UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Implementación de un sistema de visión por computadora integrable como módulo de ROS a una plataforma robótica móvil

Trabajo de graduación presentado por María Alejandra Samayoa Gómez
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

2023

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

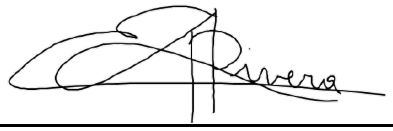
Implementación de un sistema de visión por computadora integrable como módulo de ROS a una plataforma robótica móvil

Trabajo de graduación presentado por María Alejandra Samayoa Gómez
para optar al grado académico de Licenciado en Ingeniería Mecatrónica

Guatemala,

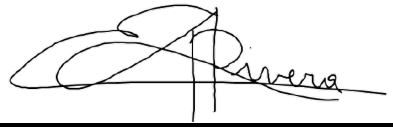
2023

Vo.Bo.:

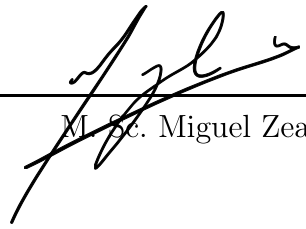
(f) 

Dr. Luis Alberto Rivera Estrada

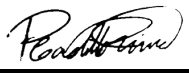
Tribunal Examinador:

(f) 

Dr. Luis Alberto Rivera Estrada

(f) 

M. Sc. Miguel Zea

(f) 

M. Sc. Pedro Iván Castillo

Fecha de aprobación: Guatemala, 4 de enero de 2023.

Prefacio

El siguiente proyecto surge de los beneficios que se le pueden agregar a un proyecto de robótica al agregar Visión por Computadora. A través del mismo, se le estará dotando al rover un grado más de autonomía, ya que podrá corroborar datos de su ubicación y tomar decisiones según esta información. El módulo de visión por computadora, además, queda integrable como módulo de ROS a cualquier proyecto que desee utilizar el sistema para realizar su control en el futuro.

Considero oportuno agradecerles a mis padres, Corina Gómez y David Samayoa, sin los cuales, junto a su amor y apoyo incondicional no me hubiera sido posible la oportunidad de estudiar esta carrera y haber llegado tan lejos. Todo su trabajo, esfuerzo y sacrificio me inspira cada día a ser una mejor persona. Quiero agradecerles también a mis dos hermanas: Emily y Fátima, que son la razón de muchas de las cosas que hago y mis mejores amigas. Gracias a mi familia por aguantarme en "modo universidad" y hacerme sentir que siempre tendré a alguien de mi lado.

Me gustaría agradecerles a todos mis compañeros y catedráticos que me acompañaron a lo largo de la carrera. En especial a mis amigos cercanos, con los que siempre encontré apoyo, consuelo y risas.

En especial me gustaría agradecerle a mi asesor, el Dr. Luis Rivera. Su apoyo, consejos y motivación siempre me alentaron a esforzarme por hacer lo que tenía que hacer y mucho más.

Índice

Prefacio v

Lista de figuras x

Lista de cuadros xi

Resumen xiii

Abstract xv

1. Introducción 1

2. Antecedentes 3
- 2.1. Cámara Kinect para visión por computadora . . . 3
- 2.2. Plataforma móvil . . . 4
- 2.3. Visión por computadora . . . 5

3. Justificación 7

4. Objetivos 9
- 4.1. Objetivo general . . . 9
- 4.2. Objetivos específicos . . . 9

5. Alcance 11

6. Marco teórico 13
- 6.1. Visión por computadora . . . 13
  - 6.1.1. Software utilizado para visión por computadora . . . 15
  - 6.1.2. Detección de marcadores ArUco . . . 15
- 6.2. Módulos de cámaras . . . 16
  - 6.2.1. JeVois Smart Machine Vision Camera . . . 16
  - 6.2.2. Raspberry Pi Camera . . . 17
  - 6.2.3. Microsoft Kinect . . . 17
- 6.3. ROS . . . 18

7. Preparación de la computadora Raspberry Pi **21**
7.1. Ubuntu Mate . . . 22
7.2. Ubuntu Desktop . . . 23
7.3. Conexión a Internet . . . 24

8. Módulos de visión por computadora **27**
8.1. Adaptación de las cámaras al sistema operativo . . . 27
8.1.1. Raspberry Cam . . . 27
8.1.2. JeVois . . . 28
8.2. Módulo detección marcadores ArUco . . . 28
8.2.1. Raspberry Cam . . . 28
8.2.2. JeVois . . . 31

9. Selección de módulo de cámara **33**

10. Microsoft Kinect **37**
10.1. Pruebas fallidas . . . 38
10.2. Pruebas exitosas . . . 38
10.2.1. Sobre Windows 11 en computadora . . . 38
10.2.2. Sobre Ubuntu Desktop en Raspberry Pi 4 . . . 41
10.3. Módulo de detección de marcadores ArUco . . . 43

11. Integración a ROS **45**

12. Características del módulo de visión **49**

13. Integración a plataforma robótica **55**
13.1. Marcadores . . . 55
13.2. Montadura sobre el robot . . . 56
13.3. Pruebas sobre el robot . . . 57
13.3.1. Prueba de identificación . . . 57
13.3.2. Aplicación de mapeo . . . 58

14. Conclusiones **61**

15. Recomendaciones **63**

16. Bibliografía **65**

17. Anexos **67**
17.1. Las imágenes creadas y sus guías . . . 67
17.2. Programación . . . 67
17.3. Videos de funcionamiento . . . 67

18. Glosario **69**

Lista de figuras

1. Robot móvil con sistema de control basado en visión de la cámara Kinect [2]. 4
2. Robot explorador que servirá como plataforma móvil [3]. 5

3. Historia de la visión por computadora [9]. 13
4. Ejemplo visión por computadora utilizando aprendizaje automático para reconocimiento [10]. 14
5. Descripción gráfica del proceso de control por visión [12]. 14
6. Ejemplo de marcador ArUco con identificación de 23 [14] 15
7. Imagen con marcadores identificados, [14] 16
8. Cámara JeVois, [15] 16
9. Cámara RaspiCam, [16] 17
10. Configuración de módulos dentro del Kinect [1]. 18

11. Vista del escritorio de la imagen creada para Raspberry Pi con Ubuntu Mate. 22
12. Imagen creada corriendo ROS desde la terminal. 23
13. Imagen creada con ROS funcionando. 23
14. Vista del escritorio de la imagen creada para Raspberry Pi 4 con Ubuntu Desktop. 24
15. ROS funcionando en imagen con Ubuntu Desktop. 24
16. Resultado de correr el icono pidiendo el input del usuario para conexión a una red nueva. 25
17. Archivo que modifica el programa con el formato correcto. 26

18. Imagen tomada por el programa de tablero de calibración. 29
19. Detección de esquinas funcionando sobre imagen tomada de tablero de calibración. 29
20. Prueba de información escrita sobre imagen e impresión de identificación en terminal. 30
21. Prueba de identificación impresa en terminal con video. 30
22. Funcionamiento de sesión de *screen* después de establecer parámetros. 31
23. Funcionamiento luego de llamar al *bash script*. 32
24. Funcionamiento del programa para extraer información desde el puerto serial. 32

25. Transmisión en vivo por medio de servidor Web utilizando la Raspberry Cam. 35

26. Kinect for Windows Desktop Toolkit v1.8 con ejemplos en C#. 39
27. Módulo de Kinect Explorer en la aplicación de Toolkit. 40
28. Módulo de Face Tracking Basics en la aplicación de Toolkit. 40
29. Resultado de correr el ejemplo *glview*. 41
30. Resultado de correr el ejemplo *chunkview*. 42
31. Resultado de correr el ejemplo *glpclview*. 42
32. Resultado de correr el ejemplo *regview*. 42
33. Funcionamiento del programa para extraer información desde el puerto serial. 43
34. Algoritmo de detección de esquinas para calibración de la cámara. 44
35. Detección de maracadores ArUco de imagen. 44

36. Información publicada en el tópico de ROS. 47

37. Prueba de profundidad. 50
38. Medias de desviaciones estándar de diferentes medidas. 51
39. Comparación de promedios de profundidades obtenidas con las medidas verdaderas. 51
40. Gráfica de error experimental de medida de profundidad. 51
41. Prueba de eje horizontal. 52
42. Rango de la cámara. 52
43. Dominio de la cámara. 53

44. Marcador ArUco que se utilizó para las pruebas. 56
45. Soporte colocado en la parte de enfrente del robot. 56
46. Prueba de identificación de marcadores con la cámara montada en el robot. . 57
47. Versión final del robot explorador con la cámara montada. 57
48. Versión final del robot explorador con la cámara montada. 58
49. Nodo de ROS funcionando durante prueba de identificación en el robot. . . . 58
50. Prueba de mapeo con la cámara montada y los marcadores 1 y 4. 59
51. Aplicación de mapeo graficado en RViz. 59

Lista de cuadros

1. Resumen de características de ambas cámaras. 35

2. Medias de desviaciones estándar de las medidas obtenidas de la cámara. . . . 50

Resumen

Uno de los proyectos que se elaboró dentro de la Universidad del Valle de Guatemala en el año 2022 fue una plataforma robótica móvil. El siguiente trabajo consistió en la implementación de un sistema de visión por computadora para esta plataforma. Lo primero que se realizó fue un estudio comparativo entre dos módulos de cámara, la JeVois A33 y la Raspberry Pi Camera, para definir el más adecuado entre las opciones. Para esto se realizaron varias pruebas y se eligió uno para ser implementado. Además, se realizaron pruebas con el módulo de cámara Kinect para Windows para determinar si el módulo sería implementable en la plataforma. El módulo elegido fue adaptado como nodo de ROS. De este modo fue integrado a la plataforma robótica. Se realizaron pruebas básicas con algoritmos de visión por computadora para comprobar su funcionamiento adecuado.

Abstract

One of the graduation projects done by the Universidad del Valle de Guatemala in 2022 was a mobile robotic platform. The following project consists in the addition of a computer vision module to said platform. To do so, a trade study was performed between two cameras, the JeVois A33 and a Raspberry Pi Camera, to determine which one is most suitable. The performance of these two cameras was compared in various tests and one was chosen to be implemented. Additionally, tests were done on the Microsoft Kinect for Windows to determine if it could be implemented on the platform. The selected camera was implemented as a ROS node. It is in this state that it was integrated into the rest of the robotic platform. Basic tests with computer vision algorithms were performed to verify the correct functioning.

CAPÍTULO 1

Introducción

La visión por computadora es uno de los campos crecientes de la inteligencia artificial que ha llegado a causar un gran impacto en el campo de la Robótica. La adición de visión por computadora a los proyectos de este campo ha permitido generar un control más eficiente, obtener más información sobre el entorno del robot y permitir la creación de sistemas más automáticos.

En el año 2022, uno de los proyectos de la Universidad del Valle fue la creación de una plataforma robótica móvil, el Rover UVG, que serviría como robot explorador y contaría con diferentes funciones unidas por el software: Robotics Operating System (ROS, por sus siglas en inglés). Este trabajo trata sobre la adición de la función de visión por computadora al rover.

Lo primero que se realizó fue definir una serie de experimentos que se realizarían en un ambiente controlado en la universidad para analizar el comportamiento de los diferentes módulos sobre la plataforma. Además, se definió el uso de la computadora Raspberry Pi: para el control de ROS y de los nodos.

Ya que se estaría trabajando la Raspberry Pi en las pruebas finales y en el desarrollo de los módulos, lo primero que se realizó para el proyecto fue determinar cuál sería el mejor sistema operativo para manejar tanto la versión de ROS como los demás módulos que se definieron.

Uno de los objetivos del proyecto fue la selección del módulo de cámara que se estaría colocando para realizar la visión por computadora. Para esto, se comparó el funcionamiento de la cámara JeVois A33 y de la Raspberry Pi Camera o RaspiCam.

El experimento que se definió para comprobar el funcionamiento del nodo de visión por computadora consistió en la identificación de diferentes "estaciones" definidas por un marcador ArUco. Por lo mismo, se trabajó un algoritmo de detección de marcadores ArUco en ambas cámaras.

Luego de ver el rendimiento de ambas, se eligió la cámara JeVois. Se creó un nodo de

ROS que estuviera enviando la información para el sistema general del robot y así se logró la integración con el resto del sistema.

Otro de los objetivos era probar la cámara Microsoft Kinect. Se realizaron pruebas con esta cámara en diferentes computadoras al igual que en sistemas operativos. Se logró identificar la manera para poder usarse en Windows 11 y sobre un sistema operacional de Linux montado en la Raspberry Pi. Sin embargo, se decidió no utilizar la cámara para esta iteración de la plataforma robótica.

CAPÍTULO 2

Antecedentes

El siguiente trabajo buscó expandir algunos trabajos realizados dentro de la Universidad del Valle de Guatemala e implementar la información de otras investigaciones realizadas en diferentes partes del mundo. En otros trabajos de investigación del exterior se han realizado estudios sobre la implementación de la cámara Kinect para realizar control de visión por computadora. Dentro de la universidad, se han trabajado proyectos de visión por computadora y de la plataforma robótica móvil. A continuación se describen los antecedentes de este proyecto.

2.1. Cámara Kinect para visión por computadora

La cámara Microsoft Kinect ha probado ser una herramienta útil y de bajo costo para realizar visión por computadora para la robótica. En el artículo *Enhanced Computer Vision With Microsoft Kinect Sensor: A Review*, los autores realizan una guía con diferentes algoritmos que se pueden utilizar para realizar la visión por computadora a tráves del módulo de Kinect [1]. Se describen las funciones de reconocimiento de posición, mapeo y de cámara 3D que se pueden lograr con el dispositivo.

En la tésis *Evaluation of Microsoft Kinect 360 and Microsoft Kinect One for robotics and computer vision applications* por Simone Zennaro [2], se logró realizar diferentes pruebas de visión específicamente para aplicarse para el control por visión para una plataforma robótica y comparar los módulos de la cámara Kinect 360 y Kinect One. La intención de este robot fue que lograra seguir a una persona por medio de un sistema de control basado en visión. En el trabajo, se desarrollan diferentes algoritmos para poder comparar reconocimiento facial, detección de contornos y percepción de profundidad. Se concluyó que ambas pueden ser implementadas, pero que la cámara de Kinect 360 funciona mejor en interiores y que la cámara Kinect One tiene mejor detalle y funcionamiento en el exterior por tener mejor contraste.

En la Figura 1 se muestra la plataforma robótica en la que se probaron las cámaras.

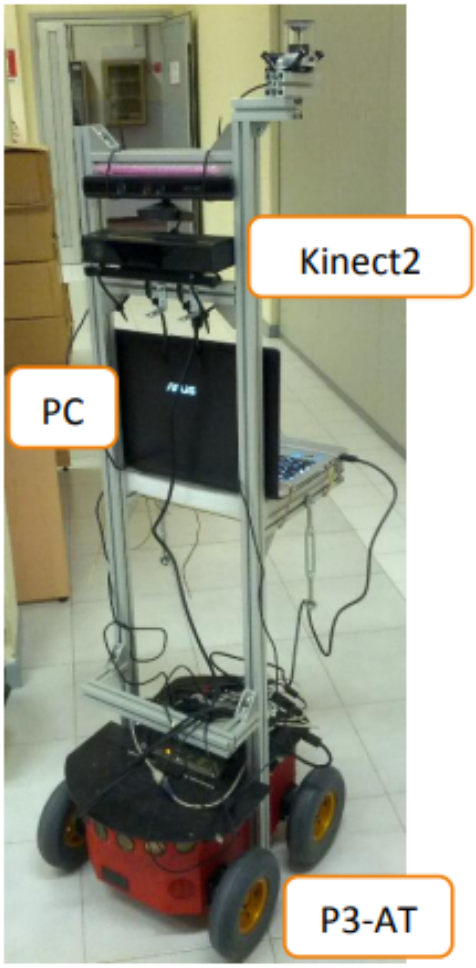


Figura 1: Robot móvil con sistema de control basado en visión de la cámara Kinect [2].

2.2. Plataforma móvil

Se han realizado diferentes iteraciones de un robot explorador en la Universidad del Valle a tráves de los años. La versión más reciente del robot se trabajó en el año 2021, donde se buscó trabajar en el desarrollo y mejora de la plataforma que no había tenido seguimiento desde el 2018. A través del trabajo de graduación de Héctor Sagastume [3] y de Javier Archila [4], se buscó obtener una plataforma que funcionaría a control remoto y contara con algunos sensores para su funcionamiento óptimo.

Sagastume cambió el sistema de transmisión de potencia, le agregó sensores infrarojos y de temperatura y realizó el diseño de la carcasa exterior. Por otro lado, Archila se enfocó en realizar el control del robot para que funcionara en modo de auto piloto. Para la lectura de los sensores y el sistema de auto piloto, se utilizó una Raspberry Pi 3 como computadora. Se obtuvo el robot que se muestra en la Figura 2 que funcionó a control remoto por medio de conexión al internet.

Este robot explorador no contó con un sistema de visión avanzado. Se utilizó un módulo

Figura 2: Robot explorador que servirá como plataforma móvil [3].

de cámara USB que se conectó a la Raspberry Pi 3 y el video se mandaba a través de la conexión de *bluetooth* que se realizó. Sin embargo, el video no formó parte del control que se realizaba para el movimiento del robot.

