DISEÑO E IMPLEMENTACIÓN DE SISTEMA DE LOCALIZACIÓN Y MAPEO SIMULTANEO (SLAM) 3D PARA RECONOCIMIENTO DE ENTORNOS MEDIANTE EL USO DE ROBOT MÓVIL

WILLIAM FERNANDO QUINTERO QUINTANA

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA
INGENIERÍA MECATRÓNICA
BUCARAMANGA
2022

DISEÑO E IMPLEMENTACIÓN DE SISTEMA DE LOCALIZACIÓN Y MAPEO SIMULTANEO (SLAM) 3D PARA RECONOCIMIENTO DE ENTORNOS MEDIANTE EL USO DE ROBOT MÓVIL

WILLIAM FERNANDO QUINTERO QUINTANA

TESIS DE GRADO PARA OPTAR AL TÍTULO DE INGENIERO MECATRÓNICO.

DIRECTOR
HERNAN GONZALEZ ACUÑA, PH.D

UNIVERSIDAD AUTÓNOMA DE BUCARAMANGA
FACULTAD DE INGENIERÍA
INGENIERÍA MECATRÓNICA
BUCARAMANGA
2022

DEDICATORIA

A mi familia y mis seres queridos que me han acompañado, a ellos me gustaría dedicar la culminación de mis estudios.

AGRADECIMIENTOS

Agradezco profundamente el apoyo recibido tanto de profesores, como de familiares por permitirme llegar hasta este momento.

RESUMEN

El presente trabajo propone la implementación de un sistema visual de localización y mapeo simultaneo VSLAM 3D, que permitirá generar un mapa de un ambiente desconocido y registrar la ubicación de un robot a medida que este se encuentra explorando una zona desconocida

De la misma forma la exploración de la zona desconocida facilitara a un robot móvil llegar a posiciones en zonas ya visitadas realizando una planificación de trayectorias y retroalimentando su posición por medio del análisis de una secuencia de imágenes para obtener información de la trayectoria realizada y del entorno del robot.
.

PALABRAS CLAVE:
VSLAM 3D

ABSTRACT

The present work proposes the implementation of a simultaneous visual location and mapping system VSLAM 3D, which will allow to generate a map of an unknown environment and record the location of a robot as it is exploring an unknown area.

In the same way, the exploration of the unknown zone will make it easier for a mobile robot to reach positions in already visited areas, carrying out trajectory planning and providing feedback on its position through the analysis of an image trajectory to obtain information on the trajectory made and the environment. of the robot.
.

KEYWORDS:
VSLAM 3D

TABLA DE CONTENIDO

LISTA DE FIGURAS

1 INTRODUCCIÓN

En robótica un elemento importante es la autonomía de los procesos, para ello, cualquier robot requiere información basada en su entorno para completar la tarea la cual se le ha programado. Normalmente se implementan sensores de proximidad que determinen la posición de estos objetos, pero ¿y si fuera posible determinar las proximidades solo con el uso de fotografías?

La visión artificial conjunta una serie de procesos que llevan a la autonomía del ordenador para la toma de decisiones que servirán para: automatizar tareas repetitivas de inspección, realizar inspecciones de objetos sin tener contacto físico, reducir el tiempo de los ciclos de un proceso automatizado, entre otros.

En este aspecto se debe tomar en cuenta dos escenarios que puede tener este robot al realizar sus funciones, las cuales son:

El primer escenario donde se tiene un robot móvil que tiene programado realizar su recorrido, a priori se tiene un mapa del entorno y con este se puede determinar la posición objetivo y también realizar una planificación de la trayectoria del robot para que llegue a esta posición.

En el segundo escenario el robot móvil no conoce la estructura o mapa del lugar, por tal razón inicialmente no es posible seleccionar una posición objetivo para ser alcanzada. Sumado a esta problemática, si el robot móvil está en un lugar de difícil acceso como minas o tuberías de acueducto donde por las condiciones del ambiente no se tiene buena recepción de señal GPS, el robot no estaría en la capacidad de posicionarse con respecto al entorno.

2 OBJETIVOS

2.1 Objetivo general

Implementar un algoritmo de localización y mapeo simultaneo (SLAM) en un robot móvil para la construcción de un mapa de su entorno estructurado.

2.2 Objetivos específicos

- Diseñar el algoritmo SLAM 3D para la creación de un mapa o entorno estático con respecto al robot.
- Determinar la arquitectura de hardware del robot móvil para la implementación del algoritmo SLAM.
- Implementar el algoritmo SLAM en un sistema embebido.
- Implementar una interfaz para la operación del robot móvil para la visualización de los resultados.
- Validar el funcionamiento del algoritmo y determinar cuál es su exactitud.

3 PLANTEAMIENTO DEL PROBLEMA Y JUSTIFICACIÓN

El factor limitante que existe en el empleo de robótica móvil es que el robot debe conocer su entorno para así permitir una planificación de su trayectoria. Esto implica que un robot sea capaz de realizar tareas que, de ser posible de solucionar el problema del mapeo y localización en su entorno de trabajo, mejoraría considerablemente las tareas que este realizará.

El uso de imágenes como única fuente de información y combinarlas con el uso de láser o GPS hace que sea posible la construcción de un modelo que permita la estimación de posición de los objetos cercanos y estimar la localización del propio robot en tiempo real, sumado al control del movimiento de este, hace realidad la creación de una técnica conocida como SLAM.

Los elementos anteriormente mencionados hacen que sea posible la conducción autónoma y se hace uso del mapa para estimar una ruta transitable del robot en un entorno. Sin embargo, cabe a señalar que, en la conducción de un robot, un mapa en 2D es suficiente para cumplir la tarea de navegación con la limitante en especial que los métodos de mapeo basados en láser lleguen a resultados imprecisos debido a la integración de la información del tipo incremental del movimiento a lo largo del tiempo y el corto alcance de estos sensores.

Documentadamente estos problemas en la implementación se ven resueltos al considerar el mapeo RGBD (3D) ya que esto permite información detallada del entorno además de proporcionar imágenes RGB que permite realizar el cálculo de posición basados en la información visual, permite el reconocimiento de lugares, mayor precisión en las estimaciones, entre otras funciones.

Existen sistemas que permiten la planificación de trayectorias por medio de visión artificial, estas poseen alto grado de precisión, pero siempre se ven limitados por el tipo de información a procesar. Siempre se busca la mayor precisión posible ya que vincula gran parte de la problemática de la realidad aumentada y la navegación visual para drones y vehículos autónomos.

Estos campos de la robótica móvil pueden ser conectados mediante técnicas independientes como la localización y mapeo simultáneos (SLAM). Por lo tanto, en este proyecto se propone como solución un consenso entre la funcionalidad de un mapa topológico en dos dimensiones y la utilidad de la información proporcionada por el mapa 3D. Mediante el procesamiento individual de los modelos 3D del entorno estructurado a trabajar.

No existe un enfoque ideal que pueda satisfacer todos o la mayoría de los requisitos posibles por lo que la selección debe considerarse como un conjunto de compromisos aplicados a un espacio de trabajo en específico y la tarea a realizar,

es por ello por lo que se plantea el registro digital del lugar enfocado a un ambiente estructurado.

Para el caso de los sistemas de inspección autónomos, SLAM es el más importante ya que incluye la planificación de rutas y los conceptos básicos del control de movimiento. Estas técnicas de localización se basan en métodos de medición, que proporcionan un bajo costo y posibilita las tareas autónomas.

4 ESTADO DEL ARTE

En **[5]** Se realiza un itinerario de diversas tareas de robótica móvil usando la inteligencia artificial. Se destaca el uso de la inteligencia artificial basada en imágenes como fuente única de información sumado al uso de sensores como láser o GPS. Se concluye que las IA brinda soluciones sólidas para tareas específicas, como la recuperación de información de escenas, mapeo, localización y exploración.

Para el mapeo brindan una solución métrica y una topológica. Se destaca la precisión del primero con respecto al segundo que conducen a una representación gráfica, es decir se limita únicamente a representar un gráfico que contenga ubicaciones de los objetos respectivos. El trabajo se segmenta en presentar las principales herramientas de IA utilizadas en el campo de la robótica, los principales métodos para describir la información (Selección de detectores) y la implementación de la técnica escogida.

Un estudio donde se compara dos metodologías para dar solución al problema de localización y mapeo es presentado en **[6].** En este documento se diseñan dos algoritmos: el primero es basado en la lógica del modelo de filtro Kalman y la segunda emplea un modelo difuso. El objetivo es determinar la precisión de ambas metodologías, teniendo como enfoque principal los sistemas no holonómicos, un caso especial y recurrente en el campo de la robótica.

Finalmente se concluye que el algoritmo basado en la teoría del filtro Kalman muestra resultados más satisfactorios. Sin embargo, se demuestra que la lógica difusa es ampliamente aplicable para la estimación de estados.

En **[7]** se realiza un diseño de un algoritmo de FastSLAM, realizando una adaptación de un algoritmo bidimensional FastSLAM implementado a uno que pueda hacer reconstrucción de entornos tridimensionales. Se utiliza un filtro Bayesiano Binario (uso del teorema de Bayes para extracción de datos basados en la probabilidad). La técnica emplea los datos de trayectoria para estimar un mapa del entorno. Los resultados experimentales muestran que el método propuesto construye un 3D de los alrededores y estima la posición del robot con precisión. Indican que método propuesto se puede aplicar con cualquier sensor de alcance 3D y que en términos de velocidad computacional no pueden procesar en tiempo real debido a limitaciones en las especificaciones de hardware para la actualización del mapa tridimensional.

Otros modelos y aplicaciones de los sensores de visión para dar solución a problemas y una alternativa robusta para el campo de robótica móvil y para capturar información necesaria del entorno donde el robot se mueve en presentada en **[8].** El trabajo se centra en la geometría de los sistemas de visión omnidireccional. Estiman que esta implementación en tareas como las del SLAM es robusta y

computacionalmente factible para la autonomía de robots móviles y que la visión omnidireccional y la robótica móvil son dos áreas activas en donde se espera una evolución rápida en los próximos años.

Un trabajo sobre SLAM 3D en el marco de la semántica visual que consiste en el uso de la información del entorno y el cómo es implementada en la robótica aérea se documenta en **[9].** El método combina odometría visual de bajo nivel con información geométrica correspondiente a superficies planas extraídas de objetos detectados. El trabajo compara el rendimiento en interiores bajo distintas condiciones ambientales. Demuestran que el algoritmo desarrollado es capaz de trabajar en entornos desafiantes y en robots aéreos cuya velocidad sea de 2 m/s.

En **[10]** describen las características clave de los algoritmos que se usan en el SLAM 3D y proponen dos taxonomías para estas técnicas las cuales consisten en clasificación de sensores y clasificación de señales virtuales utilizadas por los algoritmos visuales que son ejecutados por medio de cámara monocular. Sin embargo, se generaron mapas de baja resolución debido a potencia de la computadora y del dispositivo sensorial. Los resultados también muestran que la soluciones no requieren conocimiento previo del ambiente y son ejecutadas en tiempo real.

Múltiples robots para la reconstrucción de un entorno SLAM 3D, es propuesto en **[11].** cada uno equipado con sensor Kinect, que cooperan para explorar el entorno desconocido con posiciones desconocidas. Cada uno de ellos construye un mapa local y ese mapa es compartido entre los robots. Concluye en una mejora en la reducción del ancho de banda y mejora en la eficacia de la computación.

Un sistema de entrega automático por parte de un robot móvil que utiliza filtro de partículas SLAM 3D para completar funciones de posicionamiento y navegación. A través de la plataforma diseñada bajo el lenguaje de C++ que controlan y notifican la transferencia **[12].**

En **[13]** compara el desarrollo de un sistema SLAM 3D basado en un mapa mejorado de un mapa creado por procesos gaussianos, que permiten recuperar superficies, incluidas las de áreas escasamente escaneadas utilizando mediciones aéreas y terrestres, para demostrar un mapeo preciso y odometría robusta. Se obtuvo buena precisión y las muestras distribuidas permiten el registro de mapa en un tiempo real.

La técnica de estéreo fotométrico combinado con SLAM en un proceso automatizado para obtener una escena 3D con el uso de fotografías y múltiples vistas es usada en **[14].** Como resultado obtuvieron escenas reales, pero posee dificultad a la hora de reconstruir escenas oscuras y objetos parcialmente sin textura.

En **[15]** En este artículo proponen mejorar la localización y Mapeo 3D proporcionado por un algoritmo RGBD SLAM, utilizando un conocimiento previo del modelo 3D del entorno. Para mejorar la precisión y la robustez de la localización, proponen usar sensores RGBD que aportan color e imágenes de profundidad, pero su problema es el alcance limitado y sus mapas obtienen resultados con mucho ruido. Por lo tanto, crean un algoritmo que restrinja la localización de la cámara y los puntos de mapeo utilizando un modelo 3D del medio y obtienen resultados precisos reduciendo costos.

5 MARCO TEÓRICO

A continuación, se definen algunos conceptos necesarios para el proyecto.

5.1 Visión por computador.

La visión por computadora es una técnica que emplea el uso de cámaras para simular movimientos a partir de la adquisición de imágenes, estas conllevan a una recopilación de información que puede ser procesada en forma de datos numéricos.

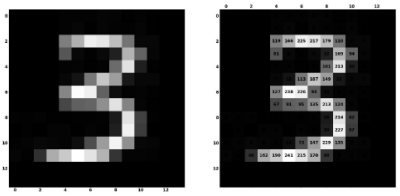


Figura 1. Ejemplo del uso de descriptores para clasificar tonos[1]

En términos de programación, una imagen es un conjunto de números agrupados como si fuera una matriz bidimensional, que contiene valores de 0 a 255 (si no hay color en la imagen), que representan los valores de los píxeles de una imagen. Para que un algoritmo pueda comprender esta información, primero debe procesarse, en [2] presentan tres ejemplos de algoritmos básicos de procesamiento de imágenes basados en lenguaje C++ que son:

Thresholding: Se basa en la creación de un entorno programado que permite al usuario modificar la imagen con el uso de un valor que el usuario establece y cada píxel se convierte a blanco o negro dependiendo del valor establecido.

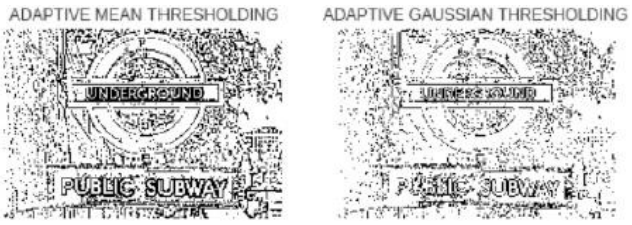


Figura 2. Reconstrucción de imagen por medio de OpenCV[2]

Morphological transformation: Consiste en un conjunto de operaciones que buscan diferenciar los textos de los fondos usando dos imágenes, una que sirve como elemento estructural o guía y otra que decide las operaciones a realizar, las

[1] Imagen extraída de [1]
[2] Imágenes extraídas de [1]

cuales son eliminar el ruido de la imagen y la eliminación de pixeles dentro del objeto identificado.

Figura 3. "Matching" o comparación de características de dos imágenes[3]

Blurring: Emplea un filtro que opera multiplicando los valores de la matriz para eliminar ruidos y bordes realizando un promedio de los pixeles que conforman el área central de la imagen, trabajando con la mediana en lugar del promedio o también reemplazando valores para mejorar la resolución a un sector en específico (método gaussiano).

5.2 Puntos característicos

Figura 4. Extracción de descriptores en una fotografía[4]

Los puntos característicos se utilizan como características que hacen coincidir el mismo objeto desde diferentes perspectivas, esto es indispensable al momento de detectar movimientos y por tanto genera una gran utilidad en el campo de la obtención de datos. Los puntos característicos se componen de dos partes: los Keypoints y Descriptor. Los primeros tienen información de la posición y el segundo es un vector que describe la información de los pixeles del alrededor.

[3] Imágenes extraídas de [1]
[4] Imagen extraída de [4]

5.3 Robótica móvil

Los robots están comprendidos en hardware y software y según estos, se definirá el propósito de su función y siempre trabajan con: un sistema de control, actuadores y sensores. Los sensores se utilizan tanto para obtener información del entorno y también para medir la posición del robot (sensores internos)

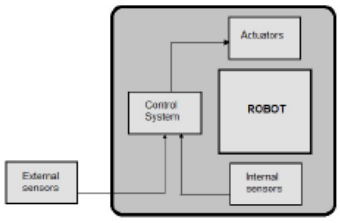


Figura 5. Esquema simple de un robot[5]

5.4 Cinemática

Lo movimientos en robótica se definen de acuerdo con el análisis de la posición inicial y final, geométricamente se analiza las rotaciones y traslaciones correspondientes a este proceso y se representan de forma matricial y este proceso se divide en los pasos:

5.5 Rotaciones y traslaciones:

Se debe tener en cuenta la cantidad de cuerpos conectados en el robot que realicen el movimiento, por ejemplo, se puede llamar A cuerpo fijo conectado al eje fijo y cuerpo B que denotará el cambio de posición con respecto al eje central.

$$\begin{array}{c|ccc} & \hat{a}_1 & \hat{a}_2 & \hat{a}_3 \\ \hat{b}_1 & 1 & 0 & 0 \\ \hat{b}_2 & 0 & c_x & s_x \\ \hat{b}_3 & 0 & -s_x & c_x \end{array} \quad \Rightarrow \left[{}^{B}C^{A}\right] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_x & s_x \\ 0 & -s_x & c_x \end{bmatrix}$$

$$\begin{array}{c|ccc} & \hat{b}_1 & \hat{b}_2 & \hat{b}_3 \\ \hat{c}_1 & c_{y'} & 0 & -s_{y'} \\ \hat{c}_2 & 0 & 1 & 0 \\ \hat{c}_3 & s_{y'} & 0 & c_{y'} \end{array} \quad \Rightarrow \left[{}^{C}C^{B}\right] = \begin{bmatrix} c_{y'} & 0 & -s_{y'} \\ 0 & 1 & 0 \\ s_{y'} & 0 & c_{y'} \end{bmatrix}$$

$$\begin{array}{c|ccc} & \hat{c}_1 & \hat{c}_2 & \hat{c}_3 \\ \hat{d}_1 & c_{z''} & s_{z''} & 0 \\ \hat{d}_2 & -s_{z''} & c_{z''} & 0 \\ \hat{d}_3 & 0 & 0 & 1 \end{array} \quad \Rightarrow \left[{}^{D}C^{C}\right] = \begin{bmatrix} c_{z''} & s_{z''} & 0 \\ -s_{z''} & c_{z''} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figura 6. Representación matricial del movimiento relativo de un cuerpo[6]

La rotación y traslación conllevan a la representación matricial del sistema y el análisis geométrico de realizar dicho movimiento, esta representación creará las

[5] Imagen extraída de [2]
[6] Extraída de [2]

matrices transformadas de rotación y traslación que representarán el movimiento que comúnmente se le conoce como vector r.

5.6 Análisis de velocidades:

La velocidad como es sabido es la razón de cambio en el tiempo que hay sobre una posición, matricialmente se obtiene con anterioridad la matriz de transformación que me define el cambio de posición general, pero para encontrar una razón de cambio que hay entre un cuerpo con otro se tiene en cuenta:

$$ {}^{Q}_{A}v = {}^{P}_{A}v + {}^{Q/P}_{B}v + {}^{B}_{A}\omega \times {}^{Q}_{P}\vec{r} \qquad (1) $$

Donde la velocidad angular es igual a la razón de cambio que hay en cada matriz de transformación

5.7 Análisis de aceleraciones:

El proceso para encontrar las aceleraciones angulares que van de un cuerpo particular B con respecto al eje de referencia se encuentra realizando la derivada de las velocidades angulares o también es posible de forma segmentada, es decir se halla la aceleración del cuerpo inicial A, se suma la a aceleración relativa que hay entre los dos cuerpos y por último se suma esto con el producto cruz de las velocidades relativas.

$$ {}^{Q}_{A}a = {}^{P}_{A}a + {}^{Q/P}_{B}a + {}^{B}_{A}a \times {}^{Q}_{P}\vec{r} + {}^{B}_{A}\omega \times \left({}^{B}_{A}\omega \times {}^{Q}_{P}\vec{r}\right) + 2{}^{B}_{A}\omega \times {}^{Q/P}_{B}v \qquad (2) $$

La aceleración por tal es producto de una serie de operaciones vectoriales entre aceleraciones y velocidades angulares determinadas inicialmente.

5.8 SLAM

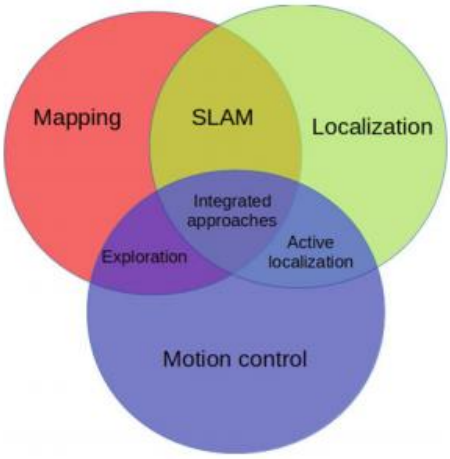


Figura 7. Enfoques de la robótica móvil[7]

SLAM es un subcampo nuevo de la robótica creado para buscar solución práctica al reconocimiento y obtención de información de entornos inaccesibles. Busca obtener información visual con la cual el operador pueda determinar la ubicación de objetos de forma automatizada. SLAM posibilita al usuario de crear mapas en sitios donde la asistencia humana es imposible. Actuaria alejada de métodos como GBS y su aplicación varía desde exploración de zonas de estructuración inestable, navegación en lugares de difícil acceso, zonas espaciales e incluso la exploración de otros planetas.

El modelo de localización se basa en cálculo de distancia y dirección. A través de la información visual, se trabaja desde la perspectiva del robot desde antes de iniciar su movimiento, esto debido a que debe posicionar un objeto para determinar el movimiento del robot con respecto al mismo objeto debido a los cambios de ubicación vistos en este. SLAM siempre trabaja con entornos estáticos por ende el movimiento calculado siempre será el del robot por lo que la información recolectada sería utilizada para dar coordenadas de posición del robot en un tiempo t. Sumado a la posibilidad de trabajar en simultaneo con el mapeo del terreno ya que la información representada de forma vectorial se almacena para la implementación de algoritmos de reconstrucción.

