

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,

2022

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,

2022

Vo.Bo.:

(f) _____
Dr. Luis Alberto Rivera Estrada

Tribunal Examinador:

(f) _____
Dr. Luis Alberto Rivera Estrada

(f) _____
(f) _____

Fecha de aprobación: Guatemala, de de .

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" hacerme sentir que siempre tendrá 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
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. Imagen para Raspberry Pi	21
7.1. Ubuntu Mate	21
7.2. Ubuntu Desktop	23
7.3. Conexión a internet	24
8. Módulos de visión por computadora	27
8.1. Adaptación de las cámaras a la imagen	27
8.1.1. Raspberry Cam	27
8.1.2. Módulo detección marcadores ArUco	28
9. Selección de módulo de cámara	33
10. Integración a ROS	37
11. Microsoft Kinect	39
12. Conclusiones	41
13. Recomendaciones	43
14. Bibliografía	45
15. Anexos	47
15.1. Las imágenes creadas y sus guías	47
15.2. Programación de Cámaras	47
16. Glosario	49

Lista de figuras

1.	Robot móvil sobre el cuál se le aplicó el 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, extraído de: [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.	<i>Desktop</i> 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.	<i>Desktop</i> 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 ícono pidiendo el input del usuario para conexión a una red nueva.	25
17.	Archivo que modifica el programa con el formato correcto.	25
18.	Imagen tomada por el programa de tablero de calibración.	29
19.	Algoritmo de 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 <i>screen</i> después de establecer parámetros. . . .	31
23.	Funcionamiento luego de llamar al <i>bash script</i>	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

Lista de cuadros

- ## 1. Resumen de características de ambas cámaras 34

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 a esta plataforma. Lo primero que se realizó fue un estudio comparativo entre dos módulos de cámara, la JeVois A33 y la Raspberry Camera, para definir uno como 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 medio fue integrado a la plataforma robótica. Se realizaron pruebas básicas con algoritmos de visión por computadora para comprobar su funcionamiento adecuado.

Abstract

Uno de los proyectos que se elaboró dentro de la Universidad del Valle de Guatemala en el año 2022 fue una plataforma One of the projects that was 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 the platform. To do so, a trade study was performed between two cameras, the JeVois A33 and a Raspberry 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, test were done to 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 to 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). El siguiente proyecto 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 en las pruebas finales y en el desarrollo de los módulos, lo primero que se realizó para el proyecto fue una imagen con la que se pudiera trabajar sobre una Raspberry Pi 3 con los programas necesarios. A esta imagen se le descargó el sistema operativo de Linux Ubuntu 20.04 y ROS Foxy, entre otras cosas necesarias para el trabajo. Se elaboró de igual manera una guía de instalación para que quede como referencia de los pasos a seguir. Se estuvieron realizando las pruebas de las cámaras sobre esta imagen, ya que la unión de todo estaría corriendo sobre una plataforma similar. Además, se elaboró otra variación de esta imagen compatible con una Raspberry Pi, que sería la computadora utilizada en las pruebas finales del proyecto.

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 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 mandando la información para el sistema general del robot y así se logró la integración con el resto de la máquina.

Otro de los objetivos era probar la cámara Microsoft Kinect. Se estuvieron realizando pruebas con esta cámara y se concluyó que no era compatible con el sistema operacional de las Raspberry Pi, ni con las librerías de la versión de ROS que se estaba utilizando. Por esto mismo, no podría ser implementado sobre esta versión de la plataforma robótica.

CAPÍTULO 2

Antecedentes

El siguiente trabajo busca 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 través 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 tesis *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, desarrolla 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.

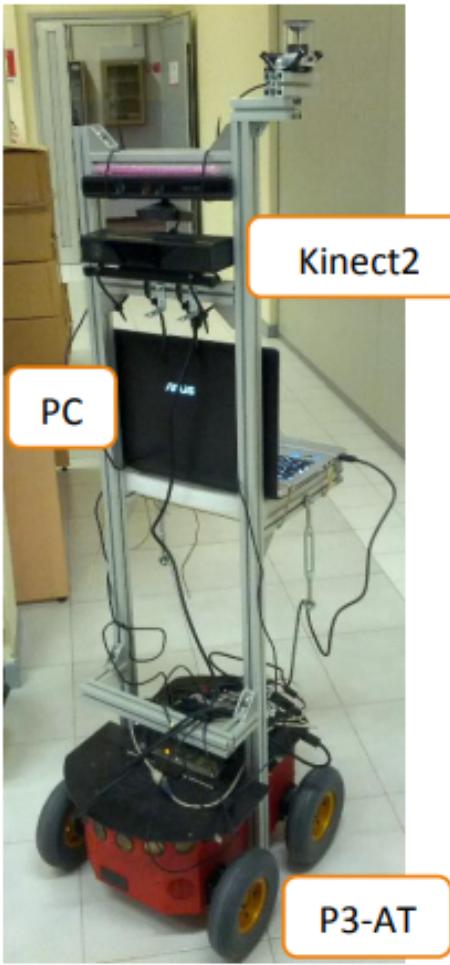


Figura 1: Robot móvil sobre el cuál se le aplicó el sistema de control basado en visión de la cámara Kinect [2].

