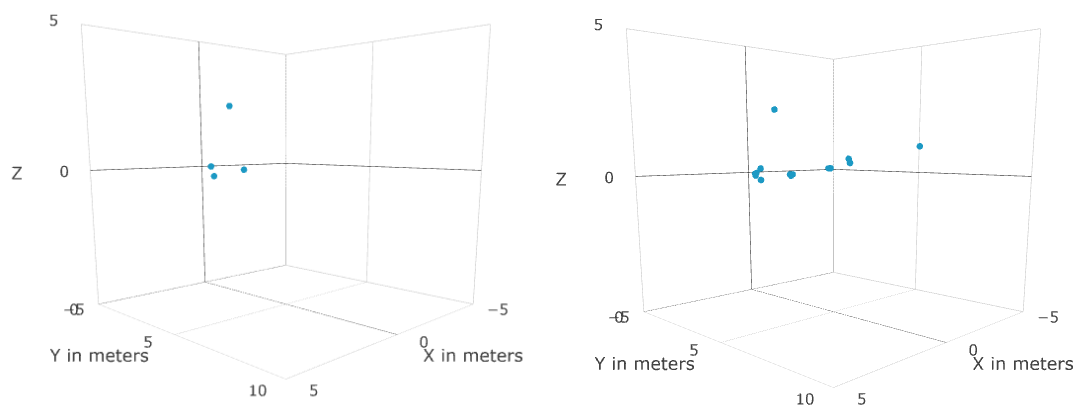


Figura 61. Peak grouping activado (izquierda) y desactivado (derecha) [35].

En resumen, *Mmwave Demo Visualizer* es una **herramienta ideal para entender cómo funciona el radar**, ya que se pueden configurar algunos parámetros y ver qué ocurre modificando cada uno de esos desplazables.

5.3 Comandos de configuración de la demo Out_of_the_box

Los comandos que se envían al radar en la herramienta *mmwave Demo Visualizer* se encuentran en un **archivo de configuración**. Los parámetros del archivo de configuración (archivo .cfg) se pueden ver en la parte derecha de la Figura 58 o descargar la configuración completa pinchando en el botón “*save config to PC*”. El archivo consiste en unas cuantas **líneas informativas** (comentadas mediante un “%” al principio de la línea) y una serie de **comandos** que contienen una serie de **parámetros para la configuración** del radar. La lista completa de comandos y parámetros, y su función está descrita en el manual de usuario del *mmwave Software Developer Kit* (mmwave SDK) [36]. En este apartado se **nombrarán todos los comandos del archivo de configuración de OOB y se describirán en detalle los más importantes** (los que serán necesarios modificar según los requerimientos de la situación).

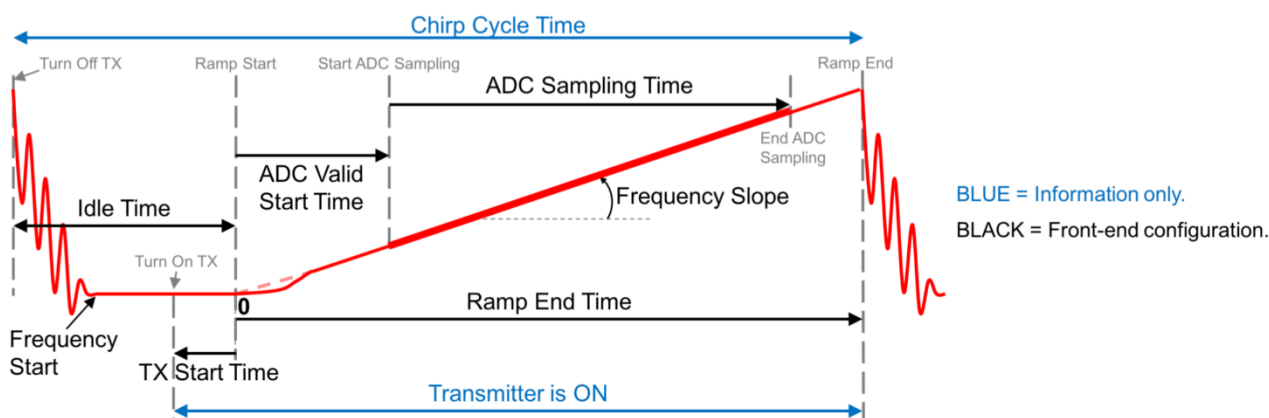


Figura 62. Parámetros de un chirp [36].

Hay una **medida de tiempo más que no está representada en la Figura 62**, el **excess time** o tiempo de exceso. Este tiempo es el que transcurre desde que se acaban de tomar las muestras (final de *ADC Sampling Time*) hasta que la rampa de frecuencia llega a su punto máximo (principio del idle time del siguiente *chirp*).

- **SensorStop: Detiene el radar.** No tiene parámetros.
- **flushCfg:** Comando que se le tiene que enviar al radar para **cargar una nueva configuración**. No tiene parámetros.
- **dfeDataOutputMode:** Configura el **modo de funcionamiento del radar**, en este caso **tendrá el valor 1** (chirps basados en frames).
- **channelCfg:** Se usa para **seleccionar el número de antenas** del radar.

channelCfg		
Número	Parámetro	Descripción
1	<rxChannelEn>	Máscaras para habilitar el número de antenas receptoras <ul style="list-style-type: none"> • 0001b =1: 1antena • 0011b =3: 2 antenas • 0111b =7: 3 antenas • 1111b =15: 4 antenas
2	<txChannelEn>	Máscaras para habilitar el número de antenas transmisoras <ul style="list-style-type: none"> • 0001b =1: 1 antena

		<ul style="list-style-type: none"> • 0011b =3: 2 antenas • 0111b =7: 3 antenas • 1111b =15: 4 antenas
3	<cascading>	No aplicable, dejarlo siempre a 0 .

Tabla 1. Parámetros del comando channelCfg.

- adcCfg: Configuración **software del convertor analógico-digital**.

adcCfg		
Número	Parámetro	Descripción
1	<numADCbits>	Selección del número de bits de cada muestra : <ul style="list-style-type: none"> • 0: 12 bits • 1: 14 bits • 2: 16 bits (Solo esta opción está soportada)
2	<adcOutputFmt>	Formato de salida : <ul style="list-style-type: none"> • 0: Real (No soportado) • 1: Complejo 1X • 2: Complejo 2X (Se usa cuando hay interferencias, ya que la mitad de las muestras son ocupadas por elementos de ruido o interferencias)

Tabla 2. Parámetros del comando adcCfg.

- adcbufCfg: Configuración **hardware del convertor analógico-digital**.
- profileCfg: En este comando se configuran algunos de los **parámetros del chirp**, que serán claves a la hora de calcular los parámetros de entorno.

profileCfg		
Número	Parámetro	Descripción
1	<profileID>	ID del perfil .
2	<startFreq>	Frecuencia de inicio en GHz (f_0).
3	<IdleTime>	Idle time en μs (T_{idle}).
4	<adcStartTime>	Tiempo de inicio de muestreo en μs (T_{ADC}).
5	<rampEndTime>	Ramp end time en μs (T_{ramp}).
6	<txOutPower>	Back-off en dB de la potencia de la antena transmisora (Pt) . Quiere decir que para 0, la potencia es 12dB. Para un valor de 1, la potencia es 11 dB [37]. El valor se debe calcular en hexadecimal por cada antena receptora y escribirlo en decimal en el archivo de configuración. Por ejemplo, para 3 antenas receptoras el valor en hexadecimal será de 6 bytes, 0x543210, donde los bytes 4 y 5 son los valores para la antena 1, los bytes 2 y 3 para la dos y los bytes 0 y 1 para la antena 3.
7	<thPhaseShifter>	Variador de la fase del chirp para las antenas transmisoras. Recomendado ponerlo a 0.
8	<freqSlopeConst>	Pendiente de la rampa en $\text{MHz}/\mu\text{s}$ (S).
9	<txStartTime>	Tx start time en μs (T_{start}).
10	<numADCSamples>	Número de muestras recogidas durante ADC Sampling time (N_{ADC}).
11	<digOutSampleRate>	Frecuencia de muestreo en ksp/s ($\text{ADC}_{\text{sampling}}$).
12	<hpfCornerFreq1>	Frecuencia de corte del filtro paso alto 1.
13	<hfpCornerFreq2>	Frecuencia de corte del filtro paso alto 2.

14	<rxGain>	Parámetro de ganancia de la antena receptora
----	----------	---

Tabla 3. Parámetros del comando *profileCfg*.

- *chirpCfg*: **Configuración de chirps** (un comando por cada antena transmisora).
- *lowPower*: Configuración del modo de **baja potencia**.
- *frameCfg*: **Configuración del frame**.

frameCfg		
Número	Parámetro	Descripción
1	<i>chirp start index</i>	Índice del chirp inicial
2	<i>chirp end index</i>	Índice del chirp final
3	<i>number of loops</i>	Número de chirps por frame (N_c)
4	<i>number of frames</i>	Número de frames, se deja a 0.
5	<i>frame periodicity</i>	Periodicidad del frame en μ s
6	<i>trigger select</i>	Cómo se le indica al radar que debe iniciar un nuevo frame, conocido como trigger . 1 si se activa mediante software y 0 si es mediante hardware.
7	<i>Frame trigger delay</i>	Retraso del “trigger” en ms.

Tabla 4. Parámetros del comando *frameCfg*.

- *guiMonitor*: Con este parámetro se elige el **tipo de datos que devuelve OOB** (Temperatura, datos estadísticos, mapas de calor, etc.).
- *cfarCfg*: Sirve para configurar los **algoritmos CFAR** que se explicarán en el apartado 5.5.
- *multiObjBeamForming*.
- *calibDcRangeSig*.
- *clutterRemoval*: Sirve para **eliminar el denominado *static clutter*** (eliminar los objetos estáticos).
- *aoaFovCfar*: **Filtrado de puntos** fuera de unos límites de **acimut y elevación**.
- *cfarFovCfg*: **Filtrado de puntos** fuera de unos límites de **rango y velocidad**.
- *compRangeBiasAndRxChanPhase*: Parámetros de ganancia, **depende de la geometría de las antenas**.
- *measureRangeBiasAndRxChanPhase*.
- *extendedMaxVelocity*.
- *CQRxSatMonitor*.
- *CQSigImgMonitor*.
- *analogMonitor*.
- *lvdsStreamCfg*.
- *bpmCfg*.
- *Calibdata*.
- *SensorStart*: Comando empleado para **iniciar el radar** después de cargar la configuración. No tiene parámetros.

5.4 Efecto de los *.cfg* en los *chirps* y los parámetros de entorno

Una vez entendido el significado de los parámetros de configuración, se pueden utilizar para **calcular de forma analítica los parámetros de la escena** (rango máximo, resolución, etc.). Antes de calcular los parámetros de la escena, hay que describir algunos elementos importantes, el **Radar Cross-Section (RCS)**, el **Signal to Noise Ratio (SNR)** y el **Noise Figure (NF)**.

5.4.1 Radar Cross-Section (RCS)

Se define el *Radar Cross-Section (RCS)* de un objeto como el **área en la que se recibe una cierta energía y, en condiciones de reflectividad perfecta e isotrópicas** (de igual forma en todas las direcciones), **la energía reflejada es la misma que la recibida** [38]. Se asume que el radar está situado a una distancia R del objeto, que la cantidad de densidad de energía (*Power density*) recibida por el objeto es P_{Di} , la cantidad de energía reflejada por el objeto P_r es:

$$P_r = \sigma P_{Di}$$

Ecuación 13

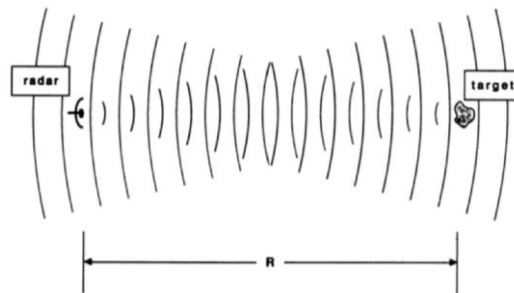


Figura 63. El radar emite ondas que rebotan en el objetivo y vuelven a él [39].

donde σ es el RCS del objeto. Se define a su vez P_{Dr} como la densidad de energía reflejada a la antena receptora, asumiendo un comportamiento isotrópico y sin pérdidas, a una distancia R la densidad de energía es radiada de forma esférica, obteniendo la siguiente expresión:

$$P_{Dr} = \frac{P_r}{4\pi R^2}$$

Ecuación 14

Combinando las dos ecuaciones anteriores:

$$\sigma = 4\pi R^2 \frac{P_{Dr}}{P_{Di}}$$

Ecuación 15

Esta expresión, también puede ser expresada como **cocientes de energía** [39]. La impedancia intrínseca del medio, Z_0 , se puede expresar como:

$$Z_0 = \frac{E}{H}$$

Ecuación 16

donde E y H son las amplitudes de los campos vectoriales que componen la onda alineados correctamente. La admitancia intrínseca del medio, Y_0 , es el inverso de Z_0 , es decir, el inverso de la Ecuación 16:

$$Y_0 = \frac{H}{E}$$

Ecuación 17

La densidad de energía de una onda electromagnética sigue la siguiente ecuación, suponiendo que ambos campos están en fase (perpendiculares entre sí y a la dirección de movimiento).

$$W = \frac{EH}{2}$$

Ecuación 18

Sustituyendo la Ecuación 16 y Ecuación 17 en la Ecuación 18, se obtiene la siguiente expresión:

$$W = \frac{E^2 Y_0}{2} = \frac{H^2 Z_0}{2}$$

Ecuación 19

Esta densidad energética es a la que antes se ha llamado P_{Dr} y P_{Di} , por lo tanto:

$$P_{Dr} = \frac{|E_r|^2 Y_0}{2}$$

Ecuación 20

$$P_{Di} = \frac{|E_i|^2 Y_0}{2}$$

Ecuación 21

Sustituyendo las ecuaciones anteriores en la Ecuación 15, se obtiene la siguiente expresión:

$$\sigma = 4\pi R^2 \frac{|E_r|^2}{|E_i|^2}$$

Ecuación 22

En cualquiera de las dos expresiones anteriores sobre el RCS, este es **función de la distancia del radar al objeto**. Debido a que la distancia es conocida para la mayoría de los ensayos o pruebas de RCS, es una variable independiente que se quiere eliminar de la ecuación. Se puede eliminar de la ecuación imponiendo que la onda incidente sea una onda plana, lo que significa que la ecuación anterior se escribe de la siguiente forma:

$$\sigma = \lim_{R \rightarrow \infty} 4\pi R^2 \frac{|E_r|^2}{|E_i|^2}$$

Ecuación 23

A pesar de lo anterior, **esta ecuación no puede ser utilizada** para la toma de mediad del RCS, ya que actualmente no existe ningún dispositivo capaz de colocarse a una distancia infinita y recolectar información útil de energía reflejada. Para poder obtener el valor del RCS de otra forma, en este caso, se realizará la **medida**

del RCS a través de la ecuación del rango de distancia [40]. Para el caso **cuasi-monoestático**, la antena emisora (T_x) y receptora (R_x) se encuentran muy juntas espacialmente, en el caso **monoestático**, se usa la misma antena para la transmisión y la recepción y en el caso **biestático**, las dos antenas se encuentran separadas. La potencia recibida por la antena receptora (P_r) se puede obtener mediante la siguiente expresión:

$$P_r = \frac{P_t G_t G_r \sigma \lambda^2}{(4\pi)^3 R^4}$$

Ecuación 24

donde P_t es la potencia transmitida, G_t y G_x son las ganancias de la antena transmisora y receptora respectivamente, λ es la longitud de onda, σ es el RCS y R es la distancia al objeto. **Todos los elementos de la ecuación excepto el RCS son conocidos**, y la potencia reflejada se puede **obtener de forma experimental**, por lo tanto, se puede calcular el RCS. Si el experimento se lleva a cabo en una **cámara anecoica** la expresión de la potencia recibida es la Ecuación 24, de lo contrario **habría que multiplicar la expresión anterior por $1/L$** , donde L representa las pérdidas de la señal por pérdidas de espacio libre.

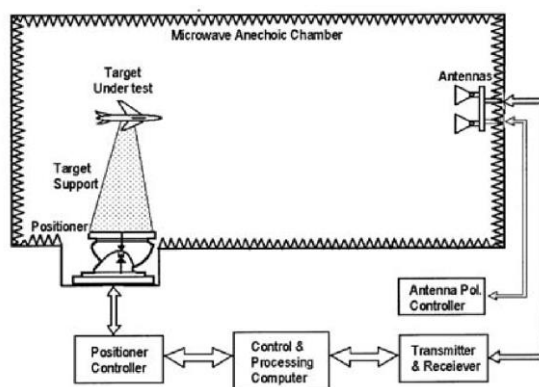


Figura 64. Esquema de una cámara anecoica para la medida de RCS [41].

Para los casos donde la antena transmisora y receptora sea la misma, el término $G_t G_x$ se sustituye por la ganancia de la antena elevada al cuadrado (G^2) [38]. El objeto se puede **colocar sobre una plataforma giratoria** para hacer que el objeto rote tanto en el plano azimutal como en el plano cenital y para ver como varía el RCS dependiendo de la orientación del objeto [40].

Si solo se realiza movimiento alrededor de un eje, se puede usar un gráfico como el de la Figura 65, y **si se rota el objeto en 2 ejes, sería conveniente utilizar un mapa de calor** (Figura 66) para poder ver el RCS para cualquier orientación posible, pero alargaría mucho el proceso de medición.

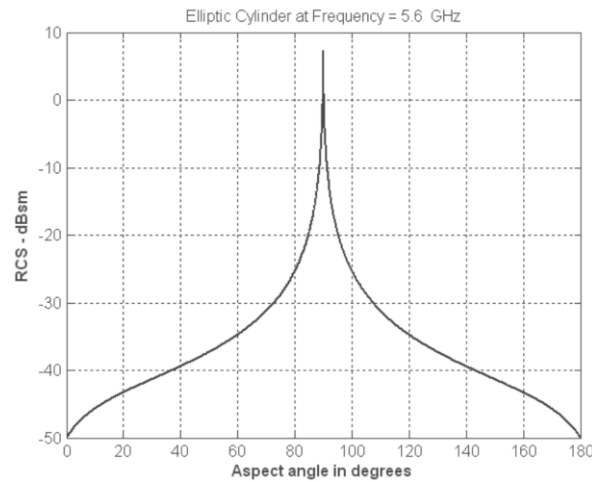


Figura 65. Medida del RCS rotando alrededor de un eje [38].

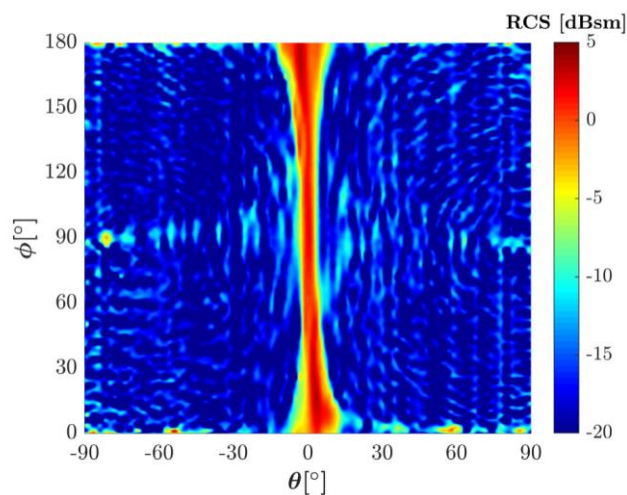


Figura 66. Medida del RCS rotando alrededor de dos ejes [40].

Para saber si las medidas del RCS son correctas, es recomendable realizar una **calibración con un objeto del que se sepa su RCS**. Por ejemplo, se puede utilizar un trapecio de aluminio, del que previamente se ha calculado su RCS mediante simulación electromagnética con el **software CST Microwave Studio**.

Hay que **tener en cuenta las distancias** a la hora de realizar las **medidas del RCS en una cámara anecoica**. La distancia R entre la antena y el objeto debe ser lo suficientemente grande para que **el objeto entero entre en el haz de la antena** (*antenna beam*).

$$R \geq \frac{D}{2 \tan\left(\frac{\theta}{2}\right)}$$

Ecuación 25

Este criterio sigue la Ecuación 25, donde D es la dimensión de la antena y θ es el ancho del haz a media potencia o *half-power beamwidth* de la antena, que es la separación angular en el que la potencia emitida (del haz principal) es menor del 50% de la potencia máxima (o decrece 3 dB).

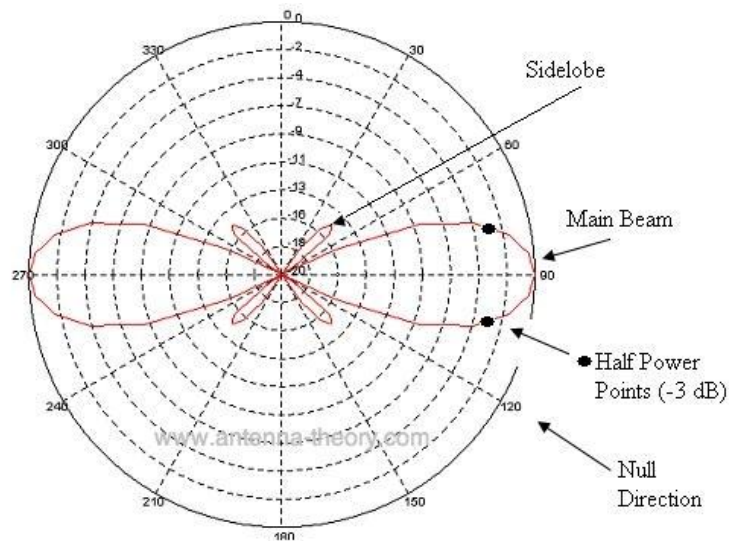


Figura 67. Ancho de haz a media potencia [42].

El ancho de banda a media potencia se puede calcular mediante la siguiente expresión [43]:

$$\theta = 70\lambda/\phi_{antena}$$

Ecuación 26

El otro aspecto para tener en cuenta es la **distancia del objeto a la antena para considerarse far field**. Generalmente se escoge una distancia que cumpla la Ecuación 27, donde en este caso D es la dimensión del objeto o de la antena, la que sea mayor.

$$R_{FF} = \frac{2D^2}{\lambda}$$

Ecuación 27

Dado que **la respuesta del objeto depende del tamaño geométrico y de la longitud de onda de la señal**, λ , los parámetros del radar se eligen teniéndolos en cuenta [8]. Por ejemplo, para una esfera de radio a, para $\lambda \ll a$, las respuestas son de alta frecuencia o **naturaleza óptica**, para λ aproximadamente a, se llaman **respuestas resonantes** y para $\lambda \gg a$ de baja frecuencia o respuestas **Rayleigh**.

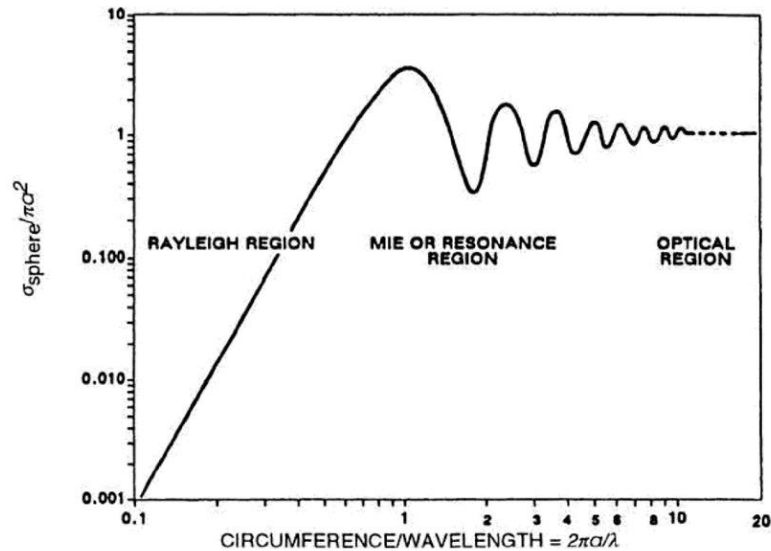


Figura 68. Tipos de respuestas [8].

Para obtener una respuesta de tipo óptico, se debe cumplir la Ecuación 28 donde c es la velocidad de la luz.

$$f_h \gg \frac{c}{a} \gg f_l$$

Ecuación 28

Para el siguiente análisis y las siguientes expresiones se asume medidas de RSC en campo abierto (o *far field*) en la **región óptica**, es decir, cuando **la longitud de onda sea mucho menor que el tamaño del objeto**, que ocurre en la mayoría de los casos. Cuando se trata de **objetos simples** (esferas, elipses, placas planas rectangulares, etc.), el cálculo del RCS se puede llevar a cabo de una manera sencilla **analíticamente** [38]. El cálculo de estos valores se puede ver en el **Anexo** (apartado 11). Cuando un objeto complejo se basa en la **coherente combinación de los RCS de los objetos sencillos** que lo componen, puede modelarse como un conjunto de objetos sencillos.

Cuando el objeto no es simple ni modelable a múltiples objetos sencillos, es decir, se trata de **objetos complejos** (que no se compongan de objetos sencillos), como aviones, barcos, etc., se debe recurrir al cálculo predictivo o por ordenador: Los métodos para predecir el RCS de objetos son **muy complejos**, incluso para objetos sencillos ya que requiere resolver **complejas ecuaciones diferenciales o integrales** (basadas en las ecuaciones de Maxwell) que describan la reflexión de ondas. Debido a las dificultades asociadas con la obtención exacta del RCS, existen **diversos métodos de aproximación**. La mayoría de esos métodos son **válidos en la región óptica**, y cada uno tiene sus ventajas y desventajas. Los errores que se cometen son lo suficientemente pequeños para que los resultados puedan ser útiles para ingenieros y desarrolladores de tecnología radar. Para este tipo de objetos **no hay forma analítica** de calcular o predecir el RCS. El campo de **Computational electromagnetics (CEM)** usa la potencia de los ordenadores para implementar las ecuaciones de Maxwell y resolver los problemas. Existen múltiples técnicas CEM para la resolución de estos problemas. De todos los métodos CEM que existen, dos de ellos son *Finite Difference Time Domain (FDTD)* y Método de elementos finitos (FEM).

Finite Difference Time Domain (FDTD) es un método que usa **ecuaciones de derivación numérica** para resolver las ecuaciones de Maxwell en el dominio del tiempo. **El objeto y sus alrededores deben ser discretizados** para

simular el objeto aislado, sin paredes a su alrededor. Por ejemplo, para la derivada primera de una función se pueden usar varias ecuaciones: la ecuación de la derivación numérica de dos puntos progresiva (Ecuación 29) la ecuación de la derivación numérica de dos puntos regresiva (Ecuación 30) y la ecuación de la derivación numérica de dos puntos central (Ecuación 31):

$$\frac{df(x_0)}{dt} = \dot{f}(x_0) = \frac{f(x_0 + \Delta x) - f(x_0)}{\Delta x}$$

Ecuación 29

$$\dot{f}(x_0) = \frac{f(x_0) - f(x_0 - \Delta x)}{\Delta x}$$

Ecuación 30

$$\dot{f}(x_0) = \frac{f(x_0 + \Delta x) - f(x_0 - \Delta x)}{2\Delta x}$$

Ecuación 31

Las derivadas de segundo orden y orden superior se pueden calcular mediante un **proceso similar** al de las derivadas de primer orden

El método de **elementos finitos (FEM)** es una técnica común para la aproximación de soluciones a ecuaciones diferenciales muy complejas. Al igual que en el método anterior, **el objeto y sus alrededores deben ser discretizados** para simular el objeto aislado. Se construye una solución aproximada usando una **combinación lineal de funciones** en la región de estudio.

Como se ha explicado anteriormente, existe una herramienta bastante común para el cálculo aproximado del RCS de forma rápida, **CST Microwave Studio**, que es una herramienta de **simulación 3D para la resolución de problemas con ondas de alta frecuencia**. Este programa utiliza simulaciones mediante **trazado de rayos (ray tracing)**, que son mucho más rápidas que la resolución de ecuaciones diferenciales o integrales y se puede conseguir **la misma precisión para objetos no muy complejos** como paralelepípedos.

5.4.2 Signal to Noise Ratio (SNR)

Para una mejor actuación del radar, **el sistema debe ser capaz de distinguir entre la señal recibida e ignorar el ruido de fondo** [44]. Hay un concepto conocido como *Signal to Noise Ratio* (SNR), es la **diferencia entre la señal recibida y el noise floor**, que es simplemente ruido que se cuela del exterior.

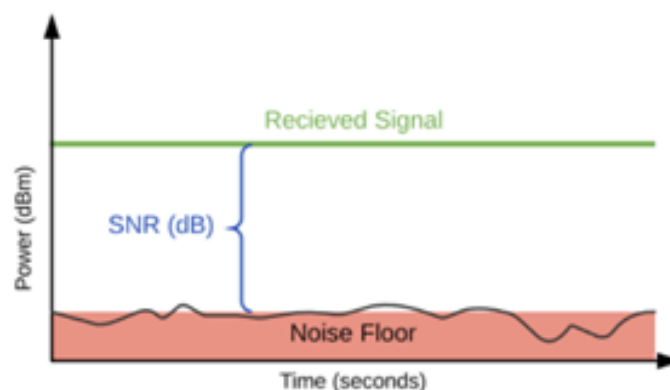


Figura 69. Signal to Noise Ratio (SNR) [44].

Por ejemplo, si llega una señal a -75 dBm y el *noise floor* es -90 dBm, el SNR efectivo es 15 dBm. Esto se reflejará una señal con una intensidad de 15dBm. **Cuanto más alejada sea la señal del noise floor, será de mejor calidad.** Las señales cerca del *noise floor* pueden resultar en datos corruptos o erróneos.

El nivel de **ruido o noise floor** se puede calcular fácilmente mediante la siguiente ecuación, donde k es la constante de Boltzmann, T es la temperatura de la antena, NF es el *Noise Figure* y IF_{max} el máximo ancho de banda [45]. Estos dos últimos valores se explicarán en los apartados 5.4.3 y 5.4.4.a, respectivamente.

$$Noise_{floor} = k \cdot T \cdot NF \cdot IF_{max}$$

Ecuación 32

5.4.3 Noise Figure (NF)

El *Noise Figure* o figura de ruido indica **cuanto es la degradación del SNR causada por los componentes** durante el procesamiento de datos [46]. Es la **diferencia en decibelios** de la señal de ruido a la salida del elemento y la señal de ruido del mismo elemento ideal (misma ganancia y ancho de banda). Esto incluye interferencias externas que afecten a los elementos, así como el ruido blanco. En el caso de los radares empleados, **este valor viene dado en la hoja de especificaciones.**

5.4.4 Cálculo analítico de parámetros

A continuación, se realizará un **análisis teórico de los parámetros de entorno** (Rango máximo, resolución, etc.) **y otros parámetros importantes** en función de la configuración de los *chirps*. Las ecuaciones de los parámetros de entorno se encuentran en la guía para la configuración de los *chirps*, de Texas Instruments [47]. También se puede encontrar la demostración de la mayoría de los parámetros en [1]. Los parámetros configurables se describieron en el apartado 5.3. Posteriormente, en el apartado 7.1, se **verificarán experimentalmente** las ecuaciones.

5.4.4.a Rango máximo limitado por parámetros del radar

La distancia máxima de visión **puede estar limitada bien por la frecuencia de los pulsos o beat frequency (IF_{max}) o bien por parámetros de ruido.** En este apartado se explicará la distancia máxima **limitada por la frecuencia de los pulsos**, que sigue la siguiente ecuación:

$$Rango_{max} = \frac{IF_{max} \cdot c}{2S}$$

Ecuación 33

La frecuencia de los pulsos (IF_{max}) es dependiente de la **frecuencia de muestreo ($ADC_{sampling}$)**. En el modo de muestreo complejo 1X, IF_{max} sigue la Ecuación 34 y en el modo complejo 2X, la Ecuación 35, donde $ADC_{sampling}$ es la frecuencia de muestreo. **Los modos de muestreo son configurables** con el comando `adcCfg` con el parámetro `<adcOutputFmt>`, en este caso, se usa la 1X. S es la pendiente del *chirp* y c es la velocidad de la luz ($3 \cdot 10^8$ m/s).

$$IF_{max} = 0.8 \cdot (ADC_{sampling})$$

Ecuación 34

$$IF_{max} = \frac{0.8 \cdot (ADC_{sampling})}{2}$$

5.4.4.b Rango máximo limitado parámetros de ruido

La distancia máxima limitada por parámetros de ruido **depende de varias magnitudes: parámetros de actuación del radar** (potencia de la antena emisora y receptora y *Noise Figure*), **parámetros del chirp**, **parámetros de la antena** (ganancias), **SNR y características del objeto** (RCS). Ignorando las pérdidas de espacio libre, la expresión es la siguiente:

$$Rango_{max,SNR} = \sqrt[4]{\frac{P_t G_{Rx} G_{Tx} c^2 \sigma N_c T_c}{f_c^2 (4\pi)^3 kT \cdot NF \cdot SNR_{min}}}$$

En el numerador, P_t es la potencia de la antena transmisora, G_{Rx} y G_{Tx} son las ganancias de la antena receptora y transmisora respectivamente, c es la velocidad de la luz, σ es el RCS del objeto, N_c es el número de chirps en un frame y T_c es el tiempo total del chirp o el tiempo de muestreo. Todos los parámetros son configurables mediante comandos de configuración o son propiedades de las antenas u objetos excepto T_c , que se calcula con la Ecuación 37, donde N_{ADC} es el número de muestras y $ADC_{sampling}$ es la frecuencia de muestreo.

$$T_c = \frac{N_{ADC}}{ADC_{sampling}}$$

En el caso del denominador, f_c es la frecuencia central (Ecuación 39), k es la constante de Boltzmann ($1.38 \cdot 10^{-23} J/K$), T es la temperatura ambiente (en grados Kelvin), NF es el "noise figure" (disponible en la hoja de especificaciones [48], 9dB) y SNR_{min} es el mínimo SNR necesario para la detección. Antes de calcular la frecuencia central, es necesario calcular el ancho de banda efectivo, que se calcula con la Ecuación 38.

$$B = ST_c$$

$$f_c = f_o + T_{ADC}S + \frac{B}{2}$$

5.4.4.c Mínimo RCS detectable en el máximo rango

Para realizar este cálculo, hay que **despejar el RCS** de la Ecuación 36.

$$\sigma = \frac{Rango_{max}^4 f_c^2 (4\pi)^3 kT \cdot NF \cdot SNR_{min}}{P_t G_{Rx} G_{Tx} c^2 N_c T_c}$$

Este valor solo tiene sentido si el **rango limitado por la frecuencia de los pulsos es mayor que el rango limitado por parámetros de ruido** ya que, de lo contrario, el rango máximo será el rango limitado por parámetros de ruido, y el valor del RCS será el mismo que el utilizado para calcular dicha distancia.

5.4.4.d Resolución de distancia/rango

La resolución de rango del radar es la **habilidad para diferenciar dos objetos cercanos**, cuanto menor sea, mejor podrá diferenciar objetos con menor separación entre ellos.

La ecuación de la resolución de rango es la Ecuación 41 **donde c es la velocidad de la luz y B es el ancho de banda efectivo**.

$$Range_{res} = \frac{c}{2B}$$

Ecuación 41

5.4.4.e Velocidad máxima

Para calcular la velocidad máxima basta con **sustituir en la Ecuación 6 $\Delta\phi$ por su valor máximo, π** . De esta forma, se obtiene que la velocidad máxima es:

$$v_{max} = \frac{\lambda}{4T_r}$$

Ecuación 42

La **longitud de onda, λ** , se calcula con la Ecuación 43.

$$\lambda = \frac{c}{f_c}$$

Ecuación 43

El **chirp repetition time/inter chirp time T_r** , es el tiempo que transcurre entre dos *chirps* de una misma antena (Ecuación 44).

$$T_r = N_{Tx}(T_{idle} + T_{ramp})$$

Ecuación 44

5.4.4.f Resolución de velocidad

De la Ecuación 5 se puede derivar la siguiente ecuación, que se trata de la **resolución de velocidad**.

$$v_{res} = \frac{\lambda}{2N_c T_r} = \frac{1}{N_c} \frac{\lambda}{2T_r}$$

Ecuación 45

Observando la Ecuación 45 y comparándola con la Ecuación 42 (Rango de velocidad) se puede ver que en ambas se encuentra el cociente $\frac{\lambda}{T_r}$ multiplicado por una constante $\frac{1}{4}$ para la velocidad máxima y $\frac{1}{2}$ para la resolución de velocidad. Por este motivo **la ecuación de la resolución de velocidad se puede escribir también en función de la velocidad máxima**.

$$v_{res} = \frac{\lambda}{2N_c T_r} = \frac{2v_{max}}{N_c}$$

Ecuación 46

5.4.4.g Ángulo máximo

Al igual que para el cálculo de la velocidad máxima, para calcular el ángulo máximo, hay que **sustituir $\Delta\phi$ por su valor máximo, π** en la Ecuación 8, obteniendo la Ecuación 47, donde l es la distancia entre antenas receptoras.

$$\theta_{max} = \sin^{-1}\left(\frac{\lambda}{2l}\right)$$

Ecuación 47

Una separación entre antenas de $\lambda/2$ resultaría en un campo de visión de $\pm 90^\circ$. Esto realmente no es así, ya que **la ganancia de la antena receptora disminuye al ir aumentando el ángulo**, hasta llegar a un punto que la señal se atenúa. Esto se puede ver en la Figura 70 donde se puede ver que a unos 70° , la ganancia empieza a decrecer. De hecho, tanto Texas Instruments como Anteral, en sus hojas de especificaciones reducen el **campo de visión a $\pm 60^\circ$** .

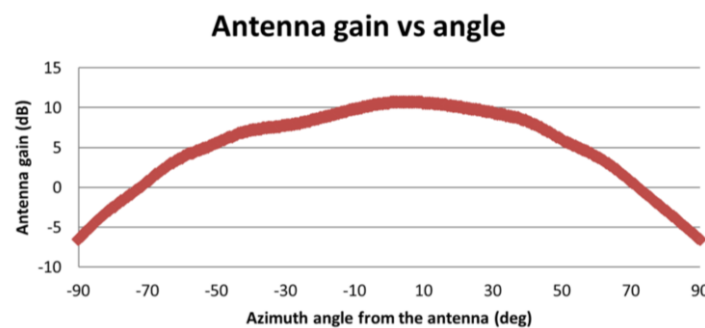


Figura 70. Ganancia de la antena respecto al ángulo (en la horizontal) [47].

5.4.4.h Resolución de ángulo

La **resolución de ángulo** sigue la siguiente expresión donde α es el ángulo de interés, es decir, el ángulo donde el objeto está presente y N_{Rx} el número de antenas receptoras:

$$\theta_{res}(\text{°}) = \frac{\lambda}{l \cdot N_{Rx} \cos \alpha} \cdot \frac{180}{\pi}$$

Ecuación 48

La resolución angular se puede mejorar con **múltiples antenas transmisoras (N_{Tx})**. La ecuación de la resolución quedaría de la siguiente forma:

$$\theta_{res}(\text{°}) = \frac{\lambda}{l \cdot N_{Rx} N_{Tx} \cos \alpha} \cdot \frac{180}{\pi}$$

Ecuación 49

5.4.4.i Otros parámetros importantes

Además de los parámetros calculados anteriormente, existen otros parámetros que, **aunque no son necesarios para los cálculos anteriores, son también muy importantes**.

Uno de los parámetros fácilmente calculables es el **excess time**, que, como se ha explicado anteriormente, es el tiempo que transcurre desde que se acaban de tomar las muestras (final de *ADC Sampling*

Time) hasta que la rampa de frecuencia llega a su punto máximo (principio del idle time del siguiente *chirp*). Observando la Figura 62, se llega a la siguiente ecuación:

$$T_{excess} = T_{ramp} - T_{ADC} - T_c$$

Ecuación 50

Es importante también calcular la **frecuencia de fin**, que servirá para calcular el ancho de banda total del *chirp*. La frecuencia de fin se calcula mediante la Ecuación 51

$$f_{end} = f_o + T_{ADC}S + B$$

Ecuación 51

Conocida la frecuencia final, el **ancho de banda total** se puede calcular con la Ecuación 52.

$$B_{max} = f_{end} - f_o$$

Ecuación 52

El **tamaño en memoria es el tamaño de los datos** después de realizar la primera FFT, que no debe superar un límite para que pueda entrar dentro de la memoria y procesarse adecuadamente. Es un parámetro muy importante, ya que hay que ajustar los parámetros de tal forma que **no se sobrepasen los 768kbits disponibles**.

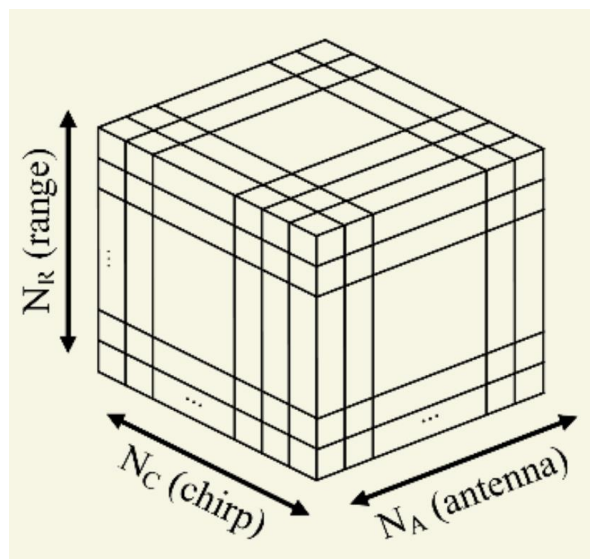


Figura 71. Cubo de datos [31].

Uno de los lados del cubo se refiere al número de **antenas virtuales**. Este valor es el producto entre el número de antenas transmisoras y el número de antenas receptoras.

$$N_A = N_{Tx} \cdot N_{Rx}$$

Ecuación 53

Otro de los lados del cubo es el *zero-padded range-FFT* o *NFFT range*, que se define como la potencia de 2 mayor o igual al número de muestras por *chirp*, N_{ADC} . El tercero de los lados es el número de *chirps* por cada *frame* (N_C). Teniendo además en cuenta que **cada muestra ocupa 4 bytes**, el tamaño del cubo (en kbytes) se calcula de la siguiente forma:

$$Cube_{size} = \frac{4 \cdot N_A \cdot N_C \cdot NFFT}{1024}$$

Ecuación 54

El **ciclo de trabajo** (D) es un parámetro importante. El ciclo de trabajo **depende de la suma de T_{idle} y T_{start}** . Si esta suma es **menor que 10 μ s**, el ciclo de trabajo se calcula de la siguiente forma:

$$D(\%) = \frac{T_c + T_{ADC}}{T_{ramp} + T_{idle}} \cdot 100$$

Ecuación 55

Si $T_{idle} + T_{start}$ es **mayor que 10 μ s**, T_{idle} es excluido de la ecuación. Esto se debe a la **opción de ahorro de energía**, que entra en funcionamiento. Se deshabilitan todas las antenas durante el periodo entre *chirps*, por lo que esa duración debe ser excluida en el cálculo del ciclo de trabajo

$$D(\%) = \frac{T_c + T_{ADC}}{T_{ramp}} \cdot 100$$

Ecuación 56

También se puede calcular el **ciclo de trabajo a lo largo de un *frame***. Este ciclo de trabajo se puede calcular mediante la siguiente expresión:

$$D_{RF}(\%) = \frac{N_{Tx} \cdot N_C \cdot (T_{ramp} + T_{idle})}{Frame\ periodicity} \cdot 100$$

Ecuación 57

5.5 Constant false alarm rate (CFAR)

De todos los mecanismos de procesamiento que emplean los radares, el primero siempre es la **detección**, que busca confirmar la presencia de un blanco o *target*. El problema principal para este proceso es que pueden existir **varios objetos que produzcan reflexiones y que no sean blancos** [49]. En cuanto a la reflexión, se pueden dar blancos y ecos o falsos blancos (una señal interferente que recibe el nombre de *clutter*). Por lo tanto, este mecanismo de procesamiento busca **resolver el problema de la discriminación entre *clutter* y *target***, que se resuelve mediante los **detectores o procesadores CFAR** (*Constant False Alarm Rate*) de **ventana deslizante**.

El principio de **ventana deslizante** significa que el procesamiento sobre las muestras se realiza sobre un **tramo**. Así, **una ventana se desliza por todas las celdas resolutivas disponibles** en el que se ejecuta un algoritmo que decide si la celda en el centro de la ventana es un blanco, mediante la comparación con las magnitudes encontradas en las celdas vecinas.

La propiedad CFAR establece una **relación entre los dos parámetros fundamentales de la detección**. El primer parámetro es la **probabilidad de detección** (P_d) que cuantifica la relación entre los blancos o *target* que se detectan y los que pasan desapercibidos. Por ejemplo, una probabilidad de detección de 0,5 significa que el procesador sólo detectará la mitad de los blancos. El segundo parámetro es la **probabilidad de falsa alarma** (P_f) que mide la proporción entre las muestras de *clutter* que son equívocamente clasificadas como blanco y aquellas que son correctamente descartadas. Por ejemplo, una probabilidad de falsa alarma de 0,1 significa que, como promedio, cada diez muestras de *clutter*, se detecta un falso *target*.

El detector ideal es uno que alcance una $P_d = 1$ y una $P_f = 0$. Generalmente, al modificar el ajuste del detector, P_d y la P_f siguen una relación de proporcionalidad inversa; esto es, al mejorar una la otra empeora.

En radares se utiliza el **criterio de Neyman-Pearson** que establece que el **valor de la P_f toma precedencia**, es decir, que hay que garantizar los requerimientos de diseño de P_f para luego proceder a maximizar la P_d . Los valores más comúnmente encontrados van desde $P_f = 10^{-4}$ hasta $P_f = 10^{-9}$.

5.5.1 CFAR_CA

CFAR_CA es el **detector clásico** que opera bajo el principio de ventana deslizante. Este decide la presencia o no de un blanco o *target* en la **celda bajo evaluación o *Cell Under Test* (CUT)** a partir de la **información de las celdas vecinas o celdas de referencia**. El número de celdas de referencia se selecciona típicamente entre los valores 8, 16, 32 y 64. Si se seleccionan 8 celdas, se colocarán 4 a cada lado de la CUT.

El propósito de las celdas vecinas es la estimación del **valor promedio aritmético del clutter**, para lo cual la medición obtenida en cada una de ellas es promediada. Cuando $N = 64$ se considera que las muestras son suficientes como para hacer una estimación acertada del clutter, ya que el aumento por encima de ese nivel no mejora significativamente el promedio hallado y trae consigo problemas vinculados al procesamiento de situaciones excepcionales como son la ocurrencia de múltiples blancos y la aparición de saltos en el nivel del clutter. Cuando se está en presencia de dichas situaciones, se dice que el **clutter es heterogéneo**. En caso contrario, se clasifica como **homogéneo**.

Otro elemento que suele estar presente en la ventana deslizante son las **celdas de guarda**. Una celda de guarda es **aquella que se omite de la estimación del promedio** para evitar el esparcimiento de la señal del blanco hacia regiones vecinas. Las celdas de guarda son seleccionadas alrededor de la CUT y su número rara vez es superior a dos.

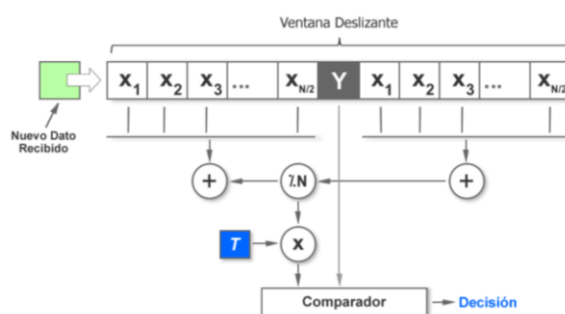


Figura 72. Diagrama de bloques de CFAR_CA [49].

Una vez hallado el promedio de las celdas, se **multiplican por un factor de escala o ajuste T**, resultando en la obtención del umbral de detección. A continuación, un **comparador se encarga de tomar las decisiones** acerca de si es un blanco o no. Si **el valor del umbral está por debajo del valor de la CUT se tomará como blanco o target**, si está por encima se tomará como **clutter**.

El factor T es un elemento decisivo en la detección ya que demasiado alto, el umbral resultante será muy elevado y gran parte de los blancos estarán por debajo de él. Por el contrario, si la T se escoge muy pequeño, aunque se detectarán la mayoría de los objetivos, el detector clasificará erróneamente como blancos a muestras de clutter. Ambas situaciones descritas se pueden ver en la Figura 73, donde son calculados dos promedios de clutter utilizando dos factores T diferentes, uno excesivamente bajo y otro excesivamente alto.

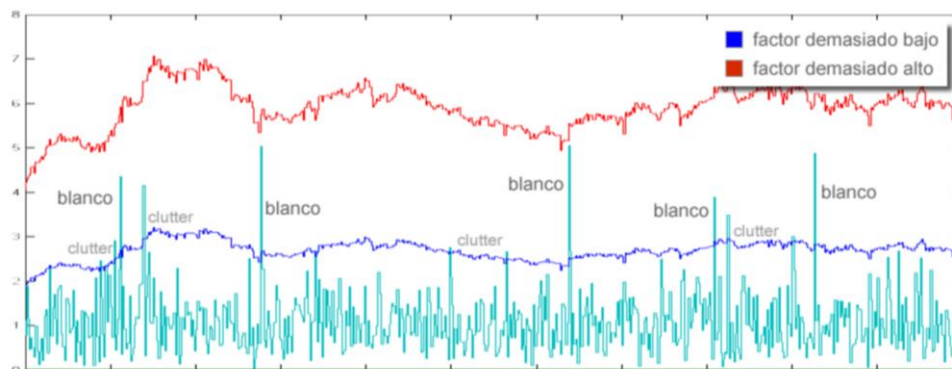


Figura 73. Diferentes umbrales de detección de CFAR_CA [49].