5.9 Mapeo del terreno

El presente trabajo se basa en un diseño e implementación de SLAM para reconstrucción de un mapa 3D. El mapeo del SLAM se genera a través de la

[7] Información extraída de [3]

información visual que se recolecta basada en el posicionamiento de los puntos característicos de la imagen, procesa la información visual en el Frontend del programa para generar un gráfico de proyección el cual es el mapa de disparidad. En el Backend se optimiza esta información para generar la proyección en el plano tridimensional generando una serie de nodos de posicionamiento.

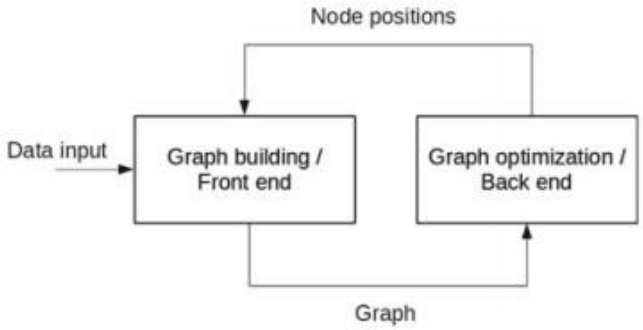


Figura 8. Esquema gráfico de construcción de mapas[8]

El sistema de mapeo incluye representaciones que garantiza un movimiento robusto y seguro del robot en terrenos desconocidos, así como mejoras en el trabajo ya que sus tareas son lo suficientemente precisas como para ser útiles en el trabajo futuro en terreno “explorado”.

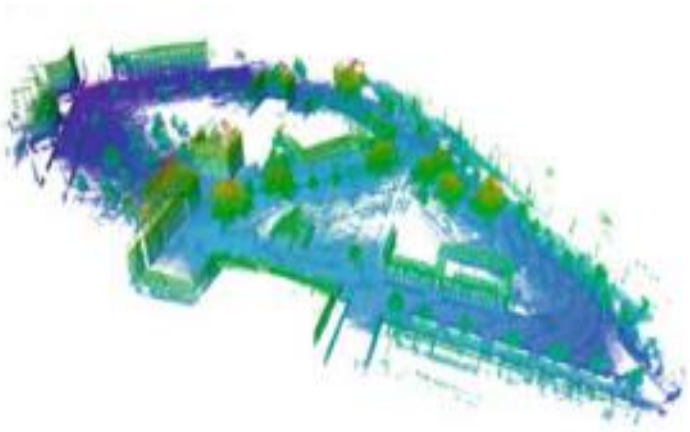


Figura 9. Reconstrucción tridimensional de un entorno topológico [9]

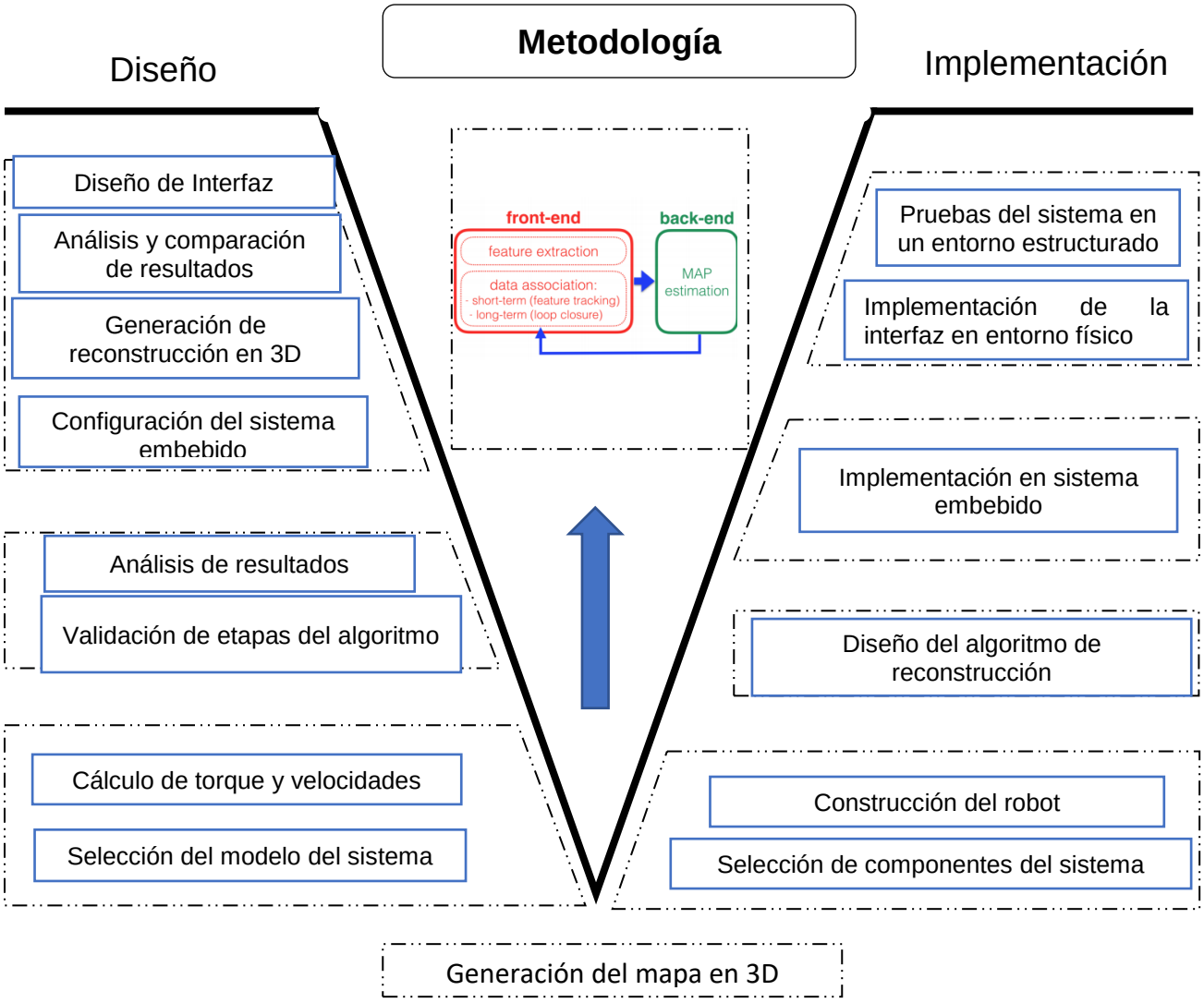
[8] Información extraída de [3]
[9] Información extraída de [33]

6 DISEÑO METODOLÓGICO

El diseño metodólogo está dividido en 4 pasos los cuales son:

1. Se enfoca en el diseño y construcción del robot móvil empleado para hacer la adquisición de información visual.
2. Se basan en el estudio e implementación de las etapas del algoritmo
3. Se basa en el estudio e implementación del algoritmo SLAM en el sistema embebido seleccionado para la generación de un mapa 3D.
4. Se enfoca en validar los resultados obtenidos tanto del diseño como de la implementación.

6.1 Diagrama en V.



7 DISEÑO DEL ROBOT MÓVIL

7.1 Configuración cinemática

Para la selección del robot móvil se compararon 3 configuración cinemáticas que fueron el robot móvil diferencia, el robot móvil tipo triciclo y el robot Ackermann.

7.1.1 Robot diferencial

Un robot de tracción diferencial es aquel que se mueve por acción de dos ruedas motrices independientes para plantear el modelo cinemático, este modelo es práctico y utiliza hipótesis que simplifican las ecuaciones que usa tomando en cuenta que el robot se mueve en una superficie plana.[24]

7.1.1.1 Modelo matemático:

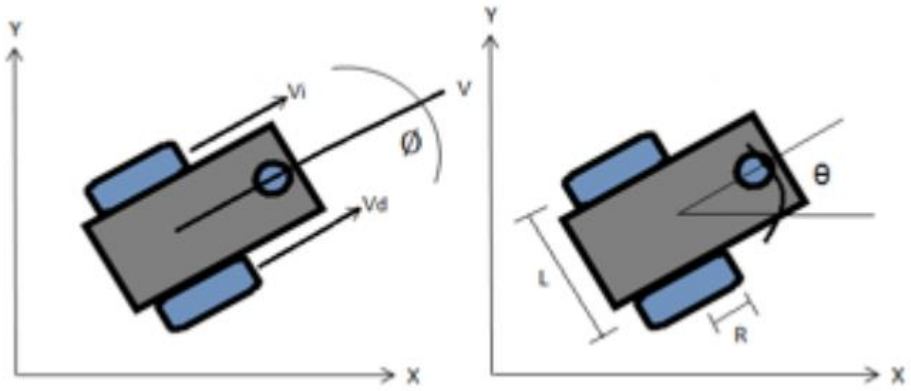


Figura 10. Variables del modelo diferencial

Para las ecuaciones del modelo se toma en cuenta que la velocidad de movimiento es linealmente proporcional al momento generado por las ruedas por lo tanto se puede definir V como el promedio de velocidades de cada rueda por el radio. El ángulo de dirección de la trayectoria puede ser definido por el teorema de Pitágoras como se muestra a continuación:

$$V = R * \frac{v_R + v_i}{2} \quad (3)$$

$$V_x = R * \frac{v_R + v_i}{2} * \cos\theta \quad (4)$$

$$V_x = R * \frac{v_R + v_i}{2} * \sin\theta \quad (5)$$

De esta manera se obtienen las ecuaciones de cinemática para un robot diferencial tomando en cuenta que la relación de (11), (12) y (13) se dan por medio de la integral de la velocidad que es la posición del robot obteniendo:

$$\dot{x} = V * \cos\theta \quad (6)$$

$$\dot{y} = V * \sin\theta \quad (7)$$

El movimiento rotacional considera las magnitudes de las velocidades como diferentes y se define como W la velocidad angular el cual varía según la longitud que separa a las ruedas (L):

$$\dot{\theta} = W = R * \frac{(V_R - V_L)}{L} \quad (8)$$

Como resultado de las ecuaciones se obtiene el modelo cinemático para el robot con un sistema de referencia inercial del robot móvil

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R * \frac{v_R + v_i}{2} \\ R * \frac{(V_R - V_L)}{L} \end{bmatrix} \quad (9)$$

En otras palabras, de usarse el modelo diferencial se trabajaría con los parámetros de las longitudes de las ruedas, la velocidad de cada una de estas y el ángulo de incidencia o rotación del robot.

7.1.2 Robot Ackerman

El robot del tipo Ackerman consta de un modelo que usa cuatro ruedas posicionadas [25]

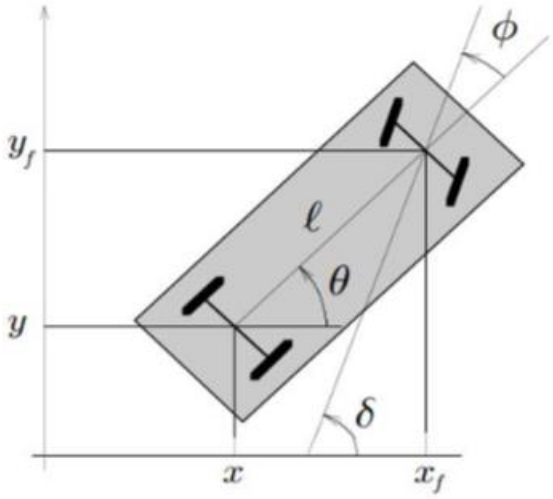


Figura 11. Variables del modelo de Ackerman

El modelo se basa en un sistema de dirección que controla las ruedas delanteras y así determinar el grado de curvatura en la trayectoria del robot, también consta de un sistema de tracción que le proporciona al sistema la velocidad para trasladarse.

Las ecuaciones no distan del todo del modelo diferencial ya que se emplean las ecuaciones: (12), (13) y (16) para el ángulo φ, y (14), (15) con el ángulo θ Sin embargo el modelo de Ackermann toma en cuenta la velocidad que proporciona el conjunto de ruedas traseras trayendo una relación entre estas con el ángulo de incidencia de las ruedas delanteras como se observa en la figura 11.

$$\dot{\theta} = \frac{V_1}{L} * \tan\varphi \tag{10}$$

$$\dot{\varphi} = V_2 \tag{11}$$

$$\dot{x} = V_1 * \cos\theta \tag{12}$$

$$\dot{y} = V_1 * \sin\theta \tag{13}$$

De esta manera tomando en cuenta todos los parámetros y variables se puede resumir el modelo en una ecuación matricial de la siguiente manera:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \\ \dot{\varphi} \end{bmatrix} = \begin{bmatrix} cos\,\theta & 0 \\ \sin\theta & 0 \\ 0 & \tan\varphi \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R * \dfrac{v_R + v_i}{2} \\ R * \dfrac{v_R + v_i}{2L} \end{bmatrix} \quad (14)$$

7.1.3 Robot triciclo

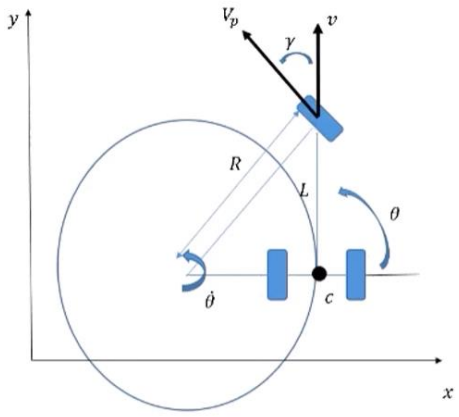


Figura 12. Variables para el modelo de robot triciclo

En este modelo se analiza el movimiento que tiene un robot a partir de un punto de operación C y la incidencia de dos parámetros como lo son el ángulo de inclinación donde v será la velocidad del par de ruedas y Vp la velocidad de la rueda delantera [26]

$$\dot{y} = v * \cos\theta \quad (15)$$

$$\dot{x} = v * \sin\theta \quad (16)$$

$$\dot{\theta} = \frac{V_p}{R} = \frac{v}{R * \cos\gamma} = \frac{v * \sin\gamma}{L * \cos\gamma} = \frac{v}{L} * \tan\gamma \quad (17)$$

$$v = V_p * \sin\gamma \quad (18)$$

La dependencia hacia el ángulo de inclinación de la primera rueda hace que el modelo resultante sea similar al del Ackerman.

Selección del robot: El sistema del robot diferencial posee un único eje donde se operan las dos ruedas activas, permite el giro sobre su propio eje, opera con dos motores. De esta manera, el robot puede moverse en espacios congestionados con cierta facilidad. Usando solo dos ruedas controladas individualmente con una rueda loca como tercer punto de apoyo.

7.2 Selección de la cámara

En esta sección se presentará la comparación de tres cámaras con la finalidad de escoger la que mejor se acople a las necesidades de diseño del tipo de robot seleccionado.

Figura 13. Intel D435i

La tecnología Realsense proporciona datos de precisión ya que están diseñadas para la detección de imagen por estereoscopía (recolección de información visual tridimensional). Este mecanismo integrado permite una óptima implementación ya que poseen dos cámaras las cuales ofrecen, visión de profundidad y visión a color RGB. El sensor de profundidad sumado a la IMU hace que los sistemas operativos robóticos (ROS) puedan obtener coordenadas de posición global y así realizar reconstrucciones densas.

Intel D435i	Intel D455	Intel D415
La resolución RGB es de **1920 x 1080** La resolución Depth es de **1280 x 720.** Velocidad por fotogramas es de **30 fps** Depth y **90 fps** en RGB. Su rango de visión llega hasta los **3 metros**	La resolución RGB es de **1280 x 800** La resolución Depth es de **1280 x 720.** Velocidad por fotogramas es de **90 fps** Depth y **30 fps** en RGB. Su rango de visión llega hasta los **6 metros**	La resolución RGB es de **1920 x 1080** La resolución Depth es de **1280 x 720.** Velocidad por fotogramas es de **30 fps** Depth y **90 fps** en RGB. Su rango de visión llega hasta los **3 metros**

Sus dimensiones son **90x25x25** Precio de **$345.00**	Sus dimensiones son **124x26x29** Precio de **$419.00**	Sus dimensiones son **99x20x23** Precio de **$272.00**

Selección final: Cada una de las cámaras presentadas presentan una ventaja considerable dependiendo de la aplicación que se le otorgue, la principal diferencia entre la 435i y la 415 es que la primera cuenta con un sistema más actualizado que le permite generar un mayor campo de visión (FOV).

7.3 Tarjetas de procesamiento

En esta sección se presentará la comparación la Raspberry pi 4 y la Jetson nano con la finalidad de escoger la que mejor se acople a las necesidades del tipo de entorno programable.

Raspberry Pi 4	Jetson Nano NVIDIA
La Raspberry Pi 4 cuenta con un sistema Quad-Core ARM Cortex de 64 bit a 1.5 GHz, posee conectividad Gigabit Ethernet y modulo Wifi, posee entrada de Micro-SD y una capacidad de memoria RAM de hasta 4 GB, la GPU de video es de **32 bit** y cuenta con un decodificador de video de **1080p60**.	La Jetson Nano de NVIDIA cuenta con un sistema Quad-Core ARM Cortex de 64 bit a 1.4 GHz, posee conectividad Gigabit Ethernet y modulo Wifi, posee entrada de Micro-SD y una capacidad de memoria RAM de hasta 4 GB, la GPU de video es NVIDIA **921 MHz** y cuenta con un decodificador de video a **4kp30**

Figura 14. Ilustración de una Jetson Nano NVIDIA

Selección final: La Jetson Nano posee mayor capacidad de disco y una mejor tarjeta gráfica que le permite decodificar videos a 4kp30. Además, su sistema es compatible con gran parte de librerías del entorno ROS.

Figura 15 Ventilador

Al seleccionar la tarjeta Jetson, es indispensable el uso de un ventilador ya que evita los sobrecalentamientos. Arduino posee puertos compatibles para su conexión ya que son los mismo que los de una CPU. Se debe considerar el peso del ventilador ya que siempre estará puesto encima de la tarjeta y anclado a la base.

7.4 Arduino para control de los motores:

En esta sección se presentará la comparación la tarjeta Arduino con la finalidad de escoger la que mejor se acople a las necesidades del diseño del robot.

Figura 16. Arduino Uno

Arduino Uno	Arduino Mega
El Arduino uno posee un tamaño que le permite el encaje en cualquier placa de prueba en un sistema de aproximadamente 2,7” x 2,1” y tiene una mayor compatibilidad a escudos de	Arduino Mega proporciona un control de bajo nivel que posee medidas de aproximadamente 4” x 2,1” posee una gran cantidad de pines que supera al Arduino micro y al Arduino uno ya que

relés que la Arduino micro, posee 14 pines de E/S digitales donde 6 incluyen modulación por ancho de pulsos PWM y 6 pines de entrada analógica	llegan a ser hasta cerca de 54 pines de E/S digitales con modulación de pulsos PWM y 16 pines de entrada analógica.

Selección final: Las dimensiones del Arduino uno favorece enormemente las necesidades de diseño del robot ya que pese a tener menos pines de entradas analógicas tiene una capacidad de procesamiento y es compatible con la tarjeta Arduino.

7.5 Selección de las ruedas

En esta sección se presentará la comparación de dos tipos de rueda para micromotor con la finalidad de escoger la que mejor se acople a las necesidades del sistema.

Figura 17. Rueda genérica todo terreno micromotor

Rueda terreno plano	Rueda todo terreno
Las ruedas para terreno están diseñadas para cualquier micromotor con shaft del tipo D ya que estos se adaptan directamente a la rueda y su ajuste se da por presión, sus medidas nominales van desde un diámetro de 3.2 cm, ancho de 0.65 cm y son vendidos a par.	Rueda genérica para micromotor con rin plástico y rueda en caucho. Tiene dientes internos en el rin para hacer detección de encoder si se requiere. Esta rueda está diseñada para ser usada con micromotores y se adaptan directamente a la rueda, su ajuste es por presión un diámetro de 4.4 cm, ancho de 1.7 cm

Selección final: Se escoge finalmente la rueda todo terreno debido a la irregularidad de superficies por las cuales se verá obligado a recorrer el robot, su

tamaño y forma dan una resistencia adicional de amortiguamiento lo cual impide grandes vibraciones al momento en el que se ponga en marcha el vehículo.

7.6 Selección de drivers

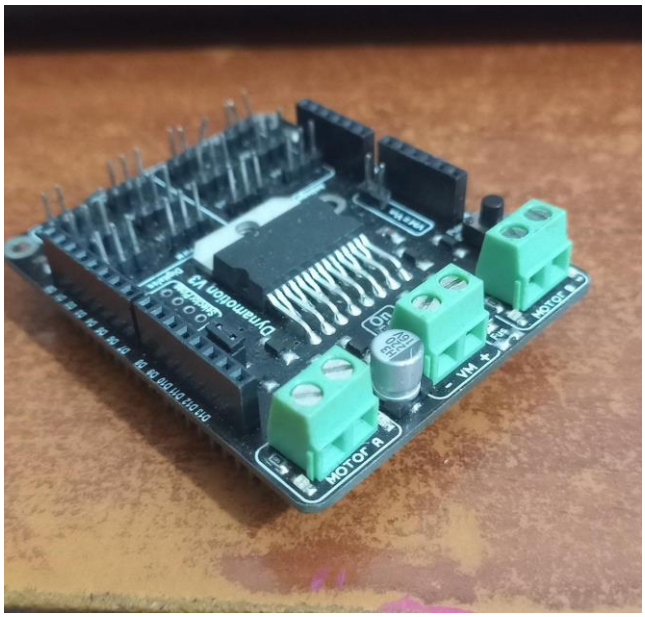


Figura 18. Dynamotion V3

Dynamotion V3	Driver Motor paso a paso TB6600	Puente H L298
La tarjeta para control de los motores cuenta con regletas que permiten su conexión con Arduino y así otorgar un control de la velocidad y dirección de dos motores de forma independiente. Entre sus características cuenta con 28 pines para Arduino, un pulsador 6 puertos digitales y 6 puertos análogos, cuenta con puertos para la conexión de motores con leds que indicarían el estado de su funcionamiento. Cuenta con un voltaje de	Modo seleccionable de micro paso (1/1, 1/2, 1/4, 1/8, 1/16 paso). Control de corriente (en 8 pasos de 0,2 A a 5 A). Fuente de alimentación de 12 ~ 42V DC Protección contra sobre voltaje, bajo voltaje, sobre corriente y cortocircuitos. Alta velocidad de partida. Torque de alta velocidad. Carcasa de plástico negro resistente y disipador de calor de aluminio negro. Orificios de montaje sobre el disipador de	Tensión del módulo de alimentación 2V-10V; Voltaje de entrada de la señal 1.8-7V; Corriente de un solo canal de 1.5A, corriente de pico de hasta 2.5A; Circuito de conducción común incorporado, el extremo de entrada se suspende, el motor no funcionará mal; Circuito integrado de protección contra el sobrecalentamiento con efecto de histéresis (TSD), no hay necesidad de preocuparse por la parada del motor;

alimentación de 7 a 12 V, soporta corriente de carga hasta 1.5 A, 2 salidas de PWM.	calor para el montaje de la unidad en paneles de la máquina.	Tamaño del producto: 24.7 * 21 * 5mm Diámetro del orificio de montaje: 2 mm. Peso: 5g

Selección final: El Dynamotion V3 es compatible con Arduino uno y sus dimensiones tanto en tamaño como en peso coinciden con las dimensiones de la tarjeta lo cual permite unificar ambos componentes de tal manera que no requiere del uso de un circuito adicional además de que no requiere de un disipador de calor.