2.2. Plataforma Móvil

Se han realizado diferentes iteraciones de un robot explorador en la Universidad del Valle a través 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 infrarrojos 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 Pi3 como computadora. Se obtuvo el robot que se muestra en la Figura 2 que funcionó a control remoto por medio de conexión al internet.



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

Este robot explorador no contó con un sistema de visión avanzado. Se utilizó un módulo de cámara USB que se conectó a la Raspberry Pi3 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 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 y la plataforma Open CV ya que este tuvo mejor desempeño. 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 dónde 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]. Además, 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 Cam y JeVois, se realizaron diferentes pruebas que detallen su funcionamiento. Tomando en consideración las pruebas que se realizarían 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. Sin embargo, se denominó que la cámara no es compatible con la plataforma.

Estas pruebas no solo detallan el funcionamiento de los 3 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

Objetivo General

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

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 una imagen compatible con la Raspberry Pi y con la versión de ROS que se estaría trabajando. Sobre esta imagen 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 Foxy FitzRoy.

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 trabajo 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 física 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.

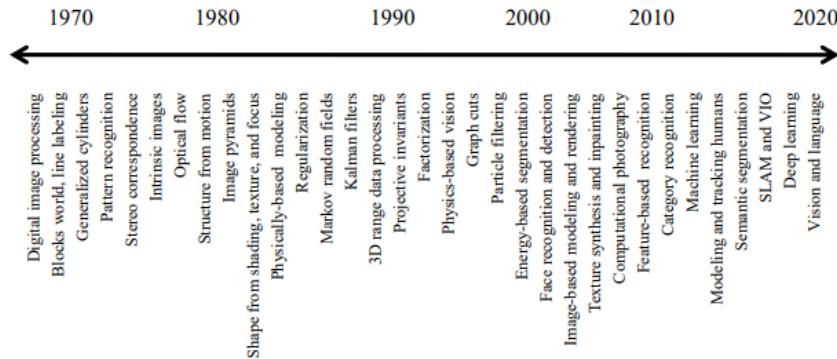


Figura 3: Historia de la visión por computadora, extraído de: [9].

En [9], se detalla el proceso de visión por computadora. La mayoría de cámaras realizan un proceso de capturar imágenes en 3 dimensiones y realizar la conversión correspondiente para obtener una imagen en 2D. A través de puntos y líneas 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.

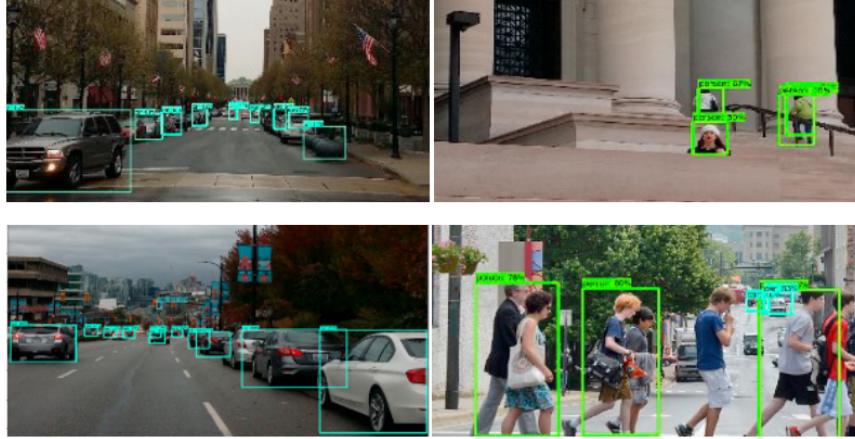


Figura 4: Ejemplo visión por computadora utilizando aprendizaje automático para reconocimiento [10].

En el campo de la robótica, la visión por computadora a 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.

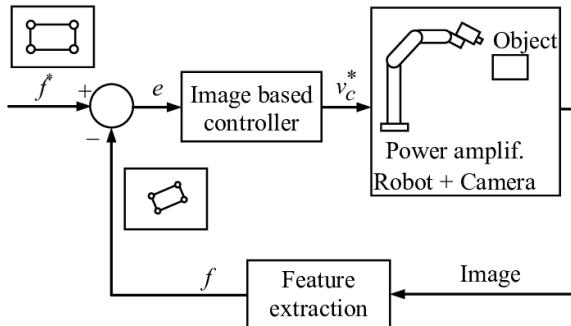


Figura 5: Descripción gráfica del proceso de control por visión [12].

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 están guardados estas matrices y sus respectivos identificadores. Lo primero que se hace en el algoritmo

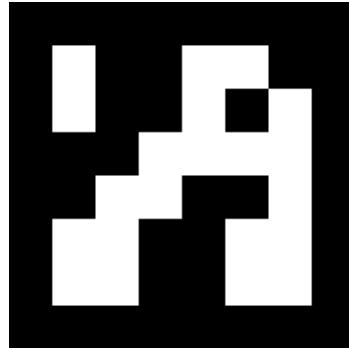


Figura 6: Ejemplo de marcador ArUco con identificación de 23, [14]

de detección es un proceso de *thresholding*, que es un proceso de visión por computadora en donde se combierte 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.