2.3. Visión por computadora

La visión por computadora no es un tema nuevo dentro de la universidad. Ya que es un tema con bastantes aplicaciones, han existido varios trabajos de graduación que la integran a otro proyecto, hacen una comparación entre diferentes módulos disponibles o continúan afinando métodos para realizar la visión por computadora. Entre los trabajos más recientes destacan algunos que serán de importancia para la aplicación que se le quiere dar a la visión por computadora en este proyecto.

En el año 2020, José Guerra, en su trabajo de graduación, utilizó la visión por computación aplicada a la robótica de enjambre [5]. Con esto, se buscó identificar las poses de los miembros del enjambre para tener un control de la posición de estos sobre una mesa de prueba. Además, realizó una comparación breve entre la programación orientada a objetos y el método multihilos para obtener la mejor visión. Esto lo realizó utilizando la plataforma Open CV y luego migrando la programación a Python.

José Ramírez continuó el estudio en el año 2021 [6], utilizando como base el trabajo realizado por Guerra. La aplicación en este caso buscaba detectar los obstáculos dentro del entorno y se buscaba crear una aplicación en la plataforma de Matlab que lograra el mapeo del área sobre la mesa de prueba Robotat. Se logró la migración de los algoritmos a Matlab y el mapeo del área sobre la mesa.

Otro trabajo de visión por computadora que será útil para el proyecto es el estudio realizado por Héctor Klée en el año 2021 [7]. Este consistió en un estudio comparativo entre diferentes módulos de visión para un sistema embebido: JeVois Smart Vision A33 y ESP32-CAM. Se implementó en un pequeño brazo robótico, con la intención de que se realizara un sistema de control basado en visión para que el brazo siguiera a un marcador. Se realizaron 8 diferentes pruebas con ambos módulos y se les otorgó una calificación según

su desempeño ante estas pruebas. Al final, se decidió utilizar el sistema JeVois A33. Además, se logró realizar el controlador para el brazo, utilizando la visión para reconocer el marcador específico.

Los trabajos de Ramírez y Guerra fueron limitados por la pandemia para realizar la cantidad de pruebas que se querían realizar, y sus aplicaciones consistieron en aplicaciones estacionarias. El trabajo de Klée, aunque se realizó para una aplicación móvil, se enfocó únicamente en la detección de un tipo de marcador y se aplicó la visión por computadora con un solo módulo de cámara.

CAPÍTULO 3

Justificación

La visión por computadora ha sido un campo creciente en los últimos años, demostrando su versatilidad y utilidad a través de diversas aplicaciones que se le han dado. Uno de los campos en donde se ha demostrado su utilidad es en el campo de la robótica, dónde se han generado aplicaciones para reconocer obstáculos, mejorar la interacción con los seres humanos y para realizar sistemas de control para el robot.

Este año se creó una plataforma robótica móvil, Rover UVG, que es un compendio de diferentes módulos sobre los cuales se podrán realizar diferentes pruebas y se juntan diferentes aplicaciones de la tecnología. Uno de estos módulos fue la visión por computadora que se le agregó con este trabajo de graduación. La plataforma que se había trabajado con anterioridad, [4], no contaba con un sistema de visión por computadora avanzado ya que dependía más de sus diferentes sensores que de la cámara que tenía implementada. Se busca mejorar la visión por computadora que tendrá la plataforma. En la universidad anteriormente se han trabajado implementaciones de visión por computadora a aplicaciones estáticas, tal como las mesas de prueba trabajadas en [6]. Adémas, se han realizado controladores a base de la visión por computadora en aplicaciones como un pequeño brazo robótico al igual que una comparación breve entre algunos módulos de cámaras, como fue el caso en [7].

El trabajo propuesto buscó agregar una parte importante a la plataforma que se estuvo trabajando este año, implementando un sistema de visión por computadora más completo. Se realizaron varias pruebas sobre la plataforma móvil, algo que se limitó en los trabajos anteriores.

A través de la comparación de los módulos Raspberry Pi Cam y JeVois A33, se realizaron diferentes pruebas que demuestran su funcionamiento. Tomando en consideración las pruebas que se realizaron y el montaje de la plataforma, se escogió el módulo de cámara JeVois. Este se implementó como nodo de ROS y se pudo integrar al control general de la plataforma. Se realizaron también pruebas para la cámara de Kinect para Windows para ver su utilidad en la plataforma y su compatibilidad con sistemas de Raspberry Pi y Linux. Se encontró qué versiones del soporte para la cámara eran las que se podrían utilizar para el proyecto.

Estas pruebas no solo detallan el funcionamiento de los tres módulos de cámaras, sino que enseñan usos que se les pueden dar en otros proyectos. Con el módulo de detección de marcadores ArUco en la cámara JeVois, se puede realizar un sistema de control más detallado en la plataforma y le agrega un nivel de independencia al robot. Como nodo de ROS, el sistema de visión se podría implementar en diferentes proyectos de robótica en el futuro.

CAPÍTULO 4

Objetivos

4.1. Objetivo general

Implementar un sistema de visión por computadora que pueda integrarse como módulo de ROS a una plataforma robótica móvil.

4.2. Objetivos específicos

- Evaluar y comparar diferentes módulos de cámara para definir el más adecuado para la plataforma robótica móvil.
- Adaptar el sistema de visión por computadora con la cámara seleccionada como módulo de ROS, para su integración a la plataforma robótica móvil.
- Adaptar la cámara Kinect como módulo de ROS, para su integración a la plataforma robótica móvil.
- Realizar pruebas básicas de algoritmos de visión por computadora para validar el sistema implementado.

CAPÍTULO 5

Alcance

Todas las pruebas realizadas en las cámaras fueron limitadas por los recursos que se iban a tener en la plataforma robótica. Por esto, lo primero que se realizó fue definir el sistema operativo a utilizar para que fuera compatible con la Raspberry Pi y con la versión de ROS que se estaría trabajando. Sobre el sistema operativo seleccionado y sobre la Raspberry Pi, ya se pudieron realizar las pruebas de las cámaras. Fue necesario investigar la manera de adaptar ambas cámaras sobre este nuevo sistema operativo y que realizaran el procesamiento adecuado. Con esto fue posible confirmar que el funcionamiento de estas fuera el que se deseaba en la plataforma.

Tomando en consideración que la cámara estaría montada sobre la misma Raspberry donde se estaría realizando el procesamiento de ROS, se decidió utilizar la cámara JeVois y que esta se comunicara por medio de un puerto serial. Es pertinente mencionar que se utilizó un modelo de Raspberry Pi 4 que cuenta con 8 GB de RAM, con una imagen que contaba con el sistema operativo Ubuntu: 20.04 para correr ROS2 FoxyFitzRoy.

Debido a los diferentes módulos que se montarían sobre la plataforma, los experimentos que se realizarían para comprobar el funcionamiento de todo se definieron en un área determinada en uno de los laboratorios de la universidad. El experimento que se definió para comprobar el funcionamiento del módulo de visión por computadora fue el de identificación de diferentes estaciones por medio de marcadores ArUco. Por ende, el algoritmo que se trabajó para comparar ambas cámaras fue el de detección de marcadores ArUco.

CAPÍTULO 6

Marco teórico

6.1. Visión por computadora

Según [8] la visión por computadora se puede definir como un campo de la inteligencia artificial que es un compendio de algoritmos y técnicas que mezclan un módulo de visión físico con la interpretación de la imagen por medio de programación y diferentes procesamientos que se le realizan a la imagen [8]. Este campo tiene una gran cantidad de aplicaciones en diferentes campos, en especial en el campo de la robótica. Es un campo que ha ido cambiando en grandes cantidades a lo largo de la historia, como se puede observar en la Figura 3.

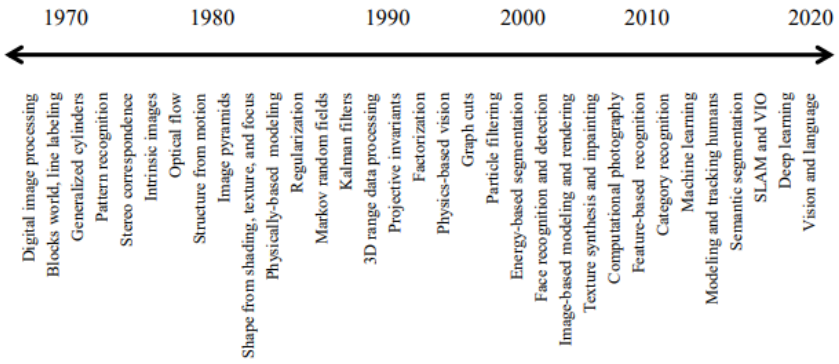


Figura 3: Historia de la visión por computadora [9].

En [9], se detalla el proceso de visión por computadora. La mayoría de cámaras realizan un proceso de capturar imágenes en 3 dimensiones y luego realizan la conversión correspondiente para obtener una imagen en 2D. A través de puntos y lineas en dos dimensiones se va formando, pixel por pixel, la imagen digital que replica la figura en 3D. Una gran parte de la industria de visión por computadora clásica se basa en mejorar la calidad de la imagen extraída a través de diferentes algoritmos [9]. Algunos algoritmos que se le atribuyen a la visión por computadora clásica son los filtros Gauseanos, detección de ejes, cambios de

saturación o color, entre otros.

La combinación de los campos de visión por computadora y el aprendizaje automático es algo que ha ido evolucionando en los últimos años y sigue en crecimiento. En esencia, los algoritmos de visión por computadora a base de aprendizaje automático tienen la intención de obtener datos y utilizar estos dentro de modelos para realizar predicciones [10]. En la industria se pueden ver en aplicaciones de reconocimiento facial, análisis de texto y clasificación de imágenes, entre otras. Este tipo de visión ha permitido aplicaciones de controladores de visión más complejas, como las que se manejan en los vehículos autónomos que reconocen diferentes señales, obstáculos y comandos y utilizan estas imágenes para reaccionar de la manera correcta.

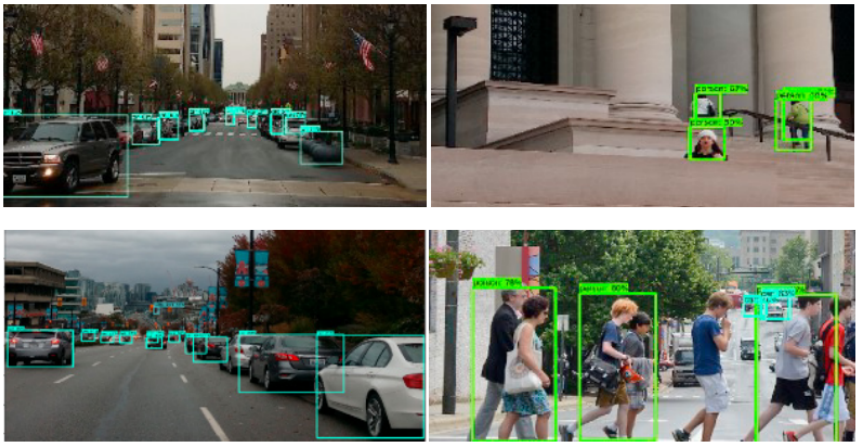


Figura 4: Ejemplo visión por computadora utilizando aprendizaje automático para reconocimiento [10].

En el campo de la robótica, la visión por computadora ha permitido realizar control por visión (*visual servoing* en inglés). Tal como indica el nombre, el control por visión permite control sobre el movimiento de un robot con retroalimentación del algún módulo de visión [11]. Esto se puede realizar de maneras diferentes según la información que se este recibiendo del procesamiento de la cámara y lo que se quiera lograr con el robot. Como se puede ver en la Figura 5, el proceso de control se realiza con la retroalimentación de la información de los rasgos que se buscan con la cámara.

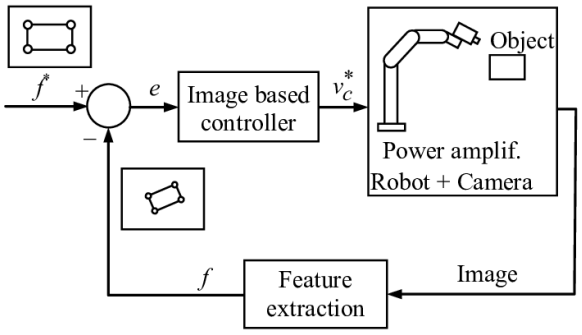


Figura 5: Descripción gráfica del proceso de control por visión [12].

6.1.1. Software utilizado para visión por computadora

El campo de visión por computadora, al ser bastante amplio, cuenta con una gran cantidad de programas y librerías que se pueden utilizar para trabajar. Entre estos, uno que es bastante común es OpenCV (*Open Source Computer Vision Library*) que es una librería que se utiliza para realizar visión por computadora en tiempo real [13]. Esta librería se pueden manejar dentro de diferentes plataformas como Python y Matlab. Además, funciona en sistemas operativos de Windows, macOS y Linux. Cuenta con funciones para realizar una gran cantidad algoritmos de visión por computadora. Varios módulos de cámara se pueden conectar a través de OpenCV. Programas como Matlab cuentan también con una gran habilidad de procesamiento de imágenes, ya que tienen funciones especializadas para la visión.

6.1.2. Detección de marcadores ArUco

Los marcadores ArUco están compuestos por un cuadrado con borde negro y una matriz interna de color blanco que corresponde a un identificador [14], como se puede ver en la Figura 6 . La librería de OpenCV ya cuenta con un diccionario donde estan guardados estas matrices y sus respectivos identificadores. Lo primero que se hace en el algoritmo

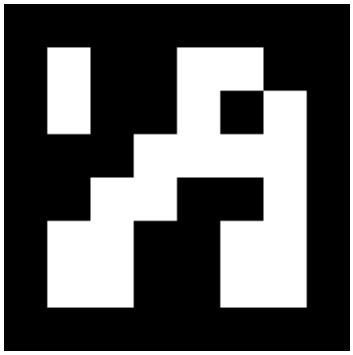


Figura 6: Ejemplo de marcador ArUco con identificación de 23 [14]

de detección es un proceso de thresholding:, que es un proceso de visión por computadora en donde se convierte en binaria una imagen y se pasa a escala de blanco y negro. Con esta imagen binaria, se identifica la forma cuadrada del marcador. Es importante para este módulo que la superficie donde se encuentre el marcador sea plana para poder identificar la figura cuadrada. Es necesario después eliminar las partes fuera del cuadrado, para que solo quede la imagen del marcador.

Esta imagen se divide en células según el tamaño del marcador y se compara la información de los colores de cada célula con el diccionario de marcadores que se tiene para verificar si es un marcador y cuál es su identificador. Por lo mismo, es posible realizar un marcador diferente a los predeterminados si se le agrega la referencia a este diccionario donde realiza la búsqueda. Esta información se obtiene y se puede sobre escribir en la imagen o guardar en otro lado. La Figura 7 a continuación demuestra los marcadores ya identificados.

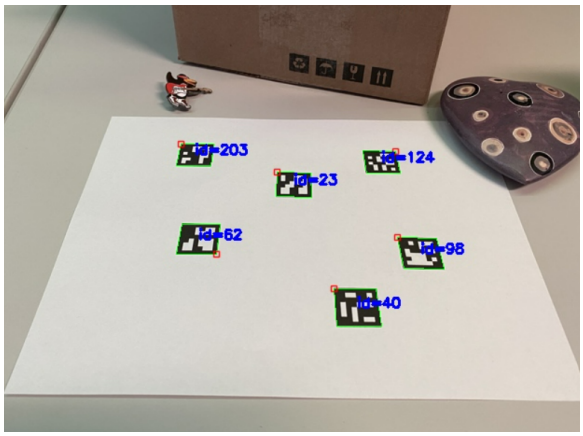


Figura 7: Imagen con marcadores identificados, [14]

6.2. Módulos de cámaras

6.2.1. JeVois Smart Machine Vision Camera

El módulo JeVois es un módulo de cámara inteligente, ya que cuenta con un CPU de cuatro núcleos capaz de correr una versión ligera de Linux, el sensor de video, puertos USB y puertos seriales [15]. . Es gracias a esto que el procesamiento de diferentes algoritmos de visión por computadora se realizan dentro de la misma cámara. Es compatible con proyectos de PC, Arduino o Raspberry Pi. La base de sus diferentes algoritmos es la librería de OpenCV [15]. Entre sus considerables ventajas están su tamaño compacto, su versatilidad de implementación y las diferentes aplicaciones que se le han dado para realizar visión por computadora en otros proyectos.

Figura 8: Cámara JeVois, [15]

Entre los módulos de visión por computadora que se pueden realizar con esta imagen se encuentran: detección de códigos ArUco, detección y reconocimiento de objetos, detección de ejes, *thresholding*, etc.

Para el usuario común, la cámara cuenta con una interfaz gráfica conocida como JeVois Inventor, que permite el visualizar la imagen de la cámara, ver y modificar el algoritmo

de visión por computadora que está seleccionado y leer los mensajes por puerto serial que emite la cámara. El sistema cuenta también con diferentes librerías para programadores que permiten modificar estos módulos de visión preestablecidos. Sin embargo, ambos recursos únicamente son compatibles con procesadores de arquitectura AMD:, lo que significa que no se pueden modificar los módulos ya establecidos desde sistemas de arquitectura ARM:, como la Raspberry Pi. Sin embargo, todavía es posible utilizar los módulos de visión por computadora con los que cuenta la cámara si se realiza la configuración correcta por medio del puerto serial.