7.7 Batería

Batería de polímero litio Ion 1000 7.4 V	Batería de polímero litio Ion 3.4 V	Batería LiPo 7.4 V
Esta batería posee tecnología LiPO, que permite implementación en aplicaciones de alta descarga de energía, tiene un valor nominal de 7.4 V a 25 C, posee un paquete de 2 celdas de 1000 mAh, peso aproximado de 85 g, y dos conectores del tipo JST-XH para la carga y JST-RCY para la descarga, sus dimensiones son 70x35x18 mm.	Es una batería de tecnología de polímero Ion-Litio, la cual integra protecciones contra sobre voltaje y sobre corriente, posee un valor nominal de hasta 3,7 V y 2000 mAh a 2C, peso aproximado de 45 g y un conector del tipo JST-PH, sus dimensiones son 4,5x55,72 mm.	Una batería de poco consumo que tiene capacidad de 1000 mAh cuyo valor nominal es de 7.4 V a 25 C conector del tipo JST-XH peso aproximado de 60 g, dimensiones de 72x35x12. Posee una descarga máxima de 10 segundos a 50 C.

Figura 19. Batería de polímero litio Ion 1000 7.4 V

Selección final: La batería de 7.4V Ion 1000 posee mayor capacidad de carga rápida. Mayor vida útil del ciclo, casi el doble que la tecnología estándar de lipo.

7.8 Selección del módulo de comunicación inalámbrico:

Figura 20. Módulo Bluetooth HC05

Al no requerirse distancias muy largas, el módulo Bluetooth es ideal para establecer la comunicación puente entre la interfaz del usuario con el Arduino encargado de programar el motor. El módulo HC05 permite una fácil configuración y es compatible con Arduino.

7.9 Selección de Motores

7.9.1 Requerimientos de torque y velocidad:

En esta sección se tomará en cuenta cada uno de los elementos seleccionados y su peso para hallar el peso del robot.

Cámara: 8.5 oz = 0.24 kg

Arduino: 25 g = 0.025 kg
Puente H: 30 g = 0.03 kg
Raspberry: 7.8 oz = 0.22 kg
Plataforma: 19 g = 0.019 kg
Separadores y tornillos: 12 g = 0.012 kg
Ruedas=0.08 kg
Motores: 20 g = 0.020 kg
Módulo HC05 = 0.0036 kg
Batería: 85 g = 0.085 kg
Ventilador: 68 g = 0.068 kg

Carga Total [kg]: 0.637 kg

Diámetro de las ruedas: 4cm
Velocidad deseada: 0.5m/s
Cantidad de motores: 2

7.9.2 Cálculo del torque

$$Torque = Fuerza\ x\ Distancia = 0.637\ * 4 = 2.548\ kg * cm \quad (19)$$

7.9.3 Cálculo de la velocidad

$$Perímetro = 2 * pi * radio = 2 * 3.1416 * 2 = 12.57\ cm \quad (20)$$

El perímetro es la distancia que recorre en dar una vuelta por lo que la velocidad será el cociente entre el espacio con el tiempo por tanto al trabajarse con RPM los motores se hace la conversión de la velocidad deseada.

$$0.5\ m/(1\ min/60) = \ 30m/min \quad (21)$$

Si en el caso particular la rueda necesita 12.57 cm para dar una revolución se puede obtener la cantidad de revoluciones por minuto que se requiere en metros

$$\frac{30m}{0.1257} = 238.66\ RPM \quad (22)$$

Con esto se concluye que se requiere un torque constante de 2.5 kg*cm y una velocidad de 239 RPM lo cual al tener en cuenta que son dos motores se puede distribuir uniformemente estos valores dando cerca de un torque stall de 1 kg*cm y velocidades de hasta 120 RPM lo que hace que los micromotores de 100:1 y 50:1 se ajusten a las necesidades siendo el de 50:1 el más económico.[10]

Figura 21. Micromotor HP 100:1

El Micromotor de 100:1 es un motor de alta potencia que cuenta con valor nominal de 6 V, pesa 9.5 g, su velocidad sin carga llega a los 120 rpm y 40 mA, sus dimensiones son 26x10x12 mm un torque stall de 9 kg-cm

Figura 22. Micromotor HP 50:1

El Micromotor de 50:1 es un motor de alta potencia que cuenta con valor nominal de 6 V, pesa 0.34 oz, su velocidad sin carga llega a los 6.25 rpm y 100 mA, sus dimensiones son 24x10x12 mm torque stall de 15 oz-in 1.0 kg-cm

[10] Información e imágenes extraídas de https://dynamoelectronics.com/

Figura 23. Motor reductor 70:1

El Micromotor reductor de 70:1 es un motor de alta potencia que cuenta con valor nominal de 12 V, pesa 205 g, su velocidad sin carga llega a los 73 rpm y 150 mA, sus dimensiones son 37Dx70L mm puede soportar hasta 27 kg*cm

Selección: Para seleccionar un motor DC se debe tener en cuenta los requerimientos de diseño que va desde el voltaje de alimentación nominal al cual se le debe aplicar al motor el cual depende de la batería que nominalmente otorgan desde 5 a 7 V, la velocidad sin carga que representa la velocidad deseada, el torque stall que representa la fuerza máxima del motor antes de llegar a 0 RPM (fuerza límite antes de dañarse). En este caso se seleccionó el motor de 50:1 según los parámetros analizados.

7.10 Diseño de la estructura del robot

Con base a una elección minuciosa de cada componente se optó por utilizar la cámara D435i, la tarjeta Jetson Nano, un Arduino y la tarjeta de dynamotion V3 basados en una relación de tamaño, costo y funcionamiento óptimo de cada componente. El modelo del robot a implementar es el diferencial debido a la cantidad reducida de variables a controlar que posee en comparación de los otros dos modelos que incluyen entre otros factores el costeo de uno o dos motores más para su adaptación.

7.11 Ensamble

En esta sección se presenta el proceso de ensamble del robot y la integración de todos sus componentes seleccionados. Para la base se usó una plataforma de acrílico trasparente, la cual será la que soportará el peso de todos los demás componentes, adherido a esta estarán instalados los motores (dos en este caso) seleccionados los cuales estarán pegados a las ruedas.

Para la instalación de los componentes programables se hizo uso de 10 tornillos y 10 tuercas pequeñas de 4 mm, separadores, 4 soportes, se instalaron tanto la tarjeta de Arduino y el puente h de tal forma que este sea el encargado de regular la potencia suministrada por la batería.

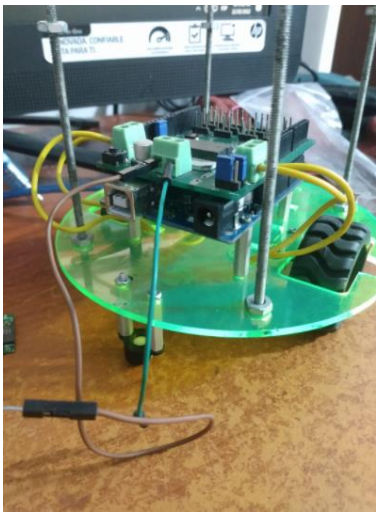


Figura 24. Ensamble de los componentes del robot

Una vez comprobado el funcionamiento correcto de los motores al ser conectados a una fuente a 5 V y el ajuste a la base, finalmente se realiza las conexiones de los cables de los motores con el puente H.

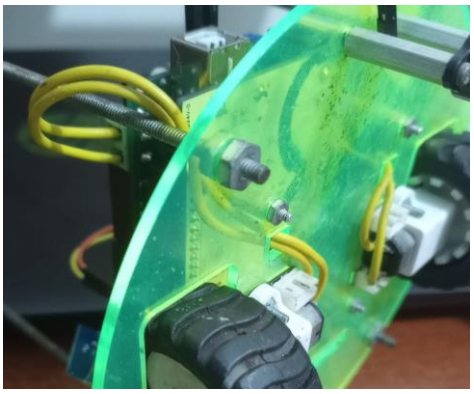


Figura 25. Conexión de los motores con la tarjeta

Con el uso de fresadora y usando una L pequeña de dos aberturas se decidió adaptar el tamaño de los orificios en base a la apertura que posee la Intel D34i (6mm) la cual se proyecta instalarse sobre la L fijada, los tornillos fueron recortados con segueta y afilados con esmeril para su encaje correspondiente, con el uso de hombre solo se ajusta finalmente el elemento en la cubierta que servirá de soporte para la cámara que se desea implementar.

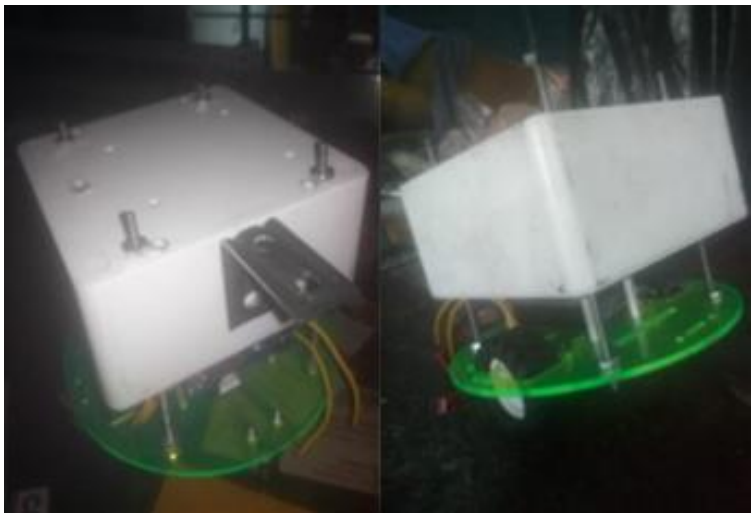


Figura 26. Soporte para la cámara y sistema embebido

Se instaló una rueda pequeña en la parte de adelante que servirá en el carrito como apoyo para estabilizar el vehículo que hasta el momento tenía solo dos ruedas. Dichos elementos son ajustados con tuercas en base a evitar conflictos evidentes de movilidad y así permitir que el robot se pueda trasladar sin problemas.

Se verifica el funcionamiento de los motores conectándolos a una fuente de alimentación a 5V, permitiendo el giro en ambos sentidos. Una vez determinado esto se realiza una prueba conectando las tarjetas Arduino Uno y Dynamotion V3. Los pines a los cuales están conectados los motores A y B, siendo el primero el que se ubica al lado derecho visto desde el pin de conexión con la fuente de alimentación, y el segundo siendo el motor de la izquierda, son dados por medio de la hoja de datos del driver en donde se definen:

D5: Control del Motor A en un sentido contrario a D10
D10: Control del Motor A en un sentido contrario a D5
D6: Control del Motor B en un sentido contrario a D9
D9: Control del Motor B en un sentido contrario a D6

Para esta ocasión no será necesario la implementación de un Joystick por lo tanto no se definirán más pines en la programación. El movimiento del robot permite la adquisición de imágenes secuenciales. Elemento fundamental en el procesamiento.

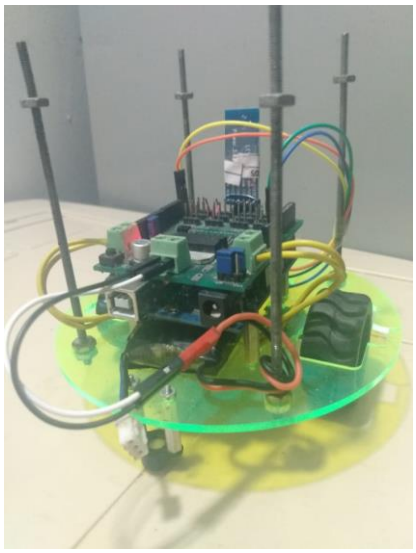


Figura 27. Robot parcialmente ensamblado

Esta configuración permite que sea apto para las pruebas de movimiento, permitiendo así que se puedan hacer con seguridad pruebas de seguridad para hacer nivel de la velocidad con la cual arrancaría cada motor, trabajan conectadas por cable USB y el diseño permite que sea sencillo el desmontar o montar la cámara junto al sistema embebido.

Figura 28. Prototipo completamente ensamblado

Para este ensamble se tomó en cuenta el uso de tuercas y tornillos que permitiesen la fijación de la tarjeta Jetson y a su vez la cámara la cual estará en una máxima altura permitiendo así tener una vista panorámica de su entorno.

8 DISEÑO DEL ALGORITMO

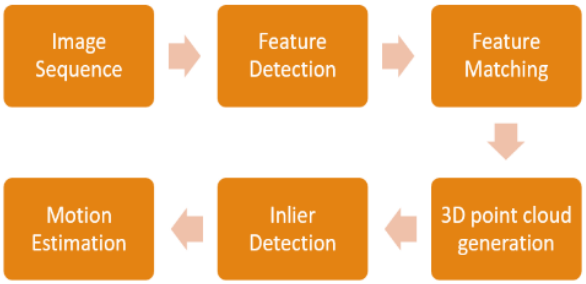


Figura 29. Pasos para la elaboración del algoritmo de odometría

El algoritmo de SLAM se divide en 6 pasos los cuales son: Lectura de imagen secuenciales **(Image sequence),** detección de características entre dos imágenes **(Feature detection),** comparación entre características de ambas imágenes **(Feature Matching),** generación de nube de puntos en 3D a través de triangulación **(3D point cloud generation),** Detección de características entre un par de imágenes comparadas con otro par de imágenes comparadas **(Inlier detection)** y extracción del movimiento **(Motion Estimation).**[11]

8.1 Extracción de puntos característicos

Cuando se habla de "características" se habla de información que determina el contenido de una imagen. Estas características ayudan a determinar patrones y ofrecen una ayuda para encontrar un mismo objeto.

Este proceso se divide en dos: la detección y comparación de puntos característicos. La descripción asignada a un punto debe concordar con la asignación de otro punto. Las propiedades indispensables de la técnica deben ser: invariante a la rotación, invariante a la escala, debido a que el ángulo y movimiento varía y no es factible que la técnica deje de cuando alguno de estos aspectos varía.

[11] https://github.com/cgarg92/Stereo-visual-odometry

	Corner Detector	Blob Detector	Rotation Invariant	Scale Invariant	Affine Invariant	Repeatability	Localization Accuracy	Robustness	Efficiency
Haris	x		x			+++	+++	++	++
Shi-Tomasi	x		x			+++	+++	++	++
FAST	x		x	x		++	++	++	++++
SIFT		x	x	x	x	+++	++	+++	+
SURF		x	x	x	x	+++	++	++	++
CENSURE		x	x	x	x	+++	++	+++	+++

[12]

Figura 30. Detectores de puntos característicos

Existen diversas técnicas para detectar las características de una imagen, entre las cuales se destacan, Harris, Shi-Tomasi, FAST, SIFT, SURF y CENSURE. Cada una genera una serie de errores que limita su uso para ciertos tipos de implementación. Es decir, Harris es un método de detección de esquinas en imágenes a escala de grises, pero no logra detectar el escalado de imagen correctamente aun así es fiable debido a que mantiene repetitividad con respecto a las otras imágenes, pero es lenta en ejecución, contrario del método FAST, que es la más rápida en cuanto ejecución sin embargo no es igual de robusta que la técnica SIFT que es capaz de hacer cálculos precisos pese a que sea la más lenta y menos apta para la implementación.

Una vez realizado el cálculo de disparidad este proceso se repite, pero trabajando con las imágenes estéreo generadas a partir del mapa de disparidad. Este segundo proceso de extracción va enfocada a analizar el movimiento del robot.

[12] Estudio extraído de la revista [32]

8.2 Cálculo de disparidad:

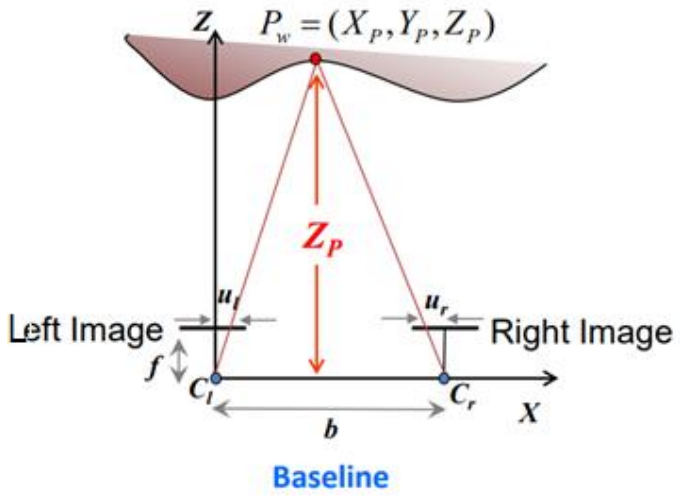


Figura 31. Profundidad a partir de una imagen estéreo[13]

Los valores de profundidad se hallan por medio de una relación geométrica, cuyos parámetros serán u, f y b. El primero es el parámetro que me permitirá encontrar el límite de convergencia de la perspectiva de las dos cámaras, el segundo equivale a la distancia que hay del punto analizado en la imagen con respecto al observador y el último la distancia constante de separación entre un lente con el otro. La disparidad resulta de la diferencia entre $u_l - ur$ es proporcional a la distancia de la separación y la distancia del punto analizado al foco, un factor importante y necesario para la reconstrucción en 3D. Es importante resaltar que la distancia Z será siempre el mayor valor.

$$Z_p = \frac{bf}{u_l - ur} \qquad (23)$$

[13] Información extraída de [3]

Figura 32. Imagen de cámara izquierda (CL) [14]

Figura 33. Imagen de cámara derecha (CR)

En este apartado se visualiza el trabajo de dos imágenes para el análisis de la primera extracción, empíricamente la cámara RealSense permite la calibración para trabajar con una imagen estéreo que es la combinación de las imágenes con ambas perspectivas. Los puntos de correspondencia entre las imágenes se analizan siempre en la misma línea horizontal.

El mapa de disparidad representa la suma de la diferencia al cuadrado que separan dos píxeles correspondientes, con este mapa se realiza la asignación de coordenadas en el plano 3D, (x,y,z).

[14] Base de datos extraída de data_odometry_gray [28]

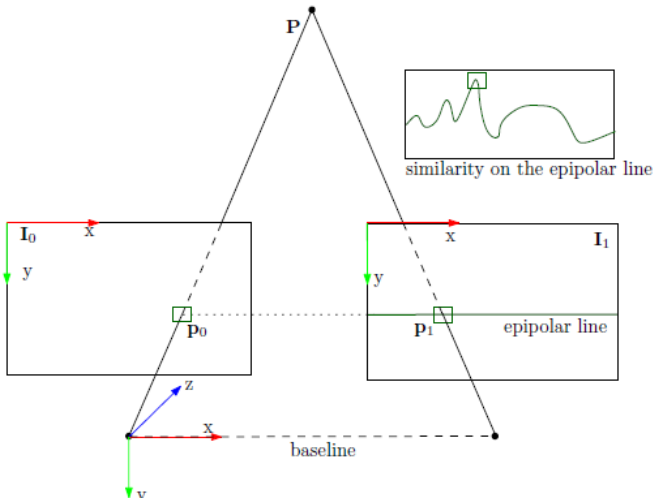


Figura 34. Localización de P0 y P1 en I0 (CL) y I1 (CR)[15]

El algoritmo determina la disparidad de los pixeles entre un marco visto desde la izquierda con uno de la derecha el punto 3D se proyecta a partir de p_0 y p_1 por un proceso de correlación entre imágenes, cada imagen es comparada desde una posición inicial en el plano epípolar (línea horizontal que las separa). Debido a la relación de distancias el punto de la derecha siempre se encontrará desplazado proporcionalmente al punto de la izquierda

Cada posición calculada se reúne en una matriz llamada matriz fundamental la cual almacenará valores de coeficientes que servirán para determinar la correspondía de los puntos generando así una nube de puntos en el plano tridimensional.

Esta nube de puntos es la que hace posible la generación de un contorno tridimensional de los objetos.