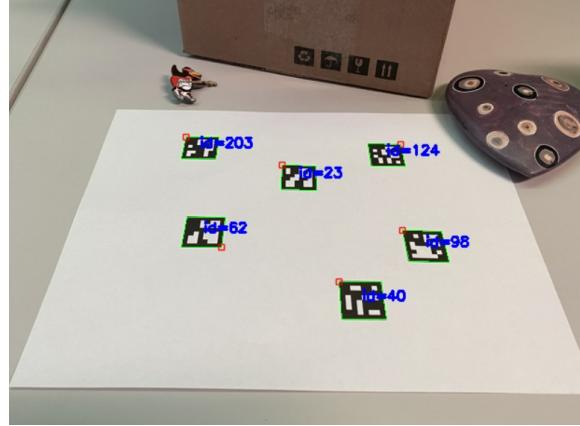


Figura 7: Imagen con marcadores identificados, [14]

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.

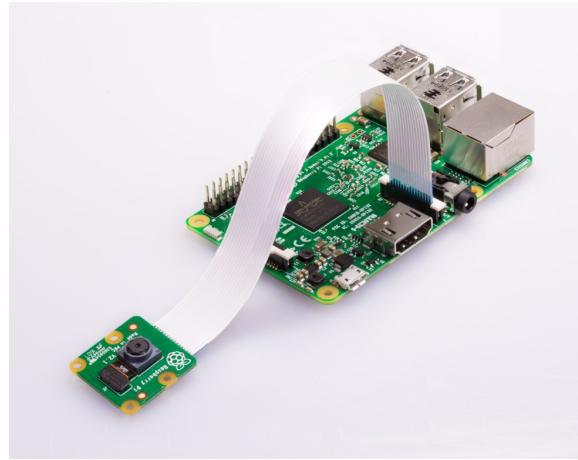


Figura 9: Cámara RaspiCam, [16]

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. 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 [17]. 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 a continuación, 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 [18]. 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

Imagen para Raspberry Pi

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 estaba trabajando todo desde un modelo Raspberry Pi 3.

Se decidió entonces crear una imagen que se podía instalar en la Raspberry 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 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 una imagen 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 [19]. 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 mismo, se decidió instalarle una interfaz gráfica simple a la imagen.

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.

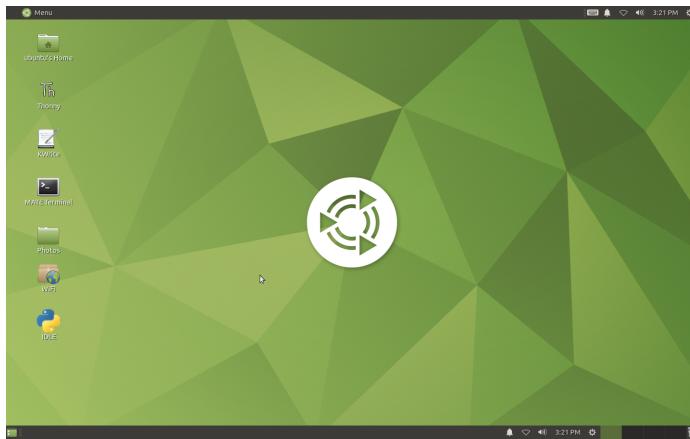


Figura 11: *Desktop* de la imagen creada para Raspberry Pi con Ubuntu Mate.

Luego se instaló la versión de ROS exitosamente y se realizó el proceso para poder llamarlo desde la terminal. Se realizaron pruebas simples tanto en ROS como en algunos otros programas y se comprobó su funcionamiento.

Esta imagen con el sistema, la interfaz gráfica, ROS y los otros programas, se creó utilizando el programa WinServer 32. Luego, se instaló esta imagen creada en otra Raspberry Pi 3 exitosamente.

Adicionalmente, se realizó una guía de uso rápido de la imagen, que incluye información de cómo conectarse al internet y el uso básico de la misma, al igual que una guía sobre la creación de la imagen desde 0, instalando la versión Server y desde ahí realizando la instalación del Desktop y los demás programas.

