# UNIVERSIDAD POLITÉCNICA DE ATLACOMULCO

Ingeniería en Robótica

**PROYECTO DE ESTADÍA**
**"APLICACIÓN DE ROBOTS MÓVILES EN UNA FORMACIÓN DE ROBOTS MÓVILES DIFERENCIALES"**

Presentado por:
Francisco Javier Meléndez Ruiz

Profesor Asesor:
Dr. Erick Axel Padilla García

Atlacomulco, México. Septiembre 2022

GOBIERNO DEL
ESTADO DE MÉXICO

UPA
UNIVERSIDAD POLITÉCNICA DE ATLACOMULCO

Atlacomulco Estado de México,
a 20 de Septiembre 2022

**Asunto:** Carta de Terminación

**M.I.S.C. JOSÉ CUAUHTÉMOC MARQUÉS MARTÍNEZ**
**SUBDIRECTOR ACADÉMICO**
**UNIVERSIDAD POLITÉCNICA DE ATLACOMULCO**
**PRESENTE.**

Por medio del presente se informa que el **C. Francisco Javier Meléndez Ruiz** con número de matrícula **1731100429** alumno (a) de la Carrera de **Ingeniería en Robótica** de la Universidad Politécnica de Atlacomulco, ha concluido la **Estadía** dentro de la Institución, desarrollando el proyecto **APLICACIÓN DE ROBOTS MÓVILES EN UNA FORMACIÓN DE ROBOTS MÓVILES DIFERENCIALES** de cubriendo un total de 600 horas, durante el periodo comprendido entre Mayo – Agosto 2022.

Cabe mencionar que la participación del alumno(a) durante su **Estadía** es considerada como: **Satisfactoria** (mayor o igual a 7.0) de acuerdo con los parámetros establecidos en la evaluación anexa con una calificación de: **9.0**

Se extiende la presente para los fines que al interesado convenga. Sin otro particular por el momento quedamos a sus órdenes para cualquier duda o aclaración al respecto.

ATENTAMENTE

Asesor Profesor
Dr. Erick Axel Padilla García

## ÍNDICE DE CONTENIDO

## ÍNDICE DE FIGURAS



## ÍNDICE DE TABLAS



## ÍNDICE DE FORMULAS

# I INTRODUCCIÓN

Según la literatura, un robot móvil puede referirse a un sistema electromecánico que es capaz de desplazarse de forma autónoma, su sistema de referencia cambia gracias a dispositivos de locomoción, ya sea usando ruedas o patas.

De forma general, los robots móviles son usados como vehículos y diseñados para moverse dentro de un ambiente, principalmente, terrenos planos. Estos sistemas cuentan con grados de libertad, que permiten estudiar y controlar los movimientos del vehículo. En este trabajo, utilizaremos kits de robots móviles del tipo diferenciales, generalmente comandadas por una velocidad lineal y su orientación.

Uno de los temas de mayor interés en robots móviles, es guiar el movimiento del robot a un punto deseado o con una trayectoria deseada. Para esto, generalmente se usa información proveniente de los sensores del robot o con otro dispositivo externo, por ejemplo, a través de la información de una cámara. Esta información externa es la referencia de un espacio de trabajo por donde el robot puede desplazarse de forma segura. La interconexión entre robots y dispositivos externos, como la cámara, requieren de estudiar y seleccionar alguna estrategia, que permita reducir cierta incertidumbre del entorno, debe ser una estrategia capaz de determinar el sistema de referencia del robot móvil y tomar decisión sobre los actuadores del mismo.

Estas plataformas donde existe información del entorno para controlar un robot permiten ampliar las aplicaciones donde se requieren más de un robot para ejecutar una tarea definida. Dentro de estas aplicaciones, puede requerirse cooperación de varios robots, transporte de un material pesado o peligroso, la acción colaborativa entre los robots, vigilancia o inspección entre otras aplicaciones. Así surgen las necesidades de reconocimiento de imágenes, sistemas de visión u otro tipo de medición externa, que permita independizar a más de un robot en un mismo entorno de control. Esto también permite desarrollar estrategias de control, identificación, cooperación y formación de robots móviles, donde existe un reto añadido en las comunicaciones de los diferentes dispositivos.

En esta propuesta, se desea implementar una plataforma experimental que pueda utilizarse en la Universidad Politécnica de Atlacomulco, por ello, proponemos seleccionar y usar componentes de bajo costo, que pudiera reproducirse y utilizarse entre docentes y estudiantes. Así surge esta primera propuesta de control de robots usando una cámara, donde el enfoque principal es lograr crear y mantener formaciones de robots móviles con cierto grado de autonomía.

Los materiales, actuadores, sensores, estrategias y características que resuelven el problema de control de más de un robot a traves de una camara, se detalla en las siguientes secciones de este documento.

## II ANTECEDENTES

Existen muchos trabajos relacionados con formaciones de robots móviles, entre ellos uno de los autores mencionados se tiene a Keisuke Uto (Alguno de los trabajos en los que ha participado es (Takano, Obayashi y Uto, 2015)). Algo relevante sobre este autor es que buscó reducir costos utilizando como base un kit de robot móvil en ves de un robot completo comercial. El autor utilizo 9 kits de robots móviles con ruedas omnidireccionales de (Tosa Electric Inc., 2009), para lograr una formación usó comunicación multi-punto con módulos Xbee, marcadores visuales de 4x4 bits con corrección de errores y una cámara web para ubicar los marcadores. Para generar la estructura de formación, usó una iPad para dibujarla. Una de las presentaciones donde mostró este trabajo, puede verse en (Uto, 2011).

Una forma de trabajo para formaciones de robots móviles es usando un método de control cooperativo, como se puede ver en el trabajo (Alonso-Mora, Baker y Rus, 2017). En este trabajo se hizo un formación de robots móviles donde 3 robots omnidireccionales trabajan de forma cooperativa para llevar un objeto usando una trayectoria dinámica para evitar obstáculos y personas.

En la presentación de (Tang, Lee y Luc, 2016) mencionan varios tipos de formaciones que se pueden utilizar dependiendo de si en el entorno existen muchos, pocos o ningún obstáculo y si se requiere que cada robot se mantenga en el campo de visión de uno y el otro. También menciona las leyes de control global y local, donde en el control global se tienen metas globales y/o conocimiento global accesible a todos los robots mediante un informante central o por interpretación mediante el modelado de los robots y en el control local que se deriva en base a los sensores de cada robot.

Una forma de tener una referencia global para un control global, es con una cámara. Para control local se pueden usar encoders en los motores de cada robot.

## III DEFINICIÓN DEL PROBLEMA O PREGUNTA A RESPONDER

Para este proyecto de investigación, se propone un método experimental que de solución al problema de formación de robots móviles. Para esto, se utilizó una cámara web para obtener la posición y dirección de cada robot usando como referencia marcadores de realidad virtual *ArUco,* kits de *robots móviles diferenciales* que incluyen chasis, ruedas, motores y encoders rotatorios, placas de desarrollo *Arduino Nano* y *Arduino Uno* y módulos de comunicación *nRF24L01*. Para control se utilizaron dos *PID* por motor, uno para posición y otro para velocidad.

Uno de los objetivos de este proyecto es utilizar una plataforma de bajo costo que puede implementarse en la universidad o incluso reproducirse en cualquier institución.

Con este trabajo podemos extender nuevas formaciones con diferentes cantidades de robots. Por ejemplo, estudiar la comunicación entre la cámara y los robots.

## IV JUSTIFICACIÓN

Esta plataforma experimental permite que la universidad tenga un sistema de bajo costo reproducible por los estudiantes y docentes, con el cual, se pueden probar diferentes temas en la misma plataforma. Por ejemplo, estrategias de control multi-agente, prácticas relacionadas con el sistema de visión, estrategias de control cooperativo, entre otros.

Es una propuesta de una plataforma de arquitectura abierta, es decir, podemos modificar cualquier elemento de la plataforma, usar cualquier software de programación o añadir nuevos elementos sin ninguna restricción por algún fabricante. Así podemos extender la plataforma o probar nuevos elementos de cualquier marca, sin ninguna restricción. Esto tiene grandes ventajas académicas, siendo un sistema accesible para los mismos estudiantes de la universidad.

# V OBJETIVOS

## V.I OBJETIVO GENERAL

Implementar un método de bajo costo para la formación de robots móviles diferenciales usando una cámara.