6.2.2. Raspberry Pi Camera

La empresa de Raspberry Pi tiene en el mercado un módulo de cámara, la Raspberry Pi Camera, o RaspiCam, como se le conoce [16]. Este es un módulo compatible con diferentes modelos de las Raspberry Pi y con librerías para su uso fácil en sistemas de Raspbian. Cuenta con la habilidad de tomar fotos y videos en definición alta. Es un módulo pequeño, pero cuenta con un sensor de visión, un cable FPC y diferentes montaduras.

Figura 9: Cámara RaspiCam, [16]

Una de las ventajas de esta cámara es que es compatible con OpenCV. Esto significa que se pueden programar módulos de visión por computadora con facilidad desde Python o utilizando el lenguaje C. Aunque las librerías nativas de la cámara solo son compatibles con el sistema operativo de Raspbian, en Linux es posible llamar el módulo con OpenCV ya que se cuentan con los *drivers* necesarios para este sistema operativo. De igual manera, su versatilidad le da la facilidad de integración con todo tipo de proyectos y para realizar diversas operaciones de visión.

6.2.3. Microsoft Kinect

El módulo de visión Kinect es un módulo de detección de movimiento de la compañía Microsoft que se utilizaba originalmente para la consola Xbox [17]. Sin embargo, después de su lanzamiento, se empezó a convertir en una opción de bajo costo para realizar visión por

computadora gracias a su procesamiento de profundidad y su sensor de visión RGB [1]. Por lo mismo, Microsoft desarrollo un módulo de Kinect para PC.

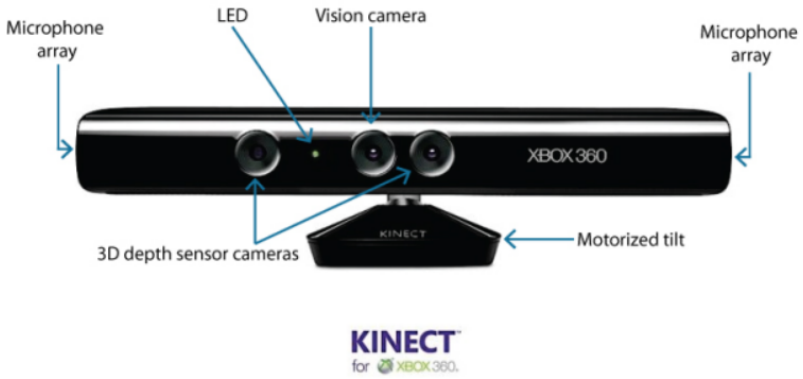


Figura 10: Configuración de módulos dentro del Kinect [1].

Sin embargo, desde el lanzamiento de Windows 8, Microsoft dejó de proveer soporte oficial para los *drivers*.

Desde ese momento, empezó a surgir un grupo de programadores colaborando para realizar una serie de librerías de *open source* llamado Open Kinect. Estas librerías buscan la integración del módulo a diferentes sistemas operativos como Linux [18]. Con este software es posible utilizar la cámara para diferentes proyectos.

En los últimos años, se han realizado estudios donde se utiliza el Kinect para aplicaciones de seguimiento de objetos, reconocimiento de posición y gestos, mapeo de áreas, entre otros. La estructura del módulo se muestra en la Figura 10.

6.3. ROS

ROS, como se conoce por sus siglas en inglés (*Robot Operating System*) es un *software* de *open source* que cuenta con una gran cantidad de librerías y herramientas que buscan integrar diferentes componentes para el funcionamiento de un robot [19]. El programa funciona al conectar diferentes nodos que representan diferentes actuadores o sensores del robot. ROS permite juntar estos nodos y realizar controladores para el robot de manera más sencilla. Los nodos de ROS pueden ser escritos en lenguaje de Python o C y la unión de esta puede ser entre diferentes lenguajes.

ROS maneja nodos por medio de temas, (*topics* en inglés) o servicios. Con los temas, se crea un nodo que este constantemente publicando información y otro que este suscrito al tema y esté escuchando cada vez que el nodo publique algo. Por otro lado, con los servicios se realiza un sistema similar a un servidor y cliente. Esto significa que se crea un servidor que le pida información al cliente antes de recibirla. Se mandan mensajes de confirmación siempre que se realiza una transacción de información.

Se puede conectar a través del programa todo tipo de sensores que se manejen con software. Existen librerías que integran módulos de ROS con software para realizar visión

por computadoras, tal como OpenCV.

La versión de ROS que se está utilizando para el proyecto es la versión de ROS 2 Foxy-FitzRoy que es más compatible con el sistema operativo de Ubuntu 20.04.

CAPÍTULO 7

Preparación de la computadora Raspberry Pi

Se definió que la computadora que se estaría utilizando sobre la plataforma robótica sería una Raspberry Pi 4. Sobre esto estaría corriendo ROS para la lectura de los sensores, el programa con el control del robot y se estarían enviando instrucciones a los motores. Para esto se necesitaba tener un sistema operativo compatible con ROS y con programas como Python y OpenCV.

Para la implementación de las cámaras para la visión por computadora del robot, fue necesario que estas estuvieran corriendo en una Raspberry Pi. Este módulo de visión, uno de ubicación y la unión en ROS de todos los demás módulos del robot estarían corriendo en Raspberry. Sin embargo, la versión de ROS2 que se utilizó para el proyecto, ROS FoxyFitzroy, corre idealmente en sistemas operativos de Ubuntu Linux 20.0, macOS y Windows. Por lo mismo, no era posible la instalación en el sistema Raspbian con el que se trabajan los módulos de Raspberry Pi 3 en la universidad.

Aunque en el proyecto se tenía planeado el uso del modelo Raspberry Pi 4 para la unión de los módulos, no se contaba con estas cuando se inició, por lo que se empezó a trabajar en una Raspberry Pi modelo 3B.

Se decidió entonces crear una imagen que se podía instalar en la Raspberry Pi que contara con Ubuntu 20.04, Ros2 Foxy, al igual que los programas básicos que fueran necesarios para estar trabajando en la programación de los diferentes módulos. La intención de la imagen fue de poder crear y probar los nodos de ROS sobre las Raspberry Pi 3 y que esta quedara disponible dentro del departamento para uso futuro.

7.1. Ubuntu Mate

Por el tamaño de la memoria RAM en las Raspberry de modelo 3, solo se permite descargarle el sistema operativo de Ubuntu 20.04 en versión Server, es decir, sin interfaz gráfica. La página oficial de Ubuntu recomienda para las Raspberry Pi 3 esta versión de 64 bits ARM [20]. Para el modelo 4, que cuenta con más memoria RAM, sí es posible desde el principio instalar Ubuntu Desktop, que ya es la versión completa del sistema operativo y cuenta ya con interfaz gráfica.

Lo primero que se realizó fue descargar la imagen de Ubuntu Server de 64 bits, ARM, a la Raspberry Pi. Se realizó la conexión a internet por WiFi y se empezaron a realizar las descargas de algunos programas y funciones necesarias.

Aunque se puede trabajar desde este sistema, no es tan amigable con el usuario y para alguien sin mucho conocimiento de Linux puede significar una extensión de tiempo al trabajar bastante significativo. Por esto, se decidió instalarle una interfaz gráfica simple al sistema operativo.

Después de algunos intentos de instalar otras opciones, la interfaz que sí funcionó y demostró mejor rendimiento fue Ubuntu Mate, una opción ligera y compatible con el sistema operativo de Linux. Además, se realizó la instalación de programas como Python, Kwrite, Thonny, etc., que son necesarios para la programación en Python o C y que son útiles para los diferentes usos que pueda tener el usuario. La vista del escritorio de la interfaz se puede ver en la Figura 11.

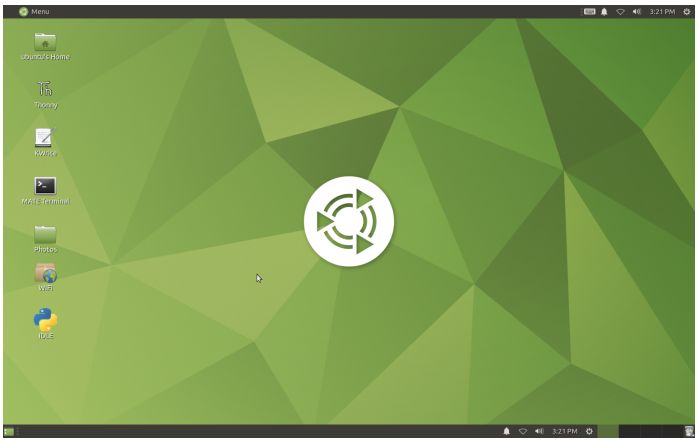


Figura 11: Vista del escritorio de la imagen creada para Raspberry Pi con Ubuntu Mate.

Luego se instaló la versión de ROS exitosamente y se realizó el proceso para poder llamarlo desde la terminal, como se puede ver en la Figura 12. Se realizaron pruebas simples tanto en ROS como en algunos otros programas y se comprobó su funcionamiento. Una prueba con una simulación de ejemplo se puede ver funcionando en la Figura 13. Esta imagen con el sistema, la interfaz gráfica, ROS y los otros programas, se creó utilizando el programa WinServer 32. Luego, se instaló este sistema creado en otra Raspberry Pi 3 exitosamente.

Adicionalmente, se realizó una guía de uso rápido del sistema, que incluye información

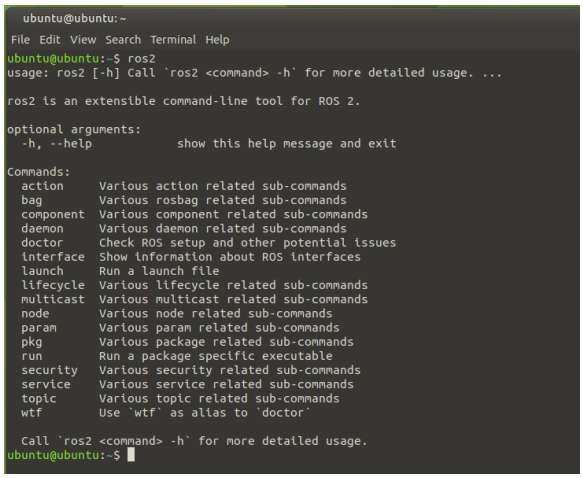


Figura 12: Imagen creada corriendo ROS desde la terminal.

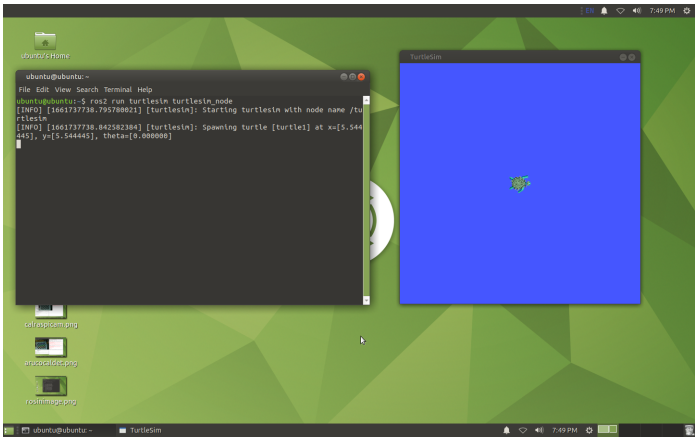


Figura 13: Imagen creada con ROS funcionando.

de cómo conectarse al internet y el uso básico de la misma, al igual que una guía sobre la creación de la imagen desde cero, instalando la versión Server y desde ahí realizando la instalación del Desktop y los demás programas. Esta instalación que se creó es compatible para los modelos Raspberry Pi 3 y 4, aunque hay una diferencia considerable en cuanto al rendimiento ya que es más rápida sobre la versión 4 de la computadora.

7.2. Ubuntu Desktop

Una vez se contó con el modelo 4, se creo también una instalación descargando directamente Ubuntu Desktop. A esta instalación de Ubuntu se le descargaron los mismos programas que al otro sistema y se realizó la instalación de ROS. La misma prueba del ejemplo de la simulación se puede visualizar en la Figura 15. Esta interfaz es un poco más completa y familiar para el usuario. La vista del escritorio de la imagen creada se puede ver en la Figura 14.

Esta interfaz se recomienda únicamente para el modelo 4. Al ser una interfaz más com-

pleta, ocupa más memoria de RAM y puede causar que los programas en el modelo 3 sean más lentos o se cierren repentinamente. Otra de las ventajas es que la instalación puede ser más rápida, ya que si se encuentra la versión Desktop del sistema operativo, se puede omitir todo el proceso de la descarga de la interfaz gráfica desde la versión Server.

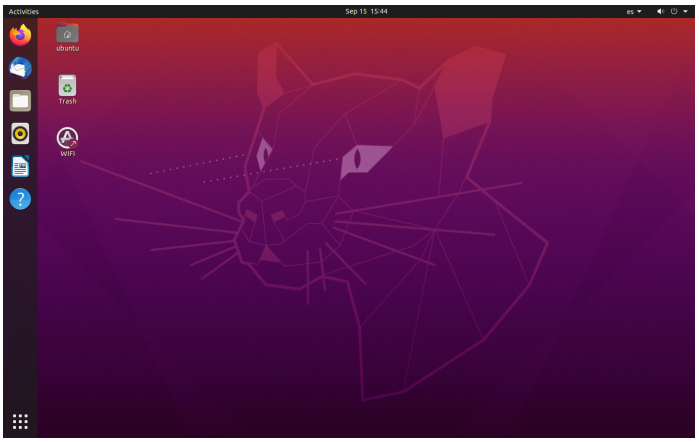


Figura 14: Vista del escritorio de la imagen creada para Raspberry Pi 4 con Ubuntu Desktop.

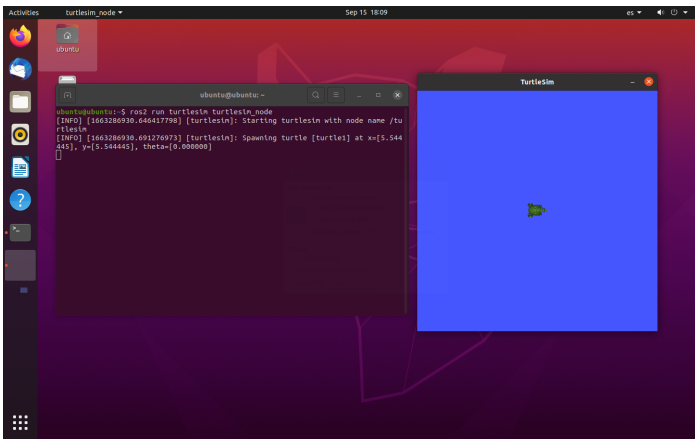


Figura 15: ROS funcionando en imagen con Ubuntu Desktop.

7.3. Conexión a Internet

Los sistemas operativos de Raspbian son nativos a los módulos de Raspberry Pi y por ende, los más recomendados para la computadora. Esto significa que aunque se pueden utilizar otros sistemas operativos de Linux, ciertas cosas tienen que adaptarse para funcionar dentro de la computadora. Por ejemplo, en Raspbian, cualquier configuración del *hardware* como habilitar la cámara o cambiar el teclado, al igual que algunas configuraciones del sistema se realizan en la aplicación de Raspi-config.

Esta librería no es compatible con sistemas diferentes a Raspbian. Los cambios necesarios se pueden hacer de manera manual, cambiando algunos archivos específicos, pero la aplicación permite realizarlo de manera más intuitiva. Tener una interfaz gráfica permite

realizar algunos de estos cambios fácilmente, como el cambio de teclado y hora del sistema. Otros, como habilitar la cámara requirieron un poco más de investigación.

Algo que se observó en las instalaciones de los sistemas operativos realizadas fue de que la interfaz gráfica muestra el ícono para realizar la conexión al Internet, pero no es posible realizar la conexión por este medio. La Raspberry Pi estaba conectada a la red, ya que se modificó un archivo en la instalación inicial para poder realizar los demás cambios e instalaciones necesarias. Sin embargo, no era posible visualizar redes disponibles o realizar el cambio a otra red sin ir a modificar el archivo. Este requiere ingresar las redes de una manera muy específica según el tipo de red qué es y es sensible a las indentaciones y tipos de espacios que ingresa el usuario. Luego de realizar el cambio, es necesario aplicar los cambios por medio de otra instrucción. Para que se realize el cambio es necesario hacer todo esto con permisos de administrador, es decir permiso *sudo*.

Por ende, se decidió hacer el proceso de conexión más fácil para el usuario. Se creó un *bash script* que por medio de una terminal le pide la información de la red al usuario, modifica el archivo con el formato necesario y aplica los cambios para que ya esté conectada la Raspberry Pi a la red deseada. Se creó además, un icono en la pantalla principal que llama al programa con los permisos necesarios para realizar el cambio y aplica el cambio de una vez. En el programa se le da la opción al usuario de escoger una de las redes predeterminadas o ingresar los datos de una nueva. Si es una de las que ya está predeterminadas, modifica de una vez el archivo. De lo contrario, le pide al usuario el nombre y la contraseña de la red a la que se quiere conectar. En la Figura 16, se puede ver el programa creado corriendo en la terminal. Los cambios en el archivo por medio de la aplicación se muestran en la Figura 17.

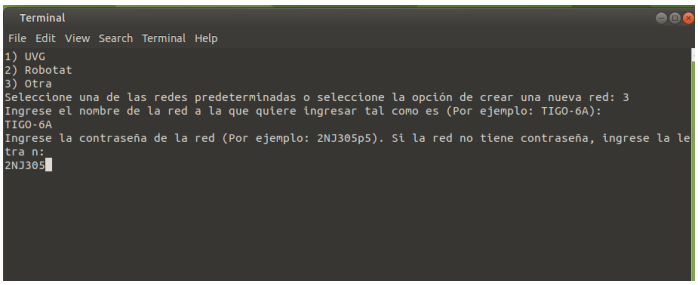


Figura 16: Resultado de correr el icono pidiendo el input del usuario para conexión a una red nueva.