El principal problema del detector CFAR_CA es el **débil comportamiento a clutter heterogéneo**. Si aparecen múltiples blancos en la ventana deslizante o con saltos en el nivel del clutter, se distorsiona el umbral calculado y empiezan a aparecer falsas alarmas u omisiones de blancos a una frecuencia mayor que la esperada.

5.5.2 CFAR_CAGO y CFAR_CASO

Las variantes **CFAR_CAGO** y **CFAR_CASO** proponen utilizar la mitad de las celdas que mayor o menor promedio tengan respectivamente. De esta forma se pueden descartar los máximos locales eligiendo el grupo de celdas anterior o posterior de la CUT. La variante **CFAR_CAGO** beneficia ligeramente a P_f y el **CFAR_CASO** beneficia a P_d .

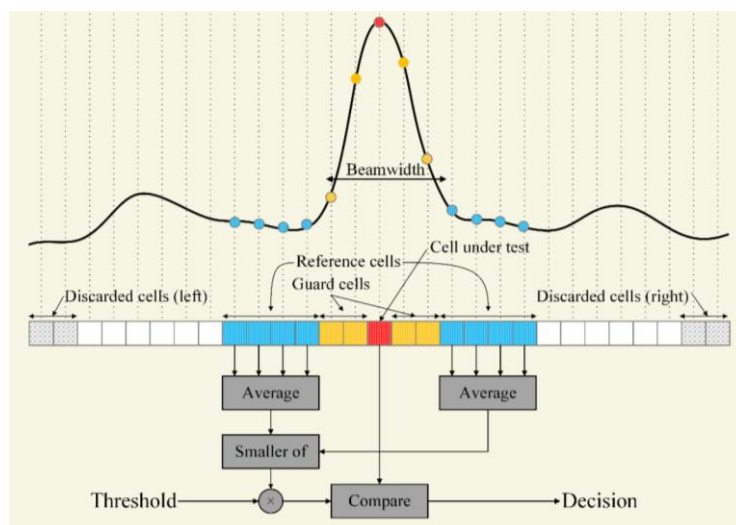


Figura 74. Ejemplo de funcionamiento del CFAR_CASO [31].

5.5.3 Configuración de CFAR en los radares de Texas Instruments (OOB)

En los dispositivos de Texas Instruments todos los **parámetros anteriores se pueden configurar** con el comando de configuración cfarCfg para la demo Out_Of_The_Box. A continuación, se explican los parámetros modificables en este comando de configuración. Para otras demos, aunque igual el algoritmo CFAR se configure con diferentes parámetros, los fundamentos teóricos son los mismos.

cfarCfg		
Número	Parámetro	Descripción
1	< subFrameIdx>	Índice del subframe.
2	<procDirection>	Dirección de procesado, 0 para elegir la detección en rango y 1 para elegir la detección en Doppler.
3	< mode>	Permite elegir entre CFAR_CA (Cell Averaging CFAR) escribiendo un 0, CFAR_CAGO (Cell Averaging Greatest Of CFAR) escribiendo un 1 y CFAR_CASO (Cell Averaging Smallest Of CFAR) con un 2.
4	< noiseWin>	Número de celdas de referencia a un lado de la CUT.
5	< guardLen>	Número de celdas de guarda a un lado de la CUT.
6	< divShift>	La suma de las muestras de clutter se divide por 2 elevado a este parámetro. Hay que tener en cuenta el modo y número de celdas de referencia. <ul style="list-style-type: none"> • <mode>=0: <divShift>=$\log_2(2 \cdot \text{noiseWin})$ • <mode>=1,2: <divShift>=$\log_2(\text{noiseWin})$ Para el caso de la Figura 3, que está en CFAR_CASO, con <noiseWin>=4, <divShift>= $2 \log_2 4 = 2$
7	Ciclic/Wrapped around mode	Elección entre modo cíclico, con un 0 y modo Wrapped around, con un 1.
8	Threshold	Rango (T) en dB, admite floats
9	Peak grouping	Habilitar el clustering con un 1 y deshabilitarlo con un 0.

Tabla 5. Parámetros del comando cfarCfg.

El rango o Threshold para lograr una probabilidad de falsa alarma (P_f) de $6.428 \cdot 10^{-4}$ es 12dB respecto al noise floor [50].

5.6 Formato de los datos de salida: Type-Lenght-Value (TLV)

Cuando un *chirp* vuelve después de rebotar contra un objeto, se mezcla con el *chirp* original para determinar distancia, velocidad y ángulo. El resultado se digitaliza y organiza en una estructura Tipo-Longitud-Valor, en inglés *Type-Length-Value* (TLV) basada en la demo cargada [51]. Esta información se empaqueta en un formato de salida y se envía al ordenador vía UART. Estos paquetes se envían cada *frame* como un paquete con una cabecera. Cada TLV tiene una estructura diferente para organizar la información. Estas estructuras se diferencian con identificadores. La longitud de cada paquete depende de la cantidad de objetos detectados y varía de *frame* a *frame*. Para mantener uniformidad entre los paquetes, cada paquete debe ser un múltiplo de 32 bytes. Si de forma natural no se llega a un múltiplo, el resto del TLV se rellena con ceros.

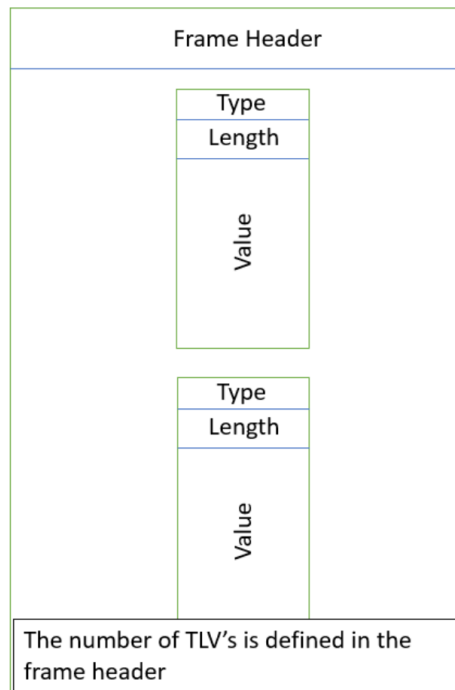


Figura 75. Esquema de un TLV.

- Cabecera del *frame*: 40 bytes

Una cabecera **siempre es enviada al principio de cada paquete**. Se debe usar la denominada *Magic Word* o **palabra mágica para detectar el inicio de cada paquete**. El contenido de la cabecera se puede ver en la Tabla 6.

Cabecera del frame				
Número	Valor/Campo	Tipo	Bytes	Descripción
1	Palabra mágica	Unit16_t	8	Palabra mágica o de sincronismo. Se inicializa a {0x0102, 0x0304, 0x0506, 0x0708}
2	Versión	Unit32_t	4	Versión del SDK representada como $(Num_{mayor} \cdot 2^{24} + Num_{menor} \cdot 2^{16} + Num_{Bugfix} \cdot 2^8 + Num_{Build})$
3	Longitud total del paquete	Unit32_t	4	Longitud total del paquete incluyendo la cabecera del <i>frame</i> en bytes
4	Plataforma	Unit32_t	4	Tipo de dispositivo (para IWR6843, 0xA6843)
5	Numero de <i>frame</i>	Unit32_t	4	Numero de <i>frame</i> (se reinicia cuando el radar es reseteado, no cuando se para)
6	Tiempo [En ciclos de la CPU]	Unit32_t	4	Tiempo en ciclos de CPU cuando el mensaje fue creado
7	Número de objetos detectados	Unit32_t	4	Número de puntos detectados durante el <i>frame</i>
8	Número de TLVs	Unit32_t	4	Número de TLV en el <i>frame</i>
9	Número de <i>subframe</i>	Unit32_t	4	Número de <i>subframe</i>

Tabla 6. Contenido de la cabecera del frame.

- Cabecera del TLV: 8 bytes

El número de TLVs en el frame, se extrae de la cabecera del frame. Cada TLV del paquete dispone de su cabecera de la que se extrae el identificador y su longitud.

Cabecera del TLV				
Número	Valor/Campo	Tipo	Bytes	Descripción
1	Tipo	Unit32_t	4	Tipo/Identificador del TLV
2	Longitud	Unit32_t	4	Longitud del TLV

Tabla 7. Contenido de la cabecera del TLV.

Para la demo de Out_of_the_box, solo se usan los TLV de identificador del 1 al 9, pero en otras demos se llegan a utilizar otros tipos de identificadores. Se puede ver en la Tabla 8 las diferentes demos que soportan cada TLV.

Tipos de identificadores		
Identificador	Tipo	Demos soportadas
1	Detected points	OOB, Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
2	Range Profile	OOB, Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
3	Noise floor profile	OOB (No soportado en xWRL6432), Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
4	Azimuth Static Heatmap	OOB (No soportado en xWRL6432), Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
5	Range-Doppler Heatmap	OOB (No soportado en xWRL6432), Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
6	Statistics (Performance)	OOB, Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
7	Side Info for Detected Points	OOB, Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
8	Azimuth/Elevation Static Heatmap	OOB (No soportado en XWRL6432), Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
9	Temperature Statistics	OOB (No soportado en XWRL6432), Small Obstacle Detection, Traffic Monitoring, Long Range People Detection (LRPD), Area Scanner, Automated Doors and Gates, Parking Garage Sensor, Mobile Tracker.
1000	Spherical Coordinates	LRPD, Traffic Monitoring.
1010	3D Target List	LRPD, (Overhead)3D People Counting, Traffic Monitoring, Vital Signs + People Counting.
1011	Target Index	LRPD, (Overhead)3D People Counting, Traffic Monitoring, Vital Signs + People Counting.
1012	Target Height	LRPD, (Overhead)3D People Counting, Traffic Monitoring, Vital Signs + People Counting.

1020	3D Spherical Compressed Point Cloud	LRPD, (Overhead)3D People Counting, Vital Signs + People Counting.
1021	Presence Detection	LRPD, (Overhead)3D People Counting, Vital Signs + People Counting.
1030	Occupancy State Machine Output	Small Obstacle Detection.

Tabla 8. Identificadores de los diferentes TLV.

Los parámetros en el comando guiMonitor sirven para activar o desactivar que tipos de TLV (de Out_of_the_box) son incluidos en el paquete de salida. A continuación, se describirán brevemente el **contenido de cada uno de los identificadores**, que será importante para el posterior desarrollo (Tabla 9). Algunos datos que devuelven los TLV se pueden ver en el **Anexo**.

Estructura de los diferentes TLV			
Identificador	Tipo	Longitud	Valores
1	Detected points	(Número de objetos detectados) x (16 Bytes)	Array de los puntos detectados. Cada punto es representado por 16 bytes, devolviendo la posición en x, y, z y la velocidad Doppler. Consultar Anexo para más información.
2	Range Profile	(Tamaño de Range FFT) x (2 Bytes) para XWR6843, XWR1843, XWR1642 y XWR1443, (Tamaño de Range FFT) x (4 Bytes) para XWRL6432	Array de los puntos en el 0 Doppler (objetos estáticos).
3	Noise floor profile	(Tamaño de Range FFT) x (2 Bytes)	Array de los puntos en el máximo Doppler (en general para escenas estáticas no debería haber objetos en la máxima velocidad o máximo Doppler).
4	Azimuth Static Heatmap	(Tamaño de Range FFT) x (Número de antenas virtuales "acimut") x (2 Bytes).	Muestras para calcular el mapa de calor estático (objetos estáticos). Los valores se usan para construir el mapa de calor en la herramienta <i>mmWave Demo Visualizer</i> .
5	Range-Doppler Heatmap	(Tamaño de Range FFT) x (Tamaño de Doppler FFT) x (2 Bytes)	Matriz de detección Rango-Doppler
6	Statistics (Performance)	24 bytes	Valores estadísticos. Consultar Anexo para más información.
7	Side Info for Detected Points	4 bytes x (Número de objetos detectados)	Valores del SNR y ruido con una resolución de 0.1 dB. Consultar Anexo para más información.
8	Azimuth/Elevation Static Heatmap	(Tamaño de Range FFT) x (Número de antenas virtuales) x (4 Bytes)	Muestras para calcular el mapa de calor estático de acimut o elevación (objetos estáticos).
9	Temperature Statistics	28 Bytes	Valores de temperatura tomados justo antes del envío de los TLV. Consultar Anexo para más información.
1000	Spherical Coordinates	(Número de puntos) x (16 Bytes)	Información sobre los puntos detectados en coordenadas esféricas. Consultar Anexo para más información.
1010	3D Target List	(Número de objetivos/targets) x (112 Bytes)	Serie de valores que describen el movimiento en 3 dimensiones del objeto, así como errores en las medidas. Consultar Anexo para más información.
1011	Target Index	(Número de puntos) x (1 Byte)	Contiene el ID del <i>target</i> . Consultar Anexo para más información.
1012	Target Height	(Número de puntos) x (12 Bytes)	Contiene el ID y la máxima y mínima altura del objeto. Consultar Anexo para más información.

1020	3D Spherical Compressed Point Cloud	(20 Bytes) + (8 Bytes x Número de puntos)	Define la nube de puntos. Se divide en 2 estructuras, la primera define la nube de puntos y la segunda define los puntos dentro de la nube de puntos. Consultar Anexo para más información.
1021	Presence Detection	4 Bytes	Indica si se ha detectado presencia de algún objeto en el rango del radar.
1030	Occupancy State Machine Output	4 Bytes	Máscara de bits indicando en que zona del radar se encuentra un objeto pequeño. Por ejemplo, si el bit 1 está a 1, la zona 1 está ocupada, pero si está a 0, la zona no está ocupada.

Tabla 9. Descripción de los diferentes TLV.

6 Desarrollo práctico

A continuación, se describirá el **desarrollo práctico**, incluyendo **todos los pasos, comprobaciones realizadas y problemas encontrados** hasta llegar a la solución final.

6.1 Descripción e instalación de hardware y software

En este apartado se describirá detalladamente tanto el **hardware** como el **software** empleado junto con el uRad Industrial v1.0 y el IWR6843AoP, descritos anteriormente en el apartado 4.4.

6.1.1 Nvidia Jetson AGX Orin Developer Kit

Para la **programación y la visualización** de los datos del radar y de otros dispositivos, se utilizará una placa **Nvidia Jetson AGX Orin Developer Kit**. El kit de desarrollo **NVIDIA Jetson AGX Orin** permite **crear robótica avanzada y aplicaciones de IA** para fabricación, logística, venta al por menor, servicios, agricultura, ciudades inteligentes, sanidad y biociencias, entre otras muchas [52].

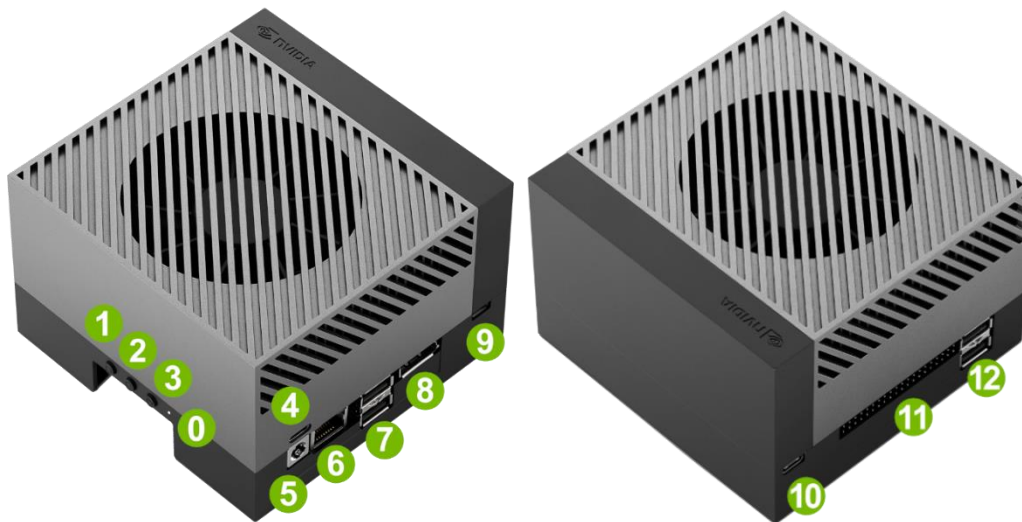


Figura 76. Vistas laterales de Nvidia AGX Orin Developer Kit [53].

Es recomendable que **los periféricos, como el radar, se conecten a los puertos USB número 7 de la Figura 76** ya que estos son puertos USB 3.2 de segunda generación y los otros dos puertos USB (número 12) son USB 3.2 de primera generación.

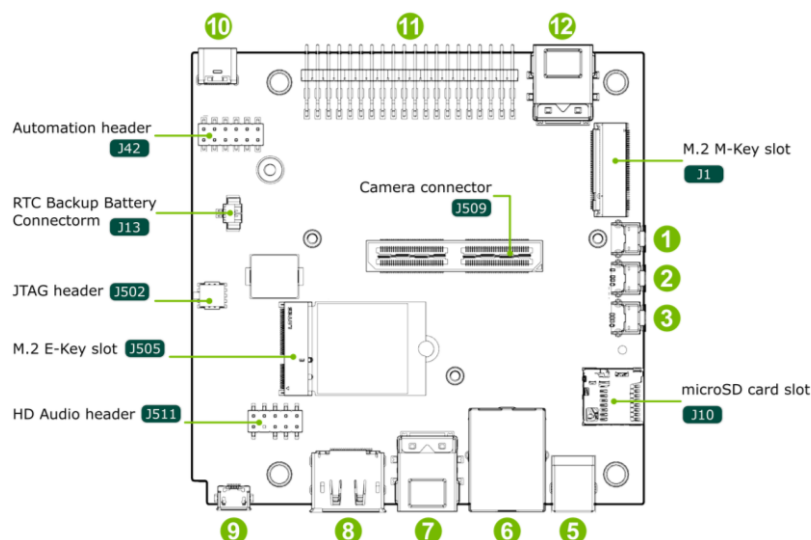


Figura 77. Vista inferior de Nvidia AGX Orin Developer Kit [53].

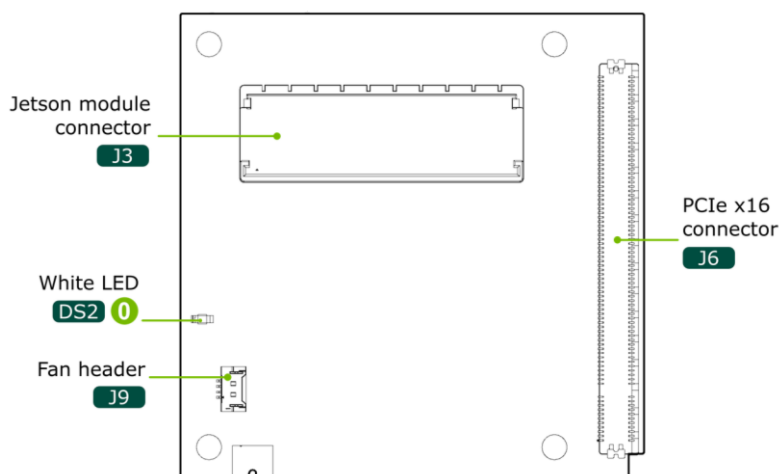


Figura 78. Vista superior de Nvidia AGX Orin Developer Kit (oculta bajo el módulo) [53].

Se puede hacer que la placa **se encienda al recibir corriente** sin necesidad de encenderla manualmente. Esto es útil para no tener que estar encendiendo y apagando la placa cada vez que se quiera iniciar el equipo en aplicaciones donde no se pueda acceder a la placa fácilmente o donde esta sea totalmente inaccesible. Esto se puede hacer mediante los pines 5 y 6 del *Automation header* (J42), Figura 77.

Sin embargo, hay un **problema con los puertos serie** de la placa, que afecta al radar. Hasta ahora, se desconoce si es un problema del hardware o del software, pero al hablar con los desarrolladores la placa [54], se ha llegado a la conclusión que posiblemente se trate de un **problema con el software**.

El radar funciona correctamente la primera vez que se ejecuta el programa. Cuando se quiere ejecutar otro programa o simplemente enviar al radar otro archivo de configuración, **desaparece el puerto serie** y ya **no se puede utilizar ningún tipo de periférico** (el puerto desaparece). Es un problema muy molesto a la hora de desarrollar y probar los programas, por lo que se ha optado por usar un modelo anterior un poco menos potente, *Jetson AGX Xavier Developer Kit*.

6.1.2 Nvidia Jetson AGX Xavier Developer Kit

Como se ha explicado en el apartado anterior, los puertos de la *Nvidia Jetson AGX Orin Developer Kit* no funcionan correctamente, por eso **se usará Nvidia Jetson AGX Xavier Developer Kit**, que es la predecesora [55]. A pesar de que la nueva placa sea **menos potente** que la *Jetson AGX Orin*, tiene potencia suficiente para procesar los datos recibidos correctamente.



Figura 79 Nvidia Jetson AGX Xavier Developer Kit [55].

6.1.3 USB Hub

Este **hub USB** se usará ya que la *Nvidia Jetson AGX Xavier Developer Kit* **sólo dispone de un único puerto USB**. Varios elementos que se utilizarán demandan un **pico de corriente** y no se podrían conectar a un *hub* normal, por lo que este tiene **alimentación externa** para soportar los picos de corriente. Además, cuenta con **botones para activar y desactivar los puertos**. Esto es muy útil ya que en vez de desconectar el USB cada vez que no se quiera usar un puerto, se le da al botón y su efecto es como si el USB estuviera desconectado.



Figura 80. Hub USB con alimentación externa.

6.1.4 Kazam

Kazam es una aplicación para la grabación del escritorio en Ubuntu. **Es capaz tanto de grabar videos como de tomar capturas de pantalla** de la pantalla completa, una aplicación o un área que determine el usuario.

6.1.5 Visual Studio Code

Visual Studio Code es un **editor de código fuente** desarrollado por Microsoft para Windows, Linux, macOS y Web [56]. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código. También es **personalizable**, por lo que los usuarios pueden cambiar el tema del editor, los atajos de teclado y las preferencias.

6.1.6 Robot operating system (ROS)

Robot Operating System (ROS) es una estructura **distribuida de procesos llamados Nodos** que permite el diseño individualizado de procesos, pero fácilmente acoplable a otros [57]. Estos procesos se pueden **agrupar en Paquetes**, que **fácilmente pueden ser intercambiados, compartidos y distribuidos**. **Los nodos generan topics**, que pueden realizar una **transferencia de información**. Cualquier nodo puede publicar o suscribirse un *topic* para enviar o recibir flujos de datos (imágenes, estéreo, láser, control, actuador, contacto, etc.)

La funcionalidad del núcleo ROS se expande con una variedad de herramientas que permiten a los desarrolladores: **visualizar y recopilar información, navegar de manera sencilla** en la estructura de paquetes y **crear código para automatizar tareas** complejas y otros procesos de configuración.

ROS actualmente **solo se ejecuta en plataformas basadas en Unix y en W10 mediante virtualización**. El software para ROS **se prueba principalmente en sistemas Ubuntu y Mac OS X**, aunque la comunidad ROS ha estado contribuyendo con soporte para otras plataformas Linux.

ROS usa mensajes simplificados para **describir valores de los datos** que los nodos pueden publicar [58]. Esto facilita a las herramientas de ROS la **generación de código fuente** para esos tipos de mensajes. Las descripciones de los mensajes se almacenan en archivos .msg en la carpeta msg/ dentro del subdirectorio del paquete de ROS.

Hay dos partes en un mensaje, **campos de datos y constantes**. Los campos de datos son donde se encuentran los datos del mensaje (por ejemplo, coordenadas de un punto) y las constantes son valores útiles que pueden ser usados para interpretar los campos.

ROS también dispone de **servicios** que se usan para **acciones de petición y respuesta**, realizadas **mediante mensajes**, uno para la petición y otro para la respuesta. Los nodos de ROS ofrecen servicios que el cliente puede solicitar enviando el mensaje de petición y esperando para recibiendo la respuesta. Se definen usando archivos .srv, que se compilan a código fuente.

La placa **Jetson AGX Orin Developer Kit** dispone de **Ubuntu 20.04 LTS**, por lo que es necesario descargar **ROS Noetic** y la a placa **Jetson AGX Xavier Developer Kit** dispone de **Ubuntu 18.04 LTS**, por lo que es necesario instalar **ROS Melodic**. Para llevar a cabo la correcta instalación, se pueden seguir los pasos de la web oficial de ROS [59] o seguir los pasos de la página de *Stereolabs* [60] (en el caso de la *Jetson AGX Orin*). Para la placa *Jetson AGX Orin*, es recomendable realizar la instalación a partir de la página de *Stereolabs*, ya que posteriormente se utilizará un dispositivo de ese fabricante. En esa página web se explica tanto la

instalación de ROS *Noetic* como la instalación de todos los elementos necesarios para que funcione el dispositivo del fabricante.

6.1.7 Rviz

Rviz es una **herramienta de simulación y visualización 3D** para robots, donde se puede visualizar el ambiente en el que estos se desempeñan y la información de sensores que estos generan [61]. Es **altamente configurable** y posee distintos tipos de *plugins* y formatos de visualización.

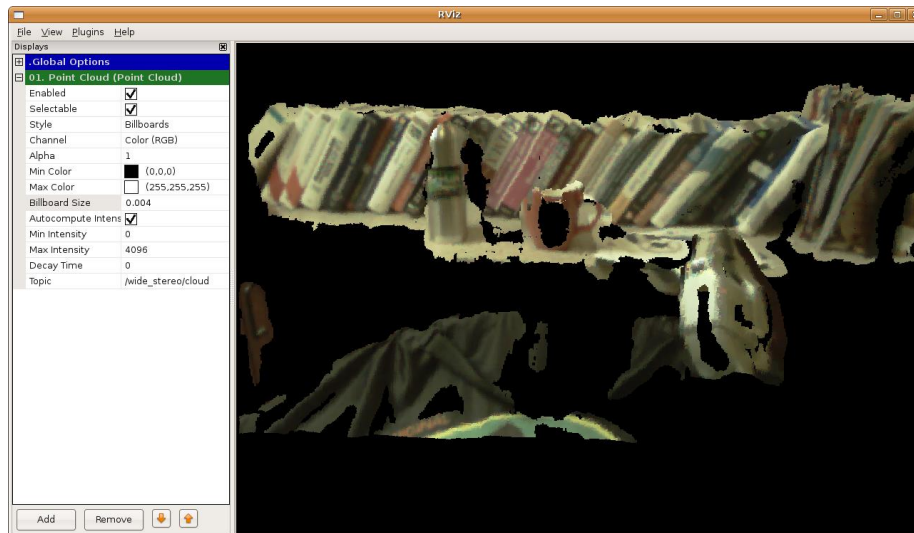


Figura 81. Ejemplo de Rviz [62].

6.1.8 Cámara estereoscópica ZED 2i

Una **cámara estereoscópica** o cámara 3D, es nombrada así debido a la visión estereoscópica humana (visión en 3D) [63]. Es una cámara **capaz de capturar imágenes** (fotografías) **en tres dimensiones**. Estas cámaras intentan imitar este comportamiento de visión humana, utilizando dos objetivos separados estratégicamente para captar la fotografía en el mismo instante y obtener las imágenes 3D. La cámara que se va a utilizar es la **cámara ZED 2i, del fabricante Stereolabs**. Esta cámara no solo muestra las imágenes en 3D, **lleva incorporada una red neuronal** que le permite realizar muchas más funcionalidades.



Figura 82. Cámara ZED 2i.

De todas las funcionalidades la cámara ZED 2i, solo se utilizarán dos de ellas: **La percepción de profundidad y la detección de objetos**. La percepción de profundidad es la **habilidad para determinar las distancias entre objetos y verlas en 3 dimensiones**. La cámara ZED 2i proporciona una nube de puntos en 3 dimensiones mostrando la profundidad de los objetos. También puede mostrar los alrededores e incluso contener información sobre el color.

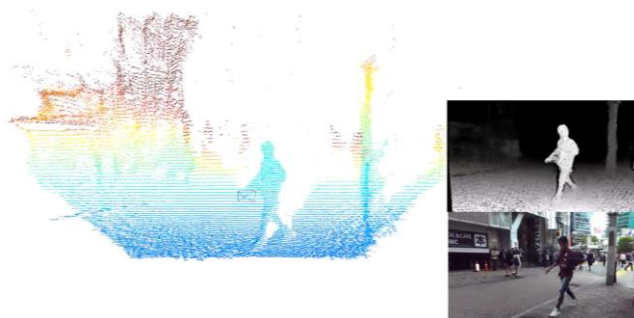


Figura 83. Ejemplo de la nube de puntos generada por la cámara ZED 2i [63].

En cuanto a la detección de objetos, la cámara ZED 2i puede **reconocer diferentes objetos gracias a la red neuronal implementada**. Es capaz de diferenciar diferentes objetos y, cuando detecta personas, es capaz de mostrar en pantalla su “**esqueleto**”.



Figura 84. Ejemplo de la detección de objetos y personas de la cámara ZED 2i [63].

6.1.9 Configuración de la cámara ZED 2i y mmwave con ROS y Rviz

Como se ha comentado en el apartado 5.2, la herramienta *mmwave Demo Visualizer* es una herramienta ideal para trabajar con el radar por primera vez con *Out_of_the_box* y entender su funcionamiento, pero la herramienta **no ofrece ninguna posibilidad de trabajar con los datos recibidos** ni de visualizar los datos proporcionados por otro dispositivo (como pueden ser los datos de la cámara ZED 2i) u otra demo. Las herramientas que permite el procesado y la visualización de los datos que suministran los sensores son **ROS y Rviz**.

6.1.9.a ZED 2i con ROS

Para poder visualizar los datos que proporciona la cámara en *Rviz*, hay que **instalar el driver de ROS**, que incluye los nodos necesarios, y algunos complementos, pero antes de eso, hay que instalar el **ZED SDK** (de la página web de *Stereolabs*) para que la cámara pueda funcionar correctamente. La guía para la instalación tanto del SDK como del *driver* de ROS y otros complementos se pueden ver desde [60] o desde su propia página web ([64] para el SDK y [65] para el *driver* y complementos). Para el caso de la **Nvidia Jetson AGX Orin Developer Kit** se instalará la versión **ZED SDK 3.8.2 para Ubuntu 20.04 LTS** y para la **Nvidia Jetson AGX Xavier Developer Kit** se instalará la versión **ZED SDK 3.8.2 para Ubuntu 18.04 LTS**. Una vez instalado el SDK, hay que instalar el *driver* de ROS, que incluye el *zed_ros_wrapper*, que es el **nodo de ROS**. Es también conveniente instalar **algunos ejemplos** (*zed_ros_examples*), para así poder entender cómo funciona la cámara. **El complemento *zed_ros_interfaces* no es necesario instalarlo ya que viene incluido en el nodo de ROS**.

Cuando ya está todo instalado se conecta la cámara a la placa de desarrollo y se pone en funcionamiento el nodo de ROS mediante un comando disponible en la guía [65]. Al iniciarse con ROS y Rviz, el resultado es el de la Figura 85.

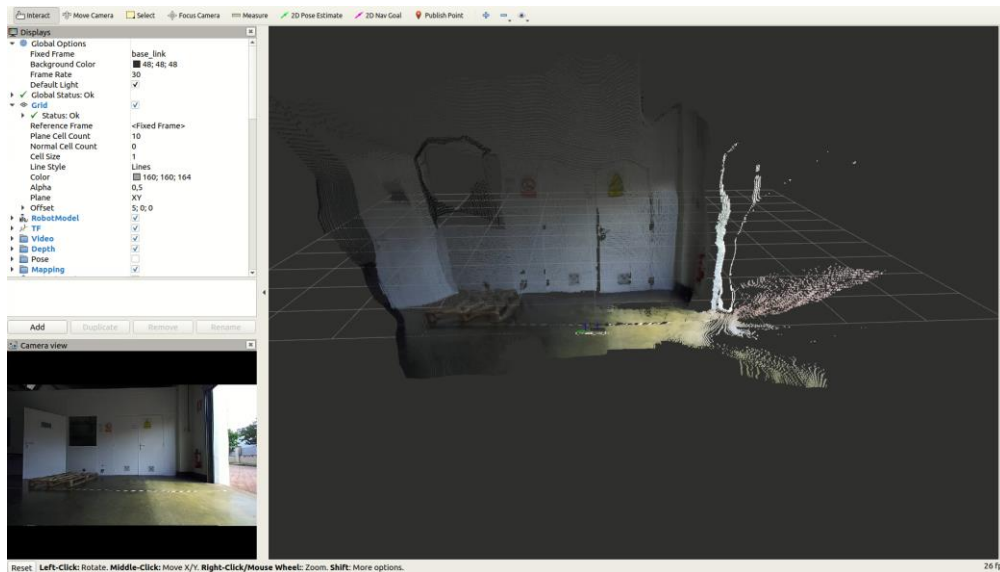


Figura 85. Visualización de la nube de puntos proporcionada por la cámara ZED 2i en Rviz.

Una de las funcionalidades que proporciona la cámara ZED 2i es la **detección de objetos**. Para ello se puede seguir la **guía de servicios de ROS** [66] para activar esa funcionalidad. Se pueden detectar múltiples objetos, personas o animales, y el resultado es el que se muestra en la Figura 86 y Figura 87.

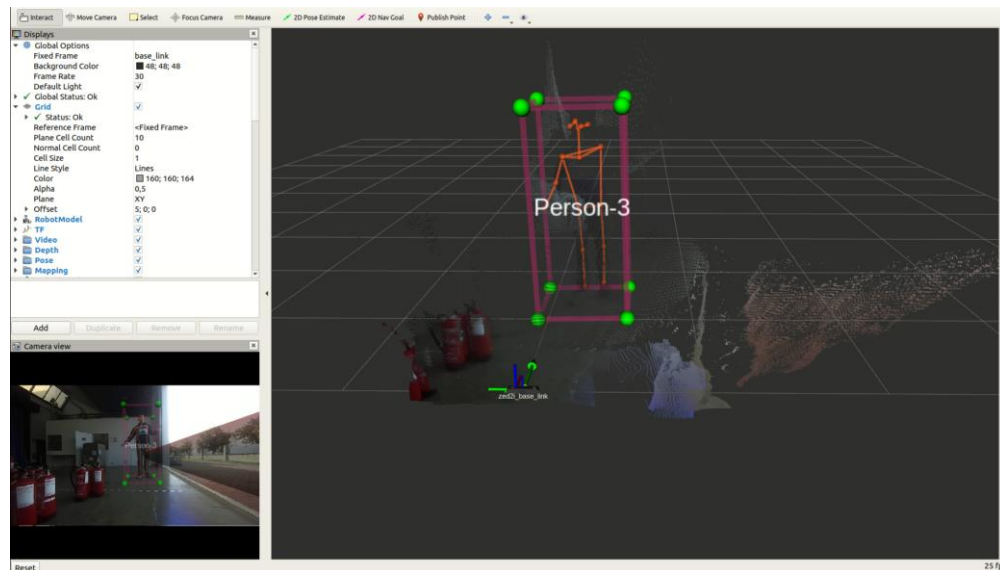


Figura 86. Detección de personas de la cámara ZED 2i.

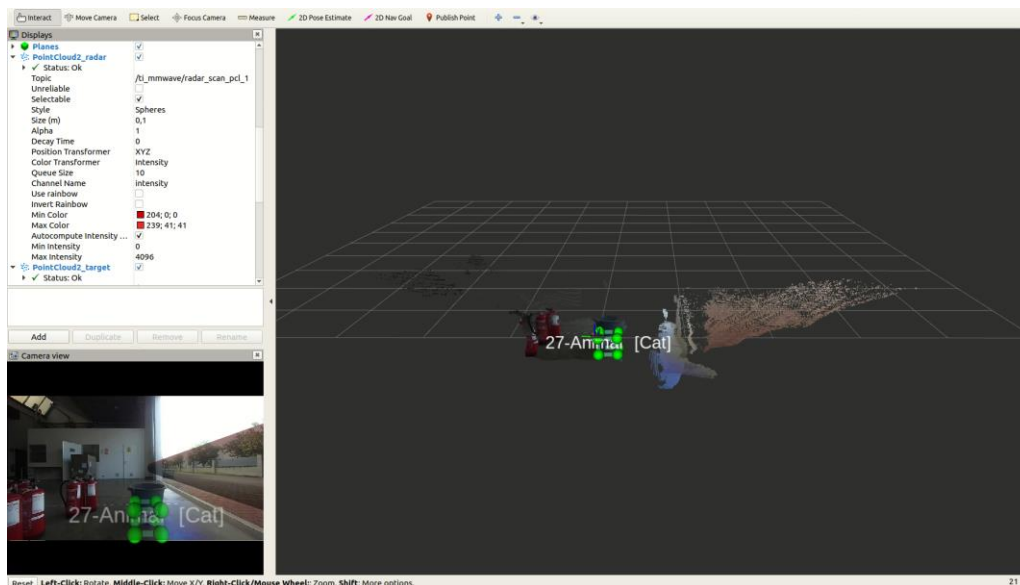


Figura 87. Detección multiclase de la cámara ZED 2i, en este caso detectando un animal.

Se pueden configurar diversos parámetros para ejecutar el servicio de ROS, como se puede ver en la siguiente tabla.

Configuración del servicio "Object Detection"			
Número	Parámetro	Descripción	Valor
1	od_enable	Habilitar o deshabilitar el módulo de detección de objetos .	true/false
2	model	Módulo de detección a usar	"0": MULTI_CLASS_BOX "1": MULTI_CLASS_BOX_ACCURATE "2": HUMAN_BODY_FAST "3": HUMAN_BODY_ACCURATE "4": MULTI_CLASS_BOX_MEDIUM "5": HUMAN_BODY_MEDIUM
3	confidence_threshold	Valor mínimo de la confianza de detección de un objeto.	Número entero [0,100]
4	max_range	Máximo rango de detección.	Por defecto, 15 m
5	object_tracking_enabled	Habilitar o deshabilitar el tracking de objetos .	true/false
6	body_fitting	Habilitar o deshabilitar el ajuste al cuerpo para el modelo "HUMAN_BODY".	true/false
7	mc_people	Habilitar o deshabilitar la detección de personas para el modelo "MULTI_CLASS_BOX".	true/false
8	mc_vehicle	Habilitar o deshabilitar la detección de vehículos para el modelo "MULTI_CLASS_BOX".	true/false
9	mc_bag	Habilitar o deshabilitar la detección de bolsos/mochilas para el modelo "MULTI_CLASS_BOX".	true/false
10	mc_animal	Habilitar o deshabilitar la detección de animales para el modelo "MULTI_CLASS_BOX".	true/false
11	mc_electronics	Habilitar o deshabilitar la detección de aparatos electrónicos para el modelo "MULTI_CLASS_BOX".	true/false
12	mc_fruit_vegetable	Habilitar/deshabilitar la detección de frutas y vegetales para el modelo "MULTI_CLASS_BOX".	true/false

Tabla 10. Parámetros del servicio de ROS Object detection.

Con todo esto ya se tendría lo básico para trabajar con la cámara ZED 2i.

6.1.9.b Radar con ROS

Hay que instalar el **driver de ROS que proporciona Texas Instruments**, para ello hay que seguir la guía de usuario [67]. Primero hay que compilar (*catkin_make*) como especifica la guía de usuario y luego compilar en el *catkin_ws* (*catkin workspace*). Una vez instalado el paquete de ROS, **hay que hacer varias modificaciones**. En primer lugar, hay que **eliminar la carpeta *autonomous_robotics*** (*catkin_ws/src/mmwave_ti_ros*), ya que produce duplicidades y no se puede compilar correctamente.

Para el caso de la **Jetson AGX Orin Developer Kit** hay que **descargar otra librería “serial”**, ya que el *driver* de ROS está preparado para Ubuntu 18 y ROS *Melodic* en vez de Ubuntu 20 ROS *Noetic*. Se debe **eliminar la carpeta “serial” existente (*catkin_ws/src/mmwave_ti_ros/ros_driver/src*)** y luego **instalar en ese directorio la nueva librería *serial*** mediante el comando **“*sudo apt install ros-noetic-serial*”**.

Para poner en funcionamiento el radar, **se hará ejecutando los archivos .launch**. En todos estos archivos **hay un error en el código que impiden su correcto funcionamiento** para el radar **IWR6843AoP**.

```

src> ti_mmwave_ropkg > launch > 6843AOP_multi_3d_0.launch
1 <!--
2 This file will launch rviz along with the mmwave sensor node and configure a TI mmwave 6843 sensor using a 3D config
3 -->
4
5 <launch>
6
7 <!-- Input arguments -->
8 <arg name="device" value="6843" doc="TI mmwave sensor device type [1443, 1642, 6843]"/>
9 <arg name="config" value="3d" doc="TI mmwave sensor device configuration [3d best range ros (not supported by 1642 EVM), 2d best range ros]"/>
10 <arg name="max_allowed_elevation_angle_deg" default="90" doc="Maximum allowed elevation angle in degrees for detected object data [0 >= value <= 90]"/>
11 <arg name="max_allowed_azimuth_angle_deg" default="90" doc="Maximum allowed azimuth angle in degrees for detected object data [0 >= value <= 90]"/>
12
13 <!-- mmWave Manager node -->
14 <node pkg="ti_mmwave_ropkg" type="ti_mmwave" name="ti_mmwave" ns="radar_0" output="screen">
15 <param name="command_port" value="/dev/ttyAMC0" />
16 <param name="command_rate" value="115200" />
17 <param name="data_port" value="/dev/ttyAMC1" />
18 <param name="data_rate" value="115200" />
19 <param name="max_allowed_elevation_angle_deg" value="$(arg max_allowed_elevation_angle_deg)" />
20 <param name="max_allowed_azimuth_angle_deg" value="$(arg max_allowed_azimuth_angle_deg)" />
21 <param name="frame_id" value="/ti_mmwave_0"/>
22 <param name="mmwaveCLI_name" value="/mmwaveCLI" />
23 <remap from="ti_mmwave/radar_scan_pcl" to="/ti_mmwave/radar_scan_pcl_0"/>
24 </node>
25
26 <!-- mmWaveQuickConfig node (terminates after configuring mmWave sensor) -->
27 <node pkg="ti_mmwave_ropkg" type="mmwaveQuickConfig" name="ti_mmwave_config" ns="radar_0" args="$(find ti_mmwave_ropkg)/cfg/6843AOP_3d.cfg" output="screen">
28 <param name="mmwaveCLI_name" value="/mmwaveCLI" />
29 </node>
30
31 <node pkg="tf" type="static_transform_publisher" name="radar_baseLink_0" args="0 0 0 0 0 ti_mmwave_pcl ti_mmwave_0 100"/>
32
33 <node pkg="rviz" type="rviz" name="rviz" args="-d $(find ti_mmwave_ropkg)/launch/rviz/ti_mmwave_multi.rviz"/>
34 </launch>
35

```

Figura 88. Archivo 6843AOP_multi_3d_0.launch por defecto.

Para el IWR6843AOP se tienen **4 archivos *launch* diferentes**, *6843AOP_multi_2d_0.launch*, *6843AOP_multi_2d_1.launch*, *6843AOP_multi_3d_0.launch* y *6843AOP_multi_3d_1.launch*. En todos ellos hay que **modificar la línea 21**, donde se especifica el nombre del *frame*, **eliminando la “/”** del nombre. Donde pone **“value=“/ti_mmwave_X”**”, modificarlo de tal forma que quede **“value=“ti_mmwave_X”**”, donde X representa el número del archivo, 0 o 1. Hay **dos archivos con diferente número para poder ejecutar 2 radares al mismo tiempo**, por eso cambia el nombre del *frame*. La única diferencia es que el archivo 0 lanza un nodo de *Rviz* y el archivo 1 no lo hace.

Además, hay que cambiar el **nombre de los puertos** a los que se conecta el radar, ya que los puertos que vienen por defecto en el archivo *.launch* pueden no ser los correctos. Para el caso del IWR6843AoP, estos puertos se encuentran en las **líneas 15 y 17**, y por defecto vienen los puertos */dev/ttyAMC0* y */dev/ttyAMC1* para los archivos con subíndice 0 y */dev/ttyAMC2* y */dev/ttyAMC3* para los que tienen de subíndice 1. Los puertos a los que se conecta el radar se pueden ver con el comando *ls/dev*. En este caso, el radar se conecta a los puertos */dev/ttyUSB0* y */dev/ttyUSB1*. Justo debajo de las líneas para seleccionar los puertos se pueden **configurar la velocidad de transmisión de datos** (en baudios). La velocidad que aparece por defecto puede que no sea la velocidad recomendada, por lo que se puede modificar. En este caso, se ha dejado la velocidad

del `command_port` por defecto y se ha modificado la velocidad del puerto de datos (`data_port`) a 921600 baudios (valores recomendados por Texas Instruments). En la Figura 89, se puede ver el nuevo archivo `.launch` una vez realizados todos los cambios anteriores.

```

13 <!-- mmWave Manager node -->
14 <node pkg="ti_mmwave_ropkg" type="ti_mmwave_ropkg" name="ti_mmwave" ns="radar_0" output="screen">
15 <param name="command_port" value="/dev/ttyUSB0" />
16 <param name="command_rate" value="115200" />
17 <param name="data_port" value="/dev/ttyUSB1" />
18 <param name="data_rate" value="921600" />
19 <param name="max_allowed_elevation_angle_deg" value="$(arg max_allowed_elevation_angle_deg)" />
20 <param name="max_allowed_azimuth_angle_deg" value="$(arg max_allowed_azimuth_angle_deg)" />
21 <param name="frame_id" value="ti_mmwave_0"/>
22 <param name="mmWaveCLI_name" value="/mmWaveCLI" />
23 <remap from="/ti_mmwave/radar_scan_pcl" to="/ti_mmwave/radar_scan_pcl_0"/>
24 </node>

```

Figura 89. Archivo `6843AOP_multi_3d_0.launch` con las correcciones.

Para el caso del **IWR6843ISK**, los errores y cosas que se deben cambiar son exactamente las mismas y en las mismas líneas de código que para el IWR6843AoP.

Con todo esto, **ya se puede lanzar el nodo de ROS** y ver los datos en *Rviz*. En la Figura 90 se puede ver la **nube de puntos que proporciona el radar mientras detecta a una persona y un objeto**.

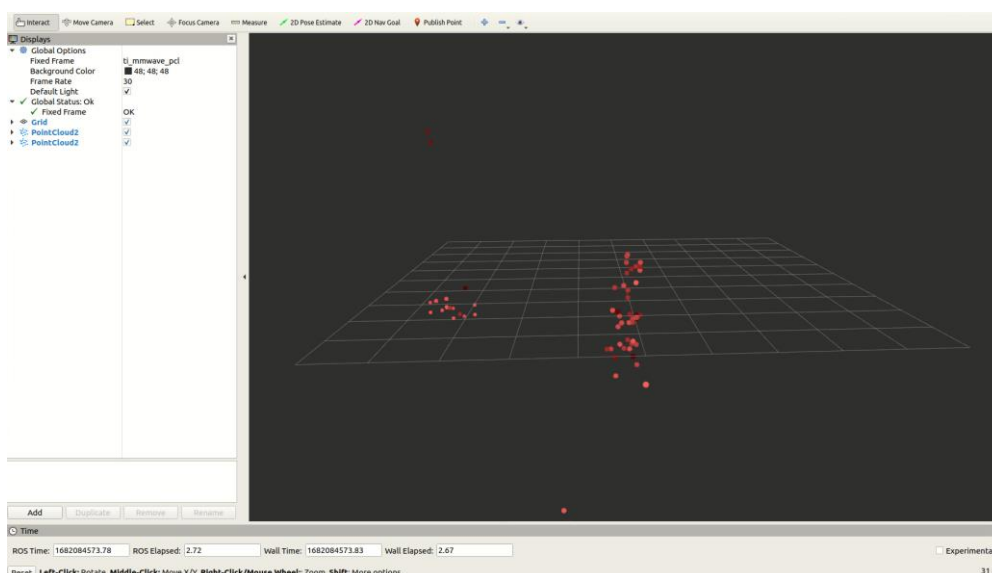


Figura 90. Nube de puntos del radar (OoB) con *Rviz* mientras detecta a una persona y un objeto a la izquierda.

6.1.9.c ZED 2i y Radar juntos

Hasta ahora, solo se han lanzado ambos nodos de ROS por separado, donde cada uno de ellos lanzaba un nodo de *Rviz* por separado. Si se lanzaran los dos nodos a la vez **sin hacer cambios en *Rviz*, no se podrían ver al mismo tiempo radar y cámara**, ya que el segundo nodo cerraría *Rviz* y lo volvería a abrir. Esto se debe a que en el archivo `.launch` hay una **línea de código que abre *Rviz*** con una configuración determinada (por ejemplo, para el radar IWR6843AoP, es línea 33 de la Figura 95). Para ello hay que **ejecutar los dos nodos y configurar *Rviz* internamente** añadiendo o eliminando `topics` para así poder ver ambas nubes de puntos a la vez, ya que cada nodo arranca *Rviz* configurado para mostrar únicamente sus `topics`. Para la configuración, se lanzará en primer lugar el nodo del radar y posteriormente el de la cámara. Como se ha comentado anteriormente, en *Rviz* sólo se podrá **ver la configuración del nodo de la cámara, y habría que añadirle únicamente la nube de puntos del radar**. Esto se hace así porque el nodo de la cámara muestra muchos más `topics`, y el radar solo muestra uno, luego es más sencillo añadir el `topic` del radar a la cámara.

