Robot móvil “Azcatl”

Responsables

Medrano Albarrán Gerardo

Merino Texta Rogelio de Jesús

Fecha de realización: 27 de junio de 2019

Descripción

Azcatl es un robot móvil de 6 motores, montados sobre una plataforma metálica. C uenta con un microcontrolador Arduino Mega que actúa como controlador de los m otores y adquisidor de los datos provenientes de los sensores, además cuenta con una microcomputadora Raspberry Pi 3 +, que cuenta como unidad de procesamie nto.

En cuanto a los sensores, cuenta con un módulo de 8 fotoresistencias que le sirve n para poder detectar fuentes luminosas, tiene 2 sensores de distancia digitales pa ra prevenir colisiones con objetos muy cercanos, y un sensor de laser con campo de detección de 360°. El módulo de las 8 fotoresistencia se puede desconectar par a conectar un módulo de visión artifical con el que el robot adquiere la funcionalida d de detectar colores e ir en dirección a la posición del color determinado.

Esta integrado con ayuda del Middleware ROS (Robot Operating System) versión Kinetic que corre sobre la microcomputadora Raspberry Pi 3 +.

Componentes

- [1] Arduino MEGA 2560
- [1] Eliminador de 12[V]B a 5[A] y regulador step-down 300W 9A para reducir el voltaje a 7.4 [V] o en su caso b]ateria Li-Po 7.4 [V] 5000 [mA]
- [1] Cable USB Arduino
- [1] Chasis metalico de dos pisos.
- [6] Clemas de plastico
- [1] Conector de tipo T para bateria
- [6] Coples metalicos
- [1] Memoria Flash microSD 16 [GB].
- [6] Motores Pololu, 100:1 Metal Gearmotor 37Dx73L [mm] with 64 CPR Encoder.
- [1] Pololu Dual VNH5019 Motor Driver Shield
- [6] Ruedas de plastico
- [1] Microcomputadora Raspberry Pi 3
- [1] Torre de alumino y acrilico para el soporte de los fotoresistores
- [1] Cámara Logitech c525
- [1] Sensor laser Hokuyo
- [1] Cable USB Hokuyo

Modificaciones

1. Se modifico el alambrado de los motores con la pila Li-Po, debido a que la polari dad de dos de los motores estaba al revés.

2. Se cambiaron las conexiones entre el Arduino Mega y los encoders de un motor , debido a que este motor ocupada el pin de intrrrupción que debía ser de otro mot or.

3. Se rediseñó el control PID, debido a que sólo se consideraban las lecturas de 2 encorders correspondientes a 2 motores; en el control actual se considera un prom edio para cada uno de los lados del robot, es decir:

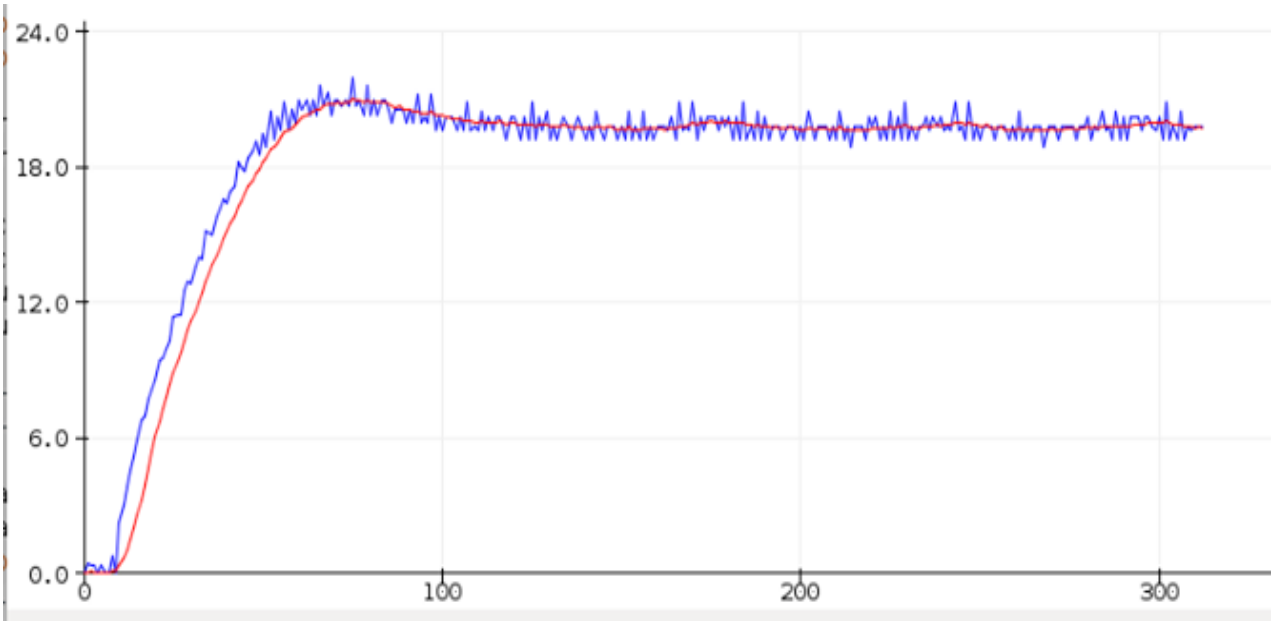
Lado derecho del robot-> (Motor2+Motor4+Motor6)/3

Lado izquierdo del robot-> (Motor1+Motor3+Motor5)/3

Esto ayuda a la eliminación del ruido y a suavizar el comportamiento.

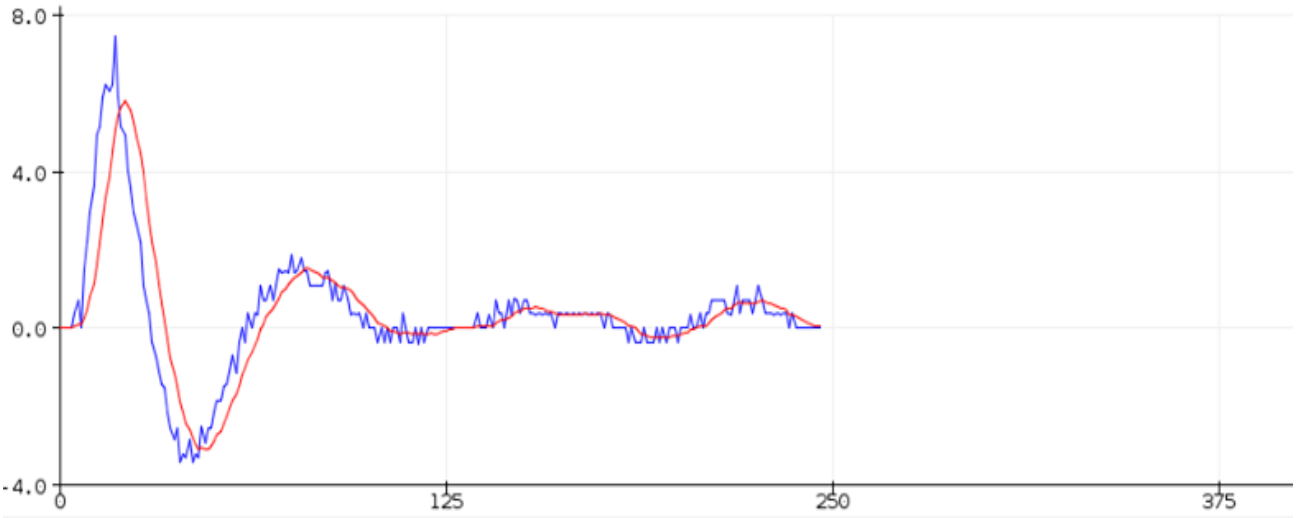
Además de una modificación en el tiempo de muestreo, con base al teorema del m uestreo de Nyquist-Shannon.

Al final se logró un comportamiento del sistema ligeramente subamortiguado, com o se puede observar en la gráfica:



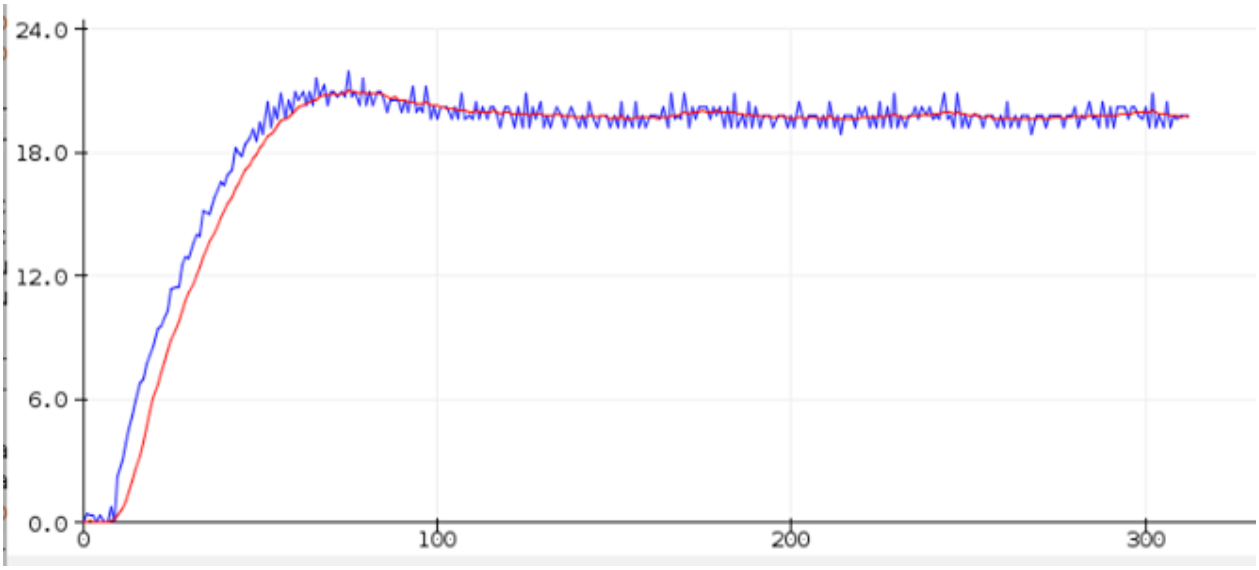
De color azul se observa el comportamiento ya con el promedio de los encoders, s in embargo aún tiene bastantes fluctuaciones por lo que se agregó un filtro inercial (curva roja) para reducir las fluctuaciones por los ruidos y a suavizar aún más la c urva, esto en contraparte significa un pequeño retraso en la respuesta debido al pr ocesamiento del filtro antes de que entregue la salida, sin embargo mejora el rendi miento en general.

Con el control pudimos lograr que el error entre las velocidades derecha e izquiera tendiera a 0 en menos de 200 [ms], además a la respuesta del error también se le aplicó un filtro inercial (curva roja) para reducir los ruidos.