## V.II OBJETIVOS ESPECÍFICOS

1. Investigar qué componentes se pueden utilizar en para formación de robots con una cámara para tener una plataforma en un mismo entorno.
2. Comprobar la viabilidad de los componentes seleccionados para poderlos aplicar a la plataforma del proyecto.
3. Corregir la imagen de la cámara según el lente de la misma y transformar la perspectiva a una vista aérea.
4. Controlar los robots de forma inalámbrica por computadora para comandar los dispositivos móviles en un mismo entorno.
5. Comandar a los robots a moverse a un punto especifico de la imagen capturada por la cámara.

## VI META DE INGENIERÍA

Para lograr una solución al problemas de formación de robots, se requieren de conocimientos de diferentes áreas de la ingeniera en conjunto. Aplicando los conocimientos de ingeniería en robótica, se obtuvo un diseño propuesto que involucra las siguientes áreas: ingeniería mecánica, electrónica, programación, sistemas de visión y teoría de control.

Para el sistema de visión se requiere de detectar *marcadores RA* ó códigos bidimensionales y para evitar poner la cámara en el techo, realizar una transformación a la imagen que nos permita trabajar en un plano bidimensional.

Para el diseño mecánico se propuso usar kits de robots diferenciales desarrollados en la universidad a los que se les va a hacer ligeras modificaciones que permitan el montado de *marcadores RA*.

Para el sistema electrónico se busca utilizar en los robots: *Puente H*, placa de desarrollo (Como Arduino Nano), módulo de comunicación, encoders para los motores. Para la computadora se utilizaría un módulo de comunicación y una placa de desarrollo para hacer de interfaz.

# VII MARCO TEÓRICO

## VII.I Robots móviles

Los robots móviles a diferencia de los robots industriales, pueden moverse libremente, solamente están limitados al terreno que le permite moverse su sistema de locomoción y su control.

Existen diversos tipos de sistemas de locomoción, existen robots terrestres con patas, con ruedas o con cadenas (orugas), robots acuáticos flotantes o sumergibles y aéreos.

Para los robots terrestres con ruedas existen varias disposiciones de ruedas, las dos mas relevantes son:

- Robot omnidireccional: Estos pueden moverse en cualquier dirección sin necesidad de reorientarse, aunque también pueden girar. Utilizan ruedas omnidireccionales con rodillos a 90 grados o a 45 grados si se trata de ruedas *suecas* (*mecanum*).
- Uniciclo: Tiene una cinemática sencilla. Su estructura consta de 2 ruedas fijas convencionales sobre el mismo eje, cada una tiene un motor y una rueda loca para estabilidad. También se le conoce como robot diferencial.

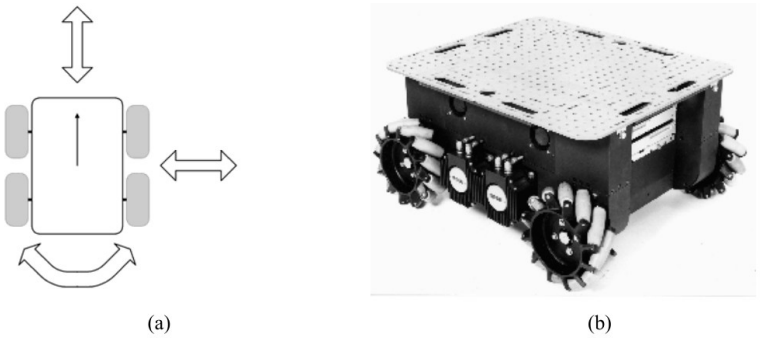


*Figura 1: Robot omnidireccional con ruedas suecas. (a) Maniobrabilidad. (b) Robot Uranus (Universidad de Michigan).*

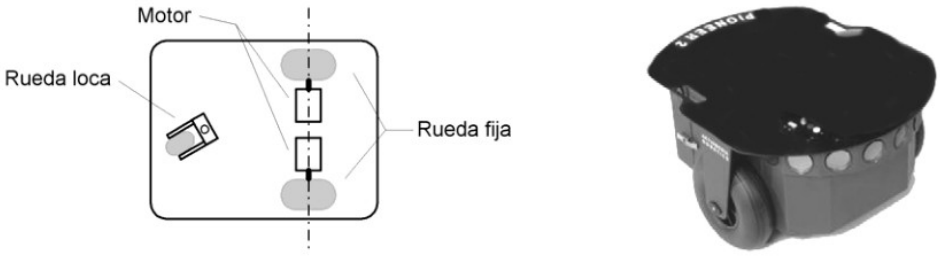


*Figura 2: Uniciclo. (a) Estructura. (b) Robot Pioneer (Active Media).*

**VII.II Formaciones multi-robot**

No existe una definición formal como tal, ya que sigue siendo un tema de investigación y está siendo explorado. Cuando se habla de formaciones multi-robot, se habla de que estos robots mantengan una estructura. Es posible mantener una estructura aun en presencia de obstáculos, usando estrategias como la subdivisión.

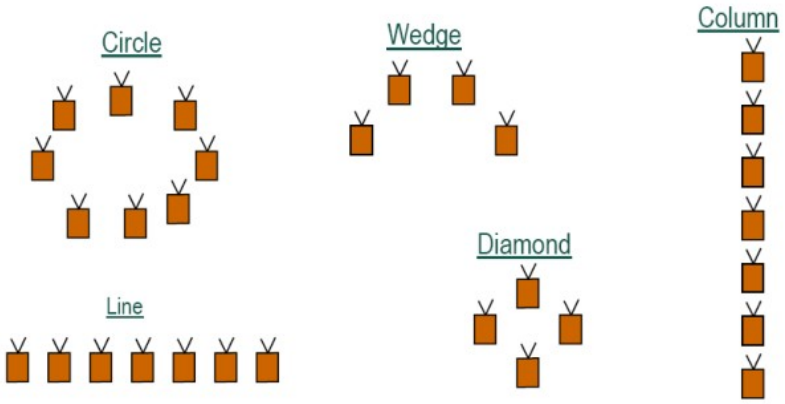


*Figura 3: Ejemplos de estructuras multi-robot (Tang, Lee y Luc, 2016).*

### VII.III OpenCV y visión

OpenCV (Open Source Computer Vision Library) es una *biblioteca* de código abierto para visión por computadora y aprendizaje automático.

La biblioteca tiene más de 2500 algoritmos optimizados. Estos algoritmos pueden utilizarse para detección y reconocimiento facial, identificar objetos, clasificar acciones humanas en video, seguimiento de movimientos de cámara, seguimiento de objetos en movimiento, producir nubes de puntos 3D por cámaras estéreo, unir imágenes para producir una imagen de alta resolución de una escena completa, eliminar los ojos rojos de fotografiás tomadas con flash, seguir los movimientos de los ojos, reconocimiento de escenario y establecer marcadores para superponerlos con realidad aumentada, etc.

Tiene interfaces de programación para C++, Python, Java y MATLAB y tiene soporte para Windows, Linux, Android y macOS.

Para este proyecto nos interesan los algoritmos para realizar corrección de lente, transformación de perspectiva y los algoritmos para obtener las matrices para hacer estos.


*Figura 4: Intro and loading Images - OpenCV with Python for Image and Video Analysis 1 por sentdex (Kinsley, 2015)*

**VII.IV ArUco**

ArUco es una *biblioteca* de código abierto para la estimación de la postura de la cámara mediante marcadores visuales cuadrados. Está escrito en C++.

Sus características principales son:

- Detección de marcadores con una sola linea de código en C++.
- Detección de varios diccionarios: ARUCO, AprilTag, ArToolKit+, ARTAG, CHILITAGS.
- Más rápido que cualquier otra biblioteca para la detección de marcadores.
- Pocas dependencias: OpenCV (>=2.4.9) y eigen3 (incluido en la biblioteca).
- Integración trivial con OpenGL y OGRE (Motor gráfico 3D).
- Multiplataforma (Windows, Linux, macOS, Android).

OpenCV incluye esta biblioteca en sus versiones oficiales.