Esta imagen que se creó es compatible para los modelos Raspberry Pi 3 y 4, aunque hay una diferencia considerable en cuanto al rendimiento. Las Raspberry Pi 4, ya que cuentan con más memoria RAM, realizan procesos con mayor rapidez y permiten realizar más procesos simultáneamente.

```

ubuntu@ubuntu:~ 
File Edit View Search Terminal Help
ubuntu@ubuntu:~$ ros2
usage: ros2 [-h] Call 'ros2 <command> -h' for more detailed usage. ...

ros2 is an extensible command-line tool for ROS 2.

optional arguments:
  -h, --help            show this help message and exit

Commands:
  action    Various action related sub-commands
  bag       Various rosbag related sub-commands
  component Various component related sub-commands
  daemon   Various daemon related sub-commands
  doctor   Check ROS setup and other potential issues
  interface Show information about ROS interfaces
  launch   Run a launch file
  lifecycle Various lifecycle related sub-commands
  multicast Various multicast related sub-commands
  node     Various node related sub-commands
  param   Various param related sub-commands
  pkg      Various package related sub-commands
  run      Run a package specific executable
  security Various security related sub-commands
  service  Various service related sub-commands
  topic   Various topic related sub-commands
  wtf      Use 'wtf' as alias to 'doctor'

Call 'ros2 <command> -h' for more detailed usage.
ubuntu@ubuntu:~$ 

```

Figura 12: Imagen creada corriendo ROS desde la terminal.

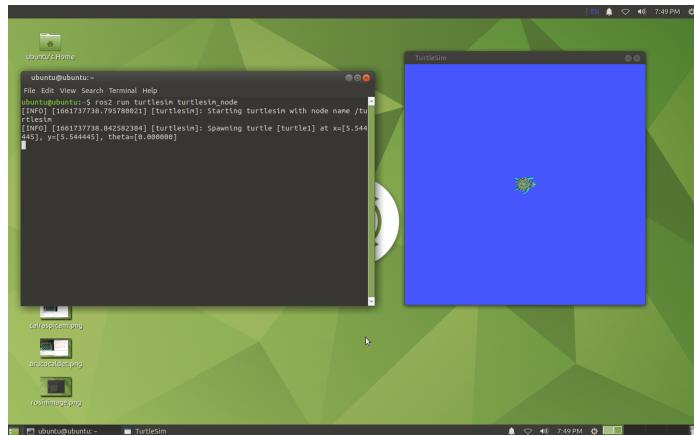


Figura 13: Imagen creada con ROS funcionando.

7.2. Ubuntu Desktop

Una vez se contó con el modelo 4, se creó también una imagen descargando Ubuntu Desktop. A esta instalación de Ubuntu se le descargaron los mismos programas que a la otra imagen y se realizó la instalación de ROS. Esta interfaz es un poco más completa y familiar para el usuario.

Esta interfaz se recomienda únicamente para el modelo 4. Al ser una interfaz más completa, 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*.

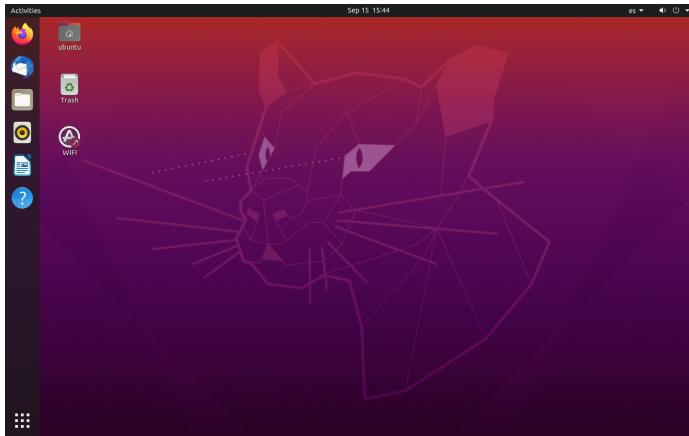


Figura 14: *Desktop* de la imagen creada para Raspberry Pi 4 con Ubuntu Desktop

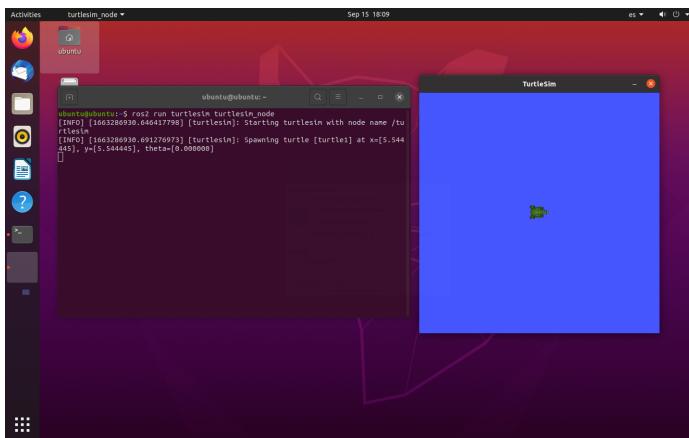


Figura 15: ROS funcionando en imagen con Ubuntu Desktop.

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 como 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 ciertos archivos 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 ambas imágenes creadas fue de que, aunque en la interfaz gráfica aparece la opción de realizar la conexión a una red eléctrica, no es posible realizarla de esta manera. 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. 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 ícono 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.