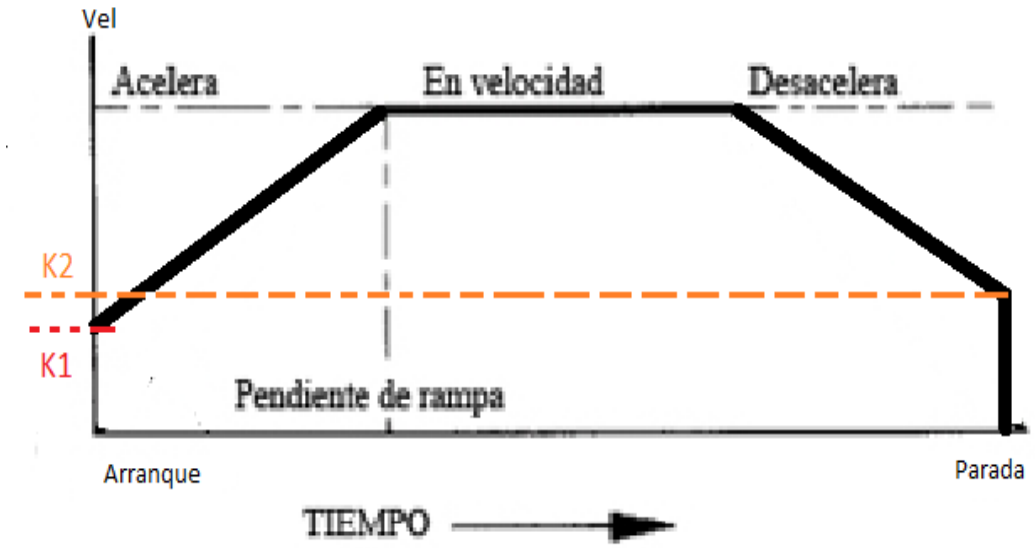


Aportes

1. Se implementó un filtro inercial (ya comentado anteriormente) para reducir el rui do en las lecturas de los encoders, es importante recalcar que entre mayor sea el t amaño de muestreo, mayor será el retardo en la respuesta.



2. Se agregó un perfil de velocidad trapezoidal modificado: Con un offset en el arra nque (K1) y otro en la parada (K2), esto para contrarrestar el efecto del efecto iner cial del sistema debido a que a determinado voltaje ya no se tiene la suficiente pot encia para mover los motores, aunque el voltaje no sea 0; en este caso los offsets son iguales, aunque pueden ser modificados.



El perfil trapezoidal consta de 3 partes, la primera describe una pendiente positiva que tiene la siguiente ecuación:

$$v = k + \frac{3 * (vm - k) * |enc[0] - posIzq0|}{\Delta}$$

La segunda parte, describe una velocidad constante, por lo que su ecuación es la siguiente:

$$v = vm$$

Por último, se describe una pendiente negativa, cuya ecuación es la siguiente:

$$v = vm - \frac{3 * (vm - k) * |enc[0] - posIzq0| - 2/3}{\Delta}$$

Donde:

v = velocidad actual

vm = velocidad máxima

k = offset del perfil trapezoidal

enc[0] = número de pulsos promedio recibido de los encodes de un lado del motor

Δ = posIzqFin – posIzq0 (para el giro)

posIzq0 = número de pulsos del encoder al momento de iniciar el movimiento

posIzqFin = posIzq0 + ((D*Pi*angulo)/(CII*360.0) (para el giro)

angulo = ángulo deseado

D = Diámetro de las llantas

Δ = av/Cll (para el avance)

posIzqFin = posIzq0 + Δ (para el avance)

av = avance deseado

Cll = Circunferencia de la llanta

Cuando se va de reversa se sigue la misma lógica de dividir el perfil en tres partes, y las ecuaciónes cambian ligeramente, son las siguientes ecuaciones respectivam ente:

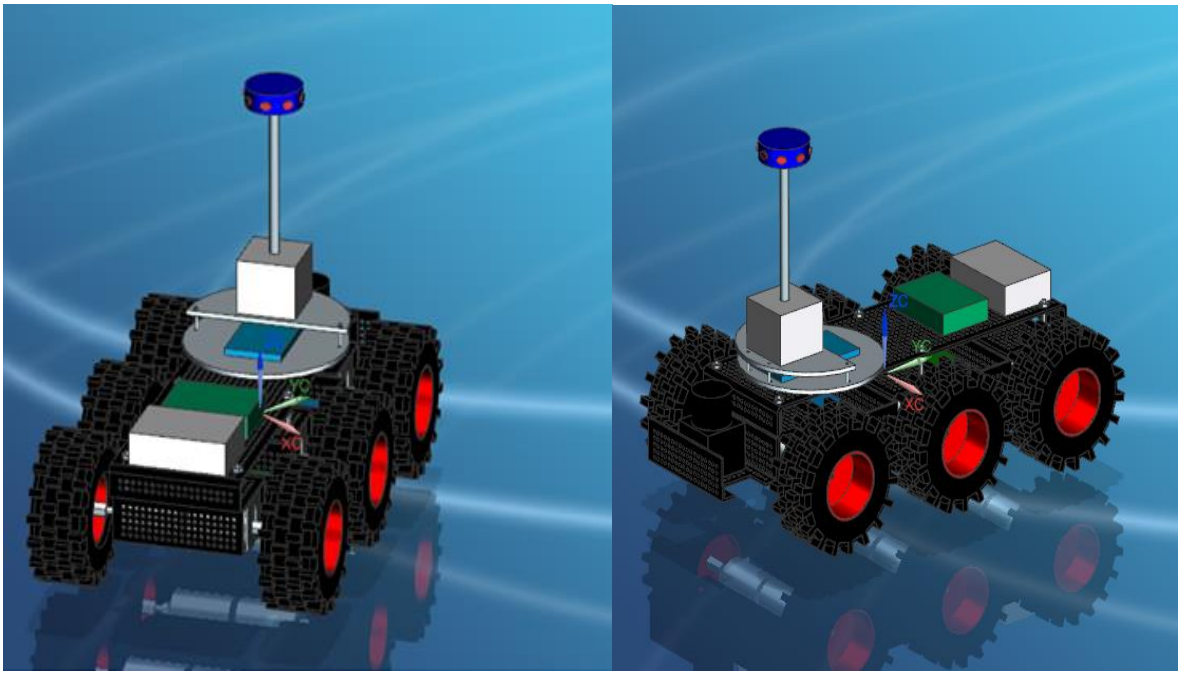
$$v = -k - \frac{3 * (vm - k) * (-1) * |enc[0] - posIzq0|}{\Delta}$$

$$v = -vm$$

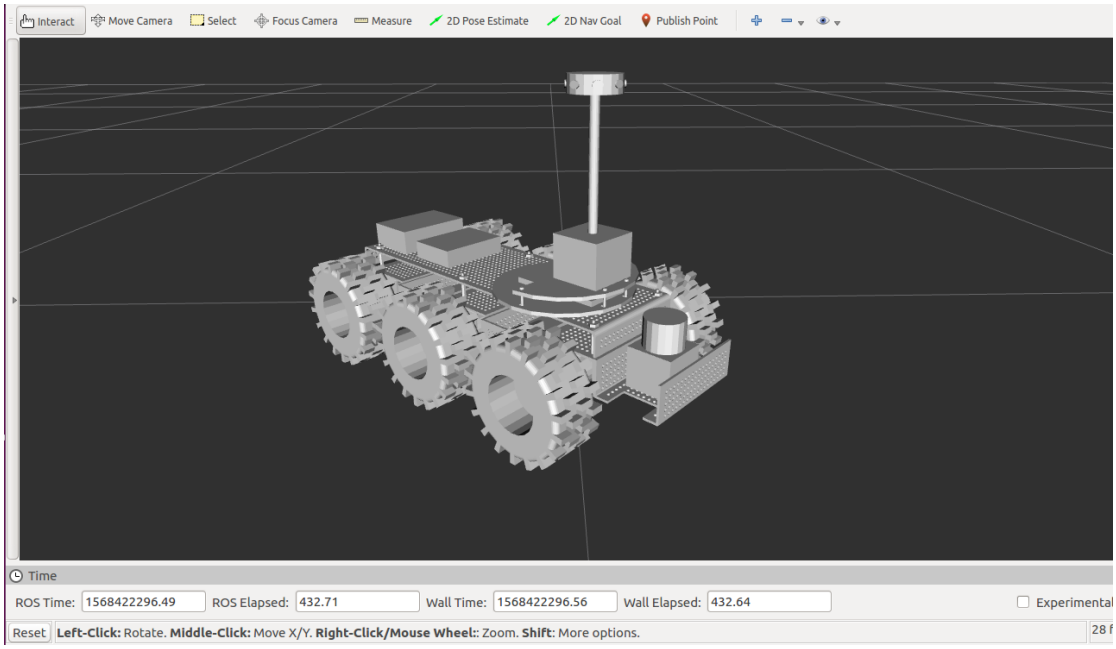
$$v = -k - \frac{3 * (vm - k) * (enc[0] - posIzq0)}{-\Delta}$$

En cuanto al comportamiento de la velocidad en los giros, es igual que el comporta miiento que presenta en el avance.

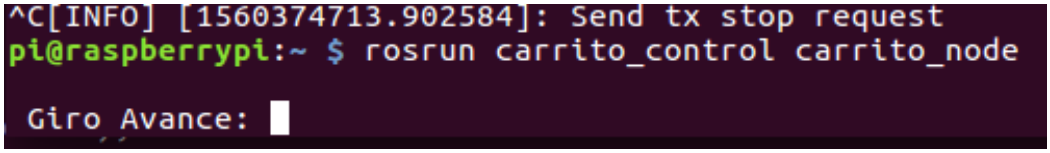
3. Se diseñó el modelo CAD del robot; es importante recalcar que se debe tener c uidado con el formato del archivo STL que es exporta, ya que debe ser tipo Binario .