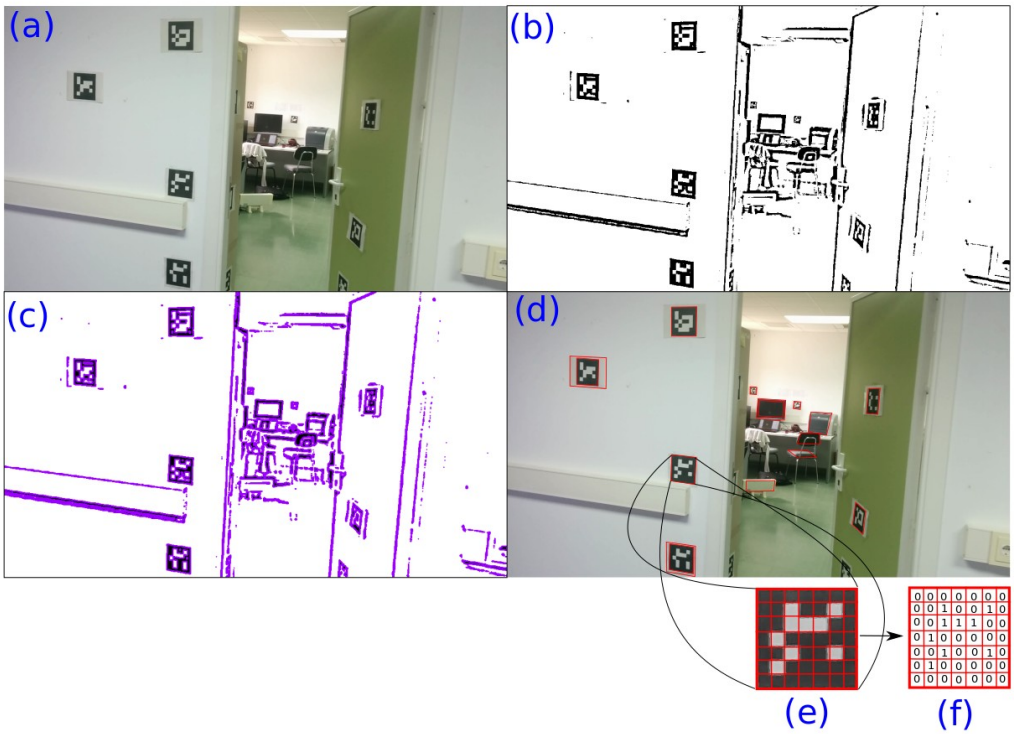


*Figura 5: Proceso de detección e identificación de ArUco (Romero-Ramirez, Muñoz-Salinas y Medina-Carnicer, 2018).*

## VIII CRONOGRAMA

*Tabla 1: Cronograma*

| Nombre | Fecha de inicio | Fecha de fin | Duración |
|---|---|---|---|
| Selección de materiales | 10/06/2022 | 15/06/2022 | 4 |
| Compra y recepción de materiales | 16/06/2022 | 28/06/2022 | 9 |
| Pruebas de comunicación entre Arduinos | 29/06/2022 | 04/07/2022 | 4 |
| Impresión de los marcadores ArUco | 05/07/2022 | 07/07/2022 | 3 |
| Detección de ArUco con OpenCV | 08/07/2022 | 13/07/2022 | 4 |
| Creación e impresión de tabla ChArUco | 14/07/2022 | 18/07/2022 | 3 |
| Creación de script de corrección de lente | 19/07/2022 | 20/07/2022 | 2 |
| Creación de script de ajuste de perspectiva | 21/07/2022 | 22/07/2022 | 2 |
| Adaptación de los marcadores a los kits de robots | 25/07/2022 | 27/07/2022 | 3 |
| Control de los robots de manera inalámbrica | 28/07/2022 | 16/08/2022 | 14 |
| Desarrollo de formulas para conversión de unidades | 17/08/2022 | 18/08/2022 | 2 |
| Movimiento de los robots con el ratón (Aplicación ejemplo) | 19/08/2022 | 23/08/2022 | 3 |

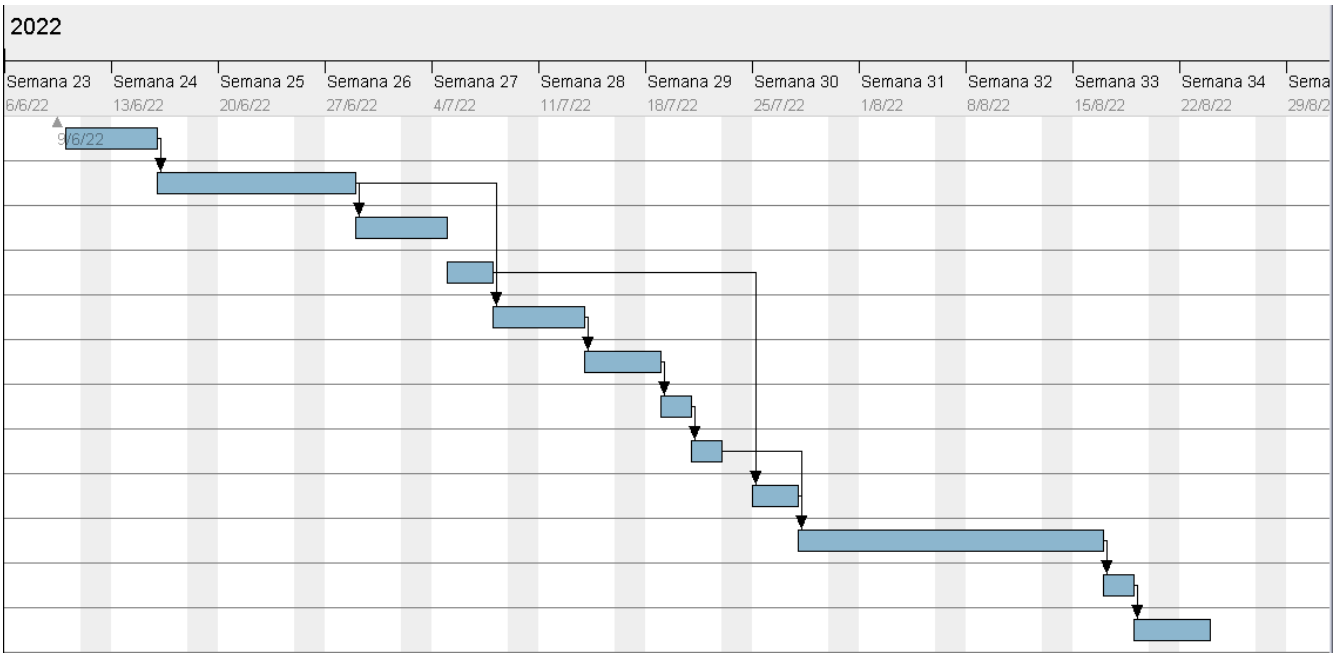


*Figura 6: Diagrama de Gantt del cronograma*

## IX MÉTODOS y/o PROCEDIMIENTOS

Para dar un mejor entendimiento de los resultados de este proyecto, la sección de procedimientos y resultados, se detallan en esta sola sección.

### IX.I Materiales

Se utilizaron tres kits de robots móvil diferenciales desarrollados en la universidad. Viene cada uno con:

- Un chasis.
- Dos motorreductores amarillos (Se desconoce la relación de reducción).
- Dos encoders rotatorios de 30 pulsos por vuelta.
- Dos ruedas anchas amarillas.
- Una rueda loca.
- Un puente H para los motores con un *CI* TB6612FNG.
- Una *PCB* para conectar un Arduino Nano y todos los componentes eléctricos y electrónicos antes mencionados.
- Tornillos, tuercas y separadores para ensamblar todo el kit.

La *PCB* incluida no se pudo utilizar debido a que no tiene las conexiones necesarias para conectar algún módulo de comunicación, se tuvo que hacer las conexiones a una protoboard. No se incluye un Arduino Nano y actualmente son costosos debido a la escasez que hay de *chips,* que en este caso es el ATmega328P, se espera que los problemas de escasez cesen pronto o que se reemplace este microcontrolador por uno más reciente ((Microchip Technology, 2022) recomienda su reemplazo por un ATmega328PB) para mantener los costos bajos en un futuro.