Ya que ambas imágenes presentaban el mismo problema a pesar de tener diferentes interfaces gráficas, se les agregó esto a ambas. Funciona en ambos modelos de Raspberry Pi.

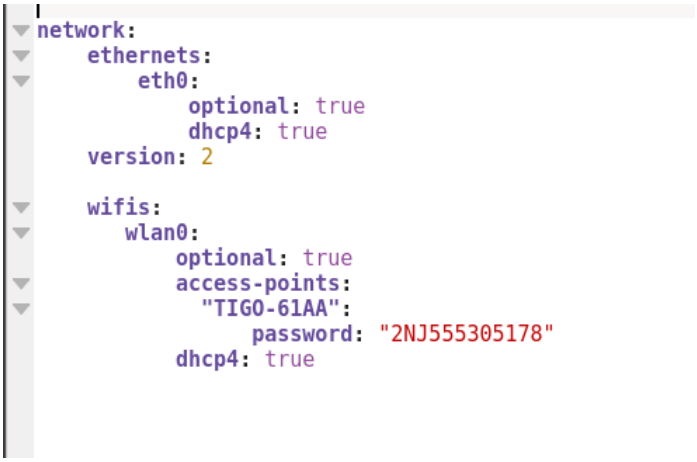


Figura 17: Archivo que modifica el programa con el formato correcto.

CAPÍTULO 8

Módulos de visión por computadora

Al inicio del proyecto, fue necesario acordar entre los otros compañeros trabajando sobre la plataforma, al igual que con los catedráticos, cuál sería el alcance de esta primera iteración del rover. Se definieron unos cuantos experimentos que comprobarían la funcionalidad de los nodos al igual que de la unión de estos. Para la visión de computadora, se definió un experimento que consistería en la identificación de "estaciones" por medio de reconocimiento de códigos ArUco. Para ello, primero fue necesario realizar al adaptación a las cámaras al sistema operativo que se definió y luego realizar las pruebas de diferentes algoritmos de visión.

8.1. Adaptación de las cámaras al sistema operativo

Las primeras pruebas con las cámaras se realizaron en una Raspberry Pi que contaba con el sistema operativo de Raspbian. Luego, después de la instalación de Ubuntu sobre la computadora, fue necesario adaptar las cámaras a este nuevo sistema operativo.

8.1.1. Raspberry Cam

No se encontraron mayores problemas con la RaspiCam. Dentro de Raspbian se puede descargar una librería de Python especializada para el uso de esta cámara llamada Picamera. Con esta librería y la de OpenCV, se lograron realizar algunas pruebas de algoritmos de visión por computadora. Sin embargo, esta librería no es compatible con el sistema operativo de Ubuntu, por lo que lo primero que se tuvo que hacer fue investigar un poco sobre cómo utilizar únicamente OpenCV para llamar a la cámara. Además fue necesario habilitar la cámara agregando una línea al final de un archivo en el folder de *firmware*.

8.1.2. JeVois

La cámara JeVois fue un poco más complicada. Esta cámara ya cuenta con una instalación creada por la compañía que corre una versión de Linux más simple y contiene ya varios módulos de visión por computadora. Esto da una gran ventaja para el usuario promedio, ya que, sin la necesidad de instalar *drivers* o software (en sistemas compatibles), se puede conectar la cámara y empezar a realizar diferentes módulos de visión por computadora al llamar diferentes configuraciones por medio de aplicaciones como Guvcview o algún programa serial como Screen. Por ejemplo, la configuración de YUYV 320 240 50.0 JeVois DemoArUco llama ya un módulo de detección de ArUco establecido en la imagen. Para modificar estos módulos como programadores, JeVois ofrece dos alternativas. Hay una interfaz gráfica llamada JeVois Inventor, que permite en la misma aplicación la visualización en tiempo real de la imagen de la cámara, del programa en Python o C del módulo y una pantalla serial donde se imprimen algunos mensajes. Aquí se pueden crear nuevos módulos si se tiene la librería *jevois-sdk-dev*. Esta es la segunda opción que se le ofrece a programadores. Con esta librería se pueden realizar módulos desde otros programas ya que básicamente permite realizar cambios en la instalación con la que cuenta la JeVois mientras se esta programando, guardando esos módulos y permitiendo la creación completa de módulos de visión utilizando librerías como OpenCV.

Es importante mencionar que la aplicación de JeVois inventor dio problemas en la imagen anterior de Raspbian que tenía la Raspberry al principio ya que no se pudo instalar. En una máquina virtual que se instaló en una computadora laptop, el programa fue bastante lento y la imagen se congelaba con frecuencia.

La librería *jevois-sdk-dev* no es compatible con el sistema operativo ARM, solo con los sistemas AMD. Dado que la arquitectura del procesador de la Raspberry Pi es ARM, no es posible esta instalación. Por lo mismo, fue imposible crear módulos de visión que llamaran la cámara y permitieran realizar cambios al sistema ya instalado sin la librería correspondiente. La solución que se encontró fue utilizar estos módulos ya creados de la cámara y llamarlos por medio de un *bash script*, donde se establecen las configuraciones necesarias para que la cámara llame el módulo. Un *bash script* es un documento de texto con una serie de comandos escritos con un lenguaje específico. Al llamarlo dentro de la terminal, se logran realizar todas las instrucciones de un solo.

De esta manera se lograron tener ambas cámaras funcionales en la imagen final de la Raspberry Pi, listas para realizar los módulos de visión por computadora deseados y unir esto como nodo de ROS para que estuvieran enviando constantemente la información requerida.

8.2. Módulo detección marcadores ArUco

8.2.1. Raspberry Cam

Luego de la adaptación de la RaspiCam a la instalación de Ubuntu que se estaría utilizando, se empezaron las pruebas con la cámara. Se empezó con lo más básico: realizar la toma de una sola imagen con la cámara utilizando OpenCV.

Se fue trabajando sobre esto para llegar al módulo de visión por computadora. Se realizó un programa de calibración para la cámara. Este contaba en dos programas separados. El primero realizaba capturas de la cámara cada 30 cuadros, y guardaba estas imágenes como `cal_image_x`, dónde x representaba el número de tomas. Se realizaron tomas de un tablero de ajedrez impreso en una hoja a diferentes distancias de la cámara, como se puede observar en la Figura 18.

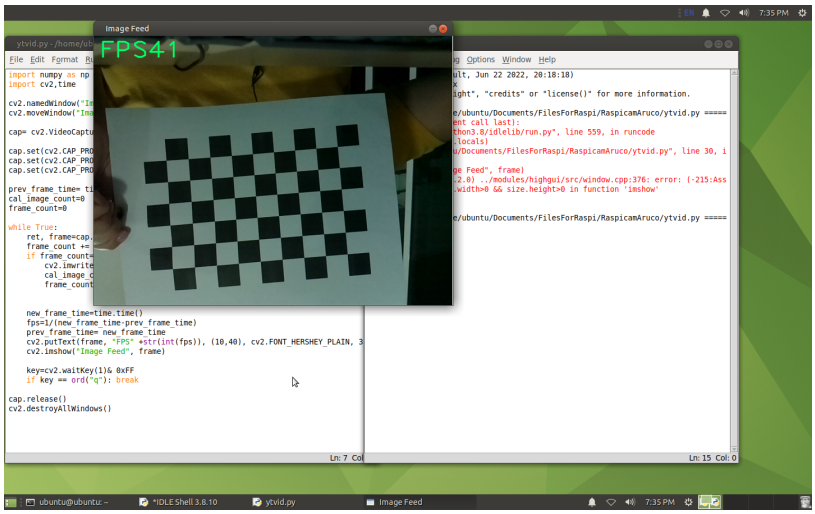


Figura 18: Imagen tomada por el programa de tablero de calibración.

El segundo programa usaba las imágenes del tablero obtenidas con el programa anterior y realizaba un algoritmo de detección de esquinas, como se puede observar en la Figura 19. La ubicación de estas según el programa se comparaba con la medida real de la distancia entre esquinas del tablero. Con esta comparación, se realiza una matriz de calibración que indica la distorsión que puede tener la cámara y se guarda en otro archivo.

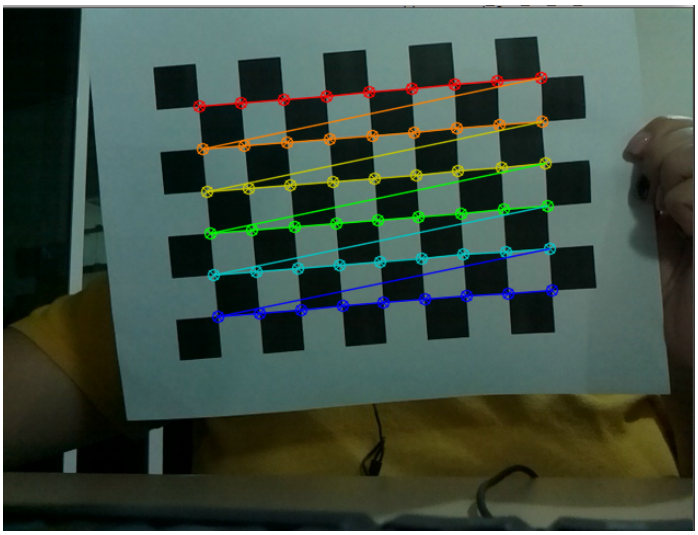


Figura 19: Detección de esquinas funcionando sobre imagen tomada de tablero de calibración.

Luego de tener la esta matriz de distorsión se llama al último programa. Este utiliza la

librería de ArUco de OpenCV que cuenta ya con un diccionario de estos marcadores para su identificación. Se llama al diccionario y se empieza a tomar un vídeo con la cámara. La misma librería de ArUco incluye una función de detección de los marcadores después de aplicar una detección de esquinas y bordes. Luego, simplemente se le indica al programa que imprima la información del marcador: su identificación, las coordenadas en x, y, z de la distancia del marcador respecto a la cámara y el ángulo de este.

Lo primero que se realizó fue una prueba en dónde únicamente se tomaba una imagen e imprimía esta información sobre la misma, como se ve en la Figura 20. Sin embargo, esta información no es relevante para el procesamiento y lo hace un poco más lento. Por lo mismo, se decidió cambiar el programa a que la cámara estuviera tomando video constante y que únicamente se mandara el dato de la identificación, como se puede observar en la Figura 21. El programa corría bien, con bastante rapidez y logró identificar todos los marcadores que se le mostraron.

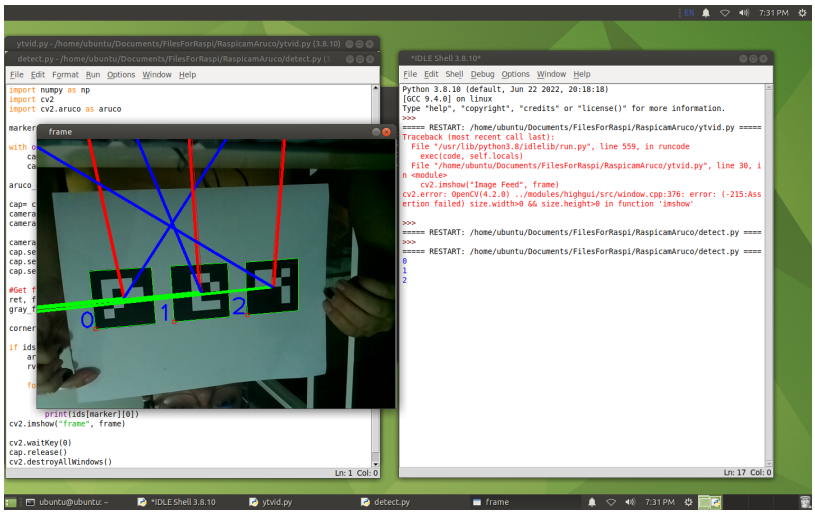


Figura 20: Prueba de información escrita sobre imagen e impresión de identificación en terminal.

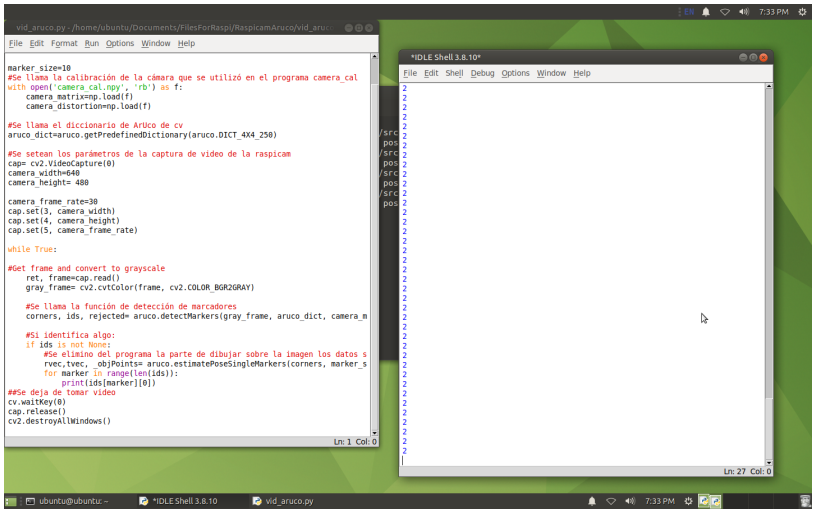


Figura 21: Prueba de identificación impresa en terminal con video.

8.2.2. JeVois

Como se mencionó anteriormente, no fue posible utilizar las librerías o la interfaz para modificar los módulos de visión por computadora que venían dentro de la imagen de esta cámara. Sin embargo, se pudo comunicar con la cámara desde su puerto serial y enviar la configuración correspondiente a el módulo de detección de marcadores que se buscaba.

Al principio, se estaba llamando y probando el funcionamiento del algoritmo de detección adentro de la función de *screen*. Esta es una funcionalidad en Linux que permite crear pantallas en dónde se pueden correr programas dentro de la terminal y ver ahí mismo su resultado. Se creaba una sesión de *screen* con el aparato y por medio de serial se le escribían los parámetros de configuración a la cámara. Con esto se le indica que corra el módulo de detección de ArUco, que despliegue la identificación y la información de ubicación de estos, se escogen otros parámetros internos y que se muestre la información en la misma terminal. El módulo de detección de marcadores de ArUco que ya trae la cámara esta basado en las librerías de ArUco de OpenCV. Por lo mismo, realiza el mismo proceso que se realizó con la otra cámara para la detección de marcadores.

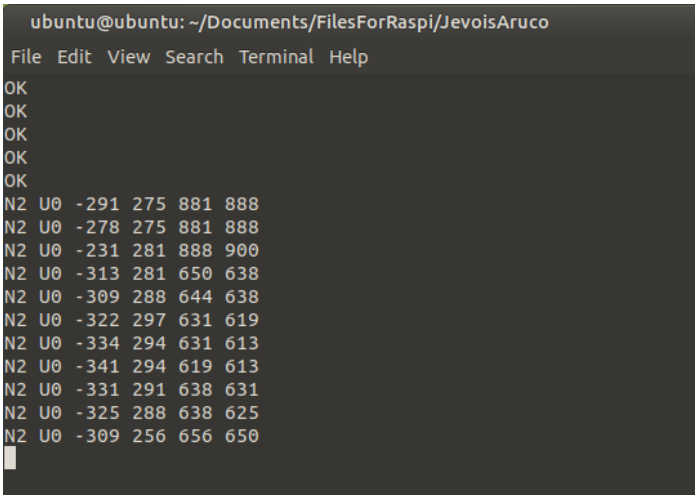


Figura 22: Funcionamiento de sesión de *screen* después de establecer parámetros.

Esto funcionaba sin problemas, pero eran cinco instrucciones diferentes que se debían de ingresar en ese momento sin errores para que lo detectará la cámara y se guardaran los parámetros.

La solución que se le dio fue la creación de un *bash script* que iniciara esta sesión y corriera las instrucciones necesarias. Esto permitía iniciar la sesión de screen y colocar los valores necesarios en los parámetros de la cámara, como se puede ver en la Figura 23. Sin embargo, luego se encontró una forma más simple de mandar la configuración directa al aparato conectado en serial igual por medio de un *bash script* sin tener la visualización en la terminal, que no era necesario para el proyecto.

Con esto se podían observar los datos de la detección de marcadores en la pantalla, pero no eran útiles para ROS. Por lo mismo, se trabajo un programa en Python utilizando la librería de PySerial. Esta librería permite mandar y leer mensajes desde un puerto serial,

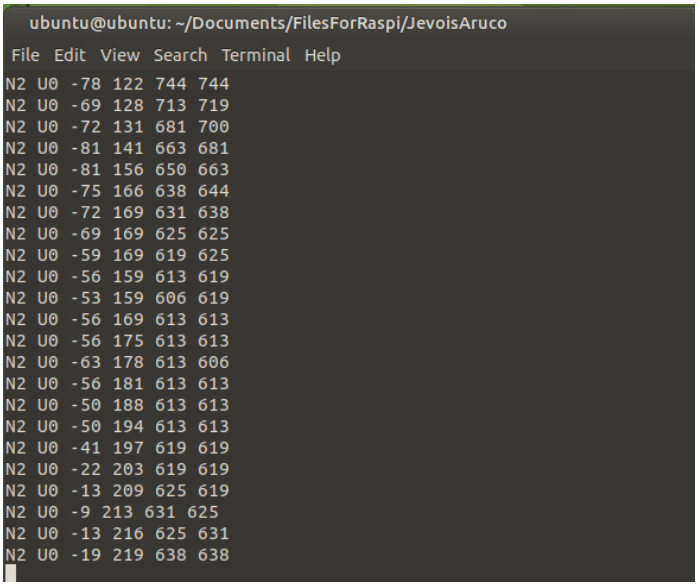


Figura 23: Funcionamiento luego de llamar al *bash script*.

como se ve en la Figura 24. Con esto, se logró crear un programa que llamara el *bash script* para mandar por el puerto las configuraciones necesarias a la cámara y que pudiera leer e imprimir esos mensajes de la información recibida. Este mensaje se separa, para tener por separado la información de la identificación, la ubicación y el tamaño del marcador. La información por este medio sí fue útil para ROS y de igual manera lograba correr rápido y detectar los marcadores indicados.