4. Se implementó la odometría con la visualización del modelo virtual en Rviz

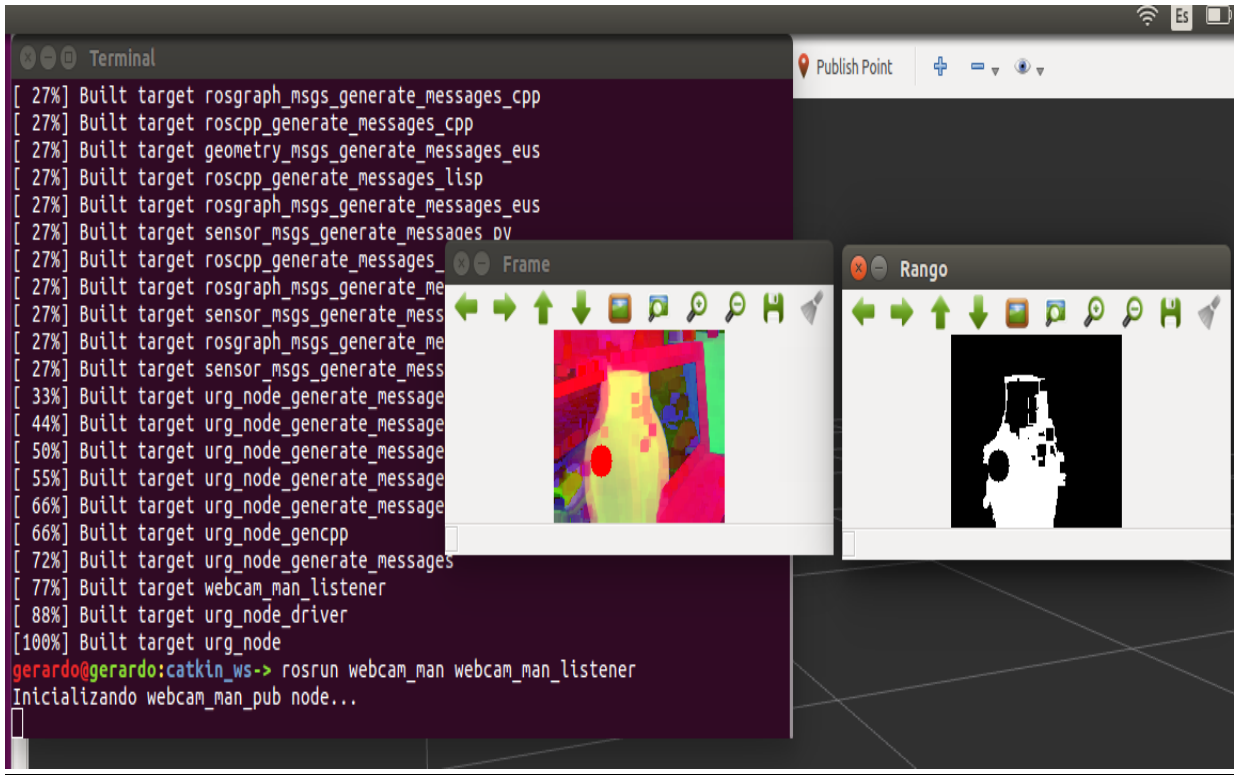


5. Se realizó un nodo de ROS, de modo que ya es posible, mandarle comandos de sde la terminal del tipo (Gira, Avanza), donde la variable Gira está expresada en gr ados, y la variable Avanza está expresada en centímetros (esta es la forma manua l, para activar la forma manual se deben descomentar las líneas 52, 54, 61 y 62 de l archivo carrito_node.cpp)

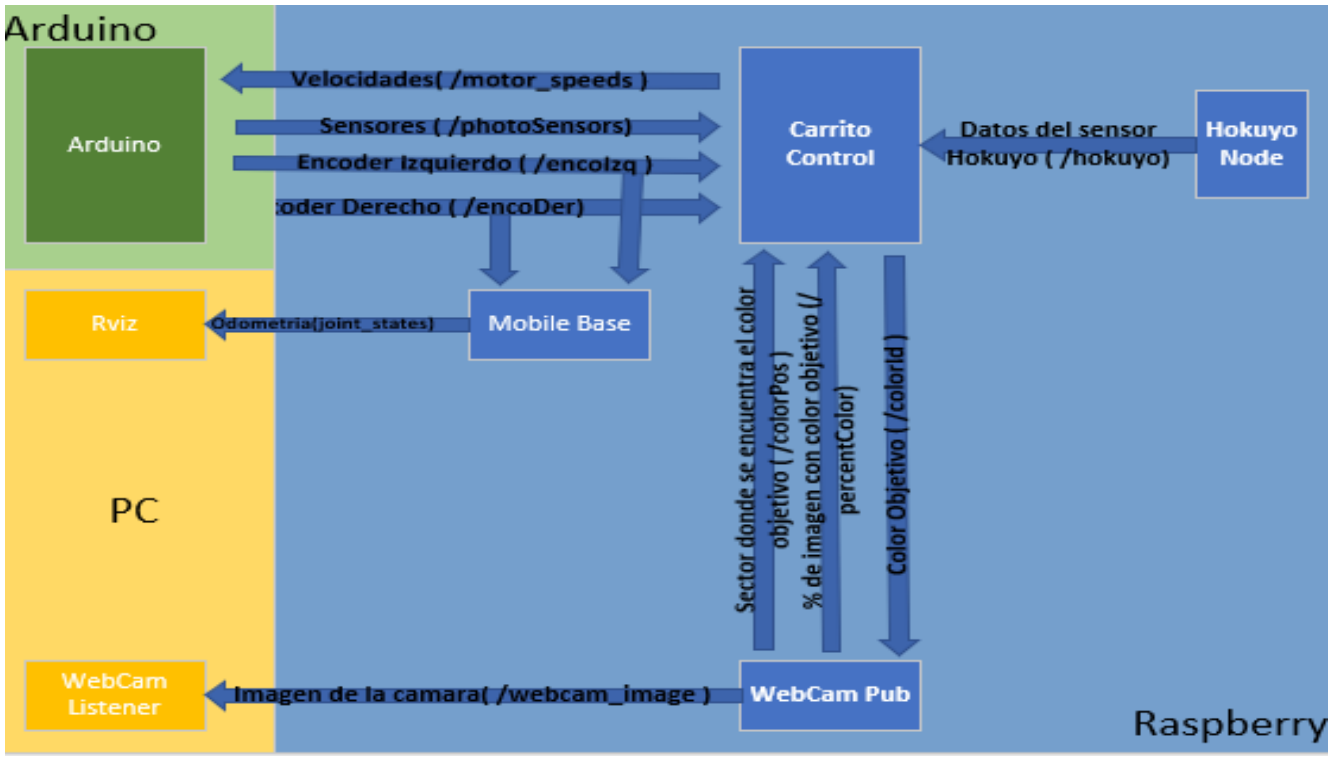


6. Se implementó el nodo del sensor laser Hokuyo con el que se realizan las medi ciones para detectar obstáculos.

7. Se implementó el nodo de visión artificial con el que se hace la detección de col ores para que el robot pueda ir en dirección a la posición de determinado color.



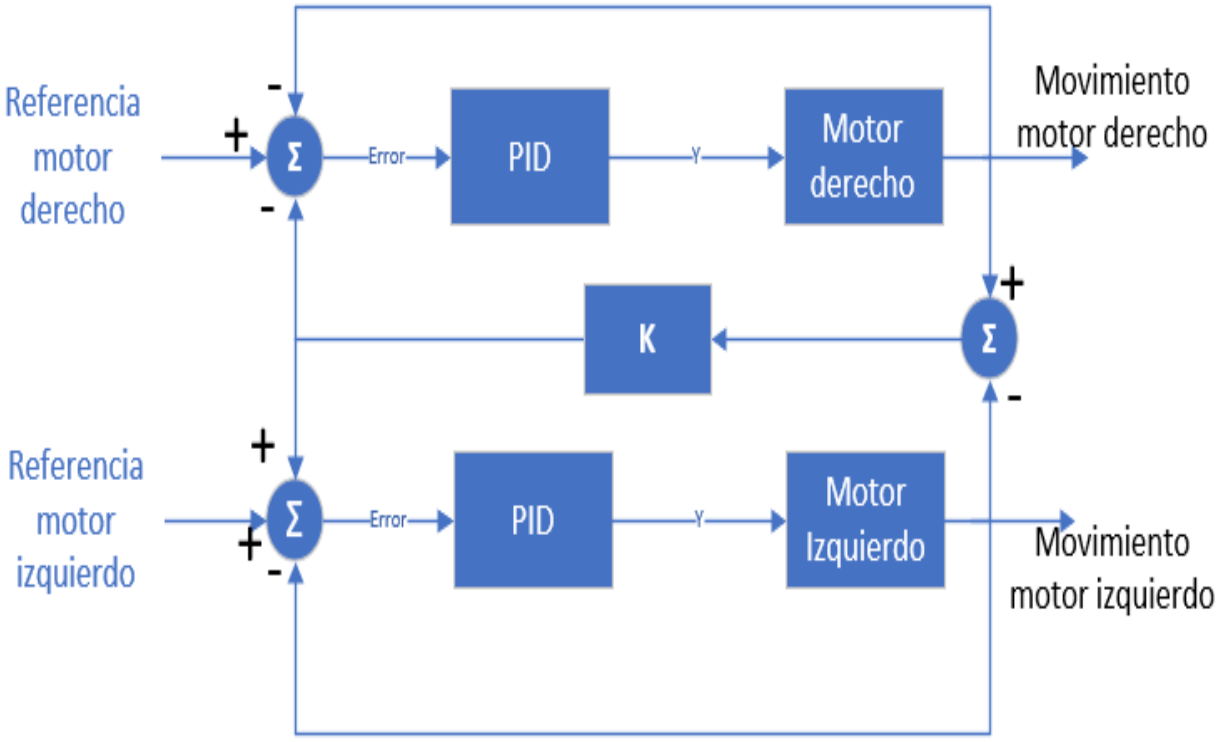
8. En general, se realizó una arquitectura de comunicación basada en nodos de R OS publicadores y subscriptores, la arquitectura se muestra en la siguiente imagen :



En la imagen anterior se muestra como es la interacción entre los diferentes nodos de ROS.

En el diagrama se representan 3 diferentes dispositivos que tienen uno o más nod os de ROS corriendo.

El primero es un Arduino Mega (mostrado en verde) este tiene un único nodo que es el encargado de controlar los motores y recibir las señales de los sensores (8 fo to resistencias). Los motores tienen un control PID independiente cada uno y a su vez tienen un control proporcional que los une para que la velocidad de los motore s sea mas uniforme. El controlador utilizado tiene el esquema de control mostrado en la siguiente figura:



Las referencias del controlador son recibidas por el tópico de ros “/motor_speeds” que vienen de un nodo montado en la Raspberry. Ademas de recibir las velocidad es que deben seguir los motores también publica el conteo total de los pulsos pro medio de los motores derecho e izquierdo.

La zona marcado en azul en la figura de la aquitectura corresponde a los nodos qu e corren directamente sobre la raspberry.