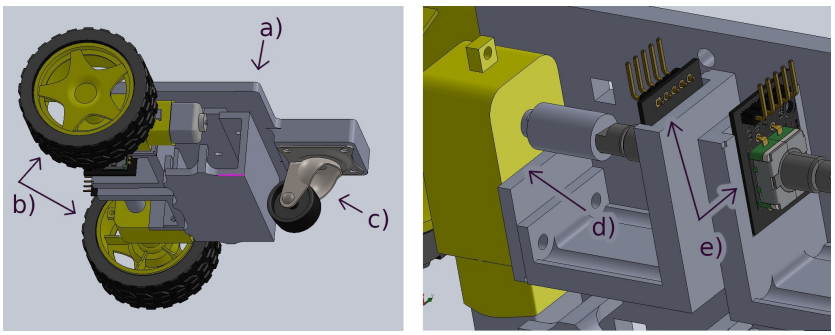


*Figura 7: Kit ensamblado: a) Chasis. b) Ruedas. c) Rueda Loca. d) Motorreductores. e) Encoders rotatorios.*

Para la comunicación se tenían cuatro opciones:

- Un módulo de comunicación Xbee con protocolo *ZigBee* e interfaz por UART y SPI, siendo este el que utilizó Uto en sus robots.
- Un módulo *Bluetooth* HC-06 de interfaz UART.
- Un módulo nRF24L01.
- Una placa de desarrollo con *WLAN* integrado como el ESP32, reemplazando al Arduino Nano.

De estos se optó por el módulo nRF24L01 porque permiten mandar y recibir paquetes de datos sin necesidad de realizar una conexión previamente, también pueden utilizar una banda ancha de hasta 2Mbps y son muy baratos. Todos los módulos y placas antes mencionados, trabajan en la banda de 2.4GHz.

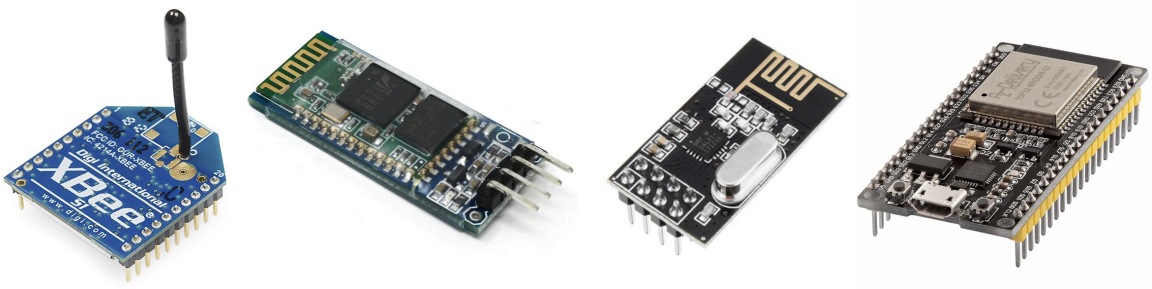

*Figura 8: Módulos de comunicación. a) Xbee b) HC-06 c) nRF24L01 d) ESP32*

Para las baterías que van a utilizar los robots se usaron celdas 18650 que fue posible adquirir a bajo costo localmente. En cada robot se utilizan 2 celdas conectadas en serie con un portapilas montado por debajo del chasis.

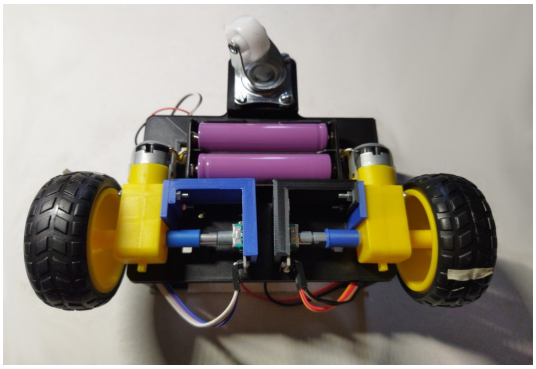

*Figura 9: Robot con celdas 18650 de color moradas.*

Para la cámara se utilizó una cámara web Logitech C920 con resolución de 1920x1080 pixeles a 30FPS, un campo visual de 78° y es posible montarlo a un trípode.

*Figura 10: Cámara web Logitech C920*

### IX.II Pruebas de comunicación

Para probar los módulos nRF24L01, se programó una simple prueba en la que se utiliza un Arduino Uno como transmisor y tres Arduino Nano como receptores. El transmisor manda paquetes con dirección "1rPON" y los datos son un solo número que va cambiando en secuencia del 1 al 3. Los receptores solo responden por su número correspondiente y encienden un LED, si reciben un paquete que no contiene su número, el LED se apaga. Todo esto fue alimentado con una batería portátil para celular.

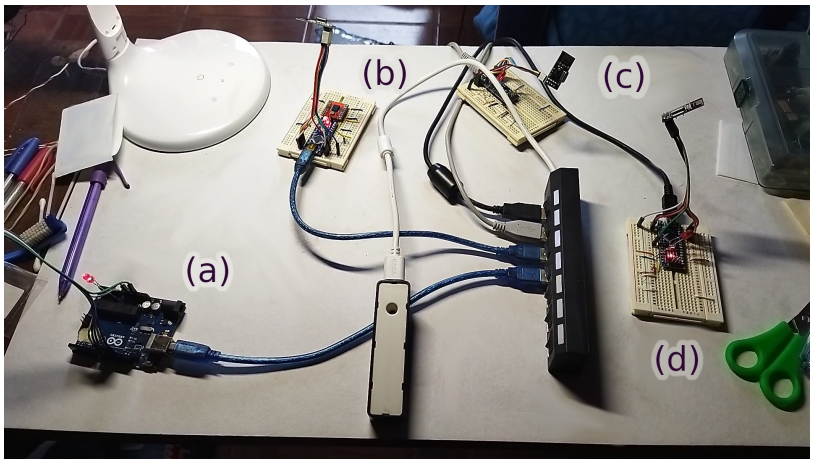


*Figura 11: Disposición de la prueba de comunicacion. (a) Transmisor. (b) Receptor 1. (c) Receptor 2. (d) Receptor 3.*

### IX.III Impresión de marcadores ArUco

Lo primero que se tiene que hacer para esto es generarlos. Se planeó que los robots utilizaran los marcadores con ID del 1 al 3, se reserva el 0 para posibles usos futuros. Para generarlos se utilizó la biblioteca de ArUco y OpenCV en Python.

*Figura 12: Marcadores ArUco del 1 al 3*

Para su impresión, se probaron dos tipos: De inyección de tinta (inkjet) y tóner. Se observo que la impresión con tinta aunque tiene menos contraste que el tóner, también refleja menos la luz. En base a esto, se optó por usar impresiones de inyección de tinta. Estos fueron impresos con dimensiones de 13 x 13 cm incluyendo un margen en blanco de 5mm en cada lado.

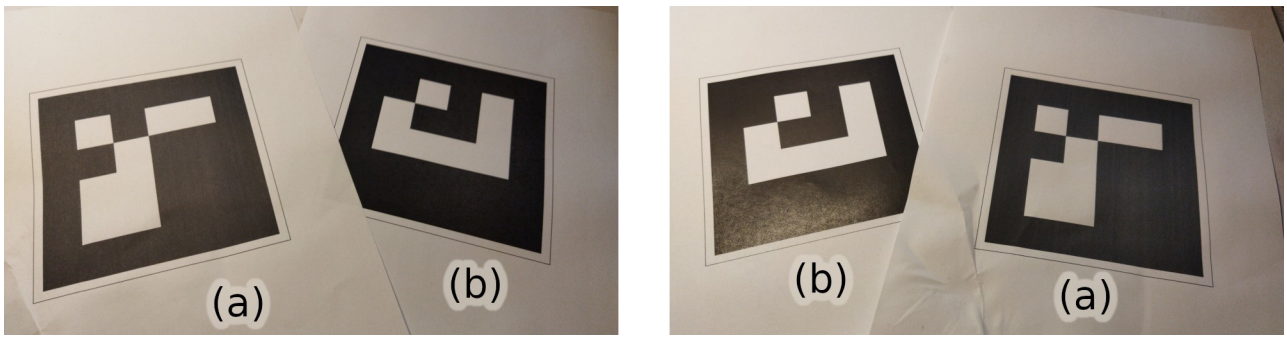


*Figura 13: Prueba de reflectividad y contraste. Las hojas se intercambian mientras que una lampara se mantiene en la misma posición. (a) Inkjet. (b) Tóner.*

Para darles más firmeza a los marcadores, estos fueron puestos y pegados sobre papel ilustración con cinta doble cara para evitar mojar las hojas con pegamento.

### IX.IV Detección de los marcadores

Para esto se utilizó un script en *Python* de ejemplo, incluido en la documentación de OpenCV. El ejemplo te encierra los marcadores en una ventana y te los marca con su ID. El script tuvo problemas para detectar los marcadores, esto debido a que se les había dejado poco margen blanco alrededor de estos. Se tuvo que volver a imprimir todos los marcadores y dejarles un margen de al menos un 1x1 bit, que en este caso, sería de 16mm en cada lado. Una vez hecho esto no hubo problemas.