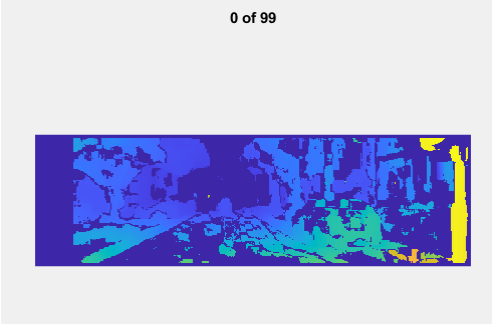


Figura 35. Mapa generado a partir de las dos primeras imágenes secuencial

[15] https://github.com/cgarg92/Stereo-visual-odometry

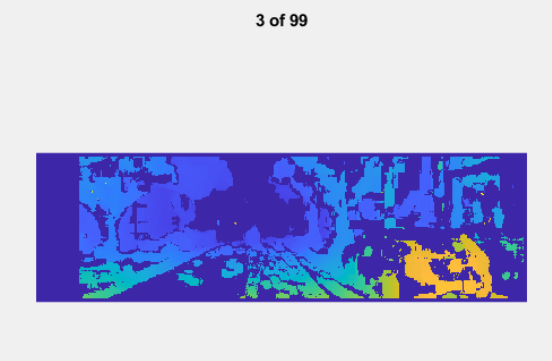


Figura 43. Mapa generado a partir de la segunda y tercera imagen secuenciales

8.3 Reconstrucción en 3D

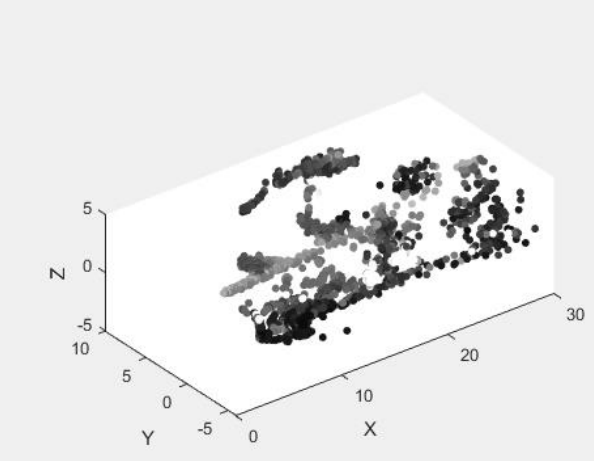


Figura 36. Generación de la nube de puntos en el plano xyz

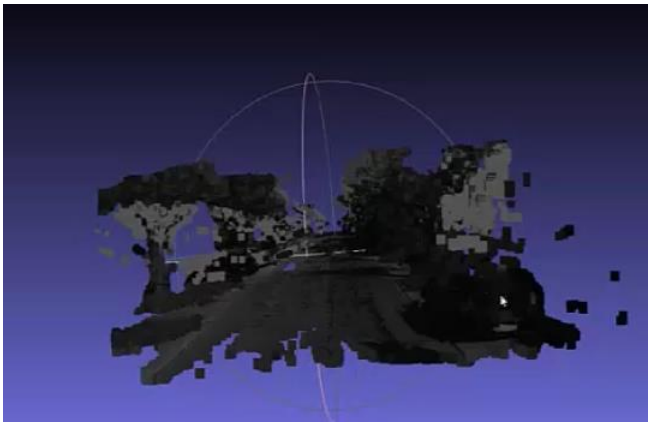
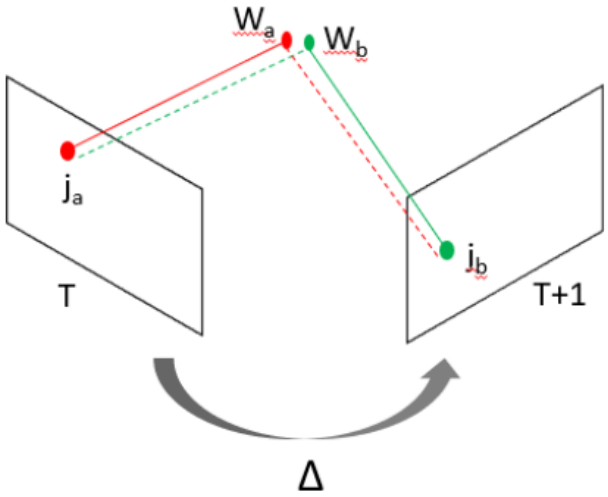


Figura 37. Reconstrucción tridimensional de las imágenes

8.4 Extracción del movimiento.



[16]

Figura 38. Proyección de un punto Wb por triangulación comparado con el punto Wa resultante de un desplazamiento delta

La extracción de movimiento se determina al tomar todos los puntos detectados de una de las imágenes (conjuntas entre la de la izquierda con la de la derecha) y se aplica un desplazamiento. Sí esto coincide entre la imagen proyectada (es decir la imagen calculada en la secuencia) se obtiene la información del desplazamiento.

Sin este desplazamiento, el mapeo resultaría incompleto debido a que no se almacenaría información y solamente resultaría en un mapeo situacional donde solo se hace una toma de datos y esta no vuelve a repetirse.

[16] https://github.com/cgarg92/Stereo-visual-odometry

9 VALIDACIÓN DEL ALGORITMO

Una vez tenido el montaje en físico del robot móvil y también hecha la conexión con la cámara se procedió a la realización de pruebas para recolección de datos a procesar. En este caso el robot fue puesto en un entorno con pequeños materiales a su alrededor que cumplen el objetivo de servir como elementos a reconstruir. El robot comienza su movimiento en este caso limitado a la conexión por cable del Arduino para comenzar un movimiento simple y recto. El robot inicia su travesía mientras graba con la cámara RGB y así tomar muestras del entorno a reconstruir.

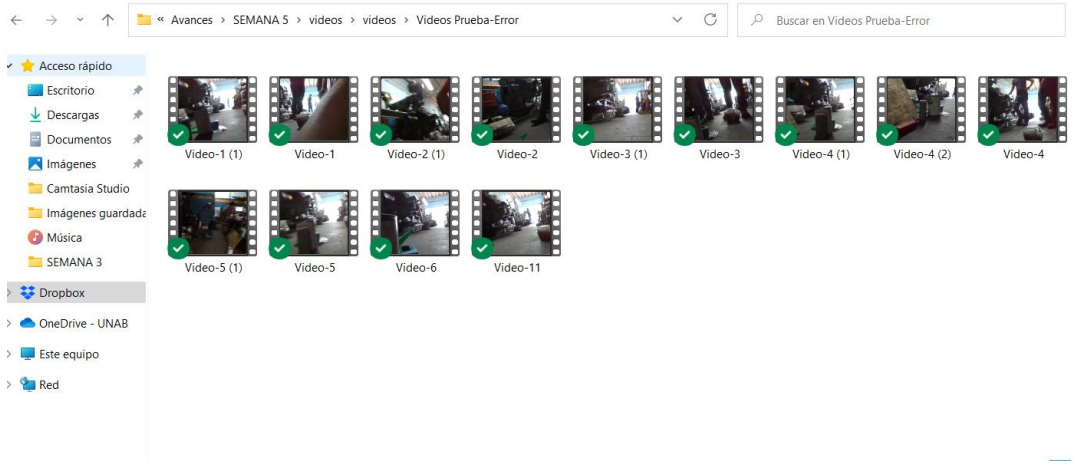


Figura 39. Pruebas de video tomada tomadas desde el robot

Se realizaron varias tomas entre cerca de 13 videos con un recorrido corto del vehículo. Se visualizaron pequeñas desviaciones en el movimiento debido a la limitación de estar conectada por cable y por ello se asistió de tal manera de que el cable no interfiriera en la trayectoria. Entre los 13 videos que se tomaron se decidió escoger uno para hacer su respectivo análisis. Esto con el objetivo de validar el algoritmo en Matlab y se muestran los resultados a continuación:

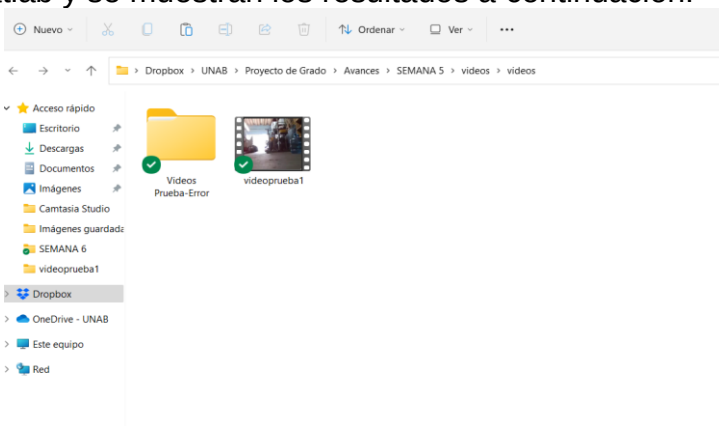


Figura 40. Video recortado con duración de 10 segundos

Este proceso es utilizado para analizar los resultados obtenidos de imagen por imagen y así dar validación al algoritmo de extracción y detección. Esta información

puede ser reconstruida que finalmente es el objetivo de esta práctica de este ya que se busca validar con una construcción 3D de los objetos vistos imagen a imagen.

Figura 41. Captura del frame correspondiente a la imagen "000000 (61)"

Figura 42. Captura del frame correspondiente a la imagen "000000 (360)"

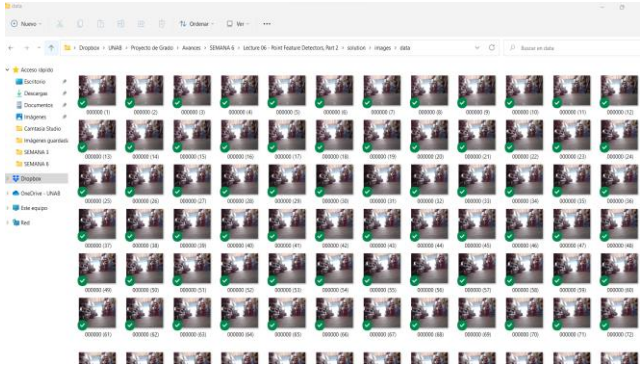


Figura 43. Extracción de imágenes (517)

La cámara tiene la capacidad de capturar 30 imágenes por segundo, es por ello por lo que para el procesamiento se decidió trabajar con 15 segundos lo cual genera un total de 517 imágenes, el objetivo es realizar el procesamiento de estas imágenes utilizando el algoritmo SLAM para extracción y correlación de puntos característicos.

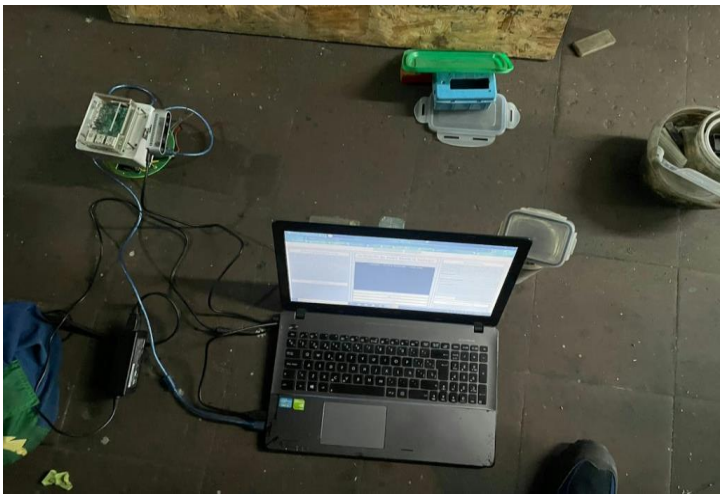


Figura 44. Implementación para prueba de reconocimiento

Para esto el programa toma un “Parche” que conlleva unos 8 pixeles cercanos y realiza esta comparación en ubicación con las imágenes consecutivas. Una vez calculados los extremos locales en escala y espacio se comparan entre las imágenes consecutivas y se verifica coincidencias con hasta 9 pixeles próximos de esta forma se genera una pirámide de datos que son pasados a un cálculo estadístico para hallar la ubicación de aquellos con mayor incidencia.

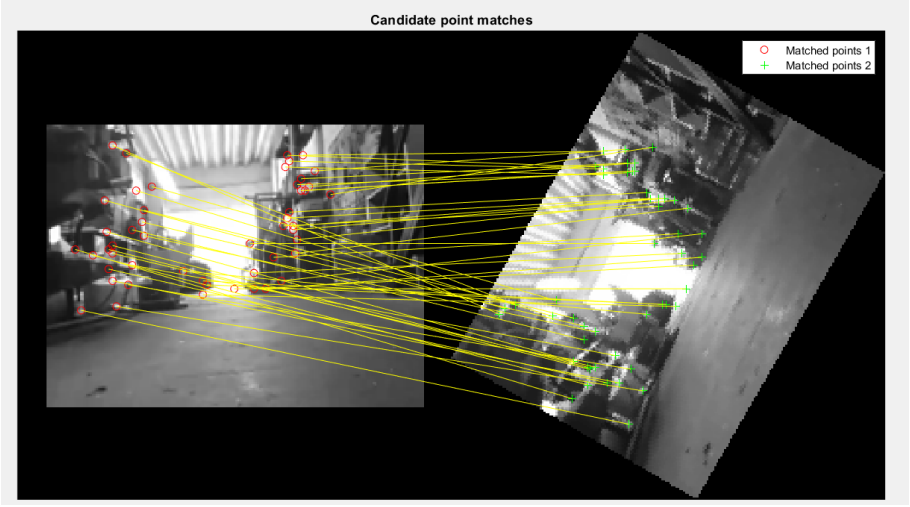


Figura 45. Correlación de los puntos en las imágenes secuenciales.

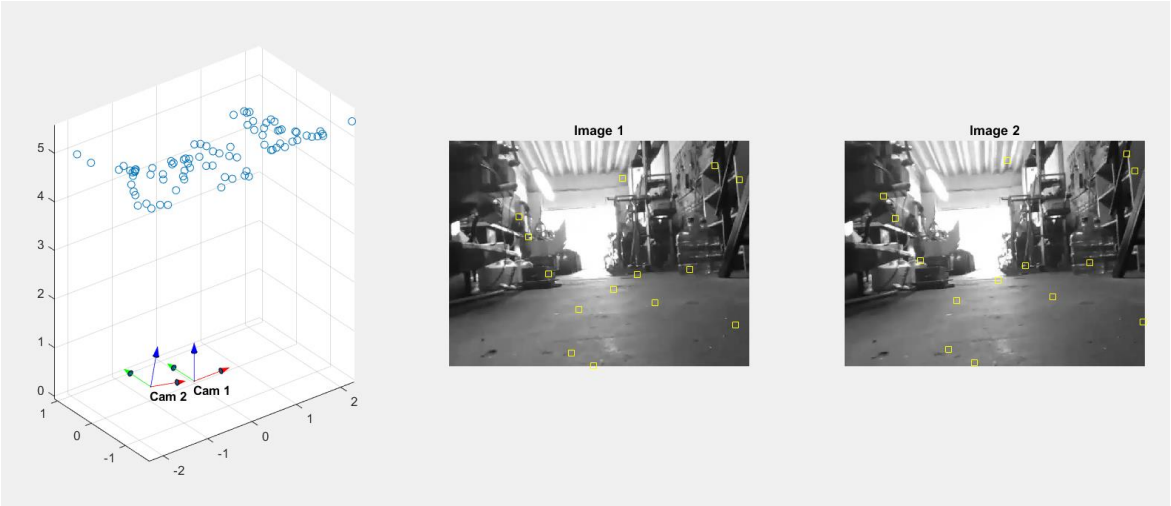


Figura 46. Triangulación para encontrar los puntos Zp

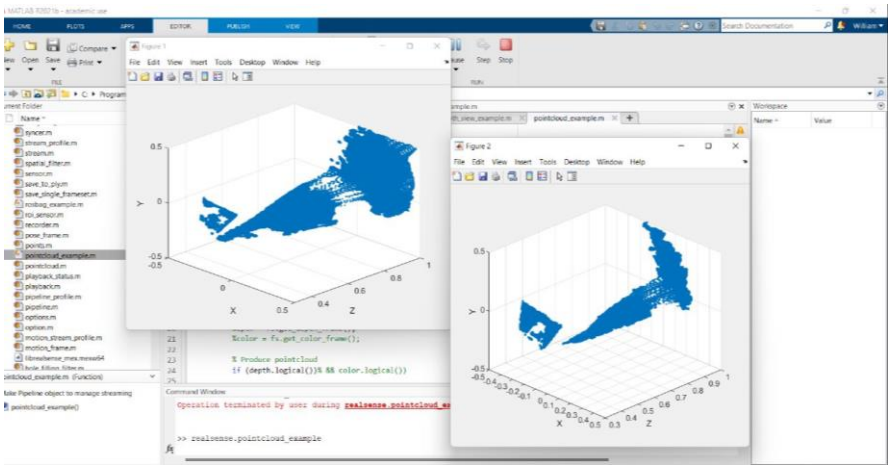


Figura 47. Nube de puntos generada con la D435i en Matlab

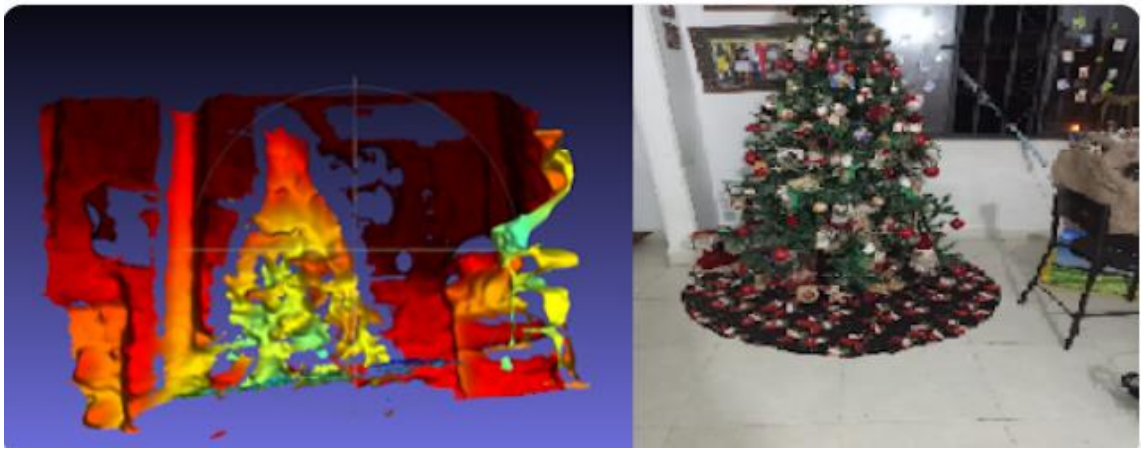


Figura 48. Reconstrucción del mapa de disparidad

10 CONFIGURACIÓN DEL SISTEMA EMBEBIDO

La Jetson Nano es resumidamente como un pequeño computador que permite ejecutar redes neuronales en paralelo, usado para la clasificación de imágenes y demás actividades relacionadas a la inteligencia artificial. Posee 4 conectores USB, Un adaptador Gigabit Ethernet, un puerto micro USB de 5V, puerto de HDMI. Su configuración se realiza internamente.

10.1 Configuración de Hardware

Se instala el Kit Jetson con cable USB del tipo C, se conecta el cable de ethernet y se da alimentación. La Jetson nano debe incluir un ventilador que permita ayudar a disipar el calor, adicional a esto se debe emplear una tarjeta Micro SD de mínimo 64 GB de espacio ya que en ella irá todos los elementos del sistema operativo para inicializarse.

Figura 49. Adaptación física de la Jetson Nano

10.2 Configuración de Software

Para trabajar la Jetson nano es importante saber o tener en cuenta el uso del sistema operativo Ubuntu, ya que es el sistema operativo compatible con la tarjeta, y como aspecto importante el entendimiento de su funcionamiento basado en paquetes y repositorios para seguir una secuencia lógica en cuanto al proceso de configuración se establece un orden de operación, que es el siguiente: En este caso se instala una imagen visual (.iso) la cual posee los siguientes elementos de software: Sistema operativo Ubuntu 16.04 LTS, l4t para conectividad con la red.

Se procede a crear un usuario y una contraseña los cuales brindaran los permisos de dominio requeridos para configurar el sistema en modo root.

10.3 Actualización del Kernel

Intel RealSense proporciona instaladores para Ubuntu desde la versión 16 hasta la 22. La comunicación de esta pasa por el Kernel que es la parte del sistema operativo de la Jetson que se encarga de dar acceso al hardware de forma segura, Al instalarse la versión 16.04 LTS. Se debe actualizar el Kernel para que sea compatible con la versión del SO trabajado a través del comando.

sudo apt-get install --install-recommends linux-generic-hwe-16.04[17]

10.4 Instalación de componentes de software de NVIDIA

10.4.1 Máquina Virtual DOCKER

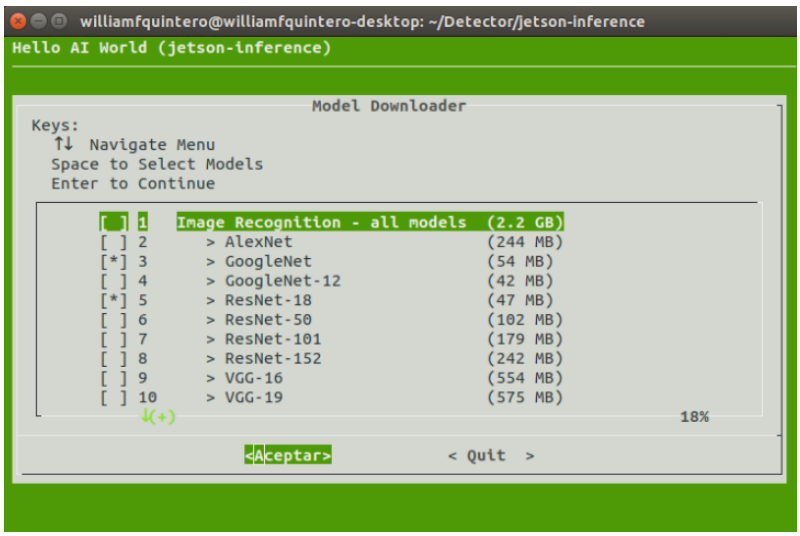


Figura 50. Máquina virtual DOCKER

Docker es un aplicativo que facilita la ejecución de procesos en forma de máquina virtual, es dependiente del sistema operativo host. Lo que la hace dependiente de la conexión a la red. Provee a su vez al programador tener la capacidad de transferencia de archivos necesarios para el funcionamiento del software de la tarjeta Jetson. Para su instalación se requiere de los comandos:

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add –
sudo apt-get install -y docker-ce
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

[17] https://ubuntu.com/kernel/lifecycle

Adicional a esto, Docker permite que a través de su plataforma se pueda instalar ciertos componentes de software de NVIDIA en el kit de desarrollo de Jetson, los cuales son:

10.4.2 JetPack versión 4.4.1[18]:

Es un SDK (kit de desarrollo de software) es indispensable para la creación de aplicaciones de inteligencia artificial. JetPack 4.4.1 es la versión más reciente y permite un soporte sobre los módulos de la Jetson Nano de NVIDIA.

10.4.3 L4T 32.4.4:

Brinda acceso directo a la GPU ya que canaliza el cómputo de los gráficos a través del api Vulkan.

git clone https://github.com/KhronosGroup/Vulkan-Loader.git

10.4.4 CUDA versión 10.2:

Es una API completa para la gestión de la memoria virtual en la tarjeta gráfica, con funciones precisas para la asignación de memoria y rangos de direcciones en la memoria.

10.4.5 Libargus:

Es una API para adquirir imágenes y metadatos asociados de cámaras. La operación libargus fundamental es una captura: adquirir una imagen de un sensor y procesarla en una imagen de salida final.

10.4.6 OpenGL® ES:

Es una API multiplataforma para gráficos 2D y 3D de funciones completas en sistemas integrados.