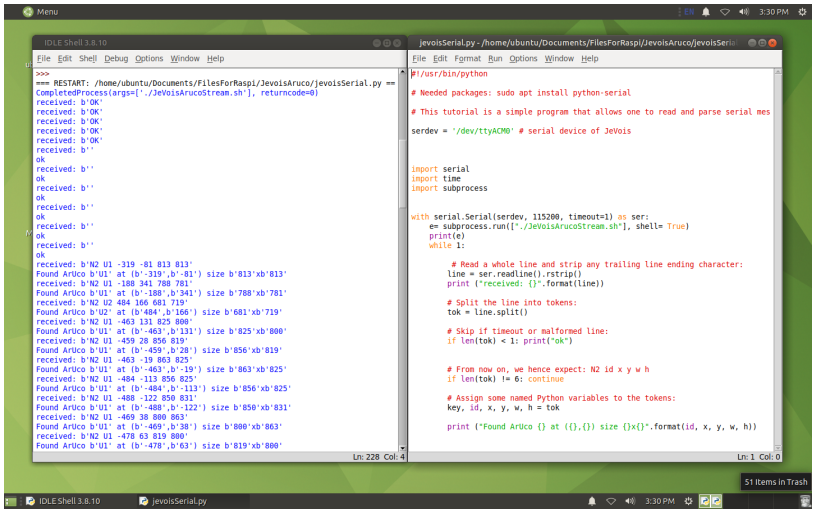


Figura 24: Funcionamiento del programa para extraer información desde el puerto serial.

CAPÍTULO 9

Selección de módulo de cámara

Uno de los objetivos del proyecto fue realizar una comparación entre la Raspberry Pi Cam y la cámara JeVois y seleccionar la más adecuada para utilizar en la plataforma robótica.

Para realizar esto, no se comparó únicamente los resultados de los módulos de visión que se implementaron. También se tomo en cuenta la compatibilidad con ROS, con el sistema operativo instalado en la Raspberry y con el hardware que se estaría utilizando en la plataforma. Además, se tomó en cuenta la aplicabilidad que se le pudiera dar en las siguientes etapas del proyecto.

Como se evidenció en las figuras 21 y 24 ambas cámaras se implementaron de manera exitosa dentro de la instalación de Ubuntu, corriendo programas de detección de marcadores ArUco. Ambas lograron la detección correcta de los marcadores a diferentes distancias y sacaron los datos necesarios de identificación y ubicación.

Para cualquier módulo de visión por computadora que se desee con la Raspberry Pi Cam es necesario crear el algoritmo desde cero. Esto trae ventajas y desventajas. Una de las ventajas es que se pueden modificar los algoritmos a lo que se necesite específicamente para el funcionamiento. También significa que se pueden realizar una gran variedad de módulos de visión con ayuda de la librería de OpenCV. No se puede usar la librería de RaspiCam creada por la compañía de Raspberry Pi, ya que no es compatible con la instalación, pero de todos modos se puede utilizar realizando las adaptaciones mencionadas. La desventaja de la creación de módulos es que el procesamiento de este sucede sobre la Raspberry Pi dónde está la cámara. Tomando en cuenta que el programa de control en ROS estaría llamando varias terminales y realizando varios procesos a la vez, este requiere bastante capacidad computacional por lo que no sería tan recomendable estar llamando el procesamiento en la misma computadora. Sería necesario contar con una segunda Raspberry donde este conectada la cámara para que el funcionamiento sea el óptimo. Se necesita entonces realizar algún tipo de comunicación entre estas para que la información se transmita al sistema de control.

Otra desventaja es que para llamar a otro módulo de visión es necesario realizar el código completo de este y realizar el procesamiento ahí mismo.

Por otro lado, la JeVois trae consigo sus propias ventajas y desventajas. Sí se pudo utilizar dentro del sistema operativo de Ubuntu, pero únicamente llamando a los diferentes módulos disponibles dentro del sistema operativo por medio del puerto serial. Las librerías para modificar estos módulos y crear nuevos desde cero y la interfaz gráfica creada por la compañía no son compatibles con la instalación que se creó. Por lo mismo, aunque los módulos que trae sí están basados en OpenCV, no se puede llamar a la cámara en los módulos creados con esta librería desde Python. Una ventaja es que ya que el funcionamiento de esta se realizó por medio del puerto serial, la información que se buscaba también se transmite por este medio. Por ende, también es posible su implementación como nodo de ROS.

Una de las características más importantes de esta cámara es que el procesamiento de los módulos de visión por computadora se realiza dentro de la cámara y solo se manda el resultado por medio de un puerto serial. Esto no implica mucha carga computacional dentro de la Raspberry Pi a la que esta conectada. Además, es fácil llamar a los otros módulos disponibles en el sistema operativo, ya que solo se necesita cambiar la configuración con la que se llama la cámara a la correspondiente al módulo que se quiere implementar. Es fácil igual cambiar la manera en la que se despliegan los resultados.

A continuación se muestra una tabla resumiendo las características que posee cada cámara.

Para la plataforma se decidió que se tendría una Raspberry Pi corriendo ROS con todos los nodos y controladores necesarios. Para no demandar mucho más de esta y que funcione más lento, no se puede conectar algo más que este realizando procesamiento. El uso de otra Raspberry Pi no solo implica tener un módulo de comunicación que podría llegar a causar problemas en algún momento, sino que también significa un costo mayor para la implementación del proyecto. Los otros módulos de la plataforma como el de ubicación con el sistema de captura de movimiento Robotat, el *LiDAR*, los motores y los demás, tenían que estar mandando datos e información a la Raspberry Pi también.

Por lo tanto, se decidió implementar el módulo de cámara JeVois en la plataforma. Este, como realiza el procesamiento del módulo en el interior de la cámara, no implica mucha carga computacional. Además, ya se esta realizando la comunicación serial al tenerla conectado a la Raspberry Pi. Otra ventaja de la JeVois sobre la Raspberry Cam es que también se pueden implementar otros módulos de visión por computadora fácilmente. Por eso, en alguna otra fase del robot o para otro experimento que se decida hacer, es más fácil únicamente cambiar los parámetros que se colocan al llamar la cámara y escoger el formato de la información que se manda por serial.

Sin embargo, se decidió crear una funcionalidad adicional al Rover utilizando la Raspi Cam. Sobre el sistema operativo de Raspbian y con un programa de Python simple, se creó un módulo de transmisión en directo del vídeo que toma la cámara. Esto se pudiera implementar sobre una Raspberry Pi mucho más simple sin nada más y permitiría tener confirmación visual del recorrido del robot. El programa crea un servidor en una página de internet simple sobre un puerto de la dirección IP de la Raspberry Pi y cualquier usuario conectado a la misma red puede ingresar a la transmisión por medio del servidor, Figura 25.

Comparación de cámaras		
Cámara	JeVois Smart Vision Camera	Raspberry Pi Camera
Ventajas	▪ Procesamiento en la cámara. ▪ Comunicación por puerto serial. ▪ Facilidad de alterar cómo se recibe la información. ▪ Facilidad de implementar otros algoritmos de visión por computadora. ▪ Acceso a información detallada, incluyendo la rotación del marcador.	▪ Se pueden crear otros algoritmos de visión por computadora. ▪ Compatibilidad con OpenCV.
Desventajas	▪ No hay acceso a imagen desde Python. ▪ Alimentado por USB. ▪ No es fácil crear algoritmos de visión por computadora.	▪ Procesamiento sobre computadora. ▪ No es compatible con otras computadoras ▪ Para obtener información detallada de la ubicación y orientación del marcador es necesario hacer cambios a la programación.

Cuadro 1: Resumen de características de ambas cámaras.

Figura 25: Transmisión en vivo por medio de servidor Web utilizando la Raspberry Cam.

CAPÍTULO 10

Microsoft Kinect

Parte del proyecto consistió en realizar pruebas para ver si se podría utilizar la cámara Microsoft Kinect para Windows en la plataforma robótica. Esta debería de ser compatible con la Raspberry Pi 4, el sistema operativo Linux y ROS 2 FoxyFitzRoy. Por lo tanto, se tenía que evaluar si esta implementación sería posible ya que el módulo como tal esta diseñado para funcionar sobre sistemas operativos de Windows y tiene una serie de requisitos que hay que cumplir para extraer información.

El Microsoft Kinect para PC perdió soporte oficial de Windows a partir de Windows 8.1. Por ende, para su uso en otros sistemas operativos, se decidió utilizar la librería *open source* de Open Kinect que es compatible con los sistemas operativos de Linux. La librería cuenta con todos los recursos necesarios para la implementación de la cámara.

Sin embargo, el Kinect como tal pide ciertas cosas. Según MathWorks, que cuenta con una *toolbox* que permite la conexión del módulo [21], los requisitos a nivel de *hardware* para correr la cámara son los siguientes:

- Procesador de doble núcleo físico o un procesador bastante rápido
- Puerto de USB 3.0
- 4 GB de RAM
- Tarjeta gráfica con soporte para *DirectX 11*

Los modelos de Raspberry Pi anteriores al modelo 4 no cuentan con puertos USB 3.0 y ningún modelo cuenta con soporte para *DirectX*. Las librerías de Open Kinect ayudan a adaptarse a sistemas que no cuenten con el soporte para esa tarjeta gráfica. Sin embargo, no es posible utilizar el Kinect en una Raspberry Pi 3 sin un adaptador adecuado de USB 2.0 a 3.0.

El Kinect cuenta con un adapatador de USB a corriente alterna, pero este únicamente sirve para darle potencia al módulo de motores. Es necesario el puerto USB 3.0 para darle potencia a las cámaras y a la LED con la que cuenta. El bus de la USB tiene que ser capaz de procesar una gran cantidad de información también para obtener acceso al video.

Se realizaron pruebas con la cámara sobre diferentes sistemas operativos para ver en cuál se podría tener acceso a la información. Se probó sobre el sistema operativo de Ubuntu Mate y Ubuntu Desktop en la Raspberry Pi 4 además de pruebas en Windows 11 y Ubuntu Desktop en una computadora normal. Los pasos realizados en las instalaciones exitosas se encuentran en la guía de instalación para Kinect adjunta en el anexo.

10.1. Pruebas fallidas

Sobre la instalación del sistema operativo de Ubuntu Mate que se realizó para el uso en la Raspberry Pi 4, se instaló la librería de Open Kinect, conocida como freenect. Antes de realizar pruebas, fue necesario agregar una serie de permisos que se tienen que modificar para que la cámara sea compatible con Linux. Ya con los permisos agregados, se trató de correr algunos módulos de ejemplos con los que cuenta la librería. Sin embargo, los ejemplos para obtener imágenes de la cámara fallaban sin sacar la imagen como tal. La cámara sí estaba siendo detectada por el sistema, pero no se pudo tener acceso a la información de sus sensores. Uno de los ejemplos que incluye la librería consiste en únicamente modificar el movimiento de los motores, por lo que debería de ser capaz de realizarse a pesar de las limitaciones de la Raspberry Pi. Sin embargo, al probarlo no se vio ningún resultado. Ya que Mate es una interfaz gráfica más ligera, puede ser que no tenga el soporte necesario para mostrar la imágen obtenida de la cámara.

Para comprobar que no fuera problema de la computadora Raspberry Pi, se trató de realizar la instalación sobre una máquina virtual en una computadora normal. La máquina virtual contaba con un sistema operativo de Ubuntu Desktop 20.04 instalado. Sobre esto, se realizó la descarga de las librerías de Open Kinect. Aunque estas se lograron instalar, no se logró obtener información de la cámara. De igual manera, el ejemplo corría y fallaba sin sacar ninguna imágen. Igual estaba siendo detectado el módulo por la computadora, pero no se logró tener comunicación por este medio. Ya que se estaba trabajando sobre una máquina virtual con Linux, puede ser que los mismos permisos de conexiones físicas con la computadora hayan causado algún tipo de interferencia que no permita tener acceso completo a la cámara.

10.2. Pruebas exitosas

10.2.1. Sobre Windows 11 en computadora

Se quiso después descartar problemas con la cámara, por lo que se probó buscar el sistema de drivers correspondiente a Windows 11 sobre una computadora personal. Para que se pueda obtener información de la cámara es necesario que las librerías de Microsoft sean compatibles con el modelo del Kinect y con el sistema operativo de Windows 11.

La versión original de Kinect for Windows SDK 1.0 no es compatible con sistemas operativos de Windows a partir de Windows 8. Además, a partir de versiones de las librerías 2.0, el aparato no es compatible, ya que estas fueron diseñadas para el modelo 2 de la cámara.

Después de varias instalaciones fallidas, con las que el sistema detectaba la cámara, pero sacaba un mensaje de error en los *drivers* al mostrar el aparato en el Administrador de Dispositivos, se encontró que el sistema compatible con Windows 11 es la versión de Kinect for Windows SDK 1.8. Esta versión puede encontrarse en el centro de descargas de Microsoft, específicamente en el siguiente enlace: [22].

Para su uso se recomienda también instalar la interfaz gráfica de Kinect for Windows Developer Toolkit v1.8, ya que cuenta con varios módulos instalados y permite una visualización rápida de ellos utilizando la cámara. A continuación, en la Figura 26, se ve una de las pantallas de la aplicación con opciones de ejemplos en C#. Este también se puede encontrar en el centro de descargas de Microsoft, específicamente en [23]. La combinación de ambos permite realizar pruebas de una gran cantidad de módulos en C#, C++ o Visual Basic, al igual de tener un acceso fácil a los parámetros de la cámara.

Figura 26: Kinect for Windows Desktop Toolkit v1.8 con ejemplos en C#.

Esta aplicación es bastante útil, ya que sí permite tener acceso a la información de la cámara y probar diferentes módulos. La aplicación tiene un botón para leer la documentación de cada uno de los módulos, pero este no lleva a la página específica. La información de esta versión del Toolkit, desde *troubleshooting* hasta la documentación de los módulos, se puede encontrar en la siguiente referencia: [24]. Esta información es bastante general, y no incluye detalles de la programación de los módulos, pero puede ser de bastante ayuda como guía.

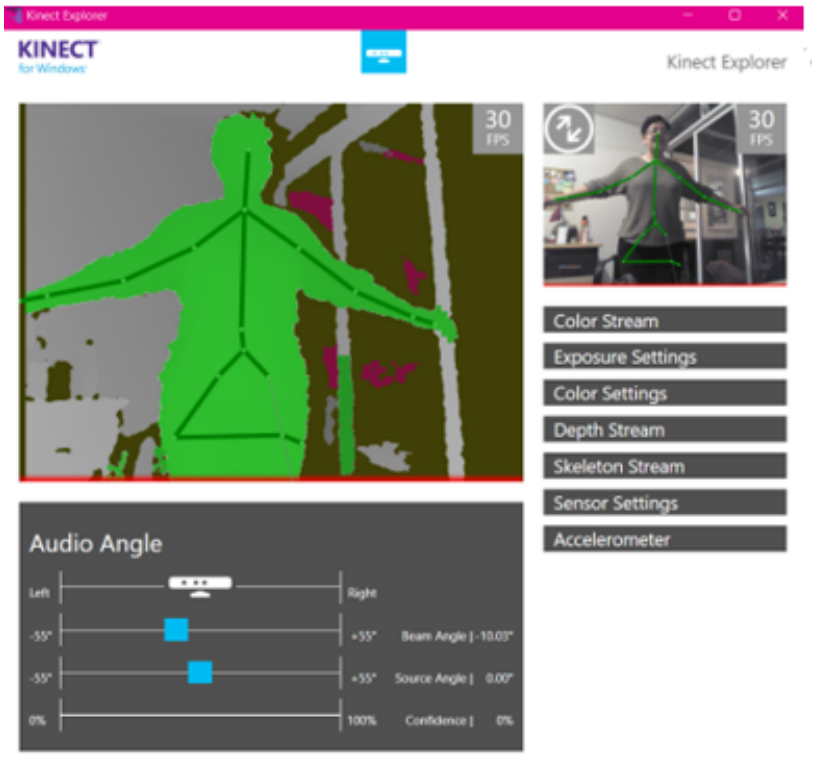


Figura 27: Módulo de Kinect Explorer en la aplicación de Toolkit.