### IX.V Obtención de coordenadas en un plano bidimensional

Con esto se busca que los robots puedan ser ubicados por la computadora como si estos fueran vistos desde arriba, aunque la cámara los este viendo desde un ángulo. Lo que facilita que sean ubicados con coordenadas cartesianas. Todo esto asumiendo que no existe una distorsión de lente en la imagen.

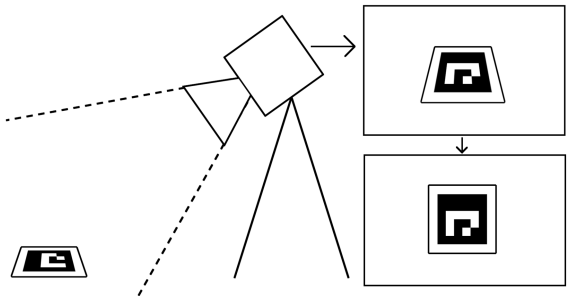

*Figura 14: Ilustración que muestra la disposición de la cámara, lo que percibe y como está con la perspectiva transformada.*

Como todos los lentes producen una distorsión en la imagen, es posible que sea necesario corregir esto si es muy significativo. OpenCV ofrece una función llamada *undistort* que nos permite, con una matriz de 4x4, corregir la imagen. Para obtener esta matriz se necesita calibrar la cámara con un tablero de cuadros o un tablero ChArUco. Es importante que antes de hacer esto, se fije el enfoque de la cámara, ya que la distorsión cambia dependiendo si el enfoque es cercano o lejano.

**IX.VI Corrección de distorsión de lente con tablero ChArUco**

Un tablero ChArUco, es un tablero de cuadros, en donde en el lugar de los cuadros blancos, se encuentran marcadores ArUco. Se generó un tablero en la página web calib.io de 11x8 cuadros, cada cuadro mide 32x32mm y cada marcador mide 25x25mm, todo esto se imprimió en una hoja de tamaño doble carta y fue puesta en papel ilustración de la misma manera que los marcadores. Como se tuvo que usar otra impresora debido al tamaño, la impresión por inkjet no fue perfecta con esta. Se tuvo que imprimir a tóner.

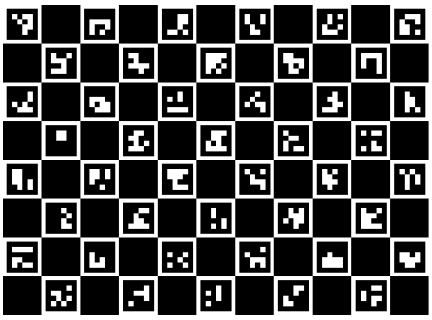

*Figura 15: Tablero ChArUco*

Con un script basado en el que está en Camera Calibration using ChArUco and Python - OpenCV Q&A Forum, se puede obtener la matriz de 4x4 necesaria para la función *undistort*. Para utilizarlo se debe mover el tablero por todo el campo de visión de la cámara, el script va guardando todos los puntos y después de 10-30 segundos, te entrega la matriz. Se hizo una modificación para que lo guardara a un archivo.

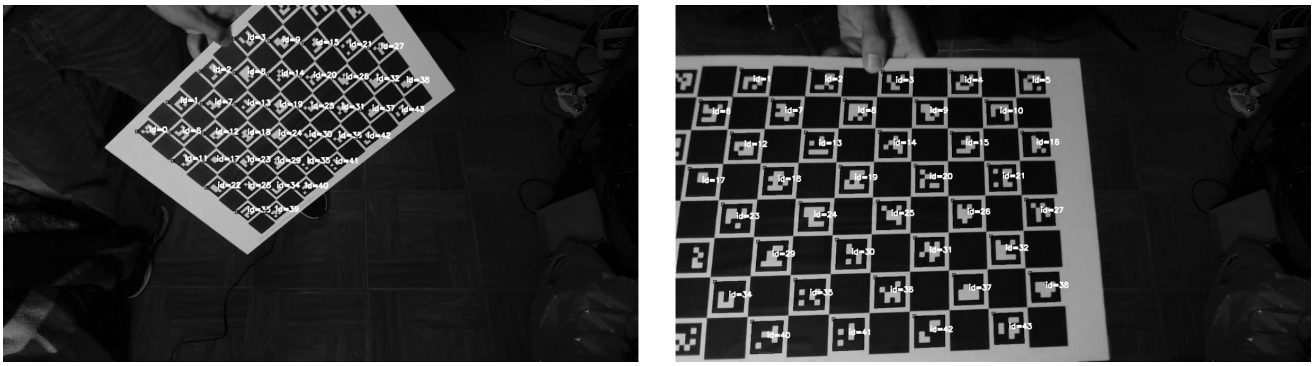

*Figura 16: Uso de script de correccion de lente*

Después de aplicar la matriz a la imagen con la función *undistort*, es difícil notar la diferencia y siendo que para aplicar la corrección se requiere de utilizar demasiado tiempo de CPU, se optó por no aplicarla.

*Figura 17: Antes y después de aplicar corrección de lente*

**IX.VII Transformación de perspectiva**

OpenCV también nos ofrece una función para alterar la perspectiva de una imagen, se llama *warpPerspective* y requiere una matriz de 3x3, además, existe la función *perspectiveTransform* que transforma vectores 2D o 3D, que puede resultar más económico en uso de CPU si no se requiere previsualizar la imagen.

Para obtener la matriz se puede usar la función *getPerspectiveTransform* que requiere 4 vértices cuadrangulares origen y 4 vértices cuadrangulares destino. Obtener estos vértices origen con la cámara y un marcador es muy impreciso, esto debido a que los vértices que nos arroja ArUco tienen ruido. Existe otra función llamada *findHomography,* que funciona con 4 o más vértices, mientras más vértices se utilicen, más tolerancia tiene al error. Para aprovechar esto se volvió a utilizar el tablero ChArUco.

El tablero se tiene que posicionar a la misma altura de donde van a ir los marcadores en los robots, aún no se le han hecho modificaciones a los robots para poder montar los marcadores. Para eso, se

diseñó una pieza, que al pegar 4 de estas en un robot, permite montar un marcador y con los tres robots, sostener el tablero a la misma altura. Se imprimieron 12 de estas piezas para todos los robots.

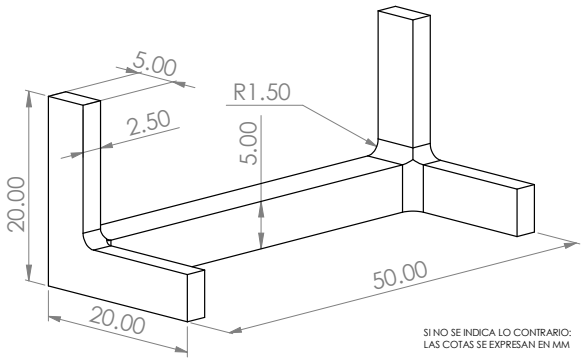


*Figura 18: Diseño de pieza para montar los marcadores*

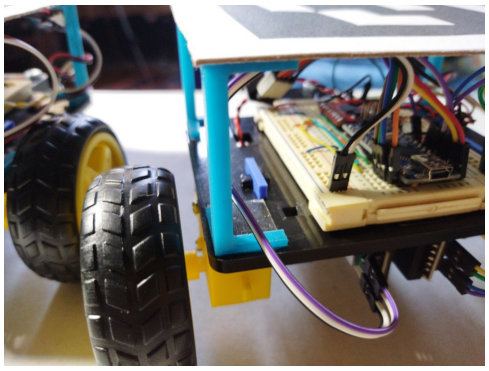

*Figura 19: Robot con las piezas y un marcador montados*

Se hizo un script para obtener la matriz 3x3 que hace lo siguiente:

```
script Calibración de perspectiva:
    genera un tablero ChArUco
    vert, ID <- detecta los marcadores dentro del tablero generado
    vertr, IDr <- ordena vert e ID en orden ascendente a ID
    vertr += offset vertical y horizontal para centrarlos

    enciende la cámara
    crea una ventana para visualizar la imagen de la cámara

    Mientras la cantidad de marcadores detectados no sea igual a los del tablero
    Hacer:
        imagen <- captura con la cámara
        vert, ID <- detecta los marcadores en la imagen
        dibuja en la imagen los marcadores detectados
        actualiza la ventana con la imagen

    vertd, IDd <- ordena vert e ID en orden ascendente a ID
    matriz <- findHomography con vertd y vertr * 0.3
    guarda la matriz a un archivo para futuras referencias

    warped <- warpPerspective a la imagen con la matriz
    crea una nueva ventana con la imagen warped
fin
```

Este script tiene la limitación que necesita detectar todos los marcadores del tablero antes de generar la matriz. Una vez que detecta todos, te crea una ventana con la transformación aplicada a la imagen de la cámara.