10.5 Configuración ROS (Sistema Operativo Robótico)

10.5.1 Ros-versión: 1.14 (Melódico)

ROS es un conjunto de librerías de software que se usan para aplicaciones tanto de robótica como de visión artificial. La versión “Melodic” es una de las versiones de ROS que permite compatibilidad con Ubuntu 16.04 LTS de 64 bits. Su peso en memoria de procesamiento es cercano a los 32 GB por lo cual es un elemento que

[18] https://developer.nvidia.com/embedded/jetpack-sdk-441-archive

consumirá con total seguridad más de la mitad del espacio de disco disponible en la memoria SD. ROS trabaja con nodos que son subprogramas en lenguaje C++ o Python. Estos se pueden modificar usando el editor bash y así se pueden programar un robot simulado que conecte con el robot real.

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
sudo apt install ros-melodic-desktop-full
```

10.6 Configuración de rosdep

Rosdep es una herramienta que instala dependencias entiéndase dependencias a aquellas aplicaciones que necesitan de otras para completar su funcionamiento, en este caso las dependencias son relacionadas al entorno que reconstruye a partir de la cámara de RealSense, debido a que ciertas factores como el mapa, la nube de puntos, todos estos aspectos dependen de la información visual que se otorgue la cámara, así como la reconstrucción también depende de la precisión con la que se genere la nube de puntos. En este caso estar siendo Rosdep la encargada de buscar estos elementos que hacen falta para la completa ejecución del programa.

```
sudo apt install python-rosdep
sudo rosdep init
rosdep update
```

La instalación de dependencias se hace de forma automática una vez este inicializado el rosdep. Esta herramienta optimiza enormemente la instalación de la cantidad de elementos y funciones que trabajan con el entorno.

Es muy importante poner la siguiente linea de código en el funcionamiento del bash:

```
source /opt/ros/rolling/setup.bash
```

10.7 Sensor RealSense™ SDK 2.0

Para el SLAM es imprescindible de que ROS pueda hacer uso de los sensores de la cámara para ello RealSense dispone de un SDK con las funciones que permiten que se ejecuten a partir de comandos de ros. Basta con clonar la librería:

https://github.com/IntelRealSense/realsense-ros

Adicional a esto se configuran repositorios para hacer uso de las librerías de RealSense.