Si se hace de esta forma, se puede observar que **todavía no se pueden ver ambas nubes de puntos**. En el apartado de *Global Options* se pueden ir cambiando el **fixed frame** para poder ver una u otra nube de puntos, pero nunca se pueden ver las dos a la vez, ya que, si una funciona, la otra dará error, como se puede ver en la Figura 91 y Figura 92.

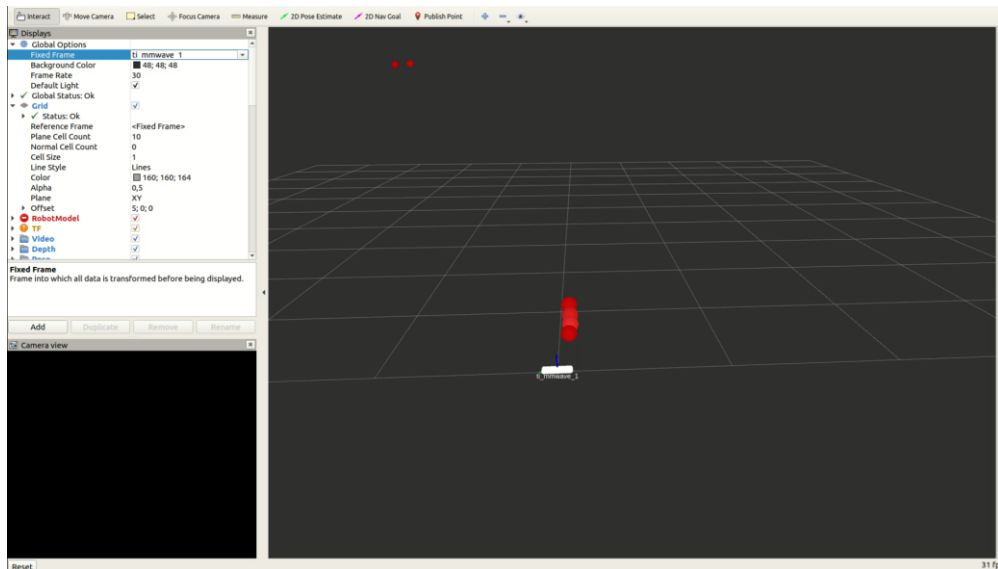


Figura 91. Funciona el radar, pero falla la cámara ZED.

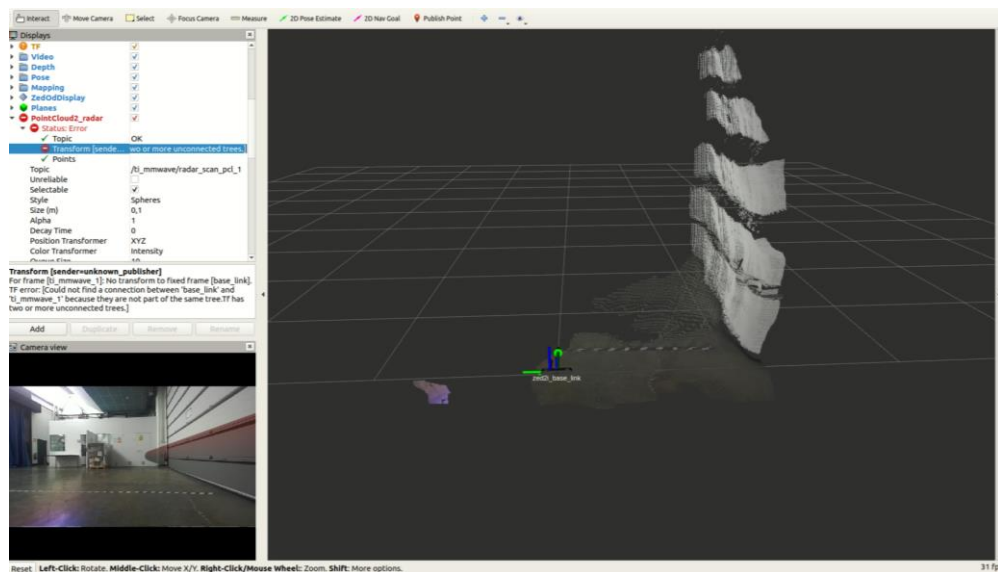


Figura 92. Funciona la cámara ZED, pero falla el radar.

Esto se debe a que los nodos de la cámara y el radar **no publican los topics en el mismo fixed frame**, que es el sistema de coordenadas. Para corregir este error, basta simplemente con pasar o **convertir un frame donde publique el radar a otro donde publique la cámara**, de tal forma que **ambos topics publiquen en el mismo fixed frame**.

A la hora de hacer la transformación de los *Fixed frames* en el archivo .launch, se utiliza el **comando static_transform_publisher** con el nombre `base_link_X` y ciertos argumentos [68]. Esos argumentos representan la **diferencia en la posición de los diferentes sensores**. Es decir, da igual que los sensores estén separados, todos los valores que proporcionan se pueden **referenciar a un único punto**.

Argumentos de static_transform_publisher		
Número	Argumento	Descripción
1	x	Offset en el eje X
2	y	Offset en el eje Y
3	z	Offset en el eje Z
4	yaw	Rotación sobre el eje Z
5	pitch	Rotación sobre el eje Y
6	roll	Rotación sobre el eje X
7	Parent_frame_id	Frame al que se van a convertir los datos
8	child_rame_id	Frame del que se van a convertir los datos

Tabla 11. Parámetros de static_transform_publisher.

Se puede ver mejor con un **ejemplo**, proporcionado por la web oficial de ROS [69]. Se supone que se tiene un láser montado sobre una base móvil a una cierta distancia (en los 3 ejes) respecto de su centro. El láser está localizando un objeto a 0.3 metros de su centro o referencia (base_laser) y se quiere transformar esa medida al centro de la base móvil. El eje X es positivo hacia la derecha y el eje Z positivo hacia arriba.

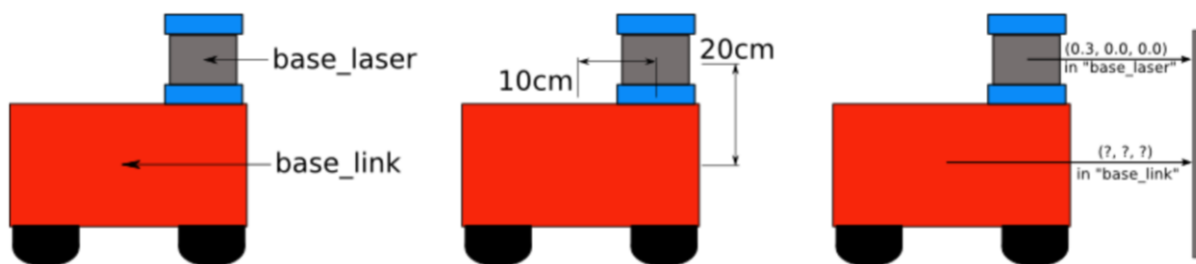


Figura 93. Esquema del ejemplo de la web oficial de ROS [69].

Para conseguir los datos en base_link de base_laser tenemos que aplicar la traslación de (x=0.1m, y=0.0m, z=0.2m), y para conseguir los datos en base_laser de base_link, hay que aplicar la traslación inversa (x=0.1m, y=0.0m, z=-0.20m).

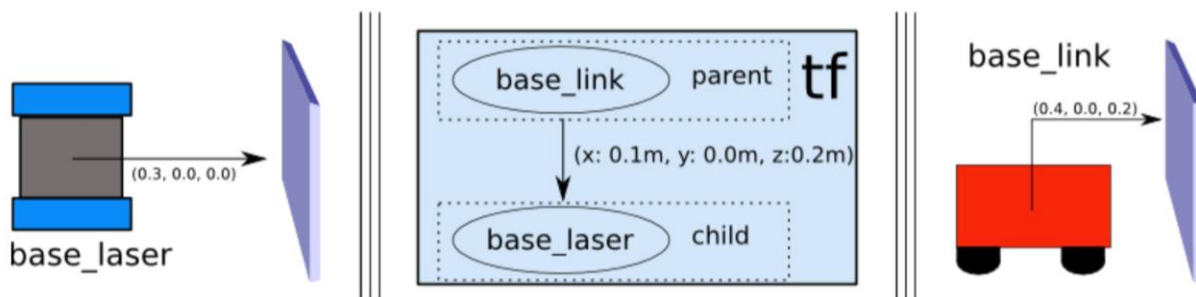


Figura 94. Datos de base_laser en base_link en el ejemplo de la web oficial de ROS [69].

En este caso, **se utilizará como referencia (base_link) la cámara ZED 2i**, ya que se pueden modificar los argumentos de la línea de código en el archivo .launch del radar. **Se hará la conversión en el archivo .launch, justo debajo de la línea 31** (que también hace otra conversión). En este caso, se quiere pasar de ti_mmwave_pcl, que es donde publica el radar a base_link, que es un *frame* común para los dos sensores. Esto se consigue añadiendo la línea de código número 33, como se puede ver en la Figura 95. De momento, **todos los offsets serán 0** ya que solo se va a realizar a modo de prueba para comprobar su correcto funcionamiento. **La cámara y el radar se colocarán muy cerca**, prácticamente en el mismo punto, de tal forma que dejando a

0 todos los argumentos de la transformada se obtenga un buen resultado. Estos argumentos serán útiles más adelante cuando la cámara y el radar no estén en el mismo punto.

```

src > ti_mmwave_rospkg > launch > 6843AOP_multi_3d_0.launch
1  <!--
2  This file will launch rviz along with the mmWave sensor node and configure a TI mmWave 6843 sensor using a 3D config
3  -->
4
5  <launch>
6
7  <!-- Input arguments -->
8  <arg name="device" value="6843" doc="TI mmWave sensor device type [1443, 1642, 6843]"/>
9  <arg name="config" value="3d" doc="TI mmWave sensor device configuration [3d best range res (not supported by 1642 EVM), 2d best range res]"/>
10 <arg name="max allowed elevation angle deg" default="90" doc="Maximum allowed elevation angle in degrees for detected object data [0 > value <= 90]"/>
11 <arg name="max allowed azimuth angle deg" default="90" doc="Maximum allowed azimuth angle in degrees for detected object data [0 > value <= 90]"/>
12
13 <!-- mmWave_Manager node -->
14 <node pkg="ti_mmwave_rospkg" type="ti_mmwave_rospkg" name="ti_mmwave" ns="radar_0" output="screen">
15   <param name="command_port" value="/dev/ttyUSB8" />
16   <param name="command_rate" value="115200" />
17   <param name="data_port" value="/dev/ttyUSB1" />
18   <param name="data_rate" value="921600" />
19   <param name="max allowed elevation angle deg" value="$(arg max allowed elevation angle deg)" />
20   <param name="max allowed azimuth angle deg" value="$(arg max allowed azimuth angle deg)" />
21   <param name="frame_id" value="ti_mmwave_0" />
22   <param name="mmWaveCLI_name" value="/mmWaveCLI" />
23   <remap from="/ti_mmwave/radar_scan_pcl" to="/ti_mmwave/radar_scan_pcl_0"/>
24 </node>
25
26 <!-- mmWaveQuickConfig node (terminates after configuring mmWave sensor) -->
27 <node pkg="ti_mmwave_rospkg" type="mmWaveQuickConfig" name="ti_mmwave_config" ns="radar_0" args="$(find ti_mmwave_rospkg)/cfg/6843AOP_3d.cfg" output="screen">
28   <param name="mmWaveCLI_name" value="/mmWaveCLI" />
29 </node>
30
31 <node pkg="tf" type="static_transform_publisher" name="radar_baseLink_0" args="0 0 0 0 0 0 ti_mmwave_pcl ti_mmwave_0 100"/>
32
33 <node pkg="tf" type="static_transform_publisher" name="radar_baseLink_1" args="0 0 0 0 0 0 base_link ti_mmwave_pcl 100"/>
34
35 <node pkg="rviz" type="rviz" name="rviz" args="-d $(find ti_mmwave_rospkg)/launch/rviz/ti_mmwave_multi.rviz"/>
36 </launch>
37

```

Figura 95. Archivo 6843AOP_multi_3d_0.launch después de realizar los cambios.

Volviendo a lanzar los archivos *launch* y seleccionando en Rviz el frame *base_link*, ya se podrían ver ambas nubes de puntos. El arranque del nodo de la cámara dejará de hacerse de la forma anterior, se usará otro archivo *.launch* pero solo del nodo de ZED, *zed_wrapper*, en vez del nodo con *Rviz*, ejecutando el comando "*roslaunch zed_wrapper zed2i.launch*" De esta forma, *Rviz* solo se abrirá una vez, cuando se lance el nodo del radar. Basta con cargar la configuración guardada anteriormente para poder visualizar los datos correctamente. Como se puede ver en la Figura 96, se han juntado ambas nubes de puntos correctamente en *Rviz*.

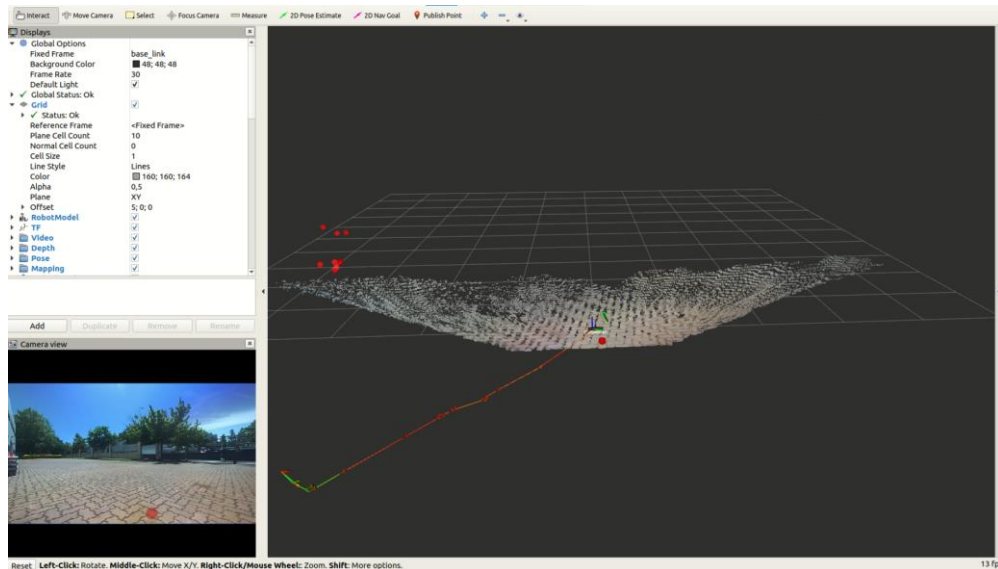


Figura 96. Cámara ZED 2i y radar (Out_of_the_box) en Rviz.

El orden de ejecución de los comandos, cada uno en **diferente terminal** es el siguiente (los comandos 2 y 3 se pueden intercambiar):

1. `roslaunch zed_wrapper zed2i.launch`
2. `roslaunch ti_mmwave_rospkg 6843AOP_multi_3d_0.launch`

- rosservice call /zed2i/zed_node/start_object_detection 2 50 6 true false true true true true true true true (Si se quiere activar la detección de personas)

6.1.10 Rover Aion Robotic R1

El **Rover Aion Robotic R1** (Se le hará referencia como **Rover**) es un robot robusto, ligero diseñado para **aplicaciones de control remoto o movimiento autónomo** [70]. El Rover ya viene listo para programar con código abierto, por ejemplo, con *Nvidia Jetson Xavier AGX*.

Este robot se usará para los **ensayos** en los que se requiera velocidad y en la aplicación final, **se colocará sobre él el radar y la cámara ZED 2i** con la *Nvidia Jetson Xavier AGX Developer Kit* para comprobar el correcto funcionamiento.

Una característica importante es la **velocidad máxima**, que se encuentra la hoja de especificaciones y es, en este caso, **2m/s o 7.2 Km/h** [71].



Figura 97. Rover Aion Robotic R1 [70].

6.2 Demo 3D_People_Counting

En este apartado se cargará en el firmware del radar la demo **3D_People_Counting**, la cual se juntará con la cámara ZED 2i. También **se hará alguna modificación**, para mejorar el funcionamiento del programa y adecuarlo a las necesidades del trabajo.

6.2.1 Comandos de configuración de la demo 3D_People_Counting

Al igual que para el resto de las demos, se le debe enviar al radar un **archivo de configuración**. Algunos comandos los **comparte con OOB**, pero otros muchos son propios de esta demo. Al igual que para OOB, **se nombrarán todos los parámetros y se explicarán los más importantes**. Estos parámetros corresponden a la **capa de detección** [31].

- dfeDataOutputMode: Parámetro de **Out_of_the_box**.
- channelCfg: Parámetro de **Out_of_the_box**.
- adcCfg: Parámetro de **Out_of_the_box**.
- adcbufCfg: Parámetro de **Out_of_the_box**.
- lowPower: Parámetro de **Out_of_the_box**.
- compRangeBiasAndRxChanPhase: Parámetro de **Out_of_the_box**.

- bpmCfg: Parámetro de **Out_of_the_box**.
- profileCfg: Parámetro de **Out_of_the_box**.
- chirpCfg: Parámetro de **Out_of_the_box**.
- frameCfg Parámetro de **Out_of_the_box**.
- dynamicRangeAngleCfg
- dynamicRACfarCfg: Parámetros de configuración para el CFAR para la escena en movimiento o dinámica. Se utiliza para la creación de un mapa de calor rango-ángulo, que posteriormente será usado por el algoritmo para la creación del mapa en 3D.

dynamicRACfarCfg		
Número	Parámetro	Descripción
1	<subFrameIdx>	Índice del <i>subframe</i>
2	<cfarDiscardLeftRange>	Celdas de descarte a la izquierda para el rango
3	<cfarDiscardRightRange>	Celdas de descarte a la derecha para el rango
4	<cfarDiscardLeftAngle>	Celdas de descarte a la izquierda para el ángulo
5	<cfarDiscardRightAngle>	Celdas de descarte a la derecha para el ángulo
6	<refWinSizeRange>	Celdas de referencia para el rango
7	<refWinSizeAngle>	Celdas de referencia para el rango
8	<guardWinSizeRange>	Celdas de guarda para el rango.
9	<guardWinSizeAngle>	Celdas de guarda para el ángulo.
10	<rangeThre>	Umbral para el rango (dB)
11	<angleThre>	Umbral para el ángulo (dB)
12	<sidelobeThre>	Se usa cuando el segundo filtro se habilita, sirve para confirmar los puntos en el dominio del rango (dB).
13	<enable2ndPass>	Habilitar el segundo filtro
14	<dynamicFlag>	Siempre a 1

Tabla 12. Parámetros del comando dynamicRACfarCfg.

- dynamic2DAngleCfg
- staticRangeAngleCfg
- staticRACfarCfg: Parámetros de configuración para el CFAR para la escena en estático. Es idéntico al de la escena en movimiento.

staticRACfarCfg		
Número	Parámetro	Descripción
1	<subFrameIdx>	Índice del subframe
2	<cfarDiscardLeftRange>	Celdas de descarte a la izquierda para el rango
3	<cfarDiscardRightRange>	Celdas de descarte a la derecha para el rango
4	<cfarDiscardLeftAngle>	Celdas de descarte a la izquierda para el ángulo
5	<cfarDiscardRightAngle>	Celdas de descarte a la derecha para el ángulo
6	<refWinSizeRange>	Celdas de referencia para el rango
7	<refWinSizeAngle>	Celdas de referencia para el rango
8	<guardWinSizeRange>	Celdas de guarda para el rango.
9	<guardWinSizeAngle>	Celdas de guarda para el ángulo.

10	<rangeThre>	Umbral para el rango (dB)
11	<angleThre>	Umbral para el ángulo (dB)
12	<sidelobeThre>	Se usa cuando el segundo filtro se habilita, sirve para confirmar los puntos en el dominio del rango (dB).
13	<enable2ndPass>	Habilitar el segundo filtro
14	<dynamicFlag>	Siempre a 0

Tabla 13. Parámetros del comando staticRACfarCfg.

- fineMotionCfg: Configuración del modo de detección para **detectar personas en estático**.
- antGeometry0: Define la **geometría de la antena**.

antGeometry0		
Número	Parámetro	Descripción
1	<virtAntIdx0>	Localización de la antena virtual 0 en el azimut.
2	<virtAntIdx1>	Localización de la antena virtual 1 en el azimut.
3	<virtAntIdx2>	Localización de la antena virtual 2 en el azimut.
4	<virtAntIdx3>	Localización de la antena virtual 3 en el azimut.
5	<virtAntIdx4>	Localización de la antena virtual 4 en el azimut.
6	<virtAntIdx5>	Localización de la antena virtual 5 en el azimut.
7	<virtAntIdx6>	Localización de la antena virtual 6 en el azimut.
8	<virtAntIdx7>	Localización de la antena virtual 7 en el azimut.
9	<virtAntIdx8>	Localización de la antena virtual 8 en el azimut.
10	<virtAntIdx9>	Localización de la antena virtual 9 en el azimut.
11	<virtAntIdx10>	Localización de la antena virtual 10 en el azimut.
12	<virtAntIdx11>	Localización de la antena virtual 11 en el azimut.

Tabla 14. Parámetros del comando antGeometry0.

- antGeometry1: Define la **geometría de la antena**.

antGeometry1		
Número	Parámetro	Descripción
1	<virtAntIdx0>	Localización de la antena virtual 0 en la elevación.
2	<virtAntIdx1>	Localización de la antena virtual 1 en la elevación.
3	<virtAntIdx2>	Localización de la antena virtual 2 en la elevación.
4	<virtAntIdx3>	Localización de la antena virtual 3 en la elevación.
5	<virtAntIdx4>	Localización de la antena virtual 4 en la elevación.
6	<virtAntIdx5>	Localización de la antena virtual 5 en la elevación.
7	<virtAntIdx6>	Localización de la antena virtual 6 en la elevación.
8	<virtAntIdx7>	Localización de la antena virtual 7 en la elevación.
9	<virtAntIdx8>	Localización de la antena virtual 8 en la elevación.
10	<virtAntIdx9>	Localización de la antena virtual 9 en la elevación.
11	<virtAntIdx10>	Localización de la antena virtual 10 en la elevación.
12	<virtAntIdx11>	Localización de la antena virtual 11 en la elevación.

Tabla 15. Parámetros del comando antGeometry1.

Para entender estos dos parámetros es necesario entender cómo se forman las **antenas virtuales**. Los radares XWR6843AoP tienen la siguiente **disposición de antenas y canales de alimentación**.

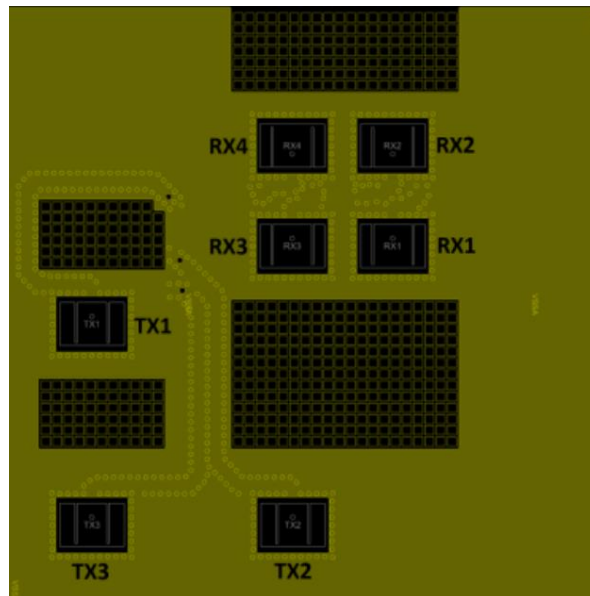


Figura 98. Antenas y canales de alimentación de los radares XWR6843AoP [31].

También se pueden **observar estas antenas directamente en el radar** (Figura 99), y se puede comprobar que la disposición de las antenas coincide con lo que dice el fabricante.

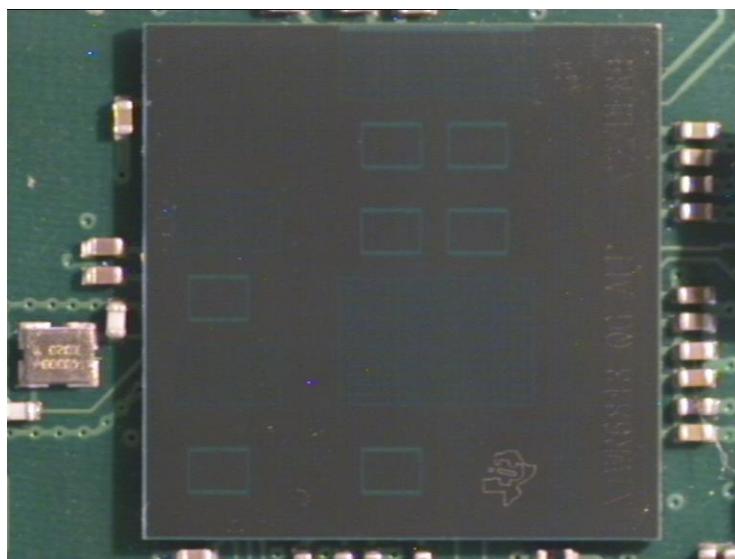


Figura 99. Colocación de las antenas del radar.

Para crear el **patrón de las antenas virtuales** se **sustituye cada antena transmisora por el bloque de antenas receptoras**. Por ejemplo, para TX1 se sustituye el bloque con las antenas RX1, RX2, RX3 y RX4. La nomenclatura de las antenas virtuales sigue el orden de la numeración de las antenas receptoras. Una vez hecho ese proceso con todas las antenas transmisoras, las antenas virtuales quedan de la siguiente forma:

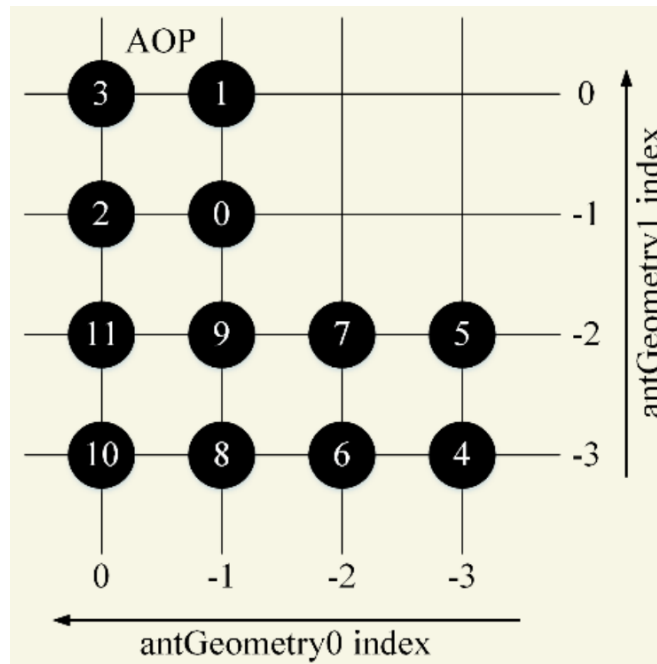


Figura 100. Patrón de antenas virtuales para el IWR6843AoP [31].

Los números que hay que colocar en los comandos `antGeometry0` y `antGeometry1` son los que se ven en la Figura 100. Para `antGeometry0` el número para la antena virtual 0 es -1, el de la antena virtual 1 es -1 otra vez, el de la antena virtual 2 es 0, y así sucesivamente hasta llegar a la antena virtual 11. Se repite exactamente el mismo proceso para `antGeometry1`.

Para el caso del IWR6843ISK, se pueden diferenciar mejor las antenas, al estar **fuera del chip**. Se puede ver tanto en el radar como en la Figura 101 que las antenas están colocadas de forma diferente que para el radar IWR6843AoP, lo que da lugar a **otro patrón de antenas virtuales**, el cual se puede ver en la Figura 102. El proceso para rellenar los comandos de configuración `antGeometry0` y `antGeometry1` es **idéntico** al caso anterior.

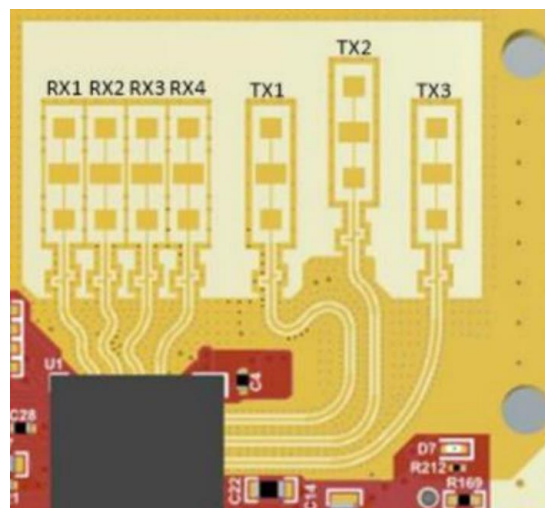


Figura 101. Antenas y canales de alimentación de los radares XWR6843ISK [31].

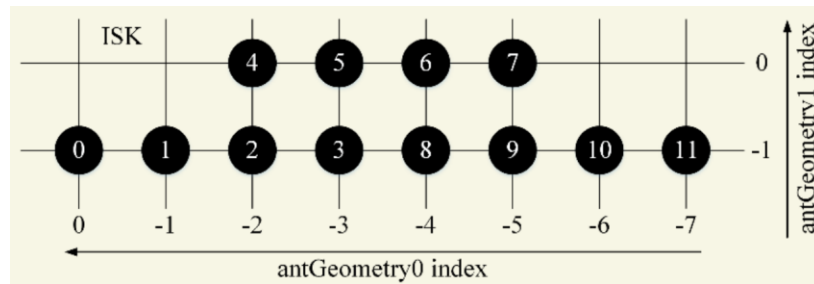


Figura 102. Patrón de antenas virtuales para el IWR6843ISK [31].

- antPhaseRot: Define la **rotación de las antenas**.

antPhaseRot		
Número	Parámetro	Descripción
1	<virtAntIdx0>	Rotación de la antena virtual 0.
2	<virtAntIdx1>	Rotación de la antena virtual 1.
3	<virtAntIdx2>	Rotación de la antena virtual 2.
4	<virtAntIdx3>	Rotación de la antena virtual 3.
5	<virtAntIdx4>	Rotación de la antena virtual 4.
6	<virtAntIdx5>	Rotación de la antena virtual 5.
7	<virtAntIdx6>	Rotación de la antena virtual 6.
8	<virtAntIdx7>	Rotación de la antena virtual 7.
9	<virtAntIdx8>	Rotación de la antena virtual 8.
10	<virtAntIdx9>	Rotación de la antena virtual 9.
11	<virtAntIdx10>	Rotación de la antena virtual 10.
12	<virtAntIdx11>	Rotación de la antena virtual 11.

Tabla 16. Parámetros del comando antPhaseRot.

Para rellenar los campos de este comando hay que **observar por dónde se alimentan las antenas receptoras**, y dependiendo de eso, colocar un 1 o un -1. Estas antenas y canales de alimentación también se pueden llegar a observar directamente en el radar. Se puede ver en la Figura 103 que los canales de alimentación de las antenas transmisoras coinciden con lo que dice el fabricante.

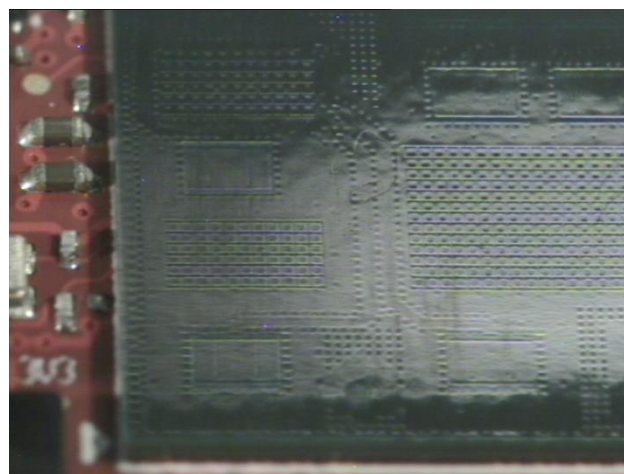


Figura 103. Antenas transmisoras del IWR6843AoP.

En la Figura 104 se puede ver que los canales de alimentación de las antenas receptoras también coinciden con lo que explica el fabricante, aunque en este caso no se distinguen con tanta claridad como las antenas transmisoras

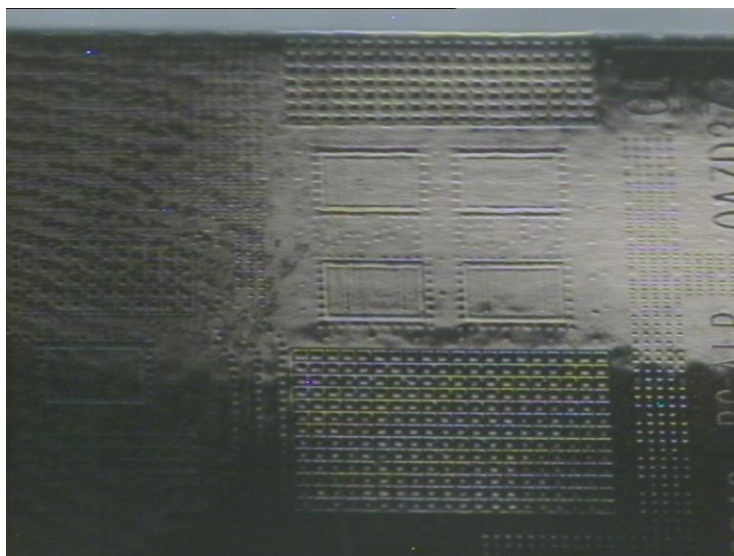


Figura 104. Antenas receptoras del IWR6843AoP.

Para el caso del IWR6843AoP, **no todas las antenas receptoras se alimentan por el mismo lugar**. La antena receptora RX1 recibe alimentación por la parte superior, se le puede asignar el valor positivo o negativo, es indiferente, pero hay que tenerlo en cuenta a la hora de asignar valores al resto de antenas. En este caso se le asignará un 1. A la antena RX2 le llega alimentación por la parte inferior, luego el valor que hay que asignarle es -1. Para la RX3 se le debe asignar un 1 y a la RX4 un -1. Con esto se habrían rellenado los campos para las 4 primeras antenas virtuales, por lo tanto, se debe repetir exactamente el mismo patrón 2 veces más.

Para el caso del IWR6843ISK es más sencillo, ya que todas las antenas están alimentadas por el mismo sitio, por la parte inferior (Figura 101), luego todos los parámetros de este comando serán 1.

- fovCfg: Configuración del **campo de visión** (acimut y elevación).

Los siguientes parámetros corresponden a **parámetros de tracking** que se pueden ver en el otro documento disponible para la configuración de parámetros [32].

- boundaryBox: En este comando se define la zona donde trabajará el **algoritmo de tracking**.

boundaryBox		
Número	Parámetro	Descripción
1	<X-min>	Valor mínimo de la coordenada X .
2	<X-max>	Valor máximo de la coordenada X .
3	<Y-min>	Valor mínimo de la coordenada Y .
4	<Y-max>	Valor máximo de la coordenada Y .
5	<Z-min>	Valor mínimo de la coordenada Z .
6	<Z-max>	Valor máximo de la coordenada Z .

Tabla 17. Parámetros del comando boundaryBox.

- **staticBoundingBox**: Zona en la que se espera que los **objetos permanezcan quietos**. Los parámetros son los mismos que los de la Tabla 17.

Es recomendable que **boundaryBox** sea mayor que **staticBoundingBox**.

- **presenceBoundingBox**: Zona donde se prevé que **haya objetos**. Los parámetros son los mismos que los de la Tabla 17.
- **sensorPosition**: **Posición en el eje z y orientación** del radar.

sensorPosition		
Número	Parámetro	Descripción
1	<sensorHeight>	Altura del radar respecto al suelo.
2	<azimTilt>	Rotación del radar respecto al eje Z.
3	<elevTilt>	Rotación del radar respecto al eje X.

Tabla 18. Parámetros del comando *sensorPosition*.

- **gatingParam**: **Límites para la detección** de puntos y objetos.
- **stateParam**: **Numero de frames para pasar de un estado a otro**.

stateParam		
Número	Parámetro	Descripción
1	<set2actThre>	Número de detecciones para empezar el tracking .
2	<set2freeThre>	Número de no detecciones para que un grupo de puntos (del que no se había empezado el tracking) deje de considerarse como posible objeto .
3	<active2freeThre>	Número de no detecciones para dejar de hacer tracking a un objeto que estaba en movimiento .
4	<static2freeThre>	Número de no detecciones para que un objeto estático desaparezca
5	<exit2freeThre>	Determina el tiempo de vida de un objeto fuera de la “static boundary-box”
6	<sleep2freeThre>	Determina el tiempo de vida de un objeto dentro de la “static boundary-box”

Tabla 19. Parámetros del comando *stateParam*.

- **allocationParam**: Determina si **un punto se puede asociar a un grupo de puntos** ya existente para formar un centroide. Permite también formar nuevos centroides.
- **maxAcceleration**: Máximo valor que se espera que cambie la aceleración.
- **trackingCfg**: **Configuración básica** del algoritmo de *tracking*.

6.2.2 TLV de la demo 3D_People_Counting

La demo 3D_People_Counting **proporciona 4 TLV**:

- MMWDEMO_OUTPUT_MSG_COMPRESSED_POINTS, **identificador 1020**
- MMWDEMO_OUTPUT_MSG_TRACKERPROC_3D_TARGET_LIST, **identificador 1010**
- MMWDEMO_OUTPUT_MSG_TRACKERPROC_TARGET_INDEX, **identificador 1011**
- MMWDEMO_OUTPUT_MSG_TRACKERPROC_TARGET_HEIGHT, **identificador 1012**

6.2.3 3D_People_Counting con ROS

Esta demo sólo se puede usar para el radar **IWR6843AoP**, por lo que se utilizará el **uRad Industrial v1.0**. Es importante **cargar en el firmware del radar la demo que viene en el paquete de ROS** (catkin_ws/src/mmwave_ti_ros/ros_driver/src/ti_mmwave_tracker_rospkg/bin). Se ha añadido recientemente el TLV MMWDEMO_OUTPUT_MSG_TRACKERPROC_TARGET_HEIGHT (identificador 1012), pero **no se ha actualizado el paquete de ROS** para poder soportarlo. Si se carga la última versión de la demo, no funcionaría tan bien como la versión que se encuentra en el paquete de ros y, además, eventualmente el radar **dejaría de dar la nube de puntos** debido al error (Std::out_of_range) por no poder soportar el nuevo TLV. Para arrancar el nodo de ROS se debe usar el comando **“roslaunch ti_mmwave_tracker_rospkg AOP_3d_Tracking.launch”**.

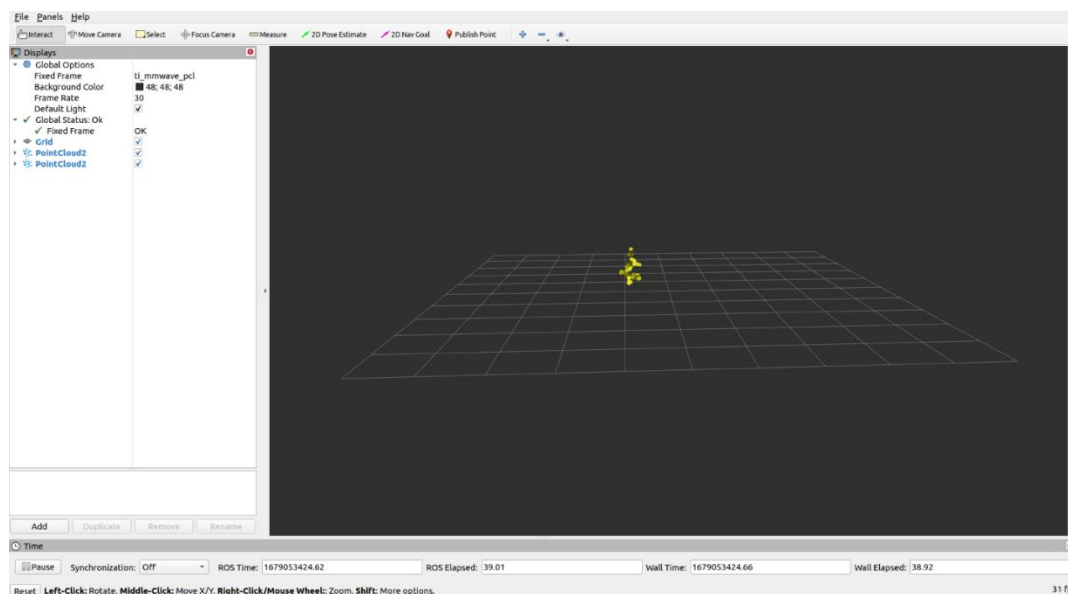


Figura 105. Nube de puntos de la detección de una persona.

6.2.4 Creación de un nuevo nodo de ROS: ti_mmwave_trackarray_to_pcl

El nodo de ROS de ti_mmwave_tracker_package (paquete de ROS proporcionado por Texas Instruments) publica los **siguientes topics**:

- /ti_mmwave/radar_scan
- /ti_mmwave/radar_scan_markers
- /ti_mmwave/radar_scan_pcl_0
- /ti_mmwave/radar_scan_pcl_1
- **/ti_mmwave/radar_trackarray**
- /ti_mmwave/radar_trackid

En la primera parte, el nuevo nodo de ROS se va a centrar en **obtener los valores que publica ti_mmwave/radar_trackarray**, es decir, se va a **suscribir** a ese **topic** y a su vez va a **publicar** cierta información. Este **topic** del radar publica, entre otras cosas, **posición, velocidad y aceleración del centroide** de un objeto en 3 dimensiones, por lo que el nuevo nodo, publicará los datos de posición en forma de nube de puntos y al lado de estos puntos una etiqueta con el módulo de su velocidad.

6.2.4.a Creación de un nuevo nodo de ROS

En primer lugar, se creará un **nuevo paquete de ROS**. Se seguirá el **formato de paquete de ROS de Antoni Rosinol**. Para ello, hay que entrar en la carpeta `ros_driver/src` y ejecutar el siguiente comando en el terminal (se realiza dentro del driver del radar ya que es algo complementario y de esta forma se tienen todos los paquetes juntos). Lo primero que hay que hacer es ejecutar el comando `"bash -c "$(curl -sLf https://raw.githubusercontent.com/ToniRV/catkin_package_maker/master/make_catkin_pkg.sh)"` para copiar los archivos necesarios para el nuevo nodo.

Una vez ejecutado, dará la opción de ponerle nombre al paquete, en este caso **se nombrará `ti_mmwave_tracker_to_pcl`**. A continuación, se deberán **modificar algunos archivos** de dentro del paquete: **CMakeLists.bit** (Figura 106), **ti_mmwave_tracker_to_pcl.launch** (Figura 107), **LICENSE** (Figura 108), **package.xml** (Figura 109) y **setup.py** (Figura 110). Todos los archivos anteriores **excepto LICENSE** son indispensables para el correcto funcionamiento.

```

1  cmake_minimum_required(VERSION 2.8.3)
2  project(ti_mmwave_tracker_to_pcl)
3
4  find_package(catkin_simple REQUIRED COMPONENTS
5    | ti_mmwave_tracker_rosepkg
6  )
7
8  catkin_python_setup()
9
10 catkin_simple()
11
12 cs_add_executable(${PROJECT_NAME}
13   | src/${PROJECT_NAME}.cpp)
14
15 cs_install()
16
17 cs_export()
18

```

Figura 106. Archivo CMakeLists.bit del nodo `ti_mmwave_tracker_to_pcl`.

```

1  <launch>
2  <arg name="verbose" default="false" doc="Activate verbose mode"/>
3
4  <node name="ti_mmwave_tracker_to_pcl" pkg="ti_mmwave_tracker_to_pcl" type="ti_mmwave_tracker_to_pcl" output="screen" args="">
5    <param name="verbose" type="bool" value="$(arg verbose)"/>
6  </node>
7 </launch>
8

```

Figura 107. Archivo `ti_mmwave_tracker_to_pcl.launch` del nodo `ti_mmwave_tracker_to_pcl`.

```

1  Copyright (c) 2023 FUNDACION I+D AUTOMOCION Y MECATRONICA (All rights reserved)

```

Figura 108. Archivo LICENSE del nodo `ti_mmwave_tracker_to_pcl`.

```

1  <?xml version="1.0"?>
2  <package format="2">
3    <name>ti_mmwave_tracker_to_pcl</name>
4    <version>1.0.0</version>
5    <description>ROS node that converts trackarray data to point cloud</description>
6
7    <maintainer email="jagarcia@naitec.es">José Antonio García</maintainer>
8    <license>Proprietary</license>
9
10   <buildtool_depend>catkin</buildtool_depend>
11   <buildtool_depend>catkin_simple</buildtool_depend>
12
13   <build_depend>roscpp</build_depend>
14   <build_depend>ti_mmwave_tracker_rosepkg</build_depend>
15
16 </package>
17

```

Figura 109. Archivo package.xml del nodo ti_mmwave_trackarray_to_pcl.

```

1  #!/usr/bin/env python
2
3  from distutils.core import setup
4  from catkin_pkg.python_setup import generate_distutils_setup
5
6  setup_args = generate_distutils_setup(
7      name='ti_mmwave_tracker_to_pcl',
8      version='1.0.0',
9      description='ROS node that converts trackarray data to point cloud',
10     packages=['ti_mmwave_tracker_to_pcl'],
11     package_dir={'': 'src'})
12 )
13
14 setup(**setup_args)
15

```

Figura 110. Archivo setup.py del nodo ti_mmwave_trackarray_to_pcl.

A continuación, se vuelve a ros_driver y se compila. Este nodo se va a **compilar mediante catkin build** en vez de catkin_make. **La única diferencia es estética**, ya que *catkin build* muestra de una forma más limpia el proceso (Figura 111). Se puede ver qué paquete se ha compilado correctamente y cuál no, así como el tiempo que ha tardado en compilarse, lo cual ayuda a la detección de errores.

```

Workspace configuration appears valid.
[build] Found 22 packages in 0.6 seconds.
[build] package table is up to date.
Starting >>> catkin_simple
Starting >>> serial
Starting >>> zed_interfaces
Finished <<< catkin_simple [ 0.3 seconds ]
Finished <<< serial [ 0.4 seconds ]
Starting >>> ti_mmwave_rosepkg
Starting >>> ti_mmwave_tracker_rosepkg
Finished <<< zed_interfaces [ 2.4 seconds ]
Starting >>> rviz_plugin_zed_od
Starting >>> zed_nodelets
Finished <<< ti_mmwave_tracker_rosepkg [ 2.3 seconds ]
Starting >>> ti_mmwave_tracker_to_pcl
Finished <<< ti_mmwave_rosepkg [ 2.1 seconds ]
Finished <<< rviz_plugin_zed_od [ 0.7 seconds ]
Finished <<< zed_nodelets [ 1.5 seconds ]
Starting >>> zed_sync_test
Starting >>> zed_wrapper
Finished <<< zed_wrapper [ 0.8 seconds ]
Starting >>> zed_ar_track_alvar_example
Starting >>> zed_depth_sub_tutorial
Starting >>> zed_display_rviz
Starting >>> zed_multicamera_example
Starting >>> zed_nodelet_example
Starting >>> zed_obj_det_sub_tutorial
Finished <<< zed_sync_test [ 1.0 seconds ]
Starting >>> zed_ros
Finished <<< zed_depth_sub_tutorial [ 0.4 seconds ]
Starting >>> zed_rtabmap_example
Finished <<< zed_ar_track_alvar_example [ 0.3 seconds ]
Starting >>> zed_sensors_sub_tutorial
Finished <<< zed_display_rviz [ 0.3 seconds ]
Finished <<< zed_nodelet_example [ 0.3 seconds ]
Starting >>> zed_tracking_sub_tutorial
Finished <<< zed_multicamera_example [ 0.4 seconds ]
Starting >>> zed_video_sub_tutorial
Finished <<< zed_ros [ 0.4 seconds ]
Finished <<< zed_obj_det_sub_tutorial [ 1.2 seconds ]
Finished <<< zed_rtabmap_example [ 0.4 seconds ]
Finished <<< zed_sensors_sub_tutorial [ 0.4 seconds ]
Finished <<< zed_video_sub_tutorial [ 0.4 seconds ]
Finished <<< zed_tracking_sub_tutorial [ 0.4 seconds ]
Starting >>> zed_examples
Finished <<< zed_examples [ 0.3 seconds ]
Finished <<< ti_mmwave_tracker_to_pcl [ 7.5 seconds ]
[build] Summary: All 22 packages succeeded!
[build] Ignored: None.
[build] Warnings: None.
[build] Abandoned: None.
[build] Failed: None.
[build] Runtime: 10.9 seconds total.

```

Figura 111. Compilación mediante catkin build.

Para ello hay que ejecutar el comando “*pip install catkin tools*” o “*pip3 install catkin tools*” si el anterior no funciona. **Si se ha hecho un catkin_make anteriormente** hay que eliminar las carpetas *build* y *devel* del *workspace* mediante el comando “*sudo rm -r build/ devel/*”. Ahora ya se puede compilar con *catkin build*. En este caso, **dará error ya que no se tiene el catkin_simple**. Para ello se ejecuta en la carpeta *src* del *catkin_ws* el comando “*git clone https://github.com/catkin/catkin_simple.git*”. Ahora sí que se podrá compilar mediante *catkin build*.