*Figura 20: Antes y después de aplicar la transformación de perspectiva en la imagen de la cámara.*

Se tiene que recalcular la matriz cada vez que se altera la altura a la que estaba la cámara o su ángulo de elevación, no es necesario si solo se mueve de posición o si se rota con respecto al suelo.

### IX.VIII Control de los robots por comandos inalámbricos

Para hacer esto se propuso un formato de paquete que se conforma de:

- ID del robot a controlar.
- Velocidad de los motores hacia el frente.
- Diferencial de velocidad de los motores para dirección (positivo en el derecho, negativo en el izquierdo).

Para probarlo, se hizo un simple script en *Python* que utiliza el *Arduino Uno* como interfaz para utilizar el módulo de comunicación por *UART* y simplemente hace que un robot avance a una velocidad de 100/255 por un segundo, se detenga por medio segundo y gire a velocidad 100/255 en sentido horario por un segundo.

También se hizo una prueba de latencia en donde, en un bucle infinito, se hace avanzar un robot por 33ms para luego detenerlo por 33ms. Se observa que esto lo hace de manera precisa y sin oscilaciones en el tiempo.

### IX.IX Control PID en los robots por medio de encoders

Ahora que se probó el movimiento de los robots, para controlarlos se decidió finalmente otro formato de paquete que se compone de:

- ID del robot a controlar.
- Pulsos de encoder a avanzar por el motor izquierdo.
- Pulsos de encoder a avanzar por el motor derecho.

Para la implementación de *PID* en base a la información de los encoders, se utilizaron las bibliotecas de Arduino:

- Encoder.h: Esta biblioteca permite contar los pulsos de los encoder en un contador global, incluyendo también el pulso que te indica la dirección y contiene implementaciones escritas en *ensamblador* para el *ATmega328P* (Usado en *Arduino Uno* y *Nano*) y las placas de desarrollo *Teensy*, todo esto lo hace que sea muy rápido y una mejor opción que escribirlo uno mismo.
- FastPID.h: Ofrece un controlador de PID usando operaciones de punto fijo de 32 bits, esto significa que no realiza operaciones de *punto flotante* que resultan ser muy lentas en un ATmega328P o cualquier microcontrolador sin unidad de *punto flotante*.

La razón por la se utilizan *bibliotecas* rápidas es porque el ATmega328P, solo tiene 2 patas con capacidad de interrupción y una de ellas se tiene que utilizar de manera forzosa en el modulo de comunicación, por lo que se tiene que revisar manualmente si hay pulsos. Si se llegara a tardar el microcontrolador demasiado en una sola función, se podrían perder pulsos.

Con esto, se implementó por motor, un *PID* que controla directamente la velocidad de este en base a la diferencia entre el contador del encoder y un contador final. Para obtener el contador final se le suma al contador actual el valor delta que recibimos de los paquetes.

Para ajustar el *PID* se hizo a prueba y error con ayuda de un script que hace avanzar a un robot 60 pulsos (una vuelta de rueda), se le puso a las ruedas un pedazo de cinta para verificar si se completa la vuelta. Se ajustaron los valores al punto de que no se pasara ni oscilara. Con esto preparado, la siguiente parte combinará la información de la cámara y marcadores con el control de robots, todo esto para rotar a un ángulo definido a un robot y avanzar cierta distancia.

### IX.X Pruebas de rotación y traslación

Para lograr esto, primero es necesario conocer las formulas necesarias para convertir de radianes de rotación y milímetros de traslación, a un número de pulsos que pueda interpretar el robot.

Primero se definen la cantidad de pulsos recibidos por el encoder que equivalen a una revolución:

$$\tau_p = 60\ pulsos$$

El ancho de un marcador se define por:

$$l_{mm} = 97.5\ mm$$
$$l_u = 60.0\ unidades$$

donde $l_{mm}$ es el ancho en milímetros y $l_u$ es el ancho en las unidades que nos da el script.

Y:

$$r = 33.25\,mm\,\frac{l_u}{l_{mm}}$$
$$\alpha = 82.5\,mm\,\frac{l_u}{l_{mm}}$$

donde $r$ es el radio de las ruedas y $\alpha$ es la distancia que hay entre las ruedas dividido sobre 2. $\frac{l_u}{l_{mm}}$ indica que se convirtieron sus medidas de milímetros a unidades del script.

Con esto definido se hicieron pruebas de rotación para hacer que el robot 1, rotara a $\frac{\pi}{2} rad$ con respecto al sistema de coordenadas de la imagen con la transformación de perspectiva aplicada. Para esto, primero se obtiene el ángulo actual del robot, esto se obtiene usando de referencia el lado inferior del marcador del robot y al calcular su $\arctan()$ con su pendiente.

$$\arctan\left(\frac{y_f - y_i}{x_f - x_i}\right)$$

Posteriormente, se calcula la diferencia entre el resultado de $\arctan()$ y $\frac{\pi}{2} rad$ para después convertirlo a la cantidad de pulsos que se tienen que mover en cada motor usando las formulas:

$$\frac{pulsos}{rad_{rueda}} = \frac{\tau_p}{2\pi} \quad \text{y} \quad \frac{pulsos}{rad_{robot}} = \frac{\alpha}{r}\,\frac{pulsos}{rad_{rueda}}$$

Después de haber transmitido un paquete para que el robot se reoriente en base a los pulsos calculados, el robot se mueve a la orientación deseada.

Ahora, para trasladar el robot, se usa como posición actual el punto medio del lado inferior del marcador y nuestra posición destino serán simplemente 50 unidades hacia el frente del robot y para simplificarlo, el robot está orientado hacia $y$ positivo. Para saber que tantas unidades tiene que avanzar se utiliza:

$$distancia = \sqrt{(x_f - x_i)^2 + (y_f - y_i)^2}$$

y para convertirlo a pulsos:

$$\frac{pulsos}{unidades} = \frac{\tau_p}{2\,r\,\pi}$$

Durante esta prueba, se notó que los motores no rotaban a la misma velocidad y ademas, derrapaban las ruedas.

Resolver esto requiere que ademas de la posición, controlemos la velocidad y mantengamos una aceleración máxima. Entonces hay que volver a programar los robots.

### IX.XI PID para control de velocidad

Para este control se necesita obtener la velocidad promedio para cada rueda. Lograr esto con un encoder de poca resolución no es trivial, el método que normalmente se ocuparía sería simplemente contar los pulsos que ocurrieron en un lapso de tiempo y si se hace esto con estos encoders, se tendría que tomar los pulsos en tiempos muy largos para tener una buena precisión y esto no nos daría un buen tiempo de respuesta.

Se propuso un algoritmo que involucra guardar en un arreglo de tres espacios el tiempo que transcurre entre cada pulso, esto se implementó para cada encoder de la siguiente forma:

```
función Lectura de encoder:
    contadorEncoder <- Encoder.read() de la biblioteca Encoder
    tiempoActual <- Contador de tiempo global en microsegundos

    Si el contadorEncoder cambió con respecto a contadorPrevio:
        tiempoDelta <- tiempoActual - tiempoPrevioPulso

        (Para evitar medir una velocidad muy rápida y posiblemente erronea)
        Si el tiempoDelta es mayor a 1000us:
            Si el contadorEncoder aumentó:
                arregloDelta[i] <- tiempoDelta
            Si disminuyó:
                arregloDelta[i] <- tiempoDelta negativo
            i incrementa en 1
            Si i superó el tamaño del arregloDelta:
                i se iguala a 0
        contadorPrevio <- contadorEncoder
        tiempoPrevioPulso <- tiempoActual
fin
```

Esta función se ejecuta en cada bucle del programa.