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

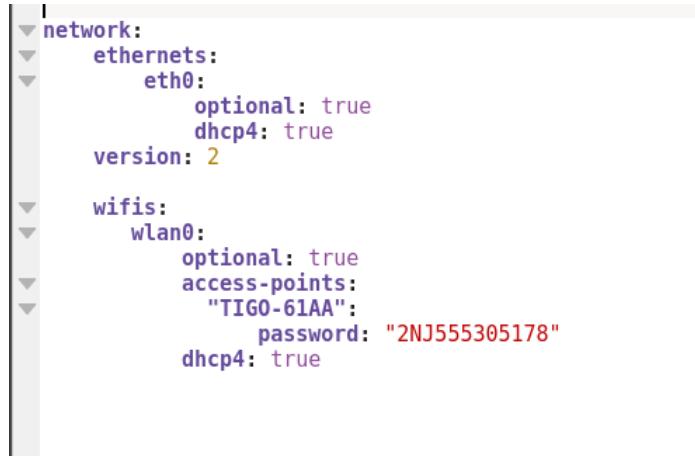


Figura 17: Archivo que modifica el programa con el formato correcto.

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.

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 consistía en la identificación de “estaciones” por medio de reconocimiento de códigos ArUco.

8.1. Adaptación de las cámaras a la imagen

Las primeras pruebas con las cámaras se realizaron en una Raspberry Pi que contaba con el sistema operativo de Raspbian. Luego, después de la creación de la imagen de Ubuntu, 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 Linux, 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*. Una vez se logró esto, se empezaron la creación de los módulos de visión acordados.

JeVois

La cámara JeVois fue un poco más complicada. Esta cámara ya cuenta con una imagen 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 2 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 imagen con la que cuenta la JeVois mientras se está 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 archivo de Bash, donde se establecen las configuraciones necesarias para que la cámara llame el módulo.

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

8.1.2. Módulo detección marcadores ArUco

Raspberry Cam

Luego de la adaptación de la RaspiCam a la imagen 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 frames, 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.

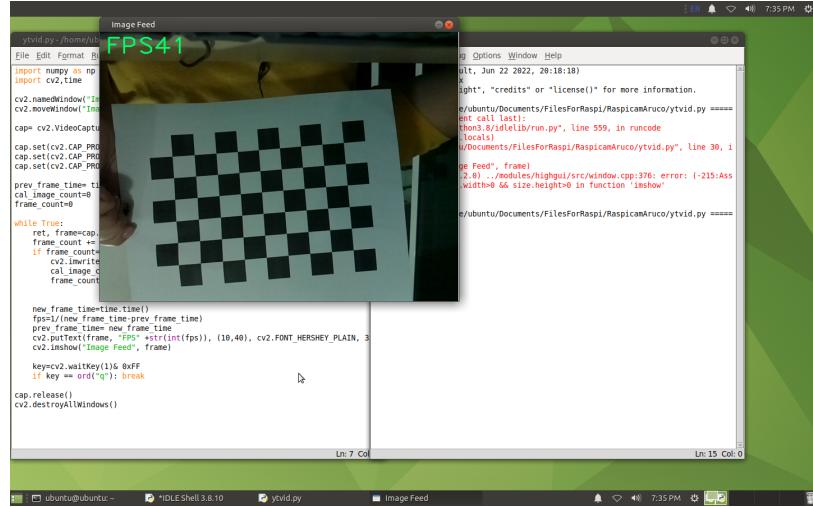


Figura 18: Imagen tomada por el programa de tablero de calibración.

El segundo programa usaba estas imágenes 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.

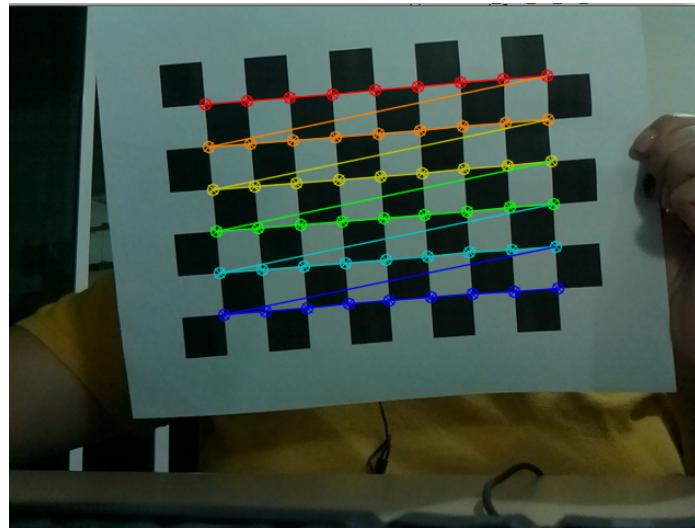


Figura 19: Algoritmo de detección de esquinas funcionando sobre imagen tomada de tablero de calibración.

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, Figura 21. El programa corría bien, con bastante rapidez y logró identificar todos los marcadores que se le mostraron.