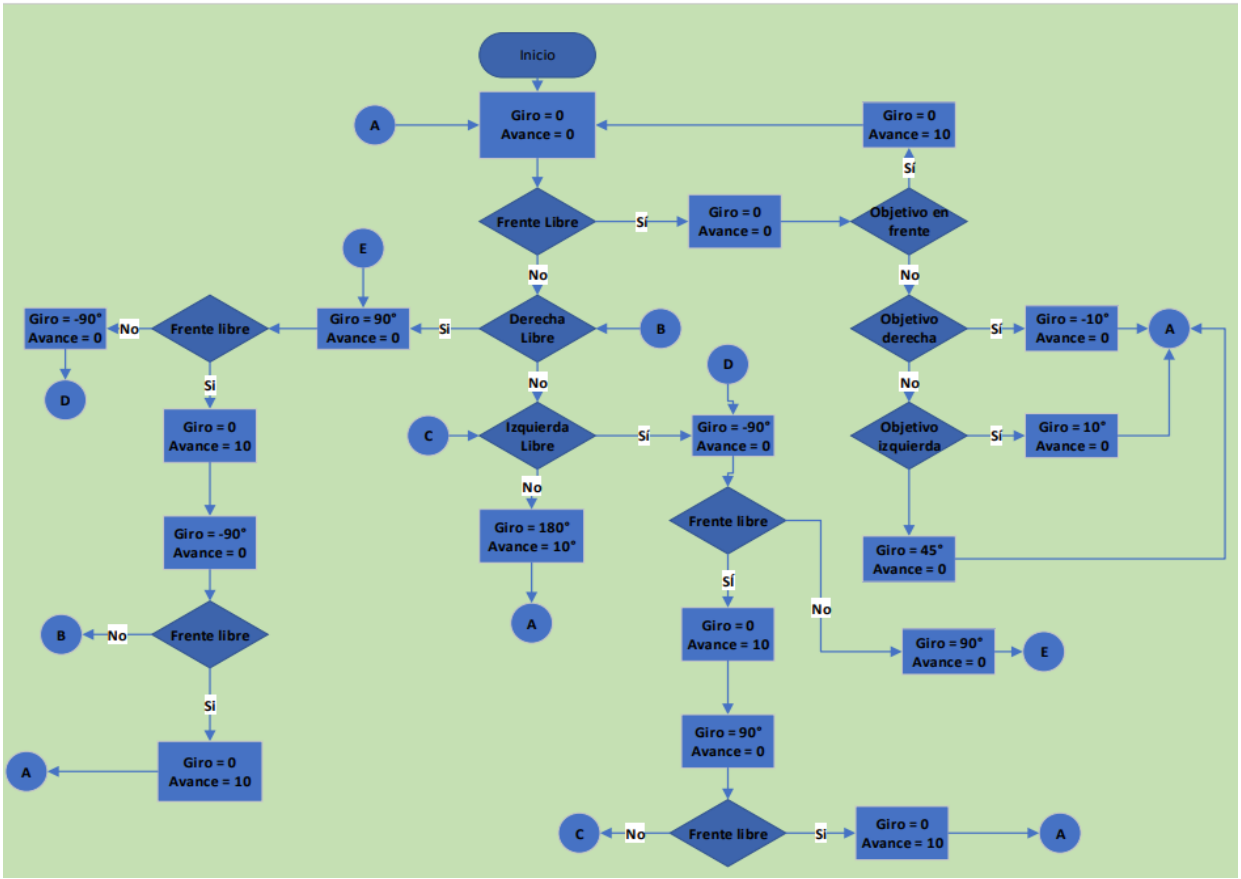
El nodo de Hokuyo con el nombre “Hokuyo node” es el encargado de recibir por un puerto USB de la raspberry la información del sensor Hokuyo y procesar la inform ación para tener distancias asociadas a un ángulo en forma de un arreglo. Este no

do (es un nodo de ROS en lenguaje de Python) publica la información procesada d el Hokuyo.

El nodo de WebCam Pub es el encargado de recibir la imagen de la cámara web y de procesar la información para decidir si el objetivo se encuentra en el rango de visión. Una vez el nodo procesa la información, éste publica el sector donde se en cuentra el objetivo(sectores: Izquierda, Derecha y Centro) en el tópico "/colorPos". Adicionalmente a esto publica un numero entre cero y uno que representa el porce ntaje que el ocolor objetivo cubre la imagen de la cámara, esto para aproximar que tan lejos esta el objetivo del robot.

Uno de los nodos más importantes es el encargado de comunicar y procesar la inf ormación de los demás nodos. El nodo carrito control es el encargado de hacer est o. Él recibe la información procesada de la cámara, la información del sensor hok uyo (la depura), utiliza la información de los encoders para moverse lo que la maq uina de estados decida. Es por esto que este es posiblemente el nodo mas import ante pues es el que decide que acción tomar y da la información al arduino del per fil de velocidad que este debe seguir.

9. Como se mencionó anteriormente, se diseñó una máquina de estados para evit ar obstáculos y seguir un objetivo de color, cuyo diagrama es el siguiente:



El nodo de “Mobile Base” toma la información de los sensores y aplica transformac iones para poder representar la posición y orientación del robot en función de un p unto de partida.

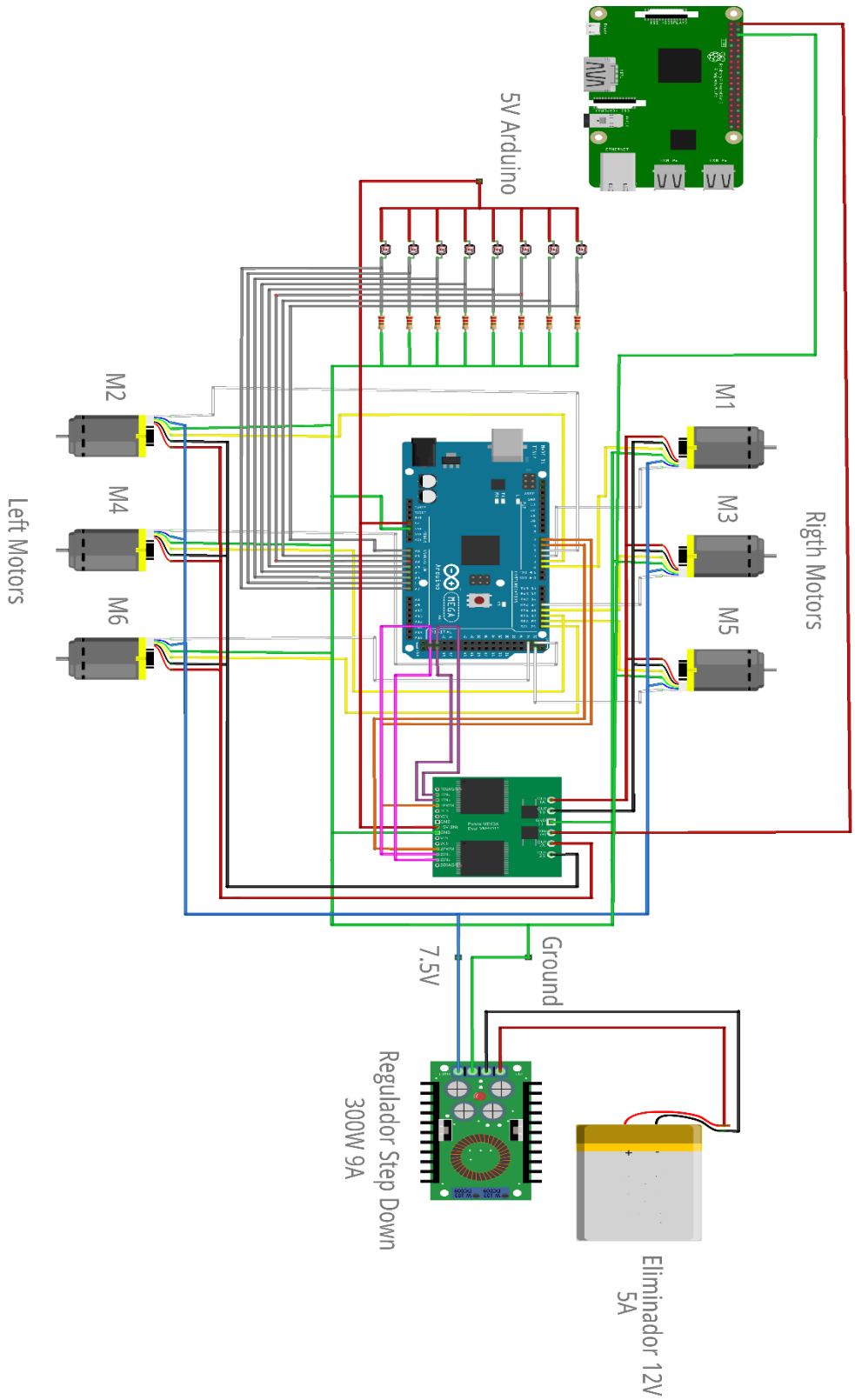
Por otro lado en la computadora corren 2 nodos diferentes, estos son:

WebCam Listener que su única función es de apoyo visual para ratificar que el nod o “WebCam Pub” de Raspberry está funcionando correctamente.

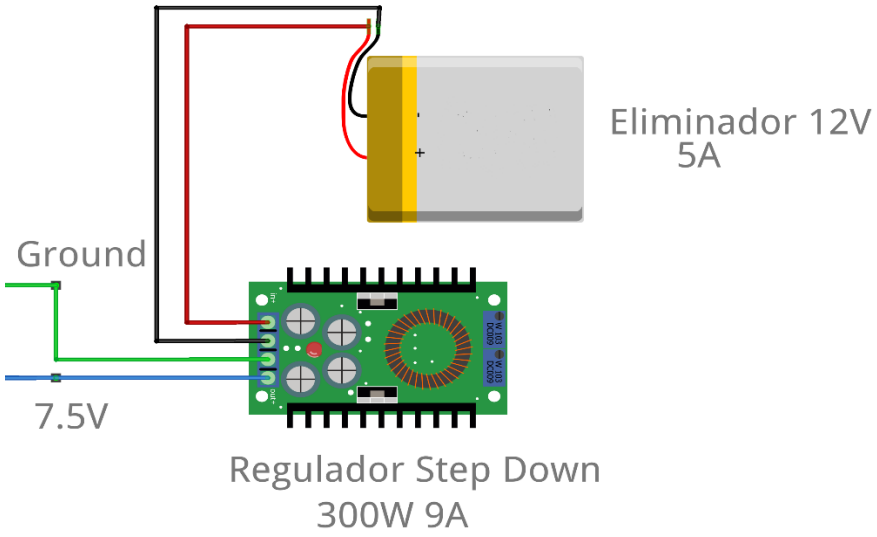
El nodo que corre la interfaz grafica de Rviz es la encargada de mostrar gráficame nte la posición del robot en un entorno virtual.