6.2.4.b Publicación de un centroide como nube de puntos

Este programa buscará recoger la información de la posición de objetos y publicarla como nube de puntos para poder verla en *Rviz*. En la primera parte, se **recogerá únicamente la información de la posición** y se mostrará en el terminal (sin publicarlo). En la segunda parte, transformará esos valores de las posiciones en

una **nube de puntos**, y se **publicará**. Seguidamente **Rviz se suscribirá a ese topic** para que se puedan visualizar los centroides. Es importante que **el nodo del radar esté en funcionamiento**, ya que de lo contrario no se recibirían datos de centroides.

El programa del nodo se escribirá en el archivo **ti_mmwave_tracker_to_pcl.cpp**, dentro de la carpeta **src/**. Como la extensión del archivo lo indica, **el lenguaje de programación será C++**. Antes de suscribirse al **topic** deseado, se hará un **programa que se suscriba a un topic creado mediante un comando** que publique la palabra **“Hello”**. De esta forma se comprobará que se suscribe correctamente. El código se puede ver en la Figura 112.

La primera parte del programa son **#include**, que sirve para **incluir las librerías necesarias** (las 3 primeras líneas). De la 5 a la 8 es lo que se conoce como **“callback”**. Únicamente **recibe el mensaje y lo muestra en pantalla**. En el bucle principal o bucle **main**, de la línea 11 a la 30 sirven para **inicializar el nodo de ROS, definir variables, etc**. En las líneas de la 32 a la 36 se encuentran algunos logs (comentados) que sirven para dar información o avisos según se requiera. Por último, las líneas 41 (la que llama a la función recibido) y 42 sirven para **suscribirse al topic**.

```

1 #include <ros/ros.h>
2 #include "std_msgs/String.h"
3 #include "ti_mmwave_tracker_to_pcl/ti_mmwave_tracker_to_pcl.h"
4
5 void recibido(const std_msgs::String msg)//Mensaje tipo RadarTrackArray, lo guardo en msg
6 {
7     ROS_INFO("I heard: [%s]", msg->data.c_str());
8 }
9
10 int main(int argc, char *argv[]) {
11     bool verbose;
12
13     // Initialize ROS node
14     ros::init(argc, argv, "ti_mmwave_tracker_to_pcl");
15
16     ROS_INFO("Starting tracker to pcl node...");
17
18     ros::NodeHandle nh;
19
20     //Get parameters from launch file
21     nh.param<bool>("ti_mmwave_tracker_to_pcl/verbose",verbose,false);
22
23     if(verbose)
24     {
25         | ros::console::set_logger_level(ROSCONSOLE_DEFAULT_NAME, ros::console::levels::Debug);
26     }
27     else
28     {
29         | ros::console::set_logger_level(ROSCONSOLE_DEFAULT_NAME, ros::console::levels::Info);
30     }
31
32     //ROS_INFO("This is a INFO log");
33     //ROS_DEBUG("This is a DEBUG log");
34     //ROS_WARN("This is a WARN log");
35     //ROS_ERROR("This is a ERROR log");
36     //ROS_FATAL("This is a FATAL log");
37
38     ros::Rate rate(100); //Hz
39     while (ros::ok())
40     {
41         | ROS::Subscriber sub = nh.subscribe("/mystring", 10, recibido);
42         | ros::spin();
43     }
44     return EXIT_SUCCESS;
45 }
46

```

Figura 112. Programa para leer el topic /mystring.

El comando para **generar el topic /mystring** se puede ver en la Figura 113, la **comprobación de que el topic publica correctamente** en la Figura 114 y el nuevo nodo **recogiendo los datos y mostrándolos en el terminal** en la Figura 115.

```

~/catkin_radar_ws$ rostopic pub /mystring std_msgs/String 'Hello'
publishing and latching message. Press ctrl-C to terminate

Debug Options
Node Name          ti_mmwave_tracker_to_pcl
Background Color   #000000

```

Figura 113. Comando que genera el topic mystring que publica el mensaje "Hello".

```

~/catkin_radar_ws$ rostopic echo /mystring
data: 'Hello'
***

Debug Options
Node Name          ti_mmwave_tracker_to_pcl
Background Color   #000000

```


Figura 114. Comando para comprobar que el topic mystring publica correctamente.

Para ejecutar el nuevo nodo hay que usar el comando **“roslaunch ti_mmwave_tracker_to_pcl ti_mmwave_tracker_to_pcl.launch verbose:=true”**.

```

~/catkin_radar_ws$ roslaunch ti_mmwave_tracker_to_pcl ti_mmwave_tracker_to_pcl.launch verbose:=true
... logging to /home/.../.ros/log/99841f8e-cd54-11ed-80ff-48b02d185740/roslaunch-...-agx-32605.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://.../

SUMMARY
=====
PARAMETERS
* /roscpp: melodic
* /rosversion: 1.14.13
* /ti_mmwave_tracker_to_pcl/verbose: True

NODES
/
  ti_mmwave_tracker_to_pcl (ti_mmwave_tracker_to_pcl/ti_mmwave_tracker_to_pcl)

ROS_MASTER_URI=http://localhost:11311

process[ti_mmwave_tracker_to_pcl-1]: started with pid [32629]
[ INFO] [1680015452.894595706]: Starting tracker to pcl node...
[ INFO] [1680015453.144187306]: I heard: [Hello]
    
```

Figura 115. Nodo ti_mmwave_trackerarray_to_pcl publicando el mensaje "Hello".

Una vez comprobado que se suscribe correctamente hay que **recoger los datos de la posición** del centroide del objeto. Antes de ello, hay que entender cómo se utiliza un mensaje.

El topic ti_mmwave/radar_scan_trackarray publica un tipo de mensaje que **no es un mensaje que proporciona ROS**, es un mensaje personalizado (Figura 116), por lo que se debe **definir su formato**. El formato del mensaje se puede ver detalladamente en los archivos .msg, en la carpeta ti_mmwavetracker_ropkg/msg (Figura 117 y Figura 118).

```

---
header:
  seq: 193
  stamp:
    secs: 1680014876
    nsecs: 74104785
  frame_id: "ti_mmwave_1"
num_tracks: 1
track:
  header:
    seq: 0
    stamp:
      secs: 1680014876
      nsecs: 74106513
    frame_id: "ti_mmwave_1"
  tid: 3
  posx: 0.04657086134
  posy: -0.0850017517805
  posz: 0.00223898515105
  velx: 0.0578896030784
  vely: 0.0977546349168
  velz: 0.216250896454
  accx: 0.0211008302867
  accy: 0.0253685060889
  accz: 0.0577674992383
---
    
```

Figura 116. Datos que publica el topic ti_mmwave/radar_scan_trackarray



```
Open RadarTrackArray.msg
Header header
uint32 num_tracks
RadarTrackContents[] track
```

Plain Text Tab width: 8 Ln. 3, Col. 77 INS

Figura 117. Contenido del mensaje RadarTrackArray.msg.

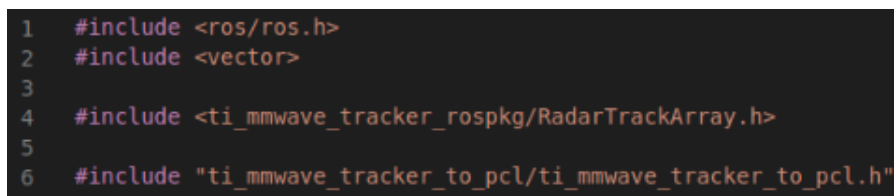


```
Open RadarTrackContents.msg
Header header
uint32 id
float32 posx
float32 posy
float32 posz
float32 velx
float32 vely
float32 velz
float32 accx
float32 accy
float32 accz
```

Plain Text Tab width: 8 Ln. 3, Col. 8 INS

Figura 118. Contenido del mensaje RadarTrackContents.msg.

Sabiendo como es el formato del mensaje, ya se puede realizar la programación para recibir los datos. Habrá que añadir los `#include` necesarios para poder crear un **vector dinámico** y leer el *topic* del que queremos extraer los datos (Figura 119).



```
1 #include <ros/ros.h>
2 #include <vector>
3
4 #include <ti_mmwave_tracker_rospkg/RadarTrackArray.h>
5
6 #include "ti_mmwave_tracker_to_pcl/ti_mmwave_tracker_to_pcl.h"
```

Figura 119. #include necesarios para leer el topic.

La función “recibido”, también tendrá que cambiar (Figura 120). **Se le cambiará el nombre a TrackarrayCallback** ya que el nombre anterior era únicamente para hacer la prueba. Se asignarán a las variables y vectores dinámicos (que serán definidos si es necesario) **los valores del mensaje RadarTrackArray**

(líneas 11, 15, 19, 21, 23 y 25) y se **mostrarán en el terminal** mediante `ROS_DEBUG()`. El **bucle `for`** es para obtener todos los datos en el caso de que haya **más de un objeto a la vez**.

```

8 void TrackarrayCallback(const ti_mmwave_tracker_rospkg::RadarTrackArray msg)//Mensaje tipo RadarTrackArray, lo guardo en msg
9 {
10 ROS_DEBUG("Received TrackArray data...");
11 uint32_t numTrack = msg.num_tracks; //Convierto a unsigned int 32 el numero total de tracks
12 ROS_DEBUG("> Number of tracks: %d",numTrack);
13
14 std::vector<ti_mmwave_tracker_rospkg::RadarTrackContents> tracks; //Creo un vector dinamico llamado tracks
15 tracks = msg.track; //Al vector dinamico le doy los valores de tracks
16
17 for(unsigned int i = 0; i < tracks.size();i++)
18 {
19     uint32_t tid = tracks[i].tid; //Convierto a unsigned int el parametro tid del vector tracks
20     ROS_DEBUG(">> Track [%d] tid: %d",i,tid);
21     float posx = tracks[i].posx;
22     ROS_DEBUG(">> Track [%d] posx: %f",i,posx);
23     float posy = tracks[i].posy;
24     ROS_DEBUG(">> Track [%d] posy: %f",i,posy);
25     float posz = tracks[i].posz;
26     ROS_DEBUG(">> Track [%d] posz: %f",i,posz);
27 }
28 }
    
```

Figura 120. Función `TrackarrayCallback`.

El bucle **`while (ros::ok())`** se eliminará, ya que no es necesario. La función `ros::spin()` hace que el programa no avance más allá de esa línea de código (se queda ejecutándose un bucle infinito dentro de él). El cambio que se hace es en la línea 59 (Figura 121), donde se **cambia el nombre del `topic` al que debe suscribirse**.

```

57
58 ros::Rate rate(100); //Hz
59 ros::Subscriber sub = nh.subscribe("/ti_mmwave/radar_trackarray", 10, TrackarrayCallback);
60 ros::spin();
61
62 return EXIT_SUCCESS;
63 }
    
```

Figura 121. Cambio en la función del suscriptor.

Para una mejor comprensión del programa, y en concreto, del bucle `main`, en la Figura 122 y en la Figura 123 se pueden ver **diagramas de flujo** tanto del programa como del bucle `main`, ya que la estructura de estos no se volverá a cambiar.



Figura 122. Diagrama de flujo del programa.

Lo primero que se hace es **incluir todos los elementos necesarios** y definir todas las variables y elementos necesarios. A continuación, se **define el bucle TrackArrayCallback** y se **entra al bucle main**. El diagrama de flujo del bucle main se puede ver en la **¡Error! No se encuentra el origen de la referencia.**

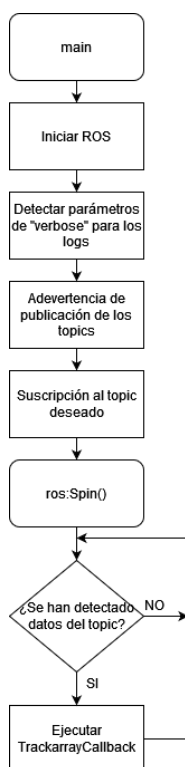


Figura 123. Diagrama de flujo del bucle main.

En el bucle principal, lo primero que se hace es **iniciar ROS** y **detectar el parámetro “verbose”** para mostrar logs por el terminal. A continuación, se **advertir de la publicación de los topics** y se **suscribe al topic deseado**.

Tras ello se llega a **“ros::Spin()”**, que es un **bucle infinito** del que nunca se sale. Cuando detecte algún *topic*, ejecutará el *Callback*, que se introduce como parámetro en la suscripción. Si no se detecta ningún *topic* publicado, se sigue buscando hasta encontrar uno.

Se diferencia de un bucle *main* en que al entrar a *ros::Spin()* no se ejecuta ninguna línea de comando del programa. Esta función pertenece a la **librería de ROS**, y, por tanto, hay un cierto código que se ejecuta. Este código que se ejecuta tendrá un **bucle infinito ejecutándose e impidiendo que el programa en C++ avance a la siguiente instrucción**. Con *ros::SpinOnce()* se lleva a cabo el mismo procedimiento, pero ese bucle que se ejecuta infinitamente dentro de *ros::Spin()* se ejecute **una única vez** y se pueda ejecutar la siguiente instrucción del programa en C++.

En el momento que se detectan datos, se **ejecuta TrackArrayCallback** y se vuelve a comprobar si han llegado nuevos datos. Este es un **programa basado en eventos**, donde el evento que hace que se empiecen a ejecutar otros bucles (en este caso *TrackArrayCallback*) es la **llegada de datos del topic** de ROS

Ya estaría **completado el programa suscriptor**, y se puede ver en la Figura 124 que los datos recibidos se muestran en el terminal correctamente.

```

DEBUG [1688015523.908517925] ==> Track [0] posiz: 0.065310
DEBUG [1688015523.908564507] ==> Track [0] posiz: 0.165371
DEBUG [1688015523.908621340] Received TrackArray data...
DEBUG [1688015523.908633131] ==> Number of Tracks: 1
DEBUG [1688015523.908633924] ==> Track [0] kid: 3
DEBUG [1688015523.908631528] ==> Track [0] posiz: 0.087667
DEBUG [1688015523.908629032] ==> Track [0] posiz: 0.062958
DEBUG [1688015523.908638397] ==> Track [0] posiz: 0.132499
DEBUG [1688015524.031020840] Received TrackArray data...
DEBUG [1688015524.031033793] ==> Number of Tracks: 1
DEBUG [1688015524.031208241] ==> Track [0] kid: 3
DEBUG [1688015524.031044972] ==> Track [0] posiz: 0.087667
DEBUG [1688015524.031308902] ==> Track [0] posiz: 0.062958
DEBUG [1688015524.031346743] ==> Track [0] posiz: 0.132499
DEBUG [1688015524.008231923] Received TrackArray data...
DEBUG [1688015524.008485720] ==> Number of Tracks: 1
DEBUG [1688015524.008607889] ==> Track [0] kid: 3
DEBUG [1688015524.008629032] ==> Track [0] posiz: 0.087667
DEBUG [1688015524.008109315] ==> Track [0] posiz: 0.062958
DEBUG [1688015524.008197927] ==> Track [0] posiz: 0.132499
DEBUG [1688015524.138227836] Received TrackArray data...
DEBUG [1688015524.138358197] ==> Number of Tracks: 1
DEBUG [1688015524.138544355] ==> Track [0] kid: 3
DEBUG [1688015524.138727317] ==> Track [0] posiz: 0.703310
DEBUG [1688015524.138800409] ==> Track [0] posiz: 0.060906
DEBUG [1688015524.138952101] ==> Track [0] posiz: 0.084803
DEBUG [1688015524.180213129] Received TrackArray data...
DEBUG [1688015524.180408803] ==> Number of Tracks: 1
DEBUG [1688015524.180643614] ==> Track [0] kid: 3
DEBUG [1688015524.180803593] ==> Track [0] posiz: 0.703310
DEBUG [1688015524.180909451] ==> Track [0] posiz: 0.060906
DEBUG [1688015524.181097401] ==> Track [0] posiz: 0.084803
DEBUG [1688015524.208079791] Received TrackArray data...
DEBUG [1688015524.208279591] ==> Number of Tracks: 1
DEBUG [1688015524.230086122] ==> Track [0] kid: 3
DEBUG [1688015524.230273803] ==> Track [0] posiz: 0.703310
DEBUG [1688015524.230937254] ==> Track [0] posiz: 0.060906
DEBUG [1688015524.230991933] ==> Track [0] posiz: 0.084803
DEBUG [1688015524.208099351] Received TrackArray data...
DEBUG [1688015524.208452926] ==> Number of Tracks: 1
DEBUG [1688015524.208631413] ==> Track [0] kid: 3
DEBUG [1688015524.208631413] ==> Track [0] posiz: 0.703310
DEBUG [1688015524.208628451] ==> Track [0] posiz: 0.060906
DEBUG [1688015524.208182044] ==> Track [0] posiz: 0.084803
DEBUG [1688015524.330118105] Received TrackArray data...
DEBUG [1688015524.330291191] ==> Number of Tracks: 1
DEBUG [1688015524.330466917] ==> Track [0] kid: 3
DEBUG [1688015524.330629342] ==> Track [0] posiz: 0.703310
DEBUG [1688015524.330670997] ==> Track [0] posiz: 0.060906
DEBUG [1688015524.330724051] ==> Track [0] posiz: 0.084803
DEBUG [1688015524.380237283] Received TrackArray data...
DEBUG [1688015524.380409001] ==> Number of Tracks: 1
DEBUG [1688015524.380671382] ==> Track [0] kid: 3
DEBUG [1688015524.380824723] ==> Track [0] posiz: 0.703310
DEBUG [1688015524.381130315] ==> Track [0] posiz: 0.060906
DEBUG [1688015524.381378283] ==> Track [0] posiz: 0.084803

```

Figura 124. Topic `ti_mmwave_trackarray_to_pcl` publicando los datos de posición del objeto.

El segundo paso es transformar esas coordenadas obtenidas en una **nube de puntos** para poder suscribirse con *Rviz*, **visualizarla** y, más adelante, combinarla con la cámara ZED 2i. Antes de publicar la nube de puntos, se realizará un **publicador simple** para comprobar que el programa recibe las coordenadas de los objetos correctamente y, además, es capaz de publicar un *topic*. En la web de ROS, hay un apartado de **tutoriales** [72], donde enseñan los principios básicos de este programa. Uno de esos tutoriales trata sobre **realizar un publicador y un suscriptor, para C++ y Python** [73]. En este caso se usará el **publicador para C++ que se encuentra en ese tutorial**, al que se le hará referencia con el nombre **“publicador simple”** que publicará **“hello world”**.

En primer lugar, se deben añadir **nuevos #include**, necesarios para el funcionamiento del publicador (Figura 125). Además, se define el **nombre del publicador** (línea 10) fuera de los bucles para definirla como una variable global y que no haya problemas más adelante.

```

1 #include <ros/ros.h>
2 #include <vector>
3
4 #include "std_msgs/String.h" //Para lo de la web
5 #include <ti_mmwave_tracker_rospkg/RadarTrackArray.h>
6 #include <sstream> //Para lo de la web
7
8 #include "ti_mmwave_tracker_to_pcl/ti_mmwave_tracker_to_pcl.h"
9
10 ros::Publisher chatter_pub;
11

```

Figura 125. #Include y definición del publicador simple.

En el bucle *TrackarrayCallback*, es donde estará el **código del publicador simple**. Este código se añade al final del bucle *for* (de la línea 33 a la 38), como se puede ver en la Figura 126.

```

12 void TrackarrayCallback(const ti_mmwave_tracker_rospkg::RadarTrackArray msg)//Mensaje tipo RadarTrackArray, lo guardo en msg
13 {
14     ROS_DEBUG("Received TrackArray data...");
15     uint32_t numTrack = msg.num_tracks; //Convierto a unsigned int 32 el numero total de tracks
16     ROS_DEBUG("> Number of tracks: %d",numTrack);
17
18     std::vector<ti_mmwave_tracker_rospkg::RadarTrackContents> tracks; //Creo un vector dinamico llamado tracks
19     tracks = msg.track; //Al vector dinamico le doy los valores de tracks
20
21     for(unsigned int i = 0; i < tracks.size();i++) //para el resto de parametros, float
22     {
23         uint32_t tid = tracks[i].tid; //Convierto a unsigned int el parametro tid del vector tracks
24         ROS_DEBUG("> Track [%d] tid: %d",i,tid);
25         float posx = tracks[i].posx;
26         ROS_DEBUG("> Track [%d] posx: %f",i,posx);
27         float posy = tracks[i].posy;
28         ROS_DEBUG("> Track [%d] posy: %f",i,posy);
29         float posz = tracks[i].posz;
30         ROS_DEBUG("> Track [%d] posz: %f",i,posz);
31     }
32
33     std_msgs::String msg2;
34     std::stringstream ss;
35     ss << "hello world ";
36     msg2.data = ss.str();
37     ROS_INFO("%s", msg2.data.c_str());
38     chatter_pub.publish(msg2);
39 }

```

Figura 126. Bucle TrackarrayCallback para añadir el publicador simple.

Por último, al igual que en el suscriptor, hay que añadir una línea de código antes del `ros::spin()`. En este caso se añade una **advertencia de publicación** (línea 71), es decir, advierte a todos los nodos que están suscritos a ese *topic* que se van a empezar a publicar datos.

```

69 ros::Rate rate(100); //Hz
70 ros::Subscriber sub = nh.subscribe("/ti_mmwave/radar_trackarray", 10, TrackarrayCallback);
71 chatter_pub = nh.advertise<std_msgs::String>("chatter", 1000);
72 ros::spin();
73
74 return EXIT_SUCCESS;
75 }
76

```

Figura 127. Advertencia del publicador simple.

Ahora, se lanza el nodo y se comprueba que, además de recibir los datos, publica “hello world”. En la Figura 128 se puede ver que **la recepción de datos no ha cambiado** y que se produce el mensaje informativo de que se ha publicado algo. Para comprobar que **publica correctamente** basta con ejecutar el comando “`rostopic echo /chatter`” (Figura 129) en otro terminal.

```

DEBUG [1681295054.952890001]: Received TrackArray data...
DEBUG [1681295054.952891045]: > Number of tracks: 1
DEBUG [1681295054.952400558]: > Track [0] tid: 0
DEBUG [1681295054.952400118]: > Track [0] posx: 0.694755
DEBUG [1681295054.952424487]: > Track [0] posy: 0.221217
DEBUG [1681295054.953190421]: > Track [0] posz: 0.000542
[ INFO ] [1681295054.953505092]: hello world
DEBUG [1681295056.002022073]: Received TrackArray data...
DEBUG [1681295056.002280926]: > Number of tracks: 1
DEBUG [1681295056.002360045]: > Track [0] tid: 7
DEBUG [1681295056.002400934]: > Track [0] posx: 0.608034
DEBUG [1681295056.002710890]: > Track [0] posy: 0.137925
DEBUG [1681295056.003020043]: > Track [0] posz: -0.036497
[ INFO ] [1681295056.003542779]: hello world
DEBUG [1681295056.051806992]: Received TrackArray data...
DEBUG [1681295056.052261059]: > Number of tracks: 1
DEBUG [1681295056.053020523]: > Track [0] tid: 7
DEBUG [1681295056.053335128]: > Track [0] posx: 0.624394
DEBUG [1681295056.054036378]: > Track [0] posy: 0.142637
DEBUG [1681295056.054070928]: > Track [0] posz: -0.083508
[ INFO ] [1681295056.055315064]: hello world
DEBUG [1681295056.102110307]: Received TrackArray data...
DEBUG [1681295056.102272779]: > Number of tracks: 1
DEBUG [1681295056.102380073]: > Track [0] tid: 7
DEBUG [1681295056.102531920]: > Track [0] posx: 0.607754
DEBUG [1681295056.102930091]: > Track [0] posy: 0.147358
DEBUG [1681295056.103400379]: > Track [0] posz: -0.086525
[ INFO ] [1681295056.103632878]: hello world
DEBUG [1681295056.152399752]: Received TrackArray data...
DEBUG [1681295056.152077257]: > Number of tracks: 1
DEBUG [1681295056.152451515]: > Track [0] tid: 7
DEBUG [1681295056.152735113]: > Track [0] posx: 0.607754
DEBUG [1681295056.152790094]: > Track [0] posy: 0.147358
DEBUG [1681295056.152870043]: > Track [0] posz: -0.086525
[ INFO ] [1681295056.152937427]: hello world

```

Figura 128. Recepción de datos del topic deseado.

```

data: "hello world "
---
data: "hello world "
---
data: "hello world "
---
data: "hello world "
---
data: "hello world "
---
data: "hello world "
---
data: "hello world "
---

```

Figura 129. Comprobación de que el publicador simple publica correctamente.

Una vez comprobado que el nodo se suscribe y publica correctamente, el siguiente paso es transformar los datos en el formato de una **nube de puntos**. Para ver como es el mensaje se puede ejecutar el comando “`rostopic echo ti_mmwave/radar_scan_pcl_1`” para ver los datos que publica el *topic* que genera la nube de puntos del radar. En la Figura 130 se puede ver un ejemplo de este tipo de mensaje, donde **únicamente se ha detectado un punto**.

```

---
header:
  seq: 2514
  stamp:
    secs: 1681383470
    nsecs: 232976000
  frame_id: "ti_mmwave_1"
height: 1
width: 1
fields:
- name: "x"
  offset: 0
  datatype: 7
  count: 1
- name: "y"
  offset: 4
  datatype: 7
  count: 1
- name: "z"
  offset: 8
  datatype: 7
  count: 1
- name: "intensity"
  offset: 16
  datatype: 7
  count: 1
- name: "velocity"
  offset: 20
  datatype: 7
  count: 1
is_bigendian: False
point_step: 32
row_step: 32
data: [143, 248, 47, 63, 150, 34, 41, 61, 87, 27, 37, 62, 0, 0, 0, 0, 245, 40, 124, 65, 79, 180, 11, 64, 0, 0, 0, 0, 0, 0, 0, 0]
is_dense: True
---

```

Figura 130. Mensaje de nube de puntos del topic `ti_mmwave/radar_scan_pcl_1`.

El formato de nube de puntos es un **mensaje de ROS** de tipo `sensor_msgs`. En concreto la nube de puntos deseada es el mensaje `sensor_msgs/PointCloud2`. Toda la documentación acerca de los mensajes se encuentra en la web de ROS. Para el mensaje `sensor_msgs/PointCloud2` [74] se puede observar que hay **9 valores** para rellenar (Figura 131).

File: `sensor_msgs/PointCloud2.msg`

Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool is_bigendian # Is this data bigendian?
uint32 point_step # Length of a point in bytes
uint32 row_step # Length of a row in bytes
uint8[] data # Actual point data, size is (row_step*height)

bool is_dense # True if there are no invalid points
```

Compact Message Definition

```
std_msgs/Header header
uint32 height
uint32 width
sensor_msgs/PointField[] fields
bool is_bigendian
uint32 point_step
uint32 row_step
uint8[] data
bool is_dense
```

Figura 131. Mensaje `sensor_msgs/PointCloud2` [74].

7 de los 9 valores son **campos o constantes**, pero 2 de ellos corresponden a **otros tipos de mensajes** diferentes, `pointfield[] fields`, que se corresponde con el mensaje `sensor_msgs/PointField` [75] y `Header header`, que se corresponde con `std_msgs/Header` [76], cuya documentación también está disponible.

File: `std_msgs/Header.msg`

Raw Message Definition

```
# Standard metadata for higher-level stamped data types.
# This is generally used to communicate timestamped data
# in a particular coordinate frame.
#
# sequence ID: consecutively increasing ID
uint32 seq

# Two-integer timestamp that is expressed as:
# * stamp.sec: seconds (stamp_secs) since epoch (in Python the variable is called 'secs')
# * stamp.nsec: nanoseconds since stamp_secs (in Python the variable is called 'nsecs')
# time-handling sugar is provided by the client library
time stamp
#Frame this data is associated with
string frame_id
```

Compact Message Definition

```
uint32 seq
time stamp
string frame_id
```

Figura 132. Mensaje `std_msgs/Header` [76].

File: `sensor_msgs/PointCloud2.msg`

Raw Message Definition

```
# This message holds the description of one point entry in the
# PointCloud2 message format.
uint8 INT8 = 1
uint8 UINT8 = 2
uint8 INT16 = 3
uint8 UINT16 = 4
uint8 INT32 = 5
uint8 UINT32 = 6
uint8 FLOAT32 = 7
uint8 FLOAT64 = 8

string name # Name of field
uint32 offset # Offset from start of point struct
uint8 datatype # Datatype enumeration, see above
uint32 count # How many elements in the field
```

Compact Message Definition

```
uint8 INT8=1
uint8 UINT8=2
uint8 INT16=3
uint8 UINT16=4
uint8 INT32=5
uint8 UINT32=6
uint8 FLOAT32=7
uint8 FLOAT64=8
string name
uint32 offset
uint8 datatype
uint32 count
```

Figura 133. Mensaje `sensor_msgs/PointCloud` [75].

Antes de realizar la nueva programación, se deben **modificar 2 archivos** para que el nodo de ROS pueda reconocer este tipo de mensajes. El primero de ellos es el `CMakeLists.txt`, al que hay que añadirle las **dependencias de los nuevos tipos de mensajes** y el segundo de ellos el `ti_mmwave_tracker_to_pcl.h`, que hay que **incluir el mensaje del tipo `sensor_msgs/PointCloud2`**.

```
1 cmake_minimum_required(VERSION 2.8.3)
2 project(ti_mmwave_tracker_to_pcl)
3
4 find_package(catkin_simple REQUIRED COMPONENTS
5   | ti_mmwave_tracker_rospkg
6   | )
7
8 catkin_python_setup()
9
10 catkin_simple()
11
12 cs_add_executable(${PROJECT_NAME}
13   | src/${PROJECT_NAME}.cpp)
14
15 find_package(catkin REQUIRED COMPONENTS
16   | roscpp
17   | std_msgs
18   | sensor_msgs
19   | )
20
21 cs_install()
22
23 cs_export()
```

Figura 134. Archivo `CMakeLists.txt` del nodo `ti_mmwave_tracker_to_pcl` para publicar una nube de puntos.

```
1 #pragma once
2
3 #include "sensor_msgs/PointCloud2.h"
```

Figura 135. Archivo `ti_mmwave_tracker_to_pcl.h` del nodo `ti_mmwave_tracker_to_pcl` para publicar una nube de puntos.

Este tipo de mensajes es uno de los **más difíciles** que proporciona ROS, así que, en vez de buscar publicar la nube de puntos completa, en primer lugar, se buscará **publicar un punto fijo de coordenadas (3.3; 4.4; 2)** y conseguir publicar un mensaje como el de la Figura 130.

En primer lugar, al igual que en todos los casos anteriores, hay que **incluir los mensajes correspondientes a PointCloud2** (línea 5). Se puede observar que **los #include necesarios para realizar el publicador simple (sstream y std_msgs/String) han sido eliminados** ya que no se va a usar ese publicador.

```

1  #include <ros/ros.h>
2  #include <vector>
3
4  #include <ti_mmwave_tracker_rospkg/RadarTrackArray.h>
5  #include <sensor_msgs/PointCloud2.h>
6  #include "ti_mmwave_tracker_to_pcl/ti_mmwave_tracker_to_pcl.h"

```

Figura 136. #include para publicar un punto.

En el bucle TrackarrayCallback, **se eliminarán las líneas de código del publicador simple** y se escribirá lo necesario para publicar un punto fijo (Figura 137).

```

35  sensor_msgs::PointCloud2 PCL;
36
37  PCL.header.seq = 1; //tiene que ir incrementando
38  PCL.header.stamp = ros::Time::now();
39  PCL.header.frame_id = "ti_mmwave_1";
40
41  PCL.height = 1;
42  PCL.width = numTrack;
43
44  sensor_msgs::PointField field0;
45  field0.name = "x";
46  field0.offset = 0;
47  field0.datatype = 7;
48  field0.count = 4;
49  PCL.fields.push_back(field0);
50
51  sensor_msgs::PointField field1;
52  field1.name = "y";
53  field1.offset = 4;
54  field1.datatype = 7;
55  field1.count = 4;
56  PCL.fields.push_back(field1);
57
58  sensor_msgs::PointField field2;
59  field2.name = "z";
60  field2.offset = 8;
61  field2.datatype = 7;
62  field2.count = 4;
63  PCL.fields.push_back(field2);
64
65  sensor_msgs::PointField field3;
66  field3.name = "velocidad";
67  field3.offset = 16;
68  field3.datatype = 7;
69  field3.count = 4;
70  PCL.fields.push_back(field3);
71
72  std::vector<uint8_t> data(32);
73  float x = (float) 3.3;
74
75  data[0] = FLOAT_TO_UINT(x) & 0x000000FF;
76  data[1] = (FLOAT_TO_UINT(x) & 0x0000FF00) >> 8;
77  data[2] = (FLOAT_TO_UINT(x) & 0x00FF0000) >> 16;
78  data[3] = (FLOAT_TO_UINT(x) & 0xFF000000) >> 24;
79
80  PCL.is_bigendian = false;
81  PCL.point_step = 32;
82  PCL.row_step = 32;
83  PCL.data = data;
84  PCL.is_dense = true;
85  target_publisher.publish(PCL);

```

Figura 137. Código necesario para publicar un punto.

La primera línea (línea 35) **se copia el formato de sensor_msgs/PointCloud2** en la variable PCL. Las líneas 37, 38 y 39 se usan para **definir los valores de la cabecera o mensaje std_msgs/Header** (Figura 138). El número de secuencia (**PCL.header.seq**) **debe ir incrementando** cada vez que se publique un punto. Esto puede ser útil en aplicaciones donde sea vital recibir llevar un registro de que se han recibido todos los paquetes. En este caso no tiene mucha importancia aunque se hará el programa para incrementar ese valor cada vez que se publique una nube de puntos. De momento, como solo se busca publicar un punto para ver que el mensaje es del tipo PointCloud2, sin importar los datos, se dejará con el valor fijo "1". En parámetro **PCL.header.stamp** se **coloca el tiempo actual** y en el **PCL.header.frame_id** se **debe colocar el mismo frame donde publica el topic /ti_mmwave_radar_trackarray**, que en este caso es "ti_mmwave_1".

```

37  PCL.header.seq = 1; //tiene que ir incrementando
38  PCL.header.stamp = ros::Time::now();
39  PCL.header.frame_id = "ti_mmwave_1";
40

```

Figura 138. Rellenar mensaje std_msgs/Header para publicar un punto.

Las líneas 41 y 42 son **parámetros propios del mensaje sensor_msgs/PointCloud2**, **PCL.height** y **PCL.width**. El primero se refiere al **orden de los puntos**, si no hay orden, como es este caso, se deja con un 1. El segundo de los parámetros se refiere **número total de puntos**.

```

40
41   PCL.height = 1;
42   PCL.width = numTrack;
43

```

Figura 139. Rellenar dos de los parámetros del sensor_msgs/PointCloud2.

A continuación, está el campo **PCL.fields**, que es un mensaje de ROS (**sensor_msgs/PointField**). Como hay que rellenar los campos dependiendo de las coordenadas (x, y, z) y de otros valores deseados como, por ejemplo, la velocidad, no basta con hacerlo como hasta ahora. **Para todos los casos se realiza el mismo procedimiento**, por lo que solo se explicará para un caso, la **coordenada x**. En primer lugar, se **copia el formato del mensaje sensor_msgs/PointField en field0** (línea 44). Las 4 siguientes líneas sirven para **darle valores a los constantes** del mensaje. El primero de ellos, **field0.name es el nombre**, en este caso como es la coordenada X, se le llama "x". El **field0.offset hace referencia al número de byte desde el cual empiezan los valores** y **field0.count** es el número de bytes que ocupa. El tipo de datos o **field0.datatype depende del tipo de datos** (el número correspondiente a cada tipo de dato se puede ver en la Figura 133), **en este caso 7**, ya que se tienen que poner los datos en tipo float32. Por último, **se le añade al vector PCL.fields el nuevo vector que se ha formado, field0** (línea 49) mediante la función *push_back*.

```

43
44   sensor_msgs::PointField field0;
45   field0.name = "x";
46   field0.offset = 0;
47   field0.datatype = 7;
48   field0.count = 4;
49   PCL.fields.push_back(field0);

```

Figura 140. Rellenar mensaje sensor_msgs/PointField para publicar un punto.

Ahora toca **rellenar el vector de datos, que será el PCL.data**. Lo primero que hay que hacer es **crear un vector dinámico** para almacenar los datos (línea 71). En este caso los datos serán del tipo `uint8_t`, ya que son los tipos de datos que se necesitan. **El vector dinámico tendrá 32 datos del tipo uint8_t**. En la línea siguiente se define una **coordenada con un valor arbitrario para la x**, en este caso 3.3. Por defecto, al definir un dato *float* se toma como un dato en coma flotante de doble precisión o *double* (tipo de dato `float64`, que no coincide con lo que puede leer *Rviz*). Por este motivo **se le añade "(float)" antes del valor numérico**, para convertir de `float64` a `float32`.

Para rellenar el vector de datos, en primer lugar, **se reinterpreta el valor float como un número del tipo int (de 32 bits) mediante una macro**. Las macros son un **conjunto de comandos que se invocan con una palabra clave, opcionalmente seguidas de parámetros**. Se sustituye esa palabra clave por el código de las macros (lo realiza el compilador). En este código la macro se define al principio del código con la palabra clave **FLOAT_TO_UINT(x)**, donde x es el parámetro (Figura 142).

El formato de datos que se necesitan para introducir en los datos es de 8 bits, es decir, 2 bytes. Para ello es conveniente **utilizar máscaras**. Para el primero de los datos, `data[0]`, **se realiza el AND lógico entre el dato y una máscara**, en este caso `0x000000FF` (línea 75). Esto quiere decir que el resultado, los 6 primeros bytes serán 0 y los dos últimos bytes serán los dos últimos bytes del dato. Esto se puede explicar mejor de forma gráfica:

$$\begin{array}{r} 0100\ 0000\ 0101\ 0011\ 0011\ 0011\ 0011\ 0011 \\ \& \underline{0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 1111\ 1111} \\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0011\ 0011 \end{array}$$

El tipo de dato, al ser de 8 bits, **se usarán los 8 últimos bits del número obtenido**, en este caso 0011 0011, que en decimal es 51. Esto funciona bien para el primero de los datos ya que se buscan los 2 últimos bytes, **pero con los siguientes datos no se buscan esos bytes**. Por ejemplo, para el segundo dato (data[1]) (línea 76):

$$\begin{array}{r} 0100\ 0000\ 0101\ 0011\ 0011\ 0011\ 0011\ 0011 \\ \& \underline{0000\ 0000\ 0000\ 0000\ 1111\ 1111\ 0000\ 0000} \\ 0000\ 0000\ 0000\ 0000\ 0011\ 0011\ 0000\ 0000 \end{array}$$

Si no se hiciera nada, el dato que recogería sería el 0000 0000, pero el dato que necesitamos es el de los dos bytes siguientes, 0011 0011 (que casualmente coinciden con el data[0], pero no tiene porqué). **Para que recoja los bits deseados hay que usar operadores de desplazamiento de bits**. Para este caso en concreto, será necesario desplazar los bits **8 posiciones a la derecha**, para obtener 0000 0000 0000 0000 0000 0000 0011 0011, y que al introducir ese dato en el vector se obtenga 0011 0011.

Para el **tercer dato** (data[2]), **la máscara es 0x00FF0000** y habría que mover los bits **16 posiciones** a la derecha.

$$\begin{array}{r} 0100\ 0000\ 0101\ 0011\ 0011\ 0011\ 0011\ 0011 \\ \& \underline{0000\ 0000\ 1111\ 1111\ 0000\ 0000\ 0000\ 0000} \\ 0000\ 0000\ 0101\ 0011\ 0000\ 0000\ 0000\ 0000 \end{array}$$

Para el **cuarto y último dato** (data[3]), **la máscara es 0xFF000000** y habría que mover los bits **24 posiciones** a la derecha.

$$\begin{array}{r} 0100\ 0000\ 0101\ 0011\ 0011\ 0011\ 0011\ 0011 \\ \& \underline{1111\ 1111\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000} \\ 0100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000 \end{array}$$

Es importante el **orden de la colocación de los bytes**, es decir, si el orden de los datos es **Big-Endian** o **Little-Endian**. Esto designa el formato en el que se deben **ordenar o almacenar los bytes**. **Big-Endian es el orden intuitivo, ordenando los bytes de derecha a izquierda**. El orden de los datos **Little-Endian es al revés, el byte menos significativo va a la izquierda**. Por ejemplo, para el número 0x3D2C1B0A, en el sistema **Big-Endian** se ordenaría como {3D, 2C, 1B, 0A} y en el sistema **Little-Endian** {0A, 1B, 2C, 3D}. Se va a seguir el orden en el que se ordena la nube de puntos que proporciona el radar, que en este caso es **Little-Endian**. Por ese motivo, a la hora de ordenar los bytes en el vector data, **el byte menos significativo es el primer valor**.

Para el resto de los datos (coordenadas y velocidad) el **procedimiento es idéntico** al de la coordenada X.

```

72     std::vector<uint8_t> data(32);
73     float x = (float) 3.3;
74
75     data[0] = FLOAT_TO_UINT(x) & 0X000000FF;
76     data[1] = (FLOAT_TO_UINT(x) & 0X0000FF00) >> 8;
77     data[2] = (FLOAT_TO_UINT(x) & 0X00FF0000) >> 16;
78     data[3] = (FLOAT_TO_UINT(x) & 0XFF000000) >> 24;
79

```

Figura 141. Obtención de los datos para completar el vector de datos de una coordenada.

```

7
8     #define FLOAT_TO_UINT(X) *((reinterpret_cast<int*>(&X))
9

```


de tamaño 0.01 metros, casi inapreciable. **Aumentando el tamaño y cambiando la forma a un cubo se puede ver perfectamente la nube de puntos del radar y el target fijo que publica el nuevo nodo (Figura 148).**

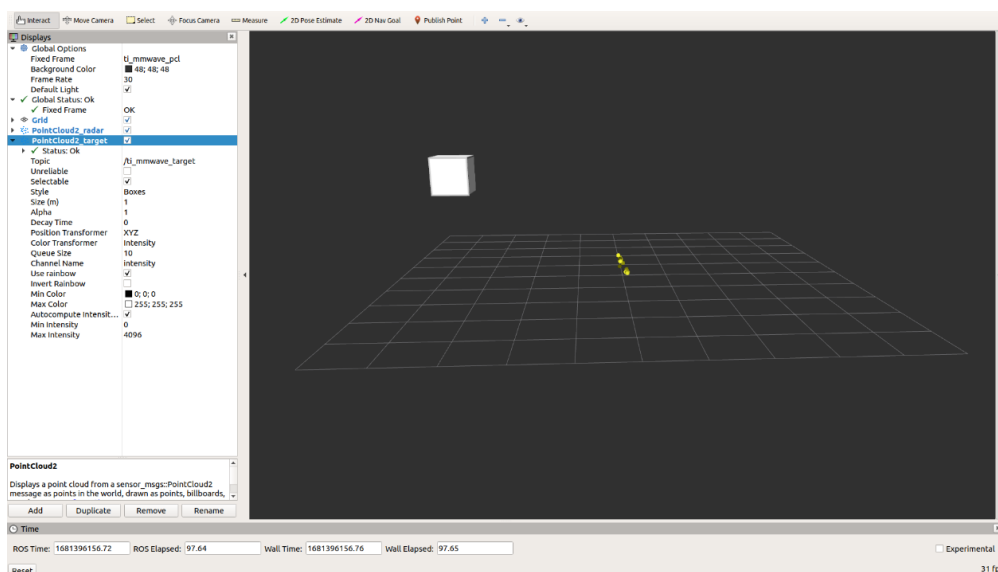


Figura 148. Visualización target y nube de puntos en Rviz.

Cabe destacar, que al configurar todos los parámetros manualmente excepto PCL.width **dará fallo cuando el radar detecte 2 objetos diferentes**. No hay que olvidar que a pesar de que esto sea un punto fijo se publica cuando recibe los datos de targets del radar, así que, si el radar detecta 2 objetos, PCL.width será 2 y hará que no coincidan los valores y de fallo. Rviz detecta que solo han llegado 32 datos cuando en realidad deberían llegar 64, simplemente **desecha el punto y no lo muestra en pantalla**. Este error se puede ver en la parte izquierda de la Figura 149.

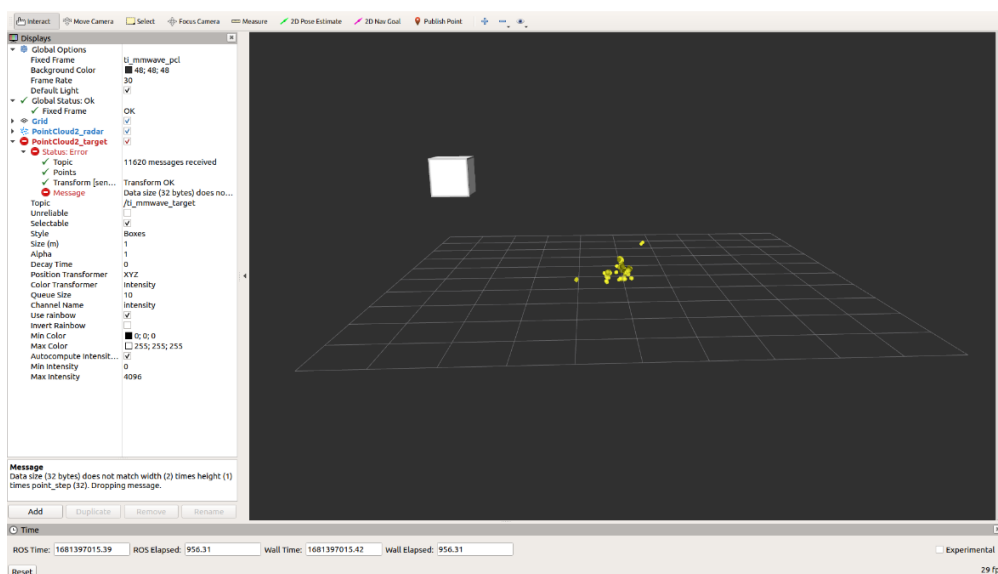


Figura 149. Error al detectar 2 objetos a la vez.

El siguiente y último paso es modificar el código de tal forma que se **muestren todos los targets que detecta el radar** en vez de uno fijo. Lo fundamental del código ya está realizado al publicar un punto fijo, ahora hay que **adecuarlo para poder publicar todos los puntos**. Para esto hay que **introducir el código desde las líneas 72 a la 90 (Figura 146) dentro del bucle for**, para que si llegaran más de un punto se pueda mostrar en Rviz. Además, para lograrlo, hay que **redimensionar el vector dinámico** cada vez que reciba datos (línea 35).

Como cada dato son 32 bytes, el tamaño total del vector será 32 bytes multiplicado por el número de puntos (variable NumTrack). Es importante también **definir el vector dinámico fuera del bucle for** (línea 22), ya que si no dará error al compilar. A la hora de recibir los datos y rellenar el vector, se debe **cambiar los valores fijos de las coordenadas por los valores de los puntos** que se recogen y que son los que se quieren representar. Además, hay que cambiar la **posición donde se deben colocar los datos** dentro del vector, multiplicando al número de posición por $(32 \cdot i)$. De esta forma, si llega un solo punto, por ejemplo, la primera posición será $0 \cdot (32 \cdot 0) = 0$. Si llegan dos puntos, la primera posición del segundo punto será $0 \cdot (32 \cdot 1) = 32$, es decir, la primera posición del segundo punto. Todos estos cambios se pueden ver en la Figura 150.

```

21
22 std::vector<uint8_t> data(32);
23
24 for(unsigned int i = 0; i < tracks.size();i++) //para el resto de parametros, float
25 {
26     uint32_t tid = tracks[i].tid; //Convierto a unsigned int el parametro tid del vector tracks
27     ROS_DEBUG(">> Track [%d] tid: %d",i,tid);
28     float posx = tracks[i].posx;
29     ROS_DEBUG(">> Track [%d] posx: %f",i,posx);
30     float posy = tracks[i].posy;
31     ROS_DEBUG(">> Track [%d] posy: %f",i,posy);
32     float posz = tracks[i].posz;
33     ROS_DEBUG(">> Track [%d] posz: %f",i,posz);
34
35     data.resize(32*numTrack);
36
37     float x = (float) posx;
38     float y = (float) posy;
39     float z = (float) posz;
40
41     data[0+(32*i)] = FLOAT_TO_UINT(x) & 0x000000FF;
42     data[1+(32*i)] = (FLOAT_TO_UINT(x) & 0x0000FF00) >> 8;
43     data[2+(32*i)] = (FLOAT_TO_UINT(x) & 0x00FF0000) >> 16;
44     data[3+(32*i)] = (FLOAT_TO_UINT(x) & 0xFF000000) >> 24;
45
46     data[4+(32*i)] = FLOAT_TO_UINT(y) & 0x000000FF;
47     data[5+(32*i)] = (FLOAT_TO_UINT(y) & 0x0000FF00) >> 8;
48     data[6+(32*i)] = (FLOAT_TO_UINT(y) & 0x00FF0000) >> 16;
49     data[7+(32*i)] = (FLOAT_TO_UINT(y) & 0xFF000000) >> 24;
50
51     data[8+(32*i)] = FLOAT_TO_UINT(z) & 0x000000FF;
52     data[9+(32*i)] = (FLOAT_TO_UINT(z) & 0x0000FF00) >> 8;
53     data[10+(32*i)] = (FLOAT_TO_UINT(z) & 0x00FF0000) >> 16;
54     data[11+(32*i)] = (FLOAT_TO_UINT(z) & 0xFF000000) >> 24;
55 }

```

Figura 150. Toma de datos y construcción del vector de datos.

Por último, hay que **cambiar el valor de PCL.row_step** (línea 100 de la Figura 151), que debe ser el número total de bytes, es decir, el **tamaño total del vector de datos**.