Para calcular la velocidad se hace la sumatoria de todos los valores del arreglo para luego usarlo en dividir un numero muy grande. Con esto se obtiene un valor de tipo entero con buena resolución. Su implementación es así:

```
función Calculo de velocidad:
    tiempoActual <- Contador de tiempo global en microsegundos

    (Si el motor no se ha movido en 50ms, la velocidad es cero)
    Si el tiempoActual - tiempoPrevioPulso es mayor a 50000us:
        Todos los valores en arregloDelta se igualan a cero
```

```
        velocidadPromedio <- 0
    Si es menor o igual:
        sumatoria <- suma de todos los valores en arregloDelta
        (usamos un numero muy grande (2^20 * 3) para evitar usar decimales y
        se agrega un 1 a la sumatoria para evitar la división entre cero)
        velocidadPromedio <- 3145728 / (sumatoria + 1)
fin
```

Esta función se ejecuta justo antes de calcular los PID, en nuestro caso es 60 veces por segundo.

Ahora, para el control de velocidad se utiliza un PID para obtener la velocidad deseada en el motor en base a la diferencia entre el contador actual y el deseado y con otro PID, controlamos el "acelerador" del motor, teniendo cuidado también de no hacer cambios bruscos para evitar derrapar.

La función que maneja el PID de posición, es muy breve, debido a que utilizamos la biblioteca de FastPID. Esta función está de esta forma:

```
función PID de posición:
    error <- contadorMeta - contadorEncoder

    (Esto establece un rango aceptable de error para detener el motor)
    Si el valor absoluto del error es menor a 1:
        Reinicializa las variables internas del PID
        velocidadMeta <- 0
        velocidadMotor <- 0
    Si no es así:
        velocidadMeta <- cálculo de PID por la biblioteca para la posición
fin
```

Para el PID de velocidad, primero configuramos la biblioteca para limitarlo a un rango de salida de -25 a 25 y la salida del PID la sumamos a una variable global para la velocidad final del motor, en pseudocódigo es así:

```
velocidadMáxima = 150
función PID de velocidad:
    aceleración <- cálculo de PID por la biblioteca para la velocidad usando
                   la diferencia entre la velocidadMeta y la velocidadPromedio
    velocidadMotor <- velocidadMotor + aceleración

    Si la velocidadMotor supera la velocidadMáxima: limitalo a la velocidadMáxima
    Si es menor a la velocidadMáxima negativa: limitalo a la velocidadMáxima
negativa
fin
```

Después de implementar todas estas funciones, se hicieron de nuevo las pruebas de rotación y traslación, en base a esto, se ajustaron las constantes de PID. El movimiento del robot de prueba se volvió mas suave y ambas ruedas se mantienen a la misma velocidad al moverse en linea recta. Observando esto, los demás robots fueron reprogramados.

Las constantes que se obtuvieron para los PID son:

- kp = 2.0, ki = 0.05 y kd = 0.1 para posición.
- kp = 0.04, ki = 0.1 y kd = 0.0 para velocidad.

### IX.XII Movimiento de los robots con el ratón

Para darle una aplicación interactiva a todo lo que se ha desarrollado anteriormente, se creó un script para que un usuario pueda seleccionar un robot y hacer que este se mueva a cualquier parte de la pantalla.

OpenCV tiene una función llamada *setMouseCallback()* que nos permite escuchar los eventos de ratón en una ventana controlada por la *biblioteca*, solo necesitamos darle el nombre de la ventana a escuchar y el de una función escrita para responder a estos eventos. Cuando esta función es llamada por algún evento*,* recibe como argumentos: El tipo de evento recibido (clic izquierdo, clic derecho, movimiento, etc), coordenadas *x* y *y* del cursor en la ventana y banderas (¿Se mantuvo presionado shift al momento del evento?, etc).

La función que se usó para escuchar los eventos del mouse, realiza lo siguiente:

```
función Callback de mouse:
    Si se presionó el clic izquierdo:
        esquinas <- obtén los vértices de los marcadores, ordenados por id
        Para cada id en esquinas:
            Si existen esquinas para el id actual:
                colisión <- usando pointPolygonTest de OpenCV, revisa si hay una
                            colisión entre el punto (x, y) y las esquinas del
                            marcador actual.
                Si ocurrió una colisión:
                    robotSeleccionado <- id actual
                    Retorna
        robotSeleccionado <- 0

    Si se presionó el clic derecho:
        Si el robotSeleccionado no es igual a 0:
            Crea nuevo hilo de controlRobots, incluyendo en los argumentos el
                robotSeleccionado y puntos (x, y)
            Retorna
fin
```

Para la función controlRobots, mencionada en el pseudocódigo, simplemente hace girar el robot hacía donde está el punto (x, y) y después se traslada hacia el. Se implementó así:

```
función controlRobots (id, xDestino, yDestino):
    posición, ánguloRobot <- obtén la posición y ángulo del robot con cierto id

    ánguloDestino <- atan2(posición[y] - yDestino, posición[x] - xDestino)
    ánguloDelta <- la distancia mas corta entre el ánguloRobot y ánguloDestino
    distancia <- la distancia entre el punto posición y (xDestino, yDestino)
```

```
    Rota el robot con id un ánguloDelta
    Espera 2 segundos para que termine el giro
    Traslada el robot con id a la distancia obtenida
fin
```

Todo esto, nos da la siguiente secuencia:

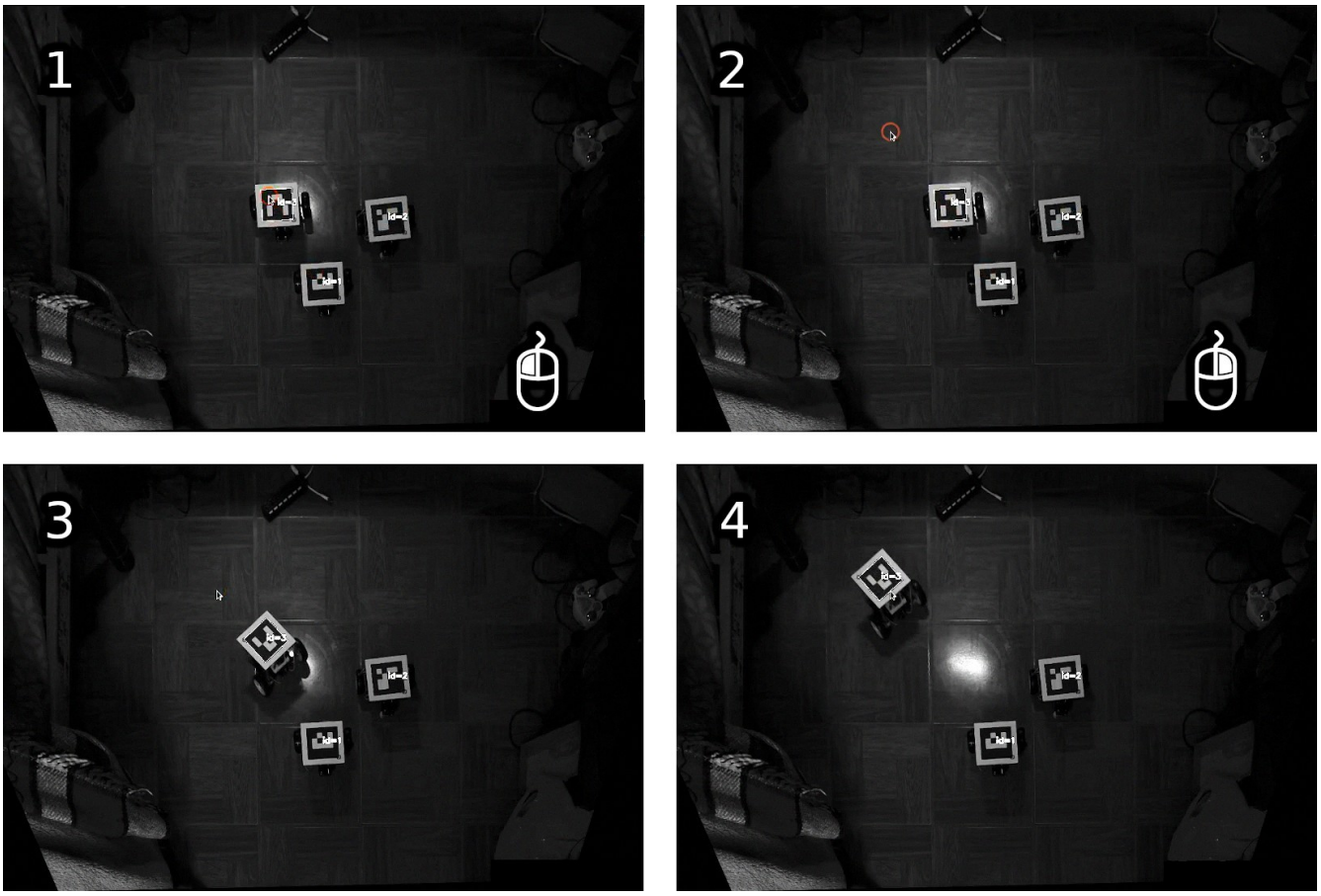


*Figura 21: Secuencia del control de robots con el ratón.*

## X ASPECTOS FINANCIEROS

No se incluye el costo de las protoboard, los jumpers ni del trípode ya que ya se contaba con ellos.