Diagramas electrónicos

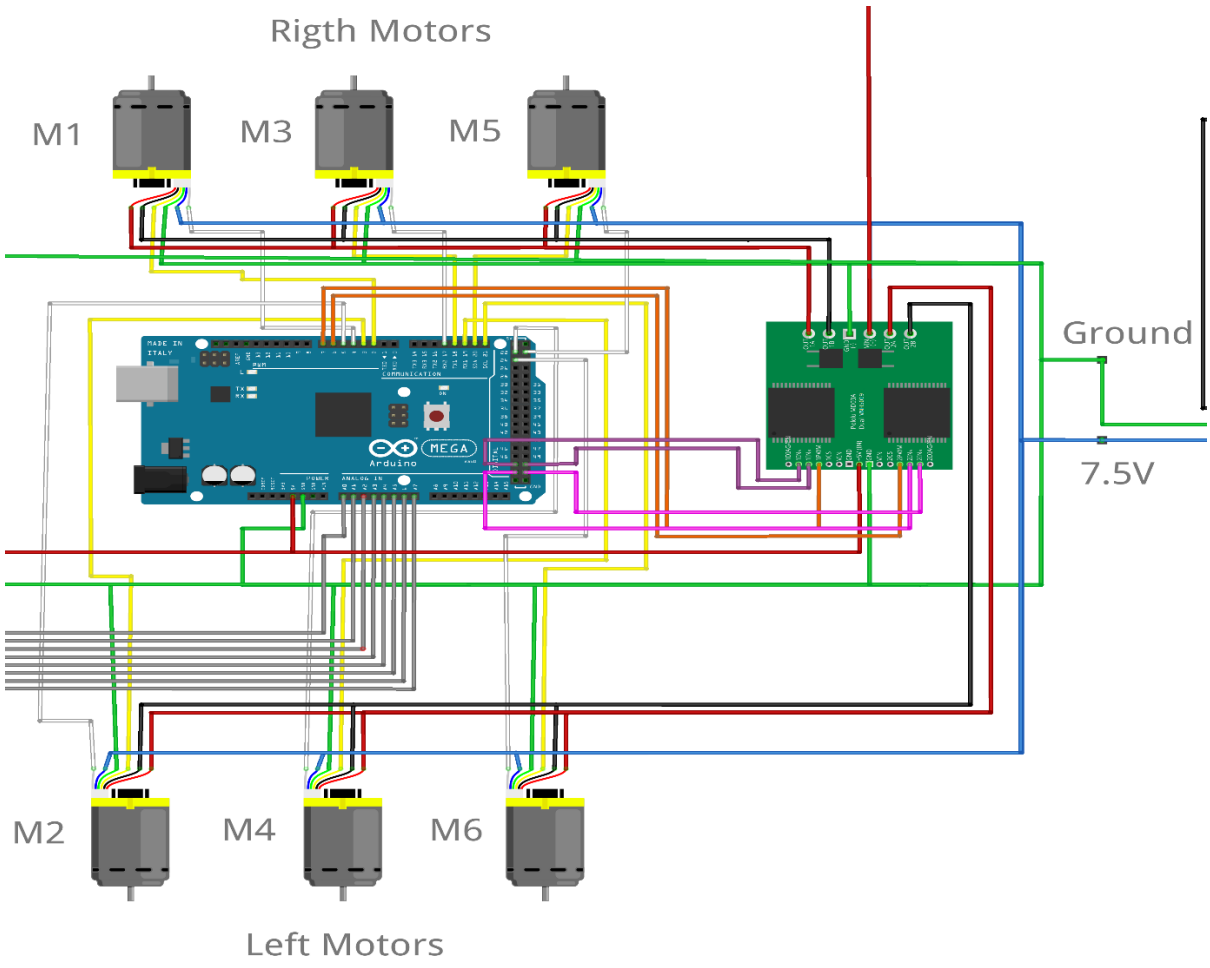
- Diagrama general:



- Etapa de regulación de voltaje



- Conexión de los motores al Arduino y al Driver VNH5019



- Conexión de las fotoresistencias al Arduino

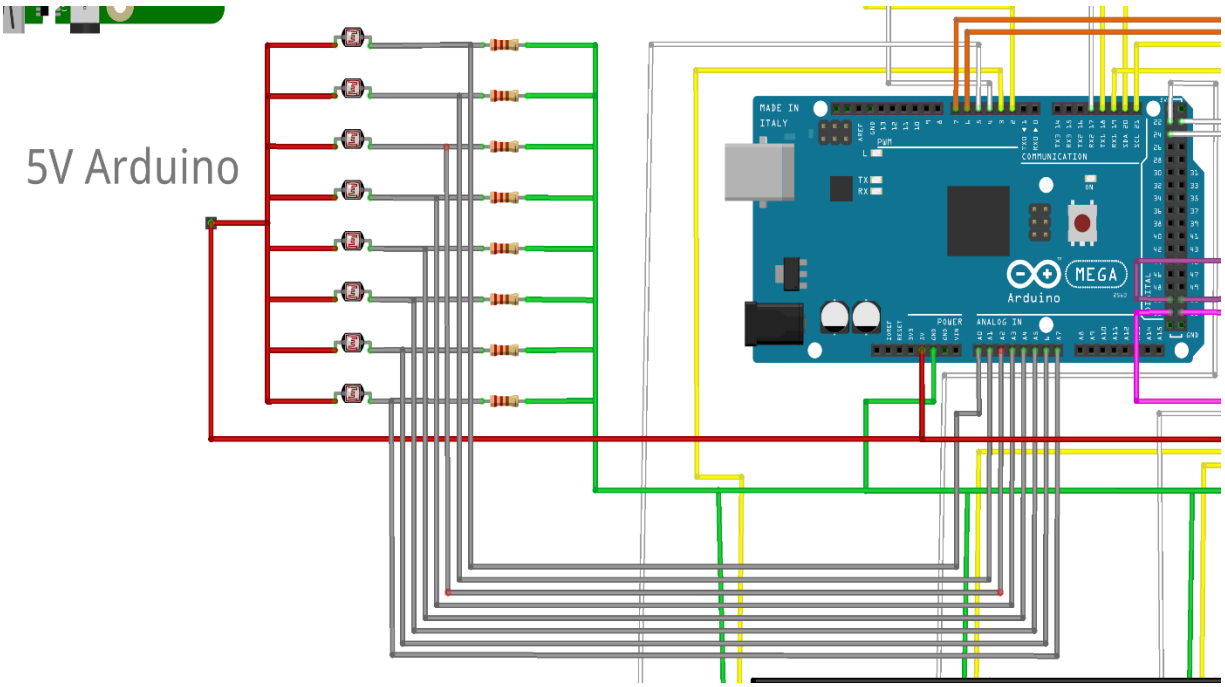


Tabla de conexiones

Pines Arduino	Destino
2 (Interrupt 0)	EncA_Motor1
4	EncB_Motor1
3 (Interrupt 1)	EncA_Motor2
5	EncB_Motor2
18 (Interrupt 5)	EncA_Motor3
17	EncB_Motor3
19 (Interrupt 4)	EncA_Motor4
22	EncB_Motor4
20 (Interrupt 3)	EncA_Motor5
23	EncB_Motor5
21 (Interrupt 2)	EncA_Motor6
24	EncB_Motor6
6	PWM_1 (Der)
7	PWM_2 (Izq)
48	2INA (Izq)
49	2INB (Izq)
50	1INA (Der)
51	1INB (Der)
A0	Fotoresistencia 1
A1	Fotoresistencia 2
A2	Fotoresistencia 3
A3	Fotoresistencia 4

A4	Fotoresistencia 5
A5	Fotoresistencia 6
A6	Fotoresistencia 7
A7	Fotoresistencia 8

Ejecución del sistema

1. Asegurarse de que el Arduino tiene cargado el programa correcto.

2. Asegurarse de que la pila tenga carga suficiente y esté conectada su alarma de Low Battery.

3. Encender la Raspberry y asegurarse de que esté comunicada con el Arduino m ediante el cable USB.

4. Conectarse a la red inalámbrica llamada "FrankyAP" e ingresar la siguiente cont raseña: raspberry.

5. Conectarse vía ssh con la Raspberry, como se observa en la siguiente imagen, l a contraseñar es *raspberry*:

```
knight@key:~-> ssh pi@172.24.1.1
pi@172.24.1.1's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l
```

6. Una vez conectados, se procede alevantar todos los nodos, esto se realiza ejec uta el paquete launch, como se muestra a continuación:

```
gerardo@gerardo:catkin_ws-> roslaunch surge_et_ambula Azcatl.launch
```

En el archivo .launch se describen todos los nodos que se quieren levantar cuando se ejecute el comando launch. Nota: El comando launch tambien lanza el roscore, en caso de no tenerlo levantado.

A continuación se muestra el contenido del archivo Azcatl.launch donde se escribe n los nodos a levantar:

```
<launch>
    <param name="robot_description" command="cat $(find mobile_base)/urdf/Azcatl.urdf"/>
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>
    <node name="serial_node" pkg="rosserial_python" type="serial_node.py">
        <param name="port" value="/dev/ttyUSB0"/>
        <param name="baud" value="115200"/>
    </node>
    <node name="mobile_base_node" type="mobile_base_node" pkg="mobile_base" output="screen"/>
    <node name="simple_move_node" type="simple_move_node" pkg="simple_move" output="screen"/>
    <node name="carrito_control_node" type="carrito_node" pkg="carrito_control" output="screen"/>
    <node name="hokuyo_node" type="hokuyo_node.py" pkg="hokuyo" output="screen">
    <node name="webcam_man_node" type="webcam_man_pub" pkg="webcam_man" output="screen"/>
</launch>
<!--
    El launch levanta el roscore, si es que éste último no está levantado
    name->Es un identificador único para el nodo en el xml
    type->Es el archivo que se quiere lanzar (.cpp/.py)
    pkg->Es la ruta relativa
    output->Para que se muestre la salida en la consola o a un archivo /log
-->
```

7. En caso de no querer levantar todos los nodos, lo cual se recomienda si es que se se está desarrollando alguna funcionalidad o se está debuggeando, se tienen q ue levantar uno por uno aquellos nodos que se deseen levantar, pero primero se ti ene que levantar el roscore:

```
pi@raspberrypi:~/catkin_ws $ roscore
... logging to /home/pi/.ros/log/e3e20036-8d53-11e9-a1b0-b827eb6ef589/roslaunch-
raspberrypi-1251.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://raspberrypi:33707/
ros_comm version 1.12.9
```

Para cada uno de los nodos se introduce el siguiente comando:

rosrun <nombre del paquete> <nombre del archivo>

Ejemplo: rosrun carrito_control carrito_node

```
pi@raspberrypi:~ $ rosrun carrito_control carrito_node

 Giro Avance:
```

Ejemplo: rosrun hokuyo_node hokuyo_node.py

```
gerardo@gerardo:catkin_ws-> rosrun hokuyo hokuyo_node.py
```

En el caso del nodo de conexión serial, se tienen que indicar unos parámetros (los cuales se pueden ver en el archivo .launch, son variables dependiendo las caract erísticas de la conexión), estos parámetros se introducen de la siguiente manera c uando se levanta individualmente el nodo:

```
pi@raspberrypi:~ $ rosrun rosserial_python serial_node.py /dev/ttyACM0 _baud:=11
5200
[INFO] [1560374704.488549]: ROS Serial Python Node
[INFO] [1560374704.557214]: Connecting to /dev/ttyACM0 at 115200 baud
```

Nota: Para cada nodo que se levante, se necesita una terminal conectada vía SSH con las Raspberry y mantenerlas abiertar para que los nodos sigan corriendo.