```

97
98 PCL.is_bigendian = false;
99 PCL.point_step = 32;
100 PCL.row_step = 32*numTrack;
101 PCL.data = data;
102 PCL.is_dense = true;
103 target_publisher.publish(PCL);

```

Figura 151. Ajustar los valores del mensaje para publicar todos los puntos.

Se puede comprobar en la Figura 152 que **el target sigue correctamente la nube de puntos**. En este caso se ha modificado el tamaño y la forma del *target* para que se pueda ver mejor que coincide con la nube de puntos que publica el radar. **Para más de un target, el programa seguiría funcionando perfectamente** (Figura 153).

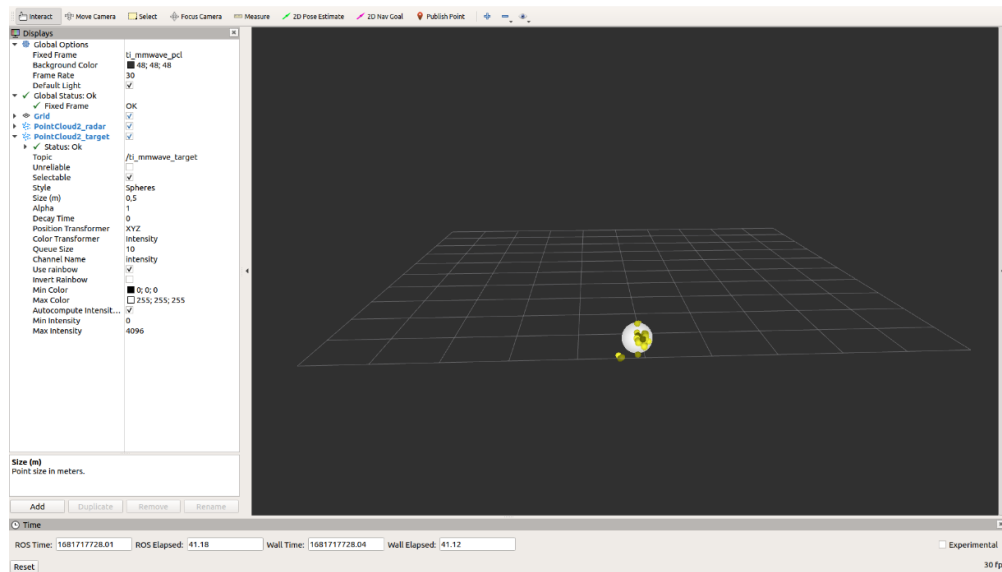


Figura 152. Target siguiendo la nube de puntos.

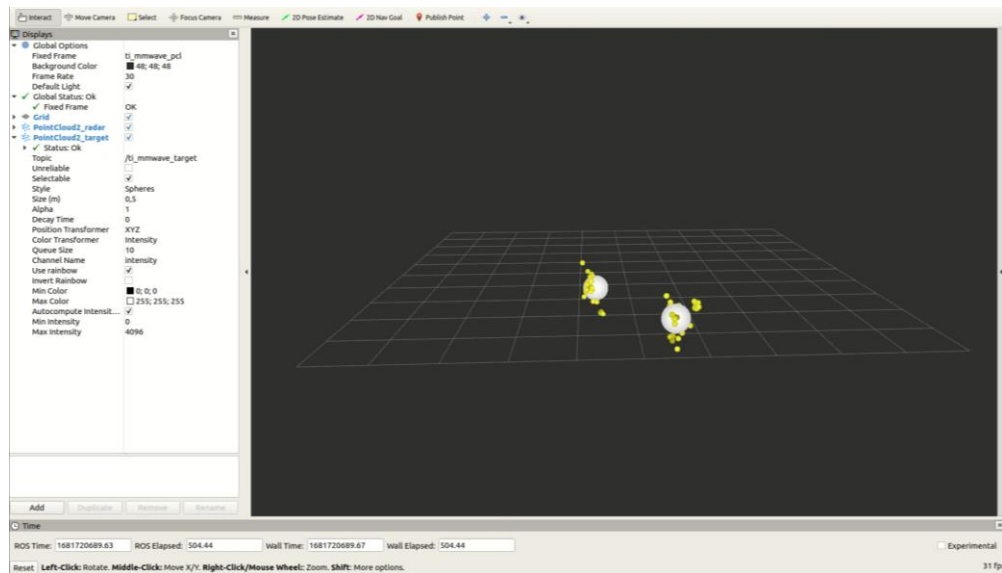


Figura 153. Dos targets siguiendo las nubes de puntos.

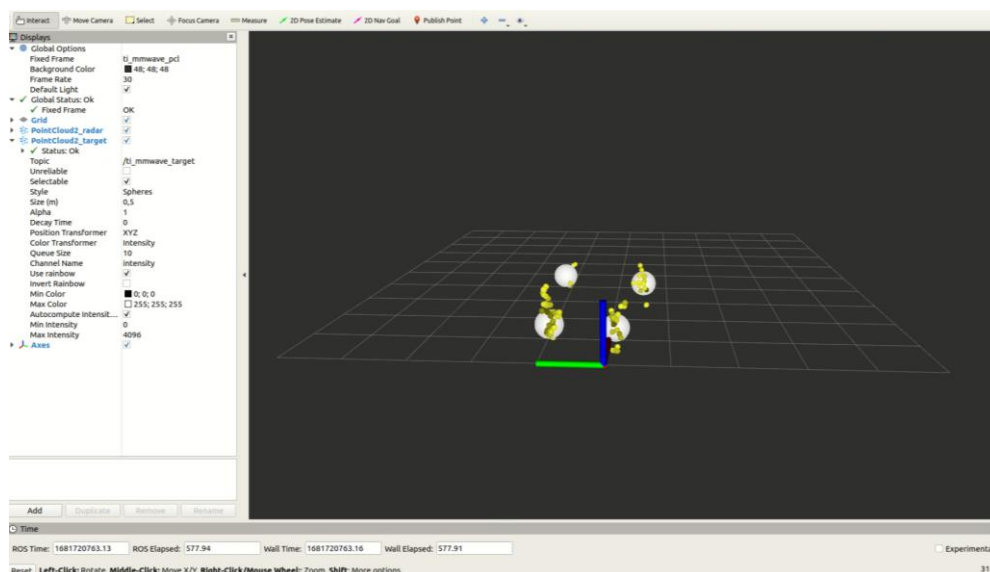


Figura 154. Varios targets siguiendo las nubes de puntos (con ejes de coordenadas).

El orden de ejecución de los comandos, cada uno en **diferente terminal** es el siguiente (El comando 4 se puede ejecutar cuando se quiera, pero siempre después de ejecutar el primero de todos):

1. `roslaunch zed_wrapper zed2i.launch`
2. `roslaunch ti_mmwave_tracker_rospkg AOP_3d_Tracking.launch`
3. `roslaunch ti_mmwave_tracker_to_pcl ti_mmwave_tracker_to_pcl.launch verbose:=true`
4. `rosservice call /zed2i/zed_node/start_object_detection 2 50 6 true false true true true true true true` (Si se quiere activar la detección de personas)

6.2.4.c Publicación de la velocidad del centroide como una etiqueta

Como el topic `/ti_mmwave/radar_trackarray` **publica también la velocidad** en 3 dimensiones, se puede poner una **etiqueta al lado del centroide indicando su velocidad**. Al igual que para publicar la nube de puntos, se irá poco a poco, verificando que funciona correctamente. En primer lugar, se buscará publicar una etiqueta de **texto con la velocidad del punto en una posición fija**, y en segundo lugar se buscará publicar **tantas etiquetas de velocidades como targets haya** (cerca de la posición de los targets).

Hay **múltiples objetos o markers** que se pueden publicar y visualizar en Rviz [77]. De todos los que hay, el *marker* que se va a publicar es de **tipo texto**. Para publicar un *marker*, hay que publicar un tipo de mensaje en específico, en concreto hay que publicar un mensaje del tipo `visualization_msgs/Marker`, que es un mensaje de ROS. El formato de este mensaje se puede ver en la Figura 155.

File: `visualization_msgs/Marker.msg`

Raw Message Definition

```
# See http://www.ros.org/wiki/rviz/DisplayTypes/Marker and http://www.ros.org/wiki/rviz/Tutorials/Markers
%3A%20Basic%20Shapes for more information on using this message with rviz

uint8 ARROW=0
uint8 CUBE=1
uint8 SPHERE=2
uint8 CYLINDER=3
uint8 LINE_STRIP=4
uint8 LINE_LIST=5
uint8 CUBE_LIST=6
uint8 SPHERE_LIST=7
uint8 POINTS=8
uint8 TEXT_VIEW_FACING=9
uint8 MESH_RESOURCE=10
uint8 TRIANGLE_LIST=11

uint8 ADD=0
uint8 MODIFY=0
uint8 DELETE=2
uint8 DELETEALL=3

Header header           # header for time/frame information
string ns               # Namespace to place this object in... used in conjunction with id to create a unique name for the object
int32 id                # object ID useful in conjunction with the namespace for manipulating and deleting the object later
int32 type              # Type of object
int32 action            # 0 add/modify an object, 1 (deprecated), 2 deletes an object, 3 deletes all objects
geometry_msgs/Pose pose # Pose of the object
geometry_msgs/Vector3 scale # Scale of the object 1,1,1 means default (usually 1 meter square)
std_msgs/ColorRGBA color # Color [0.0-1.0]
duration lifetime      # How long the object should last before being automatically deleted. 0 means forever
bool frame_locked      # If this marker should be frame-locked, i.e. retransformed into its frame every timestep

#Only used if the type specified has some use for them (eg. POINTS, LINE_STRIP, ...)
geometry_msgs/Point[] points
#Only used if the type specified has some use for them (eg. POINTS, LINE_STRIP, ...)
#number of colors must either be 0 or equal to the number of points
#NOTE: alpha is not yet used
std_msgs/ColorRGBA[] colors

# NOTE: only used for text markers
string text

# NOTE: only used for MESH_RESOURCE markers
string mesh_resource
bool mesh_use_embedded_materials
```

Figura 155. Mensaje `visualization_msgs/Marker` [78].

Este mensaje es bastante largo y contiene **múltiples mensajes de ROS** en su interior. Uno de ellos es el mensaje `std_msgs/Header`, que ya se ha utilizado en el apartado anterior y cuyo formato se puede ver en la Figura 132. El siguiente mensaje es `geometry_msgs/Pose` (Figura 156), que a su vez **contiene dos mensajes en su interior**, `geometry_msgs/Point` (Figura 157) y `geometry_msgs/Quaternion` (Figura 158). Estos se utilizan para **definir el punto y la orientación** del marker.

File: `geometry_msgs/Pose.msg`

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Figura 156. Mensaje `geometry_msgs/Pose` [79].

File: `geometry_msgs/Point.msg`

Raw Message Definition

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Figura 157. Mensaje `geometry_msgs/Point` [80].

File: `geometry_msgs/Quaternion.msg`

Raw Message Definition

```
# This represents an orientation in free space in quaternion form.
float64 x
float64 y
float64 z
float64 w
```

Figura 158. Mensaje `geometry_msgs/Quaternion` [81].

El mensaje `geometry_msgs/Vector3` se utiliza para **modificar la escala** o el tamaño del marker, cuyo formato se puede ver en la Figura 159.

File: `geometry_msgs/Vector3.msg`

Raw Message Definition

```
# This represents a vector in free space.
# It is only meant to represent a direction. Therefore, it does not
# make sense to apply a translation to it (e.g., when applying a
# generic rigid transformation to a Vector3, tf2 will only apply the
# rotation). If you want your data to be translatable too, use the
# geometry_msgs/Point message instead.

float64 x
float64 y
float64 z
```

Figura 159. Mensaje `geometry_msgs/Vector3` [82].

El último de los mensajes obligatorios es `std_msgs/ColorRGBA`, que se usa para **darle color** al marker. El formato se puede ver en la Figura 160.

File: `std_msgs/ColorRGBA.msg`

Raw Message Definition

```
float32 r
float32 g
float32 b
float32 a
```

Figura 160. Mensaje `std_msgs/ColorRGBA` [83].

Los 5 últimos valores del mensaje `visualization_msgs/Marker` no son obligatorias para todos los *markers*, se deberán rellenar los valores **dependiendo el marker** que se quiera publicar. En este caso, al querer publicar un texto, **sólo se rellenará el valor "string text"** con el texto que se quiera mostrar en pantalla.

Ahora, hay que crear un **nuevo publicador** para publicar estos *markers*, que se hará en el archivo del apartado anterior. En primer lugar, hay que **incluir los mensajes de ROS** que se van a usar (desde la línea 9 hasta la 14) y las **librerías** que serán necesarias más adelante (líneas 3, 4 y 5).

```
1 #include <ros/ros.h>
2 #include <vector>
3 #include <string>
4 #include <iostream>
5 #include <sstream>
6
7 #include <ti_mmwave_tracker_rospkg/RadarTrackArray.h>
8 #include <sensor_msgs/PointCloud2.h>
9 #include <visualization_msgs/Marker.h>
10 #include <geometry_msgs/Pose.h>
11 #include <geometry_msgs/Vector3.h>
12 #include <std_msgs/ColorRGBA.h>
13 #include <geometry_msgs/Point.h>
14 #include <std_msgs/ColorRGBA.h>
15 #include "ti_mmwave_tracker_to_pcl/ti_mmwave_tracker_to_pcl.h"
16
```

Figura 161. `#include` necesarios para publicar un marker de texto.

Justo después hay que **definir el nuevo publicador**, ya que se quiere publicar el *marker* sin dejar de publicar los objetos. Esto se hace en la línea 20 de la Figura 162.

```
18
19 ros::Publisher target_publisher;
20 ros::Publisher marker_publisher;
21
```

Figura 162. Definición del nuevo publicador de markers.

En la Figura 163 se puede ver el código necesario para **crear el mensaje que publica un texto con un número fijo**. Para rellenar el mensaje se sigue el mismo procedimiento que para la nube de puntos, se copia el formato del mensaje en la variable Text y se rellenan sus valores.

```

135 visualization_msgs::Marker Text;
136
137 Text.header.seq = 1; //tiene que ir incrementando
138 Text.header.stamp = ros::Time::now();
139 Text.header.frame_id = "ti_mmwave_1";
140
141 Text.ns = "name";
142 Text.id = 0;
143 Text.type = 9; //Texto
144 Text.action = 0; //Add or modify
145
146 geometry_msgs::Pose pose;
147
148 geometry_msgs::Point position;
149 position.x = 0;
150 position.y = 0;
151 position.z = 0;
152 pose.position = position;
153
154 geometry_msgs::Quaternion orientation;
155 orientation.x = 0;
156 orientation.y = 0;
157 orientation.z = 0;
158 orientation.w = 0;
159 pose.orientation = orientation;
160
161 Text.pose = pose;
162
163 geometry_msgs::Vector3 scale;
164 scale.x = 1;
165 scale.y = 1;
166 scale.z = 1;
167 Text.scale = scale;
168
169 std_msgs::ColorRGBA color;
170 color.r = 1;
171 color.g = 0;
172 color.b = 0;
173 color.a = 1;
174 Text.color = color;
175
176 //Text.lifetime = 0;
177 Text.frame_locked = false;
178
179 //float to string converter
180 float vel = (float) 4.25894;
181 std::stringstream ss;
182 ss << vel;
183 std::string velo = ss.str();
184 velo=std::to_string(vel);
185 Text.text = velo;

```

Figura 163. Código para rellenar el mensaje visualization_msgs/Marker.

El mensaje visualization_msgs/Marker se ha copiado en Text. La cabecera se rellena de la misma forma que en el apartado anterior (Figura 138). En la Figura 164 se puede ver como **se rellenan los primeros valores del mensaje** visualization_msgs/Marker. **El nombre y el ID se ha dejado fijos** debido a que solo se quiere publicar un texto. **El tipo es 9** ya que es el valor que corresponde al *marker* del tipo texto y **la acción es 0** que corresponde con añadir o modificar el *marker* (en este caso el texto)

```

141 Text.ns = "name";
142 Text.id = 0;
143 Text.type = 9; //Texto
144 Text.action = 0; //Add or modify

```

Figura 164. Rellenar algunos valores del mensaje visualization_msgs/Marker.

A continuación, se define la **posición y orientación del texto** (Figura 165). Por ahora se colocará en el **origen**, es decir, las 3 coordenadas de posición a 0, y se pondrá también **sin orientación**, ya que el texto siempre se orienta automáticamente a la vista de Rviz.

```

146 geometry_msgs::Pose pose;
147
148 geometry_msgs::Point position;
149 position.x = 0;
150 position.y = 0;
151 position.z = 0;
152 pose.position = position;
153
154 geometry_msgs::Quaternion orientation;
155 orientation.x = 0;
156 orientation.y = 0;
157 orientation.z = 0;
158 orientation.w = 0;
159 pose.orientation = orientation;
160
161 Text.pose = pose;

```

Figura 165. Definición de la posición para un texto fijo en el origen.

En la Figura 166, se puede ver la parte del código en la que se define la **escala y el color**. La escala se va a dejar en 1 para los 3 ejes, es decir, **tamaño normal**, que suele ser 1 m². El color, se dejará como **rojo sin transparencia** ($a = 1$).

```

163 geometry_msgs::Vector3 scale;
164 scale.x = 1;
165 scale.y = 1;
166 scale.z = 1;
167 Text.scale = scale;
168
169 std_msgs::ColorRGBA color;
170 color.r = 1;
171 color.g = 0;
172 color.b = 0;
173 color.a = 1;
174 Text.color = color;
175

```

Figura 166. Escala y color de un texto fijo en el origen.

El *lifetime* o tiempo de vida del *marker* se va a dejar a 0, es decir, el **marker se queda para siempre**, solo puede ser modificado (o eliminado si se cambiara el Text.action de la Figura 164). En este caso se ha comentado ya que **por defecto está a 0**. La segunda parte de la Figura 167 se usa para **convertir un número float en un string** o cadena de caracteres (desde la línea 181 hasta la 184). La línea 180 define el valor de velocidad fijo que se va a mostrar y en la línea 185 se añade al mensaje la velocidad en forma de cadena de caracteres. El parámetro **frame_locked** se usa para variar la posición del *marker* si varían las posiciones de los sistemas de coordenadas. Si la posición del *marker* no se modifica y este parámetro está en false, aunque cambien las coordenadas, el **marker se quedará en la posición en la que estaba**, pero si está en true, el **marker se moverá en referencia al nuevo sistema de coordenadas**. En este caso, da igual que valor ponerle, ya que **no van a cambiar los sistemas de coordenadas**.

```

175
176 //Text.lifetime = 0;
177 Text.frame_locked = false;
178
179 //float to string converter
180 float vel = (float) 4.25894;
181 std::stringstream ss;
182 ss << vel;
183 std::string velo = ss.str();
184 velo=std::to_string(vel);
185 Text.text = velo;

```

Figura 167. Formato que se le da al texto fijo que se verá en Rviz.

Por último, **se publica el mensaje** (Figura 168) y se define el publicador justo debajo del publicador de la nube de puntos (línea 221 de la Figura 169).

```
186  
187     marker_publisher.publish(Text);  
188 }  
189
```

Figura 168. Publicador del marker texto para un texto fijo.

```
217  
218     ros::Rate rate(100); //Hz  
219     ros::Subscriber sub = nh.subscribe("/ti_mmwave/radar_trackarray", 10, TrackarrayCallback);  
220     target_publisher = nh.advertise<sensor_msgs::PointCloud2>("ti_mmwave_target", 1000);  
221     marker_publisher = nh.advertise<visualization_msgs::Marker>("ti_mmwave_marker", 1000);  
222     ros::spin();  
223  
224     return EXIT_SUCCESS;  
225 }
```

Figura 169. Advertencia de publicación de un marker.

En el terminal mediante el comando “*rostopic echo /ti_mmwave_marker*” se puede observar el formato del mensaje del *marker* (Figura 170).

```
header:  
  seq: 159  
  stamp:  
    secs: 1681909230  
    nsecs: 419628848  
  frame_id: "ti_mmwave_1"  
ns: "name"  
id: 0  
type: 9  
action: 0  
pose:  
  position:  
    x: 0.0  
    y: 0.0  
    z: 0.0  
  orientation:  
    x: 0.0  
    y: 0.0  
    z: 0.0  
    w: 0.0  
scale:  
  x: 1.0  
  y: 1.0  
  z: 1.0  
color:  
  r: 1.0  
  g: 0.0  
  b: 0.0  
  a: 1.0  
lifetime:  
  secs: 0  
  nsecs: 0  
frame_locked: False  
points: []  
colors: []  
text: "4.258940"  
mesh_resource: ""  
mesh_use_embedded_materials: False
```

Figura 170. Formato del mensaje del marker.

Se puede comprobar suscribiéndose al *topic* con Rviz que el texto se publica correctamente y aparece en el origen de coordenadas (Figura 171).

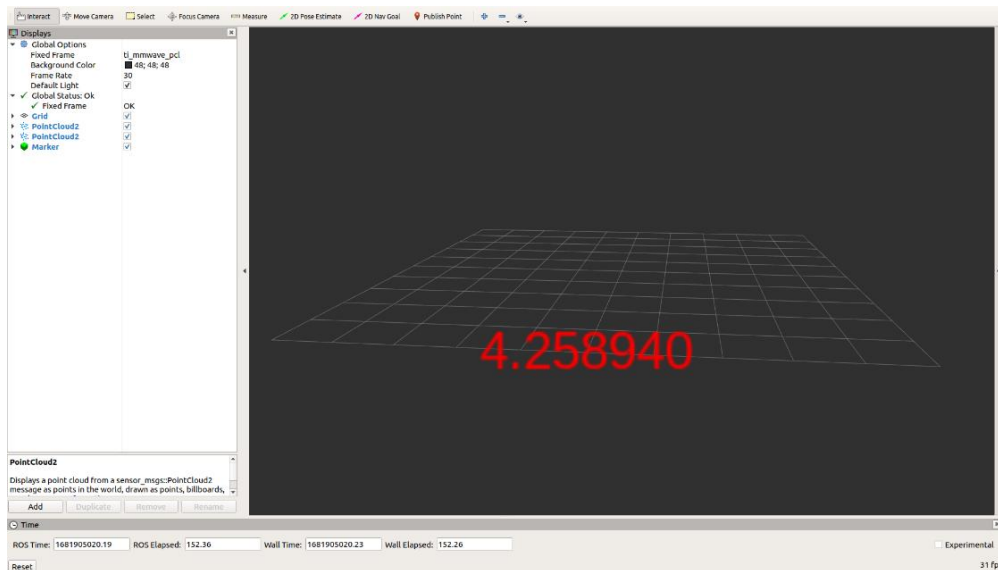


Figura 171. Visualización en Rviz de un mensaje fijo.

El segundo paso es **recoger los datos de la velocidad de cada target y hacer que aparezca a su lado**. En este caso **no se pueden almacenar datos** de texto y publicarlos todos a la vez (como se hacía con los targets con el vector dinámico). Hay que procesar un *target* y **antes de que se procese el siguiente publicar el texto**. Para ello basta con **añadir el código de la Figura 163 y el publicador (Figura 168) al final del bucle for**, pero para que funcione correctamente hay que hacer algunos cambios y, lo más importante, obtener la velocidad.

La velocidad se obtiene de la misma forma que se obtienen los datos de posición (desde la línea 50 a la 55). Como el topic devuelve los datos del vector de velocidades, se puede calcular su módulo como se hace en la línea 57, y se muestra en el terminal mediante ROS_DEBUG().

```

50     float velx = tracks[i].velx;
51     ROS_DEBUG(">> Track [%d] velx: %f",i,velx);
52     float vely = tracks[i].vely;
53     ROS_DEBUG(">> Track [%d] vely: %f",i,vely);
54     float velz = tracks[i].velz;
55     ROS_DEBUG(">> Track [%d] velz: %f",i,velz);
56
57     float velocidad = sqrt((velx*velx)+(vely*vely)+(velz*velz));
58     ROS_DEBUG(">> Track [%d] velocidad (m/s): %f",i,velocidad);
59

```

Figura 172. Obtención y cálculo de la velocidad de cada objeto.

Los cambios que hay que hacer en el código se ven en la Figura 173. Se ha hecho un **cambio en la escala**, para que no se vea tan grande el texto. El tiempo de vida se ha descomentado ya que se quiere que **la etiqueta de velocidad desaparezca a los 0.2 segundos**. Por último, hay que **modificar el código que convierte un float a una cadena de caracteres**, para que en vez del número fijo (cuya definición se ha eliminado) convierta el valor de la velocidad del objeto.


```

112 geometry_msgs::Vector3 scale;
113 scale.x = 0.5;
114 scale.y = 0.5;
115 scale.z = 0.5;
116 Text.scale = scale;
117
118 std_msgs::ColorRGBA color;
119 color.r = 1;
120 color.g = 0;
121 color.b = 0;
122 color.a = 1;
123 Text.color = color;
124
125 Text.lifetime = ros::Duration(0.2);
126 Text.frame_locked = false;
127
128 //float to string converter
129 std::stringstream ss;
130 ss << v;
131 std::string vel = ss.str();
132 vel=std::to_string(v);
133 Text.text = vel;

```

Figura 173. Cambios para publicar la velocidad por cada punto.

Se puede ver en la Figura 174 que la velocidad sigue a los targets y varía según la velocidad de estos.

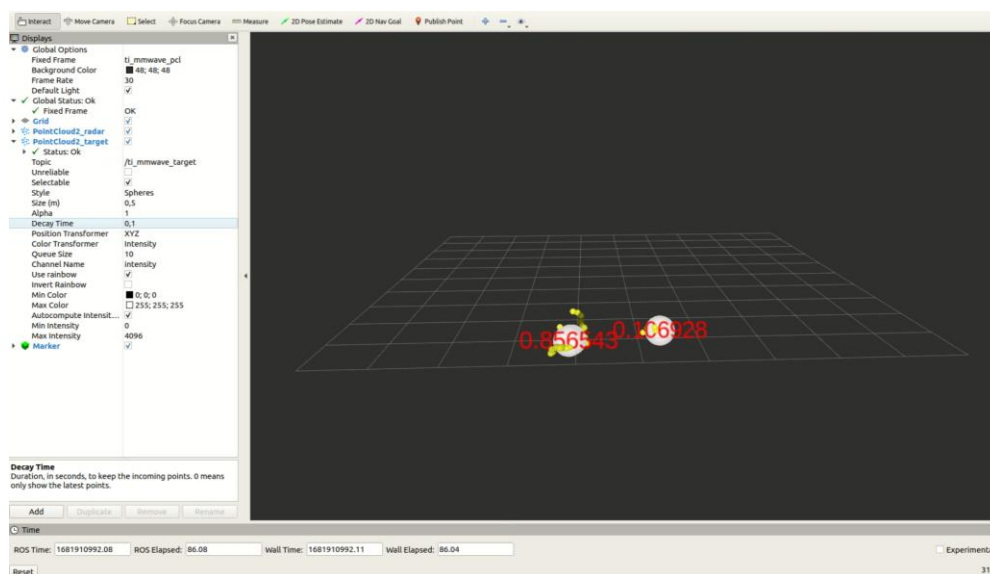


Figura 174. Visualización en Rviz de 2 puntos con su velocidad.

6.2.5 3D_People_Counting y ZED 2i

En este apartado se mostrarán a la vez la nube de puntos de la cámara ZED 2i, la nube de puntos del radar y los targets con su velocidad correspondiente. Antes de eso, se modificará un poco el código que muestra los targets y la velocidad para que quede estéticamente mejor.

Antes de modificar el código, se ha modificado también el comando de configuración *stateParam*, en concreto el cuarto parámetro, *<static2freeThre>*, para hacer que los objetos que tras moverse se vuelven estáticos permanezcan durante más tiempo. El valor de este comando se ha cambiado de 20 a 150 frames.

En primer lugar, se mejorará la nube de puntos de targets. El radar muestra los objetos y hace tracking o seguimiento de ellos, pero hasta ahora solo se ha mostrado los objetos. Una forma de ver los objetos y hacia dónde se mueven es mediante colores ya que Rviz da la posibilidad de que se muestren distintos colores en función de un parámetro (que se puede elegir) de la nube de puntos. El vector de datos de la nube de puntos es de 32 bytes, pero solo se usan 12 de ellos para las coordenadas. Hay 5 campos extras de 4 bytes cada uno para poner otros valores. Estos campos se utilizarán para escribir la velocidad (aunque no se utilice en la nube de puntos) y el identificador. Se realiza de la misma forma que para las coordenadas, como se puede ver en la Figura 175.

```

82 data[16+(32*i)] = FLOAT_TO_UINT(v) & 0X000000FF;
83 data[17+(32*i)] = (FLOAT_TO_UINT(v) & 0X0000FF00) >> 8;
84 data[18+(32*i)] = (FLOAT_TO_UINT(v) & 0X00FF0000) >> 16;
85 data[19+(32*i)] = (FLOAT_TO_UINT(v) & 0XFF000000) >> 24;
86
87 data[20+(32*i)] = FLOAT_TO_UINT(i) & 0X000000FF;
88 data[21+(32*i)] = (FLOAT_TO_UINT(i) & 0X0000FF00) >> 8;
89 data[22+(32*i)] = (FLOAT_TO_UINT(i) & 0X00FF0000) >> 16;
90 data[23+(32*i)] = (FLOAT_TO_UINT(i) & 0XFF000000) >> 24;

```

Figura 175. Agregación de velocidad e identificador en el vector de datos de la nube de puntos.

A la hora de rellenar el mensaje sensor_msgs/PointField, en la constante *datatype* se escribía un 7 para indicar que el tipo de datos es float32. A la hora de programar, **es mejor utilizar etiquetas que den información** en vez de valores numéricos, ya que otro usuario puede reconocer el tipo de dato fácilmente. Por este motivo **se ha sustituido el número 7 por sensor_msgs::PointField::FLOAT32**, como se puede ver en las líneas 170, 177, 184, 191 y 198 de la Figura 176. Se han añadido la velocidad (aunque el campo de velocidad estaba definido en field3, el vector de datos no se rellenaba con la velocidad) y el identificador con los parámetros correspondientes para poder incluirlos en el mensaje de la nube de puntos.

```

167 sensor_msgs::PointField field0;
168 field0.name = "x";
169 field0.offset = 0;
170 field0.datatype = sensor_msgs::PointField::FLOAT32;
171 field0.count = 4;
172 PCL.fields.push_back(field0);
173
174 sensor_msgs::PointField field1;
175 field1.name = "y";
176 field1.offset = 4;
177 field1.datatype = sensor_msgs::PointField::FLOAT32;
178 field1.count = 4;
179 PCL.fields.push_back(field1);
180
181 sensor_msgs::PointField field2;
182 field2.name = "z";
183 field2.offset = 8;
184 field2.datatype = sensor_msgs::PointField::FLOAT32;
185 field2.count = 4;
186 PCL.fields.push_back(field2);
187
188 sensor_msgs::PointField field3;
189 field3.name = "velocidad";
190 field3.offset = 16;
191 field3.datatype = sensor_msgs::PointField::FLOAT32;
192 field3.count = 4;
193 PCL.fields.push_back(field3);
194
195 sensor_msgs::PointField field4;
196 field4.name = "identificador";
197 field4.offset = 20;
198 field4.datatype = sensor_msgs::PointField::FLOAT32;
199 field4.count = 4;
200 PCL.fields.push_back(field4);

```

Figura 176. Sensor_msgs/PointField para rellenar los campos de coordenadas, velocidad y nube de puntos.

En cuanto a las etiquetas de velocidad, se ha hecho un código para que **cuando se detecte velocidad nula, su etiqueta aparezca en rojo, y para el resto aparezca en verde**. Se definen las variables de los colores fuera del bucle for (Figura 177), se crea el código para distinguir las distintas velocidades (Figura 178) y a los valores del mensaje std_msgs/ColorRGBA se les da las variables de los colores.

```

32
33 float red = 0;
34 float green = 1;
35

```

Figura 177. Definición de variables para el color de las etiquetas.

```

95 if (v==0){
96     red =1;
97     green = 0;
98 }
99 else{
100     green = 1;
101     red = 0;
102 }

```

Figura 178. Código para distinguir las velocidades no nulas.

```
137     std_msgs::ColorRGBA color;  
138     color.r = red;  
139     color.g = green;  
140     color.b = 0;  
141     color.a = 1;  
142     Text.color = color;
```

Figura 179. Valores del mensaje `std_msgs/ColorRGBA` para diferentes colores.

Al igual que en el caso anterior, **es preferible usar etiquetas en vez de números**. En este caso se puede realizar para **definir el tipo de *marker* y la acción** (Figura 180).

```
109     Text.ns = "Texto_velocidad";  
110     Text.id = i;  
111     Text.type = visualization_msgs::Marker::TEXT_VIEW_FACING;  
112     Text.action = visualization_msgs::Marker::MODIFY; //Add or modify  
113
```

Figura 180. Etiquetas para definir tipo y acción.

Además, se ha modificado **la posición del texto**, para que quede en la parte superior del *target*. Esto se realiza modificando el mensaje `geometry_msgs/Point` dentro del mensaje `geometry_msgs/Pose`. En este caso con **aumentar la posición en el eje x y la posición en el eje z 0.3 metros**, es suficiente para que se vea la velocidad de una forma clara y limpia.

```
114     geometry_msgs::Pose pose;  
115  
116     geometry_msgs::Point position;  
117     position.x = (posx+0.3);  
118     position.y = posy;  
119     position.z = (posz+0.3);  
120     pose.position = position;  
121
```

Figura 181. Modificación de la posición de la etiqueta de velocidad.

El diagrama de flujo del bucle *TrackArrayCallback* se puede ver en la Figura 182.

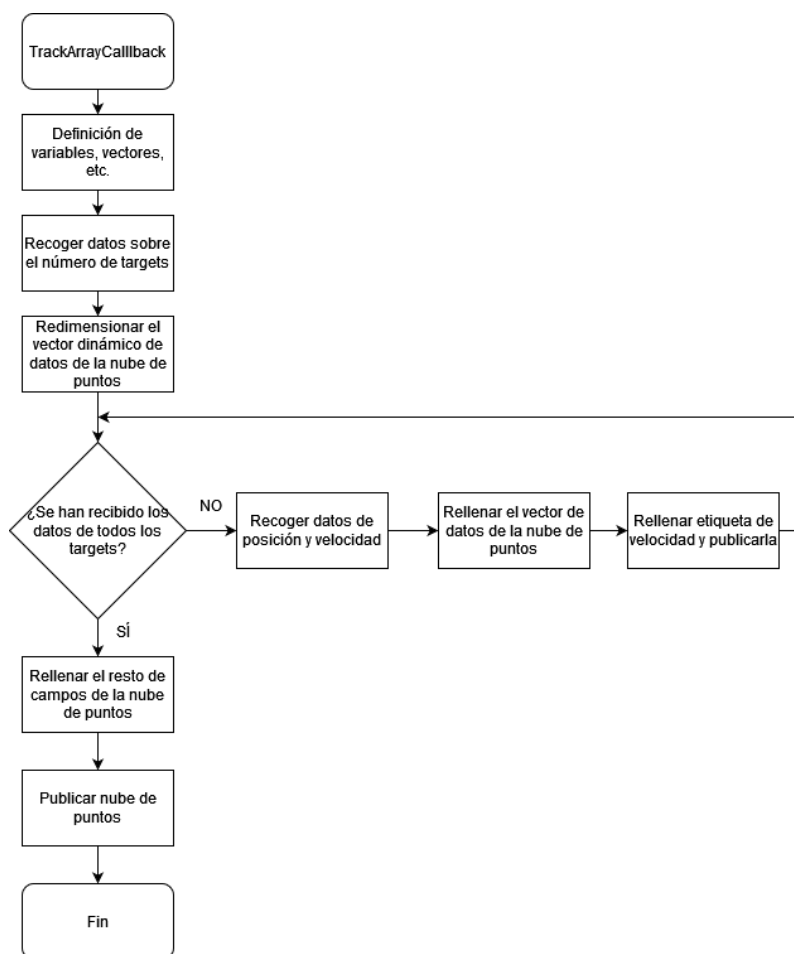


Figura 182. Diagrama de flujo del bucle TrackArrayCallback.

Que se ejecute este bucle significa que **han llegado datos del *topic* suscrito**. En primer lugar, se definen las variables locales, vectores dinámicos, etc. Después, se recogen los datos sobre el **número de targets** detectados y se redimensiona el vector dinámico. Una vez recogidos estos datos, se entra al bucle *for*, que se ejecutará tantas veces como targets haya. En ese bucle se **recogen los datos de posición y velocidad**, se rellena el vector dinámico (vector de datos de la nube de puntos) con los datos necesarios, se rellenan todos los datos necesarios para el mensaje que genera la etiqueta de velocidad y se **publica**. Una vez se hayan procesado los datos de todos los targets, se rellenan los campos restantes del mensaje de la nube de puntos y se **publica**.

Dentro de *Rviz*, se pueden configurar diferentes parámetros para poder ver todo de una forma más clara. Para los targets, se ha cambiado el formato para que se vean mejor. Como se ha explicado anteriormente, **en función del identificador el *target* se verá de diferente color**. Además, se van a volver a **mostrar los targets en forma de cubos, pero más pequeños que antes y transparentes**, para que se pueda ver la nube de puntos que hay detrás o para que se distingan dos targets si se superponen. Todos estos cambios se pueden apreciar en la Figura 183.

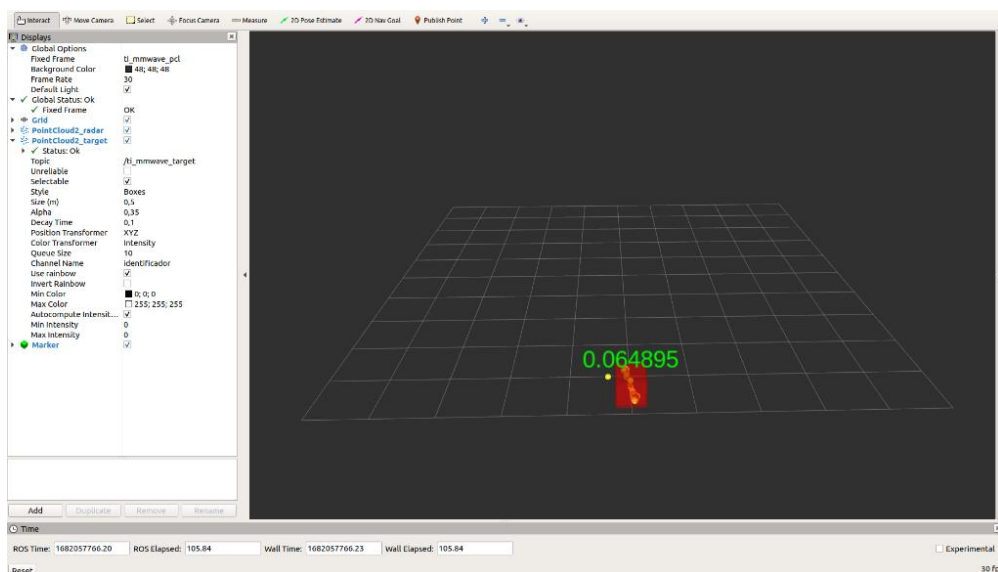


Figura 183. Cambios en el formato del target.

Ahora solo queda poner todo junto en Rviz y ver el **resultado final** para esta demo, es decir, juntar la cámara y el radar. El resultado se puede ver en la Figura 184. **La caja perteneciente al target se encuentra dentro de la caja o boundary-box que genera el radar**, y se puede apreciar la velocidad del target en m/s correctamente.

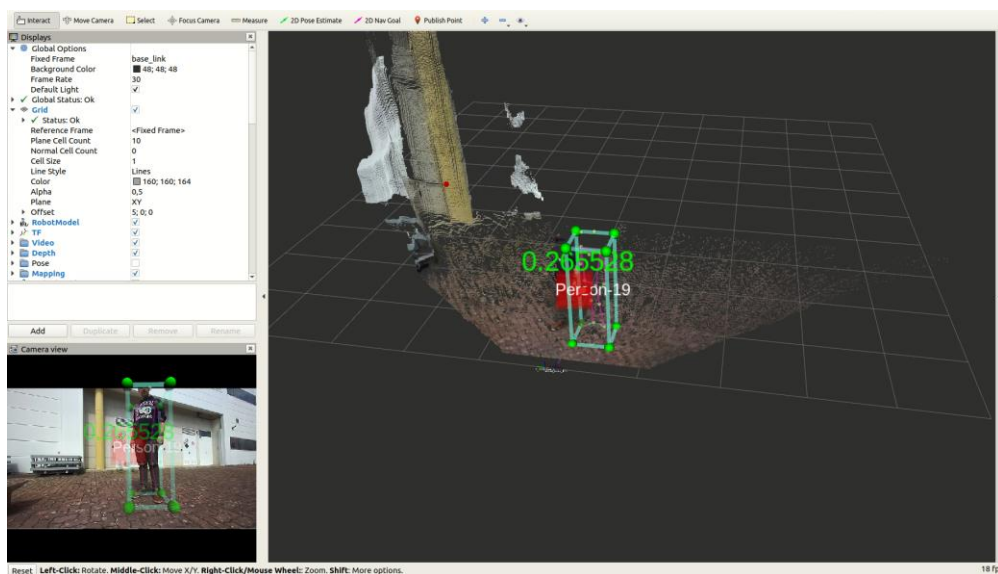


Figura 184. Resultado final para la demo 3D_People_Counting.

6.3 Mobile_Tracker

En este apartado se cargará en el firmware del radar la demo **Mobile_Tracker**, la cual se juntará con la cámara ZED 2i. El driver de ROS no está preparado para aceptar algunos de los TLV nuevos de la demo, por lo que habrá que modificar el paquete de ROS para que sea compatible la demo y funcione correctamente. Esta demo solo se puede usar para el radar **IWR6843ISK**, por lo que se utilizará ese modelo de radar.

6.3.1 Comandos de configuración de la demo Mobile_Tracker

Esta demo **no tiene parámetros nuevos**. Los parámetros son exactamente **los mismos que la demo 3D_People_Counting**. De hecho, en la guía de usuario [34], no se explican los comandos y se hace referencia a las guías de usuario de la demo 3D_People_Counting ([31] y [32]).

6.3.2 TLV de la demo Mobile_Tracker

La demo Mobile_Tracker proporciona 4 TLV:

- MMWDEMO_OUTPUT_MSG_DETECTED_POINTS_SIDE_INFO, **identificador 7**
- MMWDEMO_OUTPUT_MSG_SPHERICAL_POINTS, **identificador 1000**
- MMWDEMO_OUTPUT_MSG_TRACKERPROC_3D_TARGET_LIST, **identificador 1010**
- MMWDEMO_OUTPUT_MSG_TRACKERPROC_TARGET_INDEX, **identificador 1011**

6.3.3 Adaptación del paquete de ROS para Mobile_Tracker

Como se ha comentado anteriormente, **el driver de ROS no está preparado para esta demo**. Que el paquete esté o no preparado no depende de para qué esté hecha la demo, sino de los **TLV que esa demo proporciona**. Dos **demos completamente diferentes pero que devuelvan los mismos TLV serían compatibles para el mismo paquete de ROS**.

Antes de elegir qué paquete de ROS editar, hay que tener en cuenta una **simplificación** que se va a realizar. La demo devuelve un TLV con la nube de puntos (identificador 1000), un TLV con información adicional de los puntos (identificador 7) y dos TLV de tracking (identificadores 1010 y 1011). La información adicional de los puntos no interesa, ya que es el ruido o *noise floor* y el SNR, por lo que **se detectará que ha llegado ese TLV y se desechará la información**. Con la nube de puntos se hará lo mismo. Aunque puede resultar útil ver la nube de puntos, realmente **lo que interesa son los targets**, por lo que se hará lo mismo que para el TLV de información adicional.

El ti_mmwave_ropkg está preparado para la demo Out_of_the_box (TLV con identificador del 1 al 9) y **ti_mmwave_tracker_ropkg para 3D_People_Counting** (TLV con identificadores 1010, 1011 y 1020). Los identificadores de los TLV que devuelve Mobile_Tracker son 7, 1000, 1010 y 1011. Se ha decidido **modificar ti_mmwave_tracker_ropkg** por diversos motivos.

En primer lugar, en ti_mmwave_tracker_ropkg están **implementados dos TLV**, uno más que en el paquete ti_mmwave_ropkg. Además, los TLV soportados son los que tienen identificadores 1010 y 1011, es decir, proporcionan información importante, ya que son los **TLV de tracking**. En el otro paquete, ti_mmwave_ropkg, el identificador del TLV soportado es el 7, cuya información se desechará. Si se quisiera modificar este último paquete, habría que recoger los datos de los TLV de *tracking*, modificarlos y publicarlos. Esto podría llegar a ser muy complicado, por lo que este es el motivo principal por el que se ha elegido no editar este paquete. **El TLV con identificador 1000 no está implementado en ningún paquete**, por lo que habría que implementarlo en ambos paquetes.

Además, al modificar el paquete **ti_mmwave_tracker_ropkg**, se hará compatible con la demo de Mobile_tracker, pero no dejará de serlo con la demo 3D_People_Counting, solo hay que tener en cuenta que el archivo de configuración que se utilice sea el correcto.

Hay que **editar 2 archivos** dentro del paquete de ROS, DataHandlerClass.cpp y mmWave.h. En este último es en el que menos código hay que editar, únicamente **hay que definir los nuevos TLV**. Los TLV anteriormente definidos son los 3 últimos de la Figura 209 (desde la línea 63 a la línea 70). Se definen los

nuevos TLV de la misma forma que los anteriores, con su nombre completo igualado a su identificador. El resto de los valores y definiciones no se deben tocar, ya que no son TLV y pueden influir en el funcionamiento del nodo.

```

53  enum MmwDemo_Output_TLV_Types
54  {
55      MMWDEMO_OUTPUT_MSG_NULL = 0,
56
57      /*! @brief List of detected points side information */
58      MMWDEMO_OUTPUT_MSG_DETECTED_POINTS_SIDE_INFO = 7,
59
60      /*! @brief Tracker TLV's */
61      MMWDEMO_OUTPUT_MSG_SPHERICAL_POINTS = 1000,
62
63      /*! @brief Tracker TLV's */
64      MMWDEMO_OUTPUT_MSG_TRACKERPROC_3D_TARGET_LIST = 1010,
65
66      /*! @brief Tracker TLV's*/
67      MMWDEMO_OUTPUT_MSG_TRACKERPROC_TARGET_INDEX = 1011,
68
69      /*! @brief 3D Spherical Compressed Point Cloud */
70      MMWDEMO_OUTPUT_MSG_COMPRESSED_POINTS = 1020,
71
72      MMWDEMO_OUTPUT_MSG_MAX = 3
73  };
    
```

Figura 185. Archivo mmWave.h modificado.

Además, hay que **definir los nuevos estados de SorterState** para que, cuando se detecte alguno de los TLV nuevos, se pueda acceder a la parte del código (que se escribirá en el archivo **DataHandlerClass.cpp**) donde se escribirá el código para cada uno de ellos.

```

74
75  enum SorterState { READ_HEADER,
76                  CHECK_TLV_TYPE,      Pedrhom Nafis
77                  READ_SPHERE_POINT_CLOUD,
78                  READ_3D_TARGET_LIST,
79                  READ_SPHERICAL_POINTS,
80                  READ_TARGET_INDEX,
81                  SWAP_BUFFERS,
82                  READ_SIDE_INFO };
83
    
```

Figura 186. Definición de los nuevos estados SorterState.

En el archivo DataHandlerClass.cpp hay que añadir código en lugares concretos, por lo que hay que tener claro **cómo funciona el programa**. Una vez ejecutada la parte principal del programa, descifrada la cabecera del TLV, etc., llegará a la **instrucción SorterState = CHECK_TLV_TYPE**, y se irá a la parte del programa **case CHECK_TLV_TYPE**. En esta parte del programa **se comprueba si han llegado todos los TLV**. Si han llegado todos, se hacen los ajustes y se publica lo necesario para finalizar el programa, pero si no han llegado todos lo primero que se hace es **recoger el identificador del nuevo TLV y su longitud**. Posteriormente se irá a la **zona donde esté el código del TLV que corresponda** con el identificador recibido, se ejecutará y volverá a CHECK_TLV_TYPE. Antes de eso es muy importante actualizar el **puntero currentDatap**. Este puntero tiene la **posición del paquete de datos** que ya se ha interpretado o se ha realizado el *parsing* y hay que **incrementar su valor** cada vez que se procese un TLV para poder avanzar en el paquete de datos (hay un paquete por frame). El procedimiento se puede entender mejor con el esquema de la Figura 187.

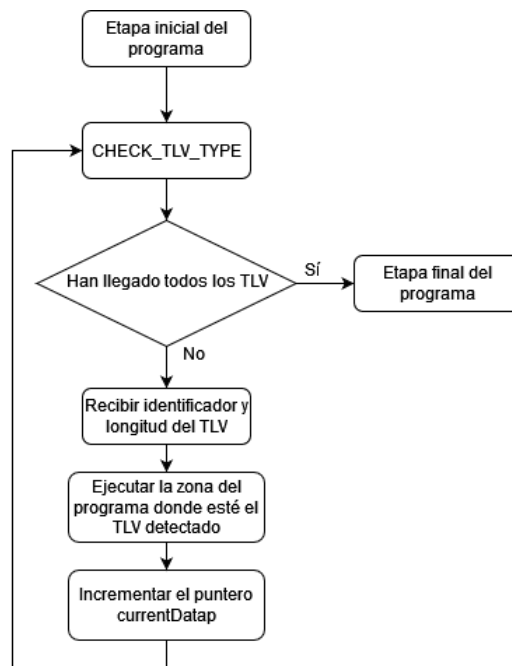


Figura 187. Esquema de interpretación de los TLV.

En primer lugar, hay que **definir los nuevos TLV dentro del CHECK_TLV_TYPE**. El código que se tiene que añadir se puede ver en la Figura 188 (desde la línea 774 hasta la línea 783)

```

774 case MMWDEMO_OUTPUT_MSG_SPHERICAL_POINTS:
775 //ROS_INFO("DataUARHandler Sort Thread : Compressed Points TLV");
776 sorterState = READ_SPHERICAL_POINTS;
777 break;
778
779
780 case MMWDEMO_OUTPUT_MSG_DETECTED_POINTS_SIDE_INFO:
781 //ROS_INFO("DataUARHandler Sort Thread : Side info TLV");
782 sorterState = READ_SIDE_INFO;
783 break;
784
785
786 case MMWDEMO_OUTPUT_MSG_COMPRESSED_POINTS:
787 //ROS_INFO("DataUARHandler Sort Thread : Compressed Points TLV");
788 sorterState = READ_SPHERE_POINT_CLOUD;
789 break;
790
791 case MMWDEMO_OUTPUT_MSG_TRACKERPROC_3D_TARGET_LIST:
792 //ROS_INFO("DataUARHandler Sort Thread : 3D Target List TLV");
793 sorterState = READ_3D_TARGET_LIST;
794 break;
795
796 case MMWDEMO_OUTPUT_MSG_TRACKERPROC_TARGET_INDEX:
797 //ROS_INFO("DataUARHandler Sort Thread : Target Index TLV");
798 sorterState = READ_TARGET_INDEX;
799 break;
800
801 case MMWDEMO_OUTPUT_MSG_MAX:
802 //ROS_INFO("DataUARHandler Sort Thread : Header TLV");
803 sorterState = READ_HEADER;
804 break;
805 default: break;
806 }
  
```

Figura 188. Definición de lo TLV dentro de CHECK_TLV_TYPE.

Una vez definidos hay que **escribir el código correspondiente a cada uno de ellos**. Para el identificador 7 es relativamente fácil, ya que es un TLV soportado en el otro paquete de ROS, ti_mmwave_ropkg. Basta con **copiar el código y eliminar las líneas donde se realice la toma de datos**. Es importante también **mover la línea de código del puntero fuera de los bucles**, porque si no este daría problemas (Figura 189). De esta forma, se detecta cuando ha llegado el TLV, se aumenta el valor del puntero y se vuelve a CHECK_TLV_TYPE.