```
sudo add-apt-repository "deb https://librealsense.intel.com/Debian/apt-repo $(lsb_release -cs) main" -u
```

sudo apt-get install librealsense2-dkms
sudo apt-get install librealsense2-utils
sudo apt-get install librealsense2-dev
sudo apt-get install librealsense2-dbg

Los paquetes constantemente deben de actualizarse ya que de no hacerse no se ejecutarán o simplemente el terminal no logrará localizar la carpeta contenedora. Por temas de orden y distribución se decidió guardar las librerías necesarias en carpetas para que al momento de su operación sea más fácil para el programador abrir el respectivo archivo ejecutable. Esto por medio de la creación de un entorno de trabajo o Workspace.

mkdir -p ~/catkin_ws/src
cd ~/ros2_ws/src[19]

mkdir es un comando que emplea la creación de un subdirectorio en este caso se direcciona a la carpeta catkin_ws y src que es el sitio donde se almacenan por defecto todos los elementos o paquetes. La instalación de paquetes es similar a un proceso de desempaquetado lo cual le permite gestionar simultáneamente diversos paquetes que se obtienen con el comando git clone.

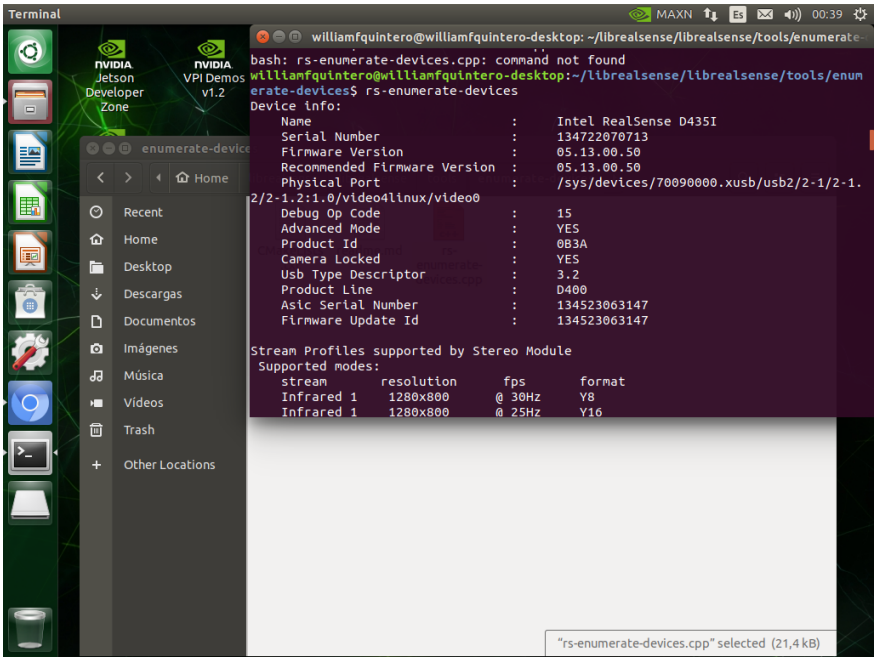


Figura 51. Configuración de la Intel RealSense D435I en la Jetson Nano

[19] https://manpages.ubuntu.com/manpages/bionic/es/man1/apt-src.1.html

11 IMPLEMENTACIÓN EN EL SISTEMA EMBEBIDO

11.1 Simuladores RVIZ y Gazebo

En esta sección se presenta el trabajo realizado desde el sistema maestro de ROS, en el cual se diseña una simulación para un robot diferencial con el objetivo de comunicar este con la tarjeta del sistema embebido.

El simulador permite reemplazar el robot para la realización de pruebas en cuanto a que es capaz de tomar las imágenes de profundidad, semántica y RGB lo cual permite trabajar en la implementación del algoritmo a través de cámara de profundidad con el fin de conseguir una comunicación a tiempo real de la toma de datos.

Un aspecto fundamental a tener en cuenta en cuanto a la programación en La Jetson e implementación del entorno de ROS es el tipo de distribución a implementar y este depende del sistema operativo en el que se trabaje, es decir Ubuntu 16.04 emplea o requiere el uso de ROS Kinetic, el Ubuntu 18.04 emplea el uso del ROS Melodic y la versión 20.04 del mismo requiere obligatoriamente el ROS Noetic. La variación de alguno de estos modelos determina la compatibilidad del sistema con el simulador que servirá de interfaz que modela el seguimiento del programa.

Los paquetes principales de ROS para crear y administrar el espacio de trabajo son descargados de acuerdo con los requerimientos de la plataforma. En este caso de instalar ROS Melodic a través de packages.ros.org, me permite tener acceso las diferentes herramientas que poseen el paquete ROS y adaptarlo al sistema Ubuntu desde el cual se solicita (lsb_release -sc).

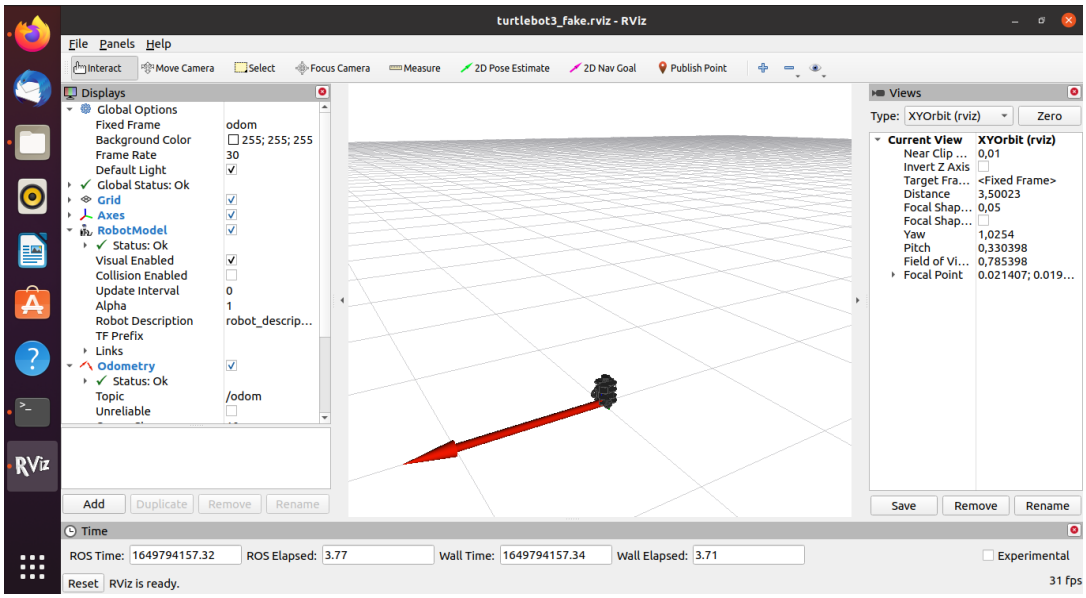


Figura 52. Simulación del robot en el entorno RVIZ

Configurar la herramienta CMake en la Jetson Nano que se realiza con el programa msys2 el cual permite descargar los paquetes necesarios para la programación de trayectoria del robot simulado desde el lenguaje de Python. Estas pruebas permiten identificar elementos de la plataforma de visualización RVIZ. La cual permite modificar elementos de su interfaz y se usa para comunicar el robot simulado con el robot real.

Ubuntu provee una facilidad en programación que permite el fácil manejo de librerías como GIT y permite una programación amena con Python o C++ que son lenguajes requeridos en la implementación de código en las tarjetas de procesamiento, un punto importante a tener en cuenta es que su programación manual permite la ejecución de comandos que pueden traducirse a una interfaz o a una tarjeta de procesamiento de forma paralela.

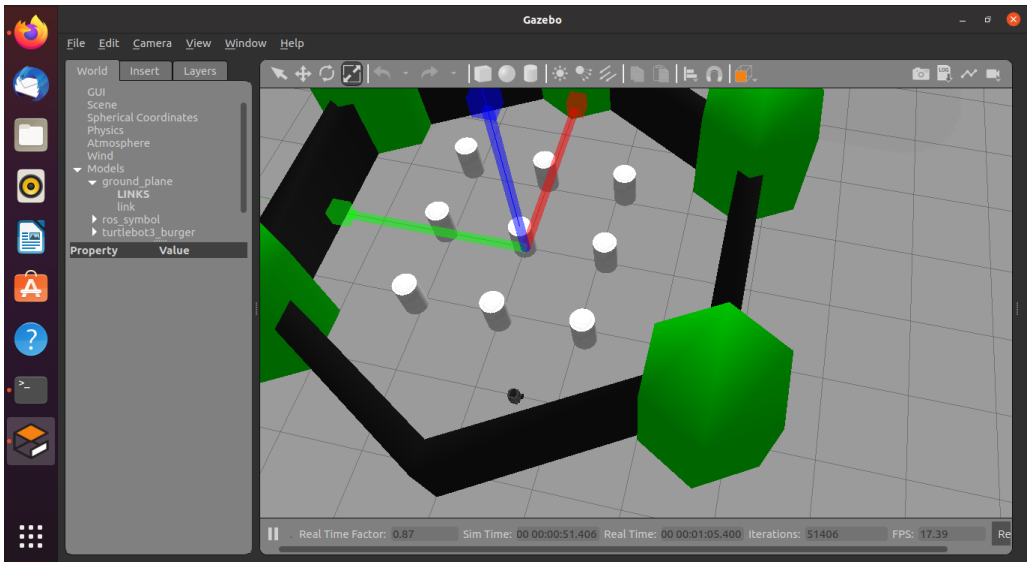


Figura 53. Navegación del robot simulado en un entorno estructurado con GAZEBO

Gazebo y Rviz son herramientas de visualización tridimensional para robots, sensores y algoritmos. Esto permite una visualización en tiempo de real del estado de un robot o su entorno lo cual es indispensable en la elaboración de un programa de SLAM.

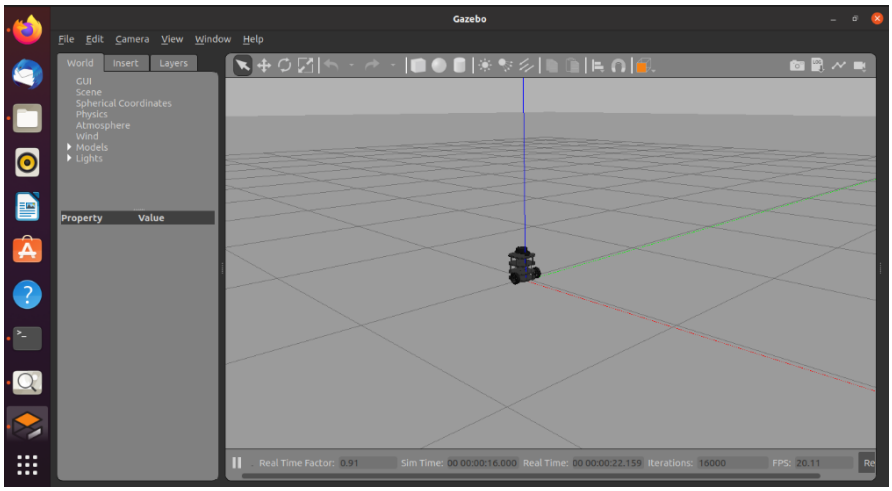


Figura 54. Robot moviéndose con los comandos programados en Python

Para programar el movimiento del robot se puede emplear lenguaje C o Python en este caso el movimiento es procesado mediante la ejecución del programa de movimiento del robot esto se visualiza en las gráficas donde el robot que es visto desde la vista superior se desplaza en un movimiento recto en RVIZ.

ROS Melodic es un sistema operativo de código que se usa comúnmente para la programación de robots y su sistema incluye la implementación de hardware para el control de este. En este caso es un sistema operativo ideal que trabaja con los objetos en tiempo real. Para su uso óptimo se optó por la instalación de Ubuntu, el cual facilita de gran manera su programación e implementación de Drivers y librerías.

Cada plataforma virtual de software que trabaja ROS para la robótica tiene el objetivo de utilizar la mayor cantidad de funciones y es por ello por lo que se subdivide en Nodos y Repositorios, los primeros se encargan de operaciones o cálculos independientes que se necesiten implementar y paquetes que son los que contienen los procesos como si de una biblioteca se tratase.

Gazebo como se visualiza en pantalla es otra alternativa de entorno que permite visualizar en sincronía con la realidad lo que hace el robot, se busca que en este entorno se reconstruya el entorno tridimensional del SLAM.

En general para marcar las trayectorias el nodo para publicar las posiciones que debe alcanzar el robot, otro donde debe recibir esas posiciones.

ros::Subscriber target_sub = n.subscribe<geometry_msgs::Pose>("/pose_target", 1, targetCallback);

Se configura un suscriptor que ejecuta la función de arranque a posición y ejecute la función callback que se encarga de guardar la posición. ¡Para ejecutar las funciones del paquete Moveit! Que es el encargado de resolver la cinemática de los robots de acuerdo a su tipología. En este caso se realiza la configuración basada en un sistema diferencial que es el robot físico y posteriormente se definen tanto coordenada articulares del robot como también el tipo de trayectoria a recorrer.

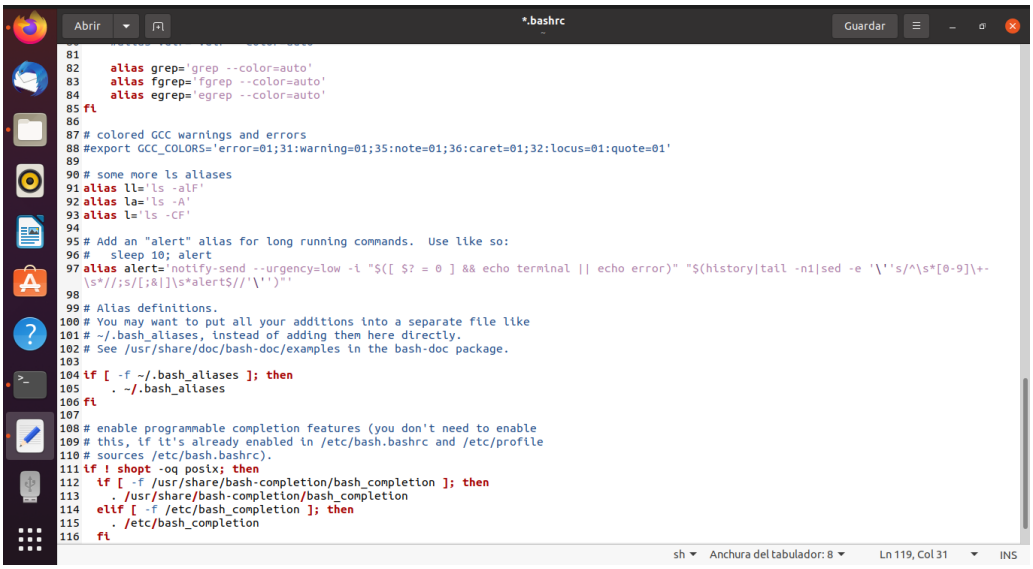


Figura 55. Configuración del bashrc

El bashrc es configurado de tal manera que permita al ROS melodic llamar paquetes utilizando directorios que se especifiquen y así proveer un control para el manejo de las operaciones que se requirieran ejecutar.

Roscore es un entorno que permite visualizar los requisitos previos para ejecutar un determinado programa y este se usa para llamar al Roslaunch que será el encargado de ejecutar un determinado programa que en este caso serán o Gazebo o RVIZ

Catkin es una herramienta que permite la creación de paquetes de ROS, normalmente viene incluida al momento de instalarse ROS, pero este no fue el caso así que se procedió a instalarse y en este caso que se usase a partir de Python que es un lenguaje de alto nivel y es el más compatible con las tarjetas de procesamiento.
Anteriormente se visualiza los resultados de implementar Gazebo o Rviz en un sistema operativo con Ubuntu lo cual comprueba el funcionamiento de estas interfaces que servirán de guía para parametrizar la información resultante de la cámara y su posición respecto al espacio.

11.2 Protocolo de comunicación en ROS

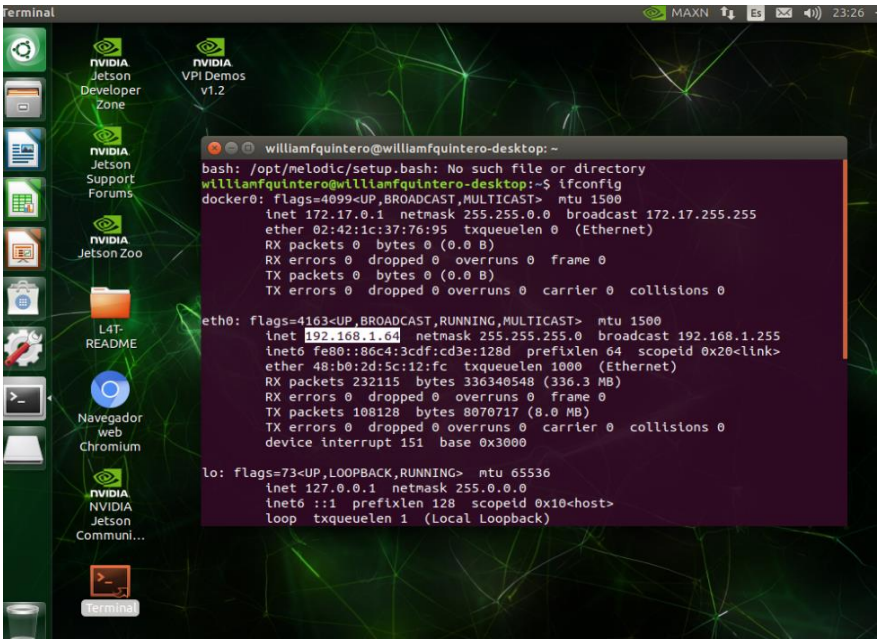


Figura 56. Comunicación Usuario-máquina

La comunicación en ROS se realiza configurando la dirección IP, en esta existen dos direcciones que conectan con las API del entorno. ROS_MASTER_URI y ROS_IP. El primero comunica el sistema embebido con el entorno roscore. El segundo es el localhost que usa dirección IPv4 o IPv6.

El DNS, broadcast entre otros van de acuerdo con la dirección local del sistema. Para su configuración se usan los comandos

export ROS_MASTER_URI=http://192.168.1.64:11311
export ROS_IP=192.168.1.64

El Sistema de ROS debe contar con dos sistemas, una computadora que permita correr la simulación (preferiblemente con SO compatible) y una placa remota en este caso la Jetson Nano. La placa es el sitio donde se prueba el algoritmo mientras que la simulación se corre en la computadora (ambas con dirección IP diferentes).

Cuando no se tiene la posibilidad de tener un segundo sistema es posible usar un servidor de red que haga la tarea de maestro, al conectarse físicamente con el puerto de ethernet la placa Jetson reconoce el servidor y automáticamente cambia a la dirección IP de esta. Sin embargo, para su correcto funcionamiento se debe configurar el ROS_IP de acuerdo con la dirección IP del localhost para que no haya interferencias entre servidores, esto basta con poner “localhost” en lugar de la dirección IP del servidor.

12 RESULTADOS DE LA IMPLEMENTACIÓN

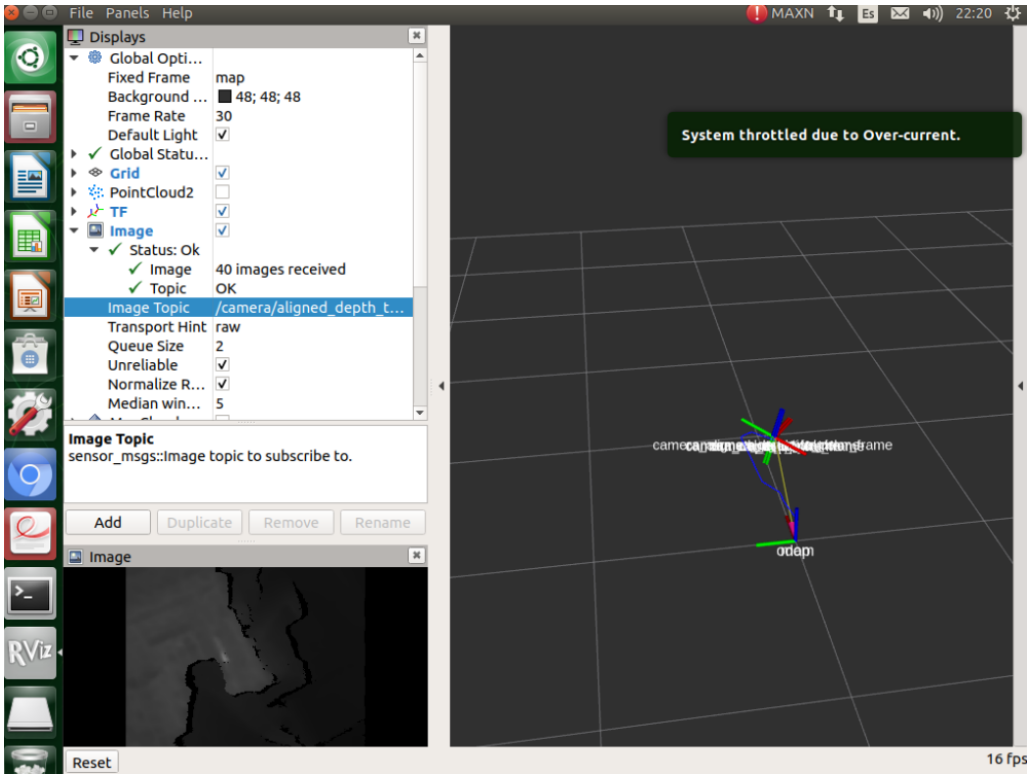


Figura 57. Implementación del RVIZ con la cámara Intel D435i

En esta sección se muestran los resultados de la implementación en donde se realizó una comunicación maestro-esclavo siendo el maestro el servidor red, como se explicó en la sección de protocolos. Este es el primer paso para dar validación al funcionamiento del sistema ROS y de esta manera una vez esté funcional se busque conectar al maestro diseñado en la sección anterior desde un computador con SO Ubuntu 16.04 y sincronizar el robot simulado con el real.

La configuración de RVIZ nos permite de forma clasificada elegir el dato a simular siendo separados por Maps donde se encuentra las simulaciones generadas para la reconstrucción del entorno, PointClouds donde genera la nube de puntos de los objetos localizados entre otras características.

En este caso se debe garantizar que cada API instalada en ROS se ejecute correctamente y eso se demarca por los iconos en verde. En caso de aparecer en rojo se debe realizar una revisión del ros-update para verificar si el sistema ha sido actualizado correctamente.

El RealSense™ D435i está equipado con una IMU integrada. La nube de puntos y un montón de flechas, o marcas de ejes, apareen en la pantalla sistematizando que representan todos los diferentes sistemas de coordenadas involucrados.

Las IMU son las Unidades de Medición Inercial (IMU) son sensores que permiten medir tanto el movimiento direccional como la rotación. El sensor está compuesto por sensores giroscópicos triaxiales de aceleración lineal de 12 bits y de velocidad angular de 16 bits.

La cámara de profundidad D435i genera y transmite de 200 a 400 Hz muestras del giroscopio interno y el acelerómetro la ubicación y orientación del sensor va en relación con los sensores de profundidad y se multiplica cada muestra por la matriz extrínseca, de esta forma se obtienen los datos de la nube de puntos generada en las imágenes.

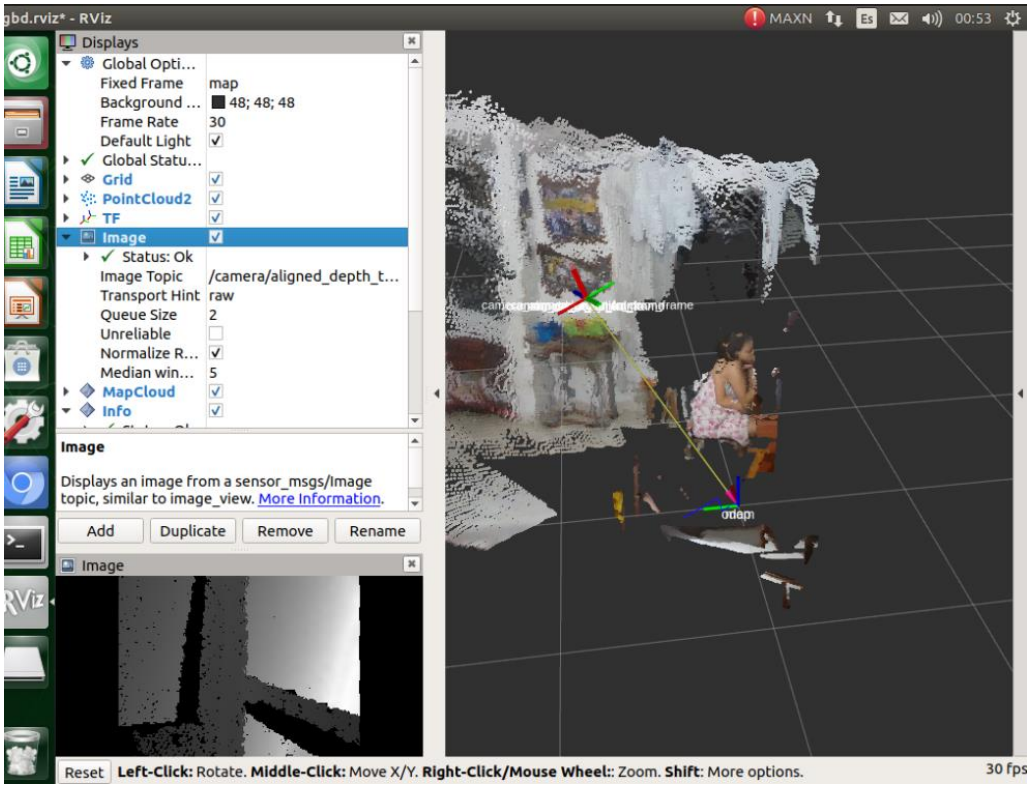


Figura 58. Reconstrucción tridimensional del entorno

Rviz es un visualizador que se usa para ejecutar los datos procesados en el ROS siento así el frontend del algoritmo, permite mostrar los resultados en el sistema de coordenadas por lo que es posible visualizar los datos en tiempo real al estar conectada con la cámara.

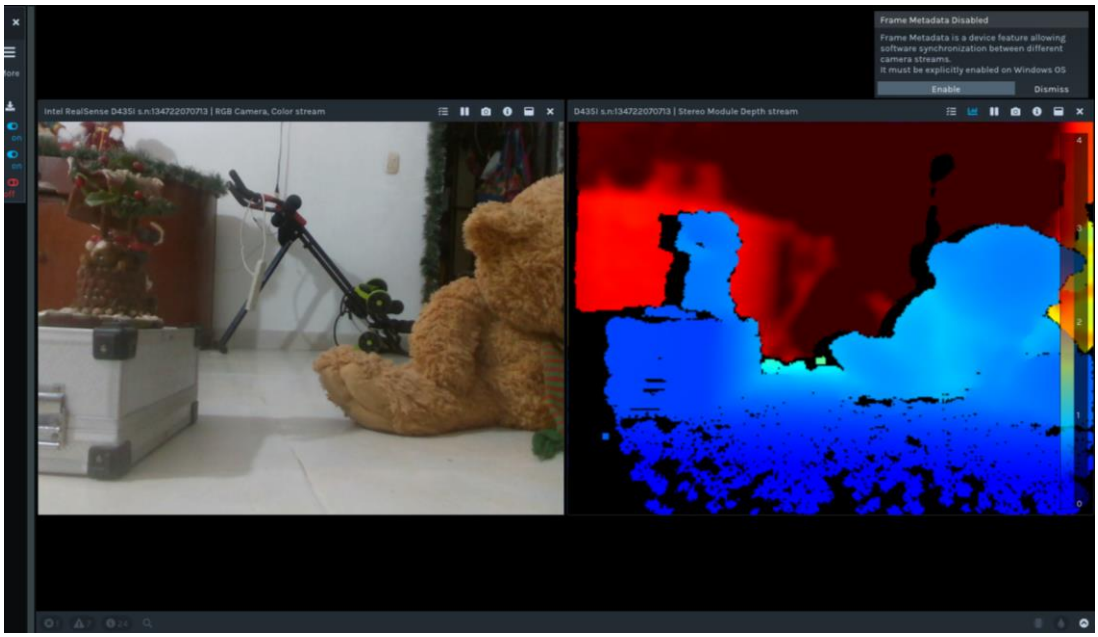


Figura 59. Imagen normal vs mapa de disparidad

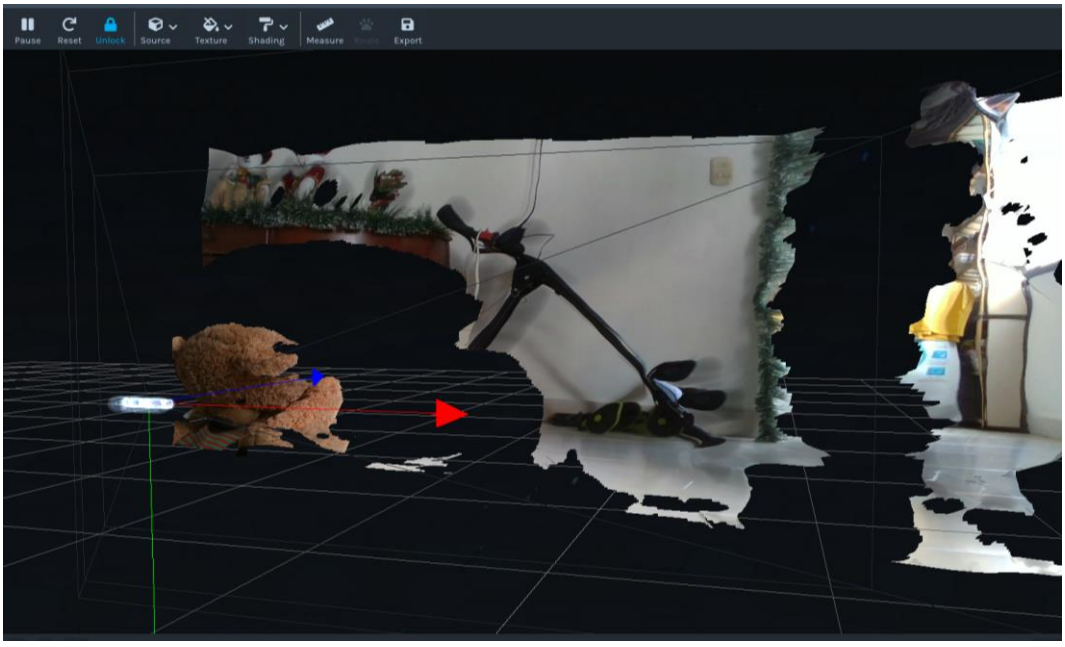


Figura 60. Proyección de profundidad de la figura 59

Como se puede observar el programa conecta a tiempo real con la cámara Intel D435i física y reconstruye tanto la imagen normal RGB como la imagen de profundidad añadiéndole los colores que permiten una fácil identificación de la

profundidad del objeto, en este caso parte de una habitación que conecta a la cocina.

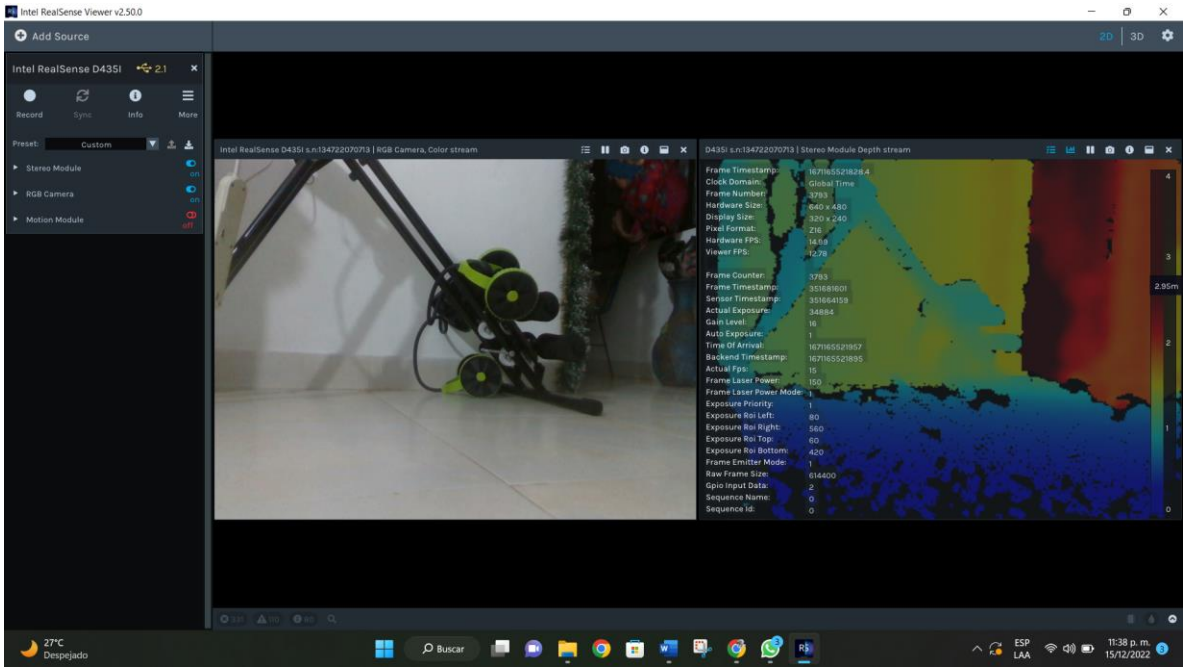


Figura 61. Información de profundidad

Como resultado la IMU integrada solo puede realizar un seguimiento durante un tiempo muy corto. Se generan conflictos en los movimientos bruscos y moverse o girar demasiado rápido romperá la secuencia de coincidencias de nubes de puntos exitosas y hará que el sistema pierda la pista.

13 CREACIÓN DE LA INTERFAZ DE USUARIO

Se busca conectar la interfaz con el módulo bluetooth. Los puertos Rx y Tx se conectan de forma inversa en el Arduino debió a que la comunicación debe seguir una secuencia lógica, es decir, si el módulo trasmite, el Arduino debe recibir la información.

El módulo Bluetooth HC-05 viene configurado de fábrica como si fuera esclavo sin embargo esto es posible de cambiarse para la funcionalidad de brindar comunicación entre el programa en Arduino y la placa física de forma inalámbrica.

Los leds parpadean cuando no se hay una comunicación con el dispositivo y se debe realizar el protocolo de comunicación inalámbrica a 2.4 Ghz y de corto alcance es el ancho de banda ISM, apto para operar cortas distancias. La conexión al PIN TX indicará que es el trasmisor y RX el receptor.

La configuración del HC05 como esclavo se da por medio de los comandos AT+NAME, AT+PSWD, AT+UART y AT+ROLE0. Siendo los primeros los que permiten el ingreso del nombre y contraseña del dispositivo, seguido de la velocidad y el rol que recibirá el módulo. El punto clave para la programación es conocer cuando usar el “?” y el “=” ya que son usados para lectura y asignación de valores respectivamente.

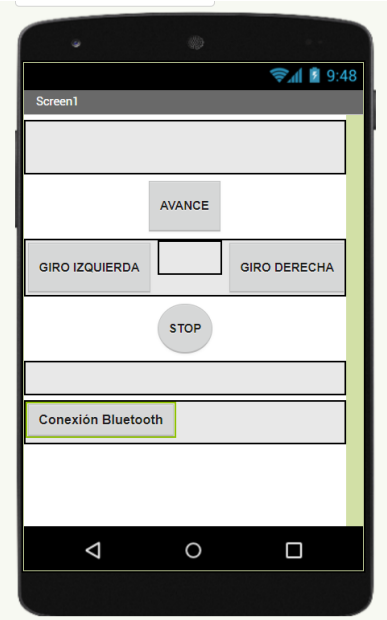


Figura 62. Interfaz de control del carrito

Para la programación se realiza el diseño de cada botón y según la lectura que se haga a través del módulo se envía un texto de salida que leerá el programa de Arduino que determinará que motor será accionado.

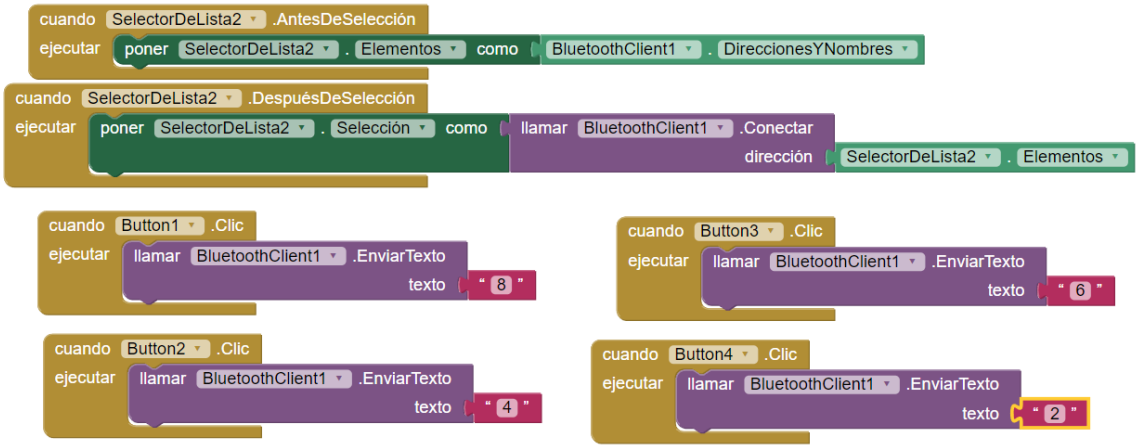


Figura 63. Configuración de la interfaz

El programa de Arduino leerá los valores a través de una variable que será la que almacene el valor comandado desde el teléfono. Los ciclos if determinarán los comandos a seguir estos siempre después de haberse confirmado la comunicación entre el maestro y el esclavo.

La función principal del programa es realizar una traducción de código y servir como puente entre el Software Matlab y la tarjeta Arduino para facilitar la programación y la implementación real. En este caso el programa simplemente vario las señales recibidas de PWM y según su valor hará que una de las ruedas gire en un sentido o en el otro dependiendo de los pines que indique la hoja de datos de la Dynamotion V3. Esto se traslada al entorno de Arduino en el cual se utiliza como medio el módulo de bluetooth.

13.1 Tabla de verdad

Tabla 1. Tabla de estados para control de motores