| Cantidad | Componente | Costo unitario | Costo Total |
|---:|---|---:|---:|
| 3 | Kit de robot diferencial | ~$600 | ~$1800 |
| 3 | Arduino Nano | $120 | $360 |
| 1 | Arduino Uno | $200 | $200 |
| 4 | Módulos nRF24L01 | $51 | $204 |
| 6 | Celdas de batería 18650 | $35 | $210 |
| 3 | Portapilas de 2 celdas 18650 | $35 | $105 |
| 1 | Servicio de impresión 3D para las bases de marcador | $30 | $30 |
| 1 | ½ de papel ilustración | $40 | $40 |
| 1 | Impresión de todos los marcadores | $32 | $32 |
| 1 | Cámara Logitech C920 | $1200 | $1200 |
| | | | ~$4181 |

## XI CONCLUSIONES

En este trabajo, se investigó sobre los diferentes componentes que se pueden utilizar para construir una propuesta experimental para control de robots móviles usando una cámara. Esta propuesta forma una base que puede extenderse para proyectos futuros de robots móviles.

Con los componentes seleccionados, fue posible crear experimentos con robots móviles, donde la cámara utilizada pudo lograr una interacción con los robots. Para lograr esta interacción, se utilizaron marcadores en los robots para que la cámara tuviera una referencia de sus posiciones y orientaciones; así como, el uso de módulos de radiofrecuencia para mantener una comunicación inalámbrica entre los dispositivos del sistema.

En esta etapa del proyecto, no se hizo una corrección de imagen, sin embargo, si se hizo una modificación en la perspectiva de la imagen para que los marcadores de los robots se pudieran visualizar en un plano 2D. Esto ayudó a tener una corrección visual de los diferentes marcadores de los robots, permitiendo que la cámara no necesite estar posicionada sobre los marcadores.

Los robot fueron controlados usando una técnica de control PID, mediante una arquitectura cerrada por cada robot y a través de la computadora, se le comandaba la posición deseada. Dicha posición deseada, se enviaba a partir de una posición seleccionada con el ratón de la computadora, siendo esta posición la referencia deseada de cada robot y cada PID del robot, se encargaba del accionamiento de los motores. La referencia de posición del PID, es retroalimentada usando los encoders de cada motor.

## XII RECOMENDACIONES Y TRABAJOS A FUTURO

Algunas recomendaciones de mejora para este trabajo, se mencionan a continuación:

- Se sugieren modificaciones en el diseño de los robots, especialmente en el tipo de encoders usados, ya que estos se desgastaban muy rápidamente. Una propuesta sería utilizar encoders ópticos, aunque para estos requieran cambios en el programa debido a que no toman en cuenta la dirección.
- Se recomienda realizar otra selección de cámara, a una de menor costo, con resolución de al menos 720p.
- Se sugiere mejorar el script para obtener la matriz de transformación, para que este no requiera detectar todos los marcadores y con esto, funcione en entornos de menor luz y a mayor distancia.

Como trabajo futuro, se propone extender esta propuesta hasta lograr crear y mantener una estructura en formación para los robots móviles.

## XIII REFERENCIAS BIBLIOGRÁFICAS


Alonso-Mora, J., Baker, S. y Rus, D. (2017) "Multi-robot formation control and object transport in dynamic environments via constrained optimization", *International Journal of Robotics Research*, 36(9), pp. 1000–1021. Disponible en: https://doi.org/10.1177/0278364917719333.

Kinsley, H. (2015) "Intro and loading Images - OpenCV with Python for Image and Video Analysis 1 - YouTube". Disponible en: https://www.youtube.com/watch?v=Z78zbnLlPUA&.

Microchip Technology (2022) *ATmega328P*. Disponible en: https://web.archive.org/web/20221006162927/https://www.microchip.com/en-us/product/ATmega328P.

Romero-Ramirez, F.J., Muñoz-Salinas, R. y Medina-Carnicer, R. (2018) "Speeded up detection of squared fiducial markers", *Image and Vision Computing*, 76(June), pp. 38–47. Disponible en: https://doi.org/10.1016/j.imavis.2018.05.004.

Takano, G., Obayashi, M. y Uto, K. (2015) "Path planning for autonomous car to avoid moving obstacles by steering using tangent-arc-tangent-arc-tangent model", *2015 IEEE Conference on Control and Applications, CCA 2015 - Proceedings*, pp. 1702–1709. Disponible en: https://doi.org/10.1109/CCA.2015.7320855.

Tang, D., Lee, I. y Luc, R. (2016) "Multi-Robot Formation". Disponible en: https://www.cpp.edu/~ftang/courses/CS599-DI/notes/formation.pdf.

Tosa Electric Inc. (2009) *18cm OmniKit*. Disponible en: http://web.archive.org/web/20091002234637/http://www.tosadenshi.co.jp/blog/18cm.html.

Uto, K. (2011) "How To Make Multi-Robot Formation Control System". Disponible en: https://www.slideshare.net/utotch/how-to-make-multirobots-formation-control-system.

## XIV ANEXOS

Todo el código utilizado para este proyecto se puede encontrar en https://github.com/ponjadito23 y en el propio disco.

## XV GLOSARIO

- Arduino Nano y Uno: Placas de desarrollo muy populares, desarrolladas por la compañía Arduino.
- Biblioteca (programación): Es una colección de funciones y algoritmos, re-utilizables para múltiples programas.
- Bluetooth: Es un estándar inalámbrico de bajo alcance y baja potencia, para intercambio de datos entre dispositivos y para redes de área personal (PAN).
- ChArUco: Acrónimo de checkerboard (Tablero de cuadros) y ArUco. Utilizado para calibración y estimación de pose de una cámara.
- Chip: Otra forma de llamar a los circuitos integrados. Estos permiten integrar una alta cantidad de transistores configurados, en un pequeño paquete.
- Cámara web: Cámara diseñada para conectarse directamente a una computadora. Permiten capturar video e imágenes para almacenarlas o distribuirlas a una red.
- Encoder: Permite codificar en pulsos o voltaje, el movimiento o posición de un motor.
- Ensamblador (Lenguaje de programación): Es cualquier lenguaje de bajo nivel, correspondiente unicamente a una arquitectura de instrucciones de procesador (x86, ARM, MIPS, etc.) en donde se declara directamente, que instrucciones debe ejecutar un procesador.
- PID (Control): Es un mecanismo de control de lazo cerrado (usando retroalimentación) ampliamente utilizado en sistemas de control industriales. Utiliza tres términos para control que son: Proporcional, integral, derivativo.
- Puente H: Circuito que permite controlar un motor en ambos sentidos, esto mediante la inversión del voltaje de entrada. Su nombre viene del parecido a la letra H con su diseño en diagrama.
- Punto flotante: Formato de número binario que permite almacenar un número en fracción y exponente. Como manejan un sistema binario, su representación de números decimales solo es aproximada. No debe ser utilizado para sistemas financieros porque acumulan errores.
- Python: Lenguaje de programación de alto nivel que no requiere compilación previa, su filosofía de diseño enfatiza la legibilidad del código. Aunque no es muy rápido, es muy sencillo

de utilizar y permite hacer esbozos de un programa antes de pasarlo a un lenguaje de más bajo nivel.

- WLAN: Conocido popularmente como Wi-Fi. Es un estándar de red inalámbrica para computadoras que permite enlazar dos o más dispositivos a una LAN (Red de área local) y si existe una puerta de enlace, a la Internet.
- Xbee: Estandar de comunicaciones diseñado para conexiones punto a punto o estrella. Ampliamente usado en la robótica y automatización del hogar.

## XVI ABREVIATURAS

CI: Circuito integrado.

CPU: Unidad de procesamiento central.

ID: Número de identificación.

PCB: Placa de circuito impreso.

PID: Control Proporcional-Integral-Derivativo.

RA: Realidad aumentada.

UART: Transmisor-Receptor Asíncrono Universal.

WLAN: Red de área local inalámbrica.