```

609
610     case READ_SIDE_INFO:
611
612         // Make sure we already received and parsed detected obj list (READ_OBJ_STRUCT)
613         if (mmwData.numObjOut > 0)
614         {
615             for (i = 0; i < mmwData.numObjOut; i++)
616             {
617             }
618         }
619         else // else just skip side info section if we have not already received and parsed detected obj list
620         {
621             i = 0;
622
623             while (i++ < tlVLen - 1)
624             {
625                 //ROS_INFO("DataUARTHandler Sort Thread : Parsing Side Info i=%d and tlVLen = %u", i, tlVLen);
626             }
627             //currentDatap += tlVLen;
628         }
629         currentDatap += tlVLen;
630         sorterState = CHECK_TLV_TYPE;
631
632
633
634

```

Figura 189. Código del TLV de identificador 7.

Si se hiciera lo mismo con el otro TLV, a nivel de código estaría bien hecho. El problema es que el algoritmo de Texas Instruments necesita al menos un punto para que el árbol de transformadas (o ejes de coordenadas) funcione correctamente. Esto se puede ver en la Figura 190, donde aparece una advertencia o Warning hasta que detecta un punto.

```

[ WARN ] [1682079266.946559754]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079266.99667988]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.046662964]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.096580759]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.146715425]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.196688035]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.246785639]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.296696124]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.346657326]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.396634480]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.446650349]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.496681386]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.546657149]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.596580573]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.64664053]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.696780735]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.746664141]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.79661934]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.846644953]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.896741549]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.946667994]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079267.996559115]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.04668261]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.096538834]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.146664284]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.196492663]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.246551982]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.296661152]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.346530517]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.396598956]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.446785798]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.496981923]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.546577543]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.596529913]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.646715958]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.696604971]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.746873332]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.796642836]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.846593887]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
[ WARN ] [1682079268.897466525]: Invalid argument passed to canTransform argument source_frame in tf2 frame_ids cannot be empty
QObject::connect: Cannot queue arguments of type 'QVector<int>'
(Make sure 'QVector<int>' is registered using qRegisterMetaType().)
QObject::connect: Cannot queue arguments of type 'QVector<int>'
(Make sure 'QVector<int>' is registered using qRegisterMetaType().)

```

Figura 190. Necesidad de tener al menos 1 punto para que pueda mostrarse en Rviz.

Esto no sucede únicamente para esta demo, para 3D_People_Counting ocurre lo mismo. Por este motivo, cada vez que se detecte el TLV de identificador 1000 se publicarán (no se publica en el topic de la nube de puntos) los datos de un punto fijo (aunque no se necesite ni se muestre en Rviz al no estar en el formato adecuado). Se copiará el formato de publicación de un punto de la parte superior del código (TLV con identificador 1020), y se editarán algunos valores para adecuar el código a un solo punto.