n	n2	n3	n4	ACCIÓN
0	0	0	0	STOP
0	0	0	1	IZQ
0	1	1	0	STOP
0	1	1	1	STOP
0	1	1	0	DER
0	1	1	0	STOP

0	1	1	0	STOP
0	1	1	1	STOP
1	0	0	0	ADEL
1	0	0	1	ADEL - IZQ
1	0	0	0	STOP
1	0	0	1	STOP
1	1	1	0	ADEL- DER
1	1	1	1	STOP
1	1	1	0	STOP
1	1	1	1	STOP

A continuación, se presenta el modelo ya con el módulo HC05 instalada y una extracción de la prueba realizada con el aplicativo. [más detalle consultar prueba 8]

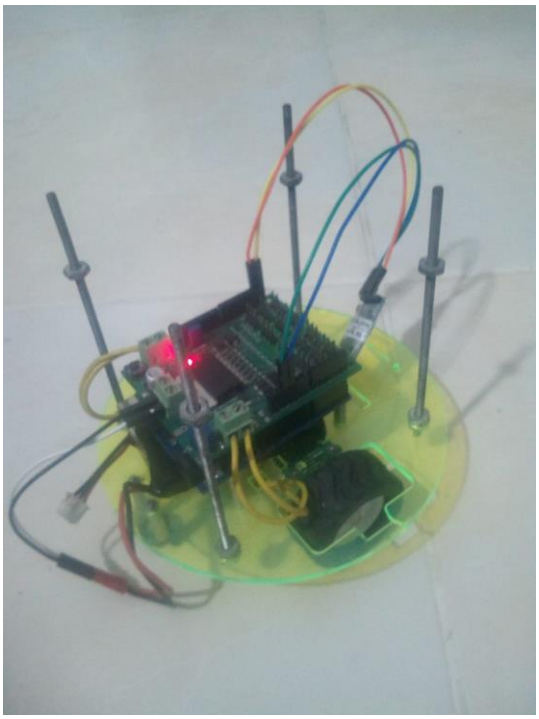


Figura 64. Arquitectura funcional del carrito

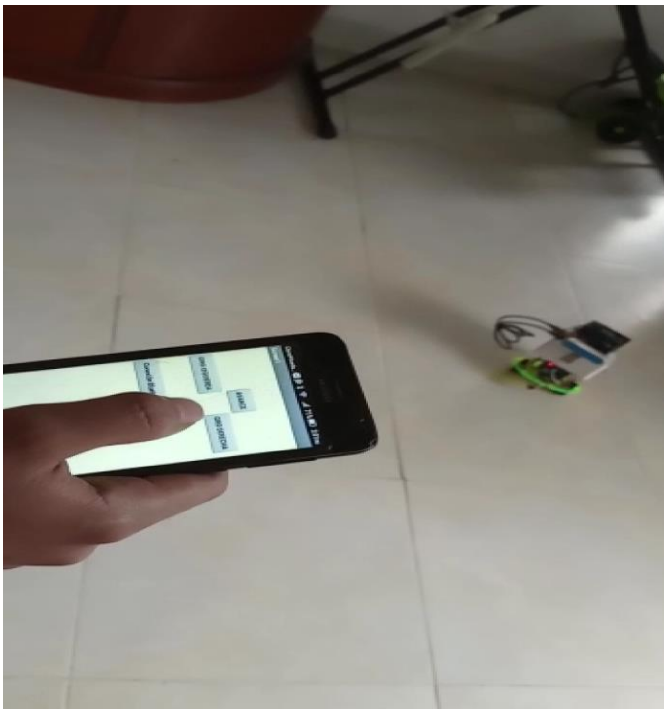


Figura 65. Prueba de funcionamiento

14 VALIDACIÓN EN UN ENTORNO ESTRUCTURADO

Figura 66. Reconstrucción a partir de ROS

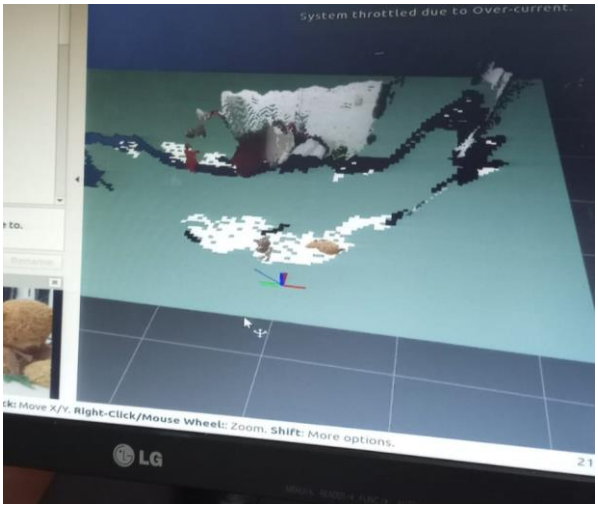


Figura 67. Mapa generado de un entorno estructurado

[para más detalle ver Prueba 4]

15 TRABAJO A FUTURO

Combinación de técnicas ROS-BAG y Matlab. A nivel académico permitirá estudiar las diferencias de las dos técnicas y comparar el nivel de precisión que estas llegasen a tener.

Unificar las partes del algoritmo trabajado a partir del software MATLAB permitiría la posibilidad de unificar cada uno de los objetivos en un único producto debido a su compatibilidad con Arduino y eliminaría la necesidad del uso de un sistema embebido y una cámara con sensores de profundidad.

El código de Matlab no permite una reconstrucción a tiempo real y debe de configurarse las imágenes de la cámara para hacer la reconstrucción, el propósito a futuro sería unificar el sistema de reconstrucción en Matlab empleando las imágenes a tiempo real generadas por la cámara y hacer una reconstrucción alternativa para realizar comparaciones con las reconstrucciones generadas en ROS.

El entorno de ROS funciona en la medida de que puede hacer una reconstrucción, esto es ajeno a las operaciones del robot. Según lo estudiado en el presente trabajo se es posible unificar el control del robot con el programa de ROS como se vio en las simulaciones de Gazebo y RVIZ. En la medida que la reconstrucción percibe la posición de la cámara, sería posible ubicar un robot en su lugar el cual puede ser operado con los códigos diseñados en Python y de esta forma unificar la interfaz de operación.

16 CONCLUSIONES

- La cámara Intel D435i posee una gran precisión a la hora de realizar el cálculo de la disparidad entre imágenes lo cual la hace una herramienta muy útil que simplifica en gran medida la implementación del algoritmo
- Según lo experimentado, Windows provee funciones que le permiten ser una alternativa de desarrollo ya que por medio de Matlab y las LTS de soporte para Ubuntu permiten una implementación parcial de algunas simulaciones en ROS usando Visual Studio sin embargo no está completamente desarrollado esta máquina virtual por ende es indispensable como herramienta de práctica un ordenador con el sistema operativo de Ubuntu.
- ROS depende del tipo del sistema operativo ya que no todas las plataformas son compatibles con todas las versiones instaladas, esto se vio reflejado en que para el Ubuntu 18 de escritorio era compatible el ROS Noetic y para la tarjeta Jetson Nano que poseía el Ubuntu 17, solo era compatible el ROS Melodic.
- ROS permite sincronización a tiempo real con sus simulaciones, estas simulaciones pueden ser desde el movimiento de un robot simulado hasta la definición de un entorno para que el robot real evite esas zonas.
- Las aplicaciones prácticas de la adquisición de modelos 3D dependen en gran medida de coincidir características de los colores y formas contra las características de los objetos analizados para obtener un modelo firme de localización y mapeo.
- Las salidas de nuestro algoritmo en un modelo 3D globalmente consistente del entorno percibido, representado como una nube de puntos. Esta permite una proyección de los objetos vistos en las imágenes para recrear un mapa virtual tridimensional con distancias promedios.
- Matlab es una herramienta de diseño útil para diseñar y desglosar el algoritmo de tipo SLAM en sus diversas etapas, Sin embargo. El procesamiento de imagen por imagen y la dificultad impide el funcionamiento a tiempo real del mismo algoritmo dificultando sus aplicaciones enfocadas a un entorno real.
- La comunicación serial del HC05 permite un óptimo desempeño para la transmisión de datos de forma inalámbrica, sin embargo, el tiempo de respuesta es más largo comparado a las pruebas usando directamente el cable. Esto conlleva a una reducción de velocidad para impedir colisiones durante su funcionamiento.

17 BIBLIOGRAFÍA Y REFERENCIAS

Libros

[1] Baturone, A. O. (2001). Robotica, Manipuladores Y Robots Moviles (Spanish Edition). MARCOMBO, S.A.

[2] Alberola, Á. M., Gallego, G. M., & Maestre, U. G. (2019). Artificial Vision and Language Processing for Robotics. Packt Publishing.

[3] Kudriashov, A., Buratowski, T., Giergiel, M., & Malka, P. (2020). Slam Techniques Application for Mobile Robot in Rough Terrain: 87 (2020 ed.). Springer.

[4] Trucco, E., & Verri, A. (1998). Introductory Techniques for 3-D Computer Vision. Prentice Hall.

Artículos

[5] Cebollada, S., Payá, L., Flores, M., Peidro, A., & Reinoso, Ó. (2021). A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data. Expert Syst. Appl., 167, 114195.

[6] Watanabe, K., Pathiranage, C.D., & Izumi, K. (2008). A Fuzzy Kalman Filter Approach to the SLAM Problem of Nonholonomic Mobile Robots. IFAC Proceedings Volumes, 41, 4600-4605.

[7] H. Jo, S. Jo, E. Kim, C. Yoon and S. Jun, "3D FastSLAM algorithm with Kinect sensor," 2014 Joint 7th International Conference on Soft Computing and Intelligent Systems (SCIS) and 15th International Symposium on Advanced Intelligent Systems (ISIS), 2014, pp. 214-217, doi: 10.1109/SCIS-ISIS.2014.7044862.

[8] Payá, L., Gil, A., & Reinoso, Ó. (2017). A State-of-the-Art Review on Mapping and Localization of Mobile Robots Using Omnidirectional Vision Sensors. J. Sensors, 2017, 3497650:1-3497650:20.

[9] H. Bavle, P. De La Puente, J. P. How and P. Campoy, "VPS-SLAM: Visual Planar Semantic SLAM for Aerial Robotic Systems," in IEEE Access, vol. 8, pp. 60704-60718, 2020, doi: 10.1109/ACCESS.2020.2983121.

[10] J. P. M. Covolan, A. C. Sementille and S. R. R. Sanches, "A mapping of visual SLAM algorithms and their applications in augmented reality," 2020 22nd

Symposium on Virtual and Augmented Reality (SVR), 2020, pp. 20-29, doi: 10.1109/SVR51698.2020.00019.

[11] S. Shi, W. Zhou and S. Liu, "An Efficient Multi-Robot 3D SLAM Algorithm," 2017 IEEE 7th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER), 2017, pp. 1200-1205, doi: 10.1109/CYBER.2017.8446394.

[12] S. Cui, M. Chen, L. He, Y. Zhang, M. Sun and X. Duan, "Design of Mobile Robot Interaction System Based on Slam Particle Filter," 2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2021, pp. 93-97, doi: 10.1109/IAEAC50856.2021.9390996.

[13] J. Ruan, B. Li, Y. Wang and Z. Fang, "GP-SLAM+: real-time 3D lidar SLAM based on improved regionalized Gaussian process map reconstruction," *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5171-5178, doi: 10.1109/IROS45743.2020.9341028.

[14] R. Maxence, H. Uchiyama, H. Kawasaki, D. Thomas, V. Nozick and H. Saito, "Mobile Photometric Stereo with Keypoint-Based SLAM for Dense 3D Reconstruction," *2019 International Conference on 3D Vision (3DV)*, 2019, pp. 574-582, doi: 10.1109/3DV.2019.00069.

Otras referencias:

[15] K. Melbouci, S. N. Collette, V. Gay-Bellile, O. Ait-Aider and M. Dhome, "Model based RGBD SLAM," 2016 IEEE International Conference on Image Processing (ICIP), 2016, pp. 2618-2622, doi: 10.1109/ICIP.2016.7532833.

[16] *DEPTH AND Semantic Segmentation Visualization Using Unreal Engine Simulation- MATLAB & Simulink- MathWorks América Latina MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink - MATLAB & Simulink [página web]. Disponible en Internet: <https://la.mathworks.com/help/driving/ug/visualize-depth-semantic-segmentation-3d-simulation.html>*

[17] CAMERA SENSOR model with lens in 3D simulation environment - Simulink- MathWorks América Latina MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink - MATLAB & Simulink [página web]. Disponible en Internet: <https://la.mathworks.com/help/driving/ref/simulation3dcamera.html>.

[18] *RUN LINUX GUI apps with WSL Developer tools, technical documentation, and coding examples | Microsoft Docs [página web]. [Consultado el 23, junio, 2022]. Disponible en Internet: <https://docs.microsoft.com/en-us/windows/wsl/tutorials/gui-apps>.*

[19] [*RESUELTA] COMMAND-LINE | Puede ' t ejecutar ni archivos. Preguntas de servidores en español. Comunidad de hispanos - EnMiMaquinaFunciona.com [página web]. Disponible en Internet: <https://www.enmimaquinafunciona.com/pregunta/46924/puede-39-t-ejecutar-ni-archivos-obtener-permiso-denegado>.*

[20] *FIXED FRAME [map] does not exist · Issue #1669 · cartographer-project/cartographer_ros. GitHub [página web]. Disponible en Internet: <https://github.com/cartographer-project/cartographer_ros/issues/1669>.*

[21] *FIRMWARE FOR Intel® RealSense™ D400 Product Family. Intel [página web. Disponible en Internet: <https://www.intel.com/content/www/us/en/download/19242/27522/firmware-for-intel-realsense-d400-product-family.html?v=t>.*

[22] *IN UBUNTU 18.04, using melodic ROS, cannot install octomap_rviz_plugins · Issue #15 · floatlazer/semantic_slam [Anónimo]. GitHub [página web]. [Consultado el 23, junio, 2022]. Disponible en Internet: <https://github.com/floatlazer/semantic_slam/issues/15>.*

[23] *SLAM WITH D435i · IntelRealSense/realsense-ros Wiki [Anónimo]. GitHub [página web]. [Consultado el 23, junio, 2022]. Disponible en Internet: <https://github.com/IntelRealSense/realsense-ros/wiki/SLAM-with-D435i>.*

[24] VARGAS MUÑOZ, Jorge Leonardo y PEREZ SALAZAR, Pedro Elias. DISEÑO DE UN ROBOT MÓVIL CON TRACCIÓN DIFERENCIAL PARA SEGUIMIENTO DE TRAYECTORIAS. CICI - Unillanos [página web]. Disponible en Internet: <https://cici.unillanos.edu.co/media2020/memorias/CICI_2020_paper_125.pdf>.

[25] PIÑERA GARCÍA, J., et al. Diseño de un robot móvil con modelo cinemático Ackermann - PDF Free Download. Le proporcionamos las herramientas cómodas y gratuitas para publicar y compartir la información. [página web]. Disponible en Internet: <https://docplayer.es/83663588-Diseno-de-un-robot-movil-con-modelo-cinematico-ackermann.html>.

[26] GUIADO DE un robot móvil con cinemática de triciclo [Anónimo]. BIOMEC - Biomechanical Engineering Lab - UPC Barcelona [página web]. Disponible en Internet: <https://biomec.upc.edu/wp-content/uploads/2016/09/Batlle-CNIM2004-Guiado-de-un-robot-movil-con-cinematica-de-triciclo.pdf>.

[27] HOW-TO: GETTING IMU data from D435i and T265. Intel® RealSense™ Depth and Tracking Cameras [página web]. [Consultado el 29, noviembre, 2022]. Disponible en Internet: <https://www.intelrealsense.com/how-to-getting-imu-data-from-d435i-and-t265/>.

[28] THE KITTI Vision Benchmark Suite [Anónimo]. Andreas Geiger [página web]. Disponible en Internet: <https://www.cvlibs.net/datasets/kitti/eval_odometry.php>.

[29] PYTHON. Intel® RealSense™ Developer Documentation [página web]. Disponible en Internet: <https://dev.intelrealsense.com/docs/python2>.

[30] GITHUB - cgarg92/Stereo-visual-odometry. Disponible en Internet: <https://github.com/cgarg92/Stereo-visual-odometry>.

[31] SCARAMUZZA, Anna. Algorithms for Projectivity and Extremal Classes of a Smooth Toric Variety. En: Experimental Mathematics [en línea]. Enero, 2009. vol. 18, no. 1, p. 71-84. Disponible en Internet: <https://doi.org/10.1080/10586458.2009.10128887>. ISSN 1944-950X.

[32] FRAUNDORFER, Friedrich y SCARAMUZZA, Davide. Visual Odometry: Part II: Matching, Robustness, Optimization, and Applications. En: IEEE Robotics & Automation Magazine [en línea]. Junio, 2012. vol. 19, no. 2. p. 78-90. Disponible en Internet: <https://doi.org/10.1109/mra.2012.2182810>. ISSN 1070-9932.

[33] OCTOMAP 3D scan dataset - Arbeitsgruppe: Autonome Intelligente Systeme [Anónimo]. Herzlich willkommen! - Arbeitsgruppe: Autonome Intelligente Systeme [página web]. Disponible en Internet: <http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/>.

[34] Robotics and Perception Group [página web]. Disponible en Internet: <https://rpg.ifi.uzh.ch/docs/VO_Part_I_Scaramuzza.pdf>.

[35] PROGRAMMING ROBOTS with ROS and ROS2 using MATLAB and Simulink Video. MathWorks - Creadores de MATLAB y Simulink - MATLAB y Simulink - MATLAB & Simulink]. Disponible en Internet: <https://la.mathworks.com/videos/programming-robots-with-ros-and-ros2-using-matlab-and-simulink-1600976607866.html?fbclid=IwAR07cEf_K666Rip_GwFvZb9yda7S_EyeG6IdPiqtXv8Jdfz1FskvQagJtWk>.

18 ANEXOS

PRUEBAS DE IMPLEMENTACIÓN:

[Prueba 1] Puesta en marcha del robot

https://www.youtube.com/watch?v=lhX0cp3ywKA&ab_channel=WILLIAMFERNANDOQUINTEROQUINTANA

[Prueba 2] Reconstrucción SLAM 3D a partir de Matlab

https://www.youtube.com/watch?v=eGKMv0mrQIQ&ab_channel=WILLIAMFERNANDOQUINTEROQUINTANA

[Prueba 3] Mapa de profundidad vs RGB - Puesta en marcha del carrito.

https://www.youtube.com/watch?v=XSUwJCNMjmc&ab_channel=WILLIAMFERNANDOQUINTEROQUINTANA

[Prueba 4] Resultados de reconstrucción 3D en ROS

https://www.youtube.com/watch?v=3ZW_cyj6qzY&ab_channel=WILLIAMFERNANDOQUINTEROQUINTANA

[Prueba 6] Conexión a ROS desde un servidor diferente

https://www.youtube.com/shorts/DshQQX_3d3g

[Prueba 7] Prueba de control de dirección de los motores

https://www.youtube.com/shorts/YXDPvuITgyc

[Prueba 8] Validación del funcionamiento de la interfaz

https://www.youtube.com/shorts/yuJUNHVPjug

HOJA DE DATOS USADAS:

[Datasheet 1] DYNAMOTION V3
[Datasheet 2] Intel RealSense D400

CONFIGURACIONES ADICIONALES Y OTROS RESULTADOS:

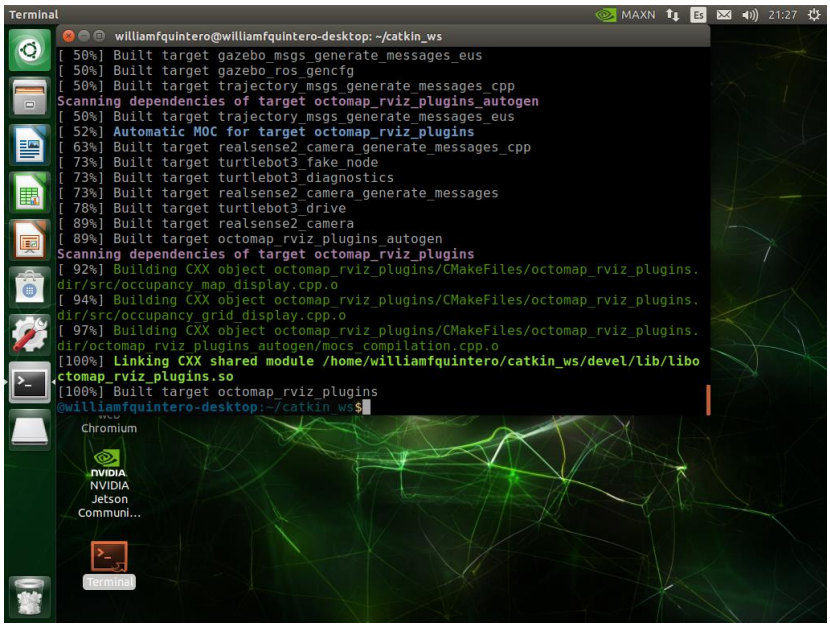


Figura 68. Configuración del Catkin make

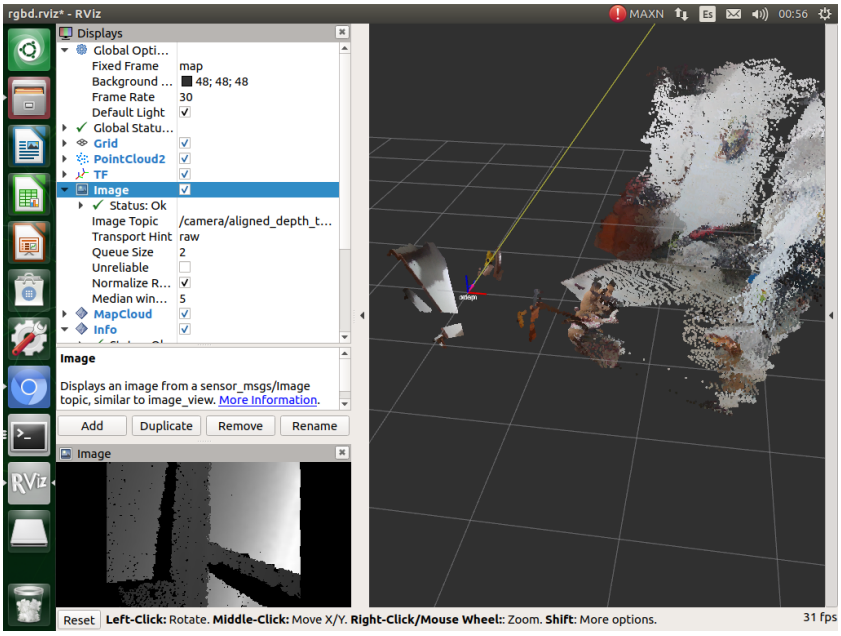


Figura 69. Reconstrucción de un entorno visto por la cámara en ROS

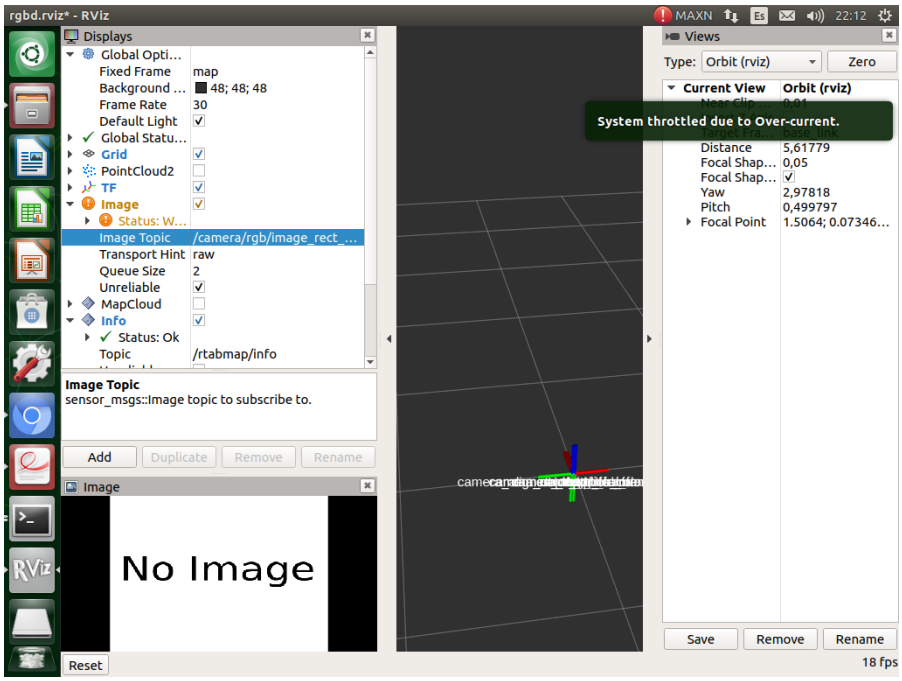


Figura 70. Problemas para reconocimiento de la cámara

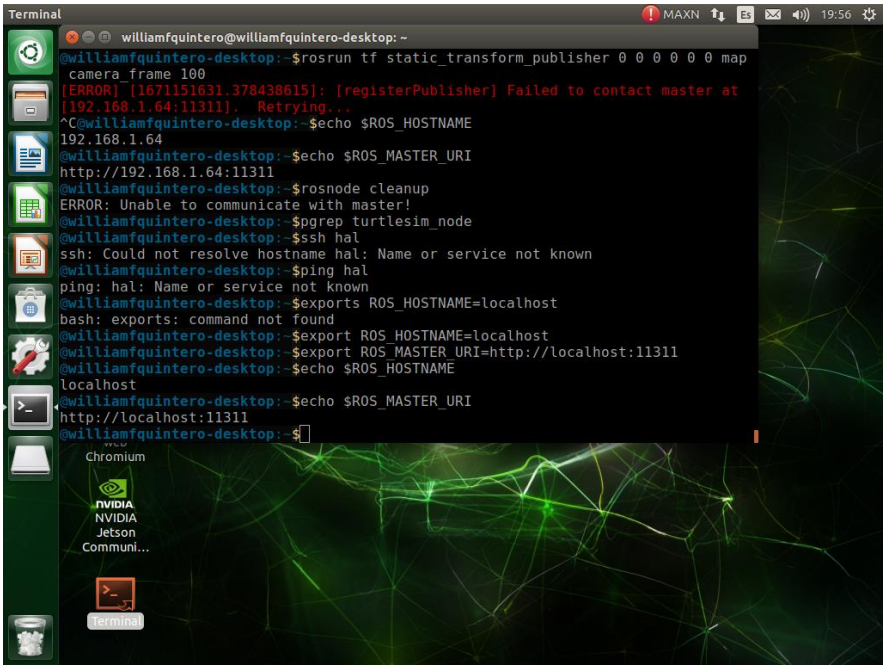


Figura 71. Errores al reconocer la comunicación maestro-esclavo

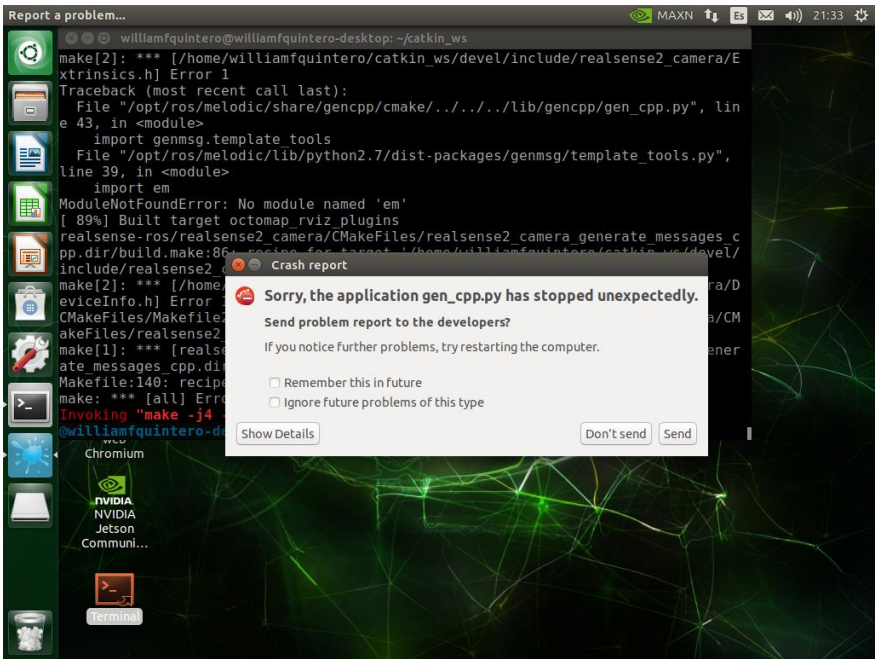


Figura 72. Errores debido a actualizaciones e incompatibilidades

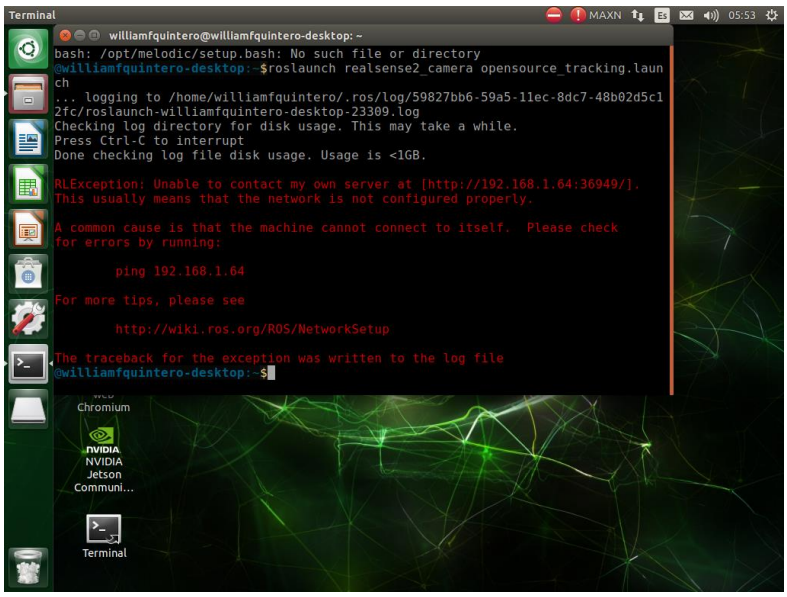


Figura 73. Problemas de comunicación con servidor

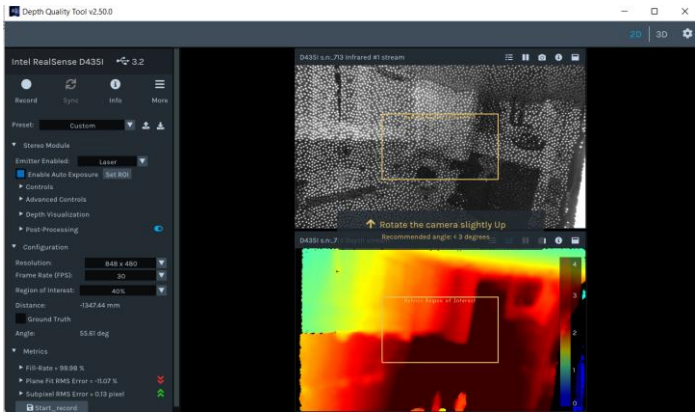


Figura 74. Generación de nube de puntos a partir de la cámara 435i

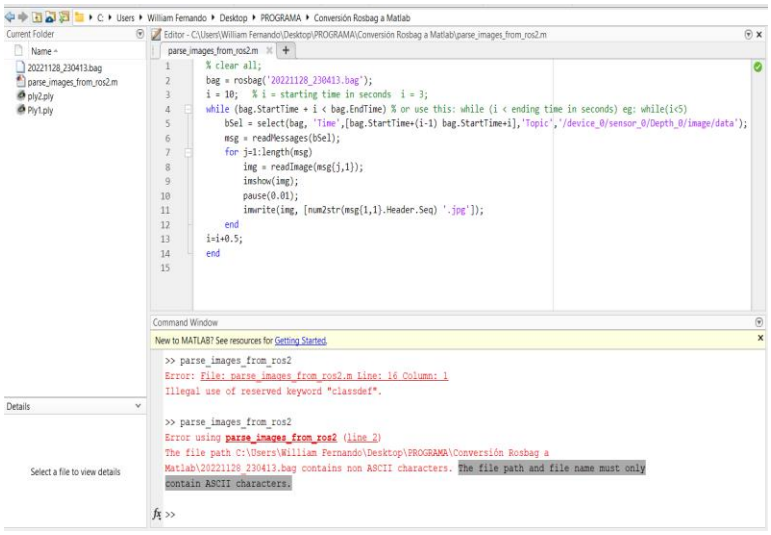


Figura 75. Errores de formato para reconocimiento rosbag

Figura 76. Resultados control del robot a través del monitor serial de Arduino

EXTRACIÓN Y COMPARACIÓN DE PUNTOS ENTRE DOS IMAGENNES