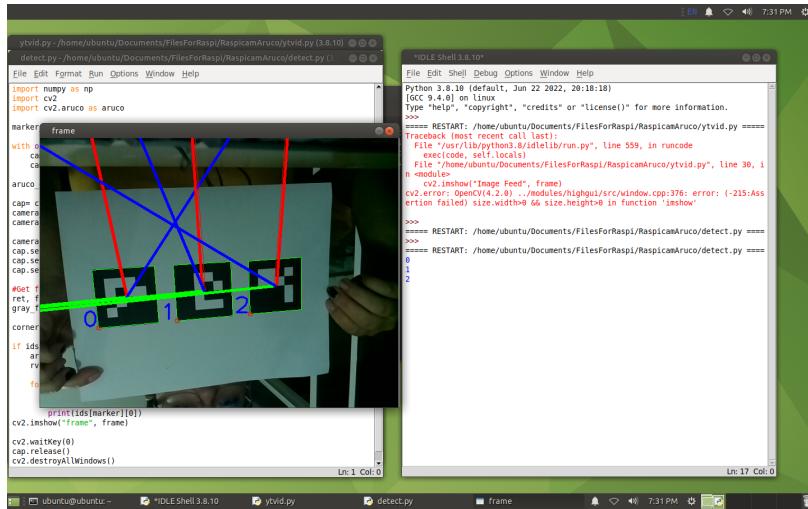


Figura 20: Prueba de información escrita sobre imagen e impresión de identificación en terminal.

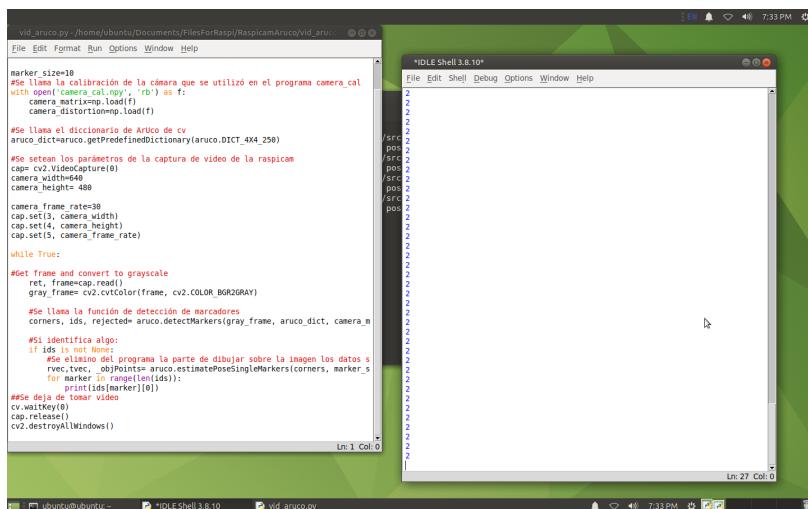
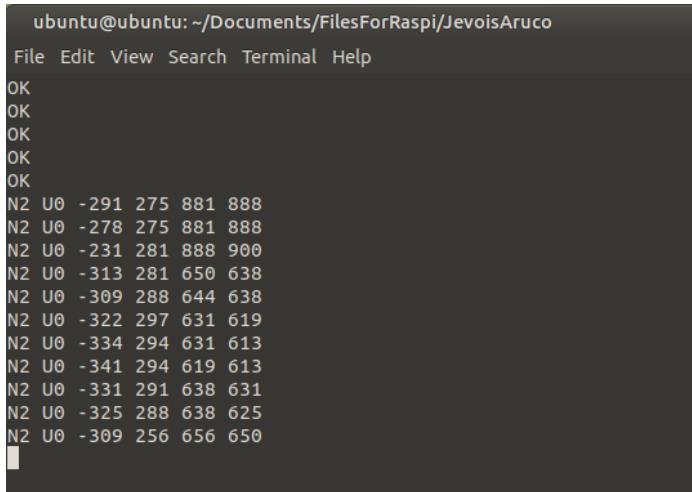


Figura 21: Prueba de identificación impresa en terminal con video.

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 llamar desde serial 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.



```
ubuntu@ubuntu: ~/Documents/FilesForRaspi/JevoisAruco
File Edit View Search Terminal Help
OK
OK
OK
OK
OK
N2 U0 -291 275 881 888
N2 U0 -278 275 881 888
N2 U0 -231 281 888 900
N2 U0 -313 281 650 638
N2 U0 -309 288 644 638
N2 U0 -322 297 631 619
N2 U0 -334 294 631 613
N2 U0 -341 294 619 613
N2 U0 -331 291 638 631
N2 U0 -325 288 638 625
N2 U0 -309 256 656 650
```

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

Esto funcionaba sin problemas, pero eran 5 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. 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 los mismos de una vez. Esto permitía iniciar la sesión de *screen* y colocar los valores necesarios en los parámetros de la cámara, como se puede ver en 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

```
ubuntu@ubuntu: ~/Documents/FilesForRaspi/JevoisAruco
File Edit View Search Terminal Help
N2 U0 -78 122 744 744
N2 U0 -69 128 713 719
N2 U0 -72 131 681 700
N2 U0 -81 141 663 681
N2 U0 -81 156 650 663
N2 U0 -75 166 638 644
N2 U0 -72 169 631 638
N2 U0 -69 169 625 625
N2 U0 -59 169 619 625
N2 U0 -56 159 613 619
N2 U0 -53 159 606 619
N2 U0 -56 169 613 613
N2 U0 -56 175 613 613
N2 U0 -63 178 613 606
N2 U0 -56 181 613 613
N2 U0 -50 188 613 613
N2 U0 -50 194 613 613
N2 U0 -41 197 619 619
N2 U0 -22 203 619 619
N2 U0 -13 209 625 619
N2 U0 -9 213 631 625
N2 U0 -13 216 625 631
N2 U0 -19 219 638 638
```

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

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, como se ve en 24. Con esto, se logró crear un programa que llamara el archivo de bash 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.