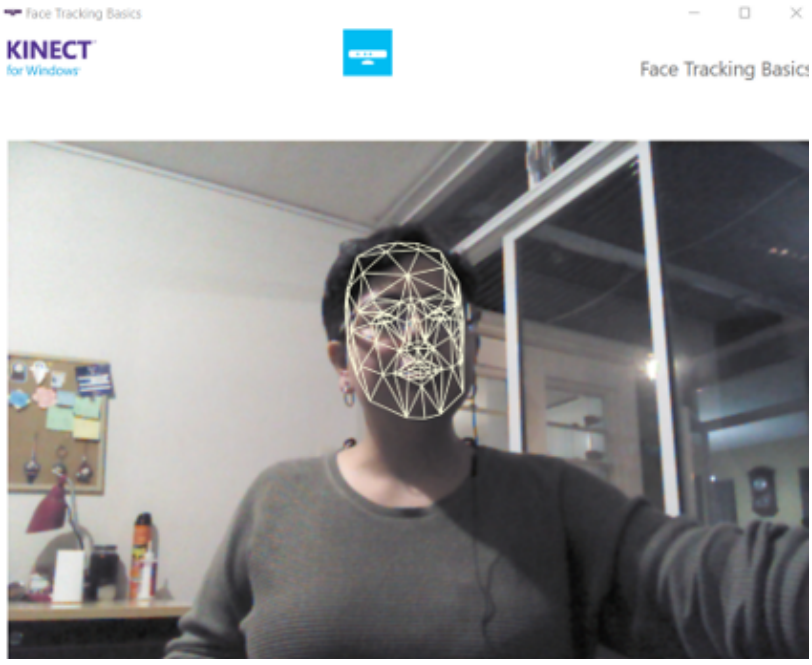


Figura 28: Módulo de Face Tracking Basics en la aplicación de Toolkit.

Se muestran también algunos ejemplos en la aplicación de Toolkit, como el Kinect Explorer que permite modificar ciertos parámetros de la cámara (Figura 27) y el Face Tracking Basic que permite rastrear la figura del rostro sobre la imagen de la cámara (Figura 27) .

10.2.2. Sobre Ubuntu Desktop en Raspberry Pi 4

En la instalación del sistema operativo de Ubuntu Desktop que se realizó para la Raspberry Pi 4 se realizó la instalación de los *drivers* de la librería de Open Kinect y se pudo obtener información de la cámara exitosamente. Sin embargo, fue necesario utilizar la versión más reciente de los repositorios en GitHub de las librerías y descargar esos archivos en lugar de realizar la instalación directa, como indica la página principal de la documentación oficial de Open Kinect. Esto se debe a que la instalación directa de la librería descarga la versión de esta compatible con versiones anteriores de Ubuntu 12.

Además de realizar la instalación de esta manera, es necesario modificar un archivo específico y agregar ahí una serie de "reglas" que modifican los permisos de la cámara. Además, es necesario hacer el cambio a un parámetro en uno de los archivos de la instalación antes de llamar directamente la instalación de los *drivers* de audio. Ya con esta modificación la computadora lista tres entradas al conectar el Kinect: la parte de la cámara, la del audio y la de los motores. Los detalles del método de instalación se encuentran en la guía de instalación para Microsoft Kinect adjunta en el anexo.

Una vez se logró la instalación exitosa, se empezaron a probar los ejemplos que traen ya la librería. Se logró obtener del Kinect la imagen de la cámara normal al igual que de la cámara de profundidad. A continuación se muestran los ejemplos de *glview* (Figura 29), *chunkview* (Figura 30), *glpclview* (Figura 31) y *regview* (Figura 32). Los ejemplos de *glview* y *chunkview* utilizan la cámara de profundidad y colocan colores sobre la imagen para mostrar los diferentes niveles que se muestran en la toma. El ejemplo de *glpclview* muestra el módulo de la cámara de eliminación de fondo, identificando a una persona y borrando el fondo detrás. Por otro lado, el ejemplo de *regview* es un filtro de colores de la cámara normal, causando un efecto de distorsión sobre la imagen obtenida. En teoría, después de la instalación, ya se debería de contar con los *drivers* necesarios para el manejo del audio y de los motores, sin embargo, los ejemplos de las librerías para modificar estos parámetros no funcionaron.

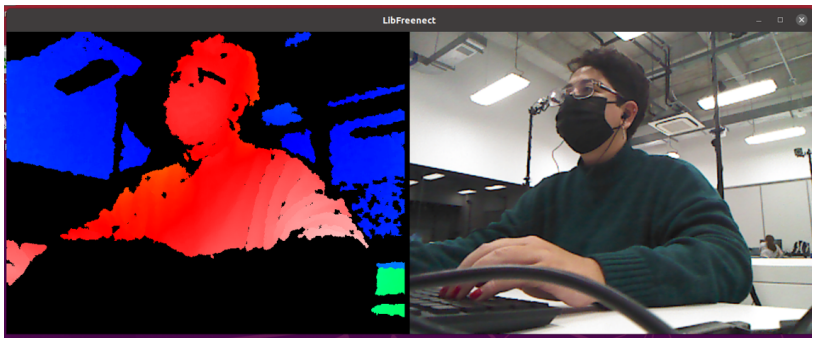


Figura 29: Resultado de correr el ejemplo *glview*.

Además, al modificar cierto permiso con la instrucción *modprobe* se puede utilizar la cámara como una cámara normal conectada a la computadora. Esto significa que se puede utilizar un programa de visualización de vídeo, como VLC y tener acceso a esta. En la Figura 33 se muestra la captura de vídeo tomada con el Kinect seleccionado como cámara. Al ser reconocido como aparato de cámara, puede ser posible llamar al módulo

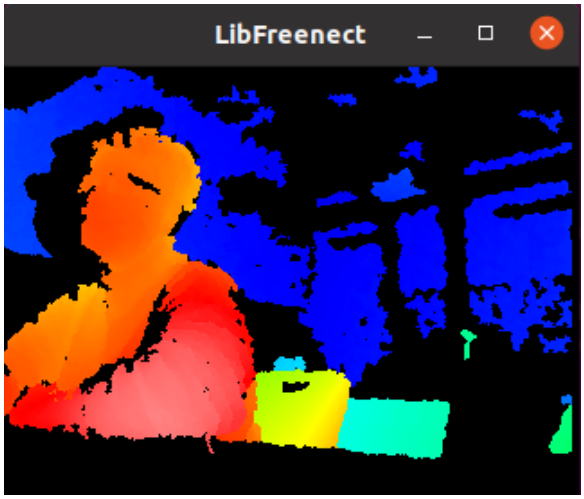


Figura 30: Resultado de correr el ejemplo *chunkview*.

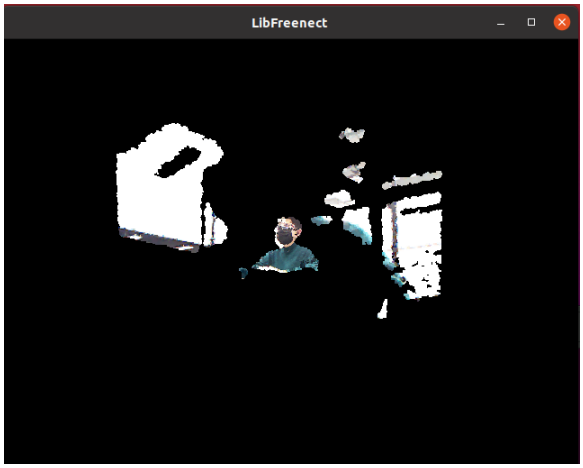


Figura 31: Resultado de correr el ejemplo *glpclview*.

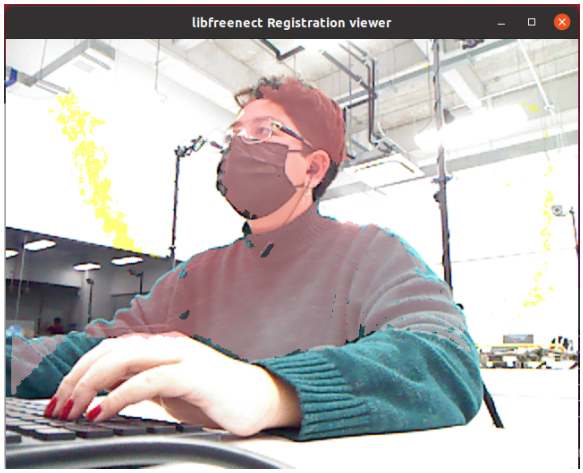


Figura 32: Resultado de correr el ejemplo *regview*.

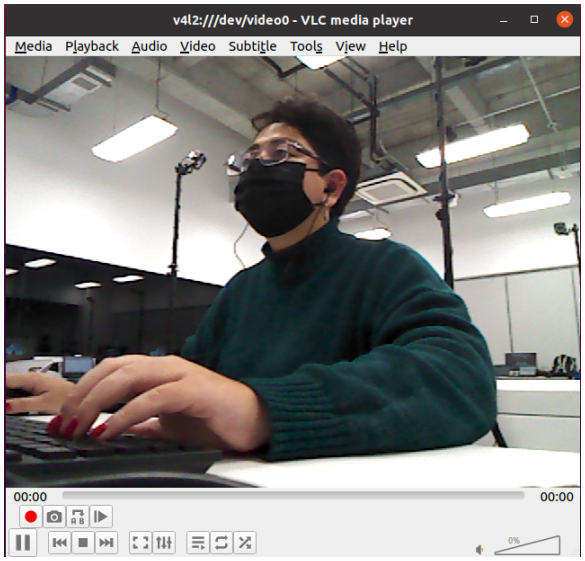


Figura 33: Funcionamiento del programa para extraer información desde el puerto serial.

10.3. Módulo de detección de marcadores ArUco

Una vez el sistema reconoce al Kinect como cámara, se puede utilizar OpenCv para realizar módulos de visión por computadora. Para probar su funcionamiento, se manejo la cámara igual que la Raspberry Pi Cam con los 3 programas y se obtuvo un módulo de detección de marcadores ArUco funcionando con la cámara Kinect. De igual manera que con la RaspiCam, todo el procesamiento de la imagen se realiza sobre la computadora en la que esta montada la cámara.

Al igual que con la otra cámara lo primero que se realizó fue la calibración de la cámara, utilizando imágenes tomadas por la cámara de un tablero de calibración, aplicarles un algoritmo de detección de esquinas y comparar los datos obtenidos con la distancia real entre los cuadros de los tableros. Este algoritmo se puede observar en la Figura 34.

Ya con estos datos, se puede realizar el módulo de detección de marcadores ArUco, utilizando el diccionario de OpenCV para obtener los datos de identificación de los marcadores. De igual manera, primero se imprimieron los datos sobre una sola imagen (Figura 35) y luego se modificó para estar imprimiendo la información del marcador según se detectara en el video.

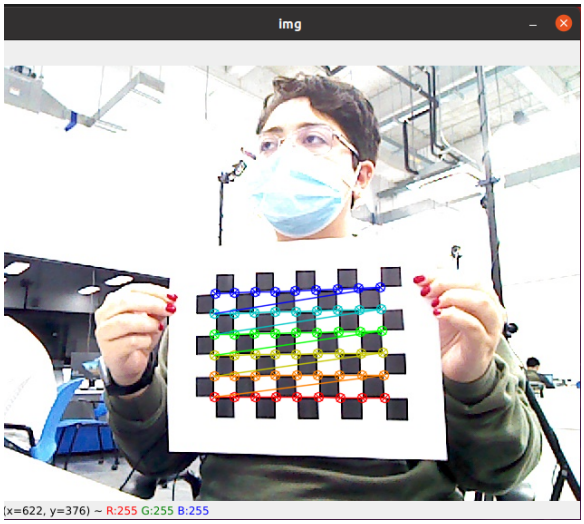


Figura 34: Algoritmo de detección de esquinas para calibración de la cámara.

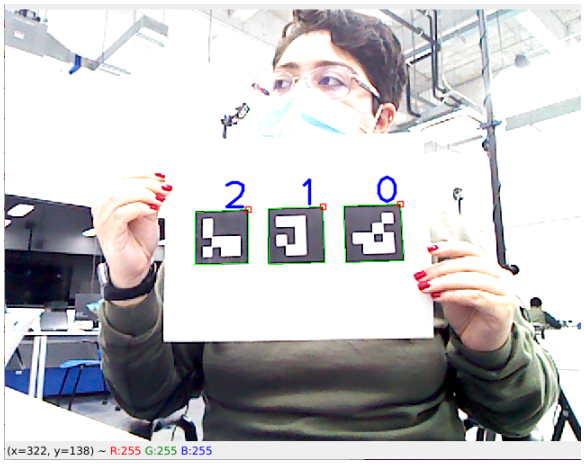


Figura 35: Detección de maracadores ArUco de imagen.

CAPÍTULO 11

Integración a ROS

La unión de los diferentes módulos del robot, al igual que el control de este, se realizaron en el sistema operativo de ROS FoxyFitzRoy. Por lo tanto, fue necesario realizar la integración de la cámara como nodo de ROS.

Los módulos de ROS tienen dos modelos de comunicación: a través de tópicos con un modelo de *publisher/subscriber* (en donde un módulo esta constantemente publicando información y hay otro módulo "suscrito" al canal constantemente recibiéndola) y servicios con un modelo de *client/server* (donde es necesario que el servidor escriba un mensaje pidiendo la información al cliente, el cliente responde con un mensaje de confirmación y la información necesaria y el servidor responde con un mensaje de recibido). Además, existen algunos librerías de ROS que ya cuentan con la estructura para implementar ciertos tipos de comunicación, tal como el paquete de odometría que se utilizó.

Para realizar la conexión de la cámara a ROS, fue necesario integrarlo como *publisher* de odometría, tomando en cuenta la cantidad y el tipo de datos que se quería transmitir. Fue necesario enviar la información se podía obtener de la cámara y que pudiera servir en el control general del robot.

Para probar la comunicación, se creo un espacio de trabajo sobre la computadora Raspberry Pi 4 que serviría de prueba. Sobre esto, se creo un nodo y un tópico para estar publicando la información. Aunque se intentaron algunas alternativas de comunicación, lo que funcionó mejor fue el programa que utilizaba la librería para mandar la información de odometría de ROS. Esta librería ya se estaba utilizando en la comunicación con los módulos DWM1001 del rover, por lo que se utilizó ese programa como base. Con esta base, se logró el envío constante de las variables de identificación de los marcadores, las coordenadas de posición y los cuaterniones.

Lo primero que hace el archivo es llamar todas las librerías que se utilizan para ROS y

para la conexión con la cámara. De parte de ROS, del paquete de *nav msg* se utiliza la librería de odometría y del paquete de *geometry msg* se utilizan librerías de pose, cuaterniones, punto, entre otras. Luego, se realiza la conexión del programa con la cámara, utilizando la librería de serial de Python. Se llama también el *bash script* que llama la configuración correcta de la cámara para obtener los datos necesarios para asegurar que no sea necesario realizar esta configuración por aparte.

La siguiente función inicia el modo de publicador del nodo y configura el intervalo de tiempo entre mensajes como 0.5 segundos. Luego, lee la información obtenida de la cámara y separa los datos. El mensaje que envía la cámara al detectar un marcador contiene 10 variables:

- Clave
- Identificación
- Coordenada x
- Coordenada y
- Coordenada z
- Tamaño de marcador 1
- Tamaño de marcador 2
- Dimensión (valor siempre igual a 1)
- Cuaternión 1
- Cuaternión 2
- Cuaternión 3
- Cuaternión 4

Por lo tanto, se tiene la condición que solo se publica la información si el mensaje que recibe de la cámara esta completo. Con las librerías de odometría, se envían las coordenadas como variable de posición mientras que los cuaterniones se envían como orientación. La variable de identificación del marcador se envía también como parte de la información. Los tamaños de marcador y el valor de dimensión se mantienen siempre constantes, por lo que no es necesario enviarlos al robot.

Al correr el paquete desde ROS y realizar una instrucción de *echo* del tópico en otra terminal para "escucharlo", se puede observar como se publica la información. Esto se puede visualizar en la Figura 36.

Este archivo entonces llama a la configuración correcta de la cámara, recibe y separa la información que le ingresa para tener el valor de cada parámetro guardado y publica esta información para ser escuchado por el módulo suscrito que es el de control del rover. Con esto, se cumple el objetivo de unir el sistema de visión por computadora por medio de un nodo de ROS para ser implementado en la plataforma.