Se sugiere la siguiente secuencia de lanzamiento:

-Nodo para que los demás nodos puedan comunicarse (*roscore*)

-Nodo de conexión serial (*rosrun rosserial_python serial_node.py*)

-Nodo del sensor laser (*rosrun hokuyo_node hokuyo_node.py*)

-Nodo de la odometría (*rosrun mobile_base mobile_base_node*)

-Nodo del publicador de la cámara (*rosrun webcam_man webcam_man_pub*)

-Exportar el nodo maestro de la Raspberry a la computador. Luego,el nodo del "list ener" de la cámara (*rosrun webcam_man webcam_man_listener*)

-Nodo del control (*rosrun carrito_control carrito_node*)

También, si es que se está desarrollando una funcionalidad o se está debuggeand o se recomienda hacer llamadas individuales a los tópicos para confirmar su funcio namiento. Para listarlos se usa el comando *rostopic list*, y para llamar a alguno se usa el comando *rostopic echo /<nombre del tópico>*, por ejemplo:

```
pi@raspberrypi:~ $ rostopic list
/diagnostics
/encoDer
/encoIzq
/motor_speeds
/photo_sensors
/rosout
/rosout_agg
pi@raspberrypi:~ $ rostopic echo /encoIzq
```

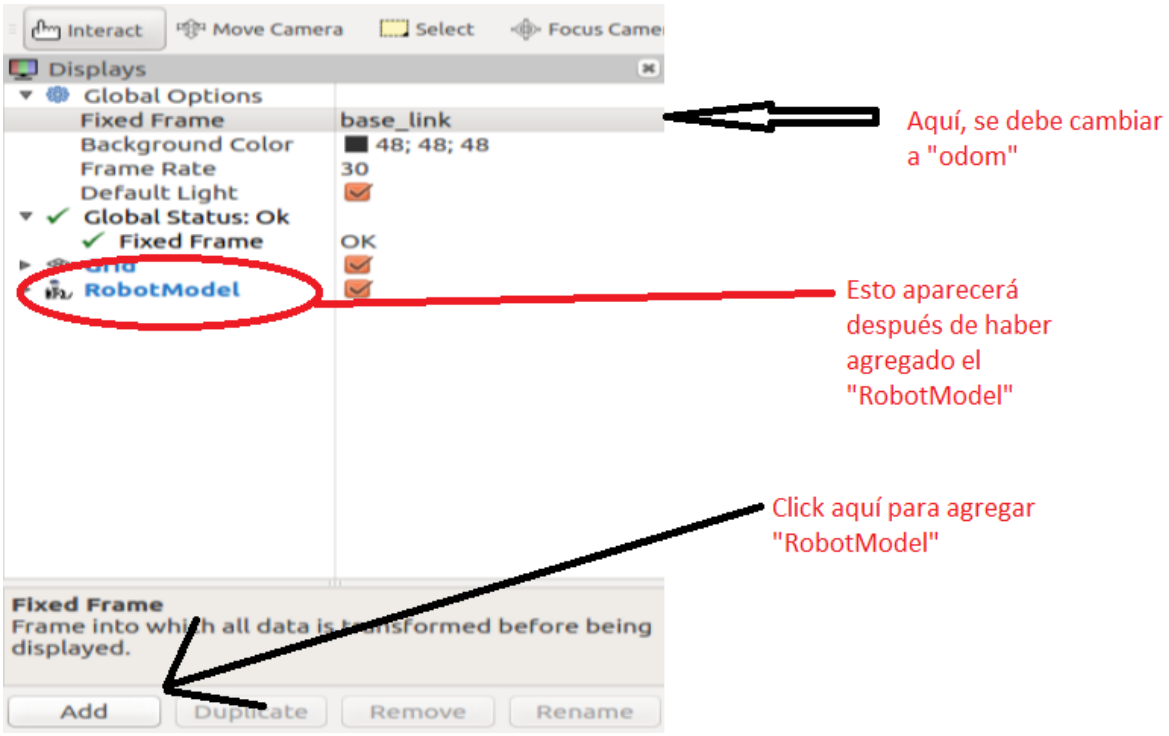
8. Existe un nodo que se corre en la computadora, corresponde al "listener" de la c ámara, para levantar ese nodo primero se tiene que exportar el nodo maestro de l a Raspberry a la computadora. Desde una terminal de la computadora se ingresa el siguiente comando:

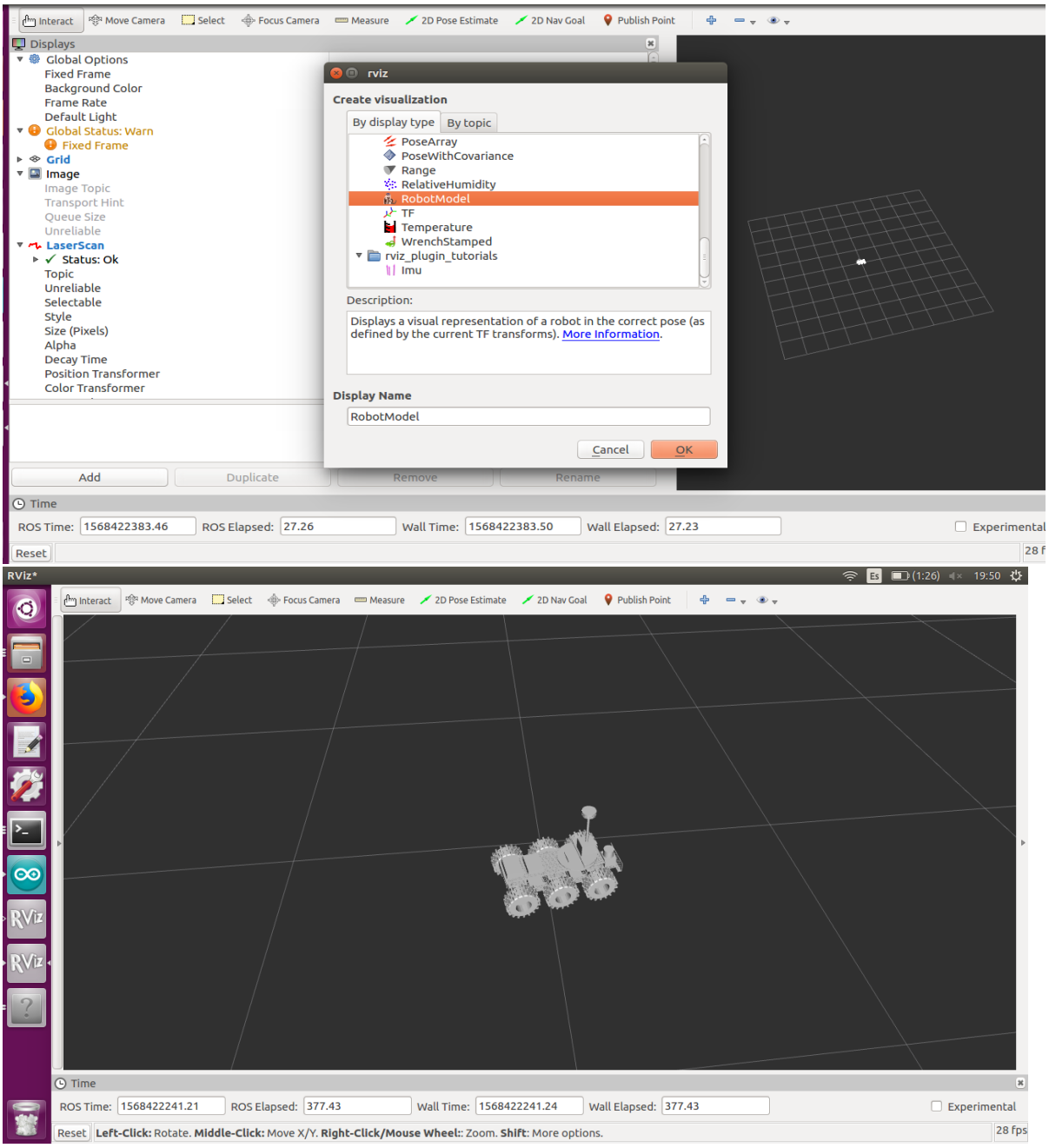
```
gerardo@gerardo:catkin_ws-> export ROS_MASTER_URI=http://172.24.1.1:11311
gerardo@gerardo:catkin_ws->
```

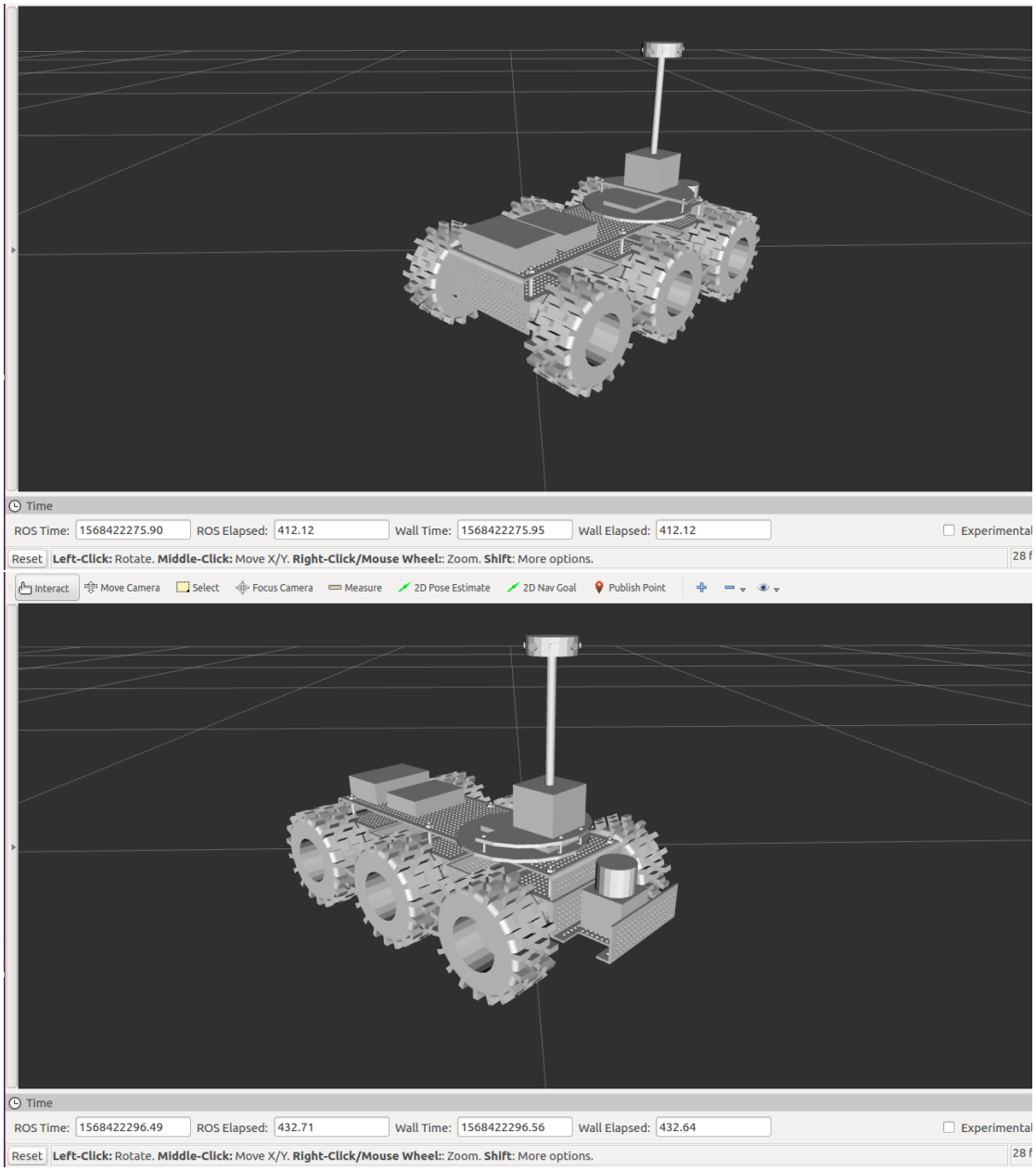
9. Una vez exportado el nodo, ahora sí se levanta el nodo con el comando *rosrun webcam_man_listener.*

```
gerardo@gerardo:catkin_ws-> rosrun webcam_man webcam_man_listener
Inicializando webcam_man_pub node...
```