The screenshot shows two side-by-side IDLE shells. The left shell is titled 'idleSerial.py' and displays Python code for reading serial data from a Jevois camera. The right shell is titled 'jevoisSerial.py' and shows the raw serial data being received by the script.

Left Shell (idleSerial.py):

```
>>> #!/usr/bin/python
# Needed packages: sudo apt install python-serial
# This tutorial is a simple program that allows one to read and parse serial messages
serdev = '/dev/ttyACM0' # serial device of Jevois

import serial
import time
import subprocess

with serial.Serial(serdev, 115200, timeout=1) as ser:
    ser.write(b'HelloWorld\r\n')
    print(ser.read())
    print('Read a whole line and strip any trailing line ending character:')
    line = ser.readline().rstrip()
    print('Received: %s' % line)
    print(repr(line))

    # Split the line into tokens:
    tok = line.split()

    # Skip if timeout or malformed line:
    if len(tok) < 1: print('OK')

    # From now on, we hence expect: !D id x y w h
    if len(tok) != 6: continue

    # Assign some named Python variables to the tokens:
    key, id, x, y, w, h = tok

    print('Found Aruco %s () at (%d,%d) size %d x %d' % (key, id, x, y, w, h))
```

Right Shell (jevoisSerial.py):

```
File Edit Format Run Options Window Help
File Edit Format Run Options Window Help
#!/usr/bin/python
# Needed packages: sudo apt install python-serial
# This tutorial is a simple program that allows one to read and parse serial messages
serdev = '/dev/ttyACM0' # serial device of Jevois

import serial
import time
import subprocess

with serial.Serial(serdev, 115200, timeout=1) as ser:
    ser.write(b'HelloWorld\r\n')
    print(ser.read())
    print('Read a whole line and strip any trailing line ending character:')
    line = ser.readline().rstrip()
    print('Received: %s' % line)
    print(repr(line))

    # Split the line into tokens:
    tok = line.split()

    # Skip if timeout or malformed line:
    if len(tok) < 1: print('OK')

    # From now on, we hence expect: !D id x y w h
    if len(tok) != 6: continue

    # Assign some named Python variables to the tokens:
    key, id, x, y, w, h = tok

    print('Found Aruco %s () at (%d,%d) size %d x %d' % (key, id, x, y, w, h))
```

Bottom status bar: Un 228 Col:4 | Un 1 Col: 0 | 51 items in Trash

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 Raspi 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 la imagen de 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 imagen 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.

El nodo de ROS con el que se planeaba trabajar era capaz de obtener la información que se le mandaba al puerto serial y trabajar con esta.

Para cualquier módulo de visión por computadora que se desee con la Raspberry Cam es necesario crear el algoritmo desde 0. 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 imagen, 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 sistema operativo de ROS es una operación cargada, no sería tan recomendable estar llamando el procesamiento de la 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. Además, el nodo de ROS con el que se estaba trabajando requería la información obtenida desde un puerto serial. Mandar información de la cámara a un puerto serial implica más procesamiento sobre la computadora.

Por otro lado, la JeVois trae consigo sus propias ventajas y desventajas. Sí se pudo utilizar dentro de la imagen, pero únicamente llamando a los diferentes módulos disponibles dentro de la imagen por medio del puerto serial. Las librerías para modificar estos módulos y crear nuevos desde 0 y la interfaz gráfica creada por la compañía no son compatibles con la imagen que se creó. Por lo mismo, aunque los módulos que trae sí están a base de OpenCV, no se puede llamar a la cámara en los módulos creados con esta librería. 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 serial. No ocupa mucho espacio computacional dentro de la Raspberry Pi a la que esta conectada. Además, es fácil llamar a los otros módulos disponibles en la imagen, 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.

Característica	Raspberry Cam	JeVois
Compatibilidad con imagen	Sí (con OpenCV)	Sí (desde puerto serial)
Detección de marcadores ArUco	Sí	Sí
Comunicación por serial sin programación adicional	No	Sí
Compatibilidad con OpenCV	Sí	No
Implementación como nodo ROS	Sí (con comunicación a serial)	Sí
Facilidad de creación de nuevo módulo de visión	Sí	No
Procesamiento de módulo de visión	Sobre la Raspberry	Dentro de la cámara
Facilidad de llamar otros módulos de visión	No	Sí
Compatibilidad de librerías propias con la imagen	No	No

Cuadro 1: Resumen de características de ambas cámaras

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.