```

637     case READ_SPHERICAL_POINTS:
638
639         i = 0;
640         offset = 0;
641
642         mmwData.numObjOut = 1;
643         pcl_conversions::toPCL(ros::Time::now(), RScan->header.stamp);
644         RScan->header.frame_id = frameID;
645         RScan->height = 1;
646         RScan->width = 1;
647         RScan->is_dense = 1;
648         RScan->points.resize(32);
649
650         realElevation = 0;
651         realAzimuth = 0;
652         realDoppler = 0;
653         realRange = 0;
654         realSNR = 0;
655
656         radarscan.header.frame_id = frameID;
657         radarscan.header.stamp = ros::Time::now();
658
659         radarscan.point_id = i;
660         radarscan.x = 1;
661         radarscan.y = 1;
662         radarscan.z = 1;
663         radarscan.range = 2;
664         radarscan.velocity = 2;
665         radarscan.intensity = 2.5;
666
667         radar_scan_pub.publish(radarscan);
668         while( i < tlvLen ) {
669             i++;
670         }
671     You, 3 minutes ago · Uncommitted changes
672     currentDatap += tlvLen;
673     sorterState = CHECK_TLV_TYPE;
674     break;

```

Figura 191. Código para el TLV con identificador 1000 añadiéndole un punto fijo.

Al haberse realizado ajustes sobre el `ti_mmwave_tracker_ropkg` el **nodo creado en el apartado 6.2.4 sirve para esta demo** también y se utiliza de la misma forma que para la demo `3D_People_Counting`. El funcionamiento se puede ver en la Figura 192.

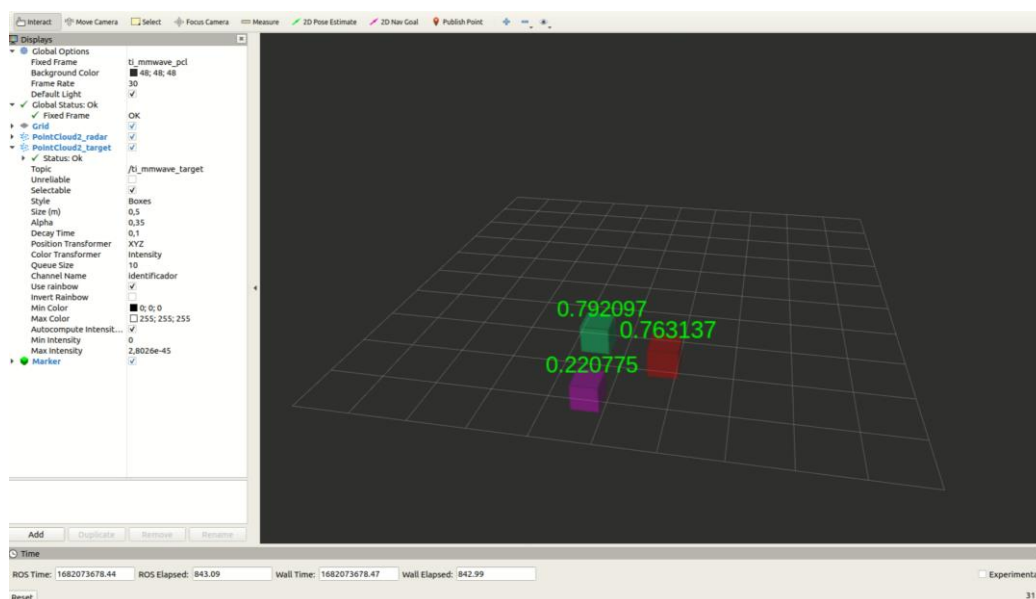


Figura 192. Demo `Mobile_Tracker` con Rviz.

6.3.4 `Mobile_Tracker` con ZED 2i

Para juntar el radar con la cámara ZED 2i basta con ejecutar los nodos de ROS exactamente igual que en el caso de la demo `3D_People_Counting` (apartado 6.2.5). El resultado se puede ver en la Figura 193.

Figura 193

Figura 193. Resultado final para la demo `Mobile_tracker`.

7 Ensayos en exteriores

En este apartado se describirá con detalle todo lo relacionado con los **ensayos en exteriores**, que incluye el **montaje y puesta a punto del Rover**, la **validación de las ecuaciones** de entorno y la comprobación del **funcionamiento de las demos**.

7.1 Validación experimentalmente de las ecuaciones usando OOB

En este apartado se **validarán experimentalmente** las ecuaciones para los parámetros de entorno del apartado 5.4.4: rango, velocidad y ángulo máximos (la resolución de velocidad, rango y ángulo son parámetros complicados de validar). Para ello **se configurarán los chirps del radar dependiendo de lo que se quiera conseguir**. Una herramienta muy buena para ello es **mmWave Sensing Estimator** [84], de Texas Instruments. En ella, se pueden **seleccionar los parámetros de entorno deseados y devuelve los parámetros del chirp** para que se cumplan esos requisitos. A veces se hará utilizar a esta herramienta para poder determinar los parámetros de una forma optimizada, ya que a **en varias ocasiones, habrá que fijar ciertos valores** para poder calcular otros. Estos valores se podrían fijar arbitrariamente (dentro de los rangos que permite el radar) y configurar el resto de los parámetros, pero se corre el riesgo que no estén configurados de forma óptima, ya que **mmwave Sensing Estimator utiliza un algoritmo que realiza esos cálculos** y tiene en cuenta muchos otros factores que afectan al funcionamiento del radar. Todos estos ensayos se realizarán con la **demo 3D_People_Counting y con el nuevo nodo de ROS**.

El resto de los parámetros que dependen del RCS **no son fáciles de medir experimentalmente** sin saber exactamente el RCS ni otros parámetros de pérdidas. A pesar de ello se estimará el **RCS de una persona adulta** (1 m²) y se supondrán **valores típicos** para los parámetros que no sean fácilmente medibles. Los resultados aparecerán en las tablas a modo de **aproximación teórica**.

7.1.1 Configuración de corto alcance de *Mmwave Sensing Estimator*

La primera comprobación que se va a realizar es con la **configuración del chirp por defecto** (de corto alcance) de la herramienta *mmwave Sensing Estimator*. Los parámetros del *chirp* se pueden ver en la Tabla 20 y los resultados para los parámetros de entorno en la Tabla 21.

Parámetros del chirp configurables			
Variable	Unidad	Valor	Configuración
Antenas receptoras (N_{Rx})	-	15 (4)	channelCfg>rxChannelEn
Antenas transmisoras (N_{Tx})	-	7 (3)	channelCfg>txChannelEn
Start frequency (f_o)	MHz	60000	profileCfg>startFreq
Idle time (T_{idle})	μs	7	profileCfg>IdleTime
ADC valid start time (T_{ADC})	μs	5,7	profileCfg>adcStartTime
Ramp end time (T_{ramp})	μs	49	profileCfg>rampEndTime
Tx output power (P_t)	dBm	657930 (2)	profileCfg>txOutPower
	W	0,016	$P_t(W) = 10^{P_t(dBm)/10}/1000$
frequency slope (S)	MHz/ μs	70,9	profileCfg>freqSlopeConst
Tx Start time	μs	1	profileCfg>txStartTime
Number of ADC Samples (N_{ADC})	-	250	profileCfg>numADCSamples
Sampling Rate ($ADC_{sampling}$)	MspS	5.91	profileCfg>digOutSampleRate
Number of chirp loops (N_c)	-	27	frameCfg>number of loops

Number of frames	-	0 (∞)	frameCfg>number of frames
Frame periodicity	ms	100	frameCfg>frame periodicity

Tabla 20. Parámetros de configuración por defecto mmwave Sensing Estimator.

Parámetros de entorno		
Variable	Unidad	Valor
Rango máximo	m	10,00
Rango máximo (SNR)	m	48,09
RCS mínimo al rango máximo	m ²	0,002
Resolución de rango	cm	5,00
Velocidad máxima	m/s	7,21
Resolución de velocidad	m/s	0,53

Tabla 21. Parámetros de entorno de la configuración por defecto mmwave Sensing Estimator.

En la siguiente figura se puede ver la **forma del chirp**, representando frecuencia en función del tiempo.

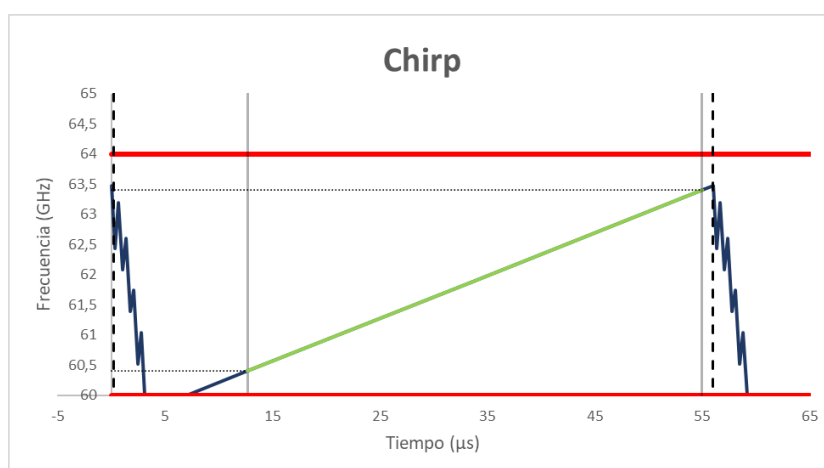


Figura 194. Chirp de la configuración por defecto de mmwave Demo Visualizer.

El radar se colocó en una **posición elevada**, aproximadamente a medio metro de altura, de tal forma que detectara una persona cuando pasara por delante. Debido a esto, el primer parámetro del comando sensorPosition será 0.5.

En un primer momento, el radar **no funcionó**. El problema es que **el número de muestras (N_{ADC}) no puede sobrepasar las 128**. Se desconoce el motivo de porque esto sucede, pero es importante tenerlo en cuenta ya que hay que configurar todo en consecuencia. Otra cuestión a tener en cuenta es que **los bytes del tamaño de memoria debe ser múltiplo de 4** para que se dé una **correcta comunicación** entre el microcontrolador y el DSP. Esto es fácilmente solucionable haciendo que el número de *chirps* por *frame* sea múltiplo de 4, ya que todo lo demás lo será (El número de antenas virtuales es 12 y el número de muestras es 128). En este caso el radar no dejaría de funcionar, pero se verían **comportamientos atípicos** y muchos puntos fantasmas.

En este caso, para que todo funcione correctamente, únicamente se ha modificado el **número de muestras a 128**, obteniendo la siguiente forma del *chirp*.

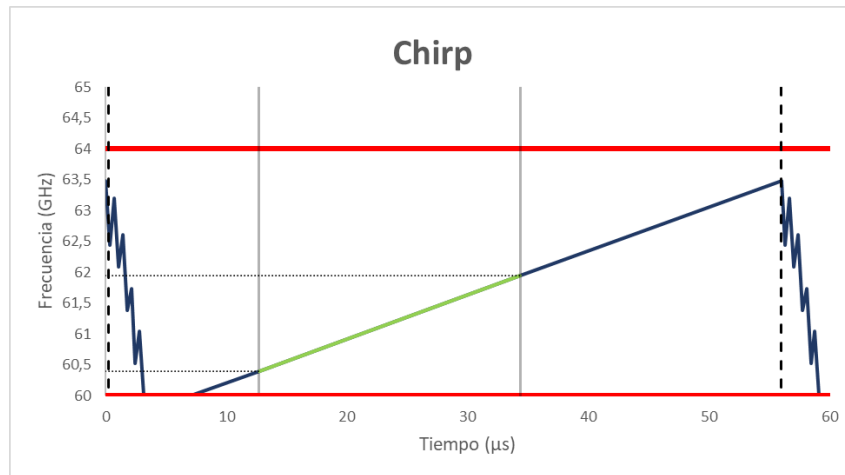


Figura 195. Chirp de la configuración por defecto de mmwave Demo Visualizer con $N_{ADC} = 128$.

Se puede ver en la Figura 196 que el límite del radar está en torno a los 5 metros y medio de distancia máxima (cada celda mide un metro cuadrado).

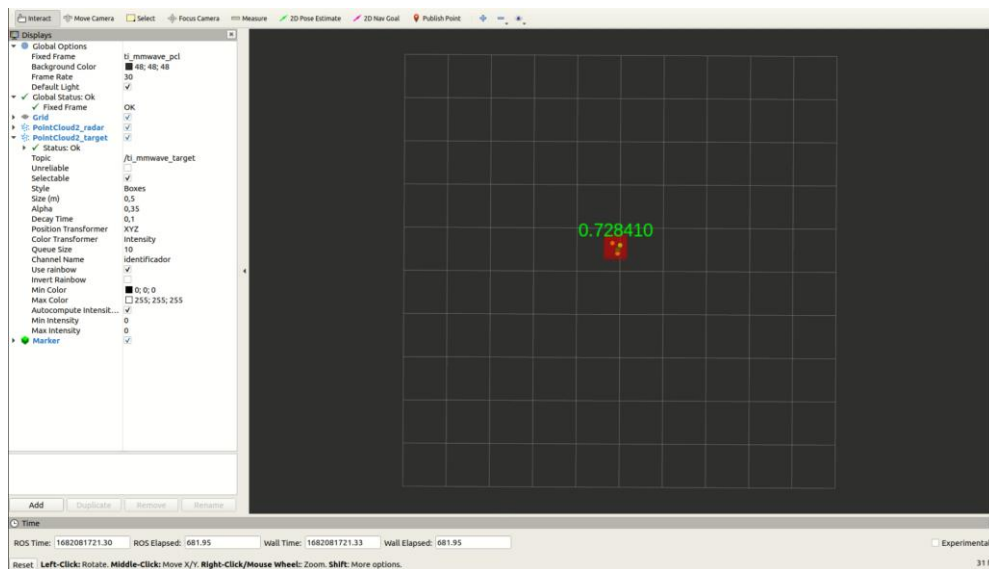


Figura 196. Detección de un objeto a 5,5 metros.

El rango máximo se puede calcular mediante la Ecuación 33. Sustituyendo el valor de IF_{max} (Ecuación 34) en la ecuación del rango, se obtiene la siguiente expresión:

$$Rango_{max} = \frac{0.8 \cdot (ADC_{sampling}) \cdot c}{2S}$$

Ecuación 58

Se puede ver que el rango máximo depende directamente de la frecuencia de muestreo. A pesar de que esta sea casi 6000 ksp/s, no se alcanzan los 10 m de distancia máxima.

El tiempo de muestreo, el número de muestras y la frecuencia de muestreo deben tener una relación, que se puede ver en la Ecuación 59.

$$\frac{N_{ADC}}{(T_{ramp} - T_{ADC})} = ADC_{Sampling}$$

Ecuación 59

Siguiendo la ecuación anterior se puede estimar la nueva frecuencia de muestreo:

$$\frac{128}{(49 - 5.7)} = 2.957 \text{ ksps}$$

Ecuación 60

El tiempo de muestreo se reduciría a unas 3000 ksps, obteniendo un **rango máximo de unos 5,7 m**, que es más o menos la máxima distancia que se observa. A pesar de que no se sabe exactamente porqué ocurre, la distancia máxima es aproximadamente esos 5.7 m, obtenido con la frecuencia de muestreo de la Ecuación 60, **independientemente de si se aumenta la frecuencia de muestreo**. Por lo tanto se dejará el **valor de la frecuencia de muestreo como 2957**.

Este error con la distancia máxima se podría arreglar **reduciendo la pendiente de la recta**, pero se corre el riesgo de afectar a otros parámetros de entorno. En este caso, como no importan los otros parámetros se ha **reducido a 35 MHz/μs**.

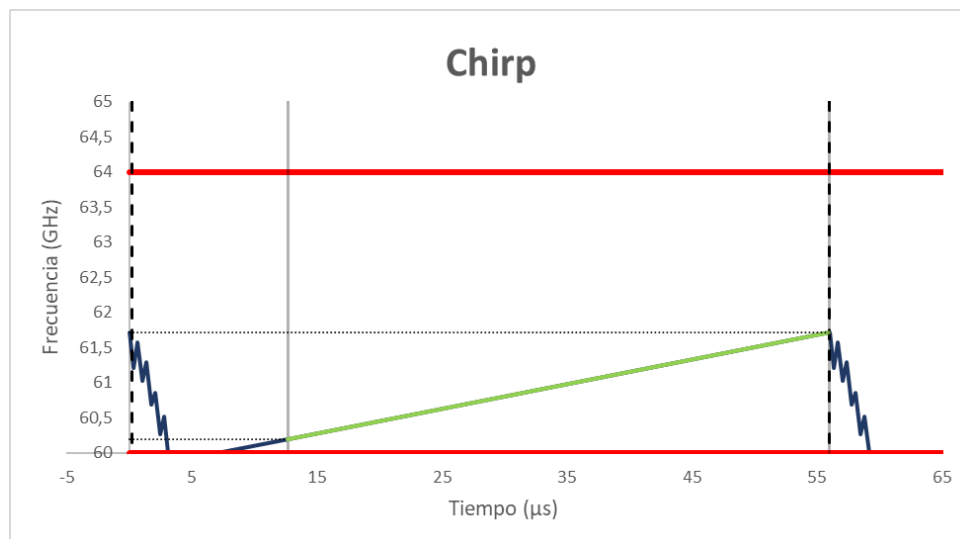


Figura 197. Chirp de la configuración por defecto de mmwave Demo Visualizer con S = 35.

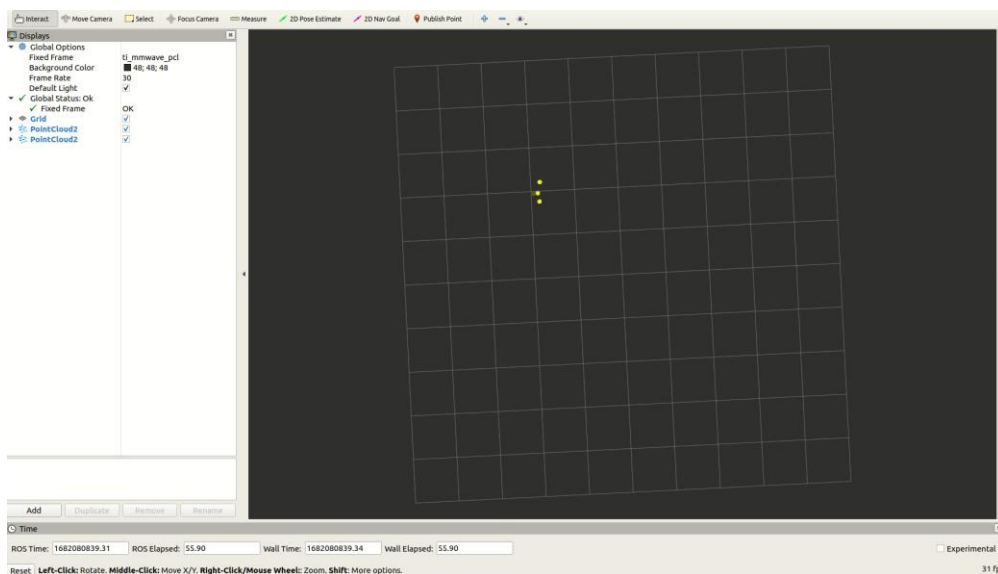


Figura 198. Detección de una persona tras reducir la pendiente de la recta.

Lo que se consigue con esta configuración es **aumentar el rango máximo un par de metros**, aunque se puede ver que alguna vez se detectan unos pocos puntos más allá de esa distancia (entre los 7,5 y los 10 metros), como es el caso de la Figura 198.

Otra opción sería **modificar los tiempos de duración del chirp**, para poder aumentar la frecuencia de muestreo, y por tanto la distancia máxima. **Bajar a 28 el T_{ramp}** haría que la frecuencia de muestreo fuera alrededor de 5740 ksp/s, que daría un rango de algo más de casi 11 m.

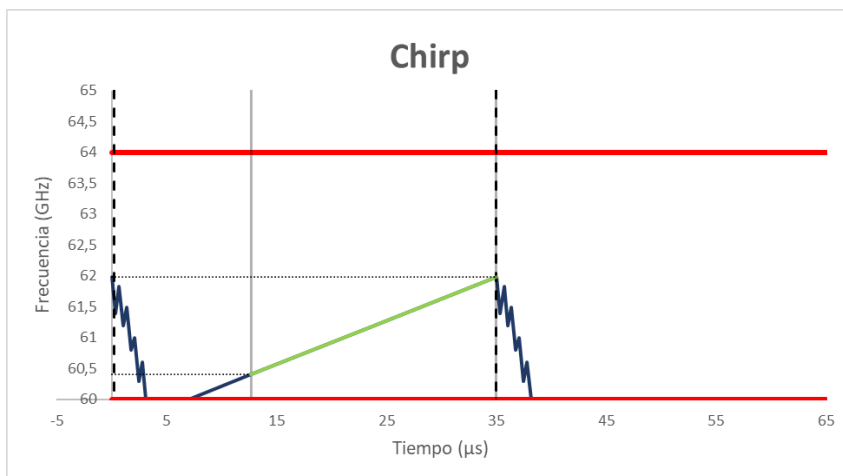


Figura 199. Chirp de la configuración por defecto de mmwave Demo Visualizer con $T_{ramp} = 28$.

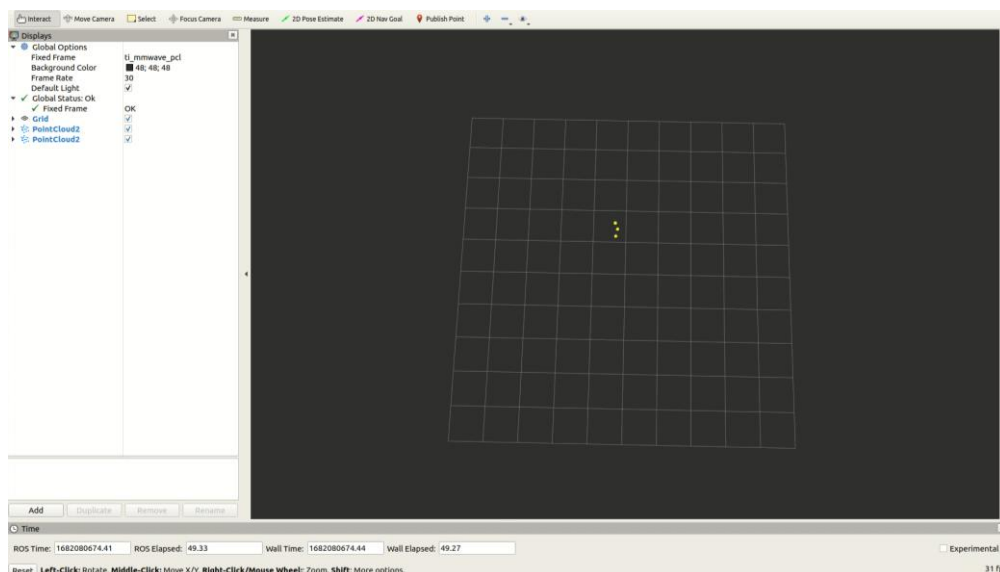


Figura 200. Detección de una persona tras reducir el tiempo del chirp.

El resultado es prácticamente el mismo que al disminuir la pendiente, incluso en este caso es un poquito peor, ya que **deja de detectar bien alrededor de los 6,5 metros**.

7.1.2 Aumento del rango máximo por encima de los 10 metros:

Según la Ecuación 58, el rango máximo depende de la frecuencia de muestreo ($ADC_{sampling}$), y de la pendiente del chirp (S). En este caso, lo ideal sería aumentar la frecuencia de muestreo (y en consecuencia N_{ADC} para que se cumpla la Ecuación 59), por ejemplo, a 11820 ksp/s y N_{ADC} a 510 pero, como se ha visto en el apartado anterior, no se puede aumentar el número de muestras por encima de 128, luego la configuración no funcionaría correctamente. Debido a esto, **se partirá de la configuración del apartado 7.1.1** y se modificará para aumentar el rango.

Se partirá de la configuración anterior tras disminuir la pendiente a $35 \text{ MHz}/\mu\text{s}$. Para conseguir todavía más rango **se disminuye a $25 \text{ MHz}/\mu\text{s}$** , lo que resultaría en **unos 14 metros de rango máximo**. Realizando la prueba se ve el **mismo resultado que antes**, no pasa de los 7,5 metros.

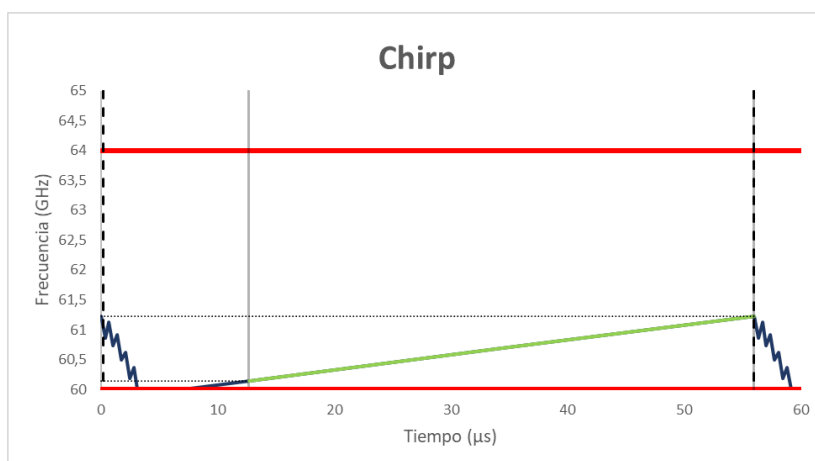


Figura 201. Chirp para aumentar el rango por encima de los 7.5 metros.

Aquí es donde entra el **algoritmo CFAR**. Lo que puede estar ocurriendo es que **el *threshold* o umbral esté demasiado alto**, haciendo que las ondas que llegan de objeto más lejanos de 7,5 metros sean detectadas

como *clutter* (umbral rojo de la Figura 73), lo que explicaría que a veces saliera algún punto más allá de esa distancia. Para configurar bien el umbral del algoritmo CFAR hay que ir **probando las diferentes configuraciones y diferentes umbrales** para comprobar que no haya muchos objetos fantasmas. Estos umbrales se configuran mediante los **comandos `dynamicRACfarCfg` y `staticRACfarCfg`**, en concreto los parámetros 10 y 11 de ambos comandos. Por defecto, estos dos comandos son:

- `dynamicRACfarCfg -1 4 1 2 2 8 12 4 8 15.00 18.00 0.40 1 1`
- `staticRACfarCfg -1 4 1 2 2 8 8 6 4 12.00 19.00 0.30 0 0`.

Para conseguir que objetos más débiles pasen ese primer filtro, **los umbrales de este algoritmo se bajarán en 7 dB** cada uno, quedando de la siguiente forma:

- `dynamicRACfarCfg -1 4 1 2 2 8 12 4 8 8.00 11.00 0.40 1 1`
- `staticRACfarCfg -1 4 1 2 2 8 8 6 4 5.00 12.00 0.30 0 0`.

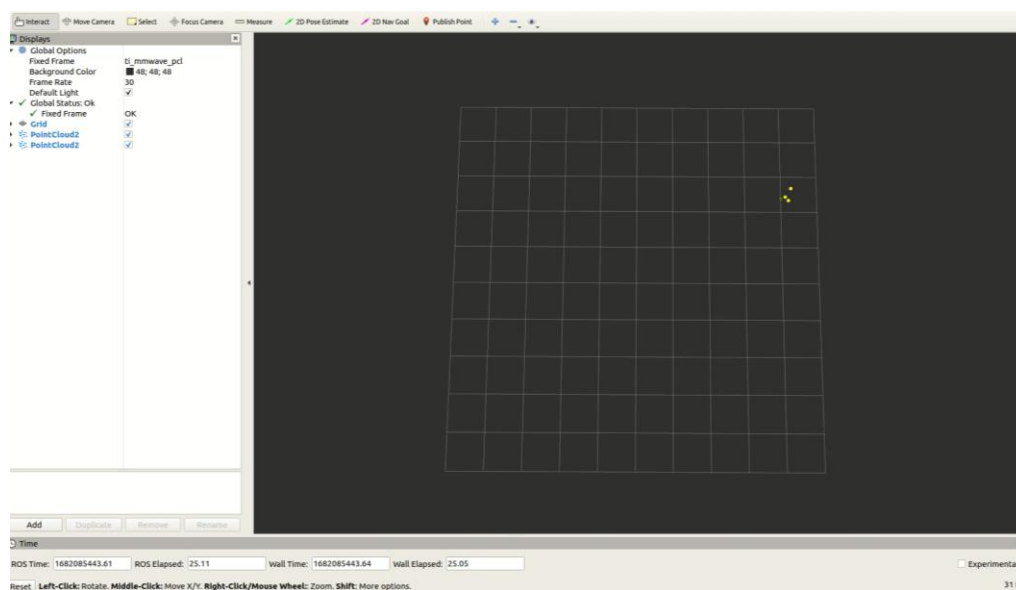


Figura 202. Resultado después de bajar 7 dB los umbrales del CFAR.

Se puede observar que se ha aumentado el rango máximo a **unos 7 metros** y que no hay **ningún punto raro** que pueda provenir de ruido detectado, por lo que se **bajará aún más el umbral** del algoritmo CFAR, por ejemplo, a los siguientes valores:

- `dynamicRACfarCfg -1 4 1 2 2 8 12 4 8 4.00 7.00 0.40 1 1`
- `staticRACfarCfg -1 4 1 2 2 8 8 6 4 2.00 9.00 0.30 0 0`.

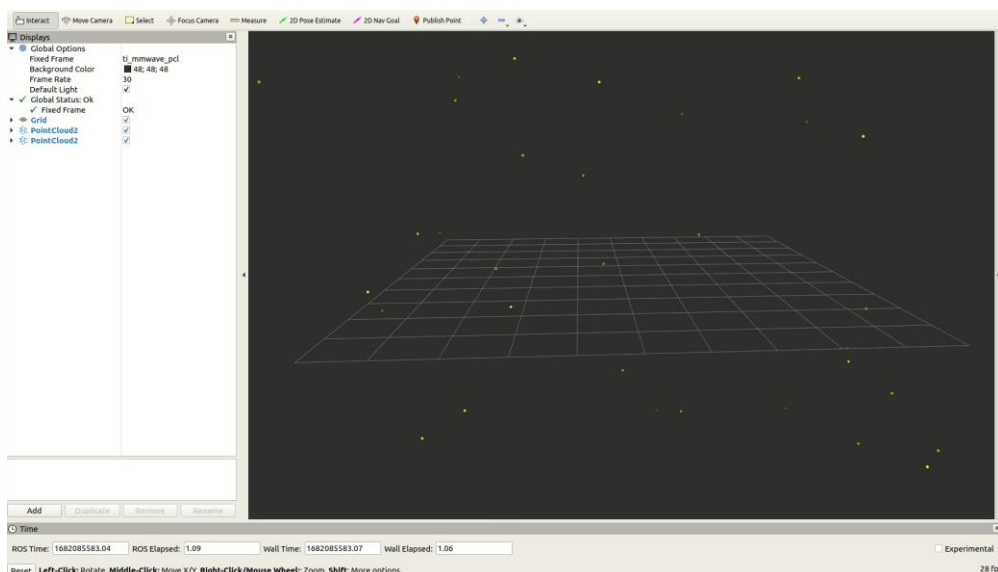


Figura 203. Umbrales del CFAR demasiado bajos.

En este caso se ha **bajado demasiado**, ya que el mapa 3D aparece lleno de puntos aleatorios cambiando constantemente (Figura 203). Este caso se corresponde con el umbral azul de la Figura 73, el umbral es demasiado bajo que el ruido se reconoce como un objeto. Dado que se cuele **mucho ruido en las medidas**, hay que **aumentar el umbral** un poco más:

- dynamicRACfarCfg -1 4 1 2 2 8 12 4 8 **6.00 9.00** 0.40 1 1
- staticRACfarCfg -1 4 1 2 2 8 8 6 4 **4.00 11.00** 0.30 0 0.

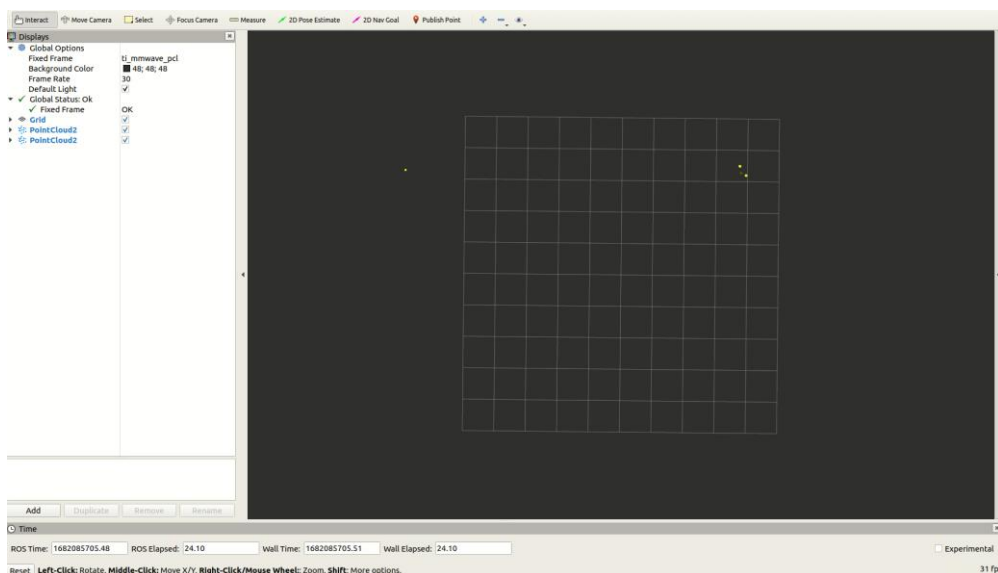


Figura 204. Resultado con los umbrales finales para el CFAR.

En este caso, **no se observan casi puntos debido al ruido**, por lo que esta configuración de umbral de CFAR sirve, pero no detecta más allá de 10 metros.

Lo siguiente que se puede hacer es aumentar la **potencia del radar**, ya que no estaba a máxima potencia. Sustituyendo el valor 657930 por 0 (para obtener la potencia máxima) se obtiene el resultado de la Figura 205. Al hacer que el radar trabaje a potencia máxima, la onda se emitirá con mayor energía y se reflejará

también con más energía, por lo que pasará con mayor facilidad el filtro del algoritmo CFAR (y habrá un poco más de ruido).

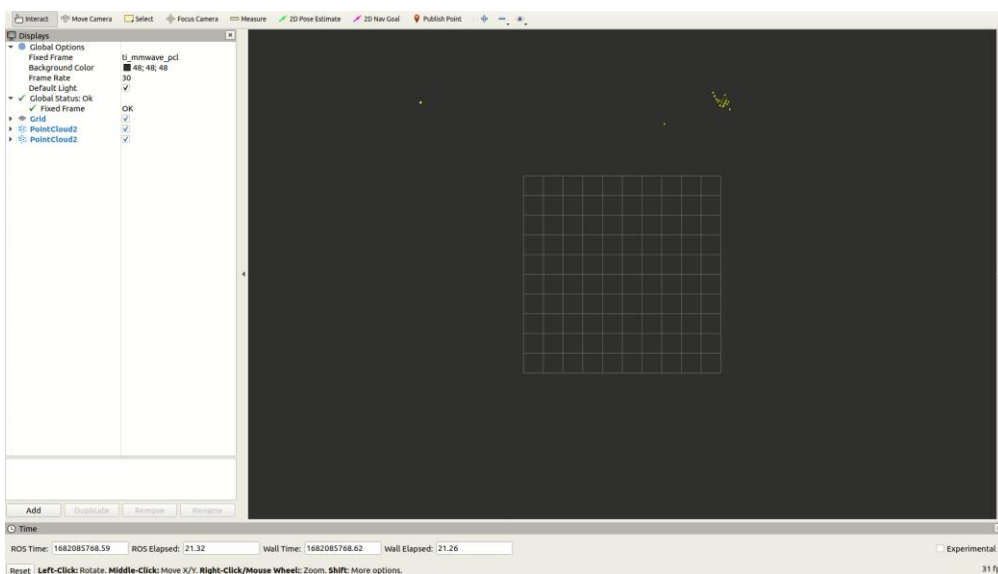


Figura 205. Medida de distancia de 14 metros.

Como se puede ver en la Figura 205, el radar es capaz de **detectar hasta 14 metros** de distancia, a pesar de que los puntos se salgan de las celdas en Rviz, ya que este estaba configurado para 10 metros

En cuanto a las **boundary boxes**, aparecen dentro de un **rango delimitado por un parámetro** de configuración. En este caso, no se detectará el objeto cuando esté fuera de los límites marcados por el comando, que en este caso son ± 2 metros en el eje X y 8 metros en la dirección del eje Y. Fuera de esa zona

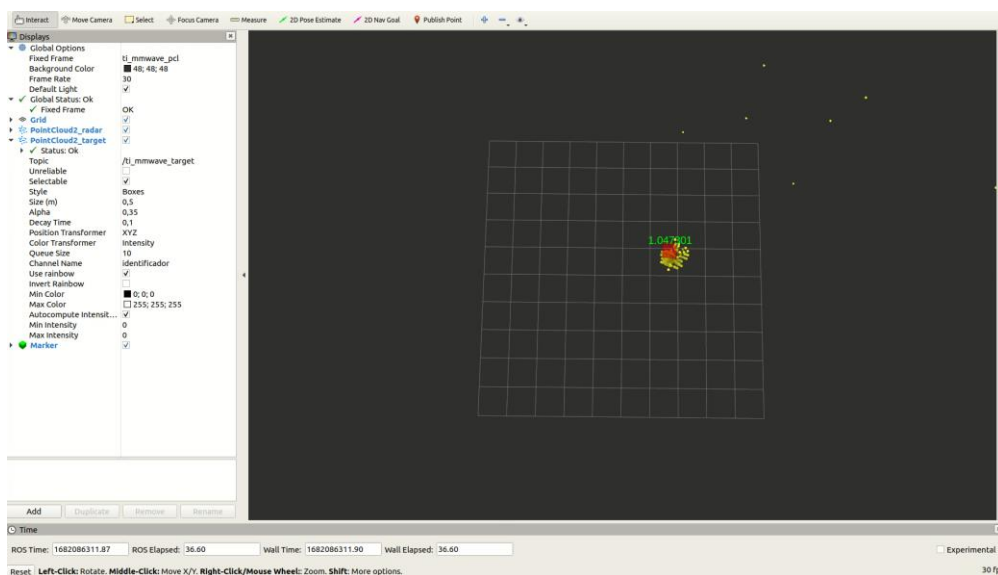


Figura 206. Tracking de una persona con la configuración de 14 metros.

Parámetros del chirp configurables

Variable	Unidad	Valor	Configuración
Antenas receptoras (N_{Rx})	-	15 (4)	channelCfg>rxChannelEn
Antenas transmisoras (N_{Tx})	-	7 (3)	channelCfg>txChannelEn

Start frequency (f_0)	MHz	60000	profileCfg>startFreq
Idle time (T_{idle})	μs	7	profileCfg>idleTime
ADC valid start time (T_{ADC})	μs	5,7	profileCfg>adcStartTime
Ramp end time (T_{ramp})	μs	49	profileCfg>rampEndTime
Tx output power (P_t)	dBm	0 (12)	profileCfg>txOutPower
	W	0,02	$P_t(W) = 10^{P_t(dBm)/10}/1000$
frequency slope (S)	MHz/ μs	25	profileCfg>freqSlopeConst
Tx Start time	μs	1	profileCfg>txStartTime
Number of ADC Samples (N_{ADC})	-	128	profileCfg>numADCsSamples
Sampling Rate ($ADC_{sampling}$)	Msps	2,95	profileCfg>digOutSampleRate
Number of chirp loops (N_c)	-	27	frameCfg>number of loops
Number of frames	-	0 (∞)	frameCfg>number of frames
Frame periodicity	ms	100	frameCfg>frame periodicity

Tabla 22. Parámetros de configuración para obtener un rango máximo de más de 10 m.

Parámetros de entorno			
Variable	Unidad	Valor	
Rango máximo	m	14,16	
Rango máximo (SNR)	m	51,00	
RCS mínimo al rango máximo	m^2	0,0059	
Resolución de rango	cm	13,83	
Velocidad máxima	m/s	7,36	
Resolución de velocidad	m/s	0,46	

Tabla 23. Parámetros de entorno de la configuración para obtener un rango máximo de más de 10 m.

Tras hacer los cálculos, la distancia máxima que permite el radar es 14.16 metros (Tabla 23), por lo que si se modificaran nuevamente parámetros del radar **sería posible aumentar ese rango** un poco más.

7.1.3 Aumentar el rango máximo por encima de los 14 metros

Para conseguir aumentar más el rango, si no hubiera problema en cuanto a número de muestras, habría que aumentar la frecuencia de muestreo a 17.75 Msps, pero **este valor supera el valor máximo permitido**, que es 12.5 Msps. Por lo tanto, habría que **modificar la pendiente** del *chirp*. Para la frecuencia de muestreo máxima (12.5 Msps), hará falta reducir la pendiente a 50 MHz/ μs , obteniendo así el rango de medida de 30 metros. Sería conveniente alejarse un poco de la frecuencia de muestreo máxima, por ejemplo, a **12,010 Msps** (se elige un número un poco raro que el *excess time* sea positivo, es decir, se cumpla la Ecuación 59), que sigue siendo bastante alta por lo que se necesitaría una **pendiente de 48 MHz/ μs** .

Parámetros configurables			
Variable	Unidad	Valor	Configuración
Antenas receptoras (N_{Rx})	-	15 (4)	channelCfg>rxChannelEn
Antenas transmisoras (N_{Tx})	-	7 (3)	channelCfg>txChannelEn
Start frequency (f_0)	MHz	60000	profileCfg>startFreq
Idle time (T_{idle})	μs	7	profileCfg>idleTime
ADC valid start time (T_{ADC})	μs	5,7	profileCfg>adcStartTime

Ramp end time (T_{ramp})	μs	49	profileCfg>rampEndTime
Tx output power (P_t)	dBm	0 (12)	profileCfg>txOutPower
	W	0,02	$P_t(W) = 10^{P_t(dBm)/10}/1000$
frequency slope (S)	MHz/ μs	48	profileCfg>freqSlopeConst
Tx Start time	μs	1	profileCfg>txStartTime
Number of ADC Samples (N_{ADC})	-	520	profileCfg>numADCSamples
Sampling Rate ($ADC_{sampling}$)	Msps	12,010	profileCfg>digOutSampleRate
Number of chirp loops (N_c)	-	27	frameCfg>number of loops
Number of frames	-	0 (∞)	frameCfg>number of frames
Frame periodicity	ms	100	frameCfg>frame periodicity

Tabla 24. Parámetros de configuración para obtener un rango máximo de 30 m.

Parámetros de entorno		
Variable	Unidad	Valor
Rango máximo	m	30,00
Rango máximo (SNR)	m	40,69
RCS mínimo al rango máximo	m ²	0,296
Resolución de rango	cm	15,00
Velocidad máxima	m/s	7,35
Resolución de velocidad	m/s	0,54

Tabla 25. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m.

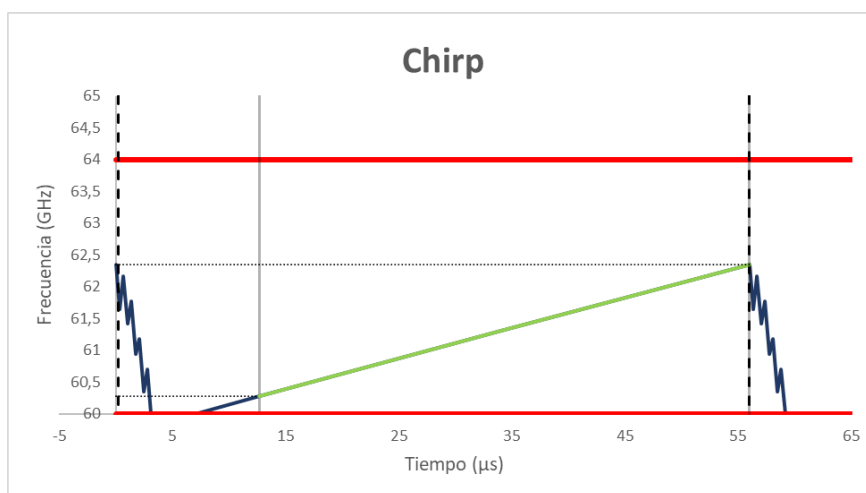


Figura 207. Chirp para conseguir un rango máximo de 30 m.

Esta configuración **no se puede aplicar en la demo 3D_People_Counting**, ya tiene el número de muestras es 520, mayor que 128. La forma para conseguir una distancia mayor es de la misma forma que en los apartados anteriores, **reducir la pendiente del chirp, reducir el tiempo del chirp o modificar los valores de los comandos de configuración de CFAR.**

7.1.4 Resolución de 5 cm

Siguiendo la Ecuación 41, **la resolución de rango solo depende del ancho de banda efectivo (B)**, que sigue la Ecuación 38. Sustituyendo la Ecuación 37 en la Ecuación 38 se obtiene:

$$B = ST_c = S \frac{N_{ADC}}{ADC_{sampling}}$$

Ecuación 61

Sustituyendo la expresión anterior en la ecuación de la ecuación de la resolución de rango:

$$Range_{res} = \frac{c}{2S \frac{N_{ADC}}{ADC_{sampling}}} = \frac{cADC_{sampling}}{2SN_{ADC}}$$

Ecuación 62

Esto demuestra que, la resolución aumenta (empeora) al aumentar la velocidad de muestreo ($ADC_{sampling}$) y al disminuir la pendiente (S) y el número de muestras por *chirp*, N_{ADC} . En este ensayo **se modificarán los parámetros del chirp sin tener en cuenta lo que se modifiquen los parámetros de rango máximo y velocidad.**

Aumentar N_{ADC} de 250 a 750 disminuye mínimamente la velocidad máxima, que depende de T_c , pero no importaría modificar ese parámetro. Fijando N_{ADC} en 750 habría que modificar la velocidad de muestreo, como mínimo, a 8.591 Msps, lo que hace que haya que disminuir todavía más la pendiente, hasta un valor de 35.1 MHz/ μ s.

Con esto, se habría modificado la resolución sin apenas modificar los parámetros de velocidad y con un rango de hasta 30 metros, pero no es del todo cierto ya **que esta configuración no puede ser aplicable al radar**, se dan dos errores que hay que solucionar. El primero de ellos tiene que ver con el **tamaño en memoria**, aumentar N_{ADC} a 750 haría que el tamaño en memoria superara los 768 kbits. Ya que el tamaño en memoria sigue la Ecuación 54, sólo se puede **reducir el número de chirps en un frame de 27 a 16, empeorando drásticamente la resolución de velocidad.** Ya se tiene una **configuración válida para el chirp**, cuyos parámetros de configuración y parámetros de entorno se pueden ver en la Tabla 26 y Tabla 27 respectivamente.

Parámetros configurables			
Variable	Unidad	Valor	Configuración
Antenas receptoras (N_{Rx})	-	15 (4)	channelCfg>rxChannelEn
Antenas transmisoras (N_{Tx})	-	7 (3)	channelCfg>txChannelEn
Start frequency (f_0)	MHz	60000	profileCfg>startFreq
Idle time (T_{idle})	μ s	7	profileCfg>idleTime
ADC valid start time (T_{ADC})	μ s	5,7	profileCfg>adcStartTime
Ramp end time (T_{ramp})	μ s	93	profileCfg>rampEndTime
Tx output power (P_t)	dBm	0 (12)	profileCfg>txOutPower
	W	0,02	$P_t(W) = 10^{P_t(dBm)/10}/1000$
frequency slope (S)	MHz/ μ s	35.1	profileCfg>freqSlopeConst
Tx Start time	μ s	1	profileCfg>txStartTime
Number of ADC Samples (N_{ADC})	-	750	profileCfg>numADCsSamples
Sampling Rate ($ADC_{sampling}$)	Msps	8.592	profileCfg>digOutSampleRate
Number of chirp loops (N_c)	-	27	frameCfg>number of loops
Number of frames	-	0 (∞)	frameCfg>number of frames
Frame periodicity	ms	100	frameCfg>frame periodicity

Tabla 26. Parámetros de configuración para obtener un rango máximo de 30 m y una resolución de rango de 5 cm.

Parámetros de entorno		
Variable	Unidad	Valor
Rango máximo	m	30,00
Rango máximo (SNR)	m	50,39
RCS mínimo al rango máximo	m ²	0,126
Resolución de rango	cm	5,00
Velocidad máxima	m/s	4,05
Resolución de velocidad	m/s	0,51

Tabla 27. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m y una resolución de rango de 5 cm.

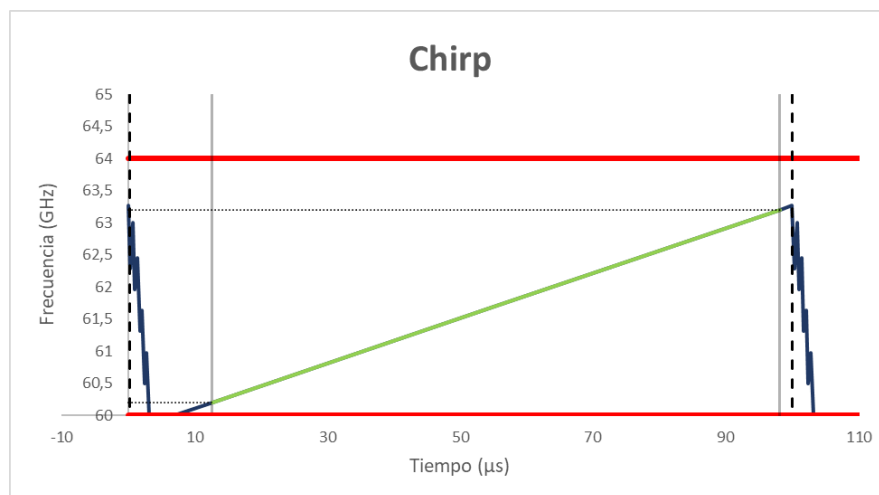


Figura 208. Chirp para obtener un rango máximo de 30 m y una resolución de rango de 5 cm.

Nuevamente se tiene el mismo problema, **el número de muestras es mayor que 128**. Usando la configuración del ensayo 1 (apartado 7.1.1) se consigue una resolución de casi 5 cm. Este ensayo no se ha realizado debido a la dificultad de colocar objetos pequeños separados por 5 cm y hacer que el radar se mueva mientras tanto (ya que esta demo no detecta los objetos estáticos)

7.1.5 Velocidad máxima de 5.5 Km/h o 1.5m/s

En este ensayo se modificará la velocidad máxima para que el **radar no detecte objetos que se muevan a más de 5.5 Km/h** (Ya que la velocidad límite del Rover puede llegar a los 7.2 Km/h). La ecuación de la velocidad máxima es la Ecuación 42. Sustituyendo la Ecuación 43 y la Ecuación 44 en la Ecuación 42, se obtiene:

$$v_{max} = \frac{c}{4N_{Tx}(T_{idle} + T_{ramp})f_c}$$

Ecuación 63

La frecuencia central, se calcula con la Ecuación 39. **Dejando esa ecuación únicamente en función de parámetros configurables, y sustituyendo en la Ecuación 63** se llega a lo siguiente.

$$v_{max} = \frac{c}{4 \left(f_o + T_{ADC}S + S \frac{N_{ADC}}{2ADC_{sampling}} \right) N_{Tx}(T_{idle} + T_{ramp})}$$

Ecuación 64

Cuando solo se tiene que modificar el rango sin tener en cuenta los parámetros de velocidad, el razonamiento es sencillo, ya que se trata de productos y cocientes simples con tres parámetros. Como se puede ver, **la velocidad depende de los mismos parámetros que el rango y alguno más**. Resulta **muy complejo determinar los valores** de los parámetros ya que N_{ADC} , $ADC_{sampling}$ y S como se ha realizado en los ensayos anteriores, para ello es necesario **resolver un sistema de 3 ecuaciones** (Ecuación 58, Ecuación 62 y Ecuación 64) **con 6 incógnitas** ($ADC_{sampling}$, S , N_{ADC} , T_{ADC} , T_{idle} y T_{ramp}).

$$\left\{ \begin{array}{l} Rango_{max} = \frac{0.8 \cdot (ADC_{sampling}) \cdot c}{2S} \\ Range_{res} = \frac{cADC_{sampling}}{2SN_{ADC}} \\ v_{max} = \frac{c}{4 \left(f_o + T_{ADC}S + S \frac{N_{ADC}}{2ADC_{sampling}} \right) N_{Tx} (T_{idle} + T_{ramp})} \end{array} \right.$$

T_{idle} se puede llegar a conocer fácilmente, ya que Texas Instruments recomienda un cierto valor en función del ancho de banda efectivo (B). Por lo que se reduciría el número de incógnitas a 5.

Table 3. Typical Synthesizer Ramp Down Times for Different Modulation Bandwidths

Ramp Bandwidth	Synthesizer Ramp Down Time (µs)
< 1 GHz	2
> 1 GHz and < 2 GHz	3.5
> 2 GHz and < 3 GHz	5
> 3 GHz	7

Figura 209. Valores recomendables del idle time para ciertos anchos de banda [47].

Para resolverlo, hay que **darle valor a 3 de las incógnitas y calcular las restantes**. La herramienta de Texas Instruments, **mmwave Sensing Estimator** realiza cálculos para hallar las incógnitas, dando los valores **idóneos** para conseguir las mejores características y calidad del *chirp* posible. Introduciendo los parámetros de escena requeridos (rango 30 m, resolución de 5 cm y velocidad máxima de 5.5 Km/h), la herramienta devuelve los valores de los parámetros que se pueden ver en la Tabla 28, obteniendo los parámetros de entorno de la Tabla 29.

Parámetros configurables			
Variable	Unidad	Valor	Configuración
Antenas receptoras (N_{Rx})	-	15 (4)	channelCfg>rxChannelEn
Antenas transmisoras (N_{Tx})	-	7 (3)	channelCfg>txChannelEn
Start frequency (f_o)	MHz	60000	profileCfg>startFreq
Idle time (T_{idle})	µs	7	profileCfg>idleTime
ADC valid start time (T_{ADC})	µs	6,4	profileCfg>adcStartTime
Ramp end time (T_{ramp})	µs	258,733	profileCfg>rampEndTime
Tx output power (P_t)	dBm	0 (12)	profileCfg>txOutPower
	W	0,02	$P_t(W) = 10^{P_t(dBm)/10}/1000$
frequency slope (S)	MHz/µs	11,925	profileCfg>freqSlopeConst
Tx Start time	µs	1	profileCfg>txStartTime

Number of ADC Samples (N_{ADC})	-	754	profileCfg>numADCSamples
Sampling Rate ($ADC_{sampling}$)	MspS	3	profileCfg>digOutSampleRate
Number of chirp loops (N_c)	-	6	frameCfg>number of loops
Number of frames	-	0 (∞)	frameCfg>number of frames
Frame periodicity	ms	100	frameCfg>frame periodicity

Tabla 28. Parámetros de configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm y velocidad máxima de 5.5 Km/h.

Parámetros de entorno			
Variable		Unidad	Valor
Rango máximo		m	30,19
Rango máximo (SNR)		m	51,69
RCS mínimo al rango máximo		m ²	0,12
Resolución de rango		cm	5,00
Velocidad máxima		m/s	1,53
Resolución de velocidad		m/s	0,51

Tabla 29. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm y velocidad máxima de 5.5 Km/h.

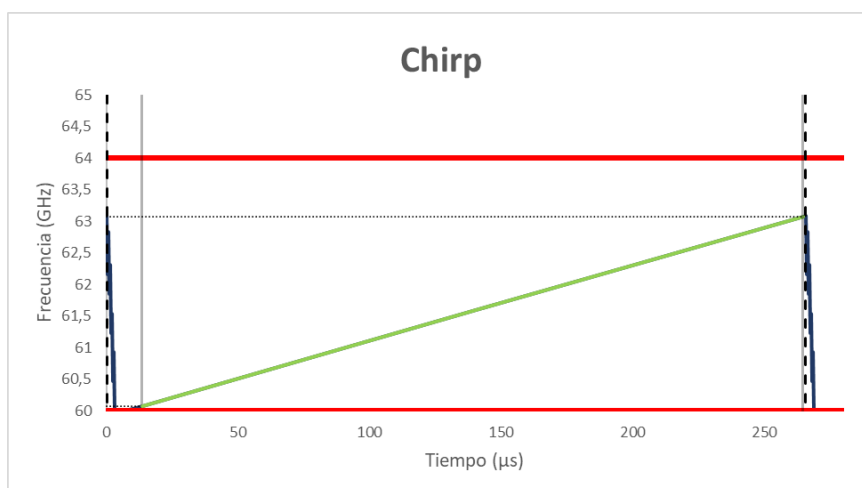


Figura 210. Chirp para obtener un rango máximo de 30 m, una resolución de rango de 5 cm y velocidad máxima de 5.5 Km/h.

Con las muestras en 128, la frecuencia de muestreo en este caso será de 0.5 MspS, que hace una **distancia máxima de 5 metros y no afecta significativamente a la medida de velocidad**, manteniéndose en prácticamente el mismo valor.

Como se le han añadido elementos al Rover (elementos básicos para su funcionamiento) **la velocidad máxima puede no ser la que dice el fabricante**. Por ello, se hará una **comprobación de la velocidad experimental**. El Rover se controlará con un **mando de Xbox** gracias a la aplicación **Mission Planner**. Esta aplicación permite seleccionar el **máximo de velocidad** del Rover mediante un **porcentaje**. Darle el valor de 50% significaría que como máximo puede ir a la mitad de su velocidad máxima, con 25%, a un cuarto de su velocidad máxima, etc.

El Rover **recorrerá una distancia de 10 metros**, pero empezará desde más atrás, para así no tener en cuenta el tiempo mientras está acelerando. Se **medirá el tiempo** que tarda en recorrer esos 10 metros, para

posteriormente obtener los datos de velocidad. Los resultados obtenidos se pueden ver en la Tabla 30, obteniendo la gráfica de la Figura 211.

Velocidades del Rover			
Porcentaje (%)	Distancia (m)	Tiempo (s)	Velocidad (m/s)
0	10	∞	0
30	10	18	0,56
50	10	11,15	0,9
65	10	8,5	1,18
85	10	6,6	1,52
100	10	5,5	1,82

Tabla 30. Velocidades del Rover.

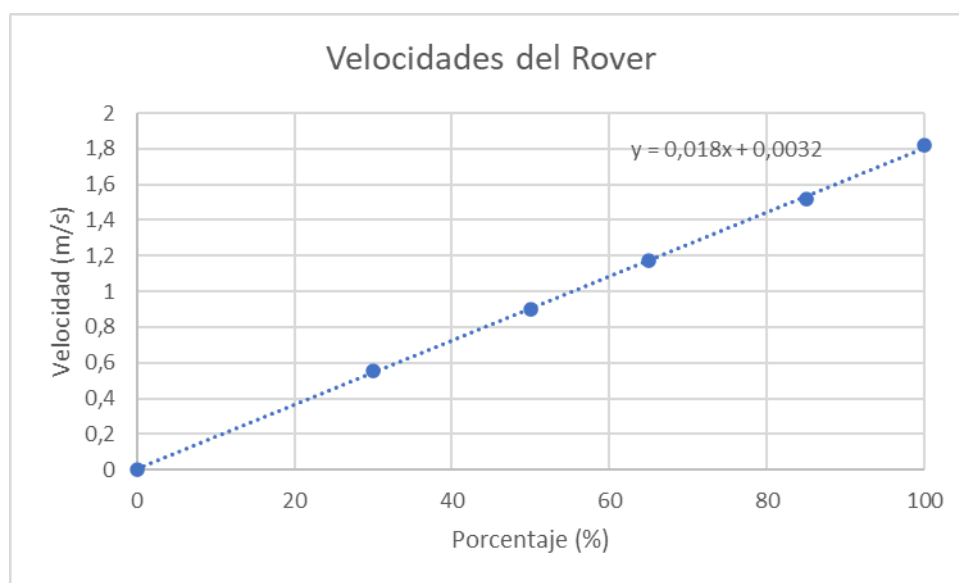


Figura 211. Gráfica de las velocidades del Rover.

Se decidió realizar este experimento ya que se creía que la velocidad del Rover no iba a tener un comportamiento lineal, pero los resultados fueron todo lo contrario, **el comportamiento de la velocidad es muy lineal**, de hecho, la ecuación de la recta es la que se puede ver en la Figura 211 y el coeficiente R^2 es 0.99964, es decir, es un **ajuste casi perfecto**.

La velocidad del Rover al 85% es prácticamente la velocidad **máxima que puede detectar el radar** con esta configuración. El Rover pasará por delante del radar al 100% de su velocidad, donde no debería ser detectado y al 80% de su velocidad, donde sí debería ser detectado.

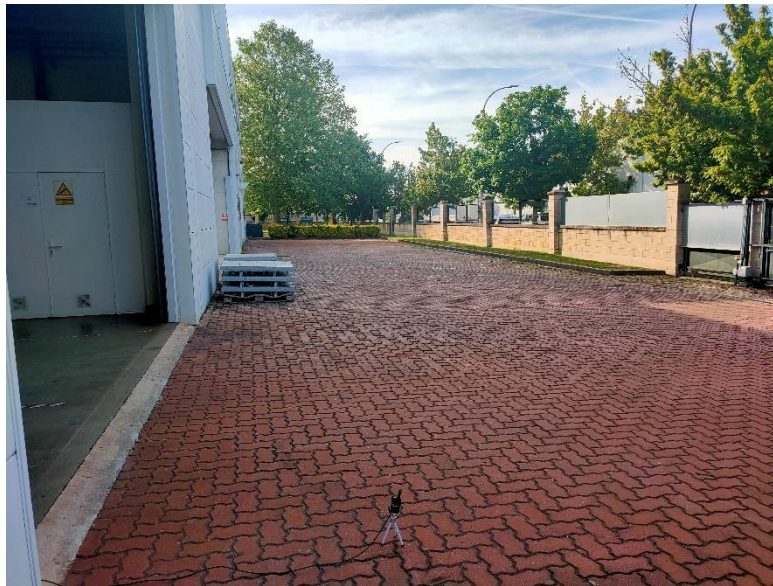


Figura 212. Posición del radar para el ensayo de velocidad.

En este caso el radar se **colocará en el suelo**, para poder detectar al Rover, como se puede ver en la Figura 212. Como también se puede ver, existe la **limitación del cable**, y debido a que **el Rover tiene que pasar bastante cerca del radar**, el espacio es limitado y no siempre alcanzará su velocidad máxima antes de entrar en el rango del radar.

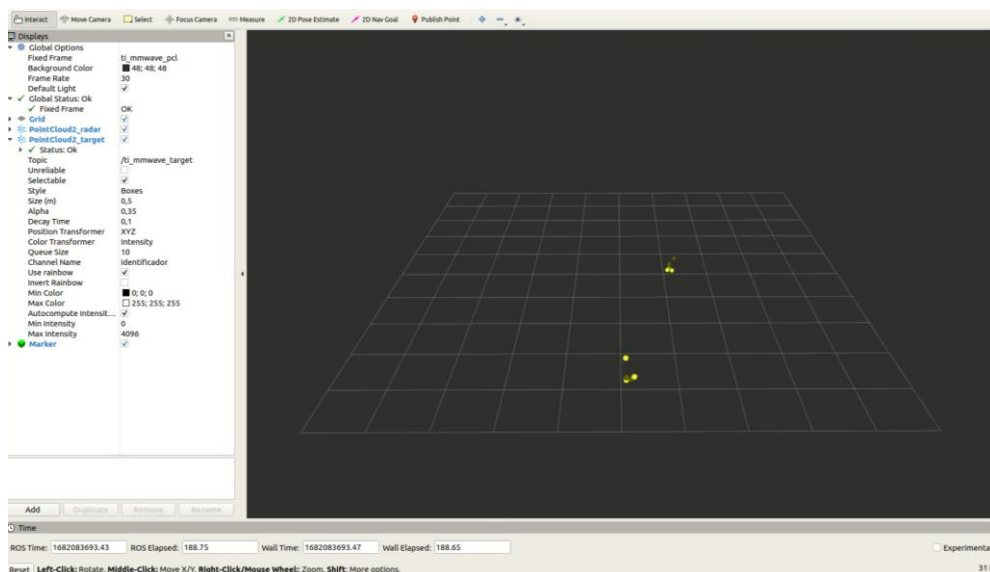


Figura 213. Detección del Rover hasta que su velocidad sobrepasó el límite del radar.

En el caso de la Figura 213, el Rover pasó de **izquierda a derecha** del radar, siendo **detectado hasta el momento que se muestra en la imagen**. Mas allá de este momento no se detectó al Rover. Después se limitó la velocidad del Rover al 80%, y este fue detectado **todas las veces que pasó por delante** del radar.

Después se usó la configuración del apartado *Configuración de corto alcance de Mmwave Sensing Estimator* (apartado 7.1.1) para tener más rango y velocidad máximos para **poder medir más cómodamente la velocidad** del Rover (al 80%, que son 1.44 m/s). Como se puede ver en la Figura 214 y en la Figura 215, **la medida de velocidad es bastante exacta** cuando el Rover iba al 80% su máxima velocidad, tanto si se movía hacia el radar o en la dirección perpendicular.

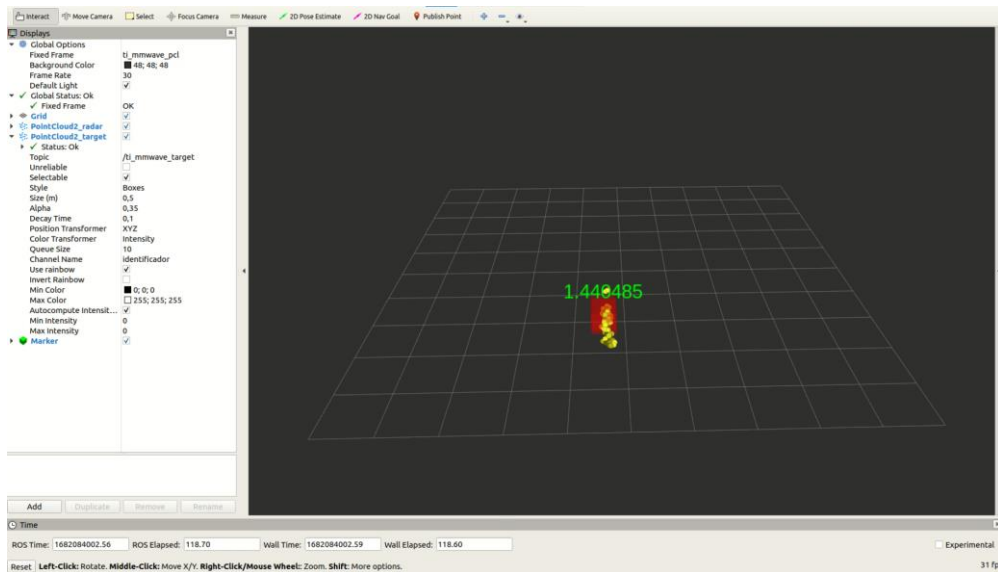


Figura 214. Detección del Rover moviéndose hacia el radar.

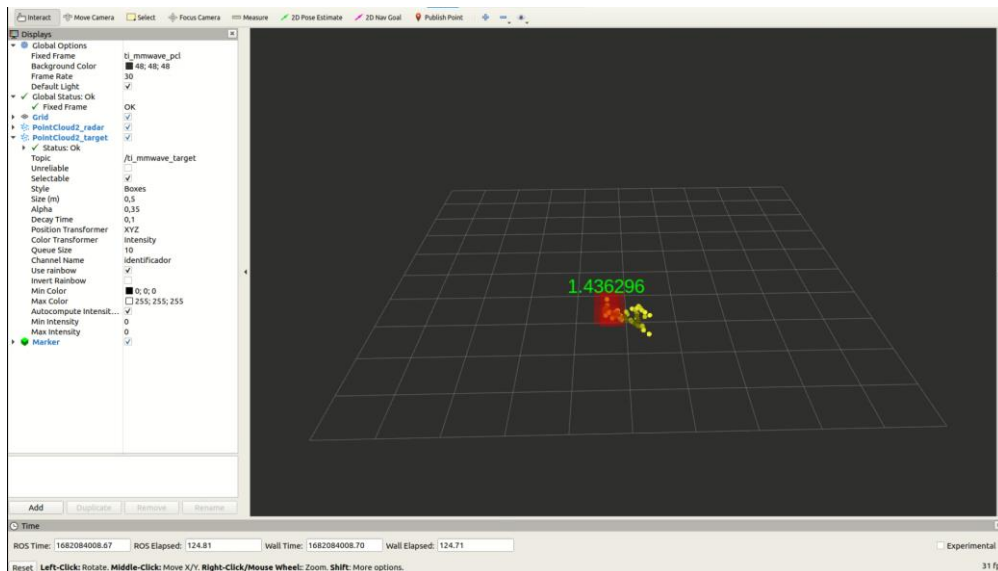


Figura 215. Detección del Rover moviéndose en perpendicular a la dirección del radar.

7.1.6 Resolución de velocidad

La ecuación de la resolución de velocidad (Ecuación 46) se puede reescribir de la siguiente forma, para obtener la resolución de velocidad en función de parámetros configurables.

$$v_{res} = \frac{2v_{max}}{N_c} = \frac{c}{2 \left(f_o + T_{ADC}S + S \frac{N_{ADC}}{2ADC_{sampling}} \right) N_{Tx} (T_{idle} + T_{ramp}) N_c}$$

Ecuación 65

Para la resolución de velocidad, es necesario resolver un **sistema de 4 ecuaciones** (Ecuación 58, Ecuación 62, Ecuación 64 y Ecuación 65) **con 7 incógnitas** ($ADC_{sampling}$, S , N_{ADC} , T_{ADC} , T_{idle} , T_{ramp} y N_c). La resolución de velocidad solo añade una ecuación con una incógnita nueva, por lo que el **sistema de ecuaciones es prácticamente el mismo que para el caso anterior**.

$$\left\{ \begin{array}{l} \text{Rango}_{max} = \frac{0.8 \cdot (ADC_{sampling}) \cdot c}{2S} \\ \text{Range}_{res} = \frac{cADC_{sampling}}{2SN_{ADC}} \\ v_{max} = \frac{c}{4 \left(f_o + T_{ADC}S + S \frac{N_{ADC}}{2ADC_{sampling}} \right) N_{Tx} (T_{idle} + T_{ramp})} \\ v_{res} = \frac{2v_{max}}{N_c} \end{array} \right.$$

Los parámetros se obtendrán de la misma forma que para la velocidad, introduciendo los parámetros de entorno requeridos en mmwave Sensing Estimator para obtener los parámetros del chirp.

Parámetros configurables			
Variable	Unidad	Valor	Configuración
Antenas receptoras (N _{Rx})	-	15 (4)	channelCfg>rxChannelEn
Antenas transmisoras (N _{Tx})	-	7 (3)	channelCfg>txChannelEn
Start frequency (f _o)	MHz	60000	profileCfg>startFreq
Idle time (T _{idle})	µs	7	profileCfg>idleTime
ADC valid start time (T _{ADC})	µs	5.2	profileCfg>adcStartTime
Ramp end time (T _{ramp})	µs	201.602	profileCfg>rampEndTime
Tx output power (P _t)	dBm	0 (12)	profileCfg>txOutPower
	W	0,02	$P_t(W) = 10^{P_t(dBm)/10}/1000$
frequency slope (S)	MHz/µs	15,353	profileCfg>freqSlopeConst
Tx Start time	µs	1	profileCfg>txStartTime
Number of ADC Samples (N _{ADC})	-	750	profileCfg>numADCsSamples
Sampling Rate (ADC _{sampling})	Msp/s	3,838	profileCfg>digOutSampleRate
Number of chirp loops (N _c)	-	15	frameCfg>number of loops
Number of frames	-	0 (∞)	frameCfg>number of frames
Frame periodicity	ms	100	frameCfg>frame periodicity

Tabla 31. Parámetros de configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm, velocidad máxima de 7 Km/h y resolución de velocidad de 1 Km/h.

Parámetros de entorno		
Variable	Unidad	Valor
Rango máximo	m	30,00
Rango máximo (SNR)	m	61,03
RCS mínimo al rango máximo	m ²	0,058
Resolución de rango	cm	5,00
Velocidad máxima	m/s	1,95
Resolución de velocidad	m/s	0,26

Tabla 32. Parámetros de entorno de la configuración para obtener un rango máximo de 30 m, una resolución de rango de 5 cm, velocidad máxima de 7 Km/h y resolución de velocidad de 1 Km/h.

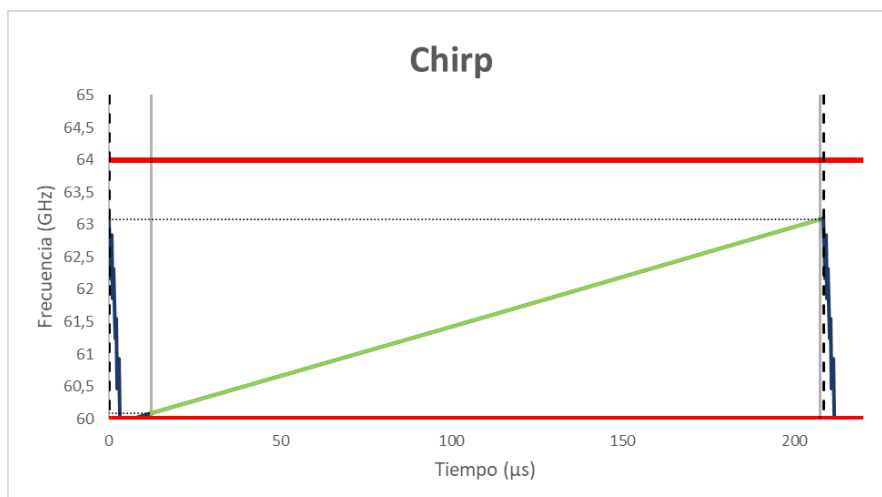


Figura 216. Chirp para obtener un rango máximo de 30 m, una resolución de rango de 5 cm, velocidad máxima de 7 Km/h y resolución de velocidad de 1 Km/h.

Es muy difícil que cuadren todos los parámetros teniendo en cuenta la limitación de la demo en cuanto al número de muestras, por lo que este ensayo no se ha realizado experimentalmente, además de por la dificultad para replicar un caso con velocidades similares.

7.1.7 Comprobación del ángulo máximo

Para determinar el ángulo máximo, una persona se colocó fuera del rango que dice el fabricante (más de 60°) y poco a poco se fue moviendo en la dirección del radar hasta que fuera detectado. En la Figura 217 se puede ver el momento donde el radar empezó a detectar a la persona (a pesar de que hubo algunos puntos sueltos anteriormente)

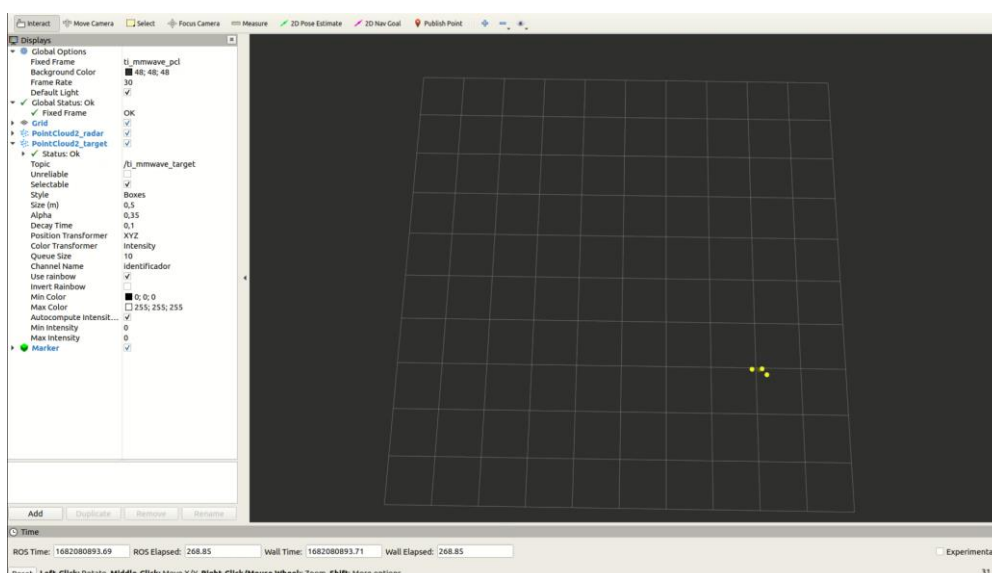


Figura 217. Medición del ángulo máximo del radar.

Como se puede ver, el ángulo de visión es prácticamente de 45° , es decir, un total de 90° , que se queda un poco lejos de los 60° que dice el fabricante. Esto se podría aumentar modificando los parámetros del algoritmo CFAR o la potencia del radar.

7.2 Montaje de los componentes en el Rover

En esta parte se describirán los elementos necesarios y el procedimiento del **montaje de los componentes (Radar, cámara, etc.) en el Rover** para su correcto funcionamiento.

El Rover se puede dividir en dos partes diferentes, la parte **interior**, donde se encuentra todo lo necesario para su funcionamiento, como los motores, y la **tapa** o la parte exterior, que es donde se colocarán la cámara y el radar.

Además de lo esencial para controlar el Rover, en él se deberá introducir la **placa de desarrollo Jetson AGX Xavier**, que irá sujeta a la parte inferior mediante **velcro**. Al estar dentro del Rover, no se le podrá conectar el cable HDMI para verlo en la pantalla como se hacía normalmente. Para poder ver y controlar la placa sin necesidad de estar conectada a la pantalla, será necesario establecer una **conexión remota** desde el ordenador. Para ello se le debe **añadir un módulo WIFI** en la parte inferior, además de **dos antenas** (Figura 218 y Figura 219).

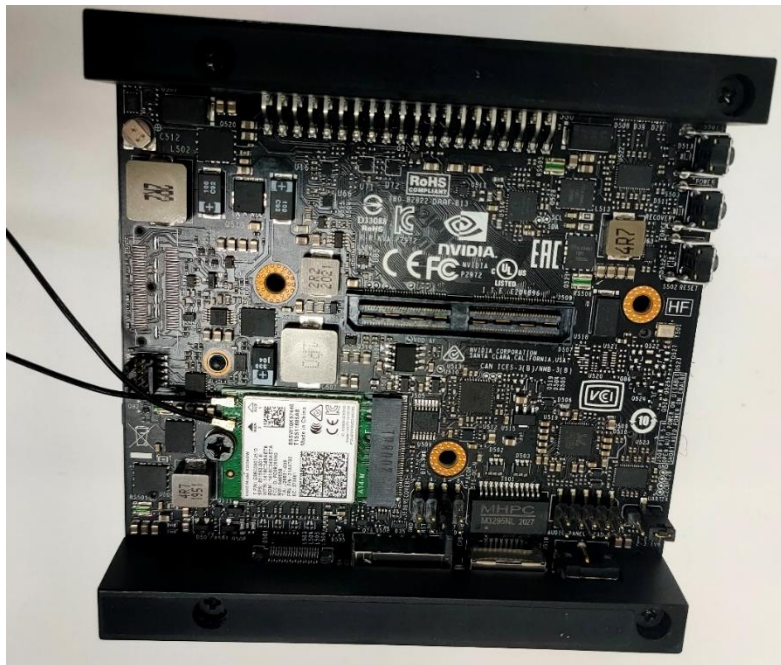


Figura 218. Módulo WIFI añadido en la parte inferior de la placa de desarrollo.