10. Para abrir el visualizador RVIZ, también se debe exportar el nodo, en caso de no haberlo exportado, entonces exportar el nodo para poder levantar el visualizado r. Luego, ingresar el siguiente comando *rosrun rviz rviz*, y deberá desplegarse lo si guiente:




Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Displays
Global Options
Fixed Frame
Background Color
Frame Rate
Default Light
Global Status: Warn
Fixed Frame
Grid
Image
Image Topic
Transport Hint
Queue Size
Unreliable
LaserScan
Status: Ok
Topic
Unreliable
Selectable
Style
Size (Pixels)
Alpha
Decay Time
Position Transformer
Color Transformer
rviz
Create visualization
By display type
By topic
PoseArray
PoseWithCovariance
Range
RelativeHumidity
RobotModel
TF
Temperature
WrenchStamped
rviz_plugin_tutorials
Imu
Description:
Displays a visual representation of a robot in the correct pose (as defined by the current TF transforms). More Information.
Display Name
RobotModel
Cancel
OK
Add
Duplicate
Remove
Rename
Time
ROS Time: 1568422383.46
ROS Elapsed: 27.26
Wall Time: 1568422383.50
Wall Elapsed: 27.23
Experimental
Reset
RViz*
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Nav Goal
Publish Point
Time
ROS Time: 1568422241.21
ROS Elapsed: 377.43
Wall Time: 1568422241.24
Wall Elapsed: 377.43
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
28 fps



11. En la caja de texto donde dice Fixed Frame estará seleccionado por defecto la opción "base_link", cambiar a "odom" y en la parte de abajo de la interfaz dar Click en "Add" y seleccionar la opción de "Robot Model" para que se pueda visualizar el comportamiento del robot.

Objetivos a alcanzar

- Implemetar control de velocidad exponencial y comparar rendimiento contra el co ntrol PID.

- Programar los algortimos de inteligencia artificial: Depth First Search (DFS), Bran ch First Search (BFS), Dijkstra, A*.

- Implementar un nodo de conexión entre el robot y el simulador.

- Corregir el error de deriva en el avance con la implementación de una IMU.

- Implementación de un filtro de Kalman extendido para poder predecir los errores i nherentes a la instrumentación.

- Generar mapas de representación del ambiente mediante la técnica de SLAM (Si moultaneous Localization And Mapping.

Recomendaciones

- Mantenimiento del sistema mecánico: Cambio de los brackets que sostienen a lo s motores por unos que no provoquen que el motor quede en Cantilever (voladizo) , cambio de los coples de los motores debido a desgaste.

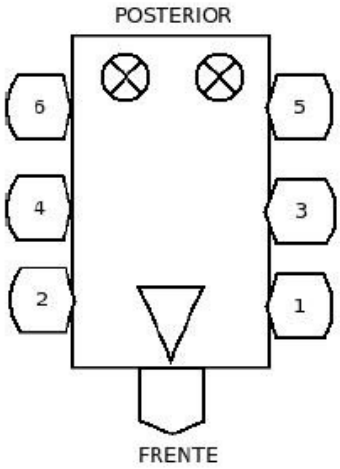
- Cambiar la pila que se tiene actualmente por una de mayor capacidad, para aum entar la autonomía del robot.

Anexo

Identificación de colores de las conexiones del motor:

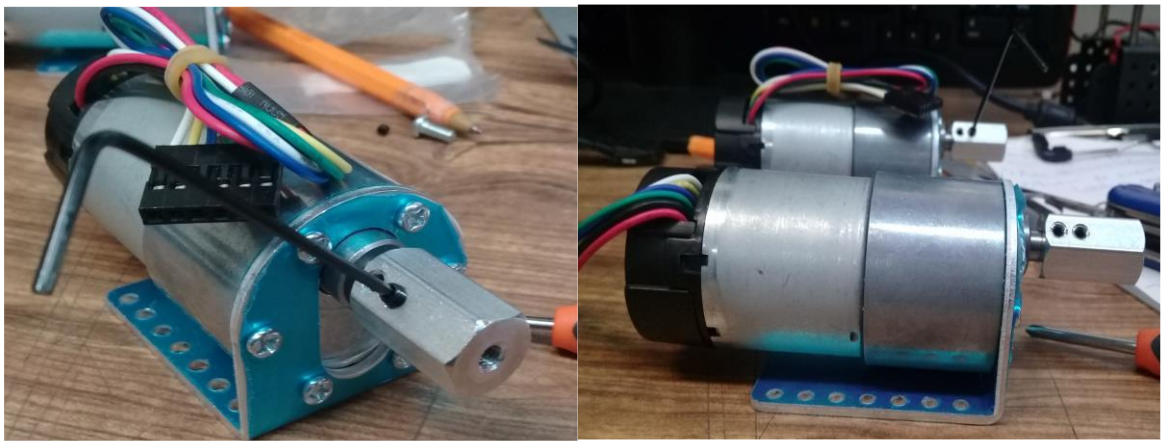
Lead Color	Function
Red	Motor power
Black	Motor power
Green	Encoder ground
Blue	Encoder Vcc (3.5 V to 20 V)
Yellow	Encoder A output
White	Encoder B output

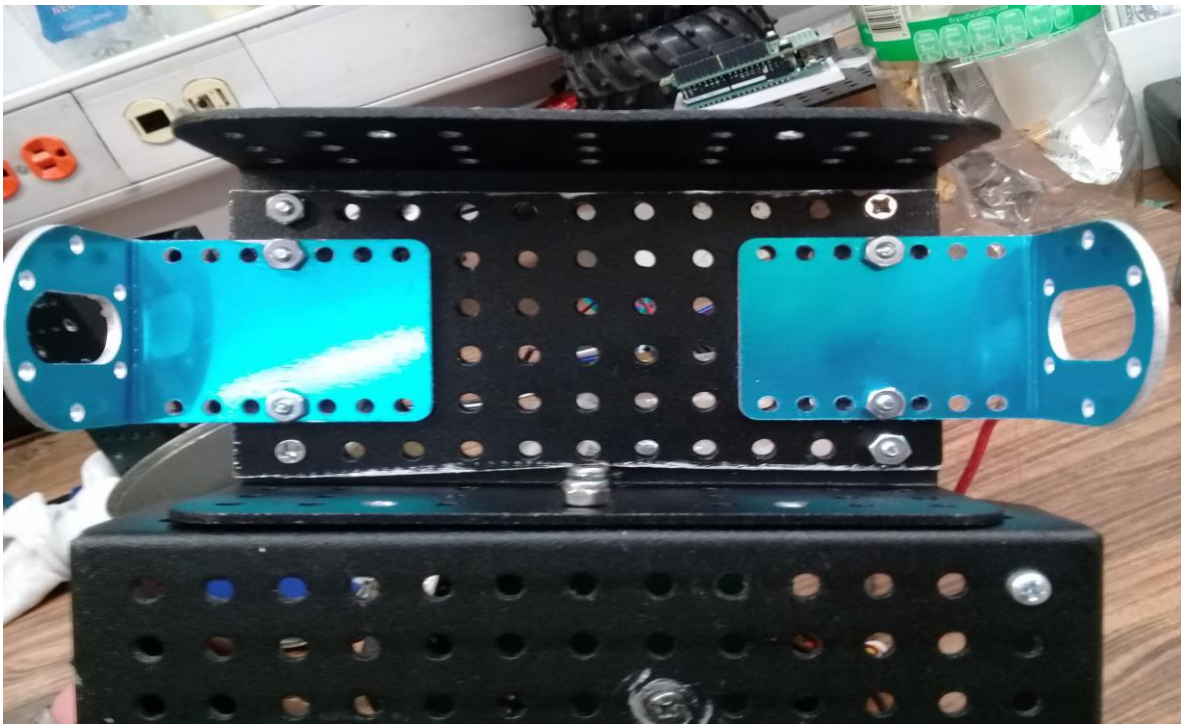
Referencia de la posición de los motores:

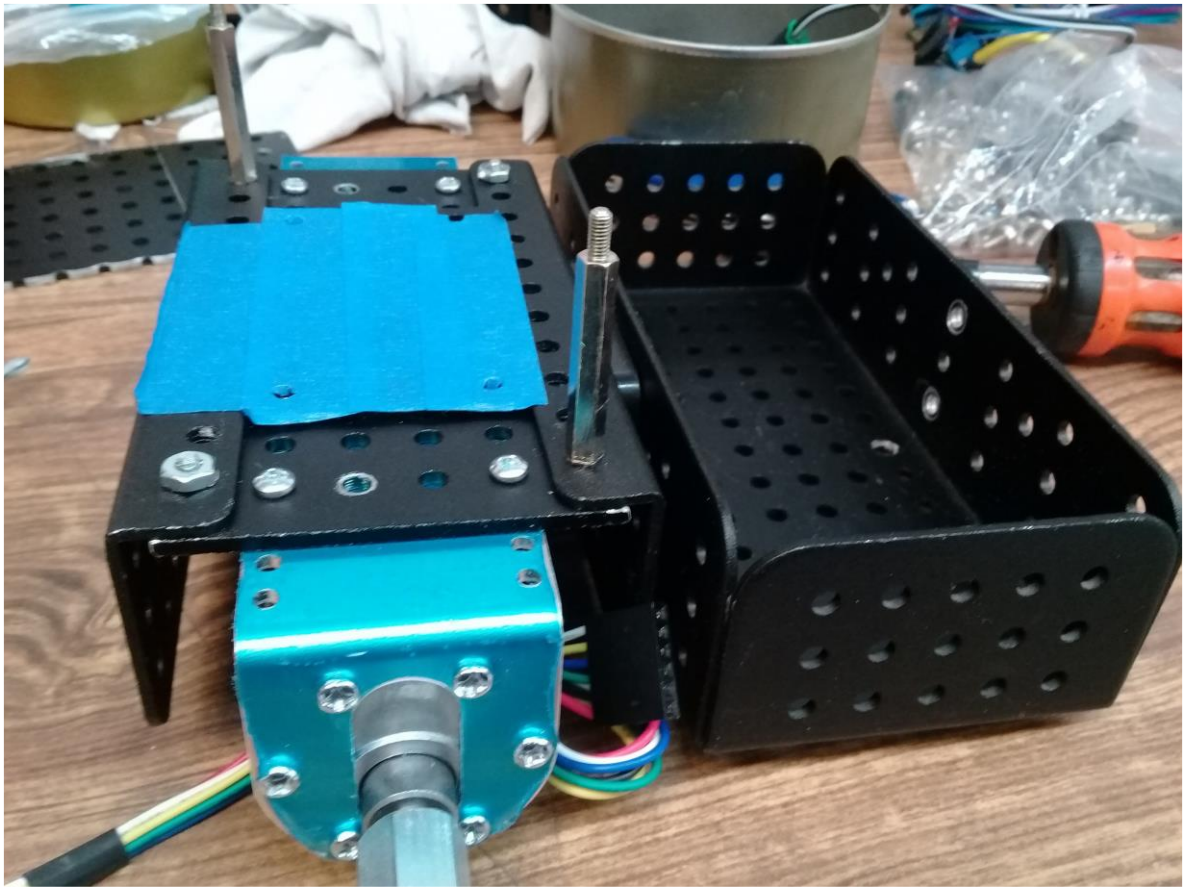


- **MOTOR 1::** Frente Derecha
- **MOTOR 2::** Frente Izquierda
- **MOTOR 3::** Medio Derecha
- **MOTOR 4::** Medio Izquierda
- **MOTOR 5::** Atrás Derecha
- **MOTOR 6::** Atrás Izquierda