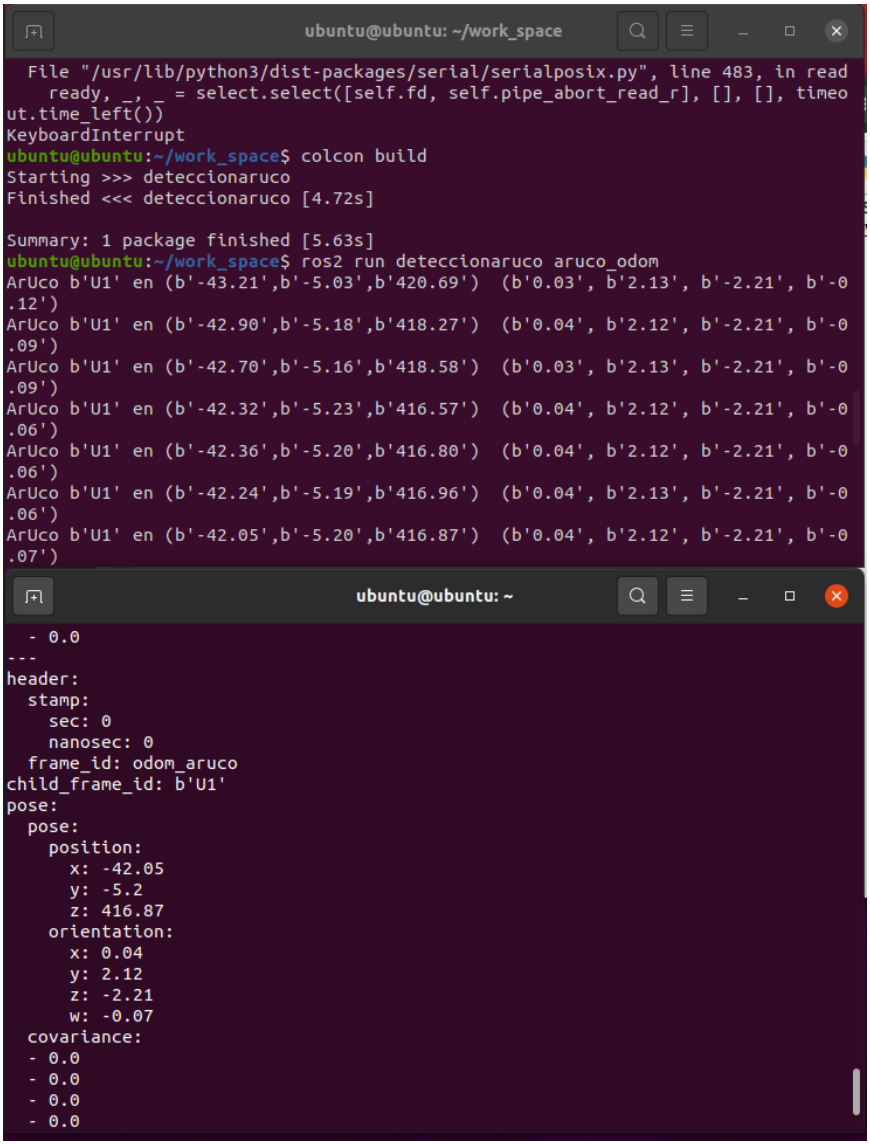


Figura 36: Información publicada en el tópico de ROS.

CAPÍTULO 12

Características del módulo de visión

El módulo de cámara que se eligió para el proyecto fue la JeVois Smart Machine Vision Camera. Una vez se decidió esto, fue necesario realizar pruebas para comprobar la información que se recibía de esta y poder determinar sus características para la implementación óptima sobre la plataforma robótica. Además, se analizaron los datos que se reciben de la cámara para obtener la desviación estándar de las medidas ya que esto se utilizó en el Filtro de Filtro de Kalman: de integración de sensores en el control de la plataforma robótica.

Como se mencionó en el capitulo anterior, la cámara despliega 10 datos con la configuración que se le colocó para detectar el marcador. De estos, 8 son importantes: la identificación, las coordenadas X, Y & Z y los 4 cuaterniones. Para su integración al sistema de control del robot fue necesario obtener un valor de desviación estándar para los parámetros cuantitativos. Es importante mencionar que una de las suposiciones del filtro de Kalman que se está utilizando es que los datos obtenidos tienen una media guassiana para poder implementarse sin problema dentro del filtro, lo que significa que se puede asumir que se mantiene constante y se puede calcular.

Se definieron varias pruebas para determinar la información del alcance de la cámara al igual que la confiabilidad de sus datos. La primera prueba consistió en colocar los marcadores separados en incrementos de 10 cm y realizar el algoritmo de detección de marcadores, como se puede ver en la Figura 37. Esto se ingresó a un ciclo que, por 50 iteraciones, guardara los parámetros de los marcadores en un documento de texto. Esto permitiría confirmar la precisión de la lectura de profundidad (Z), obtener la desviación estándar de los datos (ya que no se altera la posición del marcador), y observar los límites de profundidad de la cámara.

Los datos de los documentos de textos se analizaron en Excel. La desviación estándar que se utilizó en la plataforma de cada uno de los datos fue el promedio de todas las desviaciones a cada uno de las distancias medidas. Estas desviaciones se encuentran detalladas en el Cuadro 2. Estas medias se pueden también observar en la Figura 38 de manera gráfica.

Figura 37: Prueba de profundidad.

Medida	Media de Desviación Estándar (mm)
X	0.52
Y	0.62
Z	9.25
$Q1$	0.64
$Q2$	1.06
$Q1$	0.01
$Q4$	0.79

Cuadro 2: Medias de desviaciones estándar de las medidas obtenidas de la cámara.

Algo que se puede concluir de estos datos es que la desviación estándar de la mayoría de los datos es relativamente baja, a excepción de la variable para los datos de profundidad. La desviación estándar de la medida de profundidad incrementaba exponencialmente mientras incrementaba la distancia desde la cámara, por lo que el promedio de la desviación aumento.

Esta prueba también permitió realizar la comparación de las medidas de profundidad que se obtenían de la cámara y los datos en la vida real. La comparación de las medias obtenidas con la profundidad real se puede observar en la Figura 39. Además, se puede ver la gráfica del error experimental de las medidas obtenidas en la Figura 40. Se puede observar que las medidas obtenidas de profundidades cercanas son las que más se asemejan a sus valores reales. El valor obtenido a profundidades mayores varía un poco a partir de 800 mm. De igual manera, el error experimental incrementa exponencialmente a partir de 800 mm hasta llegar a un 14 % de error. Por esto mismo se decidió tomar en cuenta los datos hasta 1000 mm o 1 metro.

Una vez se analizaron estos datos, se realizaron pruebas para observar los límites en el eje horizontal y vertical a diferentes distancias de las cámaras también. Para obtener los

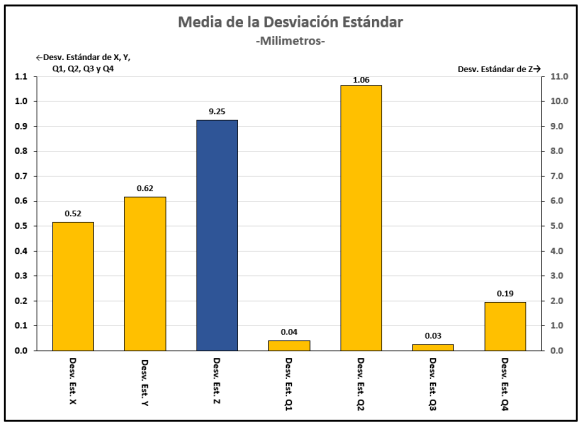


Figura 38: Medias de desviaciones estándar de diferentes medidas.

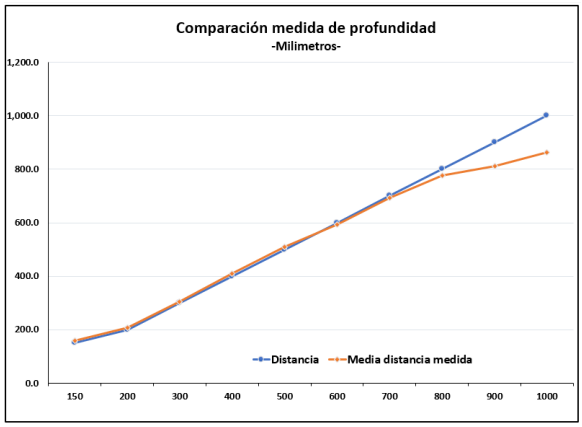


Figura 39: Comparación de promedios de profundidades obtenidas con las medidas verdaderas.

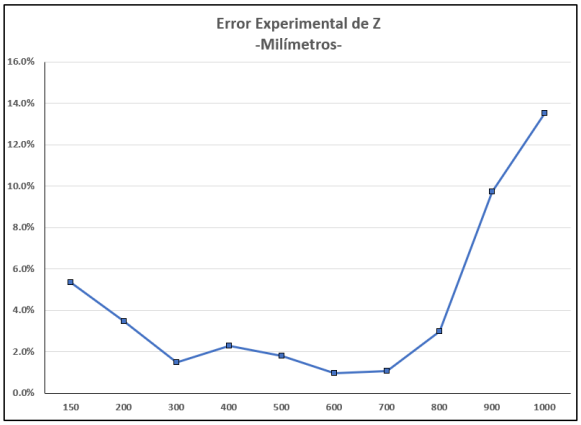


Figura 40: Gráfica de error experimental de medida de profundidad.

límites en el eje horizontal, se cambió un poco la prueba, colocando el marcador en diferentes posiciones del eje horizontal y tomando en cuenta las distancias mínimas y máximas a las que detectaba el marcador.

Con esta prueba, realizada a 150, 250, 500, 750 y 1000 mm de la cámara, se logró obtener

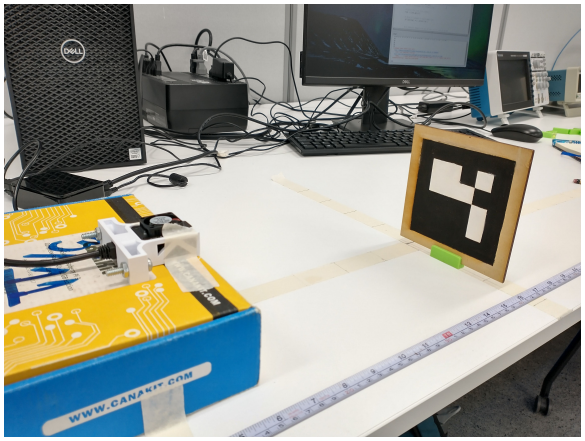


Figura 41: Prueba de eje horizontal.

el rango de la cámara. Como se puede esperar, el rango de la cámara va incrementando con la profundidad del marcador a la cámara. Es necesario tomar en cuenta para estas dos pruebas que para ser reconocido, el marcador tiene que estar completamente visto por la cámara, por lo que las distancias mínimas y máximas que se obtuvieron son las distancias a las que se debe encontrar el centro del marcador para ser reconocido.

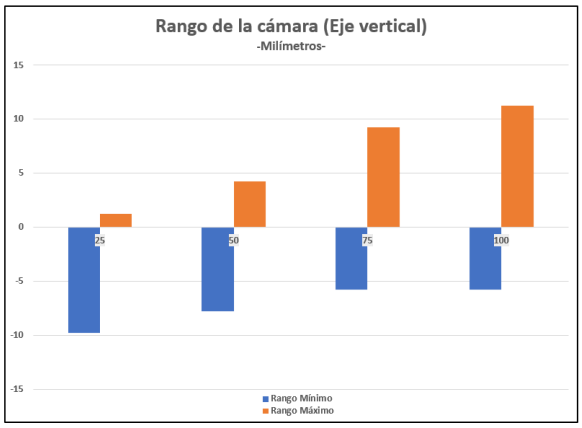


Figura 42: Rango de la cámara.

Para el eje vertical, se tomó en cuenta la altura a la que estaba la cámara para compararla con las alturas a las que todavía se detectaba el marcador. De igual manera se tomaron las medidas a diferentes profundidades para tener idea del rango completo de la cámara. Esto se puede observar en la Figura 43.

Luego de todas estas pruebas realizadas se obtuvieron los datos completos de la cámara, se comprobó la confiabilidad de los datos obtenidos de esta y se pudo determinar la ubicación que deben tener los marcadores para las pruebas según la posición de la cámara sobre la plataforma robótica.

Otra cosa que se comprobó de la cámara fue su manera de comunicación de datos. Ya se había comprobado la comunicación serial por medio de una sola conexión a puerto USB 3.0 con la Raspberry Pi 4 o con dos conexiones a puerto USB 2.0. Sin embargo, otra de

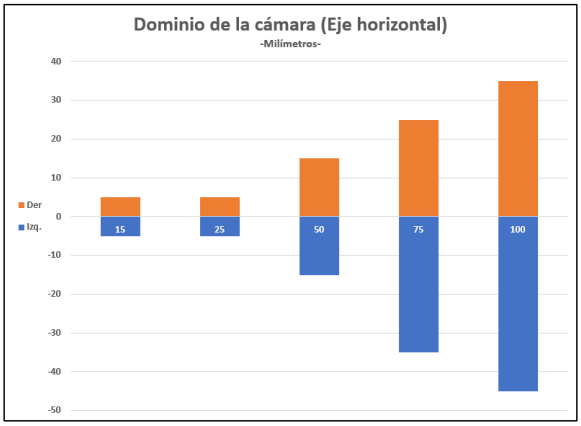


Figura 43: Dominio de la cámara.

las ventajas de la cámara es que también puede enviar datos por medio de la comunicación UART con conexión de 4 pines. Estos se conectan a los pines de tierra, voltaje de referencia, el pin TX y el pin RX. Sin embargo, es necesario igual conectar la cámara a alguna fuente de poder por medio del cable USB ya que por ahí se obtiene la corriente y el voltaje para hacer funcionar la cámara. Lo único que se debe cambiar en el programa es la manera en la que se llama la configuración de la cámara para hacer que los datos se transfieran por medio de los pines conectados.

CAPÍTULO 13

Integración a plataforma robótica

Una vez seleccionada la cámara, realizado el estudio de las propiedades de la cámara y probado el nodo de ROS, se empezó la integración del módulo de visión a la plataforma robótica móvil. Al inicio del proyecto, se definió una prueba que utilizaría la cámara para identificar "estaciones" denotadas por marcadores ArUco. Por cuestiones de tiempo, no se llegó a integrar la visión por computadora al control del proyecto. Sin embargo, sí se montó la cámara al robot y este hacía una función de mapeo de los marcadores. Esto puede llegar a integrase al control en un futuro o podría utilizarse para verificar los datos obtenidos de los demás módulos de ubicación y mapeo del robot.

13.1. Marcadores

Para la identificación de los marcadores ArUco, es importante considerar también los marcadores que se estén tratando de identificar con la cámara. Estos no pueden estar convexos, deben tener un tamaño definido, tiene que tener un fondo para poder diferenciar el marcador y tienen que tener un contraste claro en sus colores para poder identificar la matriz interior. Considerando que las pruebas se iban a hacer en la plataforma del laboratorio de la Universidad, se decidió crear marcadores especiales para las pruebas.

Los marcadores que se crearon se mandaron a cortar por láser en madera MDF. En este material, es posible rasterizar y cortar la madera. Esto permitió el corte de cuadrados de 12.5 cm por 12.5 cm. Dentro de esta figura, se colocó el marcador con medidas estándar (10 cm por 10 cm) creado con [25]. Esto permitió rasterizar la parte negra del marcador. Por último, los marcadores cortados se pintaron para agregar más contraste. Los marcadores finales se pueden observar en la Figura 44. Estos marcadores sirvieron para todas las pruebas de la cámara y se colocaron en la plataforma a la hora de hacer pruebas ya con el robot.

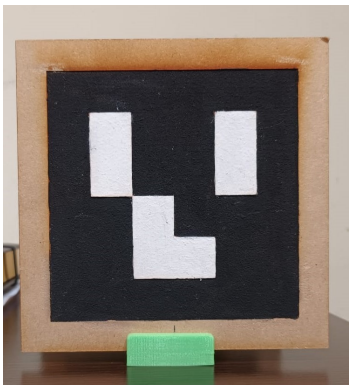


Figura 44: Marcador ArUco que se utilizó para las pruebas.

13.2. Montadura sobre el robot

Luego, fue necesario coordinar la posición de la cámara en el robot. Esta debería estar colocada en el frente del robot. Además, debería de estar asegurada al robot de alguna manera.

Lo primero que se realizó fue un soporte para la cámara. Este se imprimió en una impresora 3D y tenía el espacio para la cámara, sus cables y dos tornillos para asegurar el soporte. Su montadura se puede ver en la Figura 45 señalada con el cuadrado rojo.

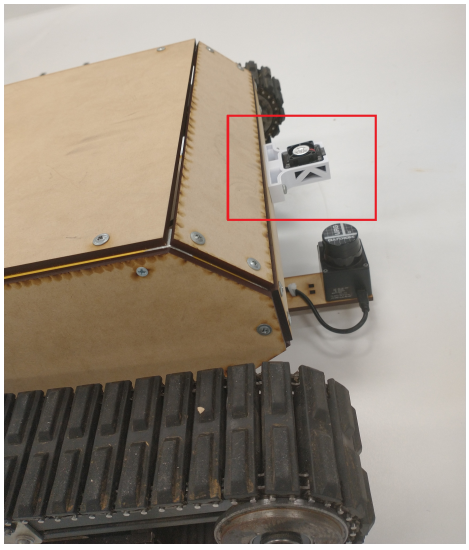


Figura 45: Soporte colocado en la parte de enfrente del robot.

Antes de probar el movimiento del robot o el funcionamiento con el nodo de ROS, se probó la identificación de los marcadores en la plataforma. Esto se hizo para comprobar que a la altura en la que estaba la cámara, se pudieran ver los marcadores, para confirmar el dato del rango de la cámara obtenido anteriormente. Esta prueba se puede ver la Figura 47. Como se puede ver, los marcadores son identificados por la cámara en la pantalla de la computadora.

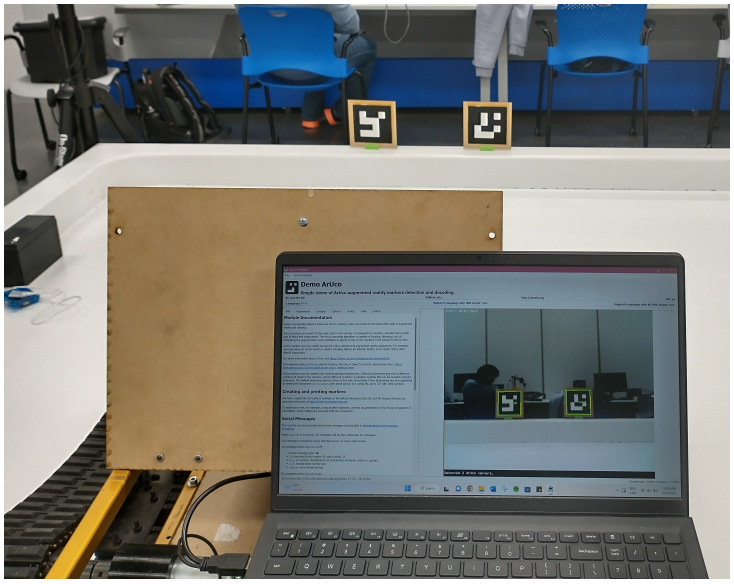


Figura 46: Prueba de identificación de marcadores con la cámara montada en el robot.