Es posible usar una sola antena, pero es **recomendable el uso de ambas**, ya que la señal de WIFI será mejor. Estas antenas se colocarán de tal manera que **sobresalgan de la carcasa del Rover**, para poder emitir una señal más potente y no ser atenuadas por la carcasa, que, además, es metálica.

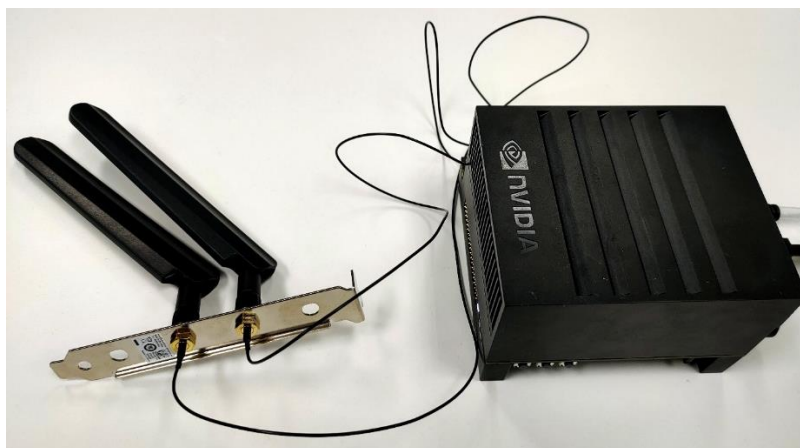


Figura 219. Antenas WIFI de la placa Jetson AGX Xavier Developer Kit.

La placa debe **generar una señal de WIFI**, a la que se conectará el ordenador para que pueda controlarse mediante VNC.

De la misma forma que la placa no puede conectarse a la pantalla mediante el cable HDMI, **tampoco se podrá conectar a la red eléctrica**. Para solventar este problema se hará uso de una **batería**, la cual, mediante un adaptador y regulador, se conectará a la placa de desarrollo para darle la energía.

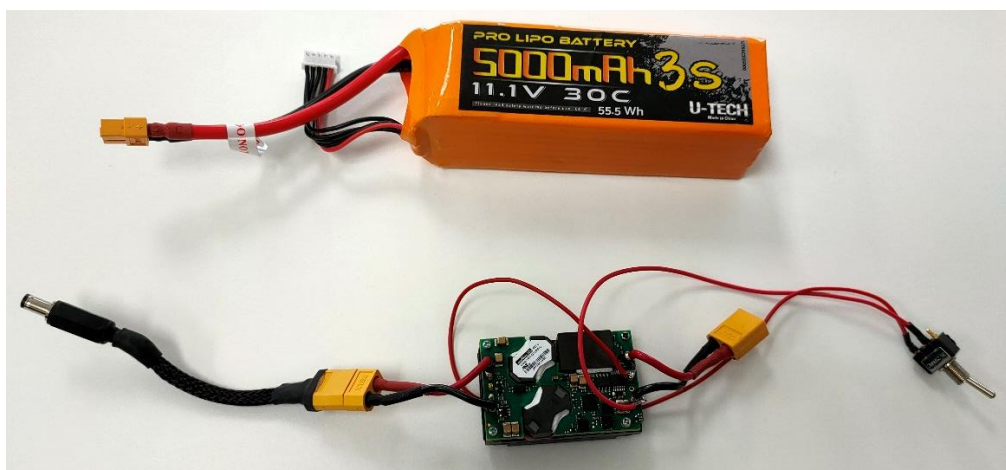


Figura 220. Batería y regulador.

Además de la batería y el módulo WIFI, a la placa se le conectará el **radar** por su único puerto USB y la cámara **ZED 2i** por uno de los puertos tipo C. Todo este montaje se puede ver en la Figura 221.

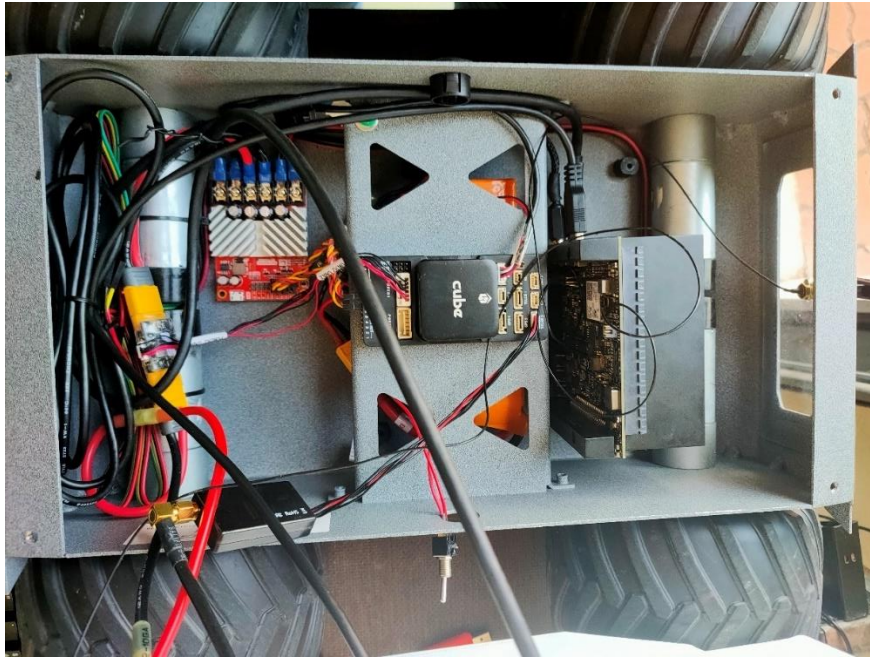


Figura 221. Parte inferior del Rover.

En la parte exterior se montará una estructura con **5 barras de metal** atornilladas a la tapa del Rover y entre sí, sobre la que se colocarán tanto la cámara como el radar. Esta estructura se puede ver en la Figura 222.

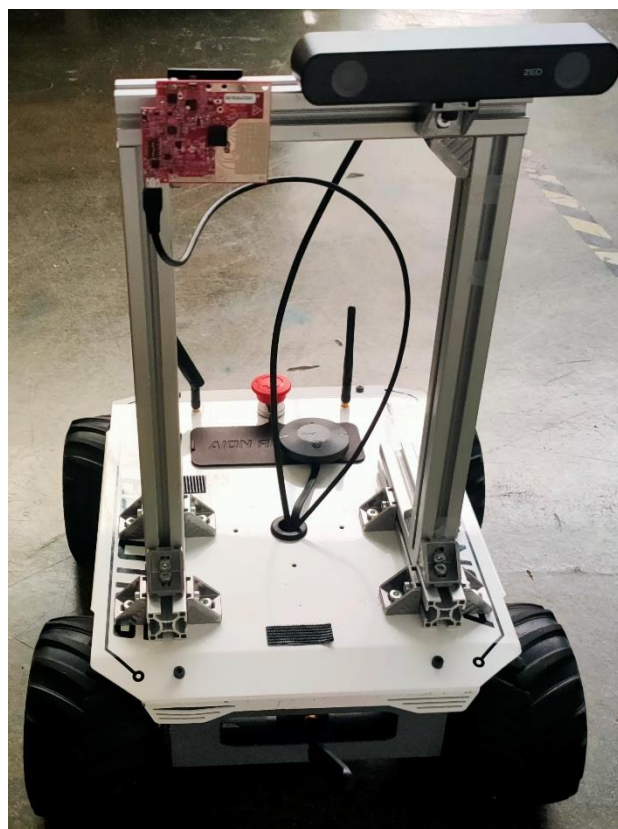


Figura 222. Estructura metálica atornillada al Rover con los elementos colocados.

Los cables de la cámara y el radar se sacarán al exterior mediante el **agujero en el centro de la tapa**. La cámara se colocará en la parte izquierda de la barra superior y el radar en la parte derecha. La distancia entre el radar y la cámara es pequeña, 14 centímetros, pero para una mejor precisión, se puede introducir ese valor en el **archivo .launch** del radar. Hay que trasladar los puntos del radar 0.14 metros en la dirección positiva del eje Y, luego el segundo argumento de la transformada será 0.14 y el resto serán 0, ya que no está girado y la desviación en los ejes X y Z se consideran despreciables.

7.3 Demo 3D_People_Counting

La demo 3D_People_Counting funciona perfectamente cuando el radar se encuentra en una posición **estática**. Una muestra de ello son los ensayos realizados en el apartado 7.1, donde en todos ellos el radar estaba estático.

Tras colocar el radar y la cámara en el Rover, funciona bien cuando se mueve a una **velocidad baja** y con **giros muy suaves**. Cuando se aumenta la velocidad, el radar empieza a detectar **puntos donde realmente no hay objetos**. Esto se puede observar sobre todo cuando el Rover gira, ya que tiene un giro muy brusco y difícil de controlar.

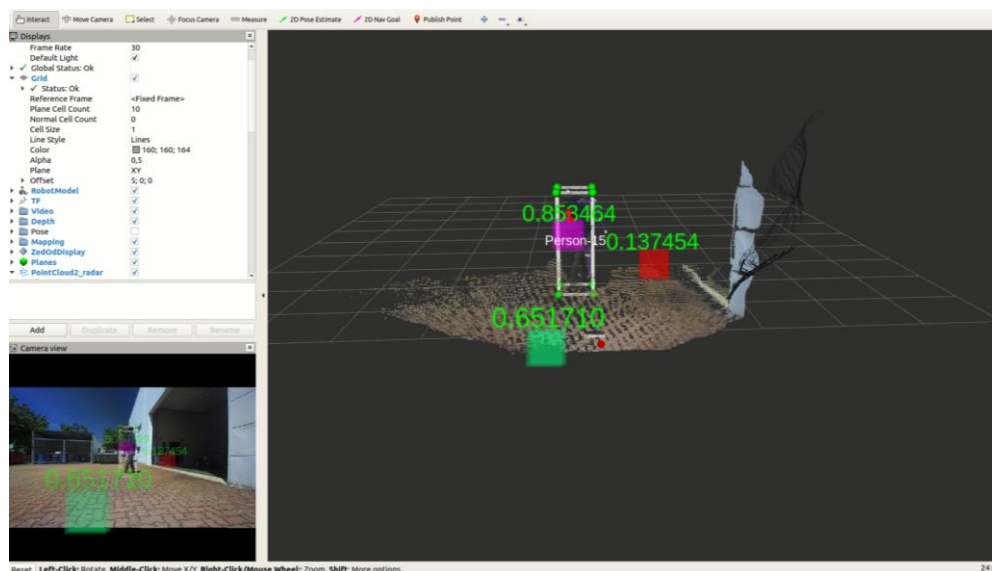


Figura 223. Objetos detectados por el radar y la cámara durante un giro brusco del Rover.

Se colocó un cubo y una persona mientras el Rover se movía en línea recta a una **velocidad reducida**, y se puede ver en la parte inferior izquierda de la Figura 224 que se han detectado ambos objetos, con una precisión de posición aceptable, pero no tanto de velocidad, ya que al solo estar moviéndose el Rover ambos objetos deberían ir a la misma velocidad.

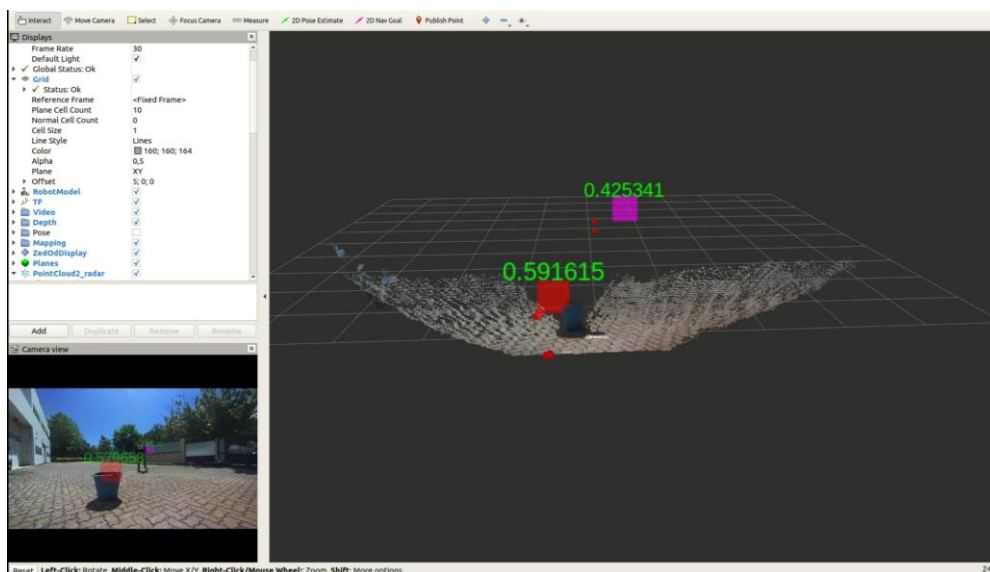


Figura 224. Detección de un objeto y una persona por el radar.

Cabe destacar que la cámara no detecta a la persona porque está algo lejos, pero cuando el Rover se acercó más, tanto el radar como la cámara fueron capaces de detectar a la persona, como se puede ver en la Figura 225.

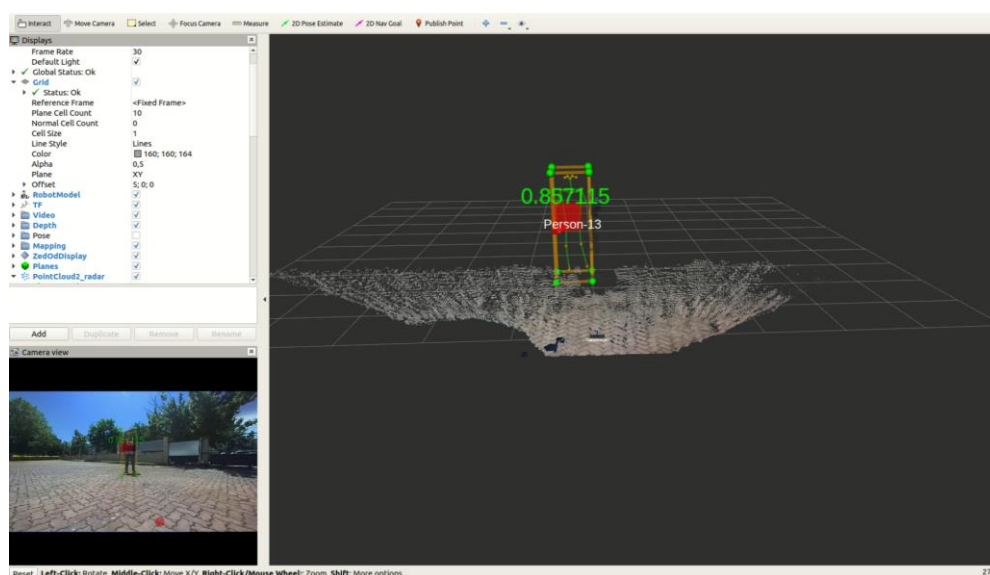


Figura 225. Detección de la persona cuando ya ha entrado en el rango de la cámara.

Como conclusión, esta demo, 3D_People_Counting, funciona perfectamente cuando el radar se encuentra estático, devolviendo la nube de puntos y los targets correctamente. La velocidad también es bastante precisa, sobre todo se puede ver en los ensayos en los que el Rover va al 80% de su velocidad. Al estar en movimiento, funciona bien con movimientos suaves, aunque la velocidad empieza a volverse un poco más inexacta. Al estar en movimiento a aproximadamente 1 m/s los targets son detectados con un poco de desviación de la realidad, pero dentro de un margen aceptable. La velocidad no es nada precisa, ya que cuando el Rover se está moviendo y detecta dos objetos estáticos debería marcar la misma velocidad para ambos objetos, ya que la velocidad relativa al Rover debería ser la misma. Realmente no ocurre eso, la velocidad en ambos objetos es bastante diferente y no es una medida fiable

7.4 Demo Mobile_Tracker

Para esta prueba se colocaron varios objetos y 2 personas. Esta demo **no funciona como se esperaba**. No funciona bien ni cuando el radar está estático ni cuando está en movimiento. Aparecen **múltiples objetos donde no debería haberlos, o no aparecen objetos o personas cuando realmente deberían aparecer**.

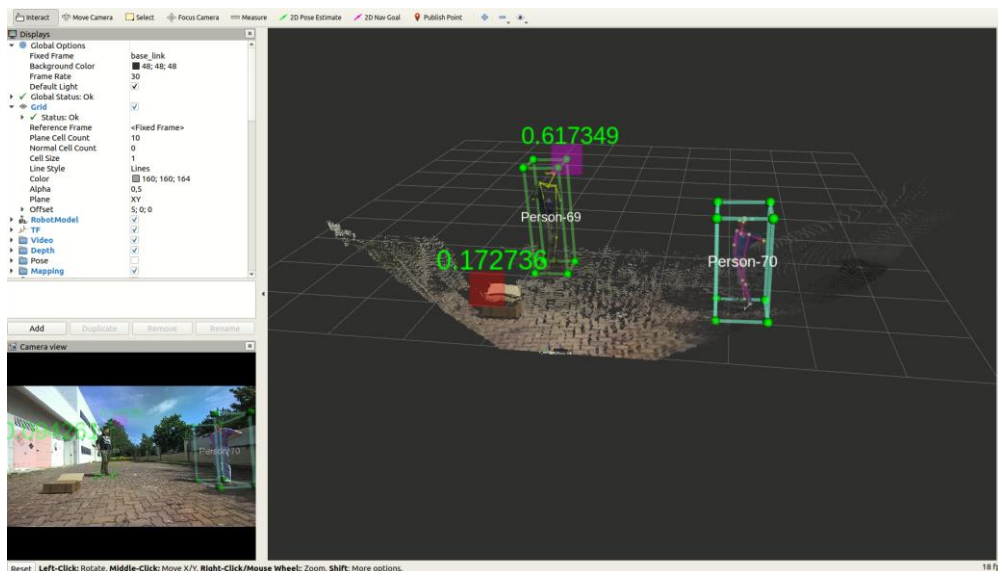


Figura 226. Detección de 3 objetos con Mobile_Tracker.

En la Figura 226, se puede ver el caso en el que **detecta 2 de los 3 objetos o personas** que hay en ese instante. De todo ello, el objeto se detecta bien, el centroide (caja morada) de la persona de la izquierda no está encima suya, y la persona de la derecha no es detectada. Además, las velocidades son muy diferentes.

En la Figura 227 y Figura 228 se puede ver que el radar **no detectó bien ni a la persona ni a la caja**, generando unos cuantos targets donde no debería haber.

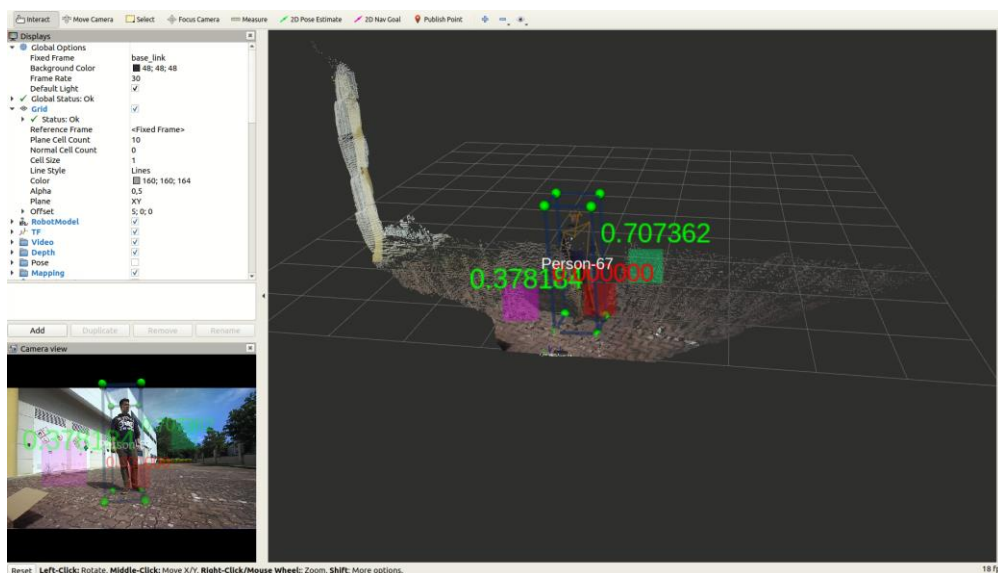


Figura 227. Detección errónea de una persona.

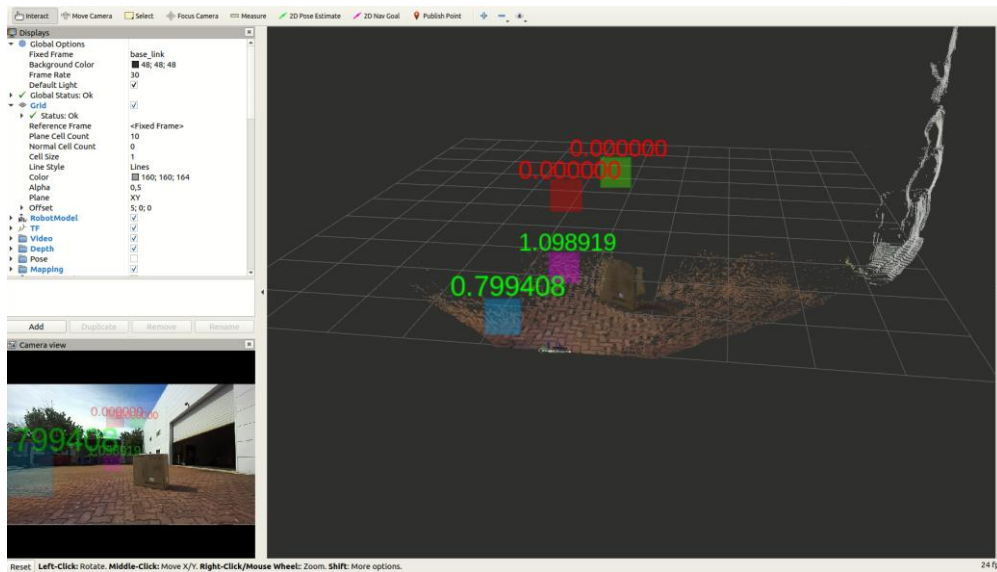


Figura 228. Detección errónea de un objeto.

En resumen, la demo *Mobile_tracker*, **no funciona del todo bien en ningún caso**, ni estático ni en movimiento, que es para lo que debería estar preparada. A pesar de esto, **en estático funciona bastante mejor que en movimiento**, muestra los targets que realmente existen, aunque a veces aparecen objetos fantasmas que no deberían aparecer. Esto podría ser algo normal ya que **esta demo está preparada para que el radar se mueva**, pero con movimientos suaves ocurre lo mismo que en el caso en estático aunque los objetos fantasmas no son tan frecuentes. Con movimientos fuertes empieza a fallar mucho, **la detección de objetos deja de ser fiable** ya que se generan muchos objetos fantasmas. Estos se podrían **reducir mediante software**, ya que algunos de ellos están en lugares que no tiene sentido que estén (por ejemplo, por debajo del suelo), pero no seguiría siendo una detección fiable, ya que algunos *targets* pueden aparecer en la zona de interés. **La velocidad es imprecisa** para todos los modos de operación.

Que haya algunos objetos fantasmas, en algunos casos, puede deberse a que la **pared izquierda sea de metal y a la derecha haya chapas metálicas**, produciendo reflexiones no deseadas, pero los comportamientos atípicos se veían aunque no hubiera elementos metálicos. En la demo *3D_People_Counting* **apenas se veían objetos fantasmas**, en comparación con la demo *Mobile_tracker*. Se cree que esta última está **preparada para movimientos más suaves y objetos más pequeños**, y que ese es el motivo por el cual se ven tantos objetos fantasmas con esa demo.

8 Conclusiones sobre las demos

La **detección de personas y objetos** es muy importante en el ámbito de la industria, evitando colisiones indeseadas. Para ello, la tecnología mmwave es una buena elección, ya que los radares que proporciona Texas Instruments junto con ROS, los hace la **opción ideal** para aplicaciones en las que el radar se implemente en vehículos pesados que se muevan a baja velocidad, ya que, de esta forma, la detección de objetos es bastante precisa.

Para la demo 3D_People_Counting se puede concluir que **en estático funciona perfectamente**, con **movimientos leves empieza a fallar la velocidad** y con **movimientos más fuertes la velocidad falla mucho** y la posición empieza a fallar un poco, pero sigue siendo lo **suficientemente precisa** para ser aceptable. Esto contrasta con lo visto con Mobile_tracker, en la que la detección de objetos y velocidad **no son del todo buenas bien en ningún caso**, ni estático ni en movimiento.

Estos radares al ser **altamente configurables** se pueden utilizar para múltiples aplicaciones, no solo relacionadas con la industria, debido también a la gran versatilidad que proporciona ROS, pudiendo recoger y procesar los datos relevantes para cada aplicación e implementarlo de una forma sencilla con otros sensores.

Uno de esos sensores puede ser la **cámara ZED 2i**, que, al basarse en el tratamiento de imágenes, **funciona perfectamente** tanto en estático como a una velocidad un poco elevada.

9 Investigación y líneas futuras

En primer lugar, se podría **transferir los nodos de ROS a la otra placa de desarrollo, la Nvidia Jetson AGX Orin Developer Kit**. Para ello habría que instalar la versión del *Jetpack* de Nvidia más actual, que hasta la fecha de publicación de este estudio es la **versión 5.1**, junto con otro archivo (USB *firmware*) que busca exclusivamente solucionar los problemas con el puerto serie. Si se comprobara que el problema se ha solucionado, **sería conveniente cambiar de placa de desarrollo**, ya que de esta forma se verá todo mejor, pues la *Nvidia Jetson AGX Orin Developer Kit* tiene más potencia que la *Nvidia Jetson AGX Xavier Developer Kit*.

La cámara publica un *topic* con los 8 puntos de la *boundary-box* (en el caso de que sea 3D). Se podría utilizar esa información para **detectar cuando el centroide proporcionado por el radar se encuentra dentro de esa *boundary-box*** proporcionada por la cámara, para así enviar un **mensaje de aviso** al robot.

En cuanto al **paquete de ROS**, se podría seguir **actualizando y añadiendo TLV** para que **un solo paquete de ROS fuera compatible con todas las demos**, es decir, con todos los TLV. Además, se podría modificar el código del TLV con identificador 1000 para **publicar la nube de puntos que proporciona la demo Mobile_tracker** (y otras demos que devuelvan el mismo TLV). Se podría hacer lo mismo con el TLV con identificador 7 para obtener los datos del SNR y del ruido.

10 Bibliografía

- [1] C. Lovescu y S. Rao, «The fundamentals of millimeter wave radar sensors,» Texas Instruments, Julio 2020. [En línea]. Available: <https://www.ti.com/lit/wp/spyy005a/spyy005a.pdf>. [Último acceso: 6 Febrero 2023].
- [2] R. Jacobi y A. Aginskiy, «Choosing 60-GHz mmWave sensors over 24-GHz to enable smarter industrial applications,» Texas Instruments, Noviembre 2018. [En línea]. Available: <https://www.ti.com/lit/wp/spry328/spry328.pdf>. [Último acceso: 27 Abril 2023].
- [3] K. Ramasubramanian, K. Ramaiah y A. Aginskiy, «Moving from legacy 24GHz to state-of-the-art 77GHz radar,» Texas Instruments, Octubre 2017. [En línea]. Available: <https://www.ti.com/lit/wp/spry312/spry312.pdf>. [Último acceso: 27 Abril 2023].
- [4] Z. Yang y A. Mani, «Interference mitigation for AWR/IWR devices,» Texas Instruments, Enero 2020. [En línea]. Available: <https://www.ti.com/lit/swra662>. [Último acceso: 15 Marzo 2023].
- [5] G. M. Brooker, «Mutual Interference of Millimeter-Wave Radar Systems,» *IEEE Transactions on Electromagnetic Compatibility*, vol. 49, nº 1, pp. 170-181, Febrero 2007.
- [6] D. Barrett y A. Alvarez, «mmWave radar sensors in robotics applications,» Texas Instruments, Octubre 2017. [En línea]. Available: <https://www.ti.com/lit/wp/spry311a/spry311a.pdf>. [Último acceso: 25 Abril 2023].
- [7] M. Malajner, D. Šipoš y D. Gleich, «Design of a Low-Cost Ultra-Wide-Band Radar Platform. Sensors (Basel),» *PMC*, Mayo 2020.
- [8] J. D. Taylor, *Ultrawideband Radar: Applications and Design*, CRC Press, 2012.
- [9] J. Suárez Páez y G. Llano Ramírez, «Revisión del estado del arte de IR-Ultra-Wideband y simulación de la respuesta impulsiva del canal IEEE 802.15.4a,» *SciELO*, vol. 6, nº 11, 2010.
- [10] T. Kaiser y F. Zheng, *Ultra Wideband Systems with MIMO*, John Wiley & Sons, 2010.
- [11] M. Aftanas, *Through wall imaging with UWB radar system*, 2009.
- [12] D. Valderas, *Ultrawideband Antennas: Design and Applications*, Londres: World Scientific, 2011, pp. 1-17.
- [13] J. Pan, «Medical Applications of Ultra-WideBand (UWB),» Abril 2008.
- [14] Y. S. Koo, L. Ren, Y. Wang y A. E. Fathy, «UWB MicroDoppler Radar for human Gait analysis, tracking more than one person, and vital sign detection of moving persons,» *2013 IEEE MTT-S International Microwave Symposium Digest (MTT)*, pp. 1-4, 2013.
- [15] Texas Instruments Inc., «Texas Instruments: What we do,» 2023. [En línea]. Available: <https://www.ti.com/about-ti/company/what-we-do.html>. [Último acceso: 11 Abril 2023].

- [16] Texas Instruments Inc., «Mmwave radar sensors,» 2023. [En línea]. Available: <https://www.ti.com/sensors/mmwave-radar/overview.html>. [Último acceso: 7 Febrero 2023].
- [17] A. Mani, S. Rao, J. Nayyar, M. Yan y B. Johnson, «Introduction to the DSP Subsystem in the IWR6843,» Texas Instruments, 18 Junio 2018. [En línea]. Available: <https://www.ti.com/lit/pdf/swra621>. [Último acceso: 27 Abril 2023].
- [18] Texas Instruments Inc., «Sensors forum: IWR1642: difference between AWR and IWR parts,» [En línea]. Available: <https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/742730/iwr1642-difference-between-awr-and-iwr-parts>. [Último acceso: 23 Febrero 2023].
- [19] Texas Instruments Inc., «Sensors forum: AWR1642: Can't choose between AWR and IWR range,» [En línea]. Available: <https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/601728/awr1642-can-t-choose-between-awr-and-iwr-range?tisearch=e2e-quicksearch&keymatch=awr%20+%20IWR>. [Último acceso: 23 Febrero 2023].
- [20] TME, «Linecard - SEED STUDIO,» 2023. [En línea]. Available: https://www.tme.eu/es/linecard/p,seeed-studio_1325/. [Último acceso: 11 Abril 2023].
- [21] Seeed Studio Inc., [En línea]. Available: <https://www.seeedstudio.com/24GHz-mmWave-Sensor-Human-Static-Presence-Module-Lite-p-5524.html>. [Último acceso: 27 Febrero 2023].
- [22] Seeed Studio Inc., «24 GHz mmWave Sensor - Human Static Presence Module Lite (MR24HPC1),» 2023. [En línea]. Available: https://wiki.seeedstudio.com/Radar_MR24HPC1/. [Último acceso: 27 Febrero 2023].
- [23] Anteral, [En línea]. Available: <https://anteral.com/>. [Último acceso: 11 Abril 2023].
- [24] Anteral, «uRad Industrial v1.0».
- [25] Anteral, «uRad Manual de usuario».
- [26] Texas Instruments Inc., «60GHz mmWave Sensor EVMs,» 2022. [En línea]. Available: <https://www.ti.com/lit/pdf/swru546e>. [Último acceso: 13 Abril 2023].
- [27] Texas Instruments Inc., «IWR6843ISK,» 2023. [En línea]. Available: <https://www.ti.com/product/es-mx/IWR6843ISK/part-details/IWR6843ISK>. [Último acceso: 26 Abril 2023].
- [28] Texas Instruments Inc., «Uniflash,» [En línea]. Available: <https://dev.ti.com/uniflash/#!/>. [Último acceso: 15 Febrero 2023].
- [29] Silicon Laboratories Inc., «CP210x USB to UART Bridge VCP Drivers,» 9 Febrero 2023. [En línea]. Available: <https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers?tab=downloads>.
- [30] Texas Instruments Inc., «3D people counting demo software implementation guide,» Agosto 2022.
- [31] Texas Instruments Inc., «Detection layer parameter tuning guide for the 3D people counting demo,» Marzo 2022.
- [32] Texas Instruments Inc., «Group tracker parameter tuning guide for the 3D people counting demo».

- [33] Texas Instruments Inc., «Tracking radar targets with multiple reflection points,» 16 Febrero 2021.
- [34] Texas Instruments, «Mobile Tracker User Guide,» 12 Diciembre 2022. [En línea]. [Último acceso: 19 Abril 2023].
- [35] Texas Instruments Inc., «mmWave Demo Visualizer,» 14 Julio 2022. [En línea]. Available: https://dev.ti.com/gallery/view/mmwave/mmWave_Demo_Visualizer/ver/3.6.0/. [Último acceso: 15 Febrero 2023].
- [36] Texas Instruments Inc., «Mmwave SDK User guide,» Septiembre 2020.
- [37] Texas Instruments Inc., «TI Sensors Forum: IWR6843: How to change the TX power using the configuration file.,» [En línea]. Available: <https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/945799/iwr6843-how-to-change-the-tx-power-using-the-configuration-file>. [Último acceso: 21 Marzo 2023].
- [38] B. R. Mahafza, Radar system analysis and design using Matlab®, 6000 Broken Sound Parkway NW, Suite 300: Taylor & Francis Group, 2013.
- [39] E. F. Knott, Radar Cross Section Measurements, Nueva York, NY 10003: Van Nostrand Reinhold, 1993.
- [40] V. Semkin, J. Haarla, T. Pairon, C. Slezak, S. Rangan, V. Viikari y C. Oestges, «Analyzing Radar Cross Section Signatures of Diverse Drone Models at mmWave Frequencies,» *IEEE Access*, vol. 8, pp. 48958-48969, 2020.
- [41] V. Borkar, A. Ghosh, R. Singh y N. Chourasia, «Radar Cross-section Measurement Techniques,» *Defence Science Journa*, vol. 60, nº 2, pp. 204-212, 2010.
- [42] Antenna-Theory.com, «Beamwidths and Sidelobe Levels,» 2015. [En línea]. Available: <https://www.antenna-theory.com/basics/radPatDefs.php>. [Último acceso: 11 Mayo 2023].
- [43] Tutorialspoint, «Antenna Theory - Beam Width,» 2023. [En línea]. Available: https://www.tutorialspoint.com/antenna_theory/antenna_theory_beam_width.htm. [Último acceso: 15 Mayo 2023].
- [44] Cisco Systems Inc., «Signal-to-Noise Ratio (SNR) and Wireless Signal Strength,» 2021. [En línea]. Available: [https://documentation.meraki.com/MR/Wi-Fi_Basics_and_Best_Practices/Signal-to-Noise_Ratio_\(SNR\)_and_Wireless_Signal_Strength](https://documentation.meraki.com/MR/Wi-Fi_Basics_and_Best_Practices/Signal-to-Noise_Ratio_(SNR)_and_Wireless_Signal_Strength). [Último acceso: 14 Marzo 2023].
- [45] Texas Instruments Inc., «Sensors Forum: AWR6843AOP: Specification,» [En línea]. Available: <https://e2e.ti.com/support/sensors-group/sensors/f/sensors-forum/1054741/awr6843aop-specification>. [Último acceso: 17 Marzo 2023].
- [46] «Noise Figure,» Wikipedia, 27 Marzo 2023. [En línea]. Available: https://en.wikipedia.org/wiki/Noise_figure. [Último acceso: 8 Abril 2023].
- [47] V. Dham, «Programming chirp parameters in TI radar devices,» 13 Febrero 2020. [En línea]. Available: <https://www.ti.com/lit/pdf/swra553>. [Último acceso: 23 Febrero 2023].

- [48] Texas Instruments Inc., «IWR6843AOP Single-Chirp 60 to 64 GHz mmWave Sensor Antennas-On-Package (AOP),» Julio 2022.
- [49] J. R. M. Fernández, «Revisión de los detectores CFAR de ventana,» *Revista Telemática*, vol. 16, nº 1, pp. 81-100, Enero-Abril 2017.
- [50] R. Amar, M. Alae-Kerahroodi y M. R. B. Shankar, «FMCW-FMCW Interference Analysis in mm-Wave Radars; An indoor case study and validation by measurements,» *IEEE Xplore*, pp. 1-11, 2021.
- [51] Texas Instruments, «Understanding UART Data Output Format,» 2023.
- [52] Nvidia Corporation, «NVIDIA Jetson Orin NX 16GB Module,» 2023. [En línea]. Available: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/>. [Último acceso: 22 Febrero 2023].
- [53] Nvidia Corporation, «Jetson AGX Orin Developer Kit Hardware Layout,» 2023. [En línea]. Available: https://developer.nvidia.com/embedded/learn/jetson-agx-orin-devkit-user-guide/developer_kit_layout.html. [Último acceso: 9 Marzo 2023].
- [54] Nvidia Corporation, «Nvidia forum: Nvidia AGX Orin USB connection issues,» 2023. [En línea]. Available: <https://forums.developer.nvidia.com/t/nvidia-agx-orin-usb-connection-issues/245341/20>. [Último acceso: 18 Abril 2023].
- [55] Nvidia Corporation, «Serie Jetson AGX Xavier,» 2022. [En línea]. Available: <https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-agx-xavier/>. [Último acceso: 15 Marzo 2023].
- [56] «Visual Studio Code,» Wikipedia, 11 Abril 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Visual_Studio_Code. [Último acceso: 17 Abril 2023].
- [57] Open Source Robotics Foundation Inc., «ROS/Introduction,» 8 Agosto 2018. [En línea]. Available: <http://wiki.ros.org/ROS/Introduction>. [Último acceso: 11 Marzo 2023].
- [58] Open Source Robotics Foundation Inc., «msg,» 31 Enero 2023. [En línea]. Available: <http://wiki.ros.org/msg>. [Último acceso: 18 Abril 2023].
- [59] Open Source Robotics Foundation Inc., «ROS/Installation,» 14 Junio 2022. [En línea]. Available: <http://wiki.ros.org/ROS/Installation>. [Último acceso: 22 Febrero 2023].
- [60] Stereolabs Inc., «Getting Started with ROS on Jetson AGX Orin,» [En línea]. Available: <https://www.stereolabs.com/blog/getting-started-with-ros-and-ros2-on-jetson-agx-orin/>. [Último acceso: 22 Febrero 2023].
- [61] «Robot Operating System,» Wikipedia, 28 Febrero 2023. [En línea]. Available: https://es.wikipedia.org/wiki/Robot_Operating_System#rviz. [Último acceso: 17 Abril 2023].
- [62] Open Source Robotics Foundation Inc., «rviz/DisplayTypes/PointCloud,» 8 Enero 2014. [En línea]. Available: <http://wiki.ros.org/rviz/DisplayTypes/PointCloud>. [Último acceso: 10 Abril 2023].
- [63] Stereolabs Inc., «ZED 2i,» [En línea]. Available: <https://www.stereolabs.com/zed-2i/>. [Último acceso: 17 Marzo 2023].

- [64] Stereolabs Inc., «How to Install ZED SDK on Linux,» [En línea]. Available: <https://www.stereolabs.com/docs/installation/linux/>. [Último acceso: 2 Marzo 2023].
- [65] Stereolabs Inc., «Getting Started with ROS and ZED,» [En línea]. Available: <https://www.stereolabs.com/docs/ros/>. [Último acceso: 2 Marzo 2023].
- [66] Stereolabs Inc., «ROS - ZED Node,» 2021. [En línea]. Available: <https://www.stereolabs.com/docs/ros/zed-node/#services>. [Último acceso: 3 Marzo 2023].
- [67] Texas Instruments Inc., «TI mmwave ROS driver user guide,» 2023.
- [68] Open Source Robotics Foundation Inc., «tf2_ros,» 5 Julio 2017. [En línea]. Available: http://wiki.ros.org/tf2_ros. [Último acceso: 20 Marzo 2023].
- [69] Open Source Robotics Foundation Inc., «Setting up your robot using tf,» 1 Abril 2021. [En línea]. Available: <http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF>. [Último acceso: 20 Marzo 2023].
- [70] Aion Robotics, «R6 ArduROS UGV,» 2019. [En línea]. Available: <https://www.aionrobotics.com/r6>. [Último acceso: 8 Abril 2023].
- [71] Aion Robotics, «M6 UGV,» 2022. [En línea]. Available: <https://docs.aionrobotics.com/commercial-vehicles/vehicles/m6-ugv>. [Último acceso: 8 Abril 2023].
- [72] Open Source Robotics Foundation Inc., «ROS Tutorials,» 2 Noviembre 2022. [En línea]. Available: <http://wiki.ros.org/ROS/Tutorials>. [Último acceso: 28 Febrero 2023].
- [73] Open Source Robotics Foundation Inc., «Writing a Simple Publisher and Subscriber (C++),» 18 Julio 2019. [En línea]. Available: <http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29>. [Último acceso: 28 Febrero 2023].
- [74] Open Source Robotics Foundation Inc., «sensor_msgs/PointCloud2 Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/sensor_msgs/html/msg/PointCloud2.html. [Último acceso: 11 Abril 2023].
- [75] Open Source Robotics Foundation Inc., «sensor_msgs/PointField Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/sensor_msgs/html/msg/PointField.html. [Último acceso: 11 Abril 2023].
- [76] Open Source Robotics Foundation Inc., «std_msgs/Header Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/std_msgs/html/msg/Header.html. [Último acceso: 11 Abril 2023].
- [77] Open Source Robotics Foundation Inc., «rviz/DisplayTypes/Marker,» 17 Abril 2021. [En línea]. Available: <http://wiki.ros.org/rviz/DisplayTypes/Marker>. [Último acceso: 19 Abril 2023].
- [78] Open Source Robotics Foundation Inc., «visualization_msgs/Marker Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/visualization_msgs/html/msg/Marker.html. [Último acceso: 19 Abril 2023].

- [79] Open Source Robotics Foundation Inc., «geometry_msgs/Pose Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/geometry_msgs/html/msg/Pose.html. [Último acceso: 19 Abril 2023].
- [80] Open Source Robotics Foundation Inc., «geometry_msgs/Point Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/geometry_msgs/html/msg/Point.html. [Último acceso: 19 Abril 2023].
- [81] Open Source Robotics Foundation Inc., «geometry_msgs/Quaternion Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/geometry_msgs/html/msg/Quaternion.html. [Último acceso: 19 Abril 2023].
- [82] Open Source Robotics Foundation Inc., «geometry_msgs/Vector3 Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/geometry_msgs/html/msg/Vector3.html. [Último acceso: 19 Abril 2023].
- [83] Open Source Robotics Foundation Inc., «std_msgs/ColorRGBA Message,» 2 Marzo 2022. [En línea]. Available: http://docs.ros.org/en/api/std_msgs/html/msg/ColorRGBA.html. [Último acceso: 19 Abril 2023].
- [84] Texas Instruments Inc., «mmWaveSensingEstimator,» [En línea]. Available: <https://dev.ti.com/gallery/view/mmwave/mmWaveSensingEstimator/ver/2.2.2/>. [Último acceso: 24 Mayo 2023].

11 Ecuaciones para el cálculo del RCS de objetos simples

Para el cálculo teórico del RCS de objetos simples se asume que la antena transmisora radia energía en la dirección positiva del eje Z para todos los siguientes casos. Además, se consideran objetos con **reflectividad perfecta**.

11.1 Esfera

El cálculo del RCS de la esfera es de los **más simples** de todos.

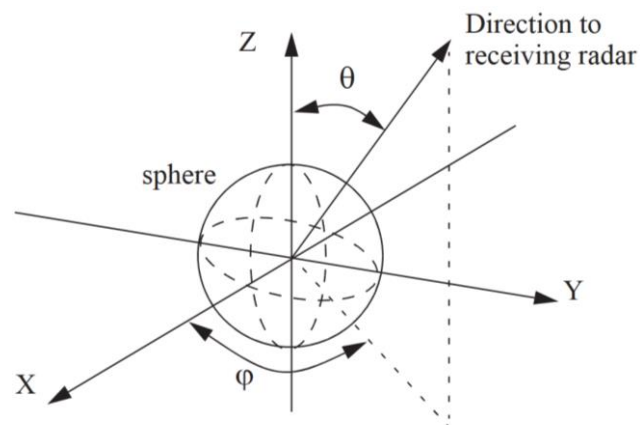


Figura 229. Esfera [38].

Para el caso de la esfera con **radio bastante mayor que la longitud de onda** (Región óptica):

$$\sigma = \pi r^2$$

Ecuación 66

y para el caso donde el **radio sea menor que la longitud de onda** (región Rayleigh):

$$\sigma \approx 9\pi r^2 (kr)^4$$

Ecuación 67

donde $k = 2\pi/\lambda$

11.2 Elipsoide

Un elipsoide centrado en el origen de coordenadas, como se puede ver en la figura, está definido por la Ecuación 68.

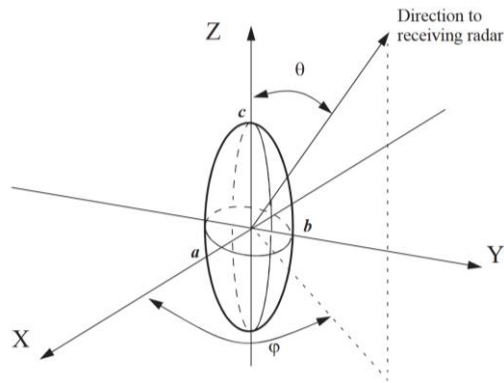


Figura 230. Elipsoide [38].

$$\left(\frac{x}{a}\right)^2 + \left(\frac{y}{b}\right)^2 + \left(\frac{z}{c}\right)^2 = 1$$

Ecuación 68

El RCS del elipsoide donde $a \neq b \neq c$ es:

$$\sigma = \frac{\pi a^2 b^2 c^2}{(a^2 (\sin \theta)^2 (\cos \varphi)^2 + b^2 (\sin \theta)^2 (\sin \varphi)^2 + c^2 (\cos \theta)^2)^2}$$

Ecuación 69

Para el caso donde $a=b \neq c$, el elipsoide se vuelve simétrico, por lo tanto, RCS es **independiente de φ** , y la ecuación anterior se simplifica.

$$\sigma = \frac{\pi b^4 c^2}{(a^2 (\sin \theta)^2 + c^2 (\cos \theta)^2)^2}$$

Ecuación 70

Por último, para el caso donde $a=b=c$:

$$\sigma = \pi c^2$$

Ecuación 71

La Ecuación 71 es la misma que la Ecuación 66, esto se debe a que el elipsoide cuando $a=b=c$, se convierte en una esfera.

11.3 Plato circular plano

En la figura siguiente se muestra un **plato plano circular de radio r**, centrado en el origen.

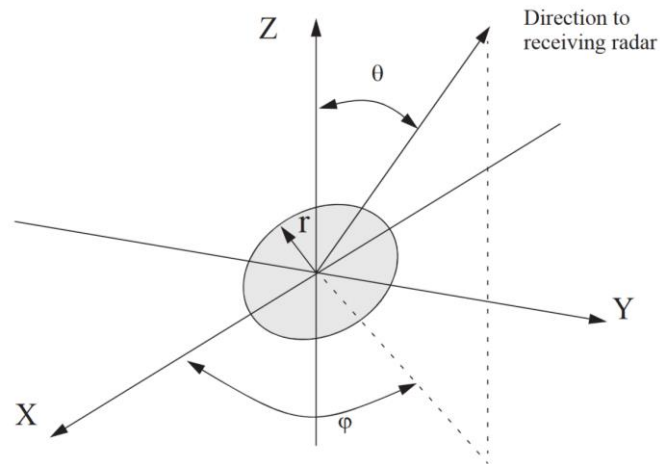


Figura 231. Plato circular plano. [38]

Debido a la geometría circular del plato, el RCS **no depende del ángulo φ** , solo depende del ángulo de incidencia. Para un ángulo de incidencia normal al plato ($\theta = 0^\circ$), la ecuación del RCS es la siguiente.

$$\sigma = \frac{4\pi^3 r^4}{\lambda^2}$$

Ecuación 72

Para un ángulo de incidencia que **no sea perpendicular al plato**, hay **dos aproximaciones** para el cálculo del RCS:

$$\sigma = \frac{\lambda r}{8\pi \sin \theta (\tan \theta)^2}$$

Ecuación 73

$$\sigma = \pi k^2 r^4 \left(\frac{2J_1(2kr \sin \theta)}{2kr \sin \theta} \right)^2 (\cos \theta)^2$$

Ecuación 74

Donde $k = 2\pi/\lambda$, y J_1 es la función Bessel esférica de primer orden.

11.4 Cono truncado (Fustrum)

El ángulo α , **asociado a la geometría** del cono viene dado por la siguiente expresión:

$$\tan \alpha = \frac{r_2 - r_1}{H} = \frac{r_2}{L}$$

Ecuación 75

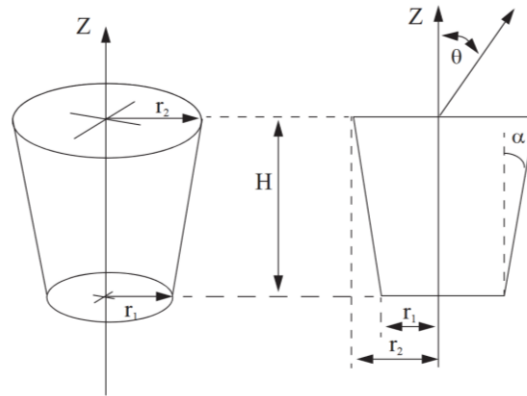


Figura 232. Definición del ángulo de un cono. [38]

Se define el **ángulo normal de incidencia** respecto a la cara del *frustum* como:

$$\theta_n = 90^\circ - \alpha$$

Ecuación 76

Alternativamente, el **ángulo normal de incidencia** ocurre también a

$$\theta_n = 90^\circ + \alpha$$

Ecuación 77

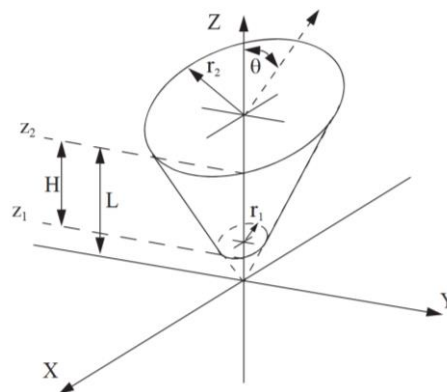


Figura 233. Cono truncado o frustum [38].

En el ángulo normal de incidencia, el RCS es:

$$\sigma_{\theta_n} = \frac{8\pi \left(z_2^{3/2} - z_1^{3/2} \right)^2}{9\lambda \sin \theta_n} \tan \alpha (\sin \theta_n - \cos \theta_n \tan \alpha)^2$$

Ecuación 78

$$\sigma_{\theta_n} = \frac{8\pi \left(z_2^{3/2} - z_1^{3/2} \right)^2}{9\lambda} \frac{\sin \alpha}{(\cos \alpha)^4}$$

Ecuación 79

Para un ángulo de incidencia no normal, el RCS del *frustum* es:

$$\sigma = \frac{\lambda z \tan \alpha}{8\pi \sin \theta} \left(\frac{\sin \theta - \cos \theta \tan \alpha}{\sin \theta \tan \alpha + \cos \theta} \right)^2$$

Ecuación 80

donde z es z_1 o z_2 dependiendo de si la contribución del RCS sea de la parte grande o la parte pequeña. Utilizando identidades trigonométricas, se puede simplificar la ecuación anterior. Para el **lado grande** se obtiene:

$$\sigma = \frac{\lambda z \tan \alpha}{8\pi \sin \theta} (\tan(\theta - \alpha))^2$$

Ecuación 81

Para la **parte pequeña**, la ecuación anterior debe ser modificada de la siguiente forma:

$$\sigma = \frac{\lambda z \tan \alpha}{8\pi \sin \theta} (\tan(\theta + \alpha))^2$$

Ecuación 82

11.5 Cilindro

En la siguiente figura se muestra la geometría asociada a un **cilindro de longitud finita**. Se presentan **2 casos diferentes**. El primero se debe a cuando el cilindro es de **sección elíptica** y cuando es de **sección circular**. El RCS del cilindro de sección elíptica, donde el radio mayor y el radio menor es r_1 y r_2 respectivamente. Cuando la onda incidente es en la dirección normal (dirección del eje z) el RCS es diferente a cuando tiene un cierto ángulo θ .

$$\sigma_{\theta_n} = \frac{2\pi H^2 r_1^2 r_2^2}{\lambda [r_1^2 (\cos \varphi)^2 + r_2^2 (\sin \varphi)^2]^{1.5}}$$

Ecuación 83

$$\sigma = \frac{\lambda r_1^2 r_2^2 \sin \theta}{8\pi (\cos \theta)^2 [r_1^2 (\cos \varphi)^2 + r_2^2 (\sin \varphi)^2]^{1.5}}$$

Ecuación 84

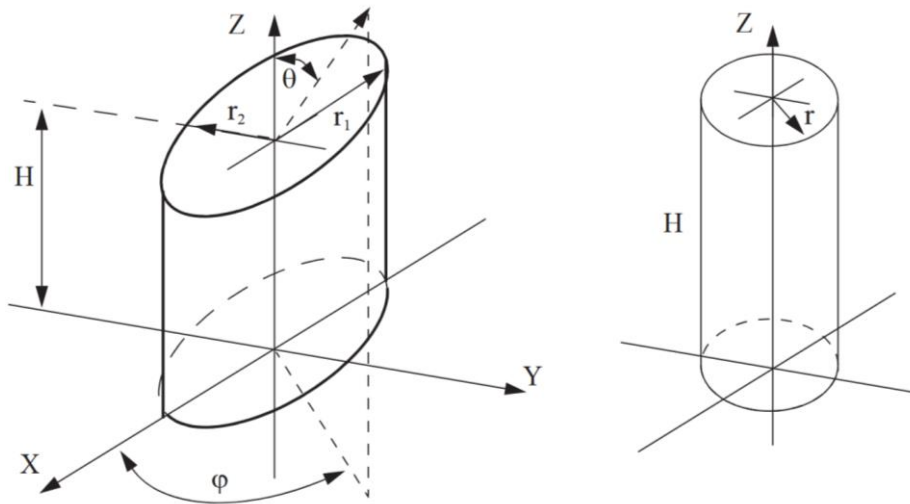


Figura 234. Cilindro de sección elíptica (izquierda) y cilindro de sección circular (derecha). [38]

Para un cilindro de **radio circular**, ya no depende de φ , por lo tanto, las ecuaciones se reducen considerablemente:

$$\sigma_{\theta_n} = \frac{2\pi H^2 r}{\lambda}$$

Ecuación 85

$$\sigma = \frac{\lambda H^2 r \sin \theta}{8\pi (\cos \theta)^2}$$

Ecuación 86

11.6 Plato rectangular plano

El desarrollo teórico para este elemento es **bastante complejo**, por lo que únicamente se muestran la ecuación **para cualquier ángulo θ y φ** (ángulo en el plano XY respecto de X en sentido antihorario)

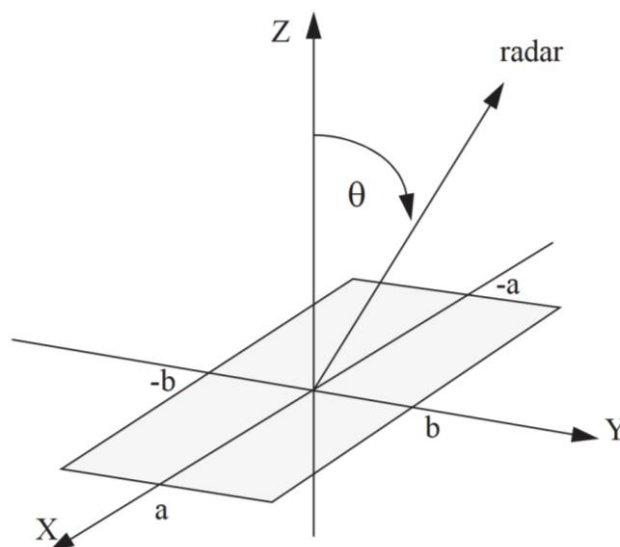


Figura 235. Plato rectangular plano. [38]

$$\sigma = \frac{4\pi a^2 b^2}{\lambda^2} \left(\frac{\sin(ak \sin \theta \cos \varphi) \sin(bk \sin \theta \sin \varphi)}{ak \sin \theta \cos \varphi \quad bk \sin \theta \sin \varphi} \right) (\cos \theta)^2$$

Ecuación 87

11.7 Plato triangular plano

Se considera un plato triangular plano definido por un **triángulo isósceles** como el que se muestra en la siguiente figura. Cabe destacar que este análisis solo es válido para ángulos pequeños, es decir, **para $\theta < 30^\circ$** .

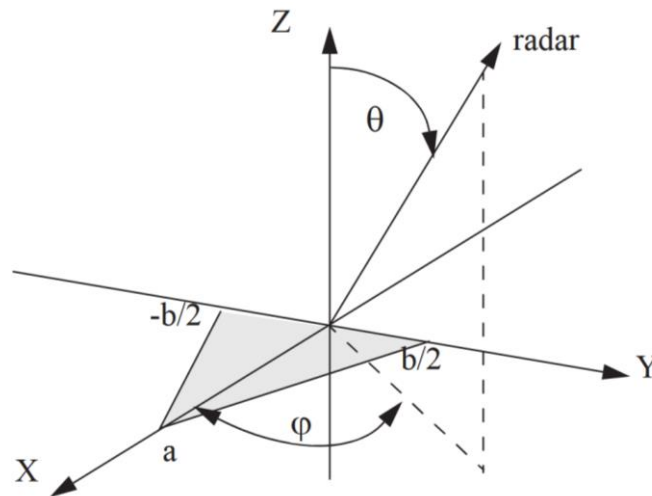


Figura 236. Plato triangular plano. [38]

$$\sigma = \frac{4\pi A^2}{\lambda^2} (\cos \theta)^2 \sigma_0$$

Ecuación 88

$$\sigma_0 = \frac{[(\sin \alpha)^2 - (\sin(\beta/2))^2] l^2 + \sigma_{01}}{\alpha^2 - (\beta/2)^2}$$

Ecuación 89

$$\sigma_{01} = 0.25(\sin \varphi)^2 [(2a/b) \cos \varphi \sin \beta - \sin \varphi \sin 2\alpha]^2$$

Ecuación 90

Donde $\alpha = ka \sin \theta \cos \varphi$, $\beta = kb \sin \theta \sin \varphi$ y $A = ab/2$. Para **ondas incidentes con $\varphi = 0$** , la ecuación se reduce a:

$$\sigma = \frac{4\pi A^2}{\lambda^2} (\cos \theta)^2 \left[\frac{(\sin \alpha)^4}{\alpha^4} + \frac{(\sin 2\alpha - 2\alpha)^2}{4\alpha^4} \right]$$

Y para $\varphi = \pi/2$:

$$\sigma = \frac{4\pi A^2}{\lambda^2} (\cos \theta)^2 \left[\frac{(\sin(\beta/2))^4}{(\beta/2)^4} \right]$$

12 Tablas de contenido de los diferentes TLV

En este apartado del anexo se mostrará el **contenido** de aquellos TLV cuyo contenido se entienda mejor presentándolo en una tabla.

Identificador 1: Detected points		
Valor	Tipo	Bytes
Posición en X (m)	float	4
Posición en Y (m)	float	4
Posición en Z (m)	float	4
Velocidad Doppler (m/s)	float	4

Tabla 33. Estructura de datos del TLV Detected points.

Identificador 6: Statics para XWR6843, XWR1843, XWR1642 y XWR1443		
Valor	Tipo	Bytes
interFrameProcessingTime [µsec]	uint32_t	4
transmitOutputTime[µsec]	uint32_t	4
interFrameProcessingMargin [µsec]	uint32_t	4
interChirpProcessingMargin [µsec]	uint32_t	4
activeFrameCPULoad [%]	uint32_t	4
interFrameCPULoad [%]	uint32_t	4

Tabla 34. Estructura de datos del TLV Statics para XWR6843, XWR1843, XWR1642 y XWR1443.

Identificador 6: Statics para XWRL6432		
Valor	Tipo	Bytes
interFrameProcessingTime [µsec]	uint32_t	4
transmitOutputTime[µsec]	uint32_t	4
powerMeasured[4] (1LSB = 100 uW)	uint16_t[4]	8
tempReading[4] (1LSB = 1°C)	uint16_t[4]	8

Tabla 35. Estructura de datos del TLV Statics para XWRL6432.

Identificador 7: Side info for detected points		
Valor	Tipo	Bytes
SNR [dB]	uint16_t	2
Noise [dB]	uint16_t	2

Tabla 36. Estructura de datos del TLV Side info for detected points.

Identificador 9: Temperature statics		
Valor	Tipo	Bytes
tempReportValid (usado para comprobar que los valores son válidos)	uint32_t	4
time (radarSS local Time from device powerup) [1LSB = 1ms]	uint32_t	4
tmpRx0Sens [1 LSB = 1 deg C]	uint16_t	2
tmpRx1Sens [1 LSB = 1 deg C]	uint16_t	2
tmpRx2Sens [1 LSB = 1 deg C]	uint16_t	2
tmpRx3Sens [1 LSB = 1 deg C]	uint16_t	2
tmpTx0Sens [1 LSB = 1 deg C]	uint16_t	2
tmpTx1Sens [1 LSB = 1 deg C]	uint16_t	2
tmpTx2Sens [1 LSB = 1 deg C]	uint16_t	2
tmpPmSens [1 LSB = 1 deg C]	uint16_t	2
tmpDig0Sens [1 LSB = 1 deg C]	uint16_t	2
tmpDig1Sens [1 LSB = 1 deg C] (No es válido para radares sin DSP)	uint16_t	2

Tabla 37. Estructura de datos del TLV Temperature statics.

Identificador 1000: Spherical coordinates		
Valor	Tipo	Bytes
Range [m]	float	4
Azimuth [rad]	float	4
Elevation [rad]	float	4
Doppler [m/s]	float	4

Tabla 38. Estructura de datos del TLV Spherical coordinates.

Identificador 1010: 3D Target List		
Valor	Tipo	Bytes
tid - Track ID	uint32_t	4
posX [m]	float	4
posY [m]	float	4
posZ [m]	float	4
velX [m/s]	float	4
velY [m/s]	float	4
velZ [m/s]	float	4
accX [m/s ²]	float	4
accY [m/s ²]	float	4
accZ [m/s ²]	float	4
ec[16] - Tracking error covariance matrix	float	64
g - Gating function gain	float	4
confidenceLevel	float	4

Tabla 39. Estructura de datos del TLV 3D Target List.

Identificador 1011: Target index		
Valor	Tipo	Bytes
targetID - Target ID	uint8_t	1

Tabla 40. Estructura de datos del TLV Target index.

Identificador 1012: Target Height		
Valor	Tipo	Bytes
targetID - Target ID	uint32_t	4
maxZ - Target maxZ estimate	float	4
minZ - Target minZ estimate	float	4

Tabla 41. Estructura de datos del TLV Target height.

Identificador 1020: 3D Spherical Compressed Point Cloud: Estructura de la nube de puntos		
Valor	Tipo	Bytes
elevationUnit	float	4
azimuthUnit	float	4
dopplerUnit	float	4
rangeUnit	float	4
snrUnit	float	4

Tabla 42. Estructura de datos del TLV 3D Spherical Compressed Point Cloud: Estructura de la nube de puntos.

Identificador 1020: 3D Spherical Compressed Point Cloud: Estructura de los puntos		
Valor	Tipo	Bytes
elevation	int8	1
azimuth	int8	1
doppler	int16	2
range	uint16	2
snr	uint16	2

Tabla 43. Estructura de datos del TLV 3D Spherical Compressed Point Cloud: Estructura de los puntos.