Una solución que se probó fue de mandar por comunicación TCP/IP la información de los datos de una Raspberry Pi a otra, pero el nodo de ROS requería comunicación serial. Las Raspberry Pi no cuentan con más de un puerto físico para recibir comunicación serial y este ya estaba ocupado por el módulo de potencia. Es por esto que se decidió que lo mejor sería conectar sobre la Raspberry Pi que estaría corriendo ROS la cámara que se seleccionaría.

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 ocupa mucho espacio computacional. Además, ya se está realizando la comunicación serial al tenerla conectada a la Raspberry Pi. Otra ventaja de la JeVois sobre la Raspberry 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 una imagen de Raspbian y con un programa de Python simple, se creó un módulo de transmisión en directo del video que toma la cámara. Esto se pudiera implementar sobre una Raspberry 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: Transmisión en vivo por medio de servidor Web utilizando la Raspberry Cam.

CAPÍTULO 10

Integración a ROS

Los módulos de ROS tienen dos modelos de comunicación: a través de *topics* con un modelo de *publisher/subscriber* (en donde un módulo esta constantemente publicando información y hay otro módulo “suscripto” al canal constantemente recibiéndola) y *services* 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 server responde con un mensaje de recibido).

Para realizar la conexión de la cámara a ROS fue necesario integrarlo como *topic*, ya que es de esta manera que se manejó el nodo en el control del robot como tal. Para realizar esto, se modificó el archivo dónde se tenía ya la configuración de la cámara.

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 suscripto que es el de control del rover.

CAPÍTULO 11

Microsoft Kinect

Para considerar la funcionalidad de cualquier módulo de cámara para el proyecto es importante tomar en cuenta los recursos disponibles de espacio y *hardware*. El Microsoft Kinect para Windows es una cámara bastante avanzada, que cuenta con recursos aptos para realizar un gran número de aplicaciones en el campo de la robótica, pero tiene una serie de requisitos que no son compatibles con el modelo de la Raspberry Pi.

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 tiene cuenta con una *toolbox* que permite la conexión del módulo [20], los requisitos a nivel de para correr la cámara son los siguientes:

- Procesador de bits
- 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.

El Kinect cuenta con un adaptador 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 al LED. El bus de la USB tiene que ser capaz de procesar una gran cantidad de información también.

Sobre la imagen de Ubuntu que se realizó para el uso en la Raspberry Pi 4, se instaló la librería de 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. Esto pudo haber sucedido por que el tamaño de información que puede pasar por los puertos USB 3.0 de la Raspberry Pi 4 no fue lo suficiente para procesar la imagen. Otro ejemplo que debía sacar la información del sensor de profundidad sacaba únicamente el valor de 0. La cámara sí estaba siendo detectada 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. Sin embargo, al probarlo no se vio ningún resultado.

Se concluyó entonces que no sería posible utilizar el módulo de Microsoft Kinect para Windows sobre la plataforma en una Raspberry Pi 4.

CAPÍTULO 12

Conclusiones

1. Fue necesaria la creación de una imagen para trabajar sobre la computadora Raspberry Pi 4 que contara con el sistema operativo de Linux Ubunut 20.04 y ROS 2 FoxyFitzRoy para la integración de los diferentes nodos de ROS sobre la plataforma robótica.
2. 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.
3. 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.
4. Se logró la implementación de este modulo de visión por computadora por medio de un nodo de ROS.
5. Se realizaron las pruebas necesarias y se determinó que el Microsoft Kinect para Windows no es implementable sobre una Raspberry Pi 4.

CAPÍTULO 13

Recomendaciones

1. Se recomienda que en las próximas iteraciones del rover, 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 dónde se este 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. La visión por computadora y el control por visión pueden ser un reemplazo para algunos de estos módulos que restringieron el área y le pueden dar al robot un sentido de mayor autonomía.
3. Aunque el Microsoft Kinect para Windows es una cámara con bastantes habilidades, su pérdida de soporte oficial y la complicación de utilizarlo en sistemas fuera de Windows es una gran desventaja y debe considerarse antes de querer utilizarlo en un proyecto.

CAPÍTULO 14

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,” 2014, Tesis de licenciatura.
- [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 Iași. Secția Automatică și 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/d5dae/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] OpenKinect, *Main page*. dirección: https://openkinect.org/wiki/Main_Page.
- [18] O. Robotics, *Why Ros?* 2021. dirección: <https://www.ros.org/blog/why-ros/>.
- [19] *Install ubuntu on a raspberry pi*. dirección: <https://ubuntu.com/download/raspberry-pi>.
- [20] MathWorks, *Kinect for Windows Hardware*. dirección: <https://la.mathworks.com/help/imaq/kinect-for-windows-hardware.html>.

CAPÍTULO 15

Anexos

15.1. Las imágenes creadas y sus guías

En este folder de Drive estan subidas las imágenes de Ubuntu Mate y Desktop creadas para este proyecto para descargar y flashear 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: <https://bit.ly/3eJ9ShA>

15.2. Programación de Cámaras

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 creo 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.

CAPÍTULO 16

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

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 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. 16

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.. 1