La versión final del robot se puede observar en la Figura 47. Abajo de la cámara se encontraba el módulo de LiDAR y en la parte de arriba el espacio para los módulos DWM1001 y el marcador del sistema Robotat.

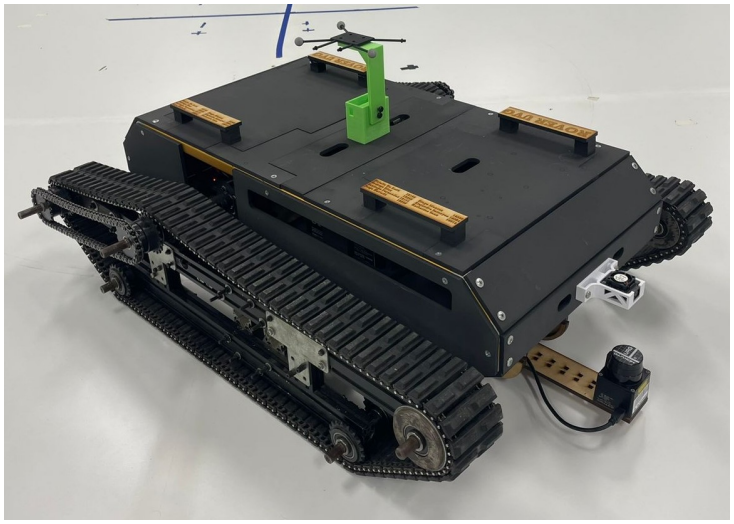


Figura 47: Versión final del robot explorador con la cámara montada.

13.3. Pruebas sobre el robot

13.3.1. Prueba de identificación

Una vez montado el módulo, se quiso probar la identificación de marcadores con el robot en movimiento. Para este punto, el robot no formaba parte del control ni estaba integrado

al nodo de control del robot. Ya que el movimiento del robot se podía realizar por medio de una aplicación separada, se utilizó esta aplicación para mover el rover por la plataforma y que fuera reconociendo los marcadores colocados alrededor utilizando el nodo de ROS que se creó anteriormente. En la Figura 48 se puede ver el robot en movimiento, con los marcadores colocados en diferentes partes de la plataforma. Por otro lado, se puede ver en la Figura 49 el nodo de ROS corriendo. Se puede ver como se identifican los marcadores y cómo va cambiando la distancia del marcador al irse acercando el rover.

Figura 48: Versión final del robot explorador con la cámara montada.

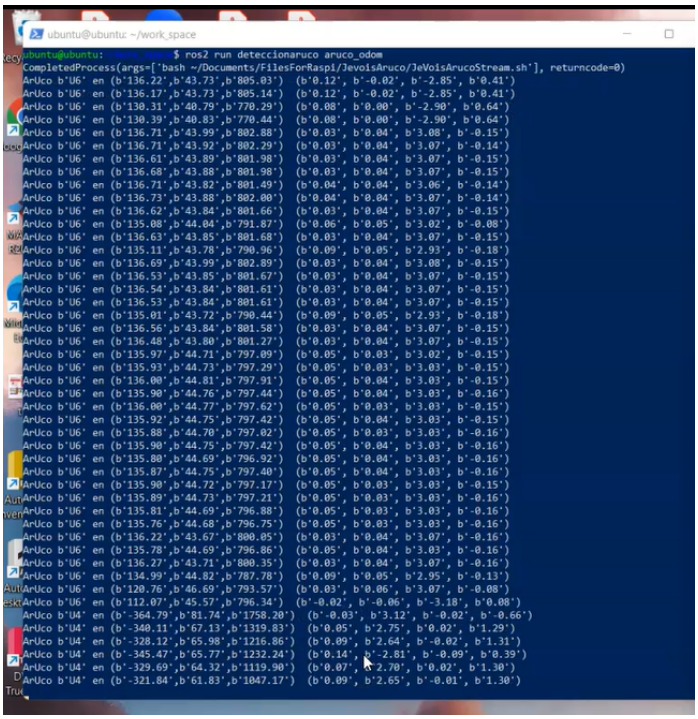


Figura 49: Nodo de ROS funcionando durante prueba de identificación en el robot.

13.3.2. Aplicación de mapeo

Aunque no formara parte del control, se logró integrar la visión por computadora al robot con una aplicación de mapeo. Esto se realizó con otro nodo de ROS, el cuál estaba

suscrito al nodo publicador que se creó para la lectura de datos y utilizaba esa información para crear transformadas.

Con ambos nodos corriendo, la posición de la cámara y de los marcadores se graficaba en la interfaz gráfica de ROS RViz. La prueba con el robot se puede ver en la Figura 50. Se estaban utilizando los marcadores 1 y 4. El mapeo de los marcadores se puede observar en 51. De esta manera, el modulo de visión por computadora quedó integrado a la plataforma robótica móvil por medio de un nodo de ROS. El enlace para los videos de funcionamiento de esto se encuentran en el anexo.

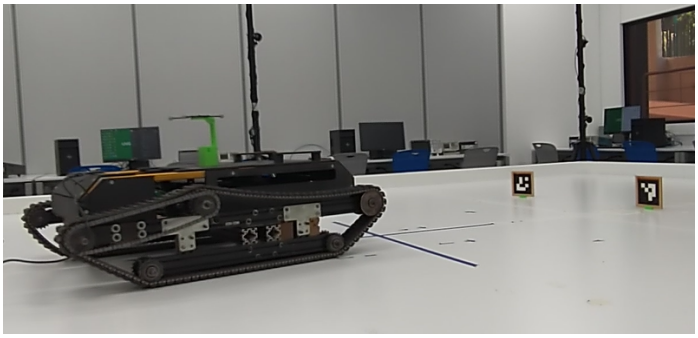


Figura 50: Prueba de mapeo con la cámara montada y los marcadores 1 y 4.

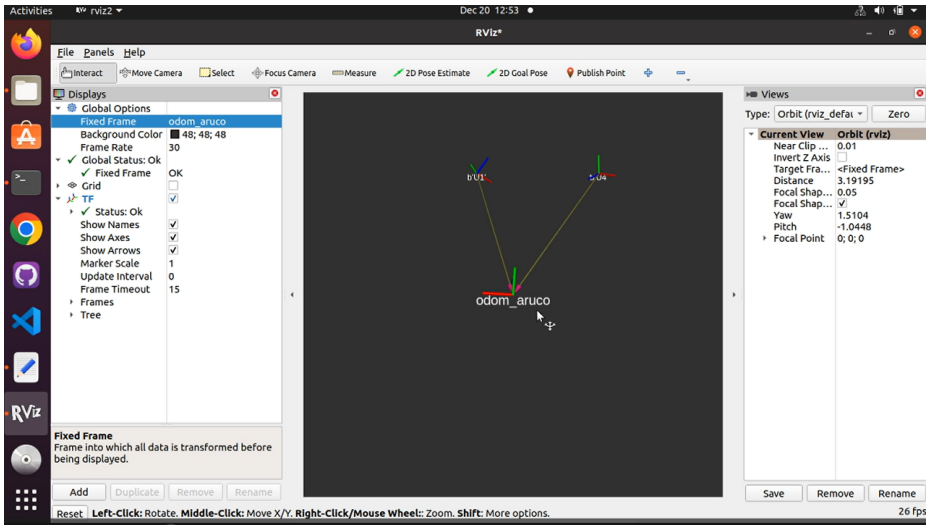


Figura 51: Aplicación de mapeo graficado en RViz.

CAPÍTULO 14

Conclusiones

1. Se logró la implementación de un módulo de visión por computadora de detección de marcadores ArUco a la plataforma robótica por medio de un nodo de ROS para una aplicación de mapeo.

2. Fue necesario determinar un sistema operativo compatible para trabajar sobre la computadora Raspberry Pi 4 y realizar una instalación que contara con el sistema operativo de Linux Ubuntu 20.04 y ROS 2 FoxyFitzRoy para la integración de los diferentes nodos de ROS sobre la plataforma robótica.

3. Para cumplir con la prueba que se le designó al modulo de visión por computadora sobre la plataforma, se logró un módulo de detección de marcadores ArUco en ambas cámaras: la Raspberry Pi Camera y la JeVois Smart Machine Vision Camera compatible con el sistema operativo de Linux que se estuvo utilizando sobre la plataforma robótica.

4. Tomando en consideración la configuración de los diferentes módulos de la plataforma robótica, se decidió que la cámara más adecuada para la visión por computadora es la JeVois Smart Vision Camera.

5. Se realizaron varias pruebas con el Microsoft Kinect para Windows y se logró encontrar la versión del soporte compatible con el sistema operativo que se estaría manejando en la plataforma.

6. Se realizaron varias pruebas para determinar las propiedades de la JeVois Smart Machine Vision Camera y confirmar la confiabilidad de los datos obtenidos de la cámara.

CAPÍTULO 15

Recomendaciones

1. Se recomienda que en las próximas iteraciones del Rover, si se va a cambiar la versión de ROS, se tenga en cuenta qué versión de ROS es la adecuada para el control del proyecto según el soporte que tenga, la compatibilidad con los módulos que se piensen agregar y el sistema operativo que se pueda implementar en donde se esté corriendo el programa.

2. Por los módulos que se decidieron integrar en esta ocasión, las pruebas se limitaron a realizarse dentro de un espacio definido en el laboratorio. En otra iteración del proyecto se pudiera utilizar la visión por computadora y el control por visión para reemplazar módulos como el de ubicación en el entorno de Robotat que restringió el área de trabajo y le puede dar al robot un sentido de mayor autonomía.

3. Se recomienda realizar más investigación con la camára de Microsoft Kinect para Windows. Desde Windows, donde tiene más soporte, se pueden realizar una gran variedad de aplicaciones con programas como Visual Studio.

4. En la instalación de *drivers* para el Microsoft Kinect, se recomienda seguir los pasos descritos en este trabajo y en las guías realizadas.

CAPÍTULO 16

Bibliografía

[1] J. Han, L. Shao, D. Xu y J. Shotton, "*Enhanced Computer Vision With Microsoft Kinect Sensor: A Review*," *IEEE Transactions on Cybernetics*, vol. 43, n.º 5, págs. 1318-1334, 2013. DOI: 10.1109/TCYB.2013.2265378.

[2] S. Zennaro, "*Evaluation of Microsoft Kinect 360 and Microsoft Kinect One for robotics and computer vision applications*," Tesis de licenciatura, Universidad de Padua, 2014.

[3] H. J. Sagastume, "Diseño Mecánico, Selección de Motores e Implementación de Sensores para un Robot Explorador Modular," Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[4] J. E. Archila, "Diseño e implementación de capacidades automáticas de navegación para un Robot Explorador Modular," Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[5] J. Guerra, "Algoritmos de Visión por Computadora para el Reconocimiento de la Pose de Agentes Empleando Programación Orientada a Objetos y Multihilos," Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[6] J. I. Ramírez, "Herramienta de Software de Visión por Computadora para Aplicaciones de Robótica de Enjambre en una Mesa de Prueba - Fase III," Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[7] H. A. Klée, "Desarrollo e implementación de algoritmos de visión por computadora clásicos empleando OpenCV en sistemas embebidos.," Tesis de licenciatura, Universidad del Valle de Guatemala, 2021.

[8] D. Forsyth y J. Ponce, *Computer Vision: A Modern Approach. (Second edition)*. Prentice Hall, nov. de 2011, pág. 792. dirección: https://hal.inria.fr/hal-01063327.

[9] R. Szeliski, *Computer vision: Algorithms and applications*, 2.ª ed. Springer Nature, 2022.

[10] A. I. Khan y S. Al-Habsi, "Machine Learning in Computer Vision," *Procedia Computer Science*, vol. 167, págs. 1444-1451, 2020, International Conference on Computational Intelligence and Data Science, ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2020.03.355. dirección: https://www.sciencedirect.com/science/article/pii/S1877050920308218.

[11] B. Siciliano, *Springer Handbook of Robotics*. Springer International Publishing, 2016.

[12] A. Burlacu y C. Lazar, "IMAGE BASED CONTROLLER FOR VISUAL SERVOING SYSTEMS," *Buletinul Institutului Politehnic din Iaşi. Secţia Automatică şi Calculatoare*, vol. 1, sep. de 2022.

[13] OpenCV, *OpenCV modules*, 2021. dirección: https://docs.opencv.org/4.5.5/.

[14] ——, *Detection of ARUCO markers*. dirección: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html.

[15] JeVois, *About JeVois*, 2019. dirección: https://www.jevoisinc.com/pages/what-is-jevois.

[16] R. Pi, *Raspberry pi documentation*, 2019. dirección: https://www.raspberrypi.com/documentation/accessories/camera.html.

[17] S. Hickey, A. Buck y M. Obajemu, *Kinect for windows - windows apps*, 2022. dirección: https://learn.microsoft.com/en-us/windows/apps/design/devices/kinect-for-windows.

[18] OpenKinect, *Main page*. dirección: https://openkinect.org/wiki/Main_Page.

[19] O. Robotics, *Why Ros?* 2021. dirección: https://www.ros.org/blog/why-ros/.

[20] *Install ubuntu on a raspberry pi*. dirección: https://ubuntu.com/download/raspberry-pi.

[21] MathWorks, *Kinect for Windows Hardware*. dirección: https://la.mathworks.com/help/imaq/kinect-for-windows-hardware.html.

[22] Microsoft, *Kinect for windows SDK*. dirección: https://www.microsoft.com/en-us/download/details.aspx?id=40278.

[23] ——, *Kinect for windows developer toolkit*. dirección: https://www.microsoft.com/en-us/download/details.aspx?id=40276.

[24] M. Kexugit, *Kinect for Windows SDK 1.8*. dirección: https://learn.microsoft.com/en-us/previous-versions/windows/kinect-1.8/hh855347(v=ieb.10).

[25] O. Kalachev, *ARUCO markers generator*. dirección: https://chev.me/arucogen/.

[26] K. Alexis, *The Kalman filter*. dirección: https://www.autonomousrobotslab.com/the-kalman-filter.html.

[27] Ubuntu, *Enterprise open source and linux*. dirección: https://ubuntu.com/.

CAPÍTULO 17

Anexos

17.1. Las imágenes creadas y sus guías

Se creó un fólder de Drive donde están subidas las imágenes de Ubuntu Mate y Desktop creadas para este proyecto para descargar e instalar para uso en Raspberry Pi 3 y 4. Además, se encuentran la guía creadas para la creación de las imágenes desde 0 y la guía creada para su instalación y primeros pasos de uso. Se puede acceder al fólder con el siguiente enlace: https://bit.ly/3eJ9ShA

17.2. Programación

Se creó un repositorio en GitHub el cual cuenta con la programación de las cámaras Raspberry Pi Cam y JeVois para el módulo de detección de marcadores ArUco, la documentación que se creó para la imagen y el archivo Bash que se utilizó para la conexión a internet, al igual que la programación de la transmisión en vivo utilizando la cámara RaspiCam. Además, cuenta con todo trabajado para la cámara Microsoft Kinect y el nodo de ROS. https://github.com/AleSamayoa/Vision-por-comptutadora-para-Rover-2022

17.3. Videos de funcionamiento

Los videos del funcionamiento de las pruebas de la cámara sobre el robot se pueden ver en el folder de Drive: https://bit.ly/3Ih4jUz

CAPÍTULO 18

Glosario

AMD: Se refiere a cierta manera de construir procesadores. Esta puede ser de 32 o 64 bits y es un tipo de arquitectura más común en computadoras modernas. 17

ARM: Se refiere a cierta manera de construir procesadores. Esta puede ser de 32 o 64 bits y es el tipo de arquitectura que utilizan las Raspberry Pi. 17

Filtro de Kalman: Es una implementación de un filtro matemático que se divide en una etapa de predicción y una de corrección que se utiliza en el campo de la robótica para integrar sensores y realizar control. [26]. 49

Raspberry Pi: Las Raspberry Pi son ordenadores de placas reducidas que manejan varios tipos de sistemas operativos. En el proyecto se utilizan los modelos de Raspberry Pi 3B y 4. A comparación de sus modelos anteriores, las Raspberry Pi 4 cuentan con 4 GB de memoria RAM, 2 USB 3.0 y 2 USB 2.0, más velocidad en su procesamiento, entre otras cosas. 1

thresholding: Según [9], es un proceso en la visión por computadora que se utiliza para hacer imágenes binarias. Esto quiere decir que se toma una imagen y según el valor de color de cada uno de los pixeles este se torna blanco o negro, según sea su valor y la referencia del proceso. 15

Ubuntu: Es un sistema operativo moderno y *open source* compatible con Linux. Existen diferentes versiones (se utiliza la 20.04 en el proyecto) y diferentes presentaciones como Ubuntu Server (sin interfaz gráfica), Ubuntu Desktop, entre otros. [27]. 11