```
rotation_inv = 1;
rotation_img2_deg = 60;

num_scales = 3; % Scales per octave.
num_octaves = 5; % Number of octaves.
sigma = 1.0;
contrast_threshold = 0.04;
image_file_1 = '000000 (1).png';
image_file_2 = '000000 (2).png';
rescale_factor = 0.3; % Rescaling of the original image for speed.

left_img = getImage(image_file_1, rescale_factor);
right_img = getImage(image_file_2, rescale_factor);

% to test rotation invariance of SIFT

if rotation_img2_deg ~= 0
   right_img = imrotate(right_img, rotation_img2_deg);
end

images = {left_img, right_img};

kpt_locations = cell(1, 2);
descriptors = cell(1, 2);

for img_idx = 1:2
   image_pyramid = computeImagePyramid(images{img_idx}, num_octaves);
   blurred_images = computeBlurredImages(image_pyramid, num_scales, sigma);
   DoGs = computeDoGs(blurred_images);
   tmp_kpt_locations = extractKeypoints(DoGs, contrast_threshold);
   [descriptors{img_idx}, kpt_locations{img_idx}] =...
      computeDescriptors(blurred_images, tmp_kpt_locations, rotation_inv);
end

indexPairs = matchFeatures(descriptors{1}, descriptors{2},...
   'MatchThreshold', 100, 'MaxRatio', 0.7, 'Unique', true);
% Flip row and column to change to image coordinate system.
% Before         Now
% -----> y       -----> x
% |              |
% |              |
% ˅              ˅
% x              y
kpt_matched_1 = fliplr(kpt_locations{1}(indexPairs(:,1), :));
```

```
kpt_matched_2 = fliplr(kpt_locations{2}(indexPairs(:,2), :));

figure; ax = axes;
showMatchedFeatures(images{1}, images{2}, kpt_matched_1, kpt_matched_2, ...
    'montage','Parent',ax);
title(ax, 'Candidate point matches');
legend(ax, 'Matched points 1','Matched points 2');
```

MODELO DE TRIÁNGULACIÓN

```
addpath('8point/');
addpath('triangulation/');
addpath('plot/');

img = imread('C:\Users\William Fernando\Desktop\3D Point Cloud Generation\Two-view Geometry\code\data\0001.png');
img_2 = imread('C:\Users\William Fernando\Desktop\3D Point Cloud Generation\Two-view Geometry\code\data\0002.png');

K = [1379.74 0 760.35
    0 1382.08 503.41
    0 0 1 ];

%% Load outlier-free point correspondences

p1 = load('C:\Users\William Fernando\Desktop\3D Point Cloud Generation\Two-view Geometry\code\data\matches0001.txt');
p2 = load('C:\Users\William Fernando\Desktop\3D Point Cloud Generation\Two-view Geometry\code\data\matches0002.txt');

p1 = [p1;ones(1,length(p1))];
p2 = [p2;ones(1,length(p2))];

%% Estimate the essential matrix E using the 8-point algorithm

E = estimateEssentialMatrix(p1, p2, K, K);

%% Extract the relative camera positions (R,T) from the essential matrix

% Obtain extrinsic parameters (R,t) from E
[Rots,u3] = decomposeEssentialMatrix(E);

% Disambiguate among the four possible configurations
[R_C2_W,T_C2_W] = disambiguateRelativePose(Rots,u3,p1,p2,K,K);

% Triangulate a point cloud using the final transformation (R,T)
```

```
M1 = K * eye(3,4);
M2 = K * [R_C2_W, T_C2_W];
P = linearTriangulation(p1,p2,M1,M2);

%% Visualize the 3-D scene
figure(1),
subplot(1,3,1)

% R,T should encode the pose of camera 2, such that M1 = [I|0] and M2=[R|t]

% P is a [4xN] matrix containing the triangulated point cloud (in
% homogeneous coordinates), given by the function linearTriangulation
plot3(P(1,:), P(2,:), P(3,:), 'o');

% Display camera pose

plotCoordinateFrame(eye(3),zeros(3,1), 0.8);
text(-0.1,-0.1,-0.1,'Cam 1','fontsize',10,'color','k','FontWeight','bold');

center_cam2_W = -R_C2_W'*T_C2_W;
plotCoordinateFrame(R_C2_W',center_cam2_W, 0.8);
text(center_cam2_W(1)-0.1,  center_cam2_W(2)-0.1,  center_cam2_W(3)-0.1,'Cam
2','fontsize',10,'color','k','FontWeight','bold');

axis equal
rotate3d on;
grid

% Display matched points
subplot(1,3,2)
imshow(img,[]);
hold on
plot(p1(1,:), p1(2,:), 'ys');
title('Image 1')

subplot(1,3,3)
imshow(img_2,[]);
hold on
plot(p2(1,:), p2(2,:), 'ys');
title('Image 2')
```

GENERACIÓN DE RECONSTRUCCIÓN A PARTIR DE LA MATRIZ K

```
% Scaling down by a factor of 2, otherwise too slow.
left_img = imresize(imread('..\data\left\000000.png'), 0.5);
```

```
right_img = imresize(imread('..\data\right\000000.png'), 0.5);
K = load('C:\Users\William Fernando\Desktop\3D Point Cloud Generation\data\K.txt');
K(1:2, :) = K(1:2, :) / 2;

poses = load('C:\Users\William Fernando\Desktop\3D Point Cloud Generation\data\poses.txt');

% Given by the KITTI dataset:
baseline = 0.54;

% Carefully tuned by the TAs:
patch_radius = 5;
min_disp = 5;
max_disp = 50;
xlims = [7 20];
ylims = [-6 10];
zlims = [-5 5];

%% Parts 1, 2 and 4: Disparity on one image pair

tic;
disp_img = getDisparity(...
    left_img, right_img, patch_radius, min_disp, max_disp);
toc
figure(1);
imagesc(disp_img);
axis equal;
axis off;

%% Optional (only if fast enough): Disparity movie
warning(['Visualizing disparity over sequence! This is optional and' ...
 ' could take a lot of time (100 stereo pairs)!']);
figure(2);
maxi = 10;
for i = 0:maxi
    l = imresize(imread(sprintf('../data/left/%06d.png',i)), 0.5);
    r = imresize(imread(sprintf('../data/right/%06d.png',i)), 0.5);
    disp_img_i = getDisparity(...
        l, r, patch_radius, min_disp, max_disp);
    imagesc(disp_img_i);
    axis equal;
    axis off;
    title([num2str(i) ' of ' num2str(maxi)]);
    pause(0.01);
```

```
end

%% Part 3: Create point cloud for first pair

[p_C_points, intensities] = disparityToPointCloud(...
    disp_img, K, baseline, left_img);
% From camera frame to world frame:
p_F_points = [0 -1 0; 0 0 -1; 1 0 0]^-1 * p_C_points(:, 1:10:end);

figure(3);
scatter3(p_F_points(1, :), p_F_points(2, :), p_F_points(3, :), ...
    20 * ones(1, length(p_F_points)), ...
    repmat(single(intensities(1:10:end))'/255, [1 3]), 'filled');
axis equal;
axis([0 30 ylims zlims]);
axis vis3d;
grid off;
xlabel('X');
ylabel('Y');
zlabel('Z');

%% Part 4: Accumulate point clouds over sequence and write PLY

warning(['Accumulating point cloud over 100 stereo pairs! This can ' ...
    'take a lot of time (~an hour or more)! The progress bar can take ' ...
    'several seconds to update.']);

all_points = [];
all_intensities = [];

% Shorten to get a point cloud faster.
image_range = 0:99;

h = waitbar(0, 'Accumulating point clouds...');

for i = image_range
    l = imresize(imread(sprintf('../data/left/%06d.png',i)), 0.5);
    r = imresize(imread(sprintf('../data/right/%06d.png',i)), 0.5);
    disp_img = getDisparity(...
        l, r, patch_radius, min_disp, max_disp);
    [p_C_points, intensities] = disparityToPointCloud(...
        disp_img, K, baseline, l);
    % From camera frame to world frame:
    R_C_frame = [0 -1 0; 0 0 -1; 1 0 0];
    p_F_points = R_C_frame^-1 * p_C_points;
```

```
    % Use only points within limits:
    filter = ...
        (p_F_points(1, :) > xlims(1)) & (p_F_points(1, :) < xlims(2)) & ...
        (p_F_points(2, :) > ylims(1)) & (p_F_points(2, :) < ylims(2)) & ...
        (p_F_points(3, :) > zlims(1)) & (p_F_points(3, :) < zlims(2));
    p_F_points = p_F_points(:, filter);
    intensities = intensities(:, filter);

    T_W_C = reshape(poses(i+1, :), 4, 3)';
    T_W_F = T_W_C * [R_C_frame zeros(3, 1); zeros(1, 3) 1];
    all_points = [all_points ...
        (T_W_F(1:3, 1:3) * p_F_points + T_W_F(1:3, end))];
    all_intensities = [all_intensities intensities];

    if 0
        figure(4);
        scatter3(all_points(1, :), all_points(2, :), all_points(3, :), ...
            20 * ones(1, length(all_points)), ...
            repmat(single(all_intensities)'/255, [1 3]), 'filled');
        axis equal;
        axis vis3d;
        grid off;
        pause(0.01);
    end

    waitbar(i/image_range(end), h);
end

close(h);

file = fopen('points.ply', 'w');
fprintf(file, ['ply\nformat ascii 1.0\nelement vertex %d\nproperty '...
    'double x\nproperty double y\nproperty double z\nproperty uchar red'...
    '\nproperty uchar green\nproperty uchar blue\nend_header\n'], ...
    length(all_points));
fclose(file);
dlmwrite('points.ply', ...
    [all_points' repmat(single(all_intensities)', [1 3])], ...
    '-append', 'delimiter', ' ');
```

CÓDIGOS PARA CONTROL DE MOTORES:

```
//#include <SoftwareSerial.h>
//
//SoftwareSerial miBT(10, 11);
```

```
/*

Pines Arduino

D5      Control Motor A un sentido
D10     Control Motor A un sentido contrario
D6      Control Motor B un sentido
D9      Control Motor A un sentido contrario

Partes
- Arduino UNO
- Dynamotion Shield
- Dos motores DC

Descripción
Se definen las variables y pines de salida
el programa lee las variable de posicion del Joystick y de acuerdo al valor sensado
mueve los motores a un lado o al otro

*/


//int val1;   //Se crea la variale tipo entero Val1
//int val2;   //Se crea la variale tipo entero Val2
int numcar=0;
int numcar2=0;
int numcar3=0;
int numcar4=0;

void setup()
{
//Los pines 10 y 5 controlan el motor A, los pines 6 y 9 controlan el motor B

Serial.begin(9600);

pinMode(10, OUTPUT);    // Se define el pin 10 como salida
pinMode(5, OUTPUT);     // Se define el pin 5 como salida
pinMode(6, OUTPUT);     // Se define el pin 6 como salida
pinMode(9, OUTPUT);     // Se define el pin 9 como salida

digitalWrite(10, LOW);   //Se inician los motores apagados
digitalWrite(5, LOW);
digitalWrite(6, LOW);
digitalWrite(9, LOW);
}
```

```
void loop() {
// if (miBT.available()) {  // Lee BT y envía a Arduino
  //serial.write(miBT.read());

 if(Serial.available() > 0) {
  //Serial.println("Digite 8 para ir hacia adelante, 6 para giro a la derecha, 4 para giro
a la izquierda y 2 para detenerse ");
  int nun=Serial.parseInt();
//  int nun=miBT.read();

  if (nun==8){
      numcar=1;
  }
  if (nun==4){
      numcar2=1;
  }
  if (nun==2){
      numcar3=1;
  }
  if (nun==6){
      numcar4=1;
  }
//  miBT.write(Serial.read()); //Lee Arduino y envía a BT
 }
// numcar = Serial.parseInt();
// Serial.println(numcar);
// numcar2 = Serial.parseInt();
// Serial.println(numcar2);
// numcar3 = Serial.parseInt();
// Serial.println(numcar3);
// numcar4 = Serial.parseInt();
// Serial.println(numcar4);
// }
//if (val1 >550) {          //Si el valor leido es superior a 550 el motor A gira en sentido
horario
//  digitalWrite(10, LOW);
//  digitalWrite(5, HIGH);
//}
//
//else if (val1 <500) {             //Si el valor leido es inferior a 500 el motor A gira en
sentido antihorario
//  digitalWrite(10, HIGH);
//  digitalWrite(5, LOW);
//}
```

```
//else
//{
//  digitalWrite(10, LOW);   // Motor A apagado
//  digitalWrite(5, LOW);
//}
//
//// logica para motore B
//
//if (val2 >550) {          //Si el valor leido es superior a 550 el motor B gira en sentido horario
//  digitalWrite(9, LOW);
//  digitalWrite(6, HIGH);
//}
//
//else if (val2 <500) {           //Si el valor leido es inferior a 500 el motor B gira en sentido antihorario
//  digitalWrite(9, HIGH);
//  digitalWrite(6, LOW);
// }
//else {
//  digitalWrite(9, LOW);   // Motor B apagado
//  digitalWrite(6, LOW);
//}}

//D5   D10   D6    D9
//0     0     0    0          stop
//0     0     0    1          giro der
//0     0     1    0          giro izq
//0     0     1    1          stop
//0     1     0    0          giro izq
//0     1     0    1          atras
//0     1     1    0          giro izq
//0     1     1    1          stop
//1     0     0    0          giro der
//1     0     0    1          giro der
//1     0     1    0          adelante
//1     0     1    1          stop
//1     1     0    0          stop
//1     1     0    1          stop
//1     1     1    0          stop
//1     1     1    1          stop

// n n2 n3 n4
// 0 0  0  0    stop
// 0 0  0  1    izq
```

```
// 0 0  1  0    abajo
// 0 0  1  1    abajo izq
// 0 1  0  0    der
// 0 1  0  1    stop
// 0 1  1  0    abajo der
// 0 1  1  1    stop
// 1 0  0  0    adel
// 1 0  0  1    adel izq
// 1 0  1  0    stop
// 1 0  1  1    stop
// 1 1  0  0    adel der
// 1 1  0  1    stop
// 1 1  1  0    stop
// 1 1  1  1    stop

if (numcar==0 & numcar2==0 & numcar3==0 & numcar4==1 ) { // GIRO DER
  //Motor A
   digitalWrite(10, LOW);
   digitalWrite(5, 210);
  //Motor B
   digitalWrite(9, LOW);
   digitalWrite(6, LOW);
   delay (1000);
   numcar =0, numcar2=0, numcar3=0, numcar4=0;
}
if (numcar==0 & numcar2==1 & numcar3==0 & numcar4==0 ) { // GIRO IZQ
  //Motor A
   digitalWrite(10, LOW);
   digitalWrite(5, LOW);
  //Motor B
   digitalWrite(9, 215);
   digitalWrite(6, LOW);
   delay (1000);
   numcar =0, numcar2=0, numcar3=0, numcar4=0;
}
if (numcar==1 & numcar2==0 & numcar3==0 & numcar4==0 ) { // INICIA
  //Motor A
   digitalWrite(10, LOW); //horario
   digitalWrite(5, 210);
  //Motor B
   digitalWrite(9, 215); //antihorario
   digitalWrite(6, LOW);
   delay (1000);
   numcar =0, numcar2=0, numcar3=0, numcar4=0;
}
```

```
if (numcar==0 & numcar2==0 & numcar3==1 & numcar4==0 ) { // STOP
//  //Motor A
//   digitalWrite(10, HIGH);
//   digitalWrite(5, LOW);
//  //Motor B
//   digitalWrite(9, HIGH);
//   digitalWrite(6, LOW);
//   delay (1000);
//Motor A
   digitalWrite(10, LOW);
   digitalWrite(5, LOW);
//Motor B
   digitalWrite(9, LOW);
   digitalWrite(6, LOW);
   delay (1000);
   numcar =0, numcar2=0, numcar3=0, numcar4=0;
}
else {  numcar =0, numcar2=0, numcar3=0, numcar4=0;
  }
 }
```