Estructura mecánica:

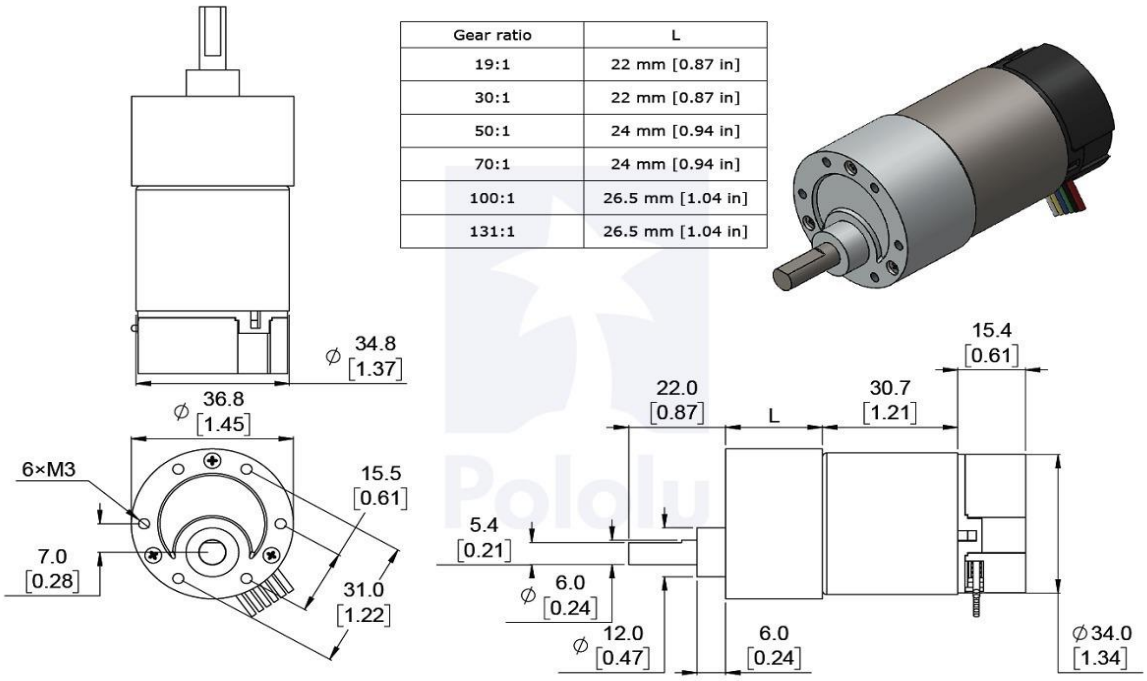






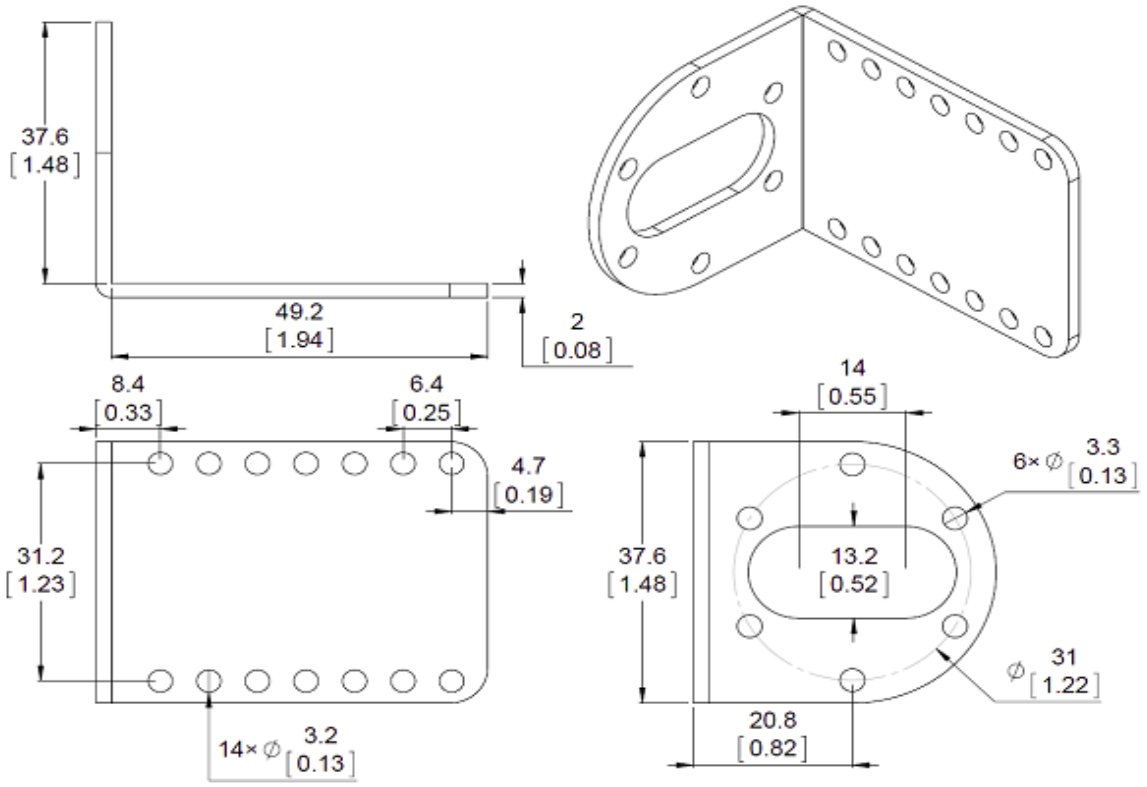
Planos de los motores:

Gear ratio	L
19:1	22 mm [0.87 in]
30:1	22 mm [0.87 in]
50:1	24 mm [0.94 in]
70:1	24 mm [0.94 in]
100:1	26.5 mm [1.04 in]
131:1	26.5 mm [1.04 in]

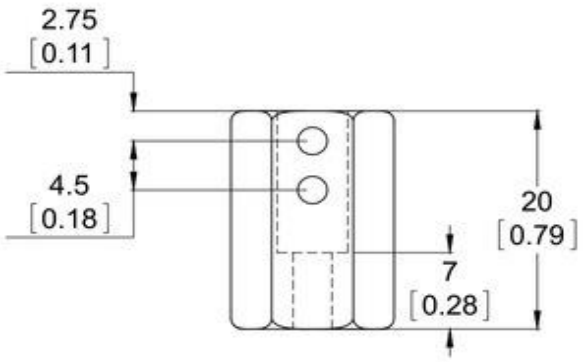


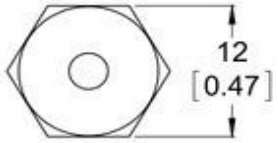
www.pololu.com

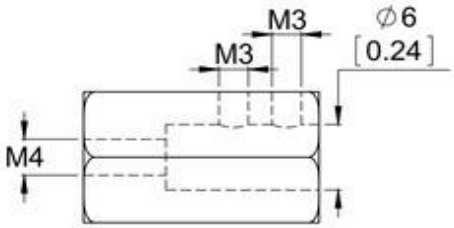
Planos de los brackets:



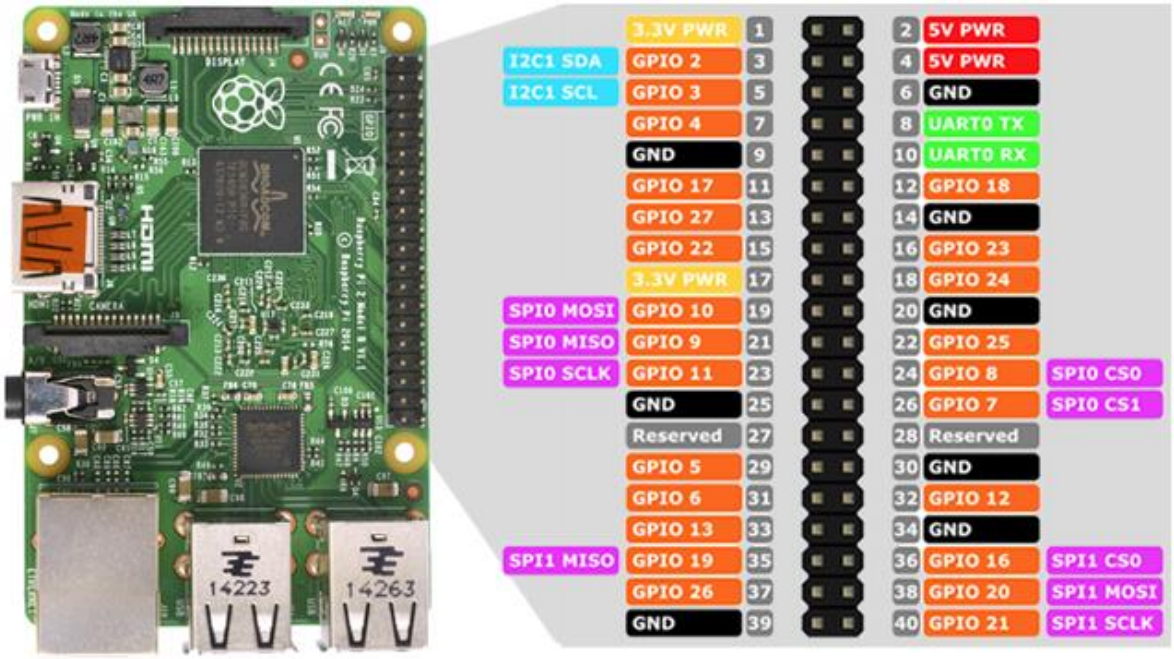
Planos de los coples:







Pines de conexión de la Raspberry Pi 3:



- Modelo del robot en físico


MULTI CHASSIS

- Recursos:

Regulador Step Down 300W 9A

https://www.cdmxelectronica.com/producto/regulador-step-down-300w-9a/?fbclid=IwAR3JRuNbcl_9QSioPW32zaR0T1-Aq35sZNkNYnjzi3sgV4ZUsWTse4NYn50

Fuente de poder 12V 5A

https://store.robodacta.mx/energia/cargadores-y-fuentes/fuente-de-voltaje-12vdc-5a-es/?fbclid=IwAR1nP6GjLCAyWinclW8oVgHblWdWyC5VXIaRbKn9u9xP3tmx2dU6lXpUHy8